GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Sistema de monitorización agrícola y riego automático mediante comunicación LoRa

Autor: Alvaro Lorenzo Castro

Director: José Daniel Muñoz Frías

Codirector: Romano Giannetti

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Sistema de monitorización agrícola y riego automático mediante comunicación LoRa

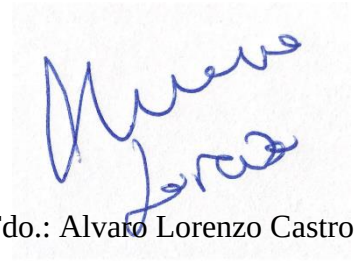
en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2021/22 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.



Fdo.: Alvaro Lorenzo Castro Fecha: 02/ 07/ 2022

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: José Daniel Muñoz Frías Fecha: ……/ ……/ ……

Firmado por MUÑOZ FRIAS JOSE DANIEL - ***7384** el día 11/07/2022 con un certificado emitido

Fdo.: Romano Giannetti Fecha: ……/ ……/ ……

Firmado por GIANNETTI ROMANO - X2593425Z el día 11/07/2022 con un certificado emitido por AC FNMT Usuarios

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Sistema de monitorización agrícola y riego automático mediante comunicación LoRa

Autor: Alvaro Lorenzo Castro

Director: José Daniel Muñoz Frías

Codirector: Romano Giannetti

Madrid

SISTEMA DE MONITORIZACIÓN AGRÍCOLA Y RIEGO AUTOMÁTICO MEDIANTE COMUNICACIÓN LORA

Autor: Lorenzo Castro, Alvaro.
Director: Muñoz Frías, José Daniel.
Codirector: Romano Giannetti
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

Es este proyecto se diseña un sistema inalámbrico de riego que permite la monitorización y control de la humedad del terreno para hacer más eficiente el desarrollo de una explotación agrícola.

Palabras clave: LoRa, IoT, Smart Irrigation, Arduino.

1. Introducción

Este Proyecto en colaboración con la INEA y la Universitat Ramón LLull busca mejorar los TFG de Miguel Campano García [1] y Luis Carlos Parra Rebollo [2]. En el caso de Luis Carlos Parra, se enfocó en utilizar una tecnología más tradicional como es WIFI lo que le produjo diversos problemas en cuanto al rango efectivo de utilización en pruebas reales. En cuanto a Miguel Campano, mantendremos el uso del protocolo LoRa y expandiremos su trabajo con el añadido de sensores de humedad que permitan monitorizar el estado del terreno de manera automática y buscaremos aumentar la autonomía del sistema utilizando formas alternativas de alimentación y menor consumo.

2. Definición del proyecto

El objetivo del proyecto será crear un sistema funcional que pueda ser desplegado en una explotación agrícola y nos permita recibir datos de manera periódica sobre el estado actual del terreno, y enviar a la zona un esquema de riego concreto según especificaciones del usuario. Se desarrollará una aplicación de escritorio que permita realizar estas acciones abstrayendo lo máximo posible al usuario de su funcionamiento permitiendo su uso a

personas con bajos niveles de conocimientos informáticos. El esquema de conexiones y distribución de los elementos quedará como se expone en la Ilustración 1.

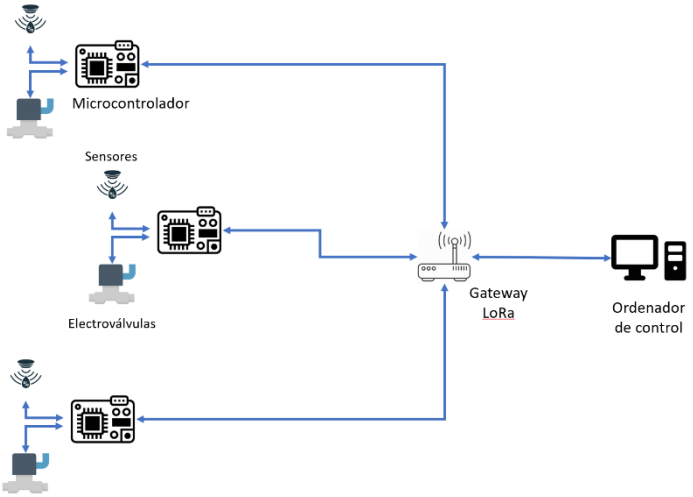


Ilustración 1 – Esquema del sistema

3. Descripción del modelo/sistema/herramienta

Se conecta la placa de Arduino MKR a una serie de válvulas y un sensor de humedad. Una vez arrancada la aplicación en el ordenador también, estos se empezarán a comunicar. El Arduino enviará cada 7 minutos un mensaje con información sobre el estado actual de la humedad del terreno. Teniendo en cuenta esto, la aplicación aprovechará para enviar cualquier paquete que tenga esperando. Esto se debe al funcionamiento de nuestro dispositivo LoRa ya que los dispositivos de clase A abren una ventana de recepción sólo tras haber enviado como se puede ver en la Ilustración 2.

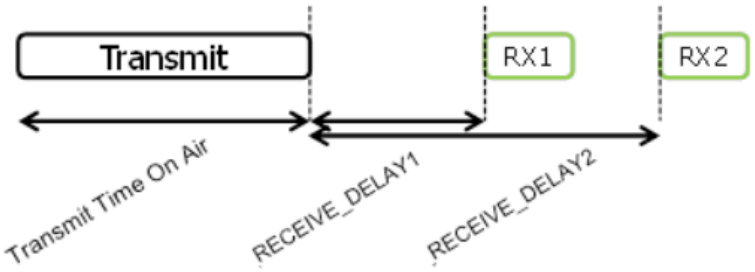


Ilustración 2 – Ventana de recepción de LoRa (3)

La aplicación recopilará los datos recibidos y los guardará pudiendo posteriormente mostrarse en una gráfica para poder realizar una apropiado seguimiento. Además, esta aplicación permitirá crear un esquema de riego para cada válvula conectada, permitiendo modificar el tiempo de riego y la distancia entre riegos pudiendo optimizarse en zonas si el usuario lo requiere.

3.1 Metodología

El proyecto se ha divido en varias fases, siendo la primera un estudio y entendimiento del funcionamiento del protocolo LoRa, y sus implementaciones en distintos proyectos. Una vez se ha estudiado su funcionamiento, se pasó a desarrollar el programa del Arduino que controlaría tanto los sensores como las válvulas. Tras esto, habiendo hecho pruebas para comprobar que el funcionamiento es el esperado, se pasó a crear un prototipo en una placa perforada donde se crearon diversos circuitos de adaptación para los componentes como reguladores de tensión y drivers para las válvulas. Durante este proceso se colocaron circuitos para instalar más de un sensor. Esto es debido a que se han investigado diferentes alternativas en cuanto a los sensores de humedad con diferentes tipos de medidas y precios con el objetivo de encontrar alternativas más económicas. Por último, se desarrolló la aplicación previamente mencionada y últimamente se comprobó el funcionamiento de todo el sistema en su conjunto.

4. Resultados

Una vez completadas todas las fases descritas, se ha podido comprobar el funcionamiento completo del sistema en un entorno controlado con el sensor colocado en una maceta. El sistema ha sido enviado a Valladolid para probar su correcto funcionamiento en una situación real. Resultados de la recopilación de datos a lo largo de un fin de semana se pueden observar en la Ilustración 3. En esta se puede observar como a lo largo de las horas va disminuyendo la humedad del terreno. Existe un espacio entre las 20:00 del día 4 y las 10 del día 5 en el que no se estuvieron recopilando datos ya que el equipo se apagó durante ese periodo. Después, por las 15 del día 5 se regó la maceta lo que causó el aumento repentino de humedad como era de esperar.

Ilustración 3 – Datos obtenidos

5. Conclusiones

Los objetivos propuestos en este proyecto han sido cumplidos y el sistema funciona como se esperaba. Eso no quita que no presente puntos de mejora o mejoras futuras que se le pudiesen hacer. En primer lugar, se ha desarrollado un prototipo, y se ha diseñado una PCB para poder permitir un uso industrial. Además, se podría desplegar un pequeño servidor que permitiese recoger los datos y que la aplicación del usuario solo cumpla las funciones básicas de mostrar la información que guardase el servidor y mandar los esquemas que se requieran.

6. Referencias

[1] M. C. García, «Diseño y desarrollo de un sistema de monitorización y control del riego en una explotación agrícola con cobertura extendida,» 2020.

[2] L. C. P. Rebollo, «Diseño de un sistema de monitorización y control del riego en una explotación agrícola,» 2020.

[3] LoRa Alliance, «LoRaWAN 1.0.3 Specification,» 2018. [En línea]. Available: https://lora-alliance.org/wp-content/uploads/2020/11/lorawan1.0.3.pdf. [Último acceso: 10 9 2021].

AUTOMATIC AGRICULTURAL MONITORIZATION AND IRRIGATION SYSTEM THORUGH LORA COMMUNICATION

Author: Lorenzo Castro, Alvaro.
Supervisor: Muñoz Frías, José Daniel.
Codirector: Romano Gianetti
Colaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

In this project, a system is designed such as it permits the monitorization and control of the humidity of the terrain in order to make a more efficient use of water during the development of a farm.

Keywords: LoRa, IoT, Smart Irrigation, Arduino.

1. Introduction

This project in collaboration with INEA and the Universitat Ramón LLull looks to improve the end of degree projects made by Miguel Campano García [1] and Luis Carlos Parra Rebollo [2]. For Luis Carlos Parra, he focused his work in more traditional technologies like WIFI which developed into problems regarding effective range in the communication during real-world testing. Regarding Miguel Campano, we Will maintain the use of LoRa and expand his work adding humidity sensors which would allow monitoring of the state of the terrain remotely and we will investigate more ways to increase the autonomy of the system studying alternatives around consumption reduction and power delivery.

2. Definition of the project

The main goal for this Project is to create a functional system which can be deployed in a farm, and which would allow to receive periodic data about the current state of the soil and to send a schema of the desired irrigation timings specified by the user. A desktop app will also be developed that would allow to perform these actions meanwhile abstracting its processes to the end user, allowing users with low levels of computer knowledge to use it

successfully. The connection and distribution of the different elements of the system can be found in Illustration 1.

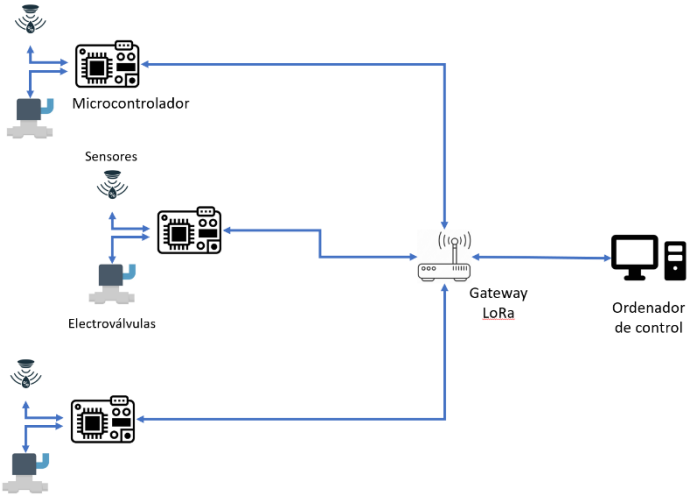


Illustration 1 – System diagram

3. Description of the system and methodology

Several electro valves and a humidity sensor are connected to an Arduino MKR board. Once the desktop app is launched, the board and the computer will start communicating. The Arduino will send every 7 minutes updates about the current humidity of the soil. Taking this into account, the app will send any packet that has in its queue. This made this way because of how LoRa devices communicate where the receiving window is only opened right after sending for class A devices as it is shown in Illustration 2.

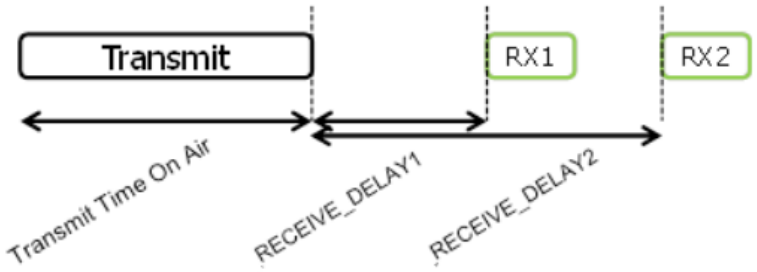


Illustration 2 – Receiving window of LoRa device (3)

The app will store the received data and will allow the user to see it in a graph to make a proper monitoring of the terrain. In addition, the app will allow to create an irrigation schema for each connected valve, being able to modify the irrigation times and the gap in between irrigations optimizing each zone if the user desires.

3.1 Methodology

The Project has been divided in several phases, the first being a study and understanding of the LoRa protocol, its workings, and its implementations in other projects. Once this has been achieved, we went for the development of an Arduino app that controls sensors and valves. After that, and the testing of the previous phase been made, we went onto creating a functioning prototype in a perfboard where adaptation circuits were created for each component such as voltage regulators and drivers for the valves. During this process, several circuits for controlling sensors where placed. This was in order to test several alternatives for sensors that use different technologies and process with the goal of finding cheaper options. Lastly, the desktop app was developed and the system in conjunction was tested.

4. Results

Once all the described phases were complete, the functions of the system have been checked in a controlled environment in a flowerpot. The system has been sent to Valladolid to make real world testing. Results of the data gathering throughout a weekend can be seen in Illustration 3. In it can be seen how as time passes, the soil dries up. A gap between 20:00 on the 4th and 10 on the 5th can be seen as the computer that gathered the data shutdown. After that, around hour 15 of the 5th, the flowerpot was irrigated which caused the humidity to skyrocket as expected.

Ilustración 3 – Datos obtenidos

5. Conclusions

The proposed goals have been accomplished and the system functions as expected. That doesn't mean that improvements or future advances can't be made. A prototype has been developed, as well as a PCB was designed as to allow its use in industrial settings.

6. References

[1] M. C. García, «Diseño y desarrollo de un sistema de monitorización y control del riego en una explotación agrícola con cobertura extendida,» 2020.

[2] L. C. P. Rebollo, «Diseño de un sistema de monitorización y control del riego en una explotación agrícola,» 2020.

[3] LoRa Alliance, «LoRaWAN 1.0.3 Specification,» 2018. [En línea]. Available: https://lora-alliance.org/wp-content/uploads/2020/11/lorawan1.0.3.pdf. [Último acceso: 10 9 2021].

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

1.1 MOTIVACIÓN DEL PROYECTO

Hoy en día existe la tecnología capaz de desarrollar sistemas más eficientes, avanzados y económicos que en tiempos pasados. Esta situación crea la oportunidad perfecta para poder crear un producto capaz de ser utilizado a gran escala y, especialmente, que sea más accesible para todos y no sea perjudicial para el entorno. Un sistema de riego automático no solo traería ventajas a nivel de optimización del desarrollo del cultivo, sino que aumentaría la eficiencia en el uso del agua disponible y prevendría su uso excesivo.

Con el desarrollo de este trabajo se pretende desarrollar un sistema IoT que permita automatizar el sistema de riego de una explotación agrícola permitiendo ser adaptada a los diferentes cambios que se puedan hacer en el clima y pudiendo implementar los distintos avances que se han desarrollado en cuanto a las técnicas de riego como el riego por goteo. Además, este sistema deberá ser apropiado para poder ser implementado en áreas grandes donde el tendido eléctrico no sería una opción así que la eficiencia energética será un factor muy importante. Acompañando la eficiencia energética estará la reducción del coste con la utilización de sistemas ampliamente disponibles y económicos como Arduino y utilización de comunicaciones que no requieran de suscripciones y operen en frecuencias de uso libre.

El desarrollo de este proyecto ha sido realizado en colaboración con otras universidades como INEA y la Universitat Ramón LLull y pretende avanzar los progresos de otros proyectos como el de Miguel Campano García [1] y Luis Carlos Parra Rebollo [2]. En el caso de Luis Carlos Parra, se enfocó en utilizar una tecnología más tradicional como es WIFI lo que le produjo diversos problemas en cuanto al rango efectivo de utilización en pruebas reales. En cuanto a Miguel Campano, expandiremos su trabajo con el añadido de sensores de humedad que permitan monitorizar el estado del terreno de manera automática y buscaremos aumentar la autonomía del sistema utilizando formas alternativas de alimentación y menor consumo.

1.2 RECURSOS POR EMPLEAR

Aquí se describirán que programas y recursos se han utilizado para el desarrollo de este proyecto.

1.2.1 RECURSOS HARDWARE Y COMPONENTES ELECTRÓNICOS

Como el objetivo de este proyecto es crear un sistema que permita comunicaciones entre un ordenador y un microcontrolador a través de LoRa como realizó Miguel Campano, se han escogido los elementos acordes a hacer esta tarea más simple y funcional. Por parte del microcontrolador, se ha escogido un Arduino MKR WAN 1300 [3] el cual incluye en su PCB el módulo de radio LoRa que necesitamos para comunicarnos. Además, este presenta un módulo RTC[1] que nos permitirá organizar los tiempos de riego que se reciban de una manera más eficiente y simple.

Para permitir las comunicaciones entre un PC y un dispositivo que envía señales a través de LoRa, se requiere de un Gateway que traduzca los paquetes LoRa a paquetes IP que pueda recibir. En nuestro caso, esto lo hacemos a través del Indoor Gateway de The Things Industries [4] adaptado a la frecuencia europea. En cuanto al PC, cualquier ordenador con soporte Python debe ser capaz de usar este sistema.

Junto al Arduino, se han necesitado realizar diversos ajustes para permitir utilizar distintos sensores y las electroválvulas por sus diversas características eléctricas e incompatibilidades. Es por esto que el Arduino se colocó en una perfboard y se realizaron diversos circuitos de adaptación que se explicarán en el Capítulo 5. Para estos circuitos, se han necesitado los siguientes componentes:

- Regulador de tensión a 5V L7805 [5]
- Driver doble L298 [6]
- Diodos Schottky
- Elementos pasivos (Resistencias y condensadores)

[1] Real Time Clock

- Transistores 2N3906 y 2N3904

Por otro lado, los sensores utilizados ha sido los siguientes:

- Teros 10 [7]
- Sensor Resistivo de Elecfreaks [8]
- Sensor capacitivo de Seedstudio [9]

Por último, para la válvula se ha utilizado una electroválvula de Rain Bird [10].

1.2.2 RECURSOS SOFTWARE

A lo largo de este proyecto se han tenido que desarrollar dos programas, uno para controlar el Arduino, y otro para manejar las comunicaciones desde el ordenador. Como se ha dicho antes, el programa de escritorio se ha desarrollado en Python, y para su desarrollo se ha utilizado el IDE de VSCode. Por otro lado, el desarrollo en Arduino se ha realizado en el propio IDE que proporcionan.

Otros programas que se han utilizado han sido KiCad para el diseño de circuitos y creación del esquema de conexiones, y la plataforma de TheThingsIndustries para monitorizar las comunicaciones y conectar la aplicación con el Arduino.

1.2.3 HERRAMIENTAS DE LABORATORIO

Casi todo el desarrollo del proyecto se ha realizado en los laboratorios de electrónica de ICAI. Para el análisis de señales eléctricas como las que vienen de los sensores cuando miden, se ha utilizado un osciloscopio Tektronix TDS 210 con sondas conectadas a x10 de atenuación. Por otro lado, para alimentar el Arduino a través del regulador de 5V, se utilizó la fuente de alimentación regulable KAISE DF1731SB5A del laboratorio para generar una tensión continua de 9 y 12V, ya que finalmente se alimentará el sistema con una batería de 12V. Por último, se utilizaron polímetros para comprobar las soldaduras en la perfboard y que no se crease ningún cortocircuito.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

En esta sección se comentarán las diferentes tecnologías a utilizar a lo largo del desarrollo del proyecto las cuales incluirán elementos tanto de software como hardware.

2.1 MICROCONTROLADOR

Un microcontrolador consiste en un circuito integrado que permite realizar cálculos y procesar datos además de interactuar con las distintas conexiones que pueda tener a través de sus pines de entrada y salida. La principal diferencia con un microprocesador es que los microprocesadores suelen presentar un mayor rendimiento lo que en muchos casos significa un mayor consumo.

Para este proyecto se utilizará el microcontrolador SAMD21 Cortex-M0+ de 32bits [11] presente en el Arduino MKR WAN 1300 [3]. El aspecto más importante de este procesador de cara a este proyecto es su tensión de operación en los pines I/O que es de 3.3V, y que contiene un módulo RTC el cual resultará muy útil para organizar envíos y recepción de datos de manera sincronizada.

2.2 SENSORES DE HUMEDAD

La medición de la cantidad de agua del terreno se realizará a través de diferentes sensores que utilizarán diferentes características del terreno para obtener sus medidas. Entre estas, destacan por su uso los sensores que utilizan parámetros resistivos, capacitivos, y de permitividad dieléctrica.

El más usado en la industria para obtener mediciones más precisas es el de permitividad ya que es un concepto muy estudiado para diferentes tipos de terreno y composiciones. Para traducir las medidas de permitividad a contenido volumétrico de agua (VWC en inglés), se

utilizan variaciones de la ecuación de Topp [12], modificando los coeficientes para aproximarse más al terreno en concreto.

En cuanto a la toma de medidas, los sensores suelen tener una serie de requerimientos en cuanto a volumen de tierra alrededor como procedimientos específicos de instalación ya que pequeños cambios pueden producir variaciones en la medida como en el caso de que existan burbujas de aire en las terminaciones o insuficiente cantidad de terreno alrededor del sensor lo que hace que sus medidas sean erróneas.

2.3 *ELECTROVÁLVULAS*

Para posibilitar el riego utilizando un microcontrolador, se utilizan electroválvulas que suelen consistir en un solenoide al que se actúa aplicando una diferencia de tensión lo que hace que empuje un diafragma (parte azul de la Ilustración 1) y esto, en función de la dirección de la tensión, abre y cierra la válvula permitiendo o no el paso de agua.

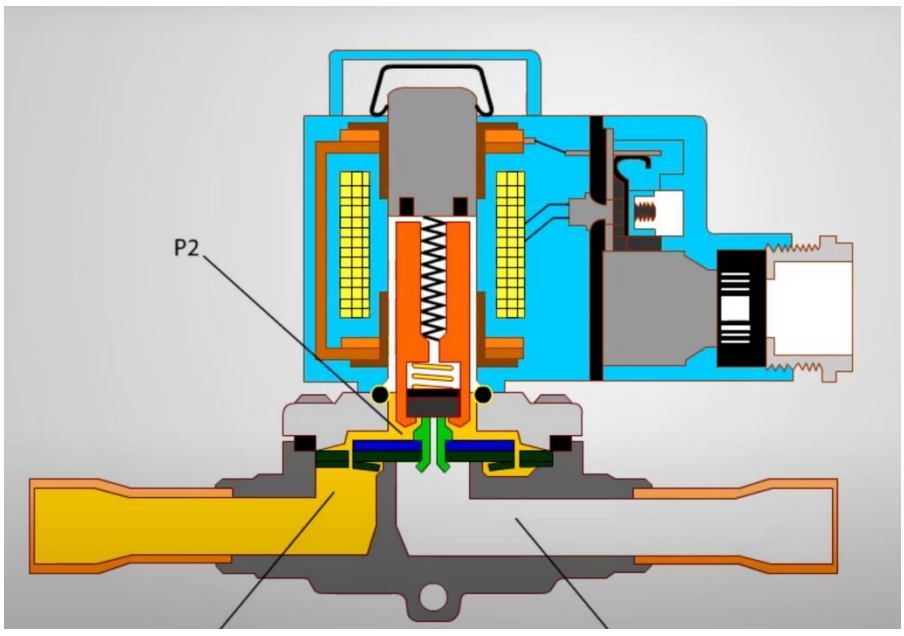


Ilustración 1 - Esquema de una electroválvula

2.4 GATEWAY

Para facilitar la comunicación entre nuestro microcontrolador y la aplicación que se desarrollará, se utilizará un Gateway que es un dispositivo que permite traducir la información de un protocolo a otro usado en otras redes. En nuestro caso, el Gateway utilizado será el The Things Indoor Gateway [4] que traducirá las señales LoRa (este protocolo se explicará a continuación) a mensajes IP que serán enviados por la red para que sean interpretados por la aplicación.

2.5 PROTOCOLOS DE COMUNICACIÓN

Estos son una serie de normas y reglas que permiten la comunicación entre dispositivos asegurando su legibilidad y consistencia en cuanto al formato de los mensajes. En el caso de este proyecto, nos apoyaremos en dos protocolos diferentes: LoRa, y MQTT.

LoRaWAN o **LoRa** [13] describe una especificación para comunicaciones de largo alcance y poca potencia, especializándose en su uso para dispositivos de bajo consumo y largo alcance. Esta forma parte de un conjunto de tecnologías llamadas LPWAN o Low-Power Wide Area Network cuyo objetivo es el explicado, buscando el alcance y optimizando la eficiencia energética. Este protocolo es el que se utilizará para la comunicación entre el microcontrolador y el Gateway utilizando el módulo de radio CMWX1ZZABZ-078 [14].

MQTT es un protocolo de comunicación de tipo publish/subscribe orientado a IoT. Este protocolo está concretamente optimizado hacia el uso mínimo de recursos de red y tamaño total de cada paquete [15]. La arquitectura de comunicaciones publish/subscribe se basan en la existencia de dos tipos de cliente, productores y consumidores. Los productores son los que publican mensajes en un canal, mientras que los consumidores se suscriben y reciben los mensajes publicados en un cierto canal. Estos se utilizará para recibir y enviar mensajes entre la aplicación final y el microcontrolador.

2.6 SOFTWARE Y PLATAFORMAS

2.6.1 LENGUAJES DE PROGRAMACIÓN

2.6.1.1 Python

Python es un lenguaje interpretado de alto nivel que destaca por su énfasis en legibilidad, soportando varios paradigmas de programación como programación orientada a objetos y programación funcional [16]. Este es el lenguaje en el que se ha decidió desarrollar la aplicación debido a su amplia disponibilidad en todas las plataformas, amplia gama de librerías y facilidad de desarrollo.

2.6.1.2 Arduino/C++

El uso de un microcontrolador Arduino implica que para su programación se deberá utilizar el framework diseñado para estas placas. El lenguaje es una versión simplificada de C++ que hace su programación mucho más alcanzable para aquellos que no presentan una gran experiencia en programación, especialmente la orientada a hardware.

2.6.2 THE THINGS INDUSTRIES/THE THINGS STACK

Esta es una plataforma web que suele ser utilizada en proyectos en los que está involucrado algún módulo LoRa debido a que crea un entorno de monitorización de los dispositivos conectados en nuestra aplicación pudiendo realizar envíos y recepciones de prueba además de disponer de integraciones como la de MQTT que permiten conectar de una manera más simple los dispositivos con aplicaciones finales.

La plataforma puede ser utilizada de manera gratuita pero dependiendo de su uso, puede requerir de una suscripción, esto se puede evitar utilizando alternativas que se explicarán posteriormente.

Capítulo 3. ESTADO DEL ARTE

Los sistemas de riego automático no son nada nuevo hoy en día, cuando uno anda por calle y pasa algún parche de césped lo más probable es que este se riegue regularmente a través de algún tipo de sistema de riego automático. Más aún a nivel industrial, existen soluciones de empresas tanto a nivel individual de controladores como sistemas completos que permiten funcionalidades muy similares a las que se propone desarrollar en este proyecto. Sin embargo, estas soluciones pueden llegar a ser muy costosas para un individuo o pueden no estar adaptadas para terrenos más pequeños y es ahí donde se ha enfocado este proyecto. Si se consiguen los objetivos propuestos, el sistema creado sería capaz de ser desplegado en una plantación agrícola de una manera mucho más económica sin requerir de suscripciones, además de ser todo el código y funciones desarrolladas open-source permitiendo a cualquiera adaptar este proyecto a sus necesidades específicas o proyectos diferentes.

Antes de empezar a diseñar el sistema, se han evaluado las diferentes opciones que existen en el mercado en cuanto a alternativas para las comunicaciones, hardware y proyectos anteriores, como se explica a continuación.

3.1 COMUNICACIÓN DE LARGO ALCANCE Y POCO CONSUMO

Un proyecto con fines como los propuestos tiene dos requisitos principales. En primer lugar, sería su bajo consumo debido a las grandes distancias que existirán entre los dispositivos conectados por lo que serán alimentados por baterías que se podrán recargar en intervalos de tiempo largos. En segundo lugar y más importante, la implantación de una tecnología inalámbrica para permitir la conectividad de una manera fácil y eficiente con el servidor que aloje la aplicación. Esto agregado a la consideración de la implantación en una explotación agrícola donde la instalación tanto de tendidos eléctricos como de comunicaciones no sería ideal ni práctico.

Es por estas razones que se ha optado por la utilización de comunicaciones de tipo ***LPWAN*** (Low-Power Wide Area Network) que están especializadas en estos aspectos en concreto con algunas permitiendo la conectividad a varios kilómetros de distancia.

Como primera opción se tendría probablemente la más conocida que es a través de **conexión celular** con una tarjeta SIM debido a la facilidad de conexión y la infraestructura que ya existe. Entre sus beneficios se encuentran la capacidad de transmisión con **LTE** o a día de hoy incluso ***5G*** que puede llegar a ratios bastante altos, aunque para nuestra aplicación no es necesario; y la facilidad de conectar el dispositivo controlador con nuestra base ya que solo requeriría instalar la tarjeta y realizar el acuerdo con el operador correspondiente. Sin embargo, esta opción no es la más eficiente energéticamente y tanto la compra de las tarjetas como la suscripción necesaria con el operador escogido puede resultar en un gran coste cuando entramos en terrenos muy extensos. Además, en algunos casos las grandes superficies que puede abarcar una explotación y su distancia con respecto a poblaciones pueden resultar en carencia de cobertura en algunas zonas del terreno.

Como alternativa y mejora a la conexión tradicional existe la tecnología **LTE-M** que pretende utilizar la misma banda que el LTE tradicional, pero de una manera que mejora la duración de la batería y sea más aplicable a conexiones para dispositivos IoT. Su coste es más económico que en el caso anterior y proporciona velocidades de subida y bajada de hasta 1Mbps (más que suficiente para nuestra aplicación) y latencias inferiores a 100ms. Además de aportar más que suficientes prestaciones para nuestro caso, la red LTE-M no está desplegada a nivel nacional en España [17] lo cual presenta un problema a la hora de implementarlo.

Teniendo en cuenta estas limitaciones, existe el sistema **NB-IoT** [18] (Narrowband-IoT) que está orientado a dispositivos distribuidos de manera masiva. Esta tecnología también usaría las redes de telefonía móvil pero una de sus grandes ventajas es que es más barata que otras tecnologías como GPRS, LTE y LTE-M. Además, esta tecnología permitiría conectar en una misma red hasta 100.000 conexiones debido al poco uso de ancho de banda que tiene de unos 180kHz.

SigFox es una tecnología que ha ganado bastante popularidad en los últimos años. Con un gran despliegue en España y una tecnología especialmente enfocada en el bajo consumo de energía y bajo coste, es una tecnología muy atractiva para cualquier proyecto de IoT. Dentro de sus características se encuentran [19]:

- Diseño enfocado a largas distancias con capacidad de llegar a 50km en áreas rurales [20]
- Paquetes pequeños de 12 bytes para intercambiar información con límite de 6 mensajes por hora [21] (cumpliendo con la regulación ETSI europea)
- Utilización de banda sub-GHz a 868MHz y modulación BPSK. Los mensajes se envían en un ancho de banda de 100Hz

Como inconveniente de esta tecnología sería la necesidad de utilizar los servicios de la empresa como intermediarios y la necesidad de un pago por la suscripción al servicio y el uso de la tecnología que, aunque puede personalizarse el plan según las necesidades de cada uno, es algo que debemos tener en cuenta debido al fin de este proyecto.

Por último, consideramos la tecnología **LoRa** [13] la cual es muy similar a la ya expuesta tecnología SigFox en cuanto a que utiliza la misma banda de frecuencia libre de 868MHz que existe en Europa y también busca aportar conectividad de muy largo alcance. La gran diferencia de esta tecnología frente a las demás es que el pago que hay que realizar a la compañía que ha desarrollado esta tecnología viene a cargo del fabricante que debe pagar para poder incorporar los chips en sus productos mientras que para el usuario final el servicio no requiere de ninguna suscripción ni pago extra. Es por esto que se ha considerado esta tecnología como la más apropiada para el proyecto ya que además de cumplir con los requisitos técnicos de conectividad que se requieren, nos ofrece la operación más económica de todas y con un diseño que no depende de infraestructuras ajenas.

3.2 HARDWARE Y SOFTWARE PARA EL SISTEMA

Tanto en la planificación como durante el desarrollo del proyecto se han podido evaluar distintas opciones para la implementación del proyecto. Siguiendo el esquema que se muestra en la Ilustración 2, en cuanto al hardware de los periféricos del microcontrolador, se eligieron para los sensores los 3 mencionados en el apartado 1.2.1. En concreto, el Teros 10 se eligió para ser el sensor de referencia al venir calibrado de fábrica permitiendo luego comparar sus medidas con los demás y así observar si son válidos para sustituirle.

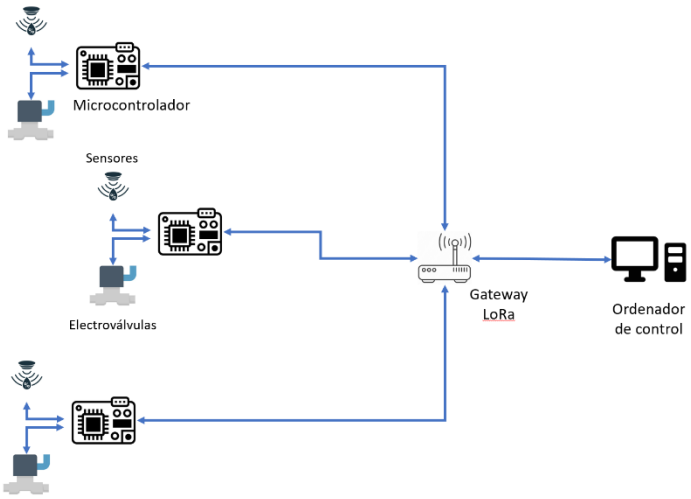


Ilustración 2 - Esquema del sistema

3.2.1 MICROCONTROLADORES

Durante la fase de planteamiento de cómo se comunicarían los dispositivos, se decidió que la inteligencia del programa recayese en la aplicación de escritorio ya que actúa como un nodo central en la red. De esta manera, el microcontrolador en nuestro caso no requiere de una gran capacidad de cómputo ni puertos específicos. Mirando las soluciones existentes en el mercado para un caso como el nuestro, las alternativas más considerables serían: Arduino, Raspberry Pi, o algún microprocesador como el ESP-32.

Raspberry Pi son una serie de ordenadores SoC[2] que se enfocan en la versatilidad y el bajo coste. Por esta misma razón son una buena opción para múltiples tipos de proyectos. Estos junto a productos de placas auxiliares con módulos y puertos pueden hacer que esta sea una buena opción. El modelo más adecuado para nuestro caso sería el Raspberry Pi Zero o el Pico. Sin embargo, no incluyen hardware de comunicación con el LoRa y una shield apropiada no estaba disponible debido a los sucesos recientes de escasez de chips además de aumentar considerablemente el precio del sistema. Esto junto a una capacidad de cómputo exagerada para nuestro caso, hace que sea una opción inferior a las demás.

Otros procesadores como el ESP-32 que suelen ser muy populares también, nos generan el mismo problema que la Raspberry Pi al requerir de algún tipo de shield o placa auxiliar para instalar el módem de LoRa.

Por estas razones, se decidió optar por una palca de Arduino como es la MKR WAN 1300, la cual nos proporciona directamente el módem, y nos deja el resto de los pines libres para poder interactuar con los sensores y válvulas que podamos conectar. Presenta también una gran cantidad de librerías que nos ayudarán en gran medida durante la programación para interactuar con el módem o el módulo RTC que también tiene este modelo. Además, resulta ser la opción que en conjunto es más económica, costando el kit entero menos de 40 euros.

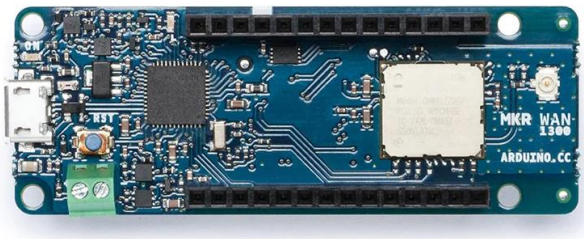


Ilustración 3 - Arduino MKR WAN 1300

[2] System on a chip

3.2.2 GATEWAY LORA

El proyecto se empezó con el Indoor Gateway de The Things Industries ya que al ser una de las empresas que soportan la red de LoRa parecía una opción muy buena. Sin embargo, recientemente han cambiado de plataforma a una que a pesar de que tiene una opción gratis, en el caso de querer algún tipo de soporte, requiere de una suscripción, lo cual es algo que se quiere evitar a toda costa en el proyecto ya que uno de los puntos principales es la búsqueda del menor coste posible.

Por ello, se investigaron alternativas que sustituyesen el uso de la plataforma. La mejor sería utilizar Chirpstack [22], un software open-source que nos permitiría crear un servidor de red propio que recibiese los mensajes del LoRa y así ignorar la plataforma por completo. Este software se podría instalar en un dispositivo ajeno tanto al microcontrolador o el PC. Especialmente si usamos como Gateway una Raspberry Pi con un módulo LoRa como se indica en la documentación de Chirpstack, podríamos levantar los servidores de Red, Aplicación y el Gateway Bridge dentro de este (incluso una base de datos para guardar mensajes), y así hacer las aplicaciones aún más simples y permitiendo una comunicación asíncrona desde el punto de vista del PC y el microcontrolador. Sin embargo, esta opción en concreto se encontraba fuera de stock durante el desarrollo del proyecto y otras alternativas soportadas por Chirpstack se salían de presupuesto al ser Gateways empresariales por lo que su precio superaba los 400 euros [23].

Habiendo mirado otras opciones, se decidió volver a la plataforma de The Things Industries, esta vez a su nueva versión TheThingsStack utilizando su versión comunitaria que no requiere de pagos.

3.3 PROYECTOS RELACIONADOS

Este proyecto pretende culminar el sistema de riego de explotación agrícola desarrollado en los trabajos de Miguel Campano García [1] y Luis Carlos Parra Rebollo [2] donde cada alumno utilizó una tecnología para la comunicación y sistemas diferentes. El uso de la tecnología LoRa además de un enfoque mayor en la eficiencia energética y de coste, se buscará perfeccionar el sistema y obtener un producto final apropiado para su distribución y uso.

Existen una gran cantidad de proyectos de nivel doméstico que intentan incorporar la tecnología LoRa para un pequeño jardín, pero su adopción en la industria no está desarrollada en gran medida. Existe otro caso [24] en el que un grupo de estudiantes creó un sistema similar pero su implementación fue solo una prueba de concepto y se instaló en los jardines del edificio donde realizaban el proyecto sin llegarse a implementar a gran escala. Además de los trabajos de los compañeros en Valladolid solo se ha podido encontrar uno realizado por un grupo de ingenieros en la India (R. K. Kodali, M. S. Kuthada y Y. K. Yogi Borra) [25] donde diseñaron un sistema similar que realizaba la conectividad de los sensores por LoRa hacia un dispositivo localizado en el terreno y este se conectaba por WiFi al resto del sistema. Sin embargo, no se ha podido encontrar ninguna solución diseñada por una empresa a una gran escala.

Capítulo 4. DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN

Viendo los trabajos anteriores realizados con la tecnología LoRa alrededor de monitorización de terrenos y riego, existe una clara diferenciación entre proyectos que pretenden ser una demostración de concepto y prueba de la tecnología como es la de los estudiantes de Galicia [24], y otros que buscan crear un producto real como es el caso de los compañeros de Valladolid, la India, y mi caso. Siendo este proyecto una revisión y avance con respecto a los trabajos de Valladolid, considero que este trabajo se compara más con el de último, donde se crea un sistema más similar al que se ha propuesto. Sin embargo, en este proyecto solo se comenta el esquema seguido sin especificar sobre su funcionamiento concreto, y los resultados explicados son poco precisos y generales. A continuación, se explicarán varios puntos donde nuestro trabajo destacará frente a los anteriores. Como se verá, algunos de los puntos estarán presentes en alguno de los trabajos anteriores. La inclusión de todas estas ventajas hará el sistema desarrollado más avanzado y maduro como producto final.

4.1.1 APLICACIÓN DE ESCRITORIO PARA EL CONTROL DEL SISTEMA

Es el caso de los trabajos anteriores de Valladolid, donde uno de los enfoques principales de sus trabajos fue el desarrollo de una aplicación que permitiese interactuar con el nodo final pudiendo manejar la distribución de estos. Sin embargo, en este caso se ha desarrollado una aplicación tal que la funcionalidad ha quedado lo más abstraída posible con el fin de habilitar su uso por el mayor número de personas posible, tengan o no conocimientos informáticos.

Además, el uso de protocolos ligeros como MQTT que están pensados para despliegues IoT permiten que tanto la aplicación como la comunicación entre plataformas sea mucho más ligera, limitando el uso tanto de banda ancha como de memoria.

4.1.2 ESTUDIO DE SENSORES

Algo completamente innovador frente a otros proyectos, será el estudio de diferentes sensores de humedad, estudiando su efectividad a distintos niveles, y determinando y adaptando aquellos que puedan ser útiles para el producto final. Para esto, se utilizarán componentes de calidad con reconocimiento previo a partir del cual se examinarán opciones más económicas. Esto profundizará en una de las motivaciones del proyecto que era la eficiencia económica y la disponibilidad del proyecto al público.

4.1.3 AUTONOMÍA

Algo más importante como es el medio de comunicación de los nodos finales con el ordenador de control, es la alimentación de estos. En el caso de trabajos como el de Kodali, Kuthada y Yogi Borra, a pesar de tener un nodo que en principio podría alimentarse utilizando una batería, se puede observar en la Ilustración 4, que para actuar la bomba que riega el terreno requiere de alimentación alterna al *relay* para poder actuarla.

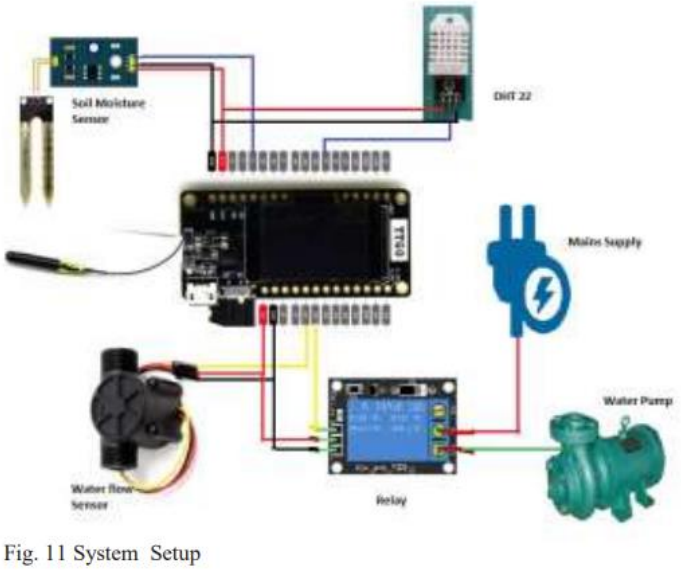


Fig. 11 System Setup

Ilustración 4 - Disposición de nodos por Kodali, Kuthada y Yogi Borra

Esto requeriría de un despliegue de cableado que en algunos casos para algunas explotaciones no es posible tanto por razones económicas como físicas. Tanto por el tratado

continuo de la tierra siendo revuelta y movida para airearla u otros procedimientos, puede ser necesario una alimentación autónoma in situ, que, en caso de ser renovable, puede eliminar preocupaciones medioambientales de este sistema. Es por esto que para nuestro sistema se crearán los circuitos y adaptaciones necesarias para que esto sea posible como se explicará tanto en los Objetivos, como a lo largo del Desarrollo del Sistema.

4.1.4 PUBLICACIÓN Y EXPOSICIÓN

Una de las principales motivaciones del proyecto como se indicó, es la disponibilidad y el alcance para el mayor número de personas posibles. Por esto, tanto el desarrollo como los resultados intentarán ser explicados y mostrados con la intención de facilitar la replicación de este proyecto y el desarrollo de este hacia futuras mejoras y aplicaciones.

4.2 OBJETIVOS

Los objetivos del proyecto han sido modificados a lo largo de su desarrollo debido a la colaboración con el INEA. En cuanto a los objetivos finales acordados, en primer lugar, consiste en la creación de un prototipo funcional del sistema descrito. Este será uno formado por una placa Arduino con capacidad de comunicación LoRa a la que se le conectarán una serie de sensores de humedad y electroválvulas de riego. Junto a un diseño electrónico capaz de adaptar tanto las alimentaciones y señales recibidas de los distintos periféricos, será necesario crear un circuito regulador que nos permita alimentar la placa con una fuente de 12V la cual será una placa solar conectada a una batería.

Por otro lado, también se requerirá el desarrollo software del sistema tanto por parte de la placa Arduino, como una aplicación en Python que permita la visualización de datos históricos sobre la humedad del terreno como el establecimiento de un horario para el riego del terreno.

El diseño físico se buscará traspasar a un diseño en una PCB que permita su fabricación en serie y poder ser producido de una manera más eficiente y consistente.

Por último, se pretende probar el funcionamiento del sistema en condiciones reales por lo que se intentará implementar en una explotación real en pruebas de campo para verificar su completo y adecuado funcionamiento y puesta a punto.

4.3 METODOLOGÍA Y PLAN DE TRABAJO

El desarrollo del proyecto se realizará de manera muy secuencial ya que sin un prototipo que funcione no se podrá seguir. Para el desarrollo de la placa de Arduino y su interconexión con los componentes se utilizarán tanto herramientas físicas como de simulación. De cara a la programación del Arduino, se utilizará la aplicación propia para el desarrollo. En cuanto a la conexión de componentes y su alimentación externa como es para la electroválvula, se utilizarán programas de simulación como LTSpice que nos permitirán hacer un primer diseño antes de implementarlo y, tras esto, comprobar su funcionamiento en el laboratorio con la ayuda de multímetros y osciloscopios. El esquema de conexión y diseño de PCB serán capturados a través de KiCad.

Para la parte del servidor, se desarrollará en Python utilizando diferentes librerías existentes. Para la interfaz de usuario se utilizará la librería de PySimpleGUI, para la representación de datos de humedad de la placa se utilizará matplotlib, y en cuanto a la organización de horarios se utilizarán librerías de fechas como dateutil. En cuanto a la interconectividad con el Arduino, la placa se comunicará con un Gateway de TheThingsNetwork el cuál traducirá el mensaje a IP y lo enviará a la plataforma de TheThingsStack donde monitorizamos las comunicaciones. A partir de aquí, para comunicarnos con la aplicación que crearemos se utilizará la integración existente de MQTT con subscripciones a los distintos eventos que se generarán tanto de envío de datos como recepciones.

La plataforma de TheThingsStack nos serviría en caso de realizar el proyecto a un nivel más de usuario ya que la edición comunitaria de esta infraestructura es gratuita. Sin embargo, como el objetivo de este proyecto sería poder ser implementado a un nivel mayor, esto requeriría de un pago por lo que como opción alternativa se intentará utilizar ChirpStack que es un servicio completamente open-source y tiene prestaciones muy similares a las de TheThingsStack. Habrá que valorar también si el coste del diseño y mantenimiento propios superan el coste de una suscripción al servicio en una aplicación de carácter industrial.

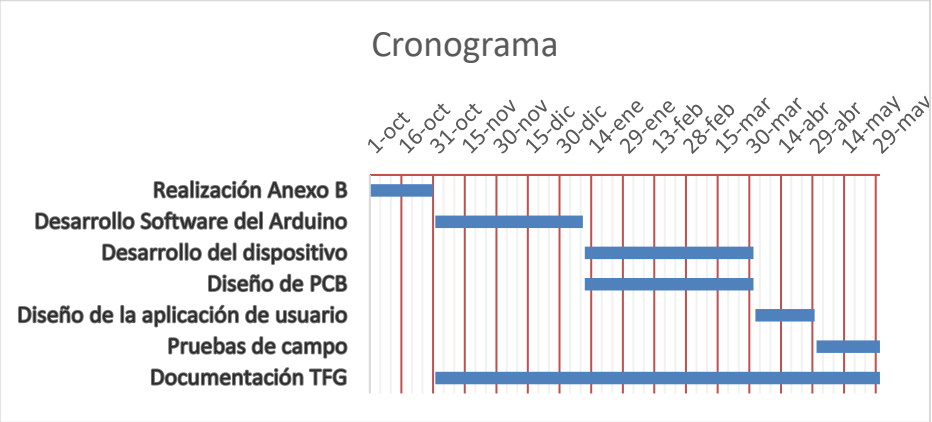


Ilustración 5 - Cronograma de planificación del proyecto

4.4 ESTIMACIÓN ECONÓMICA

En este caso, como uno de los aspectos a los que le damos más importancia es el coste económico, presentamos las distintas alternativas que se han visto con sus respectivos costes para que, en el caso de ser replicado, pueda elegirse según convenga más al futuro usuario.

Tabla 1 - Estimación económica del proyecto

Componente	Número	Precio Unitario[3]	Precio Total
Arduino MKRWAN 1300	1	40,3 €	40,3 €
Sensor Teros 10	1	144 €	144 €
Sensor Cap. Seeduino	1	6,47 €	6,47 €
Sensor Resis. Elecfraks	1	3,47 €	3,47 €
Regulador L7805	1	0,85 €	0,85 €
Resistencias	9	0,21 €	1,89 €
Resistencias de baja tolerancia	6	0,17 €	1,02 €
Diodos Schottky	6	0,54 €	3,24 €
Condensadores	4	0,63 €	2,52 €
Driver L298	1	9,62 €	9,62 €
Diodos	4	0,54 €	2,16 €
Transistores 2N3906	3	0,29 €	0,87 €
Transistores 2N3904	3	0,29 €	0,87 €
Terminal de tornillo	5	0,6 €	3 €
Gateway LoRa	1	113,89 €	113,89 €
Panel Solar + cargador simple	1	70 €	70 €
Batería LFP	1	55,9 €	55,9 €
Electroválvula	1	43,68 €	43,68 €
		Total	**503,75 €**

[3] Los precios han sido seleccionados por el precio de venta en suministros de 1 por lo que pueden aparecer mayores de lo que se puede obtener al comprarse en mayores cantidades. En caso de no haber de 1, se ha escogido el más barato en la cantidad menor.

En esta tabla podemos observar como el elemento más costoso es el sensor de humedad Teros. Debido a esta razón principalmente, se decidió hacer el estudio de distintos sensores buscando alternativas más económicas. Por otro lado, también tenemos otros elementos que ocupan una gran parte del presupuesto como es el Gateway LoRa. En la tabla se ha puesto el precio actual el cual ha subido considerablemente desde el inicio del proyecto. En esto, además del requerimiento de trabajar con la plataforma de TheThingsStack, es lo que nos llevó a considerar utilizar Chirpstack y otras alternativas como se explica en la sección 3.2.2.

El precio final del proyecto puede parecer alto para buscar ser un producto finalizado, pero teniendo en cuenta que se han tomado precios de compra en cantidades muy limitadas, tanto los componentes como los sensores presentan una gran reducción en el coste por unidad si se compran en grandes cantidades. Incluso el Teros 10 tiene opciones si se compra en cantidades mayores. Esto añadiendo el desarrollo de una PCB que permitiría el montaje más rápido y barato, permitiría hacer el proyecto mucho más alcanzable y disponible.

Capítulo 5. DESARROLLO DEL SISTEMA

En este capítulo se describirá lo realizado a lo largo del proyecto separado en secciones que pueden o no estar en orden cronológico.

5.1 CIRCUITO DE ALIMENTACIÓN DEL MICROCONTROLADOR

Como se ha explicado anteriormente, el objetivo final de este trabajo es que el sistema se alimente a través de un sistema autónomo de alimentación. Especialmente pensando en una alimentación de origen renovable y simple, se basaría en una pequeña placa solar y una batería. Esta al proporcionar una tensión de 12V, presenta un problema para el Arduino ya que el MKRWAN 1300 requiere de una alimentación de 5V estable en caso de alimentarse por una fuente que no sea el USB incorporado [3].

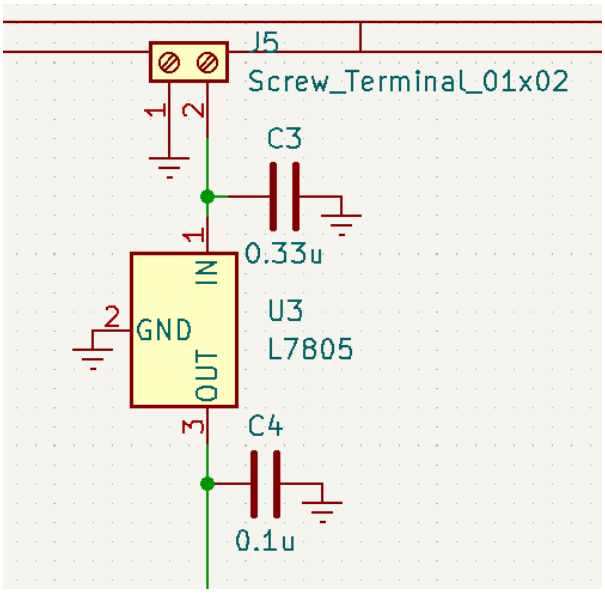


Ilustración 6 - Circuito de regulación de tensión

El circuito que se ha diseñado es uno muy simple basado en el regulador de tensión a 5V L7805 [5] al que le acoplamos un par de condensadores siguiendo lo indicado en la *datasheet*. Tras esto, la salida se conecta directamente al pin *VIn* de entrada del microcontrolador y la entrada a un terminal de tornillo al que conectaremos la batería. A lo largo de las pruebas que se realizan en el laboratorio, este terminal se conecta a una fuente de alimentación a 9V.

5.2 CIRCUITO DE CONTROL DE LAS ELECTROVÁLVULAS

Las electroválvulas son periféricos que requieren de mayor tensión y potencia de la que puede dar el microcontrolador por los pines digitales. Concretamente, la válvula utilizada requiere de un pulso de 9V para abrirse y uno de -9V para cerrarse. Es por esto que se ha decidido utilizar el driver L298 [6] que contiene dos controladores *full-bridge*. Debido a sus características, con cada controlador podremos abrir y cerrar una válvula dependiendo de que pines activemos y cuáles no. Para esto, se ha seguido el ejemplo propuesto en el datasheet [6] (Pág 6, Fig 6) sin conectar ninguna resistencia para medir la corriente al pin 15. En caso de querer conectar más válvulas, solo habría que usar el otro controlador y si se quisiesen usar aún más, replicar el circuito.

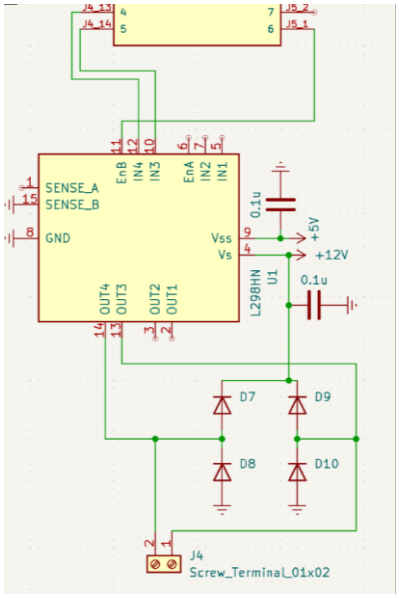


Ilustración 7 - Circuito de control de válvulas

Dentro del circuito que se puede ver en la Ilustración 7, cabe destacar los 4 diodos que hay entre la terminal de tornillo y el controlador. Estos tendrán como función descargar la bobina que hay dentro de la válvula. En caso de utilizarse este circuito para alimentar dispositivos similares basados en bobina, se debe observar que la bobina se descargue cada iteración para asegurar que no se sature.

5.3 *ESTUDIO DE DIFERENTES SENSORES EN EL MERCADO*

Aplicando el aspecto buscado de eficiencia económica, se han estudiado diferentes sensores que se podrían utilizar para monitorizar la humedad del terreno en una explotación agrícola. Para esto, se han utilizado tres sensores distintos: el sensor de humedad Teros10 como referencia, un sensor capacitivo y uno resistivo, ambos de bajo coste.

En el caso del Teros10, mide la permitividad eléctrica del terreno. Esto tiene sus pros y sus contras como que la presencia de ciertas sales en el terreno puede afectar a las lecturas en este tipo de sensores. Sin embargo, el Teros10 utiliza una frecuencia de 70MHz para realizar las lecturas lo que le permite minimizar este tipo de efectos. Debido a esto y otros factores de su construcción es un sensor muy confiable de gran precisión cuya documentación nos proporciona ecuaciones que nos permiten obtener el contenido volumétrico de agua (VWC en inglés) según el tipo de suelo en el que estemos colocando el sensor. Una vez alimentado correctamente y aplicada la ecuación 4 del *datasheet* [7] que se muestra a continuación, se puede leer la humedad directamente a través del monitor serie de Arduino. La tensión indicada en la ecuación es la del sensor medida en mV.

Ecuación 1 - Ecuación 4 de calibración Teros 10

$$\Theta\left(\frac{m^3}{m^3}\right) = 5.439 \times 10^{-10} \times V^3 - 2.731 \times 10^{-6} \times V^2 + 4.868 \times 10^{-3} \times V - 2.683$$

Es importante tener en cuenta que esta ecuación ha sido usada debido a que las pruebas se han realizado en una maceta en un laboratorio. Para su implantación real, se debe utilizar la ecuación 2 dada. Estas ecuaciones presentan una precisión más que necesaria para nuestro

caso, pero en caso de requerir una aún mayor, el fabricante recomienda realizar nuestra propia calibración.

Tras conseguir el correcto funcionamiento del Teros 10, se pasó a realizar diferentes medidas a diferentes niveles de humedad con el objetivo de realizarlas a la vez con los otros dos sensores y así poder construir una curva de calibración para cada uno que nos permitiese utilizarlos en vez del Teros10 ya que los sensores utilizados serían mucho más económicos que este. Las medidas se realizaron en una maceta donde, en primer lugar, se media con el Teros 10. Después, se sacaba y se introducía el siguiente asegurándonos de que no queden bolsas de aire cerca del sensor y así.

En la Ilustración 8, se pueden observar las tensiones que se han obtenido a cada nivel de humedad comparando los niveles de humedad que se han obtenido con el sensor de referencia, frente a los dos utilizados.

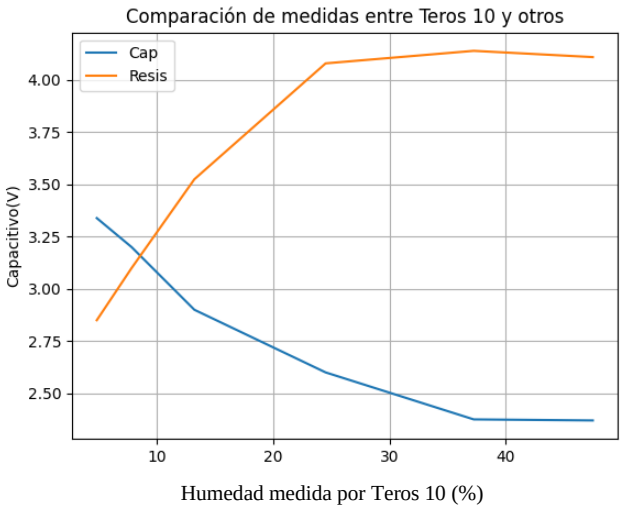


Ilustración 8 - Comparación de tensiones entre Teros10 y otros

Analizando los resultados, el sensor capacitivo utilizado que en nuestro caso es el *Grove capacitive Moisture Sensor* [9], se ha mostrado útil para determinar la humedad dentro del rango entre 0 y 30 %. A partir de este valor, el sensor parece saturarse y no consigue

diferenciar mayores concentraciones de agua en el sustrato. Por esto, dependiendo del caso de uso, podría ser una opción real que utilizarse. En concreto, analizando las distintas medidas obtenidas, se hizo una regresión polinómica de tercer orden obteniendo la Ecuación 2, donde introduciendo la salida del sensor capacitivo, se debería obtener un resultado similar al del Teros10 dentro del rango mencionado.

Ecuación 2 - Regresión polinómica para sensor capacitivo

$$\Theta\left(\frac{m^3}{m^3}\right) = 1281.605 - 1149.338 \times V + 348.9791 \times V^2 - 35.70235 \times V^3$$

Por otro lado, el sensor resistivo *Octopus Moisture Sensor* de *Elecfreaks* [8] ha resultado ser bastante impredecible en las lecturas. En la Ilustración 8, se muestra una versión obtenida a través de la realización de varias medidas, obteniendo la media y en algún caso eliminando casos extremos. Moviendo el sensor ligeramente en cualquier sentido, introduciéndolo o sacándolo 1mm o menos genera grandes cambios (variaciones de 1V) en las lecturas lo que lo hace poco fiable para prácticamente cualquier uso que no fuese binario en cuanto a detectar casos extremos como el encharcamiento del suelo o situaciones parecidas. En caso de ser útil para alguien que intente recrear el experimento, la ecuación utilizada para adaptar las tensiones a las del Teros10 desde el sensor resistivo es la Ecuación 3.

Ecuación 3 - Regresión polinómica para sensor resistivo

$$\Theta\left(\frac{m^3}{m^3}\right) = -348.004 + 358.8872 \times V - 122.6154 \times V^2 + 14.12773 \times V^3$$

5.3.1 CIRCUITO DE ADAPTACIÓN DE LOS SENSORES

En nuestro caso, cada sensor presentaba un rango diferente de tensiones disponibles para su actuación y posterior lectura. En el caso del Teros10, este rango va desde 3V hasta 15V mientras que para los otros dos sensores estudiados es entre 3.3V y 5V. Esto implicaría que podríamos haber utilizado los puertos digitales del Arduino directamente para alimentar los sensores, pero previamente se estaba intentando utilizar un sensor que requería 5V para su

funcionamiento por lo que se decidió diseñar un circuito que nos permitiese alimentar estos sensores con 5V. Este consiste en un par de transistores 2N3904 y 2N3906 a los que actuaremos con el pin digital del Arduino activando el sensor que haya conectado a la terminal de tornillo. El esquema de conexión para cada sensor será como está mostrado en la Ilustración 9: 1:pin de alimentación; 2:salida del sensor; y 3:tierra.

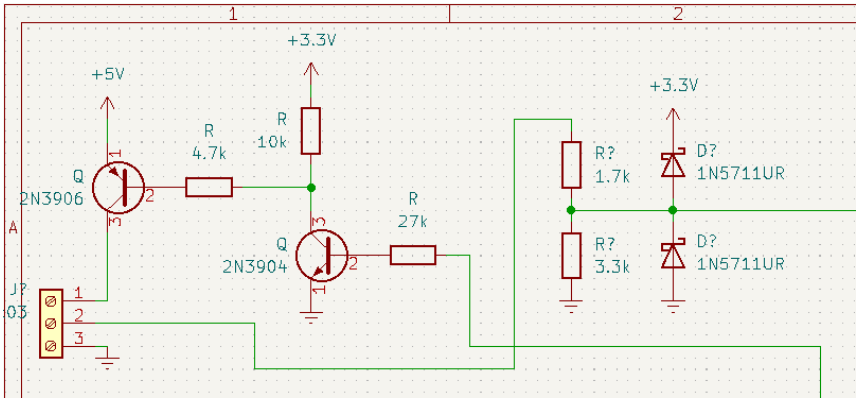


Ilustración 9 - Circuito de adaptación de sensores

Una vez activado el sensor, la señal de lectura se hace pasar por un pequeño circuito de adaptación y protección que previene que llegue al Arduino una señal de una tensión excesiva. Este circuito consiste en un divisor de tensión que permite adaptar una señal de 5V a 3.3V. Además del divisor, como último frente, se han utilizado un par de diodos Schottky. Estos diodos en conjunto evitarán el caso extremo de una señal mayor de 5V. La elección concreta de utilizar diodos de este tipo en vez de normales es debido a su baja pérdida de conducción, esto nos permite ajustar el límite de tensión mejor a los 3.3V ya que las pérdidas causan que el punto de actuación del circuito sea menor de los 3.3V que se desean. Una vez pasado por esto, la señal se conecta directamente a un puerto analógico del Arduino.

5.4 *ESQUEMA HARDWARE FINAL*

Una vez conectados todos los elementos en la placa, queda como se puede ver en la Ilustración 10. Un apunte sobre la figura es que se puede ver que el microcontrolador que aparece es el Arduino MKR Zero en vez del MKR 1300, esto se debe a que la librería de KiCad no tenía incluido nuestro modelo, pero, por suerte, se encontró en una página el modelo del MKR ZERO, que tiene la misma disposición de puertos que el MKR WAN 1300 y las mismas dimensiones, por lo que sirve de cara al esquema, y el diseño de la PCB.

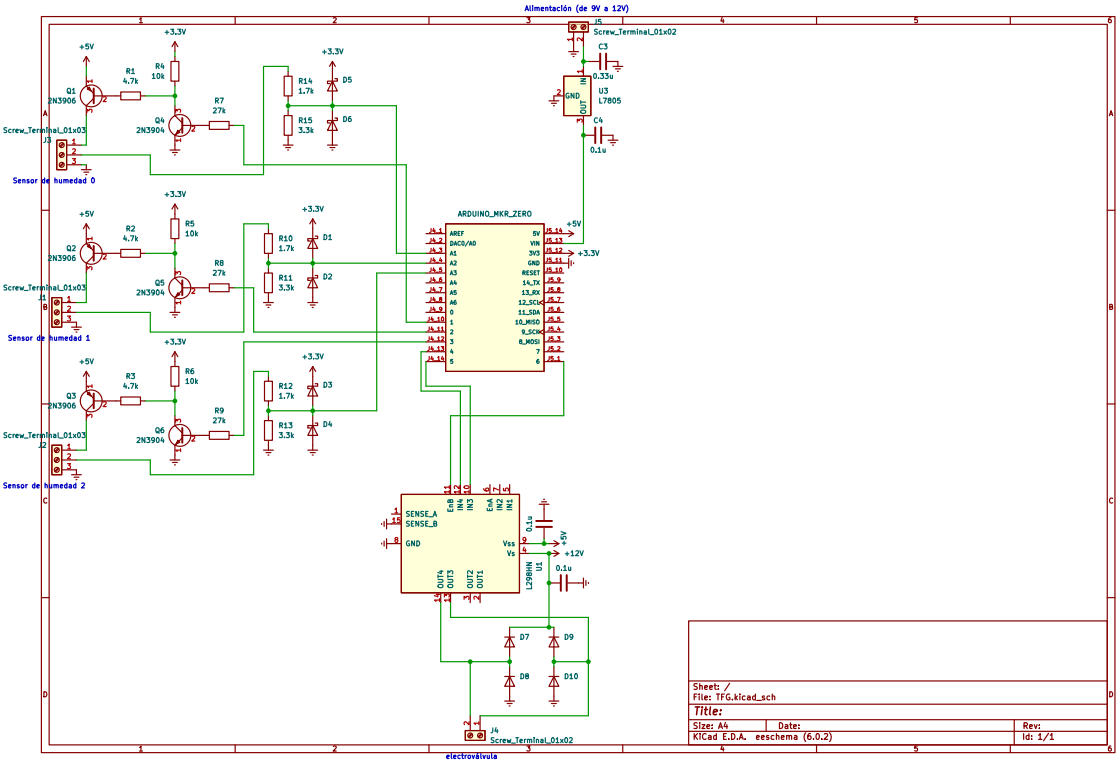


Ilustración 10 - Esquema final

La disposición física se puede ver en la Ilustración 10 y la Ilustración 11, donde se podrá observar que la disposición en el esquema en kiCad es la misma que en la protoboard. Esta nos ha permitido separar la parte de alimentación con la válvula, y la de los sensores al funcionar estos a diferentes tensiones. En la palca al estar separados una gran distancia no es necesario, pero para luego pasar el diseño a la PCB es más deseable.

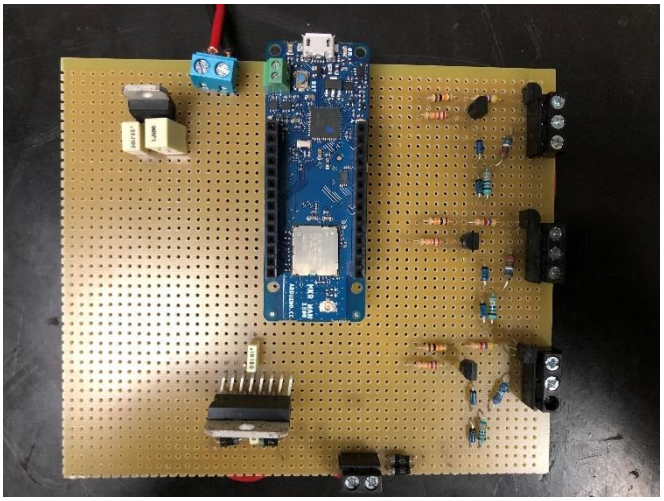


Ilustración 11 - Protoboard del proyecto por delante

5.5 SOFTWARE DEL MICROCONTROLADOR

5.5.1 COMUNICACIÓN LORA

Es en esta parte en la que se ha tenido la oportunidad de trabajar con la tecnología LoRa explicada en apartados anteriores. Esta especificación nos proporciona distintas maneras de operar entre el microcontrolador y la plataforma. Primero se debe determinar en qué modo quiere uno que el dispositivo funcione. Dentro de la especificación, se describen tres clases de dispositivos: clase A, B y C [26, p. 8]. La principal diferencia entre ellos es cuándo el dispositivo atiende a recepción de paquetes. En el caso de la clase A, solo permite la recepción inmediatamente después de enviar este un paquete, haciéndolo óptimo para sistemas donde la inteligencia este en un nodo central y los dispositivos solo actúen como transmisores de datos de un sensor, por ejemplo. La clase B permite abrir la ventana de recepción periódicamente a través de pings regulares, lo que permite realizar más fácilmente pequeñas correcciones o acciones de control sobre los nodos externos de nuestra red. Por último, la clase C destaca por poder abrir su ventana de recepción de manera continua. Aunque estas características den a entender que la clase C es superior, hay que tener en cuenta que abrir la ventana de recepción de mensajes aumenta el consumo y es por esto que

la clase C en especial solo se recomienda en caso de ser estrictamente necesario, ya que en algunos casos puede incluso infringir la legislación local con respecto a ocupación del espectro radioeléctrico. En nuestro caso, se decidió utilizar el dispositivo en modo de clase B ya que requerimos de poder ir subiendo y modificando el esquema de riego en función de nuestras necesidades.

Otro elemento a tener en cuenta a la hora de desarrollar nuestro programa es el modo de conexión del microcontrolador con la plataforma de TheThingsStack. En el caso de LoRa, existen dos métodos [27]; *Over-the-air-activation* (**OTAA**), y *Activation By Personalization* (**ABP**). En el caso de ABP, las direcciones utilizadas para conexión son escritas en el dispositivo, mientras que con OTAA las direcciones y claves son negociadas entre la red y el dispositivo aportando una mayor seguridad. Es por esto que por defecto se recomienda el uso del modo OTAA, ya que no requiere mucho mayor esfuerzo que usar ABP.

Una vez teniendo claro cómo va a actuar nuestro módulo LoRa, para conectar el microcontrolador con la plataforma hay que registrarlo utilizando su *DevEUI* y obtener una *APP-EUI* y una *APP-KEY*. Estos valores se deberán guardar en un archivo aparte y se utilizarán al arrancar el Arduino para que se conecte.

Para configurar y operar el módem se utiliza la librería de ***MKRWAN*** [28] que incluye todas las funciones que podamos necesitar.

5.5.2 USO DEL MÓDULO RTC

Este modelo concreto de Arduino incluye en su procesador un módulo RTC que nos permitirá organizar más simplemente el esquema de riego que recibamos. Para el manejo del módulo, se utiliza la librería de ***RTCZero*** [29] que incluye las funciones para establecer la hora, y poner alarmas que activaran interrupciones. Esto será lo que utilizaremos para actuar o no las válvulas.

5.5.3 EXPLICACIÓN DEL CÓDIGO

El código del programa se puede encontrar en el ANEXO I. Sin embargo, explicaremos aquí alguno de los bloques. El bucle principal consistirá en una serie de etapas por las que pasará con el fin de conectarse correctamente y poder entrar en el ciclo de envío y recepción. Se puede ver este ciclo de una manera muy simplificada en el siguiente diagrama:

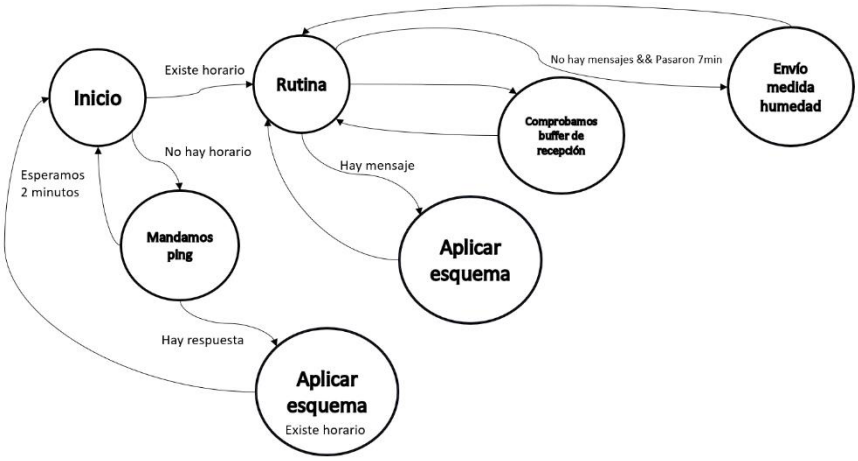


Ilustración 12 - Diagrama del bucle principal del programa

En este, se comprueba si se ha recibido ya algún esquema, en caso de no haberse recibido, enviará un paquete cada 2 minutos a modo de ping abriendo así su ventana de recepción permitiendo que llegue el primer esquema. Esto se comprueba con la variable "con_horario". Una vez recibido, ordenará cronológicamente las acciones programadas y pasará a la rutina principal. Dentro de esta, se comprobará si se ha recibido algún otro paquete y en caso de que pase se aplicará como se hace con el primero. En caso de no haber paquetes, se habrá lanzado un temporizador que, al llegar a 7 minutos, mandará un mensaje con el valor actual de la humedad. En caso de recibir un mensaje, se tratará como al principio y se procesará.

5.5.3.1 Recepción de paquetes

Para las comunicaciones desde el PC al Arduino, se ha diseñado un paquete que contará con secciones para pasarle un esquema de riego, además de acciones instantáneas de apertura y cierre de las válvulas. El paquete se estructura de la siguiente manera:

bit	0	1	2	3	4	5	6	7
0	Abrir V0	Cerrar V0	Abrir V1	Cerrar V1	Abrir V2	Cerrar V2	Abrir V3	Cerrar V3
8	Gap V0							
16	Tiempo de riego V0							
24	Gap V1							
32	Tiempo de riego V1							
40	Gap V2							
48	Tiempo de riego V2							
56	Gap V3							
64	Tiempo de riego V3							

Tabla 2 - Estructura de paquete LoRa

Se puede ver en la sección de los datos del paquete (las cabeceras no se han incluido) que es de solo 9 bytes, esto nos permitirá poder enviar más paquetes sin llegar a romper los límites por estar utilizando una frecuencia de radio libre. El primer byte incluye las acciones instantáneas en cada válvula, pudiendo abrir o cerrar. En caso de poner a 1 el bit tanto de apertura como cierre, la válvula quedará cerrada. Por otro lado, se puede especificar cada cuántas horas se querrá regar, y cuánto tiempo. Este paquete se analiza en la función *manageSchedule* en la cual simplemente se recorre el paquete y si se ha rellenado el campo de *gap* y de *tiempo de riego*, se guardará en una estructura y ésta en un vector. Una vez recorrido todo, se ordenarán de manera cronológica y se pone el RTC a 0 para que cuando se cumpla el primer *gap*, se ejecute la acción. Las acciones se guardan dos veces; una vez como encendido de válvula a la hora del *gap*, y otra a la hora de *gap* + *duración* como cierre. Así, cuando pasen las horas especificadas, la válvula regará, y una vez haya pasado el tiempo que se quiere regar, se cerrará.

5.5.3.2 Ejecución de acciones

Como se explica en el apartado anterior, la función *manageSchedule* es la que se encarga de organizar el esquema que se recibe por el paquete. Una vez cargados todos los valores ordenados en el vector global de *acciones*, se habilitan las interrupciones del RTC. Estas se ejecutarán si hay un match entre el valor actual del RTC y la hora puesta en la alarma. Esto se realiza con las funciones que hay en el loop de *setAlarmTime*, *enableAlarm*, y *attachInterrupt*. Una vez hecho el setup por primera vez, el Arduino es capaz de mantener el esquema y manejarlo sin ninguna intervención. La lógica se encuentra en la propia función de interrupción que se puede ver en el Extracto de código 1.

Extracto de código 1 - Interrupción de RTC

```
void alarma(){
    Serial.println("Ejecutando alarma: ");
    mostrarHorario(acciones[0]);
    actuarElectrovalvula(valvulas[acciones[0].id_periferico*2],
valvulas[(acciones[0].id_periferico*2)+1],
enableValvula[acciones[0].id_periferico], acciones[0].on_off);
    PeripherialAction nextProgram;
    nextProgram = acciones[0];
    if(nextProgram.gap+rtc.getHours()<24){
      nextProgram.hora.hour = nextProgram.gap +rtc.getHours();
    }else{
      nextProgram.hora.hour = nextProgram.gap +rtc.getHours()-24;
      nextProgram.nextDay += 1;
    }
    while(inAcciones(nextProgram)){
        nextProgram.hora.second += 10;
    }
    acciones[0] = nextProgram;

    Serial.println("Ordenando");
    sortActions();
    Serial.println("Terminado de ordenar");

    for(int j=0; j<indice_acc; j++){
      mostrarHorario(acciones[j]);
    }

rtc.setAlarmTime(acciones[0].hora.hour,acciones[0].hora.minute,acciones[0].hora.s
econd);

    return;
}
```

En este se puede observar como cuando salta la interrupción, ignorando las partes de debugging y monitorización, activamos la válvula que hay en la primera posición del vector de *acciones* tanto para abrirla como cerrarla. Para localizar el puerto del Arduino que hay que activar nos apoyamos de una serie de vectores constantes globales que se crearon con los pines asignados a cada sensor y válvula, además de el pin de *enable* del controlador.

En siguiente lugar, creamos una nueva acción que será la que sustituye a la que se acaba de ejecutar. Se ha decidido hacerse de esa manera ya que, como la estructura elegida es tal que se especifican las separaciones entre riegos en vez de momentos exactos de riego, lo que tenemos es una lista que al ejecutarse un elemento se crea una nuevo igual pero desplazado en el tiempo una cantidad de tiempo igual a la separación entre riegos. Tras esto, se vuelve a ordenar el vector, y se pone el primer elemento (el que primero debe suceder) como condición de la interrupción.

A la hora de crear la nueva acción existe una lógica que lo que hace es simplemente separar ligeramente en el tiempo las acciones en caso de coincidir de manera que le pueda dar tiempo al RTC a ejecutar las interrupciones y no se salte elementos de la lista. También cabe destacar la variable dentro de la estructura de *nextDay*. Esta existe ya que en caso de dejar el sistema funcionando más de 24 horas, la variable de la hora pasará a ser menor por pasarse de 24h, pero tendrá que ejecutarse después que las que tienen un valor de hora mayor. De esta manera, si se pasa de la hora 24, se suma 1 a *nextDay*, y al ordenar el vector se tendrá en cuenta y el orden se mantendrá.

5.6 SOFTWARE DE APLICACIÓN

La aplicación de escritorio se ha pensado con el objetivo de que sea esta la que maneje los datos que se reciben periódicamente del Arduino, además de que contenga una interfaz gráfica para un manejo más simple de esta.

En primer lugar, me gustaría comentar las librerías principales que se han utilizado para desarrollar esta aplicación en Python. Para la interfaz gráfica, se debatió utilizar distintas librerías como Tkinter y PyQT5. Finalmente nos decantamos por **PySimpleGUI** [30] ya que presenta una manera más fácil y simple de disponer e interactuar con los elementos de la ventana. Para la representación de datos se ha decidió utilizar **matplotlib** [31] al ser una de las librerías más conocidas y utilizadas además de tener previos conocimientos de su funcionamiento. Por último, tal vez la más importante es la librería sobre la que basaremos la comunicación con la plataforma que en este caso es **paho-mqtt** [32] la cual nos permitirá utilizar MQTT de una manera simple y funcional.

5.6.1 CONEXIÓN CON LA PLATAFORMA THETHINGSSTACK

Una vez se realizó el setup de una cuenta en TheThingsStack, se nos proporciona un *tenant-id*. Este es el id que nos diferencia como usuario, y es con este con el que nos referiremos a nuestra aplicación y a nuestros dispositivos. El dispositivo se registra en la plataforma y nos proporcionan un appKey y un appEui los cuales serán necesarios para el Arduino para enviar paquetes a la plataforma. Una vez establecida nuestra aplicación y comprobada la conectividad, la conexión con distintas aplicaciones se puede hacer de diferentes maneras a través de las integraciones que presenta. En nuestro caso, se ha utilizado la de MQTT, para la que generará una *API key* que utilizaremos para conectarnos desde la aplicación de escritorio [33].

Esta conexión se puede ver en el código dentro de la sección del *main* mostrada en la Extracto de código 2. En este caso, para la escucha de mensajes se ha creado un hilo con un *loop_forever*, como alternativa a esto también se podría meter en un while True un *loop_once* como indica la documentación, pero como utilizamos PySimpleGUI, esta ya presenta una función bloqueante que interferiría con el resto del ejecución.

Extracto de código 2 - Conexión MQTT

```
# setup MQTT
    access = 'XXXXXXXXXXXXXXXX' #aqui se coloca la API key generada
    mqtt_address = 'eu2.cloud.thethings.industries'
    client = mqtt.Client()
    client.username_pw_set('tfg-icai@tfg-icai', password=access)
```

```
    client.on_connect = on_connect
    client.on_message = on_message
    client.on_disconnect = on_disconnect
    client.connect(host=mqtt_address, port=1883, keepalive=60)
    hilo = threading.Thread(target=mqtt_client_listener, args=(client,))
    hilo.start()
```

5.6.2 INTERFAZ GRÁFICA

Como se ha indicado antes, se ha utilizado la librería de PySimpleGUI para crear la interfaz gráfica. Para el *layout*, se distribuye en un *grid*, este se define a través de una lista, cada elemento de la lista será una fila y cada subelemento dentro de una fila creará una columna. De esta manera, nos dará mucho control sobre cómo queremos crear esta interfaz. En nuestro caso, se han creado dos pestañas, una que controla la muestra de datos, que se puede ver en la Ilustración 13, y otra que controla el envío de paquetes al Arduino, la cual se puede ver en la Ilustración 14.

En el caso de la primera pestaña, se tiene un slider y una lista con los que podremos seleccionar qué porción del histórico de mensajes queremos visualizar, se puede ver desde el último día, a los últimos meses. Por otro lado, en la segunda pestaña tendremos una serie de campos a rellenar con el esquema de riego que se quiera seguir, especificando el espacio entre riegos, y la duración de estos. En caso de desconectar una válvula y querer quitar su esquema, simplemente se debe dejar vacío y dar al segundo botón, este eliminará la sección en el siguiente envío de manera que el Arduino ignorará esa válvula.

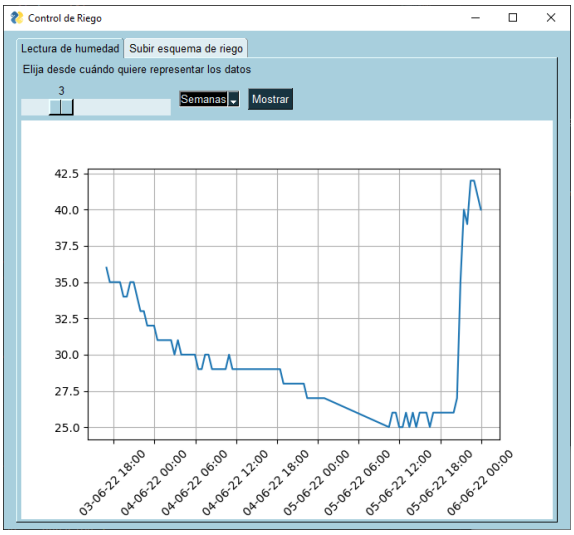


Ilustración 13 - Pestaña de lecturas

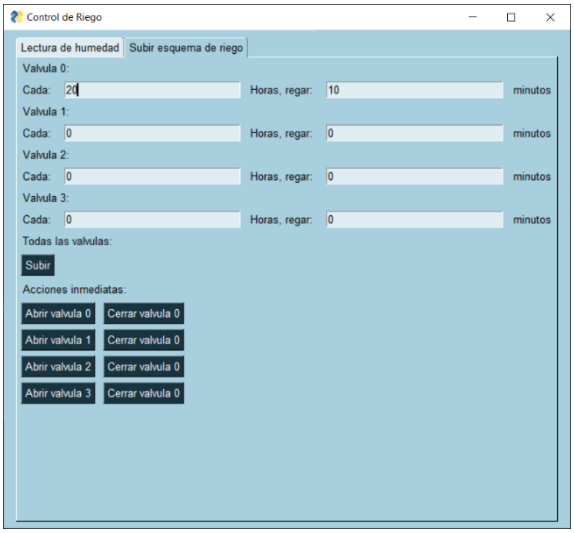


Ilustración 14 - Pestaña de subida de esquemas

En caso de solo querer abrir o cerrar alguna válvula de manera puntual, solo hará falta darle al botón correcto y en el próximo envío esta acción se llevará a cabo. Se debe tener en cuenta que las acciones instantáneas pueden tener un retraso de hasta 7 minutos ya que el Arduino sólo abrirá su ventana de recepción en estos intervalos.

5.6.3 ENVÍO Y RECEPCIÓN DE PAQUETES

Las comunicaciones como se indicaba anteriormente se realizan a través de MQTT. Al ser un protocolo de tipo publish/subscribe, tendremos que suscribirnos al canal correcto para poder enviar y recibir los paquetes correctamente. Siguiendo las indicaciones de la documentación de TheThingsStack [34], los canales seguirán un esquema como el siguiente:

- `v3/{application id}@{tenant id}/devices/{device id}`

Este será el canal general de nuestra aplicación. A partir de este, según añadamos más parámetros, accederemos a los canales de *Uplink*, *Downlink* y otros. Para nuestro proyecto, se han utilizado los siguientes dos canales para recibir los paquetes del Arduino y para enviarlos:

- `v3/{application id}@{tenant id}/devices/{device id}/down/replace`
- `v3/{application id}@{tenant id}/devices/{device id}/up`

Suscribiéndonos al canal de *Uplink* podemos interceptar todos los mensajes provenientes del Arduino, lo que nos permite recibir tanto los mensajes de ping del inicio como los mensajes periódicos con el valor de humedad del terreno. Por otro lado, si queremos enviar un mensaje, solo se tendrá que publicar un mensaje al canal de Downlink. En nuestro caso, usamos el *replace* para evitar que lleguen paquetes de mayor longitud por apilarse en la cola antes de ser entregados. En cualquier caso, se han puesto mecanismos para que, en principio, eso no ocurra.

Fijándonos más en el código, los mensajes de los canales a los que se está suscrito generan una interrupción a una función que definamos, que en este caso será la función *on_message*. Aquí separamos la funcionalidad en dos partes: la primera donde si el mensaje que se recibe es uno de los pings iniciales, se ignora. Por el otro, en caso de recibir un valor de humedad, se guarda este más la hora de recepción en un JSON con el formato {fecha y hora: humedad}.

Para el envío de paquetes, este tiene que estar sincronizado en cierta medida con el Arduino ya que la ventana de recepción no está siempre abierta. La plataforma de TheThingsStack mantiene los mensajes que reciba durante un tiempo para ser reenviados al Arduino, pero no 7 minutos. Por esto, cuando se recibe un paquete del Arduino se lanza un temporizador que a los 7 minutos enviará cualquier paquete pendiente y así será recibido sin problemas.

Para generar el paquete, esto se hace con la función de *enviarPaquete* que se puede ver en el Extracto de código 3. Previamente, cuando el usuario rellena los campos de envío y le da a subir, esta configuración se guarda en el archivo config.json, de manera que si el programa se cerrase o sólo se quisiese cambiar un valor, el resto de valores se cogerán del archivo. Una vez hecho click en el botón de subir, la aplicación se queda en modo de espera para el temporizador, y cuando termina se enviará el esquema propuesto.

Los botones instantáneos funcionan de una manera parecida, pero cuando se hace click en uno, simplemente se pone el modo de espera y cada uno se añade a una lista que se recorre

en la función de enviarPaquete donde, si está en la lista el botón, se pone a 1, y sino a 0. De manera similar, si se ha hecho click en la subida del esquema, se cogerá la configuración guardada y se añadirá al paquete. De esta manera, si uno solo le da al botón de abrir válvula 0, no se cambia el esquema actual en el Arduino, solo se ejecuta la acción. Por último, cabe mencionar que el conjunto de los datos a enviar se codifican primero en hexadecimal, y luego en base64 ya que el *payload* en los paquetes hacia la plataforma debe de estar codificado de esta manera.

Extracto de código 3 - Función enviarPaquete

```
def enviarPaquete(client):
    listaEnvio = []
    inmediatos = 0
    global listaAcciones
    for num in ('0', '1', '2', '3'):
        for letra in ('A', 'C'):
            if "".join([letra, num]) in listaAcciones:
                inmediatos = (inmediatos << 1) + 1
            else:
                inmediatos = inmediatos << 1
    inmediatos = hex(int(inmediatos))
    inmediatos = inmediatos.split('x')[1] if len(inmediatos.split('x')[1]) == 2
else '0' + inmediatos.split('x')[1]
    listaEnvio.append(inmediatos)

    if 'horario' in listaAcciones:
        conf = getConfig()
        for valor in conf.values():
            for i in valor:
                if i == '0':
                    listaEnvio.append('00')
                else:
                    elem = hex(int(i))
                    elem = '0' + elem[elem.index('x') + 1:] if
len(elem[elem.index('x') + 1:]) == 1 else elem[elem.index('x') + 1:]
                    listaEnvio.append(elem)
    else:
        for i in range(8):
            listaEnvio.append('00')

    payload = ''.join([str(item) for item in listaEnvio])
    print(payload)
    pay_b64 = b64encode(bytes.fromhex(payload)).decode()
    pay = '{"downlinks":[{"f_port": 1,"frm_payload":"' + pay_b64 + '","priority":
"NORMAL"}]}'
    client.publish(topic='v3/tfg-icai@tfg-icai/devices/arduino-tfg/down/replace',
payload=pay, qos=0, retain=False)
    listaAcciones = []
```

5.7 DISEÑO DE PCB

Para este paso, como ya se realizó el esquema en kiCad, el paso a diseñar la PCB solo requiere de asignar las huellas de los componentes según se vayan a escoger. Esto implica decidir si se van a utilizar componentes SMD o de agujero pasante (*THD*), el tamaño en el que se quiere diseñar al PCB, y el *layout* final que se esté buscando.

Para nuestro caso, se ha decidido utilizar componentes SMD en todo lo posible para la PCB por sus principales ventajas que son: tamaño reducido, y coste inferior. Es verdad que para un prototipo como el que se ha diseñado, es mejor utilizar componentes *THD* ya que es más fácil trabajar con ellos en un entorno de pruebas y desarrollo, pero para un proyecto que busca ser fabricado en serie y más económico, las ventajas del SMD prevalecen sobre las del *THD.*

Dentro del diseño propio de la PCB, se ha decidido separar la parte del control de las válvulas con la de los sensores. Esto es debido a que trabajan en tensiones muy dispares (12V frente a 3.3V), y para aumentar la seguridad frente a cortocircuitos como la conveniencia en el desarrollo, quedaron los circuitos de control de los sensores en un lado, y la alimentación y los circuitos de las válvulas a otro.

Teniendo en cuenta estas consideraciones, el resultado es el que se puede ver en la Ilustración 15. con esta disposición, cuando se utilizase el sistema, se podría colocar tres sensores, los cuales estarían distribuidos verticalmente dentro del terreno, permitiendo ajustarse aún más a distintos tipos de plantas que puedan estar creciendo. Después, la alimentación se conectaría justo al lado del Arduino, dando fácil acceso al puerto del microcontrolador en caso de querer hacer modificaciones en el código. Tanto para el regulador de tensión (U3), como el controlador de las válvulas (U1), no se ha dejado espacio para poner disipadores ya que, en las pruebas realizadas, estos nunca se han puesto suficientemente calientes para que se produzca ningún problema. En caso de que se necesitase, se podrían colocar con otra orientación y no habría problemas.

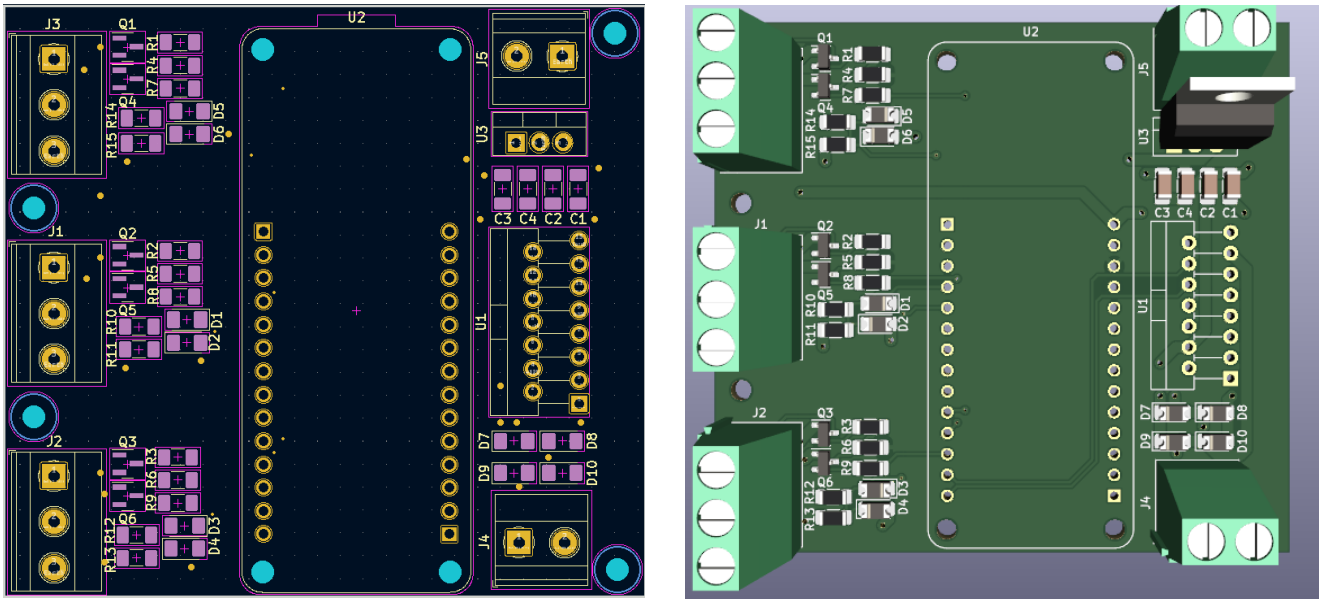


Ilustración 15 - Vista del esquema de la PCB y vista 3D

Por otro lado, en cuanto al trazado de las alimentaciones, se han utilizado dos zonas, una en la capa superior que corresponde con la alimentación de 3.3V, y otra en la capa inferior que corresponde con la GND, como se puede ver en la Ilustración 16. Debido a que la PCB no es compleja, el trazado no ha requerido de utilizar ningún elemento especial, y se ha seguido el uso de la capa inferior para trazado verticales, y la superior para horizontales. Por último, se han creado 4 agujeros para facilitar su montaje en una caja.

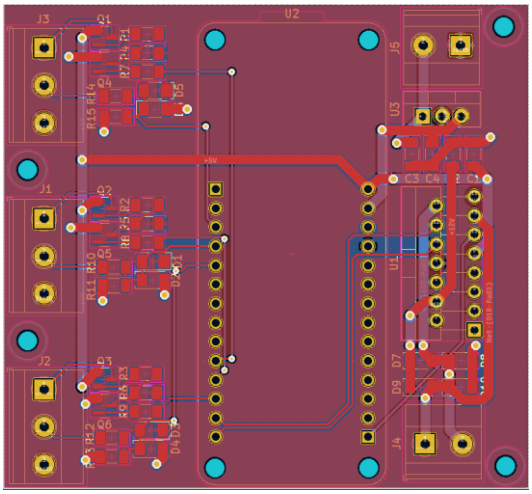


Ilustración 16 - Vista de las pistas de la PCB

Capítulo 6. ANÁLISIS DE RESULTADOS

6.1 *FUNCIONAMIENTO DEL SISTEMA*

Finalmente, el Arduino tendrá conectado varios sensores y una o varias válvulas. En caso de utilizar un sensor u otro habría que modificar la fórmula que adapta la tensión leída al valor de humedad porcentual. La instalación de cada sensor se debe realizar siguiendo la guía del fabricante para conseguir lecturas lo más consistentes posibles. Una vez se conecte el Arduino a una fuente alimentación, este empezará a buscar el Gateway para conectarse a la aplicación de la plataforma y comenzar con la sincronización con la aplicación. Mandará mensajes cada dos minutos hasta recibir algún mensaje de la aplicación. A partir de ese momento, pasará a enviar mensajes cada 7 minutos con el valor leído de la humedad y, si recibe algún paquete con el esquema a seguir del riego, empezará a llevarlo a cabo.

El Gateway se conecta a una red WIFI y será el que traduzca los mensajes LoRa a IP y viceversa. Por otro lado, desde el lado de la aplicación, esta debe estar abierta para recibir los datos y, desde esta, se controlará el riego y se visualizará el estado del terreno.

6.2 *PRUEBAS REALIZADAS*

Se han realizado diversas pruebas a lo largo del desarrollo del proyecto como se requiere cuando se desarrollan elementos tanto en hardware como software.

6.2.1 PRUEBAS DE CIRCUITOS

A la hora de implementar los circuitos que se desarrollaron en el laboratorio, se soldaron en una perfboard. Para probar su funcionamiento y evitar dañar el Arduino, se realizaron medidas con diferentes instrumentos. En primer lugar, se comprobaron las soldaduras en

caso de haber cortocircuitado elementos cercanos y se comprobó la continuidad de líneas como la de 3.3V, 5V y tierras. Esto se realizó con el detector de cortos del polímetro.

Para los circuitos de alimentación de los sensores, se puso el Arduino con el puerto digital correspondiente activo, y se midió la tensión en el terminal de tornillo buscando una tensión de 5V. Por el otro lado, el circuito de protección de los sensores, se conectó el generador de señales del laboratorio y se inyectó una señal desde 0V hasta 5V y se comprobó como la tensión no superaba los 3.3V en la salida.

El circuito de alimentación de la válvula requería poder aportar una corriente puntual de prácticamente 2 Amperios, es por esto que se conectó al terminal una resistencia de 8Ω y 20W. Una vez comprobado su correcto funcionamiento se pasó a realizar pruebas con la válvula real. En primer lugar, se tuvieron diversos problemas que después se vió que era un problema con la válvula. Una vez sustituida, se alimentó el sistema con una fuente a 9V que no fue suficiente ya que su límite era de 1A. Probando después con una fuente de laboratorio, se vio que, durante funcionamiento de la válvula, el pico de corriente llega a los 1,25A, lo cual se debe de tener en cuenta a la hora de hacer la instalación final. Por otro lado, se comprobó la duración necesaria del pulso para abrir y cerrar la válvula. Para esto, se pasaron pulsos desde los 1000ms y se fue disminuyendo. Según las pruebas realizadas, el sistema funciona correctamente hasta los 10ms, debajo de este valor no llega a cerrarse por completo.

6.2.2 PRUEBAS DE SOFTWARE

A lo largo del desarrollo, se tuvieron que hacer pruebas de cada programa y, finalmente, de ambos en conjunto. En primer lugar, para el programa de Arduino, al carecer la plataforma de herramientas de testing y debugging, las pruebas ser realizaron mostrando elementos en la terminal serie. Esto se puede ver a lo largo del código con secciones y funciones como la de *mostrarHorario*, utilizada para comprobar el correcto funcionamiento de la función de ordenación de las acciones. Para comprobar que las comunicaciones funcionaban de manera correcta, se utilizó la plataforma de TheThingsStack donde se simularon mensajes de

Downlink, y se monitorizaron los mensajes que se recibían del Arduino traduciendo los datos que venían en base hexadecimal a cadenas de caracteres.

La aplicación de escritorio siguió un camino de debugging parecido. Sin embargo, al usarse un IDE, este muestra mensajes de error que ayudan en parte a solucionar los problemas. Se tuvo que llevar el desarrollo de cada sección por separado con el fin de entender y aprender a utilizar correctamente cada librería. Primero se comprobó la conectividad con la plataforma de TheThingsStack a través del monitor de esta mirando los mensajes de Downlink que aparecían, y se comprobó la recepción simulando mensajes de *Uplink*. En segundo lugar, se comprobó el funcionamiento de la interfaz gráfica, como lanzaba eventos, y el diseño en general que fuese el correcto. Por último, se probó el funcionamiento de la función de muestra de los datos ya que, al estar embebido en la interfaz, esto trajo diversos problemas. Una vez todos los módulos separados se diseñaron, se pasó a integrar todo junto y, de nuevo se probó el funcionamiento haciéndolo funcionar contra la plataforma.

En último lugar, se hicieron una pequeñas pruebas utilizando la aplicación junto al Arduino comprobando que ambos funcionasen de manera correcta. Se tuvieron que realizar ajustes para lograr una sincronización correcta pero finalmente funcionaba como era de esperar.

6.3 PRUEBAS FINALES

En la fecha en la que se ha entregado esta memoria, no se ha podido probar su funcionamiento en condiciones reales en una explotación agrícola como había sido planeado. Sin embargo, se ha podido probar su funcionamiento a lo largo de un fin de semana en una maceta de un domicilio. Los resultados se pueden ver en la Ilustración 17, donde se pueden observar diferentes medidas a lo largo de los días. Como era de esperar, a lo largo de las horas, la humedad fue disminuyendo, hasta el último día donde se regó y la humedad creció rápidamente. Cabe destacar como entre las 12h del 5 de junio y las 8h del mismo día, existe un hueco de muestras. Esto se debe a que el ordenador que se utilizó para tomar las muestras tuvo un error no relacionado con el proyecto lo que hizo que no recogiese los datos. Además,

las muestras en estas pruebas se recibieron cada hora, ya que en esos momentos la aplicación estaba configurada de esa manera, que ahora ha sido modificada.

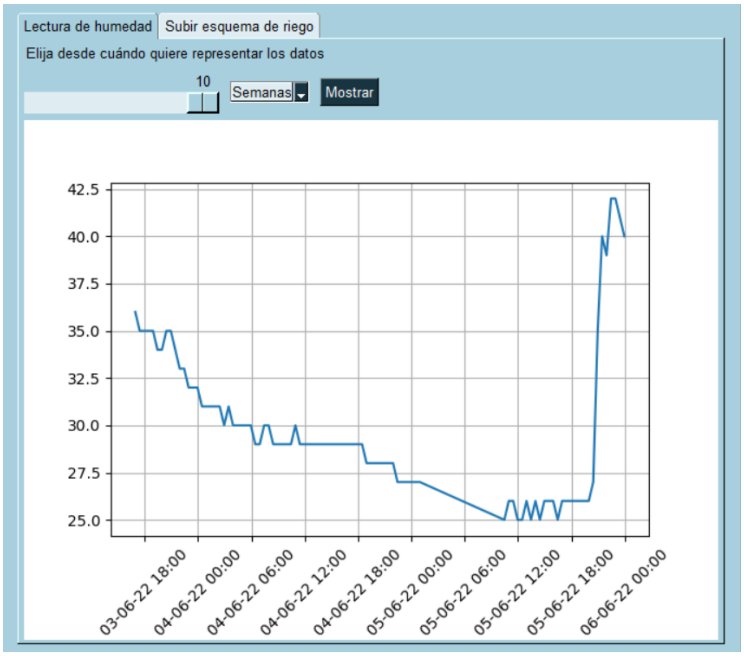


Ilustración 17 - Resultados de lectura

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

En este capítulo analizaremos las conclusiones principales del proyecto y comentaremos los principales puntos de mejora y avances que se pudiesen hacer en futuros trabajos.

7.1 CONCLUSIONES PRINCIPALES

Con el desarrollo de este proyecto se ha conseguido desarrollar un sistema de riego inalámbrico utilizando el protocolo LoRa con el fin de obtener un mayor rango que con tecnologías convencionales, y una mayor eficiencia energética. Esto se ha conseguido creando una aplicación de escritorio como centro de control, la cual interactúa con la placa que se colocaría en el punto que se quiera monitorizar.

A través de las funciones disponibles, el administrador podrá cómodamente realizar un uso más eficiente del agua y su tiempo, conllevando implicaciones de ahorro de energía, agua y dinero. Dentro de los objetivos propuestos en el inicio de este proyecto, se han podido cumplir en su mayor parte. En primer lugar, el desarrollo de la aplicación en Python, aunque esta puede presentar mejoras en diferentes aspectos como se mostrará a continuación, cumple la función propuesta. Por otro lado, tanto el microcontrolador como los distintos circuitos y componentes que se le han conectado se ha conseguido que funcionen correctamente y en sincronía, realizando las funciones esperadas y sin problemas.

Por último, se ha conseguido realizar el diseño de la PCB del proyecto, lo cual permitiría la distribución y construcción de este sistema de una manera más simple y accesible. Este diseño, sin embargo, no ha podido ser probado junto a las pruebas de campo, debido a problemas de tiempo. Tanto revisiones del proyecto como problemas con distintos componentes han causado un retraso en el proyecto que implica que las pruebas finales se realizarán fuera de plazo del plazo de entrega del proyecto.

A pesar de estos inconvenientes, se considera un paso adelante frente a los trabajos anteriores y un punto a partir del cual se puede conseguir el objetivo final de obtener un producto distribuible y utilizable a gran escala.

7.2 TRABAJOS FUTUROS

A pesar de que el sistema en si funciona y cumple con las funciones esperadas, esto no indica que sea perfecto o que no haya puntos de mejora. En este apartado aplicaremos la autocrítica y buscaremos encontrar tanto puntos que pudiesen haberse cambiado por completo, como pequeñas modificaciones.

7.2.1 PERSISTENCIA DE DATOS

Uno de los puntos que pueden destacar más al aplicar este proyecto, es como, al residir la lógica principal en la aplicación, esta debe estar abierta para poder recibir los datos de la humedad. Esto, dependiendo de qué equipo se utilice para monitorizar el terreno, puede ser algo sin importancia o un fallo más o menos grave.

Reflexionando sobre esto, se han encontrado dos mejoras principales. La primera, sería hacer que las placas que haya dentro de nuestro sistema guardasen esta información en su memoria y, en el momento que reciban una petición de recolección, envíen la información al equipo. De esta manera, el quipo no tendría que estar encendido y atento continuamente. Las ventajas de esto sería el paso a una comunicación más asíncrona, pero en caso de querer visualizar los datos en otro lado, sin tener el archivo no se podrá ver.

La otra opción valorada sería utilizar un ordenador como una Raspberri Pi, el cual podría actuar de punto intermedio, y sería este quien tuviese realmente el control del sistema. Se encargaría de recibir los datos en todo momento, y lo guardaría en una base de datos que podría consultarse a través de una API Rest, siendo así accesible por cualquier persona con la aplicación de escritorio. Esta aplicación de escritorio pasaría a ser una aplicación puramente gráfica que realizase peticiones. Este enfoque, permitiría conectar la Raspberri Pi

a un enchufe, y olvidarse del sistema hasta que se quiera consultar. El único problema de este sería el aumento tanto en coste como de consumo del sistema total.

7.2.2 PRUEBA Y REVISIÓN DE LA PCB

Al no haberse completado la fabricación y pruebas de la PCB, se coloca como principal objetivo de un trabajo futuro sobre un proyecto similar. El desarrollo de esta, como se ha indicado anteriormente, permitiría muchos avances. La placa final podría ser más compacta, se podrían utilizar conectores más estándares en vez de terminales de tornillo, y tendríamos la principal ventaja de ser más simple tanto replicar el sistema como su montaje completo. Además, como la placa acabará en el terreno, permite crear una carcasa a prueba de agua en el caso de fuertes lluvias o tormentas.

7.2.3 MEJORAS EN LAS COMUNICACIONES

El paquete que se ha desarrollado en el proyecto para enviar la información de riego cumple sus funciones, pero, de cara a un producto final, siendo el protocolo utilizado uno como el LoRa, sería una gran mejora la implementación de algunos sistemas de comprobación de errores. Esto es debido a que, con el uso de LoRa, el alcance de las comunicaciones puede llegar a ser de varios kilómetros, creando una situación más propensa a errores y menor SNR.

Teniendo en cuenta esto, sería recomendable el uso de protocolos como bits de paridad. Junto a esto, podría crearse algún campo que permitiese seleccionar entre distintas funciones y placas para expandir la aplicación del sistema. Por ejemplo, utilizando sistemas similares, pero a los que conectamos otros elementos, realizar en esas placas distintas acciones como lecturas de otros parámetros.

7.2.4 ADAPTAR EL SISTEMA PARA MÚLTIPLES END-POINTS

Siendo el objetivo del sistema completo de ser utilizado en explotaciones agrícolas reales, la expansión física de estas puede cubrir varias hectáreas, por lo que la colocación de más de un Arduino con válvulas y sensores sería necesaria. Por esto, sería deseable modificar ligeramente la aplicación para poder por ejemplo elegir entre placas tanto para leer como para enviar el esquema en caso de tener distintas plantaciones. Esto junto al punto anterior modificando ligeramente los paquetes para poder enviar los datos a una placa concreta, sería lo necesario para implementarse.

7.2.5 USO DE TECNOLOGÍAS OPEN-SOURCE

Aunque este proyecto será público y accesible tanto este documento como el proyecto completo en GitHub [35], se ha utilizado la plataforma de TheThingsStack para su desarrollo, que como se explicó, puede requerir de pago sen caso de usarse de manera comercial o requerir de soporte. Por esto se propone el uso de Chirpstack como alternativa, que en nuestro caso no pudo desarrollarse por el estado actual de la industria de microchips y las faltas de stock. Este, junto al desarrollo de la PCB, considero que serían los principales objetivos de un trabajo futuro en un sistema como este.

Capítulo 8. ALINEACIÓN DEL PROYECTO CON LOS ODS

En este capítulo se discutirá la relación que tiene el proyecto desarrollado con los Objetivos de Desarrollo Sostenible [36]. Estas son unas propuestas realizadas por las Naciones Unidas, firmadas por los países miembros, donde se buscan eliminar las distintas desigualdades que se pueden ver en el mundo como pobreza, hambre, acceso a recursos, etc. Firmados en 2015, se espera alcanzarlos para 2030. Todos los objetivos se pueden ver en la siguiente ilustración:

Ilustración 18 - Objetivos de Desarrollo Sostenible

Dentro de cada objetivo, se describen distintas submetas, cada organismo deberá intentar aplicar el mayor número de objetivos en desarrollo de su actividad. La creación y reconocimiento de estos objetivos ha hecho que no solo organismos gubernamentales los acepten y diseñen prácticas y programas que ayuden a solucionar estos problemas, sino que también empresas han incluido estos objetivos en sus reportes anuales, y los individuos de la comunidad científica los tienen en cuenta al desarrollar nuevos productos e infraestructura.

8.1 RELACIÓN CON EL PROYECTO REALIZADO

En la Motivación del proyecto se expusieron brevemente varias razones del desarrollo del proyecto, que es crear un sistema accesible, eficiente y útil que se pueda usar de manera extendida. Es por esto que el proyecto puede colaborar en el desarrollo de varios objetivos, sobre todo en el caso de colaborar con proyectos del mismo tema en sincronía.

El objetivo que consideramos que este Trabajo de Fin de Grado aporta más, sería el "**Objetivo 10:** Reducir la desigualdad en y entre los países". Dentro de este objetivo, nos referiríamos a metas como "10.2 De aquí a 2030, potenciar y promover la inclusión social, económica y política de todas las personas, independientemente de su edad, sexo, discapacidad, raza, etnia, origen, religión o situación económica u otra condición". En especial, con el enfoque que se le ha dado al factor económico a lo largo del proyecto, logramos disminuir la desigualdad económica, dándole acceso a más personas y agricultores a sistemas de riego automático.

A pesar de ser el objetivo 10 el principal, también se puede enfocar a otros como puede ser el "12: Garantizar modalidades de consumo y producción sostenibles", y dentro de este la meta de "12.2 De aquí a 2030, lograr la gestión sostenible y el uso eficiente de los recursos naturales". Siendo el agua dulce un recurso escaso en nuestro planeta, procurar su uso eficiente es clave para nuestro desarrollo como en los diversos ecosistemas que dependen de esta. Un sistema como este puede ayudar a utilizar el agua de una manera más eficiente y evitando malgastos. La monitorización de la humedad permite al administrador del terreno tomar una decisión más educada sobre si debe o no en esos momentos regar la tierra, como puede ser después de algún evento de lluvias o tormenta, donde puede parecer que ha llovido poco, pero en realidad no es necesario el uso de más agua.

El uso sostenible de los recursos también se puede enfocar a la energía, como puede ser en el caso de utilizar energías no renovables o fósiles. La simple adaptación del Arduino para poder utilizarse con un panel solar, además de haber reducido el consumo de energía para asegurar su correcto funcionamiento, permite utilizar las placas en el terreno con un uso

neto de energía negativo, no teniendo que utilizar ninguna fuente de energía externa y generando la placa solar energía más que suficiente para la placa.

Capítulo 9. BIBLIOGRAFÍA

[1] M. C. García, «Diseño y desarrollo de un sistema de monitorización y control del riego en una explotación agrícola con cobertura extendida,» 2020.

[2] L. C. P. Rebollo, «Diseño de un sistema de monitorización y control del riego en una explotación agrícola,» 2020.

[3] Arduino, «Arduino MKR WAN 1300,» [En línea]. Available: https://www.mouser.es/catalog/specsheets/Arduino_01172018_ABX00017.pdf. [Último acceso: 5 2 2022].

[4] Rs-Online, «The Things Indoor Gateway TTIG-868,» [En línea]. Available: https://docs.rs-online.com/827b/0900766b816c5f09.pdf. [Último acceso: 11 10 2021].

[5] STMicroelectronics, «L78 Positive Voltage Regulators,» [En línea]. Available: https://www.mouser.es/datasheet/2/389/cd00000444-1795274.pdf. [Último acceso: 2 2 2022].

[6] STMicroelectronics, «L298 Dual Full-Bridge Driver,» . [En línea]. Available: https://www.mouser.es/datasheet/2/389/cd00000240-1795236.pdf. [Último acceso: 5 3 2022].

[7] metergroup, «Teros10 humidity sensor,» [En línea]. Available: http://publications.metergroup.com/Manuals/20788_TEROS10_Manual_Web.pdf. [Último acceso: 11 2 2022].

[8] «ElecFreaks,» 2022. [En línea]. Available: https://www.elecfreaks.com/learn-en/microbitOctopus/sensor/octopus_ef04027.html. [Último acceso: 11 3 2022].

[9] «SeeedStudio,» [En línea]. Available: https://wiki.seeedstudio.com/Grove-Capacitive_Moisture_Sensor-Corrosion-Resistant/#play-with-arduino. [Último acceso: 2 3 2022].

[10] «Rain Bird,» [En línea]. Available: https://www.rainbird.com/es/products/valvulas-de-la-serie-dvdvf. [Último acceso: 3 2 2022].

[11] Microchip, «Low-Power, 32-bit Cortex-M0+ MCU with Advanced Analog,» [En línea]. Available: https://www.mouser.es/datasheet/2/268/SAMD21_Family_DataSheet_DS40001882D-1660038.pdf. [Último acceso: 11 10 2021].

[12] G. C. Topp, "Electromagnetic determination of soil water content: Measurements in coaxial transmission lines," 1980.

[13] "LoRa Alliance," 2021. [Online]. Available: https://lora-alliance.org/. [Accessed 15 Octubre 2021].

[14] Murata, «CMWX1ZZABZ-078 LoRa module,» [En línea]. Available: https://www.murata.com/products/connectivitymodule/lpwa/overview/lineup/type-abz-078. [Último acceso: 11 10 2021].

[15] MQTT, «MQTT,» [En línea]. Available: https://mqtt.org/. [Último acceso: 3 1 2022].

[16] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Python. [Último acceso: 10 6 2022].

[17] «GSMA,» Julio 2021. [En línea]. Available: https://www.gsma.com/iot/deployment-map/. [Último acceso: 4 noviembre 2021].

[18] «GSMA,» Junio 2019. [En línea]. Available: https://www.gsma.com/iot/wp-content/uploads/2019/07/201906-GSMA-NB-IoT-Deployment-Guide-v3.pdf. [Último acceso: 4 noviembre 2021].

[19] «SigFox,» 2021. [En línea]. Available: https://www.sigfox.com/en/what-sigfox/technology#id_technology. [Último acceso: 6 11 2021].

[20] A. Matthews, «Electronic Specifier,» 9 Marzo 2017. [En línea]. Available: https://www.electronicspecifier.com/products/iot/overview-of-long-range-wireless-iot-solutions. [Último acceso: 1 Noviembre 2021].

[21] «SigFox Build,» [En línea]. Available: https://build.sigfox.com/study. [Último acceso: 14 Noviembre 2021].

[22] Chirpstack, «ChirpStack, open-source LoRaWAN® Network Server stack,» [En línea]. Available: https://www.chirpstack.io/. [Último acceso: 3 10 2021].

[23] «Gateway options Chirpstack,» [En línea]. Available: https://www.chirpstack.io/gateway-bridge/gateway/. [Último acceso: 3 10 2021].

[24] P. L.-I. F.-L. C.-E. ,. B.-N. A. F. M. F.-C. Iván Froiz-Míguez, «Design, Implementation, and Empirical Validation of an IoT Smart Irrigation System for Fog Computing Applications Based on LoRa and LoRaWAN Sensor Nodes,» A Coruña, 2020.

[25] M. S. K. y. Y. K. Y. B. R. K. Kodali, «LoRa Based Smart Irrigation System,» 2018.

[26] LoRa Alliance, «LoRaWAN Specification,» 2015. [En línea]. Available: https://lora-alliance.org/wp-content/uploads/2020/11/2015_-_lorawan_specification_1r0_611_1.pdf. [Último acceso: 11 10 2021].

[27] The Things Network, «End Device Activation,» [En línea]. Available: https://www.thethingsnetwork.org/docs/lorawan/end-device-activation/. [Último acceso: 20 10 2021].

[28] Arduino, «MKRWan library,» [En línea]. Available: https://www.arduino.cc/reference/en/libraries/mkrwan/. [Último acceso: 3 11 2021].

[29] Arduino, «RTCZero library,» [En línea]. Available: https://www.arduino.cc/reference/en/libraries/rtczero/. [Último acceso: 3 11 2021].

[30] «PySimpleGUI,» [En línea]. Available: https://pysimplegui.readthedocs.io/en/latest/. [Último acceso: 1 4 2022].

[31] «matplotlib,» [En línea]. Available: https://matplotlib.org/. [Último acceso: 1 4 2022].

[32] R. PierreF, «paho-mqtt 1.6.1,» 21 Octubre 2021. [En línea]. Available: https://pypi.org/project/paho-mqtt/. [Último acceso: 1 4 2022].

[33] TheThingsStack, «Setting Up MQTT Client,» [En línea]. Available: https://www.thethingsindustries.com/docs/integrations/pubsub/mqtt-client/. [Último acceso: 3 4 2022].

[34] TheThingsStack, «MQTT Server,» [En línea]. Available: https://www.thethingsindustries.com/docs/integrations/mqtt/. [Último acceso: 3 4 2022].

[35] A. Lorenzo, «GitHub - Irrigation-System-LoRa,» [En línea]. Available: https://github.com/Lawrence4U/Irrigation-System-LoRa-TFG. [Último acceso: 25 6 2022].

[36] UN, «Objetivos de Desarrollo Sostenible,» [En línea]. Available: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/. [Último acceso: 20 6 2022].

[37] The Things Industries, «Getting Started with TTIG,» [En línea]. Available: https://docs.rs-online.com/9a8e/0900766b816c5f05.pdf. [Último acceso: 10 10 2021].

[38] Arduino, "arduino," [Online]. Available: https://store-usa.arduino.cc/products/arduino-mkr-wan-1300-lora-connectivity.

[39] LoRa Alliance, «LoRaWAN 1.0.3 Specification,» 2018. [En línea]. Available: https://lora-alliance.org/wp-content/uploads/2020/11/lorawan1.0.3.pdf. [Último acceso: 10 9 2021].

[40] UN, «Objetivo 11: Lograr que las ciudades sean más inclusivas, seguras, resilientes y sostenibles,» [En línea]. Available: https://www.un.org/sustainabledevelopment/es/cities/. [Último acceso: 20 6 2022].

ANEXO I: GUÍA DE INSTALACIÓN

En este anexo se explicará como instalar y crear el entorno de trabajo que se ha usado en este proyecto con el fin de facilitar la replicación de este y aclarar posibles dudas. Todos los ejemplos se harán con instalación en Windows.

I.1 IDE ARDUINO

Para poder programar en el Arduino se ha utilizado el IDE propio que se puede encontrar tanto en la Microsoft Store o, como se ha hecho en nuestro caso, de la página oficial de Arduino (https://www.arduino.cc/en/software). Una vez descargado el instalador, se irán dando a siguiente hasta que se instale el programa. A parte de la ruta de instalación, es recomendable prestar atención a la pantalla de opciones de instalación en caso de no querer crear accesos directos. El resto de las opciones si no se ha descargado el programa antes, son muy recomendables y, a veces, necesarias para que funcione correctamente.

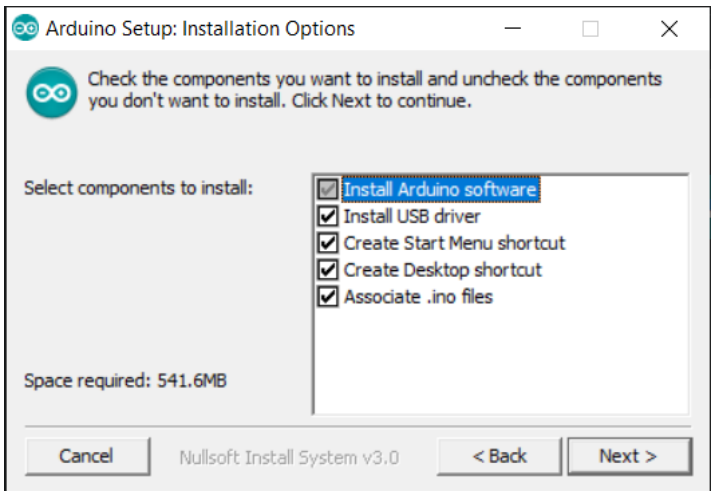


Ilustración 19 - Opciones de instalación

Una vez instalado el programa, tendremos que instalar nuestra placa y las librerías para poder programar la tarjeta MKRWAN 1300. Para la tarjeta en sí, tendremos que instalar el gestor de tarjetas adecuado. Para esto, habrá que ir a Herramientas → Placa → Gestor de tarjetas.

Allí buscaremos Arduino AVR Boards e instalaremos el gestor. Para las librerías, iremos de nuevo a Herramientas → Administar Bibliotecas, y buscaremos tanto MKRWAN como RTCZero, y las instalaremos. Con esto quedaría dispuesto el programa para empezar a programar. A la hora de querer subir el programa, conectaremos el Arduino con un cable micro-usb, seleccionamos en placa el modelo de nuestro Arduino, y, en el puerto, el que salga que está nuestra placa.

I.2 VSCODE

VSCode es un programa desarrollado por Microsoft que permite desarrollar software en diversos lenguajes de programación. Para este proyecto solo será necesario utilizarlo para desarrollar en Python. Para su instalación, se descarga el instalador de la página oficial (https://code.visualstudio.com/). Mientras se ejecuta el instalador asegurarse de que esté seleccionada la opción "Agregar a PATH".

Tras instalarlo y abrir la aplicación se verá una interfaz como la de la Ilustración 20. Dentro de esta, seleccionaremos el icono con los cubos que es el correspondiente a las extensiones. Aquí, instalaremos la extensión oficial de Microsoft para Python. Con esto ya podremos crear un archivo Python y empezar a programar.

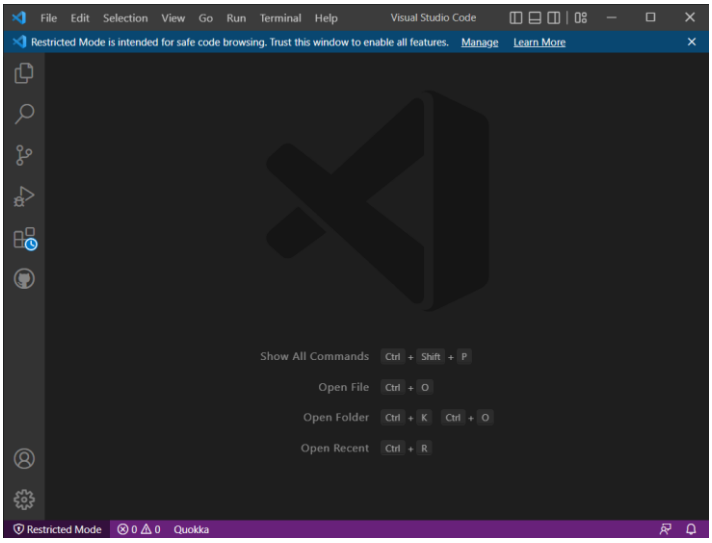


Ilustración 20 - VSCode

Para poder ejecutar el programa, deberemos tener el lenguaje instalado en nuestro ordenador y la manera más simple es entrar en la Microsoft Store y descargarlo buscando Python. Una vez instalado, debemos entrar en VSCode, apretar Ctrl + P para abrir la barra de comandos, y elegir Python interpreter. Aquí, seleccionaremos la versión de Python que tenemos instalada en el ordenador y con esto ya funcionará.

En cuanto a las librerías que tendremos que instalar como la de PySimpleGUI, simplemente habrá que abrir una línea de comandos y escribir pip install *{libreria}*. Esto servirá para instalar todas las librerías que necesitamos para este proyecto.

I.3 THETHINGSSTACK

Para crear nuestra aplicación y que puedan comunicarse los Arduino y el PC, utilizamos la plataforma de TheThingsStack. Para crear la aplicación primero debemos obtener un *tentant-id*, el cual se obtiene al crearnos una cuenta. Para esto, debemos ir a la página de TheThingsIndustries (https://www.thethingsindustries.com/) y darle al botón de "Get Started". Aquí nos saldrán las opciones para elegir en función de nuestro proyecto. En nuestro caso, como solo utilizamos un Gateway y menos de 10 nodos, seleccionamos "The Things Stack Discovery" en el paso 1 y "DIY" en el paso 2, lo que nos permitirá utilizar la plataforma de manera gratuita.

En la siguiente pantalla se nos pedirá rellenar diferente información, como elegir *el tenant-id* y la zona, que elegiremos Europa. A partir de aquí pedirá datos financieros en el caso de que posteriormente se elija mejorar la tarifa a una de pago. Una vez obtenido el tenant-id, iremos al URL "https://{tenant-id}.{zona}.cloud.thethings.industries". El tenant-id deberá ser el propio y la zona dependerá de la elegida, en mi caso eu2.

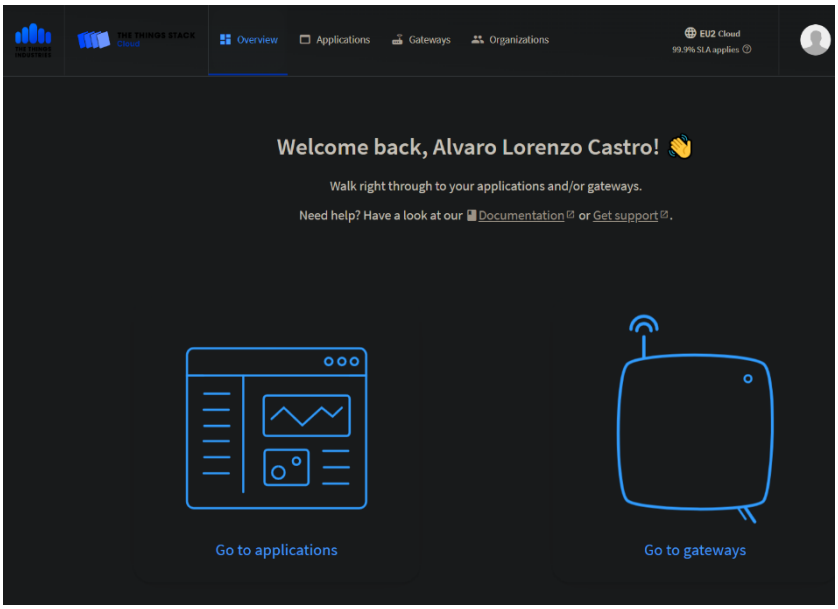


Ilustración 21 - Pantalla inicial de TheThingsStack

Una vez logueado, se nos presentará una pantalla como la de la Ilustración 21. Aquí pasaremos a crear nuestra aplicación, que solo requiere darle un nombre único. Después registraremos nuestro Gateway, primero le pondremos un nombre único, y después tendremos que introducir el EUI del Gateway. Para esto, depende del que se utilice pero para el de TheThingsIndustries está escrito en el aparato en la parte de abajo. También se deberá seleccionar la frecuencia que se va a utilizar que será la de la zona propia que en nuestro caso es la Europe SF9. El resto de los parámetros se han dejado por defecto. Si todo se ha hecho correctamente, y el Gateway está conectado correctamente, debería aparecer en la consola un mensaje de conexión del Gateway. Esta consola se encuentra dentro del menú del Gateway en la sección de live-data.

Para añadir nuestros nodos a la aplicación, entraremos dentro de esta, haremos click en la barra izquierda en End Devices, y crearemos uno según cual es el nuestro. En este caso, seleccionamos Arduino, MKRWAN 1310, firmware 1.2.3, y zona Europa _863_870.

Después en el registro pondremos la misma frecuencia que al añadir el Gateway, generaremos un AppEUI y un AppKey (son importantes y no se deben compartir con otros). Para el DevEUI debemos obtenerlo del Arduino en si. Para obtenerlo, debemos cargar el programa de ejemplo dentro de la librería MKRWAN de LoraSendandReceive. Al incio de este programa, nos indicará el DevEUI de nuestro Arduino. Por último, le daremos un nombre único y proceso completado.

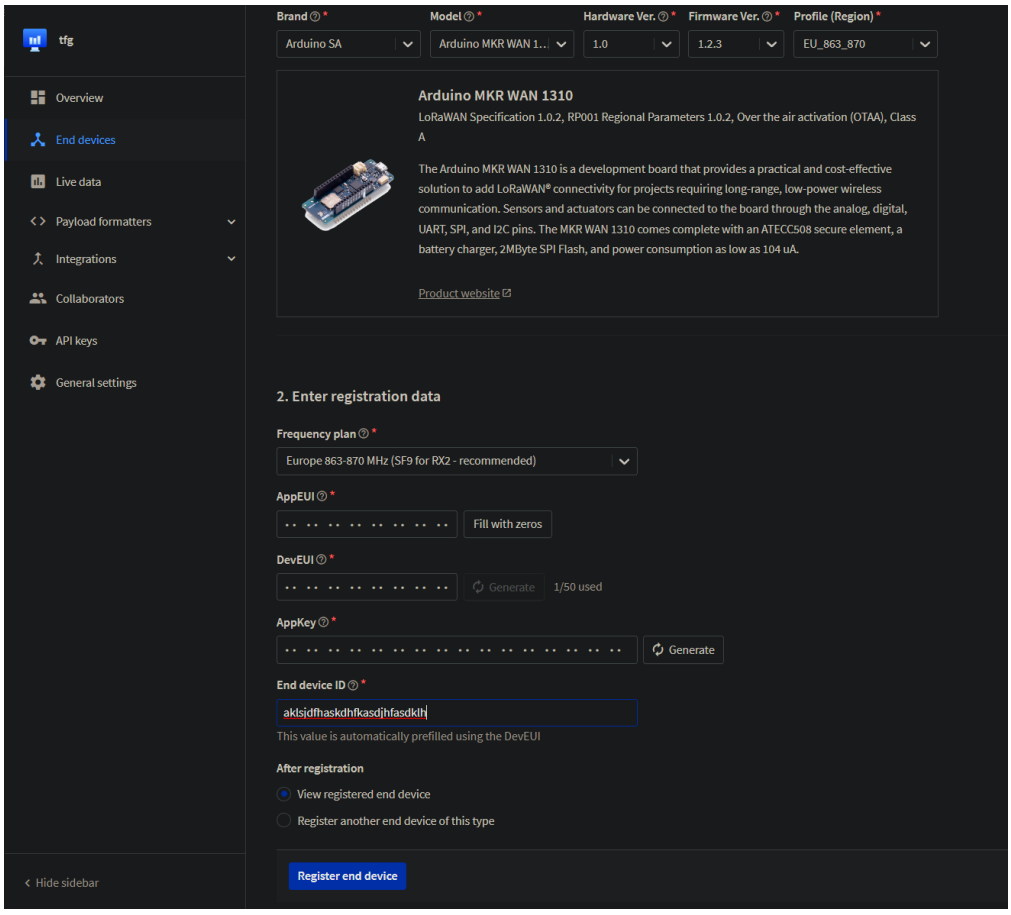


Ilustración 22 - Registro de dispositivo en la plataforma

Una vez registrado todo, podremos probar con el mismo programa del Arduino, como se conecta y hablan entre ellos a través de la consola del Arduino. En una pestaña siguiente, se podrán simular mensajes de Uplink y Downlink para probar la funcionalidad de nuestra aplicación.

Por último para este proyecto, tendremos que habilitar la integración de MQTT. Para esto iremos a la sección de Integrations, MQTT, y le daremos a generar nueva API Key. Es importante que al generarla guardemos el valor y no lo perdamos ya que será la contraseña para conectarnos desde la aplicación. Con esto, la plataforma quedaría completamente configurada.

I.4 INSTALACIÓN DE GATEWAY TTIG

Para instalar el Gateway, se deberán seguir unos simples pasos. Primero, resetearlo manteniendo el botón reset (al lado del puerto USB-C) durante 5 segundos hasta que el LED parpadee rápidamente Verde y Rojo. Una vez hecho esto, mantendremos el botón de SETUP (arriba) durante 10 segundos hasta que parpadee en rojo. Ahora debermos ir a nuestro PC, y buscar una red cuyo nombre sea MINIHUB-xxxxxx donde xxxxxx son los 6 últimos dígitos del ID del Gateway. La contraseña de este red será la que pone en el aparato debajo de WiFi PW. Una vez conectado, habrá que meterse en el navegador a la dirección 192.168.4.1. Aquí se mostrarán las redes que ve el Gateway actualmente. Cabe destacar que el Gateway sólo funciona con redes de 2.4Ghz. Dentro del menú, elegiremos nuestra red, y en el pop-up que aparece, debemos poner la contraseña. Una vez hecho esto, hacer click en “save & reboot”. Esto reiniciará el Gateway e intentará conectarse a la red. Si la configuración es correcta, primero parpadeará verde mientras se conecta a la red y después verde/rojo mientras se conecta a CUPS. Tras esto, el LED brillará verde sin parpadear [37].

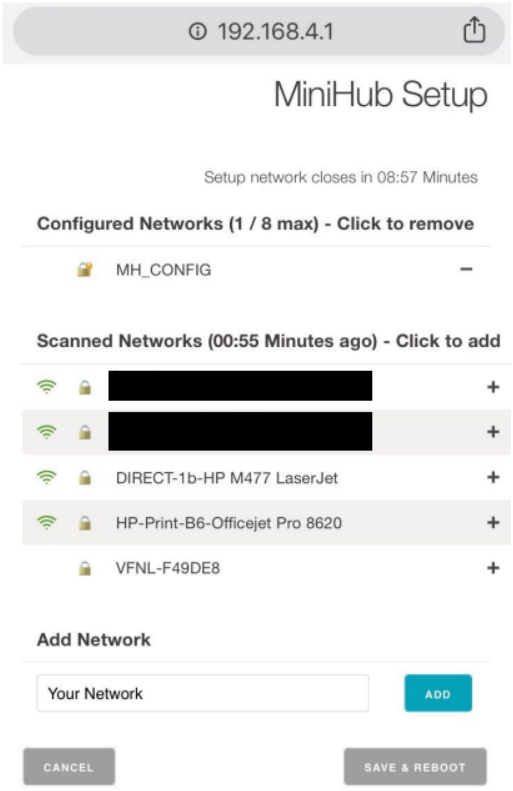


Ilustración 23 - Pantalla de conexión del Gateway

I.5 CONEXIÓN DE PERIFÉRICOS A LA PLACA

Si se decide seguir el esquema propuesto de conexiones, para conectar los sensores, tomando como referencia los pines del esquema en kiCad (Ilustración 24), al pin 1 conectaremos VCC; al pin 2, la salida del sensor; y al pin 3, la tierra.

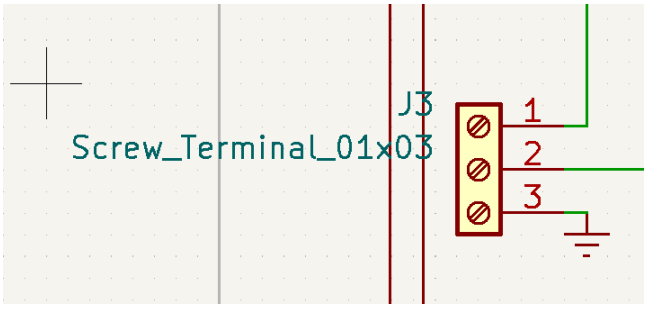


Ilustración 24 - Terminal de tornillo del sensor

Para la alimentación, siguiendo de nuevo el esquema de kiCad (Ilustración 25), se colocará el positivo de 12V al puerto 2, y la tierra al 1.

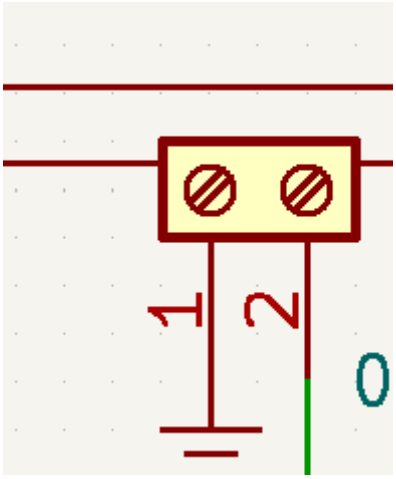


Ilustración 25 - Terminal de alimentación

ANEXO II : CÓDIGOS DESARROLLADOS

I.1 CÓDIGO FUENTE DEL MICROCONTROLADOR

LoraSendAndReceiveUpdated.ino:

```
#include <MKRWAN.h>
#include <RTCZero.h>
#include "arduino_secrets.h"

RTCZero rtc;
LoRaModem modem(Serial1);

struct Time{
  byte hour;
  byte minute;
  byte second;
};

struct PeripherialAction{
  Time hora;
  byte gap;
  byte id_periferico;
  bool on_off;
  int nextDay; //este campo servirá para ordenar las acciones correctamente
habiendo pasado un dia
};

struct PeripherialAction acciones[10];
int indice_acc=0;
int accion_sel=0;

const int sensor1 = 1;
const int sensor2 = 2;
const int sensor3 = 3;
const int sensores[3] = {sensor1, sensor2, sensor3};

const int lectura1 = A1;
const int lectura2 = A2;
const int lectura3 = A3;

bool con_horario = false;
long timer_start = 0;
const long OFFSET = 420000;//7minutos en milisegundos

const int valvula0[2] = {4,5}; //pines de cada valvula
const int valvula1[2] = {7,7}; //7 indica no usado en el HW
```

```
const int valvula2[2] = {7,7};
const int valvula3[2] = {7,7};
int valvulas[8] =
{valvula0[0],valvula0[1],valvula1[0],valvula1[1],valvula2[0],valvula2[1],valvula3
[0],valvula3[1]};
int enableValvula[4] = {6,7,7,7};

String appEui = SECRET_APP_EUI;
String appKey = SECRET_APP_KEY;

void setup() {
  Serial.begin(115200);
  while (!Serial);
  if (!modem.begin(EU868)) {
    Serial.println("Failed to start module");
    while (1) {}
  };
  Serial.print("Your module version is: ");
  Serial.println(modem.version());
  Serial.print("Your device EUI is: ");
  Serial.println(modem.deviceEUI());

  int connected = modem.joinOTAA(appEui, appKey);
  if (!connected) {
    Serial.println("Something went wrong; are you indoor? Move near a window and
retry");
    while (1) {}
  }
  analogReadResolution(12);//el Teros 10 envia datos de 12bits
  modem.minPollInterval(60);

  pinMode(valvula0[0], OUTPUT);
  pinMode(valvula1[0], OUTPUT);
  pinMode(valvula2[0], OUTPUT);
  pinMode(valvula3[0], OUTPUT);
  pinMode(valvula0[1], OUTPUT);
  pinMode(valvula1[1], OUTPUT);
  pinMode(valvula2[1], OUTPUT);
  pinMode(valvula3[1], OUTPUT);
  pinMode(enableValvula[0], OUTPUT);
  pinMode(enableValvula[1], OUTPUT);
  pinMode(enableValvula[2], OUTPUT);
  pinMode(enableValvula[2], OUTPUT);

  pinMode(sensor1, OUTPUT);
  pinMode(sensor2, OUTPUT);
  pinMode(sensor3, OUTPUT);
  pinMode(lectura1, INPUT);
  pinMode(lectura2, INPUT);
  pinMode(lectura3, INPUT);

  digitalWrite(sensor1, LOW);
  digitalWrite(sensor2, LOW);
```

```
  digitalWrite(sensor3, LOW);
  digitalWrite(enableValvula[0], LOW);
  digitalWrite(enableValvula[1], LOW);
  digitalWrite(enableValvula[2], LOW);
  digitalWrite(enableValvula[3], LOW);

  rtc.begin();
  Serial.println("Setup complete");
  /*char mensaje[7] = "AABBCC";
  sendMessage(mensaje, 6);*/
}

void loop() {
  if(!con_horario){
    //mandar mensaje
    Serial.println("Checking for messages...");
    char mensaje[5] = "Init";
    sendMessage(mensaje, 4);
    delay(1000);//damos tiempo para recibir el mensaje de vuelta
    if(!modem.available()){
      Serial.println("Sin mensajes");
      delay(20000);//2 minutos para reenviar paquete
      return;
    }else{
      Serial.println("received, now checking...");
      con_horario = 1;
      char rcv[64];
      //rcv[54] = 63;//pongo un valor aleatorio para luego comprobar que el
paquete tiene la longitud necesaria
      int i = 0; //i tendra la longitud de la cadena recibida
      while (modem.available()) {
        rcv[i++] = (char)modem.read();
      }
      /*if(rcv[54] == 63){
      char msg[6] = "ERROR";//error de recepcion
      sendMessage(msg, 5);
      con_horario = 0;
      return;
      }*/
      Serial.print("Received: ");
      for (int j = 0; j < i; j++) {
        Serial.print(rcv[j] >> 4, HEX);
        Serial.print(rcv[j] & 0xF, HEX);
        Serial.print(" ");
      }
      Serial.println();
      indice_acc = 0;
      manageSchedule(rcv);
      timer_start = millis();
      //poner alarmas etc
      if(indice_acc!=0){
        Serial.println("Poniendo alarmas...");
        //ponemos la primera alarma en funcionamiento
```

```
rtc.setAlarmTime(acciones[0].hora.hour,acciones[0].hora.minute,acciones[0].hora.s
econd);
        rtc.enableAlarm(rtc.MATCH_HHMMSS);
        rtc.attachInterrupt(alarma);
        Serial.println("Setup completo");
      }
    }
  }
  if(!modem.available()){
    if(millis() - timer_start >= OFFSET){
      timer_start = millis();
      //en vez de enviar basura, mandar el valor leido por el sensor actual
      double humedad, valorSensor;
      digitalWrite(sensor2, HIGH); //alimentamos el sensor
      delay(20);
      valorSensor = analogRead(lectura2);
      digitalWrite(sensor2,LOW); //desconectamos para ahorrar energía
      double tension = valorSensor*3.33/4096;
      tension = tension*1000/0.638;
      humedad = 5.439*pow(10,-10)*pow(tension,3) -2.731*pow(10,-6)*pow(tension,2)
+4.868*pow(10, -3)*tension -2.683; //ecuación para Teros10
      Serial.print("Humedad(%): ");
      Serial.println(humedad*100, 2);
      int valorMensaje = round(humedad*100);
      char mensaje[4];
      if(valorMensaje<0){
        valorMensaje = 0;
      }else if(valorMensaje>100){
        valorMensaje = 100;
      }
      Serial.print("RTC: ");
      Serial.print(rtc.getHours());
      Serial.print(":");
      Serial.print(rtc.getMinutes());
      Serial.print(":");
      Serial.println(rtc.getSeconds());
      itoa(valorMensaje, mensaje, 10);
      sendMessage(mensaje, 3);

    }
  }else{//recibo paquete despues del envio
    char rcv[64];
    //rcv[54] = 63;//pongo un valor aleatorio para luego comprobar que el paquete
tiene la longitud necesaria
    int i = 0; //i tendra la longitud de la cadena recibida
    while (modem.available()) {
      rcv[i++] = (char)modem.read();
    }
    /*if(rcv[54] == 63){
      char msg[6] = "ERROR";//error de recepcion
      sendMessage(msg, 5);
      con_horario = 0;
```

```
    }*/
    Serial.print("Received: ");
    for (int j = 0; j < i; j++) {
      Serial.print(rcv[j] >> 4, HEX);
      Serial.print(rcv[j] & 0xF, HEX);
      Serial.print(" ");
    }
    Serial.println();
    indice_acc = 0;
    manageSchedule(rcv);
    Serial.print("Numero de acciones recibidas: ");
    Serial.println(indice_acc);
    //poner alarmas etc
    if(indice_acc!=0){
      Serial.println("Poniendo alarmas...");
      //ponemos la primera alarma en funcionamiento

rtc.setAlarmTime(acciones[0].hora.hour,acciones[0].hora.minute,acciones[0].hora.s
econd);
      rtc.enableAlarm(rtc.MATCH_HHMMSS);
      rtc.attachInterrupt(alarma);
      Serial.println("Setup completo");
    }
  }


}

void sendMessage(char *message, int longitud){
  char *pt_msg = message;
  Serial.print("Sending:");
  for (int i = 0; i < longitud; i++) {
    Serial.print(*pt_msg >> 4, HEX);
    Serial.print(*pt_msg & 0xF, HEX);
    pt_msg++;
    Serial.print(" ");
  }
  Serial.println();
  int succ = 0;
  modem.beginPacket();
  while(!succ){
    modem.print(message);
    succ = modem.endPacket(true);
    if (succ > 0) {
      Serial.println("Message sent correctly!");
    } else {
      Serial.println("Error sending message :(");
    }
  }
  delay(1000);
}

void manageSchedule(char *paquete){
```

```
  char *esquema = paquete;
  struct PeripherialAction accion;
  char accionesInstantaneas = *esquema;
  esquema++;
  int *ptrVal = valvulas;
  int *ptrEn = enableValvula;
  //Serial.print("botones: ");
  //Serial.println(accionesInstantaneas, BIN);

  for(int i=0;i<8;i++){
    if(i%2==0 && i!=0){
      ptrVal+=2;
      ptrEn++;
    }
    if((accionesInstantaneas>>(7-i))&1){
      actuarElectrovalvula(ptrVal[0],ptrVal[1], ptrEn[0],!(i%2));//si i par es
abrir sino cerrar
    }
  }

  for(int i=0;i<4;i++){
    /*Serial.print("valor del puntero[0]: ");
    Serial.print(*esquema >> 4, HEX);
    Serial.println(*esquema & 0xF, HEX);*/
    if(*esquema==0){
      continue;
    }else{
      //hora,duracion,id_periferico,pinEnable
      accion.hora.hour = esquema[0];
      accion.hora.minute = 0;
      accion.hora.second = 0;
      accion.nextDay = 0;
      while(inAcciones(accion)){
        accion.hora.second += 10;
      }
      accion.gap = esquema[0];
      accion.id_periferico = i;
      accion.on_off = true;
      acciones[indice_acc] = accion;
      indice_acc++;

      //con la duracion creamos otra accion para cerrar la valvula
      accion.hora.hour = esquema[0];
      int horasRiego = esquema[1]/60;
      if(accion.hora.hour + horasRiego<24){
        accion.hora.hour += horasRiego;
        accion.nextDay = 0;
      }else{
        accion.hora.hour += horasRiego - 24;
        accion.nextDay = 1;
      }
      accion.hora.minute = esquema[1]%60;
```

```
      accion.hora.second = 0;
      while(inAcciones(accion)){
        accion.hora.second += 10;
      }
      accion.gap = esquema[0];
      accion.id_periferico = i;
      accion.on_off = false;
      acciones[indice_acc] = accion;
      indice_acc++;
    }

    esquema+=2;
  }

  Serial.println("Ordenando");
  sortActions();
  Serial.println("Terminado de ordenar");

  for(int j=0; j<indice_acc; j++){
    mostrarHorario(acciones[j]);
  }

  rtc.setTime(0,0,0);
  //PRUEBAS
  //rtc.setTime(0,59,0);

  return;
}

bool inAcciones(PeripherialAction accion){

    for(int i=0;i<indice_acc;i++){
      if(accion.hora.hour == acciones[i].hora.hour && accion.hora.minute ==
acciones[i].hora.minute && accion.hora.second == acciones[i].hora.second){
        return true;
      }
    }
    return false;
}

void mostrarHorario(const PeripherialAction accion){

  Serial.print("Valvula: ");
  Serial.print(accion.id_periferico, DEC);
  Serial.print("\t");
  Serial.print("Hora: ");
  Serial.print(accion.hora.hour, DEC);
  Serial.print(":");
  Serial.print(accion.hora.minute, DEC);
  Serial.print(":");
  Serial.print(accion.hora.second, DEC);
  Serial.print(",nextDay: ");
  Serial.print(accion.nextDay);
```

```
  Serial.print(",gap: ");
  Serial.print(accion.gap, DEC);
  Serial.print(",on_off: ");
  Serial.println(accion.on_off);
  return;
}

void alarma(){
    Serial.println("Ejecutando alarma: ");
    mostrarHorario(acciones[0]);
    actuarElectrovalvula(valvulas[acciones[0].id_periferico*2],
valvulas[(acciones[0].id_periferico*2)+1],
enableValvula[acciones[0].id_periferico], acciones[0].on_off);
    PeripherialAction nextProgram;
    nextProgram = acciones[0];
    if(nextProgram.gap+rtc.getHours()<24){
      nextProgram.hora.hour = nextProgram.gap +rtc.getHours();
    }else{
      nextProgram.hora.hour = nextProgram.gap +rtc.getHours()-24;
      nextProgram.nextDay += 1;
    }
    while(inAcciones(nextProgram)){
        nextProgram.hora.second += 10;
    }
    acciones[0] = nextProgram;

    Serial.println("Ordenando");
    sortActions();
    Serial.println("Terminado de ordenar");

    for(int j=0; j<indice_acc; j++){
      mostrarHorario(acciones[j]);
    }

rtc.setAlarmTime(acciones[0].hora.hour,acciones[0].hora.minute,acciones[0].hora.s
econd);
    //PRUEBAS
    /*if(acciones[0].hora.minute !=0){
      rtc.setTime(acciones[0].hora.hour,acciones[0].hora.minute-
1,acciones[0].hora.second);
    }else if(acciones[0].hora.hour != 0){
      rtc.setTime(acciones[0].hora.hour-1,59,acciones[0].hora.second);
    }*/

    return;
}

void sortActions(){

  for (int i=0; i<indice_acc-1;i++){
    for(int j=0; j<indice_acc-i-1; j++){
      if(!compare(acciones[j], acciones[j+1])){ //si acciones[j] es antes
devolvera un 1 => no se cambia
```

```
        swap(&acciones[j], &acciones[j+1]);
      }
    }
  }


  return;
}

bool compare(const PeripherialAction accion1, const PeripherialAction accion2){
  if(accion1.nextDay < accion2.nextDay){
    return true;
  }else if(accion1.nextDay == accion2.nextDay && accion1.hora.hour <
accion2.hora.hour){
     return true;
  }else if(accion1.nextDay == accion2.nextDay && accion1.hora.hour ==
accion2.hora.hour && accion1.hora.minute < accion2.hora.minute){
    return true;
  }else if(accion1.nextDay == accion2.nextDay && accion1.hora.hour ==
accion2.hora.hour && accion1.hora.minute == accion2.hora.minute &&
accion1.hora.second < accion2.hora.second){
    return true;
  }
  return false;

}

void swap(PeripherialAction *xp, PeripherialAction *yp){
    struct PeripherialAction temp = *xp;
    *xp = *yp;
    *yp = temp;
    return;
}

void actuarElectrovalvula(int inp1, int inp2, int puertoEnable, bool on_off){
  //Logica para encender/apagar valvulas
  /*Serial.print("Argumentos para actuacion válvula:");
  Serial.print(inp1);
  Serial.print(",");
  Serial.print(inp2);
  Serial.print(",");
  Serial.print(puertoEnable);
  Serial.print(",");
  Serial.println(on_off);*/

  pinMode(puertoEnable, HIGH);
  if(on_off){
    Serial.println("Abriendo valvula en el puerto");
    digitalWrite(inp1, HIGH);
    digitalWrite(inp2, LOW);
    delay(10);//tiempo que tarde en abrirse/cerrarse
    digitalWrite(inp2, HIGH);
```

```
  }else{
    Serial.println("Cerrando valvula en el puerto");
    digitalWrite(inp1, LOW);
    digitalWrite(inp2, HIGH);
    delay(10);//timepo que tarde en abrirse/cerrarse
    digitalWrite(inp2, LOW);
  }
  pinMode(puertoEnable, LOW);

  return;
  }
```

I.2 APLICACIÓN DE ESCRITORIO (GUI.PY)

```
import PySimpleGUI as sg
import matplotlib
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure
import re
import paho.mqtt.client as mqtt
import json as json
from base64 import b64encode, b64decode
import threading
from threading import Timer
import codecs
from dateutil import parser
from dateutil.relativedelta import relativedelta
from datetime import datetime, timedelta
import pytz

matplotlib.use('TkAgg')
sg.theme('LightBlue3')

opciones = ['Dias', 'Semanas', 'Meses']
lay1 = [
    [sg.T('Elija desde cuándo quiere representar los datos')],
    [sg.Slider((1, 10), orientation='h', key='slider'),
     sg.Combo(values=opciones, default_value=opciones[0], enable_events=True,
readonly=True, k='-COMBO-'),
     sg.B('Mostrar', key='graph')],
    [sg.Canvas(key='Canvas')]
]

lay3 = [
    [sg.T('Valvula 0:')],
    [sg.T('Cada: '), sg.Input(expand_x=True, size=(2, 1), key='gap0'),
sg.T('Horas, regar: '),
     sg.Input(expand_x=True, size=(2, 1), key='duracion0'), sg.T('minutos')],
```

```
    [sg.T('Valvula 1:')],
    [sg.T('Cada: '), sg.Input(expand_x=True, size=(2, 1), key='gap1'),
sg.T('Horas, regar: '),
     sg.Input(expand_x=True, size=(2, 1), key='duracion1'), sg.T('minutos')],
    [sg.T('Valvula 2:')],
    [sg.T('Cada: '), sg.Input(expand_x=True, size=(2, 1), key='gap2'),
sg.T('Horas, regar: '),
     sg.Input(expand_x=True, size=(2, 1), key='duracion2'), sg.T('minutos')],
    [sg.T('Valvula 3:')],
    [sg.T('Cada: '), sg.Input(expand_x=True, size=(2, 1), key='gap3'),
sg.T('Horas, regar: '),
     sg.Input(expand_x=True, size=(2, 1), key='duracion3'), sg.T('minutos')],
    [sg.T('Todas las valvulas:')],
    [sg.B('Subir', key='SubirA')],
    [sg.T('Acciones inmediatas:', expand_x=True)],
    [sg.B('Abrir valvula 0', key='A0'), sg.B('Cerrar valvula 0', key='C0')],
    [sg.B('Abrir valvula 1', key='A1'), sg.B('Cerrar valvula 0', key='C1')],
    [sg.B('Abrir valvula 2', key='A2'), sg.B('Cerrar valvula 0', key='C2')],
    [sg.B('Abrir valvula 3', key='A3'), sg.B('Cerrar valvula 0', key='C3')],
    [sg.T(visible=False, key='warning')]
]
lay_grp = [[sg.Tab('Lectura de humedad', layout=lay1), sg.Tab('Subir esquema de
riego', layout=lay3)]]
layout = [
    [sg.TabGroup(layout=lay_grp,
                 tab_location='topleft',
                 key='tabGroup',
                 expand_x=True,
                 expand_y=True,
                 visible=True)],
]


def find2nd(string, char):
    """Funcion que obtiene la posicion del segundo caracter en un string

    Args:
        string (string): cadena
        char (char): caracter a buscar

    Returns:
        int: posicion
    """
    start = string.find(char)
    index = string[start + 1:].find(char)
    return index + start + 1

def getDatos():
    f = open("datos.json")
    return json.load(f)

def getConfig():
    f = open("config.json")
```

```
    return json.load(f)

def guardarConfig(values):

    f = open("config.json", "r")
    conf = json.load(f)

    for num in ('0', '1', '2', '3'):
        for i,tipo in enumerate(('gap', 'duracion')):
            if values[tipo + num]=="":
                conf["".join(['V',num])][i] = '0'
            elif values[tipo + num]!="":
                conf["".join(['V',num])][i] = values[tipo + num]

    print(conf)
    json_completo = json.dumps(conf)

    with open("config.json", "w") as outfile:
        outfile.write(json_completo)

    return

def cargarConfig(window):
    f = open("config.json", "r")
    conf = json.load(f)
    for num in ('0', '1', '2', '3'):
        for i,tipo in enumerate(('gap', 'duracion')):
            print(window[tipo + num])
            window[tipo + num].update(conf["V"+num][i])


    return

def on_connect(client, userdata, flags, rc):
    print("Connected with result code " + str(rc))
    # Subscribing in on_connect() means that if we lose the connection and
    # reconnect then subscriptions will be renewed.
    client.subscribe("#")


# The callback for when a PUBLISH message is received from the server.
def on_message(client, userdata, msg):
    # print(msg.topic)
    print(msg.payload.decode('utf-8'))
    recibido = json.loads(msg.payload.decode('utf-8'))
    # mensajes desde la placa
    if msg.topic == 'v3/tfg-icai@tfg-icai/devices/arduino-tfg/up':

print(codecs.decode(str(b64decode(recibido['uplink_message']['frm_payload']).hex(
)), 'hex').decode(
            'utf-8'))
```

```
        if
codecs.decode(str(b64decode(recibido['uplink_message']['frm_payload']).hex()),
'hex').decode(
                'utf-8') == 'ERROR':
            window['warning'].update(visible=True, value='Error en el envío,
pruebe a enviarlo de nuevo')
            return

        if
codecs.decode(str(b64decode(recibido['uplink_message']['frm_payload']).hex()),
'hex').decode(
                'utf-8') == 'Init':
            return

        try:
            humedad = int(

codecs.decode(str(b64decode(recibido['uplink_message']['frm_payload']).hex()),
'hex').decode(
                    'utf-8'))
            t = Timer(390, timerEnd, args=[client])
            t.start()

            print('% de humedad:', humedad)
            madridtz = pytz.timezone("Europe/Madrid")
            try:
                tiempo =
parser.isoparse(recibido['uplink_message']['received_at']).astimezone(madridtz)
            except:
                tiempo =
parser.isoparse(recibido['received_at']).astimezone(madridtz)#al hacer pruebas
cuando el paqeute viene de la plataforma es distinto
            print(tiempo)
            fecha = str(tiempo.date())
            hora = str(tiempo.time())
            key = str(fecha + " " + hora[:find2nd(hora, ':')])
            datos_prev = getDatos()

            #print(datos_prev)
            if key not in datos_prev.keys():
                datos_prev[key] = humedad
            json_completo = json.dumps(datos_prev)

            with open("datos.json", "w") as outfile:
                outfile.write(json_completo)
        except:
            print("valor del arduino no correcto")
        # los datos se guardaran en un dict de manera {(Fecha, Hora) : Valor}



    # mensajes hacia la placa
    elif msg.topic == 'v3/tfg-icai@tfg-icai/devices/arduino-tfg/down/queued':
```

```
print(b64decode(recibido['downlink_queued']['frm_payload']).hex().upper())


def on_disconnect(client, userdata, rc):
    if rc != 0:
        print("Unexpected disconnection.")


def draw_figure(canvas, figure):
    figure_canvas_agg = FigureCanvasTkAgg(figure, canvas)
    figure_canvas_agg.draw()
    figure_canvas_agg.get_tk_widget().pack(side='top', fill='both', expand=1)
    return figure_canvas_agg


# funcion que teniendo los parámetros coje los elementos de una fuente de datos
# el tiempo se puede coger del propio mensaje del ttn
def mostrarGrafico(values, figura, ax):
    """funcion que muestra los datos obtenidos

    Args:
        values (list): lista de valores de los eventos
        figura (figure): figura de matplotlib
        ax (axis): objeto de ejes de matplotlib
    """
    # int(values['slider'])
    data = getDatos()
    datos = {}
    print(data)
    # ordenamos los datos en caso de que no esten ordenados en el archivo
    ordered_data = dict(sorted(data.items(), key=lambda x:
datetime.strptime(x[0], '%Y-%m-%d %H:%M'), reverse=False))

    for i, j in ordered_data.items():
        datos[datetime.strptime(i, '%Y-%m-%d %H:%M')] = j
    del data
    data = {}
    if values['-COMBO-'] == 'Meses':
        for i, j in datos.items():
            if i > (datetime.now() -
relativedelta(months=int(values['slider']))):
                data[i] = j
    elif values['-COMBO-'] == 'Semanas':
        for i, j in datos.items():
            if i > (datetime.now() - timedelta(weeks=int(values['slider']))):
                data[i] = j
    else:
        for i, j in datos.items():
            if i > (datetime.now() - timedelta(days=int(values['slider']))):
                data[i] = j
    del datos
```

```
    # al creacion del fig debe tener en cuenta los parametros
    ax.cla()
    ax.grid()

    for tick in ax.get_xticklabels():
        tick.set_rotation(45)
    ax.plot(list(data.keys()), list(data.values()))

    xfmt = matplotlib.dates.DateFormatter('%d-%m-%y %H:%M')
    ax.xaxis.set_major_formatter(xfmt)
    figura.draw()
    return


def enviarPaquete(client):
    listaEnvio = []
    inmediatos = 0
    global listaAcciones
    for num in ('0', '1', '2', '3'):
        for letra in ('A', 'C'):
            if "".join([letra, num]) in listaAcciones:
                inmediatos = (inmediatos << 1) + 1
            else:
                inmediatos = inmediatos << 1
    inmediatos = hex(int(inmediatos))
    inmediatos = inmediatos.split('x')[1] if len(inmediatos.split('x')[1]) == 2
else '0' + inmediatos.split('x')[1]
    listaEnvio.append(inmediatos)

    if 'horario' in listaAcciones:
        conf = getConfig()
        for valor in conf.values():
            for i in valor:
                if i == '0':
                    listaEnvio.append('00')
                else:
                    elem = hex(int(i))
                    elem = '0' + elem[elem.index('x') + 1:] if
len(elem[elem.index('x') + 1:]) == 1 else elem[elem.index('x') + 1:]
                    listaEnvio.append(elem)
    else:
        for i in range(8):
            listaEnvio.append('00')



    payload = ''.join([str(item) for item in listaEnvio])
    print(payload)
    pay_b64 = b64encode(bytes.fromhex(payload)).decode()
    pay = '{"downlinks":[{"f_port": 1,"frm_payload":"' + pay_b64 + '","priority":
"NORMAL"}]}'
    client.publish(topic='v3/tfg-icai@tfg-icai/devices/arduino-tfg/down/replace',
payload=pay, qos=0, retain=False)
```

```
    listaAcciones = []
    return


def mqtt_client_listener(client):
    client.loop_forever()

def timerEnd(client):
    global queued
    print('Activated timer')
    if queued:
        #window['warning'].update(visible=True, value='Enviado')
        enviarPaquete(client)
        queued = False

def checkFormat(values):
    tipo = ('gap', 'duracion')
    for num in ('0', '1', '2', '3'):
        primVacio = False
        for tpo in tipo:
            if tpo=='duracion':
                if primVacio and values["".join([tpo,num])]!="":
                    return False
                elif not primVacio and values["".join([tpo,num])]=="":
                    return False


            if not values["".join([tpo,num])].isdigit() and
values["".join([tpo,num])]!="":
                return False
            elif values["".join([tpo,num])]=="" and tpo=='gap':
                primVacio=True


    return True


def main():
    # setup MQTT
    access =
'NNSXS.VCOASZCJR2TD4MZ4DYCTDDIJHJDI52LQYWNXWHA.76LAW2FWLS56JOYBJIQB6Z7GTDWZ2F4WKD
MVESCENZXH6QGYVY3A'
    mqtt_address = 'eu2.cloud.thethings.industries'
    client = mqtt.Client()
    client.username_pw_set('tfg-icai@tfg-icai', password=access)
    client.on_connect = on_connect
    client.on_message = on_message
    client.on_disconnect = on_disconnect
    client.connect(host=mqtt_address, port=1883, keepalive=60)
    hilo = threading.Thread(target=mqtt_client_listener, args=(client,))
    hilo.start()

    # setup GUI
```

```
    fig = Figure()
    fig.autofmt_xdate()
    ax = fig.add_subplot(111)
    ax.set_xlabel("Fecha")
    ax.set_ylabel("% Humedad")
    ax.grid()
    fig_agg = draw_figure(window['Canvas'].TKCanvas, fig)
    global values
    while True:
        event, values = window.read()
        global listaAcciones, queued, primEnvio
        if event == sg.WIN_CLOSED:
            client.disconnect()
            break
        if event == 'graph':
            mostrarGrafico(values, fig_agg, ax)
        if event in ('A0', 'C0', 'A1', 'C1', 'A2', 'C2', 'A3', 'C3', 'Subir0',
'Subir1', 'Subir2', 'Subir3'):
            listaAcciones.append(event)
            if not primEnvio:
                window['warning'].update(visible=True, value='Enviado')
                enviarPaquete(client)
                primEnvio = True
            else:
                window['warning'].update(visible=True, value='el paquete se
enviará en los próximos 7 minutos')
                queued = True

        if event in ('SubirA'):
            if checkFormat(values):
                guardarConfig(values)
                listaAcciones.append('horario')
                if not primEnvio:
                    enviarPaquete(client)
                    window['warning'].update(visible=True, value='Enviado')
                    primEnvio = True
                else:
                    window['warning'].update(visible=True, value='el paquete se
enviará en los próximos 7 minutos')
                    queued = True
            else:
                window['warning'].update(visible=True, value='Valores incorrectos
en los campos(ambos de una misma válvula deben de estar rellenos y sólo de
números)')

if __name__ == "__main__":
    window = sg.Window('Control de Riego', layout=layout, resizable=True,
finalize=True)
    cargarConfig(window)
    primEnvio = False
    queued = False
    listaAcciones = []
    main()
```