UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Universitat Politècnica de València
Escuela Técnica Superior de Ingeniería Del Diseño

Sistema de desinfección mediante ozono con monitorización IoT

Trabajo Final del

Máster Universitario en Sensores para Aplicaciones Industriales

Curso Académico 2020/2021

Alumno: Héctor Pous Casas

Director: Rafael Masot Peris

Valencia, Junio de 2021

Resumen

En el presente trabajo se define el diseño e implementación de un sistema de desinfección mediante ozono con monitorización IoT que es capaz de eliminar patógenos como bacterias, hongos, protozoos o virus presentes en una estancia, en este proyecto se centra en el virus Sars-Cov-2. Además, adquiere y gestiona datos sobre el lugar en tiempo real, de manera que posibilita la consulta tiempo después.

Para el control del sistema se ha elegido la placa de desarrollo ESP32, que dispone de conectividad WIFI y bluetooth. En esta aplicación se utiliza una conexión WIFI para obtener acceso a la red.

Para el proceso de desinfección, utilizaremos cuatro actuadores: generador de ozono para producir la molécula O_3 la cual se encarga de la desinfección, ventiladores para proporcionar al sistema de la ventilación adecuada, pantalla LCD para la visualización del temporizador y zumbador para advertir del fin del proceso.

Los sensores empleados serán tres: MQ131 para la adquisición de la concentración de ozono, MQ135 para cuantificar la cantidad de CO_2 en el ambiente y DHT11 para obtener valores de temperatura y humedad. Durante este trabajo explicaremos tanto sus principios de funcionamiento como sus comunicaciones con el micro para poder gestionar los datos.

Por último, se realizará el montaje del sistema con las conexiones de cada uno de los distintos sensores y se explicará cómo observar los datos obtenidos en la aplicación móvil Blynk y en el servidor web ThingSpeak.

PALABRAS CLAVE: desinfección, ozono, *IoT, ESP32, NodeMCU,* sensores, *Blynk, ThingSpeak*

Abstract

This work defines the design and implementation of an ozone disinfection system with IoT monitoring that is capable of eliminating pathogens such as bacteria, fungi, protozoa or viruses present in a room, in this project it focuses on the Sars-Cov virus -2. In addition, it acquires and manages data about the place in real time, so that it can be consulted later.

To control the system, the ESP32 development board has been chosen, which has WIFI and bluetooth connectivity. In this application a WIFI connection is used to gain access to the network.

For the disinfection process, we will use four actuators: ozone generator to produce the O_3 molecule which is responsible for disinfection, fans to provide the system with adequate ventilation, LCD screen to display the timer and buzzer to warn of the End of the process.

The sensors used will be three: MQ131 to acquire the ozone concentration, MQ135 to quantify the amount of CO_2 in the environment and DHT11 to obtain temperature and humidity values. During this work we will explain both its operating principles and its communications with the micro in order to manage the data.

Finally, the assembly of the system will be carried out with the connections of each of the different sensors and it will be explained how to observe the data obtained in the Blynk mobile application and in the ThingSpeak web server.

KEYWORDS: disinfection, ozone, IoT, ESP32, NodeMCU, sensors, Blynk, ThingSpeak

Resum

En el present treball es defineix el disseny i implementació d'un sistema de desinfecció mitjançant ozó amb monitoratge IoT que és capaç d'eliminar patògens com a bacteris, fongs, protozous o virus presents en una estada, en aquest projecte se centra en el virus Sars-Cov-2. A més, adquireix i gestiona dades sobre el lloc en temps real, de manera que possibilita la consulta temps després.

Per al control del sistema s'ha triat la placa de desenvolupament ESP32, que disposa de connectivitat WIFI i bluetooth. En aquesta aplicació s'utilitza una connexió WIFI per a obtindre accés a la xarxa.

Per al procés de desinfecció, utilitzarem quatre actuadors: generador d'ozó per a produir la molècula O_3 la qual s'encarrega de la desinfecció, ventiladors per a proporcionar al sistema de la ventilació adequada, pantalla LCD per a la visualització del temporitzador i brunzidor per a advertir de la fi del procés.

Els sensors emprats seran tres: MQ131 per a l'adquisició de la concentració d'ozó, MQ135 per a quantificar la quantitat de CO_2 en l'ambient i DHT11 per a obtindre valors de temperatura i humitat. Durant aquest treball explicarem tant els seus principis de funcionament com les seues comunicacions amb el micro per a poder gestionar les dades.

Finalment, es realitzarà el muntatge del sistema amb les connexions de cadascun dels diferents sensors i s'explicarà com observar les dades obtingudes en l'aplicació mòbil Blynk i en el servidor web ThingSpeak.

PARAULES CLAU: desinfecció, ozó, IoT, ESP32, NodeMCU, sensors, Blynk, ThingSpeak

Índice general

Resumen III

Abstract V

Resum VII

Índice general IX

1 Introducción y aspectos generales 1

1.1 Introducción 1

1.2 Objetivo y motivación del presente trabajo 2

1.3 Diagrama de conexiones 2

2 Material y métodos 5

2.1 Sensores y tratamientos previos 5

2.2 Actuadores y procesos 21

2.3 La placa NodeMCU 25

2.4 Programación 29

2.5 Prototipo 32

3 Resultados 36

4 Conclusión 39

Bibliografía 41

APÉNDICES 43

Índice de figuras

Figura 1.1: Generador ozono comercial 1
Figura 1.2: Organización Mundial de la salud 2
Figura 1.3: Ecosistema IoT 2
Figura 1. 4: Diagrama de conexiones. 3
Figura 1. 5: Diagrama de bloques. 3

Figura 2. 1: Encapsulado 3 pines DHT11 6
Figura 2. 2: Aplicación estándar. 6
Figura 2. 3: Inicio de la comunicación Micro-DHT11 7
Figura 2. 4: Respuesta del DHT11 al micro. 8
Figura 2. 5: Estructura de la trama de datos. 8
Figura 2. 6: Sensor de calidad del aire MQ-135 9
Figura 2. 7: Circuito eléctrico MQ-135 10
Figura 2. 8: Sensor detector de $O3$ MQ-131 11
Figura 2. 9: Circuito eléctrico MQ 131 12
Figura 2. 10: Circuito eléctrico placa MQ131 14
Figura 2. 11: Curva de sensibilidad del MQ135 y condiciones ambientales idóneas 15
Figura 2. 12: Ecuación MQ135 para detección de $CO2$ 16
Figura 2. 13: Modificación de rzero en librería MQ135.h 17
Figura 2. 14: Programa calibración y obtención rzero MQ135 17
Figura 2. 15: Curva de sensibilidad del sensor MQ-131 / condiciones de temperatura 18
Figura 2. 16: Ecuación MQ131 para detección de $O3$ 18
Figura 2. 17: Cambio valor RL en librería MQ131 19
Figura 2. 18: Cambio conversión valor voltaje en librería ESP32 20
Figura 2. 19: Modificación de rzero en librería MQ135.h 20
Figura 2. 20: Programa calibración y obtención rzero MQ131 21
Figura 2. 21: Generador de ozono con sistema de ventilación 22
Figura 2. 22: Ventiladores del sistema de desinfección 23
Figura 2. 23: Protocolo de comunicación I2C 24
Figura 2. 24: Conexiones pantalla LCD 1602 24
Figura 2. 25: Zumbador 25
Figura 2. 26: Diagrama de bloques del SoC ESP32 26
Figura 2. 27: Placa de desarrollo ESP32 DEV KIT DOIT 29
Figura 2. 28: Contraseña única del proyecto de Blynk 30
Figura 2. 29: Configuración comunicación con Blynk 31
Figura 2. 30: Envío de datos a Blynk y ThingSpeak 31
Figura 2. 31: Obtención datos de los sensores 31

Figura 2. 32: Parámetros canal ThingSpeak 32
Figura 2. 33: Inicialización de la comunicación con ThingSpeak 32
Figura 2. 34: Esquema eléctrico del prototipo 33
Figura 2. 35: PCB proyecto previo 34
Figura 2. 36: Estructura para instalar el sistema en el armario 34

Figura 3. 1: App Blynk 36
Figura 3. 2: Visualización de los datos de temperatura DHT11 37
Figura 3. 3: Visualización de los datos de humedad DHT11..... 37
Figura 3. 4: Visualización de los datos de concentración de $CO2$ MQ-135 37
Figura 3. 5: Visualización datos de concentración de ozono MQ-131 37
Figura 3. 6: Alarma máxima de concentración de ozono permitida por la OMS 38
Figura 3. 7: Alarma máxima de concentración de $CO2$ permitida por la OMS..... 38
Figura 3. 8: Piloto para visualizar cuando está funcionando el proceso 38

Índice de tablas

Tabla 2. 1: Características técnicas DHT11 7
Tabla 2. 2: Rango dinámico sensor MQ131 después de modificación 14
Tabla 2. 3: Diferencias en librería MQ131 por microcontrolador 19
Tabla 2. 4: Características generador ozono cerámico 22
Tabla 2. 5: Características ventilador del sistema de desinfección 23
Tabla 2. 6: Comparativa ESP8266 / ESP32 27
Tabla 2. 7: Diferencias entre los distintos chips de la familia ESP32 28
Tabla 2. 8: Diferencias entre los distintos módulos de la familia ESP32 28

Capítulo 1

Introducción y aspectos generales

1.1 Introducción

En el presente trabajo de estudio se expondrá información sobre la eficacia del ozono como desinfectante frente al coronavirus SARS-CoV-2, todo motivado por la actual crisis sanitaria que afecta a toda la población mundial. Además, se mostrará el diseño e implementación de un sistema de desinfección mediante ozono con monitorización IoT (Internet of Things).

La desinfección es un proceso químico que mata o erradica los microorganismos sin discriminación como las bacterias, virus y protozoos impidiendo el crecimiento de microorganismos patógenos en fase vegetativa que se encuentren en objetos inertes.

El ser humano fue mejorando sus métodos de desinfección a lo largo de la historia, primero con métodos simples como el ahumado, desecación, salazón o la condimentación practicada desde la época prehistórica. Continuando con la utilización de sahumerios de azufre y alquitrán por los griegos y romanos para desinfectar las cloacas. Así se fueron mejorando los métodos de desinfección hasta llegar en la actualidad.

A día de hoy la ciencia ha podido desarrollar una gran cantidad de métodos de desinfección con los cuales asegurar unas condiciones óptimas en diferentes ámbitos de la vida tanto personal como laboral.

Para garantizar una eficiente desinfección se debe monitorizar diferentes parámetros antes, durante y después de esta como la temperatura, humedad, concentración del desinfectante, concentración de gases, etc. Con este propósito se utilizan los sensores que son dispositivos capaces de obtener magnitudes físicas o químicas de su entorno.

Motivados por la pandemia mundial causada por el SARS-CoV-2, la búsqueda de mejoras en la desinfección se ha acentuado notablemente. Un ejemplo importante son los generadores de ozono, los cuales mediante la formación de la molécula O_3 pretenden desinfectar una estancia, superficie o materiales inertes.

Figura 1.1: Generador ozono comercial

1.2 Objetivo y motivación del presente trabajo

El objeto de este trabajo es poder realizar el diseño e implementación de un sistema de desinfección IoT capaz de mostrar los datos a través de un dispositivo Smartphone o de una web mediante un PC.

Principalmente el proyecto presenta dos motivaciones diferentes, primero se tiene una motivación sanitaria debida a la situación mundial por el SARS-CoV-2, el cual impulsa a encontrar nuevas formas más eficientes de combatir el virus.

Por otro lado, se busca tener la certeza de una desinfección eficiente, por este motivo, se monitorizan los parámetros del entorno y se almacenan de forma adecuada para una visualización práctica. Por tanto, la segunda motivación del trabajo es la posibilidad del usuario en saber en todo momento y lugar del estado del proceso.

Figura 1.2: Organización Mundial de la salud

Figura 1.3: Ecosistema IoT

1.3 Diagrama de conexiones

Diversos fabricantes han desarrollado generadores de ozono para diversas aplicaciones dentro de diferentes campos como la alimentación, textil, automovilismo, sanitario, hogar, etc.

Los generadores de ozono que están en el mercado principalmente cuentan con un generador, ventiladores y un temporizador analógico o digital para programar la duración de la desinfección. El generador que se presenta en este trabajo, a diferencia de los demás generadores, cuenta con una conexión wifi con la cual se puede realizar el almacenamiento y visualización de los parámetros del entorno durante el proceso de desinfección. Por lo tanto, se consigue un generador de ozono IoT que da la posibilidad al usuario de conocer el estado del proceso en todo momento.

Para su montaje, se ha utilizado una placa basada en el ESP-WROOM-32, la NodeMCU que dispone de conectividad Wifi y Bluetooth para comunicar, en este caso se emplea el modo Wifi, con un router que disponga de acceso a internet. La programación de este controlador es intuitiva, pues se puede realizar utilizando el IDE de arduino, un entorno muy extendido por estudiantes en ingeniería y que contiene numerosas librerías de multitud de sensores que facilitan la estandarización para la programación.

A continuación, se presenta el diagrama de conexiones del proyecto y el diagrama de bloques del proceso completo.

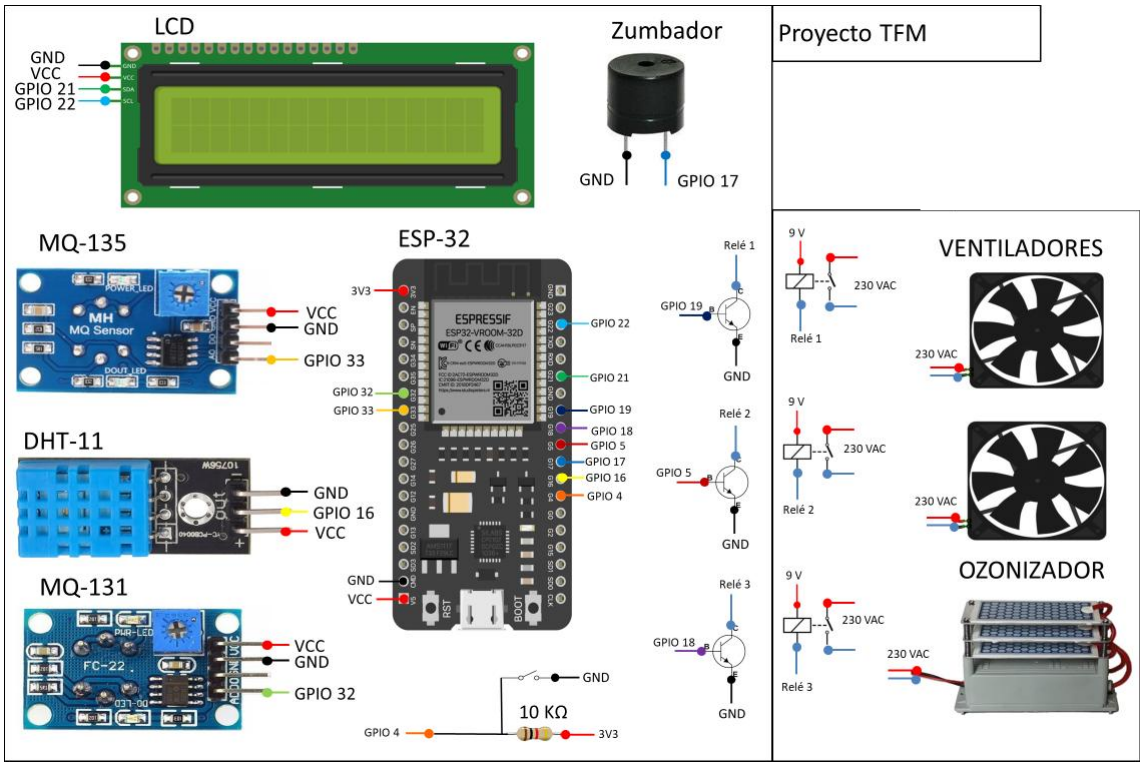


Figura 1. 4: Diagrama de conexiones eléctricas.

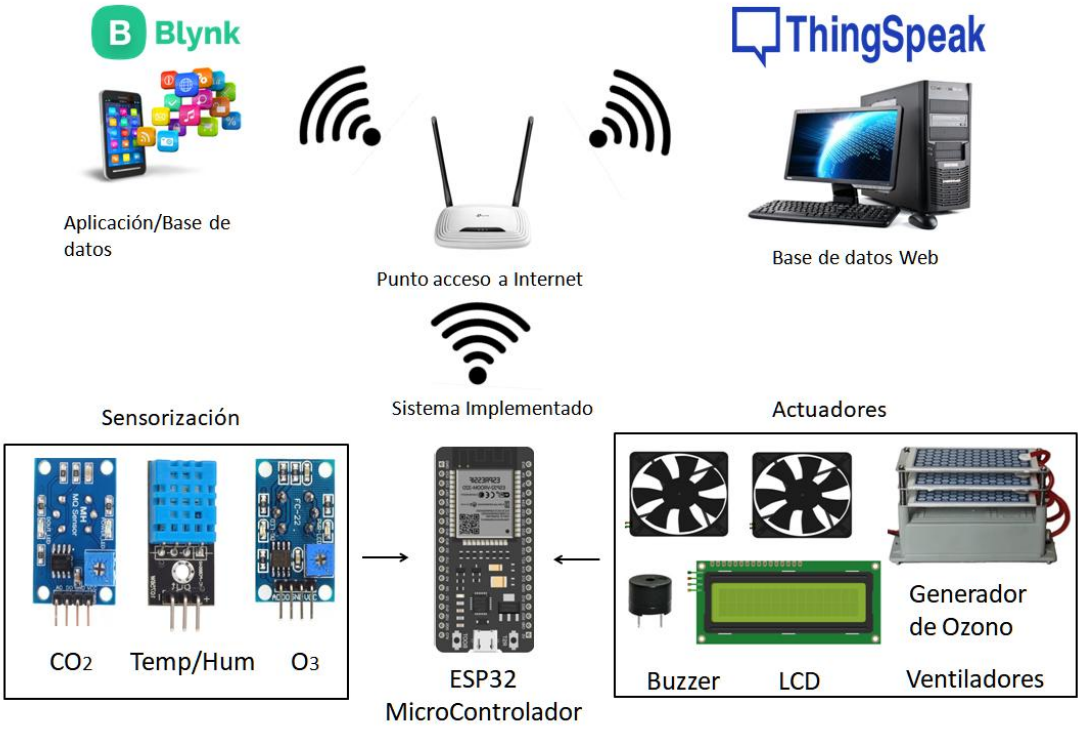


Figura 1. 5: Diagrama de bloques.

Capítulo 2

Material y métodos

En este capítulo, se expondrán los materiales empleados, especialmente en los sensores y tratamientos previos para su utilización. Además, se explicará el proceso de desinfección que lleva a cabo el sistema y se profundizará en cada uno de sus actuadores, en especial el generador de ozono. También se describirá el diseño y construcción de la placa de circuito impreso (PCB) que se encarga de controlar el proceso, mediante el NodeMCU.

Por último, se expondrán el diseño a nivel eléctrico del prototipo, así como los aspectos más importantes de la programación para el microcontrolador en el entorno IDE de arduino.

2.1 Sensores y tratamientos previos

Durante este apartado, se describirán meticulosamente los diferentes sensores elegidos para el diseño del sistema de desinfección y sus tratamientos previos necesarios. Se realizará un estudio exhaustivo de cada uno de ellos, explicando su funcionamiento y como se han acondicionado para el diseño, capacitando de una adquisición de datos de humedad, temperatura, concentración de ozono y concentración de CO_2 en el aire.

Para cada una de estas mediciones, se han elegido una clasificación de sensores de bajo coste que posibilitan la adquisición de datos con la precisión y estabilidad requeridas para la aplicación, estos sensores son adecuados porque el proceso no necesita de elevada precisión para ser fiable.

A continuación, se enumeran los diferentes sensores utilizados y la magnitud que son capaces de detectar.

1. DHT11 para la medida de temperatura y humedad
2. MQ-135 para la medida de concentración de CO_2
3. MQ-131 para la medida de la concentración de O_3

2.1.1 DHT11

Descripción y principio de funcionamiento

El sensor DHT11 es un sensor físico con la capacidad de medir la temperatura y la humedad del entorno con una idónea alta fiabilidad y estabilidad a causa de la señal digital calibrada que nos proporciona y que evita la transmisión de ruido.

El principio de funcionamiento de este sensor está basado en dos elementos, por un lado, tenemos una resistencia variable en función de la humedad y un termistor NTC sensible a la temperatura que se conectan a un microcontrolador de 8 bits.

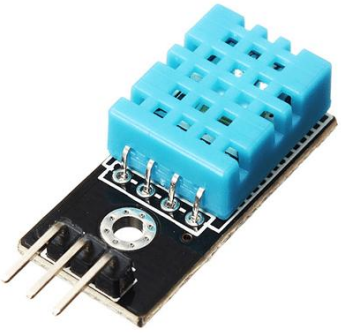


Figura 2. 1: Encapsulado 3 pines DHT11

El usuario no necesita calibrar el sensor, debido a que los fabricantes nos garantizan que cada componente ha sido calibrado en sus instalaciones y que, cada coeficiente de calibración ha sido colocado en el firmware de la memoria OTP del micro.

Existen dos versiones del DHT11. Por una parte, encontramos el sensor aislado que dispone de cuatro pines y necesita la colocación de una resistencia de Pull-Up de 5 k entre el pin de la alimentación y el pin de datos, por otro lado, está la versión con 3 pines (la escogida en el diseño), como la que se observa en la figura 2.1 en la que dicha resistencia de Pull-Up ha sido incluida en la PCB, por lo que no es necesario su colocación por parte del usuario.

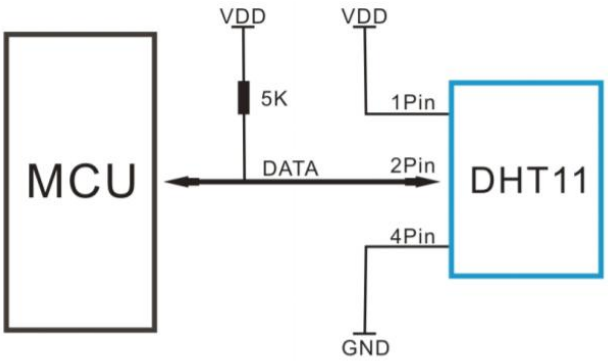


Figura 2. 2: Aplicación estándar.

Características técnicas

Las características técnicas del DHT11 se han obtenido del Datasheet de un fabricante en particular, pero todos son capaces de proporcionar valores precisos. Hay que tener en cuenta que se puede alimentar desde los 3,5V hasta los 5V con una intensidad de consumo de 2,5 mA, los distintos factores relevantes en referencia a la sensorización se muestran en la tabla 2.1.

DHT11	Temperatura (ºC)	Humedad (% RH)
Rango	0 - 50	20 - 90
Precisión	A 25 ºC ± 2	De 0ºC – 50 ºC ±5
Resolución	1	1

Tabla 2. 1: Características técnicas DHT11

Transmisión de datos

El DHT11 no utiliza un protocolo de datos serie estándar como puede ser el SPI, el I2C, o 1Wire, aunque si se trata de una comunicación a un solo hilo de manera bidireccional bastante sencillo.

El micro externo es quien debe iniciar la comunicación manteniendo la línea de datos en estado bajo durante al menos 18µs (en reposo, la señal está en estado alto). Después, el DHT11 responde con un pulso a nivel bajo de 80µs y deja la línea flotando (a nivel alto de nuevo) durante 80µs, preparándose para enviar el dato, como se puede observar en la figura 2.3.

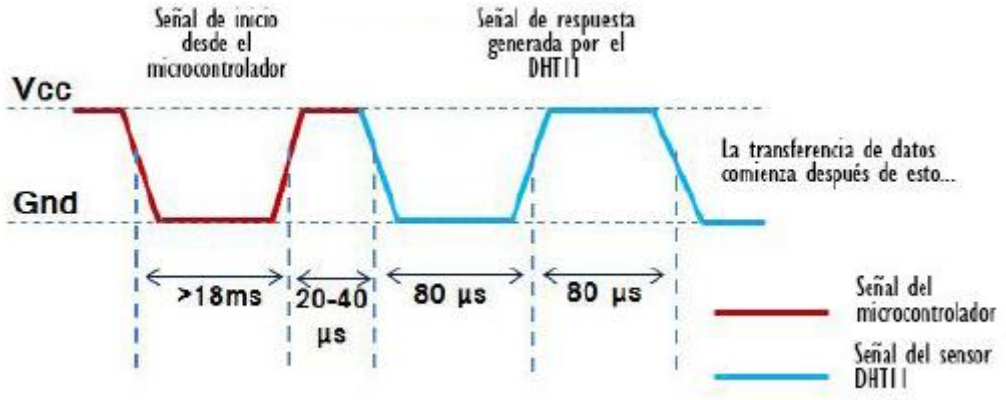


Figura 2. 3: Inicio de la comunicación Micro-DHT11

Cuando el DHT cambia el estado del bus a bajo, ya está enviando primer bit del dato. Todos los bits de dato comienzan con un pulso bajo que dura 50µs que las librerías emplean como pulso para la sincronización.

Tras esto, se envía un pulso que, dependiendo del dato, tiene una duración entre 26µs y 28µs si es un "0", o de 70µs si se trata de un "1", tal y como se observa en la figura 2.4.

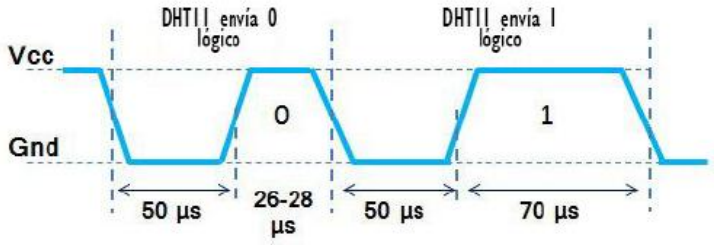


Figura 2. 4: Respuesta del DHT11 al micro.

El micro del DHT11 permanece en reposo hasta que recibe la señal de Start del microcontrolador externo para enviar la trama de datos.

Esta trama está codificada como se muestra en la figura 2.5, es decir, consta de 40 bits en los que se incluyen todos los datos que el sensor es capaz de proporcionar de manera que el primer byte corresponde a la parte entera de la medición de la humedad relativa, el segundo a la parte decimal de esta (que siempre es 0), el tercer byte corresponde a la parte entera de la temperatura, mientras que el cuarto a la parte decimal de esta (se puede emplear, aunque debido a la baja precisión del sensor en muchas ocasiones no se muestra).

Por último, el quinto byte es el checksum de todos los bytes anteriores, es decir, la suma de comprobación.

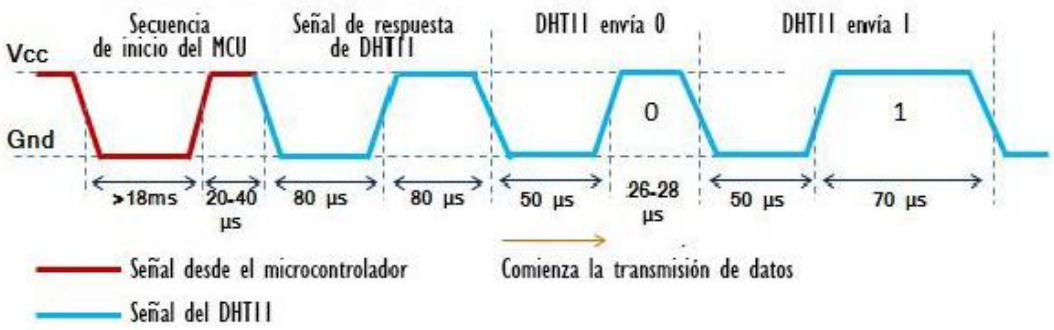


Figura 2. 5: Estructura de la trama de datos.

2.1.2 MQ-135

Descripción y principio de funcionamiento

El MQ135 (figura 2.6) es un sensor electro-químico analógico con la capacidad de detectar gases tóxicos que disminuyen la calidad del aire en recintos cerrados de un tamaño considerable, lo que le hace óptimo para emplearse tanto en hogares como en oficinas.

Figura 2. 6: Sensor de calidad del aire MQ-135

El elemento sensor está constituido por dióxido de estaño (SnO_2) cuya conductividad es directamente proporcional a la contaminación del aire en el que se encuentra.

Este sensor es capaz de reaccionar con:

- Amoníaco (NH_3)
- Cualquier óxido de nitrógeno (NOx)
- Benceno
- Humo
- Dióxido de carbono (CO_2)

La sensibilidad del sensor para todos estos gases es prácticamente la misma. Si se tiene en cuenta que la cantidad de CO_2 presente en la atmósfera es mucho mayor en proporción a los demás gases que detecta, podemos determinar que en la atmósfera de un hogar familiar este gas será el único con presencia efectiva.

Por esto, se decide que este sensor se calibrará y tendrá en cuenta solo para la detección de CO_2 en espacios cerrados.

La PCB donde se encuentra el sensor dispone de cuatro pines. Estos cuatro pines son para la tensión de alimentación, la masa y dos salidas, una analógica con el valor medido por el sensor y otra digital que se puede pre-configurar para activarla si el valor medido supera un umbral definido manualmente mediante un potenciómetro que también se ubica en la placa.

Uno de los pilares de este sensor es la presencia de una resistencia de heater interna que agiliza la reacción del gas con el elemento sensor, como se observa en la figura 2.7.

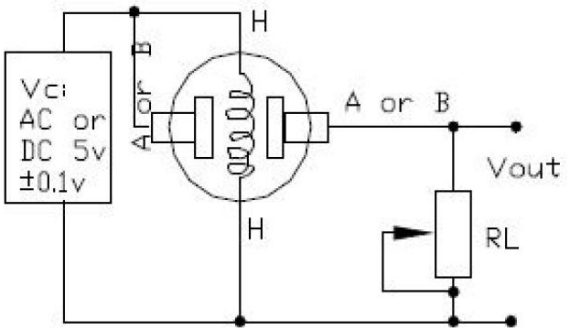


Figura 2. 7: Circuito eléctrico MQ-135

Esta resistencia favorece a la variación de la resistencia entre A y B que junto a la resistencia de carga RL empleada para cerrar el circuito, se forma un divisor de tensión que proporciona la medida para ser leída por el NodeMCU.

Características técnicas

El circuito del MQ-135 se alimenta a 5V, así como su resistencia de calentamiento; se pueden alimentar ambos elementos con los mismos 5V sin ningún tipo de problema.

Las características técnicas más importantes y que hacen referencia al entorno en el que se encuentra, son las siguientes:

- Temperatura de trabajo entre -10 ºC y 45 ºC
- Humedad relativa máxima del 95% Rh
- Valores de oxígeno entre 2% y 21%
- Precalentamiento de 24h

Transmisión de datos

Este sensor es capaz de comunicarse con un micro que lo controle, de dos formas distintas.

Por un lado, dispone de un pin analógico que nos ofrece el valor de voltaje en bornes de la resistencia de carga RL que es procesado en el ESP32 para dar una medida de concentración de CO_2.

Por el otro lado, dispone de un pin digital que nos ofrece un valor de 1 o 0 en función del voltaje medido, es decir, depende si la medida es mayor o menor al umbral prefijado. Este umbral se define por el usuario con una resistencia variable que viene en la PCB donde se encuentra el sensor.

En ninguno de los casos se emplea un protocolo de comunicación establecido, ya que lo único que realiza el micro es la lectura del estado de ambos pines, el valor analógico medido o el nivel alto/bajo del pin digital.

2.1.3 MQ-131

Descripción y principio de funcionamiento

El MQ-131 (figura 2.8) es un sensor electro-químico analógico capaz de detectar la molécula de ozono (O_3), el sensor disminuye su conductividad a mayor concentración de O_3. Presenta una aceptable precisión en la medida, por lo cual, es ideal para aplicaciones como detección de ozono doméstico, en un ámbito industrial o en generadores portátiles.

Figura 2. 8: Sensor detector de O_3 MQ-131

El elemento sensor está constituido por trióxido de wolframio (WO_3) cuya conductividad es inversamente proporcional a la concentración de ozono (O_3) en el ambiente.

Aparte de reaccionar con el ozono, este sensor es capaz de reaccionar a otros óxidos fuertes como:

- Cloro (Cl_2)
- Cualquier óxido de nitrógeno (NOx)

La mayor sensibilidad del sensor es para la molécula de ozono. Hay que tener en cuenta que el sensor está dentro de un entorno en el cual dispondremos de un generador de ozono, por tanto, este gas estará en mayor proporción que los demás, se puede de determinar que en nuestra atmosfera objetivo este gas será el único en presencia efectiva.

Por esto, se decide que este sensor se calibrará y utilizará solo para la detección de O_3 en espacios cerrados.

El sensor elegido esta colocado en una PCB (placa de circuito impreso), el cual posee cuatro pines. Estos cuatro pines son la tensión de alimentación, la masa y dos salidas, una digital que se puede pre-configurar para poder activarla si la medida es mayor a un umbral que se define mediante un potenciómetro que se haya en la PCB, y una analógica que proporciona el valor medido por el sensor.

Como en toda la familia de sensores MQ, este sensor presenta una resistencia calefactora interna que facilita la reacción del gas con el elemento sensor, como se observa en la figura 2.9.

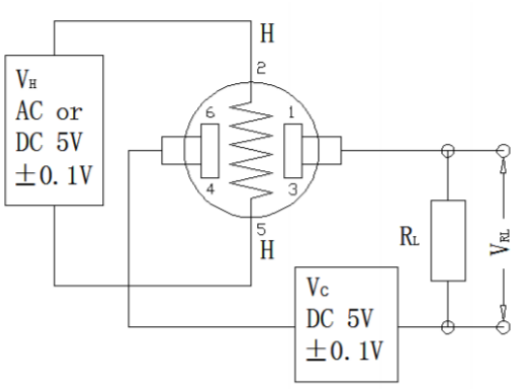


Figura 2. 9: Circuito eléctrico MQ 131

Esta resistencia facilita la variación de la resistencia existente entre A y B que con la resistencia de carga RL utilizada para cerrar el circuito, se forma un divisor de tensión que proporciona la medida para ser leída por el NodeMCU.

Características técnicas

El circuito del MQ-131 se alimenta a 5V, así como la resistencia calefactora; ambas se pueden alimentar con los mismos 5V sin ningún inconveniente.

Las características técnicas más significativas y que también se refieren al ambiente donde se halla el sensor, son:

- Concentración de detección: 0,01>>>1ppm de ozono
- Consumo de energía de calefacción PH: ≤ 900 Mw
- Precalentamiento: ≥ 48 horas
- Condiciones Temperatura: 20ºC ± 2ºC
- Condiciones Humedad: 55% ± 5%

Transmisión de datos

El MQ-131 presenta el mismo modo de transmisión que el sensor anteriormente estudiado MQ-135.

Por un lado, dispone de un pin analógico que nos ofrece el valor de voltaje en bornes de la resistencia de carga RL que es procesado en el ESP32 para dar una medida de concentración de O_3.

Por otro lado, también dispone de una salida digital la cual dependiendo del umbral prefijado nos proporciona un 1 o 0 lógico. Este umbral es fijado manualmente por el usuario a través del potenciómetro de la PCB.

No se puede considerar como un protocolo de comunicación establecida como tal, porque lo único que realiza el micro es la lectura del estado de estos dos pines, tanto el valor analógico como el digital.

2.1.4 Tratamientos previos

En este apartado se expondrán los procesos de calibración necesarios para la utilización de los diferentes sensores del proyecto. No necesitaremos preconfigurarlos porque vienen configurados de fábrica o no presentan un protocolo de comunicación que lo necesite.

Estos procesos se han de realizar para cada sensor en específico, es decir, para cada hardware físico y no por modelo ya que cada uno es único por comprador, esto se realiza para garantizar una medida fiable y racional.

DHT11

Como se expone en el apartado 2.1.1, el sensor de temperatura y humedad DHT11 no necesita de ningún tipo de preconfiguración por parte del usuario. Esto es debido a que los fabricantes aseguran que se realiza una calibración de cada una de las unidades que van a ser vendidas en sus instalaciones, además, se someten a un test minucioso de funcionamiento.

MQ135

El sensor MQ135 precisa de una precalibración exhaustiva antes de ser utilizado para realizar medidas. Esto se debe a que el sensor presenta diferentes curvas de sensibilidad para diferentes gases, aunque esta sensibilidad es parecida entre gases, se debe seguir la curva correspondiente al gas objetivo, en este caso el CO_2.

MQ131

El sensor MQ131 presenta el mismo proceso de calibración que el MQ135. Esto es debido a que son sensores de la misma familia y que también presenta diferentes sensibilidades a diferentes gases. En este caso el gas objetivo será el ozono (O_3). Además, este sensor precisa de unas modificaciones físicas en la placa para mejorar su funcionamiento.

2.1.5 Procesos de preconfiguración realizados

MQ131

En este apartado se expondrán los cambios realizados a la placa del MQ131 y sus motivos. Principalmente se han realizado dos cambios, la modificación de la resistencia de carga y el aumento de la resistencia calefactora.

El primer cambio es debido a que con la resistencia original de 10kΩ, el rango dinámico del sensor era muy pequeño, es decir, para diferentes valores de concentración de ozono el sensor nos proporcionaba valores de tensión demasiado pequeños para su tratamiento. Por tanto, después de hacer varias pruebas se optó por una RL de 680 kΩ.

Concentración Ozono (o_3)	Tensión salida (V_{RL})
10 ppb	4,20 V
1000 ppb	2,22 V
2000 ppb	1,86 V

Tabla 2. 2: Rango dinámico sensor MQ131 después de modificación

El segundo cambio se realizó para que la resistencia calefactora obtuviera su temperatura óptima más rápido, porque si no el tiempo hasta estabilizarse es demasiado extenso, dificultando su calibración y obteniendo medidas inestables. Esto provoca que a medida que pasa el tiempo las medidas para un mismo valor varían. Por tanto, se añadió una resistencia en serie de 47Ω a la R3 de 5R1Ω. Para ello, se tuvo que cortar la pista que unía el pin 5 con la R3 para poder soldar en serie la nueva resistencia.

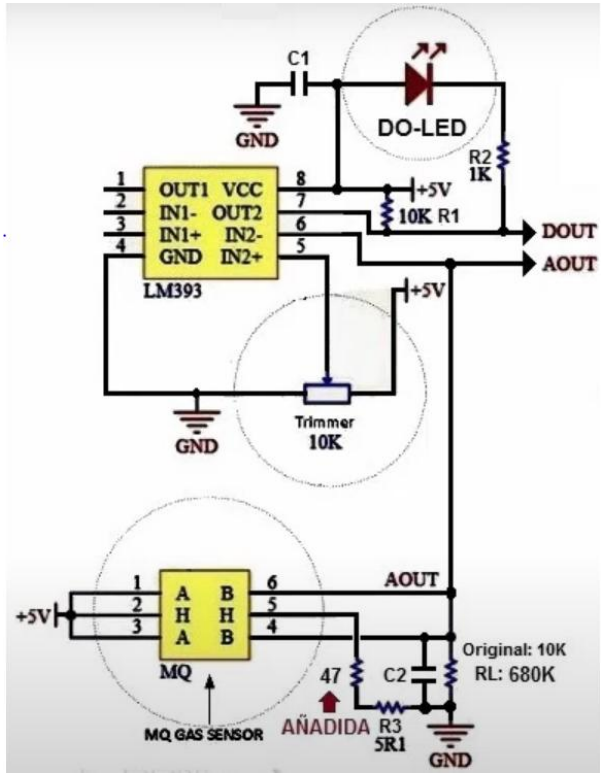


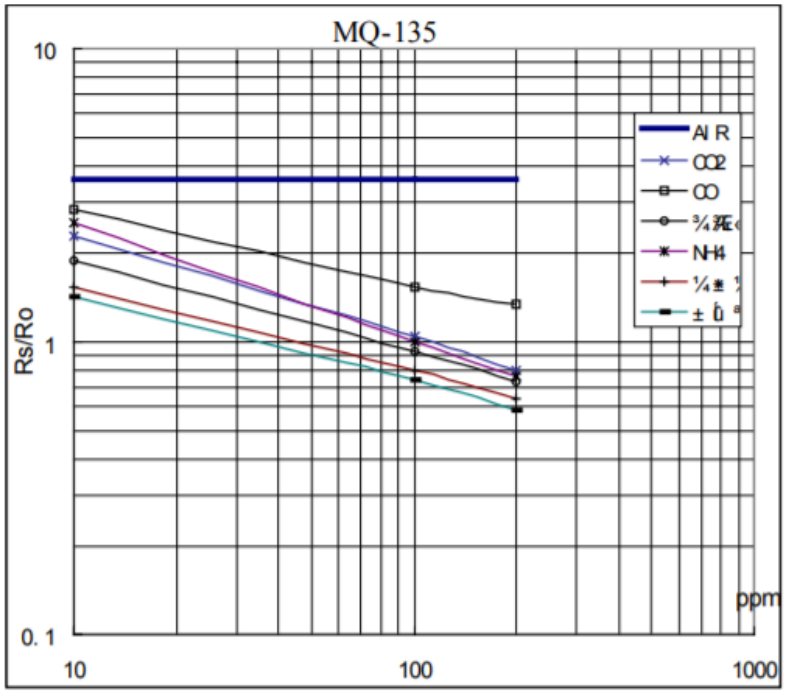
Figura 2. 10: Circuito eléctrico placa MQ131

En la figura (2.10) se pueden observar los dos cambios realizados, con esto limitamos la corriente, reducimos el consumo del circuito y el calentamiento de la resistencia calefactora se estabiliza mucho antes. Esto es debido, basándonos en la ley de Ohm, que al aumentar la resistencia (R) disminuye la corriente (I) y por tanto el consumo. Además, esto favorece al calentamiento de la resistencia calefactora.

2.1.6 Procesos de calibración empleados

MQ135

La calibración de este sensor precisa de un método manual para escalar adecuadamente el valor leído. El problema que se observa, es que la relación entre la medida y el valor real no es lineal, por lo que se debe estimar la curva de sensibilidad que presenta el fabricante en su datasheet.



Curvas de sensibilidad del MQ-135 para distintos gases, en donde:
Tª: 20 ˚C; Humedad: 65%;
Concentración de O_2: 21%;
R_L= 20 KΩ
Ro= Resistencia del sensor a 100 ppm de NH_3 en el aire limpio
R_S = Resistencia del sensor a varias concentraciones de gases

Figura 2. 11: Curva de sensibilidad del MQ135 y condiciones ambientales idóneas

Es fundamental que se presenten las condiciones ambientales idóneas como temperatura a 20ºC, humedad relativa rondando los 65% de HR con una concentración de oxígeno del 21% (presente en cualquier atmósfera exterior normal) y con la resistencia de carga RL asignada a 20 kΩ.

El fabricante nos proporciona la curva de sensibilidad (figura 2.11) y no una ecuación. Por tanto, se ha de hallar de forma empírica la ecuación de funcionamiento de cada sensor individual.

En este caso, la ecuación que se obtiene tras representar gráficamente la curva de CO_2 es la siguiente en la figura 2.12.

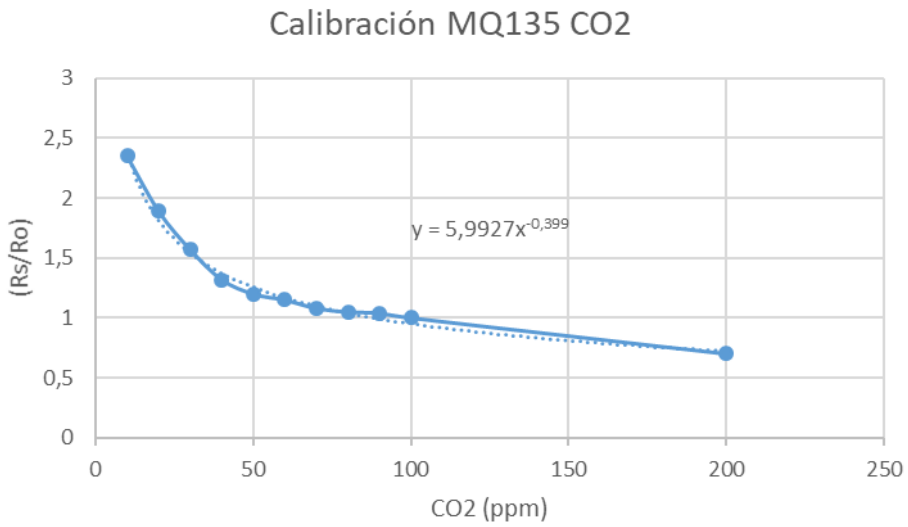


Figura 2. 12: Ecuación MQ135 para detección de CO_2

Por tanto, si sustituimos para los valores que queremos:

(2.1)

$$\left(\frac{Rs}{Ro}\right) = 5{,}9927 * CO_2{}^{(-0.399)}$$

Donde CO_2 es el valor de la concentración de CO_2 en ppm, R_o es la constante única de cada sensor cuando se expone a una concentración de 100 ppm de NH_3 y R_s la resistencia del sensor que se debe leer desde el controlador.

En estos tiempos, es muy difícil asegurar una atmósfera con las condiciones específicas de 100 ppm de NH_3, por lo que se ha tenido que buscar otra metodología de calibración para este sensor.

Este método alternativo de calibración para el sensor MQ-135 fue descrito por Georg Krocker que realizó una biblioteca específica, para el entorno del IDE de arduino, para poder realizar el calibrado en condiciones ambientales normales.

Krocker se basa en que el CO_2 es el cuarto gas más abundante en la atmósfera terrestre con una concentración aproximada de 400 ppm en exteriores (solo superado por el N_2, el O_2 y el Ar que son, por orden, los tres más abundantes).

El primer paso de la calibración es la carga del programa que se puede ver en la figura 2.13. En dicho programa, se incluye la biblioteca MQ135 que él mismo desarrolló.

En segundo lugar, se debe alimentar el sensor durante 12/24 horas. En este caso se tuvo 24h conectado para quemar bien la resistencia calefactora.

El tercer paso consiste en colocar el montaje en una habitación o lugar en interior, con unas condiciones ambientales de 20ºC con un 35% de humedad relativa (aunque la

humedad no es tan importante). En este caso, la temperatura inicial fue de 21.3ºC y la final de 20.2ºC; mientras que la humedad relativa fue constante durante el proceso con un valor del 50%.

Durante este paso, el valor de *rzero* va oscilando. Hay que esperar a que este valor se estabilice, lo que puede tardar entre 30 minutos y una hora.

Para acabar la calibración, cuando el valor de *rzero* obtenido por el puerto serie del IDE de Arduino se estabilice, se debe abrir el archivo MQ135.h con un editor de texto y sustituir el valor de RZero por el obtenido por el puerto serie. En este caso, el valor se estabilizó en 151 como se observa en la figura 2.13. Esta biblioteca modificada es la que se debe incluir en el programa de Arduino y posteriormente cargar a la placa del NodeMCU, ya que así tendremos en sensor calibrado y listo para obtener medidas fiables.

```
#ifndef MQ135_H
#define MQ135_H
#if ARDUINO >= 100
 #include "Arduino.h"
#else
 #include "WProgram.h"
#endif

/// The load resistance on the board
#define RLOAD 10.0
/// Calibration resistance at atmospheric CO2 level
#define RZERO 151
/// Parameters for calculating ppm of CO2 from sensor resistance
#define PARA 116.6020682
#define PARB 2.769034857
```

Figura 2. 13: Modificación de rzero en librería MQ135.h

```
#include "MQ135.h"

#define pinA 33

MQ135 gasSensor = MQ135(pinA);
int rzero;
float Res = 0;
float ppm_C02 = 0;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
}

void loop() {
  // put your main code here, to run repeatedly:

  rzero = gasSensor.getRZero();
  Res = gasSensor.getResistance();
  ppm_C02 = gasSensor.getPPM();

  Serial.print("Zero: ");
  Serial.println(rzero);
  Serial.print("Res: ");
  Serial.println(Res);
  Serial.print("ppm CO2: ");
  Serial.println(ppm_C02);
  delay(5000);
```

Figura 2. 14: Programa calibración y obtención rzero MQ135

MQ131

Este sensor presenta el mismo procedimiento que el MQ131, debido a que son de la misma familia de sensores. Es necesario seguir un método manual para escalar adecuadamente el valor medido. Este sensor presenta el mismo problema que el anterior, no tiene una respuesta lineal entre el valor real y el valor medido, por tanto, se debe estimar la recta de sensibilidad que proporciona el fabricante en un datasheet.

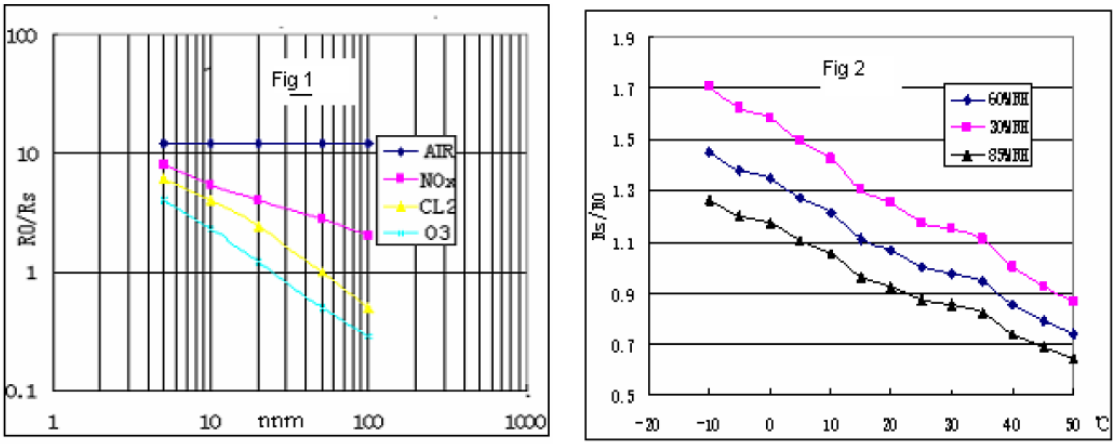


Figura 2. 15: Curva de sensibilidad del sensor MQ-131 / condiciones de temperatura

Es fundamental que se presenten las condiciones ambientales idóneas como temperatura a 20ºC, humedad relativa rondando los 55% de HR con una concentración de oxígeno del 21% y con la resistencia de carga RL asignada a 680 kΩ, en vez de la RL previa de 10kΩ para tener un rango dinámico mayor.

El fabricante nos proporciona la curva de sensibilidad (figura 2.15) y no una ecuación. Como en el caso anterior del MQ131. Por tanto, se ha de hallar de forma empírica la ecuación de funcionamiento de cada sensor individual.

En este caso, la ecuación que se obtiene tras representar gráficamente la curva de O_3 es la siguiente en la figura 2.16.

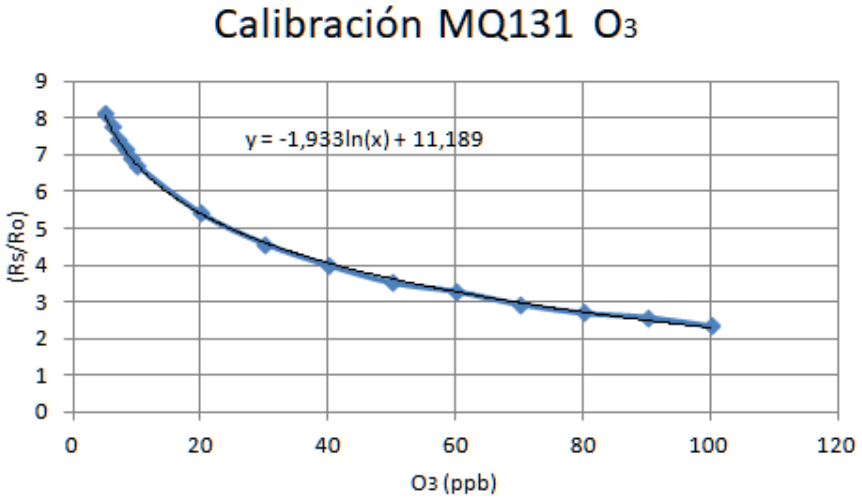


Figura 2. 16: Ecuación MQ131 para detección de O_3

Por tanto, si sustituimos para los valores que queremos:

(2.2)

$$\left(\frac{Rs}{Ro}\right) = -1{,}933\ln(O_3) + 11{,}189$$

Donde O_3 es el valor de la concentración de O_3 en ppb, R_o es la constante única de cada sensor cuando se expone a una concentración de 200 ppb de O_3 y R_s la resistencia del sensor que se debe leer desde el controlador.

En estos tiempos, es muy difícil asegurar una atmósfera con las condiciones específicas de 200 ppb de O_3, por lo que se ha tenido que buscar otra metodología de calibración para este sensor.

Este método alternativo de calibración para el sensor MQ-131 fue descrito por Olivier Staquet que realizó una biblioteca específica, para el entorno del IDE de arduino, para poder realizar el calibrado en condiciones ambientales normales.

Como se ha visto previamente se ha tenido que cambiar la resistencia de carga (RL), por lo que se debe cambiar este valor en la librería de 10kΩ por 680kΩ. Además, se tuvo que cambiar algunas funciones porque esta librería está preparada para Arduino Atmega y este proyecto utiliza la placa ESP32.

Hay varias diferencias, en primer lugar, el valor de voltaje de trabajo que utiliza cada microcontrolador en sus entradas analógicas. Por otro lado, también tiene un rango de resolución diferente. Estas diferencias se expresan en la siguiente tabla:

	Arduino Atmega	ESP32 (NodeMCU)
Voltaje de trabajo	5V	3,3V
Rango de resolución	10bits (0-1023)	12 bits (0-4095)

Tabla 2. 3: Diferencias en librería MQ131 por microcontrolador

A continuación, se presentan los cambios realizados en la librería para poder realizar el método de calibración.

```
15 // Default values
16 #define MQ131_DEFAULT_RL       680000   // Default load resistance of 10KOhms >>> 680KOhms Modified by: J_RPM
17 #define MQ131_DEFAULT_STABLE_CYCLE    20     // Number of cycles with low deviation to consider
```

Figura 2. 17: Cambio valor RL en librería MQ131

```
float MQ131Class::readRs() {
 // Read the value
 int valueSensor = analogRead(pinSensor);
 // Compute the voltage on load resistance (for 5V Arduino)
 //float vRL = ((float)valueSensor) / 1024.0 * 5.0;
 // Compute the voltage on load resistance (for 3,3V ESP32)
 float vRL = ((float)valueSensor) / 4095.0 * 3.3;
 // Compute the resistance of the sensor (for 5V Arduino)
 //float rS = (5.0 / vRL - 1.0) * valueRL;
 // Compute the resistance of the sensor (for 3,3V ESP32)
 float rS = (3.3 / vRL - 1.0) * valueRL;
 return rS;
}
```

Figura 2. 18: Cambio conversión valor voltaje en librería ESP32

El primer paso de la calibración es la carga del programa que se puede ver en la figura 2.20. En dicho programa, se incluye la biblioteca MQ131 que él mismo desarrolló.

En segundo lugar, se debe alimentar el sensor durante 24/48 horas. En este caso se tuvo 48h conectado para quemar bien la resistencia calefactora.

El tercer paso consiste en colocar el montaje en una habitación o lugar en interior, con unas condiciones ambientales de 20ºC con un 55% de humedad relativa. En este caso, la temperatura inicial fue de 21.3ºC y la final de 20.2ºC; mientras que la humedad relativa fue constante durante el proceso con un valor del 50%.

Durante este paso, el valor de *rzero* va oscilando. Hay que esperar a que este valor se estabilice, esto puede tardar varios minutos.

Para acabar la calibración, cuando el valor de *rzero* obtenido por el puerto serie del IDE de Arduino se estabilice, se debe abrir el archivo MQ131.h con el IDE de arduino y proporcionar con la función setR0 el valor obtenido por el puerto serie. En este caso, el valor se estabilizó en 364740.87 como se observa en la figura 2.19. Esta biblioteca modificada es la que se debe incluir en el programa de Arduino y posteriormente cargar a la placa del NodeMCU, ya que así tendremos en sensor calibrado y listo para obtener medidas fiables.

```
MQ131.begin(2,PinMQ_A, LOW_CONCENTRATION, 680000);

//Resistencia R0 del sensor MQ-131 instalado, para 10 ppb

MQ131.setR0(364740.87);

// Segundos entre lecturas
// No es necesario más tiempo, porque el calefactor del sensor está conectado de forma permanente
MQ131.setTimeToRead(1);

// Temperatura y Humedad (Habilitar para modificar los valores, por defecto 20°C / 60%HR)
// En caso de incorporar un sensor de temperatura y humedad, actualizar aquí con los valores del sensor
MQ131.setEnv(20, 60); // Sin corrección: 20°C / 60 %HR
```

Figura 2. 19: Modificación de rzero en librería MQ135.h

```
#include "MQ131.h"

#define pinA 32


int rzero;
float O3_ppm = 0;

void setup() {
  // put your setup code here, to run once:
Serial.begin(115200);

MQ131.begin(2,pinA, LOW_CONCENTRATION, 680000);

}

void loop() {
  // put your main code here, to run repeatedly:


rzero = MQ131.getR0();
O3_ppm = MQ131.getO3(PPM);

Serial.print("Zero: ");
Serial.println(rzero);
Serial.print("ppm O3: ");
Serial.println(O3_ppm);
delay(5000);
```

Figura 2. 20: Programa calibración y obtención rzero MQ131

2.2 Actuadores y procesos

En este apartado se expondrán los diferentes actuadores que se emplean durante el proceso de desinfección del sistema. Además, se explican las diferentes características de cada uno.

2.2.1 Generador de ozono

Es una máquina que genera ozono artificial, a partir del oxígeno del ambiente, mediante una alta tensión eléctrica por dos electrodos, esto produce el "efecto corona" el cual consiste en un fenómeno eléctrico que se produce por la ionización del gas que rodea a un conductor cargado. Este fenómeno aparece en las líneas de alta tensión y se manifiesta como un halo luminoso.

Esto provoca una corriente de electrones en el espacio delimitado por los electrodos, que es por donde se conduce el aire. Estos electrones provocan la disociación de las moléculas de oxígeno que posteriormente crearán el ozono.

En la siguiente tabla se exponen las características principales del generador:

	Generador Ozono
Tipo generador	Cerámico
Voltaje funcionamiento	220 V
Consumo	90 W
Producción Ozono	10g/h

Tabla 2. 4: Características generador ozono cerámico

El ozono es considerado un gas irritante exclusivamente por sus concentraciones en el aire, es decir, por problemas derivados de su inhalación, que dependen concentración del gas en el ambiente y del tiempo de exposición del individuo.

Según la OMS la cantidad máxima en general recomendada de ozono en el aire es de 0'05 ppm (0'1mg/m^3). Aunque para periodos inferiores a 2 horas puede ser una concentración de 0'2 ppm. Los valores para presentar efectos agudos letales son altos, unos 15 ppm, una concentración prácticamente inalcanzable por métodos convencionales.

El ozono es un gas muy inestable, es decir, rápidamente se descompone formando oxígeno, por lo cual no se puede almacenar de forma eficiente. Es por esto que se debe producir el ozono *In situ.* Además, es necesario de un sistema para esparcir el gas por el entorno, esto es debido a que el ozono presenta una densidad mayor al aire.

Figura 2. 21: Generador de ozono con sistema de ventilación

2.2.2 Ventiladores

En este apartado se explicará las características y funciones de los ventiladores integrados en el sistema de desinfección. El equipo dispone de dos ventiladores diferentes, el primero se encarga de la distribución del gas ozono durante la creación de

este. El segundo tiene la función de acabar de distribuir el ozono una vez acabado el tiempo de funcionamiento del generador.

Estos ventiladores son necesarios por varios motivos, en primer lugar, porque el ozono es una molécula muy inestable por lo que se debe esparcir por la estancia antes de que la molécula se descomponga. Por otro lado, porque al poseer una densidad mayor que el aire el ozono tiende a acumularse en vez de distribuirse de forma uniforme, por lo cual, se utilizan estos ventiladores.

	Ventilador
Tipo ventilador	Fan / Axiale ventilator
Voltaje funcionamiento	12 V
Consumo	10,1 W
Revoluciones	3100RPM

Tabla 2. 5: Características ventilador del sistema de desinfección

Figura 2. 22: Ventiladores del sistema de desinfección

2.2.3 Pantalla LCD

La pantalla LCD es un instrumento informativo para el usuario para poder saber en qué punto del proceso se encuentra el sistema. Existen diferentes tipos de pantallas LCD, en este caso, se utiliza el modelo LCD1602 con el módulo hardware adaptador LCD a I2C. Este adaptador permite conectar la pantalla LCD a un microcontrolador, en este caso al ESP32, mediante el protocolo de comunicación I2C.

Esta comunicación tiene una arquitectura maestro-esclavo. Por tanto, el maestro inicia la comunicación, tanto para enviar datos como para recibir. Los esclavos por el contrario reaccionan a las instrucciones del maestro.

Esta comunicación requiere de únicamente de dos cables, una señal de reloj (SCL) y una señal de datos (SDA). El bus I2C es síncrono, por tanto, el maestro proporciona una señal de reloj, que mantiene sincronizados a todos los dispositivos del bus.

El bus I2C emplea una trama amplia para poder realizar la comunicación entre los dispositivos. La comunicación costa de:

- 7 bits a la dirección del dispositivo esclavo a comunicar.
- Un bit restante indica si enviar o recibir información.
- Un bit de validación.
- Uno o más bytes son los datos enviados o recibidos del esclavo.
- Un bit de validación.

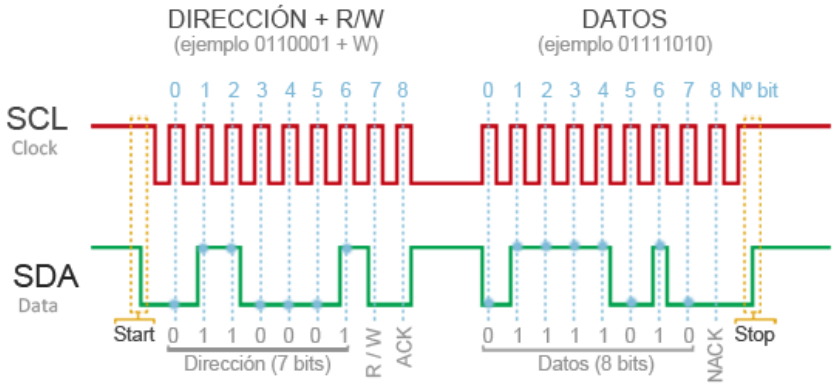


Figura 2. 23: Protocolo de comunicación I2C

La pantalla requiere de 4 conexiones: VCC, GND, SCL, SDA. Para las cuales contamos con pines preconfigurados en la Placa del ESP32. Aunque con una librería (Wire.h) podemos configurar casi cualquier GPIO como SCL o SDA.

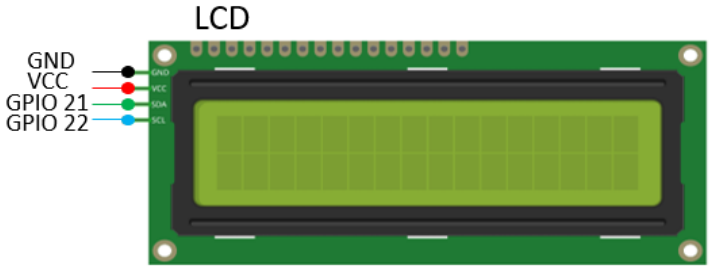


Figura 2. 24: Conexiones pantalla LCD 1602

2.2.4 Zumbador

Como último actuador se presenta el zumbador, el cual es un instrumento señalizador, es decir, advierte en qué momento el sistema llega a un determinado proceso. En el sistema implementado se emplea cunado el proceso de desinfección ha finalizado para advertir al usuario.

El zumbador es un instrumento muy sencillo de utilizar, esto es debido a que solo consta de 2 pines, (GND-VCC). Por lo tanto, para utilizarlo se conecta a una salida del ESP32 y cuando sea preciso se aplica un voltaje de 5V.

Figura 2. 25: Zumbador

2.3 La placa ESP32

A lo largo de este apartado se analizará de forma minuciosa la placa de gobernar el sistema de desinfección que se quiere desarrollar. Para ello, se expondrán los principales aspectos del SoC que la gestiona, el ESP32-D0WDQ6, y se comentarán las diferentes posibilidades que presenta.

2.3.1 ESP32-D0WDQ6, *SoC* de la *NodeMCU*

La NodeMCU (Microcontroller Unit) es una placa de desarrollo abierta tanto a nivel de Software como de Hardware. Esta placa está diseñada con el objetivo de facilitar la programación del microcontrolador que la controla.

El SoC (System on a Chip) encargado de gobernar esta placa es el ESP32-D0WDQ6, que tiene como corazón el microcontrolador Tensilica Xtensa LX6 con arquitectura de 32-bit Dual core.

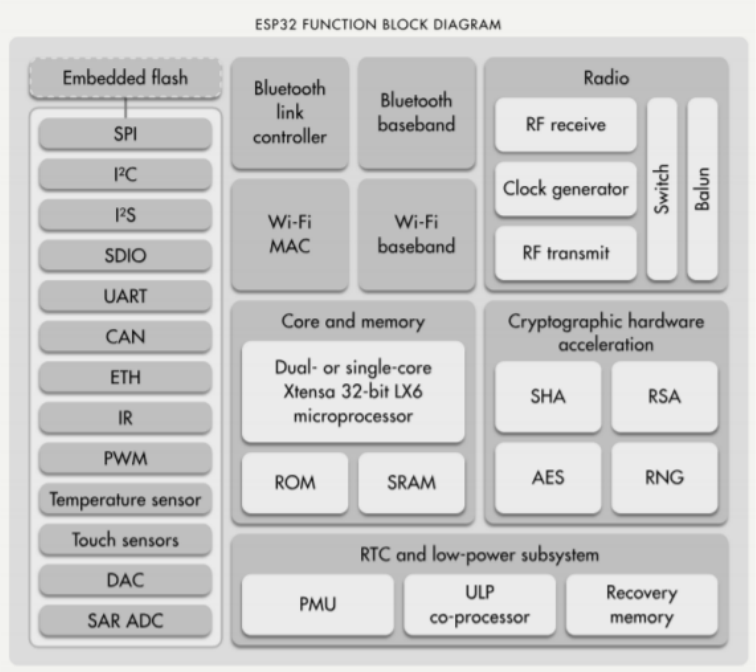


Figura 2. 26: Diagrama de bloques del SoC ESP32

Por tanto, la arquitectura de este *SoC* permite realizar operaciones con números de tamaño de 0 a 4.294.967.295 o de -2.147.483.648 a 2.147.483.647 con una frecuencia de trabajo de 160 MHz, aunque puede llegar a trabajar con 240MHz, esto es una velocidad bastante elevada, aún más comparado con su hermano menor ESP8266 que más adelante se verán las diferencias entra ambos.

Es decir, el ESP32 tiene integrado todo lo necesario para actuar prácticamente como un ordenador autónomo, incluso dispone de una ROM para guardar programas, cosa que el ESP8266 no dispone.

Por tanto, las características principales del ESP32 son las siguientes:

- MCU Tensilica Xtensa LX6 32 bit Dual core
- Módulo Wifi de 2.4 GHz
- Módulo Bluetooth v4.2 BR/EDR y BLE
- Memoria RAM 520 kB
- Memoria ROM 448 Kb
- 18 entradas analógicas de 12 bits
- 36 pines de entradas y salidas (GPIO)

Es necesario comentar que el sistema de desinfección parte de un proyecto previo en el cual se utilizaba la placa ESP8266 para llevar a cabo el proceso de desinfección. En este trabajo se escogió la placa ESP32 por sus mayores ventajas frente al ESP8266.

Esto facilitó notablemente la programación del NodeMCU debido a su mayor cantidad de GPIO disponibles. Además, una de las mayores ventajas que supuso este cambio fue la posibilidad de integrar diversos sensores, ya que muchos de estos sensores utilizan una salida analógica para proporcionar su salida. Esto era un problema con la placa ESP8266 porque solamente presenta una entrada analógica y era imposible conectar más de un sensor al NodeMCU.

Al realizar este cambio se obtuvo un sistema más completo y robusto para llevar a cabo el proceso de desinfección, aún más importante permitió realizar el nuevo proceso de monitorización mediante los sensores para controlar el ambiente donde se produce la desinfección, además de poder visualizarlo tanto vía web con la plataforma ThingSpeak o via Bluetooth con la App Android.

Por tanto, se presentan todas las diferencias entre la placa anterior ESP8266 y la nueva ESP32 empleadas:

Características	ESP8266	ESP32
Procesador	Tensilica LX106	Tensilica Xtensa LX6
Nº bits	32 bits	
Nº nucleos	Single core	Dual core
Frecuencia de trabajo	80 MHz (hasta 160 MHz)	160 MHz (hasta 240 MHz)
SRAM	160 kB	512kB
SPI FLASH	Hasta 16 MiB	
Alimentación	3.0 a 3.6 V	2.2 A 3.6 V
Temperatura	-40ºC a 125ºC	
Consumo corriente	80 mA(promedio), 225 mA(máximo)	
Wifi	802.11 b/g/n (hasta +20 dBm) WEP, WPA	
Bluetooth	No	V4.2 BR/EDR + BLE
Ethernet	No	10/100 Mbps
GPIO (utilizables)	17	36
Hardware/Software PWM	No/8	1/16
ADC	1(10 bits)	18(12 bits)
DAC	No	2 (8 bits)
UART	2	4
SPI	2	4
I2C	1	2
Sensor efecto Hall	No	Si
Sensor temperatura	No	Si

Tabla 2. 6: Comparativa ESP8266 / ESP32

Una vez vistas estas diferencias, también es importante estudiar las diferencias entre los diferentes modelos para el mismo chip ESP32. Es importante diferenciar entre chip, módulo y tarjeta de desarrollo, aunque comúnmente se le pueda llamar módulo ESP32 o chip ESP32.

A continuación, se expondrán las diferencias por los distintos chips disponibles:

Ordering code	Core	Embedded flash	Connection	Package
ESP32-D0WDQ6	Dual core	No embedded flash	Wi-Fi b/g/n + BT/BLE Dual Mode	QFN 6*6
ESP32-D0WD	Dual core	No embedded flash	Wi-Fi b/g/n + BT/BLE Dual Mode	QFN 5*5
ESP32-D2WD	Dual core	16-Mbit embedded flash (40 MHz)	Wi-Fi b/g/n + BT/BLE Dual Mode	QFN 5*5
ESP32-S0WD	Single core	No embedded flash	Wi-Fi b/g/n + BT/BLE Dual Mode	QFN 5*5

Tabla 2. 7: Diferencias entre los distintos chips de la familia ESP32

Después de ver los distintos chips, se puede exponer los distintos módulos existentes en el mercado junto al chip que llevan integrado:

–	Key Components				Dimensions [mm]		
Module	**Chip**	**Flash**	**RAM**	**Ant.**	**L**	**W**	**D**
ESP32-WROOM-32	ESP32-D0WDQ6	4MB	–	MIFA	25.5	18	3.1
ESP32-WROOM-32D	ESP32-D0WD	4MB	–	MIFA	25.5	18	3.1
ESP32-WROOM-32U	ESP32-D0WD	4MB	–	U.FL	19.2	18	3.2
ESP32-SOLO-1	ESP32-S0WD	4MB	–	MIFA	25.5	18	3.1
ESP32-WROVER	ESP32-D0WDQ6	4MB	4MB	MIFA	31.4	18	3.2
ESP32-WROVER-I	ESP32-D0WDQ6	4MB	4MB	U.FL	31.4	18	3.5

Tabla 2. 8: Diferencias entre los distintos módulos de la familia ESP32

2.3.2 Placa de desarrollo *NodeMCU*

En este apartado se expondrán las características de la placa de desarrollo utilizada en el proyecto. Primero hay que diferenciar entre los chips integrados, anteriormente mencionados en el apartado 2.3.1, con las placas donde van integrados.

Por si solo el chip necesita que le incorporen toda la electrónica necesaria para realizar las tareas como la memoria flash, el cristal oscilador de 40 MHz, configuración de la antena o las resistencias pull-up. Estas placas de desarrollo incorporan ya todos estos elementos para facilitar su montaje.

En este proyecto se optó por la placa de desarrollo ESP32 DEV KIT DOIT, el cual cuanta con dos versiones diferentes, una con 30 y otra con 36 GPIOs que es la utilizada en el sistema. Las dos versiones se basan en el módulo PCB ESP-WROOM-32.

Estas dos versiones presentan estas características principales:

- Conversor Serie-USB para programación y alimentación
- Fácil acceso a los pines del MCU
- Presencia de pines de alimentación para sensores y actuadores
- Presencia de LED's indicadores de estado
- Presencia de botón de reset

Figura 2. 27: Placa de desarrollo ESP32 DEV KIT DOIT

Por último, esta placa emplea el chip de comunicación USB CP2102, por tanto, se debe instalar en el PC donde se programe el driver necesario para poder programar elESP32.

2.4 Programación

A lo largo de este apartado se mostrarán como se realizado la programación del NodeMCU utilizando el IDE de Arduino y las diferentes librerías aportadas por la comunidad que facilitan el manejo y el uso de los distintos sensores y actuadores.

Los principales puntos a mencionar serán las partes de código que configuran y establecen la comunicación con la aplicación Blynk y el servidor ThingSpeak, además se comentará el código para el proceso de desinfección.

A continuación, se enumeran las distintas librerías empleadas en la programación junto a una breve explicación de su utilidad:

1. <Wire.h>
2. <hd44780.h>
3. <LiquidCrystal_I2C.h>
4. <DHT.h>
5. <MQ131.h>
6. <MQ135.h>
7. <ThingSpeak.h>
8. <WiFi.h>
9. <BlynkSimpleEsp32.h>

El código del programa completo se encuentra en el apéndice A

2.4.1 Blynk

En este apartado se expondrán los puntos clave sobre la programación para poder establecer comunicación con la App Blynk y enviar la información deseada.

Autenticación e inicio de la comunicación

En la figura 2.28, se puede observar la sintaxis necesaria para obtener la autenticación de la app con el hardware que se desea comunicar. Cada proyecto creado con Blynk tiene un Token único (contraseña).

```
31 // clave de seguridad para establecer la comunicación con APP Blynk
32 #define SECRET_AUTH "Sgj3QShg2x6VX4G4ONAVXdt-898HVtg-"
```

Figura 2. 28: Contraseña única del proyecto de Blynk

En la figura 2.29 se muestra la sintaxis necesaria para establecer la comunicación con la App Android. Esta función envía el identificativo único, el nombre de la red a la que se conecta el NodeMCU y la contraseña de acceso de esta red inalámbrica. También se define que el envío de información sea cada 5 segundos para no saturar la comunicación.

```
150 Blynk.begin(auth, ssid, pass); //inicialización de la comunicación con Blynk
151 timer.setInterval(5000L, sendvalue1); //intervalo de tiempo de 5 segundos
```

Figura 2. 29: Configuración comunicación con Blynk

Envío de datos

La función *sendvalue1* es la encargada de enviar los datos de los diferentes sensores cíclicamente cada 5 segundos, tanto para Blynk como para ThingSpeak.

```
363 void sendvalue1()  //creamos un bucle para la APP de blynk
364 {
365  //establecer las variables que se van a enviar al servidor y a qué campo va a pertenecer cada una
366  ThingSpeak.setField(1,temp);
367  ThingSpeak.setField(2,hum);
368  ThingSpeak.setField(3,O3_ugm3);
369  ThingSpeak.setField(4,ppm_C02);
370
371  ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey); //escribir los valores establecidos
372
373  //establecer las variables que se van a enviar a la App y a qué campo va a pertenecer cada una
374  Blynk.virtualWrite(V5, temp);
375  Blynk.virtualWrite(V6, hum);
376  Blynk.virtualWrite(V7, O3_ugm3);
377  Blynk.virtualWrite(V8, ppm_C02);
378  Blynk.virtualWrite(V9, O3_ppb);
379  Blynk.virtualWrite(V10, O3_ppm);
380
381 }
```

Figura 2. 30: Envío de datos a Blynk y ThingSpeak

Los valores de los diferentes sensores se obtienen a través de las funciones específicas de cada uno de ellos. Estas funciones son *leer_MQ131*, *leer_MQ135* y *leerdht11*. En ellas se presentan como obtener los datos y su visualización por el puerto serie.

```
345 void leer_MQ135(){
346
347   ppm_C02 = gasSensor.getPPM();
348
349   Serial.print("ppm MQ-135:");
350   Serial.println(ppm_C02);
```

```
278 void leerdht11(){
279
280   temp = dht11.readTemperature();
281   hum =  dht11.readHumidity();
```

```
306 void leerMQ_131(){
307
308   lectura_ozono = analogRead(PinMQ_A);
309
310   // Carga los valores obtenidos en sus variables
311   O3_ppm = MQ131.getO3(PPM);
312   O3_ppb = MQ131.getO3(PPB);
313   O3_mgm3 = MQ131.getO3(MG_M3);
314   O3_ugm3 = MQ131.getO3(UG_M3);
```

Figura 2. 31: Obtención datos de los sensores

2.4.2 ThingSpeak

Como en el anterior apartado, se expondrán los puntos clave de la programación para establecer la comunicación y enviar los datos al servidor web de ThingSpeak.

Autenticación e inicio de la comunicación

En la figura 2.32 se puede observar la información necesaria para poder conectar con el servidor ThingSpeak, la cual es el ID del canal y la API key, estos parámetros son únicos para cada canal diferente.

```
28 #define SECRET_CH_ID 1231108     // ID del canal de thingspeak con el que se quiere establecer la comunicación
29 #define SECRET_WRITE_APIKEY "53XBAKBUCHNCYPFY"   // replace XYZ with your channel write API Key
30 unsigned long myChannelNumber = SECRET_CH_ID;
31 const char * myWriteAPIKey = SECRET_WRITE_APIKEY;
```

Figura 2. 32: Parámetros canal ThingSpeak

Se debe inicializar la comunicación a ThingSpeak con la función ThingSpeak.begin, la cual necesita un cliente wifi para establecer dicha comunicación.

```
153       ThingSpeak.begin(client);  // Initialize ThingSpeak
154       Serial.print(".");
155       delay(2000);
```

Figura 2. 33: Inicialización de la comunicación con ThingSpeak

Por último, como se observó en la figura 2.30 se debe establecer los datos a cada gráfico y con la función *ThingSpeak.wrietFields* escribir los datos en su respectivo gráfico.

2.5 Prototipo

En este apartado se expondrá el diseño del prototipo para el sistema de desinfección tanto del proyecto previo como del proyecto posterior con sus respectivos sensores.

Se debe tener en cuenta que el prototipo irá instalado en armarios con unas dimensiones de 1,80 metros de altura y 1,20 metros de ancho. Estos armarios son de aluminio, por lo cual se utilizará una estructura también de aluminio para acoplar el sistema de control al armario.

2.5.1 Esquema electrónico

El diseño del sistema se ha realizado de forma compacta porque no debe ocupar demasiado espacio debido a que el espacio en los armarios es reducido. Además, este requisito fue expresamente pedido por el mismo cliente del proyecto.

Este diseño consta de dos partes diferenciadas: una sección de control y otra de actuación.

Por un lado, el control consta de los diferentes sensores como el MQ-131, MQ-135 y DHT11 gobernados y alimentados por el NodeMCU. Por otro lado, se encuentra la sección con los diferentes actuadores como el generador de ozono, los ventiladores, el zumbador y la pantalla LCD.

El sistema no requiere de una batería externa debido a que en todo momento estará conectada a la red eléctrica. Por lo tanto, la alimentación del sistema se realiza con un transformador de 230VAC – 5VDC.

El prototipo se debía realizar con un diseño PCB de la placa con todos los componentes integrados, sin embargo, debido a una situación económica la empresa no vio oportuno de momento materializar la placa. Por tanto, se realizó un prototipo utilizando una protoboard, en la cual la sección de control permanece intacta pero la parte de los actuadores se simula mediante 3 leds, el motivo de esto es que no se puede proveer de una red de 230 V para activar el generador de ozono.

Además, otro motivo por el cual se descartó realizar la nueva placa fue porque el proyecto previo todavía estaba en fase de pruebas. Durante la realización del nuevo prototipo, todavía se estaba analizando la evolución en el mercado del proyecto.

A continuación, se mostrará en la figura 2.34 es esquema eléctrico del prototipo, además se mostrará en la figura 2.35 la PCB del proyecto previo y en la figura 2.36 la estructura donde se coloca dentro del armario.

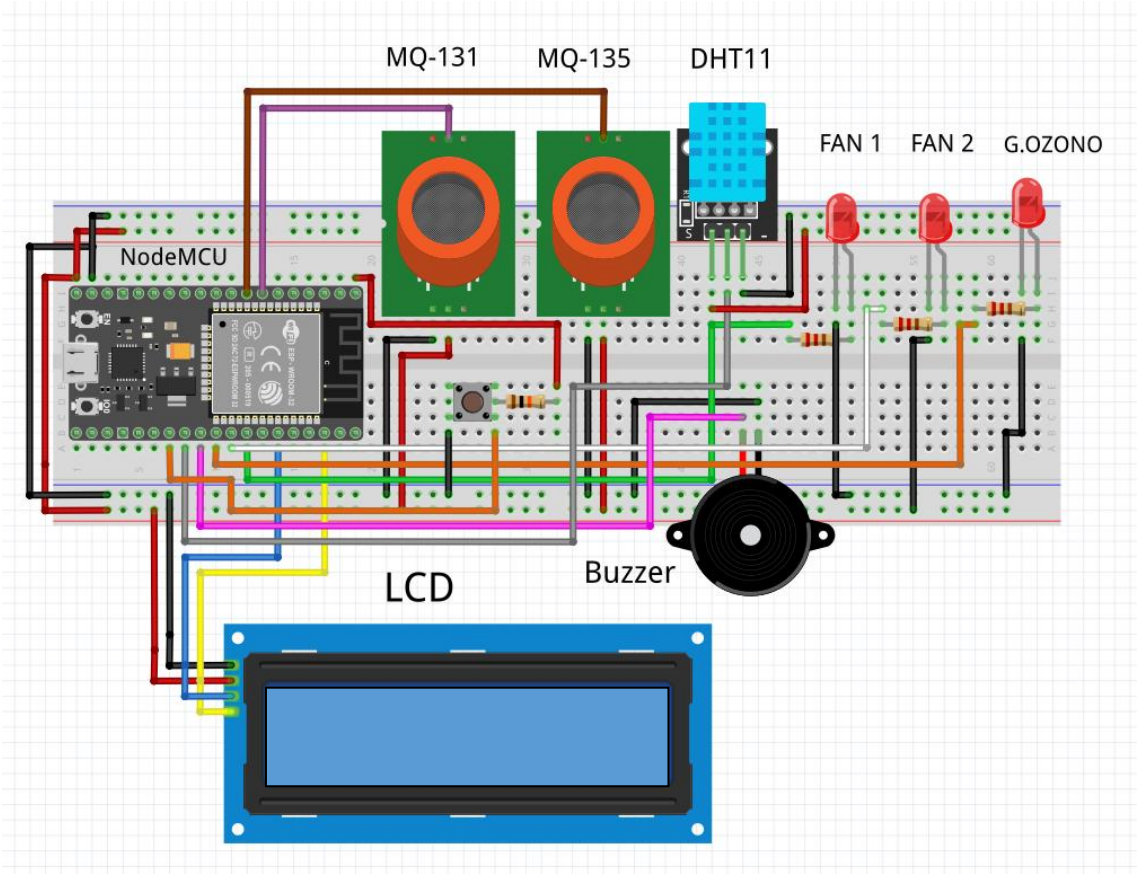


Figura 2. 34: Esquema eléctrico del prototipo

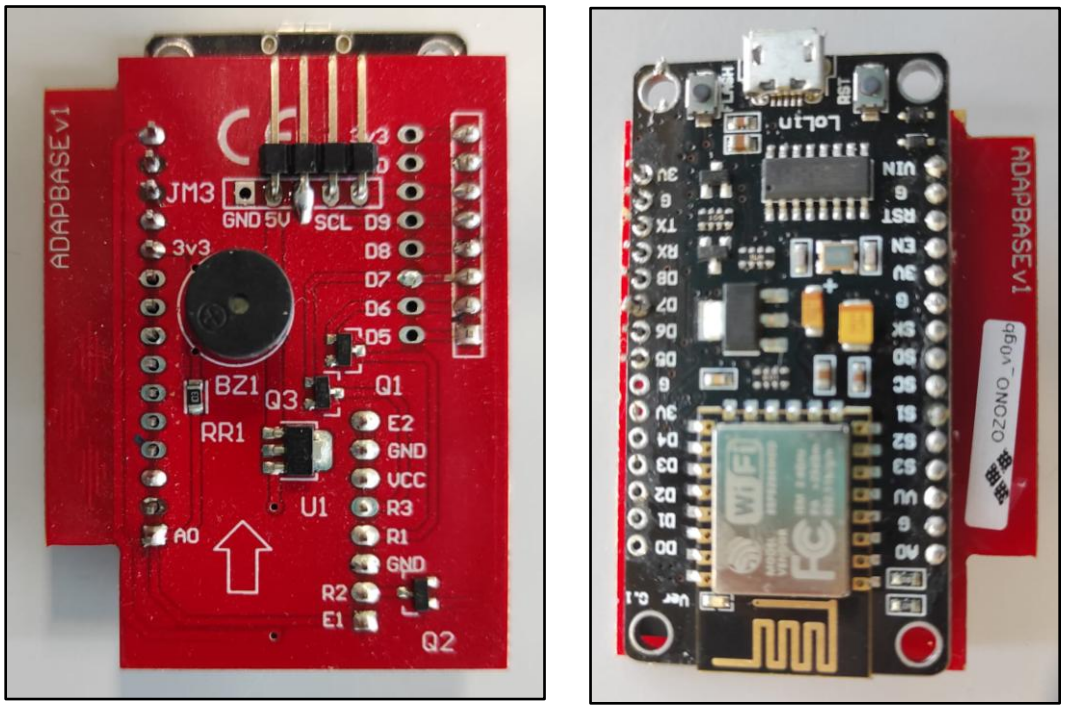


Figura 2. 35: PCB proyecto previo

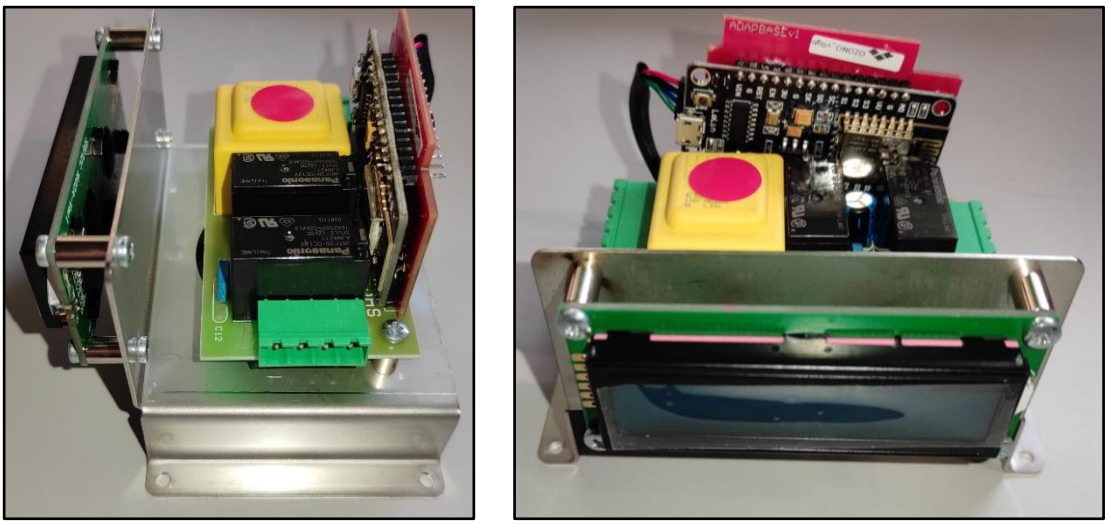


Figura 2. 36: Estructura para instalar el sistema en el armario

Capítulo 3

Resultados

A lo largo de este apartado se expone como se muestran los datos del sistema de desinfección, que puede ser a través de Smartphone o Tablet, vía la app Blynk o bien por el navegador web de cualquier dispositivo mediante el servidor de ThingSpeak.

3.1 Visualización de los datos vía Blynk

Para la visualización de los datos a través de Blynk se ha diseñado una App la cual muestra en pantalla 6 calibres donde cada uno muestra un campo de datos diferente. Existen diferentes formas de mostrar los datos como indicadores numéricos o gráficas pero cada componente gasta una cierta cantidad de energía en la aplicación, al empezar a realizar un proyecto nuevo se tiene una cantidad limitada de energía.

Esta energía es bastante limitada, además si se quiere aumentar esta energía se debe hacer una micro-transacción en la aplicación. Esto limita bastante las opciones de una App prototipo para una empresa que todavía no sabe cómo evolucionará el producto.

A continuación, se muestra en la figura 3.1 la App realizada para el proyecto.

Figura 3. 1: App Blynk

3.2 Visualización de los datos vía ThingSpeak

En el servidor web de ThingSpeak se pueden mostrar tanto datos en tiempo real como históricos. Para esta plataforma se ha elegido 3 tipos de presentación.

- Tiempo real: indicadores numéricos
- Históricos: gráficas
- Alarmas: pilotos indicadores

En las siguientes figuras 3.2, figura 3.3, figura 3.4, figura 3.5, figura 3.6 y figura 3.7 se muestran las diferentes representaciones de los datos procesados por el sistema.

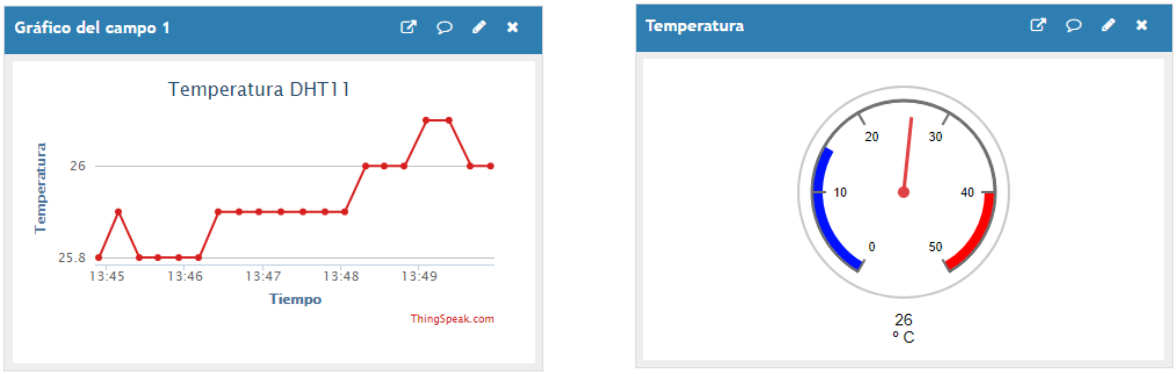


Figura 3. 2: Visualización de los datos de temperatura *DHT11*

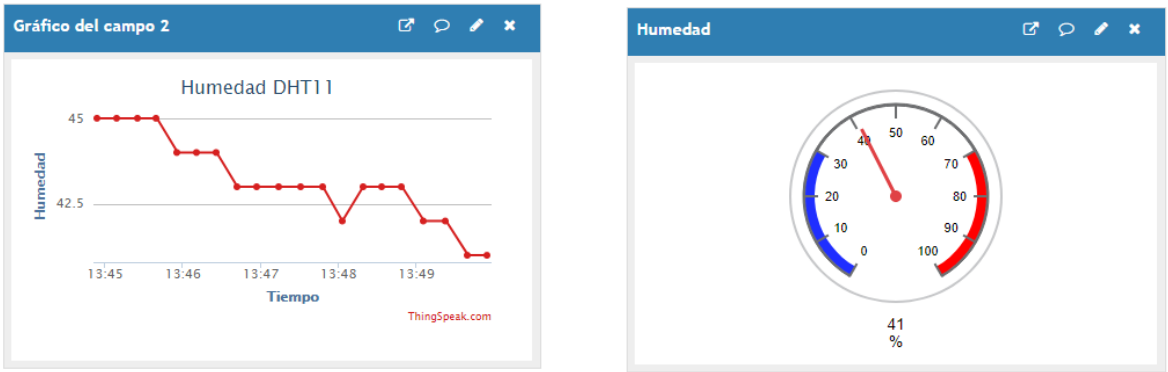


Figura 3. 3: Visualización de los datos de humedad *DHT11*

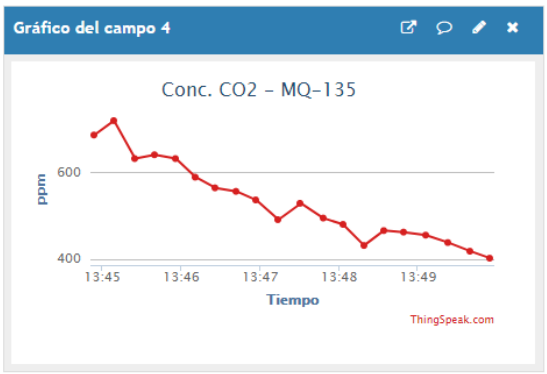


Figura 3. 5: Visualización datos de concentración de ozono *MQ-131*

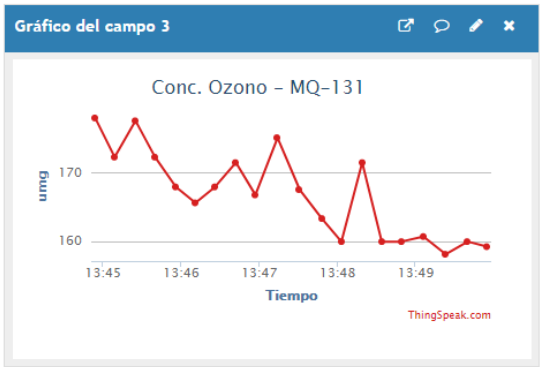


Figura 3. 4: Visualización de los datos de concentración de CO_2 *MQ-135*

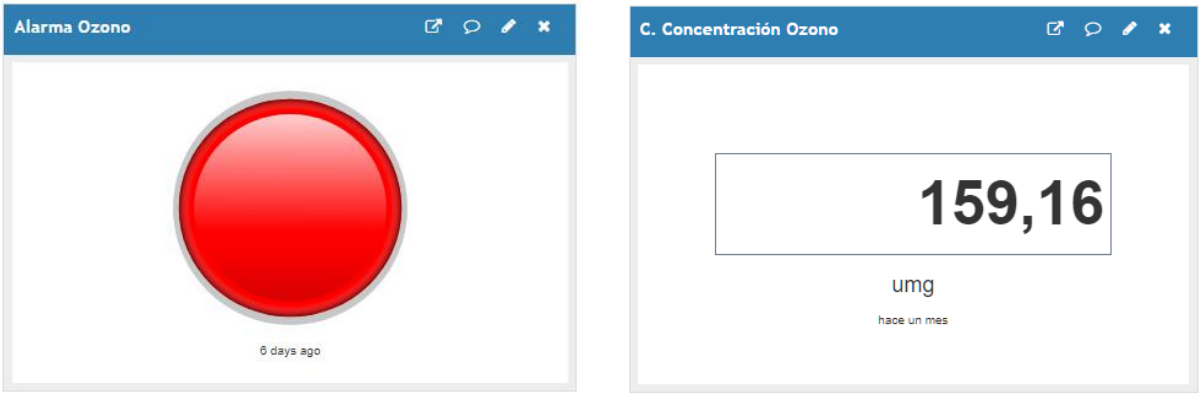


Figura 3. 6: Alarma máxima de concentración de ozono permitida por la *OMS*

Esta alarma máxima está establecida por la *OMS* en 0,10 mg/m^3 o 100 µg/m^3 para un máximo de 8 horas al día. Una vez pasemos de este umbral el piloto se volverá rojo advirtiendo que se superó la concentración aconsejada.

Figura 3. 7: Alarma máxima de concentración de CO_2 permitida por la *OMS*

Esta alarma está establecida entre 400 y 800 ppm para una habitación o recinto cerrado. En este caso cuando sobrepasemos los 800 ppm de CO_2 , la alarma saltará advirtiendo al usuario de este hecho.

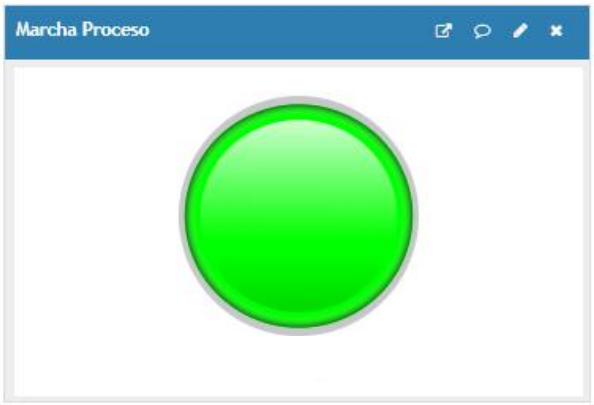


Figura 3. 8: *Piloto para visualizar cuando está funcionando el proceso*

Capítulo 4

Conclusión

Durante el periodo de realización del presente proyecto, se ha definido el diseño e implementación de un sistema de desinfección capaz de eliminar microorganismos y adquirir datos del entorno tanto en tiempo real como almacenarlos en un servidor web, lo cual permite su visualización pasado un largo periodo de tiempo.

El sistema de desinfección se divide en dos procesos bien diferenciados. Estas actividades están gobernadas por una placa NodeMCU que dispone de un procesador ESP32 con conectividad WiFi, por lo tanto, el sistema está conectado continuamente a la red con acceso a internet.

Se ha diseñado para la desinfección de áreas pequeñas y cerradas, con la intención de eliminar principalmente el virus SARS-CoV-2 debido a la crisis sanitaria actual. Además, se ha diseñado para monitorizar la estancia donde se encuentra el equipo, esto es útil para cumplir los requisitos sanitarios por la OMS tanto por la utilización del gas ozono como por la calidad del aire de la estancia. Para la monitorización se ha hecho especial hincapié en la elección de los diferentes sensores como el MQ-131, MQ-135 y DHT11 con los cuales se puede medir la temperatura, la humedad, concentración de ozono y la concentración de CO_2 en el ambiente.

Se debe destacar que el sensor MQ-131 no muestra medidas estables y constantes, después de innumerable intentos y búsquedas de información se ha llegado a la conclusión que se debería optar por un sensor de mayor calidad. Este sensor puede servir como detector del gas ozono, pero no para estimar una cantidad precisa del mismo.

La NodeMCU, gobernada por el SoC ESP32 se ha programado mediante el IDE de Arduino. Esta placa permite su programación con este entorno que supone una ventaja debido a que esta plataforma está extendida y posee una gran comunidad, donde poder encontrar librerías y múltiples ejemplos.

El diseño electrónico del proyecto ha sido realizado pensando en futuras ampliaciones, por este motivo se cambió la placa NodeMCU de un procesador ESP8266 a un ESP32. Esto dota al sistema de una mayor versatilidad por la mayor cantidad de pines I/O que dispone, con lo cual se podría añadir más sensores, actuadores o módulos.

Bibliografía

CO, *Espressif Systems (2021). ESP32EX. Data sheet.*

CO, *HANWEI ELECTRONICS LTD (No especificado). TECHNICAL DATA MQ-135 GAS SENSOR. Data sheet.*

CO, *Mouser Electronics (No especificado). DHT11 Humidity Temperature Sensor. Data sheet.*

CO, Henan Hanwei (No especificado). TECHNICAL DATA MQ-131 OZONO SENSOR. Data sheet.

Comunidad GITHUB. Documentación relativa a NodeMCU (Consultado en Marzo y Abril de 2021). https://github.com/nodemcu.

Foro de fritzing (Consultado Mayo de 2021). https://forum.fritzing.org/.

Llamas (2016). DETECTOR DE GASES CON ARDUINO Y LA FAMILIA DE SENSORES MQ (Consultado en Marzo de 2021). https://www.luisllamas.es/arduino-detectorgas- mq/.

Llamas (2016). MEDIR TEMPERATURA Y HUMEDAD CON ARDUINO Y SENSOR DHT11 DHT22 (Consultado en Abril de 2021). https://www.luisllamas.es/arduino-dht11-dht22/.

Página web ayuda de Blynk (Consultado en Marzo y Abril de 2021). https://docs.blynk.cc/.

Página web Youtube (Consultado en Febrero de 2021). https://www.youtube.com/watch?v=YSMRpG5tzK4&t=741s

Página web de Random Nerd Tutorials (Consultado en Marzo de 2021). https://randomnerdtutorials.com/esp32-adc-analog-read-arduino-ide/

Página web de Handson Technology (Consultado en Febrero de 2021). http://www.handsontec.com/dataspecs/module/I2C_1602_LCD.pdf

Página web ayuda de MathWorks (Consultado en Marzo y Abril de 2021). https: / / es . mathworks.com/help/thingspeak/.

APÉNDICES

PROGRAMACIÓN

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Clcd.h>
#include <LiquidCrystal_I2C.h>
#include <DHT.h>
#include "MQ131.h"
#include "MQ135.h"
#include "ThingSpeak.h"
#include <WiFi.h>
#include <Blynk.h> //incluir librería de Blynk
#include <BlynkSimpleEsp32.h> //incluir librería del módulo WiFi con
Blynk

#define ENTRADA     4     // 4
#define RELE1       19    // G.OZONO
#define RELE2       18    // 2 VENTILADORES
#define RELE3       5     // FAN OZONO
#define ZUMB        17    // 14 ZUMBADOR
#define d_h_t       16
//#define PinMQ_A     15   //15
#define PinMQ_A     32   //34
#define PinMQ_135   33   //35
#define Umbral_O3   0
#define Modo        10

#define SECRET_SSID "ONOBDC5"    // nombre de red WiFi
#define SECRET_PASS "cV10mxhwKNky"  // contraseña de red WiFi
#define SECRET_CH_ID 1231108     // ID del canal de thingspeak con el
que se quiere establecer la comunicación
#define SECRET_WRITE_APIKEY "53XBAKBUCHNCYPFY"   // replace XYZ with
your channel write API Key
unsigned long myChannelNumber = SECRET_CH_ID;
const char * myWriteAPIKey = SECRET_WRITE_APIKEY;
// clave de seguridad para establecer la comunicación con APP Blynk
#define SECRET_AUTH "Sgj3QShg2x6VX4G4ONAVXdt-898HVtg-"


MQ135 gasSensor = MQ135(PinMQ_135);

char auth[] = SECRET_AUTH;
char ssid[] = SECRET_SSID;
char pass[] = SECRET_PASS;
int keyIndex = 0;            // your network key Index number (needed
only for WEP)

WiFiClient  client;

unsigned long myChannelNumber = SECRET_CH_ID;
const char * myWriteAPIKey = SECRET_WRITE_APIKEY;
int aleatorio, h;

BlynkTimer timer; //declaramos el timer que va a regir el tiempo entre
ejecuciones del bucle correspondiente a la APP de blynk
```

```
DHT dht11(d_h_t, DHT11);

#if defined(ARDUINO) && ARDUINO >= 100
#define printByte(args)  write(args);
#else
#define printByte(args)  print(args,BYTE);
#endif

LiquidCrystal_I2C lcd(0x27,16,2);
//hd44780_I2Clcd lcd;

float ref_R0;
float O3_ppm;
float O3_ppb;
float O3_mgm3;
float O3_ugm3;
int concentracion;
int voltaje;
int Res;
boolean leer = false;

int calibrar = 1;
float temp = 0;
float hum = 0;
float ppm_C02 = 0;
int cinco = 5;
int program = 0;
int lectura_ozono = 0;
int lectura_co2 = 0;
int ESTADO = 0;
int valor = 0;
int reset = 0;
int cuenta = 0;
int cuenta2 = 0;
int cuenta3 = 0;
int cuentabarra = 0;
int parar = 0;
int parar2 = 0;
int parar3 = 0;
int barra = 0;
int apagado_zumb = 0;
int encendido_zumb = 1;

long previousMillis = 0;        // will store last time LED was
updated
long previousMillisPant = 0;
long previousMillis3 = 0;
long previousMillisZumb = 0;
long previousMillis5seg = 0;
long previousMillisDHT = 0;

long timer1 = 30000;                      // 30 segundos   ozono
long timer2 = 300000;                     // 270 segundos  espera
(270+30)
long timer3 = 600000;                     // 300 segundos  ventilacion
(300+300)
long timertotal = 610000;                 // tiempo total
 desinfeccion (600+10)
long tiempocuadrostotal = timertotal/16;
long timerZumb = 1000;                    // 1 segundo
```

```
long timer5seg = 5000;                      // 5 segundos
long timerDHT = 5000;
int  timerlcd = timer3/1000;
long currentMillis;
long currentMillisPant;
long currentMillisZumb;
long currentMillis5seg;
long currentMillisDHT;

byte N[8]={
B11111,
B11111,
B11111,
B11111,
B11111,
B11111,
B11111,
B11111,
};

const int LCD_COLS = 16;
const int LCD_ROWS = 2;

// the setup function runs once when you press reset or power the
board
void setup(){

  int status;
  // initialize digital pin LED_BUILTIN as an output.

  Serial.begin(115200);

  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, pass);  // Connect to WPA/WPA2 network. Change this
line if using open or WEP network

  //timer.setInterval(31000L, sendvalue1);
  timer.setInterval(5000L, sendvalue1);

 if(WiFi.status() != WL_CONNECTED){
    Serial.print("Attempting to connect to SSID: ");
    Serial.println(SECRET_SSID);

    while(WiFi.status() != WL_CONNECTED){

      Blynk.begin(auth, ssid, pass); //inicialización de la
comunicación con Blynk

      ThingSpeak.begin(client);  // Initialize ThingSpeak
      Serial.print(".");
      delay(2000);
    }
    Serial.println("\nConnected.");
  }

  //Wire.begin(17,16);
  Wire.begin();
  pinMode(ZUMB, OUTPUT);
  pinMode(RELE1, OUTPUT);
  pinMode(RELE2, OUTPUT);
  pinMode(RELE3, OUTPUT);
```

```
  pinMode(ENTRADA, INPUT);
  //digitalWrite(Umbral_O3, HIGH);   // Pull-Up (Alto = Desconectar el
generador de Ozono)

  dht11.begin();
  lcd.init();  // Inicialzar el LCD
  lcd.backlight();//Encendre la llum de fons.
  lcd.print("PREPARADO");// Escribim missatge inicial a la LCD
  lcd.createChar(0,N);
  digitalWrite(ZUMB, LOW);

  delay(100);

   ESTADO = digitalRead(ENTRADA);
   //ESTADO = analogRead(ENTRADA);

  delay(100);

  if((ESTADO == 0)){

    program = 1;
    valor=1;
    reset=1;

    timer1 = 2000;                        // 2 segundos   ozono (30)
    timer2 = 4000;                       // 2 segundos  espera (270+30)
    timer3 = 6000;                       // 2 segundos  ventilacion
(300+300)
    timertotal = 7000;                   // 7 segundos tiempo total
 desinfeccion (600+10)
    tiempocuadrostotal = timertotal/16;
    timerlcd = timer3/1000;
    timer5seg = 1000;
    cinco = 1;

    previousMillis=millis();
    previousMillisPant=millis();

  }

   // Inicializa el sensor MQ-131
  // - Control de temperatura en pin 2 (no se usa)
  // - Lectura del sensor O3 en pin A1
  // - Modelo: LOW_CONCENTRATION
  // - Valor RL = 680KOhms (680000 Ohms)

  MQ131.begin(2,PinMQ_A, LOW_CONCENTRATION, 680000);

  //Resistencia R0 del sensor MQ-131 instalado, para 10 ppb

  MQ131.setR0(364740.87);

  // Segundos entre lecturas
  // No es necesario más tiempo, porque el calefactor del sensor está
conectado de forma permanente
  MQ131.setTimeToRead(1);

  // Temperatura y Humedad (Habilitar para modificar los valores, por
defecto 20°C / 60%HR)
  // En caso de incorporar un sensor de temperatura y humedad,
actualizar aquí con los valores del sensor
```

```
  MQ131.setEnv(20, 60); // Sin corrección: 20°C / 60 %HR

  //if(digitalRead (Modo)==LOW)

  /*if(digitalRead (Modo)==LOW){
    Serial.println(F(">>> Calibrando el sensor MQ-131 <<<"));
    //lcd.print(F("Calibrando: RO3!"));
    MQ131.calibrate();  // Inicia la calibración del sensor
    Serial.println(F("-----------------------------------"));
    ref_R0 = MQ131.getR0();  // Carga el valor de la resistencia
actual del sensor MQ-131

    MQ131.setR0(ref_R0);

  Serial.println(F("MQ-131 calibrado"));
  Serial.print(F("RO3 = "));
  Serial.print(ref_R0);
  Serial.println(F(" Ohms"));
  Serial.print(F("Intervalo = "));
  Serial.print(MQ131.getTimeToRead());
  Serial.println(F(" segundos"));

  MQ131.setTimeToRead(1);
  delay(5000);
  } else {
    ref_R0 = 158605.95; // Resistencia por defecto del sensor MQ-131
instalado, para 10 ppb

    MQ131.setR0(ref_R0);

  Serial.println(F("MQ-131 calibrado"));
  Serial.print(F("RO3 = "));
  Serial.print(ref_R0);
  Serial.println(F(" Ohms"));
  Serial.print(F("Intervalo = "));
  Serial.print(MQ131.getTimeToRead());
  Serial.println(F(" segundos"));

  MQ131.setTimeToRead(1);
  delay(5000);
  }
*/
  //
**********************************************************************
********** //
  // >>> Habilitar para una concentración de ozono diferente a 10 ppb
al arrancar <<< //
  //
**********************************************************************
********** //
  //ref_R0 = ref_R0 - (ref_R0/6);             // 15 ppb
  //ref_R0 = ref_R0 - (ref_R0/3);             // 25 ppb
  //ref_R0 = ref_R0 - (ref_R0/2);             // 50 ppb
  //ref_R0 = ref_R0 - (ref_R0/2 + ref_R0/12); // 75 ppb
  //
**********************************************************************
********** //
}

void reinicio(){
```

```
  lcd.init();  // Inicialzar el LCD
  lcd.backlight();//Encendre la llum de fons.
  lcd.createChar(0,N);

}

void leerdht11(){

  temp = dht11.readTemperature();
  hum =  dht11.readHumidity();

  while(isnan(temp)||isnan(hum)){

    Serial.println("lectura fallida en el sensor");

    delay(100);

    temp = dht11.readTemperature();
    hum =  dht11.readHumidity();

  }
      Serial.print ( " Temperatura DHT11: " );
      Serial.print (temp);
      Serial.println ( " °C. " );

      Serial.print ( " Humedad DHT11: " );
      Serial.print (hum);
      Serial.println ( " %. " );

      Serial.println ( " ----------------------- " );

}


void leerMQ_131(){

  lectura_ozono = analogRead(PinMQ_A);

  // Lectura y presentación SERIE
  Serial.println(F("------------------------------------"));
  Serial.println(F("Leyendo ..."));

  // Lee los valores de concentración de O3 en el aire
  MQ131.sample();

  // Carga los valores obtenidos en sus variables
  O3_ppm = MQ131.getO3(PPM);
  O3_ppb = MQ131.getO3(PPB);
  O3_mgm3 = MQ131.getO3(MG_M3);
  O3_ugm3 = MQ131.getO3(UG_M3);

  // Muestra los valor por el puerto serie
  Serial.print(O3_ppm);
  Serial.println(F(" ppm"));
  Serial.print(O3_ppb);
  Serial.println(F(" ppb"));
  Serial.print(O3_mgm3);
  Serial.println(F(" mg/m3"));
  Serial.print(O3_ugm3);
  Serial.println(F(" ug/m3"));
  Serial.print("Voltaje MQ-131:");
```

```
  Serial.println(lectura_ozono);


}


void leer_MQ135(){

  ppm_C02 = gasSensor.getPPM();



  /*lectura_co2 = analogRead(PinMQ_135);
  voltaje = lectura_co2 * (3.3 / 4095.0); //Convertimos la lectura en
un val

  Res=20000*((3.3-voltaje)/voltaje);  //Calculamos Rs con un RL de 1k

  concentracion=116.602*pow(Res/151, -2.769); // calculamos la
concentración*/
  Serial.print("ppm MQ-135:");
  Serial.println(ppm_C02);

  /*Serial.print("Concentracion MQ-135:");
  Serial.println(concentracion);
  Serial.print("Voltaje MQ-135:");
  Serial.println(lectura_co2);*/

}


 void sendvalue1()  //creamos un bucle para la APP de blynk
{
  //establecer las variables que se van a enviar al servidor y a qué
campo va a pertenecer cada una
  ThingSpeak.setField(1,temp);
  ThingSpeak.setField(2,hum);
  ThingSpeak.setField(3,O3_ugm3);
  ThingSpeak.setField(4,ppm_C02);

  ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey); //escribir
los valores establecidos

  //establecer las variables que se van a enviar a la App y a qué
campo va a pertenecer cada una
  Blynk.virtualWrite(V5, temp);
  Blynk.virtualWrite(V6, hum);
  Blynk.virtualWrite(V7, O3_ugm3);
  Blynk.virtualWrite(V8, ppm_C02);
  Blynk.virtualWrite(V9, O3_ppb);
  Blynk.virtualWrite(V10, O3_ppm);

 }



// the loop function runs over and over again forever

void loop() {
```

```
  Blynk.run(); //se inicializa la APP de blynk
  timer.run(); //se inicializa el timer de la APP

  currentMillisDHT = millis();

  if(currentMillisDHT-previousMillisDHT > timerDHT){

  previousMillisDHT=millis();

  leerdht11();

  leerMQ_131();

  leer_MQ135();

  }

  //ESTADO = analogRead(ENTRADA);
  ESTADO = digitalRead(ENTRADA);
  //Serial.println(ESTADO);


if((ESTADO == 0) && (reset == 0) && (program==0)){

    valor=1;
    reset=1;

    previousMillis=millis();
    previousMillisPant=millis();

}

    if(valor == 1){


      currentMillis = millis();

        if(currentMillis-previousMillis < timer1){

          cuenta++;

          }

          if(cuenta==1&&parar==0){

            digitalWrite(RELE1, HIGH);
            digitalWrite(RELE3, HIGH);
            delay(100);
            reinicio();
            lcd.clear();
            lcd.setCursor(0,0);
            lcd.print("DESINFECCION "+String(timerlcd));
          }

        if((currentMillis-previousMillis < timertotal)) {

          cuentabarra++;
          currentMillis5seg = millis();

                  if(currentMillis5seg-previousMillis5seg >
timer5seg){
```

```
                        previousMillis5seg=millis();

                        if(timerlcd<0){
                          timerlcd=0;
                        }

                        reinicio();
                        lcd.clear();
                        lcd.setCursor(0,0);
                        lcd.print("DESINFECCION "+String(timerlcd));

                        timerlcd=timerlcd-cinco;

                        for(int i = 0 ; i <= barra ; i++){

                            lcd.setCursor(i,1);
                            lcd.write(byte(0));
                            //delay(10);

                        }

                    }
            }


if(cuentabarra==1){

  switch (barra){

            case 0:
            lcd.setCursor(0,1);
            lcd.write(byte(0));
            break;

            case 1:
            lcd.setCursor(1,1);
            lcd.write(byte(0));
            break;

            case 2:
            lcd.setCursor(2,1);
            lcd.write(byte(0));
            break;

            case 3:
            lcd.setCursor(3,1);
            lcd.write(byte(0));
            break;

            case 4:
            lcd.setCursor(4,1);
            lcd.write(byte(0));
            break;

            case 5:
            lcd.setCursor(5,1);
            lcd.write(byte(0));
            break;

            case 6:
```

```
              lcd.setCursor(6,1);
              lcd.write(byte(0));
              break;

              case 7:
              lcd.setCursor(7,1);
              lcd.write(byte(0));
              break;

              case 8:
              lcd.setCursor(8,1);
              lcd.write(byte(0));
              break;

              case 9:
              lcd.setCursor(9,1);
              lcd.write(byte(0));
              break;

              case 10:
              lcd.setCursor(10,1);
              lcd.write(byte(0));
              break;

              case 11:
              lcd.setCursor(11,1);
              lcd.write(byte(0));
              break;

              case 12:
              lcd.setCursor(12,1);
              lcd.write(byte(0));
              break;

              case 13:
              lcd.setCursor(13,1);
              lcd.write(byte(0));
              break;

              case 14:
              lcd.setCursor(14,1);
              lcd.write(byte(0));
              break;

              case 15:
              lcd.setCursor(15,1);
              lcd.write(byte(0));
              delay(100);
              break;

           }
}


         currentMillisPant = millis();

            if(currentMillisPant-previousMillisPant >
tiempocuadrostotal){

              previousMillisPant=millis();
              barra++;
```

```
            cuenta=0;
            cuentabarra=0;
            parar=1;

          }


        if((currentMillis-previousMillis > timer1) && (currentMillis -
previousMillis < timer2)){

          cuenta2++;

          }

          if(cuenta2==1&&parar2==0){


            digitalWrite(RELE1, LOW);
            digitalWrite(RELE2, LOW);
            digitalWrite(RELE3, LOW);
              delay(100);

          parar2=1;

          }


          if(currentMillis - previousMillis > timer2){

          cuenta3++;

          }


          if(cuenta3==1&&parar3==0){


                digitalWrite(RELE3, LOW);
                digitalWrite(RELE2, HIGH);
                digitalWrite(RELE1, HIGH);
                  delay(100);

              parar3=1;

          }


         if((currentMillis-previousMillis > timer3)) {

              currentMillisZumb = millis();

                digitalWrite(RELE1, LOW);
                digitalWrite(RELE2, LOW);
                digitalWrite(RELE3, LOW);

                if(timerlcd<0){
                timerlcd=0;
                }
                reinicio();
                lcd.clear();
```

```
                    lcd.setCursor(0,0);
                    lcd.print("DESINFECCION "+String(timerlcd));

                    timerlcd=timerlcd-cinco;

                    for(int i = 0 ; i <= barra ; i++){

                        lcd.setCursor(i,1);
                        lcd.write(byte(0));
                        //delay(10);

                    }

   if(currentMillisZumb-previousMillisZumb > timerZumb){

      previousMillisZumb=millis();

        if(apagado_zumb==1){

             digitalWrite(ZUMB,LOW);
              encendido_zumb=1;
              apagado_zumb=0;

              } else if(encendido_zumb==1){

                  digitalWrite(ZUMB,HIGH);
                  encendido_zumb=0;
                  apagado_zumb=1;
               }
        }
     }



if((currentMillis-previousMillis > timertotal)) {

      delay(10);
      digitalWrite(ZUMB,LOW);
      reinicio();

      lcd.clear();
      lcd.print("PREPARADO");

      reset = 0;
      valor=0;
        parar=0;
        parar2=0;
        parar3=0;
          cuenta=0;
          cuenta2=0;
          cuenta3=0;
            barra=0;
            program=0;
              encendido_zumb=1;
              apagado_zumb=0;
                timerlcd = timer3/1000;
      }

 }

}
```