Arduino. Sensorización y domotización de sistema de riego.

Rubén Jorge López-Tarruella Pereo
Grado en ingeniería informática
Arduino

Antoni Morell Pérez
Pere Tuset Peiró

Junio de 2018

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Arduino. Sensorización y domotización de sistema de riego.*
Nombre del autor:	*Rubén Jorge López-Tarruella Pereo*
Nombre del consultor/a:	*Antoni Morell Pérez*
Nombre del PRA:	*Pere Tuset Peiró*
Fecha de entrega (mm/aaaa):	06/2018
Titulación::	*Grado de Ingeniería Informática*
Área del Trabajo Final:	*TFG – Arduino*
Idioma del trabajo:	*Castellano*
Palabras clave	*Arduino, riego, sensores*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

El proyecto pretende abordar las necesidades existentes de actualización para los sistemas de riego actuales. Dichos sistemas no cubren satisfactoriamente las necesidades de programación en cultivos de nueva generación como los de riego hidropónico, además de una deficiente sensorización y de la imposibilidad de interacción telemática con ellos a bajo coste.

El dispositivo se desarrolla utilizando la plataforma de Arduino. Además, el dispositivo cuenta con sensores de temperatura y humedad del aire, sensores de humedad de la tierra (uno por zona a controlar) y la capacidad de servidor web conectado a través de un cable de red *Ethernet* la cual facilita la interacción con el dispositivo de forma telemática para configurarlo y consultar las alarmas registradas por los sensores.

El dispositivo permite configurar valores de alerta para los sensores de temperatura, humedad del aire y humedad de la tierra, así como horarios (cuatro horas de inicio diferente) o frecuencia (para los riegos hidropónicos que requieren una frecuencia mayor a cuatro riegos diarios) y duración del riego de forma totalmente independiente para cada zona.

Como conclusión, los dispositivos IoT cada vez están más extendidos gracias en gran medida a las posibilidades que da Arduino a la hora de desarrollar a bajo coste dispositivos con los que se satisfacen totalmente las necesidades existentes en cualquier ámbito, y no únicamente de forma parcial como sucede con los dispositivos comerciales.

Abstract (in English, 250 words or less):

The project aims to address the existing updating needs for current irrigation systems. These systems do not satisfactorily cover the programming needs of new generation crops such as hydroponic irrigation, as well as poor sensorization and the impossibility of telematic interaction at a low cost.

The device is developed using the Arduino platform. In addition, the device has air temperature and humidity sensors, soil humidity sensors (one per controlled zone), and web server capability connected via Ethernet cable, which facilitates the interaction with the device to configure it and check the alarms recorded by the sensors.

The device allows setting alert values for the sensors of temperature, air humidity and soil moisture, as well as schedule starting times (up to four different starting times) or adjusting the frequency (hydroponic cultivation could require more than four times a day) and independent irrigation duration, completely independent for each zone.

In conclusion, IoT devices are becoming more widespread thanks largely to Arduino's ability to develop low cost devices that fulfill the needs of any field, not only partially as with commercial devices.

Índice

1. Introducción

Se diseñará un sistema de control de riego automático que dispondrá de distintos tipos de sensores (humedad del aire, temperatura y humedad de la tierra). Dichos sensores, unido a los parámetros configurados (horario de riego, duración, repetición, etc.) decidirá de forma autónoma si se ha de realizar un riego no.

Haremos una breve introducción de los distintos sistemas de riego, y algunas particularidades de los mismos.

1.1. ¿Qué es un riego?

El riego consiste en aportar agua a los cultivos por medio del suelo para satisfacer sus necesidades hídricas que no fueron cubiertos mediante la precipitación. Se utiliza en la agricultura y en jardinería [1].

Dichos sistemas se pueden dividir en dos subfamilias, los riegos tradicionales y los sistemas actuales.

1.1.2. Sistemas de riego tradicionales

Se entiende por riego tradicional aquel que se construyen por canales que transportan el agua hacia las zonas agrícolas. Los canales desembocan normalmente en unas arquetas que disponen de compuertas que al abrir permiten la salida del agua. Estos sistemas de riego, ya en desuso, suponen un consumo elevado de agua y un desaprovechamiento de recursos, por lo que la tendencia es a cambiar hacia los sistemas modernos. Entre los sistemas de riego tradicionales se encuentran los siguientes tipos:

- Por arroyamiento o surcos.
- Por inundación o sumersión.
- Por infiltración o canales.
- Por drenaje.

[1]

1.1.3. Sistemas actuales de riego

Estos sistemas de riego suponen una reducción en el consumo de agua, y por lo tanto una optimización de recursos; un riego uniforme, obteniendo un caudal regulado mediante la presión del agua. A diferencia de los riegos tradicionales, la instalación de estos sistemas no requieren de construcciones, sino que la conducción de agua se realiza mediante tuberías de PVC hasta los puntos de riego determinado, en función del sistema de instalado. Dichos sistemas se encuentras controlados por unos sistemas programadores que controlan el paso del agua. Dentro de estos sistemas de riego encontramos los siguientes tipos:

- Por aspersión o difusión.

- Por goteo o riego localizado.
- Por hidroponía.

[1]

1.1.3.1. Riego por aspersión o difusión

En este tipo de sistema los elementos de riego son unas turbinas, aspersores o difusores. Aunque la utilización variará en función de las necesidades de cada zona, como elemento común destacar que se tratan de elementos de riego sectoriales. Es decir, cada elemento abarca unos determinados metros de radio a los que regará cuando funcionen. Esto quiere decir que los elementos deberán de ir sucediéndose a lo largo de la zona que se quiera regar hasta cubrir toda la extensión.

El tiempo de riego de unas zonas a otras con este tipo de riego puede variar, así como las veces que es necesario regar al día según las necesidades del terreno. En el caso de un jardín, no excederá de 3 o 4 veces al día y unos minutos cada vez.

[2]

1.1.3.2. Riego por goteo o riego localizado

Los elementos de riego en estas instalaciones son unos goteros que son instalados justo en las plantas que queramos que sean regadas. Al ser una acción localizada el consumo de agua se reduce considerablemente, al no ser un riego indiscriminado. Está muy extendido su uso en la agricultura, sobre todo en aquellos terrenos más secos donde los recursos de agua son muy limitados.

Para estos tipos de riego, la programación (tiempo de riego y horarios) varía al igual que en el caso anterior. Lo normal es una programación de un par veces al día de unos minutos cada una.

[3]

1.1.3.3. Riego por hidroponía

Este tipo de riego, cada vez más extendido en la agricultura, es algo especial y diferente a los vistos anteriormente. En este tipo de riego las plantas no se encuentran plantadas en tierra (no usan como sustrato la tierra), sino que en su lugar se usa fibra de coco, u otros elementos que directamente sustentan a la planta. En el riego por hidroponía la planta toma los nutrientes directamente del agua (solución nutritiva), por lo que tiene características especiales con respecto a los vistos anteriormente.

Las necesidades de riego de alguno de este tipo de riego, requieren que el riego se produzca cada pocos minutos, dado que se trata de mantener hidratada la planta en todo momento y vaya cogiendo los nutrientes de forma paulatina. El agua en ningún caso se pierde (excepto por evaporación) y forma parte de un circuito cerrado en el que cada cierto tiempo se ha de reponer el agua perdida por evaporación y equilibrar los nutrientes disueltos en el agua.

[4]

1.2. Sistemas programadores de riego

Se entiende por un sistema programador de riego aquel que nos permite definir unos horarios para automatizar el proceso de riego. Dicho sistema, normalmente, está encargado de controlar las válvulas (electroválvulas) que dan servicio a las diferentes zonas (áreas) en las que se suele dividir una parcela, plantación, invernadero, etc. Para su estudio lo dividiremos en dos tipos, los electromecánicos y electrónicos, y los inteligentes, en función del grado de la tecnología que utilizan en su implementación.

1.2.1. Sistemas electromecánicos y electrónicos

Los sistemas programadores electromecánicos y electrónicos engloban desde un programador de grifo (Ilustración 2), hasta un programador que controla múltiples zonas y con posibilidad de programar diferentes franjas horarias (Ilustración 1).

Ilustración 2

Ilustración 1

Estos sistemas presentan las siguientes limitaciones:

- Franja de riego única para todas las zonas. (Todas las zonas regarán a la misma hora, de forma secuencial. Para aclararlo con un ejemplo, no se puede configurar una hora de inicio para la zona 1 y otra hora de inicio para la zona 2; ambas deben de ser la misma hora, aunque la zona 2 no comenzará hasta haber terminado la zona 1.
- En el caso de riegos por hidroponía es importante realizar riegos de varios minutos en repeticiones cortas (cada hora o cada media hora), este tipo de programadores normalmente tienen limitadas la capacidad de poder gestionar este tipo de riegos, y o bien disponen de unas limitadas franjas horarias (de tres a cuatro franjas), o bien repeticiones cada cierto tiempo, siendo el mínimo de una hora. No existe un sistema que permita compatibilizar ambas opciones y tener una zona con un tipo de programación y otra con otro tipo de programación.
- Falta de sensorización. Los programadores más avanzados sí cuentan con cierta sensorización, en la que un sensor de lluvia determinará si se ha de regar o no, pero no controlan la humedad del aire, de la tierra en las diferentes zonas o la temperatura.

- Falta de domotización. Esta es la diferencia clave con respecto a lo que describiremos como sistemas de riego inteligentes. Estos sistemas no proveen de una configuración remota vía web, y en algunos casos no disponen ni de pantalla para poder configurarlos.

1.2.2. Sistemas de riego inteligentes

En esta área se incluiría el sistema que estamos desarrollar. En el mercado se pueden encontrar soluciones que permiten la domotización y sensorización del riego, aunque con ciertas limitaciones:

- Los sistemas consultados no permiten configurar la posibilidad de repetición cada cierto tiempo que si permiten los tradicionales de grifo, sino que permiten configurar diferentes franjas, y como ocurría en el caso de los tradicionales riegan de forma secuencial.
- Los sistemas requieren de una gran inversión. Existen diferentes tipos, desde los que permiten reutilizar la instalación de las electroválvulas tradicionales (GreenIQ.com) hasta los que hay de sustituir todas las electroválvulas y cuya inversión es aún mayor (Fliwer.com).
- El coste de los sistemas es desorbitado, teniendo en cuenta que un sistema como el propuesto podría igualar a uno de los sistemas tradicionales, mientras que estos sistemas disponibles llegan a ser entre cuatro y cinco veces el precio los sistemas tradicionales.

1.3. ¿Qué es Arduino?

Arduino es una plataforma electrónica de código abierto. Sus placas de están preparadas para leer entradas (sensores de luz, temperatura, sonido, pulsaciones de un botón, etc.) y convertir esto en una salida (encendiendo un LED, activando un motor, mandando un mensaje por internet, etc.).

Éstas utilizan diferentes microcontroladores y microprocesadores. En general, la placa Arduino consiste en un microcontrolador sobre una placa de circuito impreso que dispone de múltiples conectores que funcionan como puertos de entrada y salida. En dichos puertos se pueden conectar placas de expansión que incrementan la funcionalidad del modelo empleado.

El software que se utiliza para programar Arduino es un entorno de desarrollo (IDE) basado en entorno de *processing*, y la estructura del lenguaje de programación *Wiring*. Las placas son programadas mediante un ordenador donde se ejecuta el entorno de desarrollo y se carga la información mediante comunicación del puerto serie a la palca Arduino.

[5], [6]

1.3.1. ¿Por qué Arduino?

Arduino ofrece gran versatilidad a la hora de poder agregar componentes, tanto para sensorización como para emitir respuestas programadas en la placa. Es por ello que su uso se ha visto muy extendido en los últimos años creando

una gran comunidad internacional que cuenta con estudiantes, profesores, aficionados a la tecnología, etc.

Existen miles de proyectos realizados con Arduino, desde objetos de uso diario (conectar una tostadora a la red y accionarla a través del wifi), hasta complejos instrumentos para experimentos y mediciones científicas.

Aunque existen otras placas programables con microcontroladores disponibles en el mercado actualmente; Arduino ofrece varias ventajas:

- Es económico. El precio es bastante más bajos que sus competidoras.
- Multiplataforma. El software de desarrollo se puede ejecutar en cualquier sistema operativo para ordenadores (Windows, Macintosh OSX o Linux).
- Un entorno de desarrollo simple y limpio. Posee una interfaz de uso sencilla para los principiantes.
- Código abierto y software ampliable. El software de Arduino está publicado como herramientas de código abierto, disponible para que los programadores expertos puedan incrementar las funcionalidades del mismo.
- Código abierto y hardware ampliable. Los planos de las placas de Arduino se encuentran publicados bajo licencia *Creative Commons* lo que permite a los diseñadores de circuito experimentados crear sus propias versiones de los módulos, ahorrando así bastante dinero.

[5], [6]

2. Objetivos del proyecto

El proyecto tratará de dar una solución económica y viable a los sistemas de automatización de riego existentes. La implementación del dispositivo podrá realizarse en cualquier ámbito, desde un sistema de riego doméstico hasta una instalación de regadío para la agricultura.

Los sensores dotarán de la capacidad necesaria al sistema para decidir de forma autónoma si es necesario realizar un riego extra o por el contrario determinar si no es necesario realizar el que estaba previsto por configuración.

La configuración que el usuario podrá realizar por cada zona (el prototipo cuenta con dos zonas) será de los siguientes parámetros:

- Nombre de la zona, para una correcta identificación.
- Cuatro horas de inicio diferentes. Así cubrirá las necesidades de los riegos de aspersión y por goteo.
- Duración de cada riego.
- Repetición del riego, para los sistemas en los que no sea suficiente con cuatro horas de inicio se configurará cada cuántos minutos se ha de regar. Así cubrirá las necesidades de los riegos por hidroponía.
- Límites de humedad relativa del aire y humedad de la tierra. Por encima del valor máximo no hará falta regar y por debajo del valor mínimo habrá que aumentar el tiempo de riego al doble para el siguiente riego.
- Límite de temperatura, por encima del máximo se aumentará el tiempo de riego al doble y por debajo del mínimo no se regará.
- Encendido, donde se podrá activar y desactivar cada zona de forma independiente.

Además de los parámetros anteriores, el usuario podrá configurar la fecha y hora del sistema.

La interfaz web proporcionará la configuración telemática del dispositivo facilitando al usuario no tener que estar presentes físicamente en donde se encuentre el sistema de riego, asistiendo en el control y gestión del mismo.

Por último, el sistema generará una alarma al superar los umbrales máximos y mínimos establecidos en la configuración para los diferentes sensores. Dichas alarmas quedarán registradas en un *log* (un histórico de eventos), accesible de forma remota, junto con la fecha y hora a la que se produjo.

Con el fin de cumplir las expectativas y las necesidades del sistema se fijan como objetivos del proyecto los siguientes:

- El dispositivo ha de poder ser utilizado en la franja de temperaturas de 0º-50ºC sin distorsión de su funcionamiento.
- El dispositivo debe permitir conocer su configuración mediante la web (programación y hora actual) y permitir actualizarla desde la misma web que sirve.
- La latencia de respuesta del dispositivo a una conexión realizada mediante un navegador web dentro de la red local ha de ser inferior a los 2s.

- El acceso desde la web al dispositivo no debe bloquear el funcionamiento habitual del dispositivo.
- El dispositivo ha de revisar las lecturas de los sensores cada 10 minutos, y registrar una alerta cuando se superen los umbrales establecidos mediante la configuración web.
- El dispositivo no ha de perder la configuración actual en caso de corte del suministro de luz.
- El dispositivo debe de regar en las horas establecidas en la configuración web, con la duración determinada en función de la configuración web y de si existe alguna alarma registrada para dicho riego (cancelación o duplicación de la duración configurada).
- Tras una nueva configuración, el dispositivo ha de reiniciar las alertas registradas, para que no tengan efectos sobre la nueva configuración.
- El dispositivo debe permitir la consulta del log de alertas registradas, y su borrado completo desde la página web servida desde el propio dispositivo.

3. Descripción del proyecto

El diseño conceptual es el siguiente:

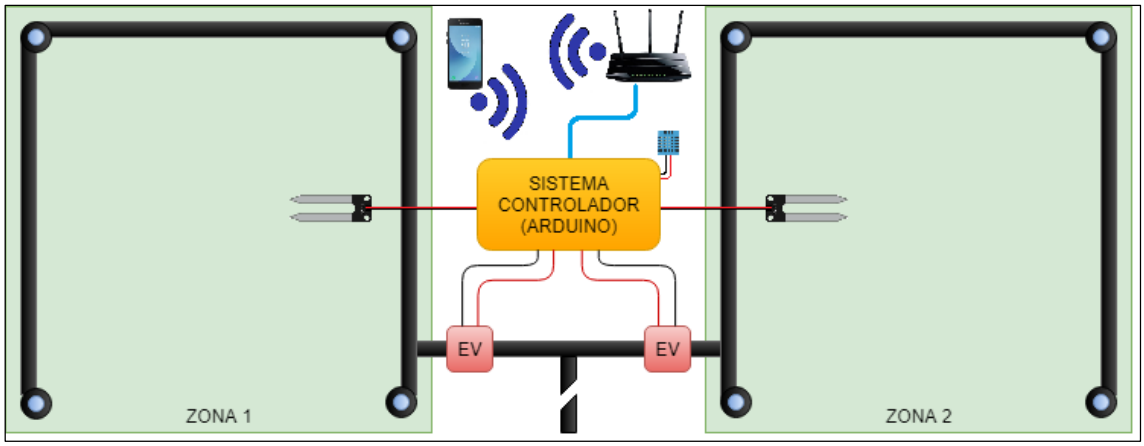


Ilustración 3

Los elementos de la Ilustración 3 marcados en las esquinas con puntos azules son los aspersores de riego para ambas zonas. Los dos cuadros rojos etiquetados como “EV” corresponden a las dos electroválvulas del sistema de riego. Ambas interpuestas en la tubería que está conectada a la corriente de agua (tubos en negro). Desde las dos electroválvulas parten dos cables, los cuales van conectados con el sistema controlador que es objeto de este proyecto. Dicho sistema controlador, está conectado a un *router*, y desde cualquier dispositivo (en la imagen se ha utilizado un teléfono móvil a modo de ejemplo) se podrá acceder a la configuración del sistema controlador e interactuar directamente con la web servida desde dicho dispositivo. El diseño no requiere de ningún servidor web externo, sino que es el propio Arduino el que hará las funciones de servidor de forma directa. Además de los elementos mencionados, el sistema de riego contará con tres sensores, dos para la humedad de la tierra de cada zona, y uno para la temperatura y humedad del aire. Dichos sensores tal como se aprecia en la imagen van conectados al sistema controlador Arduino.

El sistema consistirá en una placa programable Arduino, al que le conectaremos un módulo wifi para que el usuario pueda conectarse a una página web servida por el propio Arduino que permitirá configurar los valores límites de humedades (aire y tierra) y temperatura así como la programación temporal de las diferentes zonas de riego.

El prototipo constará con de los siguientes componentes:

- Placa programable Arduino Mega 2560 R3.
- Placa Ethernet tipo Wiznet W5100 con lector de micro-sd.
- Sensor de humedad del aire y temperatura del tipo DHT11.
- Dos sensores de humedad de tierra (higrómetro) tipo YL-69 o HL-69. (La diferencia entre los modelos es la resistencia a la corrosión).
- Módulo de reloj de precisión del tipo DS3231.

- Placa de dos relés.
- Tarjeta micro-SD.
- Fuente de alimentación de 5V para alimentar la placa programable.
- Fuente de alimentación de 24V para la apertura de las electroválvulas.
- Dos Electroválvulas para dar servicio a dos zonas de riego.

Esquema de conexiones de componentes:

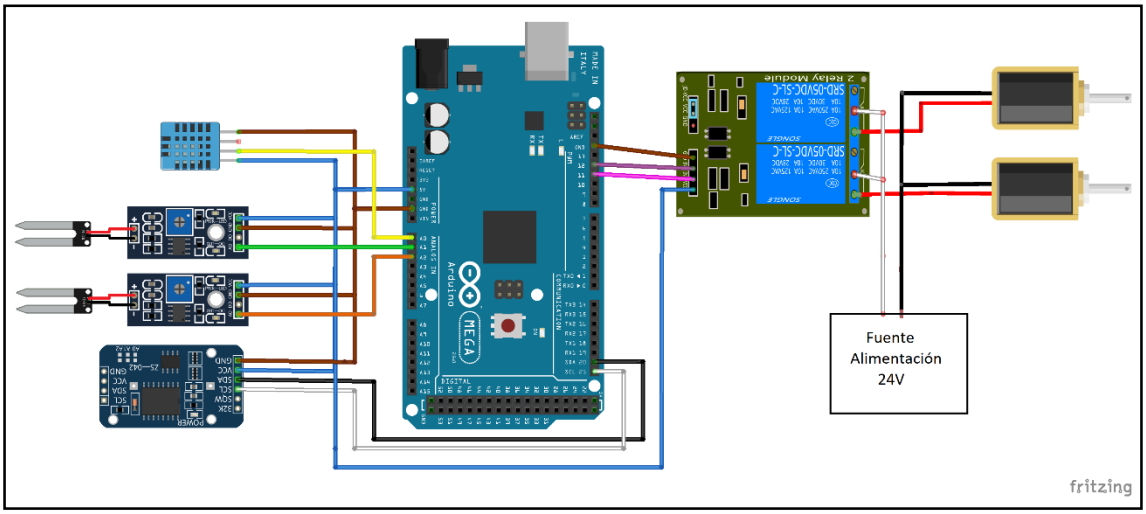


Ilustración 4

Esquema de la web de configuración:

Ilustración 5

4. Estudio de viabilidad económica

Se detallan a continuación los costes de fabricación del sistema de riego. Se trata del coste propio del prototipo, por lo que los costes asociados a una posible versión comercial del mismo serían o igual o inferiores en todo caso.

Costes de componentes:

Tabla 1

Componente	Sin IVA	Con IVA	Cantidad	Total
Arduino Mega 2560 R3	30,25 €	36,60 €	1	36,60 €
Ethernet Wiznet W5100	10,12 €	12,25 €	1	12,25 €
Sensor higrómetro YL-69	1,15 €	1,39 €	2	2,78 €
Sensor aire DHT11	2,30 €	2,78 €	1	2,78 €
Módulo DS3231	2,85 €	3,45 €	1	3,45 €
Placa de relés	2,72 €	3,30 €	1	3,30 €
Tarjeta micro-sd	8,26 €	10,00 €	1	10,00 €
Fuente alimentación 5V	8,26 €	10,00 €	1	10,00 €
Cableado interconexión	1,65 €	2,00 €	1	2,00 €
Total				**83,16 €**

Coste de componentes adicionales (se desglosan a parte dado que en una instalación común de riego estos componentes ya los tendrían, o bien son comunes a cualquier instalación de riego tradicional:

Tabla 2

Componente	Sin IVA	Con IVA	Cantidad	Total
Electroválvula 24V	9,04 €	10,94 €	2	21,88 €
Fuente alimentación 24V	8,43 €	10,20 €	1	10,20 €
Total				**32,08 €**

El tiempo medio de montaje para el hardware, teniendo en cuenta una posible comercialización futura sería de unas 2 horas por dispositivo. Evidentemente, con un incremento de 15 minutos más por cada dispositivo adicional que se realice a la vez.

Realizaremos el cálculo del coste de una hora teniendo en cuenta las tablas salariales actuales para un técnico electrónico (https://www.indeed.es/salaries/T%C3%A9cnico/a-electronico/a-Salaries):

Tabla 3

Concepto	Coste	Coste por hora (1826 h total)
Salario Bruto Anual	18.000 €	9,86 €
Contribución Contingencia común (23,60%)	4.248 €	2,33 €
Desempleo (6,70%)	1.206 €	0,66 €
Otros (0,80%)	144 €	0,08 €
Total	**23.598 €**	**12,93 €**

Tabla de coste por dispositivos (estimación):

Tabla 4

Número de dispositivos	Horas	Total coste	Coste unitario
1	2	25,86 €	**25,86 €**
2	2,25	29,09 €	**14,54 €**
3	2,5	32,32 €	**10,77 €**
10	4,25	54,95 €	**5,49 €**
100	26,75	345,87 €	**3,45 €**

Como se aprecia, el coste de montaje a medida que aumenta la producción disminuye drásticamente, por lo que los costes asociados al montaje serían insignificantes. Al día un técnico podría montar unos 20 dispositivos.

No se han tenido en cuenta hasta ahora otros costes, como el de la carcasa que contendría el dispositivo, pero realizando un sondeo estarían estimados en 1€ por dispositivo para pedidos de 100 unidades.

Coste total del prototipo (sin elementos comunes ni horas de desarrollo software):

Tabla 5

Concepto	Total coste
Materiales	83,16 €
Mano de obra	25,86 €
Total	**109,02 €**

En base a estos cálculos, el coste del prototipo sigue siendo inferior a los actuales sistemas de riegos comerciales que aportan similares características.

A todo ello, hay que tener en cuenta que los costes de los materiales puede verse disminuidos drásticamente si se opta por otras placas de montaje más baratas con las mismas prestaciones (Elengoo tiene una placa totalmente compatible por 12€, XCSOURCE tiene otra por 13€), o bien una vez terminado el código analizar las necesidades de computación y de memoria reales y cambiar el modelo de la placa de montaje utilizada por una de inferiores características. (Arduino UNO 20€, Arduino Leonardo 16€, Arduino Micro 16€); o una combinación de ambas opciones.

Además, como ya comentamos antes, los costes de los materiales al comprarse de forma individual son más elevados que si se comparan en grandes cantidades para poder llevar a cabo una producción con fines comerciales.

Los costes asociados a la creación y mantenimiento de una empresa creemos están exentos de ser estudiados en este apartado, y corresponderían a un posterior estudio de viabilidad de empresas y no al de viabilidad propia del proyecto en cuestión.

5. Planificación del proyecto

5.1. Primera aproximación

El sistema lo dividiremos en dos partes, una la correspondiente al hardware y otra al software.

El Hardware, a su vez lo dividiremos en otras dos, una parte lógica y una funcional:

- **Lógica:** Contemplará la instalación del hardware requerido y los componentes adicionales (placa wifi, relés, sensor de humedad y temperatura, sensor higrómetro y reloj).
- **Funcional:** Las electroválvulas para el control del riego.

El software se diseñará por módulos. Atendiendo a las diferentes necesidades del sistema:

- Módulo de inicialización, donde se realizarán los *setup* iniciales que requiera el hardware y los sensores instalados. También se realizará la lectura de los valores configurados previamente los cuales quedarán almacenados en un fichero en una tarjeta de memoria micro-sd.
- Módulo de servidor web, el cual atenderá a las diferentes peticiones que se realicen desde los clientes sirviéndole una web para configurar las opciones del sistema y que mostrará los valores de temperatura y humedad relativa del aire actuales. Dicho módulo realizará una llamada al módulo de lectura de la tarjeta de memoria para servir el fichero solicitado.
- Módulo de sensores, inicialmente se recogerán en un único módulo las lecturas de todos los sensores, aunque se prevé que cada tipo de sensor tenga una función específica que será llamada por la función principal. Este módulo llamará al módulo de escritura en caso que sea necesario registrar una incidencia en las condiciones ambientales.
- Módulo de riego, en el que teniendo en cuenta las lecturas de los sensores y la configuración actual del sistema se decidirá en función de la hora si se ha de regar o si se ha de parar de regar.
- Módulo de lectura/escritura de ficheros de la tarjeta de memoria. Se utilizará tanto para cargar los valores de configuración actuales al arrancare sistema, como para la escritura de los mismos tras ser modificado por el usuario a través de la web. También leerá la página web almacenada para poder devolverla al módulo de servidor web. Seguramente dicho módulo habrá que dividirlo en dos funciones, una para la lectura y otra para la escritura.
- Módulo de reloj, el cual permitirá obtener la hora actual y modificarla en caso que lo solicite el usuario a través de la web.
- Módulo principal, en el que en principio se ejecutarán los módulos de servidor web, sensores y riego de forma continuada con la técnica de *polling*. Se intentará implementar un sistema de interrupciones (ISR) atendiendo a las posibilidades de Arduino de atender a las interrupciones de tipo *Timers* y de tipo hardware, aunque únicamente atiende a las interrupciones digitales sí sería conveniente implementar los *timers* para que la lectura de sensores no

se realice excesivas veces, sino una vez por segundo o incluso cada más tiempo.

- Página web. El desarrollo de la página web se realizará en HTML y JavaScript y será almacenada en la tarjeta de memoria.

- Fichero de configuración en JSON o XML para almacenar los diferentes valores configurados.

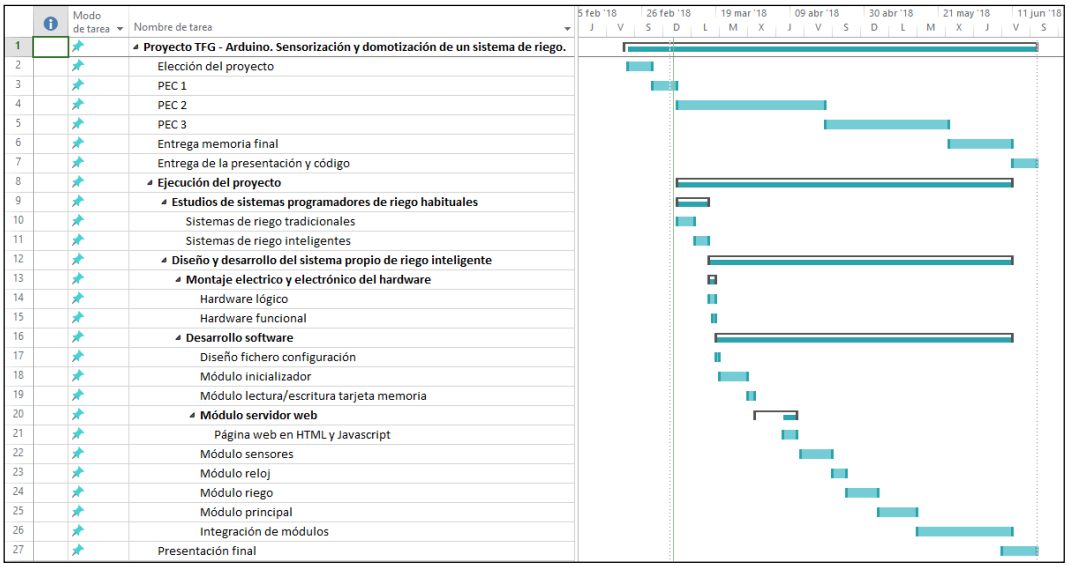
5.2. Planificación temporal

A continuación la lista de tareas del proyecto:

	Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin
1		**Proyecto TFG - Arduino. Sensorización y domotización de un sistema de riego.**	**84 días**	**mié 21/02/18**	**dom 17/06/18**
2		Elección del proyecto	5 días	jue 22/02/18	mié 28/02/18
3		PEC 1	5 días	jue 01/03/18	mié 07/03/18
4		PEC 2	30 días	jue 08/03/18	mié 18/04/18
5		PEC 3	25 días	jue 19/04/18	mié 23/05/18
6		Entrega memoria final	13 días	jue 24/05/18	dom 10/06/18
7		Entrega de la presentación y código	6 días	lun 11/06/18	dom 17/06/18
8		**Ejecución del proyecto**	**68 días**	**jue 08/03/18**	**dom 10/06/18**
9		**Estudios de sistemas programadores de riego habituales**	**7 días**	**jue 08/03/18**	**vie 16/03/18**
10		Sistemas de riego tradicionales	3 días	jue 08/03/18	lun 12/03/18
11		Sistemas de riego inteligentes	4 días	mar 13/03/18	vie 16/03/18
12		**Diseño y desarrollo del sistema propio de riego inteligente**	**62 días**	**sáb 17/03/18**	**dom 10/06/18**
13		**Montaje electrico y electrónico del hardware**	**2 días**	**sáb 17/03/18**	**dom 18/03/18**
14		Hardware lógico	2 días	sáb 17/03/18	dom 18/03/18
15		Hardware funcional	1 día	dom 18/03/18	dom 18/03/18
16		**Desarrollo software**	**61 días**	**lun 19/03/18**	**dom 10/06/18**
17		Diseño fichero configuración	1 día	lun 19/03/18	lun 19/03/18
18		Módulo inicializador	6 días	mar 20/03/18	mar 27/03/18
19		Módulo lectura/escritura tarjeta memoria	2 días	mié 28/03/18	jue 29/03/18
20		**Módulo servidor web**	**8 días**	**vie 30/03/18**	**mar 10/04/18**
21		Página web en HTML y Javascript	3 días	sáb 07/04/18	mar 10/04/18
22		Módulo sensores	7 días	jue 12/04/18	vie 20/04/18
23		Módulo reloj	3 días	sáb 21/04/18	mar 24/04/18
24		Módulo riego	7 días	mié 25/04/18	jue 03/05/18
25		Módulo principal	7 días	vie 04/05/18	lun 14/05/18
26		Integración de módulos	20 días	mar 15/05/18	dom 10/06/18
27		Presentación final	7 días	vie 08/06/18	dom 17/06/18

Ilustración 6

Diagrama de Gantt:



	Modo de tarea	Nombre de tarea
1		**Proyecto TFG - Arduino. Sensorización y domotización de un sistema de riego.**
2		Elección del proyecto
3		PEC 1
4		PEC 2
5		PEC 3
6		Entrega memoria final
7		Entrega de la presentación y código
8		**Ejecución del proyecto**
9		**Estudios de sistemas programadores de riego habituales**
10		Sistemas de riego tradicionales
11		Sistemas de riego inteligentes
12		**Diseño y desarrollo del sistema propio de riego inteligente**
13		**Montaje electrico y electrónico del hardware**
14		Hardware lógico
15		Hardware funcional
16		**Desarrollo software**
17		Diseño fichero configuración
18		Módulo inicializador
19		Módulo lectura/escritura tarjeta memoria
20		**Módulo servidor web**
21		Página web en HTML y Javascript
22		Módulo sensores
23		Módulo reloj
24		Módulo riego
25		Módulo principal
26		Integración de módulos
27		Presentación final

Ilustración 7

6. Diseño y desarrollo

6.1. Desarrollo Hardware

A continuación se explicará cuáles han sido los componentes utilizados y sus características, cómo se han llevado a cabo las conexiones, y los problemas encontrados durante el proceso.

6.1.1. Componentes seleccionados

6.1.1.1 Placa programable Arduino Mega 2560 R3

La selección de la placa Arduino Mega 2560 R3 fue debido a su gran versatilidad a la hora de poder agregarle componentes para aumentar su conectividad. El interés principal del proyecto es el de realizar un sistema de riego que pudiera ser controlado de forma telemática. Esta palca, a diferencia de otras placas Arduino, debido a la cantidad de pines disponibles permite poder optar por diferentes soluciones, siendo una de ellas la conexión de un módulo *Ethernet* o wifi.

Si bien otras placas ya incluyen el módulo Wifi y *ethernet* integrados (Arduino Yun), el coste de la misma es muy elevado.

Por otro lado, las características que expondremos a continuación (tabla 6), son superiores al resto de la gama de precios similares, y por la cantidad de pines disponibles para la conexión de sensores y relés futuros, nos permitía en un futuro incrementar el número de zonas que controlaría, sin que esto conllevara un cambio de la placa programable.

Tabla 6 [7]

Modelo	Micro	One	Leonardo	Mega	DUE
Microcontrolador	ATmega32U4	ATmega328	ATmega32U4	ATmega2560	ATmelSAM3U4E
Frecuencia Reloj	16 MHz	16 MHz	16 MHz	16 MHz	96 MHz
Memoria Flash	32 KB	32KB	32KB	256KB	256KB
SDRAM	2.5 KB	2 KB	3.3 KB	8KB	50KB
Pines de E/S Digitales	20	14	14	54	54
Pines Analógicos	12	6	6	16	16

La selección de la placa también se basó en el desconocimiento inicial de cuánta memoria requeriría el programa a desarrollar en cuestión, tanto flash como SDRAM. Esto, aunado a la necesidad de tener pines de conexiones analógicos y digitales en cantidad para posibles ampliaciones, dejo como posibles opciones la placa Arduino Mega y DUE.

El precio, similar de ambas, hacía que la placa DUE fuera más atractiva, dada sus mejores características técnicas (mayor velocidad de procesamiento y mayor memoria SDRAM). La elección final se basó únicamente en disponibilidad de stock y rapidez a la hora de recibir los componentes, por lo que se optó por la placa Mega.

Dicha placa contiene 16 pin analógicos, lo que a priori nos permitiría conectar hasta 16 sensores higrómetros, lo que es un número muy respetable a la hora de una posible comercialización futura con vistas a ámbitos profesionales.

Una vez concluya el desarrollo del prototipo, y para una reducción de costes se debería de contemplar opciones inferiores, para ello deberíamos analizar las necesidades reales del código generado.

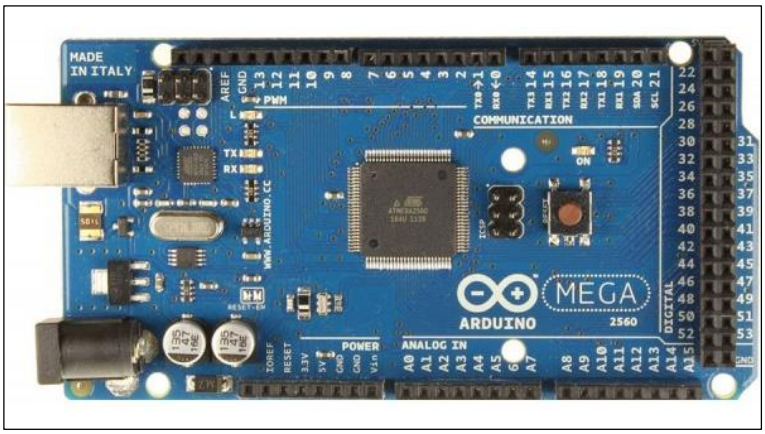


Ilustración 8

6.1.1.2. Placa Ethernet tipo Wiznet W5100 con lector micro-SD.

Entre las diferentes opciones de placa *Ethernet*, la selección de esta placa se ha debido a que se requería disponer de un lector micro-SD para el proyecto, y con esta placa se solucionaban ambos problemas a la vez.

La placa utiliza un controlador *Ethernet* del tipo W5100 con un buffer interno de 16K. Con una velocidad de conexión de 10/100Mb, lo cual nos es más que suficiente para el proyecto que queremos realizar, dado que únicamente tiene que servir una pequeña página web con datos. La conexión con la placa arduino se realiza a través del puerto SPI a través de los pines digitales 10, 11, 12 y 13. Para el manejo de la SD además utilizará el pin 4 como SS (*Slave Select*, pin utilizado para habilitar o deshabilitar un dispositivo específico).

Tiene soporte para ser alimentada a través del cable *Ethernet* (PoE), por lo que permitiría que la alimentación llegara al prototipo únicamente con un cable *Ethernet*, sin necesidad de tener otra conexión de corriente en el punto de instalación del mismo.

[8], [9]

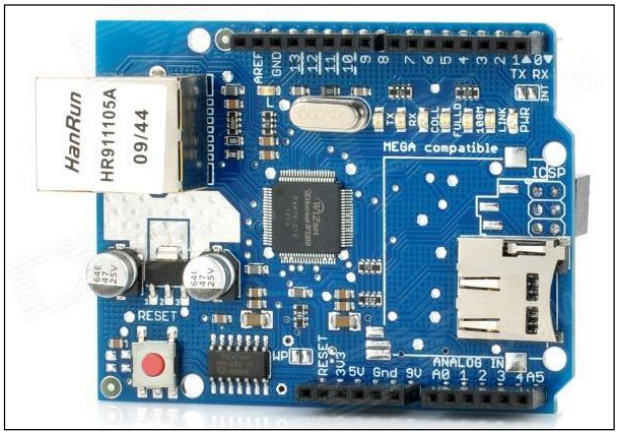


Ilustración 9

6.1.1.3. Sensor de humedad del aire y temperatura

A la hora de elegir el sensor de humedad y temperatura, existían dos factores a tener en cuenta, uno de ellos era el económico, y otro el rango de los valores obtenidos. A continuación comparativa entre los distintos sensores disponibles:

Tabla 7 [10]

Modelo	DHT11	DHT22
Precio	2-3€	6-10€
Potencia de trabajo	3 a 5 V	3 a 5 V
Rango de humedad	20 - 80% con precisión del 5%	0-100% con precisión del 2-5%
Rango de temperatura	0 - 50ºC con precisión ±2ºC	-40 – 80ºC con precisión ±0.5ºC
Muestras máximas	Una por segundo	Una cada dos segundos

Teniendo en cuenta las necesidades del proyecto, y el ambiente en el que se va a probar, finalmente elegimos el modelo DHT11. No se espera que la temperatura decaiga por debajo de 0ºC, y tampoco una humedad inferior al 20%, en cuanto a la exactitud de los valores, el prototipo no requiere de una exactitud máxima a la hora de tomar los valores. Por lo que finalmente primó el factor económico en la elección.

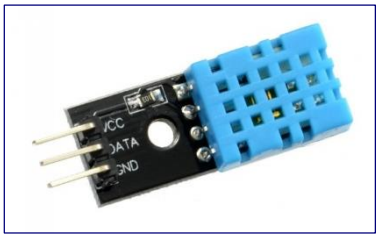


Ilustración 10

6.1.1.4. Sensor de humedad de tierra (higrómetro)

Para la elección de los sensores higrómetros, analizando la escasa información encontrada de las diferencias entre un tipo y otro (YL, FC), optamos finalmente por la solución más económica y que menos tardaban en suministrarnos. El precio oscila de los 0,4€ a los 0,8€ por unidad.

Las características del higrómetro seleccionado finalmente (YL-69) son las siguientes:

Tabla 8 [11]

Voltaje de alimentación	2 – 6V
Voltaje de salida	0-4.2V
Comparador	LM393
Dimensiones	60 x 20 mm

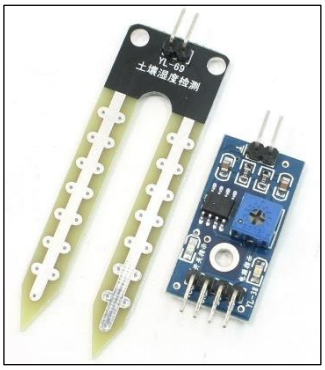


Ilustración 11

6.1.1.5. Módulo de reloj de precisión

En el proyecto se requiere un módulo de reloj para poder almacenar la hora configurada en el dispositivo para poder realizar los riegos a las horas programadas. Existen multitud de opciones disponibles en el mercado, siendo el más común el que hemos utilizado en el proyecto finalmente. Únicamente debía de cumplir el requisito de que almacenara la hora, y la guardara en caso de perder la corriente, por ello el factor económico ha sido determinante finalmente como en otros componentes.

El módulo DS3231 se trata de un reloj en tiempo real de precisión, el cual cuenta con un oscilador a cristal con compensación de temperaturas. Posee además una batería auxiliar que le permitirá mantener la hora sin perderla.

Las características del módulo a continuación:

Tabla 9 [12]

Voltaje de alimentación	3.3 – 5V
Exactitud	±2ppm
Rango de trabajo	0ºC – 40ºC
Reloj	DS3231
Memoria	EEPROM I2C

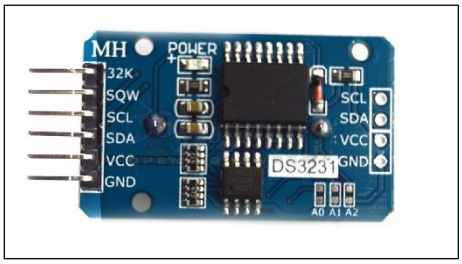


Ilustración 12

6.1.1.6. Resto de componentes

El resto de componentes seleccionados han sido los siguientes:

- Placa con 2 relés.
- Tarjeta micro-SD de 2 GB.
- Fuente de alimentación de 5V para alimentar la placa programable.
- Fuente de alimentación de 24V para la apertura de las electroválvulas.
- Dos Electroválvulas para dar servicio a dos zonas de riego.

6.1.2. Esquema de conexiones

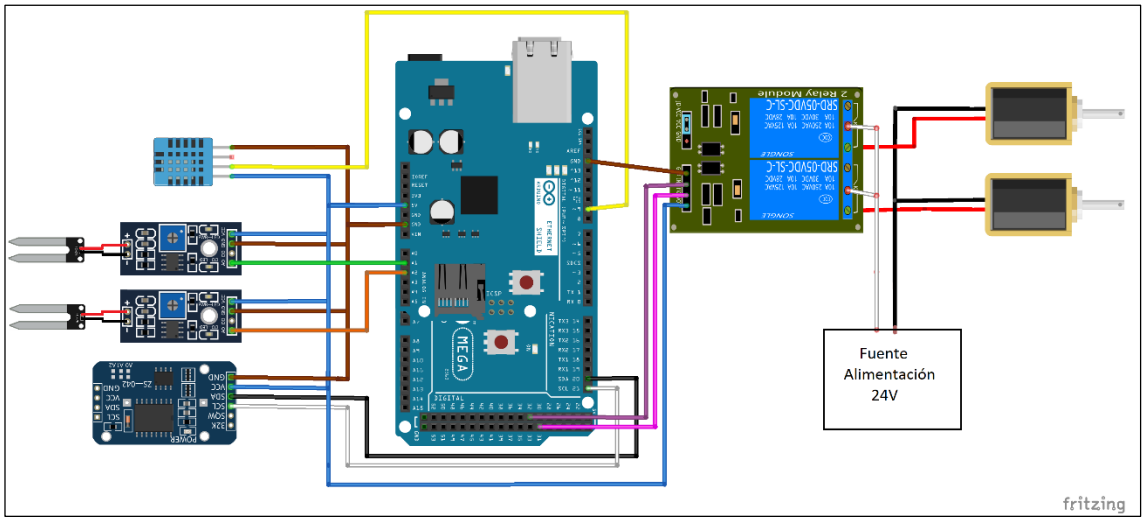


Ilustración 13

6.1.2.1. Detalle de las conexiones

Dado que en la Ilustración 13 no se puede apreciar correctamente, a continuación pondré un esquema de los diferentes componentes y de las conexiones existentes entre ellos, así como la utilidad de cada conexión.

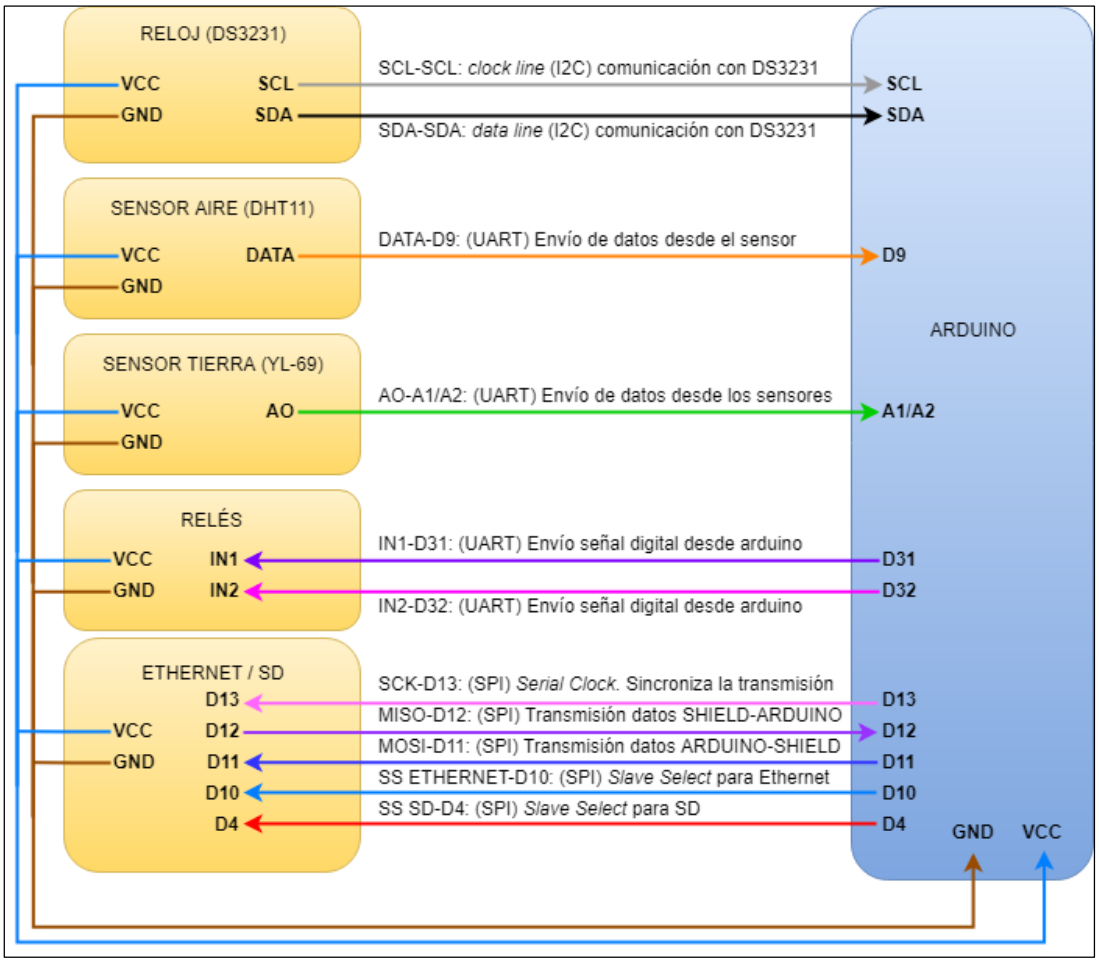


Ilustración 14

Resumen de las conexiones indicadas en la Ilustración 14:

Tabla 10

Componente	**PIN**	**Función**	**Arduino**
Reloj (DS3231)	SCL	Sincronización con Arduino	SCL
	SDA	Transmisión de datos	SDA
Sensor Aire (DHT11)	DATA	Envío de datos desde sensor	D9
Sensor Tierra 1 (YL-69)	AO	Envío de datos desde sensor	A1
Sensor Tierra 2 (YL-69)	AO	Envío de datos desde sensor	A2
Relés	IN1	Envío señal desde Arduino	D31
	IN2	Envío señal desde Arduino	D32
Ethernet/SD	SCK	Sincronización de transmisión	D13
	MISO	Envío de datos desde *shield*	D12
	MOSI	Envío de datos desde Arduino	D11
	SS ETHERNET	*Slave Select* para *Ethernet*	D10
	SS SD	*Slave Select* para SD	D4

En la comunicación entre los diferentes componentes interviene tres tipos de protocolo, en función de cómo se realice la transmisión de información:

- UART: Recepción y transmisión asíncrona universal. Utiliza una línea de datos simple para transmitir. Es el caso de los sensores de utilizados y los relés.
- I2C: Es un protocolo síncrono. Requiere de dos líneas de conexión entre el dispositivo y el arduino. Una de ellas es el enlace que se utiliza para sincronizar la señal de reloj desde el arduino al componente (SCL – *Clock Line*), la otra se utiliza para realizar la transmisión de datos (SDA – *Data Line*). Es el caso del reloj (DS3231).
- SPI: Es un protocolo síncrono. Utiliza mínimo tres líneas de comunicación. Una que se utiliza como línea para sincronizar la transmisión (SCK – *Serial Clock*) desde el arduino hacia el componente. Las otras dos líneas son para la transmisión de los datos, una desde el arduino hacia el componente (MOSI – *Master Out Slave In*) y otra desde el componente al arduino (MISO – *Master In Slave Out*). Además de estas tres líneas de comunicación, es necesaria una más por cada componente esclavo, que se utilizará para controlar la selección del esclavo a utilizar desde el arduino. Es el caso de las líneas indicadas como SS (*Slave Select*). Pla placa *Ethernet*/SD utiliza este protocolo de comunicaciones con el microcontrolador Arduino.

[13]

Además de las conexiones existentes entre los componentes propios del controlador de riego (Arduino) vistos en Ilustración 14 y Tabla 10, dicho controlador estará conectado a través de un cable *Ethernet* con un *router*, y a través de esta conexión será posible realizar la comunicación desde cualquier dispositivo (móvil, Tablet, portátil) a la web que será servida desde el Arduino. Para ello se utilizará el protocolo de comunicaciones HTTP, implementado sobre el protocolo IPv4. El dispositivo arduino gestionará y dará respuesta directamente a peticiones HTTP del tipo POST, GET y PUT.

6.1.3. Problemas encontrados

Un problema detectado durante las pruebas, ha sido la existencia de una incompatibilidad entre el pin seleccionado inicialmente para conectar el sensor DHT11 y la placa *Ethernet*. El servidor *Ethernet* testeado durante las pruebas funcionaba con normalidad, y los sensores también cuando fueron testeados funcionaban sin problemas. Durante una de las pruebas realizadas se conectaron todos los sensores en las posiciones inicialmente designadas, pero sin llevar a cabo ninguna configuración de los mismos, ni tampoco ninguna interactuación con los mismos, pero en dichas pruebas el servidor *Ethernet* dejaba de responder a las peticiones de los clientes. Se procedió a aislar uno a uno los componentes, hasta determinar que el fallo venía producido por el sensor DHT11, el cual se encontraba conectado en el D10. En su lugar, se probó a conectar en el pin D9, dándose por resuelto el problema de bloqueo. Esto se debía a que la placa *Ethernet* utiliza los pines 10-13 para comunicarse con la placa Arduino.

Aunque inicialmente los pines escogidos para los relés eran el D11 y D12, durante el montaje, y por el problema detectado con la placa *Ethernet*, se eligió conectarlos a los pines 31 y 32.

A continuación imágenes del conexionado real:

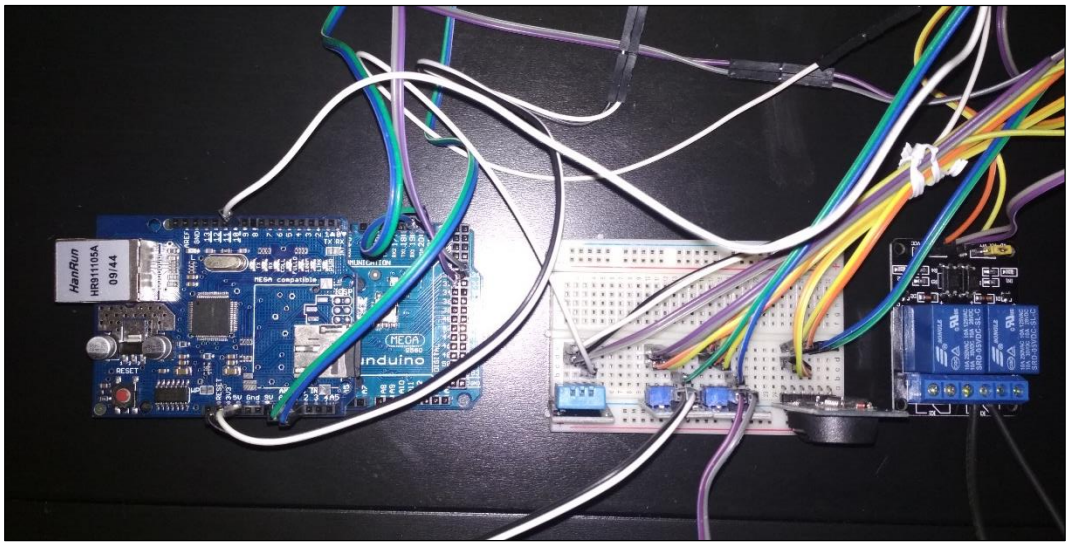


Ilustración 15

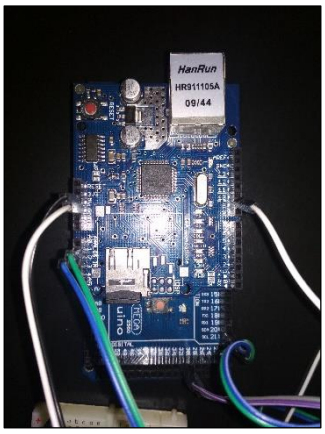


Ilustración 17

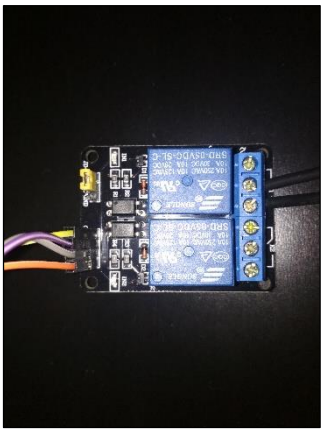


Ilustración 16

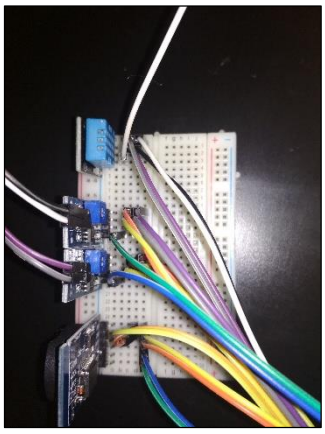


Ilustración 18

6.2. Desarrollo Software

6.2.1. Entorno de desarrollo

Se ha decidido utilizar como entorno de desarrollo Visual Code Studio, ya que este incluye soporte para Arduino y es una interfaz más cómoda que la propia IDE de Arduino. Aun así, hemos instalado igualmente la IDE para evitar posibles problemas de drivers.

El entorno Visual Code Studio tiene la capacidad de implementar *plugins* para un desarrollo software más fácil y rápido.

En dicho entorno hemos procedido a instalar las librerías que hemos ido necesitando para realizar los test y para el desarrollo propio del software.

6.2.2. Fichero de configuración

Se ha optado por un fichero de configuración tipo JSON. La estructura de dicho fichero se basa en un objeto inicial que contiene un *array* de objetos, uno por cada zona de riego.

Aunque inicialmente el fichero de configuración se diseñó con el nombre de los campos en español, para ahorrar posibles problemas futuros con el mapa de caracteres se ha optado finalmente por traducirlos al inglés.

Podemos observar que se ha escogido una estructura en la que los programas están recogidos como un conjunto de objetos también, y en la que cada sensor tiene asignada una estructura con dos atributos cada uno.

La estructura del fichero es la que sigue:

```
{    "zones": [
       {
          "name": "zona 1",
          "program": [ {"hour": 8, "minute": 0},
             {"hour": 11, "minute": 0},
             {"hour": 20, "minute": 20},
             {"hour": 22, "minute": 30}],
          "duration": 10,
          "repetition": {"on": false, "frequency": 0},
          "air": {"max": 80, "min": 40},
          "land": {"max": 800, "min": 500},
          "temp": {"max": 40,"min": 5},
          "on": true},
       {…}]}
```

Código 1

6.2.3. Test de hardware y librerías

Especificaremos los test llevados a cabo dividiéndolos por módulos:

- Módulo RTC (Reloj). Para ello se han utilizado las librerías *RTClib.h* versión 1.2.1, desarrollada por Adafruit (https://github.com/adafruit/RTClib). Se realizaron pruebas de escritura de la hora en la memoria del dispositivo DS3231, y lectura de la misma. Todas las pruebas fueron bien.
- Módulo SD. Se han utilizado las librerías *SD.h* versión 1.1.1, desarrollada por Arduino y SparkFun (https://www.arduino.cc/en/Reference/SD); y *SPI.h* versión 1.0.0, desarrollada por Arduino

(https://www.arduino.cc/en/Reference/SPI). La librería SD.h utiliza los protocolos de comunicación SPI, es por ello que se ha de incluir la librería *SPI.h* en dicho módulo.

Cuando se abre un fichero utilizando la librería, se crea un puntero de memoria que apunta al inicio del fichero. Dicho puntero se va actualizando a medida que avanzamos en la lectura del fichero. En el caso de la escritura, el puntero apunta al final del fichero abierto, lo que permite continuar agregando líneas al final del mismo.

Se realizaron pruebas de lectura y escritura de ficheros en una tarjeta SD. Todas las pruebas fueron bien. Tal y como se ha indicado, al escribir en un fichero existente se incrementaba, no se creaba de nuevo, por lo que en nuestro proyecto tendremos que borrar el fichero de configuración antes de guardar la nueva configuración. También se ha detectado que la librería no maneja ficheros con extensiones superiores a cuatro caracteres, por lo que el nombre de los ficheros de configuración y web (inicialmente *config.json* e *index.html*) pasará a tener extensión *.txt* y *.htm* respectivamente.

- Módulo Servidor. Se ha utilizado la librería *Ethernet.h* versión 1.1.2, desarrollada por Arduino (https://www.arduino.cc/en/Reference/Ethernet). Las funciones de comunicación utilizando el componente *Ethernet* utiliza la librería *Ethernet.h* mencionada anteriormente. Además de esta, dado que el componente utiliza el protocolo de comunicaciones SPI con la placa arduino, se ha de incluir en dicho módulo el uso de la librería *SPI.h*.

  La librería *Ethernet* establece un buffer de comunicación entre el cliente conectado y el servidor (arduino), el cual se utiliza tanto para leer datos enviados desde el cliente (peticiones) o para enviar datos desde el servidor (respuestas).

  Se realizaron pruebas de crear un servidor web que atendiera a las peticiones lanzadas desde un navegador. Posteriormente, tras finalizar las pruebas de todos los módulos se procedió a realizar pruebas de enviar una web alojada en una tarjeta micro-SD. Para ello se utilizó las librerías del módulo SD mencionadas anteriormente. Durante las pruebas, se encontró la incidencia indicada con anterioridad con la conexión del pin D10 del sensor de DHT11, la cual se solucionó reasignando al pin D9.

- Módulo Sensores. Se ha utilizado la librería *DHT.h* versión 1.3.0, desarrollada por Adafruit (https://github.com/adafruit/DHT-sensor-library). Se realizaron pruebas de lecturas para los diferentes sensores, cambiando las condiciones en las que se encontraban. Se detectó en concreto que para el módulo DTH las mediciones en intervalos inferiores a dos segundos dejan de ser muy precisas, por lo que se aconseja que como mínimo el intervalo de lectura sea de dos segundos.

- Módulo Relés. Se utilizó la librería *SPI.h* versión 1.0.0, desarrollada por Arduino (https://www.arduino.cc/en/Reference/SPI). Se realizaron pruebas de activar y desactivar ambos relés de forma separada y conjunta. Todas las pruebas fueron correctas.

Además de las librerías mencionadas anteriormente para los diferentes módulos también se ha utilizado la librería *ArduinoJson.h* versión 5.13.1, desarrollada por Benoit Blanchon (https://github.com/bblanchon/ArduinoJson). Con dicha librería, unida a las librerías del módulo SD para la lectura y escritura

de ficheros, se realizaron pruebas para la carga de datos desde un fichero con estructura JSON, y el guardado del mismo en un fichero del mismo tipo.

Dicha librería crea un buffer donde se captura la estructura JSON procedente de un buffer de lectura de un fichero directamente, o de una cadena de caracteres. La librería permite crear diferente tipos de objetos (arrays u objetos) que a través de punteros de memoria apuntan al dato en concreto que queremos acceder, o a los contenidos dentro de este encadenando una sucesión de objetos y *arrrays*.

Aunque se valoró la utilización de la librería *EasyWebServer*, desarrollada por Kalle Lundberg (https://github.com/llelundberg/EasyWebServer) para el manejo de las peticiones HTTP al servidor finalmente se decidió a programar una solución particular dado que la mencionada librería no gestionaba las peticiones del tipo POST y PUT generadas por los formularios web.

Se ha incluido la librería *TaskScheduler* versión 2.6.1, desarrollada por Anatoli Arkhipenko (https://github.com/arkhipenko/TaskScheduler) para crear y gestionar dos tareas de las tres tareas repetitivas (*checkProgram* y *regAlarm*).

6.2.4. Estructura del programa

Antes de entrar en cada uno de los módulos, analizaremos un diagrama para comprender mejor el funcionamiento del mismo (ver Ilustración 19).

En el diagrama se puede observar divida en dos zonas, una correspondiente a la función setup (en azul), donde se realiza la inicialización de todos los módulos y configuraciones iniciales pertinentes, y otra a la *loop* (en verde), que es la parte del programa que se iterará permanentemente.

Durante el desarrollo del software, se ha encontrado incompatibilidades a la hora de realizar la toma de valores en lugar de mediante *pooling* hacerlo con *timers*, por lo que finalmente se ha optado por una solución intermedia. Utilizando la librería *TaskScheduler* hemos logrado implementar un gestor de tareas al cual le especificamos dos tareas (*taskCheck* y *taskSensors*) y le asignamos una periodicidad a cada una de ellas (1 minuto y 10 minutos respectivamente). Llamando a la función *runner.execute()* el solo se encarga de comprobar si trascurrió los ciclos de reloj convenidos o no para proceder a ejecutar cada una de las tareas asignadas.

Se ha convenido realizarlo así, dado que no existe una necesidad de tener una respuesta inmediata a las variaciones de los sensores, y no existen fluctuaciones tan grandes en los indicadores como para requerir una lectura cada menos tiempo. En todo caso se podría aumentar el tiempo entre lectura y lectura.

Por otra parte, dado que los riegos se realizan en espacios como poco de un minuto, la temporización de la comprobación del programa de riego se ha estimado conveniente fijarla en espacios de un minuto. Dicha función comprueba tanto si ha de iniciar el riego como si ha de parar.

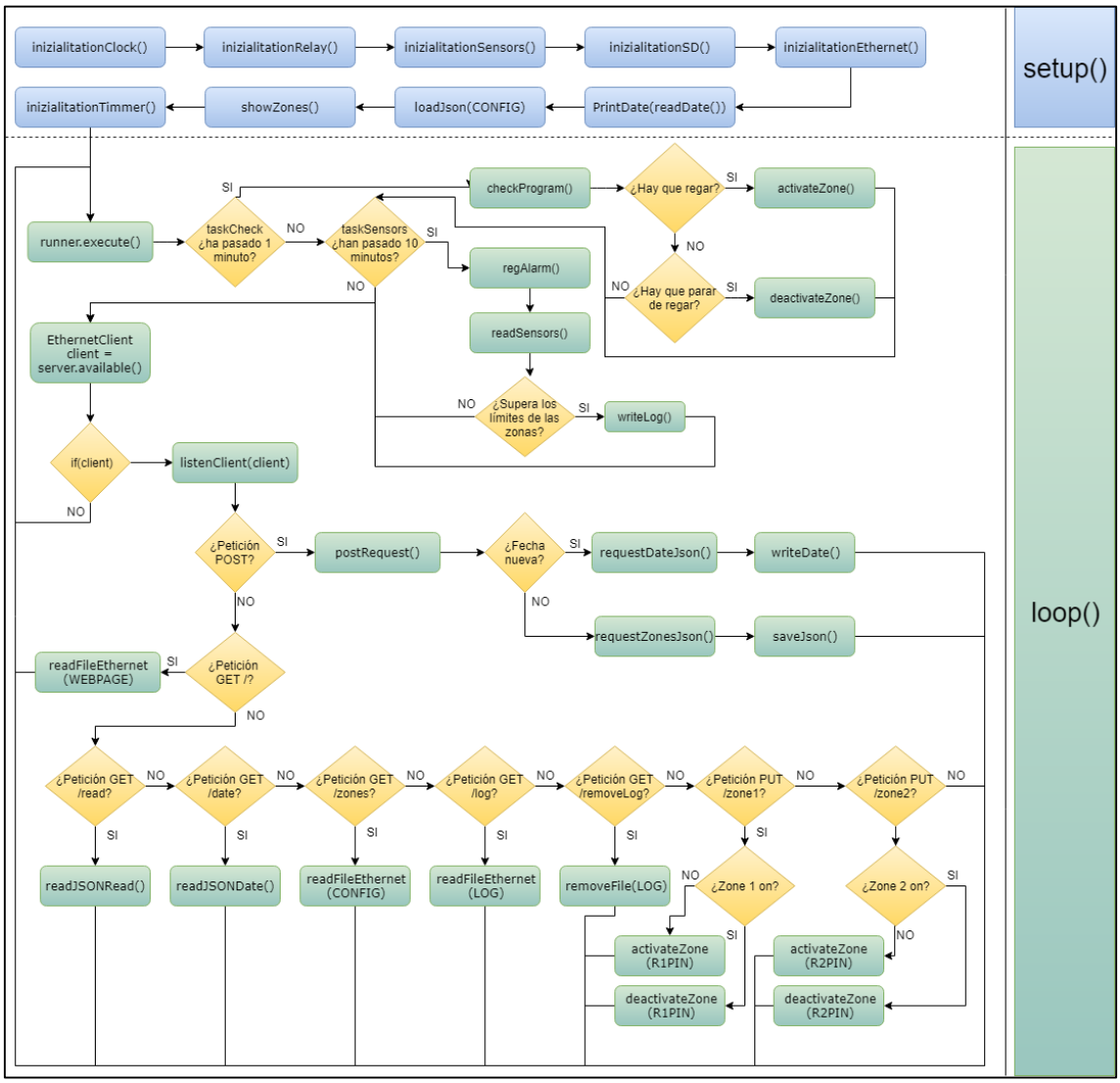


Ilustración 19

Dentro del bucle *loop*, una vez comprobada la necesidad de ejecutar o no las tareas asignadas nos encontramos con la función que recoge las peticiones de conexión al servidor. En caso de recibir una petición, se le dará respuesta dentro de la función *listenClient*, la cual determinará si se trata de una función POST, derivándola a la función *postRequest* donde se realiza una limpieza de la cadena recibida antes de su tratamiento como datos a almacenar.

Se ha diseñado diferentes respuestas para el resto de peticiones que lleguen al servidor, las cuales servirán para cubrir todas las necesidades de nuestro prototipo:

- Web de configuración de la hora y programación de las zonas (*readFileEthernet(WEBPAGE)*).
- Lectura de sensores (*readJSONRead*).

- Lectura de datos de configuración de fecha (*readJSONDate*).
- Lectura de datos de configuración de zonas (*readFileEthernet(CONFIG)*).
- Leer el fichero de registro de alarmas (*readFileEthernet(LOG)*).
- Eliminar el fichero de registro de alarmas (*removeFile(LOG)*).

La gestión de activar y desactivar las zonas directamente desde la web no se ha gestionado a parte debido a que no entrañaba complejidad, por lo que se gestionan las llamadas a las diferentes funciones directamente en la propia función *listenClient*.

Tras la breve explicación de la estructura del programa, pasaré a comentar las partes del código más importantes de los diferentes módulos.

6.2.4.1. Módulo inicializador

Se trata del módulo encargado de realizar todos los *setups* de los diferentes componentes. También se encarga de definir las estructuras de datos que serán utilizadas a posteriori, y de recuperar la configuración almacenada en el fichero de configuración y registrarla en la estructura de datos actual.

```
void setup()
{
Serial.begin(115200);

//Inicialización de los diferentes módulos
inizialitationClock();
inizialitationRelay();
inizialitationSensors();
inizialitationSd();
inizialitationEthernet();
printDate(readDate());
loadJson(CONFIG);
showZones();
delay(3000);
inizialitationScheduler();
}
```

Código 2

Podemos observar que antes de realizar la inicialización del planificador (*inizialitationScheduler()*), se produce un *delay* de 3 segundos. Esto es debido a que el sensor de humedad y temperatura requiere de al menos 2 segundos para su correcto funcionamiento, y al inicializar el gestor de tareas se ejecutan ambas tareas planificadas por primera vez (*tasksensors, taskCheck*), por lo que la primera lectura de los sensores podría ser errónea.

También existen funciones meramente de control, como *printDate* y *showZones*; están destinadas a poder realizar un seguimiento a través del puerto serie de la configuración que se está cargando.

Ahora, entraremos en detalle de la función *inizialitationScheduler()* y *loadJson(CONFIG)*, dado que el resto poseen una configuración bastante estándar.

La función *inizialitationScheduler* posee el siguiente código:

```
void inizialitationScheduler ()
{
   runner.init();
   runner.addTask(taskCheck);
   runner.addTask(taskSensors);
   taskCheck.enable();
   taskSensors.enable();
}
```

Código 3

Como podemos observar (Código 3), el gestor de tareas se inicia y se procede a agregar dos tareas, y posteriormente se habilitan. Dichas tareas, junto con el programador, se han definido de manera global en el código mediante las siguientes líneas (Código 4):

```
Scheduler runner;
Task taskSensors(600000, TASK_FOREVER, &regAlarm);
Task taskCheck(60000, TASK_FOREVER, &checkProgram);
```

Código 4

Para la definición de las tareas es necesario asignarle cada cuanto tiempo han de ejecutarse (expresado en milisegundos), cuantas veces se ejecuta (en nuestro caso por siempre, pero podríamos pasar un valor entero si solo quisiéramos que se ejecutara un determinado número de veces), y por último la dirección de la función que se ha de ejecutar. Dichas funciones han de ser del tipo *void* y no recibir parámetros tampoco, en nuestro caso no suponía ningún problema.

A continuación (Código 5) la función *loadJson(CONFIG),* que carga los datos almacenados en el fichero de configuración en la estructura de datos actual para llevar a cabo las tareas de riego pertinente:

```
void loadJson(String filename)
{
   fichero = SD.open(filename);
   DynamicJsonBuffer jsonBuffer;

   JsonObject &root = jsonBuffer.parseObject(fichero);
   JsonArray &filezones = root["zones"];

   int i = 0;
   for (auto &zone : filezones)
   {
      char *aux = zone["name"];
      String aux2(aux);
      zones[i].name = aux2;
      JsonArray &programs = zone["program"];
      int j = 0;
      for (auto &program : programs)
      {
         zones[i].program[j].hour = program["hour"];
         zones[i].program[j].minute = program["minute"];
         j++;
      }
      zones[i].duration = zone["duration"];
      JsonObject &repetition = zone["repetition"];
      zones[i].repetition.on = repetition["on"];
      zones[i].repetition.frequency = repetition["frequency"];
      JsonObject &air = zone["air"];
      zones[i].air.max = air["max"];
```

```
            zones[i].air.min = air["min"];
            JsonObject &land = zone["land"];
            zones[i].land.max = land["max"];
            zones[i].land.min = land["min"];
            JsonObject &temp = zone["temp"];
            zones[i].temp.max = temp["max"];
            zones[i].temp.min = temp["min"];
            zones[i].on = zone["on"];
            i++;
        }

        fichero.close();
        jsonBuffer.clear();
    }
```

Código 5

Para ello utilizamos las líneas correspondientes para proceder a abrir el fichero para lectura (Código 5), y nos valemos de las estructuras de datos creadas en la parte global del código (Código 6):

```
struct zone_t
{
    String name;
    program_t program[4];
    int duration;
    struct repetition_t
    {
        bool on;
        int frequency;
    } repetition;
    sensor_t air;
    sensor_t land;
    sensor_t temp;
    bool on;
};
zone_t zones[2];
```

Código 6

Como se puede apreciar, existe una correlación directa entre el fichero de configuración mostrado anteriormente y la estructura generada en el programa. Esto nos permite realizar una traslación de los datos almacenados en el fichero *config.txt* sin problemas.

Para poder realizar la lectura del fichero utilizando la librería *ArduinoJson* se ha de definir un buffer del tipo *DinamicJsonBuffer* el cual recogerá los datos leídos del fichero directamente. Existen dos tipos de buffer dentro de la librería, *DinamicJsonBuffer* y *StaticJsonBuffer*. Según indica el propio creador de la librería se recomienda el uso del buffer dinámico, dado que en dispositivos con escasa memoria se producen reservas innecesarias. El buffer dinámico va tomando la memoria a medida que la necesita, dejando libre el resto.

En la librería existen dos objetos principalmente para poder “leer” los datos de forma estructurada; JsonObject y JsonArray. Como se puede ver en el código, se va alternando el uso de ambos tipos de objetos en función de si se trata de un array de objetos o de un objeto propiamente dicho. A modo de aclaración, en Json, se entiende que un objeto ha de tener la siguiente estructura *“{“atributo”: valor}”*, por lo tanto, si el objeto en cuestión presenta una estructura diferente dará error.

Para poder acceder al valor del objeto se ha de invocar a la variable seguida del nombre del atributo que queremos acceder entre corchetes, como si de una posición de un array se tratara. Si el valor al que estamos accediendo es un valor que podemos almacenar directamente en la estructura procedemos a ello, pero en cambio si se trata de un objeto JSON, o de un array de objetos creamos una nueva variable del tipo *JsonObject* o *JsonArray* para poder acceder a todo el contenido de dicho campo. En nuestro caso esto fue necesario para poder acceder a las programaciones configuradas, así como a los valores límites de los diferentes sensores.

Una vez concluida la función procedemos a limpiar el buffer. Este paso en teoría no es necesario, aunque nos encontramos con problemas de desbordamiento de memoria durante el desarrollo y gracias a estas limpiezas logramos solucionarlos. Durante la fase de desarrollo se probó a utilizar buffer estáticos, persistiendo el problema e incluso agravándose algunas veces. Los problemas no se presentaban en esta función sino en la de lectura de los datos desde el servidor para proceder a grabarlos en el fichero, por ello, la función de recogida de datos desde el servidor presenta ciertas modificaciones que veremos posteriormente.

6.2.4.2. Módulo RTC

Es el módulo encargado de la comunicación con el componente DS3231. Estas funciones son las de almacenar los datos de configuración relativos a la hora en el componente de precisión, o bien obtener de éste la hora actual para enviarla a la web, para utilizarla a la hora de registrar una alerta en el log, o bien para decidir si se ha de regar o no.

A continuación la función que devuelve los datos a la web en una cadena con el formato de datos JSON:

```
char *getJSONDate()
{
   DateTime dateNow = readDate();
   String date = "{\"hour\":";
   date += dateNow.hour();
   date += ",\"minute\":";
   date += dateNow.minute();
   date += ",\"day\":";
   date += dateNow.day();
   date += ",\"month\":";
   date += dateNow.month();
   date += ",\"year\":";
   date += dateNow.year();
   date += "}";

   char aux[70];
   date.toCharArray(aux, 70);
   return aux;
}
```

Código 7

Como se puede apreciar en el código, la función tras recibir los datos en forma de variable *DateTime*, accede a cada uno de los parámetros que nos interesa con un método específico que tiene dicha clase. Monta una cadena con la estructura JSON y la devuelve para que sea devuelta a la web directamente.

6.2.4.3. Módulo SD

Las funciones propias del módulo SD se han incorporado dentro de otras funciones, como la mencionada anteriormente (Código 5). También en la de almacenamiento de las alertas en el fichero log, y borrado del mismo desde el servidor web. Para la lectura de la web, así como respuesta a la solicitud de los datos de configuración para mostrarlos vía web, o a la visualización del log de alertas también se ha utilizado código correspondiente a este módulo.

Las funciones son bastantes estándares, por lo que únicamente analizaremos dos funciones a modo de muestra.

Lectura de un fichero para servirlo vía web:

```
void readFileEthernet(String filename, EthernetClient *client)
{
   fichero = SD.open(filename);
   if (fichero)
   {
      while (fichero.available())
      {
         client->write(fichero.read());
      }
      fichero.close();
   }
   else
   {
      char *error = "ERROR abriendo fichero ";
      Serial.print(error);
      Serial.println(filename);
      client->write(error);
   }
}
```

Código 8

En este caso (Código 8), a la función se le pasa el nombre del fichero que ha de mostrar y el puntero al cliente *Ethernet* al que le ha de servir los datos leídos. Se procede a la escritura directamente en el la función de envío al cliente del buffer de lectura del fichero en cuestión. En caso de no encontrarse el fichero, o no poder acceder a él la función también devuelve un error que será mostrado por pantalla y a través del puerto serie.

Escritura del fichero log:

```
void writeLog(String filename, String alarm) //SD: Escritura de fichero de registro
{
   fichero = SD.open(filename, FILE_WRITE);
   if (fichero)    {
      fichero.println(alarm);
      fichero.close();
   }   else    {
      Serial.println("ERROR abriendo " + filename);
   }
}
```

Código 9

En ese caso (Código 9) la función recibe como parámetro el nombre del fichero y además la cadena de la alarma (creada en la función *regAlarm*) para escribirla directamente en el fichero LOG. En caso de no existir se crearía, y en caso de existir se agrega la línea al final del fichero.

6.2.4.4. Módulo Servidor

En el desarrollo de este módulo se ha incluido el desarrollo de la página web a través de la cual se podrá ver y modificar la configuración del programador. Para el desarrollo de la web se ha utilizado HTML y JavaScript. Se ha respetado el diseño original, y por lo tanto se ha incluido una parte para modificar las zonas programables, otra para la hora del reloj; y por último una zona donde aparecen representas las lecturas de humedad y temperatura ambientales.

Además, se han agregado dos botones para poder visualizar los datos del log, y proceder a borrarlo, además de otros dos botones para poder activar y desactivar las zonas de riego manualmente, para poder facilitar así los posibles trabajos de mantenimiento que se deban de hacer en los riegos y no tener que modificar la programación para ello. A continuación una captura de la web final:

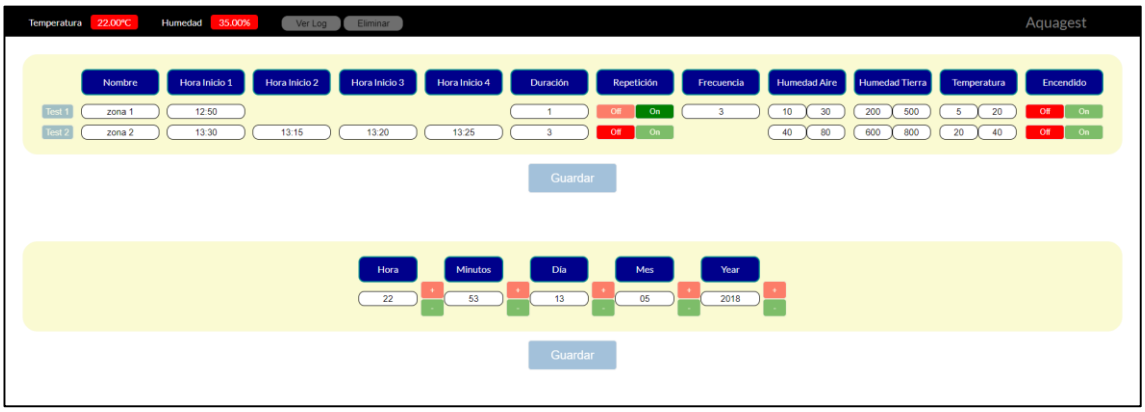


Ilustración 20

Analizaremos la parte de código correspondiente a la web (Código 10) para "traer" a la web los datos relativos a las medidas, fecha y programación, dado que aunque no es el objeto de este proyecto si resulta interesante averiguar cómo se toman dichos valores haciendo una web dinámica en lugar de servir una web estática que es lo que tradicionalmente se ha hecho en los diferentes ejemplos de web servidas por arduino. Se mostrarán las líneas de código correspondiente a la lectura de sensores y a su posterior "bindeo" en la web:

```
<script>
…
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function () {
        if (this.readyState == 4 && this.status == 200) {
            this.read = xhttp.responseText;
            _writeRead(this.read);
        }
    };
    xhttp.open("GET", "/read", true);
    xhttp.send();
    function _writeRead(str) {
        var read = JSON.parse(str);
        document.getElementById('read.temp').innerHTML = read.temp;
        document.getElementById('read.hum').innerHTML = read.hum;
    }
…
</script>
```

Código 10

Tal y como se observa, tras la carga de la estructura de la web (en blanco) se procede a lanzar una petición “GET” al servidor, en concreto a la dirección “/read”, esto provocará que el servidor al recibir dicha petición ejecute la función *getJSONRead()*, la cual devuelve una cadena formada tras la lectura de los sensores (*readSensors()*):

```
char *getJSONRead()
{
    readSensors();
    String r = "{\"temp\":\"";
    r += read.temp;
    r += "ºC\",\"hum\":\"";
    r += read.hum;
    r += "%\"}";

    char aux[40];
    r.toCharArray(aux, 40);
    return aux;
}
```

Código 11

Tras recoger la respuesta desde la web se procede a llamar a la función javaScript *_writeRead*, la cual escribirá los datos recogidos en su etiqueta web correspondiente (Código 10).

Similares a estas dos funciones en JavaScript existen otras correspondientes a la toma de datos de la hora actual y a la configuración de las zonas, las cuales emiten sendas peticiones al servidor a las direcciones “/Date” y “/Zones” respectivamente. Dichas peticiones, tal y como hemos visto en el diagrama de flujo, serán respondidas mediante llamadas a las funciones *getJSONDate()* y *readFileEthernet(CONFIG, client)*, esta última en lugar de servir el fichero web servirá el contenido del fichero de configuración almacenado en la SD de forma íntegra, dado que ya tiene estructura de datos JSON.

En la parte del servidor, ya hemos analizado una función que devuelve cualquier fichero almacenado en la SD (Código 8) y otra que devuelve la fecha almacenada directamente en el componente DS3231 (Código 7). Por ello, analizaremos la gestión que se realiza de las peticiones POST (*postRequest*) (Código 12) y el código de la función (*requestZonesJson*) (Código 13).

Función *postRequest*:

```
void postRequest(String *req_str)
{
    String data = req_str->substring(6, req_str->indexOf("HTTP"));
    data.replace("%7B", "{");
    data.replace("%22", "\"");
    data.replace("%7D", "}");
    data.replace("%20", " ");
    if (data.indexOf("year") >= 0)
    {
        requestDateJson(data);
    }
    else
    {
        requestZonesJson(data);
    }
}
```

Código 12

Tal y como vemos en la función *postRequest* (Código 12) se procede a limpiar la cadena recibida desde el cliente web con los datos que se han de almacenar tanto en el reloj (DS3231) como en el fichero de configuración de las zonas de riego. Posteriormente se llama a la función específica para almacenar dicha información. A continuación el código correspondiente a la configuración de las zonas:

```
void requestZonesJson(String data)
{
   Serial.println("** REQUEST ZONES **");

   DynamicJsonBuffer jsonBuffer;
   JsonObject &root = jsonBuffer.parseObject(data);

   int i = 0;
   String a = "";
   switch (data[2])
   {
   case '1':
      i = 0;
      a = "1";
      break;
   case '2':
      i = 1;
      a = "2";
      break;
   }

   JsonObject &zone = root[a];
   char *aux = zone["name"];
   String aux2(aux);
   zones[i].name = aux2;
   JsonArray &programs = zone["program"];
   int j = 0;
   for (auto &program : programs)
   {
      zones[i].program[j].hour = program["hour"];
      zones[i].program[j].minute = program["minute"];
      j++;
   }
   zones[i].duration = zone["duration"];
   JsonObject &repetition = zone["repetition"];
   zones[i].repetition.on = repetition["on"];
   zones[i].repetition.frequency = repetition["frequency"];
   JsonObject &air = zone["air"];
   zones[i].air.max = air["max"];
   zones[i].air.min = air["min"];
   JsonObject &land = zone["land"];
   zones[i].land.max = land["max"];
   zones[i].land.min = land["min"];
   JsonObject &temp = zone["temp"];
   zones[i].temp.max = temp["max"];
   zones[i].temp.min = temp["min"];
   zones[i].on = zone["on"];

   jsonBuffer.clear();

   saveJson(CONFIG);

   if (i == 1)
      software_Reset();
}
```

Código 13

Como se ha mencionado anteriormente, se producían desbordamientos de memoria al recibir los datos de configuración de la zona. Es por ello que finalmente se ha optado por una solución más creativa, generándose dos peticiones del tipo POST que contienen la información de cada una de las zonas.

En ambas peticiones se almacena la información en la estructura de datos del programa (*zones*) y se procede a salvarla en el fichero de configuración. Para hacer efectiva la configuración se procede a resetear el arduino mediante la función *software_Reset()* (Código 13). Esto es debido a que entendemos que tras la nueva configuración para las zonas es conveniente "resetear" las posibles alertas que quedan almacenadas en el software, dado que ya no corresponden con la configuración actualmente vigente.

6.2.4.5. Módulo Sensores

Este módulo se compone de varias funciones: la inicialización de los sensores, otra que realiza la lectura de los datos y lo almacena en la estructura de datos, otra para el registro de alarma en caso de ser necesario al superar los límites programados encargada de cancelar el siguiente riego que se vaya a realizar o de aumentar al doble el tiempo programado según sea la alerta por superar el límite máximo o mínimo.

Analizaremos la función de toma de datos:

```
void readSensors()
{
   read.hum = dht.readHumidity();
   read.temp = dht.readTemperature();
   read.humZ1 = readLand(Z1PIN);
   read.humZ2 = readLand(Z2PIN);
   Serial.println("** READ SENSORS **");
}
```

Código 14

En este código (Código 14) se ejecutan llamadas a tres tipos de funciones diferente, dos de ellas correspondientes al sensor de temperatura y humedad DHT definido en la parte global del código. Dicho sensor requiere de inicialización, mientras que para los sensores que mide la humedad de la tierra no. La lectura se almacena directamente en la estructura de datos definida como variable global del software.

Esta es la misma función que es llamada cuando se solicitan los datos de lectura desde la web al cargarse, y la misma que se invoca desde la función *regAlerta()* que analizaremos parcialmente ahora (Código 15), de la cual analizaremos una de las alarmas únicamente.

```
void regAlarm()
{
    readSensors();
    for (int i = 0; i < 2; i++)
    {
        if (zones[i].on)
        {
            if (read.temp >= zones[i].temp.max)
            {
                String alarm = "<br>";
                alarm += getStringDate();
                alarm += ", TEMPERATURA: ";
                alarm += read.temp;
                alarm += ", ZONA: ";
                alarm += zones[i].name;
                alarm += ", Se aumenta el tiempo para el siguiente riego.";
                alarm += "</br>";
                alarms[i].increment = INCREMENT;
                writeLog(LOG, alarm);
            }
        …
        }
       …
    }
}
```

Código 15

La función (Código 15), la cual es ejecutada por el planificador cada 60 segundos, realiza una llamada a la función *readSensors* (Código 14), y procede a verificar si se han sobrepasado los límites marcados según la programación. De ser así realiza una llamada a la otra función también analizada anteriormente *writeLog* (Código 9) con el dato registrado, la zona implicada y la acción que se llevará a cabo (cancelar el riego siguiente o aumentar al doble el tiempo de riego).

6.2.4.6. Módulo Riego (Relés)

Las funciones englobadas dentro del módulo de riego son las encargadas propiamente de activar o desactivar los relés (*activateZone* y *deactivateZone*) y la encargada de realizar la comprobación de si es la hora de regar o de parar el riego *checkProgram*, la cual analizaremos a continuación:

```
void checkProgram()
{
    DateTime now = readDate();
    for (int i = 0; i < 2; i++)
    {
        if (zones[i].on) {
            if (!zones[i].repetition.on) {
                for (program_t program : zones[i].program) {
                    if (program.hour == now.hour() && program.minute == now.minute()) {
                        if (alarms[i].cancel) {
                            alarms[i].cancel = false;
                            return;
                        }
                        Serial.print("** Encenciendo ");
                        Serial.print(zones[i].name);
                        Serial.println(" **");
                        if (i == 0)
                            activateZone(R1PIN);
                        if (i == 1)
```

```
                    activateZone(R2PIN);
                }
                if (program.hour == now.hour() && ((zones[i].duration * alarms[i].increment) +
program.minute) == now.minute()) {
                    Serial.print("** Apagando ");
                    Serial.print(zones[i].name);
                    Serial.println(" **");
                    if (i == 0)
                        deactivateZone(R1PIN);
                    if (i == 1)
                        deactivateZone(R2PIN);
                    alarms[i].increment = 1;
        } } }
        else {
            if ((now.minute() - zones[i].program[0].minute) % zones[i].repetition.frequency == 0) {
                if (alarms[i].cancel) {
                    alarms[i].cancel = false;
                    return;
                }
                Serial.print("** Encenciendo ");
                Serial.print(zones[i].name);
                Serial.println(" **");
                if (i == 0)
                    activateZone(R1PIN);
                if (i == 1)
                    activateZone(R2PIN);
            }
            if ((now.minute() - zones[i].program[0].minute - (zones[i].duration * alarms[i].increment)) %
zones[i].repetition.frequency == 0) {
                Serial.print("** Apagando ");
                Serial.print(zones[i].name);
                Serial.println(" **");
                if (i == 0)
                    deactivateZone(R1PIN);
                if (i == 1)
                    deactivateZone(R2PIN);
                alarms[i].increment = 1;
} } } } }
```

Código 16

Como podemos ver, en la función se realiza primeramente una llamada a la función *readDate* la cual nos devolverá la hora actual almacenada en el componente reloj. Tras esto se realiza una diferenciación si la programación es periódica o no. En ambos casos la comparación se realizará con los datos de hora obtenidos en la lectura, la diferencia es si la comparativa se llevará a cabo con las cuatro zonas programadas, o con los minutos de la primera hora almacenada.

En ambos casos, antes de realizar el riego, en caso de estar cancelado se procederá a cambiar la variable que controla dicho evento (*alarms[i].cancel*) pasando su valor a falso, que es su valor por defecto, y se cancela el resto de la ejecución.

En el caso de que la alerta registrada sea la de aumentar el tiempo de riego al doble, se tendrá en cuenta dentro de la parte de la función que controla el apagado de la zona, dado que es el factor que multiplica a la duración (*alarms[i].increment*). Dicho valor se le devuelve su valor por defecto “1” tras realizarse el apagado de la zona.

De esta forma se controla que sea únicamente el siguiente riego el que se vea afectada por la alerta registrada. Así, si tras el riego programado (ya sea

cancelado o aumentado) se produce una nueva alarma será el siguiente riego programado el que se vea afectado.

6.2.4.7. Estructura de datos

Como hemos mencionado anteriormente, en el programan existen estructura de datos que se definen en la parte global del mismo. Una de ellas, la correspondiente a la programación de las zonas se ha visto anteriormente (Código 6), junto con la carga de valores desde el fichero de configuración JSON, por lo que procederemos a ver el resto de la estructura, para comprender mejor el funcionamiento de las funciones mencionadas:

```
struct read_t //Estructura para la lectura de sensores
{
   int humZ1;
   int humZ2;
   float hum;
   float temp;
} read;

struct date
{
   int hour;
   int minute;
   int day;
   int month;
   int year;
} date;
struct program_t
{
   int hour;
   int minute;
};
struct sensor_t
{
   int max;
   int min;
};

struct alarm_t
{
   bool cancel = false;
   int increment = 1;
};
alarm_t alarms[2];

struct test_t
{
   bool on = false;
};
test_t test[2];
```

Código 17

Como vemos, las estructuras *sensor_t* y *program_t* eran utilizadas dentro de la estructura de las zonas analizada previamente.

La estructura *read_t* es la que se utiliza para almacenar los datos de la lectura de los sensores.

La estructura *alarm_t* se emplea para almacenar las alarmas referentes a las zonas. Es la que determina si se ha de cancelar el riego (*cancel*) o aumentarlo

al doble (*increment*) el valor por defecto es *1*, y es dentro de la función *regAlarm* donde se modifica el valor por *INCREMENT*, el cual está definido al inicio de las líneas del programa como *2*.

La estructura *date_t*, se utiliza para ser el paso intermedio entre la hora configurada desde la web a la almacenada en el componente reloj del dispositivo. Existen varias funciones definidas que interactúan con el reloj que se han utilizado en las pruebas y que también utilizan dicha estructura.

Por último, la estructura *test_t*, se utiliza para registrar el estado actual de la zona (encendida o apagada) en caso de que se haya activado con el botón para la activación/desactivación manual agregado en la página web.

6.2.5. Problemas encontrados

Aunque se han ido comentando en los diferentes módulos, realizaremos un breve resumen de todos los problemas, por leves que sean, detectados durante la fase de desarrollo del software.

- Al utilizar el componente de conexión *Ethernet* no era viable utilizar los pines digitales del 10 al 13, por lo que la lectura del sensor de humedad y temperatura que durante las pruebas aisladas (único módulo conectado) se estaba realizando desde el D10 se reasignó al D9.
- La extensión de los ficheros no puede ser de longitud mayor a tres caracteres, por lo que se realizó el cambio de las mismas de *json* y *html* a *txt* y *htm* respectivamente.
- Se detectó un problema con la estructura definida para almacenar los datos de las zonas. La librería para manejar estructuras JSON (*ArduinoJson*) no trabaja con cadenas de caracteres *String*, sino que en su lugar utiliza *char**, por ello, inicialmente se realizó la definición del campo *name* como *char*.* El problema que se detectó fue que a veces, debido a los buffer de lectura y escritura de los ficheros sobrescribía la información almacenada en la estructura, por lo que se solucionó cambiando el tipo a *String* y realizando la conversión que se puede ver en el código analizado cuando se procede a almacenar los datos en la estructura de datos, ya sea desde el fichero de configuración o bien cuando se reciben datos desde la web.
- Existía un problema de desbordamiento con los buffer para manejar la estructura JSON. Tras múltiples pruebas se determinó a “limpiarlos” una vez habían dejado de ser útiles. Aunque esta solución inicialmente fue suficiente, a medida que el consumo de memoria aumentó, sobre todo al recibir los datos de configuración desde el servidor, fue necesario buscar otra solución, la cual fue enviar los datos relativos a las zonas en dos partes.
- El proyecto inicialmente estaba planteado para utilizar los *timers* de arduino para realizar interrupciones cada cierto tiempo (1 y 10 minutos) y ejecutar las funciones *checkProgram* y *regAlarm* en dichas interrupciones. Encontramos el inconveniente a la hora de realizar la programación software que la utilización de dichos *timers* presentaba incompatibilidades con la utilización del sensor DHT.

  Tras intento con varias librerías que supuestamente podían emplearse para subsanar el problema del PWM, y probar diferentes pines de conexión

(con soporte PWM para los diferentes *timers*), en ningún caso se logró obtener valores de lectura válidos, todos daban como resultado *NaN* (*Not a Number*). Finalmente, se optó por incorporar la librería *TaskScheduler*, el cual es un programador de tareas y nos permite hacer un *pooling* de tareas sin tener que controlar el número de ciclos que pasen con una variable y resetearla en cada llamada a la función.

7. Conclusiones

El proyecto ha presentado bastantes dificultades en el desarrollo de cada uno de los objetivos marcados, y se ha tenido que replantear el enfoque de alguno de ellos. Entre otros, la utilización de los programadores de tareas en lugar de *timer* e interrupciones. Si bien el resultado obtenido es el mismo, la temporización de los eventos, no se consigue un óptimo rendimiento del dispositivo de esta forma, dado que en la realidad se aplica un *polling*.

El conocimiento adquirido a lo largo de todo el grado de ingeniería informática lo he podido poner en práctica en este proyecto, sobre todo la asignatura de Sistemas Empotrados, en la cual se veían muchas de las técnicas empleadas en el desarrollo de este trabajo.

En conclusión, y teniendo en cuenta la relación de objetivos descrita al inicio del proyecto, se dan por conseguidos todos ellos; se ha obtenido un prototipo totalmente funcional, el cual cumple los requerimientos de sensorización, automatización y accesibilidad a la configuración que se marcaron.

7.1. Versiones futuras

Al inicio del proyecto se pensó en instalar un módulo wifi. Durante la fase de pruebas de hardware y librerías previas al desarrollo del software, se encontraron problemas con el módulo wifi del que se disponía. En concreto se trataba del módulo "*Eleckfreaks Easy Wi-Fi Shield*", el cual utiliza un controlado wifi del tipo CC3000. Dicho módulo se quedaba en la fase de inicialización y no avanzaba. Posiblemente por un problema hardware del propio componente.

Por dichos problemas, se optó por continuar el desarrollo utilizando el componente *Ethernet*, el cual, debido a las similitudes en las funciones de las librerías permitiría una adaptación futura a cualquier módulo wifi.

Posteriormente, más avanzada la fase de desarrollo, se obtuvo otro módulo wifi, esta vez del tipo ESP-12E. Si bien, el resultado obtenido con este segundo módulo fue similar; se quedaba bloqueado el dispositivo intentando realizar la inicialización del módulo wifi. Dichos problemas quedaban descartados que se debiera a un problema de incompatibilidad, dado que las pruebas se realizaban siendo el componente el único conectado a la palca arduino y con diferentes librerías, desde las oficiales de Arduino para el control de dispositivos wifi, como otras existentes desarrolladas por la comunidad.

Debido a estas dificultades encontradas, sumada al escaso tiempo para el desarrollo del proyecto, finalmente se descartó agregar la funcionalidad wifi, debiendo de estar conectado el dispositivo en todo momento por cable de red *Ethernet* para poder acceder a modificar la configuración del dispositivo.

Queda por lo tanto queda pendiente para futuros proyectos la implementación de la capacidad de conectividad wifi.

8. Bibliografía

[1] https://es.wikipedia.org/wiki/Riego

[2] https://es.wikipedia.org/wiki/Riego_por_aspersi%C3%B3n

[3] https://es.wikipedia.org/wiki/Riego_por_goteo

[4] https://es.wikipedia.org/wiki/Hidropon%C3%ADa

[5] https://www.arduino.cc/en/Guide/Introduction

[6] https://es.wikipedia.org/wiki/Arduino

[7] https://www.arduino.cc/en/Products/Compare

[8] https://aprendiendoarduino.wordpress.com/tag/w5100/

[9] https://www.arduino.cc/en/Reference/SPI

[10] https://learn.adafruit.com/dht/overview

[11] http://www.electronicoscaldas.com/sensores-de-humedad-lluvia-inundacion/461-sensor-de-humedad-en-suelo-yl-69.html

[12] https://www.geekfactory.mx/tienda/modulos-para-desarrollo/ds3231-modulo-reloj-en-tiempo-real/

[13] https://aprendiendoarduino.wordpress.com/2014/11/18/tema-6-comunicaciones-con-arduino-4/