UNIVERSITAT POLITÈCNICA DE VALÈNCIA

**ESCOLA POLITECNICA SUPERIOR DE GANDIA**

**GRADO EN ING. SIST. DE TELECOM., SONIDO E IMAGEN**


UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

ESCOLA POLITÈCNICA
SUPERIOR DE GANDIA

# "Diseño y desarrollo de un sensor de humedad de bajo coste a diferentes profundidades para agricultura de precisión."

***TRABAJO FINAL DE GRADO***

Autor/a:
**Álvaro Daniel Liébana Carrascosa**

Tutor/a:
**Jaime Lloret Mauri**

***GANDIA, 2019***

***Resumen***

Hoy en día, gran parte de nuestra dieta está compuesta por alimentos que provienen de los árboles. Las raíces son las encargadas de absorber, almacenar y transportar los recursos que obtienen del agua, y en este proyecto nos centraremos en conocer las características que las rodean a varios niveles de profundidad para así saber cómo se está gestionando el agua en el suelo y planificar de una manera más acertada la cantidad de agua necesaria para evitar el estrés hídrico y el riego excesivo. Para llevarlo a cabo, utilizaremos sensores de humedad del suelo controlados por una placa con el procesador ESP32 que programaremos para que lea correctamente estos datos, los interprete, y muestre de la forma más clara posible la información de los distintos niveles mencionados. Otro de los fines de este proyecto es que sea accesible para todos, para lo cual es imprescindible que sea económico y que se ha conseguido gracias a que sus componentes tienen un bajo coste. Además, se ha diseñado un sistema de carga solar para que el dispositivo funcione de manera limpia y también así alargar la duración de la batería. La placa incorpora conectividad inalámbrica Wifi que enviará los datos a ThingSpeak.

***Palabras clave:*** Sensor de humedad, esp32, bajo coste, agricultura de precisión, ThingSpeak.

***Abstract***

Today, much of our diet consists of food that comes from trees. The roots are responsible for absorbing, storing and transporting resources obtained from water, and in this project we have focused on knowing the characteristics of moisture that surround them at various levels of depth to know how water in the soil is being managed, and be able to plan more accurately the amount of water needed to avoid stress and excessive watering. To carry it out, we will use soil moisture sensors controlled by an ESP32 board, which we will have programmed to read this data correctly, and interpret and display it as clearly as possible on a screen so that any user can check the information of the different levels mentioned. Another aim of this project is that it should be accessible to all, for which it is essential that it is economical and that it has been achieved thanks to the low cost of its components. In addition, a solar charging system for the device has been designed to work cleanly and to extend the battery life. The used ESP32 board incorporates Wi-Fi that will send the data to ThingSpeak.

***Keywords:*** Humidity sensor, esp32, low-cost, precision agriculture, ThingSpeak.

***Agradecimientos***

No ha sido fácil llegar a este punto, desde pequeño soñaba con ser ingeniero de telecomunicaciones y estoy feliz de haber podido cumplir este deseo. Al final, todo esfuerzo siempre tiene su recompensa.

En primer lugar, le quiero agradecer todo el apoyo a mi madre y a mi familia en general, ya que me han animado en los momentos que más lo necesitaba y sin ellos hubiera sido más difícil llegar hasta aquí. A mis amigos, que siempre han creído en mí y han entendido todas las veces que no he podido estar con ellos pues tenía que estudiar y hacer trabajos. También quiero agradecer a todos los profesores que nos han acompañado durante estos años y que nos han tratado de una forma tan cercana.

Por último, me gustaría hacer una mención especial hacia mi tutor Jaime Lloret del que he aprendido mucho estando a su lado y que me ha ayudado en todo momento, así como la oportunidad que me ha dado de estar en su laboratorio de investigación en el que me han hecho sentir como uno más de la familia.

Muchas gracias a todos por ayudarme a hacer esto posible.

# Índice

# Índice de Figuras

# Índice de tablas

# 1. Introducción

Un factor importante en nuestra dieta es la ingesta de alimentos de origen vegetal. Tanto las verduras, como las hortalizas y las frutas son un complemento que nos ayuda a cubrir la necesidad de proteínas y nutrientes que demanda nuestro cuerpo, por tanto, son parte esencial a la hora de formar un plan saludable y equilibrado de alimentación. Conseguir estos alimentos base, forma parte de un largo proceso que debe ir evolucionando al son de nuestro planeta.

La agricultura es el arte de cultivar la tierra, una ciencia que se irá propagando por lo siglos y siempre será una fuente de sustento para la humanidad [1]. Organismos como la Organización de las Naciones Unidas para la Alimentación y la Agricultura o FAO (por sus siglas en inglés: Food and Agriculture Organization) proporcionan estadísticas que nos informan de que aproximadamente el 42% de la población mundial depende de la agricultura, caza, pesca o silvicultura para su subsistencia. En relación con el terreno comercial, estadísticamente la agricultura se considera únicamente como una actividad económica, pero una de las contribuciones más importantes no monetarias de la agricultura es el hábitat, el paisaje, la conservación del suelo y la biodiversidad, etc.

Otro punto también muy significativo, es que la agricultura es un medio para solventar el hambre, ya que en muchas zonas solo cuentan con un acceso a la comida el cual pasa por producir ellos mismos los alimentos que van a consumir, o como único sustento familiar poniendo a la venta dichos alimentos.

El cambio climático es un factor que influye en la agricultura. Los efectos producidos por el calentamiento atmosférico afectan a la época de crecimiento en diversas zonas europeas los cuales alteran el estado de las cosechas. Estos cambios se irán incrementando y debemos adaptarnos a ellos modificando las técnicas y el cuidado de nuestros cultivos.

El aumento de la población mundial también condiciona en gran medida a la agricultura, pues según estimaciones de la Organización de las Naciones Unidas (ONU), en el año 2030 el número de personas habrá aumentado en 1000 millones; este aumento conlleva un incremento del 50% en la demanda de alimentos y un 40% de la cantidad de agua necesaria para producirlos.

Para afrontar este tipo de cambios que se producirán en nuestro planeta, es necesario el uso de sistemas inteligentes que hagan lo más eficaz posible el consumo de los recursos naturales como el agua, la tierra y la energía [2]. De esta forma, una vez más la tecnología se convierte en un factor clave para el desarrollo del planeta como ya

se ha visto en este y otros ámbitos, que han servido de motivación para la realización de este proyecto [3 y 4].

El ingenio humano y una visión adecuada pueden, sin duda, acelerar el progreso en la seguridad alimentaria para todos.

## 1.1. Objetivos

Lograr abastecer a una creciente población mundial no es tarea fácil, pues la necesidad de espacio y recursos naturales hace que se deban buscar alternativas para hacerlo de forma sostenible. Para ello, los agricultores deben reducir los costes de producción y controlar, en la medida de lo posible, el uso del agua y la energía en el proceso de producción de los alimentos [5].

En primer lugar, hablaremos de la gestión del agua. En este proyecto se pretende realizar una monitorización a diferentes profundidades para conocer los valores de humedad que rodea a un árbol. Esto es esencial para conocer cómo se está filtrando el agua a través del suelo y si las raíces necesitan ser hidratadas o por el contrario están en buen estado de humedad. Esta monitorización de estado permite filtrar el riego generalizado y hacerlo en función de las necesidades independientes de cada árbol del cultivo [6].

Cuando hablamos de un nuevo producto que intenta reducir los costes actuales, es imprescindible darle un enfoque que apueste por las energías verdes. Por ello se ha pensado añadir un sistema solar que ayude a reducir el consumo energético.

El aumento de la productividad lleva implícitos costes adicionales, esto nos lleva al segundo gran objetivo: aumentar la reducción de los costes de producción que precisa el sector agrícola. Además, es importante que los equipos utilizados estén correctamente optimizados para tener un consumo eficiente y no desperdiciar recursos de energía excesivos.

## 1.2. Metodología

Una vez se han marcado los objetivos del proyecto, hay que decidir qué procedimiento se va a llevar a cabo para cumplirlos.

Cuando hablamos de monitorizar a varios niveles de profundidad, sabemos que vamos a necesitar tantos sensores como zonas se quieran conocer su estado. Estos

sensores de humedad estarán conectados a la salida de 3.3V de la placa de desarrollo y enviarán su señal a los pines analógicos seleccionados de la placa, que recogerán la información obtenida para ser visualizada por el usuario final.

Abordar la necesidad de reducir los costes de producción es una tarea que requiere analizar varios aspectos. Para lograr este objetivo, por una parte, deberemos escoger materiales que tengan la mejor calidad a un precio menor del que se suele emplear por este tipo de proyectos. Por ejemplo, en lugar de utilizar una placa Arduino UNO o Due, utilizaremos la placa de desarrollo DOIT DevKit v1 ESP32 que presenta mejores características a un precio y tamaño bastante inferior.

Por otro lado, esta reducción no solo pasa por utilizar componentes de bajo presupuesto, sino que el propio hecho de monitorizar a diferentes profundidades favorece a un ahorro del agua utilizada para regar estos cultivos pues con estas medidas podemos saber la cantidad de agua que deberemos utilizar en función de sus necesidades individuales. Además, este proyecto puede ser combinado con un sistema de riego controlado muy preciso pues este se activaría cuando las condiciones que rodean el enraizado de la planta estuvieran por debajo del nivel marcado por código y esto conllevaría un mayor ahorro en el uso del agua.

Para cubrir el deseo de utilizar una fuente de energía limpia se pretende hacer uso de un pequeño panel solar, que también implica una reducción de costes a largo plazo, pues prolonga la necesidad de un remplazo en la batería que alimenta la placa y los sensores.

Si queremos utilizar un sistema que tenga un consumo óptimo, es necesario conocer a fondo las características que nos proporciona los componentes que estamos usando, por ello se emplearán técnicas de reposo gracias a los diferentes estados de funcionamiento que posee la placa de control y así poder realizar las medidas cada un intervalo de tiempo concreto y determinado por el consumidor del producto final y cuando no se esté midiendo, reducir el consumo al mínimo posible.

## 1.3. Estructura

Esta memoria tiene una estructura similar a la que se ha utilizado en la realización del proyecto. En primer lugar, el segundo apartado introducirá cuáles serán los materiales con los que trabajaremos explicando su funcionamiento y el motivo por el que han sido seleccionados, y se sugerirán también soluciones alternativas a la empleada finalmente. A continuación, en el tercer punto se realizará el montaje completo

de la circuitería del sistema con todos los componentes presentados. Una vez montado, se mostrará el diseño del encapsulado en el apartado cuarto.

Tras haber hablado del hardware del equipo, presentaremos su software correspondiente en el quinto apartado analizando el código utilizado para recibir y gestionar los datos obtenidos por los sensores para que todo funcione correctamente.

Una parte importante es conocer cómo se visualizarán los datos, por ello se explicará que es y cómo funciona la herramienta destinada a dicho fin en el sexto punto, ThingSpeak. Además, en el séptimo apartado se hará un recuento total del consumo del sistema y se realizará una predicción en base a dicho consumo del tiempo que estará el sistema en funcionamiento ininterrumpido.

Por último, se expondrán las conclusiones en el octavo apartado, así como los problemas encontrados durante todo el proceso indicando las soluciones que se han llevado a cabo, y las futuras líneas de trabajo que se esperan incorporar para mejorar el producto final.

# 2. Material de trabajo

Como decíamos, una de las principales metas de este proyecto es conseguir que el consumo energético se reduzca lo máximo posible, y por ello se han elegido componentes de bajo consumo y se ha diseñado un módulo de carga solar que se explicará más adelante. Además, se utilizarán las funciones de la placa para entrar en estado de suspensión, activarse cada cierto tiempo para realizar las medidas y volver al estado de suspensión, lo que conllevará un ahorro considerable.

Otro factor importante será el coste final del dispositivo, el cual también será reducido ya que los componentes utilizados son muy conocidos y hay diversas páginas web que los venden en grandes cantidades y, por tanto, a un precio más ajustado por cada sensor.

Mas adelante, se hablará de la construcción del encapsulado que recogerá todo el montaje de los sensores, así como del coste total de fabricación.

## 2.1. DOIT ESP32 DevKit v1

En primer lugar, tenemos la placa que controlará todo el sistema. Está basada en el microcontrolador ESP32, sucesor del ESP8266, y cuenta con capacidad Bluetooth y Wi-Fi con los estándares 802.11 b/g/n [7] certificados por la Wifi Alliance [8] y una velocidad de hasta 150 Mbps. Hay dos versiones, con 30/36 pines; nosotros utilizaremos la de 30 que además viene con dos pines GND.

*Figura 1: Placa DOIT ESP32 DevKit v1*

Como muestra la Figura 1, la placa cuenta con dos botones, EN (para reiniciar el dispositivo) y BOOT (para activar el modo programación y cargar el programa de Arduino en la placa).

Se puede apreciar que es un dispositivo con un tamaño reducido y una gran cantidad de funciones que iremos utilizando a lo largo del proceso y que lo hacen perfecto para este tipo de proyectos.

Algunas de sus especificaciones más importantes se recogen en la Tabla 1.

| **Especificaciones – DOIT ESP32 DevKit v1** | |
|---|---|
| N.º núcleos | 2 (Dual core) |
| Wi-Fi | 2.4 GHz hasta 150 Mbit/s |
| Bluetooth | BLE (Bluetooth Low Energy) |
| Arquitectura | 32 bits |
| Frecuencia de reloj | 160 MHz (hasta 240 MHz) |
| RAM | 512 KB |
| ROM | 448 KB |
| Pines | 30 |
| Consumo de corriente | 80 mA (promedio) – 225 mA |
| Bajo consumo | 150 µA |
| Consumo Deep Sleep | 2.5 µA |
| Temperatura | -40ºC a 125ºC |
| Dimensiones | 51 x 28 x 10 mm |

*Tabla 1: Especificaciones DOIT ESP32 DevKit v1*

Para conectar los sensores que describiremos en adelante, será necesario conocer el esquema de pines o Pinout que posee nuestra placa y que podemos ver en la Figura 2. En él se muestran las propiedades de cada pin de la placa para identificar dónde conectar correctamente cada módulo.

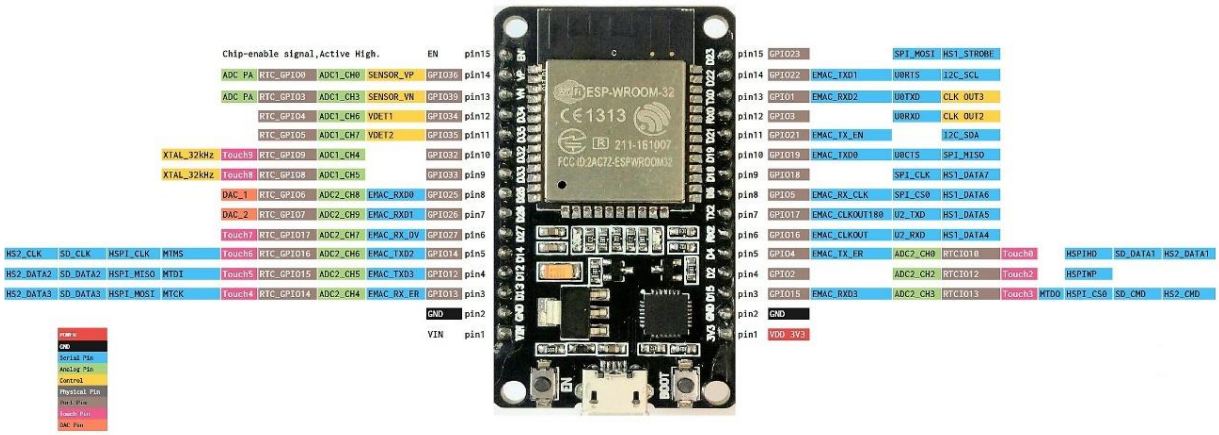


*Figura 2: Pinout DOIT ESP32 DevKit v1*

Para la alimentación del dispositivo tenemos diferentes opciones:

- Método tradicional mediante la entrada micro USB.

- Utilizando el pin Vin, al cual podemos entregarle una tensión de entre 5 (se recomienda 7) y 12 V. Este voltaje pasa a través del regulador integrado AMS1117 que lo reduce a ≈ 3.3 V. Pasar por este regulador supone una pequeña perdida en la eficiencia.

- Alimentar directamente el pin de 3.3 V. Esto se daría en caso de no disponer de USB o de la tensión necesaria para utilizar Vin. Es importante saber que aplicar cualquier tensión superior a 3.6 V (máxima tensión de funcionamiento) podría dañar algunos de los componentes de la placa, así como también los que tengamos conectados externamente.

## 2.2. Sensores

A continuación, se van a describir todos los componentes relacionados con las mediciones de humedad que formarán parte del dispositivo final. Se especificará detalladamente tanto su funcionamiento como la forma en la que serán conectados a la placa elegida ESP32.

### 2.2.1. Sensor de humedad YL-69

El YL-69 que podemos ver en la Figura 3 es un sensor de humedad del suelo de muy bajo coste (0.5€) y fácil de instalar.

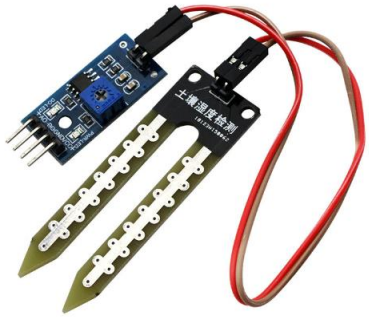

*Figura 3: Sensor YL-69 + YL-38*

El funcionamiento de este módulo consiste en dos tiras recubiertas de material conductor separadas entre sí, las cuales en presencia de humedad crean un puente entre una y otra.

Al comprar este sensor, se incluye el circuito de control YL-38 que podemos ver en la Figura 4 donde se identifican sus componentes más importantes.

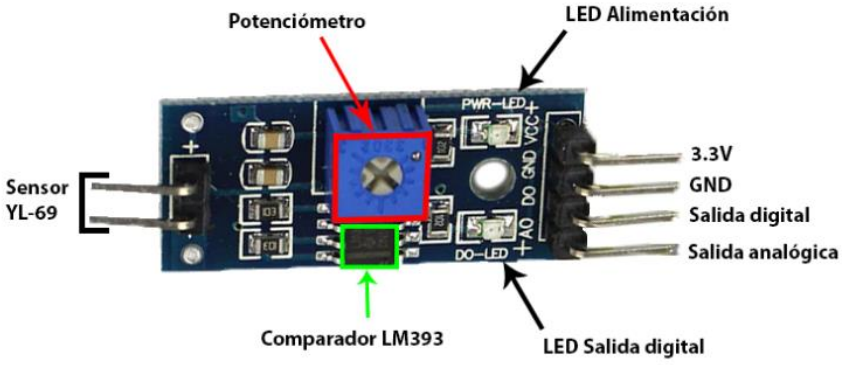


*Figura 4: Componentes YL-38*

Este circuito incorpora un amplificador operacional con función de comparador no inversor el cual convertirá la conductividad generada por el sensor en un valor analógico o digital para trabajar con él desde Arduino, y dos LED, uno de la salida digital y otro de la alimentación. En caso de usar la salida digital podremos ajustar mediante el potenciómetro la sensibilidad a partir de la que, al comparar con el valor analógico de entrada, se sobrepase el umbral seleccionado el cual encenderá el LED correspondiente y enviará un pulso de bajo nivel. Por otro lado, tenemos la salida analógica, la cual utilizaremos en este proyecto. Esta salida genera un valor de tensión que depende de

la cantidad de conductividad y Arduino lo convertirá en un valor entre 0 y 1023 (para la salida de 5V) o entre 0 y 4095 (si utilizamos la alimentación de 3.3V). En nuestro caso, como utilizamos una placa DOIT ESP32 DevKit V1 la cual únicamente tiene salida de 3.3V, nuestro rango de valores estará entre 0 y 4095, pero mediante código modificaremos dicho valor a un rango de 0-100 para facilitar la visualización de datos a cualquier usuario, donde 0 representará un estado seco mientras que 100 será el valor máximo de humedad.

Las especificaciones del sensor vienen dadas en la Tabla 2.

| Parámetro | Condiciones técnicas |
|---|---|
| Tensión de entrada | 3.3 - 5 VDC |
| Tensión de salida | 0 - 4.2 V |
| Corriente | 35 mA |
| Dimensiones YL-69 | 60 x 30 mm |
| Dimensiones YL-38 | 30 x 16 mm |

*Tabla 2: Especificaciones sensor de humedad YL-69*

Existen otros sensores similares que podrían ser utilizados sin observar apenas una variación en el resultado, como por ejemplo los modelos HL-69 y FC-28.

### 2.2.2. Módulo Tarjeta SD

Para llevar un registro de todos los datos que se obtienen de los sensores, se ha añadido un módulo para tarjetas SD/microSD y así, en caso de que fallase la función Wi-Fi de nuestro ESP32, tendríamos todas las medidas almacenadas y podríamos acceder a ellas desde cualquier dispositivo al que pueda conectarse la tarjeta.

Estos módulos precisan de una conexión SPI con la placa para que funcione correctamente y la información se escriba sin problemas. Para saber cuál son los pines que corresponden a la comunicación SPI tendremos que ir al pinout de la placa que hemos especificado anteriormente. Además, se puede alimentar tanto con 5V como con 3.3V ya que posee el chip ASM1117 que regula la tensión de entrada cuando es alimentada por 5V. Esto es porque generalmente las tarjetas SD trabajan a 3.3V y alimentarlo a dicha tensión de trabajo producirá una comunicación más limpia. La conexión de este módulo con nuestra placa viene detallada en la Figura 5.

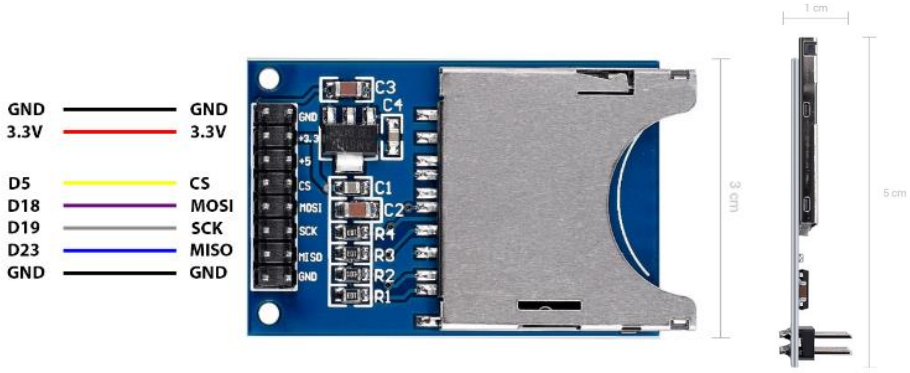


*Figura 5: Modulo SD*

Una vez tenemos todo correctamente conectado, únicamente necesitaremos la librería correspondiente SD.h que viene junto con el IDE de Arduino, con lo cual no es necesario descargar ni instalar nada. Mas adelante se detallará el código utilizado y el proceso que se lleva a cabo para escribir los datos correctamente en la SD.

Por último, cabe mencionar que el precio de este producto es de unos 0.65€, lo cual es idóneo ya que queremos que el dispositivo final sea lo más económico posible.

Las especificaciones se recogen en la Tabla 3.

| Parámetro | | Condiciones técnicas |
|---|---|---|
| Tensión de entrada | | 3.3 - 5 V |
| Consumo máx. Corriente* | Lectura | 100 mA |
| | Escritura | 100 mA |
| Dimensiones | | 50 x 30 x 10 mm |

*Tabla 3: Especificaciones módulo SD*

**Las especificaciones para el consumo de corriente son relativas al modo predeterminado de las tarjetas SD.*

## 2.3. Alimentación mediante panel solar

Como decíamos anteriormente, es importante que la batería del dispositivo dure el máximo tiempo posible para que el producto final sea totalmente independiente por largos periodos. Para ello, hemos diseñado un módulo de carga que funciona de la siguiente manera.

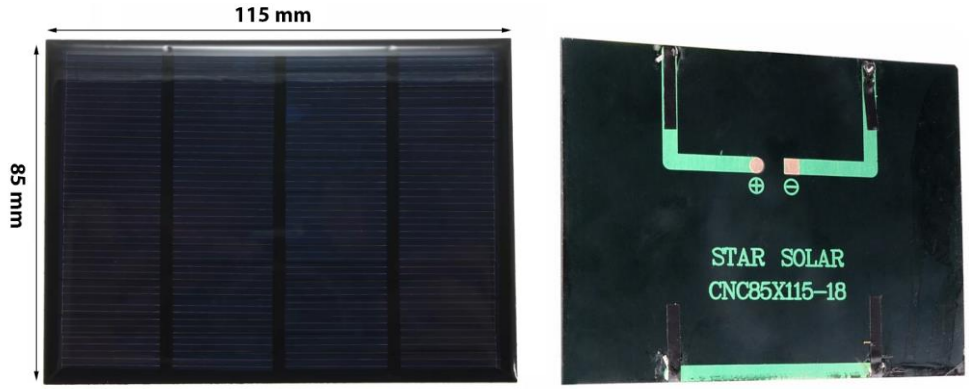


*Figura 6: Panel solar*

En primer lugar, la energía del sol la captará un panel de cuatro células como el de la Figura 6 el cual producirá un máximo 12V y 1.5W. La salida del panel se conectará al regulador LM2596 de la Figura 7 para bajar de 12 a 5V; esto es, porque el módulo de carga que vamos a utilizar necesita una entrada de entre 4.5 y 5.5V aproximadamente.

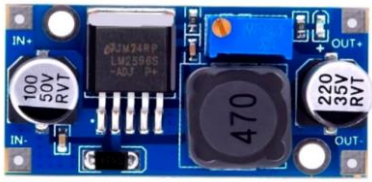


*Figura 7: Regulador Step Down LM2596*

En realidad, el panel solar no va a alimentar directamente el sistema, sino que alimentará y cargará una batería que será la que, a su vez, alimentará nuestra placa ESP32. Para ello se precisa el módulo de carga TP4056 que vemos en la Figura 8. Este módulo, que cuenta con protección, será el encargado de suministrar la energía correcta para cargar la batería y cortar el suministro cuando esta se haya cargado completamente y encenderá un LED indicativo de carga completa.

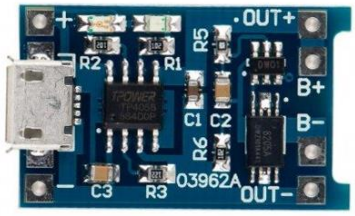


*Figura 8: Modulo de carga TP4056*

Vamos a utilizar una batería recargable de Ion-Litio Samsung modelo INR18650-30Q que se muestra en la Figura 9. El precio de esta pila es de unos 3€; es un precio elevado comparado con otros componentes, pero para acelerar el proceso de prueba

se ha utilizado este modelo que estaba disponible en el laboratorio. Tiene una capacidad de 3000 mA y una tensión con carga completa de 4.2V.


*Figura 9: Batería Samsung INR18650-30Q*

A partir de este punto tenemos tres posibles vías para finalizar la alimentación de nuestra placa:

- La primera es utilizar un Regulador Step Up MT3608 de la Figura 10 para subir la tensión de salida de la batería y suministrar energía por el pin Vin, para lo que se necesitará subir hasta entre 7 y 12V.

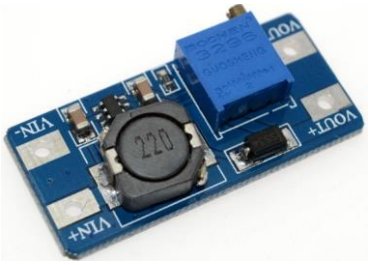


*Figura 10: Regulador Step Up MT3608*

- La segunda opción pasa por utilizar otro Regulador Step Down diferente al LM2596 (ver Figura 7) para bajar la tensión a 3.3V y alimentar directamente la placa por su pin de 3.3V.
- Como tercera y última opción tenemos el uso de un convertidor DC-DC Step Up USB que transforma una tensión de entrada de 0.9 - 5V a una salida estable de 5V a través de una toma USB, la cual proporciona una conexión sencilla. El funcionamiento de este módulo (ver Figura 11) se basa en la técnica PFM (Pulse-Frequency Modulation) que usa pulsos de duración fija y varía su tasa de repetición siendo una alternativa a la técnica PWM (Pulse-Width Modulation).


*Figura 11: Convertidor DC-DC Step Up USB*

Hemos utilizado la tercera opción por su comodidad a la hora de alimentar la placa mediante la entrada USB, aunque podríamos haber llevado a cabo cualquier de las otras opciones pues son igual de válidas.

Es muy importante que todo esté conectado correctamente y que los componentes utilizados tengan el funcionamiento que prometen. Por ello hemos comprobado mediante un multímetro digital cada parte del circuito para verificar el buen estado del sistema de alimentación. Además, para proteger el circuito ante posibles corrientes de retorno se ha añadido un diodo 1N4007 (ver Figura 12) a la salida del panel solar y previo a la conexión con el regulador LM2596.

*Figura 12: Diodo 1N4007*

Este diodo se puede encontrar en grandes lotes por un precio muy reducido. Hay que tener en cuenta, que su utilización sacrificará 1V proveniente del panel solar.

Las especificaciones de todos los componentes utilizados para alimentar la placa se pueden encontrar en la Tabla 4.

| Parámetro | Condiciones técnicas |
|---|---|
| Tensión de salida Panel | 12 V |
| Potencia Panel | 1.5 W |
| Dimensiones Panel | 115 x 85 mm |
| Vin LM2596 | 4 - 40 V |
| Vout LM2596 | 1.5 – 37 V |
| Imax LM2596 | 2 A |
| Vin TP4056 | 4.5 – 5.5 V |
| Capacidad INR18650-30Q | 3000 mA |
| Vout INR18650-30Q | 3.7 – 4.2 V |
| Vin Step Up USB | 0.9 – 5 V |
| Vout Step Up USB | 5 V |
| Imax Step Up USB | 200 – 300 mA (1 bat.) |

*Tabla 4: Especificaciones carga solar*

## 2.4. Recinto

Es necesario preservar el producto en un recipiente duradero y aislado de agentes externos que puedan afectar a su correcto funcionamiento. Por ello se ha utilizado una caja estanca como la de la Figura 13 para protegerlo y poder enterrarlo sin problemas.

*Figura 13: Recipiente estanco*

Esta caja tiene unas dimensiones de 153 x 110 x 65 mm y permite su sellado gracias a 4 tornillos de ¼ de vuelta.

## 2.5. Coste total

Uno de los factores más importantes a la hora de realizar un nuevo proyecto es el coste final de fabricación del producto, pues determinará el precio de venta al público. Como ya se ha comentado en diversas ocasiones, se ha hecho bastante hincapié para que este apartado sea uno de los grandes beneficios del proyecto. En la Tabla 5, se muestra la suma total del coste de los materiales.

| Material | Precio/Unidad (€) |
|---|---|
| DOIT ESP32 DevKit v1 | 4 |
| Sensor YL-69 + YL-38 | 0.5 |
| Modulo SD | 0.65 |
| Panel Solar | 3.5 |
| Diodo 1N4007 | 0.0079 |
| Regulador LM2596 | 0.49 |
| Regulador TP4056 | 0.25 |
| Samsung INR18650-30Q | 2.34 |
| Recinto estanco | 2 |
| Step Up USB | 0.21 |
| **Coste Total** | **13.9479** |

*Tabla 5: Coste total de los materiales*

En la Tabla 5 se han utilizado los costes de cada producto individualmente, lo cual incrementa en cierta medida su precio pues para grandes lotes este se vería reducido en aproximadamente un 35%.

Como se ha comentado anteriormente, a pesar de ser un sensor de bajo precio y debido a la raíz de su funcionamiento, puede ser enlazado con un sistema de riego generando un control y ahorro en el gasto del agua. Este ahorro provoca que el sensor sea provechoso no solo para el agricultor sino también para el medio ambiente.

# 3. Montaje de los sensores

Para mostrar el montaje de los sensores con la placa, se ha utilizado un software gratuito llamado Fritzing el cual nos permite hace diagramas de conexión de diversa índole [9]. El diseño final del montaje es el que se muestra en la Figura 14. Los elementos utilizados para este diagrama se han encontrado tanto en su página web [10] como en otras webs que integran contenido relacionado.

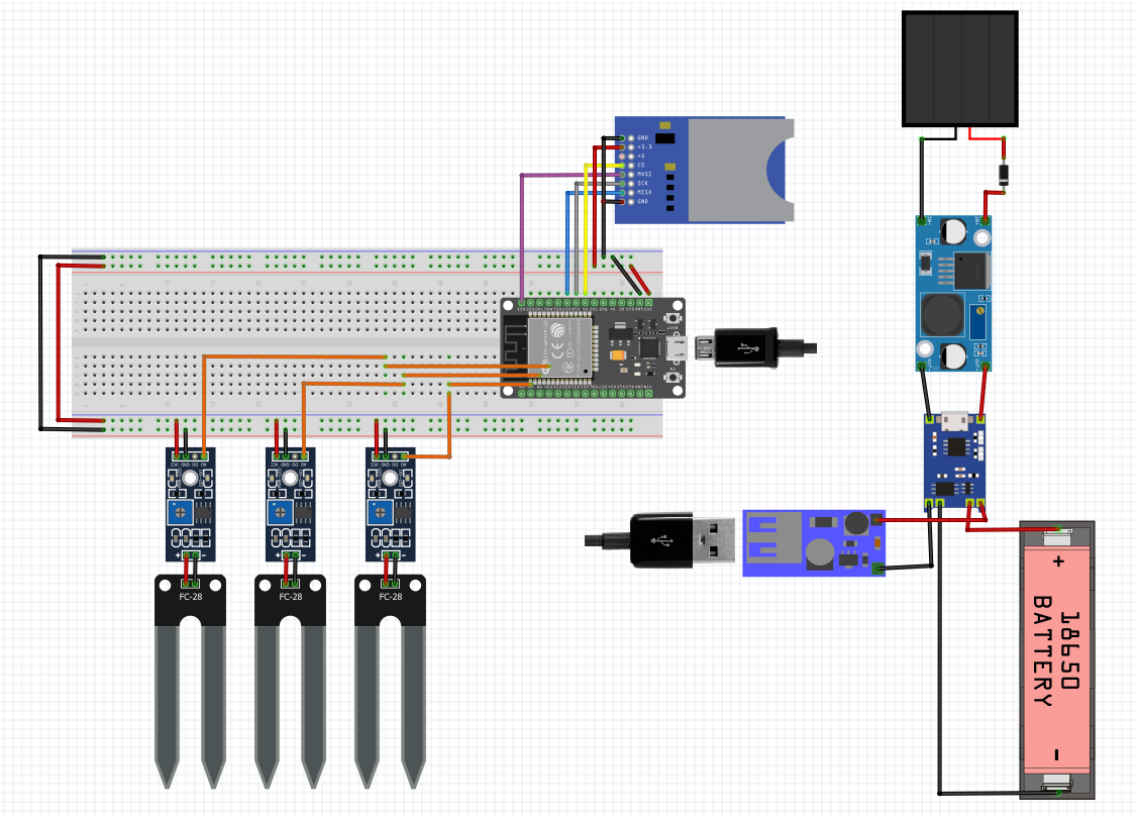


*Figura 14: Diagrama de conexión de los sensores*

La alimentación de la placa, como hemos visto, consta de un panel solar de 12V conectado a un regulador de tensión que baja el voltaje a 5V, el permitido por el módulo de carga utilizado. Este módulo de carga se conecta a la batería por los pines B+ y B- y la salida ira a un nuevo regulador para, o bien elevar la tensión a entre 7 y 12V para alimentar la placa mediante el pin Vin, bajar dicha tensión a 3.3V y alimentar el pin de 3.3V directamente o, definitivamente, la solución elegida de convertir la tensión a 5V estables por una salida USB y alimentar la placa por su entrada micro USB.

# 4. Diseño del encapsulado

Para el encapsulado del sensor hemos optado por un diseño sencillo. Debido a que esta no es la versión final, el recipiente que lo incluya estará sujeto a cambios.

Para aislarlo correctamente de los agentes externos y como hemos visto anteriormente, hemos utilizado un recipiente estanco para proteger contra líquidos, tierra u otras sustancias externas que puedan dañar la integridad de la placa que gestiona las medidas. Para la salida de los sensores, se ha realizado un agujero en la parte inferior del recinto que posteriormente ha sido sellada con silicona para mantener su estanqueidad. Estos sensores se introducirán cada uno de ellos a la profundidad deseada en función de la planta que vaya a ser analizada.

Dentro del recipiente se han distribuido el resto de los componentes separando y fijando correctamente los cables para evitar que cualquier movimiento afecte a sus conexiones con la placa. En la Figura 15 (a) podemos observar la circuitería que desde la batería alimentara nuestra placa y en la Figura 15 (b) el sistema conectado a la placa solar para suministrar energía adicional a la batería. También se puede observar la distribución del interior con todo el conjunto de sensores y cableado en la Figura 15 (c).

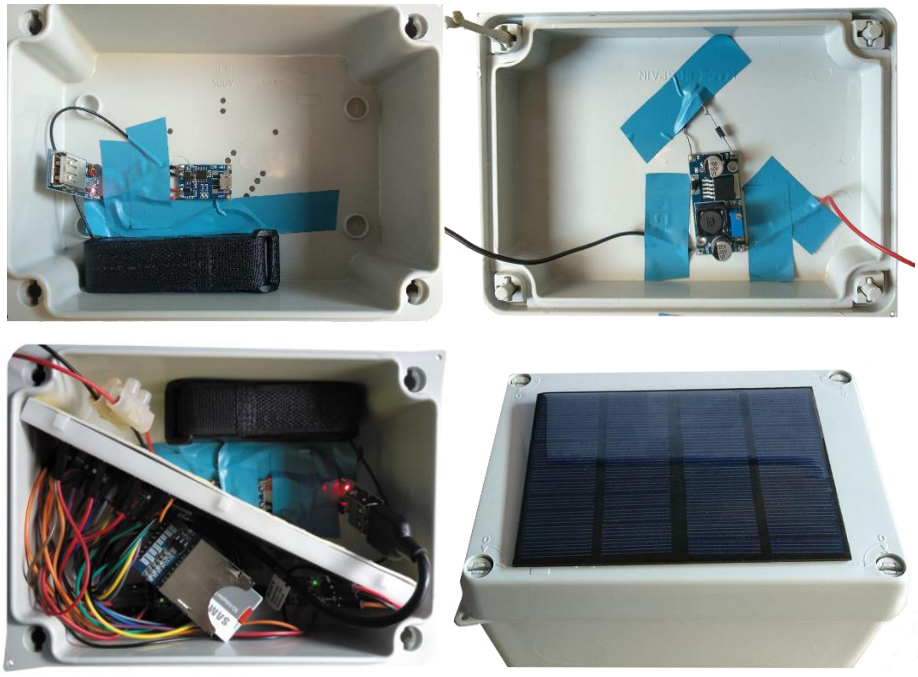

*Figura 15: Encapsulado del sensor: (a) Circuito de alimentación con la placa, (b) Circuito del panel solar, (c) Distribución del interior, (d) Dispositivo final*

Finalmente, la Figura 15 (d) muestra cómo se verá el encapsulado final con todos los componentes en su interior y sellado para su puesta en marcha.

# 5. Estructura Software

Una vez se tiene el diseño del sensor, y ya ha sido montado, comenzamos con la implementación de las funciones que este llevará a cabo. El entorno de programación utilizado es el software de Arduino mediante el cual, instalando desde el gestor de tarjetas la correspondiente al módulo ESP32, podremos trabajar con nuestra placa sin ningún problema. Arduino se programa con un lenguaje propio, aunque permite funciones de otros lenguajes como C o C++ lo cual facilita su aprendizaje y entendimiento.

```
#include <SD.h>
#include <SPI.h>
#include <sd_defines.h>
#include <sd_diskio.h>
#include "FS.h"
#include <WiFi.h>
#include <ThingSpeak.h>
#include "secrets.h"
#include <WiFiUdp.h>
#include <NTPClient.h>
//--------------------------------------------------------------------//

const int sensorPin = 36;
const int sensorPinDos = 39;
const int sensorPinTres = 34;

File logFile;

#define MOSI 23
#define MISO 19
#define SCK 18
#define CS 5

uint64_t sleep_time = 15000000;      // 15 segundos
uint64_t sleep_time_2 = 1800000000;  // 30 minutos
uint64_t sleep_time_3;               // Reposo nocturno
float t;

RTC_DATA_ATTR int bootCount = 0;

const int cs = 5;
const int mosi = 23;
const int miso = 19;
const int sck = 18;

//Parametros Wifi y Thingspeak
char ssid[] = SECRET_SSID;      // Recoge el SSID definido en secrets.h
char pass[] = SECRET_PASS;      // Recoge la contraseña definida en secrets.h

WiFiClient  client;             // Creamos el cliente

unsigned long myChannelNumber = SECRET_CH_ID; // Canal de ThingSpeak definido en secrets.h
const char * myWriteAPIKey = SECRET_WRITE_APIKEY; // Write Key de ThingSpeak definida en secrets.h

WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "2.es.pool.ntp.org", 7200, 60000);

```

En primer lugar, incluimos las librerías y declaramos todas las variables que se utilizarán más adelante. Hemos añadido el archivo secrets.h que contiene los datos del punto de acceso al que se va a conectar nuestra placa, así como el canal y otros datos relacionados con ThingSpeak a los que se enviaran los valores medidos por los sensores:

- SECRET_SSID: Nombre del punto de acceso.
- SECRET_PASS: Contraseña del punto de acceso.
- SECRET_CH_ID: Código de canal de ThingSpeak.
- SECRET_WRITE_APIKEY: Clave de escritura de ThingSpeak.

** Esta sección incluye el subapartado 5.3 que contiene el código del archivo secrets.h.*

## 5.1. Void setup ()

A continuación, iniciamos las conexiones Wi-Fi y ThingSpeak dentro del apartado void setup. Esta es la primera función en ejecutarse dentro del programa y es donde se establecen los criterios que necesitan llevarse a cabo una única vez.

```
47. void setup() {
48.   Serial.begin(9600);
49.
50.   //------------Bloque de conexion con el AP y con ThingSpeak---------------
51.
52.   WiFi.mode(WIFI_STA);
53.   WiFi.begin(ssid, pass);           // Inicio de conexion con el AP
54.
55.   Serial.println("Conectando al punto de accceso");
56.
57.   while (WiFi.status() != WL_CONNECTED) {
58.     Serial.print(".");
59.     delay(500);
60.   }
61.
62.   Serial.println();
63.   Serial.println("Conexion establecida");
64.   Serial.print("CONECTADO A: ");
65.   Serial.println(SECRET_SSID);
66.   Serial.println("Caracteristicas de la red:");
67.   Serial.print("LocalIP:"); Serial.println(WiFi.localIP());
68.   Serial.println("MAC:" + WiFi.macAddress());
69.   Serial.print("Gateway:"); Serial.println(WiFi.gatewayIP());
70.   Serial.print("AP MAC:"); Serial.println(WiFi.BSSIDstr());
71.
72.   ThingSpeak.begin(client);  // Iniciando conexion con ThingSpeak
73.
```

Además de iniciar la conexión a internet y con el servidor ThingSpeak, en esta sección también inicializaremos el módulo SD en modo escritura. Por último, antes de cerrar este bloque se iniciará el cliente que nos proporciona la fecha y hora que utilizaremos al inicio del siguiente bloque para programar los intervalos de funcionamiento del dispositivo y que posteriormente se mostrará cuando se realicen las medidas.

```
74.   //-------------------Bloque de inicio de la tarjeta SD----------------------
75.
76.   Serial.print(F("Iniciando SD ..."));
77.
78.   if (!SD.begin(cs))
79.   {
80.     Serial.println(F("Error al iniciar"));
81.     return;
82.
83.   }//cierro if
84.
85.   Serial.println(F("Iniciado correctamente"));
86.
87.   File logFile = SD.open("/datalog.txt", FILE_APPEND);
88.   logFile.println("Medidas realizadas con sensor de humedad FC-28:");
89.   logFile.println();
90.
91.   timeClient.begin();
92.
93. }//cierro void setup
94.
```

Para la conexión de la placa con el módulo SD se utiliza el bus SPI. A diferencia de otros buses, el SPI es Full-Duplex y es menos complejo que, por ejemplo, el bus I2C; además, tiene una velocidad de comunicación relativamente elevada y los dispositivos que necesitan esta conexión suelen tener un bajo precio de mercado.

Con begin () y el correspondiente pin CS (D5 en nuestra placa) estamos habilitando el integrado al que se envían los datos. Tras iniciar la comunicación, abrimos el archivo que será modificado y activamos el modo escritura mediante el comando FILE_APPEND. Este comando solo escribe a partir del último punto de escritura, por ello no hemos utilizado el comando FILE_WRITE ya que este sobrescribe lo que ya había anteriormente y en caso de fallo o reinicio de la placa eliminaría las medidas anteriores.

Una vez iniciadas las conexiones necesarias, se mostrarán por el serial los datos del punto de acceso tales como la dirección IP, MAC, etc. y la verificación de que todo ha sido conectado correctamente tras lo que se mostrarán las primeras medidas de cada sensor, como vemos en la Figura 16.

```
Conexion establecida
CONECTADO A: DESKTOP-I4PFC8A 3159
Caracteristicas de la red:
LocalIP:192.168.137.217
MAC:3C:71:BF:4C:AB:9C
Gateway:192.168.137.1
AP MAC:32:E3:7A:97:CC:28
Iniciando SD ...Iniciado correctamente

Conexion establecida con ThingSpeak
Hora de medida: 14:00:15
Humedad Sensor 1: 1023
Humedad Sensor 2: 1023
Humedad Sensor 3: 1023
```

*Figura 16: Información Monitor Serie*

Los datos de comprobación son mostrados tanto por el serial para verificar su funcionamiento en vivo, como también son escritos en la SD para que el programador pueda encontrar fallos en el dispositivo que hayan podido darse en el pasado, y las medidas con sus respectivas horas serán escritas de igual forma en un archivo de texto dentro de la tarjeta SD.

## 5.2. Void loop ()

Una vez se ha ejecutado la función de setup, entramos en la función loop. En esta función añadiremos todo lo que se repetirá de forma indefinida mientras el dispositivo este encendido.

En primer lugar, se va a definir el rango de funcionamiento del sensor. Para ahorrar el máximo de batería posible se ha utilizado el modo Deep Sleep para inducir un estado de suspensión a la placa en aquellos momentos en los que no sea necesario realizar las mediciones. El modo de ahorro utilizado se explica en profundidad en el apartado 7.1.

Gracias al servidor NTP que hemos implementado para dar constancia de las horas a las que se realizan las medidas en la tarjeta SD, podemos crear un intervalo de tiempo de funcionamiento para que el sensor únicamente este activo durante las horas que podamos necesitar los datos de humedad, por ejemplo 12 horas (6 am - 6 pm). De este modo podemos crear un periodo de sueño controlado y poner en suspensión la placa durante la noche y ahorrar gran cantidad de batería. Como se puede apreciar en el

código siguiente en primer lugar debemos actualizar la hora del servidor para que los datos extraídos de ella sean correctos para definir el intervalo del sensor. Durante este intervalo de funcionamiento se realizarán 5 medidas seguidas cada 30 minutos. El motivo de estas 5 mediciones es corroborar los datos en caso de fallo en alguna de ellas. La separación entre cada una de estas medidas será de 15 segundos pues, como veremos en el Apartado 6, ThingSpeak admite un envió de datos cada 15 segundos. Durante este tiempo de espera, también pondremos la placa en reposo y tras completarlas todas, se suspenderá la placa de nuevo durante otros 30 minutos y así sucesivamente hasta que se cumpla la hora límite marcada para que la placa inicie su reposo nocturno hasta el día siguiente. Los periodos de descanso se han definido al inicio del código mediante las variables uint64_t para poder utilizar tiempos de reposo más grandes. Además, se ha programado un sistema de despertado inteligente para que, en caso de reinicio de la placa, esta calcule el tiempo restante para iniciar el funcionamiento y envíe de nuevo la placa a un estado de suspensión durante el periodo calculado.

```
96. void loop() {
97. 
98.   timeClient.update();
99.   Serial.println(timeClient.getFullFormattedTime());
100. 
101.         if (timeClient.getHours()<6|timeClient.getHours()>=18) {
102. 
103.             if (timeClient.getHours()<6) {
104. 
105.               t=((6*60*60-timeClient.getHours()*60*60)-
    (timeClient.getMinutes()*60));
106.               sleep_time_3=t*1000000;
107. 
108.               int horas = 6-timeClient.getHours()-1;
109.               int minutos = 60-timeClient.getMinutes();
110. 
111.               esp_sleep_enable_timer_wakeup(sleep_time_3);
112.               Serial.print("Iniciando Modo Reposo Nocturno durante: " + Strin
    g (horas) + " Horas y " + String(minutos) + " Minutos");
113.               logFile.println("Iniciando Modo Reposo Nocturno durante: " + St
    ring (horas) + " Horas y " + String(minutos) + " Minutos");
114.               esp_deep_sleep_start();
115. 
116.             }
117. 
118.             else  if (timeClient.getHours()>=18) {
119. 
120.               t=((30*60*60-timeClient.getHours()*60*60)-
    (timeClient.getMinutes()*60));
121.               sleep_time_3=t*1000000;
122. 
123.               int horas = 30-timeClient.getHours()-1;
124.               int minutos = 60-timeClient.getMinutes();
125. 
126.               esp_sleep_enable_timer_wakeup(sleep_time_3);
127.               Serial.print("Iniciando Modo Reposo Nocturno durante: " + Strin
    g (horas) + " Horas y " + String(minutos) + " Minutos");
```

```
128.                logFile.println("Iniciando Modo Reposo Nocturno durante: " + St
    ring (horas) + " Horas y " + String(minutos) + " Minutos");
129.                esp_deep_sleep_start();
130.
131.                }
132.
133.          }
134.
135.          else  {
136.
137.              if (bootCount==5) {
138.                  bootCount=0;
139.                  esp_sleep_enable_timer_wakeup(sleep_time_2);
140.                  Serial.println("Iniciando Modo Reposo durante 30 Minutos");

141.                  esp_deep_sleep_start();
142.              }
143.
144.              else{
145.                  esp_sleep_enable_timer_wakeup(sleep_time);
146.              }
147.
```

Tras programar el tiempo de funcionamiento del dispositivo, se deberá comprobar que la conexión con el punto de acceso continúa operativa. Este paso es importante ya que puede haber cortes de luz que ocasionen una interrupción en el servicio de internet. De este modo, en caso de fallo el dispositivo volvería a reconectarse sin problema.

```
148.              if(WiFi.status() != WL_CONNECTED){
149.                  Serial.println();
150.                  Serial.println();
151.                  Serial.println();
152.                  Serial.print("Reconectando a: ");
153.                  Serial.println(SECRET_SSID);
154.
155.                  while(WiFi.status() != WL_CONNECTED)  {
156.                     WiFi.begin(ssid, pass);
157.                      Serial.print(".");
158.                      delay(500);
159.                  }
160.                  Serial.println("Reconectado correctamente");
161.              }
```

A continuación, recogemos el valor obtenido de los sensores el cual estará comprendido entre 0 (sensor introducido completamente en agua) y 4096 (sensor al aire libre) y los mapeamos al intervalo mencionado anteriormente de 0 a 100 para facilitar su visualización, dónde el valor mínimo es un estado de sequedad. Tras introducir los valores obtenidos de los sensores en cada variable, abrimos el archivo y escribimos. Tras ellos, seleccionamos el campo de ThingSpeak, en el que escribiremos la medida de cada sensor YL-69, mediante ThingSpeak.setField (número del campo a escribir, variable que incluye el dato).

```
162.
163.             timeClient.update();
164.
165.             int humedad = analogRead(sensorPin);
166.             humedad=map(humedad, 0, 4095, 100, 0);
167.
168.
169.             int humedadDos = analogRead(sensorPinDos);
170.             humedadDos=map(humedadDos, 0,4095, 100, 0);
171.
172.
173.             int humedadTres = analogRead(sensorPinTres);
174.             humedadTres=map(humedadTres, 0, 4095, 100, 0);
175.
176.
177.             // Abrir archivo y escribir valor
178.             logFile = SD.open("/datalog.txt", FILE_APPEND);
179.
180.             ThingSpeak.setField(1, humedad);
181.             ThingSpeak.setField(2, humedadDos);
182.             ThingSpeak.setField(3, humedadTres);
183.
184.             int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
185.
186.             if(x == 200)  {
187.                 Serial.println();
188.                 Serial.println("Conexion establecida con ThingSpeak");
189.             }
190.
191.             else  {
192.                 Serial.println("Problem updating channel. HTTP error code " +
    String(x));
193.             }
194.
```

Finalmente, con ThingSpeak.writeFields (código del canal, clave de escritura) escribiremos todos los datos del sensor en el campo especificado anteriormente.

La variable “x” guarda un código que, si es 200 significará que el dato se ha enviado correctamente a ThingSpeak y, en caso de saltar el error, se mostrará el código de dicho error para poder detectar el motivo por el que no se ha producido el envío de datos.

Una vez los datos ya se han enviado a ThingSpeak, analizaremos los valores para detectar cuando consideraremos que el suelo está húmedo y cuando no, y escribiremos los datos en la SD. El código también envía los datos por el serial para comprobar, durante la programación, como se verá en el archivo de la SD.

```
195.             Serial.print("Fecha de medida: ");
196.             Serial.println(timeClient.getFullFormattedTime());
197.             logFile.print("Hora de medida: ");
198.             logFile.println(timeClient.getFullFormattedTime());
199.
200.             if (logFile) {
201.
202.                 logFile.print("Humedad Sensor 1: ");
203.                 Serial.print("Humedad Sensor 1: ");
204.
205.                 if(humedad > 60) {
206.
```

```
                    Serial.print(humedad);
                    Serial.println("-->Suelo Humedo");
                    logFile.print(humedad);
                    logFile.println("-->Suelo Humedo");

                }//cierro if humedad

                else  {

                    Serial.println(humedad);
                    logFile.println(humedad);

                }//cierro else

```

Hemos iniciado con un if (logFile), que tiene una condición que siempre se va a cumplir mientras el módulo SD esté conectado correctamente. Como se aprecia, hemos considerado que a partir de que el sensor envíe un valor superior a 60, consideraremos que el suelo está húmedo indicando “Suelo húmedo” tras el valor medido.

Haremos lo mismo con los 2 sensores restantes:

```
                logFile.print("Humedad Sensor 2: ");
                Serial.print("Humedad Sensor 2: ");

                if(humedadDos > 60)  {

                    Serial.print(humedadDos);
                    Serial.println("-->Suelo Humedo");
                    logFile.print(humedadDos);
                    logFile.println("-->Suelo Humedo");

                }//cierro if humedadDos

                else{

                    Serial.println(humedadDos);
                    logFile.println(humedadDos);

                }//cierro else

                logFile.print("Humedad Sensor 3: ");
                Serial.print("Humedad Sensor 3: ");

                if(humedadTres > 60) {

                    Serial.print(humedadTres);
                    Serial.println("-->Suelo Humedo");
                    logFile.print(humedadTres);
                    logFile.println("-->Suelo Humedo");

                }//cierro if humedadTres

                else{

                    Serial.println(humedadTres);
                    logFile.println(humedadTres);

                }//cierro else

                logFile.close();

```

```
261.             } //cierro primer if
262.
263.             else {
264.
265.                 Serial.println("Error al abrir el archivo");
266.
267.             }//cierro else
268.
```

Tras haber escrito todos los datos en la SD, cerramos el archivo mediante el comando logFile.close(). Además, se creará un mensaje de error en caso de que haya un fallo al abrir el archivo de la SD.

Por último, puesto que hay que realizar las 5 medidas comentadas anteriormente, incrementaremos un número el valor de la cuenta y posteriormente iniciaremos el reposo de 15 segundos. Tras esto, la placa se volverá a iniciar y decidirá si debe ejecutar de nuevo el código completo o si debe volver al estado de reposo teniendo en cuenta los intervalos de funcionamiento ya definidos.

```
269.             ++bootCount;
270.             Serial.println("Medida numero: " + String(bootCount));
271.             Serial.println("Iniciando Modo Reposo durante 15 segundos");
272.             esp_deep_sleep_start();
273.
274.         }//else nocturno
275.
276.       }//cierro void loop
```

## 5.3. Secrets.h

Este archivo se ha creado para acceder fácilmente a las variables más importantes encargadas de la conexión con el punto de acceso y con el servidor ThingSpeak en caso de que sea necesario realizar algún cambio. El archivo principal accede a estos datos y los utiliza para establecer correctamente las conexiones mencionadas.

```
1.  // Datos del punto de acceso y de ThingSpeak
2.
3.  #define SECRET_SSID "DESKTOPI4PFC8A 3159"
4.  #define SECRET_PASS "4%34N99q99HP"
5.  #define SECRET_CH_ID 802891
6.  #define SECRET_WRITE_APIKEY "VCU0FVATII9R7895"
```

Como se ha comentado anteriormente, en el archivo se encuentran los datos tanto del nombre y contraseña del punto de acceso, como el canal y la clave de acceso a ThingSpeak. Este código se enlazará al inicio del archivo principal.

# 6. ThingSpeak

Como ya hemos comentado, aprovecharemos la conectividad inalámbrica de la placa para enviar los datos medidos por los sensores de humedad vía Wi-Fi a un servidor llamado ThingSpeak, utilizado en numerosos proyectos de Arduino por su versatilidad.

## 6.1. Descripción y funcionamiento

ThingSpeak se trata de una plataforma de código abierto para el Internet de las Cosas (IoT por sus siglas en inglés) que nos permite leer y almacenar datos de sensores en la nube [11]. Además, nos ofrece la posibilidad de analizar y visualizar los datos en MATLAB [12] y actuar sobre ellos ya que este servicio es propiedad de la misma empresa, Mathworks.

## 6.2. Configuración del servidor

Esta aplicación funciona con canales. Dentro de estos canales se encuentran los campos los cuales contienen los datos, ubicación y estado. Por tanto, lo primero que tenemos que hacer tras iniciar sesión en la página web [13] es crear un canal que puede ser público o privado. También se ofrece la posibilidad de guardar diferentes tipos de datos mediante el sistema de campos, permitiendo hasta 8 campos por canal.

Una vez hemos configurado el canal y los campos, necesitaremos varios datos importantes para enlazar el envío de datos de nuestra placa a este canal [14]. Para ello apuntamos el código del canal como vemos en la Figura 17 que se habrá generado automáticamente y podremos encontrar en la pestaña Channel Settings.

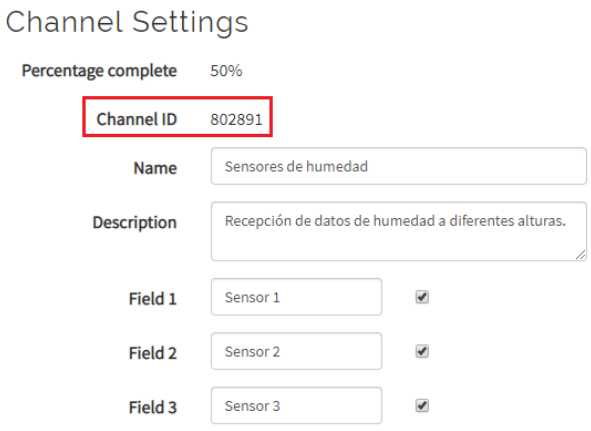


*Figura 17: Configuración del canal - ThingSpeak*

Después, vamos a la pestaña API Keys y copiamos el código Write API Key mostrado en la Figura 18, el cual es imprescindible para escribir los datos en el canal.

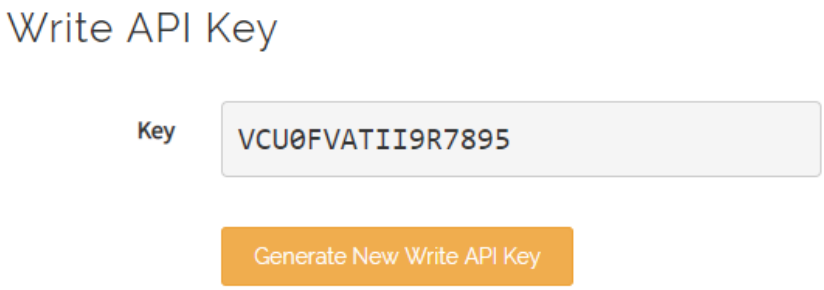


*Figura 18: Clave de escritura - ThingSpeak*

## 6.3. Visualización de los datos

Una vez esta todo configurado únicamente encendemos el sensor, y esperamos a que se conecte al punto de acceso y comience a realizar las mediciones y enviarlas a ThingSpeak.

Si lo hemos configurado y enlazado todo correctamente, en la pestaña Private View empezaremos a recibir los datos de medición en tantas graficas como sensores hayamos configurado como se observa en la Figura 19.

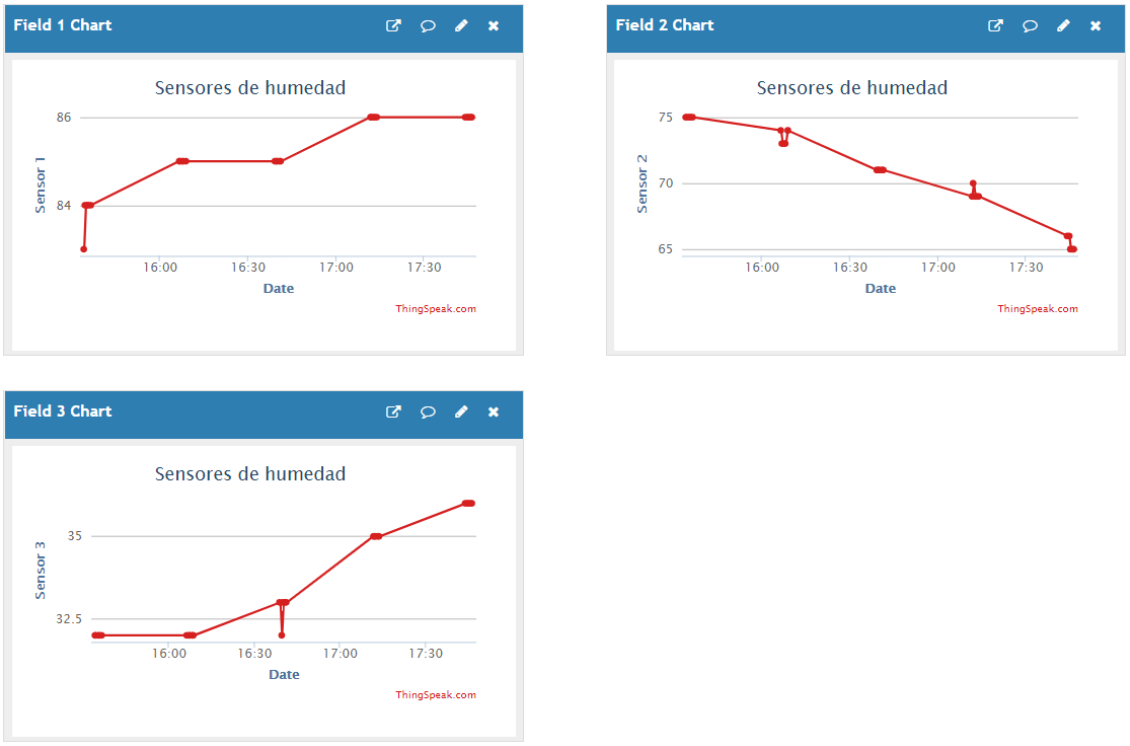


*Figura 19: Visualización de resultados - ThingSpeak*

Las mediciones se enviarán cada 15 segundos ya que la cuenta que disponemos de ThingSpeak es una versión gratuita (la versión premium ofrece medidas cada 1 segundo). Por este motivo hemos dividido las 5 mediciones en espacios de 15 segundos, para evitar que salten errores continuamente hasta pasar dichos intervalos y que se envíe correctamente.

### 6.3.1. Visualización mediante aplicación móvil

Una de las grandes ventajas de utilizar ThingSpeak para visualizar los datos de los sensores es la posibilidad de hacerlo en cualquier momento gracias a que dispone de una aplicación para los dispositivos móviles y tabletas. Esta aplicación tiene como nombre ThingView y está disponible en la tienda de Google Play de forma gratuita.

La forma de visualizar los datos se muestra en la Figura 20. Como se puede observar la aplicación es sencilla y fácil de manejar, lo cual favorece a la utilización por parte de cualquier usuario.

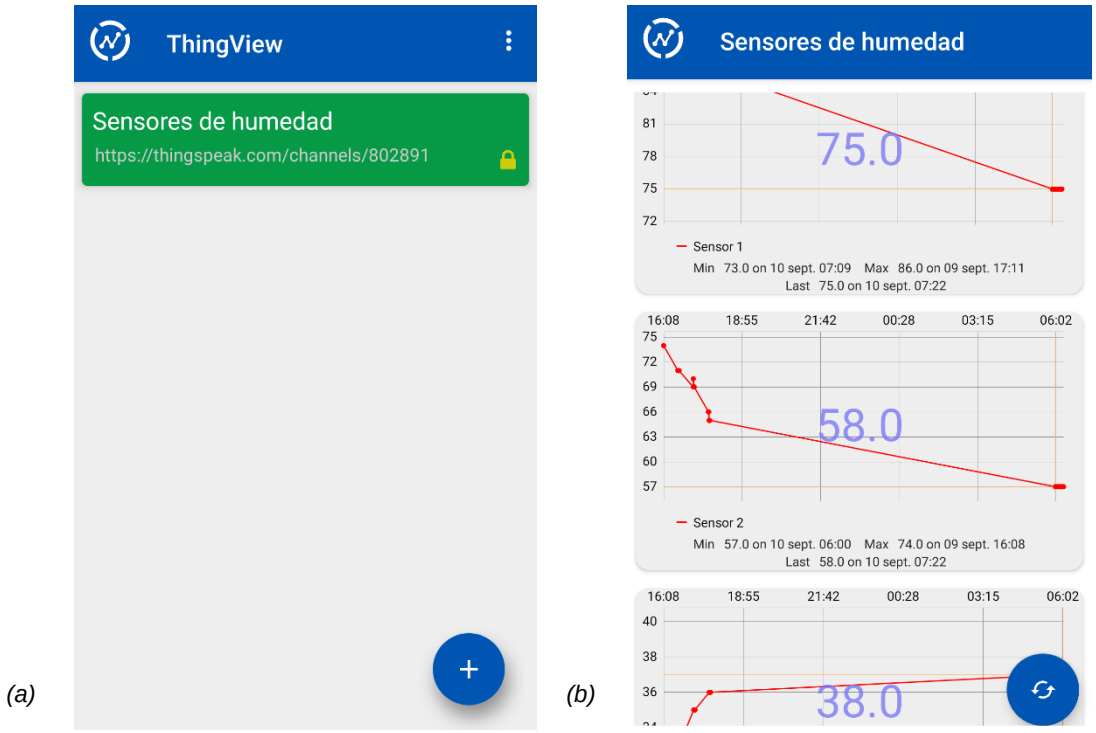


*Figura 20: Aplicación ThingView: (a) Pantalla principal. (b) Medidas sensores*

La aplicación es completamente intuitiva ya que con dos toques puedes acceder a las medidas de los sensores. Como se muestra en la Figura 20 (a), en la pantalla principal seleccionamos el canal asignado y nos aparecerán las medidas de todos los sensores de igual forma que en la Figura 20 (b). En la gráfica de cada sensor se muestra la hora de la última medida y otros datos que pueden ser de interés. En caso de querer

mayor resolución podemos tocar sobre la gráfica del sensor que queramos y podremos ampliar y reducir para ver más medidas juntas o para visualizar mejor las medidas más recientes.

## 6.4. Códigos de error

Como hemos dicho durante la explicación del código, la variable "x" recoge un código de envío, que en caso de no ser 200, nos dirá que hay un error a la hora de enviar los datos. Algunos de los códigos de error que pueden producirse y su descripción los podemos encontrar en la Tabla 5.

| Error | Descripción |
|---|---|
| 200 | La conexión se ha realizado correctamente |
| 400/404 | Clave API errónea o dirección de servidor ThingSpeak incorrecta |
| -101 | Valor fuera de rango o String demasiado largo (> 255 bytes) |
| -201 | Campo especificado invalido |
| -210 | setField() no se ha utilizado antes que writeFields() |
| -301 | Fallo durante la conexión a ThingSpeak |
| -302 | Fallo durante escritura en ThingSpeak |
| -303 | No se puede analizar la respuesta del servidor |
| -304 | El tiempo de espera para la respuesta del servidor ha vencido |
| -401 | Los datos no se han enviado |

*Tabla 6: Tabla de errores - ThingSpeak*

# 7. Mediciones del consumo real

Para realizar las mediciones de consumo hemos utilizado un multímetro situado entre la fuente de suministro de alimentación y la placa ESP32 en serie. Una vez conectado se enciende el dispositivo y comienza la ejecución del programa.

Se puede apreciar que, al iniciar, debido a las conexiones con el punto de acceso el consumo se dispara hasta los 150 mA aproximadamente durante un poco más de 1 segundo. Este valor es normal debido a que la función Wi-Fi tiene un consumo considerable pero únicamente durante un corto periodo. A continuación, empieza a ejecutarse el resto del programa y el consumo baja hasta alrededor de 80 mA entre 1 y 2 segundos hasta finalizar e iniciar el modo reposo.

Una vez estamos en el modo de reposo profundo, el consumo debería descender hasta los 10 µA, pero debido a un error en la fabricación del modelo de la placa utilizada, el consumo que obtenemos es bastante mayor. Este problema proviene de la junta, el diseño no está optimizado para el consumo de energía por lo que incluso en este modo de reposo, tenemos un valor medido de corriente de 20 mA, un aumento considerable que dista mucho de los esperados 10 µA.

## 7.1. Ahorro de energía: Modo Deep-Sleep

El consumo de energía de los ESP32 en su estado normal es un tanto elevado. Esto es debido a que, además de ser un procesador potente, contiene las conexiones de Wi-Fi y Bluetooth que necesitan una energía considerable para su funcionamiento.

En este punto entra en juego los diversos modos de ahorro de los que dispone esta placa, y que son muy necesarios para este tipo de proyectos pues permiten alargar su uso durante un mayor periodo de tiempo sin la necesidad de cambiar baterías o depender del uso de enchufes.

La placa utilizada para este proyecto contiene 3 modos diferentes de ahorro de energía, cada uno de ellos tiene un consumo pues dejan activas alguna de sus funciones. A continuación, se detalla cada uno de ellos:

- **Deep-Sleep:** Es el modo de mayor ahorro. Esto es debido a que deja la placa completamente en estado de suspensión, funcionando únicamente el reloj interno para poder reiniciarla una vez ha concluido el intervalo de tiempo designado para funcionar en este estado. Su consumo pasa a ser de 10 µA.

- **Modem-Sleep:** Este modo permite desactivar la conexión Wi-Fi cuando no vaya a ser utilizada y activarla cuando sea necesario. El consumo de la placa en este estado es de entre 3 y 20 mA.

- **Light-Sleep:** Permite mantener activa la conexión Wi-Fi, pero reduce el consumo de energía cuando no hay envío de información. Es una función interesante si no se quiere inducir la placa a un estado de reposo, pero se quiere optimizar el uso de las funciones inalámbricas. En este modo, se tiene un consumo de 0.8 mA.

El modo que se ha utilizado es el Deep-Sleep pues las medidas no se van a realizar durante todo el día. Lo que haremos, será mantener la placa operativa durante las mediciones y activar el modo de suspensión cuando no se mida.

Para conseguir el máximo ahorro posible, se han elegido diversas horas en las que el dispositivo estará completamente apagado pues no será necesario conocer el estado de las medidas como, por ejemplo, al caer la noche. El horario de funcionamiento que se ha programado es desde las 6 de la mañana hasta las 6 de la tarde. Tras este intervalo la placa será inducida a un estado de reposo las próximas 12 horas, durante las que tendrá un consumo ínfimo como se ha mostrado anteriormente.

A esta medida se ha añadido también un sistema de mediciones por cada hora limitado a 2 veces. Es decir, por cada hora de funcionamiento vamos a medir 1 vez cada 30 minutos. En cada una de estas mediciones, se medirá 5 veces para evitar cualquier fallo y asegurar el valor de cada medición.

## 7.2. Comparativa y previsión de la duración del dispositivo

Las medidas de ahorro que se han implementado deberían haber incrementado notablemente la duración de la batería. Para comprobarlo, vamos a calcular el consumo actual y será comparado con el consumo sin ningún modo de ahorro.

En primer lugar, vamos a ver la duración inicial sin usar el modo de ahorro mencionado. En los intervalos que no se realizan medidas, se ha utilizado la función de retardo en Arduino (delay) que en este caso consume 65 mA, un poco menos del promedio de la placa.

$$[12 \cdot 2 \cdot (65\ mA \cdot 5 \cdot 15s + 65\ mA \cdot 30 \cdot 60s + 150\ mA \cdot 5 \cdot 1.5s + 80\ mA \cdot 5 \cdot 1.5s)]$$
$$+ [65\ mA \cdot 12 \cdot 60 \cdot 60] = 5774400 \frac{mAs}{dia}$$

$$Duracion = \frac{3000\ mAh \cdot 3600\ s/h}{5774400\ mAs/dia} = 1.87\ dias$$

Como vemos, la duración sería muy corta, no llegaría a los 2 días. Por ello se ha decidido utilizar el modo de ahorro Deep Sleep, ya que como se podrá observar a continuación, existe una gran diferencia.

Teóricamente, la duración del dispositivo será la siguiente:

$$[12 \cdot 2 \cdot (0.010\ mA \cdot 5 \cdot 15s + 0.010\ mA \cdot 30 \cdot 60s + 150\ mA \cdot 5 \cdot 1.5s + 80\ mA \cdot 5 \cdot 1.5s)] + [0.010\ mA \cdot 12 \cdot 60 \cdot 60] = 42282 \frac{mAs}{dia}$$

$$Duracion = \frac{3000\ mAh \cdot 3600\ s/h}{42282\ mAs/dia} = 255.43\ dias$$

Se puede observar la gran duración prevista con el modo de ahorro elegido, en la que se prevé un periodo de funcionamiento independiente de 255 días.

Como hemos comentado anteriormente, debido a un error en el diseño de la placa se hace imposible conseguir este tiempo de uso por lo que necesitamos realizar una nueva estimación.

Finalmente, se obtiene la duración a partir del consumo real de la placa con el modo Deep Sleep:

$$[12 \cdot 2 \cdot (20\ mA \cdot 5 \cdot 15s + 20\ mA \cdot 30 \cdot 60s + 150\ mA \cdot 5 \cdot 1.5s + 80\ mA \cdot 5 \cdot 1.5s)] + [20\ mA \cdot 12 \cdot 60 \cdot 60] = 1805400 \frac{mAs}{dia}$$

$$Duracion = \frac{3000\ mAh \cdot 3600\ s/h}{1805400\ mAs/dia} = 5.98\ dias$$

La diferencia es notoria en comparación con la estimación teórica, pero también es cierto que inicialmente teníamos una previsión de duración menor a 2 días, por lo cual, en el peor de los casos hemos multiplicado por 3 su periodo de funcionamiento.

Cabe destacar, que en estas estimaciones no se ha incluido la carga adicional que proporciona el panel solar pues depende de diversas variables y aunque aumentaría levemente su tiempo de uso no se tienen datos contrastados.

# 8. Conclusiones

La finalidad de este proyecto es adaptar la tecnología a las necesidades de los cultivos. En este caso, queremos saber cuál es el estado de las diferentes partes de un árbol que se encuentran bajo tierra y que a simple vista no podemos averiguar. Un proyecto de estas características no solo es imprescindible para aumentar la calidad de los alimentos cultivados, sino para reducir en gran medida el impacto que estos tienen sobre los recursos naturales de nuestro planeta. Gracias a los datos recibidos por los sensores, sería idóneo la adaptación de un sistema de riego pues la gestión del agua incrementaría notablemente el ahorro total. Cuando hablamos de ahorro, no solo pensamos en los recursos del planeta sino también en los del ser humano. Es por ello por lo que uno de los factores más importantes para nosotros era reducir el coste de fabricación lo máximo posible para hacer el producto accesible al mayor número de personas.

Otros proyectos que se centran en medir la humedad de la tierra lo hacen únicamente con un sensor y en una zona poco profunda. La posibilidad de hacerlo a diferentes profundidades nos brinda información de mayor calidad sobre el estado de la planta analizada pues nos acercamos más a sus raíces y vemos cuál es su estado de humedad, un factor realmente importante.

Tras probar el producto nos damos cuenta de la facilidad que tiene para el consumidor, pues con la aplicación móvil se pueden conocer de forma rápida los datos medidos desde cualquier lugar. Con los ajustes necesarios, el sensor podría utilizarse perfectamente en los campos de cultivo para facilitar la vida a los agricultores y sobre todo para proteger al planeta del gran consumo de recursos al que se ve cada vez más expuesto.

Finalmente, en referencia al apartado de consumo, aunque no se hayan podido obtener los valores previstos desde un principio para el modo de ahorro Deep Sleep, se ha conseguido averiguar la raíz del problema para ser solucionada en el futuro. No obstante, el resto del proyecto ha cumplido con creces nuestras expectativas y no cabe duda de que su utilización es una opción viable de cara a un futuro cercano.

## 8.1. Problemas encontrados

A lo largo del proyecto han surgido diversos problemas que han afectado al proceso de formación del dispositivo final. Estos problemas han tenido que ser estudiados y solventados de la mejor manera posible para cumplir con los propósitos iniciales del proyecto.

Uno de los primeros problemas surgió cuando se instalaba el módulo de carga solar. Antes de conectarlo nos dimos cuenta de que era posible que la placa recibiera corriente de retorno y ello afectara a su funcionamiento consumiendo parte de la energía que obtenía del sol provocando, en el peor de los casos, que la batería no se recargara pues no llegaba el suministro necesario de alimentación para ello. Como solución a este problema se añadió un diodo 1N4007 que sacrifica 1V de la placa en beneficio de la protección de todo el circuito y de evitar estas posibles corrientes de retorno.

Relacionado también con el módulo solar está el hecho de que la batería utilizada no podía alimentar por si sola la placa. Esto es porque hay tres formas de alimentar dicha placa, una necesitaba entre 7 y 12V, la segunda 3.3V y por último mediante la entrada micro USB de la placa, mientras que nuestra batería proporciona un valor de tensión de 3.7 a 4.2V. Esto hacia necesaria la inclusión de, o bien un regulador de subida, uno de bajada o un convertidor USB. La solución escogida, esta última, esta detallada a lo hora de describir los materiales utilizados y el motivo de su elección.

En un principio, para alimentar la placa se había utilizado el método que incorpora un regulador Step Up mt3086, pero debido a un problema común de este modelo con el potenciómetro se hace imposible regular la tensión de entrada. Tras probar dos diferentes y experimentar el mismo problema se decidió por cambiar el componente por una solución que no implique aumentar el coste económico. Este nuevo componente es un convertidor Step Up USB que se puede encontrar por un precio similar o inferior y que brinda una mayor comodidad a la hora de alimentar directamente la placa con el cable USB conectado a este módulo.

El código utilizado también ha sufrido diversos cambios a como se había programado en un principio. Uno de esos cambios es a la hora de escribir las medidas en la tarjeta SD, pues cada vez que se reinicia (aunque no debiera ocurrir) se sobrescriben todos los datos. Esto es por el uso del comando Write. Por ello, hemos cambiado este comando para usar “Append” para que, en caso de fallo o reinicio del sistema, al iniciarse de nuevo las medidas se continúen escribiendo desde el último punto.

Como ya hemos indicado, uno de los principales propósitos del proyecto es que el dispositivo final tenga el menor consumo posible. La placa que usamos dispone de diversos modos que permiten ponerla en estado de reposo. De estos modos se ha elegido el adecuado para mantener las funciones necesarias activas y dejar en un estado de suspensión el resto. Cuando sea necesario realizar las medidas, se iniciarán todas las funciones y luego se volverá a dejar en reposo. Esto nos permite reducir en gran medida el consumo inicial que tenía el modo predeterminado.

La placa utilizada tiene una tensión de funcionamiento de 3.3V, esto provoca que el rango de valores que obtenemos con los sensores sea de 0 a 4095 a diferencia de las placas típicas de Arduino que funcionan a 5V y tienen un rango de 0-1024. Para utilizar un rango más intuitivo para el ser humano hemos mapeado estos valores para que se muestren en un intervalo entre 0-100 en el que 0 pertenece a un estado completamente seco y 100 cuando está sumergido totalmente en agua. Este cambio beneficia que cualquier usuario pueda interpretar las medidas de los sensores más fácilmente.

Al utilizar ThingSpeak también hemos tenido diversos contratiempos. Por un lado, en el código, a la hora de escribir los datos de los sensores en su correspondiente campo usábamos el comando ThingSpeak.writeField () con cada campo individualmente y por tanto debíamos de poner 3 líneas de código una por cada uno de ellos. Sin embargo, la transmisión fallaba y fue necesario recurrir a otro método con ThingSpeak.writeFields el cual escribe todos los campos a la vez y el problema fue solucionado. El segundo contratiempo con este servicio fue en relación con el tiempo de envió de datos. Esto ocurría debido a que nosotros mandábamos datos constantemente y al utilizar una cuenta en versión gratuita, el tiempo entre una medida y otra debía ser de 15 segundos.

Para realizar el prototipo del circuito en Fritzing ha sido necesario buscar cada uno de los componentes utilizados. Algunos de ellos han resultado difíciles de encontrar pues no estaban en la base de datos de Fritzing, pero gracias a la página GitHub ha sido posible encontrar los diseños de algunos de los componentes utilizados. Otros no ha sido posible obtenerlos y ha hecho falta el uso de un software para crearlos desde cero. Este software es Inkscape [15], un editor de gráficos vectoriales libre y de código abierto.

Medir el tiempo en Arduino no es una tarea sencilla, al menos, si tu finalidad es hacerlo únicamente por software. El principal inconveniente es debido a que las horas y los minutos se cuentan con un sistema sexagesimal, es decir, basado en el numero 60 por cada grado y no en el centesimal normal el cual está basado en un valor de 100 por cada grado. Además, cabe contar los cambios en los días de cada mes, años bisiestos,

etc. Todo esto impide una correcta programación por código de un reloj como tal. Una alternativa sería usar la librería Time.h la cual se apoya en un tipo especial de variable de 32 bits llamada time_t la cual almacena valores desde una fecha dada hasta el 1 de enero de 1970 y mediante diversas operaciones calcula los años, meses, días, minutos y segundos. El problema de usar esta librería es que, si por cualquier fallo la placa se reinicia, vuelve a la hora inicial, lo cual conllevaría un error en cadena en las horas de las medidas. La solución utilizada es proporcionada por la capacidad de conexión inalámbrica Wi-Fi de nuestra placa. Mediante ella, vamos a acceder a la hora exacta de la zona geográfica en la que nos encontramos a través de NTP (Network Time Protocol), un protocolo de internet usado para sincronizar los relojes de los sistemas informáticos. Este método actualiza la hora cada vez que se repite el bucle para evitar cualquier fallo.

A pesar de la solución empleada para conocer la hora de las mediciones, otro problema estrechamente relacionado con el reloj es la necesidad de despertar la placa del modo de reposo. Lo ideal sería hacerlo a horas determinadas, pero hemos utilizado una solución alternativa la cual es programar los tiempos de reposo durante un intervalo determinado para que se despierte al finalizar este, realice mediciones y luego vuelva de nuevo a su estado de reposo.

Una vez tenemos acceso a la hora, podemos llevar a cabo la programación completa del funcionamiento del dispositivo. Se pretende que la placa esté en funcionamiento ininterrumpidamente por 1 año como mínimo. En un principio se realizaban un número elevado de medidas por hora, pero, tras ver que afectaba en gran medida a la capacidad de la batería, se fue reduciendo la cantidad para favorecer un menor gasto de energía. Para lograr el máximo ahorro se ha diseñado un sistema de medidas cada 30 minutos durante 12 horas, y las 12 horas restantes iniciar el modo de ahorro. Sin embargo, surgió un problema por el que no se conseguía dormir la placa durante un periodo mayor a 36 minutos pues el tiempo definido para cada intervalo de reposo se almacenaba en variables de 16 bits. Como nosotros necesitamos hasta 12 horas en estado de reposo, para solucionar este contratiempo, hemos utilizado la variable de tipo uint64_t que permite datos de 64 bits para poder llevar a cabo estos largos periodos del modo de ahorro Deep Sleep.

Como se ha podido observar en relación con el consumo en el apartado 7, ha surgido un problema adicional debido al modelo de placa utilizado (DOIT). Este error forma parte del propio diseño, lo cual hace inevitable tener que encontrar otra alternativa. Afortunadamente, el mercado siempre nos ofrece placas ESP32 mejor diseñadas como

por ejemplo la placa FireBeetle ESP32 de DFrobot la cual si puede lograr la corriente esperada en modo Deep Sleep de 10 µA.

## 8.2. Futuras mejoras del proyecto

Como meta final, se espera la implementación en campos de cultivo para mejorar el rendimiento y la calidad de los productos plantados.

El sistema diseñado presenta un mayor potencial si lo asimilamos en conjunto con un sistema de riego automático. La naturaleza de nuestro sensor permitiría la opción de gestionar el riego en función de los niveles de las medidas obtenidas, sería un complemento ideal para llevar al máximo todos los beneficios que tiene este proyecto.

Los sensores de humedad utilizados presentan tiras chapadas en cobre que funcionan como material conductor. Cuando este tipo de sensores están expuestos al agua durante un largo periodo de tiempo, el cobre comienza a desprenderse y puede afectar a que las medidas no sean del todo exactas. Este es uno de sus mayores inconvenientes y aunque nuestros sensores no estarán constantemente expuestos al agua, pues detectarán la humedad de la tierra, es posible que se vean afectados a largo plazo. Para mejorar el uso de estos sensores en el futuro sería conveniente utilizar un proceso de chapado en oro o similar para proteger el sensor ante este fenómeno de descomposición.

Una forma más cómoda de controlar los periodos de sueño del dispositivo sería la inclusión de un reloj externo RTC. De este modo podríamos programar las horas exactas a las que despertar la placa para realizar las mediciones.

Para conocer más en profundidad el estado de las plantas analizadas sería de gran ayuda añadir varios sensores que realicen funciones para monitorizar la temperatura, la información del PH de la tierra, la concentración de nutrientes que rodea cada árbol, etc. Esto ayudaría a tratar cada planta exactamente según todas las necesidades lo cual provocaría la producción de un alimento con una calidad óptima.

Todos los dispositivos electrónicos nunca están a salvo de errores inesperados. Por ello, sería conveniente añadir una botonera para poder reiniciar la placa manualmente en caso de experimentar cualquier fallo. Una forma sencilla para lograrlo sería añadir un interruptor para quitar el paso de corriente y apagar la placa. Esto añadiría el sistema de botonera de una forma rápida y efectiva.

Cualquier aplicación IoT (Internet of Things por sus siglas en inglés o Internet de las Cosas, abreviado IdC en español) que forme parte de grandes áreas puede obtener un gran beneficio de las ventajas de la comunicación móvil GSM/3G/4G. Esta alternativa al uso actual de Wifi presenta un gran inconveniente el cual es su gran consumo y su elevado coste económico. Sin embargo, puede ser ideal para este tipo de proyectos que integran sensores y que no requieren un ancho de banda demasiado grande para enviar datos a través de internet.

Como ya hemos comentado anteriormente, el sensor en su fase de medidas realiza 5 consecutivas para evitar cualquier medida errónea que pueda darse. Sin embargo, esta solución sacrifica cierto consumo eléctrico a la hora de enviar sus datos a ThingSpeak tras cada una de las medidas. Una posibilidad para tener en cuenta en un futuro sería hacer una media aritmética de los 5 valores obtenidos del sensor para realizar un único envío a ThingSpeak y reducir el uso de la conexión a internet de la placa.

Al usar la hora que obtenemos del cliente NTP sería interesante añadir un módulo GPS para detectar la zona geográfica en la que se encuentra el sensor para acceder automáticamente al cliente adecuado desde cada lugar del planeta.

Para visualizar los datos en cualquier teléfono móvil o tableta con sistema operativo Android hemos utilizado la aplicación ya existente ThingView. Una posibilidad de futuro sería la creación de una aplicación propia para llevar a cabo nuevas formas de distribución de las medidas. Una posibilidad podría ser dividiendo la superficie en diferentes zonas y hacer una media de todos los sensores que conforman cada una de ellas. De esta forma, se podrían visualizar una mayor cantidad de datos de los árboles a la vez y no cada sensor de cada planta independientemente. Existe un gran abanico de posibilidades para poder organizar estas medidas, sin embargo, es una opción adaptativa al tipo de plantación que se esté utilizando, por ello dependerá de las necesidades del producto y de las preferencias para visualizar estos datos por parte del usuario final.

Para optimizar la construcción del sensor, resultaría conveniente utilizar una placa personalizada por medio de un circuito impreso para conectar a ella directamente algunas conexiones y eliminar el uso excesivo de cableado.

Con el fin de conocer el ahorro total que proporciona el dispositivo final, sería interesante implantarlo en un entorno real para realizar pruebas en relación con el consumo de agua y así determinar cuál es el beneficio económico que generaría la implantación de este sistema.

# 9. Bibliografía

[1] La importancia de la agricultura en la actualidad, por la FAO. Accesible online: http://www.fao.org/3/a0015s/a0015s04.htm (Enlace activo a 23 de agosto 2019)

[2] La agricultura es clave para afrontar las necesidades futuras de agua y energía, por la FAO. Accesible online: http://www.fao.org/news/story/es/item/94786/icode/ (Enlace activo a 23 de agosto 2019)

[3] Parra Boronat, L.; Sendra, S.; Lloret, J.; Bosch Roig, I. (2015). Development and test of conductivity sensor for monitoring groundwater resources to optimize the water management in Smart City environments. Sensors. 15(9):20990-21015. doi:10.3390/s150920990

[4] Rocher, J.; Parra, L.; Taha, M.; Lloret, J. (2018). Diseño de una red de sensores para monitorizar una instalación acuícola. En XIII Jornadas de Ingeniería telemática (JITEL 2017). Libro de actas. Editorial Universitat Politècnica de València. 48-54. doi:10.4995/JITEL2017.2017.6623

[5] Alberto Losada Villasante, Eficiencia técnica en la utilización del agua de riego, Revista de Estudios Agro-Sociales. Núm. 167 (enero-marzo 1994)

[6] Pasquale Steduto, Theodore C. Hsiao, Elias Fereres, Dirk Raes, Respuesta del rendimiento de los cultivos al agua. 2012

[7] Estándar IEEE 802.11: https://ieeexplore.ieee.org/servlet/opac?punumber=7786993

[8] Wi-Fi Alliance: https://www.wi-fi.org/

[9] André Knörig, Reto Wettach, Jonathan Cohen, Fritzing – A tool for advancing electronic prototyping for designers

[10] Fritzing: https://fritzing.org/home/

[11] Mohamad Jamil, Muhamad Said, Sumiyati Samuda, The Utilization Of Internet Of Things For Multi Sensor Data Acquisition Using Thingspeak. 2018

[12] Sharmad Pasha, Thingspeak Based Sensing and Monitoring System for IoT with Matlab Analysis, International Journal of New Technology and Research (IJNTR) ISSN: 2454-4116, Volume-2, Issue-6, June 2016 Pages 19-23

[13] ThingSpeak: https://thingspeak.com/

[14] David Nettikadan, Subodh Raj M.S. Smart Community Monitoring System using Thingspeak IoT Plaform, International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 17 (2018) pp. 13402-13408

[15] Inkscape: https://inkscape.org/es/