Grado en Ingeniería de Tecnologías de Telecomunicación

Escuela Politécnica Superior de Linares

UNIVERSIDAD DE JAÉN

Escuela Politécnica Superior de Linares

ESTACIÓN METEOROLÓGICA BASADA EN LORAWAN Y ALIMENTACIÓN SOLAR CON ENERGÍA SOLAR FOTOVOLTÁICA

Alumno: **Manuel Antonio Ventura Duque**

Tutor: Prof. D. Jesús de la Casa Cárdenas

Depto.: Ingeniería Electrónica y Automática.

Marzo, 2022

UNIVERSIDAD DE JAÉN

Escuela Politécnica Superior de Linares

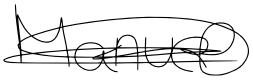
ESTACIÓN METEOROLÓGICA BASADA EN LORAWAN Y ALIMENTACIÓN SOLAR CON ENERGÍA SOLAR

Alumno: **Manuel Antonio Ventura Duque**

Tutor: Prof. D. Jesús de la Casa Cárdenas

Depto.: Ingeniería Electrónica y Automática

Firma del Alumno:



Firma del Tutor:

DE LA CASA CARDENAS JESUS - 26022699Q

Firmado digitalmente por DE LA CASA CARDENAS JESUS - 26022699Q
Fecha: 2022.02.17 09:03:41 +01'00'

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 RESUMEN

Una estación meteorológica es un sistema capaz de monitorizar las condiciones climatológicas que hay en un determinado lugar. A veces, estos sistemas no pueden transmitir información a tiempo real a un servidor, sino que almacena la información en memorias o discos duros externos para posteriormente analizarla.

Otro inconveniente es que tiene que ser capaz de funcionar de una manera autónoma sin depender de una toma de corriente eléctrica convencional, dado que se instalan en zonas de difícil acceso o remotas.

En este trabajo final de Grado se propone una solución eficaz y con tecnologías emergentes a esta problemática. Se diseñará una estación meteorológica de bajo coste, que transmitirá la información mediante la tecnología LoRa pudiendo alcanzar una comunicación de hasta 20 kilómetros.

Además, se diseñará un sistema de alimentación autónomo con placas solares fotovoltaicas para solventar el problema del lugar de instalación.

Por último, se desarrollará una interfaz gráfica de usuario, para que cualquier persona pueda ver las condiciones meteorológicas del lugar desde su móvil, *tablet* u ordenador.

2 INTRODUCCIÓN

Una estación meteorológica es un sistema conformado por distintos componentes electrónicos, que tienen como misión recolectar información de índole climatológico. Estos componentes normalmente son medidores y sensores como el de temperatura, humedad, presión, viento, o lluvia. Además, se requiere de un microcontrolador que sea capaz de procesar toda esta información y presentársela al usuario para que pueda visualizarla.

Todos estos datos se suelen emplear para varios ámbitos de aplicación, como, por ejemplo, para predicciones meteorológicas a partir de modelos matemáticos, mantenimiento en explotaciones agrarias, etc.

A día de hoy las estaciones meteorológicas se han convertido en algo fundamental en cualquier país del mundo, y que si se sigue investigando e innovando pueden ser muy valiosas para la sociedad.

En este Trabajo Fin de Grado se pretende instaurar una nueva tecnología de comunicaciones inalámbricas de alto alcance, para que en aquellas zonas más remotas se puedan instalar dichas estaciones sin necesidad de Internet o cualquier memoria para el almacenamiento. De esta manera, se puede proporcionar una alternativa más, aprovechando la gran cantidad de recursos que nos ofrece la tecnología para seguir innovando y hacer las cosas más fáciles para sectores como el agrario.

Esta nueva tecnología se llama LoRa, y permite tener comunicaciones hasta de 20 kilómetros en condiciones normales [4]. Estas condiciones normales se basan en:

- Una altura adecuada del transmisor y receptor, donde no interfiera ningún obstáculo (campo abierto o directo). Hablaremos de ello en el apartado 2.1.
- Antenas de alta ganancia.
- Bit-Rate de 250 bps a 50 Kbps [21].
- Escenario limpio y sin condiciones climatológicas desfavorables.

Además, para no depender de una toma de corriente convencional, y así instalar la estación en cualquier lugar, se propondrá un sistema de generación de energía renovable con paneles solares fotovoltaicos. Se tendrá que realizar un estudio de consumo energético, para que la autonomía del sistema meteorológico sea de 24 horas los 365 días del año. Para ello se propondrá un ejemplo de caso real, en la localidad de Linares donde calcularemos cuántas placas solares, y que batería necesitaremos en función de las condiciones climatológicas medias existentes en la zona de la instalación.

Figura 2.1. Estación meteorológica en un campo de cultivo

Toda la información será procesada por un microcontrolador ESP32, uno para la transmisión y otro para la recepción. El microcontrolador más utilizado normalmente para este tipo de proyectos son los de la marca Arduino, pero debido a las grandes ventajas que presenta ESP32 respecto Arduino, para este proyecto en concreto, hace que se decante sin lugar a duda por el ESP32.

A continuación, se muestra una breve comparativa entre los dos microcontroladores mencionados anteriormente:

	Arduino Uno	**ESP32**
Alimentación	5 V	3,3 V
Memoria Flash	32 KB	4 MB
WiFi	No	Sí
Bluetooth	No	Sí
Corriente CC por pin de E/S	40 mA	20 mA
Pines de E/S digitales	14	36
Pines Analógicos	6	15
Precio Medio	20 €	10 €

Tabla 2.1. Comparativa entre Arduino Uno y ESP32 [1]

Como se puede ver en la Tabla 2.1. Comparativa entre Arduino Uno y ESP32 [1] existen numerosas diferencias entre el Arduino Uno y el ESP32. Las conclusiones que se pueden extraer, es que ya no solo el ESP32 es un microcontrolador mucho más eficiente que el Arduino Uno, sino que tiene un coste de la mitad de precio.

Como este proyecto requiere una alta eficiencia energética al estar en un lugar remoto, y tiene como objetivo ser del menor coste posible, se escogerá ESP32.

Antes del desarrollo y de la implementación del proyecto, es necesario conocer ciertos conceptos y problemáticas que se tiene a la hora de realizar una comunicación transmisor-receptor. Así como, los aspectos legales que hay cumplir en cualquier transmisión en el territorio nacional, como la frecuencia a emplear. También se expondrá el funcionamiento del protocolo de comunicación LoRaWAN y toda la información de interés a cerca de esta tecnología.

2.1 Conceptos en las comunicaciones inalámbricas

2.1.1 Atenuación de la señal

La atenuación es la pérdida de potencia de la señal transmitida debido al medio de transmisión. Existen numerosos modelos de propagación, que determinan las pérdidas. Uno de los más conocidos es el de espacio libre, que se calcula con la ecuación (1) y (2) en los enlaces punto a punto [3]:

$$L = 20\log_{10}\left(\frac{4\pi d}{\lambda}\right)\ (dB)\ (1)$$

Donde:

- L: pérdida básica de transmisión en espacio libre.
- d: distancia en metros.
- λ: longitud de onda en metros.

Esta ecuación también se puede modelar en función de la frecuencia, que sería de la siguiente forma:

$$L = 32{,}4 + 20\ \log_{10} f + 20\ \log_{10} d\ \ (dB)\ (2)$$

Si se modela la ecuación anterior en Matlab, y se representa gráficamente se puede obtener la dependencia que tienen las pérdidas en función de la distancia, y la frecuencia

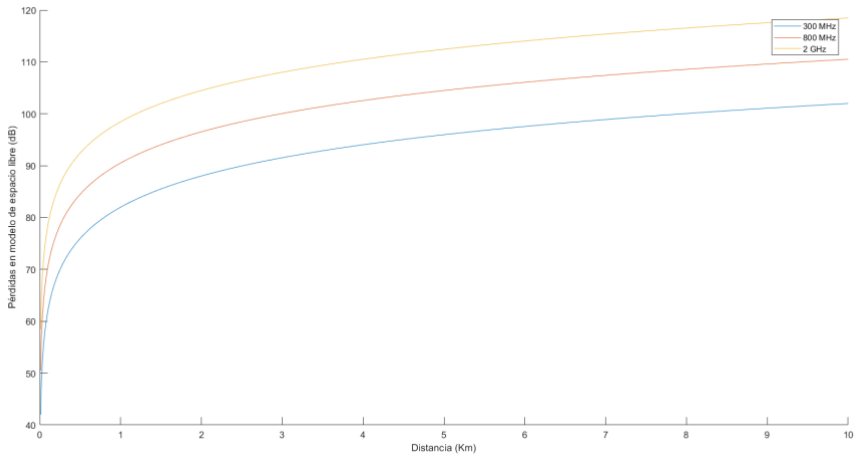


Figura 2.2. Pérdidas de propagación en espacio libre

También, hay que tener en cuenta la posibilidad de que haya obstáculos en la comunicación debido a edificios, árboles, etc. El modelo de propagación anterior no recoge estos sucesos, sino que lo hace la Recomendación UIT-R P.526 [29].

Este modelo se suele aplicar en entornos rurales (como es este caso), y permite calcular el efecto de la difracción cuando entre en el transmisor y receptor haya un obstáculo.

En el caso más idealizado, cuando el obstáculo es único, en arista en filo de cuchillo, las pérdidas suelen seguir el comportamiento de la Figura 2.3.

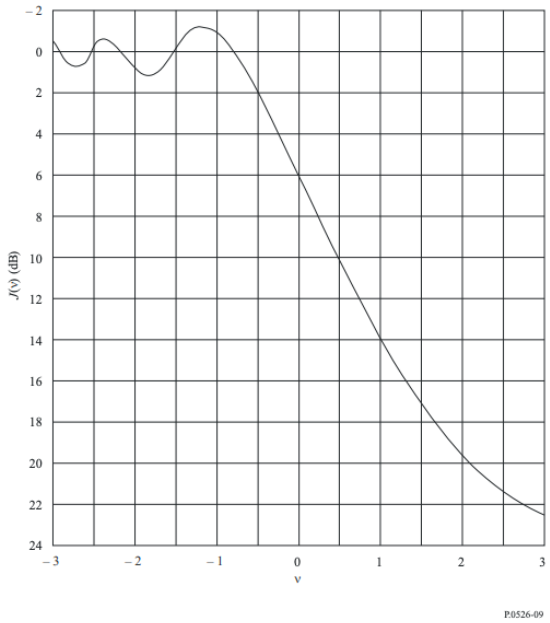


Figura 2.3. Pérdidas por difracción en obstáculos en arista de cuchillo

La Figura 2.3 está modelada por un parámetro en el eje de abscisas denominado υ de carácter adimensional. Este se denomina parámetro de difracción de Fresnel-Kirchhoff, que se determina con la ecuación (3).

$$\upsilon = h\sqrt{\frac{2(d_1 + d_2)}{\lambda d_1 d_2}} \quad (3)$$

Donde:

- *h* es la altura del obstáculo, expresada en metros, respecto de la línea vista entre transmisor y receptor, pudiendo ser tanto positiva como negativa dependiendo de si la altura del obstáculo supera o no la línea vista entre ambos extremos de la comunicación.
- d_1 y d_2 son las distancias del vértice del obstáculo al transmisor y al receptor.
- λ es la longitud de onda.

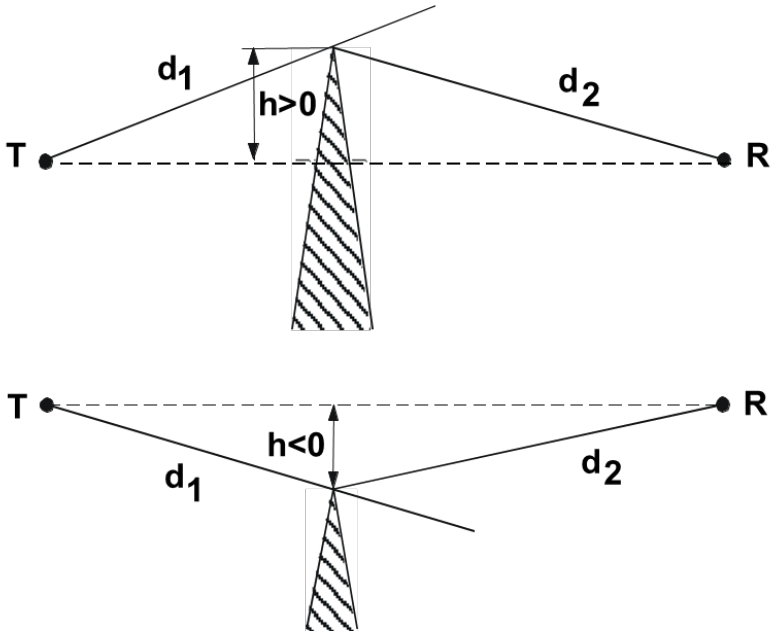


Figura 2.4. Obstrucción o despejamiento del obstáculo

El parámetro υ también se puede relacionar de la siguiente forma:

$$\upsilon = \sqrt{2}\frac{h}{R_1} \quad (4)$$

Donde R_1 es el radio de la primera zona de Fresnel. Una zona de Fresnel es una región elipsoidal que ocupa un volumen de espacio entre el emisor y el receptor, de modo que el desfase no sea mayor de 180º [13].

La primera zona de Fresnel es aquella en la que existe visión directa (LOS), y no hay ningún obstáculo. En este proyecto, esta zona va a ser fundamental para la transmisión, dado que los módulos LoRa no están preparados para pérdidas por difracción, tal y como se puede observar en el anexo 11.4.2, por lo que la comunicación no será exitosa en caso de que exista un gran obstáculo en el medio.

Las zonas de Fresnel son esenciales a la hora de diseñar la arquitectura de un radioenlace, pues permiten calcular si un obstáculo va a producir una gran pérdida de la señal. Como norma general, si la primera zona de Fresnel está despejada un 60% la comunicación puede ser aceptable [13].

Dicho de otra forma, para considerarse una comunicación aceptable la distancia entre el punto más alto del obstáculo y la línea vista de la comunicación debe ser superior a un 60% del valor de la primera región de Fresnel (R_1). En la Figura 2.5 se ve de forma gráfica esta región.

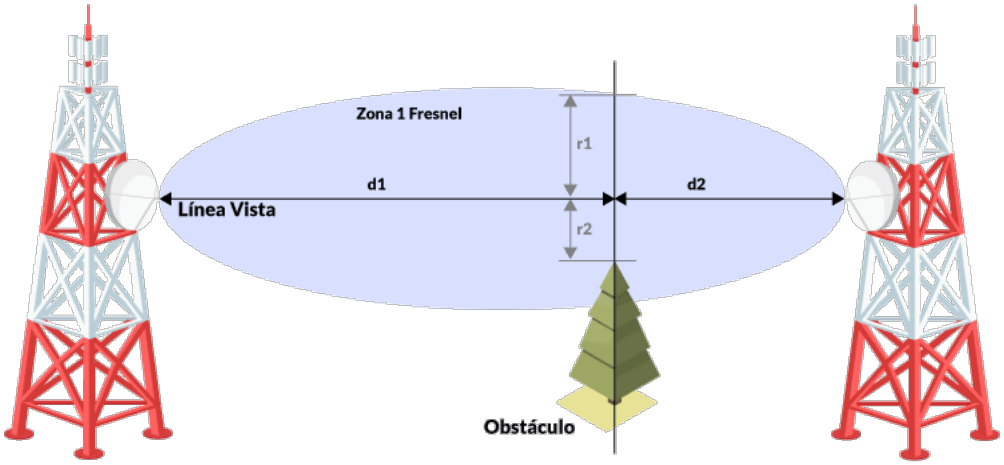


Figura 2.5. Esquema de comunicación en la primera zona de Fresnel [13]

2.1.2 Ganancia de una antena

La ganancia de una antena es otro de los conceptos fundamentales en las comunicaciones inalámbricas. Se produce por el efecto de la directividad al concentrar la potencia en aquellas zonas del diagrama de radiación interesadas. Normalmente esta ganancia se mide en dBi, si está referenciada respecto a una antena isotrópica, o dBd si es respecto un dipolo de media onda.

En aquellas comunicaciones que están limitadas en la potencia del transmisor se suele incluir antenas con una mayor ganancia para que la potencia que llegue al receptor sea la adecuada.

2.1.3 Sensibilidad

La sensibilidad es el nivel de potencia mínimo, que necesita el equipo receptor para su correcto funcionamiento. Normalmente suele medirse en dBm si nos referimos a potencia o dBμV si nos referimos a voltaje.

A la hora de diseñar un radioenlace es conveniente realizar un análisis o un balance de potencia, donde se pueda comprobar que la potencia que llegue al receptor sea como mínimo la correspondiente a la sensibilidad del receptor.

En estos radioenlances tiene un gran papel el ingeniero de telecomunicaciones, donde parte como incógnita las antenas que va a escoger y la potencia de transmisión. Para caracterizar estos parámetros, se tiene que realizar un cálculo de todas las pérdidas, y en función a estas diseñar el radioenlace con las antenas adecuadas.

Como se va a emplear la tecnología LoRa es conveniente conocer cual es la sensibilidad para que un receptor escuche la señal de un remitente perfectamente.

Normalmente, la sensibilidad suele ser de -120 dBm considerándose una señal bastante débil. Para hablar de una buena calidad de señal esta se tiene que situar en un nivel de potencia de -30 dBm. Con estos datos se podrá calcular todo lo mencionado anteriormente [21].

2.2 Tecnología LoRa

LoRa es una tecnología para comunicaciones inalámbricas desarrollada por la empresa Semtech, aunque administrada por LoRa Alliance. Su principal ventaja es su bajo consumo, lo que la hace ideal para el desarrollo de aplicaciones IoT.

Su funcionamiento se basa en técnicas de modulación de espectro ensanchado chirp (CSS), que es muy similar a la modulación FSK, pero aumentando el rango de comunicación [21].

Esta modulación se basa en utilizar pulsos chirp modulados en frecuencia lineal de banda ancha para codificar la información. La frecuencia de portadora varía de forma continua y lineal con el tiempo. Esta variación cuando llega a su limite final, retoma y vuelve al principio como se puede ver en la Figura 2.6.

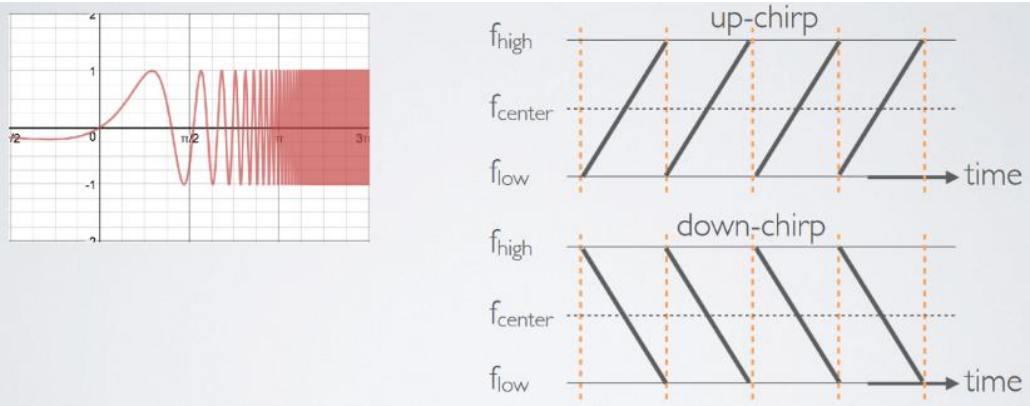


Figura 2.6. Modulación Chirp de espectro ensanchado [7]

Un ejemplo de transmisión de un mensaje con esta modulación LoRa es la siguiente:

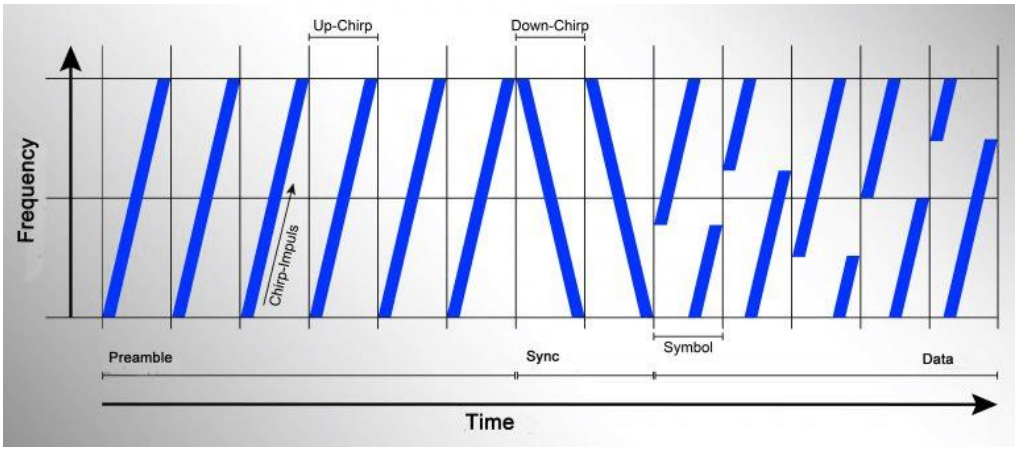


Figura 2.7. Ejemplo de modulación LoRa [7]

Como se puede analizar en la Figura 2.7 la modulación de un mensaje tiene diferentes partes:

1. **Preámbulo**. En esta parte se envían 6 pulsos chirp ascendentes para la detección de la señal LoRa.
2. **Sincronización**. Se envían 2 pulsos chirp descendentes para la sincronización en el tiempo.
3. **Envío de datos**. A partir de la sincronización comienza la transmisión de datos. Esta se caracteriza por transmitirse en ráfagas cortas, saltando entre frecuencias en secuencias pseudoaleatorias.

Visto desde un analizador de espectros:

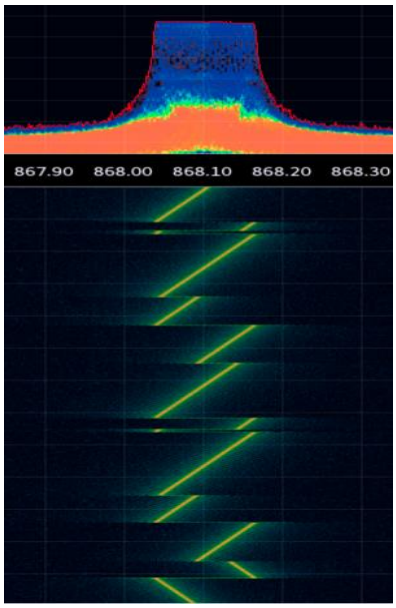


Figura 2.8. Modulación LoRa vista desde un analizador de espectros [7]

Ahora bien, para la demodulación del mensaje transmitido se coge como referencia la señal chirp descendente. Se descarta el preámbulo y se empieza a decodificar los símbolos según el esquema de la Figura 2.9

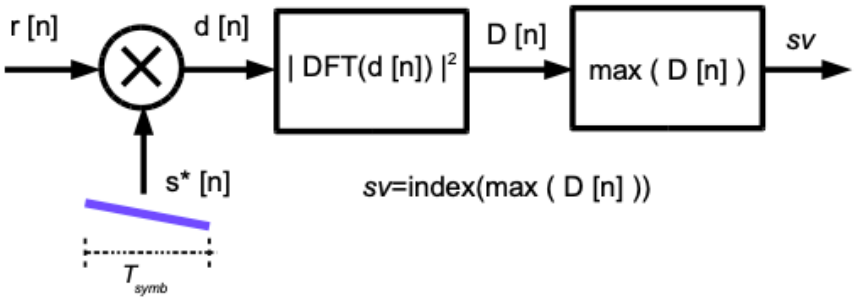


Figura 2.9. Esquema de demodulación de un mensaje LoRa [21]

Este sistema de demodulación consiste en calcular la densidad espectral de potencia, y aquellos índices que contengan la mayor potencia serán los símbolos que se han enviado.

Si por ejemplo se envían los símbolos 0, 16, y 112 por un canal con una sincronización perfecta, se tendría que recibir lo que se aprecia en la Figura 2.10 y Figura 2.11, siguiendo el esquema de demodulación de la Figura 2.9.

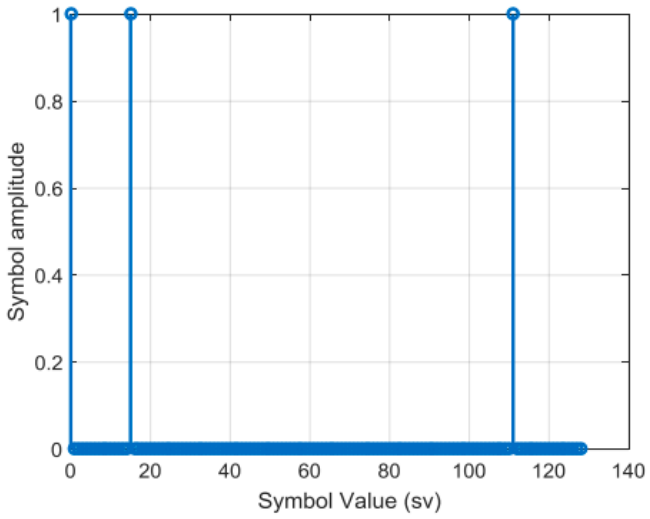


Figura 2.10. Símbolos decodificados de un mensaje LoRa [7]

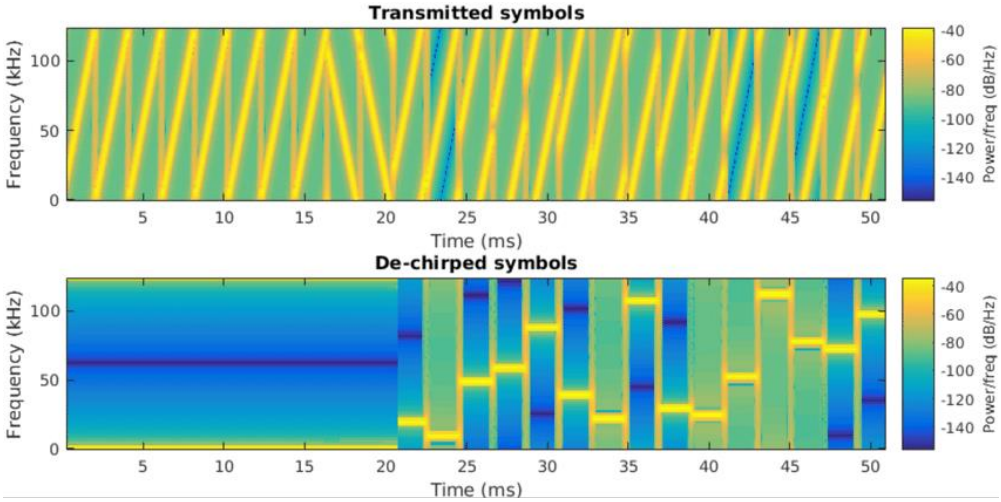


Figura 2.11. Espectrograma de demodulación del mensaje LoRa [18]

Todo lo mencionado anteriormente se corresponde a un nivel físico. A continuación, se detallará el protocolo de comunicación LoRaWAN, que se sitúa a nivel de capa de enlace, y que es el encargado de la comunicación de paquetes y administrar dispositivos LoRa.

2.2.1 Protocolo de comunicación LoRaWAN

Como se ha mencionado anteriormente, LoRa se encarga de definir la señal de radio, las frecuencias (en función de la región) y la modulación. Mientras que LoRaWAN es el protocolo de comunicación entre dispositivos de dicha tecnología. Las principales características de este protocolo son las siguientes:

- Presenta una comunicación extremo a extremo bidireccional con encriptación de la información lo que la hace totalmente segura.
- Topología en estrella.

- Alcance de hasta 20 kilómetros, en condiciones ideales.
- Baja transferencia de datos hasta los 250 bytes.
- Administración de dispositivos.

La pila de protocolos se puede observar en la Figura 2.12, donde se reflejan cada una de las capas.

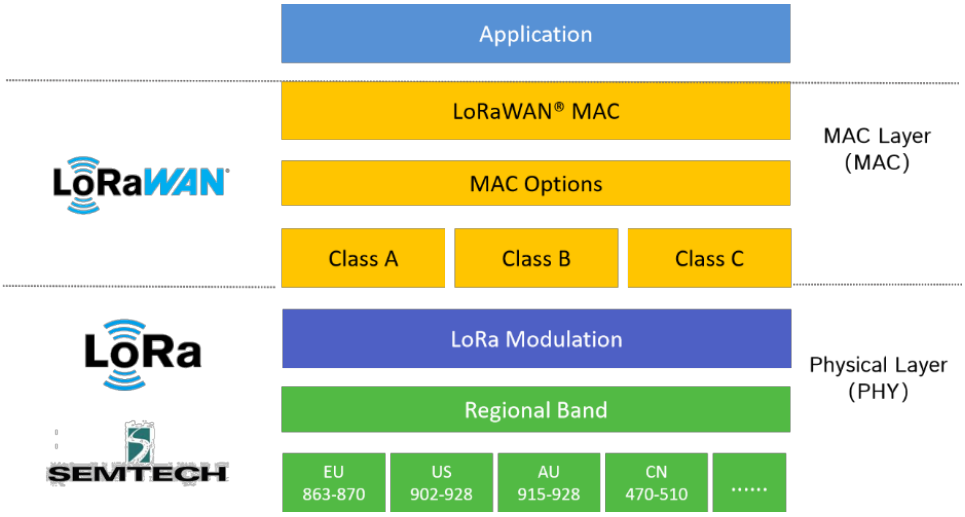


Figura 2.12. Pila de protocolos LoRa

Existen 3 clases de nodos en la red de LoRaWAN, donde cada una presenta una funcionalidad diferente para distintas aplicaciones. A continuación, se detallarán cada una de estas clases.

2.2.1.1 Dispositivos Clase A

Los dispositivos clase A son los más eficientes energéticamente. Esto se debe a que en la comunicación solo se pueden transmitir datos en el canal descendente si y solo si se ha enviado previamente un paquete para activar el canal ascendente. A esto también se le conoce como “modo escucha”.

Esta clase es válida para todos los dispositivos de la red LoRa.

2.2.1.2 Dispositivos Clase B

Los dispositivos clase B son capaces de recibir datos por el canal descendente sin necesidad de enviar un paquete de activación del canal ascendente.

Esto se consigue gracias a los *beacons,* que se envían constantemente por la puerta de enlace para estar sincronizado con el dispositivo. De esta manera se pueden programar envíos de información por el enlace descendente. Estos dispositivos se encuentran en un modo de suspensión, a la espera de negociar los tiempos para el envío de paquetes.

Su consumo de energía es mayor al estar todo el rato enviando *beacons*.

2.2.1.3 Dispositivos Clase C

Los dispositivos clase C está escuchando todo el tiempo, por lo que se pueden enviar datos o comandos por el enlace descendente en cualquier momento.

Son los que más consumo energético tienen.

2.2.1.4 Arquitectura de la red LoRaWAN

En la Figura 2.13 se puede visualizar la arquitectura de la red LoRaWAN:

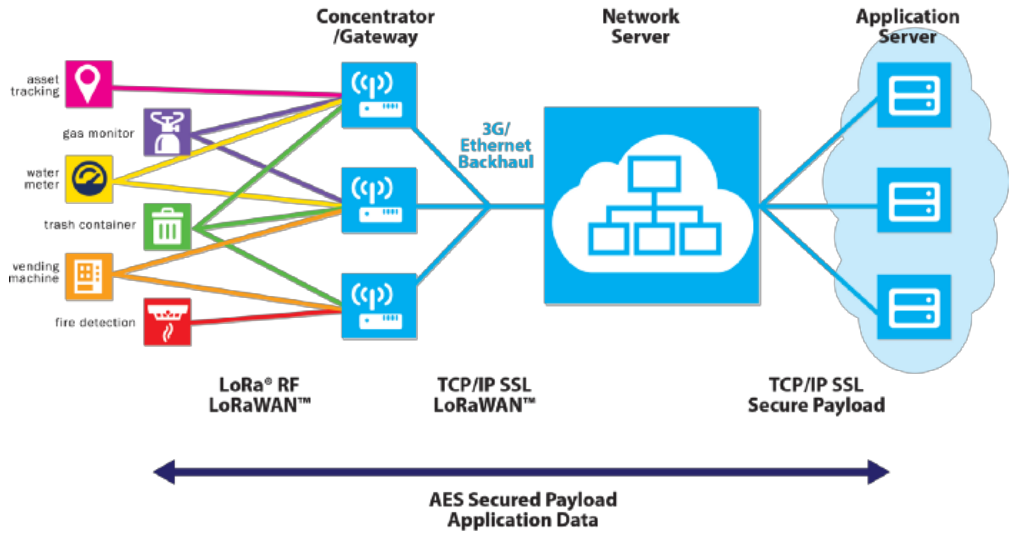


Figura 2.13. Arquitectura de la red LoRaWAN [21]

Está compuesta por:

- **Nodos o dispositivos**. Son aquellos encargados de recolectar la información y enviarla a todas las pasarelas de la red. Pueden ser sensores, u otros dispositivos.
- **Gateway** (**antenas).** Son las estaciones base de LoRa, que actúan de pasarela estableciendo una comunicación bidireccional con los nodos. Toda esta información se envía a un servidor de red.
- **Servidor de red**. Encargado de filtrar toda la información, las tareas de seguridad, control y eliminación de paquetes duplicados. Las puertas de enlace normalmente se conectan a estos servidores mediante el protocolo IP.
- **Servidor de aplicaciones**. Albergan las aplicaciones IoT, mostrando toda la información recopilada por los dispositivos. Está conformada por una interfaz, que es manejada por el servidor red.

A continuación, se expondrá uno de los servidores más empleados para el uso en esta arquitectura.

2.2.1.5 ThingSpeak y The Things Network

Uno de los servidores más utilizados en LoRaWAN es ThingSpeak. Esta plataforma permite recolectar toda la información de los dispositivos IoT y almacenarlos en la nube. Además, cuenta con una API que permite visualizar todos los datos al usuario. Al ser una empresa de Mathworks, ThingSpeak usa Matlab para la representación gráfica. El esquema de comunicación se puede visualizar en la Figura 2.14.

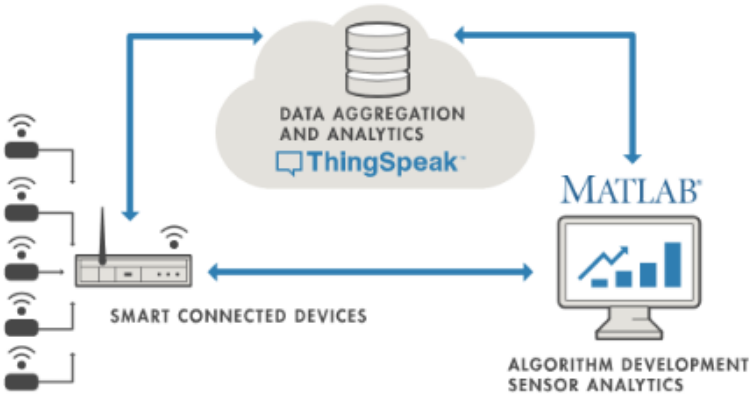


Figura 2.14. Esquema de funcionamiento de ThingSpeak [6]

Una de las principales ventajas de ThingSpeak es que se puede implementar con The Things Network, que es una infraestructura de red pública que permite que cualquier dispositivo IoT mande la información a estas estaciones base basadas en LoRaWAN. Es totalmente gratuito, y hay implementadas miles por todo el mundo.

En España, las principales ciudades cuentan con estas puertas de enlace públicas, como se puede ver en la Figura 2.15

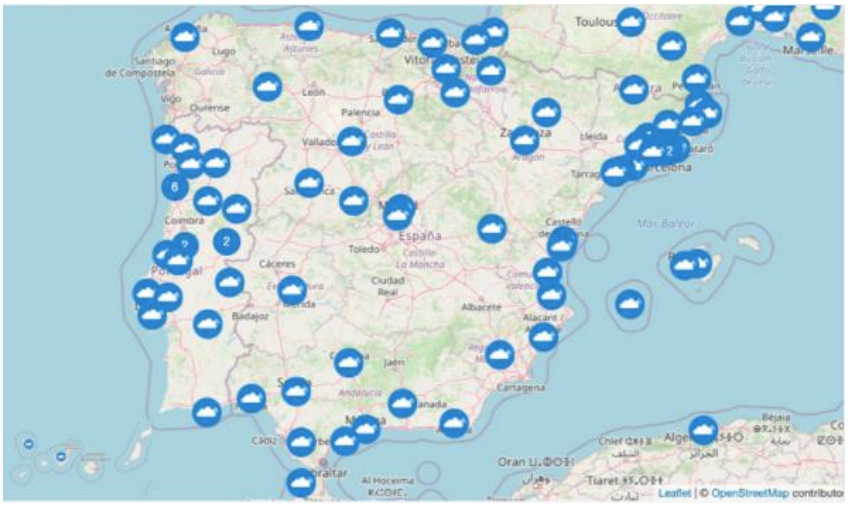


Figura 2.15. Estaciones base LoRaWAN de ámbito público en España

En el caso de la provincia de Jaén, no existe ninguna estación base LoRaWAN. Sería interesante que se fomentara a la instalación de estas por toda la provincia, para que tanto las empresas como particulares puedan desarrollar aplicaciones IoT, y así hacer de Jaén una provincia más tecnológica y aprovechar todas las ventajas que presenta el Internet de las cosas.

Toda la información a cerca de esto y de la instalación de un propio *gateway* se puede encontrar en:

https://www.thethingsindustries.com/docs/gateways/

3 OBJETIVOS

El **objetivo principal** de este proyecto es la creación de una estación meteorológica, que pueda comunicarse inalámbricamente con un servidor web, y mostrar la información climatológica a un usuario. La estación tiene que estar alimentada de una forma autónoma mediante placas solares fotovoltaicas.

Para alcanzar este objetivo principal es necesario cumplimentar una serie de **objetivos secundarios**, que se comentan a continuación:

- Implementación del transmisor y receptor mediante los microcontroladores ESP32.
- Diseño e implementación de sensores meteorológicos de bajo coste.
- Investigación, y uso de la tecnología LoRa para microcontroladores como ESP32.
- Diseño e implementación de un sistema autónomo fotovoltaico.
- Regulación de voltaje del que proporciona la placa solar para que sea compatible con el ESP32.
- Desarrollo de una interfaz de usuario mediante HTML, CSS y JavaScript.
- Creación de un servidor web en el receptor.
- Búsqueda y compra de componentes acordes al proyecto.
- Diseño de la estructura que incorporará todos los componentes de la estación meteorológica.
- Estudio fotovoltaico para incorporar el sistema en un caso real.

4 REQUISITOS

Para la implementación de la estación meteorológica es necesario cumplir una serie de requisitos. Estos se pueden dividir en dos grandes bloques: requisitos *hardware* y *software*.

4.1 Requisitos *hardware*

Los requisitos *hardware* son todos aquellos componentes físicos o materiales, que serán necesarios para la elaboración de la estación meteorológica. En este bloque, se englobará los microcontroladores, antenas, sensores, placas solares y cualquier otro dispositivo que conforme dicho sistema. A continuación, se expondrán todos los elementos que se han empleado [9].

4.1.1 ESP32 WROOM

ESP32 es una familia de microcontroladores de bajo consumo energético, que permiten procesar información de una manera muy eficiente. Además, incorporan WiFi y Bluetooth, dos de las tecnologías inalámbricas más usadas en todo el mundo. Aprovecharemos que estos dispositivos incorporan WiFi para implementar un servidor web, en el receptor, para mostrar toda la información de los sensores de la estación en una página web. [12]

Figura 4.1. ESP32 WROOM de 38 pines

Para la transmisión de datos punto a punto usaremos otra tecnología inalámbrica, que ya se ha comentado anteriormente denominada LoRa, que cuenta con un gran alcance y un consumo energético muy reducido.

Una de las ventajas principales es que al ser de bajo consumo energético se requerirá de una instalación fotovoltaica a pequeña escala, y, por lo tanto, la necesidad de una batería más pequeña. De esta manera, el sistema tendrá una ocupación de espacio muy reducido y su instalación será más sencilla. Así también, se reducirán los costes finales de la elaboración del proyecto.

4.1.2 Ebyte E32 900T20D

El E32 900T20D es un módulo, que cuenta con un chip RF SX1276, el cual incorpora la tecnología LoRa. Este modelo en concreto trabaja en un rango de frecuencias de 862 a 931 MHz, el cual permitirá emitir a la frecuencia de 868 MHz, que es la frecuencia de operación para radioaficionados en Europa cumplimentando así también la Orden IET/787/2013, de 25 de abril, por la que se aprueba el cuadro nacional de atribución de frecuencias.

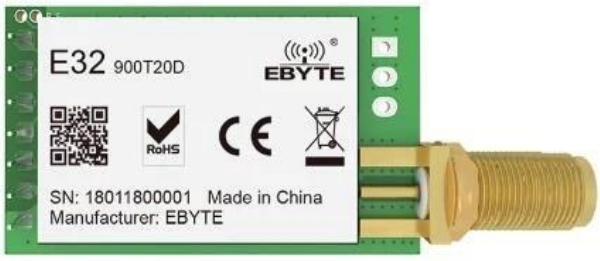


Figura 4.2. Ebyte E32 900T20D

En próximos apartados se expondrá también los diferentes modos de funcionamiento, datos técnicos como la potencia, la antena que se conectará y otra información de importancia.

4.1.3 Antena dipolo omnidireccional 5 dBi 868 MHz

Esta antena se caracteriza principalmente por radiar igual en todas las direcciones, excepto para las direcciones perpendiculares al hilo conductor donde se producen máximos de radiación [17].

Si se busca algo más específico para mejorar la cobertura, se tendría que buscar un modelo más concreto de antena, y quizás con un diagrama de directividad más específico.

Figura 4.3. Antena dipolo 5 dBi

4.1.4 Sensor BME 280

Este sensor está constituido a la vez por 3 sensores de distinta índole. Está formado por un sensor de temperatura, humedad y presión atmosférica. Además, sabiendo la presión atmosférica, se podrá calcular la altitud a la que se encuentra el dispositivo gracias a los cambios de presión. Esto será similar a la función básica de un barómetro, por lo que realmente tiene 4 funciones de la estación meteorológica en un único dispositivo.

Fabricado por la empresa Bosch, permite comunicaciones I2C o SPI con los microcontroladores, que será esencial para el ESP32 en concreto, dado que tiene pines específicos para este tipo de comunicación, como se puede ver en la Figura 4.4.

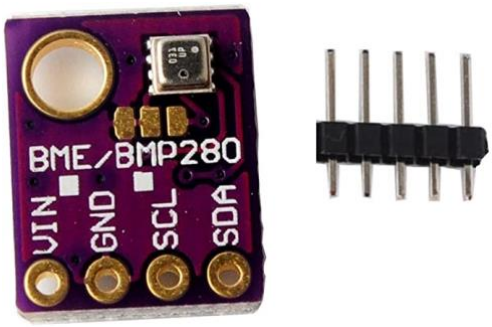


Figura 4.4. Sensor BME 280 (Temperatura, Humedad y Presión Atmosférica)

Además, como un gran punto a favor, su voltaje de operación es de 3,3 [V], que es exactamente el mismo voltaje de operación que tiene el ESP32.

Como conclusión, este sensor es el ideal para una estación meteorológica con ESP32, aunque hay muchas alternativas más, como, por ejemplo, el sensor DHT22 (humedad) o el DS18B20 (temperatura).

4.1.5 Anemómetro

El anemómetro es un sensor que permite medir la velocidad del viento. Para cumplir con los objetivos de bajo coste, se ha optado por diseñarlo desde cero mediante una impresora 3D en vez de comprarlo en una tienda *online*. El precio de venta puede rondar hasta los 100 euros. Se puede visualizar en la Figura 4.5.

El funcionamiento del diseño es el siguiente. Gracias a un motor de 5 [V] en CC (Corriente Continua), se puede estimar a que velocidad giran las aspas del anemómetro en función de la lectura analógica que se obtenga en el ESP32.

Previamente a esto, se tendrá que calibrar mediante algún sistema que permita obtener una lectura adecuada. Por ejemplo, si a 30 kilómetros por hora, la lectura en el ESP32 es de un valor de 2000, ya se podrá estimar todos los demás valores para las distintas velocidades que se produzcan dado que se trata de una relación lineal. El valor máximo del ESP32 es de 4095.

Figura 4.5. Anemómetro

Todo lo relacionado al diseño, fabricación y calibración se expondrá en los apartados de implementación, y planos.

4.1.6 Pluviómetro

En cuanto al último sensor que va a conformar esta estación meteorológica es el pluviómetro. Este se encarga de realizar mediciones en cuanto a las precipitaciones.

Al igual que el anemómetro, no existe un pluviómetro de bajo coste que permita obtener estas mediciones para el microcontrolador ESP32. Por este motivo, se ha decidido diseñar y fabricar uno con una impresora 3D.

El pluviómetro mide con precisión la cantidad de lluvia que ha caído a través del cubo de descarga automática. Tiene un interruptor de lengüeta, que se cierra cada vez que el peso del agua cubre por completo el compartimento e inclina el balancín, generando así un pulso que se puede registrar usando un contador [5].

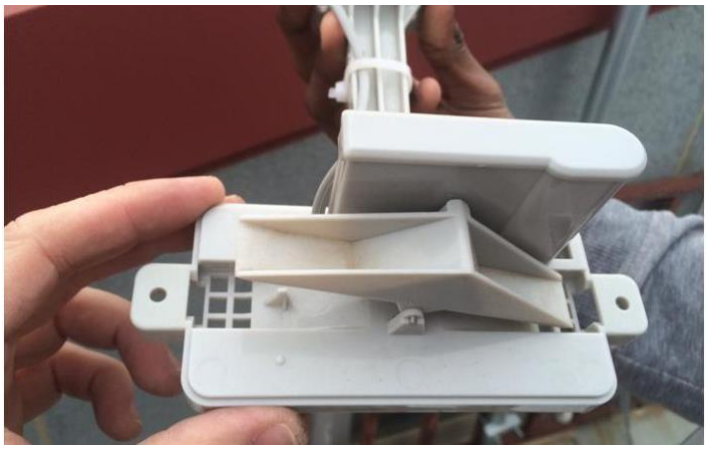


Figura 4.6. Pluviómetro de descarga automática

4.1.7 Sistema de alimentación

El sistema de alimentación será implementado en el transmisor, ya que será el que se situará en un lugar remoto, donde no existe una toma de suministro eléctrico. El receptor será alimentado por el puerto USB del ordenador, y mostrará la información en la *web*.

Este sistema estará formado por distintos componentes que serán expuestos a continuación.

4.1.7.1 Placa solar

Será el elemento principal del sistema fotovoltaico encargado de generar electricidad a partir del sol.

Todos los cálculos referentes a la elección de esta placa se pueden ver en los anexos, en concreto en el anexo 11.5.1 al final de esta memoria, donde justifica los valores escogidos.

Se ha decidido escoger una placa solar de 5 [V], que puede llegar a generar hasta 2 [W] de potencia, o, dicho de otro modo, 400 [mA] en intensidad de corriente.

Hay que conocer a priori la potencia que consume el transmisor mediante un balance de potencias, para escoger una placa que genere más o menos. Aunque siempre, como norma general, es preferible que genere de más, a que de menos.

Figura 4.7. Placa Solar 5[V] 2[W]

Como esta placa genera 5 [V] y el microcontrolador está alimentado por 3,3 [V] será un requisito fundamental regular la tensión a dicho valor. Para ello se empleará un regulador de voltaje que se explicará a continuación.

4.1.7.2 Regulador de tensión MCP1700-3302E

Este regulador de tensión es el ideal para los microcontroladores de la familia ESP, dado que las salidas de tensión que ofrece son de 3,3 [V] tal y como se puede observar en el 11.4.4, en el capítulo de anexos de esta memoria, referente a la hoja de características.

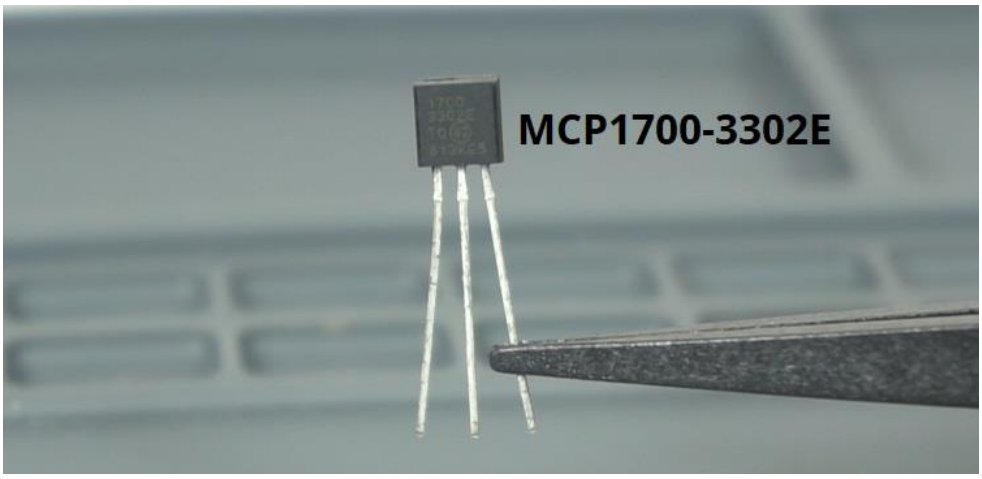


Figura 4.8. MCP1700-3302

Cabe destacar que a la salida de estos reguladores se conectarán dos condensadores en paralelo para controlar los picos de tensión. De esto se hablará más en detalle en el apartado de implementación.

4.1.7.3 Modulador de carga TP4056

El modulador de carga será fundamental para cargar la batería con la energía de la placa solar. Este modelo en concreto es el que se necesitará, dado que es específico para las baterías de litio de 3,6 [V].

Además, ofrece ventajas como que cuando la batería esté totalmente cargada dejará de suministrarle energía eléctrica, por lo que puede llegar a aumentarle la vida útil. También dispone de indicadores leds, que indican cuando la batería está cargada o no.

Y no solo esto, también protegerá el circuito de posibles picos de sobretensión, o cualquier circunstancia que pueda dañar la batería.

En la Figura 4.9 se puede visualizar el modulador de carga.

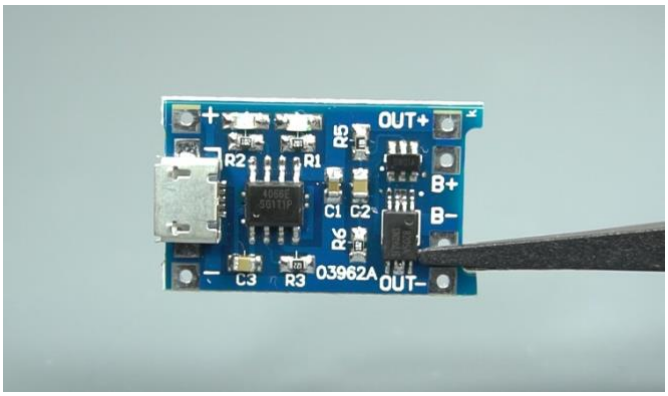


Figura 4.9. Modulador de carga TP4056

4.1.7.4 Batería de Litio de 3,6 Voltios 1300 [mAh]

Será la encargada de almacenar la energía eléctrica de la placa solar. En el apartado 11.5, de cálculos, se especificarán datos de gran relevancia como la autonomía, en función de estudios meteorológicos de la ciudad de Linares, etc. En general, el porque de esta batería.

Se trata de una pila estándar para este tipo de proyectos de bajo consumo.

Figura 4.10. Batería de Litio de 3,6 [V] a 1300 mAH

4.1.7.5 Portapilas

Por último, dentro del sistema eléctrico, será necesario un portapilas. Será el encargado de realizar contacto.

Figura 4.11. Portapilas

4.2 Requisitos *software*

En cuanto a los requisitos *software* serán todos aquellos relacionados con programas y entornos de programación que nos permitan el desarrollo de la estación meteorológica, así como la página web dónde se mostrarán todos los datos de los sensores [9].

4.2.1 Arduino IDE

El *software* de Arduino es el encargado de compilar y cargar los programas en los microcontroladores asociados. El ESP32 es uno de ellos, ya que no cuenta con una interfaz propia para el desarrollo de programas.

El lenguaje de programación, que se emplea es el C. Además, cuenta con una biblioteca de librerías para todo tipo de controladores lo que hace muy sencillo su uso. En la Figura 4.12 se expone dicha interfaz.

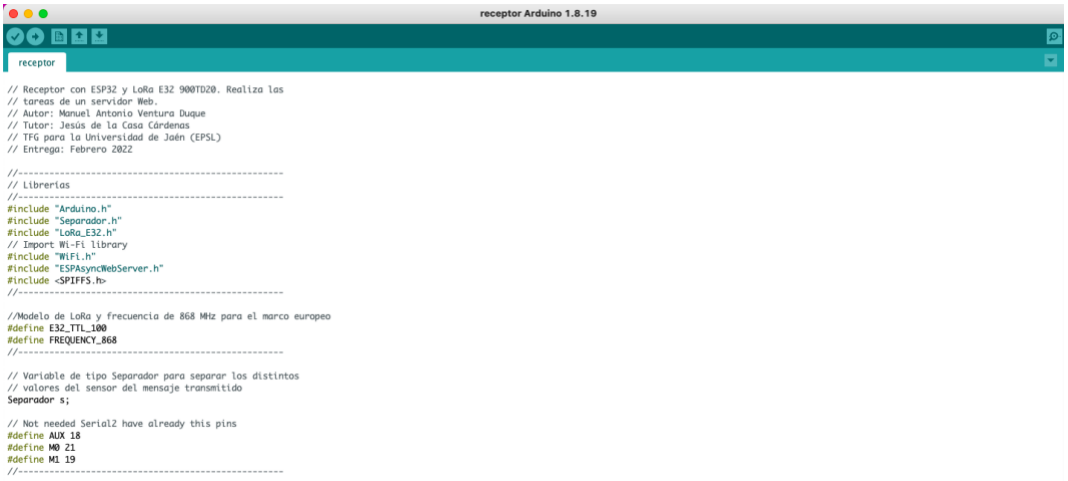


Figura 4.12. Interfaz de Arduino IDE

En este proyecto se usarán librerías como:

- **Wifi.h**: para la creación de un servidor web en el receptor.
- **LoRa_E32.h**: para la configuración de los dispositivos LoRa.
- **SPIFFS.h**: para el almacenamiento de ficheros en la memoria *flash* ESP32.

Se han expuesto las más importantes. En el capítulo 5 de implementación se detallará en más profundidad.

4.2.2 Visual Studio Code

Visual Studio Code es un editor de código fuente, que nos permite programar todo tipo de aplicaciones en múltiples lenguajes de programación. Es totalmente gratuito, y

presenta varias ventajas frente a otro tipo de editores. Las más destacadas son las siguientes [20]:

- Es un *software* multiplataforma, es decir, puede trabajar tanto en Windows, Linux o Mac OS.
- Se pueden abrir diversas terminales dentro de la interfaz, lo que permite interactuar con los distintos proyectos que están desplegados y ejecutados.
- Es gratuito.
- Permite instalar extensiones dedicadas a distintos lenguajes de programación, que incluyen servicios, temas, etc. De esta manera facilita la tarea a los programadores.

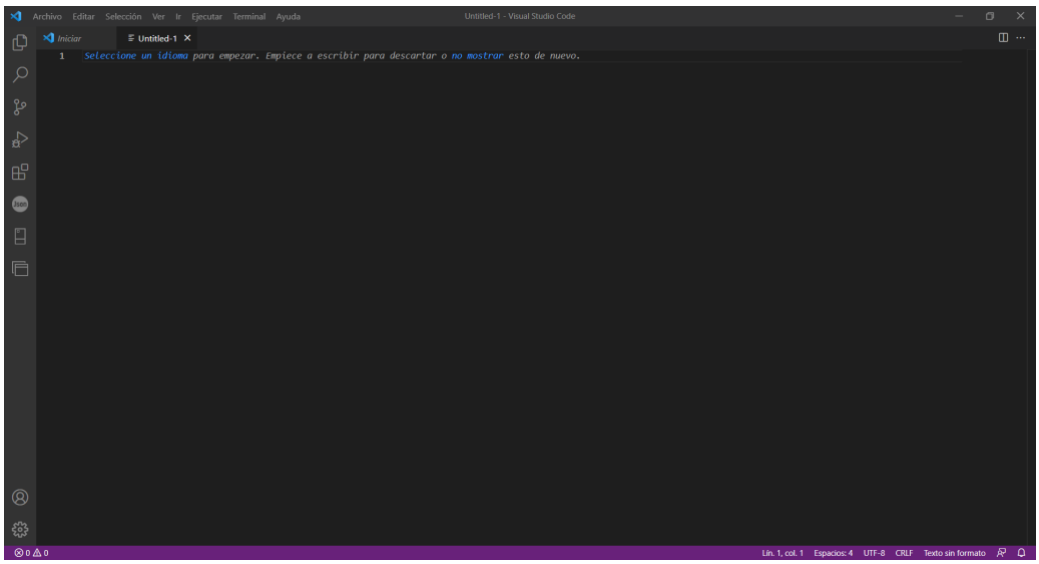


Figura 4.13. Interfaz Visual Studio Code

5 IMPLEMENTACIÓN DEL SISTEMA

Una vez se han expuesto todos los requisitos, que se necesitarán para la implementación de la estación meteorológica, se va a desarrollar, en detalle, las distintas partes que la conforman.

Se puede componer en 2 grandes bloques principales, transmisor y receptor, donde cada uno de ellos cuenta con sus particularidades.

5.1 Transmisor

El transmisor será el encargado de emitir la información recolectada por los sensores. Estará conectado al suministro eléctrico proporcionado por la placa solar y la batería, por lo que, en este bloque, se explicará todo lo relacionado con el diseño y desarrollo de este sistema eléctrico autónomo fotovoltaico.

5.1.1 Montaje del transmisor

Antes de comenzar con la configuración y programación del transmisor, se tendrá que determinar todas las conexiones de E/S (Entrada y Salida) por parte de los componentes a interconectar. En este caso habrá 4 dispositivos, que se enuncian a continuación:

- Sensor BME 280, que incluye la temperatura, humedad relativa, altitud y presión atmosférica.
- Módulo LoRa E32, que será el encargado de transmitir la información.
- Anemómetro.
- Pluviómetro.

El sensor BME 280, anemómetro y pluviómetro recolectarán datos analógicos, por lo que se comportan como entrada analógica en el ESP32.

Cabe destacar que el BME 280 presenta una diferencia con respecto a los demás sensores. Este emplea una comunicación I2C, así que se tendrá que conectar los pines de este sensor con los correspondientes a este tipo de comunicaciones en el ESP32.

I2C se rige fundamentalmente por una comunicación maestro-esclavo. Es una de las más famosas en aplicaciones IoT, ya que un único controlador (maestro) puede controlar múltiples sensores (esclavos) [11].

En la Figura 5.1 del *pinout* del BME se pueden ver principalmente 4 pines.

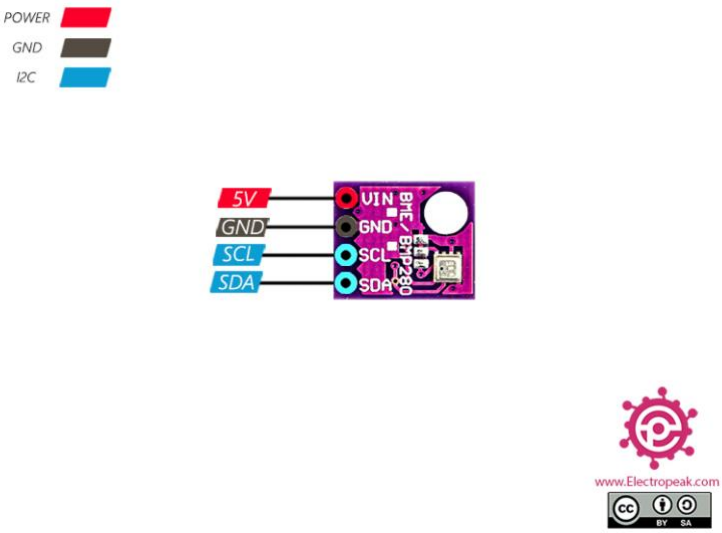


Figura 5.1. Pines del BME 280

- **VIN**. Es el pin donde se le suministrará el voltaje. Aunque en la Figura 5.1 ponga 5 [V] en VIN, a 3,3 [V] también es válido, ya que este valor se encuentra dentro del rango de funcionamiento. Esto se puede comprobar en la hoja de características, que se ha adjuntado en el anexo 11.4.3.
- **GND**. Se corresponde con masa.
- **SCL**. Es uno de los pines del protocolo I2C. Conlleva el pulso de la señal de reloj.
- **SDA**. El otro de los pines del protocolo I2C. Establece la comunicación de datos entre maestro-esclavo.

Una vez se han conocido los pines del BME, se seguirá detallando los pines de los demás dispositivos.

Ahora toca el turno del LoRa E32 900T20D. Según la hoja de características, como se puede ver en el anexo 11.4.2, se compone fundamentalmente de 7 pines. En la tabla de a continuación se recogen el funcionamiento de cada uno.

Pin	Nombre	E/S	Descripción
1	M0	Entrada	Permite escoger el modo de funcionamiento
2	M1	Entrada	Permite escoger el modo de funcionamiento
3	RXD	Entrada	Entrada TTL UART. Se conecta a RXD del micro
4	TXD	Salida	Salida TTL UART. Se conecta a TXD del micro
5	AUX	Salida	Verifica si los datos se envían o reciben
6	VCC	Entrada	Alimentación

7	GND	Entrada	Masa

Tabla 5.1. Definición de los pines del LoRa E32 900T20D

Por último, quedaría el anemómetro y el pluviómetro. Estos se corresponden con entradas analógicas al ESP32 como se ha comentado antes, por lo que se tendría que buscar entradas de este tipo.

Una vez se han estudiado todos los pines, toca analizar los pines de entrada y salida del microcontrolador. Para ello, será necesario tener la hoja de características para poder visualizar toda la información de los pines, así como del funcionamiento.

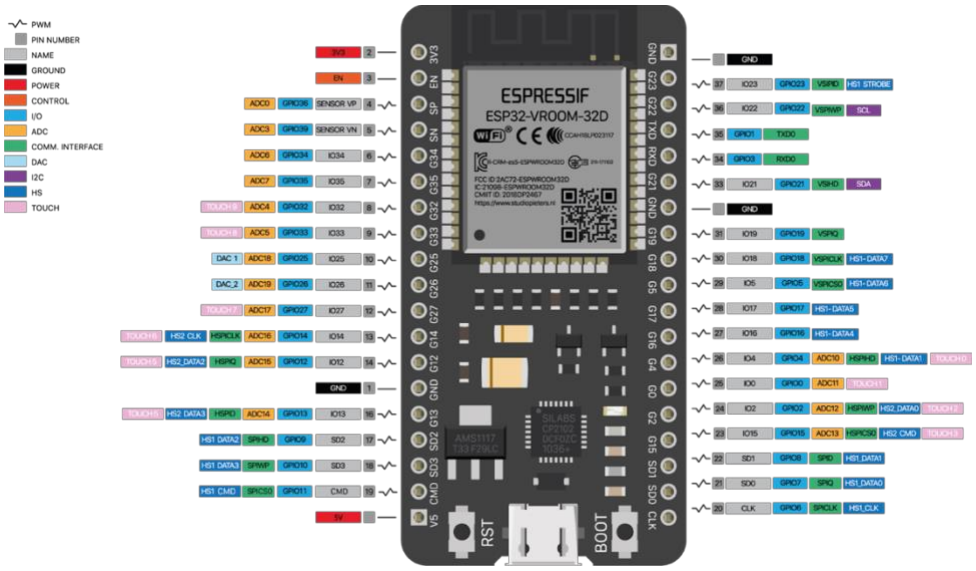


Figura 5.2. Pinout del ESP32 WROOM 32

En la Figura 5.2, se pueden observar todos los pines del ESP32, así como las características principales y a que grupo pertenecen.

Para el BME 280 se conectarán los pines SCL y SDA a los pines **G22** y **G21** que son los dedicados para el protocolo I2C.

En cuanto a los pines para el LoRa E32, la distribución será la siguiente:

- M0 → **G4.**
- M1 → **G19.**
- RXD → **G17.**
- TXD → **G16.**
- AUX → **G18.**

Los pines GPIO funcionan tanto para la entrada, como para la salida. Cualquiera de estos pines valdría para conectar los pines del módulo LoRa.

Como se puede comprobar en la Tabla 5.1, el pin **RXD** del LoRa tiene que ir conectado al **TXD** del ESP32, y viceversa. Por lo que estos pines se corresponden con los **G17** y **G16** tal y como se han establecido.

A continuación, en la Figura 5.3, se puede visualizar el montaje **a falta de la conexión del sistema eléctrico fotovoltaico**.

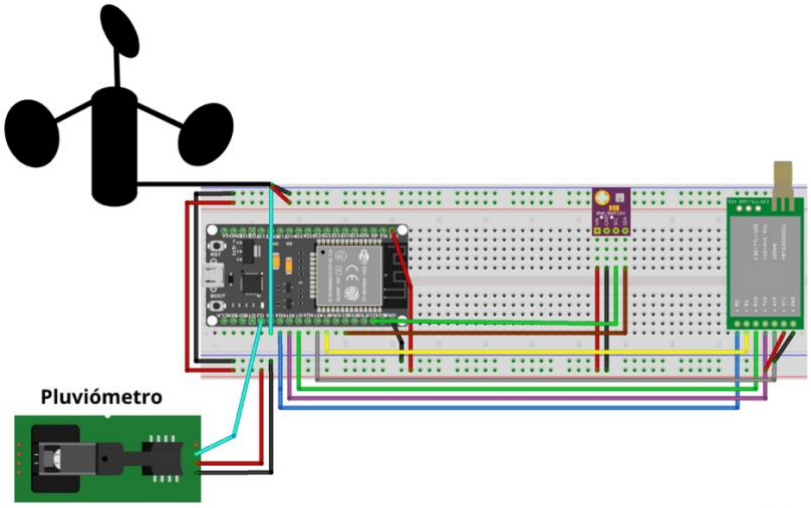


Figura 5.3.Transmisor a falta del sistema eléctrico fotovoltaico

5.1.2 Instalación y configuración de los drivers

En primer lugar, para la implementación del transmisor, se procede a la configuración del ESP32 en la interfaz de Arduino. Se instalarán todos los *drivers* necesarios, librerías, compilación y programación del transmisor.

Una vez se ha conectado el microcontrolador al ordenador inmediatamente se procederá a instalar los drivers. Si se ha conectado por primera vez, debería de instalarse de una manera automática.

En algunos casos, el ordenador no puede instalar los drivers, por lo que la interfaz de Arduino no podrá leer el microcontrolador.

Existe una solución a este problema, y es instalar los drivers de una manera manual. Para ello, se entrará en la web del fabricante, y se procederá a descargarlos desde el apartado de recursos y drivers de todos los dispositivos. Para este caso en concreto, del modelo ESP32-WROOM-32 de 38 pines el enlace será el siguiente:

https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers

Para su descarga, será tan sencillo como dirigirse al apartado de "*downloads*" y descargarse los drivers dependiendo del sistema operativo del ordenador en concreto.

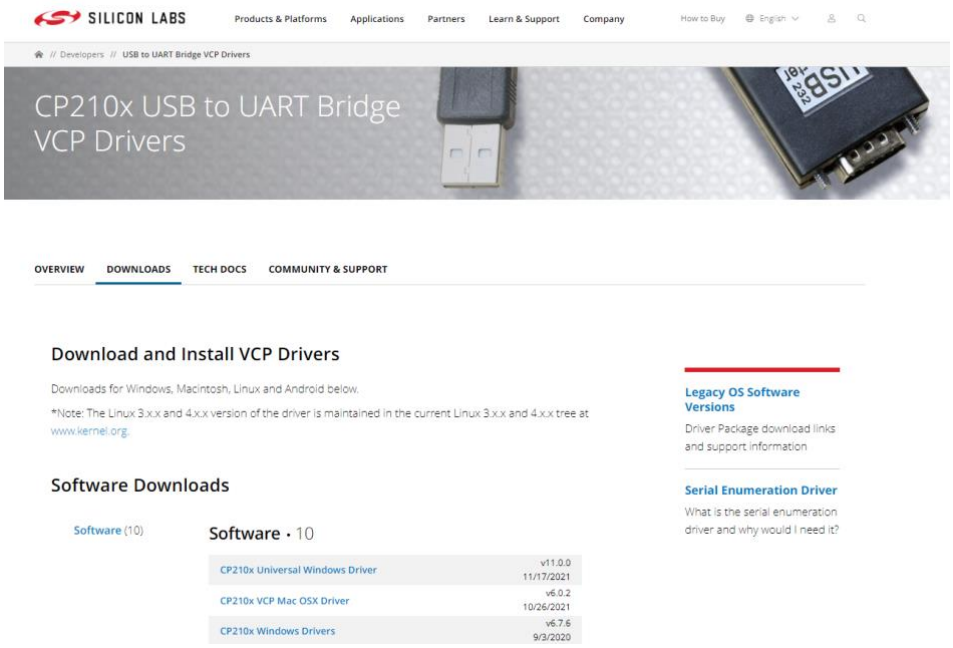


Figura 5.4. Plataforma de drivers para el ESP32 en diferentes S.O.

Por último, se seguirán los pasos de instalación típicos, como cualquier otro programa. Una vez instalados, tendrá que aparecer algo similar a lo que aparece en la Figura 5.5 en el administrador de dispositivos (en este caso en Windows).

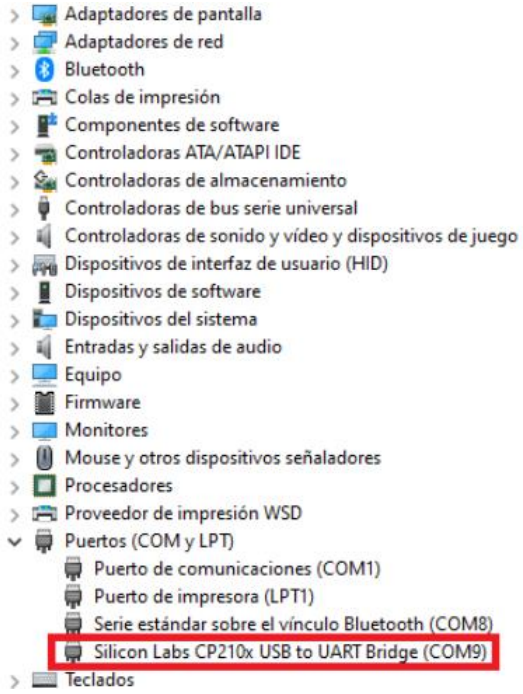


Figura 5.5. Administrador de dispositivos de Windows

A continuación, para empezar a programar con el microcontrolador es necesario instalar las tarjetas de ESP32, desde el gestor de URLs externas de Arduino IDE. Para ello, se seguirán los siguientes pasos:

1. Iniciar Arduino IDE, y dirigirse a **Archivo** → **Preferencias**. Aparecerá una ventana con un campo denominado Gestor de URLs Adicionales de Tarjetas.
2. Añadir la siguiente URL: https://dl.espressif.com/dl/package_esp32_index.json. Contiene un fichero con formato JSON con todas las placas ESP32, que están subidas a la plataforma de GitHub de la empresa Espressif.

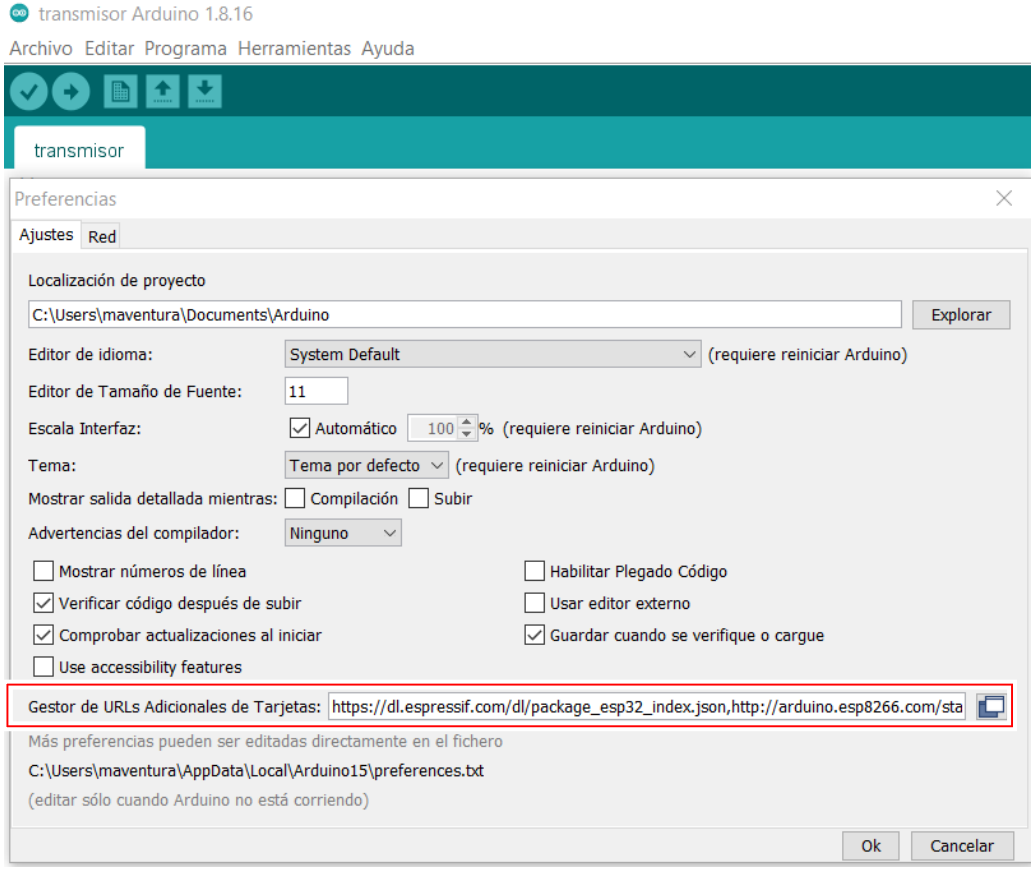


Figura 5.6. Añadir fichero JSON de los modelos ESP32 a Arduino IDE

3. En el paso anterior solamente se han añadido los distintos modelos, pero no ha instalado ninguna placa en la interfaz. Para proceder a la instalación habrá que dirigirse a **Herramientas** → **Placa** → **Gestor de Tarjetas**. Aparecerá una ventana, donde se tendrá que buscar las ESP32, e instalarlas.

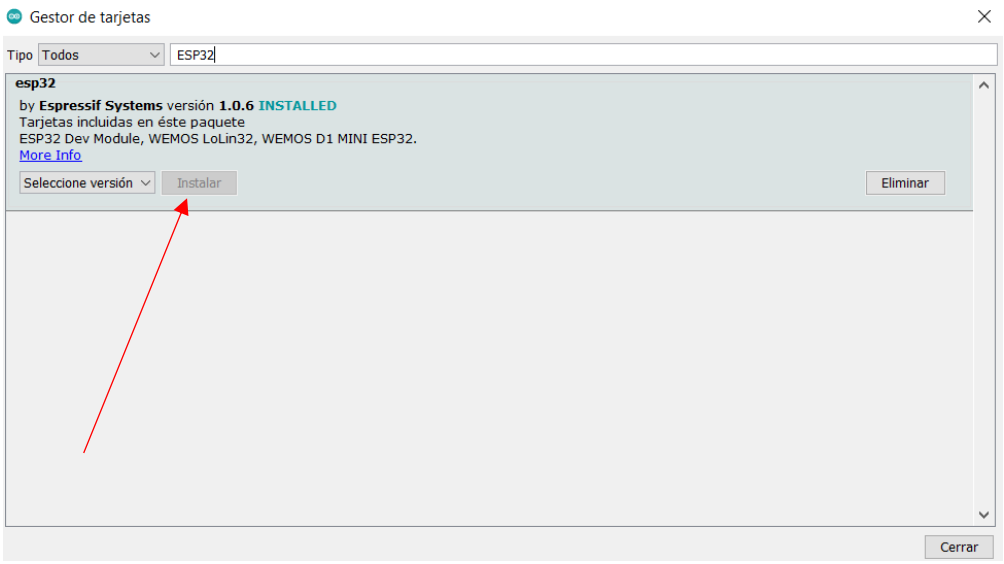


Figura 5.7. Instalación de las tarjetas ESP32 con los distintos modelos

Una vez instaladas las tarjetas ya estará el software listo para la programación del transmisor. Simplemente se seleccionará el modelo en concreto, que será la ESP32 Dev Module, en el apartado de **Herramientas → Placa.**

5.1.3 Programación del transmisor

Ahora que todo el *software* está configurado, se pasará a la parte de la programación del transmisor. Se puede descomponer en 2 partes fundamentales:

1. Programación del LoRa E32.
2. Programación de los sensores.

5.1.3.1 Programación del LoRa E32 900T20D

Programar este módulo no es una tarea sencilla, ya que la documentación técnica oficial es muy escueta y reducida. Además, para poder configurar estos dispositivos normalmente se tienen que emplear librerías externas de programadores particulares, lo que supone, que muchas de ellas no están preparadas para todos los modelos, o puede ocasionar múltiples errores, o incluso que no funcione.

En el caso del modelo usado en este trabajo final de carrera es uno de los recién sacados a mercado. Su uso está pensado para la zona geográfica europea a la frecuencia de los 868 MHz correspondiente a la banda ISM.

En primer lugar, se definirán las librerías, que se van a necesitar para la implementación del transmisor.

```
// Transmisor con ESP32 y LoRa E32 900TD20
// Autor: Manuel Antonio Ventura Duque
// Tutor: Jesús de la Casa Cárdenas
// TFG para la Universidad de Jaén (EPSL)
// Entrega: Febrero 2022

//------------------------------------------------
// Librerías
//------------------------------------------------
#include "Arduino.h"
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#include "LoRa_E32.h"
//------------------------------------------------
```

Figura 5.8. Librerías requeridas para el transmisor

Como se puede observar en la Figura 5.8 se van a requerir 5 librerías:

- **Arduino.h:** Librería estándar, que contiene las funciones más principales y básicas para el uso de microcontroladores en esta interfaz.
- **Wire.h:** Permite las comunicaciones mediante el protocolo I2C. Incluye funciones de configuración, lectura y escritura.
- **Adafruit_Sensor.h:** Proporciona una capa de abstracción, es decir, a partir de esta librería se pueden integrar otros sensores.
- **Adafruit_BME280.h:** Gracias a esta librería, que implementa la interfaz con funciones de escritura y lectura, podemos obtener los valores de temperatura, humedad y presión del sensor BME280 en Arduino IDE.
- **LoRa_E32.h:** Implementa todas las funciones necesarias para la configuración, transmisión y recepción de los datos con el módulo LoRa E32. Aunque hay otras, como LoRa.h, esta es específica para el modelo 900T20D.

Estas librerías se pueden encontrar en el siguiente enlace, donde se han almacenado en una carpeta de Drive:

https://drive.google.com/drive/folders/1ScCFrB-rpx5pB8IPGVmaRKgBrZcPBAsc?usp=sharing

En la Figura 5.9 se pueden visualizar las librerías.

Figura 5.9. Librerías para el transmisor subidas a Google Drive

En segundo lugar, se definirá la frecuencia, modelo de LoRa y los pines que se escogieron en el apartado 5.1.1 de este proyecto, tal y como se puede ver en la Figura 5.10.

```
//Modelo de LoRa y frecuencia de 868 MHz para el marco europeo
#define E32_TTL_100
#define FREQUENCY_868
//-----------------------------------------------------

//Definición de los pines
#define AUX 18
#define M0 4
#define M1 19
//-----------------------------------------------------
```

Figura 5.10. Definición del modelo, frecuencia y pines a usar

Antes de pasar a las funciones *setup*, y *loop*, que son las más conocidas en los proyectos de Arduino, se comentará una de las funciones más importantes del LoRa. Esta se le ha llamado nodeConfig, y se encarga de configurar el módulo para la transmisión.

Cabe destacar, que estos dispositivos permiten configurarse en distintos modos de transmisión. Existen varios tipos, que se detallarán a continuación, y se escogerá el más específico para este proyecto.

- **Transmisión fija punto a punto**: solo puede recibir la información un dispositivo que tenga una dirección baja, alta y canal predeterminados. En la Figura 5.11 se puede visualizar un esquema de dicha transmisión.

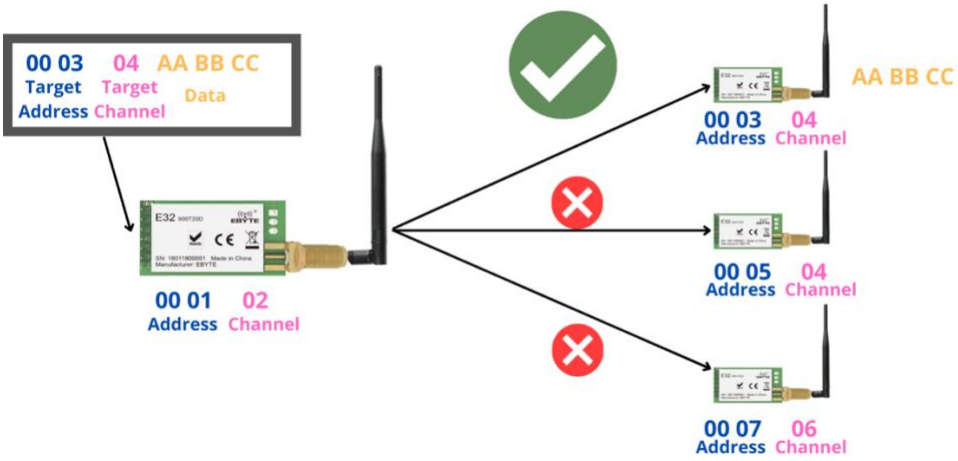


Figura 5.11. Transmisión fija punto a punto

- **Transmisión fija de difusión:** la información se transmite a todos los dispositivos que sintonicen con el canal predeterminado, no importa la dirección del receptor, es decir, solo importa el canal. En la Figura 5.12 se puede visualizar un esquema de dicha transmisión.

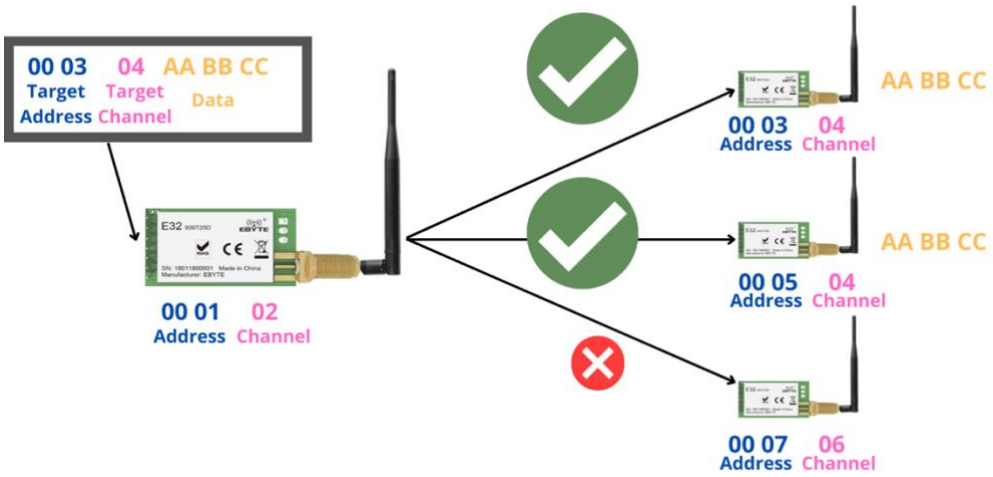


Figura 5.12. Transmisión fija de difusión

- **Transmisión transparente:** se suele emplear para realizar demostraciones, y puede enviar información a todos los dispositivos del mismo canal y dirección.

Una vez se han detallado los distintos tipos de transmisiones, la que más sentido tiene para la estación meteorológica es la fija punto a punto. Solo va a haber un receptor LoRa, por lo que esta será la escogida.

```
//Configuración del nodo LoRa
 void nodeConfig() {
  ResponseStructContainer c;
  c = e32ttl.getConfiguration();
  Configuration configuration =  *(Configuration*) c.data;
  configuration.ADDH = 0x00;
  configuration.ADDL = 0x1;
  configuration.CHAN = 0x10;
  configuration.SPED.airDataRate = AIR_DATA_RATE_100_96;
  configuration.SPED.uartBaudRate = UART_BPS_9600;
  configuration.SPED.uartParity = MODE_00_8N1;
  configuration.OPTION.fec = FEC_1_ON;
  configuration.OPTION.fixedTransmission = FT_FIXED_TRANSMISSION;
  configuration.OPTION.ioDriveMode = IO_D_MODE_PUSH_PULLS_PULL_UPS;
  configuration.OPTION.transmissionPower = POWER_20;
  configuration.OPTION.wirelessWakeupTime = WAKE_UP_1250;
// Set configuration changed and set to not hold the configuration
  ResponseStatus rs = e32ttl.setConfiguration(configuration, WRITE_CFG_PWR_DWN_SAVE);
  Serial.print("Config: ");
  Serial.println(rs.getResponseDescription());
}
//-------------------------------------------------
```

Figura 5.13. Función nodeConfig() que configura el dispositivo LoRa E32

Como se puede ver en la Figura 5.13, se ha predeterminado una dirección baja, alta y un canal. Posteriormente, en el receptor, habrá que configurarlo de la misma manera para que reciba la información (transmisión fija).

Los parámetros SPED, de esta función, definen la velocidad de la transmisión. Hay 3 tipos:

- **airDataRate**: es la tasa de datos aéreos. Tiene que ser la misma en ambos lados, y cuanto mayor sea peor será la comunicación, y mayores pueden ser las interferencias. AIR_DATA_RATE_100_96 da un valor de 9600 baudios.
- **uartBaudRate**: velocidad en baudios para el Serial. Se establece un valor de 9600 baudios.
- **uartParity**: se trata del bit de paridad. Se establece el modo predeterminado.

Posteriormente, se establecerán las opciones. Se han establecido 5 opciones principales:

- **fec**: corrección de errores hacia delante activada.
- **fixedTransmission**: se establece el modo de transmisión fijo con FT_FIXED_TRANSMISSION.
- **ioDriveMode**: empleado para establecer las resistencias pull-up a RXD, y push-pull a TXD. Esto evita falsos estados (HIGH y LOW) producido por el ruido, en los pines.
- **transmissionPower**: establece la transmisión de potencia. POWER_20 se refiere a una potencia de 20 dBm, que es la máxima para este modelo, como se puede ver en la hoja de características en el anexo 11.4.2

- **wirelessWakeupTime**: es el tiempo de activación inalámbrica. En este caso puede ser cualquier valor, ya que el LoRa trabajará en el modo 0, con M0 y M1 en nivel bajo. No importa este campo, aunque se define

Una vez se ha configurado, se establecerán todos los valores mediante la función setConfiguration.

En la función setup será donde se inicialice la función de configuración para el LoRa E32.

```
 void setup() {
//Inicio comunicacion serie con monitor
  Serial.begin(9600);
  delay(500);
  bool status;
  status = bme.begin(0x76);

    if (!status) {
    Serial.println("Could not find a valid BME280 sensor, check wiring!");
    while(1);
  }

  e32ttl.begin();

  //Configuración del nodo LoRa E32 900T20D
  nodeConfig();

}
//----------------------------------------------------
```

Figura 5.14. Función Setup en el transmisor

En cuanto a la función loop, será la encargada de transmitir la información cada cierto tiempo. Se ha propuesto enviar tramas con la información cada 3 segundos. Cabe destacar que estas tramas no pueden superar los **58 bytes**, por lo que hay que tener en cuenta la información que se manda.

En este proyecto, después de realizar una gran cantidad de pruebas en transmisión, se comprobó que no se llegó nunca a ese valor, por lo que no fue necesario realizar una subdivisión de la información.

```
void loop() {

   temperature,pressure,altitude,humidity = valoresensor();
   trama = temperature + "," + pressure+ "," +  altitude+ "," +  humidity + "," +
           wind + "," + rain
   ResponseStatus rs = e32ttl.sendFixedMessage(0x0, 0x3, 0x10, trama);
   Serial.println("Sending " + trama);
   Serial.println( rs.getResponseDescription());
   delay(3000);
}
//----------------------------------------------------
```

Figura 5.15. Función loop del transmisor

Mediante la función sendFixedMessage se envía la información que contiene la trama. Como se puede ver en la Figura 5.15 la trama tiene las direcciones que antes se habían establecido, junto al mismo canal.

5.1.3.2 Programación de los sensores

En cuanto a la programación de los sensores se emplearán las librerías, y se creará una función para cada actuador.

En el caso del sensor BME, que puede extraer la temperatura, humedad y presión, se ha desarrollado la siguiente función:

```
// Función que extrae todos los valores del sensor BMP 280 y los devuelve.
 String valoresensor(){

  temperature = bme.readTemperature();
  Serial.print("Temperature = ");
  Serial.print(temperature);
  Serial.println(" *C");

  pressure = (bme.readPressure() / 100.0F);
  Serial.print("Pressure = ");
  Serial.print(pressure);
  Serial.println(" hPa");

  altitude = bme.readAltitude(SEALEVELPRESSURE_HPA);
  Serial.print("Approx. Altitude = ");
  Serial.print(altitude);
  Serial.println(" m");

  humidity = bme.readHumidity();
  Serial.print("Humidity = ");
  Serial.print(humidity);
  Serial.println(" %");

  wind = analogRead(25);
  wind = wind*30/2000;
  Serial.print("Viento = ");
  Serial.print(wind);
  Serial.println(" %");

  rain = analogRead(26);
  if(rain>0 && rain<4095){
    rain_aux = rain_aux+10;
  }
  Serial.print("Precip. = ");
  Serial.print(rain);
  Serial.println(" %");
```

Figura 5.16. Función que extrae los valores de temperatura, humedad, presión, altitud, viento y precipitaciones del transmisor

Esta función, con la variable “bme”, usa las funciones o métodos de la librería **Adafruit_BME280.h** para extraer los valores que se leen.

Hay una particularidad en la función “readAltitude”, y es que se obtiene la altitud respecto al nivel del mar, y no a una altura desde el suelo partiendo de referencia de los 0 metros de este.

Estos valores son almacenados en la trama, como aparece en la Figura 5.15, y se envían para que el receptor los pueda recuperar.

En cuanto al anemómetro su programación se basa en una lectura analógica en cualquiera de las entradas configuradas para esto.

Para conocer la velocidad del viento habrá que calibrar este dispositivo, es decir, que cuando el motor ofrezca una tensión al microcontrolador, y se conoce la velocidad del viento, se podrá calcular el resto de valores de velocidad para los distintos valores de voltaje.

Por ejemplo, si el microcontrolador lee que el anemómetro está generando 3 [V] y hace un viento de [20 km/h], se podrá calcular el resto de valores con una simple regla de tres.

Figura 5.17. Anemómetro

Por último, queda el pluviómetro. El funcionamiento es muy sencillo. Se trata de una lengüeta con dos cavidades de 10 [mL] de capacidad. Cada vez que se llena una cavidad, esta cae por gravedad y genera una variación electromagnética, ya que está implementado con un sensor Hall, que permite un conteo al ESP32. El agua, cae hacia un depósito que a su vez libera el agua a la tierra.

Por lo que cada pulso será leído por el microcontrolador por otra entrada analógica. Las variaciones se leen, donde cada variación son 10 mL de agua.

Figura 5.18. Base del pluviómetro

En la Figura 5.16 se puede observar dicha programación.

5.1.4 Instalación del sistema eléctrico fotovoltaico

Una vez se ha comprobado el funcionamiento del transmisor, y se ha comprobado que ha sido satisfactorio, se procederá a instalar el sistema eléctrico fotovoltaico.

Este sistema solo se instalará en el transmisor, ya que el receptor no precisa de una fuente autónoma debido a que este suministro se lo proporcionará un ordenador mediante el puerto USB.

En primer lugar, se tendrá que escoger un panel fotovoltaico. Este panel tiene que suministrar potencia suficiente para hacer funcionar al ESP32, al LoRa E32 y todos los dispositivos que consuman potencia.

El panel que se ha escogido ha sido uno específico a los cálculos del anexo 11.5.1. En este caso, es capaz de generar hasta los 5 [V] y en términos de potencia hasta los 2 [W], siempre y cuando la placa esté en contacto directo con el sol, y con la inclinación óptima a sus rayos.

Este panel será instalado en un modulador de carga TP4056, que será el encargado de cargar una batería de litio de 3,6 [V]. Una de las principales ventajas que cuenta este dispositivo de carga, es que dispone de un circuito de protección, que protege a la batería en caso de un sobrevoltaje. Además, cuando la pila esté totalmente cargada se encenderá un led verde y mientras se está cargando se encenderá un led rojo.

Aquí existe un problema a solucionar, y es que debido a que el ESP32 funciona a 3,3 [V] se precisará de un sistema de regulación de voltaje que permita tener un nivel de tensión continúa adecuado para el microcontrolador. La batería, en condiciones de carga óptima, llega a emitir hasta los 4 [V], por lo que este sistema será fundamental para no estropear el ESP32 y aumentar su vida útil.

En la Figura 5.19 se puede observar el nivel voltaje de una batería aproximadamente al 90% de su carga.

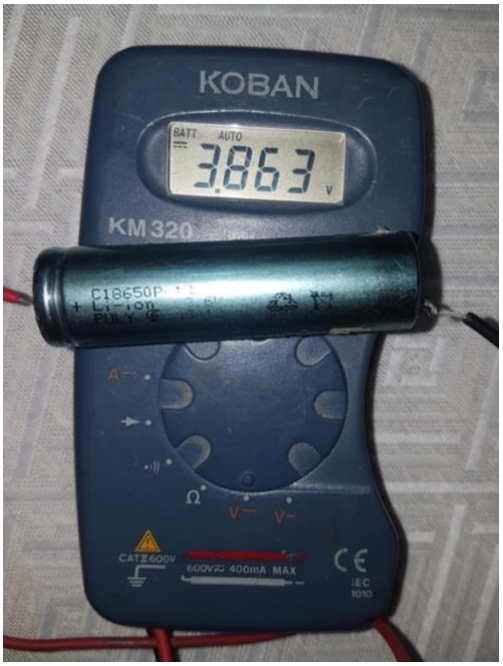


Figura 5.19. Voltaje de una batería de litio aproximadamente al 90% de su carga

Para la regulación del voltaje se empleará un MCP1700-3302E. Se trata de un LDO (Regulador de Baja Caída), que, como se puede apreciar en el anexo 11.4.4, presenta como salida estándar los 3,3 [V]. Este modelo es ideal para el microcontrolador, por lo que diseñando un buen circuito se podrá disponer de un nivel de alimentación ideal.

Por último, cabe destacar que se instalarán, a la salida del LDO, dos condensadores en paralelo para suavizar los picos de voltaje. Uno de ellos será de 100 uF y otro de 100 nF.

En la Figura 5.20 se puede ver el resultado final del sistema eléctrico fotovoltaico. Se consigue una salida en el LDO de 3,26 [V], que es suficiente para hacer funcionar el ESP32, ya que entra dentro del rango de funcionamiento tal y como se puede ver en su hoja de características.

En cuanto a la Figura 5.21se puede ver como se enciende el led rojo del modulador de carga, con la luz que suministra la linterna de un teléfono móvil, indicando que se está cargando la pila. Estas placas responden adecuadamente ante los cambios de luz, sobre todo si se trata de la luz solar.

Figura 5.20. Montaje final del sistema eléctrico fotovoltaico

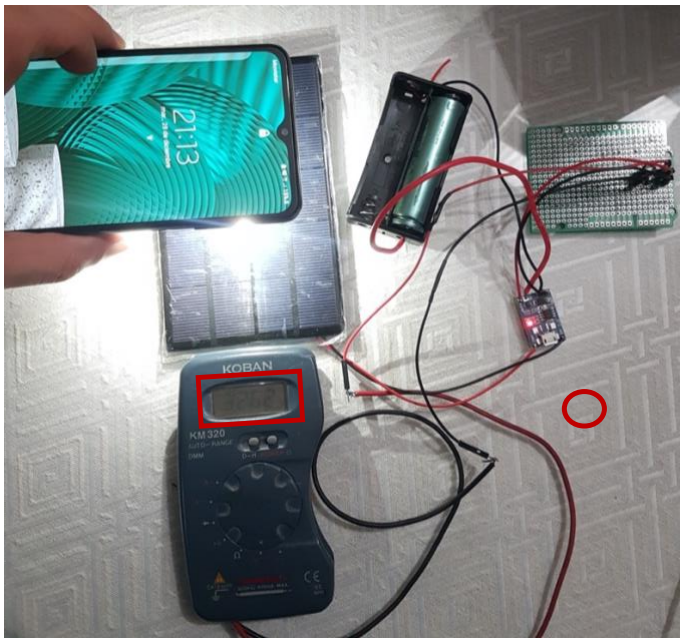


Figura 5.21. Prueba de funcionamiento del sistema fotovoltaico cargando la batería con un móvil

Visto desde un esquema de conexionado, el montaje queda como lo que se puede observar en la Figura 5.22.

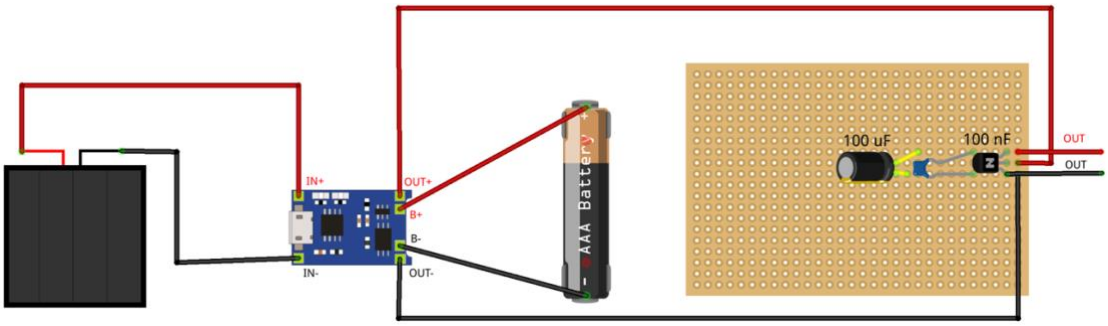


Figura 5.22. Esquema de conexionado del sistema eléctrico fotovoltaico

El esquema final del transmisor queda, por lo tanto, queda como la Figura 5.23.

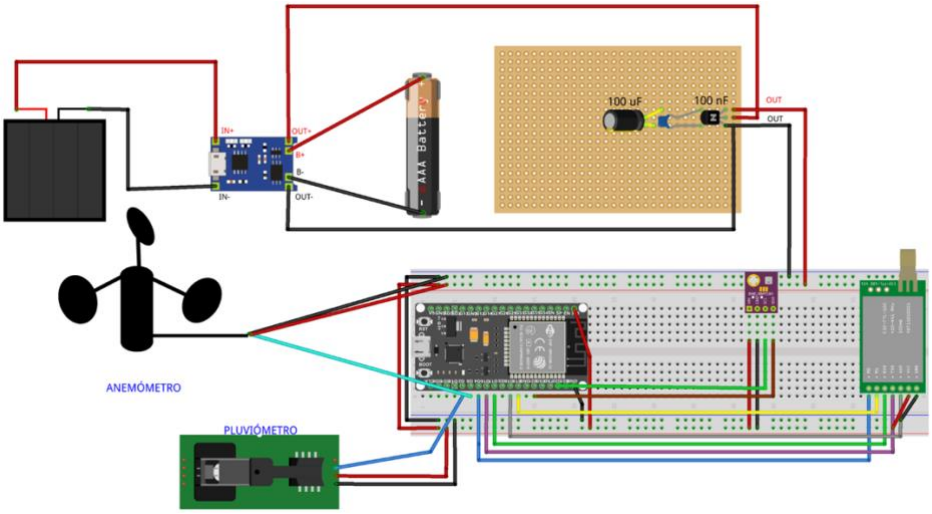


Figura 5.23. Transmisor conectado con el sistema de alimentación

5.2 Receptor

El receptor será el encargado de recibir y procesar las tramas enviadas por el transmisor. Además, toda la información que recibe de los sensores la mostrará al usuario final mediante un servidor web. Este servidor es creado por el ESP32, y nos dará una dirección IP a la que el usuario puede conectarse siempre y cuando se encuentre conectado en la misma red Wifi.

En el servidor se mostrará la página web desarrollada en HTML, CSS y JavaScript. Para ello estos ficheros se almacenarán en un sistema de archivos, que presenta el ESP32 denominada SPIFFS. Se tienen que cumplir una serie de directrices para que el microcontrolador pueda leer a la perfección estos ficheros, así como todas las imágenes que se requieran. Todo esto se expondrá más detalladamente en este apartado.

5.2.1 Montaje del receptor

En cuanto al montaje del receptor, se tendrá que decidir en que pines del microcontrolador se conectará el módulo LoRa E32. No hay más dispositivos a conectar, por lo que el montaje resultará muy sencillo.

En este caso los pines que se han escogido para el módulo LoRa E32 han sido los siguientes:

- M0 → **G21.**
- M1 → **G19.**
- RXD → **G17.**
- TXD → **G16.**
- AUX → **G18.**

Como se pueden comprobar, han sido muy similares a los conectados en el transmisor. El montaje final se puede ver en la Figura 5.24.

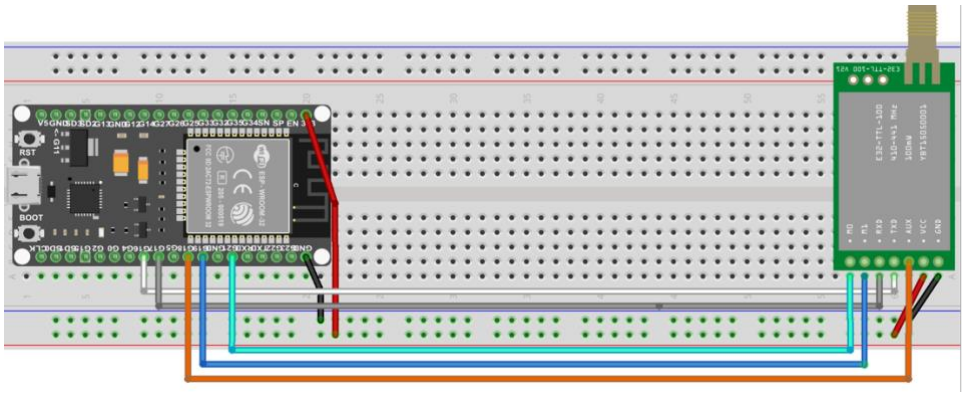


Figura 5.24. Receptor

5.2.2 Programación del receptor

En cuanto a la programación del receptor, se dividirá en dos bloques fundamentales:

1. Programación del LoRa E32 900T20D.
2. Programación del servidor web.

5.2.2.1 Programación del LoRa E32 900T20D

Una vez se han establecido todos los pines donde se conectará este dispositivo se empezará con el desarrollo.

En primer lugar, se comenzará con la definición y uso de las librerías necesarias. Las librerías para el receptor son las siguientes:

```
receptor

// Receptor con ESP32 y LoRa E32 900TD20. Realiza las
// tareas de un servidor Web.
// Autor: Manuel Antonio Ventura Duque
// Tutor: Jesús de la Casa Cárdenas
// TFG para la Universidad de Jaén (EPSL)
// Entrega: Febrero 2022

//------------------------------------------------
// Librerías
//------------------------------------------------
#include "Arduino.h"
#include "Separador.h"
#include "LoRa_E32.h"
// Import Wi-Fi library
#include "WiFi.h"
#include "ESPAsyncWebServer.h"
#include <SPIFFS.h>
//------------------------------------------------
```

Figura 5.25. Librerías para el receptor

- **Arduino.h:** Librería estándar, que contiene las funciones más principales y básicas para el uso de microcontroladores.
- **Separador.h:** Librería encargada de separar la información de la trama y almacenarlo en una variable independiente para cada sensor.
- **LoRa_E32.h:** Implementa todas las funciones necesarias para la configuración, transmisión y recepción de los datos con el módulo LoRa E32. Aunque hay otras, como LoRa.h, esta es específica para el modelo a usar.
- **Wifi.h:** Permite que el microcontrolador ESP32 se pueda conectar a una red que permita el acceso a Internet. Hay que introducir las credenciales del *router* para el acceso.
- **ESPAsyncWebServer.h:** Crea un servidor web asíncrono, que permite devolver toda la información solicitada al cliente.
- **SPIFFS.h:** Accede a la memoria flash del ESP32 y almacena los ficheros que se seleccionen. Para ello será necesario también el uso de un *plugin* denominado **ESP32 Sketch Data Upload,** que será el encargado de almacenar todos los ficheros.

Estas librerías se pueden encontrar en el siguiente enlace, donde se han almacenado en una carpeta de Drive.

https://drive.google.com/drive/folders/1E8Knq4LYh9jkRhctieNnXU4rhmb8p3_G?usp=sharing

También se pueden ver en la Figura 5.26.

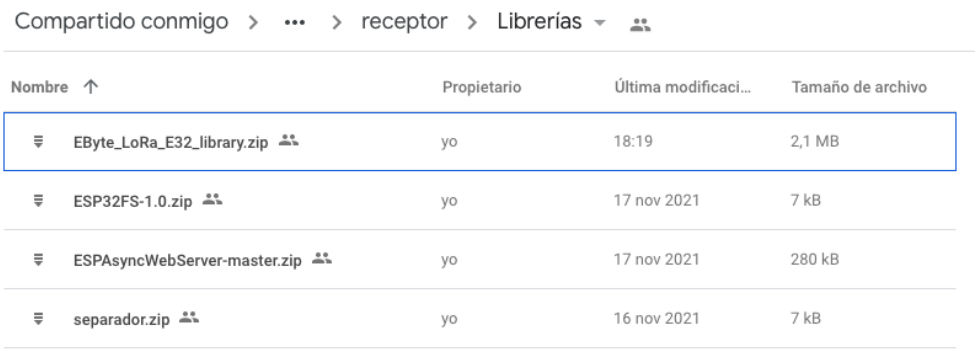


Figura 5.26. Librerías para el receptor en Google Drive

Ahora se definirán todas las variables, así como todos los parámetros de configuración de los dispositivos. Además, se establecerán las credenciales de la red WiFi donde el receptor establecerá conexión, tal y como se puede visualizar en la Figura 5.27.

receptor

```
// Variable de tipo Separador para separar los distintos
// valores del sensor del mensaje transmitido
Separador s;

// Not needed Serial2 have already this pins
#define AUX 18
#define M0 21
#define M1 19
//---------------------------------------------------

//Variables para los valores del sensor BMP 280
String temperature;
String pressure;
String altitude;
String humidity;
//---------------------------------------------------
// Credenciales del usuario
const char* ssid     = "NOMBRE_DE_LA_RED";
const char* password = "CONTRASEÑA";
// Creación de un servidor asíncrono en el puerto 80
AsyncWebServer server(80);
//---------------------------------------------------

//HardwareSerial serial1(2);
LoRa_E32 e32ttl(&Serial2, AUX, M0, M1);
void printParameters(struct Configuration configuration);
void printModuleInformation(struct ModuleInformation moduleInformation);
//---------------------------------------------------
```

Figura 5.27. Variables, y código de configuración para el receptor

Antes de pasar a las funciones principales (“setup” y “loop”) se describirá el funcionamiento de las otras funciones, ya que se realizan las llamadas en dichas funciones.

Dentro del receptor está la función nodeConfig(), que realiza la función de configuración del LoRa E32. Es exactamente igual que la del transmisor, por lo que todo quedó detallado en el apartado 5.1.3.1de esta memoria.

Como novedad se encuentra la función processor(), que tiene como objetivo reemplazar el marcador de posición con valores del BME, es decir, cuando en el servidor se encuentre con “TEMPERATURE”, “HUMIDITY”, “PRESSURE” o “ALTITUDE” se retornará su valor y se sustituirá. Esto es realmente útil a la hora de la creación de la página

web, e indexarlo con el fichero JavaScript. Así como los valores del anemómetro y pluviómetro.

Por último, se tendrá la función que realiza las tareas de conexionado con la red. Esta se denomina connectWiFi(), y devuelve una dirección IP de acceso al servidor que permite visualizar toda la información de los sensores.

```
void connectWiFi(){
  // Connect to Wi-Fi network with SSID and password
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  // Print local IP address and start web server
  Serial.println("");
  Serial.println("WiFi connected.");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}
```

Figura 5.28. Función que permite conectar el receptor a la red WiFi

Ahora se pasará a describir la función setup. Esta se inicializa únicamente una vez, y tiene como objetivo establecer una comunicación con el monitor serie, tareas de configuración que solo es necesario realizarlas una vez, como la conexión a la red, la configuración del módulo LoRa o la del sistema de archivos SPIFFS del microcontrolador.

```
void setup() {
    //Inicio comunicacion serie con monitor
    Serial.begin(9600);
    delay(500);
    // Startup all pins and UART
    e32ttl.begin();
    //Configuro el nodo LoRa
    nodeConfig();
    connectWiFi();

      if(!SPIFFS.begin(true)){
    Serial.println("An Error has occurred while mounting SPIFFS");
    return;
  }
```

Figura 5.29. Función Setup del receptor

Por último, queda comentar la función loop. Esta se ejecuta todo el tiempo, procesando la información que le llega del transmisor. Aquí es donde entra en juego la función separa, que se encarga de descomponer los valores de la trama en distintas variables. Además, toda la información se imprime por el monitor serie para tener una monitorización de lo que llega al receptor en todo momento.

```
//-----------------------------------------------

void loop() {
    if (e32ttl.available()  > 0){
    Serial.println("----------");
    ResponseContainer rc = e32ttl.receiveMessage();
    String message = rc.data;

    temperature = s.separa(message, ',' , 0);
    pressure = s.separa(message, ',' , 1);
    altitude = s.separa(message, ',' , 2);
    humidity = s.separa(message, ',' , 3);
    wind = s.separa(message, ',' , 4);
    rain = s.separa(message, ',' , 5);

    Serial.println("La temperatura es de: " + temperature + " °C");
    Serial.println("La presión es de: " + pressure + " hPa");
    Serial.println("La altitud es de: " + altitude + " m");
    Serial.println("La humedad es de: " + humidity + " %");
    Serial.println("El viento es de: " + wind + " m");
    Serial.println("Las precip. es de: " + rain + " %");
    }
}
//-----------------------------------------------
```

Figura 5.30. Función loop del receptor

5.2.2.2 Programación del servidor web

El servidor web tiene un desarrollo un poco más complicado. Se tienen que establecer todos los recursos a devolver mediante el método GET del protocolo de capa de aplicación HTTP, cada vez que se solicite.

Los recursos, que se tienen que almacenar en el gestor de archivos SPIFFS son los siguientes:

- **Index.html:** Fichero HTML de la página web.
- **Styles.css:** Fichero de estilos de la página web.
- **Index.js:** Fichero JavaScript, que contienen funciones de procesado.
- **Imágenes:** Con formato predeterminado PNG.

Es de vital importancia seguir la estructura de archivos de la Figura 5.31, ya que si no se sigue ese patrón el SPIFFS dará error. En la carpeta **data** se almacenarán todos los archivos comentados anteriormente.

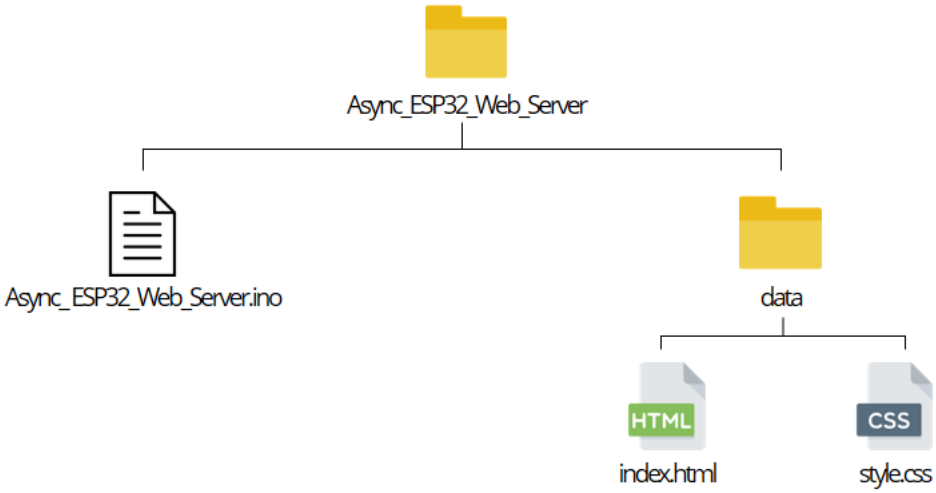


Figura 5.31. Estructura de archivos requerida [15]

El servidor web tiene que ser capaz también de retornar el valor de las variables que contienen la información de los sensores. La respuesta del servidor en estos casos será el entero 200, que indica que todo el retorno ha sido correcto.

Por último, cabe destacar que la página web ha sido diseñada para que devuelva las condiciones meteorológicas de la ciudad de Linares, es decir, cuando haga sol que se muestre un icono de soleado, etc. Para ello, se ha empleado la API de OpenWeather, que retorna esta información en un fichero JSON de cualquier ciudad. Este tendrá el siguiente formato:


```
{
"coord": {
"lon": -122.08,
"lat": 37.39
},
"weather": [
{
"id": 800,
"main": "Clear",
"description": "clear sky",
"icon": "01d"
}
],
"base": "stations",
"main": {
"temp": 282.55,
"feels_like": 281.86,
"temp_min": 280.37,
"temp_max": 284.26,
"pressure": 1023,
"humidity": 100
},
"visibility": 16093,
"wind": {
"speed": 1.5,
"deg": 350
```

```
},
"clouds": {
"all": 1
},
"dt": 1560350645,
"sys": {
"type": 1,
"id": 5122,
"message": 0.0139,
"country": "US",
"sunrise": 1560343627,
"sunset": 1560396563
},
"timezone": -25200,
"id": 420006353,
"name": "Mountain View",
"cod": 200
}
```


El icono correspondiente a la descripción del tiempo viene dado en “icon” con un valor entero y con un sufijo que indica si es de día o de noche. Solo interesa ese campo del JSON, ya que los valores más importantes vendrán dados por los sensores, que nos darán la información exacta desde la posición concreta de la estación meteorológica.

Estos iconos vienen registrados en la tabla de la Figura 5.32.

Day icon	Night icon	Description
01d.png	01n.png	clear sky
02d.png	02n.png	few clouds
03d.png	03n.png	scattered clouds
04d.png	04n.png	broken clouds
09d.png	09n.png	shower rain
10d.png	10n.png	rain
11d.png	11n.png	thunderstorm
13d.png	13n.png	snow
50d.png	50n.png	mist

Figura 5.32. Tabla de iconos de las condiciones meteorológicas

Cabe destacar que los iconos que retorna han sido modificados, y se han incorporado unos de licencia libre para evitar problemas con los derechos de autor. Para ello, es tan sencillo como fijarnos en la tabla, y cambiar los iconos en formato .svg en función de la codificación que presenta para cada descripción. Esto se realizará en el fichero JavaScript. El servidor web tiene que ser capaz de retornar estos iconos nuevos.

Toda la información necesaria viene dada en el siguiente enlace dedicado para desarrolladores de aplicaciones o páginas web:

https://openweathermap.org/api

La programación del servidor, al menos una parte ya que el resto del código viene detallado en el anexo, viene implementado en la función de setup:

```
// Route for root / web page

server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send(SPIFFS, "/index.html", String(), false, processor);
});
server.on("/styles.css", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send(SPIFFS, "/styles.css", "text/css");
});
server.on("/index.js", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send(SPIFFS, "/index.js", "text/js");
});
server.on("/temperature", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send_P(200, "text/plain", temperature.c_str());
});
server.on("/humidity", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send_P(200, "text/plain", humidity.c_str());
});
server.on("/pressure", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send_P(200, "text/plain", pressure.c_str());
});
server.on("/altitude", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send_P(200, "text/plain", altitude.c_str());
});
server.on("/assets/logo.png", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send(SPIFFS, "/assets/logo.png", "image/png");
});
```

Figura 5.33. Servidor Web

5.3 Interfaz de usuario

Cuando se habla de interfaz de usuario se hace referencia a todas las vistas donde los usuarios pueden visualizar información o interactuar con el microcontrolador. Está conformada por los archivos HTML, CSS y JavaScript, que en su conjunto realizan las tareas de procesado y representación de los datos a la parte del cliente. Esta estructura podría definirse como un modelo vista-controlador (MVC) muy conocido en la ingeniería de *software* a la hora de organizar las arquitecturas.

En la estación meteorológica, estos roles pueden ser relacionados por lo siguiente:

- **Modelo**: cuando se habla de modelo se refiere a todo lo relacionado con los datos. Esta parte, hace referencia a los sensores de la estación.
- **Vista**: se puede considerar como la interfaz de usuario, en este caso será la página web que mostrará la información de los sensores. También se puede conocer como *front-end*.
- **Controlador**: en cuanto al controlador, es la parte del sistema que se encarga de extraer la información del modelo y mostrárselo a los usuarios mediante la vista. En este caso se trata del ESP32.

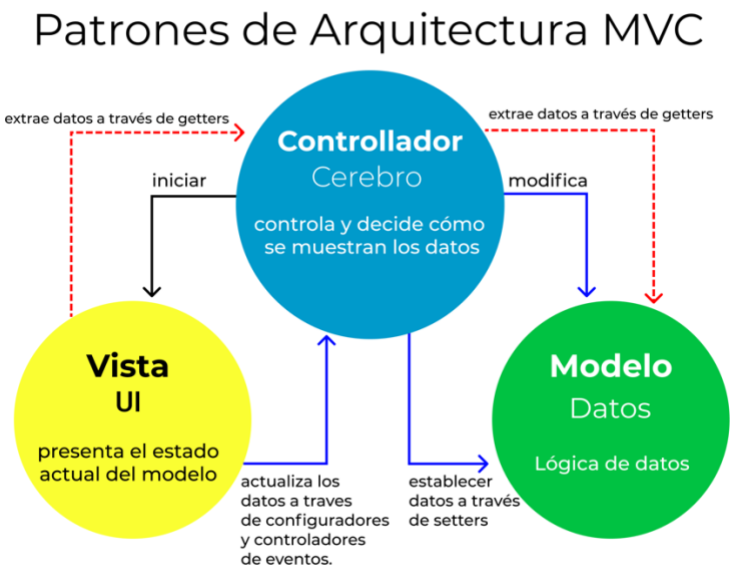


Figura 5.34. Arquitectura Modelo-Vista-Controlador

En este apartado se hablará de todo lo relacionado con las vistas, y la representación de los datos hacia el cliente.

5.3.1 Fichero index.html

El fichero **index.html** definirá la estructura de la página web. Aquí se definirán todos los títulos, cabeceras, divs (agrupaciones de contenido), imágenes y tablas que compondrán la interfaz de usuario.

Cabe destacar que en el fichero HTML no se implementa ningún estilo, si no que toda la capa de estilado vendrá luego incorporada mediante el fichero CSS, que luego se indexará.

La página web tendrá la siguiente estructura:

```
6     <title>Estación Meteorológica con LoRa</title>
7     <script type="text/javascript" src="index.js"></script>
8   </head>
9
10  <body>
11    <h1 style="text-align: center; padding-top: 4%;">Linares</h1>
12
13
14      <div id="row-temperature">
15        <div id="col-weather">
16
17        </div>
18
19        <div id="col-description">
20
21        </div>
22
23        <div id= "temperature">
24
25        </div>
26
27        <div id="table-sensores">
28          <table>
29            <tr>
30              <td>
31                <img src="./assets/Weather/Humedad.png"> Humedad
32                <div id="humidity"></div>
33              </td>
34            </div>
35              <td>
36                <img src="./assets/Weather/Presion.png"> Presión
37                <div id="pressure"></div>
38              </td>
39
40              <td>
41                <img src="./assets/Weather/Viento.png"> Viento
42              </td>
43
```

Figura 5.35. Index.html

Se pueden apreciar 4 “divs” principales, reflejados en los cuadrados azules:

1. Div id=”col-weather”: proporcionará el icono de la climatología existente en Linares. Este icono será obtenido desde el archivo JavaScript en función del JSON de la API de OpenWeather que comentamos anteriormente en el apartado 5.2.2.2 de programación del servidor *web.*
2. Div id=”col-description”: proporcionará la descripción de la climatología, es decir, si está soleado, nublado, etc. Esta información también se escoge del fichero JSON.
3. Div id=”temperature”: aparecerá la temperatura suministrada por el sensor de temperatura BME 280.

4. Div id=”table-sensores”: se mostrarán los demás valores de los sensores de la estación meteorológica.

En la Figura 5.36 se puede comprobar el resultado de dicha estructura, una vez ya estilada mediante el fichero CSS.

Figura 5.36. Index.html ya estilado

5.3.2 Styles.css

Como se ha mencionado anteriormente, el fichero styles.css contendrá todo el código de desarrollo en cuanto al estilado del HTML. El código se puede ver en los anexos de esta memoria.

5.3.3 Index.js

El fichero index.js será el encargado de obtener toda la información de los sensores gracias al servidor web, que monta el ESP32. Cada 3 segundos actualizará esa información y se mostrará en cada uno de los divs correspondientes a cada sensor.

Se hace uso de un objeto XMLHttpRequest [2], que permite hacer uso de los métodos HTTP en cualquier URL sin necesidad de refrescar la página de una manera dinámica. En este caso se hará uso del método GET que enviará la petición, al servidor,

de una manera asíncrona e imprimirá el valor en el div que le corresponda. La respuesta (responseText) será el valor que se recoge del servidor para cada sensor.

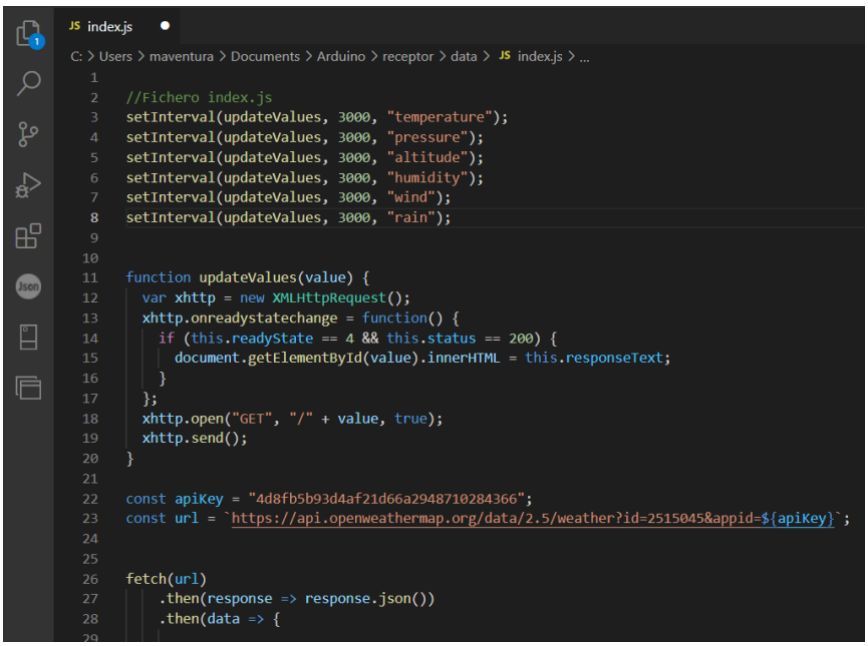


Figura 5.37. Index.js

Por ejemplo, para el caso de la temperatura, cuando se hace una llamada al método open se pasa como parámetros el método HTTP GET, seguido de la URL (/temperature) o más conocido en el mundo de la programación como *endpoint*, y un tipo de dato boolean con valor “true” indicando que la operación debe de realizarse de manera asíncrona [2].

Una vez se ha abierto, se envía o realiza la solicitud accediendo al servidor web, como se puede comprobar en la figura 5.38 devuelve la información de la temperatura en formato de texto.

```
server.on("/temperature", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send_P(200, "text/plain", temperature.c_str());
});
```

Figura 5.38. Solicitud de devolución del valor de la temperatura

6 RESULTADOS Y DISCUSIÓN

En este capítulo se comentarán todos los resultados obtenidos una vez se pone en funcionamiento la estación meteorológica y la comunicación entre transmisor y receptor.

En primer lugar, una vez se ha configurado el transmisor y emite un mensaje de verificación por el monitor serie de Arduino IDE, comenzará la conformación de la trama a enviar.

Esta trama estará compuesta por todos los valores de los sensores, y una vez finalizada se envía. Cuando llega al receptor, aparece un mensaje de verificación con texto "*success*" por el monitor serie.

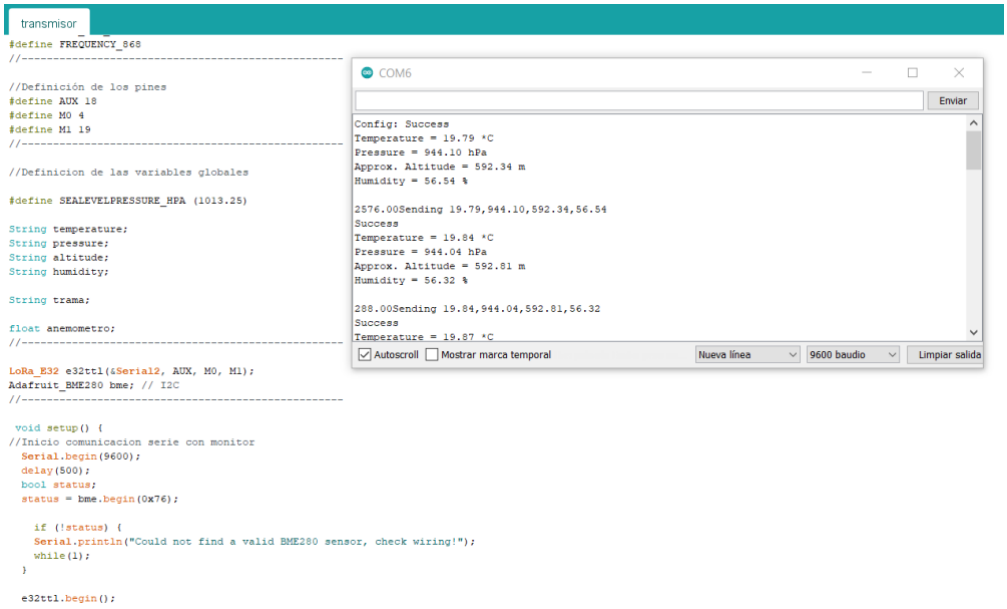


Figura 6.1. Monitor serie del transmisor y resultados

Cabe destacar, que para monitorizar el transmisor es requisito indispensable conectarlo al ordenador mediante el puerto USB. Una vez se conecte al sistema fotovoltaico será imposible visualizar la información, solo en el receptor.

A continuación, se procede a analizar los resultados obtenidos en el receptor.

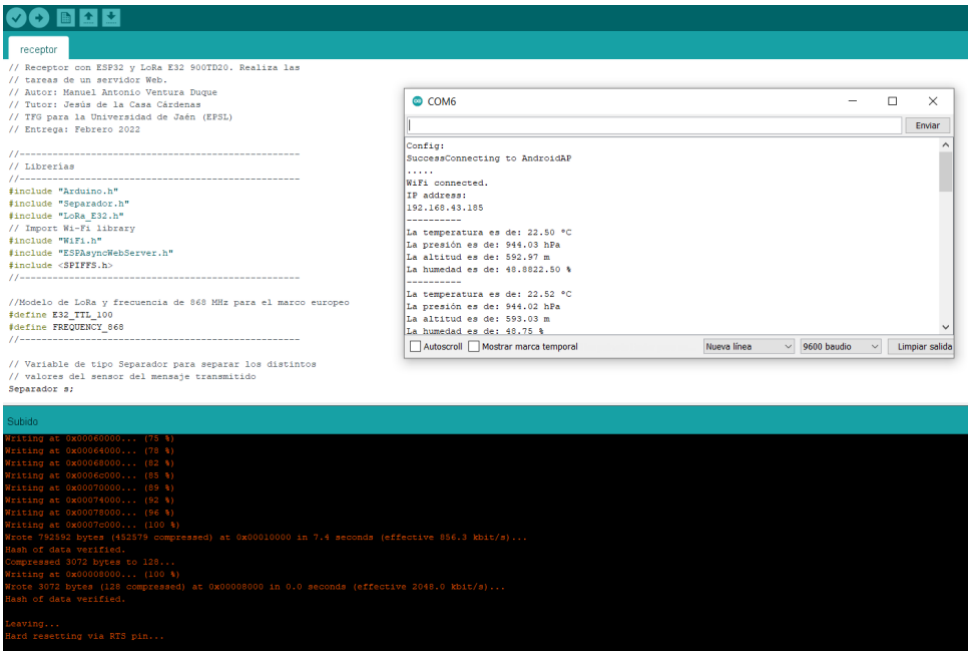


Figura 6.2. Monitor serie del receptor y resultados

Como se puede ver en la Figura 6.2 lo primero que se crea es el servidor *web*. Este se crea en la dirección **192.168.43.185**, aunque la dirección puede cambiar en función de la red a la que se conecta. En este caso, se procedió a conectar a una red móvil denominada **AndroidAP**.

Después, empieza a llegar toda la información. La función separa se encarga de descomponer la trama, y asignar a cada variable de los sensores su determinado valor. Estos valores se reflejarán en la página *web* alojada en la dirección que se ha comentado anteriormente.

En la Figura 6.3 quedan expuestos los resultados finales, que serán visualizados por la parte del cliente.

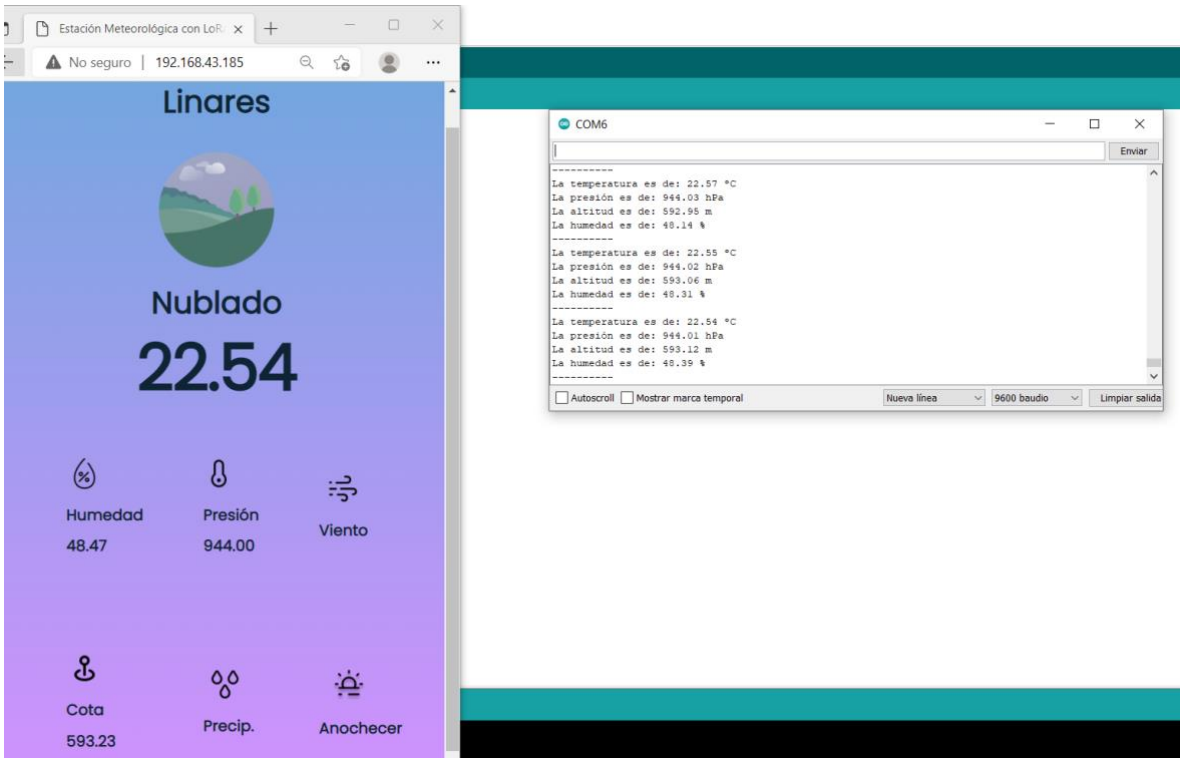


Figura 6.3. Resultados reflejados en la interfaz de usuario

7 CONCLUSIONES Y LÍNEAS FUTURAS

Tras abordar todas las tareas de investigación, desarrollo e implementación de la estación meteorológica se han podido llegar a numerosas **conclusiones,** que sin lugar a duda, ayudan a que en un futuro se pueda seguir investigando para mejorar estos sistemas tan usados a día de hoy.

Unas de las principales conclusiones que se ha podido obtener ha sido la dificultad de poner en marcha dispositivos tan recientes como el LoRa E32 900T20D. A día de hoy, ni el propio fabricante proporciona una librería para los programadores, por lo que hay que buscar librerías ya creadas por usuarios particulares, de otros modelos similares, para poner en marcha este dispositivo de comunicaciones inalámbricas de alto alcance. Una solución podría ser crear una librería propia de funciones, pero al existir una más o menos genérica no ha sido realmente necesario.

En cuanto a los sensores, cabe destacar la dificultad que existe en diseñar sensores analógicos propios con una impresora 3D (anemómetro y pluviómetro). Han hecho falta tareas de calibración, para obtener valores acordes, así como numerosas pruebas para ver que funcionan realmente y una monitorización constante.

También, la importancia que tiene las perdidas por difracción debido a los obstáculos puede decidir si se realiza la comunicación correctamente o no. Siempre se debe de buscar una visión directa (LOS) para tener unas perdidas mínimas.

Otra conclusión, que se ha reflejado a lo largo de esta memoria, ha sido la alta eficiencia energética que presenta este sistema. Con una pila recargable y una placa fotovoltaica de 2W la estación meteorológica puede ser alimentada constantemente. Si a esto le sumamos esta fuente autónoma, no es necesario preocuparse de la alimentación durante un gran período de tiempo hasta que la vida útil vaya disminuyendo. Sin duda, aplicaciones que miran por una alta eficiencia energética son el futuro.

Por último, presentar una interfaz de usuario de tan fácil acceso, posibilita que numerosas personas, que posean cultivos, puedan interesarse por este proyecto, ya que se trata de una inversión para estar al tanto constantemente de las condiciones meteorológicas de una zona geográfica. De esta forma, se podría mejorar la calidad de los cultivos, controlando acciones como el regar al tener un *feedback* de la situación de las precipitaciones o la humedad del ambiente, por ejemplo.

En cuanto a las **líneas de futuro** sin duda se pueden incorporar muchas funcionalidades más para seguir mejorando este proyecto. Como, por ejemplo, la incorporación de más sensores que puedan mostrar más información al usuario. Una idea,

para el área de cultivo, sería que mostrara al propietario de dicha área cuando se tendría que regar determinadas plantas, establecer funcionalidades de regado automáticas o manuales, etc.

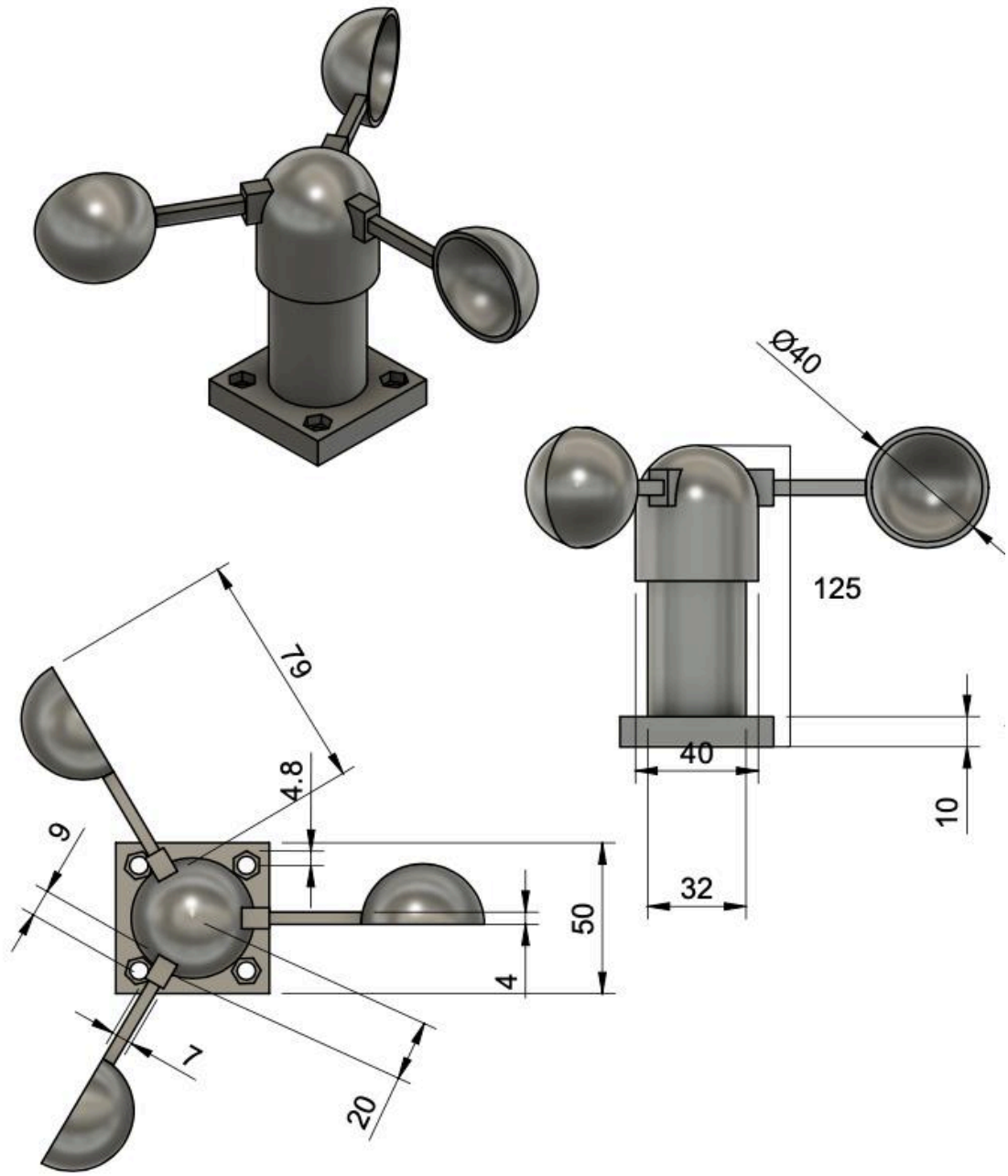
En cuanto al área de seguridad incorporar cámaras de seguridad para poder visualizar lo que sucede en el área de la estación meteorológica, o la encriptación de la señal que genera el LoRa para que no sea interferida.

Se podría también incorporar un sistema de monitorización del estado de la batería para tener un control de la vida útil, y así poder cambiarla cuando sea necesario.

Si se mira más hacia un futuro lejano, se podría incorporar una red de estaciones meteorológicas que compartieran toda la información entre ellas, para realizar predicciones meteorológicas según la zona, establecer históricos para ver valores máximos o mínimos, etc.

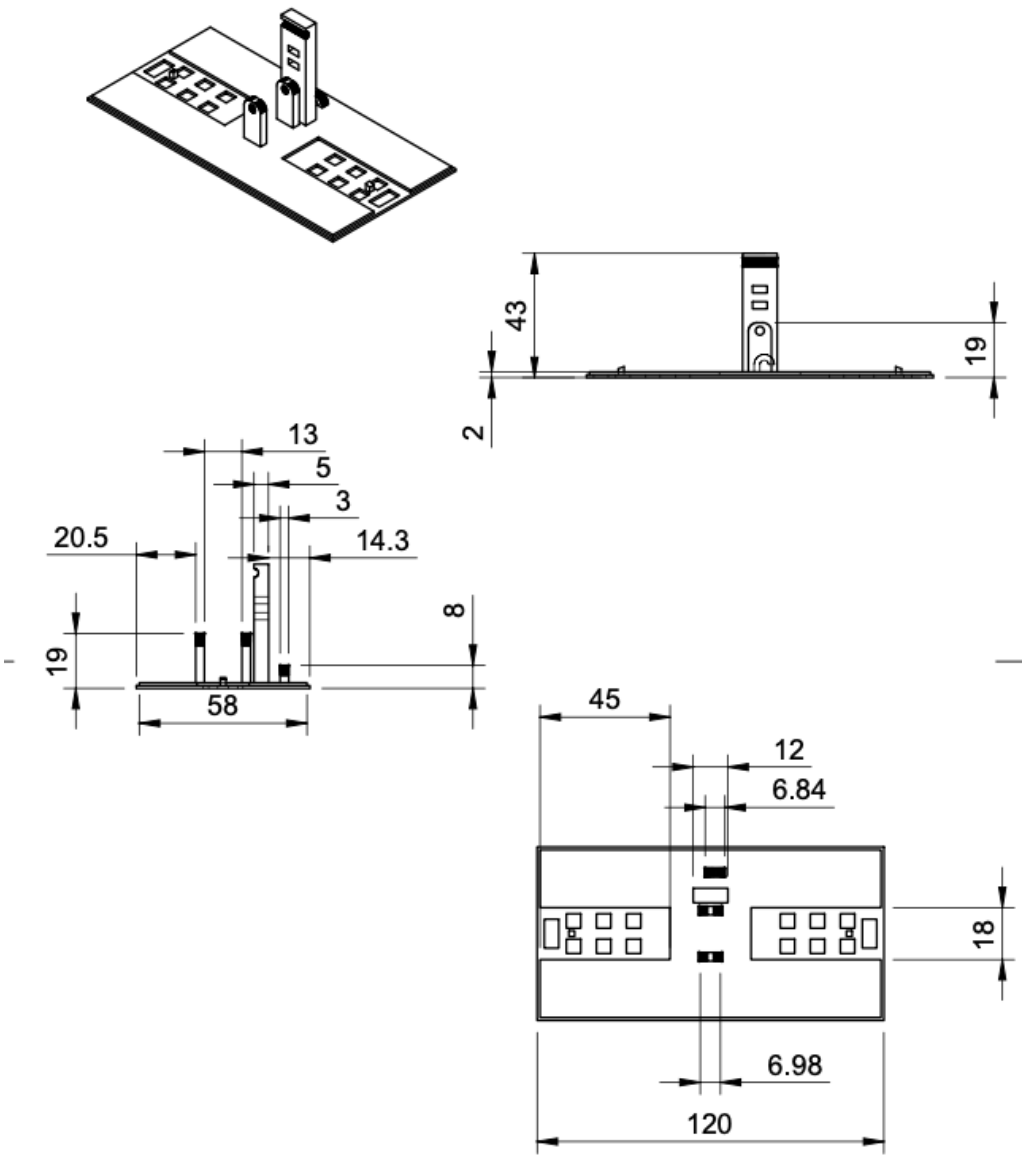
8 PLANOS

8.1 Anemómetro

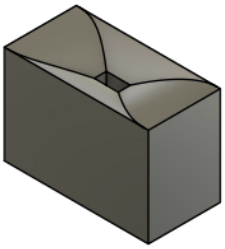


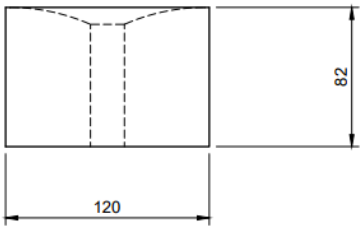
Título: Anemómetro		
Fecha: 15/12/2021	**Autor**: Manuel A. Ventura Duque	**Universidad de Jaén (EPSL)**
Escala: S/E	**Nº Plano**: 1	

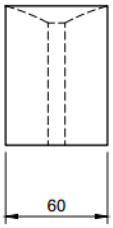
8.2 Pluviómetro



Título: Base pluviómetro		
Fecha: 15/12/2021	**Autor**: Manuel A. Ventura Duque	**EPSL**
Escala: S/E	**Nº Plano**: 1	







Título: Carcasa exterior pluviómetro		
Fecha: 15/12/2021	**Autor**: Manuel A. Ventura Duque	**EPSL**
Escala: S/E	**Nº Plano**: 1	

9 PRESUPUESTO

9.1 Presupuesto de costes

Concepto	Precio/Unidad	Cantidad	Precio
ESP32	5,50 €	2	11,00 €
LoRa E32 900T20D	6,00 €	2	12,00 €
Antenas	2,50 €	2	5 €
Placa Solar	4,00 €	1	4,00 €
Módulo TP4056	1,50 €	1	1,50 €
Protoboards	3,00 €	2	6,00 €
Cables	2,50 €	-	2,50 €
Regulador MCP1700-3302E	4,00 €	1	4,00 €
Sensor BME 280	7,00 €	1	7,00 €
Otro	5,00 €	-	5,00 €
Precio Total SIN I.V.A (21%)			51,00 €
Precio Total CON I.V.A (21%)			**61,71 €**

10 REFERENCIAS BIBLIOGRÁFICAS

[1] "El Osciloscopio". (2019). *Elosciloscopio.com*. Obtenido de Comparación de Arduino vs ESP8266 vs ESP32: https://elosciloscopio.com/comparacion-arduino-vs-esp8266-vs-esp32/

[2] (2021). Obtenido de XMLHttpRequest: https://developer.mozilla.org/es/docs/Web/API/XMLHttpRequest

[3] Asamblea de la Radiocomunicación de la UIT. (1972-1982-1994). CÁLCULO DE LA ATENUACIÓN EN EL ESPACIO LIBRE. Obtenido de Rec. UIT-R P.525-2: https://www.itu.int/dms_pubrec/itu-r/rec/p/R-REC-P.525-2-199408-I!!PDF-S.pdf

[4] Catsensors. (2020). *Catsensors.com*. Obtenido de TECNOLOGÍA LORA Y LORAWAN: https://www.catsensors.com/es/lorawan/tecnologia-lora-y-lorawan

[5] *DARRERA*. (2018). Obtenido de Pluviómetro de Balancín: https://www.darrera.com/wp/es/producto/hd2015-pluviometro-balancin/

[6] Dave Korn, C. (2019). *The MathWorks, Inc.* Obtenido de ThingSpeak for IoT Projects. Data collection in the cloud with advanced data analysis using MATLAB: https://es.mathworks.com/products/thingspeak.html

[7] Devices, W. C. (Diciembre de 2018). *Researchgate.* Obtenido de www.researchgate.net

[8] Escribano Vega, J. (2016). *Implementación de una estación metereológica con Arduino.* Gandía, Valencia, España.

[9] GCFGlobal. (s.f.). *¿Qué es hardware y software?* Obtenido de https://edu.gcfglobal.org/es/informatica-basica/que-es-hardware-y-software/1/

[10] *Github*. (2020). Obtenido de Adafruit / Adafruit_Sensor: https://github.com/adafruit/Adafruit_Sensor

[11] *Hetpro*. (2018). Obtenido de I2C – Puerto, Introducción, trama y protocolo: https://hetpro-store.com/TUTORIALES/i2c/

[12] Lozano, R. (13 de Junio de 2021). *TalosElectronics*. Obtenido de Programar ESP32 con IDE arduino: https://www.taloselectronics.com/blogs/tutoriales/programar-esp32-con-ide-arduino

[13] Martínez, J. L. (13 de Julio de 2018). *ProRed.es*. Obtenido de Zonas de Fresnel: https://www.prored.es/zonas-de-fresnel-en-un-radioenlace/

[14] Mischianti, R. (22 de October de 2019). *Arduino.cc*. Obtenido de LoRa E32 for Arduino, ESP32 or ESP8266: Specs and Base Use : https://create.arduino.cc/projecthub/xreef/lora-e32-for-arduino-esp32-or-esp8266-specs-and-base-use-804d25

[15] Nick. (2 de Mayo de 2019). *Randomnerdtutorials*. Obtenido de Power ESP32/ESP8266 with Solar Panels (includes battery level monitoring): https://randomnerdtutorials.com/power-esp32-esp8266-solar-panels-battery-level-monitoring/

[16] Rivas, J. (8 de Abril de 2021). *Modulación de LoRa*. Obtenido de LoRa Panamá: https://lora-panama.com/modulacion-de-lora/

[17] Ruesca, P. (25 de Septiembre de 2016). *Radio Comunicaciones*. Obtenido de ANTENAS DIPOLO: http://www.radiocomunicaciones.net/radio/antenas-dipolo/

[18] Solera, E. (27 de Agosto de 2018). *Modulación LoRa: Long Range Modulation*. Obtenido de Exploración de aplicaciones usando tecnología LoRa: https://medium.com/pruebas-de-laboratorio-de-la-modulaci%C3%B3n-lora/modulaci%C3%B3n-lora-4ad74cabd59e

[19] Unión Internacional de Telecomunicaciones, UIT. (2019). Programación por difracción. En *Propagación de las ondas radioeléctricas* (págs. 526-15). Obtenido de Programación de las ondas radioeléctricas: https://www.itu.int/dms_pubrec/itu-r/rec/p/R-REC-P.526-15-201910-I!!PDF-S.pdf

[20] WebDesign. (2021). *WebDesign*. Obtenido de Ventajas y Desventajas de Visual Studio Code 2021: https://webdesigncusco.com/ventajas-y-desventajas-de-visual-studio-code/

[21] Yao, J., & 2CiGroup. (22 de Marzo de 2021). *2CiGroup*. Obtenido de Conceptos de actualidad: LoRa y LoRaWan: https://www.2cigroup.com/es/conceptos-de-actualidad-lora-y-lorawan/

11 ANEXOS

Este capítulo contendrá toda la información justificativa a este proyecto, como los códigos de programación, hoja de características y los cálculos relacionados con el diseño de la estación meteorológica.

11.1 Código del transmisor

```
// Transmisor con ESP32 y LoRa E32 900TD20
// Autor: Manuel Antonio Ventura Duque
// Tutor: Jesús de la Casa Cárdenas
// TFG para la Universidad de Jaén (EPSL)
// Entrega: Marzo 2022

//------------------------------------------------
// Librerías
//------------------------------------------------
#include "Arduino.h"
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#include "LoRa_E32.h"
//------------------------------------------------

//Modelo de LoRa y frecuencia de 868 MHz para el marco europeo
#define E32_TTL_100
#define FREQUENCY_868
//------------------------------------------------

//Definición de los pines
#define AUX 18
#define M0 4
#define M1 19
//------------------------------------------------

//Definicion de las variables globales

#define SEALEVELPRESSURE_HPA (1013.25)
String trama;

String temperature;
String pressure;
String altitude;
String humidity;
String wind;
String rain;

//------------------------------------------------

LoRa_E32 e32ttl(&Serial2, AUX, M0, M1);
Adafruit_BME280 bme; // I2C
//------------------------------------------------

 void setup() {
//Inicio comunicacion serie con monitor
  Serial.begin(9600);
  delay(500);
```

```
  bool status;
  status = bme.begin(0x76);

    if (!status) {
    Serial.println("Could not find a valid BME280 sensor, check
wiring!");
    while(1);
  }

  e32ttl.begin();

  //Configuración del nodo LoRa E32 900T20D
  nodeConfig();

}
//----------------------------------------------------

void loop() {

    temperature,pressure,altitude,humidity,wind,rain = valoresensor();
    trama = temperature + "," + pressure+ "," +  altitude+ ","
+  humidity+ "," + wind + "," + rain;
    ResponseStatus rs = e32ttl.sendFixedMessage(0x0, 0x3, 0x10, trama);
    Serial.println("Sending " + trama);
    Serial.println( rs.getResponseDescription());
    delay(3000);
}
//----------------------------------------------------
//Configuración del nodo LoRa
 void nodeConfig() {
  ResponseStructContainer c;
  c = e32ttl.getConfiguration();
  Configuration configuration =  *(Configuration*) c.data;
  configuration.ADDH = 0x00;
  configuration.ADDL = 0x1;
  configuration.CHAN = 0x10;
  configuration.SPED.airDataRate = AIR_DATA_RATE_100_96;
  configuration.SPED.uartBaudRate = UART_BPS_9600;
  configuration.SPED.uartParity = MODE_00_8N1;
  configuration.OPTION.fec = FEC_1_ON;
  configuration.OPTION.fixedTransmission = FT_FIXED_TRANSMISSION;
  configuration.OPTION.ioDriveMode = IO_D_MODE_PUSH_PULLS_PULL_UPS;
  configuration.OPTION.transmissionPower = POWER_20;
  configuration.OPTION.wirelessWakeupTime = WAKE_UP_1250;
// Set configuration changed and set to not hold the configuration
  ResponseStatus rs = e32ttl.setConfiguration(configuration,
WRITE_CFG_PWR_DWN_SAVE);
  Serial.print("Config: ");
  Serial.println(rs.getResponseDescription());
}
//----------------------------------------------------
// Función que extrae todos los valores del sensor BMP 280 y los
devuelve.
 String valoresensor(){

  temperature = bme.readTemperature();
  Serial.print("Temperature = ");
  Serial.print(temperature);
  Serial.println(" *C");

  pressure = (bme.readPressure() / 100.0F);
```

```
  Serial.print("Pressure = ");
  Serial.print(pressure);
  Serial.println(" hPa");

  altitude = bme.readAltitude(SEALEVELPRESSURE_HPA);
  Serial.print("Approx. Altitude = ");
  Serial.print(altitude);
  Serial.println(" m");

  humidity = bme.readHumidity();
  Serial.print("Humidity = ");
  Serial.print(humidity);
  Serial.println(" %");

  wind = analogRead(25);
  wind = wind*30/2000;
  Serial.print("Viento = ");
  Serial.print(wind);
  Serial.println(" km/h");

  rain = analogRead(26);
  if(rain>0 && rain<4095){
    rain_aux = rain_aux+10;
  }
  Serial.print("Precip. = ");
  Serial.print(rain);
  Serial.println(" mm");

  Serial.println();

  return temperature,pressure,altitude,humidity,wind,rain;

}
//-----------------------------------------------------
```

11.2 Código del receptor

```
// Receptor con ESP32 y LoRa E32 900TD20. Realiza las
// tareas de un servidor Web.
// Autor: Manuel Antonio Ventura Duque
// Tutor: Jesús de la Casa Cárdenas
// TFG para la Universidad de Jaén (EPSL)
// Entrega: Marzo 2022

//-----------------------------------------------------
// Librerías
//-----------------------------------------------------
#include "Arduino.h"
#include "Separador.h"
#include "LoRa_E32.h"
// Import Wi-Fi library
#include "WiFi.h"
#include "ESPAsyncWebServer.h"
#include <SPIFFS.h>
//-----------------------------------------------------

//Modelo de LoRa y frecuencia de 868 MHz para el marco europeo
#define E32_TTL_100
#define FREQUENCY_868
//-----------------------------------------------------
```

```
// Variable de tipo Separador para separar los distintos
// valores del sensor del mensaje transmitido
Separador s;

// Not needed Serial2 have already this pins
#define AUX 18
#define M0 21
#define M1 19
//-----------------------------------------------------

//Variables para los valores del sensor BMP 280
String temperature;
String pressure;
String altitude;
String humidity;
//-----------------------------------------------------
// Credenciales del usuario
const char* ssid     = "NOMBRE_DE_LA_RED;
const char* password = "CONTRASEÑA";
// Creación de un servidor asíncrono en el puerto 80
AsyncWebServer server(80);
//-----------------------------------------------------

//HardwareSerial serial1(2);
LoRa_E32 e32ttl(&Serial2, AUX, M0, M1);
void printParameters(struct Configuration configuration);
void printModuleInformation(struct ModuleInformation moduleInformation);
//-----------------------------------------------------

void setup() {
    //Inicio comunicacion serie con monitor
    Serial.begin(9600);
    delay(500);
    // Startup all pins and UART
    e32ttl.begin();
    //Configuro el nodo LoRa
    nodeConfig();
    connectWiFi();

      if(!SPIFFS.begin(true)){
    Serial.println("An Error has occurred while mounting SPIFFS");
    return;
  }
  // Route for root / web page

  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/index.html", String(), false, processor);
  });
  server.on("/styles.css", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/styles.css", "text/css");
  });
  server.on("/index.js", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/index.js", "text/js");
  });
  server.on("/temperature", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", temperature.c_str());
  });
  server.on("/humidity", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", humidity.c_str());
  });
```

```
  server.on("/pressure", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", pressure.c_str());
  });
  server.on("/altitude", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", altitude.c_str());
  });
  server.on("/wind", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", wind.c_str());
  });
  server.on("/rain", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", rain.c_str());
  });

  server.on("/assets/logo.png", HTTP_GET, [](AsyncWebServerRequest
*request){
    request->send(SPIFFS, "/assets/logo.png", "image/png");
  });
  server.on("/assets/Lora.png", HTTP_GET, [](AsyncWebServerRequest
*request){
    request->send(SPIFFS, "/assets/Lora.png", "image/png");
  });
  server.on("/assets/Weather/Humedad.png", HTTP_GET,
[](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/assets/Weather/Humedad.png", "image/png");
  });
  server.on("/assets/Weather/Presion.png", HTTP_GET,
[](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/assets/Weather/Presion.png", "image/png");
  });
  server.on("/assets/Weather/Viento.png", HTTP_GET,
[](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/assets/Weather/Viento.png", "image/png");
  });
  server.on("/assets/Weather/Altitud.png", HTTP_GET,
[](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/assets/Weather/Altitud.png", "image/png");
  });

  server.on("/assets/Weather/Sunset.png", HTTP_GET,
[](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/assets/Weather/Sunset.png", "image/png");
  });
  server.on("/assets/Weather/Lluvia.png", HTTP_GET,
[](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/assets/Weather/Lluvia.png", "image/png");
  });

  server.on("/assets/Weather/Day/1.png", HTTP_GET,
[](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/assets/Weather/Day/1.png", "image/png");
  });
  server.on("/assets/Weather/Day/2.png", HTTP_GET,
[](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/assets/Weather/Day/2.png", "image/png");
  });
    server.on("/assets/Weather/Day/3.png", HTTP_GET,
[](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/assets/Weather/Day/3.png", "image/png");
  });
    server.on("/assets/Weather/Day/4.png", HTTP_GET,
[](AsyncWebServerRequest *request){
```

```
    request->send(SPIFFS, "/assets/Weather/Day/4.png", "image/png");
  });
    server.on("/assets/Weather/Day/5.png", HTTP_GET,
[](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/assets/Weather/Day/5.png", "image/png");
  });
    server.on("/assets/Weather/Day/6.png", HTTP_GET,
[](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/assets/Weather/Day/6.png", "image/png");
  });
    server.on("/assets/Weather/Day/7.png", HTTP_GET,
[](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/assets/Weather/Day/7.png", "image/png");
  });
  server.on("/assets/Weather/Night/1.png", HTTP_GET,
[](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/assets/Weather/Night/1.png", "image/png");
  });

// Iconos para la noche
  server.on("/assets/Weather/Night/1.png", HTTP_GET,
[](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/assets/Weather/Night/1.png", "image/png");
  });
    server.on("/assets/Weather/Night/2.png", HTTP_GET,
[](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/assets/Weather/Night/2.png", "image/png");
  });
    server.on("/assets/Weather/Night/4.png", HTTP_GET,
[](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/assets/Weather/Night/4.png", "image/png");
  });
    server.on("/assets/Weather/Night/5.png", HTTP_GET,
[](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/assets/Weather/Night/5.png", "image/png");
  });
    server.on("/assets/Weather/Night/6.png", HTTP_GET,
[](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/assets/Weather/Night/6.png", "image/png");
  });

  // Start server
  server.begin();
}
//-----------------------------------------------------

void loop() {
    if (e32ttl.available()  > 0){
    Serial.println("----------");
    ResponseContainer rc = e32ttl.receiveMessage();
    String message = rc.data;

    temperature = s.separa(message, ',' , 0);
    pressure = s.separa(message, ',' , 1);
    altitude = s.separa(message, ',' , 2);
    humidity = s.separa(message, ',' , 3);
    wind = s.separa(message, ',' , 4);
    rain = s.separa(message, ',' , 5);

    Serial.println("La temperatura es de: " + temperature + " °C");
    Serial.println("La presión es de: " + pressure + " hPa");
```

```
    Serial.println("La altitud es de: " + altitude + " m");
    Serial.println("La humedad es de: " + humidity + " %");
    Serial.println("El viento es de: " + wind + " km/h");
    Serial.println("Las precip. es de: " + rain + " mm");
    }
}
//----------------------------------------------------

void nodeConfig(){

  ResponseStructContainer c;
  c = e32ttl.getConfiguration();
  Configuration configuration = *(Configuration*) c.data;
  configuration.ADDH        = 0x0;
  configuration.ADDL        = 0x3;
  configuration.CHAN        = 0x10;

  configuration.SPED.airDataRate           = AIR_DATA_RATE_100_96;
  configuration.SPED.uartBaudRate          = UART_BPS_9600;
  configuration.SPED.uartParity            = MODE_00_8N1;
  configuration.OPTION.fec                 = FEC_1_ON;
  configuration.OPTION.fixedTransmission   = FT_FIXED_TRANSMISSION;
  configuration.OPTION.ioDriveMode         =
IO_D_MODE_PUSH_PULLS_PULL_UPS;
  configuration.OPTION.transmissionPower   = POWER_20;
  configuration.OPTION.wirelessWakeupTime  = WAKE_UP_1250;
  //printParameters(configuration);

  // Set configuration changed and set to not hold the configuration
  ResponseStatus rs = e32ttl.setConfiguration(configuration,
WRITE_CFG_PWR_DWN_SAVE);
  Serial.println("Config: ");
  Serial.print(rs.getResponseDescription());
}
//----------------------------------------------------
// Reemplaza el marcador de posición con valores del BME
String processor(const String& var){

  //Serial.println(var);
  if(var == "TEMPERATURE"){
    return temperature;
  }
  else if(var == "HUMIDITY"){
    return humidity;
  }
  else if(var == "PRESSURE"){
    return pressure;
  }
  else if(var == "ALTITUDE"){
    return altitude;
  }
  else if(var == "WIND"){
    return wind;
  }
  else if(var == "RAIN"){
    return rain;
  }
  return String();
}

void connectWiFi(){
```

```
  // Connect to Wi-Fi network with SSID and password
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  // Print local IP address and start web server
  Serial.println("");
  Serial.println("WiFi connected.");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}
```

11.3 Código de la interfaz de usuario

11.3.1 index.html

```
<!DOCTYPE HTML><html>
<head>
  <link rel="stylesheet" type="text/css" href="styles.css">
  <link href='https://fonts.googleapis.com/css?family=Poppins'
rel='stylesheet'>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
  <title>Estación Meteorológica con LoRa</title>
  <script type="text/javascript" src="index.js"></script>
</head>

<body>
  <h1 style="text-align: center; padding-top: 4%;">Linares</h1>


    <div id="row-temperature">
     <div id="col-weather">

     </div>

     <div id="col-description">

     </div>

     <div id= "temperature">

     </div>

     <div id="table-sensores">
        <table>
          <tr>
            <td>
              <img src="./assets/Weather/Humedad.png"> Humedad
              <div id="humidity"></div>
            </td>
          </div>
            <td>
              <img src="./assets/Weather/Presion.png"> Presión
              <div id="pressure"></div>
            </td>

            <td>
              <img src="./assets/Weather/Viento.png"> Viento
            </td>

          </tr>

          <tr>
            <td>
              <img src="./assets/Weather/Altitud.png"> Cota
              <div id="altitude"></div>
            </td>

            <td id="precipitaciones">
              <img src="./assets/Weather/Lluvia.png"> Precip.
            </td>
```

```
            <td id="anochecer">
              <img src="./assets/Weather/Sunset.png"> Anochecer
            </td>

          </tr>
        </table>
      </div>
    </div>
</body>
</html>
```

11.3.2 Styles.css

```
html{
    background: linear-gradient(180deg, #6FA7DE 0%, #D591FF 100%);

}

@media (max-width: 410px){

body{

    margin: 0 auto;
}

table{
    margin-left: auto;
    margin-right: auto;
}
}

h1{
    font-family: Poppins;
    font-style: normal;
    font-weight: bold;
    font-size: 48px;
    line-height: 49px;
    /* identical to box height, or 163% */
    padding-top: 1%;
    padding-bottom: 3%;
    color: #0F2536;

}

h3{

line-height: 2!important;

}

#temperature{

    position: relative;
    font-family: Poppins;
    font-style: normal;
    font-weight: bold;
    font-size: 101px;
    line-height: 131px;
    letter-spacing: -0.03em;
    justify-content: center;
    color: #0F2536;
```

```
}

#col-description{

    padding-top: 2%;
    font-family: Poppins;
    font-style: normal;
    font-weight: bold;
    font-size: 48px;
    color: #0F2536;



}

#row-temperature{

text-align: center;

}
body{


    height: 100%;
    width: 100%;
    position: relative;
}

th,td{
    font-family: Poppins;
    font-style: normal;
    font-weight: bold;
    font-size: 24px;
    line-height: 49px;
    padding: 10%;
    color: #0F2536;
    justify-content: center;
    text-align: -webkit-center;
}


th,td p{

    font-family: Poppins;
    font-style: normal;
    font-weight: bold;
    font-size: 22px;
    line-height: 49px;
    padding: 10%;
    color: #0F2536;
    justify-content: center;
    display: table-cell;


}

table{

display: flex;
```

```
justify-content: center;
border-spacing: 40px 10px;
border-collapse: inherit!important;
}

td#humedad img{

margin-left: 55%;

}


td#presion img{

margin-left: 50%;

}
td#viento img{

margin-left: 10%;

}

td#precipitaciones img{

margin-left: 40%;

}

td#anochecer img{

margin-left: 25%;
}
```

11.3.3 Index.js

```
//Fichero index.js
setInterval(updateValues, 3000, "temperature");
setInterval(updateValues, 3000, "pressure");
setInterval(updateValues, 3000, "altitude");
setInterval(updateValues, 3000, "humidity");
setInterval(updateValues, 3000, "wind");
setInterval(updateValues, 3000, "rain");

function updateValues(value) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById(value).innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "/" + value, true);
  xhttp.send();
}

const apiKey = "4d8fb5b93d4af21d66a2948710284366";
const url =
`https://api.openweathermap.org/data/2.5/weather?id=2515045&appid=${apiKe
y}`;

fetch(url)
```

```
    .then(response => response.json())
    .then(data => {

      if(data.weather[0].icon == "01d"){
      document.getElementById("col-weather").innerHTML = "<img
src='./assets/Weather/Day/1.png'>"
    }
    if(data.weather[0].icon == "01n"){
      document.getElementById("col-weather").innerHTML = "<img
src='./assets/Weather/Night/1.png'>"
    }
    if(data.weather[0].icon == "02d"){
      document.getElementById("col-weather").innerHTML = "<img
src='./assets/Weather/Day/2.png'>"
    }
    if(data.weather[0].icon == "02n"){
      document.getElementById("col-weather").innerHTML = "<img
src='./assets/Weather/Night/2.png'>"
    }
    if(data.weather[0].icon == "03d"){
      document.getElementById("col-weather").innerHTML = "<img
src='./assets/Weather/Day/3.png'>"
    }
    if(data.weather[0].icon == "03n"){
      document.getElementById("col-weather").innerHTML = "<img
src='./assets/Weather/Day/3.png'>"
    }
    if(data.weather[0].icon == "04d"){
      document.getElementById("col-weather").innerHTML = "<img
src='./assets/Weather/Day/3.png'>"
    }
    if(data.weather[0].icon == "04n"){
      document.getElementById("col-weather").innerHTML = "<img
src='./assets/Weather/Day/3.png'>"
    }
    if(data.weather[0].icon == "09d"){
      document.getElementById("col-weather").innerHTML = "<img
src='./assets/Weather/Day/4.png'>"
    }
    if(data.weather[0].icon == "09n"){
      document.getElementById("col-weather").innerHTML = "<img
src='./assets/Weather/Night/4.png'>"
    }
    if(data.weather[0].icon == "10d"){
      document.getElementById("col-weather").innerHTML = "<img
src='./assets/Weather/Day/4.png'>"
    }
    if(data.weather[0].icon == "10n"){
      document.getElementById("col-weather").innerHTML = "<img
src='./assets/Weather/Night/4.png'>"
    }
    if(data.weather[0].icon == "11d"){
      document.getElementById("col-weather").innerHTML = "<img
src='./assets/Weather/Day/5.png'>"
    }
    if(data.weather[0].icon == "11n"){
      document.getElementById("col-weather").innerHTML = "<img
src='./assets/Weather/Night/5.png'>"
    }
    if(data.weather[0].icon == "13d"){
```

```
      document.getElementById("col-weather").innerHTML = "<img
src='./assets/Weather/Day/7.png'>"
    }
    if(data.weather[0].icon == "13n"){
      document.getElementById("col-weather").innerHTML = "<img
src='./assets/Weather/Night/Nieve.png'>"
    }
    if(data.weather[0].icon == "50d"){
      document.getElementById("col-weather").innerHTML = "<img
src='./assets/Weather/Day/6.png'>"
    }
    if(data.weather[0].icon == "50n"){
      document.getElementById("col-weather").innerHTML = "<img
src='./assets/Weather/Night/6.png'>"
    }
    if(data.weather[0].description == "clear sky"){

      document.getElementById("col-description").innerHTML = "Cielo
Despejado";
    }
    if(data.weather[0].description == "few clouds"){

      document.getElementById("col-description").innerHTML = "Intervalos
Nubosos";
    }
    if(data.weather[0].description == "scattered clouds"){

      document.getElementById("col-description").innerHTML = "Nublado";
    }

    if(data.weather[0].description == "overcast clouds"){

      document.getElementById("col-description").innerHTML = "Nublado";
    }

    if(data.weather[0].description == "broken clouds"){

      document.getElementById("col-description").innerHTML = "Nublado";
    }
    if(data.weather[0].description == "shower rain"){

      document.getElementById("col-description").innerHTML = "LLuvia
débil";
    }
    if(data.weather[0].main == "Rain"){

      document.getElementById("col-description").innerHTML = "Lluvia";
    }
    if(data.weather[0].main == "Thunderstorm"){

      document.getElementById("col-description").innerHTML = "Tormenta";
    }
    if(data.weather[0].description == "snow"){

      document.getElementById("col-description").innerHTML = "Nieve";
    }
    if(data.weather[0].description == "mist"){

      document.getElementById("col-description").innerHTML = "Neblina";
    }
    });
```

11.4 Hojas de características

11.4.1 ESP32

Figura 11.1. Pines ESP32

3.2 External Flash and SRAM

ESP32 supports multiple external QSPI flash and SRAM chips. More details can be found in Chapter SPI in the *ESP32 Technical Reference Manual*. ESP32 also supports hardware encryption/decryption based on AES to protect developers' programs and data in flash.

ESP32 can access the external QSPI flash and SRAM through high-speed caches.

- The external flash can be mapped into CPU instruction memory space and read-only memory space simultaneously.
  - When external flash is mapped into CPU instruction memory space, up to 11 MB + 248 KB can be mapped at a time. Note that if more than 3 MB + 248 KB are mapped, cache performance will be reduced due to speculative reads by the CPU.
  - When external flash is mapped into read-only data memory space, up to 4 MB can be mapped at a time. 8-bit, 16-bit and 32-bit reads are supported.
- External SRAM can be mapped into CPU data memory space. Up to 4 MB can be mapped at a time. 8-bit, 16-bit and 32-bit reads and writes are supported.

ESP32-WROOM-32 integrates a 4 MB SPI flash, which is connected to GPIO6, GPIO7, GPIO8, GPIO9, GPIO10 and GPIO11. These six pins cannot be used as regular GPIOs.

Figura 11.2. Aspectos de la memoria Flash y SRAM

5 Electrical Characteristics

5.1 Absolute Maximum Ratings

Stresses beyond the absolute maximum ratings listed in the table below may cause permanent damage to the device. These are stress ratings only, and do not refer to the functional operation of the device that should follow the recommended operating conditions.

Table 11: Absolute Maximum Ratings

Symbol	Parameter	Min	Max	Unit
VDDA, VDD3P3, VDD3P3_RTC, VDD3P3_CPU, VDD_SDIO	Voltage applied to power supply pins per power domain	–0.3	3.6	V
I_{output} *	Cumulative IO output current	-	1200	mA
T_{store}	Storage temperature	–40	150	°C

* The chip worked properly after a 24-hour test in ambient temperature at 25 °C, and the IOs in three domains (VDD3P3_RTC, VDD3P3_CPU, VDD_SDIO) output high logic level to ground.

Figura 11.3. Características eléctricas absolutas.

5.2 Recommended Operating Conditions

Table 12: Recommended Operating Conditions

Symbol	Parameter	Min	Typ	Max	Unit
VDDA, VDD3P3_RTC,[note 1] VDD3P3, VDD_SDIO (3.3 V mode)[note 2]	Voltage applied to power supply pins per power domain	2.3/3.0[note 3]	3.3	3.6	V
VDD3P3_CPU	Voltage applied to power supply pin	1.8	3.3	3.6	V
I_{VDD}	Current delivered by external power supply	0.5	-	-	A
T [note 4]	Operating temperature	–40	-	125	°C

1. When writing eFuse, VDD3P3_RTC should be at least 3.3 V.
2. • VDD_SDIO works as the power supply for the related IO, and also for an external device. Please refer to the Appendix IO_MUX of this datasheet for more details.
   • VDD_SDIO can be sourced internally by the ESP32 from the VDD3P3_RTC power domain:
     – When VDD_SDIO operates at 3.3 V, it is driven directly by VDD3P3_RTC through a 6 Ω resistor, therefore, there will be some voltage drop from VDD3P3_RTC.
     – When VDD_SDIO operates at 1.8 V, it can be generated from ESP32's internal LDO. The maximum current this LDO can offer is 40 mA, and the output voltage range is 1.65 V ~ 2.0 V.
   • VDD_SDIO can also be driven by an external power supply.
   • Please refer to Power Scheme, section 2.3, for more information.
3. • Chips with a 3.3 V flash embedded: this minimum voltage is 3.0 V;
   • Chips with no flash: this minimum voltage is 2.3 V;
   • For more information, see Table 23 *ESP32 Ordering Information*.
4. The operating temperature of ESP32-U4WDH ranges from –40 °C to 105 °C, due to the flash embedded in it. The other chips in this series have no embedded flash, so their range of operating temperatures is –40 °C ~ 125 °C.

Figura 11.4. Condiciones de operación recomendadas del ESP32

11.4.2 LoRa E32 900T20D

2. Specification and parameter

2.1 Limit parameter

Main parameter	Performance		Remarks
	Min.	Max.	
Power supply（V）	0	5.5	Voltage over 5.5V will cause permanent damage to module
Blocking power（dBm）	-	-10	Chances of burn is slim when modules are used in short distance
Operating temperature（℃）	-40	85	

2.2 Operating parameter

Main parameter		Performance			Remark
		Min	Typ.	Max.	
Operating voltage（V）		2.3	5.0	5.5	≥5.0 V ensures output power
Communication level（V）			3.3		For 5V TTL, it may be at risk of burning down
Operating temperature（℃）		-40	-	85	Industrial design
Operating frequency（MHz）		862	868	931	Support ISM band
Power consumption	Transmitting current [mA]		120		Instant power consumption
	Receiving current [mA]		15		
	Turn-off current [μA]		4		Software is shut down
Max Tx power（dBm）			20		
Receiving sensitivity（dBm）			-123		Air data rate is 2.4kbps

Main parameter	Description	Remark
Distance for reference	5500m	Test condition：clear and open area, antenna gain: 5dBi, antenna height: 2.5m, air data rate: 2.4kbps
TX length	58 Byte	Maximum capacity of single package, automatic sub-packing after exceeding.
Buffer	512 Byte	
Modulation	LoRa™	
Communication interface	UART	@3.3V,TTL
Package	DIP	
Connector	DIP	1*7*2.54mm spacing
Size	21* 36 mm	
Antenna	SMA-K	50 ohm impedance

Figura 11.5. Aspectos eléctricos LoRa E32

6 Operating mode

There are four operating modes, which are set by M1 and M0, the details are as follows:

Mode（0-3）	M0	M1	Mode introduction	Remark
0 Normal	0	0	UART and wireless channel are open, transparent transmission is on	The receiver must work in mode 0 or mode 1
1 Wake up	1	0	UART and wireless channel are open, the only difference with mode 0 is that before transmitting data, increasing the wake up code automatically, so that it can awake the receiver under mode 3.	The receiver could be 0,1 or 2
2 Power saving	0	1	UART close, wireless is under air-awaken mode, after receiving data, UART open and send data.	Transmitter must be mode 1, unable to transmit in this mode.
3 Sleep	1	1	sleep mode, receiving parameter setting command is available.	More details on parameter specification.

Figura 11.6. Modos de operación del LoRa E32

7.1 Default parameters

	Default parameter values：：C0 00 00 1A 17 44						
Model	Frequency	Address	Channel	Air data rate	Baud rate	Parity	Transmitting power
E32-900T20D	868MHz	0x0000	0x06	2.4kbps	9600	8N1	100mW

Figura 11.7. Parámetros por defecto del LoRa E32

11.4.3 BME 280

1.1 General electrical specification

Table 1: Electrical parameter specification

Parameter	Symbol	Condition	Min	Typ	Max	Unit
Supply Voltage Internal Domains	V_{DD}	ripple max. 50 mVpp	1.71	1.8	3.6	V
Supply Voltage I/O Domain	V_{DDIO}		1.2	1.8	3.6	V
Sleep current	I_{DDSL}			0.1	0.3	µA
Standby current (inactive period of normal mode)	I_{DDSB}			0.2	0.5	µA
Current during humidity measurement	I_{DDH}	Max value at 85 °C		340		µA
Current during pressure measurement	I_{DDP}	Max value at -40 °C		714		µA
Current during temperature measurement	I_{DDT}	Max value at 85 °C		350		µA
Start-up time	$t_{startup}$	Time to first communication after both V_{DD} > 1.58 V and V_{DDIO} > 0.65 V			2	ms
Power supply rejection ratio (DC)	PSRR	full V_{DD} range			±0.01 ±5	%RH/V Pa/V
Standby time accuracy	$\Delta t_{standby}$			±5	±25	%

2. Absolute maximum ratings

The absolute maximum ratings are determined over complete temperature range using corner lots. The values are provided in Table 5.

Table 5: Absolute maximum ratings

Parameter	Condition	Min	Max	Unit
Voltage at any supply pin	V_{DD} and V_{DDIO} pin	-0.3	4.25	V
Voltage at any interface pin		-0.3	V_{DDIO} + 0.3	V
Storage temperature	≤ 65% RH	-45	+85	°C
Pressure		0	20 000	hPa
ESD	HBM, at any pin		±2	kV
	CDM		±500	V
	Machine model		±200	V
Condensation	No power supplied	Allowed		

Figura 11.8. Parámetros eléctricos del BME 280

11.4.4 MCP1700-3302e

MICROCHIP

MCP1700

Low Quiescent Current LDO

Features

- 1.6 µA Typical Quiescent Current
- Input Operating Voltage Range: 2.3V to 6.0V
- Output Voltage Range: 1.2V to 5.0V
- 250 mA Output Current for output voltages ≥ 2.5V
- 200 mA Output Current for output voltages < 2.5V
- Low Dropout (LDO) voltage
  - 178 mV typical @ 250 mA for V_{OUT} = 2.8V
- 0.4% Typical Output Voltage Tolerance
- Standard Output Voltage Options:
  - 1.2V, 1.8V, 2.5V, 3.0V, 3.3V, 5.0V
- Stable with 1.0 µF Ceramic Output capacitor
- Short-Circuit Protection
- Overtemperature Protection

Applications

- Battery-powered Devices
- Battery-powered Alarm Circuits
- Smoke Detectors
- CO^2 Detectors
- Pagers and Cellular Phones
- Smart Battery Packs
- Low Quiescent Current Voltage Reference
- PDAs
- Digital Cameras
- Microcontroller Power

Related Literature

Description

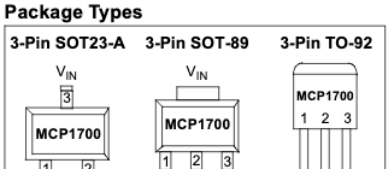
The MCP1700 is a family of CMOS low dropout (LDO) voltage regulators that can deliver up to 250 mA of current while consuming only 1.6 µA of quiescent current (typical). The input operating range is specified from 2.3V to 6.0V, making it an ideal choice for two and three primary cell battery-powered applications, as well as single cell Li-Ion-powered applications.

The MCP1700 is capable of delivering 250 mA with only 178 mV of input to output voltage differential (V_{OUT} = 2.8V). The output voltage tolerance of the MCP1700 is typically ±0.4% at +25°C and ±3% maximum over the operating junction temperature range of -40°C to +125°C.

Output voltages available for the MCP1700 range from 1.2V to 5.0V. The LDO output is stable when using only 1 µF output capacitance. Ceramic, tantalum or aluminum electrolytic capacitors can all be used for input and output. Overcurrent limit and overtemperature shutdown provide a robust solution for any application.

Package options include the SOT23, SOT89-3 and TO92.

Package Types



DC CHARACTERISTICS

Electrical Characteristics: Unless otherwise specified, all limits are established for V_{IN} = V_R + 1, I_{LOAD} = 100 µA, C_{OUT} = 1 µF (X5R), C_{IN} = 1 µF (X5R), T_A = +25°C.
Boldface type applies for junction temperatures, T_J **(Note 6)** of -40°C to +125°C.

Parameters	Sym	Min	Typ	Max	Units	Conditions
Input / Output Characteristics						
Input Operating Voltage	V_{IN}	**2.3**	—	**6.0**	V	**Note 1**
Input Quiescent Current	I_q	—	1.6	**4**	µA	I_L = 0 mA, V_{IN} = V_R +1V
Maximum Output Current	I_{OUT_mA}	**250** **200**	— —	— —	mA	For V_R ≥ 2.5V For V_R < 2.5V
Output Short Circuit Current	I_{OUT_SC}	—	408	—	mA	V_{IN} = V_R +1V, V_{OUT} = GND, Current (peak current) measured 10 ms after short is applied.
Output Voltage Regulation	V_{OUT}	**V_R-3.0%** V_R-2.0%	V_R±0.4 %	**V_R+3.0%** V_R+2.0%	V	**Note 2**
V_{OUT} Temperature Coefficient	TCV_{OUT}	—	50	—	ppm/°C	**Note 3**
Line Regulation	$\Delta V_{OUT}/(V_{OUT} X \Delta V_{IN})$	**-1.0**	±0.75	**+1.0**	%/V	(V_R+1)V ≤ V_{IN} ≤ 6V
Load Regulation	$\Delta V_{OUT}/V_{OUT}$	**-1.5**	±1.0	**+1.5**	%	I_L = 0.1 mA to 250 mA for V_R ≥ 2.5V I_L = 0.1 mA to 200 mA for V_R < 2.5V **Note 4**
Dropout Voltage V_R > 2.5V	V_{IN}-V_{OUT}	—	178	**350**	mV	I_L = 250 mA, **(Note 1, Note 5)**
Dropout Voltage V_R < 2.5V	V_{IN}-V_{OUT}	—	150	**350**	mV	I_L = 200 mA, **(Note 1, Note 5)**
Output Rise Time	T_R	—	500	—	µs	10% V_R to 90% V_R V_{IN} = 0V to 6V, R_L = 50Ω resistive

Note 1: The minimum V_{IN} must meet two conditions: V_{IN} ≥ 2.3V and V_{IN} ≥ (V_R + 3.0%) +$V_{DROPOUT}$.

2: V_R is the nominal regulator output voltage. For example: V_R = 1.2V, 1.5V, 1.8V, 2.5V, 2.8V, 3.0V, 3.3V, 4.0V, 5.0V. The input voltage (V_{IN} = V_R + 1.0V); I_{OUT} = 100 µA.

3: TCV_{OUT} = ($V_{OUT-HIGH}$ - $V_{OUT-LOW}$) *10^6 / (V_R * ΔTemperature), $V_{OUT-HIGH}$ = highest voltage measured over the temperature range. $V_{OUT-LOW}$ = lowest voltage measured over the temperature range.

4: Load regulation is measured at a constant junction temperature using low duty cycle pulse testing. Changes in output voltage due to heating effects are determined using thermal regulation specification TCV_{OUT}.

5: Dropout voltage is defined as the input to output differential at which the output voltage drops 2% below its measured value with a V_R + 1V differential applied.

6: The maximum allowable power dissipation is a function of ambient temperature, the maximum allowable junction temperature and the thermal resistance from junction to air (i.e., T_A, T_J, θ_{JA}). Exceeding the maximum allowable power dissipation will cause the device operating junction temperature to exceed the maximum 150°C rating. Sustained junction temperatures above 150°C can impact the device reliability.

Figura 11.9. Hoja de características del regulador MCP1700

11.5 Cálculos

11.5.1 Cálculos para la placa solar fotovoltaica y la batería

En primer lugar, sería conveniente reflejar en una tabla todos los datos de consumo de cada uno de los dispositivos que van a conformar la estación meteorológica, solo en el caso del transmisor.

Dispositivo	Consumo	
	[V]	[mA]
ESP32	3,3	80
LoRa E32 900T20D	3,3	120
BME 280	3,3	1
Anemómetro	[0-5]	-
Pluviómetro	[0-5]	-
Total	**201 [mA]**	

Tabla 11.1. Consumo del sistema transmisor.

Una vez se han reflejado los datos del consumo se puede realizar una estimación de la potencia de transmisión, cuando el LoRa empieza a emitir.

$$P_{tx} = V \cdot I = 3{,}3 \cdot 201 = 663{,}3\ [mW]$$

Como la transmisión se va a realizar cada 3 segundos, podremos calcular la potencia transmitida cada día. Se va a transmitir 1200 veces a la hora, es decir, 1/3 [h].

Dicho de otra manera, se transmitirían 28800 veces al día, que serían 8 horas diarias de transmisión. El sistema mientras no funciona no gasta energía.

Con esto se podrá calcular la potencia transmitida por hora y día.

$$P_{tx_h} = P_{tx} \cdot h = 663{,}3\ [mW] \cdot \frac{1}{3}[h] = 221\ [mWh]$$

A este cálculo sería conveniente sumarle un margen de seguridad del 15% para así evitar todo tipo de contingencias imprevistas.

$$P_{tx_h} + MS = P_{tx_h} \cdot 1{,}05 = 221 \cdot 1{,}15 = 254{,}15\ [mWh]$$

A continuación, se realizará un estudio solar, en la localidad de Jaén, para determinar cuantas horas de sol hay al día en el peor mes del año (siempre empleando un criterio pesimista).

Figura 11.10. Horas de luz en Jaén en el mes de diciembre

Como se puede ver, el peor mes es diciembre con una media de horas de luz al día de 9,60. Esto quiere decir, que la placa solar puede estar generando electricidad 9,60 horas al día. Esto ya refleja los días nublados.

Anteriormente se han expuesto los cálculos en niveles de potencia. En corriente los mAh de consumo son los siguientes:

$$I(mAh) = 201[mA] \cdot \frac{1}{3}[h] \cdot 1{,}15 = 77{,}05\ mAh$$

Que en un día son:

$$I(mAh/día) = 201[mA] \cdot \frac{1}{3}[h] \cdot 1{,}15 \cdot 24[h] = 1849{,}2 mAh/día$$

Es decir, el suministro de la placa tiene que ser de:

$$\frac{1849{,}2}{9{,}6} = 192{,}625\ mAh$$

Por lo que, con una placa solar de 2 [W] a 400 [mAh] será más que suficiente, además que la energía que sobre será almacenada por la batería. Si se considerara oportuno para garantizar aun más el funcionamiento del sistema se puede incorporar otra placa solar en paralelo.

La batería, por lo tanto, se pueden incorporar una de litio de 3,6 [V] a 1300 [mAh], o incluso 2 dentro del mismo portapilas.