UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Controlador de una Cámara Ambiental para ensayos físicos con lienzos

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Romero Barberá, Antonio José

Tutor/a: Sánchez López, Miguel

CURSO ACADÉMICO: 2021/2022

Resumen

Algunas variables ambientales como la temperatura o la humedad tienen un papel importante en el comportamiento mecánico de un lienzo. Este proyecto se centra tanto en el software del sistema de control de una cámara ambiental que va a controlar temperatura y humedad para que estas variables tengan el comportamiento deseado a lo largo de una sesión, como en el hardware y los diversos componentes electrónicos de la cámara ambiental.

Palabras clave:control, temperatura, humedad, cámara, ambiental, software, hardware.

Abstract

Some environmental variables such as temperature or humidity have an important role in the mechanical behavior of a canvas. This project will focus both on the control system software of an environmental chamber that will help us control temperature and humidity so these variables will have the behavior that we wish them to have as long as the session lasts, as well as on the hardware and the various electronic components of the environmental chamber.

Keywords : control, temperature, humidity, chamber, environmental, software, hardware.

Tabla de contenidos

1. Introducción

La humedad y la temperatura son unos de los principales causantes del deterioro de las obras de arte[1]. La humedad, es la cantidad de agua presente en el aire; sin embargo, el principal problema es que medimos la humedad como humedad relativa (HR), la cual es la cantidad de moléculas de agua que el aire puede contener a una temperatura concreta. Si la humedad relativa de una habitación con una obra de arte en un lienzo es muy alta, puede llegar a provocarle al lienzo: agrietamientos, deformaciones, quebraduras, desvanecimiento de la pintura, etc. Ocurre algo similar con la temperatura. Si la temperatura de una habitación es muy elevada, puede llegar a estropear el color de una pintura.

Por lo que, una temperatura y una humedad no controlada en una habitación o un museo que contenga lienzos estropeará y destruirá completamente las obras de arte y, de alguna forma, debemos poder controlarla.

1.1. Motivación

Las principales motivaciones que se han tenido para este Trabajo de Fin de Grado eran, principalmente, el pensamiento de ser capaz de ingeniar una forma de poder comunicar un PC con otro dispositivo el cual funcione como una CPU capaz de saber qué actuadores[1] tendría que activar y cuales debería desactivar con el fin de poder mantener una temperatura y una humedad que se hayan indicado en el envío de datos que se ha realizado desde el PC.

La capacidad y el reto de trabajar con tecnologías que estaban fuera de los estudios y conocimientos que he aprendido en el Grado de Ingeniería Informática y poder trabajar con ingenieros de otros campos más versados en dispositivos de control de variables ambientales y tecnologías de comunicación; todos estos factores han sido las motivaciones, tanto académicas como personales, que me han empujado a la hora de realizar este proyecto.

1.2. Objetivos

El objetivo principal de este Trabajo de Fin de Grado es, principalmente, el poder controlar la temperatura y la humedad del aire contenido en una cámara cerrada. Posteriormente este objetivo ha derivado en objetivos secundarios a la hora de ir diseñando y emprendiendo este proyecto.

Algunos objetivos secundarios que se pueden listar por su importancia son:

[1] Un actuador es un dispositivo capaz de transformar energía hidráulica, neumática o eléctrica en la activación de un proceso con la finalidad de generar un efecto sobre un proceso automatizado

- Poder comunicar un PC con un dispositivo capaz de tratar los datos enviados.
- Crear un programa capaz de enviar datos a un dispositivo y recibir los datos de este mismo.
- Crear una base de datos capaz de almacenar y registrar las variables de temperatura y humedad.
- Recibir y tratar los datos que se reciben de un PC y, en consecuencia, mandar acciones a los distintos actuadores para controlar temperatura y humedad.
- Limitar los márgenes de los datos que se envían y ser capaz de mostrar una gráfica de los cambios de temperatura y humedad.

1.3. Impacto Esperado

El impacto que se espera con el producto final/resultante de este proyecto es que se le pueda proporcionar a un usuario la capacidad de poder controlar humedad y temperatura en una cámara donde se puedan realizar ensayos con lienzos. Además de poder visualizar los cambios en estas variables utilizando unas gráficas y poder incluso hacer consultas a la base de datos si le llegase a interesar buscar datos concretos.

1.4. Metodología

Con la finalidad de abarcar los objetivos mencionados anteriormente, se ha debido de plantear una metodología que defina correctamente la estructura del trabajo a realizar. Se ha debido de analizar el estado actual de las diversas tecnologías de software que utilizan las empresas que desarrollan controladores para cámaras ambientales y, posteriormente, analizar qué tecnologías se pueden utilizar y cuales no serían óptimas para el desarrollo de nuestro trabajo. Una vez analizado el contexto tecnológico y las diversas tecnologías que se han estudiado y seleccionado, se ha tenido que diseñar una solución utilizando las tecnologías estudiadas y en base a esta solución diseñada, desarrollarla y obtener una serie de pruebas y/o conclusiones que determinen si el uso de estas mismas ha tenido un resultado esperado o el contrario.

1.5. Estructura

Con el fin de poder facilitar la lectura de este trabajo, explicaremos brevemente cada capítulo y daremos un poco de información de los anexos.

El primer (I) capítulo es la introducción al trabajo, dando un poco de contexto sobre qué ambiente se desarrolla el TFG. El segundo (II) capítulo hacemos un estudio sobre el contexto tecnológico actual sobre el que nos basamos para el desarrollo y el estudio de las distintas tecnologías. El tercer capítulo (III) discute las distintas tecnologías que existen para abordar el problema estudiado en el segundo capítulo, debatiendo sobre qué ventajas y desventajas aporta cada una y por qué hemos terminado utilizando una u otra. El cuarto (IV) capítulo muestra el cómo se ha diseñado una solución en base a las tecnologías que hemos debatido y seleccionado en el capítulo tres. El quinto (V) capítulo nos mostrará el desarrollo de la solución que hemos propuesto en el capítulo (IV) y se discuten sobre las dificultades encontradas, la solución final, etc. El sexto (VI) capítulo trata sobre las pruebas realizadas

y el cómo han influenciado estas pruebas en el desarrollo de mejoras o optimizaciones en la solución final. El séptimo (VII) capítulo es un capítulo sobre las conclusiones que se han alcanzado con el TFG, las pruebas, los problemas y qué hemos aprendido del desarrollo de este mismo proyecto, a nivel personal como profesional. Finalmente, tenemos dos capítulos extra, el octavo capítulo (VIII) contiene mejoras que se podrían haber realizado y el noveno (IX) capítulo es la bibliografía del trabajo.

1.6. Convenciones

Para poder facilitar el entendimiento, estructura y comprensión de algunos elementos de este documento, se han establecido algunos convencionalismos, entre ellos:

- El código fuente se muestra en formato `Consolas`. Y solo se empleará esta tipología para este contenido, a excepción si el código fuente sale mostrado en una fotografía anexada.
- Los nombres propios de los lenguajes de programación y tecnologías utilizadas estarán escritas en letra Bree Serif.
- Las palabras extranjeras se remarcarán en cursiva.
- El código fuente de **Arduino** en formato *`Consolas en cursiva`*

2. Estado del arte

Tras un tiempo de estudio con respecto al contexto tecnológico de las cámaras ambientales, hemos podido visualizar y analizar los diferentes datos relacionados con las especificaciones de distintos productos de venta comercial que existen en el mercado actual.

Aunque estos datos y especificaciones son públicos y pueden ser consultados por cualquier cliente y/o empresa, el tratamiento de los datos, el flujo de datos y el software que utilizan estas cámaras ambientales es información reservada de la propia empresa y, por lo tanto, limita bastante nuestro estudio sobre el estado actual del arte.

No obstante, hemos podido realizar un estudio sobre el contexto tecnológico actual utilizando y comparando la información pública de las especificaciones técnicas de empresas como: *ThermoFisher Scientific*[2] *, Instron*[3] *, Memmert*[4].

La mayoría de empresas que trabajan con cámaras ambientales o el estudio del efecto de algunas variables ambientales sobre algún producto utilizan un tipo de software llamado *SCADA,* acrónimo de *Supervisory Control And Data Acquisition,* (Control Supervisor y Adquisición de Datos). Este concepto se emplea para denominar a los tipos de Software para ordenadores que permiten controlar y supervisar procesos industriales a distancia y facilitar la retroalimentación.

Algunas empresas como por ejemplo *Memmert* tienen su propio SCADA, *Celsius*[5], cuyo software tiene una programación sencilla y lectura de datos. Este software, como bien muestran en su página web, posee unas funciones muy útiles como, por ejemplo:

- Programación gráfica y numérica.
- Conexión con RS232, RS485, USB y Ethernet para distintos equipos.
- Programación del número de rampas que se desee.
- Copia de seguridad automática de todos los datos del proceso.
- Documentación e impresión de los procesos de regulación de temperatura.

Otra empresa como por ejemplo *Instron* y su propio SCADA, poseen unas funciones de seguridad las cuales limitan las capacidades operativas de su software dependiendo del nivel de acceso que tenga el usuario/operador que haya accedido a la aplicación.

Como podemos observar por estas especificaciones, las empresas del sector de las cámaras ambientales poseen un software propio con diferentes funciones, cada uno distinto a otro.

2.1. Crítica al estado del arte

[2] https://www.thermofisher.com/es/es/home.html
[3] https://www.instron.com/en/products/materials-testing-software/bluehill-elements
[4] https://www.memmert.com/home/
[5] https://www.memmert.com/es/por-que-memmert/lideres-en-innovacion/celsius/

Como bien hemos visto en el apartado anterior, existen diferentes formas de comunicar la aplicación SCADA, existen conexiones por cable, conexiones ethernet, y existen diversas tecnologías de conexión wireless, existe incluso la tecnología que implementan muchas empresas de utilizar el protocolo de comunicación Modbus, el cual está basado en una arquitectura maestro/esclavo. Este protocolo, aunque es el más utilizado por los controladores ambientales en la industria, utiliza un protocolo TCP conectado por Ethernet a PLCs[6] conectando la aplicación SCADA a un servidor el cual pide una conexión contínua, lo cual puede suponer un problema. Otra desventaja que posee es que requiere una alta cantidad de programación y configuración, lo cual supone un problema si no has invertido una gran cantidad de tiempo para investigar su funcionamiento y configuración, además de la cantidad de recursos que tienes que invertir, entre ellos, los PLCs, los servidores, etc.

2.2. Propuesta

La propuesta y la que será la base sobre la que se construye este Trabajo de Fin de Grado es la capacidad de mejorar las limitaciones que poseen los SCADA ya existentes y hacer un cambio con respecto a las tecnologías que se utilizan. Como bien hemos visto anteriormente, el uso de tecnologías como por ejemplo PLCs , ModBUS, conexiones por cable Ethernet y demás, tienen limitaciones, tanto económicas como de estudio de la tecnología, su configuración y su programación. Por lo que nuestra aproximación para este trabajo será el uso de una tecnología sencilla, sin complicaciones de programación ni configuración y que tenga bajo coste de estudio y económico.

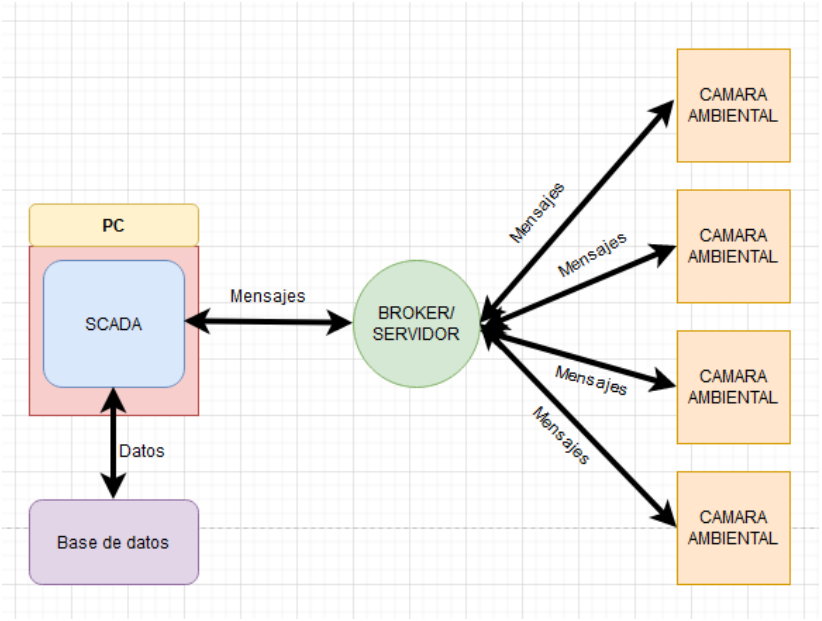
Utilizaremos tecnologías que nos permitan conectarnos desde un PC a una cámara ambiental asumiendo que puede haber problemas en la red y que tendremos un ancho de baja reducido, de esta forma podemos incrementar la distancia entre nuestro SCADA y la cámara ambiental y además de poder abarcar las posibles complicaciones que surjan por factores externos pero intentando mantener gran parte de las tecnologías que poseen la mayoría de las aplicaciones SCADA de uso industrial.

Además de utilizar la propuesta anterior, también pensamos a futuro con posibles ampliaciones por lo que se debería poder plantear un diseño escalable y fiable que nos permita seguir haciendo implementaciones y ampliaciones a largo plazo.

Nuestra propuesta es utilizar un intermediario capaz de recoger los mensajes de diversas cámaras ambientales a la vez utilizando poco ancho de banda de forma que no sature el sistema ni la red y que

[6] Un controlador lógico programable o PLC es una computadora utilizada en ingeniería automática para automatizar procesos

a su vez seamos capaces de seleccionar nosotros desde nuestra aplicación SCADA qué cámara ambiental queremos modificar o consultar sus variables.



7 Muestra abstracta de la propuesta a desarrollar de nuestro proyecto

No obstante, para el desarrollo de este proyecto, se ha ido a pequeña escala y se ha trabajado únicamente sobre una cámara ambiental. No obstante, posteriormente, en el capítulo de "Trabajos futuros", explicamos esta escalabilidad del sistema para añadir más cámaras ambientales y cómo influiría dicha ampliación en nuestro desarrollo.

[7] Muestra abstracta de nuestro diseño

3. Análisis del problema

El problema que hemos comentado en el apartado anterior sobre la comunicación entre una cámara ambiental y un PC, tiene múltiples partes que podemos analizar para encontrar posibles soluciones.

La principal innovación es el hecho de que podemos comunicarnos a distancia desde un PC a una cámara ambiental utilizando una tecnología de bajo coste y poca banda ancha. Esto puede ser muy útil si tenemos en cuenta que podemos trabajar en remoto para mantener el control de la temperatura y la humedad sin la necesidad de perder el tiempo en llegar a la habitación o al edificio donde se tenga la cámara ambiental. Además es posible que la banda ancha a la que estén conectadas las distintas cámaras ambientales esté muy limitada, lo que permitiría el uso continuo de las mismas en caso de que factores externos limiten la conexión.

Otra principal innovación es el poder guardar los datos de la cámara ambiental en una base de datos local de forma que podemos hacer consultas a la misma con parámetros como la humedad, la temperatura o un margen de horas específico.

No solo podemos guardar los datos de en una base de datos sino que también podemos utilizar estos datos para crear gráficas que nos permiten visualizar los cambios de las variables de una cámara ambiental específica.

3.1. Análisis de riesgos

Algunos riesgos que podemos analizar de este problema son, entre otros: La capacidad máxima de aumento de temperatura que puede llegar la resistencia, la capacidad mínima de disminución de temperatura que pueden llegar las células Peltier, la capacidad máxima y mínima de lectura que posee nuestro sensor, la posibilidad de un mal cableado sobre el sensor, la mala introducción de datos e intento de envío de estos.

1. Empezaremos por la capacidad máxima de aumento de la temperatura que puede llegar la resistencia, este tipo de riesgo es uno que se tendría que tener muy en cuenta en nuestra simulación de caja ambiental, puesto que, como bien sabemos, el calor puede estropear distintos materiales que componen la cámara ambiental. En nuestro caso, la cámara ambiental estaba hecha de plástico y, sabiendo que la resistencia podría calentarse hasta fundir la pared de la cámara ambiental desarrollada en nuestro proyecto. La medida que tomamos fue limitar la temperatura máxima en grados que se pueden introducir en nuestra aplicación controladora para enviar a la cámara ambiental de forma que no dañase la cámara ni el resto de componentes.

2. La capacidad mínima de disminución de temperatura de las células peltier también es un riesgo que hemos tenido que tener en cuenta, puesto que, estos componentes electrónicos tienen un límite mínimo de temperatura que pueden llegar a tener si se les introduce toda la potencia posible y, sabiendo esto, podría llegarse a dar el caso, que se hubiesen introducido unos valores deseados de temperatura inferiores a los que los componentes podrían alcanzar.

La solución y medida para este caso ha sido limitar los grados mínimos que pueden introducirse en la aplicación controladora para enviar a la cámara ambiental de forma que no se dañen los componentes eléctricos al intentar llegar a esa temperatura.

Para ambos riesgos anteriormente mencionados se han estudiado diversas pruebas, entre ellas, hacer un test de estrés a nuestras células Peltier de forma que les introducimos toda la potencia posible durante un rango de tiempo y probamos a reducir la temperatura el máximo número de grados posibles. Es importante mencionar un detalle y es que, para estas pruebas, el resultado de las mismas varía dependiendo de la zona y la habitación en donde se halle la cámara, puesto que la temperatura externa a la cámara ambiental también ha influido en estas pruebas. Si se hubiese deseado hacer un estudio más en profundidad de cuáles son los valores mínimos y máximos de temperatura que se pueden alcanzar, hubiesen sido necesarios elementos, dispositivos y componentes de mejor calidad, no obstantes, para el problema y el desarrollo del objetivo principal de este trabajo, no hemos indagado en la calidad de los componentes.

3. La capacidad máxima y mínima de temperatura y humedad que puede leer el sensor del que se disponía para la cámara ambiental debía tener unos valores máximos y mínimos de ambas variables. Y la solución fue buscar la documentación oficial del fabricante del sensor y comprobar el rango de valores que podía leer correctamente el sensor y, con estos rangos de valores, limitar el porcentaje de humedad que se puede introducir en la aplicación controladora.

4. Cabe la posibilidad de que el cableado del sensor se rompa de alguna forma o no funcionen correctamente las conexiones entre el sensor la cámara ambiental, este riesgo es poco probable pero no imposible que se llegase a ocurrir, en caso de que esto ocurriera podrían producirse fallos de lectura, de forma que sin la retroalimentación de nuestro sensor a nuestra cámara ambiental, podríamos subir o bajar las temperaturas por encima o por debajo de los límites y podría provocar errores, fallos y daños a la cámara ambiental y los componentes que posee.

5. La introducción de datos y el envío de estos mismos a la cámara ambiental también era un riesgo que podría llegar a ocurrir, que en vez de enviar un valor numérico a la cámara ambiental pudiese enviar una cadena de caracteres podría invalidar la correcta utilización de nuestra aplicación controladora.

   La solución propuesta para este caso ha sido utilizar métodos de comprobación de tipos de valores introducidos. No obstante, de este riesgo se hablará más tarde , durante el desarrollo de la solución.

3.2. Identificación y análisis de soluciones posibles.

Uno de los principales problemas que se ha tenido que resolver ha sido la comunicación entre el PC y la cámara ambiental, para resolver esto, se ha debido utilizar un dispositivo capaz de ser configurado de forma que pudiese recibir datos por protocolos de

mensajería y además pudiese ser programado. La solución para este problema ha sido utilizar una **Raspberry PI**[8].

Raspberry PI es un ordenador de bajo coste y formato compacto destinado al desarrollo de tecnologías y prototipos. Esta tecnología permite la instalación de protocolos de mensajería y además permite el paso de datos e información por puerto de serie. Por lo que, para la finalidad de este proyecto, ha sido la elección final.

Otro problema que se ha planteado ha sido la selección del tipo de protocolo de mensajería que se iba a utilizar tanto en el PC como en la Raspberry PI. Para este problema que se nos presenta se investigaron diversas posibilidades, principalmente se tuvo que investigar los diversos protocolos de mensajería para el *Internet of Things* (IOT).

Tras haber investigado sobre las distintas tecnologías de comunicación, hemos encontrado estos tres protocolos siguientes que se podrían aplicar para el caso que se nos presenta y la propuesta que hemos comentado en el apartado anterior:: **MQTT, AMQP, DDS.**

Vamos a analizar estas tecnologías y a dar una explicación de cual de estas fue la elegida para el desarrollo de nuestro trabajo antes de continuar con el resto de tecnologías y su utilización.

1. **MQTT (*Message Queuing Telemetry Transport)***

Este protocolo de mensajería es un protocolo que ha ganado mucha popularidad puesto que su transporte de mensajes tipo *publish/subscribe* es muy ligero, lo que permite la comunicación remota para dispositivos con ancho de banda limitado.
Este tipo de protocolo presenta algunas ventajas muy útiles como por ejemplo:

- Es compatible con Python y otros lenguajes de programación comunes.
- El tamaño de los paquetes es muy inferior al resto de protocolos de mensajería.
- Utiliza protocolos de mensajería tipo suscriptor/publicador
- Proporciona unos resultados rápidamente.

2. **AMQP (*Advanced Message Queuing Protocol)***

Este protocolo de mensajería es un protocolo de transferencia de mensajes de negocios entre varias aplicaciones y compañías. Aunque este protocolo realmente no está diseñado específicamente para funcionar con aplicaciones *IOT* sigue siendo un protocolo que funciona eficazmente para la comunicación de mensajes.
Este tipo de protocolo presenta algunas ventajas muy útiles como por ejemplo:

- Ayuda en la encriptación de los mensajes durante la comunicación.
- Permite la transferencia de mensajes a través de TCP[9] y UDP[10].
- Encriptación de principio a fin de la mensajería.
- Se adapta muy bien a los escenarios de mensajería

3. **DDS (*Data Distribution Service)***

[8] Una Raspberry PI es un ordenador de bajo coste y formato compacto.
[9] TCP es un protocolo de red que permite a dos anfitriones conectarse e intercambiar datos en el mismo orden que fueron enviados.
[10] UDP es un protocolo de red ligero que permite a dos anfitriones conectarse e intercambiar datos con un mecanismo que permite detectar datos corruptos pero no resuelve el orden en el que llegan los datos a diferencia de TCP.

Este protocolo de mensajería es un protocolo de transferencia de mensajes muy utilizado puesto que se utiliza como un protocolo *middleware*[11]. Este protocolo difiere de los dos protocolos anteriormente mencionados puesto que este protocolo actúa sin un broker[12] y utiliza al igual que los dos protocolos anteriores un tipo de mensajería *publish/subscribe*.
Este tipo de protocolo presenta algunas ventajas muy útiles como por ejemplo:

- Conexión directa entre los dispositivos.
- Utiliza protocolos de mensajería tipo suscriptor/publicador
- Su diseño soporta aplicaciones que trabajan en tiempo real.
- Tiene un uso muy eficaz del ancho de banda disponible.
- Escalable y efectivo.

Para nuestro trabajo, el protocolo de mensajería que se ha utilizado finalmente ha sido **MQTT.**

Este protocolo, a diferencia del resto, permite el envío de mensajes sin la necesidad de mucho ancho de banda o memoria o potencia y, dado que el receptor de la información enviada desde el PC es una Raspberry PI, su capacidad de ancho de banda no es muy grande y tampoco es necesario la encriptación de los datos puesto que posteriormente se deberían desencriptar y aumentaría la complejidad de nuestro proyecto de forma innecesaria.

Una vez seleccionado el protocolo de mensajería y el dispositivo receptor de la información ha habido que seleccionar e investigar qué lenguaje de programación se debía de utilizar para crear nuestro propio SCADA y qué frameworks se han tenido que utilizar.

Las dos opciones principales que se han investigado para el desarrollo de este SCADA han sido: **Java** y **Python.**

Tras un poco de investigación sobre la compatibilidad de **Java** y **Python** con **MQTT**, se llegó a la conclusión de que la opción más cómoda para este trabajo era utilizar Python.

Ambos lenguajes de programación son compatibles con la librería de **Eclipse Paho**[2], la librería que permite el uso de **MQTT** y **MQTT-SN** en distintos lenguajes de programación, Java y Python entre ellos.

Hay varios motivos por los que se ha utilizado **Python** por encima de **Java**. Principalmente ha sido por comodidad a la hora de desarrollar la propia funcionalidad del SCADA por parte del alumno. Otros motivos han sido la dificultad que posee **Java** para desarrollar su propio frontal, mientras que **Python** posee una librería llamada **TKinter**, una *GUI* [13]para desarrollar interfaces. No solo nos permite **Python** utilizar una GUI para desarrollar interfaces fácilmente, si no que también ha sido seleccionado por su compatibilidad con la tecnología que se ha usado para guardar los datos en una base de datos.

[11] El middleware es el software que brinda servicios y funciones comunes a las aplicaciones
[12] Un bróker de mensajería es un programa intermediario que traduce los mensajes de un sistema a través de un medio de telecomunicaciones
[13] Una GUI es una interfaz de usuario gráfica para representar la información y acciones disponibles

Para la base de datos han habido diversas opciones, algunas de las tecnologías que se han pensado en utilizar han sido: **SQL Server, SQLite y PL/SQL.**

Para la parte de la base de datos ha sido seleccionado **SQLite**[3], no por su comodidad de trabajar con él, si no por la comodidad de configurar y el tamaño del proyecto.

SQLite permite crear una base de datos sin necesidad de administración ni configuración y, según la documentación y la propia web de **SQLite**, es muy útil para dispositivos de *IOT,* además, al ser una base de datos utilizada principalmente para el desarrollo de este Trabajo de Fin de Grado, no ha sido necesaria la configuración de una base de datos permanente que pudiese integrar diversas funciones, procedimientos y tablas; como bien podría ofrecernos **SQL Server** o **PL/SQL**.

Además de estas tecnologías, utilizaremos **Arduino**[14]. El motivo por el que vamos a utilizar también **Arduino** es porque la **Raspberry PI** no contiene entradas analógicas[15], solamente contiene entradas digitales[16], por lo que, para poder leer la temperatura y la humedad de una cámara ambiental, necesitamos utilizar **Arduino**, el cual contiene entradas analógicas y digitales, además de ser programable.

3.3. Solución propuesta

La solución elegida ha sido construida en base a las tecnologías seleccionadas en el apartado anterior.

El trabajo a desarrollar consistirá en crear una aplicación SCADA que nos permita introducir los valores deseados de temperatura y humedad que vamos a enviar a la cámara ambiental mediante el protocolo de mensajería **MQTT** a la **Raspberry PI**, la cual estará conectada con una placa de Arduino por puerto en serie. En la placa de **Arduino** leeremos la información del sensor a través de una entrada analógica y devolveremos esa misma información a la **Raspberry PI** a través del puerto en serie.

No sólo utilizaremos **Arduino** para leer la información del sensor y devolverla por el puerto en serie, sino que también lo utilizaremos para apagar y encender los actuadores que aumentarán o disminuirán la temperatura y la humedad.

Una vez la **Raspberry PI** haya recibido la información de Arduino, esta misma se enviará a la aplicación SCADA de nuestro PC con el mismo protocolo de mensajería **MQTT**.

[14] Arduino es un microcontrolador con un programa integrado desde un ordenador que funciona de forma independiente a este y controla y alimenta diversos dispositivos.

[15] Una entrada analógica es una entrada que puede tomar cualquier valor en un intervalo entre 0V y 5V, por ejemplo, podría tomar un valor de 2.73V.

[16] Una entrada digital es una entrada que toma el valor de 0V o el valor de 5V, es decir, HIGH o LOW, este tipo de entradas no poseen un valor intermedio.

Finalmente, una vez hayamos recibido la información de la **Raspberry PI,** actualizaremos los datos actuales de temperatura y humedad que se mostrarán en la aplicación y posteriormente guardaremos los datos recibidos en nuestra base de datos **SQLite**.
Una vez se haya terminado de construir y configurar el proyecto, como se ha mencionado anteriormente, se deben hacer pruebas para comprobar que todo funciona correctamente y se han controlado los riesgos analizados.

3.4. Plan de Trabajo

El plan de trabajo inicial se realizó desde las aproximaciones y el optimismo de que los plazos de tiempo eran correctos.

Los plazos de tiempo asumidos en un principio han sido los siguientes:

Seguimiento de proyecto

Fecha de inicio:	03/02/2022
Fecha de finalización:	04/07/2022

Posición	Fecha de inicio	Fecha de finalización	Hito o actividad
1	03/02/2022	24/03/2022	Inicio y diseño
2	25/03/2022	02/04/2022	Adquisición de componentes
3	03/04/2022	17/04/2022	Desarrollo del SCADA
4	10/04/2022	16/04/2022	Desarrollo de la BBDD
5	14/04/2022	22/04/2022	Desarollo de MQTT
6	24/04/2022	17/05/2022	Programación del Arduino
7	24/04/2022	01/05/2022	Configuración de Node-RED
8	03/05/2022	10/05/2022	Construcción de la C.A.
9	13/05/2022	20/05/2022	Testing y cierre
10	20/05/2022	04/07/2022	Documentación y redacción

Sin embargo, uno de los factores que no se tuvo en cuenta a la hora de desarrollar un plan de trabajo inicial fue el trabajar con tecnologías y conocimientos nuevos y las curvas de aprendizaje que conllevan.

Los tiempos aproximados de inicio y finalización de las diversas tareas se han desviado de la planificación original conforme se ha ido avanzando en las distintas tareas del proyecto.

El inicio y el diseño ha ocupado mucho más tiempo del esperado por el diseño de la cámara ambiental y de las tecnologías de comunicación que se han utilizado.

El diseño y desarrollo de la aplicación SCADA ha durado menos tiempo del esperado a cambio de tener menos funcionalidades de las esperadas inicialmente.

El desarrollo de la base de datos ha ido como se esperaba.

El desarrollo de la comunicación con MQTT ha supuesto diversos problemas por la curva de aprendizaje que se ha tenido que asumir al principio, los fallos que ha habido

a la hora de configurar Node-RED y nuestra aplicación SCADA para que fuese capaz de comunicarse y pasando por el broker sin problemas.

El mayor gasto temporal y aquel que se ha desviado completamente del tiempo planificado ha sido la construcción de la cámara ambiental, el desarrollo del software para Arduino y el cableado de los actuadores con la placa de Arduino.
La construcción de la cámara ambiental ha supuesto diversos problemas puesto que ha habido muchos cambios en su diseño, más de los esperados. Estos cambios de diseño a mitad de desarrollo supusieron un coste temporal y monetario muy alto. Esto también ha ocurrido con respecto al software de Arduino, al inicio se asumió el uso de un controlador PID[17] propio para el correcto funcionamiento de los actuadores y, posteriormente, se pasó al uso de salidas digitales por la complejidad que suponía el desarrollo de un controlador PID propio y, finalmente, se volvió a utilizar un controlador PID gracias a una librería existente de Arduino que se ha podido utilizar para realizar los cálculos del controlador. Todos estos cambios han llevado a un coste temporal muy elevado.

Finalmente, la parte de testing y del cierre del proyecto ha derivado temporalmente puesto que a la hora de hacer pruebas y tests de estrés ha habido componentes electrónicos y actuadores que se han roto o han dejado de funcionar y, por lo tanto, se debían de reemplazar. La espera de que llegasen los nuevos componentes y la sustitución de los mismos ha supuesto un incremento de nuestra estimación temporal y monetaria.

Tras todos estos problemas imprevistos y cambios en el desarrollo de las funcionalidades y los diseños han llevado a que el nuevo plan de trabajo sea el siguiente:

Seguimiento de proyecto

Fecha de inicio:	03/02/2022
Fecha de finalización:	04/07/2022

Posición	Fecha de inicio	Fecha de finalización	Hito o actividad
1	03/02/2022	03/04/2022	Inicio y diseño
2	03/04/2022	06/04/2022	Adquisición de componentes
3	06/04/2022	15/04/2022	Desarrollo del SCADA
4	07/04/2022	09/04/2022	Desarrollo de la BBDD
5	11/04/2022	26/04/2022	Desarollo de MQTT
6	30/04/2022	29/05/2022	Programación del Arduino
7	30/04/2022	07/05/2022	Configuración de Node-RED
8	03/05/2022	27/05/2022	Construcción de la C.A.
9	30/05/2022	08/06/2022	Testing y cierre
10	09/06/2022	04/07/2022	Documentación y redacción

Como es visible la estimación original de trabajo y el resultado final del plan de trabajo se han desviado ampliamente, esto es debido a detalles que no se tuvieron en cuenta a la hora de estimar las horas necesarias como por ejemplo: La investigación y el estudio de las tecnologías nuevas, sus curvas de aprendizaje, los imprevistos por

[17] Un controlador PID es un mecanismo de control que a través de un lazo de retroalimentación permite regular variables de un proceso en general.

fallos eléctricos y electrónicos, las variaciones en mitad de desarrollo del diseño y la indecisión de la selección de diseños y tecnologías.

3.5. Presupuesto

El presupuesto que se ha dispuesto para este TFG ha sido separado en tres categorías, el presupuesto de Hardware, el presupuesto de Software y finalmente las horas-hombre empleadas para el desarrollo de este mismo.

El presupuesto Hardware se compone de los costes monetarios invertidos en componentes físicos, como por ejemplo:

- Sensor **x1 (3€)**
- Bomba de aire **x2 (3€)**
- Raspberry PI **x1 (30€)**
- Arduino **x1 (30€)**
- Peltiers **x3 (16€)**
- Resistencia **x1 (7.60€)**
- Humidificador **x1 (29€)**
- Silicagel **x1 (11€)**
- Mosfets **x4 (11€)**
- Fuente de alimentación **x2 (50€)**

Tras investigar un poco por internet y valorar los precios de algunos componentes, el presupuesto de hardware para el desarrollo de este Trabajo de Fin de Grado suma un total de **185,60€.**

El presupuesto software en este caso ha sido nulo, puesto que todos los elementos software utilizados para el desarrollo de este Trabajo de Fin de Grado son *open source*[18] y por lo tanto gratuitos.

Finalmente , debemos estimar el coste de horas-persona que se ha invertido en este trabajo. Como bien se ha visto en el punto anterior, el desarrollo de este trabajo, incluyendo la documentación y redacción, ha sido aproximadamente cinco meses. Asumiendo que se ha trabajado una media de entre 2 a 3 horas diarias. Consultando en internet el salario medio de un ingeniero en España, el cual ronda entre los 13 y los 16 euros por hora, se ha calculado que el coste monetario de haber desarrollado al completo este Trabajo de Fin de Grado ha sido de **5.850€.**

Si sumamos el total de costes tanto de hardware como de software y el coste monetario del trabajo del ingeniero, el presupuesto total del proyecto es de **6035,60€.**

[18] El software de código abierto es el software cuyo código fuente y otros derechos forman parte del dominio público.

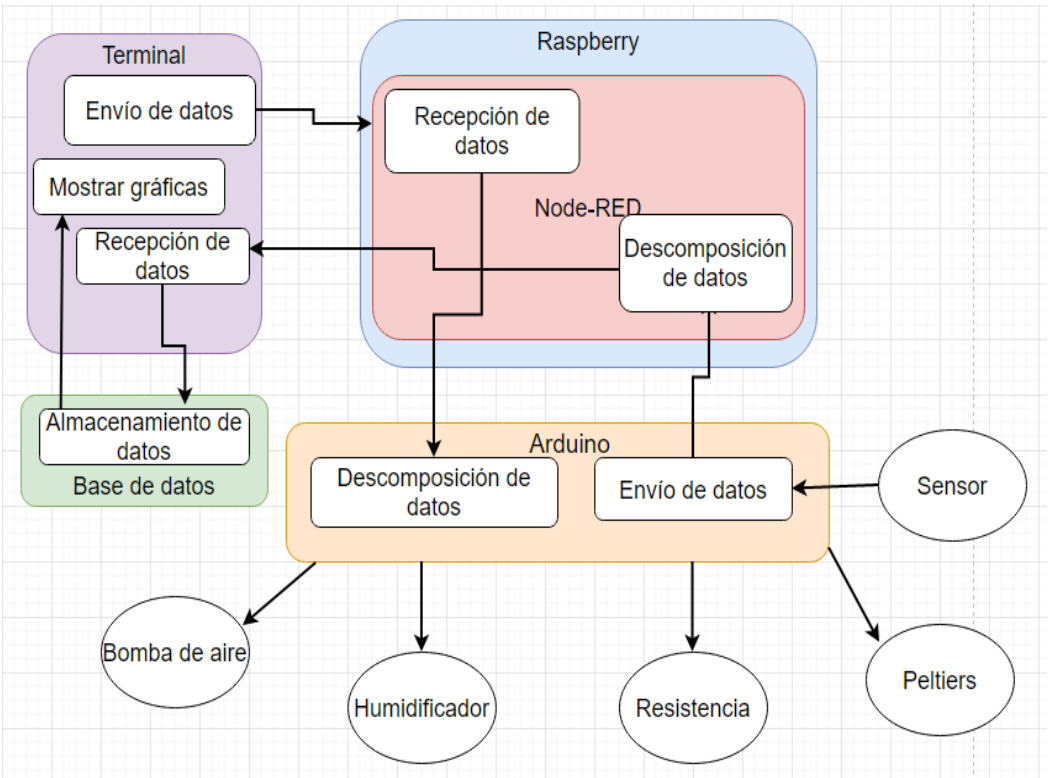
4. Diseño de la solución

A continuación veremos cómo se ha diseñado nuestra solución, empezaremos viendo la arquitectura de nuestro sistema, con imágenes y diagramas de bloques, posteriormente veremos un diseño detallado de cada uno de los bloques que hemos diseñado en nuestro diagrama de bloques y finalmente veremos las tecnologías que se han utilizado para diseñar nuestra solución.

4.1. Arquitectura del sistema

Se puede empezar hablando sobre la arquitectura que posee nuestro sistema.
Si bien nuestro proyecto consiste en crear un SCADA capaz de comunicarse *wireless*[19], también ha sido necesario el diseño y la construcción de un hardware capaz de ejecutar acciones enviadas desde el SCADA a los actuadores.

El proyecto se ha dividido en diferentes bloques diferenciados y intercomunicados como se puede ver a continuación en el siguiente diagrama de bloques:



[20] Diagrama de bloques de las distintas partes que se compone el trabajo.

[19] Wireless es una palabra del idioma inglés que puede traducirse como “sin cables” o “inalámbrico”.
[20]

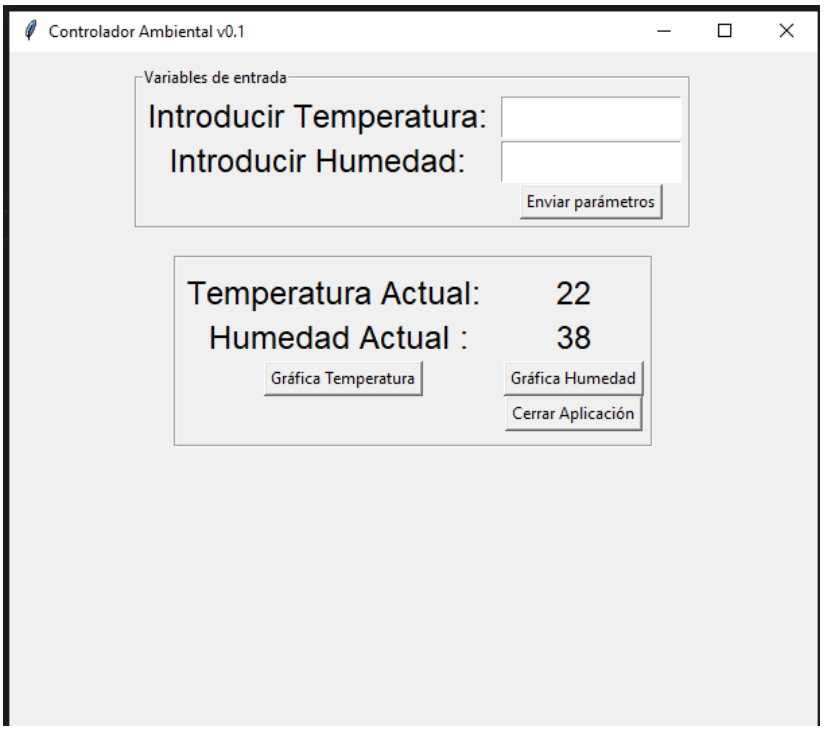
Como se puede observar en la imagen anterior, el proyecto posee diferentes bloques en los que se ha descompuesto el sistema software y hardware. Algunos de estos bloques son, por ejemplo:

- PC
- Base de datos
- Arduino
- Raspberry
- Node-RED
- Otros

4.2. Diseño Detallado

En este apartado se dispondrá a detallar, definir y mostrar cada uno de los bloques mencionados en el apartado anterior. Se comenzará mostrando las interfaces propias de cada bloque y explicando brevemente cómo funciona dicho bloque.

4.2.1. Aplicación:



Comenzando por la aplicación. Este bloque posee tres partes importantes que engloban el funcionamiento general de la aplicación y los veremos en este orden según el diagrama que hemos visualizado: “Envío de datos”, “Recepción de datos”, “Mostrar gráficas”.

Se ha diseñado esta misma con una interfaz minimalista y con la intención de que fuese simple pero eficaz.

Otras razones han sido las limitaciones de **TKinter[4] ,** la cual es una de las librerías incluidas en Python, a la hora de poder hacer buenos diseños gráficos. Sin embargo, para las intenciones de nuestro SCADA, **TKinter** ha proporcionado las herramientas necesarias para su construcción.

Como podemos observar, la aplicación posee dos partes:

1.1. Variables de entrada
1.2. Datos de salida

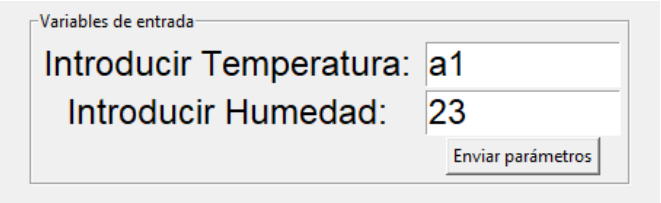
La primera sección, la cual corresponde al bloque de "Envío de datos", de Variables de entrada muestra dos campos de escritura anexos a dos campos de texto que nos indican en qué campo se debe introducir la temperatura y en qué campo se debe introducir la humedad.

Como bien se ha visto en el análisis de riesgos, la aplicación contiene posibles riesgos controlados, algunos de estos riesgos controlados son, por ejemplo:

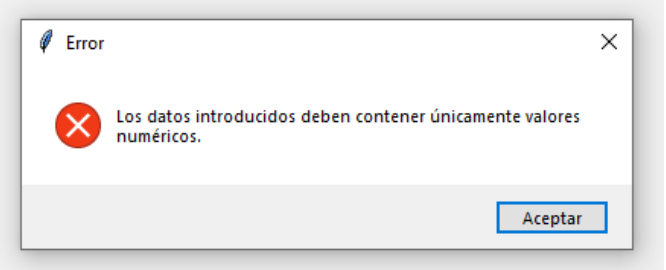
- Introducir datos NO numéricos en las variables de entrada.
- Introducir valor de temperatura por encima/debajo del rango estudiado.
- Introducir valor de humedad por encima/debajo del rango estudiado.

Podemos visualizar algunos casos donde se podría dar estos riesgos y los resultados de los mismos:

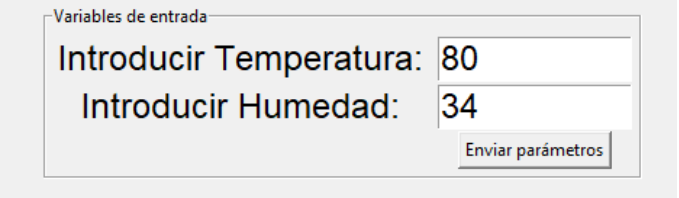
- Introducimos valores inválidos en las variables de entrada.



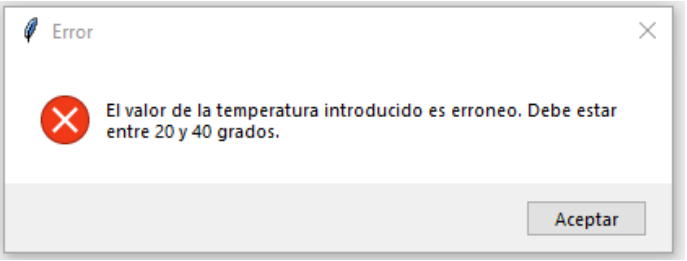
El resultado de introducir estos valores es una ventana emergente como la siguiente:



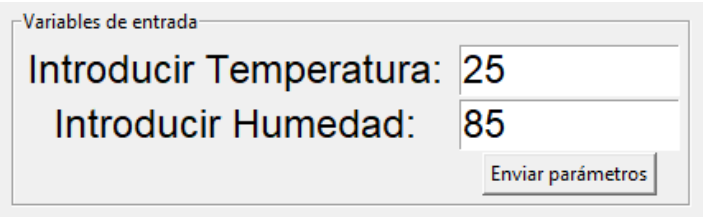
- Introducimos valores de temperatura por encima o por debajo del rango estudiado.



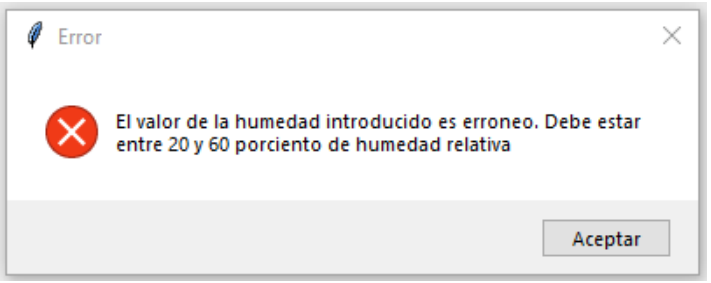
El resultado de introducir estos valores es una ventana emergente como la siguiente:



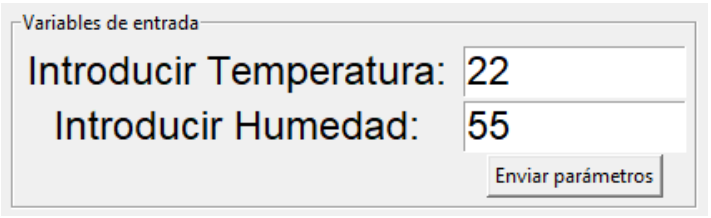
- Introducimos valores de humedad por encima o por debajo del rango estudiado.

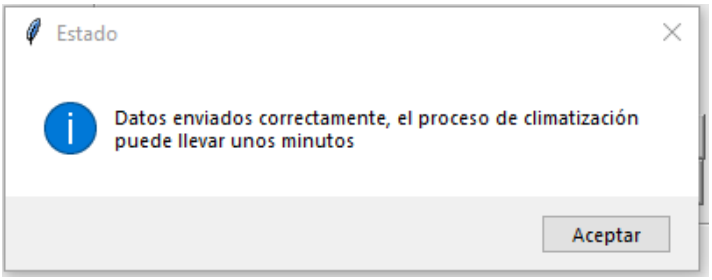


El resultado de introducir estos valores es una ventana emergente como la siguiente:

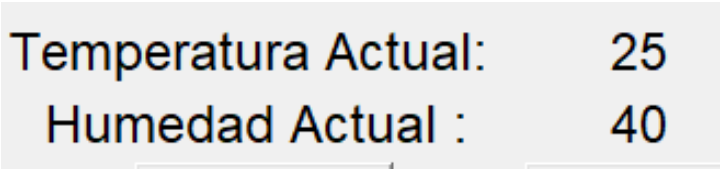


Si los datos introducidos son correctos, para tener una aplicación que nos permita tener un feedback visual de que el envío de los datos ha sido correcto, la aplicación nos mostrará una ventana emergente con un mensaje informando de que los datos se han enviado correctamente, como en el siguiente caso:





La segunda sección que contiene la aplicación contiene dos partes importantes.
La primera parte es la visualización en tiempo real del sensor de la cámara ambiental.

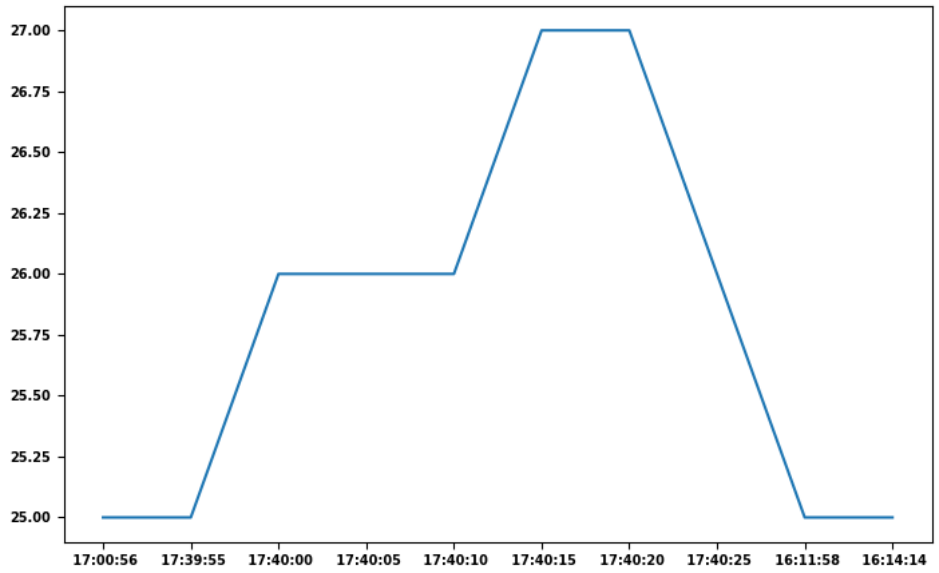


Estos datos se actualizan cada cinco segundos y nos permiten visualizar el valor de estas variables dentro de la cámara.

La segunda parte es una serie de botones y corresponde con el bloque de "Recepción de datos" y "Mostrar gráficas". Los botones de "Gráfica Temperatura" y "Gráfica Humedad" nos permiten visualizar los valores de temperatura y humedad del último minuto y cuarenta segundos.

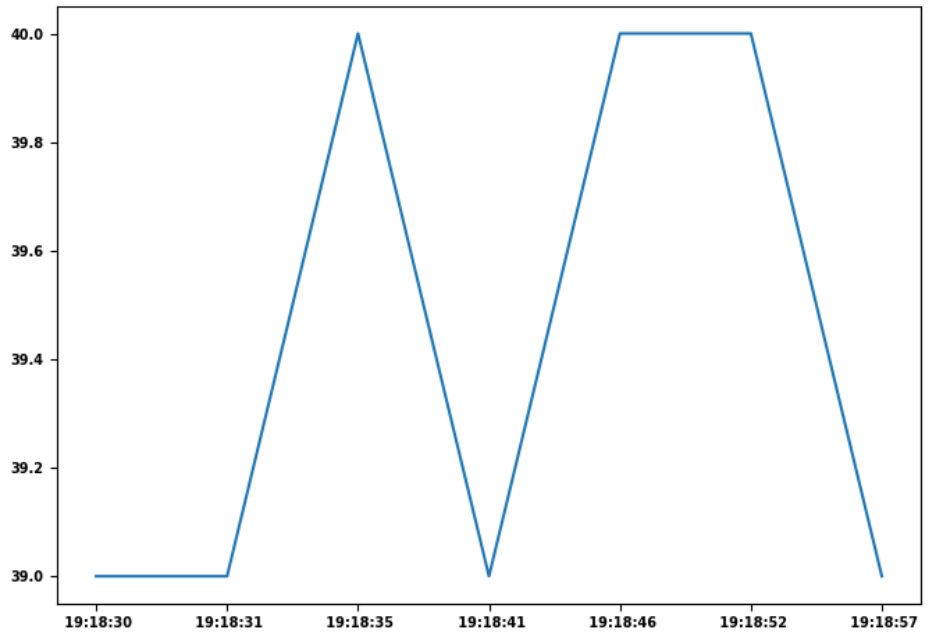
El último botón "Cerrar Aplicación" termina la conexión con la cámara ambiental y cierra la aplicación.

Si se acciona el botón de “Gráfica Temperatura” se mostrará el valor de la temperatura en forma de una ventana emergente con una gráfica, donde el valor del eje Y es el valor de la temperatura y el valor en el eje X es el valor de la hora en la que se ha tomado la temperatura:



[21] Gráfica generada por nuestra aplicación que muestra el intervalo de tiempo y los grados registrados.

Si se acciona el botón de “Gráfica Humedad” se mostrará el valor de la humedad en forma de una ventana emergente con una gráfica, donde el valor del eje Y es el valor de la humedad y el valor en el eje X es el valor de la hora en la que se ha tomado la humedad:



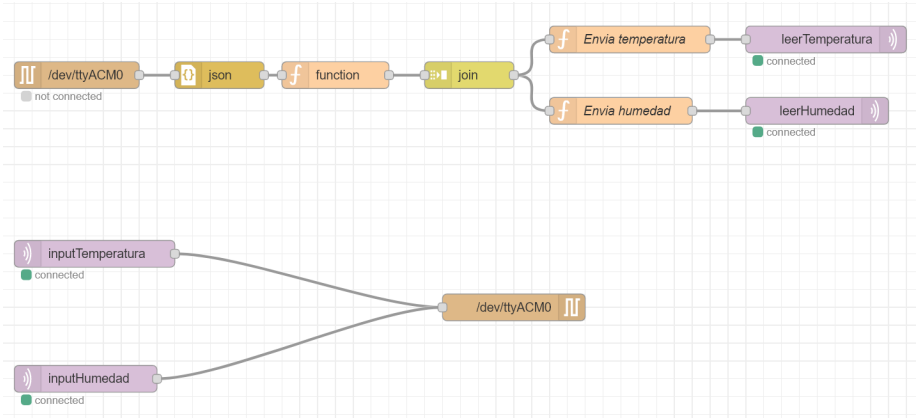
[22]Gráfica generada por nuestra apllicación que muestra el intervalo de tiempo y el porcentaje de humedad relativa

21

22

4.2.2. Raspberry/Node-RED:

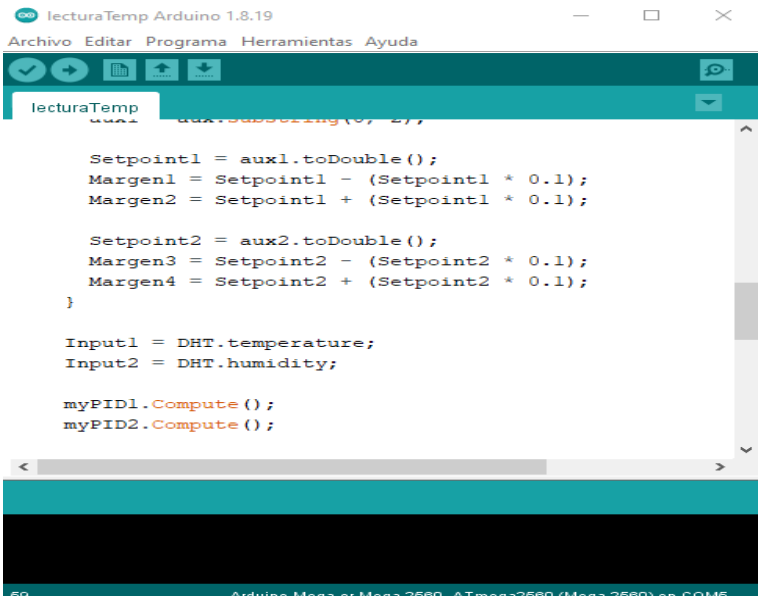
Este bloque engloba la recepción de la información por MQTT recibida del PC utilizando nodos propios y nodos importados de las librerías propias de Node-RED.



El flujo de nodos superior es el flujo de datos que recibe la información del sensor a través de un nodo de puerto serie y se envía esta misma información a través de un nodo MQTT de envío a nuestro PC.

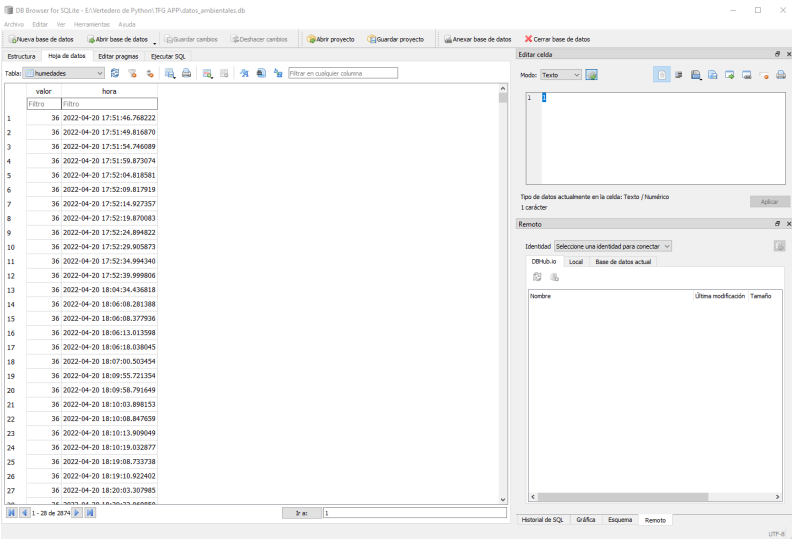
El flujo de nodos inferior es el flujo de datos que recoge la información recibida por el nodo MQTT.

4.2.3. Arduino:



Este bloque pertenece a la placa de Arduino. Esta sección tiene una serie de mecanismos de control y de cálculos que permite el control de la temperatura y la humedad así como de el envío por puerto en serie de la información captada por el sensor.

4.2.4. Base de datos:

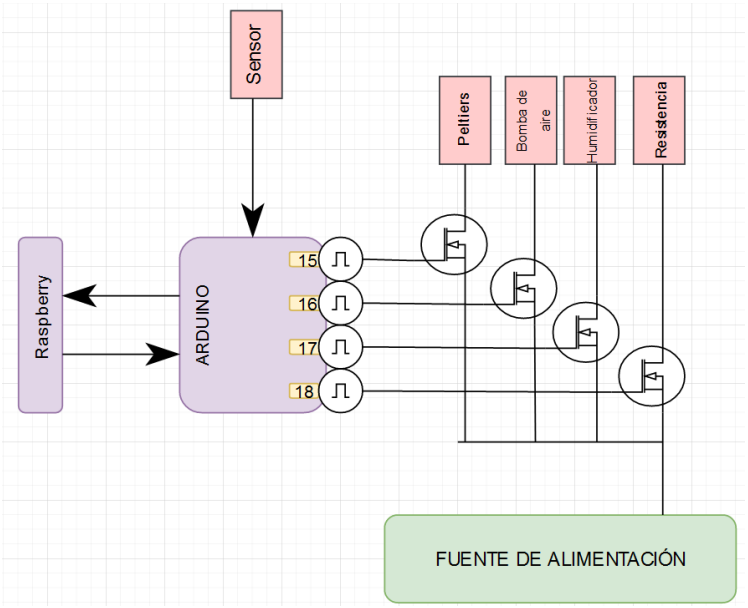


:

Este bloque define la base de datos que se utiliza para guardar los datos cuando el SCADA recibe los datos. También permite la visualización y la consulta de datos cuando en la aplicación SCADA se quiere mostrar gráficas de temperatura y humedad.

4.3. Tecnología Utilizada

A continuación se explican algunas características de las tecnologías que hemos utilizado para el desarrollo del proyecto y daremos varias razones de porqué hemos seleccionado ciertas tecnologías y no otras. Este capítulo también incluirá las tecnologías y componentes electrónicos pertenecientes al hardware conectado a nuestra placa Arduino.



- Se empezará hablando por la componente de la aplicación SCADA:

  Una de las principales características por las que se ha terminado eligiendo Python por encima de Java es la facilidad que posee Python para la visualización de datos. Pese a que ambas tecnologías están orientadas a objetos y tienen muchas librerías sobre las que se puede trabajar para construir aplicaciones, para el objetivo de nuestra aplicación SCADA, que era la comunicación por protocolo de mensajería MQTT y la posibilidad de visualizar datos, Python ha sido la opción elegida.

  El tratamiento de datos y la facilidad de programación que posee Python ha sido más cómoda a la hora de desarrollar diferentes utilidades. Su lenguaje dinámico y sus librerías de capacidad de visualización de datos, como Matplotlib[5] o Seaborn[6], han sido los factores decisivos a la hora de desarrollar la aplicación.

- Base de datos:

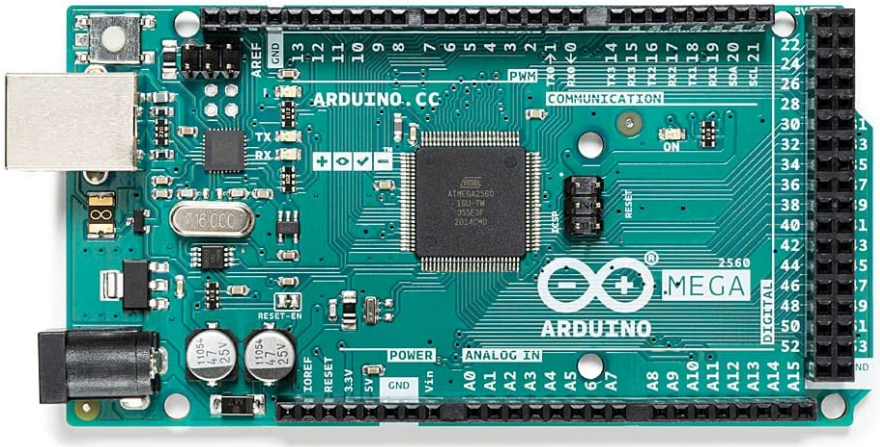
  Para la base de datos se podía haber utilizado MySQL Server o PL/SQL, sin embargo, la configuración de la base de datos podría haber sido un proceso complejo y con un coste temporal tanto de configuración como de investigación para configurar la misma.

  SQLite no necesita ser configurada y permite ser creada de forma dinámica desde el propio código de Python utilizando la librería de SQLite3, la cual funciona de forma similar a hacer scripts[23] en PL/SQL y , dada la propia experiencia profesional del alumno en scripts de bases de datos, utilizar sus herramientas ha sido lo que ha marcado la decisión de utilizar SQLite.

[23] Un script es una secuencia de instrucciones utilizada para designar a un programa relativamente simple

- Arduino:

  Arduino ha sido la tecnología que se ha decidido utilizar tanto para pasar información como para realizar los cálculos necesarios para accionar los actuadores. Arduino es una plataforma electrónica open-source programada en el lenguaje de programación C que nos permitía comunicarnos, gracias a sus librerías, por puerto en serie con nuestra Raspberry y el uso de salidas digitales ha permitido conectar los distintos actuadores con sus respectivas fuentes de alimentación limitando su activación con MOSFETs[24]. La principal razón por la que se ha elegido este modelo de Arduino es porque ya se poseía esta tecnología previamente al desarrollo del proyecto, por lo que ha supuesto una reducción muy importante del presupuesto económico. Además de ser una reducción del presupuesto económico, esta placa de Arduino posee todas las entradas necesarias para el funcionamiento de los dispositivos puesto que posee entradas digitales y entradas analógicas y otro tipo de entradas.

  El modelo de Arduino utilizado para este proyecto ha sido el modelo de Arduino Mega 2560.

[24] El MOSFET es un transistor utilizado para amplificar o conmutar señales electrónicas

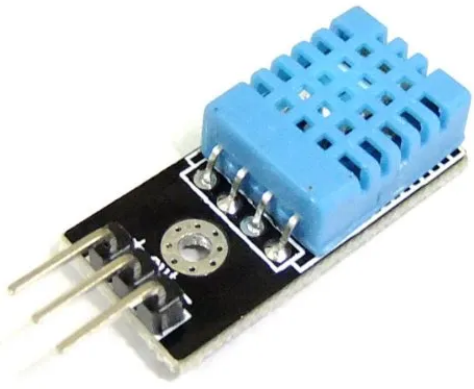
- MOSFETs:

  Como bien se ha comentado en el apartado anterior, hemos utilizando una serie de MOSFETs para limitar el paso de la corriente voltaica a los actuadores correspondientes desde sus respectivas fuentes de alimentación.

  Para este proyecto, el modelo utilizado de estos MOSFETs ha sido el modelo IRFZ44N.

- Sensor:

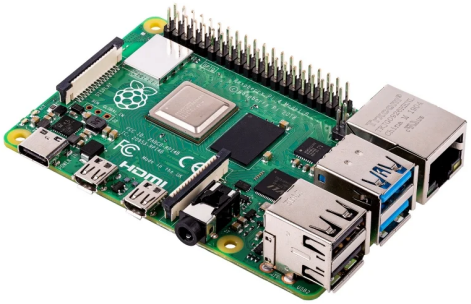
  El modelo del sensor utilizado ha sido el modelo DHT 11. No ha sido elegido por tener un alto porcentaje de fiabilidad, ha sido utilizado porque esta tecnología y este modelo ya se poseía antes de comenzar a construir el proyecto y, para escatimar en gastos de presupuesto de componentes hardware, se ha utilizado este mismo.



- Raspberry PI:

  La utilización de una Raspberry PI para este proyecto era la alternativa más económica a la utilización de un servidor para distribuir datos a distintas cámaras ambientales y , además, era la opción más cómoda por varias razones.

  La razón principal para la utilización de una Raspberry PI ha sido la posibilidad de utilizar tecnologías de protocolos de comunicación y la instalación de otras tecnologías como por ejemplo Node-RED. Otra de las razones para la utilización de Raspberry PI ha sido la opción económica puesto que ya se poseía esta tecnología previamente al desarrollo de este proyecto.

  El modelo utilizado de Raspberry ha sido Raspberry PI 4 modelo B.

- Peltiers:

  La utilización de células peltiers para poder actuar como refrigerante y bajar la temperatura ha sido una decisión puramente económica. Si bien hay otros métodos que utilizan anticongelantes y diferentes gases criogénicos, para poder plantear un modelo más económico, se han utilizado tres células peltiers que refrigeran el aire.

  El modelo utilizado de células peltiers ha sido el modelo TEC1-12705.

- Resistencia:

  La utilización de una resistencia como elemento calorífico ha sido, al igual que las células peltiers, un motivo económico y fácil de utilizar.

  El modelo utilizado de resistencia para aumentar la temperatura ha sido el modelo PTC CA DC 12V.

- Bomba de aire:

  Para poder reducir la humedad de dentro de nuestra cámara ambiental, se ha utilizado una bomba de aire. Esta bomba de aire extrae la humedad del interior de nuestra cámara ambiental y la lleva a un compartimento relleno de bolsas de gel de sílice[25] que a su vez el aire de esta cámara es devuelto a la cámara ambiental. Estas bolsas absorben la humedad que haya en el ambiente y se pueden calentar para poder expulsar la humedad que hayan absorbido.

  Ha habido una reducción económica en nuestro proyecto puesto que ambos componentes, tanto las bolsas de sílice como la bomba de aire, han sido baratos.

  Además, haciendo un modelo de reducción de humedad de esta forma, evitamos el posible efecto que tenga la humedad ambiente en nuestra cámara ambiental.

  El modelo utilizado de la bomba de aire ha sido el modelo Micro 370 Air Pump.

[25] El gel de sílices es un desecante que sirve para preservar y proteger de la humedad los productos tecnológicos

- Humidificador:

  Para poder aumentar la humedad ambiente se ha utilizado un humidificador el cual contiene en su interior una cápsula en la que se almacena el agua de forma segura y al accionarse expulsa vapor de agua dentro de la cámara ambiental. Este es el componente y actuador electrónico más caro del proyecto pero, no obstante, ha ofrecido unos resultados muy satisfactorios.

  El modelo utilizado del humidificador ha sido el modelo Beurer Mini Humidificador LB 12.

- Mosquitto:

  Mosquitto[7] es un broker de mensajería MQTT open-source que permite la fácil comunicación y distribución de mensajes ligeros y con poca capacidad de potencia, este broker ha sido seleccionado por su compatibilidad con MQTT y además por ser sencillo de utilizar y no necesitar configuración de ningún tipo ni programación.

5. Desarrollo de la solución propuesta

Ahora que ya hemos visto el diseño de la solución y las tecnologías que han sido utilizadas para el desarrollo de estas mismas, vamos a describir y a explicar paso a paso el desarrollo de la que ha sido la solución propuesta utilizando las tecnologías enseñadas y explicadas en los apartados anteriores.

Para nuestro desarrollo empezaremos explicando y mostrando cómo se ha desarrollado la aplicación SCADA y también explicaremos el desarrollo de la base de datos. Posteriormente explicaremos el desarrollo de la parte de Node-RED y sus distintos nodos y utilidades. Seguidamente explicaremos la parte del software de Arduino y finalmente, por su importancia a la hora de hacer las pruebas, explicaremos cómo se ha diseñado la electrónica de la cámara ambiental.

1. Aplicación:

Para nuestra aplicación utilizamos como base la librería de construcción de interfaces de Python, TKinter. Esta librería utiliza un sistema propio de posicionamiento por rejilla, lo que significa que divide una ventana en filas y columnas.

Utilizando la API[8] de TKinter podemos ver los distintos métodos que contiene la librería.

Empezando por la inicialización de la ventana de la aplicación. Podemos inicializar una ventana y darle un título utilizando el método siguiente:

```
app = Tk()
app.title("Controlador Ambiental v0.1")
```

A continuación creamos los recuadros que envuelven a las distintas secciones de nuestra aplicación.

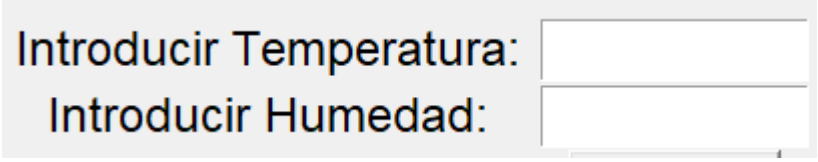
```
frame =  LabelFrame(app,text="Variables de entrada",padx=5,pady=5)
frame.pack(padx=10,pady=10)
frameBotonera = LabelFrame(app,padx=5,pady=10)
frameBotonera.pack(padx=10,pady=10)
```

Para poder visualizar correctamente los datos en las gráficas y que no haya conflictos ni solapes, hemos incluido un diccionario llamado *font* con características incluidas por nosotros mismos:

```
#Prerrequisitos
font = { 'weight' : 'bold',
         'size': 7}
plt.rc('font',**font)
```

A la hora de crear los cuadros de texto, hemos implementado la función de Entrada, se han creado dos cuadros de texto utilizando y las hemos implementado en el recuadro de las variables de entrada.

```
#Inputs de temperatura y Humedad
inputTemperatura = Entry(frame,font=("Arial",18),width=10)
inputHumedad = Entry(frame,font=("Arial",18),width=10)
```



Estos cuadros de texto son importantes porque, posteriormente, las comprobaciones que se han implementado para abarcar los análisis de riesgos encontrados al principio.

Se han utilizado etiquetas para las diferentes denominaciones de “Introducir temperatura” , “Introducir humedad” y para las denominaciones de “Temperatura” y “Humedad” de las variables en tiempo real.

```
temp_label = Label(frame, text="Introducir Temperatura:
",font=("Arial",18),compound="center")
humid_label = Label(frame, text="Introducir Humedad:
",font=("Arial",18),compound="center")
temp_lec_label2 = Label(frameBotonera,text= "Temperatura Actual:
",font=("Arial",18),compound="center")
humid_lec_label2 = Label(frameBotonera,text= "Humedad Actual :
",font=("Arial",18),compound="center")
temp_lec_label = Label(frameBotonera,text= "
",font=("Arial",18),compound="center")
humid_lec_label = Label(frameBotonera,text= "
",font=("Arial",18),compound="center")
```

Para generar las gráficas de humedad y temperatura, para salir de la aplicación y para enviar los datos a nuestra cámara ambiental, hemos necesitado generar botones.

```
enviarDatos = Button(frame,text="Enviar parámetros",
command=sendData,compound="center")
graficaTemperatura = Button(frameBotonera,text="Gráfica Temperatura",
command=generateTemp)
graficaHumedad = Button(frameBotonera,text="Gráfica Humedad",
command=generateHumi)
salir = Button(frameBotonera,text="Cerrar Aplicación",
command=funcSalir)
```

Como podemos ver, estos botones tienen una función asignada en el parámetro *command* esto implica que al pulsar el botón, las funciones con ese nombre se ejecutarán. Esto es importante

puesto que hasta que no se pulsen los botones, todas las instrucciones que contengan estas funciones estarán a la espera.

Empezando por la instrucción más importante, la función de enviar datos. Esta función comprueba utilizando una serie de mecanismos de control si el cuadro de temperatura o el cuadro de humedad es numérico:

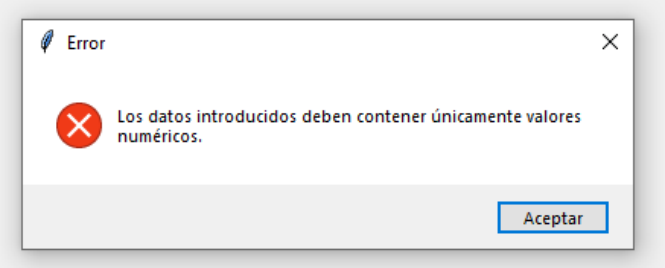
```
if(inputTemperatura.get().isnumeric()                                    and
inputHumedad.get().isnumeric()
```

Para evitar posibles errores de detección de valores nulos, además de saber si los valores introducidos son números, hemos añadido otra comprobación para ver si se ha añadido algo al cuadro de texto:

```
len(inputTemperatura.get())>0 and len(inputHumedad.get())>0
```

Si se inclumple alguna de estas condiciones la aplicación mostrará un mensaje de error

```
messagebox.showerror("Error","Los  datos  introducidos  deben  contener
únicamente valores numéricos.")
```



Este caso es analizado en el apartado de análisis de riesgos.

Si las condiciones anteriormente mencionadas se cumplen, encontramos otro mecanismo de control que comprueba si el valor numérico de los datos introducidos en los cuadros de texto está en el rango de valores que hemos analizado en los tests:

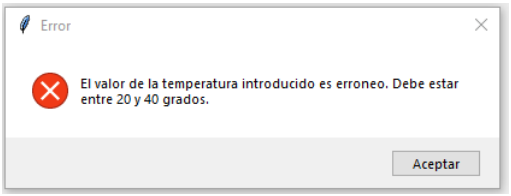
```
int(inputTemperatura.get()) < 20 or int(inputTemperatura.get()) > 40 or
int(inputHumedad.get())>60 or int(inputHumedad.get())< 20
```

Si alguna de estas comprobaciones se cumple entramos al último mecanismo de control, el cual su función es comprobar dónde ha estado el valor mal introducido, si en el cuadro de texto de la humedad o en el cuadro de texto de la temperatura:

```
int(inputTemperatura.get()) < 20 or int(inputTemperatura.get()) > 40
```

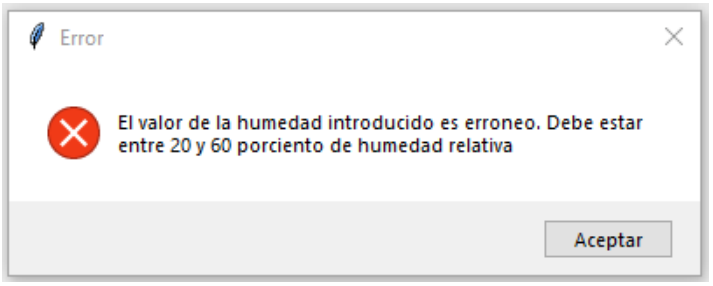
Si éstas condiciones se cumplen mostrará un mensaje de error ya controlado en el análisis de riesgos:

```
messagebox.showerror("Error","El valor de la temperatura introducido es
erróneo. Debe estar entre 20 y 40 grados.")
```



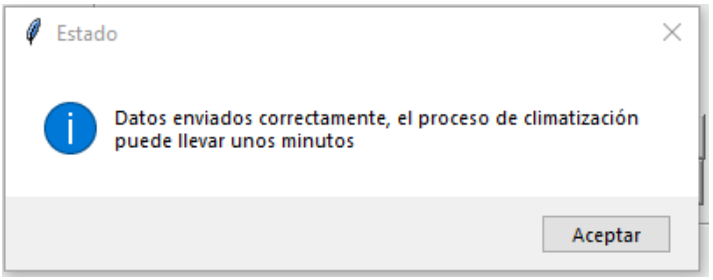
Si las condiciones anteriores no se cumplen entonces el error ha sido en la humedad y no en la temperatura y mostrará un mensaje de error ya controlado en el análisis de riesgos:

```
messagebox.showerror("Error","El valor de la humedad introducido es erróneo. Debe estar entre 20 y 60 porciento de humedad relativa")
```



Si no se cumple ninguna de las condiciones mencionadas en los mecanismos de control mencionados anteriormente, entonces los datos introducidos son correctos y podremos enviarlos correctamente:

```
client = paho.Client("controlTemperaturaHumedad")
client.connect("192.168.1.105",1883)
client.publish("inputTemperatura",int(inputTemperatura.get()))
client.publish("inputHumedad",int(inputHumedad.get()))
messagebox.showinfo("Estado","Datos enviados correctamente, el proceso de climatización puede llevar unos minutos")
```



Utilizando los métodos que se pueden consultar en la API de Eclipse Paho conectamos nuestro cliente (el PC) a nuestro broker Mosquitto introduciendo la dirección IP correspondiente.

Hemos analizado la documentación ofrecida por el broker de Mosquitto y hemos seleccionado, para facilitar el paso de información en este proyecto, el puerto 1883,

principalmente porque este puerto tiene las características de utilizar protocolo MQTT, los mensajes que pasan a través de él no se encriptan y no necesitan tampoco estar autentificados.

Una vez nos conectamos al broker utilizando el método:

```
client.connect("192.168.1.105",1883)
```

Debemos enviarle la información, para ello utilizamos los métodos de:

```
client.publish("inputTemperatura",int(inputTemperatura.get()))
client.publish("inputHumedad",int(inputHumedad.get()))
```

Estos métodos publican en el broker unos datos junto con un tema. Esto es muy importante puesto que por el tipo de mensajería publicador/suscriptor, si un suscriptor no se suscribe al mismo tema que ha publicado el publicador, nunca recibirá el mensaje correspondiente.

Con estas instrucciones enviamos al broker los valores numéricos de los cuadros de texto que hemos comentado anteriormente de temperatura y humedad.

El siguiente método que vamos a comentar es el método de generar la gráfica de temperatura y humedad. Este método empieza conectándonos a la base de datos de SQLite.

```
conexion = sqlite3.connect('datos_ambientales.db')
```

Al igual que en las bases de datos comunes, si queremos obtener una grán cantidad de datos pero seleccionandolos uno a uno, utilizamos cursores[26]. Para ello utilizamos el método de cursores que posee la librería de sqlite3:

```
c = conexion.cursor()
```

Ahora que tenemos un cursor, debemos obtener los datos, para ello podemos, utilizando el cursor, ejecutar una instrucción a nuestra base de datos:

```
c.execute("SELECT * FROM ( SELECT valor, hora FROM temperaturas ORDER
BY hora DESC LIMIT 20) ORDER BY hora ASC;")
```

La instrucción “execute” ejecuta una instrucción que se le añade entre paréntesis sobre una hoja de trabajo SQL en la base de datos sobre la que hayamos creado el cursor.

Nuestra instrucción SQL en este caso obtiene, de la tabla de temperatura, los veinte últimos registros y los ordena por la hora de forma ascendente. Esto se realiza de esta forma puesto que a la hora de ordenar los datos en la gráfica debemos rellenar los valores de más antiguos a más nuevos..

Ahora que ya hemos obtenido los datos y están almacenados en el cursor, procedemos a moverlos a una nueva variable sobre la que trabajaremos para, posteriormente, separar los datos de tiempo y los valores de temperatura. Puesto que ya no necesitamos la base de datos para nada, podemos cerrar tanto el cursor como la conexión a la base de datos utilizando las instrucciones siguientes:

```
    valores = c.fetchall()
```

[26] Un cursor, en el ámbito de bases de datos, es una estructura de control utilizada para guardar y recorrer los registros del resultado de una consulta.

```
    sorted(valores,key = lambda x: x[1])
    conexion.commit()
    conexion.close()
```

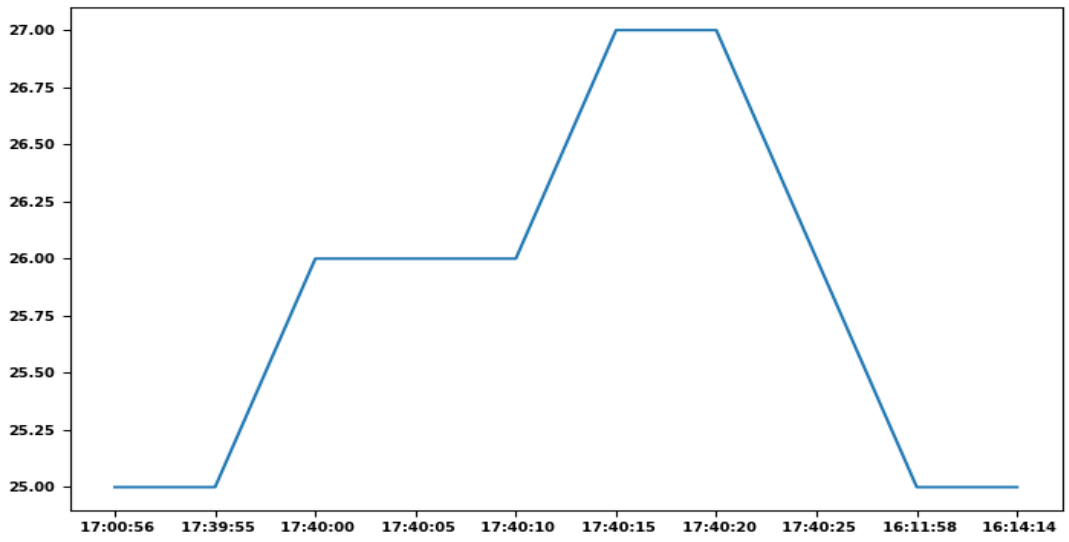
Ahora nuestros datos están en una variable llamada “valores”, la peculiaridad de esta variable es que es un diccionario[27], donde la *key* es el valor de la temperatura y el *value* es la fecha y hora completa a la que se ha registrado dicho valor de la temperatura. Para poder generar la gráfica es necesario separar los datos en dos variables diferentes donde podamos guardar los datos para el eje Y y el eje X de la gráfica. De modo que utilizamos el conjunto siguiente de instrucciones:

```
    valoresTemperatura,valoresHorarios = [],[]
    for a,b in valores:
        valoresTemperatura.append(a)
        valoresHorarios.append(b[10:19])
```

Estas instrucciones nos devuelven en una variable llamada “valoresTemperatura” los valores de temperatura y en la variable “valoresHorarios” nos devuelven los valores de la hora, minutos y segundos que se ha registrado ese valor de temperatura, es importante esto puesto que el valor original del tiempo que se guarda en el diccionario de “valores” contiene tambien el dia, el mes y el año.

Ahora utilizando las instrucciones propias de la librería de matplotlib creamos las gráficas utilizando las siguientes instrucciones:

```
    plt.figure(figsize=(15,8))
    plt.plot(valoresHorarios,valoresHumedad)
    plt.show()
```



La función de generar la gráfica de humedad es similar a la que acabamos de ver, la principal diferencia son los nombres de las variables y la consulta a la base de datos no se realiza sobre

[27] Un Diccionario es una estructura de datos que nos permite almacenar cualquier tipo de valor

la tabla de temperaturas sino sobre la tabla de humedad. Por esta razón, no hará falta analizar el código fuente de este método.

Ahora que hemos visto cómo se generan las gráficas y cómo se envían los datos, vamos a ver cómo se reciben los datos y se guardan los mismos en la base de datos.

Nuestra aplicación es, a su vez, suscriptor y publicador, por lo que debemos suscribirnos a un tema para poder recibir mensajes, para ello hemos utilizado una serie de instrucciones propias de Eclipse Paho.

```
subscriber = paho.Client("leerTemperaturaHumedad")
subscriber.connect("192.168.1.105",1883)
subscriber.subscribe([("leerTemperatura",2),("leerHumedad",2)])
```

Estas instrucciones nos permiten crear un cliente llamado "leerTemperaturaHumedad". Dicho cliente se encargará de conectarse al broker y suscribirse a los temas de "leerTemperatura" y "leerHumedad". No obstante, suscribirnos a un tema y recibir un mensaje no sirve de nada si no ocurre algo cuando nos llega el mensaje, por lo que, en nuestro caso, hemos añadido al final dos instrucciones extras que se encargan de que, cuando llegue un mensaje con el tema que nosotros seleccionemos, llame a la función que se indique. Para ello, Paho, nos ofrece las siguientes instrucciones:

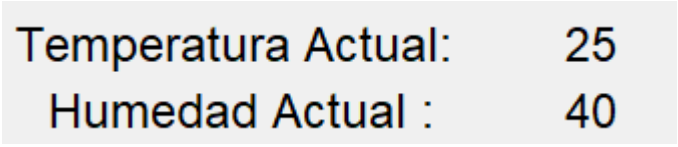
```
subscriber.message_callback_add("leerTemperatura",
on_message_temperatura)
subscriber.message_callback_add("leerHumedad", on_message_humedad)
```

Cuando llega un mensaje con el tema de "leerTemperatura", se ejecutará la función "on_message_temperatura" y cuando llegue un mensaje de "leerHumedad", se ejecutará la función "on_message_humedad".

Como ambas funciones comparten un comportamiento similar y sus funcionalidades son muy similares, solamente explicaremos la función de "on_message_temperatura", puesto que la única diferencia entre ambos métodos es la tabla donde se inserta el dato.

La función de "on_message_temperatura" es una función callback[28], esta función nos permite principalmente almacenar en la base de datos el conjunto de datos que ha llegado por MQTT. No obstante, se ha de recordar que nuestra interfaz muestra en tiempo real los datos de temperatura y humedad que tiene la cámara ambiental, por lo que habrá que actualizar los valores que se muestran, para ello hemos configurado la siguiente instrucción, que descodifica el mensaje recibido para poder introducirlo en la etiqueta de la temperatura:

```
humid_lec_label.config(text=message.payload.decode("utf-8"))
```



[28] En programación, un callback es una función que se ejecuta cuando algún evento sucede o después de que un código llega al estado deseado.

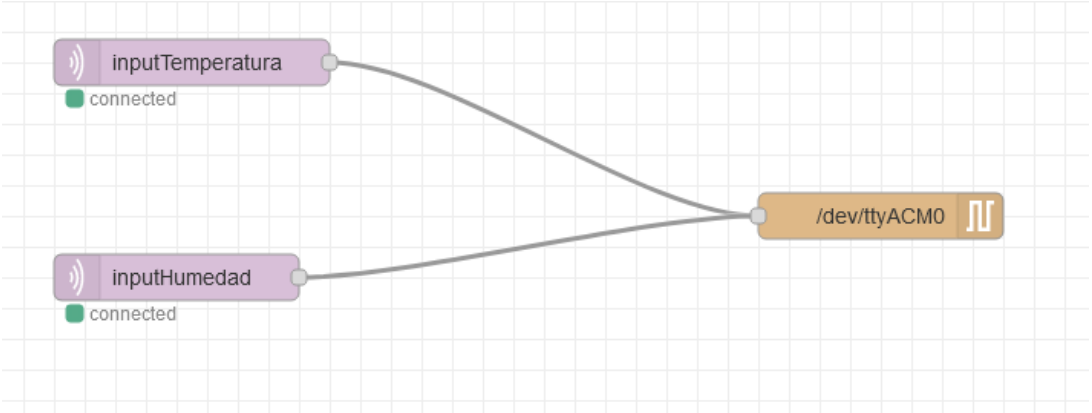
A continuación, debemos guardar los valores en la base de datos, para ello, al igual que antes, abriremos una conexión, utilizando la librería de SQLite, con la base de datos y tambien abriremos un cursor sobre el que ejecutaremos las siguientes instrucciones:

```
conexion = sqlite3.connect('datos_ambientales.db')
c = conexion.cursor()
c.execute("INSERT INTO temperaturas VALUES (:valor,:hora)",
          {
              'valor': message.payload.decode("utf-8"),
              'hora': datetime.datetime.now()
          }
          )
conexion.commit()
conexion.close()
```

Estas instrucciones nos permiten ejecutar en la base de datos la instrucción para insertar, en la tabla de temperatura, los datos del valor de la temperatura y la hora actual en la que se registra la temperatura. Finalmente procedemos a cerrar la conexión y con ello el cursor.

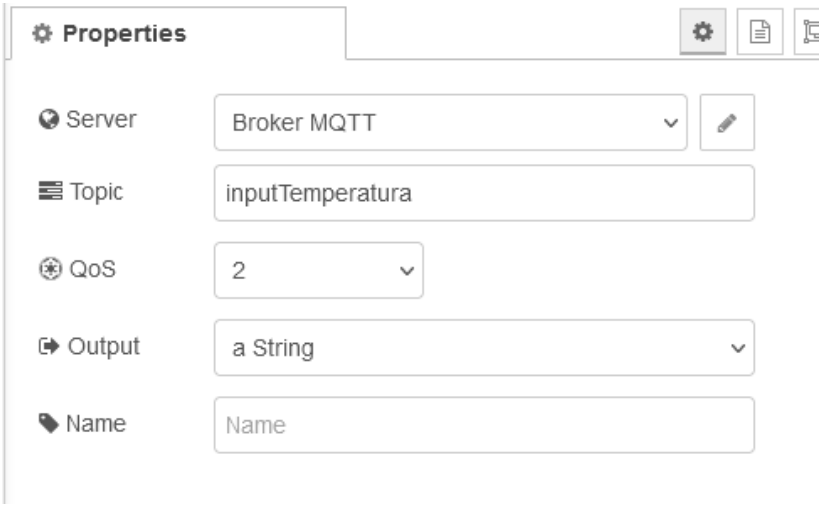
2. Node-RED:

A continuación veremos la configuración de nodos MQTT de Node-RED para la recepción de datos.

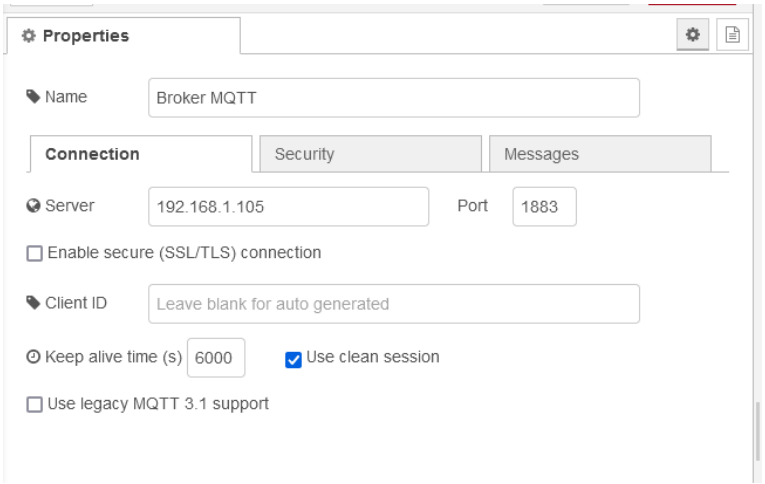


Estos nodos mostrados en la imagen superior son los nodos de conexión MQTT y el nodo de conexión por puerto de serie. Ambos tipos de nodos son nodos integrados con la paleta de nodos básicos de Node-RED.

Los nodos de la izquierda son los nodos MQTT, estos nodos se encargan de, mediante la conexión por IP a nuestro broker Mosquitto, suscribirse a cualquier mensaje que aparezca en su tópico. La configuración interna de ambos nodos es la siguiente:



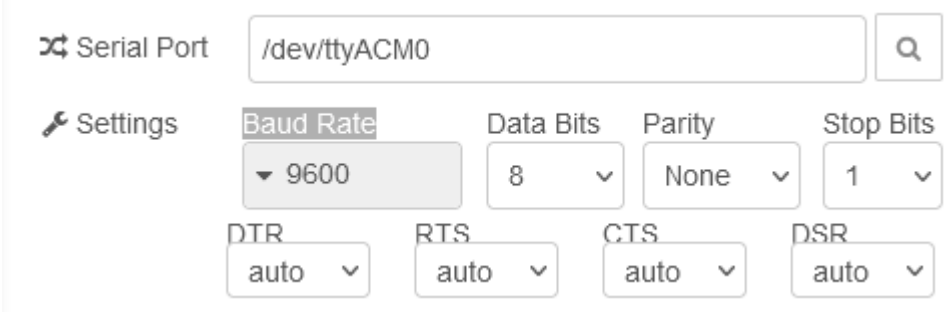
Como podemos ver, el *Topic* al que está suscrito este nodo es el mismo tema al que publicamos desde nuestra aplicación como hemos visto anteriormente. No obstante, debemos configurar nuestra conexión de servidor a nuestro broker Mosquitto. Para ello, podemos acceder a la configuración siguiente del nodo:



La configuración de nuestro servidor al cual se conectan nuestros nodos es la configuración IP y puerto de nuestro broker MQTT coincide con la misma a la que publicamos y nos suscribimos en nuestra aplicación puesto que debemos entregarle los mensajes a nuestro broker y que este haga las gestiones necesarias para recibir y enviar los mensajes.

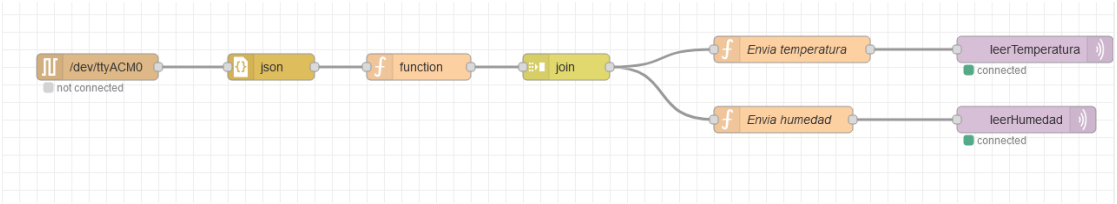
Los mensajes que se reciben en este nodo se envían directamente al nodo de salida del puerto de serie, por lo que, debemos configurar la salida del puerto de serie.

Para configurar el puerto de serie, debemos acceder a las propiedades del nodo y seleccionar la misma “tasa de baudios”[29] que deberemos coincidir posteriormente con nuestro arduino:

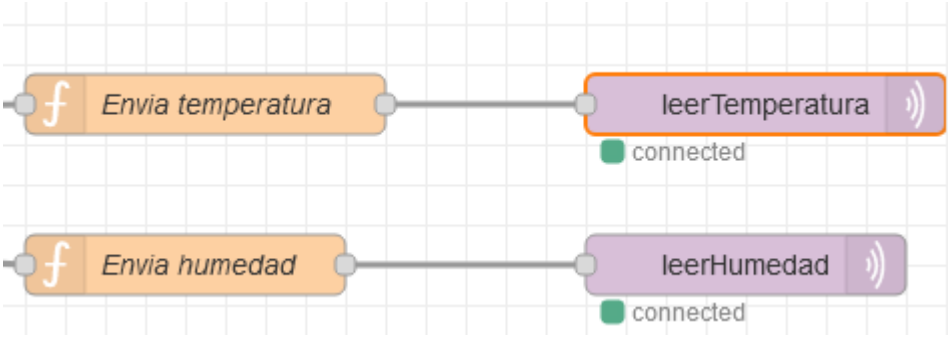


Este factor es importante, puesto que a una tasa diferente de baudios, los mensajes no llegan correctamente al puerto de serie y viceversa.

Ahora vamos a ver la recepción de mensajes desde Arduino a Raspberry y cómo se envía la información de nuevo a nuestra aplicación.



La recepción de mensajes comienza con un nodo de puerto de serie el cual posee las mismas propiedades que el nodo mencionado anteriormente para la recepción de mensajes enviados desde nuestra aplicación. Una vez recibido el mensaje, se trata, convierte y separa el mensaje de forma que podamos realizar un envío de mensajes por separado para los distintos temas que nuestra aplicación SCADA se ha suscrito.



En estos nodos la información de los mensajes se extrae y se envía por conexión MQTT utilizando los nodos de envío MQTT y, explícitamente, indicando cuales son los tópicos a los que se va a suscribir cada uno de estos nodos.

[29] La tasa de baudios es el número de unidades de señal por segundo.Un baudio puede contener varios bits.

Como se puede visualizar, son los mismos tópicos que nuestra aplicación se ha suscrito y ha indicado que, cuando se recibiesen mensajes con estos tópicos, se accionen nuestros métodos callback, los cuales hemos explicado previamente.

Ahora vamos a ver la configuración de la recepción de mensajes por puerto de serie. La transmisión de información o datos por puerto de serie funciona enviando información bit a bit, uno tras otro, y esto se aplica tanto a la hora de enviar datos como a la hora de recibirlos. En nuestro caso esto se aplica enviando la información a la placa de Arduino cuando enviamos los valores de temperatura y humedad.

En nuestro caso, y según la documentación oficial de la página de arduino, las conexiones entre un Arduino y un ordenador se hacen por cable USB y, como bien hemos comentado anteriormente, la Raspberry PI es un ordenador en miniatura, por lo que la comunicación por puerto de serie, a diferencia de si utilizáramos otros dispositivos electrónicos, se realiza por USB.

3. Arduino:

Ahora que ya sabemos cómo se transmite la información de Raspberry a Arduino vamos a ver cómo se trata dicha información.

Antes de nada, debemos hacer las configuraciones pertinentes para poder tratar los datos correctamente y poder hacer un traspaso de la información y poder accionar nuestros actuadores. Arduino posee dos métodos principales, el método “Setup” y el método “Loop”.

El método “Setup” se ejecuta la primera vez que se enciende Arduino y en nuestro caso lo utilizamos para hacer las respectivas configuraciones. Lo primero que hacemos es inicializar la tasa de baudios a 9600, para que coincida con la tasa de baudios que hemos inicializado en Node-RED.

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

Con esto permitimos el paso de la información de Raspberry a Arduino y viceversa. Ahora debemos configurar las que serán las salidas de nuestros MOSFETs para poder accionar los actuadores. Esto es necesario dado que vamos a utilizar unos controladores que nombraremos posteriormente junto con las salidas digitales, y por la configuración de Arduino, es necesario que todas las salidas digitales sean configuradas como entradas o salidas.

Para configurar las salidas Arduino posee el método “pinMode”.

```
pinMode(15,OUTPUT);
pinMode(16,OUTPUT);
pinMode(17,OUTPUT);
pinMode(18,OUTPUT);
```

Con este método le indicamos que las entradas digitales 15, 16, 17 y 18 son salidas.

Para poder implementar un sistema eficaz que fuese capaz de predecir y anticiparse a encender o apagar nuestros actuadores hemos implementado un controlador PID. No obstante, este PID no ha sido diseñado por nosotros, este controlador ha sido importado a nuestro proyecto desde la librería open-source de **Brett Beauregard**[9]**.** Dicha librería contenía toda la automatización de los procesos PID, lo único que fue necesario por nuestra parte fue la configuración de unas constantes las cuales permiten hacer correcciones en las fórmulas matemáticas de los procesos PID.

Puesto que los controladores PID son muy complejos y es un campo que se sale completamente de este proyecto, hemos dejado las constantes de la configuración del controlador PID en unas constantes estándar.

```
PID myPID1(&Input1, &Output1, &Setpoint1, 2, 5, 1, DIRECT);
PID myPID2(&Input2, &Output2, &Setpoint2, 2, 5, 1, DIRECT);
```

No indagaremos en cómo funcionan por detrás estos procesos matemáticos del controlador, no obstante sí que explicaremos qué es cada uno de los parámetros que se le pasan a la librería y qué devuelve el método.

Para empezar tenemos dos PID, el PID 1 se encarga de la temperatura y el PID 2 se encarga de la humedad, ambos PID poseen distintos parámetros. Los parámetros Input son la temperatura actual de la cámara ambiental y se actualizan cada vez que el sensor nos devuelve la información actual, el Output es la salida de los cálculos del PID y tiene un valor entre 0 y 255, finalmente tenemos los Setpoint y estas variables son los valores deseados de temperatura y/o humedad al que queremos que llegue el Input.

Los controladores PI poseen dos modos de configuración, manual o automática. La configuración manual, en pocas palabras, implica que nosotros somos los responsables de la que será la salida o Output sin ningún cálculo. Automática implica que se introduce el input, se hacen los cálculos y nos devuelve una salida o Output. Para nuestro trabajo, por razones de simplicidad y desconocimiento, nuestros PID poseen el modo automático.

```
myPID1.SetMode(AUTOMATIC);
myPID2.SetMode(AUTOMATIC);
```

Ahora que hemos visto el método de Setup de Arduino, vamos a ver el método de Loop.

El método de Loop se ejecuta todo el rato, una vez termina de ejecutar las instrucciones que posee dentro este método, vuelve a empezar desde la primera instrucción.

Como pretendemos hacer que la cámara ambiental tenga que ir adaptando la temperatura y la humedad a la que nosotros esperamos, debemos hacer varias iteraciones de este mismo método actualizando todo el rato los valores actuales de humedad y temperatura, lo primero que debemos hacer es comprobar las variables de temperatura y humedad actuales y enviarlas

por puerto de serie a la Raspberry PI. Para ello utilizamos el método disponible de Arduino para escribir cosas por puerto serie:

```
Serial.println("{ \"Temperatura\" : " + (String)DHT.temperature + " }");
Serial.println("{ \"Humedad\" : " + (String)DHT.humidity +"}");
```

Está escrito de esta forma por una razón que veremos posteriormente.

Con el envío de datos realizado vamos a ver cómo tratamos la información que recibimos.
Cuando en Arduino se recibe información por el puerto de serie, esta información se almacena en un buffer o almacén temporal. Cuando Arduino detecta que hay información en el almacén, permite la utilización de una instrucción "available". Esta instrucción devuelve un valor de 0 si no hay ningún dato en el almacén, no obstante, devuelve un valor mayor que 0 si hay valores guardados en el almacén.

Para nuestro caso, hemos utilizado la instrucción available para comprobar en el loop si hay datos de temperatura y humedad a la espera de ser tratados tal que:

```
if (Serial.available() > 0 || hayNuevoValor) {
      hayNuevoValor = true;
```

Este mecanismo de control permite el paso siempre y cuando se cumpla una de las dos condiciones, una de las condiciones es que nos haya llegado información de nuestro SCADA y esté a la espera de ser procesada. La otra condición será verdadera cuando hayamos recibido al menos una vez datos de nuestro SCADA, en caso contrario, si no hemos recibido datos en ningún momento, esta condición será falsa y por lo tanto no se cumplirá.

Este mecanismo de control se ha diseñado de esta forma puesto que cuando la cámara se inicializa por primera vez no tiene datos de ningún tipo entonces no tiene nada que calcular y por lo tanto no necesita hacer ningún proceso. Es solo cuando recibe el primer dato, que entra en el mecanismo de control y puede mantener, durante todo el tiempo que se guste, los valores que se han enviado en un principio.

Posteriormente, una vez se han recibido los datos, utilizamos un segundo mecanismo de control tal que así:

```
if (Serial.available() > 0) {
      valoresEntrada = Serial.readString();
      valorHumedad = valoresEntrada .substring(2);
      valorTemperatura = valoresEntrada.substring(0, 2);

      Setpoint1 = valorTemperatura .toDouble();
      Margen1 = Setpoint1 - (Setpoint1 * 0.1);
      Margen2 = Setpoint1 + (Setpoint1 * 0.1);

      Setpoint2 = valorHumedad .toDouble();
      Margen3 = Setpoint2 - (Setpoint2 * 0.1);
      Margen4 = Setpoint2 + (Setpoint2 * 0.1);
}
```

Este mecanismo de control permite el cambio de Setpoints, los cuales hemos explicado anteriormente, de forma que, junto con el primer mecanismo de control, se pueda acceder al bucle que controla los actuadores y, solo si se han recibido nuevos valores desde el SCADA, se cambien los valores deseados en los Setpoints.

Además, este bloque de instrucciones también nos genera márgenes. Estos márgenes son el 10% por encima y el 10% por debajo de los valores deseados de modo que le demos un margen de error al sensor al leer la temperatura y la humedad y así reduzcamos la innecesaria activación de los actuadores.

Ahora que hemos realizado los cálculos de los márgenes, vamos a ver cómo se lee la temperatura, la humedad y como se acciona qué actuador en cada caso.

Nuestro sensor posee una librería llamada “dht.h” y, gracias a esta librería, podemos utilizar las siguientes instrucciones para leer los valores de temperatura y los valores de humedad:

```
Input1 = DHT.temperature;
Input2 = DHT.humidity;
```

Ahora que tenemos guardados los valores de Temperatura y Humedad en Input1 y Input2 respectivamente, vamos a indicarle a nuestros PIDs que deben realizar los cálculos necesarios. Para esto tenemos a nuestra disposición las instrucciones de “computar”:

```
myPID1.Compute();
myPID2.Compute();
```

A continuación tendremos en los Outputs 1 y 2 los valores necesarios entre 0 y 255 que serán las señales digitales que les daremos a nuestros actuadores, pero antes, debemos comprobar qué actuador debemos accionar.

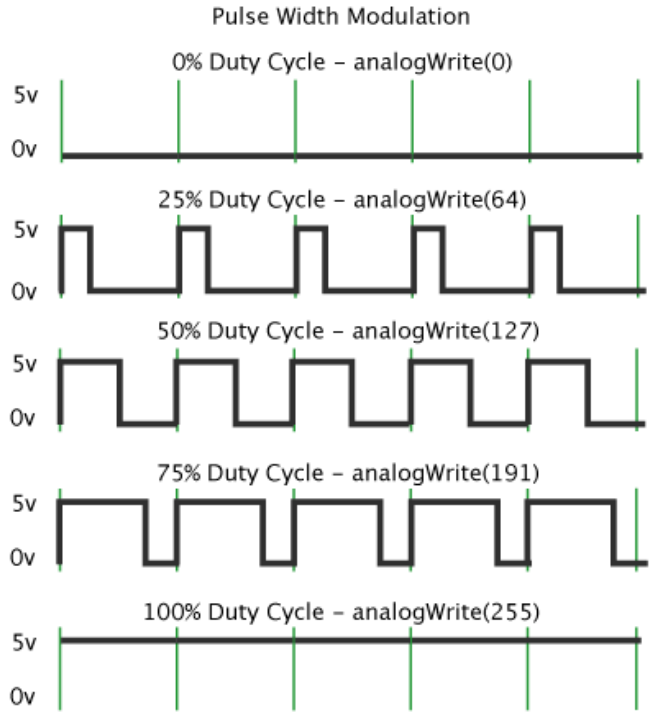
Para comprobar qué actuadores debemos accionar y cuáles no, hemos realizado una serie de comprobaciones utilizando mecanismos de control y los márgenes anteriormente indicados.

Para los mecanismos de control de la temperatura

```
if ((Input1 >= Margen1) && (Input1 <= Margen2)) {
	analogWrite(15,0);
	analogWrite(18,0);
} else {
	if (Input1 < Margen1) {
		analogWrite(18,Output1);
		analogWrite(15,0);
	}
	if (Input1 > Margen2) {
		analogWrite(18,0);
		analogWrite(15,Output1);
	}
}
```

En orden de arriba a abajo; lo primero que se hace es comprobar si el valor actual de la temperatura que hemos almacenado en Input1 está entre los márgenes de error que hemos calculado y si esto se cumple, le indicamos a los actuadores que no hagan nada; si no está en el margen, debemos comprobar si está por debajo de la temperatura deseada o si está por encima de la temperatura deseada y dependiendo de esto se accionarán las células peltier o se accionará la resistencia.

Las instrucciones de “analogWrite” son las instrucciones que se utilizan para enviar una señal de activación por la salida indicada en el primer número entre sus paréntesis, el segundo número sirve para indicar el valor de dicha señal. Este valor está representado por un valor entre 0 y 255, el mismo tipo de valor que nos devuelve el PID en el Output. Viene representado como un pulso de señales Algunos ejemplos visuales para entender el pulso son estos:



Este tipo de señal se utiliza para activar poco a poco nuestros actuadores, de forma que pueda aumentar o disminuir poco a poco el tiempo que permanecerán activos o apagados.

El siguiente código es similar al código anterior, no obstante, este es el código para los actuadores de la humedad:

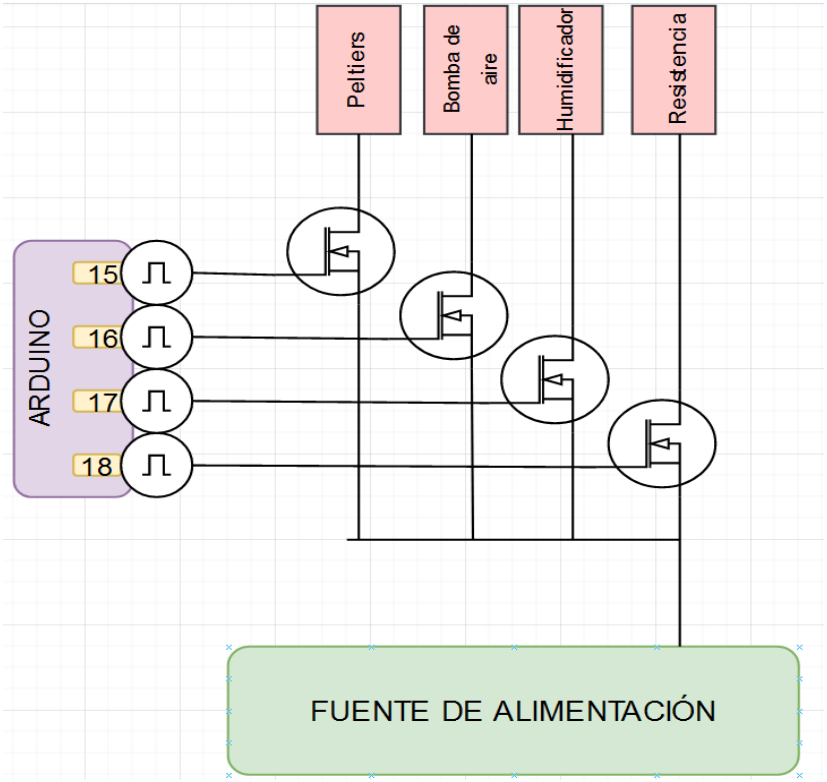
```
if ((Input2 >= Margen3) && (Input2 <= Margen4)) {
      analogWrite(16, 0);
      analogWrite(17, 0);
} else {
      if (Input2 < Margen3) {
            analogWrite(17,Output2);
            analogWrite(16,0);
      }
      if (Input2 > Margen4) {
            analogWrite(17,0);
            analogWrite(16,Output2);
      }
}
```

Por razones de repetición, no explicaremos este código, no obstante, mencionaremos que la principal diferencia es el actuador que se acciona. En lugar de la resistencia o la peltier, se acciona una bomba de aire o un humidificador.

Finalmente, después de esto, le indicamos a Arduino que se espere 5 segundos para no sobrecargar el envío de datos a nuestra aplicación SCADA y de esa forma hacer más sencilla la lectura de información y la generación de gráficas. La instrucción es propia de Arduino:

```
delay(5000);
```

Ahora que ya hemos visto cómo se tratan los datos en Arduino y cómo se accionan los actuadores, mostraremos el diseño electrónico que une Arduino con los actuadores y los componentes eléctricos.



[30] Muestra abstracta de la componente electrónica sobre la que trabaja nuestra cámara ambiental.

Los MOSFETs tienen la propiedad de que si no les pasamos corriente por una entrada, no permite el paso de la electricidad a través de él. Esta propiedad nos ha ayudado a activar o desactivar los actuadores y, junto con las salidas por pulsos de nuestro Arduino, esto permite mantener el tiempo que nosotros queramos el paso de la corriente a través de nuestros actuadores.

Cuando Arduino ejecuta una instrucción de “analogWrite” , se empieza a pasar la electricidad a través de nuestros MOSFETs permitiendo la activación de nuestros actuadores y, dependiendo de por qué salida hayamos seleccionado escribir en Arduino, podremos seleccionar qué actuador queremos encender o apagar.

30

6. Pruebas

La realización de pruebas ha sido necesaria a la hora de diseñar nuestro proyecto. Como se ha comentado en la sección de análisis de riesgos, se han debido de realizar pruebas de estrés para determinar los límites seguros de nuestro proyecto.

Se ha realizado pruebas de diversos tipos, principalmente han sido pruebas de estrés en las cuales se activan nuestros actuadores, como las peltiers o la resistencia, durante un largo periodo de tiempo y de esta forma determinar a partir de qué punto no se podía superar por mucho que se mantuviese encendido el actuador determinado.

Cuando se realizaron pruebas para las peltiers se mantuvieron encendidas las mismas durante 30 minutos y se analizó y registró que tras los primeros 23 minutos la temperatura mínima a la que se pudo llegar fue a 20 grados. Esta temperatura está determinada por diversos factores, algunos de estos factores son la propia temperatura ambiente externa a la cámara ambiental, la calidad de nuestros actuadores y la cantidad de peltiers colocadas en el interior de la cámara. La temperatura ambiente en el momento de la prueba era de 29 grados y la humedad ambiente en el interior de la cámara del 39%.

El resto de pruebas realizadas poseían un comportamiento similar a las pruebas de las células peltier. Para la resistencia se registró que su temperatura tras 10 minutos había aumentado a 37 grados. Para cuando se inició la prueba, la temperatura ambiente era de 31 grados y la humedad ambiente era del 35%.

Para las pruebas de estrés de la humedad, no fue necesario hacer pruebas para ver la cantidad de humedad ambiente que podíamos llegar a expulsar utilizando nuestro humidificador puesto que en este caso la limitación principal era debido a nuestro sensor y sus límites de lectura. Por lo que para la humedad máxima se tuvo que elegir el 80%, el cual es el límite de nuestro sensor DHT11.

A continuación la prueba de reducción de humedad fue realizada utilizando bombas de aire y activando los actuadores constantemente para que la humedad y el aire ambiente pasasen a través de la cámara auxiliar la cual contenía gel de sílice. Si bien es verdad este método puede ser poco eficiente, se diseñó de esta forma para evitar la posibilidad de que la humedad ambiente influya en el aire contenido dentro de nuestra cámara ambiental.

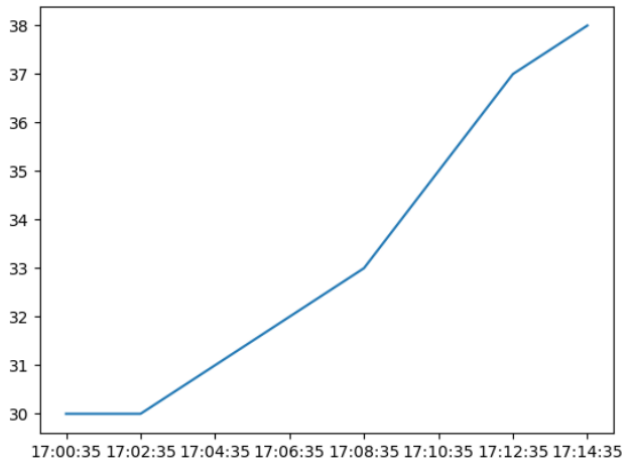
Tras accionar durante 30 minutos nuestras bombas de aire, la humedad mínima a la que se pudo llegar es del 20%.

Finalmente se realizaron pruebas para comprobar si nuestro proyecto era capaz de climatizar y modificar las variables ambientales. Las pruebas dentro de los límites establecidos fueron satisfactorias. A continuación mostraremos un ejemplo de dos pruebas:

Prueba para la subida de temperatura y humedad:

Temperatura deseada: 38ºC
Humedad deseada: 50%
Estado: Correcto
Tiempo transcurrido: 14 minutos.

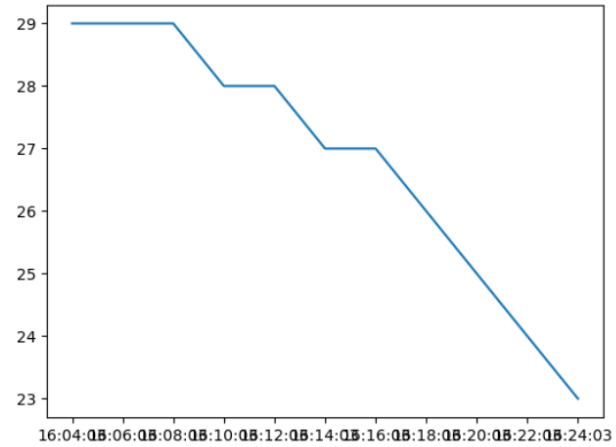
A continuación mostramos una gráfica generada a partir de la extracción de datos de nuestra base de datos sobre la temperatura registrada a lo largo de la prueba:



Prueba para la disminución de temperatura y humedad:

Temperatura deseada: 23ºC
Humedad deseada: 33%
Estado: Correcto
Tiempo transcurrido: 20 minutos.

A continuación mostramos una gráfica generada a partir de la extracción de datos de nuestra base de datos sobre la temperatura registrada a lo largo de la prueba:



Si bien estas no son todas las pruebas que se han realizado, nos permite tener una idea objetiva del resultado de nuestro proyecto y nos permite extraer unas conclusiones

7. Conclusiones

A lo largo de nuestro proyecto se han obtenido una serie de conclusiones que vamos a listar a continuación en este apartado:

1. El diseño de la cámara ambiental influye en gran parte en los resultados obtenidos. Los materiales usados para la fabricación de nuestra cámara ambiental, al menos en el exterior de la cámara, deben ser de materiales aislantes térmicos para evitar que las variables ambientales de la habitación donde esté nuestra cámara ambiental influyan a la hora de realizar pruebas y de obtener resultados.

2. La utilización de la comunicación MQTT es muy útil para un ancho de banda reducido y esto puede dar pie a la conexión de múltiples Raspberry PI conectadas simultáneamente a un punto de acceso o WiFi y por lo tanto podría ser útil para escalar nuestro proyecto a múltiples cámaras ambientales.

3. Los actuadores utilizados no han sido los más correctos puesto que ha sido un proyecto realizado con un bajo presupuesto y los resultados y los límites de estos podrían ser mejores si se hubiesen utilizado otros componentes electrónicos y actuadores.

4. La utilización de controladores PID ha sido muy acertada para el correcto control de nuestros actuadores, sin embargo, se podría haber estudiado en mayor profundidad esta tecnología controladora para ajustarla mejor a nuestros actuadores.

5. Nuestra aplicación tiene una interfaz mejorable, sin embargo, con una tecnología distinta y con mejor conocimiento de interfaces de usuarios y mayor conocimiento en cuanto a TKinter y Python se podría diseñar una interfaz más cómoda y accesible al resto de usuarios.

6. La planificación se estimó con un pensamiento demasiado optimista y no se tuvieron en cuenta muchos posibles fallos que terminaron apareciendo a la hora del desarrollo, lo cual perjudicó a nuestra previsión temporal y nuestra previsión económica.

7. Si el tiempo de diseño y planificación hubiese sido inferior o similar al tiempo optimista que se realizó en un principio, nuestra aplicación podría haber implementado mejores funcionalidades, como por ejemplo: gráficas actualizadas en tiempo real.

8. Trabajos futuros

Vamos ahora a hablar de posibles ampliaciones o trabajos futuros que se podrían realizar en base a este Trabajo de Fin de Grado. Algunas de estas ampliaciones o trabajos a futuro podrían ser:

1. La posibilidad de ampliar nuestra aplicación a diversas cámaras ambientales, para esto se podría incluir mayores entradas y salidas de datos en Node-RED y diversas modificaciones en nuestra aplicación.

2. La posibilidad de que nuestra base de datos, además de guardar temperatura y humedad junto con la hora registrada, también guardase el identificador de la cámara ambiental si se utilizan mayor cantidad de cámaras ambientales.

3. Una posible mejora sería la mejora visual de nuestra aplicación para presentar una interfaz más agradable y visiblemente cómoda.

4. Una mejora para nuestra aplicación que permita tener acceso selectivo a una cámara ambiental de un conjunto de cámaras ambientales que tengamos operativas actualmente.

5. Mejorar y afinar los parámetros constantes de nuestros controladores PID en Arduino para obtener mejor eficacia de tiempo para obtener nuestros resultados deseados.

6. Un controlador de excepciones que nos informe si ha fallado la conexión con nuestra cámara ambiental y que nos informe cuando una de nuestras cámaras haya fallado al recibir datos o simplemente no esté conectada de alguna forma a nuestra Raspberry PI.

7. Una posible ampliación de este trabajo sería el incremento en la seguridad y eficacia en la transmisión de datos entre los distintos medios de comunicación, tanto inalámbrica como por cable.

Estas son algunas, pero no todas, las posibles mejoras que se podrían llegar a implementar a futuro utilizando como base este proyecto.

9. Referencias

[1].Museo de Arte de Puerto Rico. (s. f.). *¿Cómo se deterioran las obras de arte?* Mapr.org. Recuperado 2012, de https://www.mapr.org/sites/default/files/19._hoja_educativa_sobre_conservacion_la_plena.pdf

[2].Craggs, I. (s. f.). *Eclipse Paho | The Eclipse Foundation*. Eclipse Paho. https://www.eclipse.org/paho/

[3].*SQLite Home Page*. (s. f.). SQLite. https://www.sqlite.org/index.html

[4]Love, D. (s. f.). *Tkinter GUI Programming by Example*. O'Reilly Online Learning. Recuperado 28 de junio de 2022, de https://www.oreilly.com/library/view/tkinter-gui-programming/9781788627481/cover.xhtml

[5].*Matplotlib — Visualization with Python*. (s. f.). Matplotlib. https://matplotlib.org/

[6].*Seaborn: statistical data visualization — seaborn 0.11.2 documentation*. (s. f.). Seaborn. https://seaborn.pydata.org/

[7].Eclipse Foundation. (s. f.). *Eclipse Mosquitto*. Eclipse Mosquitto. https://mosquitto.org/

[8].*Graphical User Interfaces with Tk — Python 3.10.5 documentation*. (s. f.). TKinter. https://docs.python.org/3/library/tk.html

[9].Brett. (s. f.). *GitHub - br3ttb/Arduino-PID-Library*. GitHub. https://github.com/br3ttb/Arduino-PID-Library

10. ODS

Este trabajo está fuertemente ligado con los ODS de Industria, innovación e infraestructuras. Hay diversas razones por las que este Trabajo de Fin de Grado está fuertemente ligado a este ODS, una de las principales razones es por la utilización de tecnologías comúnmente utilizadas en el ámbito de las tecnologías industriales y en el sector de la Industria. Para este proyecto, no solo se ha tenido que investigar sobre las diversas tecnologías que se utilizan hoy en día en la industria y en el ámbito de las cámaras ambientales y el control de variables ambientales, sino que también se ha tenido que innovar utilizando un protocolo de mensajería, en nuestro caso MQTT, que permitiese controlar una cámara ambiental o diversas cámaras ambientales desde muy lejos y sin la necesidad de ningún tipo de conexión por cable. Esto podría ser útil si se vincula el origen de MQTT a nuestro proyecto, puesto que este protocolo fue diseñado por Andy Stanford y Arlen Nipper para poder controlar y monitorizar los oleoductos con un sistema de control SCADA lo cual es idéntico, sino parecido, a nuestro proyecto. Utilizando un protocolo de mensajería a distancia que permitiese la comunicación eficiente utilizando paquetes de información muy ligeros sin coste de ancho de banda.

Otras razones por las que nuestro Trabajo de Fin de Grado está altamente vinculado a la Industria es por la propia investigación que se ha debido de realizar en este ámbito y sector para poder investigar sobre las distintas tecnologías y las distintas cámaras ambientales ya existentes. También se ha debido de investigar diversas tecnologías de comunicación utilizadas en el mercado actual y hacer las respectivas comparativas y obtener los pros y los contras de cada una de estas. Además, se ha debido de estudiar el funcionamiento de una cámara ambiental y simular una y para esto se han debido de investigar y buscar tecnologías de Hardware y electrónica que nos permitiese hacer la correcta construcción de la misma.

Por otra parte, nuestro trabajo también ha tenido la influencia de la industria en el sentido de las aplicaciones software industriales puesto que hemos tenido que desarrollar una aplicación controladora similar a las aplicaciones SCADA que existen en el mercado actual, si bien es una aplicación pequeña que no comparte todas las funcionalidades de una aplicación SCADA de nivel industrial, la principal funcionalidad , que es la posibilidad de monitorizar y controlar una cámara industrial o diversas cámaras industriales, sí la comparte y además permite el registro en bases de datos y la misma consulta en bases de datos sobre los datos registrados en nuestra cámara ambiental.

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
● **Fin de la pobreza.**				**X**
● **Hambre cero.**				**X**
● **Salud y bienestar.**				**X**
● **Educación de calidad.**				**X**
● **Igualdad de género.**				**X**
● **Agua limpia y saneamiento.**				**X**
● **Energía asequible y no contaminante.**				**X**
● **Trabajo decente y crecimiento económico.**				**X**
● **Industria, innovación e infraestructuras.**	**X**			
● **Reducción de las desigualdades**.				**X**
● **Ciudades y comunidades sostenibles**.				**X**
● **Producción y consumo responsables.**				**X**
● **Acción por el clima.**				**X**
● **Vida submarina.**				**X**
● **Vida de ecosistemas terrestres.**				**X**
● **Paz, justicia e instituciones sólidas.**				**X**
● **Alianzas para lograr objetivos.**				**X**