



Informe Final  
**Módulo de Calidad de Aire: BREATH**

Integrantes: Abraham Sebastian Chaffoque Arias  
Alerson Calle Huamán  
Profesores: Guido Juvenal Castillo Ocaña  
Fecha de entrega: 27 de diciembre de 2023  
Lima, Perú

# Índice de Contenidos

<b>1. Resumen</b>	<b>1</b>
<b>2. Introducción</b>	<b>1</b>
<b>3. Problemática</b>	<b>1</b>
<b>4. Objetivos</b>	<b>1</b>
4.1. Generales . . . . .	1
4.2. Específicos . . . . .	2
<b>5. Estado del Arte</b>	<b>2</b>
<b>6. Plateamiento Teórico</b>	<b>4</b>
6.1. Microcontrolador ESP32 . . . . .	4
6.2. Sensor de Infrarrojo No Dispersivo (NDIR) . . . . .	8
6.2.1. Principio de medición electroacústica: . . . . .	9
6.3. Correlación entre Temperatura, Humedad y Presión . . . . .	9
6.3.1. Funcionalidades . . . . .	9
<b>7. Metodología Planteada</b>	<b>10</b>
7.1. Selección de tecnología . . . . .	10
7.2. Rango de operación . . . . .	11
7.3. Configuración y conexiones . . . . .	12
<b>8. Métodos de calibración</b>	<b>12</b>
8.1. Sensor de <i>CO<sub>2</sub></i> SCD40 . . . . .	12
<b>9. Documentación</b>	<b>15</b>
9.1. Niveles de CO <sub>2</sub> Considerados . . . . .	15
9.2. Guía para uso e instalación de módulos de carga viral . . . . .	16
9.3. Conexiones . . . . .	16
9.4. Librerías utilizadas . . . . .	18
<b>10. Diseño mecánico</b>	<b>21</b>
10.1. Diseño Mecánico de las Componentes Usadas . . . . .	21
10.2. Consideraciones . . . . .	22
10.3. Diseño Mecánico del Case . . . . .	23
10.4. Diseño Mecánico del Ensamblaje Final del Módulo . . . . .	24
10.5. Generación del código G . . . . .	25
10.6. Impresión en 3D . . . . .	26
<b>11. Futuras versiones</b>	<b>27</b>
11.1. ESP NOW . . . . .	27
11.2. Tipos de comunicación . . . . .	27

<b>12. Conclusiones</b>	<b>29</b>
12.1. Generales . . . . .	29
12.2. Específicos . . . . .	29
<b>Referencias</b>	<b>29</b>
<b>Anexo A. Código realizado</b>	<b>30</b>

## Índice de Figuras

1. Imagen de ESP32 dev kit v1 Pinout. . . . .	5
2. Imagen de las Secciones del entorno Arduino IDE. . . . .	6
3. Imagen de la Ventana Preferencias del menú Archivo en Arduino IDE. . . . .	7
4. Imagen de la Ventana del Gestor de URLs Adicionales de Tarjetas. . . . .	7
5. Imagen del Menú de Herramientas del Arduino IDE. . . . .	8
6. Imagen de la Ventana del Gestor de Tarjetas del Arduino IDE. . . . .	8
7. Interior del sensor y el principio de medición electroacústica. . . . .	9
8. Imagen de las acciones básicas según el nivel de CO <sub>2</sub> indicado por el Módulo de Calidad de Aire. . . . .	16
9. Esquema de conexiones entre los sensores y módulos realizados en Fritzing. . . . .	17
10. Proyecto de calidad de aire en protoboard . . . . .	20
11. Imagen del diseño en SolidWorks del microcontrolador ESP32. . . . .	21
12. Imagen del diseño en SolidWorks de la pantalla LCD TFT 4". . . . .	21
13. Imagen del diseño en SolidWorks del sensor de CO <sub>2</sub> SCD40. . . . .	21
14. Imagen del diseño en SolidWorks del sensor BME280. . . . .	21
15. Imagen del diseño en SolidWorks del sensor PMS5003T. . . . .	21
16. Imagen del diseño en SolidWorks del sensor PMS5003T. . . . .	22
17. Imagen del diseño en SolidWorks del sensor PMS5003T. . . . .	22
18. Acoplamiento del sensor al entorno ambiente. (a) Buena exposición al aire ambiente gracias a una abertura grande en proximidad al sensor y un volumen muerto pequeño. (b) Acoplamiento moderado al aire ambiente debido a un volumen muerto grande y (c) solo una abertura pequeña en la carcasa del dispositivo. (d) Acoplamiento deficiente ya que el sensor no está separado del aire atrapado dentro de la carcasa del dispositivo y (e) debido a una abertura pequeña lejos del sensor. . . . .	22
19. Acoplamiento del sensor a fuentes de calor externas (vista superior). El color verde representa la PCB del cliente; los círculos rojos indican calor disipado por componentes de auto-calentamiento. (a) Excelente desacoplamiento de fuentes de calor externas gracias a una ranura en la PCB del cliente. (c) Mal desacoplamiento de fuentes de calor externas debido a la proximidad inmediata de componentes de auto-calentamiento. . . . .	23
20. Aislamiento de las turbulencias del aire (vista lateral). La estructura gris representa la carcasa del dispositivo del cliente. (a-b) Buen aislamiento. (c-d) Mal aislamiento, ya que el SCD4X está expuesto directamente a las turbulencias del viento. . . . .	23
21. Imagen del diseño en SolidWorks del cuerpo del case del Módulo de Calidad de Aire de la parte delantera. . . . .	24

22.	Imagen del diseño en SolidWorks del cuerpo del case del Módulo de Calidad de Aire de la parte interior. . . . .	24
23.	Imagen del diseño en SolidWorks del soporte del sensor SCD40. . . . .	24
24.	Imagen del diseño en SolidWorks de la tapa del case del Módulo de Calidad de Aire. . . . .	24
25.	Imagen del diseño en SolidWorks del ensamblaje final del Módulo de Calidad de Aire de la parte frontal. . . . .	25
26.	Imagen del diseño en SolidWorks del ensamblaje final del Módulo de Calidad de Aire de la parte interior. . . . .	25
27.	Imagen de la generación del código G con el programa Ultimaker Cura. . . . .	25
28.	Imagen de una porción del código G para realizar la impresión en 3D. . . . .	26
29.	Imagen de la impresión en 3D. . . . .	26
30.	Imagen de Un emisor y un receptor. . . . .	28
31.	Imagen de Un emisor y varios receptores. . . . .	28
32.	Imagen de Varios emisores y un receptor. . . . .	28
33.	Imagen de la Comunicación bidireccional entre ESP32. . . . .	28
34.	Imagen de la Arquitectura de red ESP MESH. . . . .	28
35.	Imagen de la Arquitectura de red WIFI tradicional. . . . .	29

## Índice de Tablas

1.	Especificaciones de la pantalla TFT LCD de 4"[6]. . . . .	11
2.	Especificaciones del sensor de presión BME280 [7]. . . . .	11
3.	Especificaciones del sensor de partículas por micrado PMS5003T [8]. . . . .	11
4.	Especificaciones del sensor de CO <sub>2</sub> , humedad y temperatura SCD40 [9]. . . . .	12
5.	Tabla Rango de niveles de CO <sub>2</sub> utilizados en el Módulo de Calidad de Aire como referencia. . . . .	15
6.	Conexiones TFT - ESP32. . . . .	17
7.	Conexiones BME - SCD40 - ESP32. . . . .	17
8.	Conexiones PMS5003T - ESP32. . . . .	17

## 1. Resumen

En el presente informe utilizaremos un microcontrolador ESP32 como componente maestro en la comunicación I2C que se utilizará para obtener datos como la concentración de CO<sub>2</sub> y partículas por micrómetro PM2.5, además de valores de temperatura, presión y humedad relativa a través de los componentes esclavos SCD40, BME280 y TFT de 4", siendo el sensor PMS5003T analógico, siendo la comunicación de este tipo serial UART a través de los canales RX y TX, también calibraremos el sensor SCD40 para un ambiente de 650ppm de CO<sub>2</sub>, de modo que se midan concentraciones en el rango de < 850ppm (buena calidad de aire), [850, 1100]ppm (calidad de aire decente) y una mayor a esta mostraría una calidad de aire sofocante o insalubre.

## 2. Introducción

Para desarrollar una ciudad inteligente es necesaria una buena calidad de aire, siendo esta fundamental para la salud y el bienestar de los ciudadanos. Los niveles altos de contaminación en el aire pueden tener efectos negativos en la salud, como problemas respiratorios, enfermedades cardiovasculares, cáncer y otras enfermedades crónicas. La propuesta Breath (medidor de calidad de aire) fue planteada para tomar medidas para reducir la contaminación del aire y mejorar la calidad de vida de los ciudadanos promoviendo el uso de energías renovables y fomentar el transporte público y la movilidad sostenible.

## 3. Problemática

La ciudad de Lima enfrenta un gran problema con la calidad del aire debido a los altos niveles de contaminación del aire en la ciudad. Esta contaminación es causada por la emisión de gases y partículas de los vehículos, la industria y la quema de combustibles fósiles. La contaminación del aire tiene efectos negativos en la salud de las personas, especialmente en aquellos que sufren de problemas respiratorios como el asma y la bronquitis. Además, la contaminación del aire también contribuye al cambio climático y afecta la calidad de vida de las personas en general. Para abordar esta problemática, se han implementado algunas medidas como la restricción del tráfico vehicular, la promoción del uso de transporte público y la incentivación del uso de vehículos menos contaminantes. A pesar de estas medidas, todavía queda mucho por hacer para mejorar la calidad del aire en Lima.

## 4. Objetivos

### 4.1. Generales

1. Desarrollar un módulo con la cual, permita medir la concentración de gases contaminantes en el aire en un ambiente cerrado, y de acuerdo a los niveles establecidos indique la toma de acciones de aviso y/o alerta.

## 4.2. Específicos

1. Cuantificar la concentración de CO<sub>2</sub> del aire dentro de un ambiente cerrado y posteriormente mostrar en la pantalla dicha concentración, estableciendo niveles de concentración.
2. Mediante la revisión de estudios recientes sobre la relación directa del CO<sub>2</sub> acumulado en recintos cerrados y la carga viral que estos pueden llegar a albergar, establecer los niveles de alerta para nuestros Módulos de Calidad de Aire.
3. Establecer avisos y alertas que realice el Módulo de Calidad de Aire cuando se superen los niveles de CO<sub>2</sub> recomendados para tomar acción y garantizar que el entorno dónde está el Módulo sea saludable.
4. Realizar el Diseño asistido por computadora de los componentes a usar para crear modelos digitales tridimensionales, para definir la geometría, las dimensiones y otras características, con el fin de lograr el modelo digital tridimensionales del case del Módulo de Calidad de Aire; posteriormente se realiza la impresión en 3D, para la cual se empleó una impresora 3D para fabricar físicamente el objeto según ese modelo.

## 5. Estado del Arte

Millones de personas mueren prematuramente cada año debido a enfermedades cardiovasculares, enfermedades pulmonares y cáncer causado por la contaminación del aire. Para las muertes prematuras por cáncer, la contaminación del aire es una de las principales causas ambientales. Los contaminantes en el aire existen en forma de gases y partículas sólidas y líquidas en el aire, también llamadas aerosoles. Los aerosoles se presentan en tamaños muy diversos. Entre las diferentes métricas que describen el tamaño de las partículas, la más común es el diámetro aerodinámico (diámetro de la partícula esférica con una densidad de 1 g/m<sup>3</sup> que tiene la misma velocidad de sedimentación que la partícula dada). Tres rangos de tamaño de partículas con límites superiores de 10 µm, 2,5 µm y 1 µm se denominan PM<sub>10</sub>, PM<sub>2,5</sub> y PM<sub>1</sub>, respectivamente.

Se utilizan para definir fracciones de aerosoles con fines reglamentarios. Actualmente, sólo las PM<sub>10</sub> y PM<sub>2,5</sub> están reguladas en forma de normas de calidad del aire ambiental. De estos dos, nos centramos en las PM<sub>2,5</sub>,

debido a su asociación más fuerte con efectos adversos para la salud. El componente PM<sub>2,5</sub> de la contaminación del aire fue responsable de aproximadamente 4,2 millones de muertes prematuras anuales en todo el mundo en 2015. En 2010, China tuvo 1,3 millones de muertes prematuras debido a la exposición a PM<sub>2,5</sub>, India tuvo 575.000 y Pakistán tuvo 105.000 muertes por año. Los 28 países de la Unión Europea (UE) tuvieron 173.000 y los Estados Unidos de América (EE.UU.) 52.000 muertes prematuras anuales. Por lo tanto, endurecer y hacer cumplir los estándares de calidad del aire ambiente para PM<sub>2,5</sub> podría reducir la carga de enfermedades y la mortalidad prematura. Aquí, revisamos los estándares de PM<sub>2,5</sub> en todo el mundo y comparamos estándares en diferentes jurisdicciones [1].

En el contexto de la actual pandemia de COVID-19, la necesidad de garantizar una ventilación adecuada en espacios interiores ha llevado al desarrollo y empleo de tecnologías como los medidores portátiles de dióxido de carbono (CO<sub>2</sub>). Este documento, Resolución

Ministerial RM-675-2022-MINSA - "Guía para el uso de medidores de CO<sub>2</sub> en ambientes de trabajo y escuelas," proporciona directrices detalladas sobre la utilización de estos dispositivos para evaluar y mejorar la calidad del aire en entornos cerrados. Los medidores de CO<sub>2</sub> se presentan como herramientas clave para verificar la renovación permanente del aire en ambientes internos mediante una ventilación adecuada. Se destaca su utilidad al medir el nivel de CO<sub>2</sub> como indicador de la circulación del aire y, por ende, de la acumulación de aerosoles que podrían transmitir la COVID-19. Además, se resalta la importancia de monitorear y regular el nivel de apertura de ventanas y puertas para mantener una ventilación óptima. El documento establece un nivel base de CO<sub>2</sub>, representando la concentración en un ambiente sin personas. Se establece un umbral de concentración de 400 ppm por encima del nivel base como indicador de una ventilación adecuada, recomendando la mejora de la ventilación cuando este umbral se alcanza. Existe un consenso en que los niveles de CO<sub>2</sub> en entornos como escuelas, hogares y oficinas deben mantenerse por debajo de las 1,000 ppm, según la tabla proporcionada [2].

La evaluación de la calidad del aire suele estar motivada por la necesidad de determinar si se ha excedido una norma o estándar. Pero generalmente el cumplimiento de este, oculta otro objetivo, que es proporcionar la información necesaria para estimar la población afectada a la contaminación, y la dinámica de dispersión de los contaminantes. En este contexto, la mayoría de los sistemas de vigilancia de la calidad del aire existentes en Colombia no abordan este aspecto. Para asegurar la información base para dicho estudio, se diseñará la interfaz de captura de datos para la medición y acondicionamiento de las magnitudes relacionadas con el medio ambiente o con las condiciones locales del ecosistema del lugar en donde se encuentra instalada la estación meteorológica. Lo anterior con el propósito-

to de adecuarlos al primer nodo de la red IoT [3].

También se describe un estudio sobre sensores de partículas (PM) de bajo costo y su potencial para mejorar la resolución espacio-temporal de los datos de PM en el aire. En el resumen del estado del arte, se destaca que los PM-LCS ofrecen una oportunidad rentable para mejorar la resolución de los datos. Se menciona que estudios previos se centraron en datos horarios, pero los PM-LCS pueden proporcionar mediciones a resoluciones temporales más finas. Los sensores fueron colocalizados con un monitor de referencia MCERTS (Fidas 200S) durante un año para evaluar y caracterizar su rendimiento a una resolución de 2 minutos. Se enfatiza la importancia de la calibración cuidadosa de los PM-LCS para lograr un rendimiento de grado de referencia.

La metodología involucró dos modelos de sensores (Sensirion SPS30 y Plantower PMS5003) colocalizados con un monitor de referencia, implementando modelos lineales robustos con concentraciones de partículas reportadas por el sensor y humedad relativa. Se probaron varios métodos de calibración, incluyendo regresión lineal, regresión lineal robusta, correcciones  $\kappa$ -Koehler y Laulainen, máquinas de soporte vectorial (SVM) y máquinas de aumento de gradiente (GBM).

En cuanto a los resultados, se logró un rendimiento de grado de referencia para la concentración de PM2.5, demostrando que, con calibración cuidadosa, los PM-LCS pueden complementar eficazmente los equipos de referencia en redes de múltiples nodos con alta resolución espacio-temporal. Se destaca la importancia de la metodología de calibración para lograr este rendimiento, con una caracterización detallada de varios métodos de calibración en diferentes etapas del estudio, incluyendo una evaluación mensual y una validación cruzada de los modelos desarrollados.[4]

Las ventajas de esta tecnología son evidentes, ya que implica un menor costo tanto en

la adquisición del equipo como en su operación y mantenimiento, en comparación con las tecnologías actuales utilizadas en el monitoreo de la calidad del aire. Además, la capacidad de comunicación en tiempo real de los sensores de bajo costo (SBC) permite correcciones inmediatas, mejorando la capacidad de respuesta frente a eventos o condiciones cambiantes. Asimismo, la generación de datos en volumen posibilita el desarrollo de programas de prevención, contingencia y modelos de proyección.

Sin embargo, es importante señalar algunas desventajas asociadas al uso de SBC. En primer lugar, la falta de protocolos estandarizados para calibración y operación representa un desafío, ya que puede afectar la consistencia y comparabilidad de los datos entre diferentes dispositivos. Además, las limitaciones

climáticas, especialmente en ciudades con climas extremos, también son una preocupación. Los SBC suelen tener restricciones en los rangos de operación de temperatura, con muchos sensores comerciales alcanzando hasta 55°C. Trabajar a estas temperaturas puede aumentar el margen de error de los sensores.

En conclusión, el desarrollo constante de tecnologías como los SBC para el monitoreo de la calidad del aire es crucial para lograr su aplicación óptima. Aunque los modelos actuales de SBC no cumplen con todas las características deseadas, proporcionan una base viable para la implementación de estos dispositivos. La comunidad científica demuestra un claro interés en estos temas, y la necesidad de un mayor volumen de datos para la toma de decisiones sugiere un horizonte de mejora constante en estos dispositivos.[5]

## 6. Plateamiento Teórico

### 6.1. Microcontrolador ESP32

ESP32 es la denominación de una familia de chips SoC de bajo coste y consumo de energía, con tecnología Wi-Fi y Bluetooth de modo dual integrada.

#### 1. Características del ESP32:

- Procesador:
  - CPU: microprocesador de 32-bit Xtensa LX6 de doble núcleo (o de un solo núcleo), operando a 160 o 240 MHz y rindiendo hasta 600 DMIPS
  - Co-procesador de ultra baja energía (ULP)
- Memoria RAM:
  - 520 KB SRAM
- Conectividad inalámbrica:
  - Wi-Fi: 802.11 b/g/n
  - Bluetooth: v4.2 BR/EDR y BLE
- Interfaces periféricas:
  - 12-bit SAR ADC de hasta 18 canales
  - 2 × 8-bit DACs
  - 10 × sensores de tacto (sensores capacitivos GPIOs)

- 4 × SPI
- 2 × interfaces I<sup>2</sup>S
- 2 × interfaces I<sup>2</sup>C
- 3 × UART
- Controlador host SD/SDIO/CE-ATA/MMC/eMMC
- Controlador esclavo SDIO/SPI
- Interfaz Ethernet MAC con DMA dedicado y soporte para el protocolo IEEE 1588 Precision Time Protocol
- Bus CAN 2.0
- Controlador remoto infrarrojo (TX/RX, hasta 8 canales)
- Motor PWM
- LED PWM (hasta 16 canales)
- Sensor de efecto Hall
- Pre-amplificador analógico de ultra baja potencia
- Seguridad:
  - Soporta todas las características de seguridad estándar de IEEE 802.11, incluyendo WFA, WPA/WPA2 y WAPI
  - Arranque seguro
  - Cifrado flash
  - 1024-bit OTP, hasta 768-bit para clientes
  - Criptografía acelerada por hardware: AES, SHA-2, RSA, criptografía de curva elíptica (ECC), generador de números aleatorios (RNG)
- Administración de energía:
  - Regulador interno de baja caída
  - Dominio de poder individual para RTC
  - Corriente de 5 $\mu$ A en modo de suspensión profundo
  - Despierta por interrupción de GPIO, temporizador, medidas de ADC, interrupción por sensor de tacto capacitivo

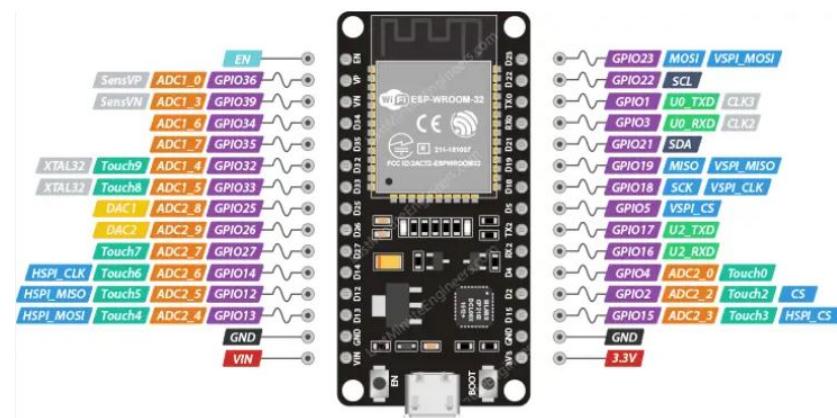


Figura 1: Imagen de ESP32 dev kit v1 Pinout.

- Entorno de desarrollo integrado de Arduino (Arduino IDE)

Arduino IDE es una aplicación multiplataforma (para Windows, macOS, Linux) que está escrita en el lenguaje de programación Java. Se utiliza para escribir y cargar programas en placas compatibles con Arduino, pero también, con la ayuda de núcleos de terceros, se puede usar con placas de desarrollo de otros proveedores.

En la Figura (2). se muestra la ventana de desarrollo del Arduino IDE, para relacionarnos con el entorno se detallan sus secciones a continuación:

- Barra de menús: En esta barra se encuentran los menús que permiten acceder a la gran mayoría de acciones y funcionalidades disponibles en el entorno de desarrollo.
- Barra de accesos directos: En esta barra se encuentran un grupo de botones con las acciones más comúnmente realizadas (como compilar y cargar el código al Arduino).
- Área de pestañas: En esta área se muestran las pestañas con los nombres correspondientes a los archivos abiertos.
- Área de edición: En esta área es dónde se escribe el código.
- Área de Estado: Esta área muestra una barra de progreso cuando se compila o carga el código al Arduino.
- Consola de salida: En esta sección se muestran las salidas generadas por el compilador.
- Barra de información: En esta área se muestra información sobre la configuración del Arduino IDE.



Figura 2: Imagen de las Secciones del entorno Arduino IDE.

- Pasos para la instalación de la placa ESP32 en el Arduino IDE

Hay que instalar la placa de ESP32 en el Arduino IDE mediante los siguientes pasos:

- Paso 1: Adicionar las URLs para la placa ESP32, ingresando a Archivo > Preferencias. Se abrirá una ventana donde se deberá hacer clic en el botón de Gestor de tarjetas adicionales.

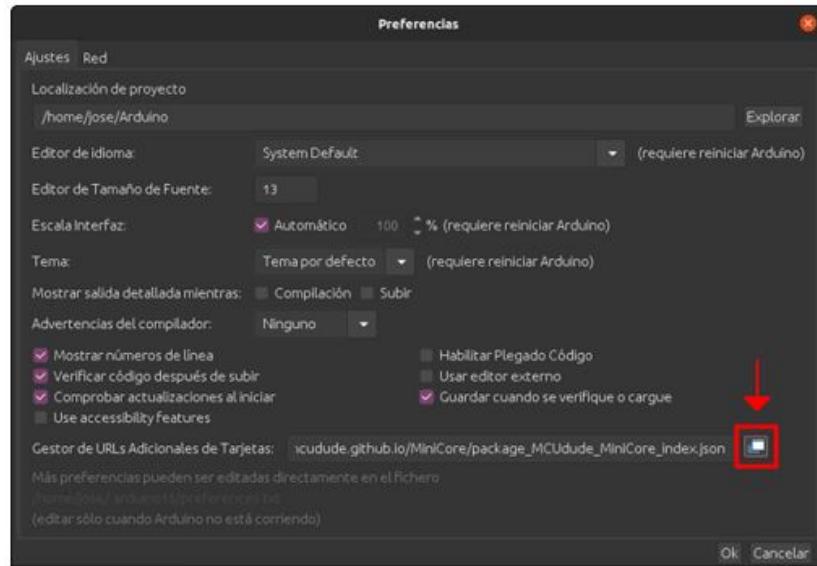


Figura 3: Imagen de la Ventana Preferencias del menú Archivo en Arduino IDE.

En la nueva ventana, pegar las siguientes URLs:

[https://dl.espressif.com/dl/package\\_esp32\\_index.json](https://dl.espressif.com/dl/package_esp32_index.json)

[https://resource.heltec.cn/download/package\\_heltec\\_esp32\\_index.json](https://resource.heltec.cn/download/package_heltec_esp32_index.json), luego presionar el botón “Ok”.



Figura 4: Imagen de la Ventana del Gestor de URLs Adicionales de Tarjetas.

- b) Paso 2: Instalar core y placa ESP32. Ingresando a Herramientas > Placas > Gestor de Tarjetas, escribir “esp32” en la barra de búsqueda. Luego seleccionar el desarrollador y la versión e instalar. De preferencia instalar la versión de “Espressif Systems” y la última versión disponible.

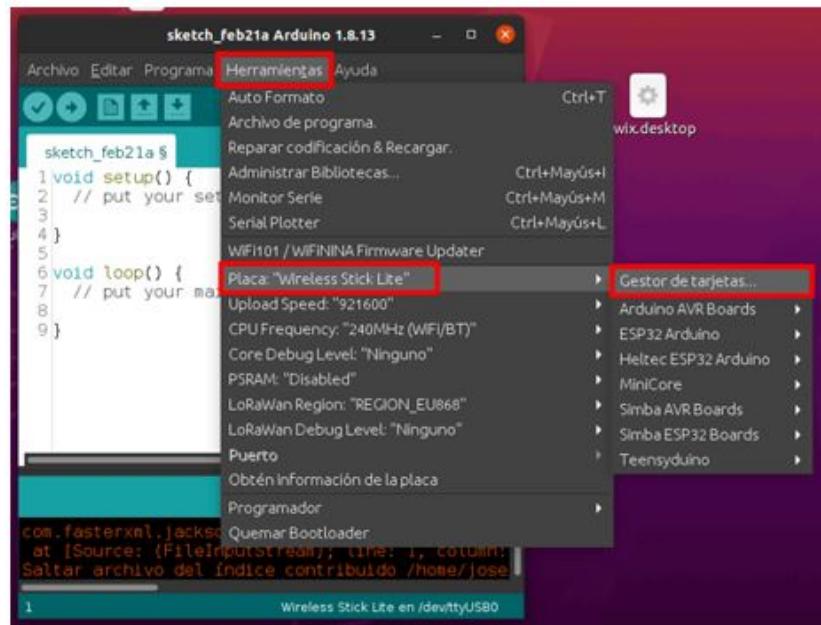


Figura 5: Imagen del Menú de Herramientas del Arduino IDE.



Figura 6: Imagen de la Ventana del Gestor de Tarjetas del Arduino IDE.

Para más información: <https://programarfácil.com/esp8266/programar-esp32-ide-arduino/>

## 6.2. Sensor de Infrarrojo No Dispersivo (NDIR)

En los sensores NDIR que solemos utilizar, el funcionamiento se basa en la absorción que el CO<sub>2</sub> realiza en un haz de infrarrojos. En un sensor NDIR, hay un emisor de infrarrojos y un sensor que detecta el infrarrojo. Cuanto más CO<sub>2</sub> haya entre el emisor de infrarrojos y el sensor que lo detecta, menos infrarrojos detecta el sensor. Los sensores electroacústicos se basan en los efectos que la concentración de CO<sub>2</sub> produce en las ondas sonoras.

### 6.2.1. Principio de medición electroacústica:

En una cámara de medición casi completamente cerrada, se emite una luz de banda estrecha que coincide con las longitudes de onda que son absorbidas por las moléculas de CO<sub>2</sub>.

Las moléculas de CO<sub>2</sub> en la cámara de medición absorben una parte de la luz radiada, mientras que otras moléculas no contribuyen a la absorción debido al espectro de la luz emitida.

Cuanta más moléculas de CO<sub>2</sub> haya en la celda de medición, más energía se absorberá.

La energía absorbida de las moléculas de CO<sub>2</sub> excita principalmente las vibraciones moleculares, lo que da lugar a un aumento en la energía translacional de las moléculas y, debido a la cámara de medición cerrada, a un aumento de la presión en la cámara.

Una modulación de la fuente de luz provoca un cambio periódico de presión en la celda de medición, que se puede medir con un micrófono. La señal del micrófono sirve así para medir la cantidad de moléculas de CO<sub>2</sub> presentes en la cámara de medición y se puede utilizar para calcular la concentración de CO<sub>2</sub>.

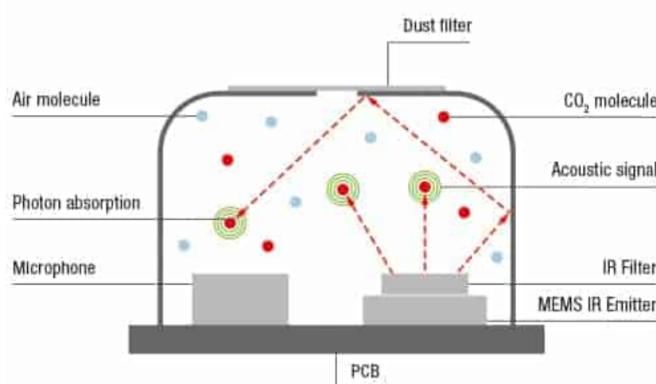


Figura 7: Interior del sensor y el principio de medición electroacústica.

## 6.3. Correlación entre Temperatura, Humedad y Presión

El sensor de precisión BME280 de Bosch se utiliza en una amplia variedad de aplicaciones, desde el monitoreo del clima hasta controles de juegos y medición de altitud donde se requiere una precisión de solo unos pocos pies.

Este sensor es fácil de usar, viene precalibrado y no requiere componentes adicionales, lo que permite medir la humedad relativa, la temperatura, la presión barométrica y la altitud sin complicaciones.

### 6.3.1. Funcionalidades

- **Medición de Temperatura:** Además de las mediciones que entran en el rango de temperaturas dispuesto en la sección **Selección de tecnología**, es importante destacar que las mediciones de temperatura se utilizan internamente para calibrar los sensores de presión y humedad. Debido a que el sensor se calienta por sí mismo, la temperatura medida suele ser ligeramente más alta que la temperatura real.
- **Medición de Humedad:** El BME280 puede medir la humedad relativa en un rango del 0 al 100 % con una precisión del ±3 %. Según la hoja de datos, el sensor puede medir hasta

un 100 % de humedad en un rango de temperatura de 0 a  $60^{\circ}C$ . Sin embargo, la máxima humedad medible disminuye a temperaturas extremadamente altas y bajas.

- **Medición de Presión:** De la misma forma que la humedad, además de medir en un rango de [300, 1100]hPa, En el rango de temperatura de 0 a  $65^{\circ}C$ , se logra una precisión completa, lo que resulta en una precisión de medición de altitud de aproximadamente  $\pm 1$  metro. Fuera de ese rango, la precisión disminuye a 1.7 hPa.
- **Cálculo de Altitud / Elevación:** Es fundamental entender la distinción entre Altitud Absoluta y Altitud Relativa. La "altitud absoluta" se refiere a la altura sobre el nivel del mar (MSL), mientras que la "altitud relativa" se refiere a la altura sobre el nivel del suelo (AGL). Es importante señalar que el BME280 no puede medir directamente la altitud, pero puede estimarla utilizando lecturas de presión. Dado que el BME280 es muy bueno para medir la presión, puede calcular la altitud relativa con precisión. Por ejemplo, si conoces la altitud de un objeto sobre una mesa y lo mueves al suelo, el BME280 mostrará una disminución de altura de 2 pies.

Sin embargo, medir la altitud absoluta se vuelve un poco más complicado, ya que el BME280 necesita conocer la presión actual al nivel del mar. Para obtener una medición precisa de la altitud absoluta, se proporciona la constante *SEALEVELPRESSURE* en el código de ejemplo a continuación, que debes actualizar con la presión al nivel del mar actual en tu ubicación.

## 7. Metodología Planteada

### 7.1. Selección de tecnología

- Utilización del microcontrolador ESP32-V1 para la integración de los sensores y la gestión de datos.
- Incorporación del sensor SCD40 para la medición de concentraciones de CO<sub>2</sub> en ppm, temperatura y humedad relativa.
- Integración del sensor BME280 para la medición de la presión barométrica.
- Inclusión del sensor PMS5003T para la medición de las partículas de tamaño  $1\mu m$ ,  $2.5\mu m$  y  $10\mu m$ .
- Integración de pantalla TFT LCD MSP4021 SPI de 4 pulgadas.

## 7.2. Rango de operación

Tabla 1: Especificaciones de la pantalla TFT LCD de 4" [6].

Nombre	Descripción
Color de pantalla	RGB 65K color
SKU	MSP4021
Tamaño de pantalla	4.0 pulgadas
Tipo/Driver IC	TFT/ST7796S
Resolución	480x320 p
Área activa	55.68x83.52 (mm)
Tamaño PCB	61.74x108.04 (mm)
Temperatura de almacenamiento	[-10, 60] °C
Temperatura de operación	[-20, 70] °C
Voltaje de funcionamiento	3.3/5 V

Tabla 2: Especificaciones del sensor de presión BME280 [7].

Parametro	Condición	ε	u
T. operacional	[-40, 85]		°C
Rango de presión	[300, 1100]		hPa
Suministro de voltaje	[1.2, 3.6]		V
	[300, 1100]	±1.7	hPa
	[-20, 0] °C		
Presión absoluta	[300, 1100]	±1.0	hPa
de presión	[0, 65] °C		
	[1100, 1250]	±1.5	hPa
	[25, 40] °C		
Presión relativa	[700, 900]	±0.12	hPa
V = 3.3 V	[25, 40] °C		

Tabla 3: Especificaciones del sensor de partículas por micrado PMS5003T [8].

Parametro	Condición	ε	u
Suministro DC	[4.5, 5.5]	-	Volt (V)
RH operacional	[0, 99]	-	%RH
T operacional	[-10, 60]	-	°C
PM 1.0	[0, 1000]	±10 %	µg/m³
PM 2.5	[0, 1000]	±10 %	µg/m³
PM 10	[0, 1000]	±10 %	µg/m³

Tabla 4: Especificaciones del sensor de CO<sub>2</sub>, humedad y temperatura SCD40 [9].

Parámetro	Condición	Valor
Suministro DC	-	[2.4, 5.5]V
Temperatura cond	-	[-10, 60] °C
Humedad cond	-	[0, 95] % RH
CO <sub>2</sub> (salida)	-	[0, 40 000]ppm
Presición	[400, 2000]ppm	±50ppm + 5 %
Humedad (salida)	-	[0, 100] % RH
Presición	[15, 35]°C, [20, 65] %RH [-10, 60]°C , [0, 100] %RH	±6 %RH ±9 %RH
Temperatura (salida)	-	[-10, 60]°C
Presición	[15, 35]°C [-10, 60]°C	±0.8°C ±1.5°C

### 7.3. Configuración y conexiones

- Integración de los sensores al microcontrolador ESP32-V1 y configuración adecuada de los parámetros de medición.
- Establecimiento de la conexión con una pantalla LCD para visualizar los datos en tiempo real.

## 8. Métodos de calibración

### 8.1. Sensor de CO<sub>2</sub> SCD40

La calibración de un sensor NDIR (Non-Dispersive Infrared) de CO<sub>2</sub> como lo es el SCD40 es un paso esencial para garantizar mediciones precisas y confiables de la concentración de dióxido de carbono en el aire. Estos sensores operan en base a la absorción de luz infrarroja por parte del CO<sub>2</sub> presente en la muestra de gas. Generalmente se implica la realización de ajustes específicos, tanto en términos de configuración electrónica como de software, para compensar factores que podrían afectar la lectura del sensor. Sin embargo, dado que en las especificaciones del fabricante de este,

- **Configuración Inicial**

```

1 #include <Wire.h>
2
3 // Dirección del sensor SCD4x
4 const int16_t SCD_ADDRESS = 0x62;
```

En esta parte, se incluye la librería Wire

para la comunicación I2C y se define la dirección del sensor SCD4x como 0x62.

- **Configuración del Entorno y Medición Inicial**

```

1 void setup() {
2     // Variables para almacenar mediciones
3     float co2, temperatura, humedad;
4     uint16_t calibracion;
```

```

5  uint8_t datos[12], contador, repeticion;
6  uint8_t retorno;
7
8  Serial.begin(9600);
9  while(!Serial);
10 Wire.begin();
11 delay(1000);
12
13 Wire.beginTransmission(SCD_ADDRESS);
14 Wire.write(0x21);
15 Wire.write(0xb1);
16 Wire.endTransmission();
17
18 Serial.println("5 min para la equilibración");
19 delay(5 * 60 * 1000);
20
21 Serial.println("CO2 antes de la calibración")
    ↪ ;

```

Esta parte del código se encarga de la configuración inicial, incluyendo la inicialización de la comunicación serial, la espera de la conexión serial y la configuración del sensor para iniciar las mediciones periódicas cada 5 segundos.

La aplicación espera 5 minutos para permitir que el sensor se estabilice antes de realizar mediciones.

- **Medición Inicial y Presentación de Datos**

```

1 // Mide 5 veces
2 for(repeticion = 0; repeticion < 5;
    ↪ repeticion++) {
3     // Lee los datos de la medición
4     Wire.requestFrom(SCD_ADDRESS, 12);
5     contador = 0;
6     while (Wire.available()) {
7         datos[contador++] = Wire.read();
8     }
9
10 // Conversión de punto flotante según la
    ↪ hoja de datos
11 co2 = (float)((uint16_t)datos[0] << 8 |
    ↪ datos[1]);
12 temperatura = -45 + 175 * (float)((
    ↪ uint16_t)datos[3] << 8 | datos[4]) /
    ↪ 65536;

```

```

13 humedad = 100 * (float)((uint16_t)datos
    ↪ [6] << 8 | datos[7]) / 65536;
14
15 // Imprime los valores
16 Serial.print("# ");
17 Serial.print(co2);
18 Serial.print("\t");
19 Serial.print(temperatura);
20 Serial.print("\t");
21 Serial.print(humedad);
22 Serial.println();
23
24 delay(2000);
25 }

```

En esta parte, se realiza la medición del sensor cinco veces. Los datos se leen, se convierten a valores de punto flotante y se imprimen en el formato deseado. Se espera un breve periodo de tiempo entre cada medición.

Esta sección es crucial para entender cómo se obtienen y procesan las mediciones del sensor.

- **Finalización de la Medición y Calibración Inicial**

```

1
2 // Detén la medición del sensor
3 Wire.beginTransmission(SCD_ADDRESS);
4 Wire.write(0x3f);
5 Wire.write(0x86);
6 retorno = Wire.endTransmission();
7 Serial.println(retorno);
8
9 // Espera al sensor
10 delay(20);
11
12 // Suponiendo que una referencia externa
    ↪ muestra 650 ppm
13 calibracion = 650;
14 Serial.print("# Calibrando con valor de
    ↪ referencia [ppm]: ");
15 Serial.println(calibracion);

```

Esta sección detiene la medición del sensor y espera brevemente antes de iniciar

el proceso de calibración. En este caso, se establece un valor de referencia para la calibración (en este ejemplo, 650 ppm de CO<sub>2</sub>).

### • Preparación y Ejecución de la Recalibración

```

1 // Prepara buffer con datos para la
   ↪ recalibración
2 datos[0] = (calibracion & 0xff00) >> 8;
3 datos[1] = calibracion & 0x00ff;
4 datos[2] = CalcCrc(datos);
5
6 // Envía comando para realizar la
   ↪ recalibración forzada
7 Wire.beginTransmission(SCD_ADDRESS);
8 Wire.write(0x36);
9 Wire.write(0x2F);
10 Wire.write(datos[0]);
11 Wire.write(datos[1]);
12 Wire.write(datos[2]);
13 retorno = Wire.endTransmission();
14 Serial.println(retorno);
15
16 delay(400);

```

Aquí se prepara un búfer con los datos necesarios para la recalibración y se envía el comando al sensor para ejecutar la recalibración forzada. También se espera un breve periodo después de la recalibración.

### • Lectura de Datos Despues de la Recalibración

```

1 // Lee datos: 2 bytes de corrección, 1 byte
   ↪ de CRC
2 Wire.requestFrom(SCD_ADDRESS, 3);
3 contador = 0;
4 while (Wire.available()) {
5   datos[contador++] = Wire.read();
6 }
7
8 if(CalcCrc(datos) != datos[2])
9   Serial.println("# ERROR: valor de
      ↪ retorno del CRC de recalibración");
10
11 calibracion = ((uint16_t)datos[0] << 8 |
   ↪ datos[1]);

```

```

12
13 Serial.print("# Valor después de la
   ↪ recalibración\n# ");
14 Serial.println(calibracion-32768);
15
16 // Formato de salida
17 Serial.println("CO2(ppm)\tTemperatura(
   ↪ degC)\tHumedadRelativa(percent)");
18
19 // Inicia la medición del sensor nuevamente
   ↪ en modo periódico
20 Wire.beginTransmission(SCD_ADDRESS);
21 Wire.write(0x21);
22 Wire.write(0xb1);
23 Wire.endTransmission();
24
25 // Espera a que la primera medición finalice
   ↪ (> 5 s)
26 delay(10000);
27 }

```

Finalmente, esta parte lee los datos después de la recalibración, verifica el CRC y presenta los resultados. Luego, configura el sensor para continuar las mediciones periódicas.

### • Bucle Principal para Mediciones Continuas

```

1 void loop() {
2   float co2, temperatura, humedad;
3   uint8_t datos[12], contador;
4
5   // Envía el comando de lectura de datos
6   Wire.beginTransmission(SCD_ADDRESS);
7   Wire.write(0xec);
8   Wire.write(0x05);
9   Wire.endTransmission();
10
11 // Lee los datos de la medición
12 Wire.requestFrom(SCD_ADDRESS, 12);
13 contador = 0;
14 while (Wire.available()) {
15   datos[contador++] = Wire.read();
16 }
17
18 // Conversión de punto flotante según la
   ↪ hoja de datos
19 co2 = (float)((uint16_t)datos[0] << 8 |
   ↪ datos[1]);

```

```

20  temperatura = -45 + 175 * (float)((
    ↪ uint16_t)datos[3] << 8 | datos[4]) /
    ↪ 65536;
21  humedad = 100 * (float)((uint16_t)datos[6]
    ↪ << 8 | datos[7]) / 65536;
22
23 // Imprime los resultados
24 Serial.print(co2);
25 Serial.print("\t");
26 Serial.print(temperatura);
27 Serial.print("\t");
28 Serial.print(humedad);
29 Serial.println();
30

```

```

31 // Espera 5 s para la próxima medición
32 delay(5000);
33 }

```

Este es el bucle principal que se encarga de realizar mediciones continuas. Envía el comando de lectura de datos al sensor, procesa los resultados y espera un breve periodo antes de realizar la siguiente medición.

## 9. Documentación

### 9.1. Niveles de CO<sub>2</sub> Considerados

Siguiendo la guía anexada en la Resolución Ministerial del MINSA [2], se tomaron los niveles base (NB) de ambientes seleccionados, que en todos los casos fue alrededor de 450 ppm, y se establecieron los rangos de salubridad de concentración de CO<sub>2</sub> que se muestran en la (5).

Los rangos de concentración de CO<sub>2</sub> establecidos servirán como referencia para que el MCV realice determinadas acciones de acuerdo a sus mediciones del momento.

Tabla 5: Tabla Rango de niveles de CO<sub>2</sub> utilizados en el Módulo de Calidad de Aire como referencia.

Concentración de CO <sub>2</sub>	Calidad	Aviso
< 850 ppm	Bueno	Color Verde
[850 – 1100] ppm	Medio	Color Anaranjado
> 1100 ppm	Insalubre	Color Rojo

Las recomendaciones básicas a realizar según el rango del nivel de CO<sub>2</sub> que indique el Módulo de Calidad de Aire son:

- Nivel BUENO:

Mantener el distanciamiento social y el uso adecuado de las mascarillas.

- Nivel MEDIO:

Mantener las recomendaciones del nivel BUENO y abrir las puertas y ventanas que permitan el flujo del aire hacia el exterior o hacia espacios mejor ventilados; para acelerar el flujo de aire hacia el exterior se puede hacer uso de un ventilador en una salida del aire. De ser posible detener el ingreso de más personas y/o reducir el tiempo de permanencia en el recinto.

- Nivel INSALUBRE:

Seguir las recomendaciones de los niveles anteriores y de preferencia evacuar el recinto por un tiempo hasta que el ambiente ventile adecuadamente y se regularice el nivel de concentración de CO<sub>2</sub>.

Estas recomendaciones se pueden observar en la Figura (8):



Figura 8: Imagen de las acciones básicas según el nivel de CO<sub>2</sub> indicado por el Módulo de Calidad de Aire.

## 9.2. Guía para uso e instalación de módulos de carga viral

Basados en la Resolución Ministerial RM-675-2022-MINSA [] y los Lineamientos para el uso adecuado del SCD40 se tomaron en cuenta las siguientes consideraciones para la ubicación de un sensor en un aula/oficina.

- A un metro o más de distancia de las personas.
- A una altura entre un metro y un metro y medio del piso.
- Lo más alejado posible de zonas de turbulencia ocasional o continua del aire, como son por ejemplo: puertas, ventanas, ventiladores y salidas de aire acondicionado.
- Evitar zonas con exposición directa al Sol o muy cercanas a fuentes de calor.
- De ser posible, ubicarlo en el centro del aula o ambiente de trabajo, para que entre otras razones excluidas las anteriores el módulo pueda censar una mayor superficie y más homogénea del aire en el ambiente.

## 9.3. Conexiones

En caso de que no queramos tener la pantalla del dispositivo encendida de forma prolongada, o queramos que esta se maneje de manera rutinaria desconectar el pin LED del 3V3 del ESP32 y asignar un nuevo pin. Tenemos que tener en cuenta que a partir de SDO(MISO) las conexiones se realizan para posterior uso del panel tactil del dispositivo (puede ser omitido).

Tabla 6: Conexiones TFT - ESP32.

4.0" TFT	ESP32
VCC	3V3
GND	GND
CS	D15
RESET	D4
DC/RS	D2
SDI (MOSI)	D23
SCK	D18
LED	3V3
SDO (MISO)	D19
$T_CLK$	SCK
$T_CS$	D5
$T_{DIN}$	SDI (MOSI)
$T_{DO}$	SDO (MISO)
$T_{IRQ}$	D27

Para la configuración de nuestro dispositivo, tenemos que tener en cuenta que el módulo que utilizamos es el ESP32 DEVIKIT V1, teniendo únicamente disponibles los pines 21 y 22 como SDA y SCL respectivamente para el

protocolo de comunicación I2C.

Tabla 7: Conexiones BME - SCD40 - ESP32.

BME280	SCD40	ESP32
GND	GND	GND
VCC	VCC	3V3
SCL	SCL	22
SDA	SDA	21

Si se posee una fuente adicional de 5V de preferencia conectar a esta, en lugar de la fuente de ESP32 conectando tierra entre estas.

Tabla 8: Conexiones PMS5003T - ESP32.

PMS5003T	ESP32
GND	GND
VCC	VIN (5V)
RX	TX2
TX	RX2

De modo que las conexiones se verían como en la figura

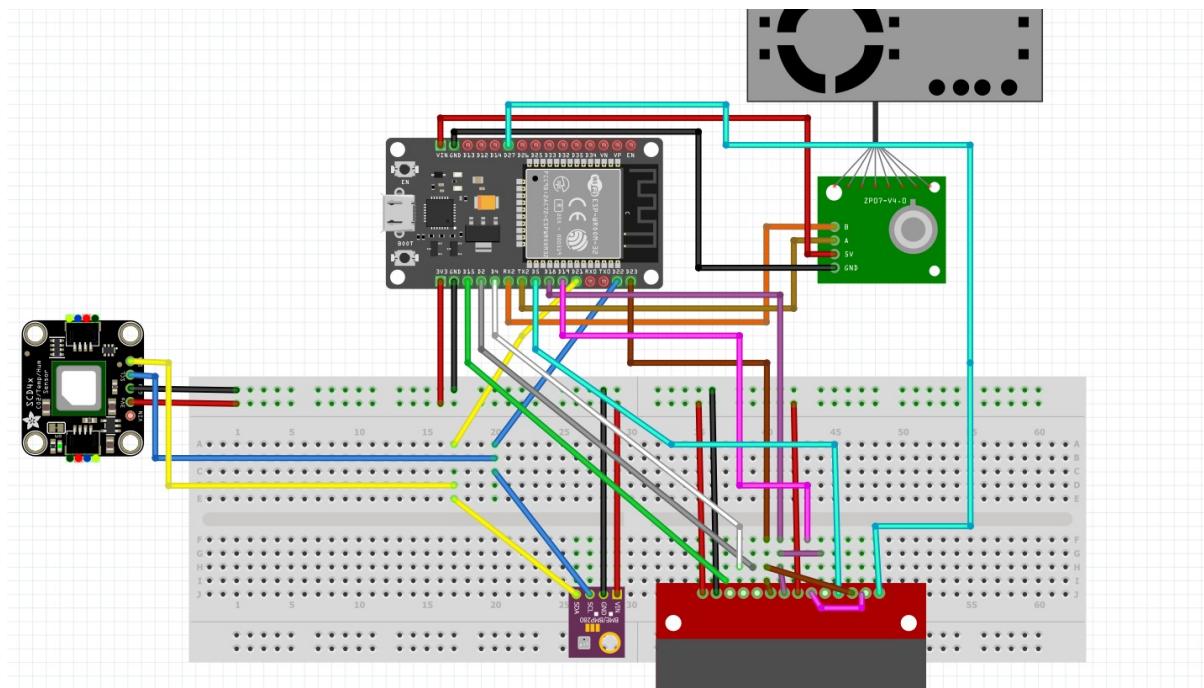


Figura 9: Esquema de conexiones entre los sensores y módulos realizados en Fritzing.

## 9.4. Librerías utilizadas

Puesto a que se utiliza la comunicación I2C para ambos sensores (SCD40, BME280) además de en la pantalla TFT estos se tendrán que conectar en paralelo al ESP32, por lo que si se requiere cambiar de sensor tenemos que tener en cuenta añadir su dirección como se verá a continuación.

```

1 #include <Adafruit_BME280.h>
2 #include "SparkFun_SCD4x_Arduino_Library.h"
3 #include <Wire.h>
4 #include <TFT_eSPI.h>
5
6 #define RXD2 16 // To sensor TXD
7 #define TXD2 17 // To sensor RXD
8
9 SCD4x SCD40;
10 Adafruit_BME280 BME280;
11
12 TFT_eSPI tft = TFT_eSPI();
13
14 void setup(void){
15   Serial.begin(9600);
16   ...
17   BME280.begin(0x76);
18   SCD40.begin(0x62);
19   ...
20 }
```

Si bien, podemos ver como se incluyen las librerías de los sensores SCD40, PMS5003T y BME280, habría una problemática con la librería <TFT-eSPI.h>, puesto a que esta librería fue realizada para varios modelos de pantalla de diferentes tamaños y características (tactibilidad, color, tamaño pixeles). Manualmente nos situamos en el archivo *UserSetup.h* librería y descomentamos el modelo utilizado además de comentar el modelo que está por defecto:

```

1 // Tell the library to use 8 bit parallel mode (
2   //→ otherwise SPI is assumed)
3 // #define TFT_PARALLEL_8_BIT
4 // Display type – only define if RPi display
```

```

5 // #define RPI_DISPLAY_TYPE // 20MHz
6   //→ maximum SPI
7 // Only define one driver, the other ones must be
8   //→ commented out
9 // #define ILI9341_DRIVER
10 // #define ST7735_DRIVER // Define
11   //→ additional parameters below for this
12   //→ display
13 // #define ILI9163_DRIVER // Define
14   //→ additional parameters below for this
15   //→ display
16 // #define S6D02A1_DRIVER
17 // #define RPI_ILI9486_DRIVER // 20MHz
18   //→ maximum SPI
19 // #define HX8357D_DRIVER
20 // #define ILI9481_DRIVER
21 // #define ILI9486_DRIVER
22 // #define ILI9488_DRIVER
23 // #define ST7789_DRIVER // Full
24   //→ configuration option, define additional
25   //→ parameters below for this display
26 // #define ST7789_2_DRIVER // Minimal
27   //→ configuration option, define additional
28   //→ parameters below for this display
29 // #define R61581_DRIVER
30 // #define RM68140_DRIVER
31 // #define ST7796_DRIVER
32 // #define SSD1963_480_DRIVER // Untested
33 // #define SSD1963_800_DRIVER // Untested
34 // #define SSD1963_800ALT_DRIVER //
35   //→ Untested
```

Una vez realizado esto, podemos hacer cambios en la asignación de pines que se visualiza en la Tabla de conexiones TFT-ESP32

```

1 // For ESP32 Dev board (only tested with ILI9341
2   //→ display)
3 // The hardware SPI can be mapped to any pins
4
5 #define TFT_MISO 19
6 #define TFT_MOSI 23
7 #define TFT_SCLK 18
8 #define TFT_CS 15 // Chip select control pin
9 #define TFT_DC 2 // Data Command control
10   //→ pin
11 #define TFT_RST 4 // Reset pin (could
12   //→ connect to RST pin)
```

```

10 // #define TFT_RST -1 // Set TFT_RST to
   ↪ -1 if display RESET is connected to
   ↪ ESP32 board RST
11
12 #define TOUCH_CS 14    // Chip select pin (
   ↪ T_CS) of touch screen
13
14 // #define TFT_WR 22    // Write strobe for
   ↪ modified Raspberry Pi TFT only
15
16 // For the M5Stack module use these #define lines
17 // #define TFT_MISO 19
18 // #define TFT_MOSI 23
19 // #define TFT_SCLK 18
20 // #define TFT_CS 14 // Chip select control pin
21 // #define TFT_DC 27 // Data Command
   ↪ control pin
22 // #define TFT_RST 33 // Reset pin (could
   ↪ connect to Arduino RESET pin)
23 #define TFT_BL 32 // LED back-light (
   ↪ required for M5Stack)

```

Como podemos ver, la parte final está comentada, esto es debido a que en nuestro caso, no se utilizará la aplicación táctil. Finalmente, añadimos la velocidad de frecuencia de renderizado y lectura.

```

1 #define SPI_FREQUENCY 27000000
2 #define SPI_READ_FREQUENCY 20000000

```

Una vez realizados estos cambios en los archivos de usuario de la librería *TFTeSPI*, la pantalla estaría habilitada para su uso correcto, por lo que continuaríamos con la lectura de los datos de presión, temperatura, humedad relativa, CO2 y PM2.5, por lo que creamos una función vacía llamada *readings()*.

```

1 void readings()
2 {
3     int co2, tscd40, tbme280, hscd40, hbme280,
        ↪ pbme280, pm25um= 0;
4     readPMSdata(&Serial1);
5     mySensor.readMeasurement();
6
7     pm25um = data.pm25_standard;

```

```

8
9     co2 = mySensor.getCO2();
10    tscd40 = mySensor.getTemperature();
11    tbme280 = bme1.readTemperature();
12    hscd40 = mySensor.getHumidity();
13    hbme280 = bme1.readHumidity();
14    pbme280 = bme1.readPressure() / 100.0F;
15
16    Serial.println();
17    Serial.print("CO2(ppm):");
18    Serial.print(co2);
19    Serial.print("\tPM 2.5:");
20    Serial.print(pm25um);
21    Serial.print("\tT1:");
22    Serial.print(tscd40);
23    Serial.print("\tRH1:");
24    Serial.print(hscd40);
25    Serial.print("\tP(hPa):");
26    Serial.print(pbme280);
27
28    char buffer1[20];
29    char buffer2[20];
30    char buffer3[20];
31
32    sprintf(buffer1, "\t%i °C", tbme280);
33    sprintf(buffer2, "\t%i \t%RH", hbme280);
34    sprintf(buffer3, "\t%i hPa", pbme280);
35    o = o + 1;
36    if (o>1){
37        tft.drawString(buffer1, 120, 255, 2);
38        tft.drawString(buffer2, 255, 255, 2);
39        tft.drawString(buffer3, 380, 255, 2);
40    }
41    CO2lecture = (int)co2;
42    pm25lecture = (int)pm25um;
43 }

```

Si analizamos el código, además de la impresión de datos en el Serial de la aplicación Arduino, se crean cadenas de caracteres llamados buffer1, 2 y 3, esto es debido a que es necesario para crear un String el cual se imprimirá o mostrará posteriormente en la pantalla con **tft.drawString(String, X, Y, font)**, esto se realizará con las funciones **ringmeter()**, **rainbow()** y **sinewave()** mostradas en la sección de código realizado en el Apéndice.

Una vez realizado las conexiones, el proyecto se reduce al ensamblaje de este y a la realización

del código. A continuación una muestra del producto en bruto.

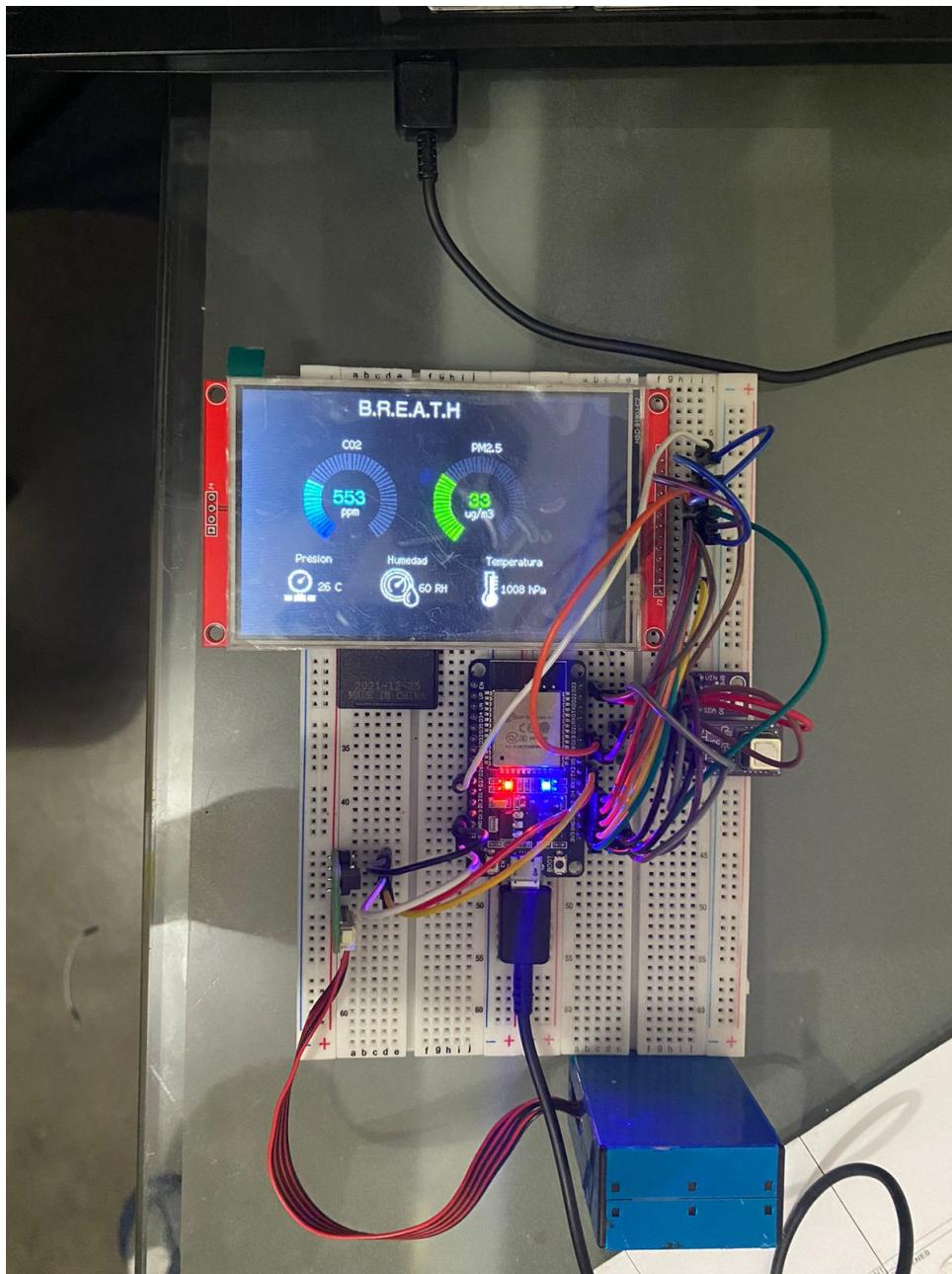


Figura 10: Proyecto de calidad de aire en protoboard

Las funciones anteriormente mencionadas, son para la realización de aplicaciones como las ruedas o arcos que muestran las cantidades de contaminación de PM2.5 y CO2 y su cambio de color a naranja-rojo a medida que este aumenta y llega a su cantidad crítica. También se muestran íconos, los cuales son imágenes en formato de matrices de datos, las cuales fueron almacenadas en la memoria flash del Esp32 a través de la función **PROGMEM** y llamadas como librerías **include iconotemperatura.h", "include iconopresion.h", "include iconohumedad.h"**, esto es debido a que resulta más eficiente en términos de recursos puesto a que un formato de imagen suele ser relativamente más grande.

## 10. Diseño mecánico

Para crear los modelos digitales tridimensionales, se usó el programa de SolidWorks versión 2022.

### 10.1. Diseño Mecánico de las Componentes Usadas

Para poder llevar a cabo la realización del proyecto se necesitó realizar el diseño tridimensional de cada componente, tomando como referencia las dimensiones obtenidas por el datasheet.

- ESP32



Figura 11: Imagen del diseño en SolidWorks del microcontrolador ESP32.

- Sensor BME280

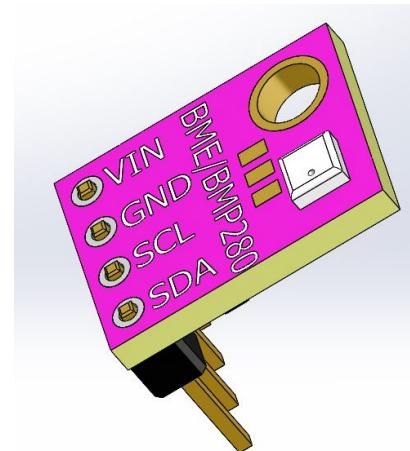


Figura 14: Imagen del diseño en SolidWorks del sensor BME280.

- TFT 4"LCD

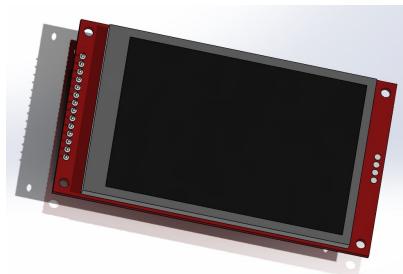


Figura 12: Imagen del diseño en SolidWorks de la pantalla LCD TFT 4'.

- Sensor PMS5003T

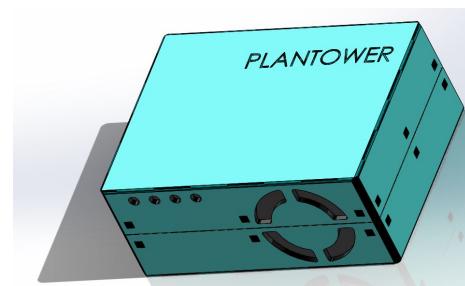


Figura 15: Imagen del diseño en SolidWorks del sensor PMS5003T.

- Sensor SCD40

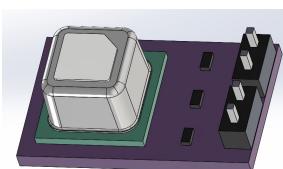


Figura 13: Imagen del diseño en SolidWorks del sensor de CO2 SCD40.

## 10.2. Consideraciones

Para el diseño del Case se tomó como referencia específica las dimensiones de la pantalla LCD TFT 4" para el ancho y alto del case Figura (16) y las dimensiones del microcontrolador ESP32 para el grosor del case (17):

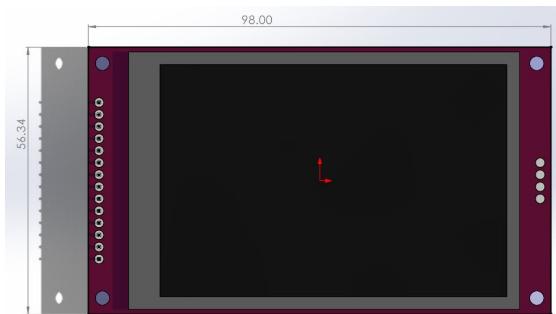


Figura 16: Imagen del diseño en SolidWorks del sensor PMS5003T.

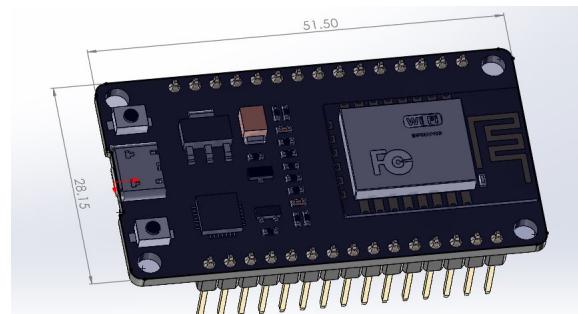


Figura 17: Imagen del diseño en SolidWorks del sensor PMS5003T.

- **Acoplamiento al aire ambiente:**

El sensor SCD4x interactúa con el entorno para medir CO<sub>2</sub>, humedad relativa y temperatura. Es esencial acoplarlo correctamente al aire ambiente para evitar tiempos de respuesta prolongados y errores de temperatura. Se recomienda colocar el sensor cerca de una abertura grande para facilitar el intercambio de aire y sellarlo para minimizar el volumen muerto, reduciendo así el tiempo de respuesta del sensor.

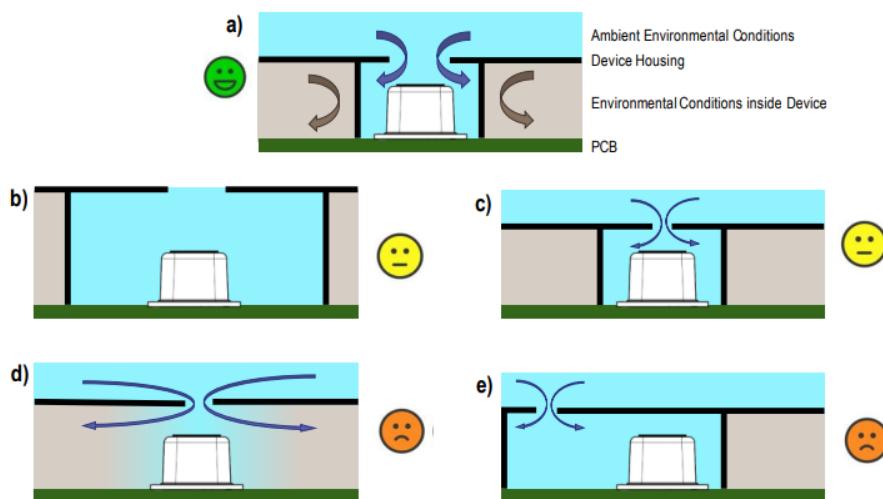


Figura 18: Acoplamiento del sensor al entorno ambiente. (a) Buena exposición al aire ambiente gracias a una abertura grande en proximidad al sensor y un volumen muerto pequeño. (b) Acoplamiento moderado al aire ambiente debido a un volumen muerto grande y (c) solo una abertura pequeña en la carcasa del dispositivo. (d) Acoplamiento deficiente ya que el sensor no está separado del aire atrapado dentro de la carcasa del dispositivo y (e) debido a una abertura pequeña lejos del sensor.

- **Desacoplamiento de fuentes de calor externas:**

Para mantener mediciones precisas, el SCD4x debe desacoplarse de fuentes de calor externas como CPU, pantalla y baterías. Este desacoplamiento es crucial, ya que las señales de humedad relativa y temperatura son esenciales para la compensación de la salida de CO<sub>2</sub> en el chip. Colocar el sensor en la parte más fría de la carcasa, lejos de las fuentes de calor, ayuda a minimizar el impacto de estas en las mediciones.

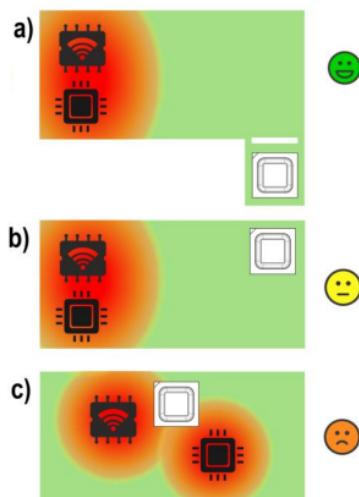


Figura 19: Acoplamiento del sensor a fuentes de calor externas (vista superior). El color verde representa la PCB del cliente; los círculos rojos indican calor disipado por componentes de auto-calentamiento. (a) Excelente desacoplamiento de fuentes de calor externas gracias a una ranura en la PCB del cliente. (c) Mal desacoplamiento de fuentes de calor externas debido a la proximidad inmediata de componentes de auto-calentamiento.

- **Aislamiento de las turbulencias del aire** El flujo de aire presente en, por ejemplo, conductos puede generar caídas de presión, contrapresión y fluctuaciones dinámicas que resultan en un aumento del ruido del sensor y una disminución de la precisión. Por lo tanto, se recomienda aislar el SCD4x del flujo de aire y las turbulencias. Esto se puede lograr colocando el sensor en un volumen separado del flujo principal de aire.

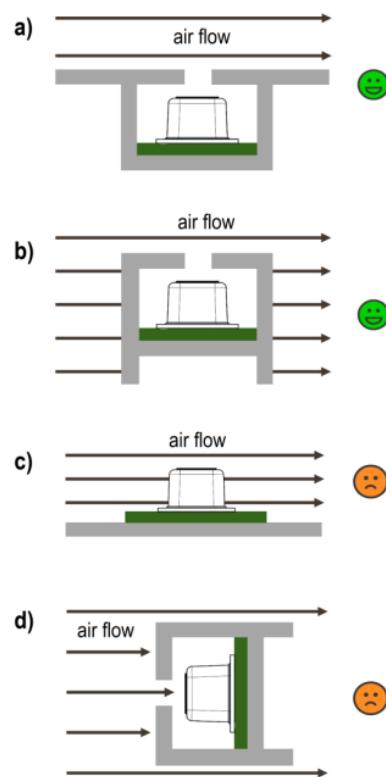


Figura 20: Aislamiento de las turbulencias del aire (vista lateral). La estructura gris representa la carcasa del dispositivo del cliente. (a-b) Buen aislamiento. (c-d) Mal aislamiento, ya que el SCD4X está expuesto directamente a las turbulencias del viento.

### 10.3. Diseño Mecánico del Case

El modelos digitales tridimensionales del case, debido a su complejidad se dividió en 3 piezas:

1. Cuerpo del Case:

Tomando como referencia las medidas de las

figuras (16) y (17) se realizó el modelos digitales tridimensionales del cuerpo del case:

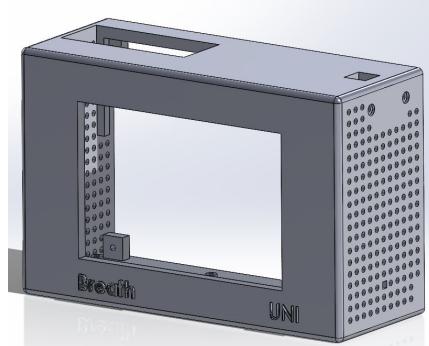


Figura 21: Imagen del diseño en SolidWorks del cuerpo del case del Módulo de Calidad de Aire de la parte delantera.

Además, tomando como referencia las medidas del microcontrolador ESP32, los sensores y la disposición en la que se deben colocar en referencia a su datasheet, se tiene:

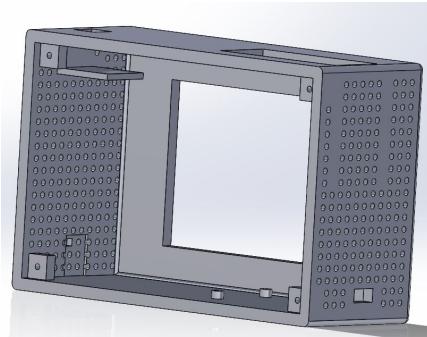


Figura 22: Imagen del diseño en SolidWorks del cuerpo del case del Módulo de Calidad de Aire de la parte interior.

## 2. Soporte para el sensor SCD40:

Tomando como referencia las recomendaciones del fabricante del sensor SCD40, se ubicó en la parte superior, además, debe

ser de fácil acceso para su mantenimiento y manipulación. Por ello, se colocó un soporte para una fácil manipulación, como se observa en la Figura (23):

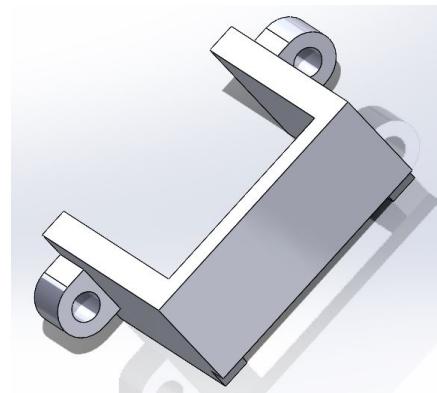


Figura 23: Imagen del diseño en SolidWorks del soporte del sensor SCD40.

## 3. Tapa del Case:

Para un correcto funcionamiento de los sensores, deben estar encerrados dentro del Case, por la cual se realizó el diseño de la tapa



Figura 24: Imagen del diseño en SolidWorks de la tapa del case del Módulo de Calidad de Aire.

## 10.4. Diseño Mecánico del Ensamblaje Final del Módulo

Usando el diseño tridimensional de cada una de las componentes y el uso de cada una de las piezas del case, se logra realizar el modelo del ensamblaje final



Figura 25: Imagen del diseño en SolidWorks del ensamblaje final del Módulo de Calidad de Aire de la parte frontal.

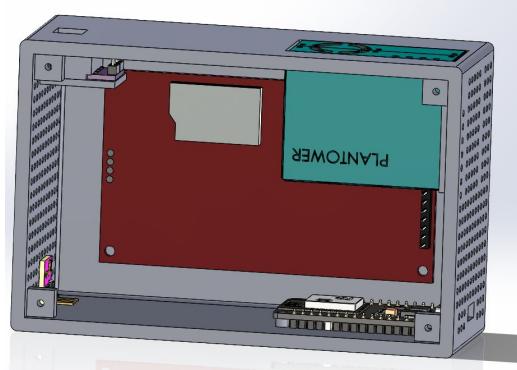


Figura 26: Imagen del diseño en SolidWorks del ensamblaje final del Módulo de Calidad de Aire de la parte interior.

## 10.5. Generación del código G

Al terminar el diseño usando el Programa SolidWorks, se guardan los archivos con la extención STL, la cual son utilizados para describir la geometría de superficies 3D mediante la representación de la malla de triángulos que compone el objeto. Mediante el uso del programa Ultimaker Cura, se exporta el archivo stl al programa:

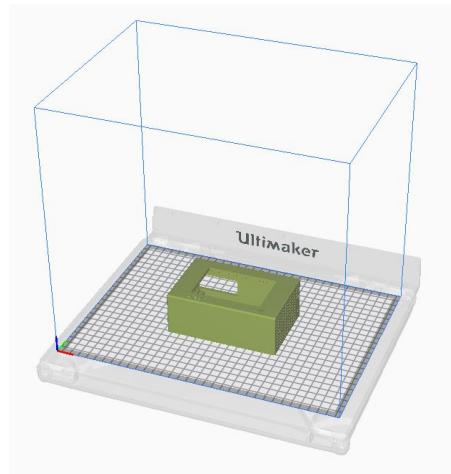


Figura 27: Imagen de la generación del código G con el programa Ultimaker Cura.

Se realiza las configuraciones necesarias para la impresión 3D, finalizando éste proceso el programa se generará un documento con códigos G, en el cual se indicará a la impresora 3D, los parámetros que debe seguir.

```
T0
M82 ;absolute extrusion mode

G92 E0
M109 S235
G280 S1
G0 Z20.001
G1 F1500 E-3
;LAYER_COUNT:339
;LAYER:0
M107
M204 S625
M205 X6 Y6
G0 F2400 X235.344 Y165.541 Z0.25
M204 S500
M205 X5 Y5
;TYPE:SKIRT
G1 F1500 E0
G1 F1200 X234.498 Y166.617 E0.02253
G1 X233.436 Y167.556 E0.04586
G1 X232.301 Y168.321 E0.06839
G1 X230.977 Y168.896 E0.09215
G1 X229.66 Y169.269 E0.11468
G1 X228.297 Y169.405 E0.13722
G1 X102.881 Y169.43 E2.20147
G1 X101.569 Y169.406 E2.22307
G1 X100.827 Y169.351 E2.23532
G1 X99.536 Y169.059 E2.2571
G1 X98.217 Y168.582 E2.28019
G1 X97.557 Y168.239 E2.29243
G1 X96.463 Y167.477 E2.31438
G1 X95.889 Y167.004 E2.32662
G1 X94.938 Y165.928 E2.35025
G1 X94.173 Y164.789 E2.37284
G1 X93.844 Y164.122 E2.38508
```

Figura 28: Imagen de una porción del código G para realizar la impresión en 3D.

## 10.6. Impresión en 3D

Con el código G generado, se guarda en una memoria y se lleva a una impresora 3D, para realizar la impresión

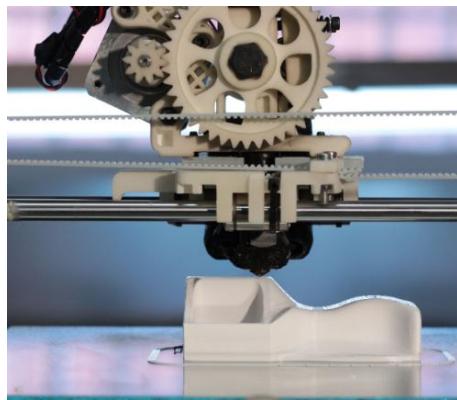


Figura 29: Imagen de la impresión en 3D.

## 11. Futuras versiones

Para un trabajo a futuro se tiene planeado el uso de varios módulos de calidad de aire que puedan hacer el testeo en tiempo real de varias zonas a la vez, logrando escalar el proyecto para un salón de mayor envergadura, todo un piso de una área de estudio, una facultad, toda la Uni, etc.

Para ello es necesario el uso de otros tipos de protocolos de comunicación:

### 11.1. ESP NOW

ESP-NOW es un protocolo de comunicación desarrollado por Espressif, que permite que múltiples dispositivos se comuniquen entre sí sin usar Wi-Fi. El protocolo es similar a la conectividad inalámbrica de bajo consumo de 2,4 GHz que suele implementarse en los ratones inalámbricos. Por lo tanto, el emparejamiento entre dispositivos es necesario antes de su comunicación. Una vez que se realiza el emparejamiento, la conexión es segura y de igual a igual.

ESP32 soporta las siguientes características:

- Comunicación unicast encriptada y sin encriptar
- Se pueden mezclar clientes con encriptación y sin encriptación
- Permite enviar hasta 250 bytes de carga útil
- Se pueden configurar callbacks para informar a la aplicación si la transmisión fue correcta
- Largo alcance, pudiendo superar los 200m en campo abierto.

Pero también tiene sus limitaciones, las cuales son:

- El número de clientes con encriptación está limitado. Esta limitación es de 10 clientes para el modo Estación, 6 como mucho en modo punto de acceso o modo mixto.
- El número total de clientes con y sin encriptación es del 20.
- Sólo se pueden enviar 250 bytes como mucho.

En palabras simples, ESP-Now es un protocolo de comunicación que nos permitirá intercambiar pequeños mensajes (hasta 250 bytes), entre nuestros microcontroladores ESP. Este protocolo es muy versátil y nos permitirá realizar conexiones en una dirección o en ambas direcciones, en diferentes configuraciones.

### 11.2. Tipos de comunicación

#### 1. Unidireccional

Este tipo de comunicación se compone de uno o varios dispositivos ESP que funcionan como maestros (emisores) y esclavos (receptores). La comunicación la iniciará el dispositivo o dispositivos maestros, y será recibida por el o los esclavos. Entre las diferentes configuraciones de las que disponemos para la configuración en una dirección, podemos distinguir las siguientes:

- Un emisor y un receptor
- Un emisor y varios receptores
- Varios emisores y un receptor

Para definir a qué receptor se va a enviar la data (o los paquetes de información) se escribe la MAC del receptor en el código del emisor.



Figura 30: Imagen de Un emisor y un receptor.



Figura 31: Imagen de Un emisor y varios receptores.



Figura 32: Imagen de Varios emisores y un receptor.

## 2. Bidireccional

Este tipo de comunicación es como el anterior, con la diferencia de que ambas placas pueden actuar como remitentes y destinatarios de los mensajes, lo cual la hace más flexible. Por ejemplo, podemos tener dos placas comunicándose entre sí simultáneamente.



Figura 33: Imagen de la Comunicación bidireccional entre ESP32.

## 3. ESP MESH

ESP-MESH permite que múltiples dispositivos (nodos) se comuniquen entre sí bajo una sola red de área local inalámbrica. Es compatible con las placas ESP32 y ESP8266. Con ESP-MESH, los nodos no necesitan conectarse a un nodo central. Los nodos son responsables de retransmitir las transmisiones de los demás. Esto permite que varios dispositivos puedan ser distribuidos en un área física grande. Los nodos pueden auto organizarse y comunicarse dinámicamente entre sí para garantizar que el paquete llegue a su destino de nodo final, en caso, por ejemplo, si se eliminara algún nodo de la red.

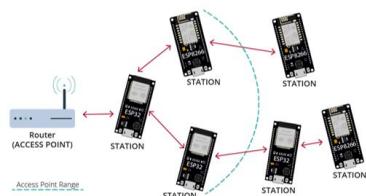


Figura 34: Imagen de la Arquitectura de red ESP MESH.

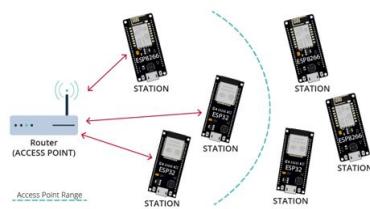


Figura 35: Imagen de la Arquitectura de red WIFI tradicional.

## 12. Conclusiones

### 12.1. Generales

1. Se desarrolló un Módulo capaz de cuantificar la Calidad de Aire presente en los lugares con gran concurrencia de personas y en ambientes cerrados. A la vez que se implementó este Módulo para alertar según el nivel de CO<sub>2</sub> en el ambiente y el envío de sus datos en tiempo real hasta una base de datos en internet () para que posteriormente se pueda acceder a ellos desde cualquier lugar.

### 12.2. Específicos

1. Así mismo, se logró cuantificar la Calidad de Aire a partir del nivel de CO<sub>2</sub> emitido producto de la respiración humana midiendo los niveles base de CO<sub>2</sub> de los ambientes durante toda la noche cuando se encuentran completamente ventilados y sin presencia alguna de personas.
2. Mediante la revisión de estudios recientes sobre la relación directa del CO<sub>2</sub> acumulado en recintos cerrados y la carga viral que estos pueden llegar a albergar, establecer los niveles de alerta para nuestros Módulos de Calidad de Aire.
3. Establecer avisos y alertas que realice el Módulo de Calidad de Aire cuando se superen los niveles de CO<sub>2</sub> recomendados para tomar acción y garantizar que el entorno dónde está el Módulo sea saludable.
4. Se diseñó e imprimió una carcasa 3D.

## Referencias

- [1] Yevgen Nazarenko, D. P. . P. A. A., “Estándares de calidad del aire para la concentración de material particulado 2.5, análisis descriptivo global,” 2020, <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7856362/pdf/BLT.19.245704.pdf/>.
- [2] de Salud, M., “Resolución ministerial 675-2022,” 2022, <https://www.gob.pe/institucion/minsa/normas-legales/3427060-675-2022-minsa>.
- [3] Hernán Paz Penagos, José Noe Poveda, A. M., “Adquisición de datos para un sistema de medición de la calidad del aire basado en iot,” 2019,

- [4] Bulot, F. M., Ossont, S. J., Morris, A. K., Basford, P. J., Easton, N. H., Mitchell, H. L., Foster, G. L., Cox, S. J., y Loxham, M., “Characterisation and calibration of low-cost pm sensors at high temporal resolution to reference-grade performance,” *Heliyon*, vol. 9, no. 5, p. e15943, 2023, doi:<https://doi.org/10.1016/j.heliyon.2023.e15943>.
- [5] García Navarrete, G. y Rico Soto, K. G., “Sensores de bajo costo para el monitoreo de calidad del aire,” *EPISTEMUS*, vol. 13, p. 30–37, 2020, doi:[10.36790/epistemus.v13i27.108](https://doi.org/10.36790/epistemus.v13i27.108).
- [6] Sitronix, 4.0 inch SPI Module MSP4020MSP4021 User Manual, 2014.
- [7] BOSCH, Combined humidity and pressure sensor BME280, 2018.
- [8] PLANTOWER, Digital universal particle concentration sensor, PMS5003 series data manual, 2016.
- [9] Sensirion, SCD4x Breaking the size barrier in CO<sub>2</sub> sensing, 2023.

## Anexo A. Código realizado

```

1 #include "iconotemperatura.h"                                ↪ library with default width and height
2 #include "iconopresion.h"
3 #include "iconohumedad.h"
4 #include <TFT_eSPI.h> // Hardware-specific
    ↪ library
5 #include <SPI.h>
6 #include <Wire.h>
7 #include <Adafruit_Sensor.h>
8 #include <Adafruit_BME280.h>
9 #include "SparkFun_SCD4x_Arduino_Library.h"
    ↪ "
10
11 int o = 0;
12
13 #define RXD2 16 // To sensor TXD
14 #define TXD2 17 // To sensor RXD
15
16 #define RED2RED 0
17 #define GREEN2GREEN 1
18 #define BLUE2BLUE 2
19 #define BLUE2RED 3
20 #define GREEN2RED 4
21 #define RED2GREEN 5
22
23 #define TFT_GREY 0x2104 // Dark grey 16 bit
    ↪ colour
24
25 SCD4x mySensor;
26 Adafruit_BME280 bme1;
27
28 TFT_eSPI tft = TFT_eSPI(); // Invoke custom
29
30
31 int CO2lecture = 0;
32 int pm25lecture = 0;
33
34 int reading = 0; // Value to be displayed
35 int d = 0;        // Variable used for the sinewave
    ↪ test waveform
36 boolean range_error = 0;
37 int8_t ramp = 1;
38
39 void setup(void)
40 {
41   Serial.begin(9600);
42
43   Serial1.begin(9600, SERIAL_8N1, RXD2,
    ↪ TXD2);
44   Wire.begin(SDA, SCL, 100000);
45   bme1.begin(0x76);
46   mySensor.begin(0x62);
47
48   tft.begin();
49   tft.setRotation(1);
50   tft.fillScreen(TFT_BLACK);
51
52   tft.setTextColor(TFT_WHITE, TFT_BLACK)
    ↪ ;
53   tft.drawString("Breath", 200, 10, 4);
54   //tft.drawString("Weather", 350, 10, 2);
55
56   tft.drawString("CO2", 130, 60, 2);
57   tft.drawString("PM2.5", 300, 60, 2);
58

```

```

59   tft.drawString("Temperatura", 65, 210, 2);
60   tft.drawString("Humedad", 190, 210, 2);
61   tft.drawString("Presion", 320, 210, 2);
62   tft.pushImage(50,230,50,50,iconotemperatura);
63   tft.pushImage(180,230,50,50,iconohumedad);
64   tft.pushImage(300,230,50,50,iconopresion);
65
66   delay(5000);
67 }
68
69 struct pms5003data {
70   uint16_t framelen;
71   uint16_t pm10_standard, pm25_standard,
72     ↪ pm100_standard;
73   uint16_t pm10_env, pm25_env, pm100_env;
74   uint16_t particles_03um, particles_05um,
75     ↪ particles_10um, particles_25um,
76     ↪ particles_50um, particles_100um;
77   uint16_t unused;
78   uint16_t checksum;
79 };
80
81 void loop()
82 {
83   readings();
84   // Test with a slowly changing value from a
85   // ↪ Sine function
86   d += 4;
87   if (d >= 360)
88     d = 0;
89
90   // Set the the position, gap between meters,
91   // ↪ and inner radius of the meters
92   int xpos = 0, ypos = 5, gap = 4, radius = 40;
93
94   //reading = 800 + 150 * sineWave(d + 90);
95   xpos = gap + ringMeter(CO2lecture, 0, 2000,
96     ↪ xpos, ypos, radius, "ppm", BLUE2RED);
97     ↪ // Draw analogue meter
98
99   //reading = 15 + 15 * sineWave(d + 150);
100  xpos = gap + ringMeter(pm25lecture, 0, 100,
101    ↪ xpos, ypos, radius, "ug/m3",
102    ↪ GREEN2GREEN); // Draw analogue
103   // ↪ meter
104
105   // ##### Draw the meter on the screen, returns x
106   // ↪ coord of righthand side
107   int ringMeter(int value, int vmin, int vmax, int x,
108     ↪ int y, int r, const char *units, byte
109     ↪ scheme)
110   {
111     // Minimum value of r is about 52 before value
112     // ↪ text intrudes on ring
113     // drawing the text first is an option
114
115     x += r;
116     y += r; // Calculate coords of centre of ring
117
118     int w = r / 3; // Width of outer ring is 1/4 of
119     // ↪ radius
120
121     int angle = 150; // Half the sweep angle of
122     // ↪ meter (300 degrees)
123
124     int v = map(value, vmin, vmax, -angle, angle);
125     // ↪ // Map the value to an angle v
126
127     byte seg = 3; // Segments are 3 degrees wide =
128     // ↪ 100 segments for 300 degrees
129     byte inc = 6; // Draw segments every 3 degrees
130     // ↪ , increase to 6 for segmented ring
131
132     // Variable to save "value" text colour from
133     // ↪ scheme and set default
134     int colour = TFT_BLUE;
135
136     // Draw colour blocks every inc degrees
137     for (int i = -angle + inc / 2; i < angle - inc /
138       ↪ 2; i += inc)
139     {
140       // Calculate pair of coordinates for segment
141       // ↪ start
142       float sx = cos((i - 90) * 0.0174532925);
143       float sy = sin((i - 90) * 0.0174532925);
144       uint16_t x0 = sx * (r - w) + x;
145       uint16_t y0 = sy * (r - w) + y;
146       uint16_t x1 = sx * r + x;
147       uint16_t y1 = sy * r + y;
148
149       // Calculate pair of coordinates for segment
150       // ↪ end
151       float sx2 = cos((i + seg - 90) * 0.0174532925)
152       // ↪ ;
153       float sy2 = sin((i + seg - 90) * 0.0174532925)
154       // ↪ ;
155       int x2 = sx2 * (r - w) + x;
156
157     }
158
159   }
160
161   delay(5000);
162 }
```

```

142     int y2 = sy2 * (r - w) + y;
143     int x3 = sx2 * r + x;
144     int y3 = sy2 * r + y;
145
146     if (i < v)
147     { // Fill in coloured segments with 2 triangles
148         switch (scheme)
149         {
150             case 0:
151                 colour = TFT_RED;
152                 break; // Fixed colour
153             case 1:
154                 colour = TFT_GREEN;
155                 break; // Fixed colour
156             case 2:
157                 colour = TFT_BLUE;
158                 break; // Fixed colour
159             case 3:
160                 colour = rainbow(map(i, -angle, angle, 0,
161                                     ↪ 127));
161                 break; // Full spectrum blue to red
162             case 4:
163                 colour = rainbow(map(i, -angle, angle, 70,
164                                     ↪ 127));
164                 break; // Green to red (high temperature
165                                     ↪ etc)
165             case 5:
166                 colour = rainbow(map(i, -angle, angle,
167                                     ↪ 127, 63));
167                 break; // Red to green (low battery etc)
168             default:
169                 colour = TFT_BLUE;
170                 break; // Fixed colour
171         }
172         tft.fillTriangle(x0, y0, x1, y1, x2, y2, colour);
173         tft.fillTriangle(x1, y1, x2, y2, x3, y3, colour);
174         //text_colour = colour; // Save the last
175                                     ↪ colour drawn
175     }
176     else // Fill in blank segments
177     {
178         tft.fillTriangle(x0, y0, x1, y1, x2, y2,
179                         ↪ TFT_GREY);
180         tft.fillTriangle(x1, y1, x2, y2, x3, y3,
181                         ↪ TFT_GREY);
181     }
182     // Convert value to a string
183     char buf[10];
184     byte len = 3;
185     if (value > 999)
186         len = 5;+
187     dtostrf(value, len, 0, buf);
188     buf[len] = ' ';
189     buf[len + 1] = 0; // Add blanking space and
190                                     ↪ terminator, helps to centre text too!
191     // Set the text colour to default
192     tft.setTextSize(1);
193     tft.setTextColor(TFT_WHITE, TFT_BLACK)
194                                     ↪ ;
195     // Uncomment next line to set the text colour
196                                     ↪ to the last segment value!
197     tft.setTextColor(colour, TFT_BLACK);
198     tft.setTextDatum(MC_DATUM);
199     // Print value, if the meter is large then use big
200                                     ↪ font 8, othewise use 4
201     if (r > 84)
202     {
203         tft.setTextPadding(55 * 3); // Allow for 3
204                                     ↪ digits each 55 pixels wide
205         tft.drawString(buf, x, y, 8); // Value in middle
206     }
207     else
208     {
209         tft.setTextPadding(3 * 14); // Allow for 3
210                                     ↪ digits each 14 pixels wide
211         tft.drawString(buf, x, y, 4); // Value in middle
212     }
213     tft.setTextSize(1);
214     tft.setTextPadding(0);
215     // Print units, if the meter is large then use big
216                                     ↪ font 4, othewise use 2
217     tft.setTextColor(TFT_WHITE, TFT_BLACK)
218                                     ↪ ;
219     if (r > 84)
220         tft.drawString(units, x, y + 60, 4); // Units
221                                     ↪ display
222     else
223         tft.drawString(units, x, y + 15, 2); // Units
224                                     ↪ display
225     // Calculate and return right hand side x
226                                     ↪ coordinate
227     return x + r;
228
229 unsigned int rainbow(byte value)
230 {
231     // Value is expected to be in range 0–127
232     // The value is converted to a spectrum colour
233                                     ↪ from 0 = blue through to 127 = red
234
235     byte red = 0; // Red is the top 5 bits of a 16
236                                     ↪ bit colour value
237     byte green = 0; // Green is the middle 6 bits
238     byte blue = 0; // Blue is the bottom 5 bits

```

```

230
231     byte quadrant = value / 32;
232
233     if (quadrant == 0)
234     {
235         blue = 31;
236         green = 2 * (value % 32);
237         red = 0;
238     }
239     if (quadrant == 1)
240     {
241         blue = 31 - (value % 32);
242         green = 63;
243         red = 0;
244     }
245     if (quadrant == 2)
246     {
247         blue = 0;
248         green = 63;
249         red = value % 32;
250     }
251     if (quadrant == 3)
252     {
253         blue = 0;
254         green = 63 - 2 * (value % 32);
255         red = 31;
256     }
257     return (red << 11) + (green << 5) + blue;
258 }
259
260 // #####
261 // Return a value in range -1 to +1 for a given
262 //      → phase angle in degrees
263 // #####
264 float sineWave(int phase)
265 {
266     return sin(phase * 0.0174532925);
267 }
268 //=====
269 // This is the function to draw the icon stored as
270 //      → an array in program memory (FLASH)
271 //=====
272 // To speed up rendering we use a 64 pixel buffer
273 #define BUFF_SIZE 64
274
275 // Draw array "icon" of defined width and height
276 //      → at coordinate x,y
277 // Maximum icon size is 255x255 pixels to avoid
278 //      → integer overflow
279
280 void drawIcon(const unsigned short *icon,
281               → int16_t x, int16_t y, int8_t width,
282               → int8_t height)
283 {
284     uint16_t pix_buffer[BUFF_SIZE]; // Pixel
285 //      → buffer (16 bits per pixel)
286     tft.startWrite();
287
288     // Set up a window the right size to stream
289     //      → pixels into
290     tft.setAddrWindow(x, y, width, height);
291
292     // Work out the number whole buffers to send
293     uint16_t nb = ((uint16_t)height * width) /
294 //      → BUFF_SIZE;
295
296     // Fill and send "nb" buffers to TFT
297     for (int i = 0; i < nb; i++)
298     {
299         for (int j = 0; j < BUFF_SIZE; j++)
300         {
301             pix_buffer[j] = pgm_read_word(&icon[i *
302 //      → BUFF_SIZE + j]);
303         }
304         tft.pushColors(pix_buffer, BUFF_SIZE);
305     }
306
307     // Work out number of pixels not yet sent
308     uint16_t np = ((uint16_t)height * width) %
309 //      → BUFF_SIZE;
310
311     // Send any partial buffer left over
312     if (np)
313     {
314         for (int i = 0; i < np; i++)
315             pix_buffer[i] = pgm_read_word(&icon[nb *
316 //      → BUFF_SIZE + i]);
317         tft.pushColors(pix_buffer, np);
318     }
319
320     tft.endWrite();
321 }
322
323 void readings()
324 {
325     int co2, tscd40, tbme280, hscd40, hbme280,
326 //      → pbme280, pm25um= 0;
327     readPMSSdata(&Serial1);
328     mySensor.readMeasurement();
329
330     pm25um = data.pm25_standard;
331
332     co2 = mySensor.getCO2();
333     tscd40 = mySensor.getTemperature();
334 }
```

```

325 tbme280 = bme1.readTemperature();
326 hscd40 = mySensor.getHumidity();
327 hbme280 = bme1.readHumidity();
328 pbme280 = bme1.readPressure() / 100.0F;
329
330 // if(pm25um >100){
331 //   pm25um = 30;
332 // }
333 // else if(isnan(pm25um))
334 // {
335 //   pm25um = 31;
336 // }
337
338 //if(isnan(pbme280)){
339 // pbme280 = 1006;
340 //}
341 Serial.println();
342 Serial.print("CO2(ppm):");
343 Serial.print(co2);
344 Serial.print("\tPM 2.5:");
345 Serial.print(pm25um);
346
347 Serial.print("\tT1:");
348 Serial.print(tbme280);
349 Serial.print("\tT2:");
350 Serial.print(tscd40);
351
352 Serial.print("\tRH1:");
353 Serial.print(hbme280);
354 Serial.print("\tRH2:");
355 Serial.print(hscd40);
356
357 Serial.print("\tP(hPa):");
358 Serial.print(pbme280);
359
360 char buffer1[20];
361 char buffer2[20];
362 char buffer3[20];
363
364 sprintf(buffer1, "%i °C", tbme280);
365 sprintf(buffer2, "%i %RH", hbme280);
366 sprintf(buffer3, "%i hPa", pbme280);
367 o = o + 1;
368 if (o>1){
369 tft.drawString(buffer1, 120, 255, 2);
370 tft.drawString(buffer2, 255, 255, 2);
371 tft.drawString(buffer3, 380, 255, 2);
372 }
373 CO2lecture = (int)co2;
374 pm25lecture = (int)pm25um;
375 }
376
377 // PMSSSSSSS
378 boolean readPMSdata(Stream *s) {
379     if (! s->available()) {
380         return false;
381     }
382
383     // Read a byte at a time until we get to the
384     // → special '0x42' start-byte
385     if (s->peek() != 0x42) {
386         s->read();
387         return false;
388     }
389
390     // Now read all 32 bytes
391     if (s->available() < 32) {
392         return false;
393     }
394
395     uint8_t buffer[32];
396     uint16_t sum = 0;
397     s->readBytes(buffer, 32);
398
399     // get checksum ready
400     for (uint8_t i = 0; i < 30; i++) {
401         sum += buffer[i];
402     }
403
404     /* debugging
405      for (uint8_t i=2; i<32; i++) {
406          Serial.print("0x");
407          Serial.print(buffer[i], HEX);
408          → Serial.print(", ");
409      }
410      Serial.println();
411 */
412
413     // The data comes in endian'd, this solves it so
414     // → it works on all platforms
415     uint16_t buffer_u16[15];
416     for (uint8_t i = 0; i < 15; i++) {
417         buffer_u16[i] = buffer[2 + i * 2 + 1];
418         buffer_u16[i] += (buffer[2 + i * 2] << 8);
419     }
420
421     // put it into a nice struct :
422     memcpy((void *)&data, (void *)buffer_u16, 30)
423     → ;
424
425     if (sum != data.checksum) {
426         Serial.println("Checksum failure");
427         return false;
428     }
429
430     // success!
431     return true;
432 }
```