UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Diseño, construcción y puesta en marcha de un sistema electrónico para el monitoreo de Compuestos Orgánicos Volátiles Totales (COVsT) y parámetros meteorológicos

TESIS

Que para obtener el título de

Ingeniero Eléctrico Electrónico

P R E S E N T A N

Eduardo Emiliano Santamaría de la Rosa
Oscar Ivan Rangel Reyes

DIRECTORA DE TESIS

Dra. María del Pilar Corona Lira

Ciudad Universitaria, Cd. Mx., 2025

Dedicatoria ...

Oscar Rangel

Dedico con amor y cariño esta tesis a mi papá y a mi mamá, por su amor incondicional y por ser un ejemplo de trabajo, honestidad y responsabilidad; a mi hermano, por su apoyo constante, su alegría y su motivación en los momentos necesarios; a mis amigos, por estar ahí cuando más lo necesité; y, por supuesto, a mis michis, por su compañía silenciosa pero constante durante este arduo proceso.

Emiliano Santamaría

Quiero dedicar esta tesis a mis padres, por el apoyo y amor incondicional que me han brindado durante toda mi vida y, en especial, durante este proceso; a mi hermana, por ser la mejor compañera de vida que pude pedir y ser un ejemplo constante; y a mi novia, amigos y primos por apoyarme en todos los momentos que lo necesité y ser una fuente inagotable de inspiración y confianza.

Agradecimientos

Agradecemos profundamente a todas las personas que hicieron posible la realización de esta tesis. En primer lugar, a nuestra tutora, la Dra. María del Pilar Corona, por su guía y constante apoyo durante el desarrollo del proyecto; así como al Ing. Manuel García, al Dr. Omar Amador y al Mtro. Darío Reyes, por el acompañamiento brindado tanto en las etapas iniciales como durante la campaña realizada en el sur del Valle del Mezquital. A Erika González, Mariana Ramos y Francisco Bernal por el apoyo técnico en la evaluación y operación de los equipos mediante el análisis químico de los compuestos orgánicos volátiles. En especial, agradecemos a Damián Carmona por su valiosa ayuda en el desarrollo inicial del proyecto.

Al Dr. Omar Amador Muñoz por la coordinación y ejecución del proyecto PRONAII No. 2024-022, así como por invitarnos a formar parte de su grupo de trabajo.

Agradecemos también al Instituto de Ciencias de la Atmósfera y Cambio Climático (ICAyCC) por las facilidades otorgadas para el diseño y construcción de los equipos de muestreos. Así como al Laboratorio Abierto a la Innovación y al Fortalecimiento de la Infraestructura en México (LemAT), por proporcionar los recursos necesarios para llevar a cabo este proyecto.

Este proyecto fue financiado por la Secretaría de Ciencia, Humanidades, Tecnología e Innovación (SECIHTI) con el proyecto: Caracterización y Diagnóstico de Contaminantes Orgánicos Atmosféricos Tóxicos No Regulados en las Regiones de Emergencia Sanitaria y Ambiental (RESA) Tolteca y Norte del Istmo de Tehuantepec. En la modalidad de Propuestas de Proyectos Nacionales de Investigación e Incidencia (PRONAII) No. 2024-022, así como por la Comisión Ambiental de la Megalópolis (CAME) de la SEMARNAT con el fideicomiso FIDAM 1490. Especialmente agradecemos las becas otorgadas por la SECIHTI.

De igual forma, extendemos nuestro agradecimiento a la comunidad del sur del Valle del Mezquital por su colaboración y apoyo durante la campaña de medición. Su disposición y apertura fueron fundamentales para el desarrollo del trabajo de campo. En especial a Aideé Mares, Francisco Ibañez, Marisol Gordillo, Yeriana León, Noe Roque, Guadalupe Sánchez, Octavio Ortíz, René Romero, Oscar Martínez y Jorge Hernández y al Dr. Refugio Choreño Gómez por las facilidades otorgadas y el apoyo en la logística y desarrollo de la campaña de muestreo.

Asimismo, expresamos nuestro reconocimiento a nuestra alma mater, la Universidad Nacional Autónoma de México (UNAM), y en particular a la Facultad de Ingeniería, por brindarnos una formación académica sólida y por ser el espacio donde se gestó este proyecto.

Finalmente, queremos agradecer a nuestras familias y amigos por su comprensión, paciencia y constante apoyo emocional a lo largo de este proceso. Su compañía y palabras de aliento fueron esenciales para seguir adelante.

Índice general

Índice de figuras

Índice de tablas

1 Introducción

Con el paso del tiempo, ha crecido el interés social por proteger tanto la salud humana como el equilibrio del entorno natural que nos rodea. En este contexto, se han identificado diversos problemas ambientales, entre ellos la presencia de compuestos microscópicos en el aire, que pueden afectar gravemente la salud tras una exposición prolongada a estos. Dichos contaminantes, conocidos como Compuestos Orgánicos Volátiles (COVs), corresponden a sustancias químicas que pueden pasar con facilidad al estado gaseoso a temperatura ambiente. En ciertas concentraciones, pueden causar efectos adversos como irritación, daños al sistema nervioso e incluso enfermedades crónicas, dependiendo del tipo de compuesto. Los principales emisores de COVs incluyen fuentes industriales, automóviles, actividades comerciales y de servicios, e incluso productos utilizados en el hogar. [1]. Si bien existen múltiples factores que elevan la concentración de estos compuestos en una zona, este proyecto se enfoca particularmente en aquellas áreas impactadas por la actividad industrial.

La identificación de contaminantes en el aire resulta especialmente compleja para las personas que no cuentan con conocimientos técnicos ni con herramientas adecuadas de monitoreo. Esta limitación impide a muchas comunidades conocer la calidad del aire que respiran y, por tanto, tomar decisiones informadas sobre su salud y entorno. El objetivo de este trabajo es desarrollar y poner a disposición una herramienta accesible, autónoma, funcional y de bajo costo que permita a las comunidades monitorear la calidad del aire que respiran cotidianamente. La información generada servirá como evidencia para conocer la presencia y concentración de contaminantes atmosféricos previamente no identificados en estas zonas, contribuyendo así a visibilizar la relación entre salud y ambiente en contextos donde las actividades productivas representan una fuente importante de emisiones contaminantes.

En este trabajo se presenta el proceso de diseño, implementación y puesta en marcha de un sistema automático de muestreo de aire activo de bajo flujo (3.2), así como la obtención de parámetros meteorológicos en la zona donde se instale. El proyecto se basa en el uso de sensores, actuadores y componentes de control de bajo costo, seleccionados por su buena relación calidad-precio, lo que permite la replicabilidad del sistema. Además, el dispositivo es autónomo, ya que no requiere conexión directa a la red eléctrica, lo que lo hace ideal para su instalación en campo y la recolección continua de datos ambientales.

2 Estado del arte

1. Importancia de la calidad del aire

La calidad del aire es un factor crítico para la salud pública y el bienestar ambiental. Desde la Revolución Industrial, el aumento de las emisiones contaminantes ha tenido un impacto significativo en la atmósfera, lo que ha impulsado la implementación de sistemas de análisis y monitoreo para evaluar y mitigar sus efectos. En las últimas décadas, los avances tecnológicos y científicos han permitido desarrollar metodologías cada vez más precisas y accesibles para medir la concentración de contaminantes en el aire. Este estado del arte tiene como objetivo revisar las tecnologías más recientes para monitorear la composición del aire, así como las causas y consecuencias de los contaminantes presentes en él.

2. Compuestos clave para medir la calidad del aire en México

En México, los compuestos y factores que se consideran para medir la calidad del aire son los siguientes:

- Dióxido de azufre (SO_2)
- Óxidos de nitrógeno (NO_x)
- Monóxido de carbono (CO)
- Ozono (O_3)
- Partículas suspendidas (PM10, PM2.5 y PM10-2.5)

Estos compuestos son medidos utilizando instrumentos de alta precisión, aprobados por la EPA (Agencia de Protección Ambiental de los Estados Unidos). Cada compuesto requiere un dispositivo específico para su medición, con un costo que oscila entre 300,000 y 800,000 pesos [2].

El proceso de medición consiste en colocar un filtro a través del cual pasa una cantidad regulada de aire proveniente del ambiente. Posteriormente, el filtro es irradiado con rayos de diferentes longitudes de onda, lo que permite comparar la cantidad de

radiación antes y después de atravesarlo. La diferencia entre la radiación inicial y final se utiliza para calcular la concentración de los compuestos presentes en la muestra de aire analizada.

De acuerdo con [3], estos dispositivos se clasifican como sensores ópticos, ya que aprovechan los cambios en las ondas electromagnéticas al interactuar con un medio para obtener mediciones. Actualmente, los sensores ópticos son una de las tecnologías más avanzadas para la detección de compuestos en el aire, debido a su capacidad de respuesta en tiempo real y la confiabilidad de sus resultados.

3. Muestreo de compuestos orgánicos volátiles (COVs)

A pesar de contar con mediciones y regulaciones para los compuestos tradicionales, diversos estudios, como [4] y [1], destacan la presencia de otros compuestos en el aire que también son perjudiciales para la salud. Estos compuestos, Compuestos Orgánicos Volátiles (COVs), no cuentan con una amplia gama de instrumentos para medir su concentración en el aire.

El método más eficaz y ampliamente utilizado para medir los COVs es la cromatografía de gases (GC). Este proceso emplea materiales adsorbentes —como Tenax, carbón activado o sílice— capaces de retener los compuestos al exponerlos a un flujo de aire, ya sea mediante muestreo pasivo (por difusión) o activo (con bomba). Posteriormente, las muestras recolectadas se analizan en un cromatógrafo de gases, que puede acoplarse a diferentes detectores, como el de ionización de flama (FID) o un espectrómetro de masas (GC–MS), lo que permite identificar y cuantificar los tipos de COVs presentes en el aire. Es importante destacar que este análisis se realiza de manera diferida en laboratorio y no en tiempo real.

De acuerdo con [5], los muestreos de COVs es pueden clasificarse en dos tipos:

3.1. Muestreo pasivo

- No requiere bombeo; la recolección se realiza por difusión natural del aire hacia el absorbente o adsorbente.
- Los tiempos de exposición varían desde 8 horas hasta 2 semanas.

La tecnología y el diseño de los muestreadores pasivos son prácticos y sencillos. Un estudio de la Universidad Rovira i Virgili, en España [6], describe el uso de este tipo de dispositivos. Aunque el artículo no incluye imágenes del muestreador, detalla que consiste en un tubo con un material adsorbente conectado a una tapa de difusión en uno de sus extremos. Todo el dispositivo estaba recubierto con una capa de acero inoxidable para resistir las condiciones climáticas. El funcionamiento del muestreador es simple: el adsorbente se expone al aire libre y comienza a captar muestras por

difusión. Sin embargo, este método tiene la desventaja de ser un proceso lento. Según el estudio, cada muestra tardó aproximadamente 14 días en recolectarse.

Figura 2.1: Ejemplo de muestreo pasivo.[1]

3.2. Muestreo activo

- Utiliza una bomba para mantener un flujo constante de aire hacia el absorbente o adsorbente.
- El tiempo de muestreo se reduce a minutos, ya que se controla el volumen de aire muestreado.

El muestreo activo emplea instrumentos especializados para el bombeo de aire. Un estudio realizado por la Universidad de Masaryk, en la República Checa [7], comparó el desempeño de tres tipos de muestreadores activos disponibles en el mercado:

- Muestreadores de alto volumen, que manejan flujos de aire entre 100 y 1700 [L/min].
- Muestreadores de bajo volumen, con flujos de aire de 1 a 16.7[L/min].
- Impactadores en cascada, que utilizan una bomba para forzar el paso del aire a través de placas de impacto y separar las partículas según su tamaño.

En la investigación se evaluaron cuatro muestreadores distintos: dos de alto volumen, un impactador en cascada y uno de bajo volumen, provenientes de tres empresas diferentes: Sven Leckel, Tisch y Digitel. Ninguna de estas compañías publica los precios de sus dispositivos en internet; sin embargo, equipos similares de marcas como Envirotechlab y PCE Instruments tienen un rango de precios que oscila entre 5,000 y 10,000 dólares.

La mayoría de estos instrumentos están diseñados principalmente para la detección de material particulado, aunque es posible adaptarlos con cartuchos de absorbente o

[1]Imagen tomada de: https://youtu.be/0uXxtunqzDI?si=Pqbzc0gyxDDB3W4j

adsorbente para recolectar compuestos orgánicos volátiles (COVs) del aire succionado por la bomba. Todos los dispositivos requieren conexión a la red eléctrica y están equipados con sensores de temperatura, humedad y flujo. Gracias al flujómetro y al controlador integrado, garantizan un flujo constante a través de los filtros y cartuchos.

Figura 2.2: Muestreador activo de bajo flujo Tisch.[2]

Los resultados del estudio indican que, aunque hubo algunas discrepancias entre los valores teóricos y los obtenidos experimentalmente, el desempeño general de los dispositivos fue satisfactorio. Además, se concluyó que la temperatura a la que se encuentra la muestra durante el proceso de medición es el factor más determinante para obtener resultados precisos.

Si bien la tecnología de los muestreadores activos está más desarrollada que la de los pasivos, no se encontraron artículos que se enfoquen exclusivamente en activos diseñados específicamente para la captura de COVs. Los dispositivos comerciales están orientados principalmente a la detección de material particulado y, como función secundaria, permiten la adaptación de un cartucho para la recolección de COVs. Además de su alto costo, los modelos de menor flujo manejan un flujo mínimo de 1 [L/min], lo que hace que estos instrumentos no sean adecuados para un muestreo con un cartucho y pruebas diseñadas para una menor masa total de aire.

4. Muestreador de bajo flujo programable por el ICAyCC

Como antecedente directo de este proyecto se encuentra el muestreador desarrollado por el Instituto de Ciencias de la Atmósfera y Cambio Climático (ICAyCC), en el área de Instrumentación Meteorológica. Este dispositivo fue diseñado para la

[2]Imagen tomada de: https://tisch-env.com/low-volume-air-sampler/

recolección activa de muestras de aire.

Cuenta con alimentación autónoma mediante una batería de 12[V] y 5[Ah], un controlador de carga y una celda solar. Además, incorpora un ventilador para enfriar las bombas de vacío, un contador de horas, un interruptor para encendido y arranque manual, y un programador con pantalla.

El equipo permite programar pruebas en fechas y horarios específicos, con una duración de hasta 12 horas, y ofrece la posibilidad de configurar hasta cinco eventos en horas distintas. No obstante, para realizar muestreos diurnos y nocturnos, es necesario cambiar manualmente los cartuchos. El flujo de aire es generado por una minibomba de vacío, cuyo ajuste se realiza de forma manual mediante una perilla, y se monitorea con un sensor de flujo de burbuja.

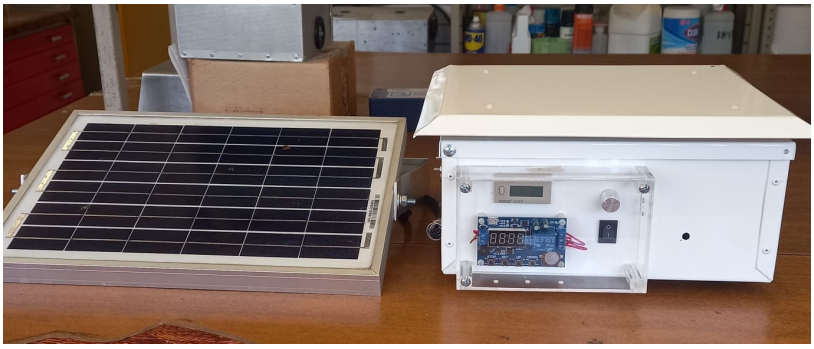


Figura 2.3: Muestreador de bajo flujo programable.

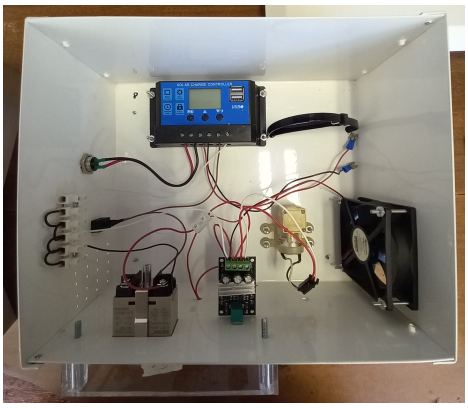


Figura 2.4: Muestreador de bajo flujo programable vista desde adentro.

3 Objetivos

1. Objetivo general

Desarrollar un sistema automático de monitoreo ambiental y colecta de Compuestos Orgánicos Volátiles Totales (COVsT) mediante sensores y actuadores de bajo costo, garantizando exactitud, precisión, confiabilidad y facilidad de uso.

2. Objetivos específicos

- Diseñar una interfaz intuitiva que facilite la configuración, operación y visualización de los datos del sistema.
- Integrar sensores de bajo costo que aseguren mediciones exactas de variables meteorológicas, equilibrando costo y exactitud.
- Garantizar la confiabilidad, exactitud y precisión del flujo controlado para una adecuada interpretación de los datos.
- Implementar un sistema eficiente para el almacenamiento y gestión de datos, facilitando su análisis a largo plazo.
- Desarrollar un sistema de alimentación autónomo que asegure la operatividad del sistema en ubicaciones remotas.

4 Metodología

1. Descripción del proyecto

El proyecto comenzó como una mejora de un muestreador autónomo ya existente, diseñado inicialmente para bombear un flujo constante de aire a un cartucho receptor durante varias horas. Cuando el tiempo de muestreo deseado se completaba, el cartucho debía ser reemplazado manualmente y la bomba recalibrada al flujo adecuado. Este sistema requería un monitoreo constante por parte del usuario, ya que, con el tiempo, el cartucho podría obstruirse y reducir el flujo de aire. Al ocurrir esto, el usuario tenía que ajustar manualmente el voltaje de la bomba para restablecer el flujo deseado. Además, no era posible verificar el flujo promedio durante toda la prueba.

Estas limitaciones causaban problemáticas tanto para los usuarios como para el análisis de las muestras. No era posible para los usuarios monitorear constantemente el proceso, y al analizar las muestras era incierto si el flujo de aire se había mantenido constante y en el nivel deseado durante toda la prueba. Resolver estas dificultades y añadir nuevas funciones constituye el objetivo principal de este proyecto.

Por lo tanto, el nuevo sistema debe ser capaz de regular automáticamente el flujo de aire hacia el cartucho, monitorear ese flujo durante toda la prueba y almacenar los datos correspondientes. También debe automatizar el cambio de cartuchos para permitir pruebas continuas de 24 horas sin intervención del usuario. Finalmente, la utilidad del proyecto se incrementará considerablemente si se integrara la capacidad de medir en tiempo real datos meteorológicos y de calidad del aire. Esto permitirá comparar los datos obtenidos mediante el muestreo activo con los datos generados por los sensores en tiempo real.

2. Descripción del sistema

El muestreador ha sido diseñado para ofrecer una experiencia intuitiva y de fácil uso, garantizando que cualquier usuario pueda operar sin complicaciones, incluso en lo referente a su mantenimiento. Su estructura modular permite el reemplazo sencillo de componentes susceptibles a daño, asegurando así una mayor durabilidad y facilidad de reparación.

El sistema cuenta con una interfaz visual accesible que permite su operación mediante un máximo de dos actuadores: una pantalla y un *joystick* como opción principal, o a través de un celular conectado por *Bluetooth*. Su diseño ligero y compacto facilita tanto el transporte como la instalación en diversas ubicaciones, adaptándose a distintos escenarios de monitoreo ambiental.

Es completamente autónomo gracias a su batería recargable y su panel solar, lo que garantiza un funcionamiento continuo, incluso en zonas remotas. Además, permite la visualización de algunas variables en tiempo real a través de internet, brindando al usuario la posibilidad de monitorear el dispositivo de manera remota. En caso de no contar con conexión a internet, el sistema genera automáticamente archivos diarios que almacenan los datos capturados: uno con registros cada 30 segundos y otro que calcula promedios cada 15 minutos.

Por último, se ha desarrollado un manual de usuario detallado que proporciona información clara sobre su funcionamiento y programación, permitiendo a los usuarios resolver cualquier duda sin necesidad de asistencia directa, fomentando así una experiencia completamente autónoma.

3. Contexto del usuario

Los usuarios elegidos cuentan con ciertas características que los vuelven un sector de interés para el proyecto. La primera característica es que son voluntarios, que no sólo brindan un espacio en sus hogares para colocar el muestreador, sino que también ofrecen su apoyo para dar mantenimiento y cooperar con la programación de pruebas del mismo. La segunda característica es que habitan en una zona donde existe una actividad industrial intensa, afectando la calidad del aire que respiran, dando como resultado un lugar donde es necesario muestrear para conocer los riesgos de salud a los que están sometidos los locales.

5 Desarrollo del trabajo

1. Diagrama de bloques del sistema

Para observar de forma resumida y gráfica las distintas partes que componen el sistema y cómo interactúan entre sí, se creó un diagrama de bloques 5.1 sencillo que agrupa los diferentes componentes que constituyen las partes esenciales del sistema.

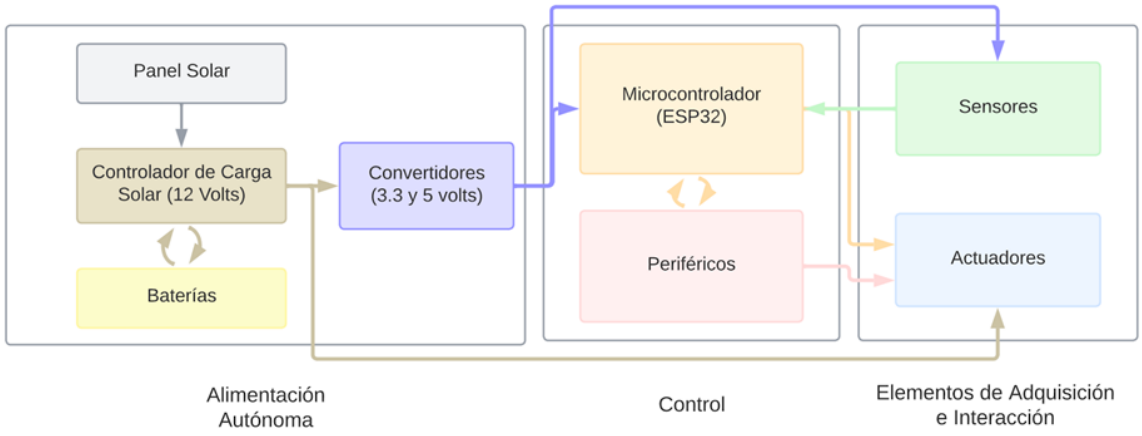


Figura 5.1: Diagrama de bloques del proyecto.

El sistema se compone de tres partes fundamentales:

1.1. Alimentación autónoma

Esta sección incluye todos los componentes encargados de suministrar y gestionar la energía del sistema. Dado que se trata de un sistema autónomo, es necesario contar con una fuente de generación de energía eléctrica. Para ello, el panel solar convierte la energía solar en electricidad, la cual es almacenada en las baterías.

Un controlador de carga supervisa el estado de las baterías y determina si la energía generada por el panel solar debe ser suministrada directamente al sistema o utilizada para recargar las baterías. Además, gestiona el momento en que las baterías deben alimentar el sistema.

Como estos elementos operan a 12 V, es necesario utilizar reguladores de voltaje que reduzcan la tensión a 5 V y 3.3 V, valores adecuados para la mayoría de los componentes electrónicos comerciales utilizados.

1.2. Control

En esta sección se incluyen todos los componentes responsables de recibir información de los sensores, procesarla y enviar comandos para que los actuadores ejecuten las acciones correspondientes. El microcontrolador (ESP32) es el núcleo de esta parte del sistema, actuando como el "cerebro" encargado de gestionar la información y coordinar el funcionamiento de los diferentes componentes.

También se consideran en esta sección los dispositivos que amplían las capacidades del microcontrolador, como convertidores de señales analógicas a digitales, el reloj en tiempo real, inversores, entre otros. A estos componentes los definimos como periféricos, ya que tienen la capacidad de recibir y enviar información desde o hacia el microcontrolador.

1.3. Elementos de adquisición e interacción

Esta sección incluye todos los elementos que, de manera física, desempeñan una función dentro del sistema. Los sensores son responsables de captar datos del entorno y transmitirlos a la unidad de control. En este caso, es necesario contar con sensores atmosféricos y un flujómetro que indique el caudal de aire extraído por el compresor.

Por otro lado, los actuadores reciben comandos desde la unidad de control y ejecutan acciones que impactan el entorno físico. En esta categoría se incluyen componentes como el compresor, las electroválvulas, los ventiladores, entre otros.

2. Especificaciones

2.1. Variables atmosféricas de interés

Además de diseñar e implementar un muestreador de flujo constante, se buscó ampliar las funciones del sistema para recolectar datos atmosféricos mediante el uso de sensores de bajo costo. La primera variable de interés fue medir la temperatura y la humedad de la ubicación, con el propósito de registrar las condiciones en las que se realizó la muestra. También se consideró necesario medir la temperatura a la que se encontraban directamente los cartuchos, ya que como se menciona en la sección 3.2 la temperatura es determinante para obtener resultados precisos.

Por otro lado, resultó fundamental determinar la dirección y la velocidad del viento, para poder asociar una magnitud y dirección específicas a los contaminantes

presentes en el aire recolectado por el muestreador. Adicionalmente, es de interés probar el funcionamiento de sensores de contaminantes en tiempo real, con el objetivo de comparar su comportamiento y utilizarlos como indicadores inmediatos del estado del aire. Estos datos debían ser recolectados en intervalos de tiempo cortos y posteriormente, promediados en intervalos más largos, principalmente para el monitoreo de estos.

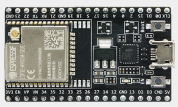
2.2. Selección de componentes

Con estas variables atmosféricas de interés, se buscaron componentes que cumplieran con el requisito de ser de bajo costo y ofrecer una fiabilidad aceptable. Un factor importante fue la necesidad de utilizar protocolos de comunicación que requirieran el menor número de cables posible, dada la diversidad de sensores involucrados. Por ello, se optó por sensores y actuadores que emplearan protocolos como Inter-Integrated Circuit (I2C), Serial Peripheral Interface (SPI), o que contaran con una interfaz digital simple de entrada o salida.

El protocolo I2C, que utiliza solo dos cables para la comunicación (más los de alimentación), es especialmente adecuado para sensores que no requieren altas velocidades de transmisión y favorece la reducción del cableado. Por su parte, el protocolo SPI usa cuatro cables para la comunicación y ofrece una mayor velocidad cuando se necesita. Con estos criterios, se procedió a buscar y cotizar sensores, actuadores y demás componentes necesarios para asegurar el correcto funcionamiento del sistema, buscando siempre la mejor relación calidad-precio para el proyecto.

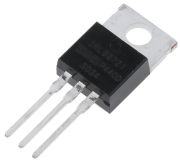
2.2.1. Control

Tabla 5.1: Componentes del sistema de control

Componente	Cantidad	Precio (USD)	Entrada [V]	Salida [V]	Protocolo	Justificaciones
ESP–WROOM–32 (Micro) [1] 	1	13	5 V	–	–	Se eligió la *ESP-WROOM-32* por su tamaño compacto, bajo consumo y buena capacidad de memoria, que permite ejecutar múltiples tareas y controlar varios periféricos. Cuenta con interfaces como I2C, SPI y Universal Asynchronous Receiver-Transmitter (UART), además de conectividad *Wi-Fi* y *Bluetooth* integradas, facilitando la comunicación inalámbrica sin módulos externos.

Continúa en la siguiente página

[1]Imagen obtenida de: Digikey - ESP32-DEVKITC-32UE

Componente	Cantidad	Precio (USD)	Entrada [V]	Salida [V]	Protocolo	Justificaciones
Módulo Micro SD [2] 	1	3.84	3.3 V	Digital	SPI	Se utiliza para almacenar los datos de los sensores en tiempo real, permitiendo crear, leer y escribir archivos de manera sencilla durante el funcionamiento del sistema. Además, ofrece una interfaz estándar y confiable para la gestión de archivos, lo que facilita el almacenamiento continuo de datos experimentales sin necesidad de conexión constante a un equipo externo.
MCP3208 (ADC) [3] 	1	8.07	5 V	Digital	SPI	Este módulo convierte hasta ocho señales analógicas en digitales mediante comunicación SPI, evitando saturar los pines del microcontrolador. Además, protege al microcontrolador de voltajes superiores a 3.3 V, mejorando la eficiencia del sistema. Cabe resaltar que en las primeras pruebas se uso este mismo modelo de Convertidor Analógico-Digital (ADC) pero de 4 canales.
DS3231 (Reloj)[4] 	1	22.75	5 V	Digital	I2C	Reloj en tiempo real que mantiene la hora y fecha actualizadas incluso cuando el sistema está apagado, gracias a su batería integrada. Es clave para registrar datos con un sello temporal preciso y programar pruebas automáticas a horas específicas.
ANT-DB1-WRT-UFL (Antena) [5] 	1	–	–	–	–	Permite ampliar el alcance de conexión inalámbrica del microcontrolador (*Wi-Fi* o *Bluetooth*). Su diseño con sellado asegura una instalación segura en exteriores, protegiendo el sistema y garantizando una conexión estable.
942-IRLB8721PB (MOSFET) [6] 	5	1.85	12 V	Analógica	–	Controla actuadores de 12 V mediante señales de 3.3 V del microcontrolador, funcionando como interruptor electrónico que permite o bloquea el paso de energía según la señal de control.

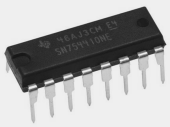
Continúa en la siguiente página

[2]Imagen obtenida de: Adafruit - Micro SD SPI
[3]Imagen obtenida de: Arrow - MCP3208-CI/P
[4]Imagen obtenida de: Distrelec - DS3231 Precision RTC
[5]Imagen obtenida de: Amazon - Antenna Dome WiFi6 2.4/5.8GHz 1.32 UFL
[6]Imagen obtenida de: Amazon - Controlador de Carga Solar PWM

Componente	Cantidad	Precio (USD)	Entrada [V]	Salida [V]	Protocolo	Justificaciones
LM4040BIZ-5.0 (Referencia de 5 V) [7] 	1	1.52	12 V	Analógica	–	Es esencial para asegurar lecturas precisas del ADC. Durante las pruebas, se detectaron variaciones en el voltaje del microcontrolador, lo que afectaba la precisión. Al integrar una referencia de 5 V estable, se garantiza un voltaje fijo y mediciones más exactas.

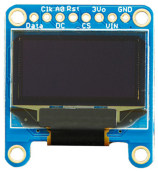
2.2.2. Adquisición e Interacción

Tabla 5.2: Componentes de adquisición e interacción

Componente	Cantidad	Precio (USD)	Entrada [V]	Salida [V]	Protocolo	Justificaciones
Termopar tipo K [8] 	2	32.44	–	Analógica	–	La temperatura de los cartuchos se monitorea usando termopares tipo K, seleccionados por su resistencia a la corrosión y amplio rango de medición (-200 °C a 1250 °C). Su señal analógica debe ser convertida para su procesamiento en el sistema.
MCP9600 (Amplificador de termopar)[9] 	2	20.74	5 V	Digital	I2C	Se integró el módulo Adafruit MCP9600 para convertir y amplificar la señal de los termopares tipo K, facilitando su lectura por el microcontrolador. Se eligió por su bajo costo, disponibilidad y compatibilidad con diversos termopares, entregando la temperatura en grados centígrados.
SN754410NEE4 (Puente H)[10] 	1	4	5 V	Analógica	–	Permite regular el voltaje que alimenta a la bomba mediante una señal PWM generada por el microcontrolador. Según el ciclo de trabajo del PWM, se ajusta el voltaje entre 0 V y 12 V, controlando así el flujo de aire que la bomba suministra al sistema.

Continúa en la siguiente página

[7]Imagen obtenida de: Mouser - LM4040BIZ-5.0/NOPB
[8]Imagen obtenida de: Mouser - Accesorios de Adafruit Thermocouple Type-K
[9]Imagen obtenida de: Top Electronics - Adafruit MCP9600 I2C Thermocouple Amplifier
[10]Imagen obtenida de: Arrow - MCP3208-CI/P

Componente	Cantidad	Precio (USD)	Entrada [V]	Salida [V]	Protocolo	Justificaciones
Pantalla OLED - 128x64[11] 	1	22.75	5 V	Digital	I2C	Esta pantalla de diodos orgánicos de emisión de luz (OLED) tiene un tamaño adecuado para integrarse en la carcasa del muestreador sin ocupar demasiado espacio ni consumir mucha energía. Su función principal es mostrar datos en tiempo real del muestreador y permitir al usuario interactuar con ciertas funciones, utilizando un *joystick* como cursor y la pantalla como visualizador
SHT30 (Termo-higrometro)[12] 	1	32.44	3.3 V	Digital	I2C	Seleccionado principalmente por su protección contra las condiciones de intemperie, ya que es necesario monitorear los datos del exterior de la caja, como la temperatura y la humedad del ambiente.
TS6T1S02A (*Joystick*)[13] 	1	265.77	5 V	Analógica	–	Seleccionado por su diseño resistente para exteriores. Permite al usuario navegar por las opciones en pantalla. Funciona como un potenciómetro, generando señales para los ejes X e Y. Incluye un botón de pulsación que envía una señal digital que es posible leer en el microcontrolador. Cabe resaltar que en primeras pruebas se uso un modulo de *joystick* más básico.
SEN55 (Sensor: COVs y PM´s)[14] 	1	30.84	5 V	Digital	I2C	Emplea tecnología óptica para medir la calidad del aire, reportando concentraciones de material particulado en distintos tamaños, además de índices de compuestos orgánicos volátiles (COVs) y óxidos de nitrógeno (NOx). Estos datos permiten identificar variaciones en los niveles de contaminantes a lo largo del día.

Continúa en la siguiente página

[11]Imagen obtenida de: Digikey - GRAPHIC DISPLAY OLED WHITE 0.96"
[12]Imagen obtenida de: Digikey - SHT-30 MESH-PROTECTED WEATHER-PR
[13]Imagen obtenida de: Digikey - SWITCH THUMBSTCK PUSHBUTTON HALL
[14]Imagen obtenida de: Mouser - LM4040BIZ-5.0/NOPB

Componente	Cantidad	Precio (USD)	Entrada [V]	Salida [V]	Protocolo	Justificaciones
SEN0392 (Sensor: calidad de aire)[15] 	1	18.85	3.3 V	Digital	I2C	Complementa las mediciones del modelo SEN55 al proporcionar un índice de calidad del aire. Este índice se interpreta mediante una tabla proporcionada por el fabricante y, al igual que el SEN55, es útil para identificar cambios en el comportamiento de la calidad del aire. Además, permite comparar los picos y valles registrados con las mediciones del otro sensor.
H7Et BVM (Contador de horas)[16] 	1	30.84	12 V	Digital	–	Este componente inicia el registro de tiempo cuando detecta voltaje en su entrada, indicando que el dispositivo monitoreado está encendido. Mide el tiempo en horas y permite verificar que las pruebas hayan cumplido la duración programada. Funciona de manera independiente, sin necesidad de conexión al microcontrolador, solo requiere estar vinculado al dispositivo que se desea supervisar.
EV-2M-12 (Electroválvulas Clippard)[17] 	2	100	12 V	Digital	–	Estas válvulas, controladas por voltaje, cuentan con dos posiciones: abiertas o cerradas. Al recibir voltaje, se activan y permiten el paso del aire; sin voltaje, permanecen cerradas, bloqueándolo. Su función principal es dirigir el flujo de aire hacia los cartuchos de muestreo según sea necesario para la recolección de muestras.
D6F-P0010A2 (Flujómetro)[18] 	1	55.54	5 V	Analógica	–	Seleccionado por su funcionamiento sencillo, su bajo costo y su rango de lectura de flujo. Cuenta con solo tres pines de conexión: dos para la alimentación y uno para la salida analógica de voltaje, que representa el flujo medido.

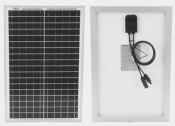
Continúa en la siguiente página

[15]Imagen obtenida de: Digikey - FERMION: SGP40 AIR QUALITY SENSO
[16]Imagen obtenida de: H7ET-BVM Contador electrónico DC 5-30 V
[17]Imagen obtenida de: Clippard - EV-2M-12
[18]Imagen obtenida de: Mouser - D6F-P0010A2

Componente	Cantidad	Precio (USD)	Entrada [V]	Salida [V]	Protocolo	Justificaciones
SEN-15901 (Sistema de Viento) [19] 	1	97.5	5 V 3.3 V	Analógica/ Digital	–	Este sistema incorpora una veleta y un anemómetro para medir la dirección y velocidad del viento. La veleta actúa como una resistencia variable, entregando una señal analógica que indica el punto cardinal del viento. El anemómetro funciona como un interruptor, generando una señal digital proporcional a la velocidad.

2.2.3. Alimentación Autónoma

Tabla 5.3: Componentes del sistema de alimentación

Componente	Cantidad	Precio (USD)	Entrada [V]	Salida [V]	Justificaciones
Celda Solar [20] 	1	–	–	18 V	Esta celda fue elegida por su bajo costo y por la capacidad de proporcionar 18 V a su salida, lo que resulta adecuado para alimentar el sistema y recargar las baterías de manera eficiente.
D24V22F5 y D24V22F3 (Reg de Voltaje (5 y 3.3[V])) [21] 	1	21.95	12 V	5 V	La ESP32 no es capaz de alimentar por sí sola todos los sensores y módulos que requieren distintos niveles de voltaje. Además, conectar los componentes directamente a la batería podría generar inestabilidad.
	1	21.95	12 V	3.3 V	Por ello, se utilizaron reguladores tipo buck, que convierten eficientemente los 12 V de las baterías a 5 V y 3.3 V, según lo requerido. Esto mejora la eficiencia energética y reduce la generación de calor.
Controlador de Carga Solar [22] 	1	–	18 V	12 V	Para garantizar que el muestreador fuera completamente autónomo, se incorporó un controlador encargado de gestionar la energía entre las baterías y el panel solar. Su función principal es maximizar el aprovechamiento de la energía solar durante los días soleados y mantener las baterías recargadas. Este modelo fue seleccionado por su bajo costo y confiabilidad comprobada en otros proyectos, asegurando la eficiencia energética del sistema.

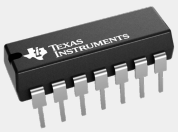
Continúa en la siguiente página

[19]Imagen obtenida de: Mouser - SEN-15901
[20]Imagen obtenida de: SYSCOM - Modulo Solar EPCOM POWER LINE
[21]Imagen obtenida de: Polulu - Step-Down Voltage Regulator D24V22F5
[22]Imagen obtenida de: Amazon - Controlador de Carga Solar PWM

Componente	Cantidad	Precio (USD)	Entrada [V]	Salida [V]	Justificaciones
Baterías 	2	–	–	12 V	Para garantizar la autonomía del sistema, se utilizaron baterías de 12 V - 10.5 Ah en paralelo, almacenan la energía producida por el panel solar y la suministran durante las noches. Esto permite que el sistema opere hasta 40 horas sin necesidad de recarga, considerando un consumo promedio de 500 mA.

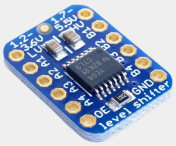
2.2.4. Evaluados y Descartados:

Tabla 5.4: Componentes evaluados y descartados

Componente	Cantidad	Precio (USD)	Entrada [V]	Salida [V]	Protocolo	Justificaciones
SN74LS04N (Compuerta *NOT*) [23] 	1	1.08	5 V	Analogica	–	Se utilizó para generar señales opuestas a las electroválvulas. La idea original era usar ventiladores cerca de la toma de muestra para evitar la contaminación de los cartuchos cuando la bomba estuviera apagada. Sin embargo, el problema se resolvió reubicando las electroválvulas al final del circuito neumático, lo que bloqueó el paso de aire a los cartuchos cuando no se realizaba una prueba, eliminando la necesidad de la compuerta.
SN74LS14 (Schmitt Trigger Inverter) [24] 	1	0.94	5 V	Digital	–	Se uso para estabilizar las señales digitales del botón del *joystick* y del interruptor del anemómetro, ya que este componente permite eliminar rebotes y ruidos eléctricos. Sin embargo, su salida de 5 V no es posible conectarla directamente al micro, que opera a 3.3 V. Durante las pruebas se observó que ambas señales eran estables y no presentaban rebotes significativos, por lo que su uso no representó una ventaja notable y se consideró innecesario.

Continúa en la siguiente página

[23] Imagen obtenida de: Texas Instruments - SN74LS04
[24] Imagen obtenida de: Pinterest - SN74LS14 Schmitt Trigger

Componente	Cantidad	Precio (USD)	Entrada [V]	Salida [V]	Protocolo	Justificaciones
TXB0104 (Bi-Directional Level Shifter) [25] 	1	73.57	5 V y 3.3 V	Digital	–	Este cambiador de nivel se usó para adaptar la salida de 5 V del *Schmitt trigger* al microcontrolador (3.3 V). Sin embargo, al descartarse el uso de este por ser innecesario, también se eliminó el cambiador de nivel.

3. Diseño del modelo funcional

Ya con todos los componentes necesarios para el proyecto, se comenzaron a probar cada uno de ellos de forma individual antes de integrarlos en un solo modelo. Estas pruebas se realizaron utilizando cables removibles (*jumpers*) y una placa de pruebasidaste marcar en (*protoboard*), lo que permitió evaluar el funcionamiento de los sensores tanto por separado como en conjunto, sin necesidad de fijar ningún componente. Esta configuración también facilitó el reemplazo de cualquier sensor o módulo en caso de ser necesario.

En esta etapa, además, se definieron los pines del microcontrolador para cada componente y se adaptaron los códigos proporcionados por algunas bibliotecas preexistentes. El objetivo fue optimizar su uso, asegurando que solo entregaran los datos requeridos y eliminando funciones innecesarias.

3.1. Primera prueba con sensores

3.1.1. Amplificador con termopar tipo K (MCP9600)

Primero se probaron los sensores encargados de convertir las señales analógicas de los termopares a señales digitales con protocolo I2C. Se buscó el módulo directamente en la página de Adafruit y se observó que contaban con una biblioteca llamada `Adafruit_MCP9600` que facilitó programar las funciones para inicializar el componente. Se conectó el módulo a pines del microcontrolador, se aprovechó el código del ejemplo 1 con el que cuenta la biblioteca, que muestra los valores de temperatura del termopar en el monitor serial y se verificó que concordaran con la temperatura del cuarto que mostraba el termohigrómetro AMPROBE modelo THWD-2.

En la gráfica 5.2 se presenta el comportamiento de ambos sensores a lo largo de un periodo de 80 segundos, con mediciones realizadas cada 5 segundos.

[25] Imagen obtenida de: Cetronic - Level Shifter Bi- Direccional Adafruit TXB0104

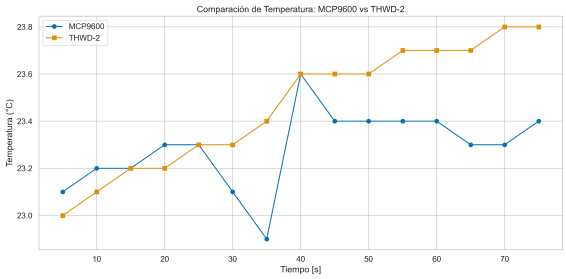


Figura 5.2: Gráfica comparativa MCP9600 - THWD-2

Estas mediciones presentaron un error absoluto medio (MAE) de 0.22 °C y un error cuadrático medio (RMSE) de 0.2768 °C. Para mediciones atmosféricas, estos valores se consideran aceptables dentro de la norma NMX-AA-166/1-SCFI-2013 15/43 [8] para estaciones meteorológicas, climatológicas e hidrológicas, por lo que no se consideró necesario agregar una compensación (*offset*) en la salida de este sensor.

En este caso, los pines del microcontrolador utilizados fueron: el encargado de transmitir la señal de reloj del sistema (SCL) y el encargado de transmitir los datos entre dispositivos (SDA), por lo que en el código solo fue necesario modificar la dirección I2C. Para las primeras pruebas, se empleó un solo termopar, pero posteriormente se añadió un segundo. Gracias a la posibilidad de cambiar su dirección I2C, su integración en el código fue sencilla.

```
#define I2C_ADDRESS_1 (0x67)
#define I2C_ADDRESS_2 (0x60)
```

Para utilizar un termopar, primero se debe crear un objeto de este tipo, asignándole un nombre. En este caso, `mcp_1` y `mcp_2` del tipo `Adafruit_MCP9600`.

```
Adafruit_MCP9600 mcp_1;
Adafruit_MCP9600 mcp_2;
```

Para facilitar la lectura de los datos, también se crearon funciones de tipo `float` que permiten obtener directamente el valor de la lectura al ser llamadas. En el caso de los termopares, se definieron dos funciones: `MCP9600_1_ReadT()` y `MCP9600_2_ReadT()`, cada una encargada de retornar la temperatura correspondiente a su sensor.

```
float MCP9600_1_ReadT(){
    float t = mcp_1.readThermocouple();
    return t;
}
float MCP9600_2_ReadT(){
    float t = mcp_2.readThermocouple();
    return t;
}
```

3.1.2. ADC MCP3208 y MCP3204

Para leer correctamente los sensores analógicos, primero fue necesario configurar el convertidor analógico a digital (ADC). Se utilizó la biblioteca `‘‘MCP320X.h’’`, que

simplifica su implementación y permite una integración rápida con el sistema. Solo fue necesario conectar los pines correspondientes del microcontrolador, respetando aquellos dedicados a la comunicación SPI. En este caso, el pin 5 se utilizó como *chip select*, el 18 como reloj, el 23 como MOSI y el 19 como MISO.

```
#define CS_PIN 5
#define CLOCK_PIN 18
#define MOSI_PIN 23
#define MISO_PIN 19
#define MCP3208 8
```

Para comenzar a usarlo basta con crear un objeto del tipo `MCP320X` que llamamos `mcp3208`, junto con algunos parámetros, los cuales básicamente son los nombres de los pines antes asignados. Para poder leer alguno de los canales, solo bastaba con llamar el objeto creado junto a la función propia `readADC(X)`, donde `X` es un número entero que indica qué canal se desea leer.

```
MCP320X mcp3208(MCP3208, CLOCK_PIN, MOSI_PIN, MISO_PIN, CS_PIN);

float lectura=mcp3208.readADC(x);
```

Para comprobar que el ADC realizará las conversiones correctamente, inicialmente conectamos una fuente de alimentación de 0 V a 5 V]directamente a la entrada del ADC. Se observó que el valor leído en el monitor serial coincidiera con el voltaje suministrado por la fuente. Posteriormente, se reemplazó la fuente por un potenciómetro analógico configurado como divisor de voltaje, donde el pin 1 recibía la entrada de voltaje, el pin 2 (central) funcionaba como salida de voltaje variable, y el pin 3 se conectaba a GND. De este modo, se verificó que al girar el potenciómetro, el valor leído por el ADC cambiará proporcionalmente. Además, se corroboró que en los extremos de su recorrido, el ADC reflejaba correctamente tanto el voltaje máximo como el mínimo (cero, correspondiente a GND).

Cabe destacar que el primer modelo de ADC que se utilizó y probó fue el MCP3204, el cual opera de manera similar al MCP3208. La principal diferencia entre ambos radica en el número de entradas analógicas disponibles: el MCP3204 cuenta con 4 canales, mientras que el MCP3208 ofrece 8. En las primeras pruebas, se utilizó el modelo de 4 canales, lo que permitió conectar únicamente el sensor de flujo al sistema. Esta prueba inicial fue útil para validar la correcta lectura y conversión de los datos antes de integrar más sensores con el modelo de mayor capacidad.

3.1.3. Puente H (SN754410NEE4), compresor y flujómetro

Posteriormente se probó el puente H. A la entrada se conectaron los pines para proporcionar el PWM y los pines para controlar el encendido, apagado y dirección de la corriente. A la salida se conectaron las terminales de alimentación del compresor. Un código sencillo se programó para que el puente H encendiera el compresor, variara el PWM y mantuviera el sentido de la corriente en una sola dirección.

Se observó cómo el compresor cambiaba su intensidad de trabajo conforme se modificaba el ciclo de trabajo de la señal de modulación por ancho de pulsos (PWM) y se identificó en qué valor de ciclo de trabajo ya no proporcionaba el voltaje suficiente para encender el compresor. Este dato era indispensable para contar con un valor límite al que tiene la capacidad de llegar el ciclo de trabajo sin que se apague el compresor.

```
void loop() {
    analogWrite(ENA, cicloTrabajo);
    cicloTrabajo += 10;
    if (cicloTrabajo > 255) {
        cicloTrabajo = 0;
    }
    delay(1000);
}
```

A continuación, se desarrollaron las funciones necesarias para controlar la bomba, permitiendo encenderla, apagarla e inicializar correctamente sus pines. Aunque son funciones simples, son fundamentales para garantizar un control preciso en todo momento, además de ahorrar código.

1. **Compresor y Puente H**

   a) **Inicialización del Compresor:** Para la configuración inicial, se implementó un control por modulación por ancho de pulso (PWM), lo que permite ajustar la velocidad de la bomba mediante cambios en su voltaje de entrada, afectando así el flujo de aire.

   En la ESP32, se definieron los siguientes parámetros para el PWM:

   - Frecuencia: 5000 Hz
   - Canal: 0
   - Resolución: 10 bits (lo que permite valores de 0 a 1024, brindando mayor precisión en comparación con una resolución de 8 bits).
   - Pin de ESP32: `PWMB - 25`

```
#define PWMB 25
const int frecuencia = 5000;
const int canal = 0;
const int resolucion = 10;
```

   Además, se configuraron los pines del puente H como salidas para controlar la dirección de la corriente y, por ende, el funcionamiento de la bomba. `bom1` corresponde al pin 33 y `bom2` al pin 32. A esta función se le nombró `Bomba_Init()`.

```
void Bomba_Init(){
    ledcSetup(canal, frecuencia, resolucion);
    ledcAttachPin(PWMB, canal);
    pinMode (bom1, OUTPUT);
```

```
    pinMode (bom2, OUTPUT);
}
```

b) **Encendido del compresor:** En esta función llamada `EncenderBom()`, se establece un pin en alto y el otro en bajo para activar el puente H, permitiendo así el flujo de corriente hacia la bomba. Luego, se asignó el valor correspondiente al PWM utilizando la variable `vel`, que más adelante estará vinculada directamente con la salida del PID para el control automático.

Adicionalmente, se implementó un booleano llamado `EstadoBom`, el cual se establece en `true` cuando la bomba se enciende. Esto permite monitorear su estado en tiempo real.

```
void EncenderBom(){
    digitalWrite (bom1, HIGH);
    digitalWrite (bom2, LOW);
    ledcWrite(canal, vel);
    EstadoBom=true;
}
```

c) **Apagado del compresor:** Para detener la bomba se creó la función `PararBom()`, en donde se enviaron ambos pines a nivel bajo, evitando que el puente H entregue corriente. Asimismo, el valor del PWM se establece en 0 para garantizar que la bomba permanezca completamente apagada.

```
void PararBom() {
    digitalWrite (bom1, LOW);
    digitalWrite (bom2, LOW);
    ledcWrite(canal, 0);
}
```

El booleano `EstadoBom` se actualiza a `false`, indicando que la bomba está apagada. Además, para asegurar un reinicio adecuado, se restablecen los valores asociados al PID, permitiendo que el control comience desde un estado inicial sin interferencias de datos previos. Esto se detalla más adelante en la sección dedicada al PID.

Asimismo, los valores relacionados con la prueba de 24 horas se reinician a sus valores iniciales. Entre ellos, se encuentra la variable `interv`, que define el intervalo de conmutación de las válvulas, junto con `TPruebaHor` y `TPruebaMin`, variables asociadas al tiempo total de prueba. Estas últimas se explican con mayor detalle en la sección sobre el control programado de pruebas.

```
PID_error=0.0;PID_p=0.0; PID_i=0.0; PID_d=0.0;previous_error
   =0;PID_value=0;
Kp=0.08; Ki=0.05; Kd=0.010;
EstadoBom=false;
```

2. **Flujómetro**

Las pruebas del puente H fueron de gran utilidad para evaluar el sensor de flujo analógico. Utilizando mangueras neumáticas, se conectó la salida de aire del compresor a la entrada del flujómetro. Este sensor se alimentó con los 5 V proporcionados por la ESP32, y su salida se midió con un multímetro. Se observó que el voltaje en la salida variaba en función de la potencia del compresor.

Teniendo el ADC configurado, se creó una función llamada `Read_MCP3208(int x)` de tipo `float` para leer el voltaje de salida del flujómetro. Esta función solicita el canal que se quiere leer para obtener un valor de voltaje de 12 bits. Al tener este valor, se multiplica por 5 (que corresponde al voltaje de referencia) y luego se divide entre 4095 (el máximo valor representado por 12 bits). De este modo, el valor leído en bits se convierte nuevamente en voltaje, y este resultado se retorna.

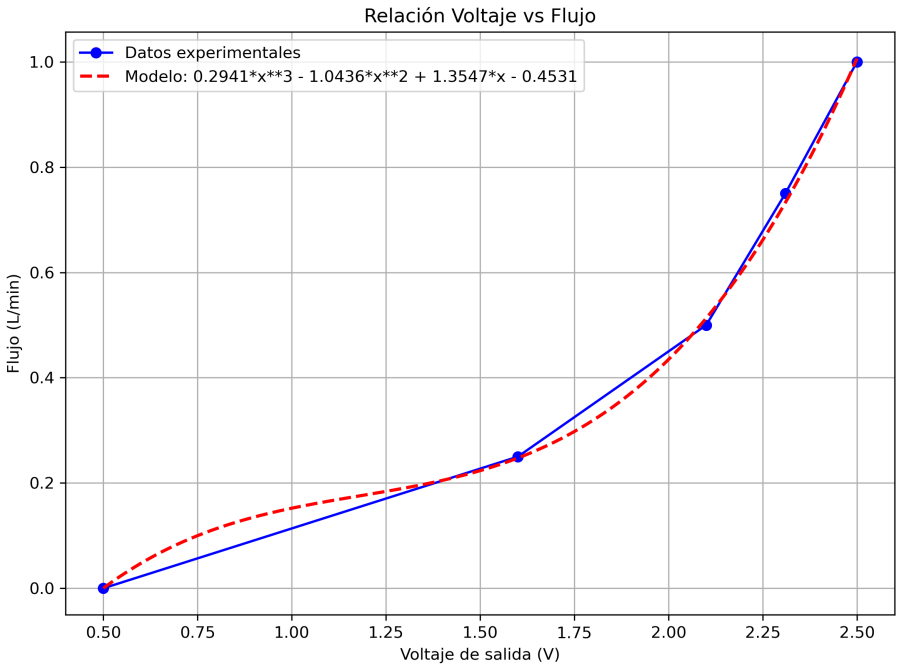
```
float Read_MCP3208(int x) {
    float lectura;
    lectura=((float)mcp3208.readADC(x)*5.0)/4095.0;
    delay(40);
    return lectura;
}
```

La hoja de datos del flujómetro incluye una gráfica que relaciona el voltaje de salida con el flujo medido. A partir de estos datos, registrados en condiciones ideales de temperatura y humedad (25 °C y 35-75 %), se obtuvo la ecuación de regresión lineal empleando Excel. Se optó por una función polinómica de tercer orden, ya que era la que mejor se ajustaba al comportamiento observado.

Flow rate L/min (normal)	0	0.25	0.50	0.75	1.00
Output voltage V	0.50 ±0.10	1.60 ±0.10	2.10 ±0.10	2.31 ±0.10	2.50 ±0.10

Measurement conditions: Power supply voltage of 5.0±0.1 VDC, ambient temperature of 25±5°C, and ambient humidity of 35% to 75%.

Figura 5.3: Mediciones de acuerdo a hoja de datos. Imagen tomada de hoja de datos D6F-P.



$$y = 0.2941x^3 - 1.0436x^2 + 1.3547x - 0.4531 \tag{5.1}$$

Figura 5.4: Gráfica con ecuación de regresión lineal.

Posteriormente, se creó otra función de tipo `float` llamada `SensorFlujo(float val)`, que toma como parámetro un valor de voltaje (en este caso, el resultado de la función `Read_MCP3208(int x)`). Esta función contiene la ecuación de la regresión lineal de tercer orden, de modo que el resultado es un valor que representa el flujo de aire. Este valor es el que se retorna.

Además, la función incluye una condición que establece que, si el valor de entrada es menor a 0.5 V, el flujo se considera 0 mL/min. Esto se debe a que, según la hoja de especificaciones del flujómetro, un voltaje inferior a 0.5 V corresponde a un flujo nulo.

```
1 float SensorFlujo(float val){
2     float y=0.2941*(pow(val,3))- 1.0436*(pow(val,2))+ 1.3547*(
    val) - 0.4531;
3 
4     if(val<=0.5){
5         y=0;
6     }
7 }
8 
```

3.1.4. Reloj de precisión RTC (DS3231)

Posteriormente, se realizaron pruebas con el reloj en tiempo real. Al ser también un dispositivo I2C, solo requiere la conexión de sus pines SDA y SCL. En este caso,

la biblioteca ‘‘RTClib.h’’ asigna automáticamente su dirección I2C (0x68), por lo que únicamente es necesario crear un objeto llamado rtc del tipo RTC_DS3231.

```
#include <Wire.h>
#include <RTClib.h>
RTC_DS3231 rtc;
```

Para su funcionamiento, basta con inicializarlo llamando a su función propia begin(). De manera análoga a los termopares, si ocurre un problema durante la inicialización, se genera un código de error asignado, en este caso, el número 2. Para mantener un código más ordenado y compacto en setup(), se creó una función específica de inicialización llamada Reloj_Init().

```
void Reloj_Init(){
    if (! rtc.begin()){
        uint8_t error = 2;
    }
    Serial.println("-----------------------------------------------")
    ;
    Serial.println("Reloj RTC inicializado correctamente!");
    Serial.println("-----------------------------------------------")
    ;
}
```

En las pruebas iniciales, se programó la fecha y hora correctas utilizando la función rtc.adjust(DateTime(2014, 1, 21, 3, 0, 0)). Posteriormente, se desconectó temporalmente el módulo para verificar que siguiera funcionando únicamente con la batería de respaldo. Al volver a conectarlo, se comprobó que mantenía la hora y fecha exactas.

Con el reloj correctamente configurado, se realizaron pruebas adicionales utilizando un LED. Al observar al LED apagándose y encendiéndose a las horas correspondientes, se corroboró la sincronización del reloj con el sistema.

3.1.5. Electroválvulas, contador de horas y ventiladores

Para probar el funcionamiento de las electroválvulas, que necesitan de 12 V para activarse, se utilizó un transistor de efecto de campo metal-óxido-semiconductor (MOSFET) como compuerta de corriente para la electroválvula. Con esta configuración, al aplicar un voltaje de 3.3 V, proporcionado por el microcontrolador, se abría la compuerta del MOSFET para permitir que la electroválvula recibiera los doce voltios que se suministraron de una fuente externa. Por seguridad del funcionamiento de la electroválvula, se agregó un diodo en sus terminales para evitar que exista un regreso de corriente que dañe el dispositivo.

Esta configuración del MOSFET como interruptor también sería útil para encender el ventilador que se encarga de enfriar todo el sistema, el contador de horas y ventiladores con los que se pensaba contar en la salida de cada toma de muestra para evitar que se contaminara mientras no estuviera recolectando. Todos estos componentes se alimentan a 12[V]. De esta forma, el control se realiza con el microcontrolador

(ESP32) y la alimentación con una fuente externa de 12[V].

Los pines utilizados en la ESP32 para este caso fueron:

- Pin 26 → Válvula 1: `electroVal1`
- Pin 27 → Válvula 2: `electroVal2`
- Pin 17 → Ventilador: `vent`

Dado que ya se cuenta con el acondicionamiento mediante MOSFETs, estos pines se configuran únicamente como salidas digitales. De este modo, cuando el pin correspondiente está en alto, el dispositivo se enciende, y cuando está en bajo, se apaga. Por lo que en el `setup()` solo se inician en `low`.

```
pinMode(electroValA, OUTPUT);
pinMode(electroValB, OUTPUT);
pinMode(vent, OUTPUT);
digitalWrite(electroValA, LOW);
digitalWrite(electroValB, LOW);
digitalWrite (vent, LOW);
```

3.1.6. Termohigrómetro (SHT30) y sensor de índice de calidad del aire (SEN0392)

Cada sensor se probó de forma individual, conectando los pines correspondientes para alimentación, transmisión y recepción de datos, siendo los pines de datos SDA y SCL. Para establecer una comunicación correcta y eficiente con estos componentes, se utilizaron bibliotecas previamente desarrolladas:

- Para el sensor SHT30, se utilizó la biblioteca `‘‘Adafruit_SHT31.h’’`.
- Para el sensor SEN0392, se utilizó la biblioteca `‘‘DFRobot_SGP40.h’’`.

Con el fin de acceder a las funciones específicas de cada sensor, se crearon objetos:

- Un objeto del tipo `Adafruit_SHT31` llamado `sht30` para el sensor `SHT30`.
- Un objeto del tipo `DFRobot_SGP40` llamado `mySgp40` para el sensor `SEN0392`.

En el caso del SHT30, fue necesario conocer su dirección de comunicación, que es `0x44`. Para el SEN0392, no fue necesario especificar la dirección manualmente, ya que la biblioteca la configura automáticamente.

Se utilizó el código de ejemplo 2 con el que cuenta la biblioteca para probar el SHT30. Este código permite imprimir en el puerto serial los datos registrados por el sensor en tiempo real. Estos datos se compararon con los valores de temperatura y humedad obtenidos con el termohigrómetro comercial AMPROBE modelo THWD-2, para verificar si era necesario contar con un *offset* en alguna de sus salidas.

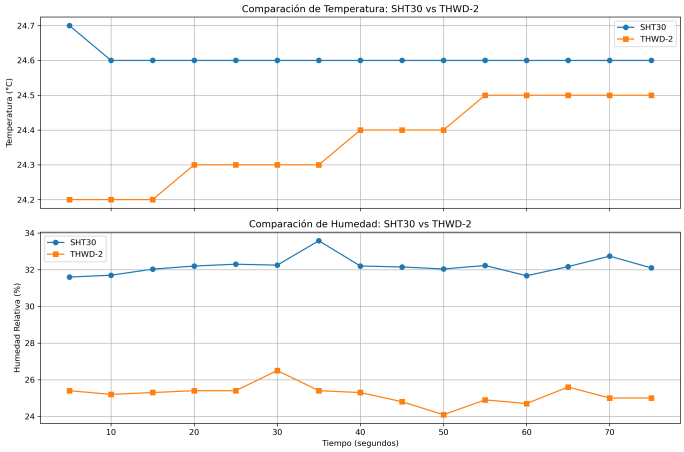


Figura 5.5: Gráfica comparativa de temperatura y humedad relativa del SHT30 vs. THWD-2.

El error absoluto medio (MAE) de la temperatura medida por el SHT30 fue de 0.24°C, mientras que el error cuadrático medio (RMSE) fue de 0.2708°C. Al tratarse de mediciones atmosféricas y con los valores de precisión para mediciones de higrómetros digitales proporcionados por la NMX-AA-166/1-SCFI-2013 15/43, este error se considera aceptable, por lo que no fue necesario aplicar ningún tipo de corrección u *offset*.

Por otro lado, en la humedad se registró un MAE de 6.998 % y un RMSE de 7.026 %. De acuerdo a la norma este error no es aceptable para mediciones de humedad relativa. Por este motivo y porque ambos instrumentos presentan un comportamiento similar, se decidió aplicar un ajuste mediante un *offset* negativo de 7 %, con el fin de obtener mediciones más precisas.

Para el componente SGP40 se utilizó el código de ejemplo 3 con el que cuenta su biblioteca, para imprimir el índice de COVs registrado por el sensor. Al tratarse de un indicador de calidad del aire y no contar con un sensor de referencia para compararlos, se evaluó al observar que sus valores cambiaran en respuesta a modificaciones en la composición del aire dentro de un perímetro reducido.

De igual manera que con los códigos anteriores, se crearon funciones específicas para la inicialización de los sensores `SHT30_Init()` y `SGP40_Init()`, asignando códigos de error en caso de falla. En este caso, el SHT30 tiene asignado el código de error 3, mientras que el SEN0392 cuenta con el código de error 6.

```
1 void SHT30_Init(){
2     if (! sht30.begin(0x44)) {
3         uint8_t error = 3;
4     }
5     Serial.println("-----------------------------------------------")
     ;
6     Serial.println("Higrometro inicializado correctamente!");
```

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

Listing 5.1: Función de inicialización SHT30

```
void SGP40_Init(){
    while(mySgp40.begin(10000) !=true){
        uint8_t error = 6;
    }
    Serial.println("-----------------------------------------------")
    ;
    Serial.println("sgp40 inicializado correctamente!");
    Serial.println("-----------------------------------------------")
    ;
}
```

Listing 5.2: Función de inicialización SGP40

Para facilitar la lectura de los datos, también se crearon funciones de tipo `float` que permiten obtener directamente el valor de la lectura al ser llamadas. En el caso del SHT30, se definieron dos funciones: `SHT30_ReadT()` y `SHT30_ReadH()`, una para temperatura y otra para humedad, del tipo `float` ambas. En cuanto al `SEN0392`, se hizo una función: `Read_SGP40()`, de tipo entero, ya que el valor es un índice entero.

```
uint16_t Read_SGP40(){
    uint16_t index = mySgp40.getVoclndex();
    return index;
}
float SHT30_ReadT(){
    float t = sht30.readTemperature();
    return t;
}
float SHT30_ReadH(){
    float h = sht30.readHumidity();
    return h;
}
```

3.1.7. Módulo micro SD

Se realizaron pruebas con el módulo micro SD para entender y corroborar su funcionamiento. En este caso, la propia biblioteca de la ESP32 incluía un ejemplo de uso para una tarjeta SD, proporcionando diversas funciones para crear, borrar y manipular archivos a través del ejemplo ‘‘`SD`’’. Para verificar su funcionamiento, se conectó el módulo a los pines SPI correspondientes en el microcontrolador, de manera similar a como se hizo con el ADC. Este usaba algunas bibliotecas tales como ‘‘`FS.h`’’, ‘‘`SD.h`’’ y ‘‘`SPI.h`’’.

```
#include "FS.h"
#include "SD.h"
#include "SPI.h"
```

Posteriormente, se programó una secuencia que generaba archivos .txt utilizando las funciones propias del ejemplo: `writeFile(fs::FS &fs, const char * path, const char * message)` y `appendFile(fs::FS &fs, const char * path, const char * message)`. La diferencia entre ambas es que la primera (`writeFile`) escribe en el archivo y, si ya existía contenido, lo sobrescribe; mientras que la segunda (`appendFile`) realiza una concatenación, manteniendo el contenido previo. Ambas funciones, al ser llamadas con un parámetro de nombre de archivo (`path`), lo crean si no existe, y si existe, simplemente escriben el contenido de `message`. El parámetro `path` corresponde al nombre del archivo, el cual comienza con una barra diagonal ‘‘/’’, y `fs` se refiere a un objeto de tipo `SD` que se crea automáticamente con las bibliotecas antes mencionadas.

```
1 void writeFile(fs::FS &fs, const char * path, const char * message){
2     File file = fs.open(path, FILE_WRITE);
3     file.close();
4 }
5
6 void appendFile(fs::FS &fs, const char * path, const char * message)
    {
7     File file = fs.open(path, FILE_APPEND);
8     file.close();
9 }
```

Se verificó que la posibilidad de crear y escribir archivos fuera de forma correcta, según los mensajes enviados al monitor serial. Luego, se retiró la tarjeta Micro SD, se insertó en una computadora y se revisó su contenido. Se corroboró que el archivo y el mensaje se habían guardado correctamente, lo que validó el funcionamiento adecuado del módulo y de las funciones proporcionadas por la biblioteca.

3.2. Primera prueba en conjunto

Una vez probados los componentes y sensores adquiridos, se desarrolló un programa que integrará el funcionamiento de todos los dispositivos en una rutina conjunta. El objetivo era evaluar su desempeño en conjunto y detectar posibles conflictos o fallas en la interacción entre ellos.

Para la conexión, se utilizó una protoboard con cables cortados a medida, permitiendo la interconexión de todos los sensores, actuadores y convertidores con el microcontrolador. Este se encargaba de recibir datos y enviar comandos para verificar el correcto funcionamiento del sistema en su conjunto.

Toda la electrónica fue alimentada a través de la ESP32, aprovechando sus salidas de 3.3[V] y 5[V] para los sensores y componentes de bajo consumo. Los actuadores que requerían 12[V] para su funcionamiento fueron alimentados mediante una fuente externa.

Al realizar esta prueba en conjunto, se observó que el módulo microSD no funcionaba correctamente. Al enviar el comando para generar los archivos correspondientes,

el módulo no respondía y no se creaba ningún archivo. Esto provocó que se comenzara a investigar exhaustivamente el origen del problema, ya que, de forma individual, el módulo funcionaba perfectamente, pero al integrarlo con el resto del sistema comenzaba a fallar.

Tras realizar múltiples pruebas y descartar posibles errores, se identificó que el problema residía en la red de comunicación SPI. Al utilizar dos dispositivos con este protocolo (el MCP3208 y el módulo micro SD), solo uno de ellos lograba comunicarse correctamente. Al investigar más a fondo, se encontró que varios desarrolladores habían reportado dificultades al conectar múltiples dispositivos SPI en la ESP32 [9].

Para solucionar el problema, se optó por utilizar los dos buses SPI disponibles en la ESP32, asignando una conexión independiente al módulo micro SD y otra al ADC. Esto fue posible porque el microcontrolador cuenta con dos buses SPI separados: el que se usa por defecto, llamado VSPI, y otro llamado HSPI.

Sin embargo, se tuvieron algunos inconvenientes al configurar el segundo bus (`HSPI`). No bastaba con definir los nuevos pines y usarlo de la manera habitual. Para esto, se tuvo que crear un objeto de tipo `SPIClass`, que ya está definido en la biblioteca `SPI.h`, y especificar el bus a utilizar (`HSPI`). Además, se definieron los nuevos pines para este bus: el pin 15 como `chip select`, el 14 como reloj, el 13 como `MOSI` y el 16 como `MISO`.

```
#define SCK  14
#define MISO  16
#define MOSI  13
#define CS  15
```

Posteriormente, se creó una función para inicializar la tarjeta SD, en la que también se inicializó el nuevo bus SPI. Para ello, se llama al objeto `spi` creado anteriormente y se utiliza su función `begin`, pasando como parámetros los pines definidos. De este modo, los pines quedaron correctamente habilitados. Para inicializar la tarjeta SD, se llamó a su función `begin`, como en el ejemplo, y se le pasaron como parámetros el pin `chip select`, el objeto `spi` creado para el nuevo bus y la velocidad de transferencia, que en este caso fue la predeterminada de `80000000`.

En este punto, también se añadieron códigos de error para manejar posibles fallos: el código `8` para cuando no se detecta el módulo micro SD y el código `9` para cuando no se detecta la memoria micro SD.

```
void SD_Init(){
    spi.begin(SCK, MISO, MOSI, CS);

    if(!SD.begin(CS,spi,80000000)){
        uint8_t error = 8;
    }
    uint8_t cardType = SD.cardType();
    if(cardType == CARD_NONE){
        uint8_t error = 9;
```

```
  }
}
```

De este modo, se corrigieron los errores al utilizar las funciones `writeFile` y `appendFile` en conjunto con los demás componentes.

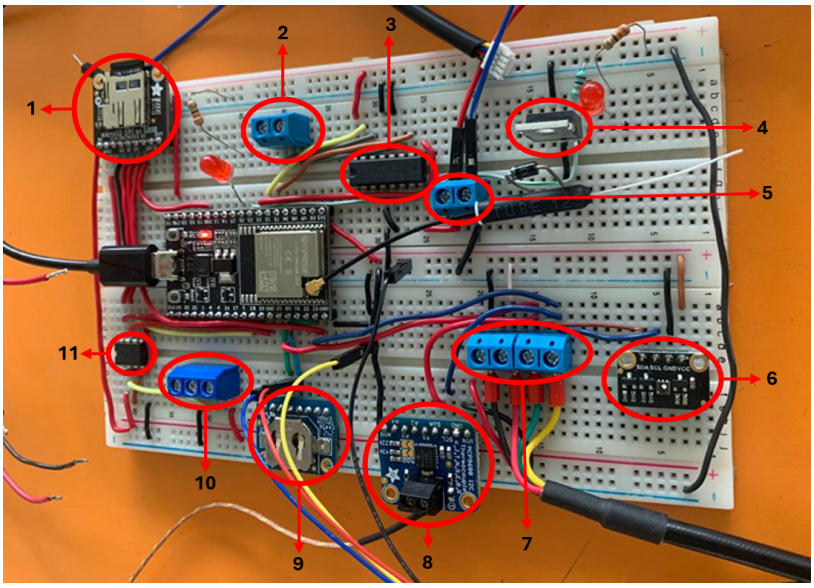


Figura 5.6: Primera prueba en conjunto

A continuación se describen los elementos enumerados en la Figura 5.6:

1. Módulo Micro SD.
2. Terminales compresor.
3. Puente H (SN754410NEE4).
4. MOSFET (942-IRLB8721PB)
5. Alimentación 12 V.
6. Sensor ICA (SEN0392).
7. Termohigrómetro (SHT30).
8. Módulo termopar (MCP9600).
9. Reloj en tiempo real (DS3231).
10. Flujómetro (D6F-P0010A2).
11. ADC de 4 canales (MCP3204).

Este fue el único inconveniente que se tuvo al juntar los módulos por primera vez. Con este problema resuelto, se utilizó este modelo como base para seguir agregando componentes y probarse de forma conjunta.

3.3. Segunda prueba con sensores

3.3.1. OLED y *Joystick*

No hubo inconvenientes al probar la pantalla OLED de forma individual. Al ser un dispositivo con comunicación I2C y contar con una biblioteca que proporcionaba funciones para desplegar información en el display, solo fue cuestión de conectar los

pines correctos a la ESP32 y subirle el código de ejemplo 5 con el que cuenta la biblioteca.

Para el *joystick*, se utilizó el ADC MCP3208 para leer los valores de voltaje de sus dos salidas, correspondientes a la posición en los ejes X y Y. Se conectó el *joystick* a la salida de 3.3[V] de la ESP32 y se codificó un programa sencillo que permitiera leer estos valores mediante el ADC y mostrarlos en pantalla. Así, fue posible identificar los rangos de voltaje para cada una de las posiciones izquierda, derecha, arriba, abajo y reposo.

Posteriormente, se integraron ambos componentes para desarrollar una pequeña interfaz en la pantalla OLED que permitiera desplazarse utilizando el *joystick*. Esta interfaz incluía tres páginas con las que se podía interactuar y navegar entre ellas. El diseño y funcionamiento detallado de esta interfaz se explica en la sección 8. Cabe destacar que este prototipo inicial no contaba con todas las funciones descritas en dicha sección.

3.3.2. Referencia de 5[V] y MCP3208

Al notar que el *joystick* requería dos entradas analógicas adicionales y que debía estar conectado a un ADC para evitar dañar el microcontrolador con posibles salidas de 5[V], se decidió cambiar el modelo de ADC (MCP3204) por otro de la misma familia, pero con más pines (MCP3208). De esta forma, la programación apenas sufrió modificaciones y ahora el sistema contaba con ocho canales para convertir señales analógicas.

Sin embargo, al realizar este cambio, se observó que los valores proporcionados por el ADC no eran muy precisos y fluctuaban, generando lecturas erróneas en la posición del *joystick*. Para solucionar esto, se añadió el componente LM4040BIZ-5.0/NOPB, una referencia de voltaje precisa que proporciona un valor constante de 5[V]. Siguiendo las recomendaciones del datasheet [**?**], se conectó la referencia con la configuración adecuada y se calculó la resistencia R_s necesaria para garantizar una salida estable de 5[V].

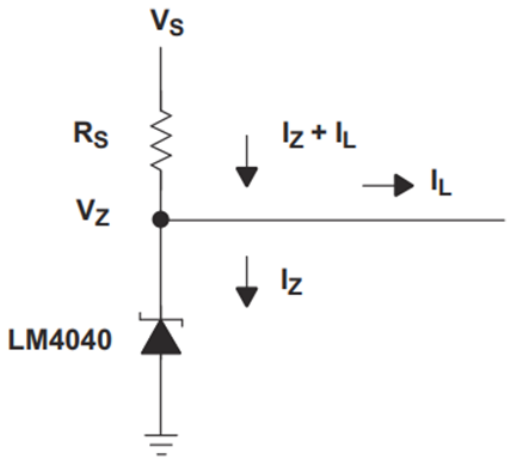


Figura 5.7: Diagrama de uso proporcionado por hoja de datos LM4040.

Como se observa en la imagen, el valor de R_s se calcula mediante la siguiente expresión:

$$R_s = \frac{V_{\text{fuente}} - V_{\text{ref}}}{I_{\text{total}}} \tag{5.2}$$

donde:

- $V_{\text{fuente}} = 12[V]$ (voltaje de alimentación)
- $V_{\text{ref}} = 5[V]$ (voltaje de referencia deseado)
- $I_{\text{total}} = 1[mA]$ (corriente recomendada por el datasheet)

La hoja de datos del `LM4040BIZ-5.0/NOPB` recomienda iniciar el cálculo con una corriente total de $1\,mA$ y ajustarla si se requiere más corriente en la salida.

Dado que el ADC solo necesita el voltaje como referencia y no demanda mucha corriente para su funcionamiento, se decidió mantener la corriente en $1\,mA$, tal como sugiere el datasheet. En nuestro caso particular:

- Voltaje de referencia requerido: $5[V]$
- Voltaje de alimentación disponible: $12[V]$

Aplicando los valores a la ecuación:

$$R_s = \frac{V_S - V_O}{I_T} = \frac{12[V] - 5[V]}{1[mA]} = 7[k\Omega] \tag{5.3}$$

Este valor no es comercial, por lo que se optó por utilizar una resistencia de kΩ6.8 para probar la referencia y verificar que mantuviera los V5 en su salida.

Con este componente, el ADC comenzó a entregar datos de forma más precisa, lo que mejoró:

- La respuesta del *joystick* en la interfaz
- El desempeño de los sensores conectados al ADC

3.3.3. Sensor de material particulado, COV y NOx (SEN55)

Se decidió agregar este sensor para contar con dos dispositivos que indicarán un aproximado de la calidad del aire. A diferencia del `SEN0392`, este sensor es mucho más completo, ya que no solo proporciona un índice de calidad del aire, sino que también mide concentraciones de:

- PM 1.0, 2.5, 6.0 y 10.0 en $\mu g/m^3$
- Niveles de COVs y NO_x presentes en el ambiente

Como el objetivo es únicamente observar su comportamiento y detectar cambios en la composición del aire, y dado que no se contaba con un sensor comercial para comparar mediciones, simplemente se conectó a la red I^2C utilizando la biblioteca existente `SensirionI2CSen5x.h`, se aprovechó el código de ejemplo porporcionado por la biblioteca 4, y se analizó cómo variaban sus lecturas ante perturbaciones en un perímetro reducido.

Al verificar que todo funcionaba correctamente, se depuró el código para dejar solo lo esencial. Para ello, se crearon dos funciones:

1. Función de inicialización del sensor (`SEN55_Init()`)
   - Mantiene la estructura del ejemplo original
   - Incluye código de error 7 para problemas de inicialización

```
void SEN55_Init() {
    while (!Serial) {
        delay(100);
    }
    Wire.begin();
    sen5x.begin(Wire);
    uint16_t err;

    err = sen5x.startMeasurement();
    if(err){
      uint8_t error = 7;
    }
    Serial.println("¡SEN55 inicializado correctamente!");
}
```

2. Función para recopilar datos de manera simplificada
   - Implementa un arreglo `sen5T[6]` para almacenar los 6 parámetros del sensor
   - Actualiza automáticamente los valores con cada llamada
   - Estructura optimizada para lectura secuencial:

- Posición 0: PM 1.0 [µg/m^3]
- Posición 1: PM 2.5 [µg/m^3]
- Posición 2: PM 4.0 [µg/m^3]
- Posición 3: PM 10.0 [µg/m^3]
- Posición 4: VOC [Índice]
- Posición 5: NO_x [Índice]

```
void LectSEN55(){
    uint16_t err;
    char errorMessage[256];

    float massConcentrationPm1p0;
    float massConcentrationPm2p5;
    float massConcentrationPm4p0;
    float massConcentrationPm10p0;
    float ambientHumidity;
    float ambientTemperature;
    float vocIndex;
    float noxIndex;

    err = sen5x.readMeasuredValues(
        massConcentrationPm1p0, massConcentrationPm2p5,
   massConcentrationPm4p0,
        massConcentrationPm10p0, ambientHumidity,
   ambientTemperature, vocIndex,
        noxIndex);

    if (err){
        Serial.println("Error en la captura");
    }else{
    sen5T[0]= massConcentrationPm1p0;
    sen5T[1]= massConcentrationPm2p5;
    sen5T[2]= massConcentrationPm4p0;
    sen5T[3]= massConcentrationPm10p0;
    sen5T[4]= vocIndex;
    sen5T[5]= noxIndex;
    }
}
```

3.3.4. Sistema de viento (Veleta y anemómetro)

El sistema de viento cuenta con dos componentes con funciones diferentes: el anemómetro y la veleta. Uno proporciona una señal analógica, mientras que el otro, al agregarle una resistencia en modo pull-up, genera una señal digital.

La veleta funciona como un divisor de voltaje, donde su resistencia varía según la dirección en la que se encuentre. Debido a que esta salida analógica se conecta al ADC, se decidió alimentar el divisor de voltaje con 5[V], ya que no se conecta directamente al microcontrolador.

Por otro lado, el anemómetro actúa como un interruptor que se activa cada vez que se completa una vuelta. Estos pulsos deben ser registrados por la ESP32 y comparados

con el tiempo transcurrido. Como su salida se conecta directamente al microcontrolador, se optó por alimentarlo con 3.3V para evitar problemas de compatibilidad de voltaje.

Para la programación del anemómetro, se utilizó un código que tomara en cuenta las revoluciones por minuto del anemómetro. Este programa se basa en una interrupción que se activa con cada flanco de subida generado por el anemómetro. En la interrupción, se incrementa una variable que cuenta los pulsos detectados. Luego, se calcula la cantidad de pulsos en función del tiempo transcurrido utilizando la función `millis()`, que devuelve los milisegundos desde que la ESP32 comenzó a ejecutar el programa. Finalmente, se realiza la conversión necesaria para obtener la velocidad del viento en metros por segundo.

```
void IRAM_ATTR isr() {
  unsigned long currentMicros = micros();
  if (currentMicros - last_micros_an >= DEBOUNCE_TIME * 1000) {
    _anemometerCounter++;
    last_micros_an = currentMicros;
  }
}

float readWindSpd() {
    noInterrupts();
    uint32_t counterCopy = _anemometerCounter;
    _anemometerCounter = 0;
    interrupts();

    float spd = (counterCopy/2.2) * 0.6671;
    return spd;
}
```

En cuanto a la veleta, dado que entrega distintos valores de voltaje según su posición, se crearon tres arreglos:

- Un arreglo con los valores máximos de voltaje para cada posición.
- Un arreglo con los valores mínimos de voltaje correspondientes.
- Un arreglo con los ángulos asociados a cada posición.

La implementación utiliza estos arreglos para determinar la dirección del viento comparando la lectura analógica actual con los rangos predefinidos.

```
float dirDeg[] =
   {112.5,67.5,90,157.5,135,202.5,180,22.5,45,247.5,225,337.5,0,
292.5,315,270};
int sensorMin[] =
   {240,296,346,411,651,851,991,1301,1651,1901,2301,2451,2701,3056,
3201,3401};
int sensorMax[] =
   {295,345,410,650,850,990,1300,1650,1900,2300,2450,2700,3055,3200,
3400,3700};
```

El sistema mide el voltaje de salida de la veleta utilizando el `ADC` y, mediante un ciclo `for`, recorre los arreglos de valores máximos (`sensorMax[]`) y mínimos (`sensorMin[]`). Si el voltaje medido se encuentra dentro de un rango válido, se obtiene el índice correspondiente en el arreglo de ángulos (`dirDeg[]`), determinando así la dirección del viento.

```
float Veleta(){
    int h = 0;
    float angle;
    lectVel=mcp3208.readADC(1);
    for(h=0; h <= 15; h++) {
        if(lectVel >= sensorMin[h] && lectVel <= sensorMax[h]){
            angle = dirDeg[h];
            break;
        }
    }
    return angle;
}
```

Se juntaron ambos programas y se añadieron instrucciones para imprimir los valores obtenidos en el monitor serial, con el fin de verificar que los datos desplegados fueran correctos. Para la veleta, la validación fue sencilla: solo se debía comprobar que el ángulo mostrado coincidiera con la posición real de la veleta.

Para la validación del anemómetro, se utilizó un modelo analógico que se había probado previamente pero se descartó. Se colocó un ventilador a la misma distancia de ambos anemómetros (el digital y el analógico) y se verificó que giraran a la misma velocidad. Los datos obtenidos del anemómetro digital eran coherentes, aunque se notó que requerían un pequeño *offset* para mostrar una velocidad más precisa.

3.4. Segunda prueba en conjunto

Se agregaron estos componentes al prototipo ya unido y al código de este prototipo se le agregaron los códigos para que los componentes realizaran sus funciones correspondientes. Con el código completo se realizó una segunda prueba de todos los componentes en conjunto.

Con esta prueba se observó que el anemómetro, al tener más líneas de código, se descalibraba y presentaba una velocidad de viento menor a la verdadera. Entonces se tuvo que aumentar el *offset* antes propuesto. Además, se determinó que cada vez que se agregaran muchas más líneas de código, se debía probar el anemómetro y corroborar sus valores.

Otro error que surgió en esta prueba fue en la lectura del anemómetro al activar el compresor. Al encenderse, el compresor generaba interferencias eléctricas que afectaban la señal digital del anemómetro, provocando lecturas erróneas con valores más altos de lo normal.

Este fenómeno se debe a que los motores eléctricos, especialmente los que funcionan con escobillas, generan ruido electromagnético y picos de voltaje debido a la conmutación rápida de corriente en sus bobinas. Es posible que este ruido se propague a través de la alimentación del sistema o por radiación electromagnética, afectando a otros componentes como el microcontrolador y los sensores.

Para reducir este ruido electromagnético, se optó por soldar un capacitor pequeño directamente en las terminales del compresor. Este capacitor actúa como un filtro de alta frecuencia, absorbiendo los picos de ruido generados y reduciendo su impacto en el resto del sistema. Con esta solución, se logró estabilizar las mediciones del anemómetro, evitando lecturas erróneas y garantizando datos más precisos y confiables.

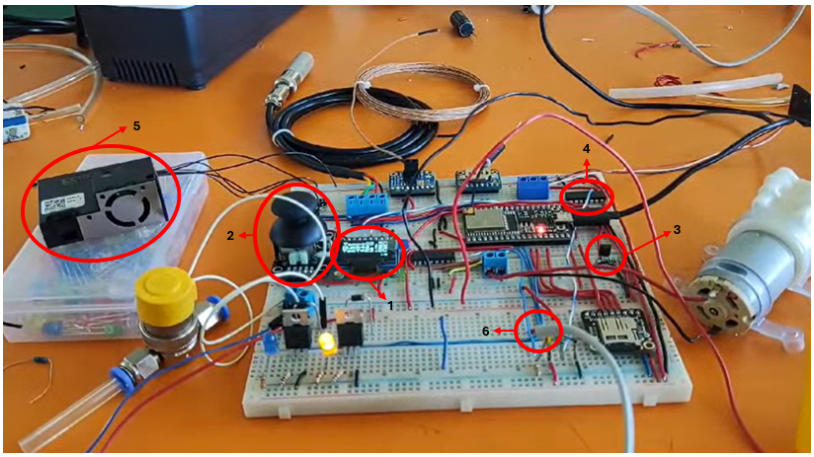


Figura 5.8: Fotografía del prototipo avanzado.

A continuación se describen los elementos enumerados en la Figura 5.8:

1. Pantalla OLED - 128x64.
2. *Joystick* (Modulo *joystick* básico).
3. Referencia de 5[V] (LM4040BIZ-5.0).
4. ADC de 8 canales (MCP3208).
5. Sensor COVs y PM´s (SEN55).
6. Sistema de Viento (SEN-15901).

3.5. Tercera prueba con sensores

3.5.1. Inversor con disparador Schmitt

Este componente fue agregado para mejorar la estabilidad de las señales digitales provenientes del *joystick* y el anemómetro. Si bien ambos funcionaban correctamente, presentaban ciertas irregularidades en la lectura debido al ruido eléctrico y a efectos de rebote.

El *Schmitt Trigger*, al contar con histéresis, permite definir mejor las transiciones entre niveles lógicos, evitando que pequeñas fluctuaciones o ruido generen cambios de

estado erróneos. Esto resultó especialmente útil en dos casos:

- **Anemómetro**:
  - Eliminó los picos repentinos en la medición de velocidad
  - Solucionó interferencias en la señal de pulsos
- ***Joystick***:
  - Redujo el efecto de rebote del botón seleccionador
  - Evitó activaciones múltiples no deseadas

Con esta implementación, se logró una respuesta más estable y precisa en ambos componentes, mejorando la fiabilidad del sistema.

3.5.2. Cambiador de Nivel

Para asegurar la compatibilidad entre los niveles lógicos de los componentes, se implementó un cambiador de nivel lógico que ajusta la salida del *Schmitt Trigger*. Este circuito genera una señal de 5[V] en estado alto, superando el límite de tolerancia de la ESP32 (que opera con 3.3[V]).

El cambiador de nivel realiza la conversión de 5[V] a 3.3[V], garantizando que:

- Las señales sean compatibles con la ESP32
- Se elimine el riesgo de daño al microcontrolador
- Se mantenga la integridad de las señales digitales

Esta modificación permitió que tanto el anemómetro como el *joystick* funcionaran dentro de los parámetros seguros del sistema, manteniendo su precisión y confiabilidad.

3.5.3. Reguladores de voltaje

Los reguladores de voltaje son esenciales para evitar que la ESP32 tenga que alimentar todos los componentes de 5[V] y 3.3[V], ya que no tiene la capacidad de potencia suficiente y, además, es necesario contar con una fuente que alimente al propio microcontrolador. Estos reguladores se conectaron a la fuente de 12[V] y se verificó que suministraran correctamente la energía a todos los componentes, incluyendo la ESP32.

3.5.4. Compuerta *NOT*

Los dos ventiladores pequeños tenían la función específica de encenderse únicamente cuando la bomba estuviera encendida y de manera inversa al estado de las electroválvulas. Para evitar utilizar más pines de la ESP32, se decidió implementar una compuerta NOT que invirtiera la señal de control de las electroválvulas.

Se conectó la compuerta de modo que:

- Su entrada recibiera la señal de control de ambas electroválvulas
- Se obtuvieran dos entradas y dos salidas invertidas

Estas salidas se conectaron a dos MOSFET configurados como interruptores para controlar los ventiladores.

Al probar el sistema, se corroboró que los ventiladores se encendían cuando se enviaba un 0 desde el microcontrolador y se apagaban cuando se enviaba un 1, cumpliendo así con la lógica de funcionamiento deseada.

3.6. Tercera prueba en conjunto

Se agregaron los componentes probados al prototipo existente y se evaluó su funcionamiento en conjunto. Se notó que el *joystick* presentaba un pequeño retraso en su respuesta y que el anemómetro, en ocasiones, seguía registrando valores erróneos y picos elevados en la velocidad del viento. Se consideró que esto podía deberse al uso de *jumpers*, cables y múltiples *protoboards*, lo que probablemente generaba ruido en las señales digitales recibidas por el microcontrolador. Se esperaba que, al integrar todos los componentes en una PCB, la respuesta del anemómetro y del *joystick* mejorara significativamente.

Adicionalmente, en este punto se solicitó medir también el voltaje de entrada del sistema para conocer el estado de las baterías. Para solucionarlo de manera sencilla, se añadió un divisor de voltaje en la entrada, de modo que este redujera los 14[V] máximos (aproximados) a un valor de alrededor de 5[V], que es el rango aceptable para el ADC. Para ello, se seleccionó un valor de R_2 de 10 kΩ y, utilizando la expresión de un divisor de voltaje, se despejó R_1:

$$R_1 = R_2 \times \frac{V_{IN} - V_{OUT}}{V_{OUT}} = 10000 \times \frac{14 - 5}{5} = 18\,\text{k}\Omega \tag{5.4}$$

Con estos valores de resistencia se protege el ADC. Por ejemplo, si el voltaje de entrada es de 12[V], la salida será:

$$V_{OUT} = \frac{10000}{18000 + 10000} \times 12 = 4.28\,\text{V} \tag{5.5}$$

De este modo, se asegura que el valor de entrada no supere los 5[V], protegiendo así el ADC. Para leer este valor:

- Se implementó una función en el ADC que utiliza `Read_MCP3208(int x)`
- El parámetro `x` corresponde al canal 4 del ADC.

Para escalar el valor devuelto por el ADC al rango de 0[V] a 3.3[V]:

$$\text{Factor de escala} = \frac{5}{13} = 0.3846 \tag{5.6}$$

$$V_{ESCALADO} = \frac{V_{ADC}}{0.3846} \tag{5.7}$$

Además se agregó un comando que le asigna el número de error 10 a cualquier fallo en el ADC. Si el ADC y el sistema siguen encendidos, pero el ADC lee un 0 en el voltaje de entrada entonces es posible asegurar que existe un error en las lecturas de este componente.

```
float Voltaje(){
    float volt;
    lectVolt=mcp3208.readADC(4);
    if(lectVolt == 0){
        uint8_t error = 10;
    }
    volt=(((lectVolt*5.0)/4095.0))/0.340599;
    return volt;
}
```

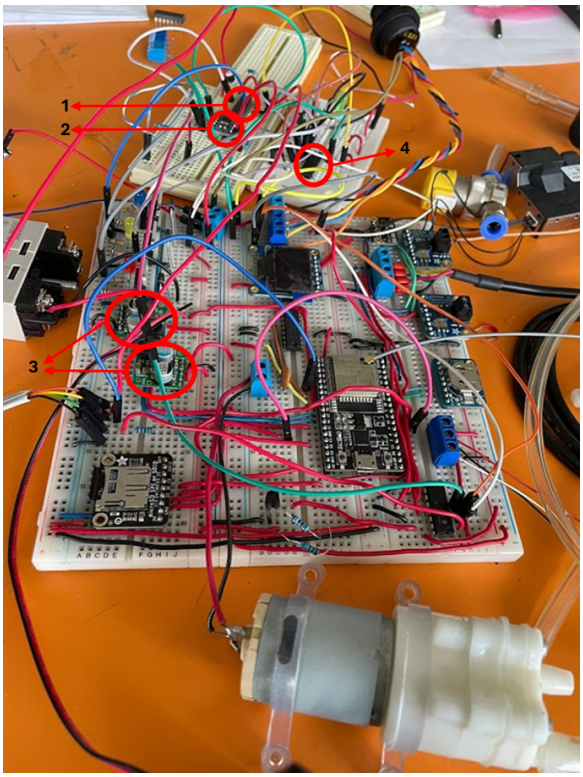


Figura 5.9: Fotografía del prototipo terminado.

A continuación se describen los elementos enumerados en la Figura 5.9:

1. Inversor con disparador *Schmitt* (SN74LS14)
2. Cambiador de Nivel (TXB0104)
3. Reguladores de Voltaje (D24V22F5 y D24V22F3)
4. Compuerta *NOT*) (SN74LS04N).

4. Programación de diferentes secuencias de adquisición de datos y de control de actuadores

En este punto absolutamente todos los componentes habían sido probados y daban o recibían datos en conjunto, por lo que se procedió a desarrollar el software que se necesitaba para que todo funcionara como se requería.

Lo que se hizo fue separar cada sensor o actuador en distintas bibliotecas, de este modo se tiene un mejor acomodo y estructura del código realizado por lo que en el archivo principal se encuentran tanto la función `setup` y `loop` propias del software de Arduino, así como las diferentes funciones desarrolladas para el control requerido, la recopilación de datos y la manipulación de los actuadores.

4.1. Creación y escritura de archivos

Para poder realizar la creación de archivos con los nombres deseados, se usaron algunas combinaciones de funciones propias de C, ya que estos nombres son variables. En un principio, la idea fue crearlos cada que iniciaba una prueba programada la cual consiste en configurar el muestreador para que inicie automáticamente el proceso de colecta en los cartuchos. Esto incluye el encendido de la bomba, el ventilador, el contador de horas y el inicio de la conmutación de válvulas, pero luego cambió a realizarlos por día. Sin embargo, los nombres no dejan de ser variables.

Para esto, se utilizaron funciones que permiten concatenar caracteres y convertir variables enteras o flotantes al tipo `string`. A continuación, se enlistan las funciones usadas y la secuencia que se siguió para poder nombrar correctamente cada archivo.

4.1.1. Funciones usadas

1. **memset(void *s, int c, size_t n)**
   - *a*) Esta función copia el valor de `c` (convertido a `unsigned char`) en cada uno de los primeros n caracteres del objeto apuntado por s.
   - *b*) Se utilizó `memset()` para inicializar el arreglo `Doc`, asegurando que contenga valores predeterminados antes de concatenar o copiar cadenas de texto.
2. **strcpy(char *s1, const char *s2)**
   - *a*) Copia la cadena apuntada por `s2` (incluyendo el carácter nulo) en la cadena apuntada por `s1`.

b) Se usó para establecer el prefijo del nombre del archivo, que comienza con una antidiagonal (/), necesaria para la correcta creación y búsqueda de archivos en el módulo SD.

- `const char* nomDoc = "/Datos";`
- `const char* nomValv = "/Valvulas";`

3. **strcat(char *string1, const char *string2)**

a) Concatena cadenas de texto.

b) Se utilizó para ir construyendo los nombres de los archivos por partes, combinando el prefijo (`nomDoc` o `nomValv`) con la fecha y la extensión del archivo.

- `const char* ext30 = "_30s.csv"; //Para el archivo de 30 seg`
- `const char* ext15 = "_15min.csv"; //Para el archivo de 15 min`
- `const char* ext = ".csv"; //Para el archivo de válvulas`

4. **sprintf(char *cadena, const char *formato, ...)**

a) Permite formatear y almacenar datos en una cadena de caracteres, convirtiendo variables numéricas en texto estructurado.

b) Se utilizó para nombrar archivos según la fecha y para escribir datos en los archivos.

1) **Nombrar archivos:**

- `char* Date = "\_\%i-\%i-\%i"; //Formato: \_Año-Mes-Día`

2) **Escribir en archivos:**

- `char* Fecha = " - \%i/\%i/\%i - \%i:\%i hrs\n"; // Formato: - año/mes/día - hora:minuto}`
- `char* msj30 = "\%i-\%i-\%i \%i:\%i:\%i,\%.2f,\%.2f,\%.2f,\%.2f,\%i,\%i,\%.1f,\%.2f,\%.1f,\%.1f,\%.1f,\%.1f,\%.1f,\%.1f,\%.2f\n"; // Formato: año-dia-mes hora:minuto:seg,temp,humedad,tempT1,tempT2,ICA,flujo,dir viento,vel viento,pm1,pm2.5,pm4,pm10,voc,nox,voltaje`
- `char* msj15 = "\%i-\%i-\%i \%i:\%i:\%i,\%.2f,\%.2f,\%.2f,\%.2f,\%.2f,\%.2f,\%.2f,\%.2f,\%.2f,\%.2f,\%.2f,\%.1f,\%.2f,\%.1f,\%.1f,\%.1f,\%.1f,\%.1f,\%.1f,\%.2f\n"; // Formato: año-dia-mes hora:minuto:seg,temperatura,tempMax,tempMin,humedad,tempT1,tempT2,ICA,flujo,dir viento,vel viento,vel vientoMax,pm1,pm2.5,pm4,pm10,voc,nox,voltaje`

4.1.2. Secuencia usada

Proceso de creación y escritura de archivos:

1. **Inicialización:** Se utiliza `memset()` para limpiar el arreglo `Doc` y asegurarse de que no contenga datos residuales.

```
memset(Doc, 0, sizeof(Doc));
```

2. **Formateo de la fecha:** Con `sprintf()`, se almacena la fecha en un `buffer` (variable auxiliar de tipo `char` usada como arreglo) utilizando el formato `Date`.

```
sprintf(buffer,Date,anio,mes,dia);  //Formato: "_año-mes-dia"
```

3. **Asignación del nombre base:** Se copia el prefijo correspondiente (`nomDoc` o `nomValv`) en el arreglo `Doc` usando `strcpy()`.

```
strcpy(Doc, nom);                   //nom: "/Datos" o "/Valvulas"
```

4. **Concatenación del formato `Date`:** Se emplea `strcat()` para agregar el contenido del `buffer` al arreglo `Doc`, quedando en la forma `nomDoc` o `nomValv` + `Date` con sus respectivas variables.

```
strcat(Doc, buffer);                    //"/Datos_año-mes-día" o
   "/Valvulas_año-mes-día"
```

5. **Concatenación de la extensión del archivo:** Se utiliza `strcat()` nuevamente para agregar la extensión deseada (`ext30`, `ext15` o `ext`), formando así el nombre final del archivo (`nomDoc` o `nomValv` + `Date` + `ext30`, `ext15` o `ext`).

```
strcat(sal, ext);                      //"/Datos_año-mes-día_30s.
   csv" o "/Datos_año-mes-día_15m.csv" o "/Valvulas_año-mes-día.
   csv"
```

6. **Escritura de datos:**

   a) En este punto, el nombre completo del archivo ya está almacenado en el arreglo `Doc`.

   b) Para escribir en el archivo, se utilizan las funciones del módulo SD, específicamente `appendFile()`.

   c) La función recibe tres parámetros: el objeto `SD`, el nombre del archivo (`Doc`) y el mensaje o texto a escribir.

   d) El mensaje puede ser un texto directo entre comillas o generado dinámicamente con `sprintf()`, utilizando los formatos `msj30` o `msj15`, según sea necesario.

```
appendFile(SD, Doc, "Mersaje a escribir");
```

Este proceso, al ser repetitivo, se optimizó mediante la creación de una función llamada `documento()`. Dicha función encapsula los pasos del 1 al 5 descritos anteriormente, permitiendo generar el nombre del archivo de manera más eficiente y evitando redundancias en el código.

La función `documento(int dia, int mes, int anio, const char* nom, const char* ext, char sal[30])` recibe los siguientes parámetros:

- `dia, mes, anio` → Representan la fecha en la que se generará el archivo, permitiendo nombrarlo de manera automática con el día, mes y año correspondientes.
- `nom` → Es el prefijo que define el inicio del nombre del archivo. Toma valores como `nomDoc` para archivos de datos o `nomValv` para archivos de conmutación de válvulas.
- `ext` → Es la extensión del archivo que define su tipo. Siendo `ext30`, `ext15` o `ext`, dependiendo de si se almacenan datos cada 30 segundos, cada 15 minutos o si corresponde a la conmutación de válvulas.
- `sal[30]` → Es un arreglo de caracteres donde se almacenará el nombre completo del archivo resultante, incluyendo el prefijo, la fecha y la extensión.

La función genera el nombre del archivo en sal, concatenando las diferentes partes proporcionadas en los parámetros. Posteriormente, este nombre se utiliza para acceder y escribir en el archivo de manera automática.

```
1 void documento(int dia, int mes, int anio, const char* nom, const
    char* ext, char sal[30]){
2     ...
3 }
```

De este modo, cada vez que se requiera escribir en un archivo, simplemente se llama a la función `documento()` con los parámetros correspondientes para generar el nombre en `Doc`. Posteriormente, solo es necesario llamar a `appendFile(SD, Doc, "Mensaje")` para escribir en el archivo de manera automática.

```
    documento(diaON, mesON, anioON, nomValv, ext, Doc);
    appendFile(SD, Doc, "Valvula A: ON -- Valvula B: OFF");
```

Listing 5.3: Ejemplo de uso de la fución documento() junto al appendFile().

(Ver Anexo: Código 7.3 - Función `documento(int dia, int mes, int anio, const char* nom, const char* ext, char sal[30])`).

4.1.3. Encabezado en archivos

Este procedimiento asegura que los archivos de datos siempre contengan un encabezado con los nombres de las columnas, facilitando la interpretación de la información cuando se abren en Excel u otro software de análisis de datos.

1. **Funcionamiento detallado:**

a) **Verificación de existencia del archivo:**

- Antes de escribir datos en el archivo, se usa una función propia del módulo SD para verificar si ya existe `(!SD.exists(Doc))`.

```
if(!SD.exists(Doc)){
    ...
}
```

- Si el archivo ya está creado, se omite este paso y se continúa con la escritura de datos.

b) **Creación del archivo y escritura de encabezados:**

- Si el archivo no existe, se llama a writeFile(), que recibe los mismos parámetros que appendFile().
- writeFile() crea el archivo y escribe la primera línea con los nombres de las columnas, asegurando que cada dato quede correctamente identificado.
- Este encabezado solo se escribe una vez, evitando repeticiones innecesarias.

```
if(!SD.exists(Doc)){
    writeFile(SD, Doc, "Fecha,Temperatura(C),Humedad(%),
   Temperatura TP_1(C),Temperatura TP_2(C),Calidad Aire,
   Flujo(mL/m),Dir. Viento(Grad),Vel. Viento(m/s),PM 1.0
    ug/m3,PM 2.5 ug/m3,PM 4.0 ug/m3,PM 10.0 ug/m3,VOC,
   NOx,Voltaje Bat(V)\n");
}
```

2. **Ejecución en dos momentos clave:**

a) **Durante el setup() del código:**

- Asegurando que, si el sistema se reinicia o inicia por primera vez, los archivos se creen con su estructura correcta.

b) **Cada vez que se registran datos cada 30 segundos o cada 15 minutos:**

- Verificando si el archivo ya existe antes de escribir.
- Además como el nombre del archivo depende de la fecha, cuando el reloj marque las 00:00 horas, el sistema detectará que la fecha cambió, creando así un nuevo documento.

4.2. Control de válvulas

Para desarrollar esta función, primero se definió qué debía hacer. Retomando los requerimientos del proyecto, se estableció lo siguiente:

- Las válvulas deberán activarse durante 12 horas cuando se requiera, sin encenderse ambas al mismo tiempo.

- Su funcionamiento será intercalado, es decir, al activarse una válvula durante 12 horas, la otra permanecerá apagada y viceversa.
- Esté encendido intercalado deberá sincronizarse con un horario establecido, funcionando como una alarma programada.

Con estos criterios claros, se diseñó la función que permitiría cumplir con estos pasos. Para facilitar las pruebas iniciales, en lugar de trabajar directamente con horas, se hizo un diseño análogo utilizando minutos, lo que permitió evaluar el comportamiento del sistema en menos tiempo.

Adicionalmente, se incorporaron dos criterios más para hacer la función más flexible y adaptable:

- Las válvulas no estarán limitadas a intervalos de 12 horas; en su lugar, el intervalo será configurable.
- La conmutación entre válvulas no ocurrirá sólo una vez, sino que continuará de forma periódica según el intervalo establecido, hasta que la función sea desactivada.

4.2.1. Proceso de conmutación

Al contar con un reloj de tiempo real, se utilizaron los datos que este proporciona. Para la primera prueba, se extrae únicamente el minuto actual y se compara con una simple condición `if`. El microcontrolador evalúa si el minuto actual coincide con el valor almacenado en la variable `minutoValv + a`.

Aquí:

- `minutoValv` representa el minuto en el que se requiere que inicie la conmutación.
- `a` se inicializa en 0 y se actualiza con el intervalo deseado para repetir la acción.

```
if(fecha.minute() == minutoValv + a){
    ...
}
```

Cuando la condición `if` se cumple, se activa una de las válvulas (`Valv A`) y se desactiva la otra (`Valv B`). Luego, la variable `a` se actualiza con la ecuación:

$$\texttt{a} = 2 \times \texttt{interval} + \texttt{a} \tag{5.8}$$

```
    ...
    digitalWrite(electroValA, HIGH);
    digitalWrite(electroValB, LOW);
    a = 2*interv + a;
    ...
```

El factor de multiplicación por 2 se debe a que, antes de volver a activar la misma válvula, se debe activar la otra válvula en el siguiente intervalo.

Para evitar que esta condición se ejecute múltiples veces en el mismo minuto, se incluyó una variable booleana (`valvBool`), la cual se establece en `false` al entrar por primera vez en la función, evitando así nuevas activaciones dentro del mismo minuto.

```
    ...
    if(valvBool == true){     //Condicion para realizar la accion una sola vez y no todos los 60 segundos del minuto
        digitalWrite(electroValA, HIGH);
        digitalWrite(electroValB, LOW);
        a = 2*interv + a;
        valvBool = false;
    }
...
```

Después, el proceso se repite, pero ahora comparando con `minutoValv` + `b`, donde:

```
...
if(fecha.minute() == minutoValv + b){
    ...
}
...
```

- `b` se inicializa con `interval`.
- Se utiliza el mismo booleano, pero con lógica invertida, además ahora se activa `Valv B` y se desactivaba `Valv A`.

```
    ...
    if(valvBool == false){
        digitalWrite(electroValA, LOW);
        digitalWrite(electroValB, HIGH);
        ...
    }
    ...
```

Luego, `b` se actualiza con la ecuación:

$$\texttt{b} = 2 \times \texttt{interval} + \texttt{b} \tag{5.9}$$

y posteriormente se manda e booleano a true.

```
        ...
        b = 2*interv + b;
        valvBool = true;
        ...
```

4.2.2. Ajuste por límite de 60 minutos

Durante las pruebas, al usar LEDs para simular la conmutación, se observó que el sistema dejaba de funcionar correctamente cuando a y b superaban el valor de 60. Esto se debe a que los minutos en el reloj no aumentan indefinidamente, sino que al llegar a 59, reinician en 0.

Para corregir esto, se agregó una nueva variable llamada `horaM`, la cual se calcula de la siguiente manera:
Si `minutoValv + a` o `minutoValv + b` era mayor o igual a 60, entonces:

$$\texttt{horaM} = \texttt{(minutoValv + a) - 60} \tag{5.10}$$

De esta manera, si la suma resulta en 60, `horaM` toma el valor de 0, reiniciando correctamente el conteo de minutos.

Adicionalmente, dentro de esta nueva condición, se reasignan los valores de a y b para mantener la secuencia de conmutación:

- Cuando la comparación era con `minutoValv + a`:

$$\texttt{a} = 2 \times \texttt{interval}, \quad \texttt{b} = \texttt{interval} \tag{5.11}$$

- Cuando la comparación era con `minutoValv + b`:

$$\texttt{b} = 2 \times \texttt{interval}, \quad \texttt{a} = \texttt{interval} \tag{5.12}$$

También asignamos el valor de `horaM` a `minutoValv` para que siguiera con la secuencia deseada.

Finalmente, en lugar de dos sentencias `if` separadas para cada válvula, ahora se implementaron dos estructuras `if-else` para manejar correctamente la lógica normal y el caso especial cuando se superan los 60 minutos.

Al volver a hacer pruebas se verificó que la lógica es correcta y funciona bien. Esta función se llamó `ControlValvulasMin()`, por lo que se procedió a realizar el ajuste correspondiente para que sea por horas.

(Ver Anexo: Código 1.2.1 - `ControlValvulasMin()`)

4.2.3. Ajuste para que la conmutación sea por horas

Para este ajuste se creó otra función llamada `ControlValvulasHor()`, se modificó la comparación para trabajar con horas en lugar de minutos. Ahora, en lugar de extraer solo los minutos del reloj de tiempo real, también se tomó el valor de la hora. De este modo, la conmutación de las válvulas ocurre cuando ambas condiciones se cumplen simultáneamente, es decir, cuando la hora del reloj coincide con `horaValv`

`+ a` y el minuto con `minutoValv`, utilizando una condición lógica `AND`.

En esta nueva estructura, el caso especial se presenta cuando el contador a alcanza o supera las 24 horas. Para manejarlo, simplemente se reemplaza el límite de 60 (utilizado en la versión por minutos) por 24, manteniendo el resto de la lógica sin cambios.

Además, se agregó una instrucción para registrar cada conmutación en un archivo llamado `"Valvulas"` seguido de la fecha y minuto, las cuales corresponden al momento en que se creó el archivo, utilizando el módulo `SD`. Esta función se implementó siguiendo la estructura explicada en la sección 4.1. Lo único que cambia es que se agregó un nuevo formato llamado `fechaValv`:

$$\text{char* fechaValv} = \text{\%i/ \%i/ \%i - \%i: \%i hrs\textbackslash n";} \tag{5.13}$$

El cual contiene el momento en el que se conmutó la válvula y luego se escribe en el archivo creado. Este registro permitió verificar el funcionamiento del sistema sin necesidad de un monitoreo constante durante varias horas.

(Ver Anexo: Código 1.2.2 - `ControlValvulasHor()`).

4.3. Control programado de pruebas

Para gestionar las pruebas programadas, se desarrollaron funciones que verifican cuándo se alcanza la hora de encendido y, una vez activadas, determinan el momento adecuado para apagarse automáticamente.

4.3.1. Función para el apagado automático

El objetivo principal del sistema es que, al configurar una prueba programada, todo el proceso sea completamente automático, sin necesidad de establecer manualmente la hora de apagado. Para ello, se diseñó una función llamada `OffAutomatico()`, que calcula la hora de apagado a partir de la hora de inicio y el intervalo de conmutación de las válvulas, ya que, en este tipo de pruebas, la conmutación solo ocurre una vez.

```
1 void OffAutomatico(){
2     ...
3 }
```

Para implementar esta lógica, se definieron las variables `horaOFF` y `minOFF`, las cuales almacenan la hora y el minuto exactos en los que debe finalizar la prueba programada. Dado que la duración máxima de estas pruebas es de 24 horas, únicamente se considera la hora y el minuto, evitando comparaciones adicionales con datos como el día o el año.

Como datos de entrada, esta función utiliza `horaON` y `minON`, que son valores configurados manualmente a través de la interfaz *Bluetooth* o la interfaz gráfica, los

cuales indican la hora y el minuto específicos en los que se encenderá la prueba programada. Además, se creó la variable `TPruebaHor`, que define la duración de la prueba programada en función del intervalo de conmutación (`interv`). Para mayor flexibilidad, se asignó:

$$\texttt{TPruebaHor} = 2 \times \texttt{interv} \tag{5.14}$$

Asegurando que el sistema pueda ajustarse a diferentes configuraciones de tiempo.

También se añadió la variable `TPruebaMin`, pensada para casos en los que sea necesario especificar el tiempo en minutos. No obstante, para este proyecto, `TPruebaMin` permanece fija con un valor inicial de 59 y no se modifica posteriormente.

```
// Valores por defecto
TPruebaHor = 24;
TPruebaMin = 59;

// Valores cuando se programa por Bluetooth
TPruebaHor = 2*(Dato5.toInt()); //Dato5.toInt() corresponde al
   intervalo
TPruebaMin = 59;
```

De esta manera, la función asigna los valores correctos de `horaOFF` y `minOFF` en función de `horaON`, `minON` y `TPruebaHor`. Para calcular la hora de apagado, simplemente se suma la hora de inicio (`horaON`) con el tiempo de prueba (`TPruebaHor`) y se asigna el resultado a `horaOFF`.

```
2     ...
3     horaOFF = horaON + TPruebaHor;
4     ...
```

Dado que el formato horario utilizado va de 0 a 23, se agregó una condición lógica que verifica si la suma es mayor o igual a 24. Si esto ocurre, se resta 24 al resultado para asegurar que el valor se mantenga dentro del rango válido de horas.

```
2     ...
3     horaOFF = horaON + TPruebaHor;
4     if(horaOFF >= 24){
5         horaOFF = horaOFF-24;
6     }
7     ...
```

Para el cálculo del minuto de apagado (`minOFF`), se siguió una lógica similar. Se compara el valor resultante con 60 y se hacen los ajustes necesarios. En este caso, `TPruebaMin` se establece con un valor de 59 para que la prueba termine un minuto antes. Esto evita que el sistema encienda y apague la prueba en el mismo instante, ya que no se compara la fecha completa (día, mes, año), sino solo la hora y el minuto actuales.

```
7     ...
8     minOFF = minON + TPruebaMin;
9     if(minOFF >= 60){
10        minOFF = minOFF - 60;
11    }
12    ...
```

Tras realizar diversas pruebas con valores arbitrarios de entrada, se verificó que la función actualiza correctamente la hora de apagado en todos los casos.

(Ver Anexo: Código 8.2.1 - Funcion `OffAutomatico()`).

4.3.2. Función para encender una prueba programada

Para gestionar el inicio de una prueba programada, se diseñó la función `BombaON()`, cuya tarea es verificar si se ha alcanzado la fecha y hora establecidas por el usuario, ya sea a través de la interfaz *Bluetooth* o mediante la interfaz gráfica.

```
1  void BombaON(){
2      ...
3  }
```

Para evitar que la función evalúe constantemente todas las condiciones de inicio, se introdujo un booleano llamado `ContrBomb`, el cual se establece en `true` una vez que el usuario ha configurado los parámetros de la prueba. Luego, se obtiene la hora y fecha actual a través del reloj de tiempo real y se comparan estos valores (`diaON`, `mesON`, `horaON` y `minON`) con los definidos por el usuario mediante una condición lógica AND dentro de un `if`. De esta manera, la función solo se ejecuta cuando todas las condiciones de tiempo coinciden exactamente.

```
2      ...
3      if(ContrBomb == true){
4          DateTime fecha = rtc.now();
5          if(fecha.day() == diaON && fecha.month() == mesON && fecha.
       hour() == horaON && fecha.minute() == minON){
6              if(ContrBombBool == true){ //Para evitar ingresar mas de
       una vez
7                  ....
8              }
9          }
10     }
11     ...
```

Cuando esto ocurre, significa que ha llegado el momento de iniciar la prueba programada. En consecuencia, se activan las funciones necesarias, como la conmutación de válvulas (`ContrValv` y `valvBool`, ambas establecidas en `true`), y se encienden los pines correspondientes al ventilador y al contador de horas (siendo este el mismo).

```
7                  ...
8                  ContrValv = true;
9                  valvBool = true;
10                 digitalWrite (vent, HIGH);
11                 ...
```

Posteriormente, se llama a las funciones `EncenderBom()` y `OffAutomatico()` previamente implementadas para garantizar el correcto funcionamiento del sistema. Adicionalmente, se envían mensajes al monitor serial únicamente para verificar que se haya entrado a esta función correctamente y se hayan establecido los parámetros correctamente, esencialmente los de apagado.

```
                ...
                EncenderBom();
                OffAutomatico();

                Serial.println(" ");
                Serial.println("Comienza Prueba");
                if(TPruebaHor != 24){
                    Serial.print("La prueba se detendrá a las: ");
  Serial.print(horaOFF); Serial.print(":");
                    Serial.print(minOFF); Serial.println(" hrs");
                    Serial.println(" ");
                }
                else{
                    Serial.print("La prueba se detendrá mañana a las
  : "); Serial.print(horaOFF); Serial.print(":");
                    Serial.print(minOFF); Serial.println(" hrs");
                    Serial.println(" ");
                }
                ...
```

Cabe resaltar que, al igual que en las funciones anteriores donde se compara la hora, se agregó también un booleano para evitar entrar más de una vez, llamado `ContrBombBool`. Este inicia en `true` y, al entrar a la función, se cambia a `false` al igual que `ContrBomb` para saber que ya no hay una prueba programada en curso. También se establecen en `true` los booleanos `ContrOFF` y `ContrOFFBool`, que se añadieron básicamente para saber en qué momento entrar a la función para apagar la prueba, ya que, mientras no se encienda, tampoco se verificará si se ha alcanzado la hora de apagado.

```
                ...
                ContrBombBool = false;
                ContrBomb = false;

                ContrOFF = true;
                ContrOFFBool = true;
```

(Ver Anexo: Código 1.6.1 - Función `BombaON()`).

4.3.3. Función para apagar una prueba programada

Se desarrolló la función `BombaFF()` para verificar automáticamente si se ha llegado a la hora configurada para el apagado de la prueba. Para que esta función sea ejecutada, el booleano `ContrOFF` debe estar en estado `true`, lo que indica que una prueba programada ya ha comenzado. En este punto, la función `BombaFF()` comprueba si es el momento de apagarla.

```
void BombaOFF(){
    if(ContrOFF == true){
        ...
    }
}
```

El proceso de apagado se basa en comparar la hora actual con los valores de hora (`horaOFF`) y minuto (`minOFF`) definidos en la función `OffAutomatico()`. Esto se logra mediante una condición lógica AND dentro de un `if`, que verifica si ambas condiciones se cumplen. Además, se utiliza el booleano `ContrOFFBool` para asegurarse de que las instrucciones de apagado se ejecuten solo una vez y no durante todo el minuto.

```
...
DateTime fecha = rtc.now();
if(fecha.hour() == horaOFF && fecha.minute() == minOFF){
    if(ContrOFFBool == true){
        ...
    }
}
...
```

Cuando la condición lógica se cumple, el booleano `ContrOFFBool` se establece en `false` para evitar ejecuciones repetidas. También se ponen en `false` los booleanos `ContrValv` y `ContrBomb`, que controlan la válvula y la bomba, respectivamente. A continuación, se llama a la función `ParарBom()` para detener la bomba. Además, se restablece el valor del *Setpoint* a 70, que es el flujo requerido, el cual se explica mejor en la sección correspondiente al PID - 6, para asegurar tambien se mandan a `LOW` las electroválvulas y ventilador.

```
            ...
            ContrOFFBool = false;
            ContrValv = false;
            ContrBomb = false;
            ParарBom();
            Setpoint = 70;
            digitalWrite(electroValA, LOW);
            digitalWrite(electroValB, LOW);
            digitalWrite(vent, LOW);
            ...
```

Finalmente, para verificar que la prueba se haya apagado correctamente, se envía un mensaje al monitor serial.

```
            ...
            Serial.println(" ");
            Serial.println("Prueba detenida");
            Serial.println(" ");
            ...
```

(Ver Anexo: Código 1.6.2 - Función `BombaOFF()`).

4.4. Obtención de datos en tiempo requerido

La obtención de datos se realiza principalmente cuando es necesario que los sensores actualicen sus valores y en momentos específicos en los que estos deben guardarse en los archivos correspondientes. Esto incluye tanto los datos instantáneos, que se registran cada 30 segundos, como los promedios, que se calculan y almacenan cada

15 minutos. Además, los datos se actualizan en la pantalla y se envían al celular mediante *Bluetooth*.

Para facilitar este proceso, se crearon varias funciones que se encargan de realizar las tareas requeridas. Estas funciones hacen uso de otras funciones específicas que fueron desarrolladas para cada sensor, lo que permite un manejo eficiente y organizado de los datos.

4.4.1. Datos cada 5 segundos.

Los datos obtenidos por cada uno de los sensores no forman parte del control de otros componentes del sistema (a excepción del sensor de flujo), por lo que la adquisición de estos no necesita ser extremadamente rápida. Además, el usuario no monitorea estos datos en todo momento. Por esta razón, se decidió que, al menos para los datos mostrados en la pantalla y enviados al celular mediante *Bluetooth*, la actualización del sensado se realice cada 5 segundos.

Para lograr esta actualización periódica de datos, se creó una función llamada `Datos5seg()`.

```
void Datos5seg(){
    ...
}
```

En esta función, se asignan a cada variable el dato leído por cada sensor, utilizando las funciones específicas que se desarrollaron para la lectura de cada uno. De este modo, las nuevas variables contendrán los valores actualizados medidos por cada sensor.

```
    ...
    T=SHT30_ReadT(); H=SHT30_ReadH(); TT_1=MCP9600_1_ReadT(); TTA_1=
    MCP9600_1_ReadTAmb(); TT_2=MCP9600_2_ReadT(); TTA_2=
    MCP9600_2_ReadTAmb(); CA=Read_SGP40(); DirVel=Veleta(); LectSEN55
    ();PM1p0=sen5T[0];PM2p5=sen5T[1];PM4p0=sen5T[2];PM10p0=sen5T[3];
    voc=sen5T[4];nox=sen5T[5]; VB=Voltaje();
    ...
```

Listing 5.4: Asignación de lecturas a variables fijas.

De esta manera, se asegura que, durante el tiempo que transcurre hasta que la función `Datos5seg()` es llamada nuevamente, las variables mantendrán el valor del último dato sensado.

Posteriormente, utilizando la función `sprintf()` (explicada anteriormente en la sección 4.1), se concatenan todos estos valores en un formato de tipo `char*` llamado `BT_Data`. Este formato separa cada variable por comas, generando así un solo renglón de datos donde cada valor está claramente delimitado.

```
char* BT_Data = "%.2f,%.2f,%.2f,%.2f,%i,%i,%.2f,%.1f,%.2f,%.1f,%.1f
    ,%.1f,%.1f,%.1f,%.1f,";
```

Para actualizar las variables en el celular, basta con llamar al objeto de *Bluetooth*, llamado `BT_VOC`, seguido de la función `printf()` y la variable que contiene los datos a enviar. En este caso, según el `sprintf()`, la variable es el buffer, por lo que se envían todos los datos concatenados y separados por comas. Para el caso de la pantalla, esta se refresca siempre, por lo que al cambiar un valor, inmediatamente se observa el cambio de medición.

```
4     ...
5     snprintf(buffer,150,BT_Data,T,H,TT_1,TT_2,CA,F,VB,DirVel,windSpd
      ,PM1p0,PM2p5,PM4p0,PM10p0,voc,nox);
6     BT_VOC.print(buffer);
7     ...
```

Se repite el mismo proceso, pero ahora utilizando las variables de promedio, las cuales almacenan los últimos promedios calculados. Sin embargo, al iniciar por primera vez, todas estas variables tienen un valor de 0 hasta que se realiza la primera toma de promedios (lo cual se explica más adelante). Estos datos se envían en un formato similar, llamado `BT_Data_prom`, que también es de tipo `char*`. La diferencia radica en que, en este último formato, todos los datos se redondean a dos decimales, mientras que en el formato anterior (`BT_Data`), algunos valores tienen solo un decimal o son enteros.

```
char* BT_Data_prom = "%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f
    ,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f";
```

```
7     snprintf(buffer,150,BT_Data_prom,promT,promH,promTT_1,promTT_2,
      promCA,promF,promVB,promDirVel,promwindSpd,prom1p0,prom2p5,
      prom4p0,prom10p0,promvoc,promnox);
8     BT_VOC.print(buffer);
```

(Ver Anexo: Código 1.3.1 - Función `Datos5Seg()`).

Para asegurar que la función se ejecute cada 5 segundos, se implementó una lógica que utiliza el reloj en tiempo real (RTC). Esta función llamada `Ctr5seg()` compara el valor de los segundos actuales con los segundos deseados (0, 5, 10, 15, ..., 55) mediante una condición `if` combinada con sentencias `OR`. Cada vez que se alcanza uno de estos valores, se verifica si la variable booleana `Ctr5SegBool` está en `true`. Esto garantiza que la acción solo se realice una vez y no durante todo el intervalo de tiempo que dura el segundo.

```
1 void Ctr5Seg(){
2     DateTime fecha = rtc.now();
3     if(fecha.second() == 1 || fecha.second() == 6 || fecha.second()
      == 11 || fecha.second() == 16 || fecha.second() == 21 || fecha.
      second() == 26 || fecha.second() == 31 || fecha.second() == 36 ||
       fecha.second() == 41 || fecha.second() == 46 || fecha.second()
      == 51 || fecha.second() == 56){
4         if(Ctr5SegBool==true){
5         ...
6         }
7     }
```

```
    ...
}
```

Cuando se cumple la condición, se llama a la función `Datos5seg()` (explicada anteriormente). Una vez que esta función termina, la variable `Ctr5SegBool` se establece en `false`.

```
        ...
        Ctr5SegBool=false;
        Datos5seg();
        ...
```

Para reactivar la variable y permitir que el proceso se repita en el siguiente intervalo de 5 segundos, se utiliza otra condición `if` similar, pero con un desfase de 1 segundo (es decir, 1, 6, 11, 16, ..., 56). De este modo, cuando se alcanza el siguiente múltiplo de 5, el proceso se reinicia según lo planeado.

```
    ...
    if(fecha.second() == 2 || fecha.second() == 7 || fecha.second()
   == 12 || fecha.second() == 17 || fecha.second() == 22 || fecha.
   second() == 27 || fecha.second() == 32 || fecha.second() == 37 ||
    fecha.second() == 42 || fecha.second() == 47 || fecha.second()
   == 52 || fecha.second() == 57){
        Ctr5SegBool=true;
    }
    ...
```

(Ver Anexo: Código 1.4.3 - Función `Ctr5Seg()`).

4.4.2. Datos cada 30 segundos.

Los datos recopilados cada 30 segundos no están destinados a ser mostrados en tiempo real a los usuarios. En su lugar, se utilizan para dos propósitos principales:

1. Generar archivos diarios que contienen los datos recopilados por el muestreador.
2. Calcular promedios cada 15 minutos para obtener valores representativos y consistentes.

Se eligió una resolución mínima de 30 segundos para la variable de tiempo por las siguientes razones:

- Permite capturar la fluctuación del flujo de aire generada por el compresor, lo que facilita la identificación de errores puntuales y el momento exacto en que ocurren.
- Enriquece los promedios de 15 minutos, asegurando que los datos sean precisos y representativos.

Esta resolución garantiza un equilibrio entre la detección de anomalías y la generación de promedios confiables.

En la función `Datos30seg()`, lo primero que se realiza es actualizar los valores del objeto `fecha`, que es de tipo `DateTime`. Esto asegura que las variables de tiempo estén actualizadas con el momento exacto en que se ingresó a la función. Para lograrlo, se utiliza la función `rtc.now()`, y el objeto `fecha` se iguala al valor devuelto por esta función.

```
1 void Datos30seg(){
2     DateTime fecha = rtc.now();
3     ...
4 }
```

En esta función, no se realiza una llamada directa a los sensores para actualizar los datos. En su lugar, se aprovechan las variables ya actualizadas en la función `Datos5seg()`. Estos valores se acumulan en variables sumadoras, que van agregando cada uno de los valores obtenidos cada 30 segundos.

Estas sumas se utilizan para calcular los promedios cada 15 minutos, por lo que se agregó un contador que solo registre cuántas veces se entró a esta función para realizar el promedio de forma correcta.

```
3     ...
4     acumT+=T; acumH+=H; acumTT_1+=TT_1; acumTTA_1+=TTA_1; acumTT_2+=
      TT_2; acumTTA_2+=TTA_2; acumCA+=CA; acumWSpd+=windSpd, acum1p0+=
      PM1p0; acum2p5+=PM2p5; acum4p0+=PM4p0; acum10p0+=PM10p0; acumvoc
      +=voc;acumnox+=nox; acumVB+=VB;
5     contP++;
6     ...
```

Se codificaron dos funciones específicas para calcular:

- Los valores máximo y mínimo de la temperatura.
- El valor máximo de la velocidad del viento.

Estas funciones comparan el valor obtenido cada 30 segundos con el valor almacenado en una variable. Si el nuevo valor es mayor (en el caso de los máximos) o menor (en el caso de los mínimos), se actualiza la variable con este nuevo valor. De lo contrario, la variable permanece sin cambios.

```
6     ...
7     TMax=tempmax(T); TMin=tempmin(T); windSpdmax=VVmax(windSpd);
8     ...
```

De esta manera, cada 15 minutos, solo es necesario acceder a estas variables para obtener:

- La temperatura máxima registrada.
- La temperatura mínima registrada.
- La velocidad máxima del viento registrada.

Este enfoque optimiza el proceso de recopilación de datos y garantiza que los valores extremos se capturen de manera eficiente.

(Ver Anexo: Código 1.3.4 - Funciones `tempmax(float temp)`, `tempmin(float temp)`, `VVmax(float vv)`).

Dado que la variable de dirección del viento se maneja en grados, no es posible calcular su promedio de la misma manera que con otras variables. Para resolver esto, se implementó una función especial que recibe el valor de la dirección del viento y, mediante condicionales, determina su seno y coseno.

Los valores de seno y coseno se almacenan en un arreglo, por lo que esta función devuelve el índice correspondiente al valor numérico de seno y coseno de la dirección dada.

Este enfoque no realiza cálculos en tiempo real de seno y coseno por dos razones principales:

1. La veleta que mide la dirección del viento tiene un número limitado de valores posibles, a los cuales les es posible asignar valores precalculados de seno y coseno.

2. Se ahorra tiempo de procesamiento al evitar cálculos repetitivos en el programa.

Los índices de los valores de seno y coseno obtenidos se almacenan en un arreglo, cuyo índice aumenta cada vez que se ejecuta esta función, utilizando un contador específico para este propósito.

```
8     ...
9     SenCos(DirVel);
10    IndicesSen[contP-1]=indSen;
11    IndicesCos[contP-1]=indCos;
12    ...
```

(Ver Anexo: Código 1.3.4 - Función `SenCos(float angulo)`).

Posteriormente, utilizando las funciones previamente descritas en la sección 4.1, se escriben estos valores de 30 segundos en el archivo con la fecha y hora correspondientes.

Finalmente, se verifica si el compresor está encendido y si hay una prueba en funcionamiento. Si esta condición es verdadera, se realiza el mismo proceso de acumulación y conteo que se aplica a las demás variables, pero exclusivamente para la variable *flujo*. En caso de que la bomba esté apagada, no se realiza ningún cambio y el valor del flujo se mantiene en 0.

```
20    ...
21    if(EstadoBom==true && ContrBomb==true){
22        acumF+=F;
23        contF++;
```

```
24      }
25      ...
```

(Ver Anexo: Código 1.3.2 - Función `Datos30seg()`).

Adicionalmente, se creó una función similar a `Ctr5seg()`, pero en este caso se le nombró `Ctr30seg()`. La diferencia radica en que ahora se compara el valor de los segundos actuales con los segundos deseados (3 y 33, para tener un pequeño desfase y evitar coincidencias con `Ctr5seg()`).

La variable booleana utilizada en esta función se llama `Ctr30SegBool`. Cuando se cumple la condición, se llama a la función `Datos30seg()`, explicada previamente, y se establece en `false` el booleano `Ctr30SegBool` para que no se ejecute más de una vez durante ese segundo. Posteriormente, se establece en `true` en los segundos 4 y 34, lo que permite reiniciar el proceso para el siguiente ciclo.

(Ver Anexo: Código 1.4.4 - Función `Ctr30Seg()`.

4.4.3. Promedios cada 15 minutos y conexión a Internet

De manera similar a los datos de 30 segundos, los datos de 15 minutos tienen los siguientes propósitos:

- Generar archivos diarios que contengan los promedios recopilados por el muestreador.
- Actualizar cada 15 minutos los promedios para monitorear, ya sea por *Bluetooth* o por Internet, los datos de cada sensor y revisar su comportamiento durante estos intervalos de tiempo.

Primeramente, se llama a la función del reloj en tiempo real (RTC) para obtener la hora exacta en que se ejecuta esta función, la cual se denominó `Prom15Min()`. Posteriormente, utilizando los datos recopilados cada 30 segundos, se accede a las variables acumuladoras de cada sensor. Estas variables contienen la suma de todos los datos obtenidos durante este período, junto con un contador general que registra el número de muestras. Con estos datos, se calcula el promedio de cada variable dividiendo el valor del acumulador entre el contador. Los resultados se almacenan en variables cuyo prefijo es `prom`.

```
1 void Prom15Min(){
2       DateTime fecha = rtc.now();
3       promT=acumT/contP, promH=acumH/contP, promTT_1=acumTT_1/contP,
     promTTA_1=acumTTA_1/contP, promTT_2=acumTT_2/contP, promTTA_2=
    acumTTA_2/contP, promCA=acumCA/contP, promwindSpd=acumWSpd/contP,
     prom1p0=acum1p0/contP, prom2p5=acum2p5/contP, prom4p0=acum4p0/
    contP, prom10p0=acum10p0/contP, promvoc=acumvoc/contP, promnox=
    acumnox/contP, promVB=acumVB/contP;
4       ...
5 }
```

Una vez que todos los promedios están listos, se procede a escribir en el archivo correspondiente utilizando las funciones explicadas en la sección 4.1. Para ello, solo se actualizan los parámetros de fecha y hora y se escriben en el archivo los nuevos valores de las variables actualizadas.

1. **Cálculo del promedio de la dirección del viento**

   Para obtener el promedio de la dirección del viento, se utiliza el arreglo en donde se almacenaron los índices de los valores de seno y coseno de cada uno de los valores obtenidos. Estos índices se asignan a los arreglos con los valores de seno y coseno mediante un ciclo `for`, en cada iteración y cada valor obtenido se suma en una variable que almacena la suma total de los senos. De manera similar, otra variable almacena la suma total de los cosenos. Por último el valor obtenido en descomposición se vuelve a convertir a ángulo mediante el arco tangente, si este valor es menor a 0, al resultado se le suma 360 grados.

```
4     ...
5     float SumSen=0,SumCos=0,convGrad=0;
6     for(int g=0; g < contP; g++){
7         SumSen = SumSen + senDeg[IndicesSen[g]];
8         SumCos = SumCos + cosDeg[IndicesCos[g]];
9     }
10    convGrad= (180.0*atan(SumSen/SumCos))/PI;
11    if(convGrad<0){
12        promDirVel=convGrad+360;
13    }else{
14        promDirVel=convGrad;
15    }
16    ...
```

2. **Implementación de la conexión *Wi-Fi***

   En este punto, también se agregó la funcionalidad de *Wi-Fi*. Para evitar saturar el servidor con una gran cantidad de datos, se decidió almacenar únicamente los promedios calculados cada 15 minutos. Esto permite monitorear el sistema de manera remota de forma eficiente.

   Para guardar los datos en un servidor, se podrían crear servidores propios; sin embargo, existen varias compañías que ofrecen servicios que facilitan esta tarea. Estas plataformas permiten almacenar y visualizar datos en tiempo real mediante el pago de una licencia, y su implementación en microcontroladores es sencilla. Algunas de las opciones más populares son *Adafruit IO, ThingsBoard y ThingSpeak*. En este proyecto, se optó por utilizar *ThingSpeak* [26]., ya que, al

[26] *ThingSpeak* es una plataforma de análisis para el Internet de las Cosas (IoT) que permite a los usuarios recopilar, visualizar y analizar flujos de datos en tiempo real desde dispositivos conectados, actúa como un servicio en la nube que facilita la gestión de datos IoT, permitiendo a los usuarios crear aplicaciones y visualizar datos sin necesidad de configurar servidores o desarrollar software web propios. https://thingspeak.mathworks.com/

estar basado en MATLAB, ofrece herramientas avanzadas para analizar, descargar y visualizar datos. Se adquirió una licencia para utilizar su servidor sin muchas restricciones. Además, *ThingSpeak* cuenta con una biblioteca fácil de implementar en la ESP32, la cual utiliza el protocolo MQTT para la comunicación.

a) **Configuración del *Wi-Fi* y conexión a *ThingSpeak***

Para hacer uso del *Wi-Fi*, se requieren dos bibliotecas: `WiFi.h` y `ThingSpeak.h`. Para conectarse a Internet, basta con proporcionar el nombre de la red (`ssid`) y la contraseña (`password`), los cuales se almacenan en variables de tipo `const char*`.

```
const char* ssid="NAME";
const char* password="*******";
```

Para conectarse al servidor de *ThingSpeak*, se debe conocer el `channelID` (de tipo `unsigned long`) y la llave de escritura del canal, denominada `WriteAPIKey` (de tipo `const char*`).

```
unsigned long channelID = #######;
const char* WriteAPIKey ="XXXXXXXXXXXXXXX";
```

A continuación, se define el tipo de conexión *Wi-Fi* que se utilizará, el cual será de tipo cliente.

```
/*Definimos el cliente WiFi que usaremos*/
WiFiClient cliente;
```

En la función `setup()`, basta con llamar a la función que inicia la conexión *Wi-Fi*, proporcionando como parámetros el nombre de la red (`ssid`) y la contraseña (`password`). Una vez que la conexión se establece, se despliegan las características de la red a la que se conectó.

```
1  WiFi.begin(ssid,password);
2  while(WiFi.status() != WL_CONNECTED){
3  }
4  /*Una vez conectado, se imprimirá una frase y se iniciará la
       conexión a la Plataforma usando el cliente definido
     anteriormente*/
5  Serial.println("-----------------------------------------");
6  Serial.println("¡Conectado al WiFi");
7  Serial.print("Current ESP32 IP: ");
8  Serial.println(WiFi.localIP());
9  Serial.print("Gateway (router) IP: ");
10 Serial.println(WiFi.gatewayIP());
11 Serial.print("Subnet Mask: " );
12 Serial.println(WiFi.subnetMask());
13 Serial.print("Primary DNS: ");
14 Serial.println(WiFi.dnsIP(0));
15 Serial.print("Secondary DNS: ");
16 Serial.println(WiFi.dnsIP(1));
17 Serial.println("-----------------------------------------");
18 ThingSpeak.begin(cliente);
19 ...
```

Finalmente, solo es necesario iniciar el servidor de ThingSpeak utilizando la función correspondiente.

```
...
ThingSpeak.begin(cliente);
```

De este modo, la conexión *Wi-Fi* y el servidor de ThingSpeak quedan listos para recibir datos.

b) **Envío de datos a ThingSpeak**

Dentro de la función `Prom15Min()`, después de escribir los datos en el archivo correspondiente, se llama a la función `ThingSpeak.setField(field, var)`. Esta función recibe dos parámetros:

- `field`: representa la posición en el servidor donde se almacenará la variable (ThingSpeak cuenta con 8 posiciones o *fields*).
- `var`: la variable que se desea enviar al servidor.

Posteriormente, se llama a `ThingSpeak.writeFields(ID, APIKey)`, la cual recibe el ID del canal y la clave de escritura (`APIKey`). De esta manera, todos los datos asignados mediante `setField` se suben directamente al servidor a través de Internet.

```
    ...
    ThingSpeak.setField(1,promT);ThingSpeak.setField(2,promH
   );ThingSpeak.setField(3,prom2p5);ThingSpeak.setField(4,
   promvoc);ThingSpeak.setField(5,promnox);ThingSpeak.
   setField(6,promF);ThingSpeak.setField(7,promDirVel);
   ThingSpeak.setField(8,promwindSpd);
    ThingSpeak.writeFields(channelID,WriteAPIKey);
    ...
```

c) **Visualización de panel en ThingSpeak**

Una vez que todo está configurado, basta con abrir un canal en la plataforma de ThingSpeak desde la cuenta con licencia, actualizar los nombres de los *fields* según los parámetros que se desean visualizar y guardar el canal.

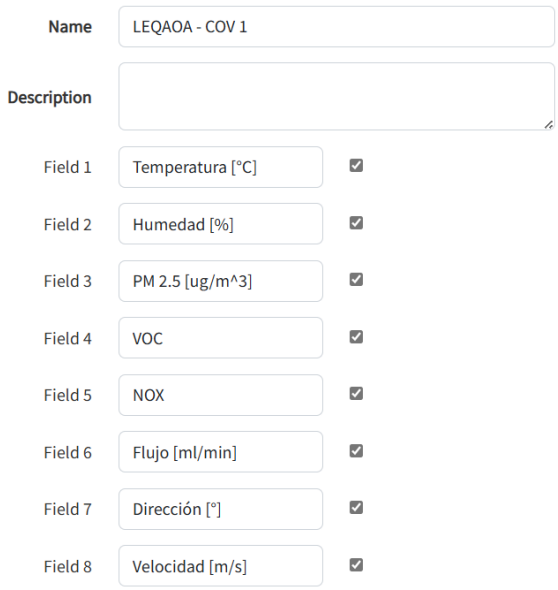
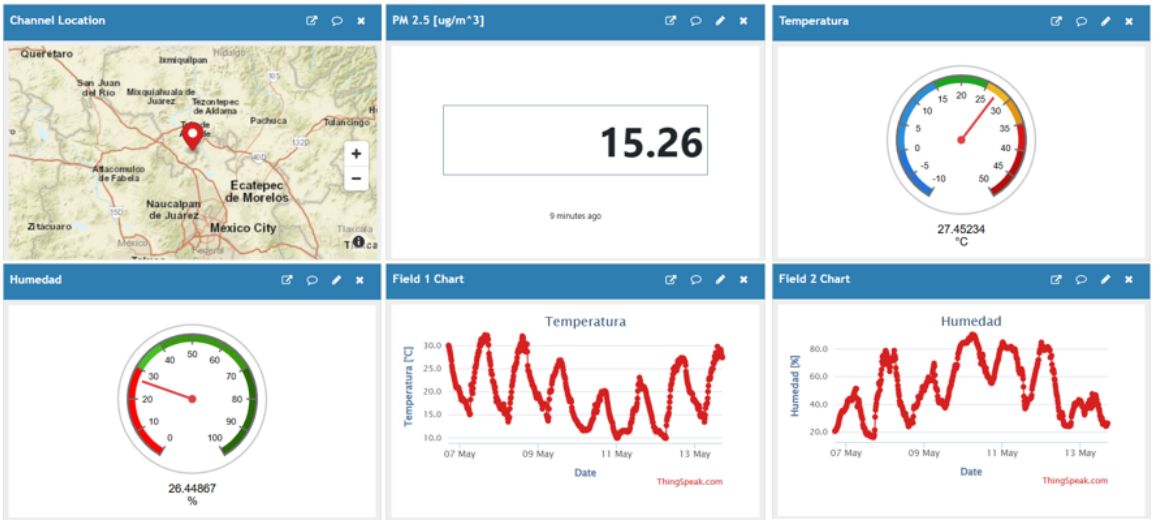
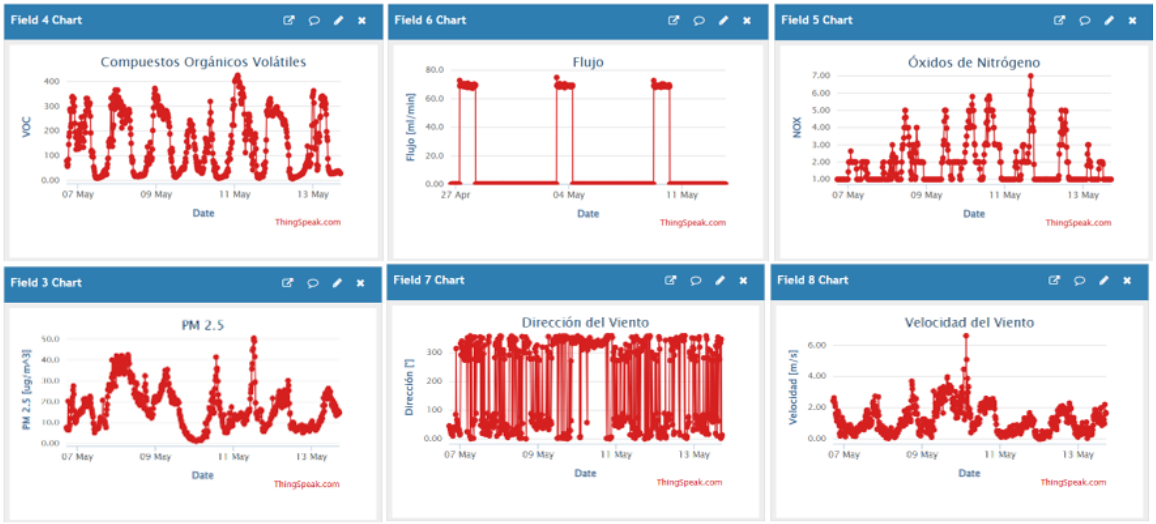


Figura 5.10: Actualización de nombres en los *fields* en ThingSpeak

Posteriormente, en la sección de visualización del canal, los datos enviados se representarán automáticamente en gráficas correspondientes a cada *field*. Además, es posible agregar diferentes tipos de visualizaciones, como la ubicación del dispositivo, indicadores de algunas variables o paneles individuales, tal como se muestra en la siguiente imagen:



((a)) Ubicación, indicadores para temperatura y humedad, ademas del ultimo dato de PM 2.5 y gráficas de temperatura y humedad.



((b)) Gráficos de COVs, Flujo, NOx, PM 2.5, dirección y velocidad del viento.

Figura 5.11: Visualización de datos en la plataforma ThingSpeak.

3. **Reinicio de variables y control del tiempo**

   Al final de la función `Prom15Min()`, todos los acumuladores y el contador utilizados en la función de 30 segundos se reinician a 0, junto con las variables que almacenan algunos valores máximos. Esto permite comenzar nuevamente el proceso de obtención de promedios para el siguiente ciclo.

```
19      ...
20      TMax=0;
21      TMin=100;
22      windSpdmax=0;
23
24      acumT=0; acumH=0; acumTT_1=0; acumTTA_1=0; acumTT_2=0;
      acumTTA_2=0; acumCA=0; acumF=0; acumWSpd=0; SumSen=0; SumCos
```

```
    =0; acum1p0=0;acum2p5=0;acum4p0=0;acum10p0=0;acumvoc=0;
    acumnox=0; acumVB=0;
25   contP=0; contF=0;
26   ...
```

(Ver Anexo: Código 1.3.3 - Función `Prom15Min()`.

Adicionalmente, se creó una función similar a `Ctr5seg()`, pero en este caso se nombró `Ctr15Min()`. La diferencia radica en que ahora se compara el valor de los minutos actuales con los minutos deseados (0, 15, 30 y 45), utilizando una condición AND en el segundo 0 para asegurar que siempre se ejecute antes de los datos de 30 segundos. La variable booleana utilizada en esta función se llama `Ctr15MinBool`.

Cuando se cumple la condición, se llama a la función `Prom15Min()` y se establece `Ctr15MinBool` en `false`. El booleano `Ctr15MinBool` se reactiva en los minutos 1, 16, 31 y 46, lo que permite reiniciar el proceso para el siguiente ciclo.

(Ver Anexo: Código 1.4.1 - Función `Ctr15Min()`.

5. Integración con aplicación móvil

Dado que el proyecto requiere ser controlado por un usuario, se decidió incorporar comandos de control que pudieran enviarse a la `ESP32` mediante `Bluetooth (BT)` para ejecutar funciones del muestreador. La `ESP32` cuenta con un módulo `BT` integrado, al cual se accedió utilizando la biblioteca desarrollada por `Espressif`, que proporciona las funciones necesarias para establecer una comunicación eficiente y confiable.

Utilizando esta biblioteca y un ejemplo de código proporcionado por el desarrollador 6, se definió el nombre del dispositivo y se inició la comunicación.

```
1  void BTVOC_Init(){
2    BT_VOC.begin(nombre); //Bluetooth device name|
3    Serial.println("-----------------------------------------------");
4    Serial.printf("El dispositivo con nombre \"%s\" está listo.\nAhora
       puede conectarse a Bluetooth!\n", nombre.c_str());
5    Serial.println("-----------------------------------------------");
6      //Serial.printf("The device with name \"%s\" and MAC address %s
      is started.\nNow you can pair it with Bluetooth!\n", nombre.c_str
      (), SerialBT.getMacString()); // Use this after the MAC method is
       implemented
7    #ifdef USE_PIN
8      BT_VOC.setPin(pin);
9      Serial.println("Using PIN");
10   #endif
11 }
```

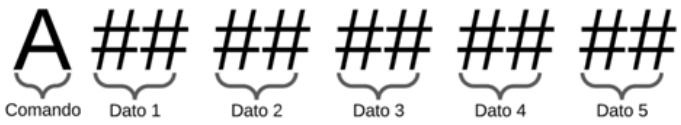
Una vez establecida la comunicación, se utilizó el prefijo `BT_VOC` en el programa para llamar las funciones encargadas de recibir o transmitir datos. Se definió una

variable que almacena el mensaje recibido y permite monitorear qué función se desea ejecutar.

```
Cadena = BT_VOC.readStringUntil('\n');
```

Para hacer más eficiente la comunicación, se estableció que los comandos de acción estarían definidos por letras mayúsculas (`A, B, C, D`, etc.), seguidas de números (cuando fuera necesario) para especificar el *qué*, *cómo* y *cuándo* de la función. Los comandos definidos fueron los siguientes:

- **Comando A: Programar una prueba.**



  El primer comando (`A`) registra los datos ingresados para programar una prueba. Los datos deben ser números de dos dígitos:

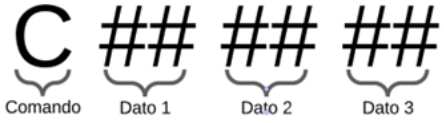
  - Dato 1: Número del día (ej. `05` para el día 5).
  - Dato 2: Mes (ej. `12` para diciembre).
  - Dato 3: Hora en formato 24h (ej. `14` para las 2 PM).
  - Dato 4: Minuto (ej. `30` para media hora).
  - Dato 5: Intervalo de conmutación de bombas en minutos (ej. `05`).

- **Comando B: Iniciar prueba.**

  El comando `B` (sin parámetros adicionales) inicia una prueba inmediatamente al ser recibido, con las siguientes características:

  - **Fecha/Hora**: Utiliza la fecha y hora exacta del momento de recepción.
  - **Flujo de operación**: Emplea valores predeterminados del sistema.
  - **Intervalo de bombas**: Usa la configuración por defecto.

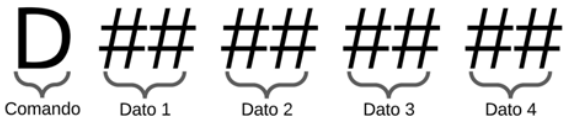
- **Comando C: Conmutación de válvulas.**



  El comando `C` permite programar la conmutación de válvulas sin activar la bomba. Se deben proporcionar tres datos de dos dígitos cada uno:

- Dato 1: Hora de apertura.
- Dato 2: Minuto de apertura.
- Dato 3: Intervalo de conmutación en horas.

- **Comando D: Conmutación de válvulas.**



El comando `D` se encarga de programar el apagado de la bomba. Todos los números deben ser de dos dígitos:

- Dato 1: Número del día.
- Dato 2: Mes.
- Dato 3: Hora en formato 24h.
- Dato 4: Minuto.

- **Comando F: Configuración de flujo.**

El comando `F` establece el flujo de operación de la bomba. Recibe un único valor numérico:

- Dato: Flujo en ml/min (1-4 dígitos, ej. `250` para 250 ml/min).

Para evitar daños en el sistema, este comando detiene cualquier prueba en curso antes de aplicar el nuevo valor de flujo.

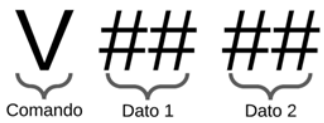
- **Comando P: Paro de prueba.**

El comando `P` detiene inmediatamente todo el sistema:

- Detiene la bomba instantáneamente.
- Cierra todas las válvulas del sistema.
- Finaliza cualquier prueba en curso.

- No requiere parámetros adicionales.

- **Comando V: Control de Válvulas.**



El comando `V` permite verificar el funcionamiento de las válvulas:

- Dato 1: Válvula a probar
  - `01`: Válvula 1.
  - `02`: Válvula 2.
  - `03`: Ambas válvulas.
- Dato 2: Parámetro de operación
  - Para válvula individual: Horas de prueba (ej. `02` = 2 horas).
  - Para ambas válvulas: Intervalo de conmutación en horas (ej. `02` = cada 2 horas).

- **Comando K: Configuración de constantes.**

El comando `K` permite la configuración dinámica de las constantes del controlador:

- **Dato 1**: Constante Proporcional (Kp) - 4 dígitos (ej. `0250`).
- **Dato 2**: Constante Integral (Ki) - 4 dígitos (ej. `0100`).
- **Dato 3**: Constante Derivativa (Kd) - 4 dígitos (ej. `0050`).

Características principales:

- Ajuste en tiempo real sin reprogramación.
- Precisión de 4 dígitos (0001-9999).
- Optimización dinámica del rendimiento.
- Sintonización rápida del controlador.

Con los comandos definidos que debe recibir el muestreador, se realizaron distintas instrucciones para separar la primera letra del mensaje recibido en la variable `Cadena`. En una variable llamada `Dato` se utilizó la función `substring()` para definir qué fragmento del mensaje extraer. En este caso del carácter 0 al carácter 1. Posteriormente se convirtió ese fragmento en un solo carácter con la función `charAt()` y se almacenó en la variable llamada `var`.

```
Dato = Cadena.substring(0,1);
var = Dato.charAt(0);
```

Este proceso de interpretación de comandos se implementó utilizando una estructura *switch-case*, permitiendo que el programa ejecute la acción correspondiente dependiendo de la letra recibida en el mensaje.

- **Caso A**: Para procesar los comandos, se utiliza la operación `Cadena.substring()` para extraer los valores de la cadena de texto, considerando las posiciones específicas de cada dato. Como este comando necesita cinco valores, la función separa la cadena en cinco partes, garantizando que cada variable obtenga el valor correspondiente según su posición en el mensaje. Por ello, es crucial que todos los números ingresados tengan exactamente dos dígitos, previniendo errores en la ubicación de los caracteres y asegurando la correcta asignación de valores.

  Una vez extraídos los datos, se transforman a números enteros y se asignan a las variables `diaON`, `mesON`, `horaON`, `minON` e `interv`. Estas variables establecen los parámetros de la prueba, de acuerdo con la estructura definida para los comandos.

  Luego, con estos valores se determina la duración total de la prueba, que corresponde al doble del valor del intervalo. Esto ocurre porque el intervalo representa el tiempo de muestreo para un cartucho, y al existir dos cartuchos en el sistema, la duración completa equivale a dos veces dicho intervalo. Además, para prevenir errores durante la ejecución, se establece el tiempo máximo de prueba en 59 minutos y se sincronizan los valores de `horaON` y `minON` con los tiempos de control de las válvulas, garantizando así el correcto funcionamiento del sistema.

```
case 'A':
     Dato1 = Cadena.substring(2,4); Dato2 = Cadena.substring
  (5,7);
     Dato3 = Cadena.substring(8,10);
     Dato4 = Cadena.substring(11,13); Dato5 = Cadena.substring
  (14,16);
     diaON = Dato1.toInt(); mesON = Dato2.toInt();
     horaON = Dato3.toInt(); minON = Dato4.toInt();
     interv = Dato5.toInt();
     TPruebaHor = 2*(Dato5.toInt());
     TPruebaMin = 59;
     horaValv = horaON; minutoValv = minON;
```

  Además, se agregó un mensaje en el monitor serial que confirma la programación de la prueba, mostrando la fecha, la hora y el flujo establecido. Este mensaje permite al usuario verificar que la prueba se ha programado correctamente y ayuda a prevenir errores en la configuración.

```
     Serial.print("Prueba establecida el (dia/mes): ");
     Serial.print(diaON); Serial.print("/");
     Serial.println(mesON); Serial.print("Hora: ");
```

```
4       Serial.print(horaON); Serial.print(":");
5       Serial.print(minON); Serial.println(" hrs");
6       Serial.print("Flujo a: "); Serial.print(Setpoint);
7       Serial.println(" mL/min");
```

Finalmente, se establecieron los valores iniciales de las variables `a` y `b` para el control de válvulas, con `a = 0` y `b = interv`. Esta configuración garantiza que el sistema inicie adecuadamente la conmutación de válvulas durante la prueba.

Además, se configuraron las variables booleanas `ContrBomb` y `ContrBombBool` en `true`, donde:

- `ContrBomb` activa la función de encendido de la bomba.
- `ContrBombBool` evita múltiples activaciones en un mismo minuto.

Esta implementación asegura que:

- La bomba se active exactamente en el momento programado.
- No ocurran encendidos repetidos durante la prueba.

```
1       a=0;
2       b=interv;
3       ContrBombBool = true;
4       ContrBomb = true;
5
6       break;
```

- **Caso B**: El caso `B` realiza funciones similares al caso `A`, con la diferencia de que, al tratarse de una prueba instantánea, obtiene los valores de fecha y hora directamente del reloj en tiempo real y los asigna a las variables `diaON`, `mesON`, `horaON` y `minON`. Posteriormente, imprime un mensaje de confirmación que indica la fecha y hora exactas de activación de la prueba.

```
1       Serial.print("Comienza prueba el dia de hoy (dia/mes): ");
2       Serial.print(fecha.day()); Serial.print("/");
3       Serial.println(fecha.month());
4       Serial.print("Hora: "); Serial.print(fecha.hour());
5       Serial.print(":"); Serial.print(fecha.minute());
6       Serial.println(" hrs");
7       horaValv = fecha.hour(); minutoValv = fecha.minute();
8       anioON = fecha.year(); diaON = fecha.day();
9       mesON = fecha.month();
10      horaON = fecha.hour(); minON = fecha.minute();
```

El flujo de la bomba se mantiene en su valor predeterminado, por lo que no es necesario modificarlo. Del mismo modo, las variables `TPruebaHor` y `TPruebaMin` se mantienen en `24` y `59`, respectivamente, definiendo la duración de la prueba en sus valores estándar.

Para garantizar el correcto funcionamiento del sistema, las variables `a` y `b` se restablecen a sus valores iniciales, y los booleanos `ContrBomb` y `ContrBombBool` se configuran en `true`.

```
      a=0;
      b=interv;

      ContrBombBool = true;
      ContrBomb = true;
      break;
```

- **Caso C**: Este caso tiene como única función la conmutación de las bombas, por lo que lo primero que se habilita son las funciones de control de las válvulas. En los casos anteriores, estas funciones se activaban dentro de la función `ContrBomb`, ya que estaban vinculadas al encendido de la bomba. Sin embargo, dado que en esta ocasión la bomba no se enciende, el control de las válvulas se habilita directamente desde el `case`.

  A partir del mensaje recibido, se extraen los datos correspondientes a:

  - Hora de encendido.
  - Minuto de encendido.
  - Intervalo de conmutación de válvulas.

  Para asegurar un inicio en estado controlado, ambas válvulas se cierran antes de comenzar el proceso.

```
    case 'C':
      ContrValv = true;
      valvBool = true;
      digitalWrite(electroValA, LOW);
      digitalWrite(electroValB, LOW);
      Dato1 = Cadena.substring(2,4);
      Dato2 = Cadena.substring(5,7);
      Dato3 = Cadena.substring(8,10);
      horaValv = Dato1.toInt();
      minutoValv = Dato2.toInt();
      interv = Dato3.toInt();
```

  Finalmente, se reinician los valores de las variables `a` y `b`, y se envía un mensaje por *Bluetooth* con la hora de activación de las válvulas y el intervalo de conmutación.

```
      a=0;
      b=interv;
      BT_VOC.print("Las valvulas conmutaran a las "); BT_VOC.
  print(horaValv); BT_VOC.print(":");
      BT_VOC.print(minutoValv); BT_VOC.println(" hrs");
      BT_VOC.print("Con un intervalo de ");
      BT_VOC.print(interv); BT_VOC.println(" hrs");
      BT_VOC.println(" ");
      break;
```

- **Caso D**: El caso D se encarga de programar el apagado de la prueba. Primero, extrae los datos del mensaje recibido y los convierte en valores enteros, asignándolos a las variables `diaOFF`, `mesOFF`, `horaOFF` y `minOFF`.

```
    Dato1 = Cadena.substring(2,4); Dato2 = Cadena.substring
(5,7);
    Dato3 = Cadena.substring(8,10);
    Dato4 = Cadena.substring(11,13);
    diaOFF = Dato1.toInt(); mesOFF = Dato2.toInt();
    horaOFF = Dato3.toInt(); minOFF = Dato4.toInt();
```

A continuación, imprime un mensaje indicando la fecha y la hora exacta en la que la prueba se apagará, proporcionando así una confirmación visual del proceso. Finalmente, habilita las funciones necesarias para el control del apagado de la bomba, asegurando que el sistema detenga su funcionamiento en el momento programado.

```
    BT_VOC.print("La prueba se detendra el (dia/mes): ");
    BT_VOC.print(diaOFF); BT_VOC.print("/");
    BT_VOC.println(mesOFF);
    BT_VOC.print("Hora: "); BT_VOC.print(horaOFF);
    BT_VOC.print(":"); BT_VOC.print(minOFF);
    BT_VOC.println(" hrs"); BT_VOC.println(" ");
    ContrOFF = true;
    ContrOFFBool = true;
```

- **Caso F**: El caso F se encarga de modificar el valor de flujo en el que se debe mantener la prueba programada. Para evitar alteraciones en el funcionamiento del sistema, se deshabilitan todas las funciones de control del compresor y las válvulas. Como medida de seguridad, el compresor y el ventilador se detienen, y las válvulas se cierran.

```
    ContrBomb = false;
    ContrOFF = false;
    ContrOFFBool = false;
    ContrValv = false;
    valvBool = false;
    PararBom();
    digitalWrite(electroValA, LOW);
    digitalWrite(electroValB, LOW);
    digitalWrite (vent, LOW);
```

Luego, se extrae el valor de flujo del mensaje recibido y se actualiza la variable *Setpoint* con el nuevo valor objetivo. Finalmente, se envía un mensaje a través del monitor serial para confirmar el nuevo flujo establecido para el compresor, asegurando que el usuario disponga de una referencia clara del cambio realizado.

```
    Dato1 = Cadena.substring(2,6);
    Setpoint = Dato1.toDouble();
    Serial.println(" ");
    Serial.print("Flujo establecido a: ");
    Serial.print(Setpoint);
    Serial.println(" mL/min");
```

```
7       Serial.println(" ");
```

- **Caso P**: En este caso, se deshabilitan todas las funciones de control relacionadas con las pruebas programadas. Se inhabilita el control de la bomba, de las válvulas y el sistema de apagado automático. La bomba y el ventilador se detienen por completo, mientras que las válvulas se cierran para evitar cualquier flujo no deseado.

```
1       ContrBomb = false;
2       ContrOFF = false;
3       ContrOFFBool = false;
4       ContrValv = false;
5       valvBool = false;
6       ParарBom();
7       digitalWrite(electroValA, LOW);
8       digitalWrite(electroValB, LOW);
9       digitalWrite (vent, LOW);
```

Finalmente, se envía un mensaje a través del monitor serial indicando que la prueba ha sido detenida exitosamente, asegurando que el usuario reciba confirmación inmediata de la acción realizada.

```
1       Serial.println(" ");
2       Serial.println("Prueba detenida");
3       Serial.println(" ");
```

- **Caso V**: Esta función primero extrae los dos datos recibidos en el mensaje. Al primer dato se le obtiene únicamente el carácter necesario para ingresar a un segundo `switch-case`, que determinará el comportamiento del sistema. Dado que el funcionamiento del compresor, el ventilador y las válvulas debe comenzar de inmediato, las variables `diaON`, `mesON`, `horaON` y `minON` se establecen con los valores actuales del reloj en tiempo real.

```
1     case 'V':{
2       Dato1 = Cadena.substring(2,3);
3       Dato2 = Cadena.substring(4,7);
4       char dat = Dato1.charAt(0);
5
6       anioON = fecha.year(); diaON = fecha.day(); mesON = fecha.
    month();
7       horaON = fecha.hour(); minON = fecha.minute();
8       horaValv = horaON; minutoValv = minON;
```

Después, se ingresa al segundo `switch-case`, en donde se presentan tres opciones:

- **Caso 1:** Se abre únicamente la válvula 1 y el tiempo de prueba se define con el segundo dato recibido, que indica cuántas horas permanecerá abierta la válvula, con el ventilador y la `bomba` en funcionamiento.

```
        case '1':
          digitalWrite(vent, HIGH);
          digitalWrite(electroValA, HIGH);
          digitalWrite(electroValB, LOW);

          TPruebaHor = Dato2.toInt();
          TPruebaMin = 59;

          EncenderBom();
          OffAutomatico();
```

- **Caso 2:** Al igual que el caso 1, se abre únicamente la válvula 2, siguiendo la misma lógica de tiempo definida por el segundo dato.

```
       case '2':
          digitalWrite(vent, HIGH);
          digitalWrite(electroValA, LOW);
          digitalWrite(electroValB, HIGH);

          TPruebaHor = Dato2.toInt();
          TPruebaMin = 59;

          EncenderBom();
          OffAutomatico();
          ContrOFF = true;
```

- **Caso 3:** Se abre primero la válvula 1 y, después del tiempo indicado en el segundo dato, las válvulas conmutan para continuar la prueba con la válvula 2 durante el mismo período.

```
        case '3':
          interv = Dato2.toInt();
          TPruebaHor = 2*(Dato2.toInt());
          TPruebaMin = 59;

          a=0;
          b=interv;

          ContrBombBool = true;
          ContrBomb = true;
```

En todos los casos, mientras una válvula esté abierta, tanto el `compresor` como el `ventilador` permanecerán encendidos.

- **Caso K**: Para este caso, lo primero que se realiza es la extracción de los datos del mensaje que corresponden a las constantes del controlador `PID`. Estos datos se almacenan en las variables correspondientes dentro del programa (`PID_Kp`, `PID_Ki`, `PID_Kd`). Dado que dichas variables están definidas como tipo `double`, es necesario convertir los valores extraídos antes de asignarlos. Esto permite que el controlador pueda operar con la precisión requerida, asegurando un ajuste adecuado de sus parámetros.

```
    case 'K':{
      Dato1 = Cadena.substring(2,6); Dato2 = Cadena.substring
   (7,11); Dato3 = Cadena.substring(12,16);
      Kp = Dato1.toDouble();
      Ki = Dato2.toDouble();
      Kd = Dato3.toDouble();
```

Estas constantes se actualizan automáticamente en cada iteración del controlador `PID`, eliminando la necesidad de llamar explícitamente a funciones adicionales para aplicar los cambios. Al finalizar, los valores de las constantes modificadas (`Kp`, `Ki`, `Kd`) se imprimen para verificar su actualización correcta antes de concluir la ejecución del caso.

```
      BT_VOC.print(Kp);BT_VOC.print(" ");
      BT_VOC.print(Ki);BT_VOC.print(" ");
      BT_VOC.print(Kd);BT_VOC.println(" ");
      break;
```

- **Caso default**: Se implementó un `case default` para manejar comandos no reconocidos. En estas situaciones, el programa ignora la entrada y no ejecuta ninguna acción, previniendo así errores o comportamientos inesperados del sistema.

```
    default:
      break;
  }
```

Al tener bien definidos los casos en la programación y en los comandos, se utilizó una aplicación para teléfonos llamada `Monitor Serial Terminal`. Esta aplicación, disponible exclusivamente para dispositivos Android, permite conectarse a cualquier dispositivo `Bluetooth` y enviar o recibir mensajes de manera similar al monitor serial de un `IDE`.

Con esta herramienta:

- Se realizaron pruebas enviando los comandos en el formato adecuado.
- Se observó el comportamiento del muestreador.
- Se verificaron los mensajes de respuesta del sistema.

De esta forma, se aseguró que la comunicación y el control funcionaran correctamente.

6. Implementación del PID para el compresor

Para regular el flujo de la bomba, se implementó un controlador `PID`, diseñado para mantener un flujo constante minimizando errores. Este control ajusta dinámicamente la respuesta del sistema mediante tres componentes:

1. **Proporcional (P)**:
   - Corrige el error actual en función de su magnitud
   - $u_P(t) = K_p \cdot e(t)$
2. **Integral (I)**:
   - Compensa errores acumulados en el tiempo
   - Elimina desviaciones persistentes
   - $u_I(t) = K_i \cdot \int_0^t e(\tau)d\tau$
3. **Derivativo (D)**:
   - Predice tendencias futuras del error
   - Responde a variaciones repentinas
   - Depende del tiempo de muestreo (T_s)
   - $u_D(t) = K_d \cdot \frac{de(t)}{dt}$

Donde:

- $e(t)$: Error en el instante t
- K_p, K_i, K_d: Constantes del controlador
- T_s: Periodo de muestreo

Este control es viable en el sistema porque cuenta con los elementos necesarios para su implementación: un sensor que mide la variable de proceso (flujo de aire), un actuador (bomba o compresor de aire) cuya salida se ajusta según la señal de control generada por el PID, y un microcontrolador que procesa la retroalimentación comparando la medición del sensor con el *setpoint* deseado. A partir de esta comparación, el microcontrolador realiza los cálculos necesarios para aplicar el control PID en su forma discreta, ajustando dinámicamente la respuesta del sistema y minimizando el error en tiempo real [10].

El código implementa un control PID para regular el flujo de la bomba, asegurando que se mantenga lo más constante posible con el menor error. La función principal, `ControlFlujo()`, inicia verificando si la bomba está encendida. Si es así, obtiene una lectura del flujo a partir del sensor y la almacena para su posterior procesamiento. Esta lectura se usa para calcular el error, que es la diferencia entre el flujo medido y el *setpoint* deseado.

```
1 if(EstadoBom==true){
2       flujoInst = SensorFlujo(Read_MCP3208(0));
3       acumFl_Ins+=flujoInst;
4       contFl_Ins++;
```

Para optimizar la respuesta del control, se implementa un ajuste dinámico de los parámetros del PID. Dependiendo de la magnitud del error, los valores de las constantes K_p, K_i y K_d cambian para hacer el control más estable y eficiente. Si el error es pequeño, se utilizan constantes más bajas para evitar oscilaciones innecesarias, mientras que si el error es grande, se aplican constantes más agresivas para corregirlo rápidamente.

```
      if(PID_error < (Setpoint*0.15) && PID_error > -(Setpoint
*0.15)){
        Kp=0.002; Ki=0.002; Kd=0.002;
      }
      else if(PID_error < (Setpoint*0.60) && PID_error > -(
Setpoint*0.60)){
        Kp=0.004, Ki=0.01, Kd=0.005;
      }
      else{
      Kp=0.07; Ki=0.05; Kd=0.010;
    }
```

Luego, se calculan las tres componentes del PID:

- La proporcional (`PID_p`), que es el producto de la constante K_p con el error obtenido.

```
    ...
        PID_p = Kp * PID_error;
    ...
```

- La integral (`PID_i`), que acumula errores pasados, con una condición *anti-windup* para evitar que siga aumentando cuando el error sea menor al 15 %, permitiendo corregir desviaciones sostenidas.

```
    ...
      PID_i += Ki * PID_error;
      if(PID_i > 500) PID_i = 500;
      if(PID_i < -500) PID_i = -500;
    ...
```

- La derivativa (`PID_d`), que toma en cuenta la diferencia entre el error actual y el error anterior, dividida por el tiempo transcurrido. Esta componente ayuda a predecir errores futuros en función de la velocidad del error.

```
    ...
      timePrev = Time;
      Time = millis();
      elapsedTime = (Time - timePrev) / 1000.0;

      if(elapsedTime > 0) {
        PID_d = Kd*((PID_error - previous_error)/elapsedTime);
      }else{
        PID_d = 0;
      }
    ...
```

Después de obtener el valor total del control PID, se aplican restricciones para evitar valores fuera de rango, asegurando que la salida permanezca dentro de los límites operativos del sistema. Finalmente, este valor se usa para ajustar la velocidad de la bomba y se actualiza el error previo para el siguiente ciclo de cálculo.

```
...
  if(PID_value < 0){
    PID_value = 0;
  }
  if(PID_value > 1000){
    PID_value = 1000;
  }
...
```

Si la bomba está apagada, el código continúa obteniendo las lecturas del sensor sin realizar ajustes en el control.

```
if(ContrBomb == true){
...
else{
  flujoInst = SensorFlujo(Read_MCP3208(0));
}
```

7. Implementación y puesta en marcha

7.1. Diseño de PCB

7.1.1. Diseño de esquemático y placa en el software KiCad

Con un modelo funcional probado en una *protoboard*, donde todos los sensores, actuadores y reguladores operaban en conjunto y de manera simultánea, se decidió dar el siguiente paso: diseñar una placa electrónica. Esto permite optimizar el espacio y mejorar la conexión entre los componentes, eliminando la cantidad excesiva de cables y facilitando las pruebas.

Aunque aún no se contaba con la versión final del software, el hardware ya estaba completamente definido, lo que permitió avanzar con el diseño de la tarjeta sin riesgo de cambios en los componentes. Hasta este punto, todo funcionaba según lo planeado, lo que ayudó a lograr una disposición más ordenada y estructurada. Esto facilitó el desarrollo y ajuste del software sin afectar la integración del sistema.

7.1.2. Diseño y obtención de huellas y símbolos de los componentes.

Antes del diseño del esquemático del muestreador y la tarjeta electrónica, se realizó una pequeña búsqueda sobre qué componentes ya contaban con su modelo de símbolo y huella para ser utilizados directamente en el software KiCad[27]. En algunos

[27] *KiCad* es una herramienta de software EDA (*Electronic Design Automation*, automatización de diseño electrónico) gratuita y de código abierto, que se utiliza para diseñar *PCB* (placas de circuito impreso o circuitos electrónicos) y esquemáticos. https://alfaiot.com/iot/kicad/

casos, también se logró obtener sus modelos en 3D, lo que facilitó aún más el diseño. A continuación, se presentan algunas de las páginas de donde se obtuvieron dichos archivos:

- **Octopart** – Fichas técnicas, búsqueda de piezas y componentes electrónicos. https://octopart.com
- **SnapEDA** – Biblioteca de símbolos y huellas PCB gratuitas. https://www.snapeda.com
- **DigiKey** – Distribuidor de componentes electrónicos con modelos disponibles. https://www.digikey.com
- **SamacSys** – Plataforma para obtener símbolos, huellas y modelos 3D de componentes electrónicos. https://www.samacsys.com

A pesar de la posibilidad de obtener los modelos correspondientes en diversas plataformas, hubo algunos componentes de los cuales no se encontró su símbolo, huella o ambos. En estos casos, fueron creados manualmente, usando como guía las hojas de datos (*datasheets*) de cada uno. Estas especificaciones fueron de gran ayuda, ya que proporcionaban las dimensiones y la composición del componente, permitiendo diseñar con precisión dentro del software.

7.1.3. Esquemático

Con el símbolo, la huella y el modelo 3D correctamente asociados a cada componente, se comenzó el diseño del esquemático en el software KiCad. En esta etapa, se realizaron las conexiones correspondientes entre los sensores, actuadores y el microcontrolador, asegurando una distribución lógica y eficiente.

Para una mejor organización y facilidad de identificación, el esquemático se estructuró en distintas hojas, dividiendo el sistema en etapas específicas: alimentación, sensores y actuadores, potencia y microcontrolador con sus extensiones. Esta segmentación permitió un diseño más ordenado, facilitando la ubicación y comprensión de cada componente dentro del circuito.

- **Hoja de Alimentación** Para la hoja de alimentación, se incorporaron los dos reguladores de voltaje (3.3 V y 5 V), junto con dos pequeños fusibles. De este modo, la entrada externa de 12 V pasa primero por los fusibles antes de llegar a los reguladores, proporcionando protección al sistema. Además, en esta sección se conectó la referencia de voltaje de 5 V previamente diseñada para el ADC, asegurando una conversión precisa de las señales analógicas. *(Ver Anexo: Esquemático 1 - Alimentación).*
- **Hoja de Sensores y Actuadores** Posteriormente, se trabajó en la hoja de sensores y actuadores, donde se organizó cada componente y se añadieron etiquetas para identificar claramente sus conexiones. En esta etapa, el enfoque fue

el asignar correctamente cada sensor y actuador a su correspondiente alimentación y a su respectiva conexión en el microcontrolador. Se aseguró que los pines de comunicación y datos, ya sean SPI, I2C, analógicos o digitales, estuvieran correctamente definidos y vinculados dentro del diseño, tal como se había probado previamente en la protoboard. *(Ver Anexo: Esquemático 1 – Sensores y Actuadores).*

- **Hoja de Potencia** La siguiente hoja correspondió a la etapa de potencia, donde se conectaron los MOSFETs previamente descritos para que funcionaran como interruptores electrónicos, permitiendo que el microcontrolador controlara la apertura o cierre del circuito. En esta sección también se integró el puente H, el cual cuenta con una entrada de 12 V para alimentar la bomba mediante una señal PWM. *(Ver Anexo: Esquemático 1– Etapa de Potencia).*

- **Hoja del Microcontrolador** Después, se continuó con la hoja del microcontrolador, donde se decidió incluir también el ADC, el reloj de tiempo real, la compuerta *NOT*, el convertidor de nivel y el Schmitt Trigger Inverter. Estos componentes actúan más como extensiones del microcontrolador, que como sensores o actuadores, por lo que tenía sentido agruparlos en esta sección. Al igual que en la hoja de sensores y actuadores, se hicieron las conexiones correspondientes mediante etiquetas, lo que facilitó la organización y comprensión del esquemático. *(Ver Anexo: Esquemático 1 – Microcontrolador).*

- **Conexiones Globales** Finalmente, se unieron las conexiones globales de cada hoja, asegurando que todo el sistema estuviera correctamente interconectado y funcional. *(Ver Anexo: Esquemático 1 – Muestreador de Bajo Flujo Automático).*

7.1.4. Diseño de Placa Electrónica

Con el esquemático finalizado y todas las conexiones correctamente establecidas, se usó la función del software *KiCad* que permite trasladar automáticamente el diseño del esquemático a la placa. Esto proporcionó los *footprints* ya asociados a cada componente, junto con las conexiones correspondientes según el diseño.

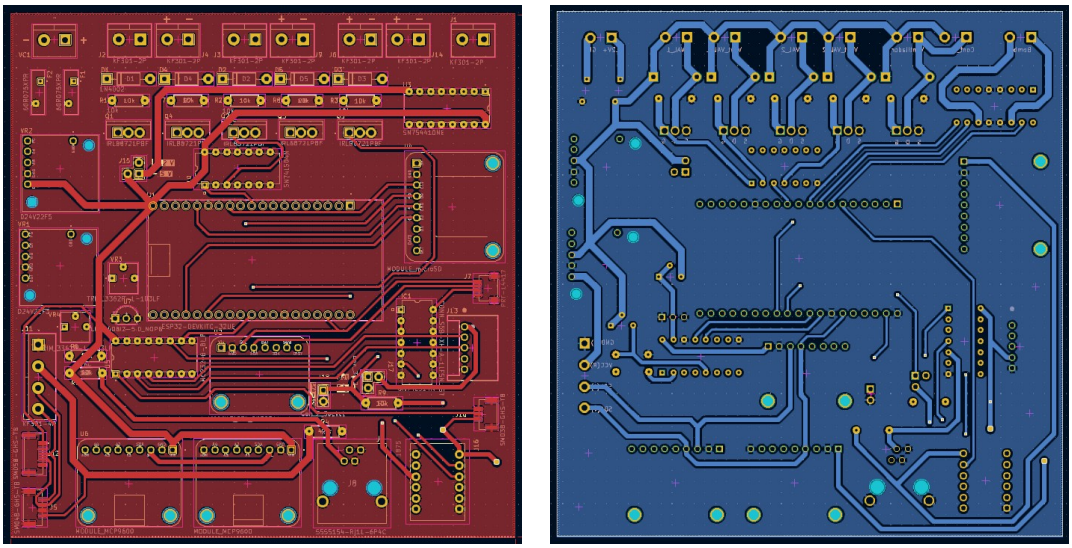
En este paso, se agregaron más componentes a la placa, principalmente conectores, con el fin de asignar a cada sensor una entrada única y evitar errores en la conexión. Además, algunos módulos se colocaron con *sockets* hembra para facilitar su montaje y desmontaje. Para reflejar estos cambios, se actualizaron los conectores en el esquemático antes de continuar con el diseño de la placa. Aunque esto no afectaba directamente al esquemático, sí se veía reflejado en la visualización en 3D del software, lo que permitió obtener una representación más realista de la tarjeta electrónica y optimizar su tamaño.

Posteriormente, se organizó la distribución de los componentes. Se colocaron todos los elementos de 12V en la parte inferior, junto a la alimentación. En la parte superior

se ubicaron los reguladores, potenciómetros, la compuerta *NOT* y el microcontrolador. Los sensores y demás componentes externos se posicionaron en los bordes de la tarjeta para facilitar su conexión.

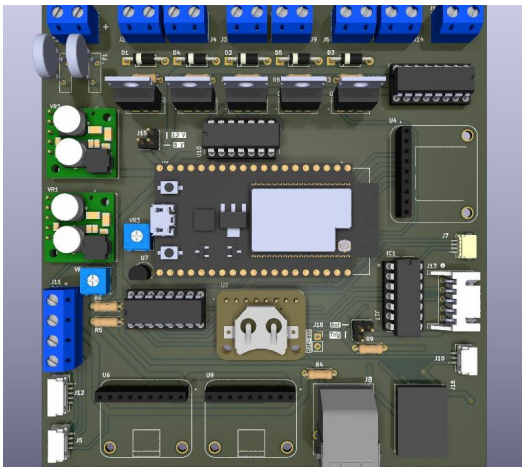
Por último, se agregaron las pistas necesarias para interconectar los componentes según el diseño del esquemático. Se optó por un diseño de doble capa (superior e inferior), lo que simplificó el enrutamiento y permitió realizar conexiones de manera más eficiente en comparación con el uso de una sola capa.

A continuación, en la siguientes figuras se muestra el diseño de la placa ya enrutada y con sus planos de tierra correspondientes, además del modelo 3D de esta.

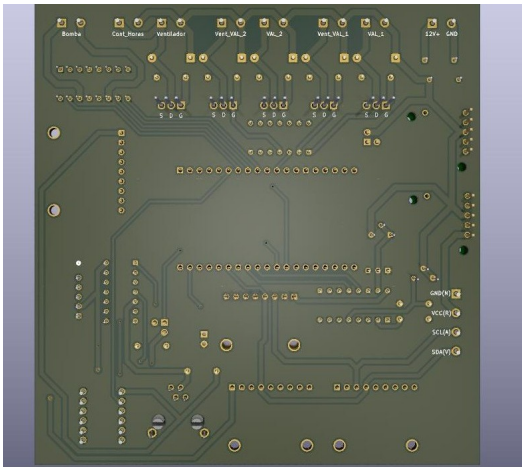


((a)) Vista frontal. ((b)) Vista trasera.

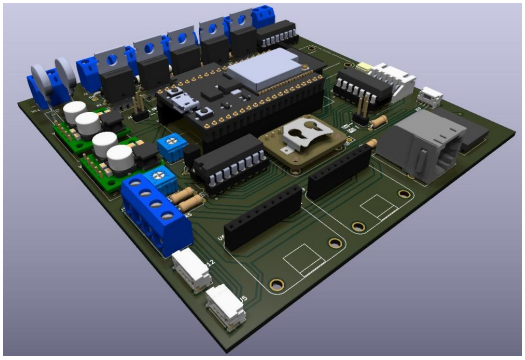
Figura 5.12: Placa electrónica diseñada vista desde KiCad.



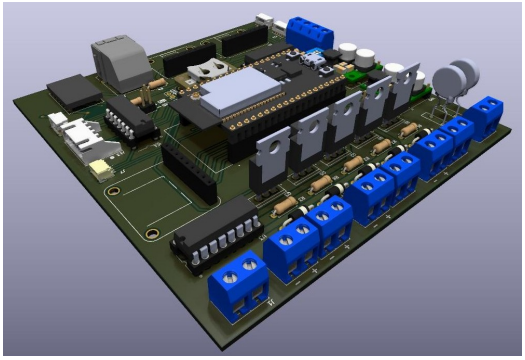
((a)) Vista frontal.



((b)) Vista trasera.



((c)) Vista lateral a 45°.



((d)) Vista lateral a 225°.

Figura 5.13: Placa electrónica diseñada vista en 3D.

7.2. Manufactura de PCB

Para la manufactura de las PCB, se optó por enviarlas a fabricar con la empresa China *JLCPCB*[28], debido a su rapidez, costos accesibles y el acabado de alta calidad. Esta opción resultó más conveniente en comparación con la fabricación manual o el uso de una CNC, ya que garantiza precisión en el diseño y una presentación más profesional.

Una vez que se recibió la placa, se procedió a soldar cada uno de los componentes o sus respectivos *sockets*, asegurando que el ensamblaje coincidiera con el modelo 3D. Con la tarjeta completamente armada, se hizo la evaluación del desempeño del sistema en su versión final.

[28] *JLCPCB* es un fabricante chino de placas de circuito impreso (PCB) que ofrece servicios de prototipado y producción en masa de PCB a precios competitivos. https://jlcpcb.com/

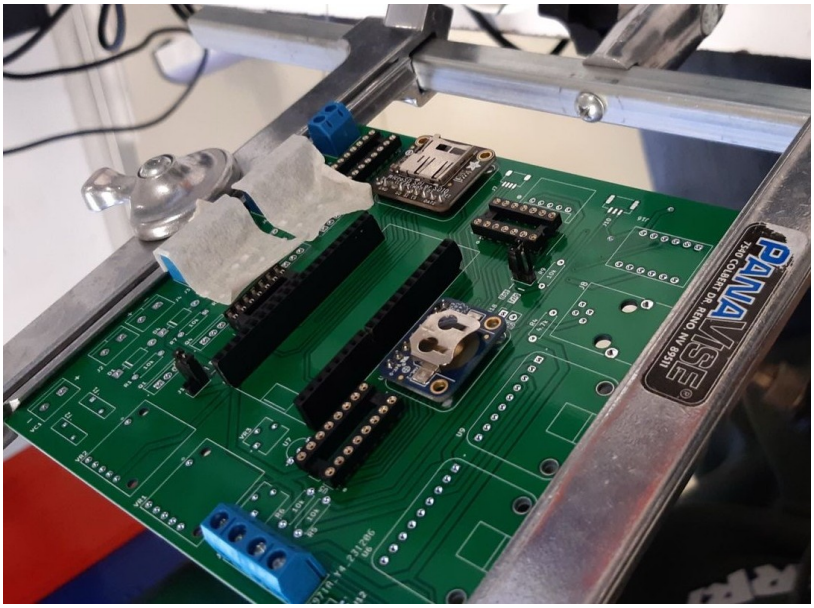


Figura 5.14: Ensamble de componentes en la placa electrónica.

7.3. Pruebas electrónicas de PCB

Con todos los componentes de la placa listos, se colocaron los sensores que van de forma "externa" a través de sus conectores correspondientes, así como los sensores y actuadores que se conectan a sus respectivas borneras.

Con todo en su lugar, se cargó la versión más reciente del software, el cual ya permitía el funcionamiento coordinado de todos los componentes con un flujo de aire constante en la bomba. Esto permitió realizar las primeras pruebas directamente en la placa electrónica.

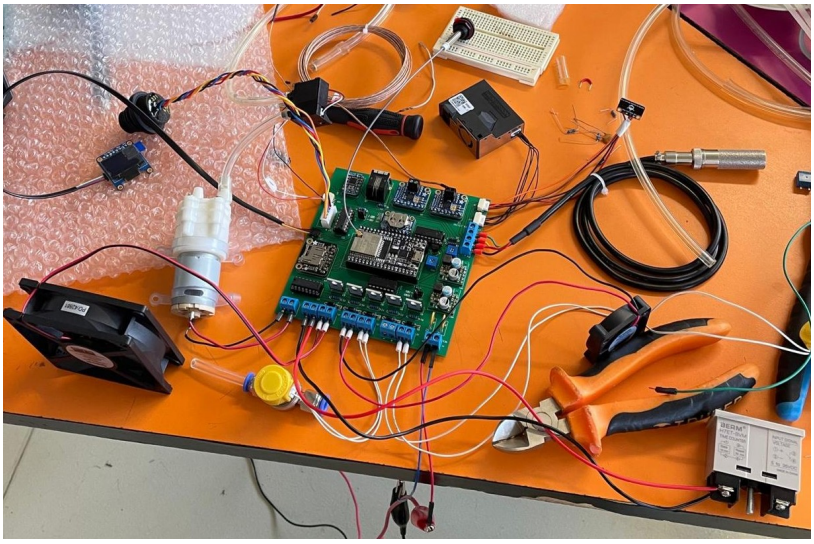


Figura 5.15: Placa electrónica ensamblada y conectada a todos sus componentes.

Inicialmente, se encendió el sistema y se verificó que todo arrancara correctamente, tal como en las pruebas previas con la *protoboard*. Luego, se enviaron algunas instrucciones a través del módulo *Bluetooth* para comprobar su funcionamiento, obteniendo las respuestas esperadas.

A continuación, se probó la interfaz manual moviendo el *joystick* y observando su respuesta en pantalla, la cual operaba correctamente. Sin embargo, al evaluar la señal del anemómetro, se detectó que la conversión no era del todo precisa. En algunas ocasiones, se generaba ruido o la señal no activaba correctamente la interrupción programada.

Para identificar la causa, se realizó un análisis con el osciloscopio y se notó que el sistema respondía mejor sin el *Schmitt Trigger* y el convertidor de nivel. Al atravesar estos componentes, la detección del cambio de estado en la señal (de 0 a 1 o viceversa) no era tan efectiva. Dado que la señal del anemómetro operaba a 3.3V, era posible enviarla directamente al microcontrolador sin necesidad de conversión

Como estos dos componentes estaban destinados únicamente a la detección de pulsos, su eliminación no afectaba al resto del sistema. Por ello, se decidió prescindir de ellos y conectar directamente la salida del anemómetro a una entrada digital del microcontrolador, utilizando un puente con ayuda de un cable.

8. Interfaz gráfica del muestreador (*Display y Joystick).*

El sistema debe ser controlado por un usuario, por lo que es necesario contar con interfaces que le permitan interactuar y manipular el muestreador de forma práctica.

No era suficiente disponer únicamente de una interfaz a través del teléfono con comunicación *Bluetooth*, ya que no se podía garantizar que todos los usuarios tuvieran un dispositivo Android o que, por alguna razón extraordinaria, no tuvieran su teléfono disponible al momento de programar una prueba.

Por esta razón, se decidió adicionar una interfaz integrada directamente en el muestreador. Esta interfaz se visualiza en un display de 128×64 pixeles, por lo que debía ser sencilla, intuitiva y funcional para facilitar el uso al operador.

Para controlar y manejar el display, se descargaron las bibliotecas proporcionadas por el fabricante (Adafruit), las cuales son `Adafruit_SSD1306.h` y `Adafruit_GFX.h`. Con estas bibliotecas se definió el tamaño de la pantalla y la dirección del dispositivo, permitiendo así una correcta inicialización y funcionamiento del display dentro del sistema.

```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define SCREEN_ADDRESS 0x3D
```

El tamaño reducido del display representó un desafío importante para el diseño de la interfaz. Por esta razón, se planeó un diseño basado en páginas, donde el display muestra diferentes pantallas dependiendo de las selecciones o acciones que el usuario realice con el *joystick*. Esta estructura facilita la navegación y permite presentar la información de forma clara y organizada.

8.1. Página menú principal

La primera página fue concebida como un menú principal. En este menú se debía incluir opciones que permitieran al usuario trasladarse fácilmente a las distintas funciones del muestreador. Se decidió incluir tres opciones principales en el menú por dos razones fundamentales:

1. **Visibilidad y legibilidad:** El texto debe tener un tamaño adecuado para ser fácilmente visible en el display de 128x64 píxeles. Al limitarse a tres opciones, se logra un equilibrio entre la cantidad de información mostrada y la legibilidad, evitando que la página se vea saturada.

2. **Funcionalidad esencial:** Era necesario proporcionar un acceso rápido a funciones clave. Por ello, se incluyó:

    - Una página para el despliegue de datos, que permite observar los datos recopilados por los sensores en tiempo real.

- Una página para la programación de pruebas, que facilita la configuración del muestreador.
- Una página para realizar pruebas instantáneas de fugas y verificar el flujo.

Estas tres páginas completan las tres opciones ideales para un menú de la interfaz completo y sencillo.

Con las funciones proporcionadas por la biblioteca, se desarrolló una función llamada `paginMenu()`, encargada de desplegar en el display la página principal con las tres opciones. Esta función se invoca únicamente cuando se requiere mostrar el menú principal.

```
void paginMenu(){
  ...
}
```

Para iniciar el proceso, se llama al objeto `display` y se utilizan diversas funciones de la biblioteca. Primero, se limpia el búfer del display para eliminar cualquier información previa y se selecciona el color blanco como el color predeterminado para todo lo que se mostrará. Posteriormente, se dibuja un círculo de 7 píxeles de diámetro con una función para dibujar elipses y dándoles valores iguales al eje menor y al eje mayor. Este círculo funciona como cursor, cuya posición se controla mediante variables que permiten su desplazamiento según la interacción del usuario con el *joystick*.

```
  ...
  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);
  display.drawRoundRect(xm, ym, 7, 7, 45, SSD1306_WHITE);
  ...
```

A continuación, se configura el tamaño del texto como el segundo más pequeño disponible para asegurar una buena legibilidad sin saturar la pantalla. El texto se posiciona cuidadosamente en tres ubicaciones distintas para las opciones del menú:

1. **MANUAL**: Posición $(20, 4)$.
2. **CONFIG.**: Posición $(20, 27)$.
3. **DATOS**: Posición $(20, 50)$.

```
  ...
  display.setTextSize(2);
  display.setCursor(20,4);
  display.println("MANUAL");
  display.setCursor(20,27);
  display.println("CONFIG.");
  display.setCursor(20,50);
  display.println("DATOS");
  ...
```

Finalmente, tras definir el texto y las posiciones, se utiliza la función `display.display()` para reflejar todos los cambios realizados en el buffer y desplegar el menú en el display. Esto garantiza una interfaz visual clara, funcional y fácil de navegar para el usuario.

```
  ...
  display.display();
}
```

De tal forma que el menú principal se ve así:

Figura 5.16: Menú principal interfaz OLED.

Para permitir el desplazamiento del cursor circular en el menú, se utilizaron las funciones del ADC (Convertidor Analógico-Digital) para medir el voltaje de las salidas del *joystick*. Dado que el movimiento en esta página se limita al eje Y y a tres posiciones posibles, se diseñaron dos funciones para registrar el movimiento hacia arriba y hacia abajo. Estas funciones actualizan la posición del cursor en el display según la interacción del usuario con el *joystick*.

```
void movAbajo(void){
  ...
}

void movArriba(void){
  ...
}
```

Además, se definió un booleano llamado `menu`, que indica si la página actual es el menú, asegurando que el movimiento del cursor solo se active en el contexto adecuado.

```
bool menu=true;
```

Cada función comienza con una condición `if` que primero verifica si la página mostrada corresponde al menú. Si es así, se realiza una segunda comprobación para garantizar que el cursor esté en una posición válida para moverse:

- Para el movimiento hacia arriba, la función verifica que el cursor no esté en la posición más alta. Si está en esa posición, se considera que ha alcanzado el límite superior y no se permite más desplazamiento.

```
void movArriba(void){
  if(menu==true){
    if(ym==54||ym==31){
```

```
      ...
    }
  }
}
```

- Para el movimiento hacia abajo, se comprueba que el cursor no esté en la posición más baja, evitando que se sobrepase el límite inferior.

```
void movAbajo(void){
  if(menu==true){
    if(ym==31||ym==8){
      ...
    }
  }
}
```

Si la posición es válida, se ajusta la posición vertical del cursor:

- Se resta un valor predefinido a la variable correspondiente al eje Y para el movimiento hacia arriba.

```
  ...
    if(ym==54||ym==31){
      ym=ym-23;
    }
  ...
```

- Se suma el mismo valor para el movimiento hacia abajo.

```
  ...
    if(ym==31||ym==8){
      ym=ym+23;
    }
  ...
```

Finalmente, cuando se invoca nuevamente la función `paginMenu()` para actualizar la pantalla, el cursor (representado por un círculo) se desplaza visualmente a la nueva posición, brindando una interacción fluida e intuitiva para el usuario.

Figura 5.17: Menú principal interfaz OLED cursor desplazado.

Con el menú principal funcionando correctamente y permitiendo un desplazamiento fluido entre las opciones, se diseñó una función que se ejecutará únicamente cuando se presionara el *joystick*. Esta función evalúa la posición del cursor en el eje Y y, dependiendo de dicha posición, modifica el estado de los booleanos encargados de indicar qué página se está desplegando. Para lograrlo, se añadieron booleanos específicos para cada una de las páginas disponibles en la interfaz.

```
bool menu=true, datos=false, manual=false;
void selecc(void){
  ...
  }
```

Dentro de la función, se implementaron estructuras condicionales para comparar la posición actual del cursor con las posiciones definidas para cada opción del menú. Cuando el usuario presiona el *joystick*, se determina la opción seleccionada y, en consecuencia, se establece en `true` el booleano correspondiente a la página seleccionada, mientras que los booleanos de las demás páginas se asignan a `false`. Esto asegura que únicamente la página deseada se mantenga activa.

```
...
  if(menu==true){
    if(ym==8){
      menu=false;
      manual=true;
      datos=false;
      ...
    }else if(ym==31){
      datos=false;
      manual=false;
      menu=false;
      ...
    }else{
      datos=true;
      manual=false;
      menu=false;
    }
...
```

Además, al ingresar a la nueva página, se restablece la posición del cursor a su nuevo valor inicial. Esto permite que al volver a acceder a dicha página, el cursor siempre comience desde una posición predeterminada, lo que mejora la experiencia del usuario y facilita la navegación dentro del sistema.

```
...
  if(menu==true){
    if(ym==8){
      ...
      x=23;
      y=1;
    }else if(ym==31){
      ...
      x=23;
      y=1;
    }
```

```
...
```

8.2. Página manual

Posteriormente se diseñó la página correspondiente a la primera opción: `MANUAL`. Esta sección fue pensada para que el usuario pueda controlar el muestreador de forma directa y sin necesidad de programar pruebas.

Para esta página, se utilizó el mismo concepto empleado en el menú principal para desplegar los datos. Se diseñó una función específica que se ejecuta únicamente cuando se desea mostrar esta sección. Al activarse, la función limpia el búfer del display y configura el color en blanco para todo lo que se desplegará en la pantalla.

```
void paginaMan(void){
  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);
  ...
}
```

El cursor de selección fue diseñado como un marco que rodea las opciones disponibles. Para lograr este efecto visual, se utilizó la función de dibujo de elipses, ajustando los valores de su eje mayor y menor para que pueda encerrar adecuadamente dos letras en mayúscula. Esta elección permitió optimizar el espacio disponible en la pantalla y ofrecer cuatro opciones seleccionables sin saturar la interfaz.

```
void paginaMan(void){
  ...
  display.drawRoundRect(x, y, 40, 22, 5, SSD1306_WHITE);
  ...
  display.setTextSize(2);
  display.setCursor(27,6);
  display.println("V1  V2");
  display.setTextSize(2);
  display.setCursor(27,27);
  display.println("F:  "+flujol[ifo]);
  display.setCursor(51,44);
  display.println("<-");
  display.display();
}
```

De tal forma que la página se ve así:

Figura 5.18: Página manualen la interfaz de la pantalla OLED.

Esta página cuenta con un desplazamiento del cursor en cuatro direcciones: arriba, abajo, izquierda y derecha. Para lograr este movimiento, se agregaron funciones específicas que permiten al microcontrolador detectar la dirección en la que se mueve el *joystick* y ajustar la posición del cursor en consecuencia.

```
void movDerecha(void){
}
void movIzquierda(void){
}
void movAbajo(void){
}
void movArriba(void){
}
```

En el caso de los desplazamientos laterales, se implementaron dos funciones: una para detectar el movimiento hacia la izquierda y otra para detectar el movimiento hacia la derecha. Cada función evalúa la posición actual del *joystick* y, si se cumple la condición correspondiente, suma o resta un valor definido a la coordenada X del cursor. Esto genera el efecto visual de desplazamiento horizontal en la pantalla, permitiendo que el usuario navegue fácilmente entre las opciones ubicadas en la misma fila.

```
void movDerecha(void){
  ...
  if(manual==true){
      if(x==23 && y==1){
      x=x+49;
      }
  }
  ...
}

void movIzquierda(void){
  ...
  if(manual==true){
      if(x==72 && y==1){
      x=x-49;
```

```
      }
    }
  ...
}
```

Para el desplazamiento hacia arriba o abajo, se mantiene la misma lógica de sumar o restar el valor correspondiente al eje Y según sea necesario. Sin embargo, se agregó una función para distinguir cuándo el movimiento corresponde al arreglo de valores de flujo. Para ello, se emplea una condicional que verifica tanto el estado del booleano de selección de la opción de cambio de flujo como la posición del cursor, asegurando que realmente se quiera modificar dicho valor. Si la condición se cumple, en lugar de mover el cursor, se incrementa o decrementa en uno el índice del arreglo de valores de flujo, permitiendo recorrer las opciones con el *joystick*.

```
void movAbajo(void){
  ...
  }else if(manual==true){
    if(x==72&&y==21&&sel==false){
      restarF();
    }
  ...
}

void movArriba(void){
  ...
  }else if(manual==true){
    if(x==72&&y==21&&sel==false){
      sumarF();
  }
  ...
}
```

Las dos primeras selecciones, ubicadas en la parte superior de la pantalla, corresponden a `V1` y `V2`. A través de estas opciones, es posible que el usuario abra la primera válvula, la segunda, o ambas simultáneamente. La activación de cualquiera de estas selecciones pone en funcionamiento tanto el compresor como el ventilador, permitiendo realizar pruebas de flujo en ambos conductos. Al posicionarse sobre `V1` o `V2` y presionar el *joystick*, se invoca la función `selecc()`, la cual identifica la opción elegida.

```
void selecc(void){
...
    }else if(manual==true){
          if(x==23 && y==1 && sel1==false){
            ...
          }else if(x==23 && y==1 && sel1==true){
            ...
          }
          if(x==72 && y==1 && sel2==false){
            ...
          }else if(x==72 && y==1 && sel2==true){
            ...
          }
```

```
}
```

Cada una de estas opciones cuenta con dos booleanos asociados: el primero indica cuándo se debe dibujar un círculo blanco que representa visualmente que la válvula está activa y funcionando; el segundo actúa como conmutador, señalando que la opción ha sido seleccionada correctamente. Si se vuelve a seleccionar la misma opción, entonces la válvula se apaga y conmutan los booleanos para no mostrar el círculo indicador y conocer el estado de la selección.

```
    ...
          if(x==23 && y==1 && sel1==false){
            bolav1=false;
            sel1=true;
          }
    ...
```

Debajo de las selecciones de válvulas (V1 y V2) se encuentra la opción para cambiar el valor del flujo. Al seleccionar esta opción y presionar el *joystick*, se accede a un arreglo de valores predeterminados de flujo. En esta etapa, el movimiento del cursor, como se muestra en la función `movArriba()`, se bloquea para evitar que el usuario navegue a otras opciones mientras define el valor de flujo.

```
void selecc(void){
    ...
      if(x==72 && y==21 && sel==true){
        ...
        sel=false;
      }
    ...
}
void movArriba(void){
  ...
  else if(manual==true){
    if(x==72&&y==21&&sel==false){
      sumarF();
    }
  ...
}
```

Dentro del arreglo, es posible que el usuario recorra los valores disponibles desplazando el *joystick* hacia arriba o hacia abajo. Este movimiento permite seleccionar de forma precisa el flujo deseado. Una vez que se ha llegado al valor adecuado, se presiona nuevamente el *joystick* para confirmar la selección. Al realizar esta acción, se desbloquea el movimiento del cursor, permitiendo regresar a las demás opciones de la interfaz, y se modifica la variable *Setpoint* al nuevo valor de flujo.

```
void selecc(void){
    ...
      }else if(x==72 && y==21 && sel==false){
        bolaf=true;
        sel=true;
        Setpoint = flujol[ifo].toDouble();
        ...
      }
```

```
}
```

Como indicativo visual de que el valor de flujo fue modificado con éxito, se dibuja un círculo blanco al costado del valor seleccionado. Esto brinda retroalimentación inmediata al usuario y asegura que el nuevo valor ha sido registrado correctamente. Además, vía `BT` se envía un mensaje indicando que el valor de flujo se ha modificado y se muestra el nuevo valor.

```
void paginaMan(void){
  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);
  ...
  if(bolaf==true){
    display.drawRoundRect(113, 30, 7, 7, 45, SSD1306_WHITE);
  }
  ...
}
```

Si el usuario vuelve a presionar la selección de cambio de flujo, el sistema restablece sus estados iniciales: el booleano correspondiente al círculo indicador se asigna a `false`, el estado de selección se desactiva y el índice del arreglo de valores de flujo se reinicia a cero. Esto garantiza que, al volver a ingresar, la lista de opciones de flujo se muestre desde el principio.

```
void selecc(void){
    ...
      if(x==72 && y==21 && sel==true){
        ifo=0;
        bolaf=false;
        sel=false;
      }
    ...
}
```

Finalmente, en la parte inferior de la pantalla se encuentra la opción para salir de la página actual, representada mediante una flecha apuntando hacia la izquierda. Al seleccionar esta opción, todos los booleanos asociados a las distintas páginas —excepto el de la página de menú— se asignan a `false`, mientras que el booleano correspondiente al menú principal se establece en `true`. Además, todos los booleanos que se usaron en esta página se cambian a sus valores iniciales. Con esto, la interfaz regresa al menú principal en donde el usuario seguirá desplazándose, y en caso de que vuelva a ingresar a la página `MANUAL`, ésta no mostrará ningún cambio realizado y se cerrarán las válvulas junto con el apagado de la bomba.

```
void selecc(void){
    ...
      if(x==49 && y==40){
          sel=true;
          sel1=true;
          sel2=true;
          menu=true;
          manual=false;
          bolav1=false;
```

```
            bolav2=false;
          }
    ...
}
```

8.3. Página config.

Esta página permite al usuario programar la fecha y hora de una prueba utilizando el flujo e intervalo predeterminados. Cuenta con seis opciones:

1. Configurar el día de la prueba.
2. Configurar el mes de la prueba.
3. Configurar la hora de inicio.
4. Configurar los minutos de inicio.
5. Salir y guardar la fecha y hora programadas.
6. Salir sin almacenar los cambios realizados.

Al igual que la página anterior, incorpora un cursor tipo marco para resaltar las opciones, indicadores circulares y arreglos desplegables son capaces de recorrerse, y sigue el mismo procedimiento para limpiar el búfer, definir el tamaño del texto y establecer el color.

```
void paginaOpc(void){
  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);
  ...
  display.setTextSize(2);
  display.setCursor(27,6);
  display.println(dia[id]+"/"+mes[im]);
  display.setTextSize(2);
  display.setCursor(27,27);
  display.println(hora[ih]+":"+minuto[imn]);
  display.setCursor(27,44);
  display.println("<-  OK");
  display.display();
}
```

De tal forma que la página se ve así:

Figura 5.19: Página de config. en la interfaz de la pantalla OLED

Esta página permite el movimiento en las cuatro direcciones: arriba, abajo, derecha e izquierda. Por ello, fue necesario modificar la función de movimiento para abarcar todas estas direcciones. El desplazamiento hacia la derecha o la izquierda resultó sencillo de implementar, ya que, al igual que en versiones anteriores, solo fue necesario sumar o restar un valor en el eje X del cursor para lograr el movimiento lateral.

```
void movIzquierda(void){
  if(menu==false){
    if(manual==false){
      if(x==72){
      x=x-49;
      }
    }
...
void movDerecha(void){
  if(menu==false){
    if(manual==false){
      if(x==23){
      x=x+49;
      }
    }
...
```

Las cuatro opciones para modificar la fecha y hora de la prueba funcionan bajo el mismo concepto. Cuando el marco resalta una opción y el usuario hace clic, se bloquea el movimiento del cursor y se habilita el desplazamiento dentro del arreglo que contiene los valores posibles para la variable seleccionada.

```
void movArriba(void){
...
  }else if(manual==false){
    if(x==23&&y==1&&sel==false){
      sumarDia();
    }
...
```

Para gestionar este comportamiento, se utiliza una variable booleana llamada `sel`, que permite al programa identificar cuándo se ha hecho clic sobre una opción para

modificar su valor. Esta variable conmuta su estado (de verdadero a falso y viceversa) con cada clic realizado dentro de esta página.

```
void selecc(void){
...
  if(sel==true){
    sel=false;
  }else{
    sel=true;
  }
...
```

En el caso de la selección “DIA”, se utiliza un arreglo de 32 cadenas. La primera cadena corresponde al letrero de la selección, mientras que las restantes representan los días del mes, desde 01 hasta 31.

```
void OLED_Init() {
  dia[0]="Dia";dia[1]="01 ";dia[2]="02 ";dia[3]="03 ";dia[4]="04 ";
  dia[5]="05 ";dia[6]="06 ";dia[7]="07 ";dia[8]="08 ";dia[9]="09 ";
  dia[10]="10 ";dia[11]="11 ";dia[12]="12 ";dia[13]="13 ";
  dia[14]="14 ";dia[15]="15 ";dia[16]="16 ";dia[17]="17 ";
  dia[18]="18 ";dia[19]="19 ";dia[20]="20 ";dia[21]="21 ";
  dia[22]="22 ";dia[23]="23 ";dia[24]="24 ";dia[25]="25 ";
  dia[26]="26 ";dia[27]="27 ";dia[28]="28 ";dia[29]="29 ";
  dia[30]="30 ";dia[31]="31 ";
  ...
```

De manera similar, para la selección “MES” se emplea un arreglo de 13 cadenas. La primera es el letrero “MES” y las siguientes doce corresponden a los meses del año, numerados del 01 al 12.

```
void OLED_Init() {
  ...
  mes[0]="Mes";mes[1]="01 ";mes[2]="02 ";mes[3]="03 ";mes[4]="04 ";
  mes[5]="05 ";mes[6]="06 ";mes[7]="07 ";mes[8]="08 ";mes[9]="09 ";
  mes[10]="10 ";mes[11]="11 ";mes[12]="12 ";
  ...
```

En la selección “Hr” se utiliza un arreglo de 25 cadenas: la primera es el letrero “Hr” y las restantes muestran las horas desde 00 hasta 23.

```
void OLED_Init() {
  ...
  hora[0]="Hr ";hora[1]="01 ";hora[2]="02 ";hora[3]="03 ";
  hora[4]="04 ";hora[5]="05 ";hora[6]="06 ";hora[7]="07 ";
  hora[8]="08 ";hora[9]="09 ";hora[10]="10 ";hora[11]="11 ";
  hora[12]="12 ";hora[13]="13 ";hora[14]="14 ";hora[15]="15 ";
  hora[16]="16 ";hora[17]="17 ";hora[18]="18 ";hora[19]="19 ";
  hora[20]="20 ";hora[21]="21 ";hora[22]="22 ";hora[23]="23 ";
  hora[24]="00 ";
  ...
```

Por último, la selección “Min” cuenta con un arreglo de 61 cadenas. La primera es el letrero “Min” y las siguientes representan los minutos desde 00 hasta 59.

```
void OLED_Init() {
```

```
  ...
  minuto[0]="Min";minuto[1]="01 ";minuto[2]="02 ";minuto[3]="03 ";
  minuto[4]="04 ";minuto[5]="05 ";minuto[6]="06 ";minuto[7]="07 ";
  minuto[8]="08 ";minuto[9]="09 ";minuto[10]="10 ";minuto[11]="11 ";
  ...
  minuto[60]="00 ";
  ...
```

Para bloquear el movimiento del cursor hacia los lados, se agregó un condicional que verifica si ninguna opción está seleccionada. Si esta condición es falsa, el sistema no ejecuta la función de movimiento hacia la derecha o izquierda. Por el contrario, si la condición es verdadera, el cursor se mueve libremente en esas direcciones.

```
    switch (varr) {
    ...
      case 2:
        if(sel==true){
        movDerecha();
        }
        break;
      case 3:
        if(sel==true){
          movIzquierda();
        }
        break;
    ...
```

Para el movimiento vertical (hacia arriba y hacia abajo), se incluyó un condicional dentro de las funciones correspondientes. Este condicional verifica si alguna opción está seleccionada y analiza la posición actual del cursor. Con esta información, el programa decide si debe simplemente sumar o restar un valor definido en el eje Y, o si debe ingresar a una función específica para recorrer el arreglo de valores de la opción seleccionada. Aquí un ejemplo con la suma de las variables Día y Mes:

```
void movArriba(void){
  ...
  }else if(manual==false){
    if(x==23&&y==1&&sel==false){
      sumarDia();
    }
    if(x==72&&y==1&&sel==false){
      sumarMes();
    }
  ...
```

Las funciones que recorren los arreglos fueron nombradas como `sumar___()` o `restar___()`, según el arreglo que se desea recorrer. En estas funciones se utiliza una variable que representa el índice del arreglo, la cual se incrementa o decrementa en 1 respecto a su valor anterior. Si el índice alcanza el límite del arreglo en cualquiera de los extremos, se reinicia al otro extremo, permitiendo un recorrido circular.

```
void sumarDia(void){
    if(id==31){
      id=1;
```

```
    }else{
      id++;
    }
}
```

Por ejemplo, si el usuario modifica la selección "DIA", llega al valor 31 y continúa moviendo el *joystick* hacia arriba, la selección cambiará a 01, permitiendo que el desplazamiento continúe. De manera similar, si el usuario está en 01 y se desplaza hacia abajo, la selección mostrará 31.

Una vez que se recorre cualquiera de las cuatro selecciones y se elige una opción, al hacer clic nuevamente se desbloquea el movimiento del cursor y aparece un círculo blanco junto a la opción elegida, indicando que ha sido seleccionada. Este comportamiento se controla mediante variables booleanas, que permiten al programa saber cuándo debe dibujarse el círculo y cuándo no. Si se vuelve a seleccionar alguna opción, el booleano correspondiente se cambia a `false` y los índices de los arreglos se reinician a 1, mostrando nuevamente el primer valor disponible para cada variable, como se observa en el ejemplo con la opción "Día".

```
void selecc(void){
  ...
  }else if(datos==false&&manual==false){
    if(x==23&&y==1&&sel==true){
      id=1;
      bolad=false;
  ...
    if(x==23&&y==1&&sel==false){
      bolad=true;
    }
  ...
```

Una vez que todas las selecciones han sido modificadas, cuentan con un valor asignado y el círculo indicador está dibujado, es posible para el usuario usar la opción "OK". Al seleccionar esta opción, el sistema verifica que los arreglos de las cuatro selecciones (DIA, MES, Hr y Min) se encuentren en una posición mayor a 0, es decir, que no muestren el letrero de la selección, sino un valor numérico válido.

```
void selecc(void){
  ...
 if(x==72&&y==40&&sel==true){
      if(im>0&&imn>0&&id>0&&ih>0){
        ...
      }
  ...
```

Si esta condición se cumple, las variables `diaON`, `mesON`, `horaON` y `minON` se actualizan con los valores correspondientes de los arreglos en su posición seleccionada, convertidos a enteros. Posteriormente, se envía un mensaje por *Bluetooth* (BT) indicando la fecha, hora y flujo programados para la prueba.

```
void selecc(void){
  ...
 if(x==72&&y==40&&sel==true){
```

```
      if(im>0&&imn>0&&id>0&&ih>0){
        diaON=dia[id].toInt();mesON=mes[im].toInt();
        horaON=hora[ih].toInt(); minON=minuto[imn].toInt();
        ...
      }
  ...
```

Además, se habilitan las funciones de control del compresor y las válvulas, y se inicializan las variables necesarias para garantizar que la prueba programada se ejecute correctamente. Finalmente, se activa la página del menú principal (cambiando su estado a `true`). Como la página `CONFIG` no cuenta con una variable booleana propia, ya que es posible controlarse observando el estado de las demás variables, no es necesario establecer a `false` ninguna otra página.

```
void selecc(void){
  ...
 if(x==72&&y==40&&sel==true){
      if(im>0&&imn>0&&id>0&&ih>0){
        ...
        a=0;b=12;
        ContrBombBool = true;
        ContrBomb = true;

        sel=false;
        menu=true;
        ...
      }
```

Si la condición de que todas las selecciones han sido modificadas es falsa, no se realiza ningún cambio y este clic no se considera como una selección válida. En este caso, la variable booleana encargada de indicar si alguna de las selecciones está en proceso de modificación se mantiene en false.

```
void selecc(void){
  ...
 if(x==72&&y==40&&sel==true){
      if(im>0&&imn>0&&id>0&&ih>0){
        ...
      }else{
        sel=false;
      }
    ...
```

8.4. Página datos

La página de datos tiene como única función desplegar información. Por esta razón, en esta sección no hay cursor ni opciones seleccionables. Dado que se cuenta con varios sensores, fue necesario utilizar múltiples páginas para aprovechar mejor el espacio disponible y garantizar que el tamaño de los caracteres fuera adecuado para su correcta visualización.

Cada una de estas páginas utiliza las mismas funciones que las anteriores. Se les asigna una variable booleana que se encarga de indicar al sistema qué página se está mostrando en el display. A continuación, se limpia el buffer del display, se define el tamaño del texto y se posiciona el cursor de texto en la coordenada inicial (0, 0). Una vez realizados estos pasos, se imprime en pantalla el letrero correspondiente a la variable que se desea mostrar, seguido del valor actual de dicha variable, el cual es obtenido mediante una función encargada de solicitar datos a los sensores cada 5 segundos.

```
void paginDatos1(){
  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(2);
  display.setCursor(0,0);
  ...
}
```

Página 1 de datos

La página 1 de datos muestra las siguientes variables:

- **Temperatura atmosférica**, obtenida a través del sensor `SHT30`, con sus unidades correspondientes.
- **Humedad atmosférica**, también obtenida mediante el sensor `SHT30`, con sus respectivas unidades.

En la parte inferior de la pantalla se dejó un renglón vacío para mantener un orden visual y asegurar que solo estas dos variables se presenten en esta página. Además, se modificó el tamaño de la fuente a 1 para incluir la hora proporcionada por el reloj en tiempo real (`RTC`). Utilizando espacios, se posicionó un índice en la parte inferior indicando que se trata de la página 1 de 6.

```
void paginDatos1(){
  ...
  display.print("Temp=");
  display.print(T,1);
  display.println("C");
  display.print("Hum=");
  display.print(H);
  display.println("%");
  display.setTextSize(1);
  display.println("");
  display.println("");
  display.print("\n");
  display.print(fecha.hour());
  display.print(":");
  display.print(fecha.minute());
  display.print("              1/6");
  display.display();
}
```

Página 2 de datos

La página 2 de datos funciona de manera similar a la anterior, pero en este caso se despliegan las siguientes variables:

- **Temperatura del primer termopar**, que mide la temperatura de uno de los trenes, en grados centígrados (°C).
- **Temperatura del segundo termopar**, que mide la temperatura del otro tren, también en grados centígrados (°C).

En la parte inferior de la pantalla se incluye el índice que indica que se trata de la página 2 de 6.

```
void paginDatos2(){
  ...
  display.print("TT1=");
  display.print(TT_1,1);
  display.println("C");
  display.print("TT2=");
  display.print(TT_2,1);
  display.println("C");
  display.setTextSize(1);
  display.println("");
  display.println("");
  display.print("\n");
  display.print(fecha.hour());
  display.print(":");
  display.print(fecha.minute());
  display.print("              2/6");
  display.display();
}
```

Página 3 de datos

La página 3 de datos muestra tres valores:

- **Índice de calidad del aire**, proporcionado por el sensor SEN0392. Este valor es adimensional, sin unidades.
- **Flujo medido por el flujómetro analógico**, expresado en mililitros por segundo (ml/s), considerando que el rango de operación del compresor es de 20 a 200 ml/s.
- **Voltaje de la batería**, obtenido mediante un divisor de voltaje, expresado en [V], ya que su rango de operación va de 12[V] a 14[V].

En este caso, el índice en la parte inferior indica que se trata de la página 3 de 6.

```
void paginDatos3(){
  ...
  display.print("ICA=");
  display.println(CA);
  display.print("F=");
  display.print(F);
  display.println("mL/m");
  display.print("VBat=");
  display.print(VB,1);
  display.println("V");
  display.setTextSize(1);
  display.print("\n");
  display.print(fecha.hour());
  display.print(":");
  display.print(fecha.minute());
  display.print("              3/6");
  display.display();
}
```

Página 4 de datos

La página 4 de datos despliega los valores obtenidos del sistema de viento:

- **Dirección del viento**, expresada en grados.
- **Velocidad del viento**, en metros por segundo (m/s).

Como solo se presentan dos variables, se incluye un renglón vacío para mantener el orden visual. En la parte inferior de la pantalla se muestra el índice correspondiente, indicando que se trata de la página 4 de 6.

```
void paginDatos4(){
  ...
  display.print("DV=");
  display.print(DirVel,1);
  display.println("G");
  display.print("VV=");
  display.print(windSpd,1);
  display.println("m/s");
  display.print(" ");
  display.setTextSize(1);
  display.println("");
  display.println("\n");
  display.print(fecha.hour());
  display.print(":");
  display.print(fecha.minute());
  display.print("              4/6");
  display.display();
}
```

Página 5 de datos

La página 5 de datos despliega los valores proporcionados por el sensor SEN55:

- **Valor de PM1.0**, en microgramos por metro cúbico ($\mu g/m^3$).
- **Valor de PM2.5**, en microgramos por metro cúbico ($\mu g/m^3$).
- **Valor de PM4.0**, en microgramos por metro cúbico ($\mu g/m^3$).

Para mostrar las unidades, se reduce el tamaño de la fuente a 1 y se colocan junto a los valores correspondientes. En la parte inferior de la pantalla, el índice indica que se trata de la página 5 de 6.

```
void paginDatos5(){
  ...
  display.print("PM");
  display.setTextSize(1);
  display.print("1.0 ");
  display.setTextSize(2);
  display.print(PM1p0,1);
  display.setTextSize(1);
  display.print("ug/m3");
  display.setTextSize(2);
  display.print("\nPM");
  display.setTextSize(1);
  display.print("2.5 ");
  display.setTextSize(2);
  display.print(PM2p5,1);
  display.setTextSize(1);
  display.print("ug/m3");
  display.setTextSize(2);
  display.print("\nPM");
  display.setTextSize(1);
  display.print("4.0 ");
  display.setTextSize(2);
  display.print(PM4p0,1);
  display.setTextSize(1);
  display.println("ug/m3");
  display.println("\n");
  display.print(fecha.hour());
  display.print(":");
  display.print(fecha.minute());
  display.print("              5/6");
  display.display();
}
```

Página 6 de datos

La página 6 de datos muestra las siguientes variables:

- **Valor de PM10.0**, expresado en microgramos por metro cúbico ($\mu g/m^3$).
- **Índice de compuestos orgánicos volátiles (COVs)**, adimensional.

- **Índice de óxidos de nitrógeno (NOX)**, también adimensional.

El índice en la parte inferior se actualiza para indicar que se trata de la página 6 de 6.

```
void paginDatos6(){
  ...
  display.print("PM");
  display.setTextSize(1);
  display.print("10  ");
  display.setTextSize(2);
  display.print(PM10p0,1);
  display.setTextSize(1);
  display.print("ug/m3");
  display.setTextSize(2);
  display.print("\nVOC=");
  display.println(voc,1);
  display.print("NOx=");
  display.println(nox,1);
  display.setTextSize(1);
  display.print("\n");
  display.print(fecha.hour());
  display.print(":");
  display.print(fecha.minute());
  display.print("              6/6");
  display.display();
}
```

De tal forma que las páginas se ven así:

((a)) Pantalla 1. ((b)) Pantalla 2. ((c)) Pantalla 3.

((d)) Pantalla 4. ((e)) Pantalla 5. ((f)) Pantalla 6.

Figura 5.20: Páginas de datos en la interfaz de la pantalla OLED

En la página de datos, no existe un cursor, y el desplazamiento entre páginas se realiza únicamente hacia la derecha o hacia la izquierda. Para gestionar esto, se modificaron las funciones de mover a la derecha y mover a la izquierda.

Dentro de estas funciones, se analiza si el sistema se encuentra actualmente en la página de datos. Si esta condición es verdadera, se ejecuta una serie de condicionales concatenados para identificar qué página de datos está actualmente visible en el display. Una vez determinada la página, se desactivan (cambiando su valor a `false`) todos los booleanos correspondientes a las demás páginas, incluyendo la que se está mostrando en ese momento. Luego, se activa (cambiando a `true`) la página siguiente o anterior, dependiendo de si el desplazamiento fue hacia la derecha o hacia la izquierda. Aquí un ejemplo con el desplazamiento a la izquierda:

```
void movIzquierda(void){
  ...
    if(datos==true){
      if (datos3==true){
        datos3=false;
        datos2=true;
        datos4=false;
        datos5=false;
        datos6=false;
        delay(100);
      }else if(datos4==true){
        datos3=true;
        datos2=false;
        datos4=false;
        datos5=false;
        datos6=false;
        delay(100);
      }else if(datos5==true){
        datos3=false;
        datos2=false;
        datos4=true;
        datos5=false;
        datos6=false;
        delay(100);
      }else if(datos6==true){
        datos3=false;
        datos2=false;
        datos4=false;
        datos5=true;
        datos6=false;
        delay(100);
      }else{
        datos2=false;
        delay(100);
      }
    }
  ...
}
```

En el caso de las páginas 1 y 6 (las páginas extremas), si el usuario intenta realizar un desplazamiento más allá de estos límites (por ejemplo, intentar mover a la izquierda desde la página 1 o mover a la derecha desde la página 6), el programa no realiza ninguna acción.

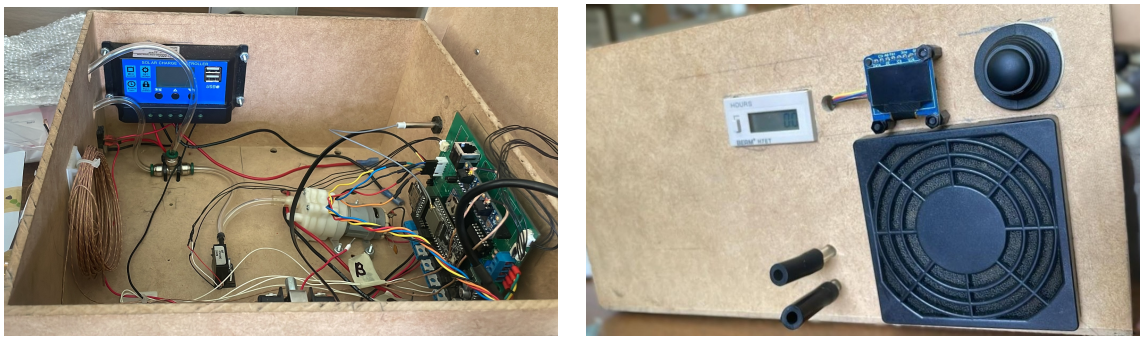
Para salir de las páginas de datos, el usuario solo necesita hacer clic con el *joystick*. En la función `selecc()`, se agregó una condición que verifica si las páginas de datos están actualmente desplegadas. Si esta condición es verdadera, la variable booleana correspondiente a la página de datos se cambia a `false`, y la variable booleana del menú principal se actualiza a `true`. Esto permite que el sistema regrese automáticamente al menú principal y comience a desplegarlo nuevamente.

```
void selecc(void){
  ...
    }else if(datos==true){
      menu=true;
      datos=false;
    }
}
```

9. Diseño de configuración física

Con la mayoría de los sensores y secuencias operando correctamente, se comenzó a diseñar la estructura del sistema. Se optó por una caja en madera MDF de tamaño comercial, suficientemente amplia para facilitar el acomodo de los componentes y su mantenimiento. Su tamaño también se definió tomando en cuenta el largo de los cartuchos utilizados para la colecta de muestras.

En el interior, la placa electrónica se fijó en una de las paredes, mientras que el controlador de energía quedó en el lado derecho. En la parte izquierda y en el exterior se colocaron la pantalla, el *joystick*, el contador de horas, el ventilador y las tomas de aire. En el suelo de la caja se instalaron la bomba, las electroválvulas y el sensor de flujo, asegurados con tornillos. La cara frontal, opuesta a la placa, se destinó a la colocación de los cartuchos.



((a)) Caja vista por dentro. ((b)) Caja vista desde la cara frontal.

Figura 5.21: Caja en MDF con componentes ensamblados

Los sensores de monitoreo ambiental se ubicaron en la parte superior, protegidos por un techo para resguardarlos de la lluvia y la exposición directa al sol.

Figura 5.22: Caja en MDF con componentes ensamblados vista de afuera

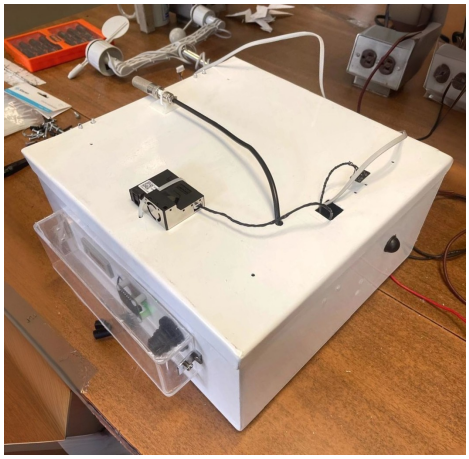
En este punto se decidió eliminar los pequeños ventiladores que salían de la compuerta *NOT*, ya que su integración con los cartuchos resultaba demasiado compleja. Inicialmente, la función de estos ventiladores era impedir el paso de aire a través del cartucho apagado mientras el otro estaba en funcionamiento. En su lugar, se replanteó la disposición de las electroválvulas, colocándolas después de la toma de aire. Con esta nueva configuración, además de permitir la selección del cartucho por el que pasará el flujo de aire, la válvula cerrada bloquea completamente el paso de aire por el cartucho inactivo.

Finalmente, se tomaron medidas precisas de cada barreno para fijar sensores, actuadores y demás componentes, preparando el diseño para una nueva versión de la caja.

Para este nuevo modelo, se decidió reubicar el ventilador junto al controlador de carga, dejando en su lugar original únicamente el contador de horas, la pantalla y el

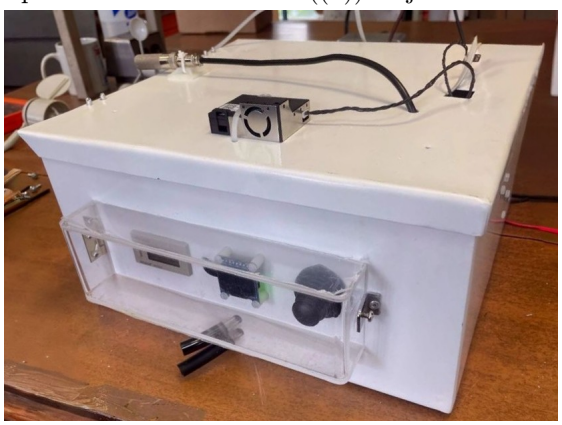
joystick. Además, se consideró añadir una protección de acrílico para resguardar estos componentes, ya que estarían completamente expuestos a la intemperie.

Dado que la caja fue fabricada externamente, al recibirla se procedió a verificar que los barrenos coincidieran con cada sensor y componente. Sin embargo, se detectaron algunas discrepancias en las posiciones y dimensiones de ciertos orificios, además de que la tapa abría hacia el lado contrario y tenía un ajuste demasiado justo. Por ello, se realizaron las correcciones necesarias y se enviaron las modificaciones al fabricante para su ajuste.



((a)) Caja vista por dentro.

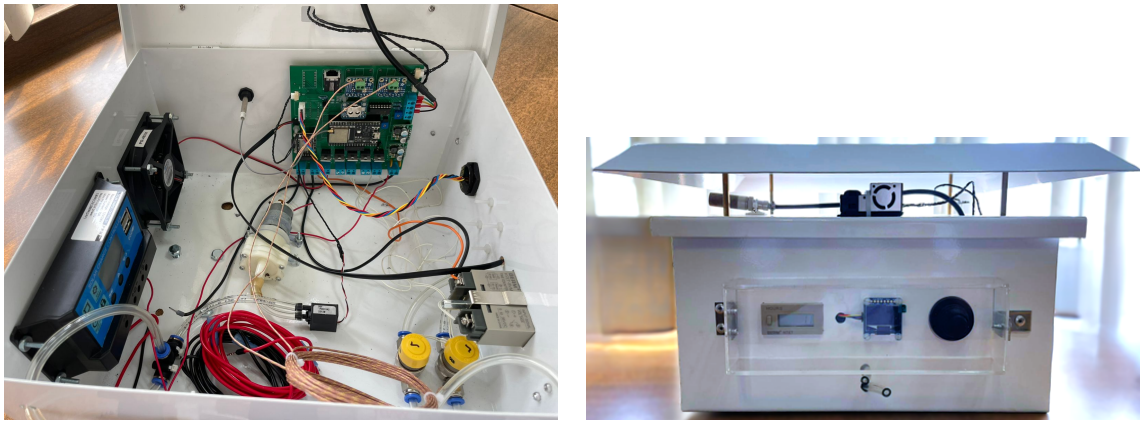
((b)) Caja vista desde la cara lateral.



((c)) Caja vista desde la cara frontal.

Figura 5.23: Caja en metal con componentes ensamblados

Con las correcciones implementadas, se procedió a instalar todos los componentes del sistema en sus posiciones correspondientes. Posteriormente, se alimentó el sistema y se verificó que todo funcionara correctamente y sin errores. Con esto, se llegó al modelo físico final del sistema, al cual denominamos **Muestreador Automático de Bajo Flujo (MABF)**.



((a)) Caja vista por dentro. ((b)) Caja vista desde la cara lateral.

Figura 5.24: Caja en metal con componentes ensamblados

En el Manual Técnico se encuentra una descripción detallada de cada uno de los componentes del sistema, así como su función y conexión dentro del muestreador. (Ver Anexo: D - Manual Técnico).

10. (Re)Diseño de aplicación móvil

Dado que era fundamental contar con una aplicación intuitiva y fácil de manejar, se optó por desarrollar la aplicación utilizando el programa gratuito MIT App Inventor, descartando el uso de la aplicación Monitor Serial Terminal para el envío de comandos. MIT App Inventor permite la programación mediante bloques, facilita la gestión de funciones con conexión *Bluetooth* (BT) y ofrece la posibilidad de crear aplicaciones con múltiples páginas.

Para la estructura de la aplicación, se adoptó un diseño similar al del display, que incluye tres páginas principales: una para pruebas, otra para programación y una tercera para la visualización de datos. Además, se agregó una página de menú principal desde la cual es posible seleccionar la sección a la que se desea acceder.

10.1. Menú principal

Esta página cuenta con tres botones que dirigen al usuario a las páginas principales, además de imágenes y títulos que mejoran la presentación y la navegación dentro del menú.

Se utilizó el bloque de inicialización de la página para cargar las imágenes de presentación y los títulos correspondientes. Asimismo, se implementaron bloques condicionales que determinan qué página debe abrirse al presionar cada uno de los botones disponibles.

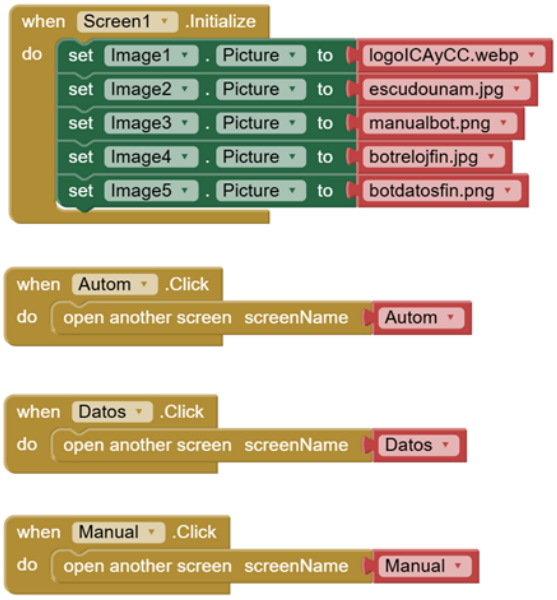


Figura 5.25: Inicialización menú principal de aplicación.

La interfaz del menú principal se muestra en la siguiente figura:

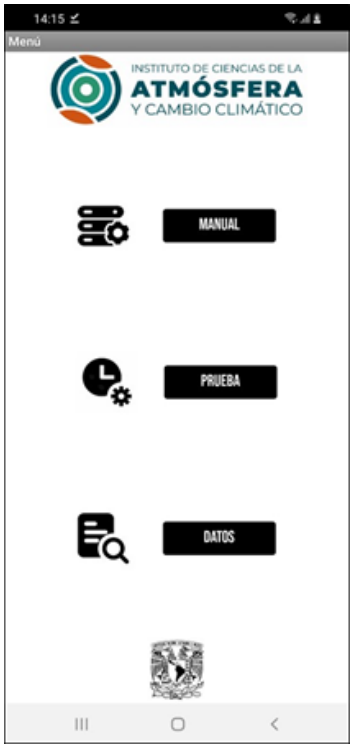


Figura 5.26: Página menú principal de aplicación.

10.2. Página manual

Esta página cuenta con una estructura de bloques extensa debido a la gran cantidad de funciones que integra. Lo primero que se implementó fue el bloque encargado de definir los elementos visuales que deben mostrarse al iniciar la página.

En este bloque, se agregó una imagen que funciona como botón para abrir la lista de dispositivos *Bluetooth* (BT) disponibles para que el usuario establezca una conexión. Además, se incluyó un letrero que indica el estado de la conexión, un botón para salir de la página y las configuraciones necesarias para definir qué opciones, textos e imágenes deben ser visibles al ingresar a esta sección.

Figura 5.27: Inicialización manual de la aplicación.

La primera pantalla que ve el usuario al ingresar en la opción manual es la siguiente:

Figura 5.28: Página manual sin conexión.

Al presionar el símbolo rojo que indica desconexión, se despliega una lista con los dispositivos disponibles. Previamente, a esta lista se le asignan los nombres de todos los dispositivos que han sido vinculados vía *Bluetooth* al teléfono.

Cuando el usuario selecciona un dispositivo con el que se pueda establecer conexión, se habilitan las funciones de la página, el letrero de estado cambia a "Conectado" y la imagen del símbolo rojo de desconexión se sustituye por una que indica conexión activa.

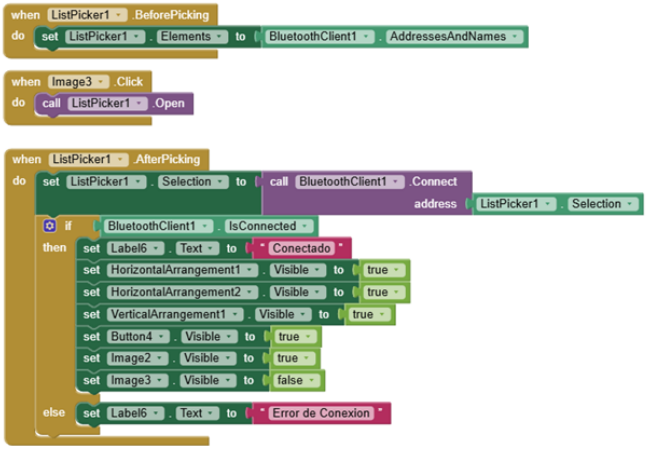


Figura 5.29: Configuración BT en la página manual.

Las funciones de esta página permiten manipular las dos válvulas, asignar un flujo objetivo al compresor y definir el tiempo de operación de las válvulas en horas. Para ello, se utilizan dos interruptores acompañados de sus respectivos letreros, los cuales determinan qué válvula se desea activar.

Estos interruptores tienen la posibilidad de generar cuatro estados posibles:

- **Solo la válvula 1 encendida**: Se activa un *slider* que permite al usuario definir cuántas horas permanecerá abierta y con flujo de aire la válvula 1.
- **Solo la válvula 2 encendida**: Se habilita un *slider* similar, pero destinado a la válvula 2.
- **Ambas válvulas encendidas**: Se muestra un *slider* que permite definir el tiempo total de operación de ambas válvulas. Estas funcionarán de forma alternada, comenzando con la válvula 1 abierta.
- **Ambos interruptores apagados**: Solo se muestra el cuadro de texto para ingresar el flujo objetivo del compresor, sin desplegar ningún *slider* ni el botón de programación.

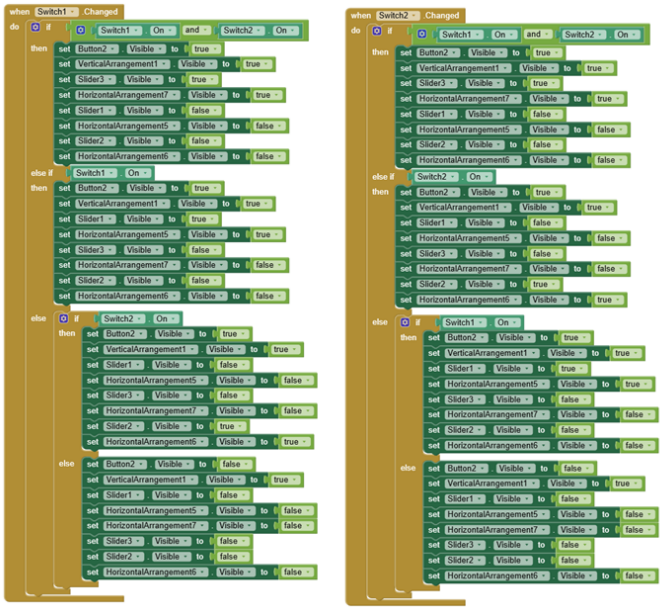


Figura 5.30: Configuración de interruptores en la página manual.

La interfaz resultante con estas opciones se muestra a continuación:

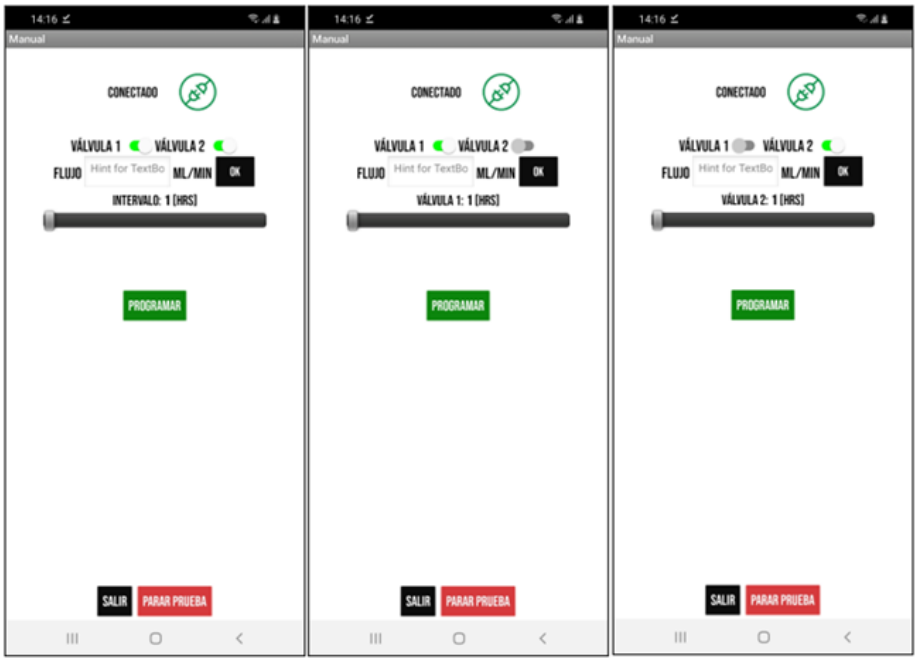


Figura 5.31: Funcionamiento de interruptores en página manual.

El botón de Programar verifica el estado de los interruptores y *sliders* para enviar un mensaje vía *Bluetooth* (BT) al microcontrolador con los datos configurados por el usuario. Este mensaje sigue una estructura diseñada para ser compatible con los

comandos previamente establecidos en la aplicación BT anterior.

El mensaje inicia con el carácter "V", que corresponde a un comando definido en la sección 5. Luego, según los interruptores activados, se añade un número que representa la configuración seleccionada:

- **1** para la válvula 1.
- **2** para la válvula 2.
- **3** para ambas válvulas.

Posteriormente, se añade el valor del tiempo de operación en horas, obtenido a partir de la posición del *slider* correspondiente.

Finalmente, se muestra un mensaje en pantalla informando al usuario que el muestreador ha sido programado correctamente.

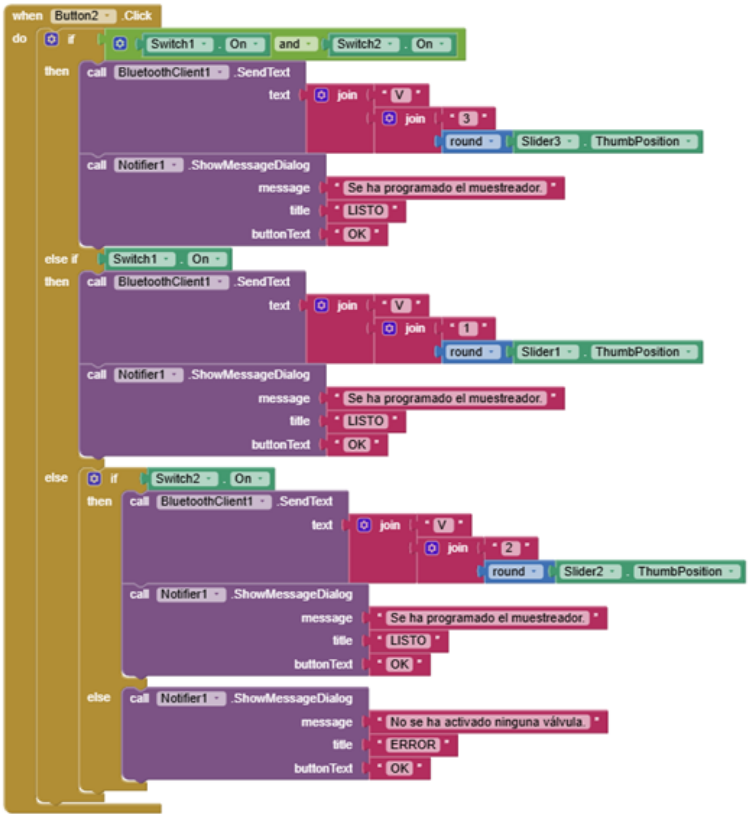


Figura 5.32: Funcionamiento de página manual.

El botón Parar Prueba envía la letra "P" vía *Bluetooth* (BT) al microcontrolador. Al recibir este comando, el sistema detiene el compresor y cierra ambas válvulas, finalizando la prueba de forma inmediata.

Por su parte, el botón "salir" habilita la página del menú principal, restaurando todas sus funciones y permitiendo al usuario navegar nuevamente por la aplicación.

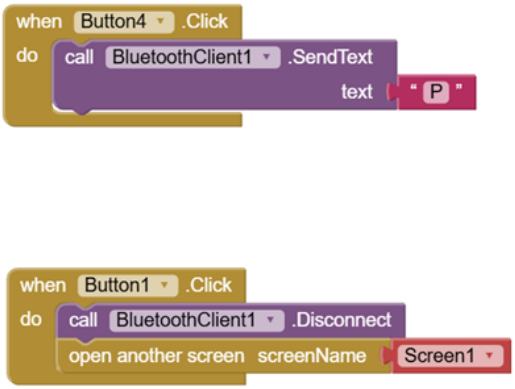


Figura 5.33: Parar prueba en página manual.

10.3. Página prueba

Esta página mantiene la misma disposición de imágenes y botones iniciales que la página manual. Al igual que en esta última, el primer paso para el usuario es establecer la conexión *Bluetooth* (BT). Una vez conectado, se habilitan y hacen visibles las demás funciones de la página, permitiendo su uso.

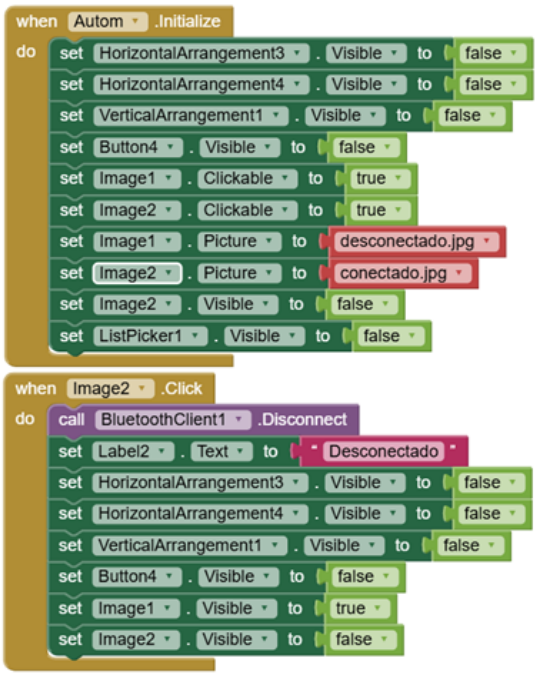


Figura 5.34: Inicialización página prueba de aplicación.

La interfaz de la página prueba sin conexión se muestra a continuación:

Figura 5.35: Página prueba sin conexión.

Al establecer la conexión *Bluetooth* (BT), las funciones de la página se activan y se vuelven visibles. Esta página incluye dos seleccionadores: uno para elegir el día, mes y año en que se desea programar la prueba, y otro para seleccionar la hora y el minuto de inicio.

Al igual que en la página manual, se incorporó un cuadro de texto que permite al usuario modificar el flujo objetivo del compresor. Además, se añadió un *slider* para ajustar el tiempo de prueba durante el cual las válvulas conmutan.

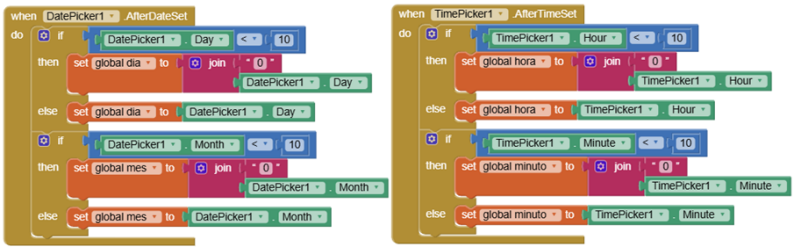


Figura 5.36: Configuración de página prueba.

La interfaz de la página prueba conectada al BT y con los seleccionadores de fecha y hora se muestra a continuación:

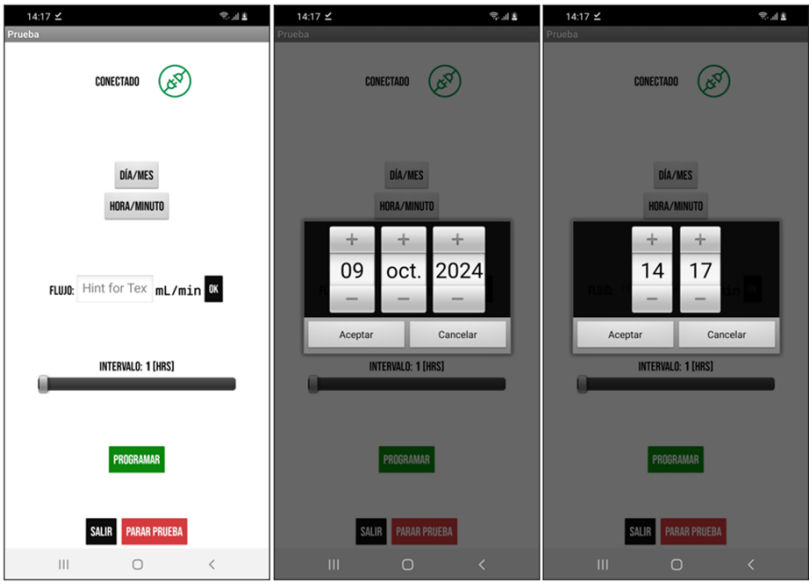


Figura 5.37: Página prueba conectada.

Una vez realizadas las modificaciones necesarias, se debe presionar el botón "programar". Este botón genera un mensaje basado en los parámetros definidos por el usuario, utilizando el comando "A" para programar una prueba. Al mensaje se le concatenan los valores de día, mes, hora y minuto seleccionados, así como el intervalo definido en el *slider*.

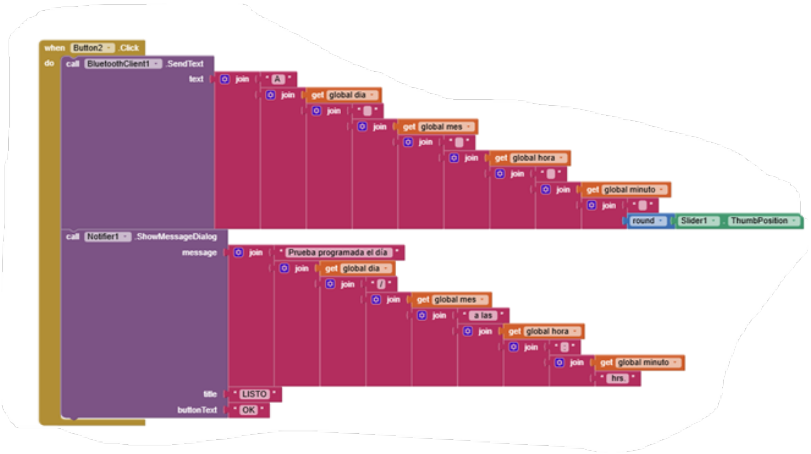


Figura 5.38: Composición del mensaje BT de página prueba.

Finalmente, se muestra un mensaje de confirmación para el usuario, indicando que la prueba se ha programado con éxito, junto con la fecha y hora registradas.

Figura 5.39: Programación exitosa en página prueba.

Los botones "salir" y "parar prueba" cuentan con la misma función y programación que en la página manual.

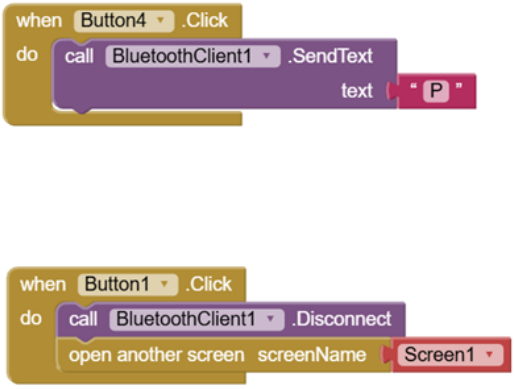


Figura 5.40: Botones "parar prueba" y "salir" de página prueba.

10.4. Página datos

Al igual que en las dos páginas anteriores, lo primero que debe hacer el usuario es conectarse al muestreador. Mientras no se establezca la conexión, no se mostrarán los letreros ni los datos recibidos.

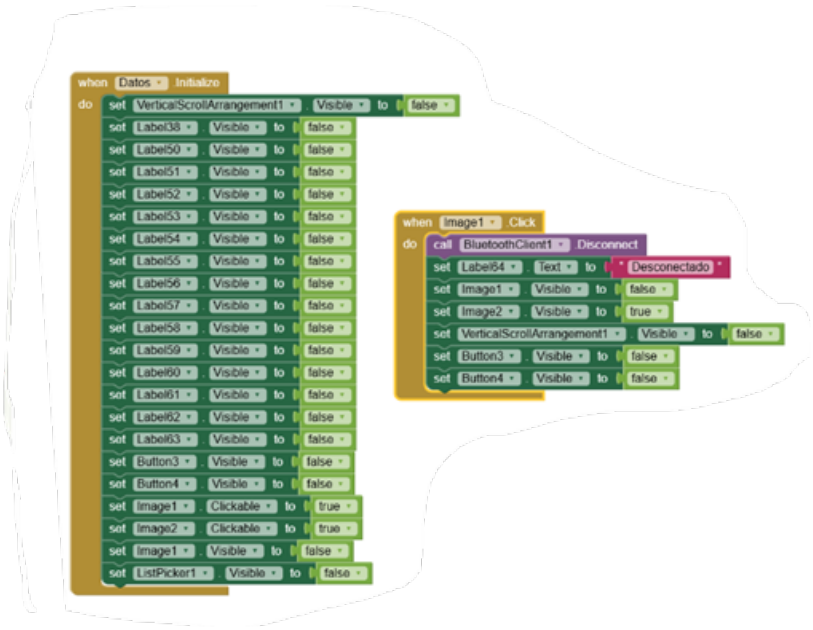


Figura 5.41: Inicialización página datos de aplicación.

Esta página utiliza un temporizador para monitorear constantemente la recepción de mensajes. Cuando se recibe un mensaje del muestreador, la aplicación lo separa en variables de tipo `string` más pequeñas cada vez que detecta una coma.

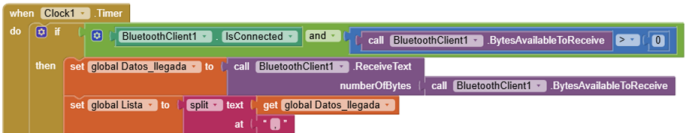


Figura 5.42: Configuración de timer en página de datos.

Luego, analiza la cantidad de estos fragmentos para verificar que el mensaje recibido sea el correcto para el despliegue de datos.

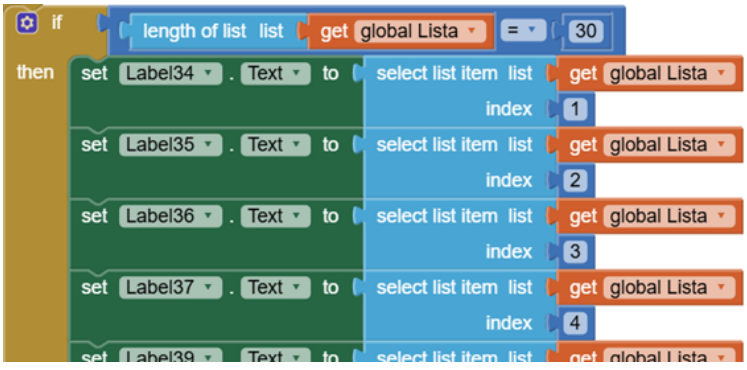


Figura 5.43: Análisis de mensaje recibido.

El mensaje que proporciona los datos a la aplicación se incorporó en la programación del muestreador para que cumpliera con el formato correspondiente. Aprovechando la función que muestra los datos en el display cada 5 segundos, se genera un mensaje utilizando la función `snprintf`, el cual se almacena en un búfer con formato separado por comas. Además, se crea otro mensaje que transmite los promedios de cada una de las variables, empleando la misma función y formato. Una vez que ambos mensajes están listos, se envían utilizando el objeto `BT_VOC` y la función `print`.

```
char* BT_Data = "%.2f,%.2f,%.2f,%.2f,%i,%i,%.2f,%.1f,
    %.2f,%.1f,%.1f,%.1f,%.1f,%.1f,";
char* BT_Data_prom = "%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,
    %.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f";
...
Datos5seg(){
  ...
  snprintf(buffer,150,BT_Data,T,H,TT_1,TT_2,CA,F,VB,
    DirVel,windSpd,PM1p0,PM2p5,PM4p0,PM10p0,voc,nox);
  BT_VOC.print(buffer);
  snprintf(buffer,150,BT_Data_prom,promT,promH,promTT_1,
    promTT_2,promCA,promF,promVB,promDirVel,promwindSpd,
    prom1p0,prom2p5,prom4p0,prom10p0,promvoc,promnox);
  BT_VOC.println(buffer);
}
```

Al recibir y separar los datos, estos se clasifican en dos tipos: datos en tiempo real y promedios. Por defecto, la pantalla muestra los datos en tiempo real, los cuales se actualizan cada 5 segundos. Se añadió un botón que permite alternar entre ambas vistas; al presionarlo, los promedios se hacen visibles y los datos en tiempo real se ocultan. De este modo, es posible que el usuario consulte tanto la información instantánea como los promedios calculados cada 15 minutos.

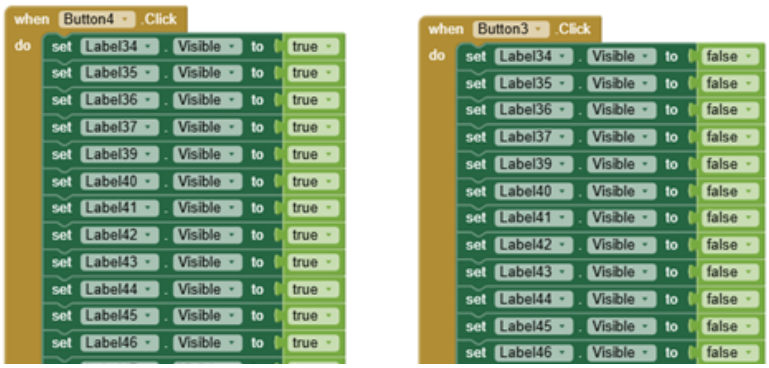
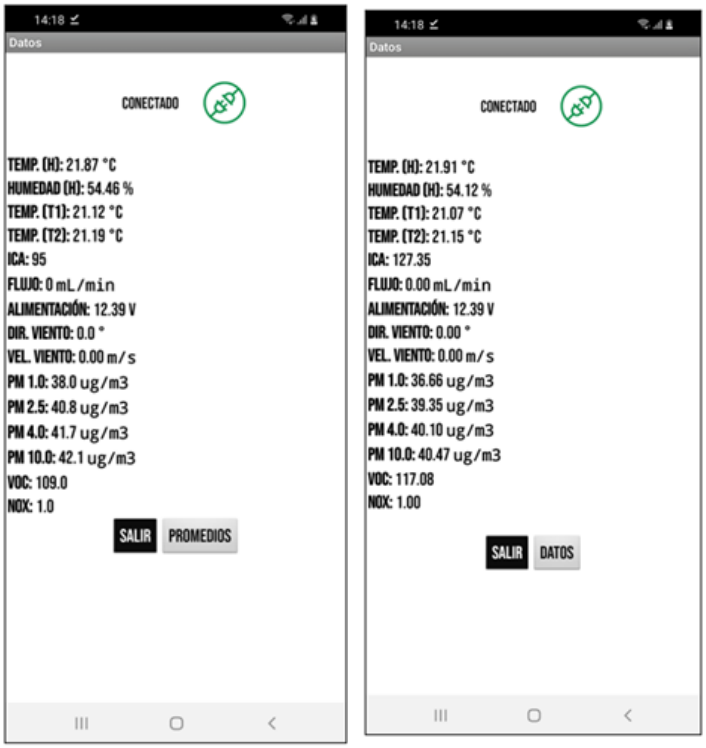


Figura 5.44: Configuración de botones en página datos.

La interfaz de la página datos y sus diferentes pantallas se muestran a continuación:



((a)) Pantalla con datos en página datos.

((b)) Pantalla con promedios en página datos.

Figura 5.45: Pantallas en página datos.

10.5. Mensajes de error

En las tres páginas que requieren conexión *Bluetooth* para habilitarse (manual, prueba y datos), se incorporó una función para la detección de errores. Esta función recibe un número enviado por el muestreador que identifica el tipo de error, y de inmediato se muestra un mensaje al usuario con la descripción correspondiente. El aviso aparece tanto al momento de establecer la conexión como en cualquier instante en que ocurra un error durante el uso de la aplicación.

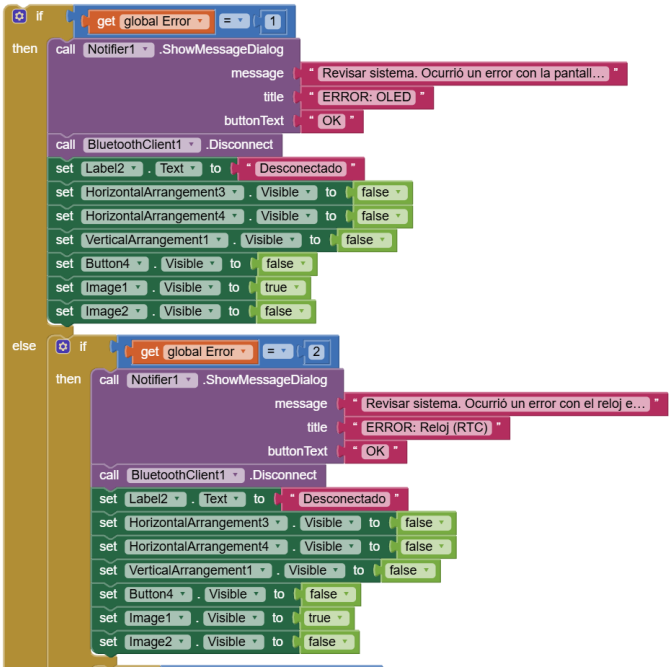


Figura 5.46: Ejemplo de configuración de dos errores.

El mensaje que indica el tipo de error fue programado directamente en el código del muestreador. Dado que algunos sensores utilizan comunicación I2C, el sistema verifica su correcto funcionamiento a través de la dirección asociada a cada uno. Si un sensor no responde conforme al protocolo, se detecta un fallo en la comunicación, lo cual es posible que se deba a un mal funcionamiento, una conexión incorrecta o la ausencia del sensor.

Cada componente que tiene la posibilidad de generar un error cuenta con una función específica que se ejecuta únicamente si se detecta dicha falla. Dentro de esta función se añadió un ciclo `while` infinito que detiene el funcionamiento del muestreador y envía el mensaje de error correspondiente. Este mensaje se transmite de forma continua con un retardo de un segundo, lo que garantiza que la aplicación lo reciba en cuanto se establezca la conexión.

```
1  void MCP9600_Init(){
2    ...
3    if (! mcp_2.begin(I2C_ADDRESS_2)){
4      ...
5      while(1){
6        BT_VOC.printf("%i,5\n",error);
7        Serial.printf("\nNúmero de error = %i\n",error);
8        delay(1000);
9      }
10   }
11 }
```

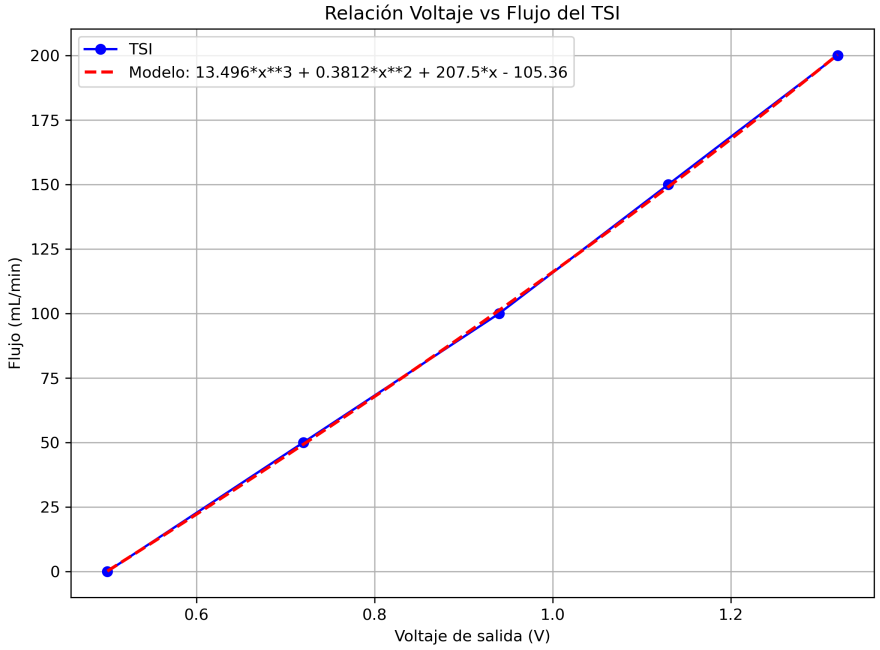
11. Validación del sistema

Para realizar las pruebas de flujo, se dejó el sistema funcionando durante 24 horas, utilizando una válvula durante las primeras 12 horas y la otra durante las siguientes 12. De esta manera, se logró capturar en los cartuchos los compuestos orgánicos volátiles (COVs) generados tanto durante el día como durante la noche.

El flujo fue verificado utilizando un flujómetro comercial (TSI), con el objetivo de validar la confiabilidad de los datos proporcionados por el sistema. Durante esta verificación, se observó una discrepancia entre los valores obtenidos y la regresión lineal proporcionada en la hoja de datos del sensor de flujo analógico. Debido a ello, se optó por realizar una calibración utilizando el flujómetro comercial (TSI), construyendo una tabla de valores de voltaje del sensor analógico frente a los valores de flujo medidos por el equipo comercial. A partir de estos datos se obtuvo una nueva regresión lineal, que permitió ajustar las mediciones del sistema con mayor precisión.

Voltaje MABF (V)	**Flujo TSI (ml/min)**
0.5	0
0.72	50
0.94	100
1.13	150
1.32	200

Tabla 5.5: Relación entre el voltaje del MABF y el flujo TSI.



$$y = 13.496x^3 + 0.3812x^2 + 207.5x - 105.36 \tag{5.15}$$

Figura 5.47: Gráfica con ecuación de regresión lineal calibrada con flujómetro comercial.

Dado que el sensor de flujo del muestreador fue previamente calibrado con el flujómetro comercial, se consideró que los datos registrados por el sistema eran confiables. Al analizar los resultados obtenidos en distintas pruebas, se comprobó que el flujo alcanzado era el requerido.

Adicionalmente, para asegurar la precisión, se realizaron verificaciones manuales colocando el flujómetro en intervalos no definidos. Estas mediciones se llevaban a cabo tras periodos prolongados de funcionamiento continuo y confirmaron que el flujo se mantenía estable en el valor deseado, que en este caso fue de aproximadamente **50 ml/min**.

Se llevaron a cabo tres semanas de pruebas con el muestreador, durante las cuales el dispositivo fue instalado en el techo del instituto para recolectar datos meteorológicos y realizar pruebas de flujo. Entre el 3 y el 25 de noviembre, se registraron variables como temperatura, humedad relativa, velocidad del viento y dirección del viento.

Los datos obtenidos se compararon con los registros oficiales proporcionados por la Red Universitaria de Observatorios Atmosféricos (RUOA), que cuenta con instrumentos de medición instalados en el mismo sitio donde se colocó el muestreador.

11.1. Temperatura: comparación RUOA vs. Muestreador

Para las mediciones de temperatura se utilizaron los datos obtenidos mediante el sensor SHT30. Al compararlos con los datos de la RUOA, se obtuvo una correlación de Pearson muy fuerte y positiva de 0.991, lo que indica un comportamiento similar entre ambas mediciones.

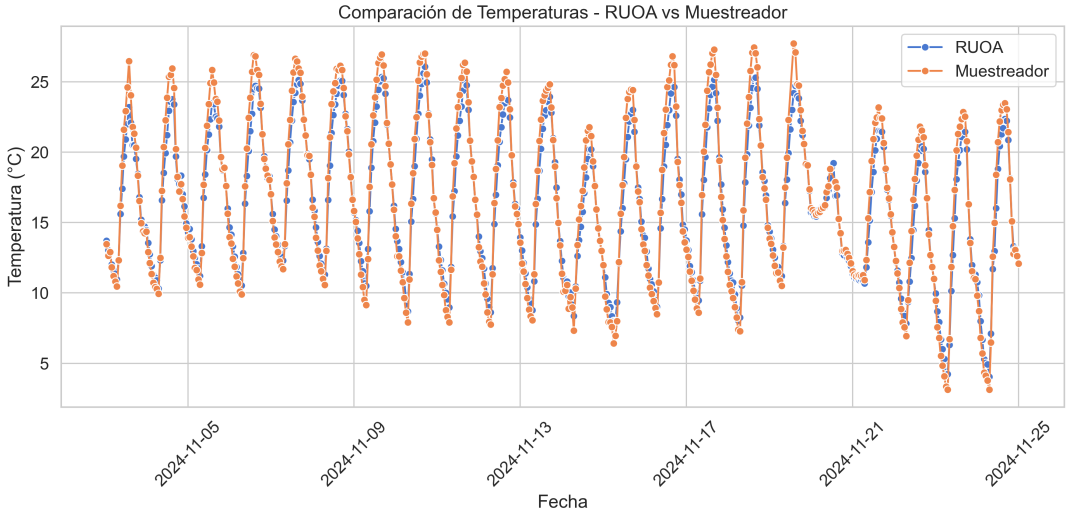


Figura 5.48: Gráfica de temperatura: RUOA vs. Muestreador.

El análisis de errores mostró un error absoluto medio (MAE) de 0.93 °C y un error cuadrático medio (RMSE) de 1.20 °C. Si bien estos valores no cumplen con los límites establecidos por la norma mexicana para estaciones meteorológicas de referencia, se consideran aceptables dentro del contexto de sensores de bajo costo, ya que permiten una representación adecuada del comportamiento térmico ambiental.

11.2. Humedad: comparación RUOA vs. Muestreador

De forma similar, las mediciones de humedad se obtuvieron con el sensor SHT30. Se alcanzó una correlación de Pearson muy fuerte y positiva de 0.991, lo que indica un comportamiento altamente consistente con los datos de referencia.

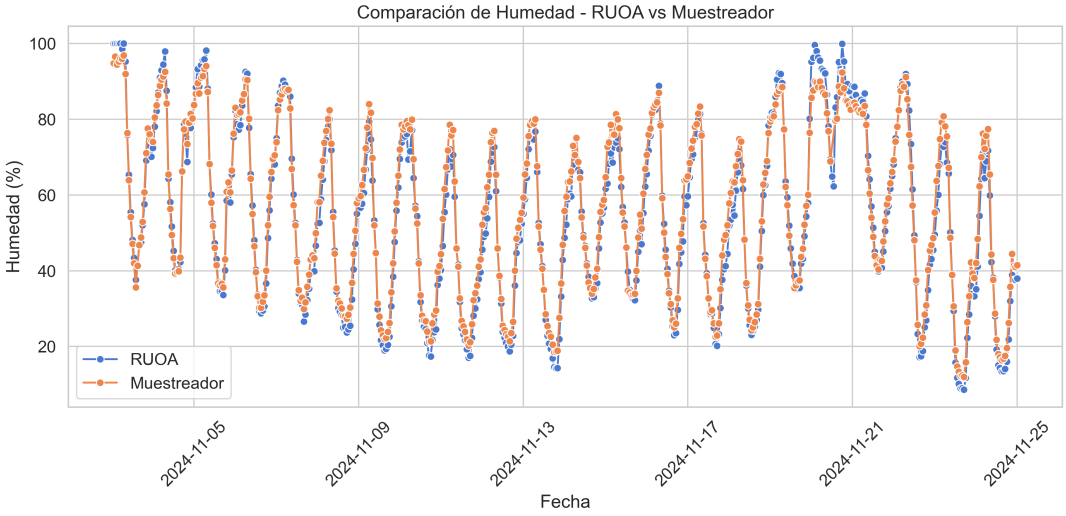


Figura 5.49: Comparación de humedad relativa: RUOA vs. Muestreador.

El error absoluto medio (MAE) fue de 3.10 %, mientras que el error cuadrático medio (RMSE) alcanzó un valor de 3.71 %. Al igual que la temperatura, estos valores no cumplen con los requisitos establecidos por la norma oficial mexicana mencionada anteriormente. Sin embargo estos resultados se consideran aceptables dentro del contexto de sensores de bajo costo, permitiendo que el dispositivo funcione como un indicador confiable de la humedad relativa ambiental.

11.3. Velocidad de viento: comparación RUOA vs. Muestreador

Los datos de velocidad del viento fueron recopilados mediante el sensor SEN15901. Al compararlos con los datos proporcionados por la RUOA, se obtuvo una correlación de Pearson de 0.878, lo cual representa una correlación fuerte, aunque ligeramente menor a lo esperado.

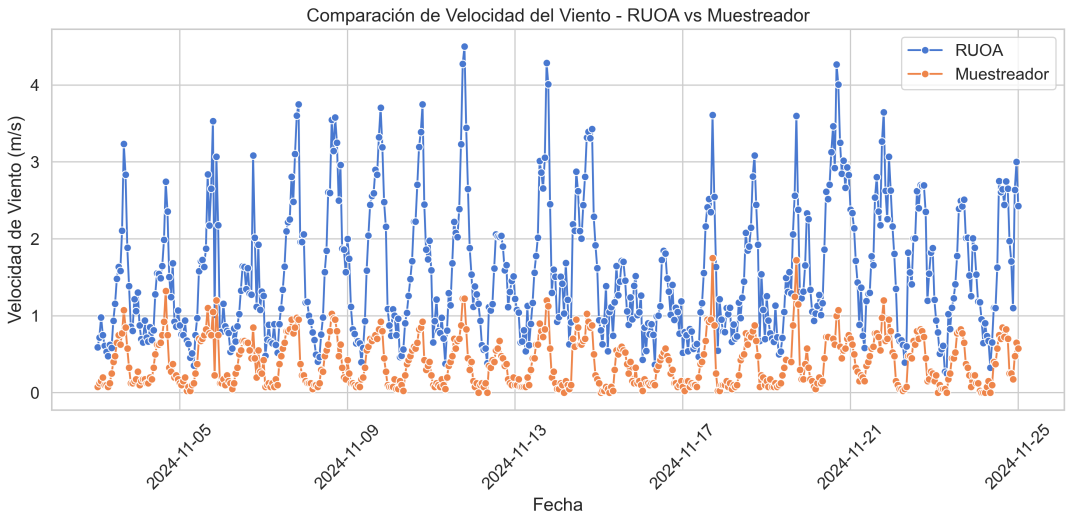


Figura 5.50: Comparación de velocidad del viento: RUOA vs. Muestreador.

Al analizar la gráfica, se observa que ambas series presentan un comportamiento similar, pero con una constante de escala diferente. Se consideró que la diferencia de altura entre ambos dispositivos pudo haber afectado las mediciones de esta variable atmosférica. Por este motivo, se decidió, como solución rápida, agregar una constante de escala para que aumentara el valor de las mediciones y posteriormente recalibrar el sensor de velocidad del viento utilizando el túnel de viento proporcionado por el Instituto de la Facultad de Ingeniería.

11.3.1. Recalibración del Anemómetro

Dentro del Laboratorio abierto a la innovación y al fortalecimiento de la infraestructura en México (LemAT), se llevó a cabo la calibración del sistema de medición de velocidad del viento. Para ello, se instaló el sensor de viento dentro del túnel de viento, mientras que el muestreador permaneció ubicado en el exterior.

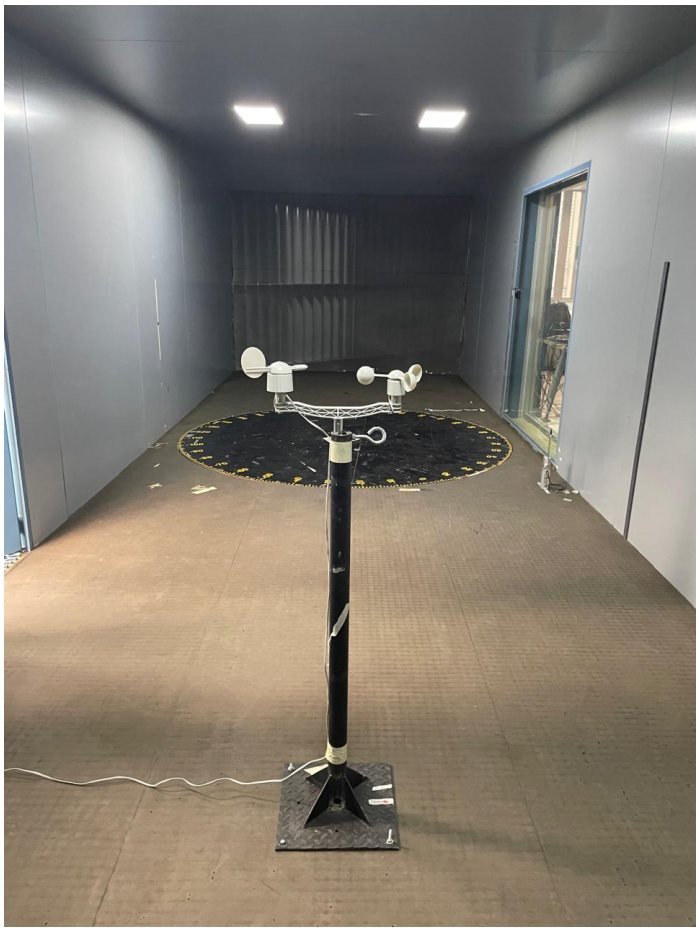


Figura 5.51: Sistema de viento instalado en el túnel de viento.

Con el túnel de viento operando en condiciones controladas (encendido y cerrado), se monitorearon en tiempo real los datos obtenidos por el anemómetro. En una primera etapa, se realizó una prueba comparativa para observar el comportamiento de las mediciones del sensor frente a los valores de referencia proporcionados por el túnel.

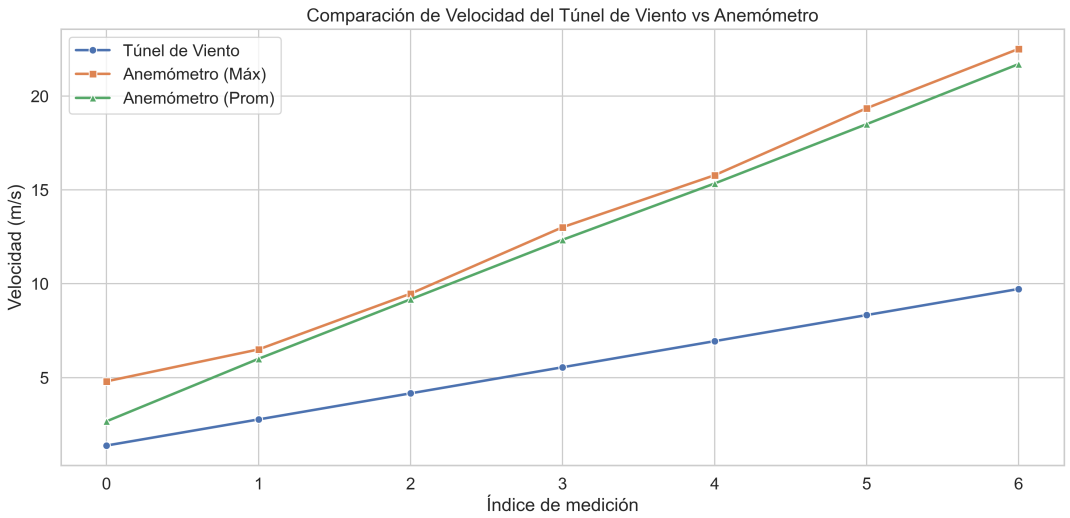


Figura 5.52: Comparación entre el anemómetro descalibrado y el túnel de viento.

Al analizar la relación entre los promedios registrados por el anemómetro y los valores de referencia del túnel, se determinó que las mediciones del sensor eran aproximadamente 2.2 veces mayores. Para corregir esta desviación, se incorporó en el código una división por dicho factor dentro de la función encargada de procesar los datos del anemómetro. Posteriormente, se realizó una nueva prueba en el túnel con la corrección aplicada, cuyos resultados se presentan a continuación:

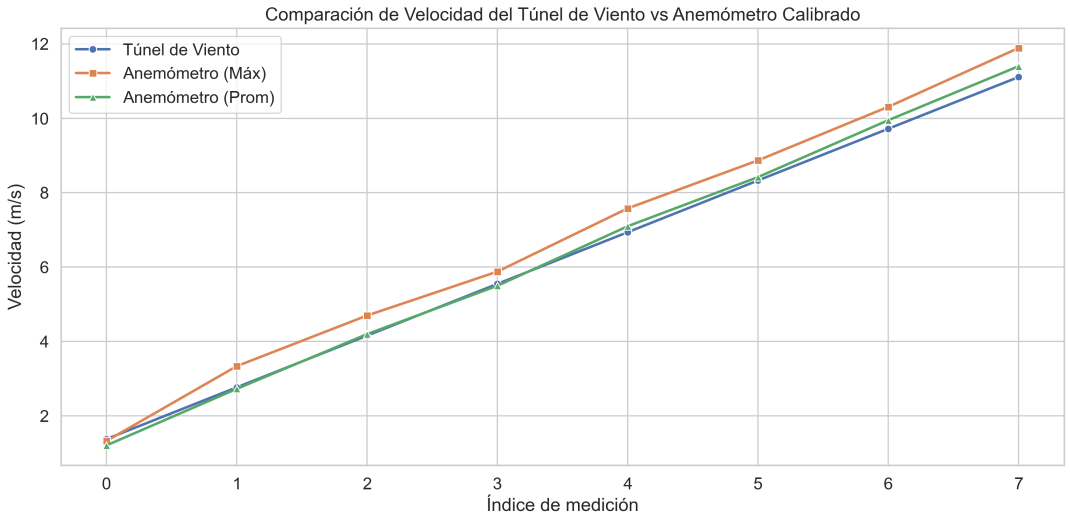


Figura 5.53: Comparación entre el anemómetro calibrado y el túnel de viento.

En la Figura 5.53 se observa cómo el anemómetro, una vez calibrado, presenta mediciones mucho más precisas y confiables. Se obtuvo un error absoluto medio (MAE)

de 0.13 m/s para los valores promedio, y de 0.50 m/s para los valores máximos. Con estos resultados, es posible concluir que el sistema de medición de velocidad de viento queda calibrado y apto para mediciones atmosféricas confiables.

11.4. Flujo: comparación Muestreador vs. Valor Objetivo

Por último, se verificó que el muestreador lograra mantener el valor objetivo durante pruebas completas de 24 horas y con los cartuchos colocados. Para ello, se instalaron cartuchos de prueba en el muestreador y se programó una prueba de 24 horas con el fin de observar el comportamiento del flujo.

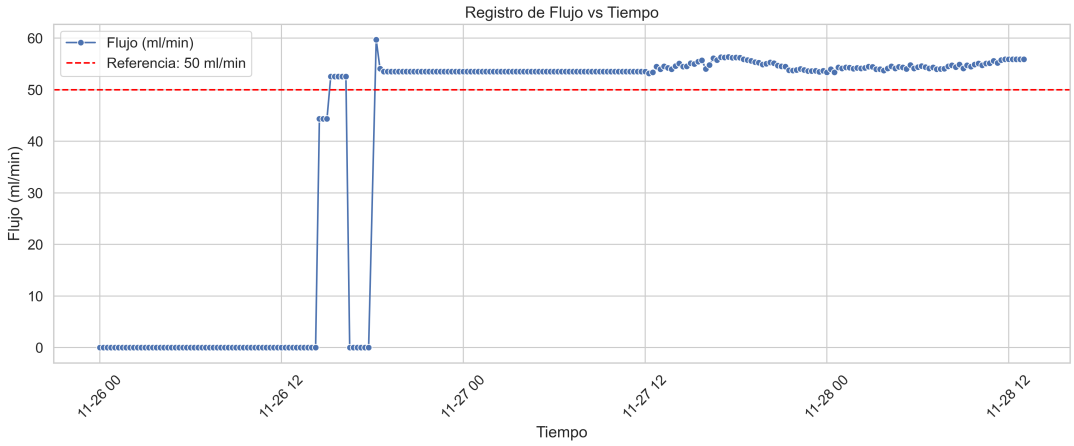


Figura 5.54: Comportamiento del Flujo vs. Valor Objetivo.

Durante estas pruebas se identificaron varios errores en el funcionamiento del muestreador. El primero se evidenció al observar que la prueba se detenía tras cumplirse exactamente una hora de operación, sin importar que la duración programada fuera mayor. Al tratarse de un problema que no se presentaba en todos los MABFs, inicialmente se asoció a un mal funcionamiento de algún componente.

Sin embargo, no fue hasta la primera prueba en campo (16 de enero) que se vinculó el fallo a un error en la programación. Al revisar el código, se detectó que el problema residía en cómo se manejaban las variables de tiempo cuando la prueba era programada justo en una hora en punto. Para resolverlo, se modificó la función que calcula la hora de apagado, agregando una condición específica para manejar este caso.

```
void OffAutomatico(){
  horaOFF = horaON + TPruebaHor;
  if(horaOFF >= 24){
    horaOFF = horaOFF - 24;
  }
  minOFF = minON + TPruebaMin;
  if(minOFF >= 60){
```

```
    minOFF = minOFF - 60;
  }
  if(minON == 0){
    horaOFF--;
  }
}
```

El segundo error se relacionaba con la visualización y almacenamiento del flujo. Como se observa en la Figura 5.54, una vez detenida la prueba el sistema continuaba mostrando el último valor de flujo registrado, tanto en el display como en los datos almacenados. Por este motivo, en la gráfica se observa una línea constante desde el momento en que se detuvo la prueba el 26 de noviembre hasta que se reprogramó otra el mediodía del 27 de noviembre.

Para corregirlo, se agregó una condición en la toma de datos en tiempo real que verifica si la bomba está activa. Si no lo está, el valor del flujo se fuerza a cero:

```
if(EstadoBom == true){
  flow_rate = acumFl_Ins / contFl_Ins;
} else {
  flow_rate = 0;
}
```

Con los errores observados y corregidos, se analizaron los datos de flujo tomados desde el 27 de noviembre al mediodía hasta el 28 de noviembre a la misma hora (24 horas continuas). A partir de esta información se obtuvieron las siguientes métricas de desempeño: un error absoluto medio (MAE) de 4.63 ml/min, un error cuadrático medio (RMSE) de 4.7 ml/min y un 28.9 % de los datos fuera del rango establecido de ± 5 ml/min.

Dado que se trabaja con un flujo objetivo tan bajo, el control resulta particularmente complejo. No obstante, los valores alcanzados se consideraron aceptables, mostrando que la bomba opera dentro de un rango funcional adecuado para los fines del muestreo ambiental propuesto.

Conclusión de la validación: Con estos datos, resultados y errores corregidos o recalibrados, se consideró que los muestreadores estaban listos para ser enviados a la zona sur del Valle del Mezquital. Se entregaron con todo lo necesario para ser operados por los usuarios, recolectar datos y realizar pruebas exitosas con los cartuchos colocados.

6 Resultados

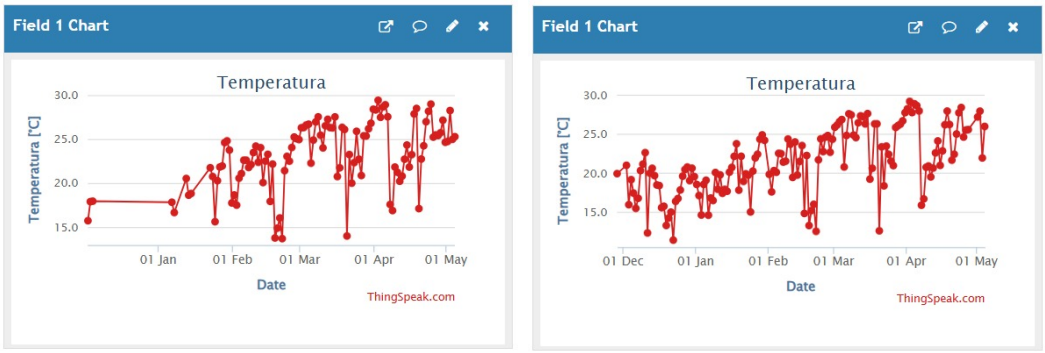
Se realizaron pruebas con el muestreador en la región sur del Valle del Mezquital, llevando todo lo necesario para su instalación: trípode, panel solar, herramientas, baterías, entre otros elementos. Ocho personas permitieron la instalación de los dispositivos en los techos de sus viviendas y brindaron apoyo en la programación de las pruebas y el monitoreo del proceso.

1. Resultados del 5 de diciembre al 13 de enero

La instalación se llevó a cabo entre el 2 y el 5 de diciembre. Durante este período, se montaron los trípodes, se colocaron los MABFs y se conectaron las baterías, el sistema de viento (anemómetro y veleta) y el panel solar. Cada dispositivo fue configurado para conectarse a la red *Wi-Fi* del hogar correspondiente, verificando su correcto funcionamiento en cada caso. Los muestreadores únicamente recopilaron datos meteorológicos y de calidad de aire con los sensores; no se realizó ninguna prueba con los cartuchos.

1.1. Funcionamiento *Wi-Fi*

A lo largo del mes, se monitorearon los datos enviados al servidor. La mayoría de los MABFs se desconectaron de la red en algún momento, lo que evidenció un error que impedía su reconexión automática. Sin embargo, gracias a la memoria de respaldo incorporada, no se produjo pérdida de datos. El MABF 7 permaneció conectado de forma continua durante todo el período, lo cual permitió comprobar la resistencia del sistema ante la intemperie, sin presentar fallas críticas.



((a)) MABF No.1. ((b)) MABF No. 7.

Figura 6.1: Comparación del panel de temperatura diario del MABF No.1 vs. MABF No.7

En la Figura 6.1(a) se observa que no se registraron datos desde principios de diciembre hasta mediados de enero. Por otro lado, en la Figura 6.1(b) se observa que el servidor no perdió ningún dato en el mismo período, indicando que no hubo desconexión.

1.2. Funcionamiento de sensores

Con los datos almacenados en la memoria fue posible observar el funcionamiento de los sensores. Al no contar con redes o dispositivos de monitoreo ambiental que abarquen esa zona, el análisis se basó en una estación Davis colocada a escasos metros del MABF 2 para comparar los datos obtenidos por este.

1.2.1. Temperatura

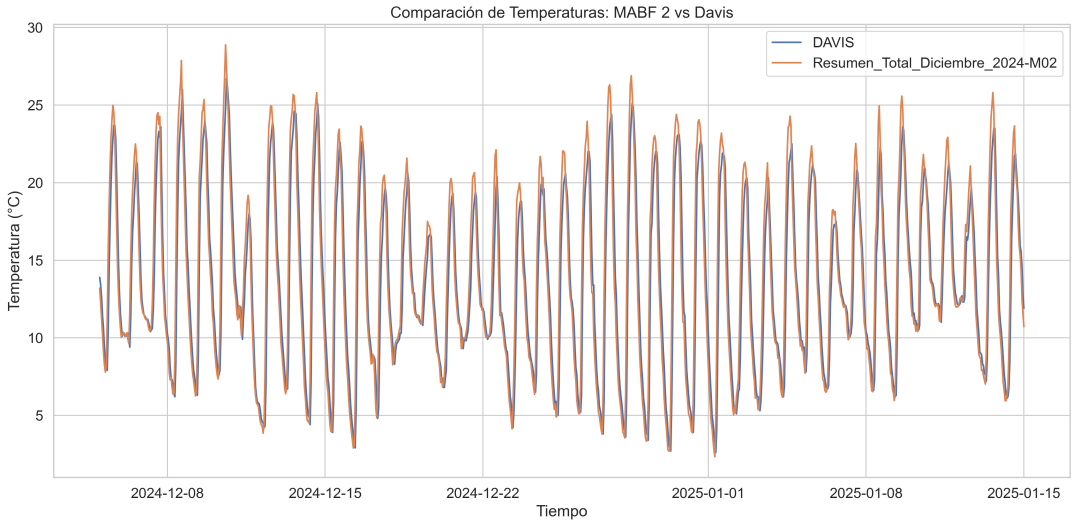


Figura 6.2: Comparación de temperatura MABF 2 vs Davis.

En la Figura 6.2, se aprecia que los valores máximos registrados por el MABF 2 son mayores que los máximos registrados por la estación Davis. Los datos presentan una correlación de Pearson de 0.92, lo cual indica una relación fuerte, con un error absoluto medio (MAE) de 1.83 °C y un error cuadrático medio (RMSE) de 2.34 °C. Aunque la correlación de Pearson es elevada, los errores superan los 1.5 °C debido a los picos máximos que alcanza el MABF 2. Este comportamiento en la temperatura se asocia con el efecto parrilla generado por la superficie sobre la que se encuentra el sensor SHT30.

1.2.2. Humedad relativa

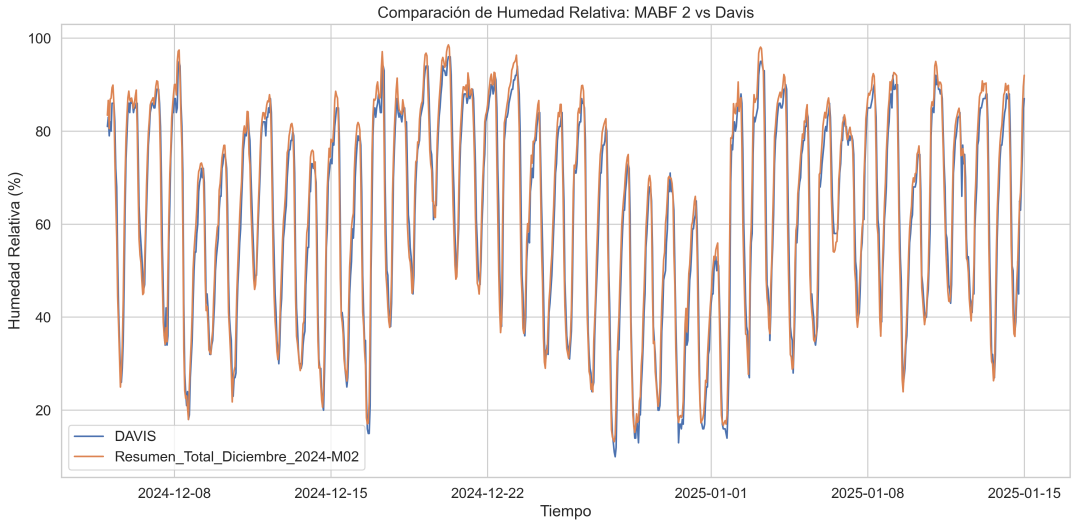


Figura 6.3: Comparación de humedad relativa: MABF 2 vs. Davis.

Al observar la Figura 6.3, se aprecia que ambos sensores muestran un comportamiento muy similar. Con un error absoluto medio (MAE) de 4.29 % y un error cuadrático medio (RMSE) de 5.23 %, además de una correlación de Pearson de 0.97 (muy fuerte), es posible afirmar que el sensor SHT30 presentó un desempeño adecuado en la medición de humedad relativa.

1.2.3. Velocidad del viento

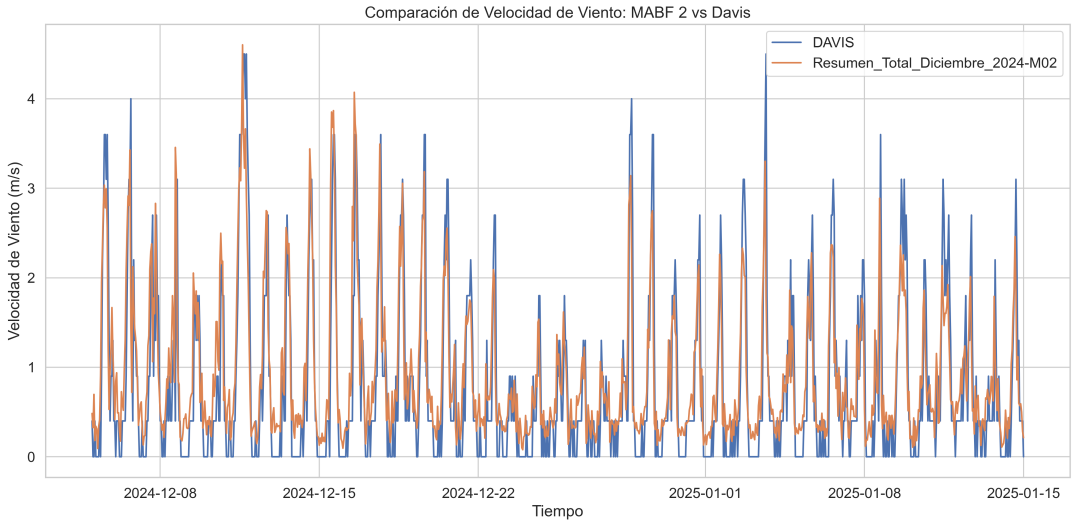


Figura 6.4: Comparación de velocidad del viento: MABF 2 vs. Davis.

Como la recalibración del anemómetro se realizó posteriormente a la instalación y al primer mes de pruebas, este resultado no refleja el comportamiento posterior a dicha calibración. Sin embargo, se observa un comportamiento similar, con un error absoluto medio (MAE) de 0.41 [m/s], un error cuadrático medio (RMSE) de 0.56 [m/s] y una correlación de Pearson de 0.82, lo que indica una relación fuerte y un funcionamiento adecuado del anemómetro.

1.2.4. Dirección del viento

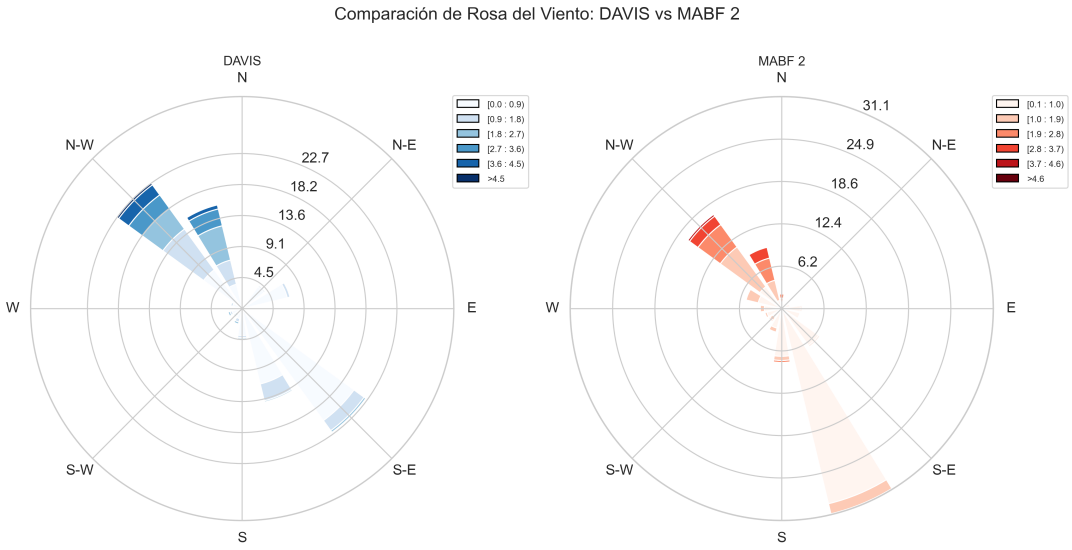


Figura 6.5: Comparación de dirección del viento: MABF 2 vs. Davis.

En la Figura 6.5 se observa que ambas veletas presentan un comportamiento similar, mostrando una alta proporción de vientos ligeros provenientes del Sureste y vientos más fuertes provenientes del Noroeste. La magnitud de los vientos registrados por el MABF 2 es ligeramente menor en comparación con las mediciones de la estación Davis, lo cual se atribuye a que, en ese momento, aún no se había realizado la recalibración del anemómetro en el túnel de viento. Con base en estas variables meteorológicas, se concluye que el sistema proporciona datos confiables para el monitoreo ambiental.

1.2.5. Índice de calidad del aire y compuestos orgánicos volátiles (COVs)

Para estos dos sensores no se cuenta con un parámetro de referencia en la zona que permita realizar una comparación directa. Sin embargo, es posible analizar el comportamiento de los sensores y verificar si los datos obtenidos presentan una lógica coherente.

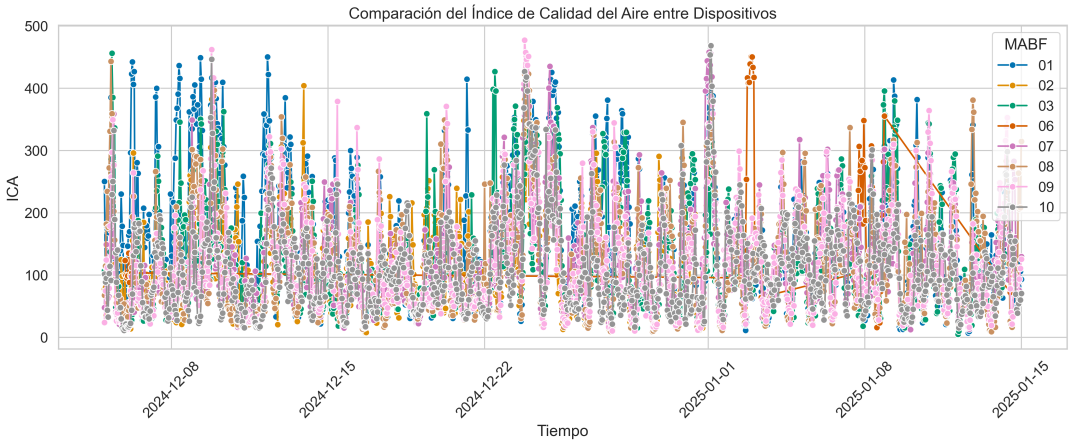


Figura 6.6: Comparación del índice de calidad del aire entre dispositivos.

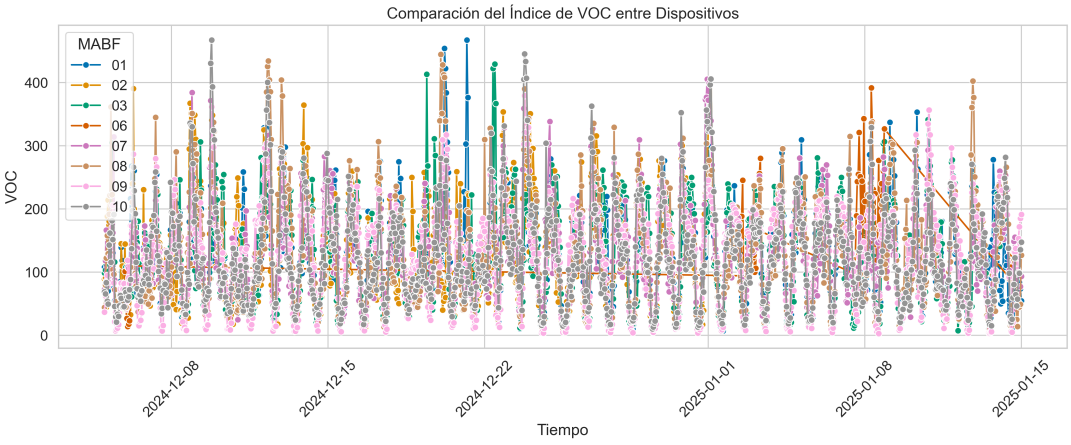


Figura 6.7: Comparación de COVs entre dispositivos.

Como se aprecia en ambas figuras, los valores mínimos muestran un comportamiento similar, alcanzando sus puntos más bajos durante las horas de la madrugada. En cambio, los valores máximos varían entre los MABFs, aunque pueden encontrarse algunas similitudes entre ciertos dispositivos. Esta variabilidad se atribuye a que los sensores están ubicados en distintas zonas, con diferentes condiciones de entrada de viento y factores ambientales.

Además, al descargar los datos almacenados en la memoria, se observó que el MABF 6 presentó una falla en alguno de sus componentes, lo que ocasionó la pérdida de datos en ciertos rangos de tiempo. Como se aprecia en ambas figuras, los registros se reanudaron únicamente después de que el usuario apagara y encendiera nuevamente la caja del dispositivo. Esta acción fue solicitada a los usuarios al menos en dos ocasiones durante el periodo de muestreo.

1.3. Solución de errores

La solución al problema de reconexión fue implementar una función similar a `Ctr15Min()`, llamada `Ctr5Min()`, la cual realiza las mismas tareas de verificación en intervalos de 5 minutos, definidos en los minutos 1, 6, 11, 16, 21, 26... 51 y 56. Estos intervalos se definieron de forma desfasada para evitar interferencias con la función `Ctr15Min()`.

```
void Ctr5Min(){
    DateTime fecha = rtc.now();
    if(fecha.minute()==1 || fecha.minute() == 6 || ... || fecha.
   minute() == 56){
    ...
    }
}
```

Dentro de la función `Ctr5Min()`, se reinicia la conexión con el servidor de ThingSpeak actuando como cliente, y se verifica el estado de la conexión a Internet. En caso de que no haya conexión, se ejecuta la función `WiFi.reconnect()` para intentar restablecerla. Estas instrucciones están contenidas dentro de una condición controlada por una variable booleana llamada `wifi`; de este modo, si se desea que una unidad MABF funcione sin conectividad a Internet, basta con establecer dicha variable en `false`.

```
        if(Wifi == true){
            ThingSpeak.begin(cliente);
            if(WiFi.status() != WL_CONNECTED){
                WiFi.reconnect();
                while(WiFi.status() != WL_CONNECTED){
                    if(j > 100){
                        break;
                    }
                    j++;
                }
            }
        }
```

Finalmente, se utiliza una variable booleana para evitar múltiples ejecuciones; esta se reinicia un minuto después del evento, es decir, en los minutos 2, 7, 12... 52 y 57.

```
    if(fecha.minute()==2 || fecha.minute() == 7 || ... || fecha.
   minute() == 57){
        Ctr5MinBool=true;
    }
```

Adicionalmente, antes de la condición `if(wifi == true)`, se implementó una verificación del error del controlador PID. Si dicho error supera el 20 % en valor absoluto, se reinician las constantes originales del controlador (`Kp`, `Ki` y `Kd`) con el objetivo de restablecer el equilibrio del sistema. Esta medida busca prevenir desviaciones prolongadas y mantener una respuesta estable y confiable en el control del flujo de aire.

```
        if(ContrOFFBool == true && (flow_rate > (Setpoint * 1.2) ||
   flow_rate < (Setpoint * 0.8))){
```

```
2             PID_error=0.0;PID_p=0.0; PID_i=0.0; PID_d=0.0;
    previous_error=0;PID_value=374;Kp=0.08; Ki=0.05; Kd=0.010;
3         }
```

Para facilitar la visualización del estado de conexión a internet, se incorporó un ícono de WiFi en la pantalla principal. Este ícono aparece cuando el dispositivo está conectado a la red, y desaparece cuando no lo está. La lógica para controlar su visualización se implementó dentro de la función `Datos5Seg()`, mediante un booleano que refleja el estado de la conexión. En la interfaz gráfica, este valor se consulta para decidir si se debe mostrar o no el ícono. (Ver sección de Anexo – Código 1.3.1 - Obtención de datos para 5 segundos y Código 8.2.3 - Funciones para el despliegue de datos).

1.4. Actualización del software a los MABFs y corrección de fallas

El 13 de enero se realizó una segunda visita para evaluar el estado de los dispositivos. Durante esta inspección, se actualizó el microcontrolador para corregir fallos como la pérdida de conexión a internet, la retención del último valor de flujo en los promedios de 15 minutos y la ausencia de reinicio de los valores PID al iniciar cada prueba y válvula. Estos dos últimos errores fueron identificados previamente en el MABF 5, que se encontraba en el laboratorio del instituto. Además, se descargaron los datos recopilados durante el periodo de prueba. En general, los registros fueron correctos, salvo en el caso del MABF 6 ya mencionado, que presentó fallas intermitentes en el módulo SD. Este problema fue rápidamente identificado gracias a los mensajes de error mostrados en la aplicación móvil, lo que permitió reemplazar la placa y restablecer su funcionamiento.

2. Resultados del 14 de enero al 30 de abril

Con la reconexión al *WiFi* corregida y con todos los MABFs funcionando de forma adecuada, comenzó el periodo de muestreo con los cartuchos colocados. Para ello, a cada usuario se le proporcionaron seis pares de cartuchos para realizar seis pruebas de 24 horas a lo largo de un periodo de seis semanas.

Además, se impartió un curso breve para que los usuarios pudieran familiarizarse con el funcionamiento del MABF, el uso de la aplicación móvil y el procedimiento correcto para la colocación de los cartuchos. Esta capacitación fue fundamental para el éxito de las pruebas, ya que una correcta colocación de los cartuchos y una programación adecuada reducían la variabilidad en los resultados obtenidos.

2.1. Funcionamiento del flujo

2.1.1. Muestreo de enero

En la primera prueba programada se identificó un problema de programación, ya que todos los muestreadores comenzaron la prueba pero se apagaron exactamente una hora después. Este error correspondía al mismo problema comentado en la Sección 11.4, que no había sido identificado hasta ese momento, cuando todos los MABF se detuvieron tras una hora de operación. En dicha sección se explica detalladamente cómo se identificó y se corrigió el problema en el código.

Dado que era necesario comenzar con las pruebas lo antes posible para no alterar por completo el itinerario programado, se optó por advertir a los usuarios que, hasta que se pudiera regresar a reprogramar y corregir este error, programaran las pruebas a las 6:01 a.m. en lugar de las 6:00 a.m., como estaba previsto inicialmente.

Acordando esto, fue posible realizar la primera prueba con cartuchos colocados:

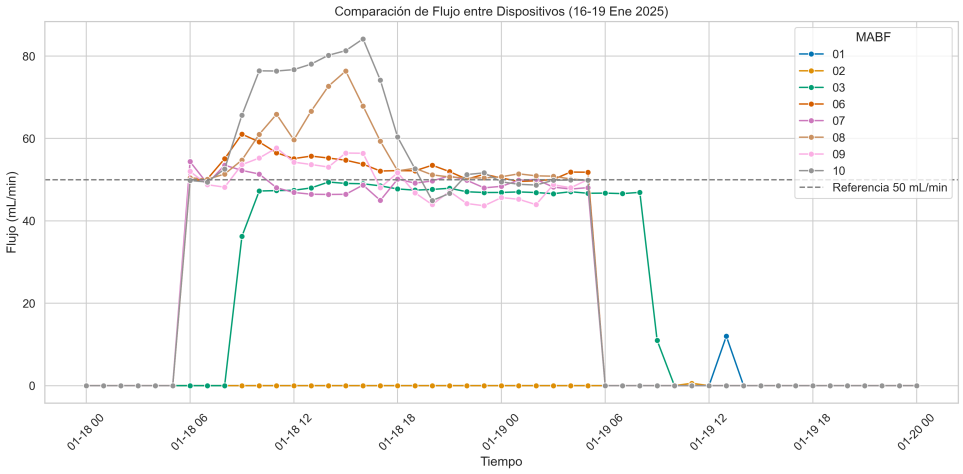


Figura 6.8: Comparación de flujo entre dispositivos durante la primera prueba).

En la Figura 6.8 se observa que tanto el MABF 1 como el MABF 2 no se encendieron, mientras que el MABF 3 inició su prueba con retraso. El MABF 1 y el MABF 2 fueron programados por el mismo usuario, por lo que se consideró que su teléfono podría no ser completamente compatible con la aplicación. En algunas pruebas realizadas junto al usuario, los dispositivos sí se encendían correctamente al programar una prueba, lo que sugiere que, en ciertas ocasiones, los datos no se transmitían adecuadamente entre el teléfono móvil y el muestreador. Por otro lado, el usuario encargado del MABF 3 indicó que él mismo programó la prueba de forma tardía.

En cuanto a la exactitud del control de flujo, se observó que muestreadores como el 10 y el 8 presentaron un comportamiento alejado a la referencia en las primeras 12 horas (Primer cartucho). En las siguientes 12 horas (Segundo cartucho) todos los

muestreadores tuvieron una mejoría en su aproximación al flujo objetivo, como se muestra en la tabla 6.1.

MABF	**Día (18 ene 6AM–6PM)**		**Noche (18–19 ene 6PM–6AM)**	
	MAE (mL/min)	**RMSE (mL/min)**	**MAE (mL/min)**	**RMSE (mL/min)**
01	50.00	50.00	50.00	50.00
02	50.00	50.00	50.00	50.00
03	13.84	24.38	2.84	2.87
06	4.70	5.59	1.34	1.69
07	2.67	3.01	1.02	1.31
08	10.64	13.56	0.98	1.24
09	3.78	4.29	3.69	4.23
10	19.76	23.04	2.30	3.63

Tabla 6.1: MAE y RMSE por dispositivo para el 18 y 19 de enero.

Esta diferencia en el comportamiento entre los cartuchos diurnos y nocturnos evidencia cómo factores como una posible fuga en el sistema o diferencias en la cantidad de compuesto en los cartuchos afectan el desempeño del muestreador para alcanzar el flujo objetivo.

Las siguientes dos muestras se realizaron de forma continua con una semana de diferencia, y los resultados se presentan a continuación:

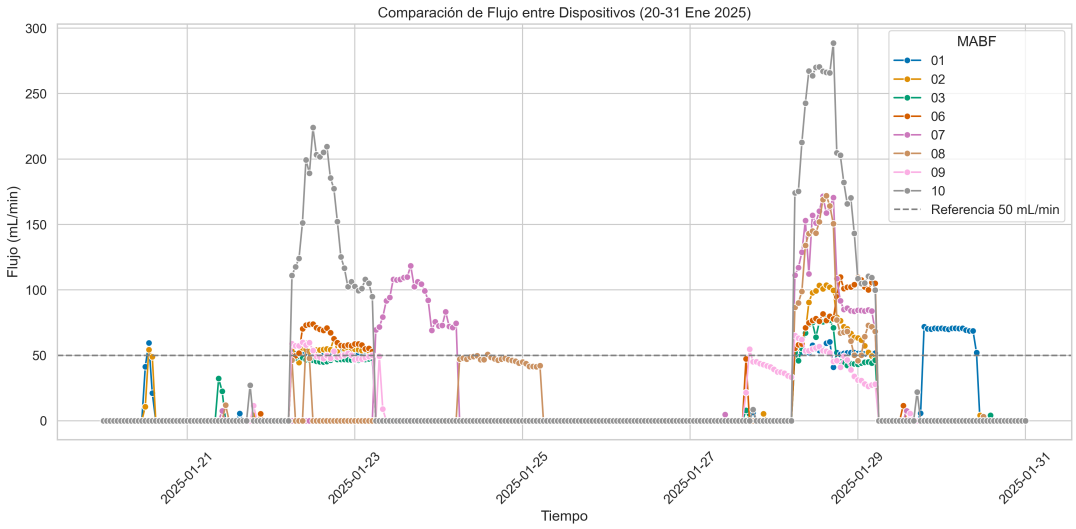


Figura 6.9: Comparación de flujo entre dispositivos (Segunda - Tercera Prueba).

Se observó un incremento súbito del flujo en algunas de las cajas. Además, los usuarios de los MABFs 07 y 08 aún se estaban familiarizando con la aplicación y presentaron inconvenientes al programar a las 6:01 a.m., como se había indicado. Por esta razón, sus muestreos se retrasaron entre uno y dos días.

MABF	**Día (22 ene 6AM–6PM)**		**Noche (22–23 ene 6PM–6AM)**	
	MAE (mL/min)	**RMSE (mL/min)**	**MAE (mL/min)**	**RMSE (mL/min)**
01	1.56	1.81	2.53	2.58
02	4.98	5.71	4.64	5.34
03	3.08	3.40	3.04	3.08
06	15.69	17.91	7.58	7.94
07	50.00	50.00	50.00	50.00
08	39.64	43.95	50.00	50.00
09	4.98	6.05	1.54	1.92
10	126.84	132.03	65.93	70.06

Tabla 6.2: MAE y RMSE por dispositivo para el 22 y 23 de enero.

Se observa en la Tabla 6.2 que los MABFs 1, 2, 3 y 9 presentaron un desempeño adecuado, con un error cuadrático medio máximo de 6.05 entre ellos. Además, se aprecia que los MABFs 6 y 10, al igual que en la primera prueba, muestran un mejor comportamiento en su segundo cartucho. En la Figura 6.9 se observa que el desempeño del MABF 8 fue correcto, a pesar de que su muestreo tuvo que ser postergado dos

días. El MABF 7 presentó un comportamiento errático, similar al observado en los MABFs 6 y 10.

MABF	**Día (28 ene 6AM–6PM)**		**Noche (28–29 ene 6PM–6AM)**	
	MAE (mL/min)	**RMSE (mL/min)**	**MAE (mL/min)**	**RMSE (mL/min)**
01	4.96	5.76	1.65	1.70
02	35.58	40.34	13.78	16.53
03	17.27	19.79	5.66	5.76
06	23.87	26.03	53.67	53.83
07	93.36	96.16	35.71	36.73
08	82.78	88.64	14.42	16.74
09	6.31	7.52	14.41	16.46
10	193.78	197.31	92.31	100.23

Tabla 6.3: MAE y RMSE por dispositivo para el 28 y 29 de enero.

En el tercer muestreo, como se observa en la Figura 6.9, varios muestreadores presentaron dificultades para alcanzar el flujo objetivo. En particular, los MABFs 2, 3, 7, 8 y 10 mostraron un mejor desempeño durante el muestreo nocturno, mientras que los MABFs 6 y 9 tuvieron un mejor comportamiento en el periodo diurno. Estos resultados evidencian que las fallas en el desempeño están relacionadas principalmente con la composición de los cartuchos y su correcta instalación, sin fugas. Al tratarse de un flujo muy bajo, los cartuchos deben ofrecer la obstrucción necesaria para que el compresor alcance el valor objetivo. Si existe una mínima fuga, se reduce la resistencia al paso del aire, lo que también impide que el sistema logre alcanzar el flujo deseado.

MABF	18-ene	22-ene	23-ene	24-ene	27-ene	28-ene	29-ene
01	–	1.42	–	–	–	3.56	4.62
02	–	3.25	–	–	–	19.67	–
03	7.59	1.12	–	–	–	14.15	–
06	3.05	8.00	–	–	–	17.58	–
07	2.28	–	16.62	–	–	35.19	–
08	8.07	–	–	2.63	–	43.44	–
09	4.43	4.40	–	–	10.67	–	–
10	13.78	44.93	–	–	–	66.07	–

Tabla 6.4: Desviación estándar mL/min por dispositivo MABF y fecha exacta de prueba - Enero 2025.

Como se muestra en la tabla 6.4, se observa una tendencia general hacia el incremento de la variabilidad en las pruebas realizadas hacia finales de mes, particularmente en las pruebas del 28 de enero. Los MABFs 01, 03, 06, 07, 08 y 09 demostraron una alta precisión inicial con desviaciones estándar inferiores a 10 mL/min en sus primeras pruebas.

Se destaca el caso del MABF 01, que mantuvo una precisión consistente a lo largo de todo el mes (1.42 - 4.62 mL/min), siendo el único dispositivo que realizó un muestreo extra en una visita que se realizó al final del mes. Por el contrario, los MABFs 07, 08 y 10 mostraron incrementos significativos en su variabilidad, llegando a desviaciones estándar de 35.19, 43.44 y 66.07 mL/min respectivamente en sus últimas pruebas.

Los días 29 y 30 de enero se realizó una visita rápida a todos los usuarios para reprogramar los dispositivos y corregir el error que impedía establecer una prueba a una hora en punto. Además, se acordó aumentar el flujo objetivo de los muestreos a 70 mL/min. Esta modificación permite obtener una mayor masa de aire muestreada y una mejor estabilidad del sistema. En la casa que aloja al MABF 1, donde se ofreció hospedaje durante la visita, se decidió realizar una prueba programada a una hora exacta y con un flujo de 70 mL/min para evaluar su desempeño. Como se muestra en el extremo derecho de la Figura 6.9, el sistema presentó una muy buena precisión y exactitud respecto al nuevo flujo objetivo.

2.1.2. Muestreo de febrero

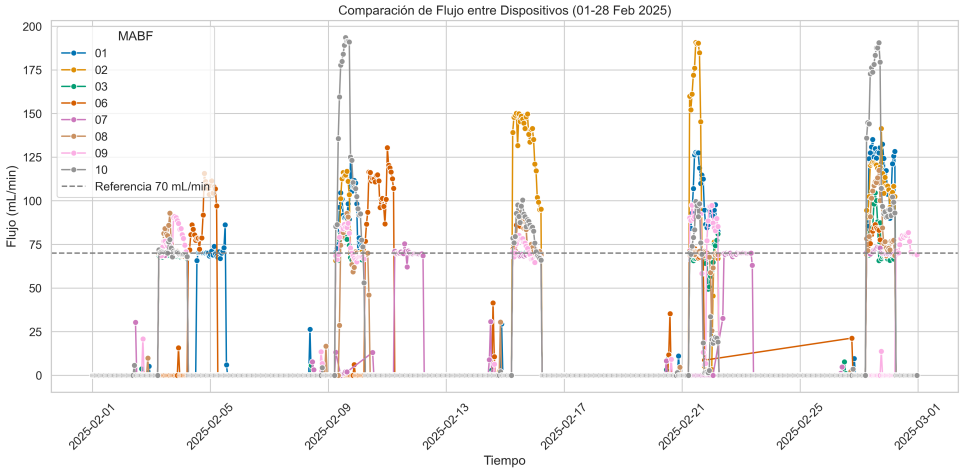


Figura 6.10: Comparación de flujo entre dispositivos durante febrero de 2025.

En la Figura 6.10 se observa que, en algunas pruebas, ciertos muestreadores alcanzaron flujos superiores al 150 % del valor objetivo. Para cuantificar estas desviaciones, se calcularon los errores absolutos medios (MAE) y los errores cuadráticos medios (RMSE) para cada dispositivo (Tabla 6.5). Los resultados obtenidos permiten identificar el desempeño general de cada muestreador durante el mes de febrero.

MABF	MAE Promedio (mL/min)	RMSE Promedio (mL/min)
01	20.44	22.65
02	33.67	36.39
03	4.81	5.90
06	14.07	17.12
07	0.84	1.46
08	8.30	12.73
09	12.28	14.25
10	31.76	38.30

Tabla 6.5: Promedio de MAE y RMSE por dispositivo durante febrero de 2025

Muestreadores como el 01, 02, 06 y 10 presentaron errores promedio superiores al 20 % del valor objetivo. En particular, los MABFs 2 y 10 registraron errores mayores a 30 mL/min en las pruebas realizadas durante febrero. Por otro lado, el muestreador 07, que tuvo el mejor desempeño, no superó los 2 mL/min de error respecto al flujo objetivo, lo que evidencia una excelente exactitud en el control del bajo flujo en todas

sus pruebas.

MABF	**03-feb**	**04-feb**	**09-feb**	**10-feb**	**11-feb**	**15-feb**	**21-feb**	**22-feb**	**27-feb**
01	–	3.78	15.80	–	–	0.55	15.56	–	16.43
02	0.66	–	21.12	–	–	18.57	42.31	–	11.32
03	0.54	–	6.19	–	–	0.43	7.15	–	12.04
06	–	15.33	–	12.60	–	8.37	1.58	–	6.22
07	1.38	–	–	–	2.06	0.59	–	1.44	1.47
08	6.94	–	10.26	–	–	8.82	9.33	–	16.71
09	7.34	–	8.29	–	–	4.87	24.83	–	6.74
10	2.30	–	46.36	–	–	10.21	21.73	–	40.87

Tabla 6.6: Desviación estándar (mL/min) por dispositivo MABF y fecha exacta de prueba - Febrero 2025.

Como se muestra en la tabla 6.6, se observa un comportamiento complejo en la precisión del control de flujo durante febrero. Los MABFs 01, 02, 03 y 10 presentaron fluctuaciones significativas entre pruebas, sin un patrón claro.

Destaca el MABF 07 como el más estable del mes, manteniendo desviaciones estándar consistentemente bajas (0.59-2.06 mL/min). Por otra parte, los MABFs 02 y 10 mostraron la mayor variabilidad, con valores atípicos de 42.31 mL/min y 46.36 mL/min respectivamente en las pruebas del 09-feb y 27-feb.

Se identifica que las pruebas del 09-feb y 27-feb presentaron la menor precisión para la mayoría de los muestreadores, mientras que el 15-feb registró los mejores resultados. El MABF 06 mostró una mejora progresiva en su precisión a lo largo del mes (de 15.33 mL/min a 1.58 mL/min), sugiriendo una posible mejora en la colocación y prueba de fugas de los trenes.

2.1.3. Muestreo de marzo

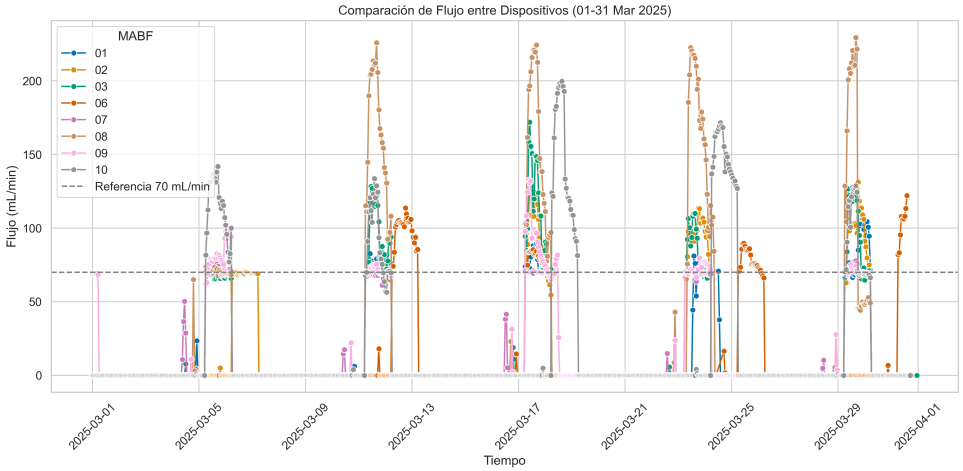


Figura 6.11: Comparación de flujo entre dispositivos durante marzo de 2025.

En la Figura 6.11 se observa que dos cajas, MABF 08 y MABF 10, se mantuvieron en rangos de flujo muy por encima del objetivo, alcanzando valores superiores a los 150 mL/min. El MABF 03 mostró un deterioro progresivo en su desempeño conforme avanzó el mes, a pesar de haber sido uno de los muestreadores con mejor rendimiento previamente. Por el contrario, el MABF 01 presentó una mejora notable en la estabilización del flujo, acercándose con mayor exactitud al valor objetivo.

MABF	MAE Promedio (mL/min)	RMSE Promedio (mL/min)
01	6.15	8.96
02	15.55	18.29
03	20.99	27.96
06	11.42	13.17
07	1.45	2.22
08	68.88	79.25
09	8.08	10.51
10	48.98	55.42

Tabla 6.7: Promedio de MAE y RMSE por dispositivo durante marzo de 2025

Al igual que en el mes anterior, el MABF 07 fue el muestreador que mostró el mejor comportamiento. Los dispositivos 01, 02, 06 y 09 mejoraron su desempeño, reduciendo sus errores y manteniéndose cerca del 20 % de error respecto al flujo objetivo. En contraste, los MABF 03, 08 y 10 presentaron un incremento en su error, reflejando un

peor rendimiento. En particular, los muestreadores 08 y 10 aumentaron su error en más de 30 mL/min, lo que evidencia una desviación significativa respecto al objetivo.

MABF	**05-mar**	**06-mar**	**11-mar**	**12-mar**	**17-mar**	**18-mar**	**23-mar**	**24-mar**	**25-mar**	**29-mar**
01	0.39	–	4.19	–	6.36	–	7.97	–	–	15.47
02	–	0.45	1.45	–	24.58	–	13.67	–	–	17.30
03	2.98	–	20.63	–	33.50	–	16.37	–	–	24.89
06	3.46	–	–	10.71	6.27	–	–	–	7.39	–
07	2.52	–	2.23	–	0.58	–	1.39	–	–	2.85
08	1.56	–	43.60	–	52.68	–	48.46	–	–	78.84
09	10.43	–	2.65	–	22.95	–	3.30	–	–	2.22
10	20.25	–	25.28	–	–	41.64	–	15.11	–	26.08

Tabla 6.8: Desviación estándar (mL/min) por dispositivo MABF y fecha exacta de prueba - Marzo 2025.

Como se muestra en la tabla 6.8, marzo presenta el patrón de variabilidad más extremo observado hasta el momento. El MABF 08 muestra una variablilidad muy alta durante todo el mes, con valores que alcanzan los 78.84 mL/min el 29-mar, siendo consistentemente el dispositivo con menor precisión.

Al igual que con la exactitud el MABF 07, mantuvo su excepcional estabilidad (0.58-2.85 mL/min), consolidándose como el dispositivo más preciso y exacto de la campaña. El MABF 01 muestra un incremento progresivo en su variabilidad a medida que avanza el mes, iniciando con 0.39 mL/min y terminando con 15.47 mL/min.

Se identifica que las pruebas del 17-mar fueron particularmente problemáticas para la mayoría de los dispositivos, con 6 de los 8 MABFs mostrando sus valores más altos o cercanos al máximo mensual. Contrariamente, el 23-mar muestra una mejora generalizada en la precisión para varios dispositivos.

El MABF 10, al igual que con el MAE y RMSE, continúa mostrando alta variabilidad (15.11-41.64 mL/min), aunque con una ligera mejora comparado con meses anteriores. Los resultados indican que los MABFs 03 y 08 fueron los que mostraron el mayor deterioro respecto al mes anterior.

2.1.4. Muestreo de abril

Los días 31 de marzo y 1° de abril se realizó una visita rápida a la zona para entregar a los usuarios nuevas mangueras de acoplamiento para los trenes, ya que las utilizadas previamente se encontraban desgastadas y presentaban múltiples fugas.

Las nuevas mangueras tenían un diámetro ligeramente menor, lo que dificultaba su instalación correcta; sin embargo, ofrecían un mejor desempeño en las pruebas de fugas, mejorando así la hermeticidad del sistema.

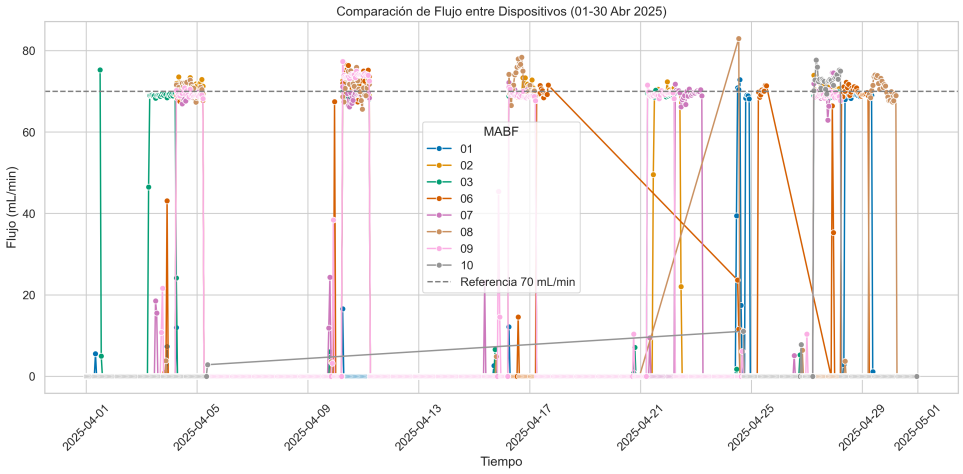


Figura 6.12: Comparación de flujo entre dispositivos durante abril de 2025.

En la Figura 6.12 se observa que los MABFs 01, 06, 08 y 10 presentaron pérdidas de datos durante el mes de abril. El usuario encargado del MABF 01 reportó que al momento de programar una prueba con su celular el dispositivo no realizaba ninguna función, por esto decidió no programar pruebas en este mes hasta que se realizara una visita para resolver errores. El usuario encargado del MABF 10 reportó que la aplicación mostraba el mensaje de error "Error en la memoria Micro SD" al inicio del mes, por lo que decidió mantener el muestreador apagado hasta que se realizó una visita técnica los días 24 y 25 de abril para corregir fallas. En el caso del MABF 08, al momento de descargar los datos, se identificaron dos archivos corruptos correspondientes a días distintos, los cuales debieron ser eliminados. Por otro lado, el usuario del MABF 06 no reportó ningún inconveniente, y se desconoce la causa por la cual no se cuenta con archivos de promedios entre el 17 y el 24 de abril.

MABF	MAE Promedio (mL/min)	RMSE Promedio (mL/min)
01	5.50	9.58
02	1.52	3.60
03	1.24	2.62
06	2.94	5.93
07	0.83	1.16
08	2.03	2.59
09	1.54	1.64
10	2.94	3.40

Tabla 6.9: Promedio de MAE y RMSE por dispositivo durante abril de 2025

Es notable la mejoría en el control de flujo observada durante este mes, tras la implementación de las nuevas mangueras. El MABF 1 fue el dispositivo con el mayor error, presentando un MAE inferior al 10 % y un RMSE cercano a 10 mL/min, lo que sugiere un buen desempeño general, aunque con algunas desviaciones puntuales respecto al flujo objetivo. Todos los demás muestreadores mantuvieron errores menores al 10 %, destacando particularmente el MABF 7, cuyo MAE y RMSE fueron cercanos a 1 mL/min, demostrando un control de flujo altamente exacto.

MABF	03-abr	04-abr	10-abr	16-abr	17-abr	21-abr	22-abr	25-abr	27-abr	28-abr	29-abr
01	–	–	–	–	–	–	–	–	–	0.32	–
02	–	0.97	1.61	1.41	–	9.76	–	–	1.04	–	–
03	9.17	–	0.37	0.33	–	0.37	–	–	0.42	–	–
06	–	0.63	2.93	–	19.80	–	–	0.99	–	0.97	–
07	–	0.90	1.16	0.65	–	–	1.19	–	–	–	–
08	–	1.41	2.59	3.27	–	–	–	–	–	–	2.15
09	–	0.63	0.86	0.69	–	0.66	–	–	0.56	–	–
10	–	–	–	–	–	–	–	–	1.85	–	–

Tabla 6.10: Desviación estándar (mL/min) por dispositivo MABF y fecha exacta de prueba - Abril 2025.

Como se muestra en la tabla 6.10 y al igual que en la exactitud, abril presenta una mejora significativa en la estabilidad general comparado con meses anteriores. La mayoría de los dispositivos muestran desviaciones estándar inferiores a 3 mL/min, indicando un control de flujo más preciso y consistente.

Se pueden observar de manera puntual tres picos en la desviación estándar de los MABFs 02, 03 y 06, siendo el pico más grande el presentado por el MABF 06 el 17-abr con 19.8 mL/min. Por otro lado, los MABFs 07, 08 y 09 mantuvieron un desempeño estable y dentro de rangos aceptables durante todo el periodo de medición.

Como se mencionó al inicio del apartado, los MABFs 01 y 10 no pudieron realizar pruebas hasta el final del mes. Sin embargo, en la única prueba que se realizó presentan una mejoría notable respecto a otros meses con una desviación estándar menor 1.85 mL/min.

2.1.5. Muestreo de mayo

En el mes de mayo se realizaron las últimas tres pruebas del programa, por lo que se decidió acotar la gráfica 6.13 hasta el 17 de mayo.

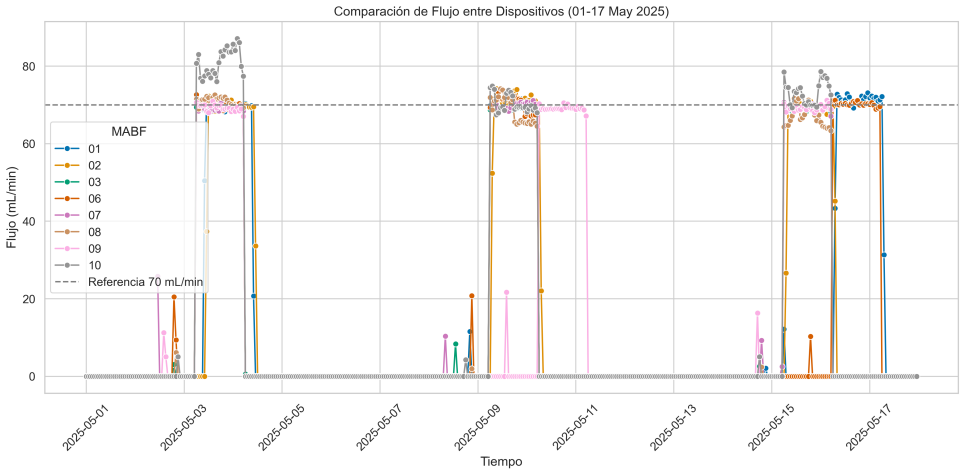


Figura 6.13: Comparación de flujo entre dispositivos durante mayo de 2025.

En la Figura 6.13 se observa que, salvo en la primera prueba del mes, el MABF 10 mantuvo un control de flujo estable. En dicha prueba inicial se presentó una ligera inestabilidad, aunque el flujo se mantuvo por debajo de los 90 mL/min, lo que indica un desempeño aceptable al permanecer, en su mayoría, dentro del margen de error del 20 % respecto al valor objetivo. Para el resto de los dispositivos se observa un comportamiento estable y muy cercano al flujo programado.

MABF	MAE Promedio (mL/min)	RMSE Promedio (mL/min)
01	2.42	6.57
02	2.52	7.65
03	0.90	0.96
06	0.99	1.19
07	0.80	0.95
08	2.70	3.04
09	0.92	1.10
10	5.42	6.10

Tabla 6.11: Promedio de MAE y RMSE por dispositivo durante mayo de 2025

Por último, en la Tabla 6.11 se observa un comportamiento ideal en todos los muestreadores durante las últimas tres pruebas. El MABF 10 presentó el mayor error, con un valor aproximado de 6 mL/min, lo cual representa menos del 10 % del flujo objetivo. En este mes, destacan los MABFs 03, 06, 07 y 09, que lograron errores cercanos a 1 mL/min, lo que demuestra una exactitud excelente.

MABF	03-may	09-may	10-may	15-may	16-may
01	10.38	1.01	–	–	8.27
02	7.45	10.46	–	4.98	–
03	0.24	0.25	–	0.58	–
06	1.03	1.88	–	–	0.56
07	0.65	0.88	–	0.69	–
08	1.25	3.45	–	2.68	–
09	0.89	–	0.64	0.81	–
10	3.53	2.33	–	3.05	–

Tabla 6.12: Desviación estándar (mL/min) por dispositivo MABF y fecha exacta de prueba - Mayo 2025.

Como se muestra en la tabla 6.12, mayo presenta algunos muestreadores con una variabilidad considerable, mientras que algunos dispositivos muestran una estabilidad excelente. Los MABFs 03, 07 y 09 mantuvieron un desempeño sobresaliente con desviaciones estándar por debajo de 1 mL/min en todas sus pruebas.

Los MABFs 01 y 02 mostraron el comportamiento más errático, con valores que oscilan entre 1.01-10.38 mL/min y 4.98-10.46 mL/min respectivamente, indicando

cierta fluctuación en la precisión con la que controlan el flujo de aire. El MABF 10 mantuvo la notable mejoría que se observó en la única prueba del mes pasado, manteniéndose en un rango de 2.33-3.53 mL/min.

El día 03-may la mayoría de los muestreadores presentaron una mayor variabilidad, con los MABFs 01 y 02 registrando los valores más altos del día (10.38 y 7.45 mL/min respectivamente). Contrariamente, el 15-may la mayoría de MABFs mostraron una mejor precisión respecto a los demás días.

Es notable el aumento tan drástico que sufrieron la exactitud y la precisión en el control de flujo en los últimos dos meses. Esto coincide con la visita que se realizó a finales de marzo para cambiar las mangueras con las que se unían los trenes.

3. Cuestionario para voluntarios - Evaluación del sistema de monitoreo de COVs

Para conocer la experiencia de los usuarios durante la instalación y operación del sistema de muestreo de COVs, se aplicó un cuestionario con preguntas tanto cuantitativas como cualitativas. Esta evaluación permitió identificar fortalezas y áreas de mejora del sistema desde el punto de vista del usuario.

Esta encuesta fue contestada por 7 de los 8 usuarios involucrados. A continuación, se presentan los resultados más relevantes relacionados con la facilidad de uso, confiabilidad y sugerencias de mejora del sistema. Las respuestas completas se incluyen en el Anexo E.

- **Encendido y operación:** El 85.7 % de los participantes indicaron que el sistema encendía correctamente y podía programarse sin complicaciones. Se destacó que el sistema respondía rápidamente a las acciones realizadas en la pantalla o desde la aplicación móvil.
- **Facilidad de uso:** En una escala del 1 al 5 (donde 5 es muy sencillo), la interfaz del sistema obtuvo en promedio una calificación de 4.42 en la navegación por menús y en cuanto a la claridad en las instrucciones iniciales una calificación de 4. Algunos usuarios señalaron que, aunque la primera vez requirieron asistencia, posteriormente el uso fue intuitivo ya que el 85.7 % se sintió cómodo usando el MABF.
- **Confiabilidad:** El 71.4 % coincidieron en que el sistema funcionó de manera continua durante semanas, sin apagarse ni presentar fallos, incluso en condiciones climáticas adversas como lluvias intensas, fuertes vientos y granizadas.

Figura 6.14: MABF operando con normalidad después de una granizada

Al analizar los datos y calcular el promedio de la relación entre la cantidad de registros faltantes (debido a desperfectos o a que el muestreador estuvo apagado) y el tiempo total en que los equipos permanecieron instalados en la zona, se obtuvo un notable 93.49 % de tiempo de operación. Esto demuestra una alta confiabilidad del sistema, incluso bajo condiciones climáticas adversas.

- **Sugerencias de mejora:** Entre las recomendaciones recurrentes destacan:
  - Mejorar el sistema de sellado de los cartuchos, ya que retirar y colocar los tapones de los trenes para cada muestreo se complicaba en ocasiones.
  - Mejorar el ajuste de los cartuchos al muestreador, ya que dependiendo de la manguera existían o desaparecían las fugas.
  - Modificar la visualización de "prueba" para ver la variación del flujo en la misma página.
- **Percepción general:** Los usuarios expresaron sentirse cómodos utilizando el sistema y les pareció bastante sencilla la interfaz, en específico al usar la aplicación, y manifestaron su disposición a participar nuevamente en futuras campañas de monitoreo.

Estos resultados fueron fundamentales para identificar aspectos técnicos y de interfaz a mejorar, así como para confirmar la confiabilidad del sistema durante su implementación en campo.

7 Conclusiones

El presente trabajo logró diseñar, implementar y validar un sistema automático de muestreo de aire activo de bajo flujo, autónomo y de bajo costo, orientado al monitoreo de Compuestos Orgánicos Volátiles (COVs) y parámetros meteorológicos en zonas afectadas por actividad industrial. Se logró que el sistema fuera repetible, integrando sensores, actuadores y componentes cuidadosamente seleccionados bajo criterios de eficiencia y costo-beneficio.

La autonomía energética, la transmisión de datos y la visualización local e inalámbrica del funcionamiento del equipo, demuestran que este sistema es capaz de operar en campo sin supervisión constante, reduciendo el esfuerzo operativo y aumentando su utilidad en aplicaciones comunitarias o académicas.

El posicionamiento adecuado de los sensores meteorológicos es fundamental para asegurar una mayor exactitud en las mediciones. En este sentido, los resultados obtenidos con sensores de bajo costo, comparados con instrumentos comerciales, demuestran un comportamiento adecuado como indicadores de parámetros meteorológicos; aunque, debido a sus limitaciones, no cumplen con las normas oficiales mexicanas para estaciones meteorológicas. Esto posiciona al sistema como una alternativa accesible, de fácil implementación y con resultados aceptables para el monitoreo meteorológico a nivel indicativo, más que como una herramienta de alta exactitud.

En cuanto al control de flujo de aire, el sistema presentó algunas complicaciones durante los primeros meses de muestreo. No obstante, al reemplazar las mangueras de unión por otras con mayor dureza y mejor agarre, se observó una mejora significativa en el desempeño de todos los MABFs durante abril y mayo. Este ajuste permitió identificar una correlación directa entre la presencia de fugas en las uniones y la estabilidad del flujo del sistema. Cuando se realiza una revisión adecuada de fugas y se emplean mangueras con las características adecuadas, el sistema logra una alta precisión en el control del flujo y una excelente exactitud respecto al valor objetivo.

Los resultados de la encuesta confirman que el diseño de las distintas interfaces resultó intuitivo y sencillo de manejar, ya que el 100 % de los usuarios la calificó entre fácil y muy fácil de utilizar. Este nivel de aceptación refleja no solo la claridad en la configuración y operación del sistema, sino también la accesibilidad para usuarios sin contexto técnico del muestreador. Por lo tanto, es notable la facilidad que presentó el

dispositivo en su uso cotidiano y en su manejo con condiciones reales de campo.

Por otro lado, al evaluar el desempeño general de los MABFs, se obtuvo un promedio de 93.49 % de tiempo de operación, lo que evidencia un alto grado de confiabilidad incluso bajo condiciones climáticas adversas. Además, las encuestas permitieron identificar que el nivel de atención y participación de los usuarios influyó directamente en la calidad de las mediciones. Quienes mostraron mayor compromiso lograron los mejores resultados, al verificar posibles fugas y realizar una instalación adecuada.

Adicionalmente, se emiten las siguientes recomendaciones técnicas:

- Considerar cuidadosamente las conexiones de los cartuchos; dependiendo del material, ya que pueden presentarse fugas. Al obstruir las tomas de aire, el flujo debe mantenerse por debajo de 15 [ml/min] si el sistema está bien sellado.
- Evitar el uso de otros dispositivos SPI en la misma red que la micro SD, ya que podrían generar conflictos, aunque las versiones actuales de las bibliotecas parecen haber mitigado este problema.
- Seleccionar la bomba adecuada según el flujo requerido. Una carga poco restrictiva impide alcanzar flujos bajos, mientras que una carga muy obstruida limita la capacidad de alcanzar flujos altos.
- Realizar mantenimientos periódicos al muestreador y al panel solar, especialmente para evitar acumulación de polvo, ya que esto podría afectar tanto la lectura de sensores como la eficiencia de carga del sistema.

8 Trabajo a futuro

Como trabajo a futuro se plantea la incorporación y/o evaluación de nuevos sensores, ya sea mejores a los usados actualmente o que midan otras variables, tales como ozono, presión atmosférica o algunos contaminantes específicos de interés.

En cuanto al software, el programa usado siempre puede estar en mejora continua, de modo que se buscaría seguir optimizando su estructura, mejorando el rendimiento y confiabilidad.De igual manera se plantea la idea de realizar reprogramaciones o actualizaciones al software de manera remota, lo que evitaría las constantes visitas a los lugares donde se encuentren instalados los equipos.

Con respecto a la especiación química, se buscaría una mejor colocación de los cartuchos, esto facilitaría mucho el trabajo al usuario y se reflejaría en mejores muestreos, ya que una buena instalación de los mismos permite que no haya fugas y que el sistema se regule y mantenga un flujo deseado y constante.

Además, se propone incorporar un sistema de enfriamiento que mantenga los trenes a una temperatura inferior a 15 [°C], con el fin de evitar la volatilización de los compuestos atrapados en el cartucho y así lograr una captura más eficiente, tal como se plantea en el artículo [7]. Este sistema debe permanecer encendido durante toda la duración de la prueba y hasta que los cartuchos sean retirados.

A Código

1. Código principal (Main)

1.1. Variables, bibliotecas y objetos utilizados

```
//Pograma principal del Muestreador Automático de Bajo Flujo (MABF)

#include "BluetoothVOC.hpp"
#include "SGP40.hpp"
#include "MicroSD.hpp"
#include "MCP3204_ADC.hpp"
#include "Oled_128x64.hpp"
#include "Anemometro.hpp"
#include "SHT30.hpp"
#include "MCP9600.hpp"
#include "SEN55.hpp"
#include <math.h>
#include <WiFi.h>
#include <ThingSpeak.h>

/*Definimos las credenciales para la conexión a la plataforma*/
  //Canal 1
  /*unsigned long channelID = 2755691;
  const char* WriteAPIKey ="M4RA39CH5EPHV48D";
  const char* ssid="SistemaVOC9";
  const char* password="C02Bia503C";*/
  //Canal 2
  /*unsigned long channelID = 2755737;
  const char* WriteAPIKey ="FO9FH9ZHZY06ITUU";
  const char* ssid="MeteoMexAeria-002";
  const char* password="FR3HYcsm6F8cvcjefdg6";*/
  //Canal 3
  /*unsigned long channelID = 2757431;
  const char* WriteAPIKey ="I4GGGE2LOKGDIUIH";
  const char* ssid="laPutaCasa";
  const char* password="Pa84*art810/SAS";*/
  //Canal 4
    unsigned long channelID = 2757447;
    const char* WriteAPIKey ="P1STTS8GRV5PSUDG";
  //const char* ssid="SistemaVOC9";
  //const char* password="C02Bia503C";
  //const char* ssid="VOC5";
```

```
38   //const char* password="86497640";
39     const char* ssid="instrumentacion";
40     const char* password="cmos$ttl";
41   //Canal 5
42   //unsigned long channelID = 2757467;
43   //const char* WriteAPIKey ="HLQQ9QJI5QE4166N";
44   //const char* ssid="Atmosfera";
45   //const char* password="53gur1d4d4tm";
46   //const char* ssid="Proyector";
47   //const char* password="53gur1d4d";*/
48   //Canal 6
49   /*unsigned long channelID = 2757483;
50   const char* WriteAPIKey ="OF6WA2WMTK0WN0WY";
51   const char* ssid="MeteoMexAeria-004";
52   const char* password="z4dokhzoooiaWesLeLf8";*/
53   //Canal 7
54   /*unsigned long channelID = 2769449;
55   const char* WriteAPIKey ="6HXG2H27T2W1X08H";
56   const char* ssid="INFINITUM008C_2.4";
57   const char* password="erkPeVU3YR";*/
58   //Canal 8
59   /*unsigned long channelID = 2769455;
60   const char* WriteAPIKey ="BRON6EKYKQDCO8E7";
61   const char* ssid="VOCDispositivo7";
62   const char* password="DC54AD065B50";*/
63   //Canal 9
64   /*unsigned long channelID = 2769500;
65   const char* WriteAPIKey ="K5AE27YQHB5N4YDL";
66   const char* ssid="INFINITUM5A38";
67   const char* password="Tr9Fb9Kh7j";*/
68   //Canal 10
69   /*unsigned long channelID = 2774045;
70   const char* WriteAPIKey ="E708ZB3G3YK7L4JN";
71   const char* ssid="TELECABLE-DF92";
72   const char* password="GPON0040DF92";*/
73 /*Definimos el cliente WiFi que usaremos*/
74 WiFiClient cliente;
75
76 bool Wifi = true;
77
78 bool Ctr15MinBool= true, Ctr5MinBool= true, Ctr30SegBool=true,
       Ctr5SegBool=true;
79 int enc = 1;
80 float flujoInst=0, acumFl_Ins=0;
81 int contFl_Ins=0;
82 float windSpdInst=0, acumWind_Ins=0;
83 int contWind_Ins=0;
84
85 float promT=0.0, promH=0.0, promTT_1=0.0, promTTA_1=0.0, promTT_2
       =0.0, promTTA_2=0.0, promCA=0.0, promwindSpd=0.0, prom1p0=0.0,
       prom2p5=0.0, prom4p0=0.0, prom10p0=0.0, promvoc=0.0, promnox=0.0,
        promVB=0.0, promDirVel=0.0;
86
87 int anem;
```

```
88
89 char* BT_Data = "%.2f,%.2f,%.2f,%.2f,%i,%i,%.2f,%.1f,%.2f,%.1f,%.1f
   ,%.1f,%.1f,%.1f,%.1f,";
90 char* BT_Data_prom = "%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f
   ,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f";
```

Listing A.1: bibliotecas y objetos utilizados en el Main.

1.2. Control de válvulas

1.2.1. Control de valvulas por minutos

```
92 //*** Control de valvulas por minutos ***//
93 ////////////////////////////////////////////////////////////////////
94 void ControlValvulasMin(){
95   DateTime fecha = rtc.now();
96
97   if(minutoValv + a >= 60){                //Condicion si se superan
     los 60 min, donde "a" actualiza los respectivos incrementos
98     horaM =(minutoValv + a) - 60;          //Variable que actualiza
     el nuevo valor de tiempo para conmutar
99     if(fecha.minute() == horaM){
100      if(valvBool == true){                //Condicion para realizar
     la accion una sola vez y no todos los 60 segundos
101        digitalWrite(electroValA, HIGH);
102        digitalWrite(electroValB, LOW);
103        a = 2*interv;
104        b = interv;
105        minutoValv = horaM;
106        valvBool = false;
107      }
108    }
109  }
110  else{                                    //Mientras no se superen
       los 60 min entrara a esta condicion
111    if(fecha.minute() == minutoValv + a){  //Compara el minuto
     actual con el requerido comenzando desde a=0
112      if(valvBool == true){                //Condicion para
     realizar la accion una sola vez y no todos los 60 segundos del
     minuto
113        digitalWrite(electroValA, HIGH);
114        digitalWrite(electroValB, LOW);
115        a = 2*interv + a;                  //Variable que actualiza
      cuando debera repetirse esta accion
116        valvBool = false;
117      }
118    }
119  }
120
121  if(minutoValv + b >= 60){                //Condicion si se superan
       los 60 min donde "b" actualiza los respectivos incrementos
122    horaM =(minutoValv + b) - 60;          //Variable que actualiza
       el nuevo valor de tiempo para conmutar
```

```
123     if(fecha.minute() == horaM){
124       if(valvBool == false){                //Condicion para
    realizar la accion una sola vez y no todos los 60 segundos
125         digitalWrite(electroValA, LOW);
126         digitalWrite(electroValB, HIGH);
127         b = 2*interv;
128         a = interv;
129         minutoValv = horaM;
130         valvBool = true;
131       }
132     }
133   }
134   else{
135     if(fecha.minute() == minutoValv + b){   //Compara el minuto
    actual con el requerido comenzando desde b=interv
136       if(valvBool == false){                //Condicion para
    realizar la accion una sola vez y no todos los 60 segundos del
    minuto
137         digitalWrite(electroValA, LOW);
138         digitalWrite(electroValB, HIGH);
139         b = 2*interv + b;                   //Variable que actualiza
     cuando debera repetirse esta accion
140         valvBool = true;
141       }
142     }
143   }
144 }
```

Listing A.2: Función para el control de válvulas por minutos.

1.2.2. Control de válvulas por horas

```
146 //*** Control de valvulas por horas ***//
147 ////////////////////////////////////////////////////////////////
148 void ControlValvulasHor(){
149   DateTime fecha = rtc.now();
150
151   if(horaValv+a >= 24){     //Condicion si se superan los 60 min,
    donde "a" actualiza los respectivos incrementos
152     horaM =(horaValv+a)-24;     //Variable que actualiza el nuevo
    valor de tiempo para conmutar
153     if(fecha.hour() == horaM && fecha.minute() == minutoValv){
154       if(valvBool == true){
155         digitalWrite(electroValA, HIGH);
156         digitalWrite(electroValB, LOW);
157         documento(diaON, mesON, anioON, nomValv, ext, Doc);
158         appendFile(SD, Doc, "Valvula A: ON -- Valvula B: OFF");
159         snprintf(buffer,30,fechaValv,fecha.day(),fecha.month(),fecha
    .year(),fecha.hour(),fecha.minute());
160         appendFile(SD, Doc, buffer);
161
162         a = 2*interv;;
163         b = interv;
```

```
164         horaValv = horaM;
165         valvBool = false;
166       }
167     }
168   }
169   else{
170     if(fecha.hour() == horaValv + a && fecha.minute() == minutoValv)
     { //Compara el minuto actual con el requerido comenzando desde a
     =0
171       if(valvBool == true){     //Condicion para realizar la accion
     una sola vez y no todos los 60 segundos
172         digitalWrite(electroValA, HIGH);
173         digitalWrite(electroValB, LOW);
174         documento(diaON, mesON, anioON, nomValv, ext, Doc);
175         appendFile(SD, Doc, "Valvula A: ON -- Valvula B: OFF");
176         snprintf(buffer,30,fechaValv,fecha.day(),fecha.month(),fecha
    .year(),fecha.hour(),fecha.minute());
177         appendFile(SD, Doc, buffer);
178
179         a = 2*interv + a;
180         valvBool = false;
181       }
182     }
183   }
184
185   if(horaValv+b >= 24){
186     horaM =(horaValv+b)-24;
187     if(fecha.hour() == horaM && fecha.minute() == minutoValv){
188       if(valvBool == false){
189         PID_error=0.0;PID_p=0.0; PID_i=0.0; PID_d=0.0;previous_error
    =0;PID_value=374;
190         Kp=0.08; Ki=0.05; Kd=0.010;
191         digitalWrite(electroValA, LOW);
192         digitalWrite(electroValB, HIGH);
193
194         documento(diaON, mesON, anioON, nomValv, ext, Doc);
195         appendFile(SD, Doc, "Valvula A: OFF -- Valvula B: ON");
196         snprintf(buffer,30,fechaValv,fecha.day(),fecha.month(),fecha
    .year(),fecha.hour(),fecha.minute());
197         appendFile(SD, Doc, buffer);
198
199         b = 2*interv;;
200         a = interv;
201         horaValv = horaM;
202         valvBool = true;
203       }
204     }
205   }
206   else{
207     if(fecha.hour() == horaValv + b && fecha.minute() == minutoValv)
     {
208       if(valvBool == false){
209         PID_error=0.0;PID_p=0.0; PID_i=0.0; PID_d=0.0;previous_error
    =0;PID_value=374;
```

```
210         Kp=0.08; Ki=0.05; Kd=0.010;
211         digitalWrite(electroValA, LOW);
212         digitalWrite(electroValB, HIGH);
213         documento(diaON, mesON, anioON, nomValv, ext, Doc);
214         appendFile(SD, Doc, "Valvula A: OFF -- Valvula B: ON");
215         snprintf(buffer,30,fechaValv,fecha.day(),fecha.month(),fecha
    .year(),fecha.hour(),fecha.minute());
216         appendFile(SD, Doc, buffer);
217
218         b = 2*interv + b;
219         valvBool = true;
220       }
221     }
222   }
223 }
```

Listing A.3: Función para el control de válvulas por horas.

1.3. Obtención de datos

1.3.1. Obtención de datos para 5 segundos

```
225 //*** Obtencion de datos para 5 segundos ***//
226 ////////////////////////////////////////////////////////////////////
227 void Datos5seg(){
228   if(WiFi.status() != WL_CONNECTED){
229     bola_wf=false;
230   }else{
231     bola_wf=true;
232   }
233
234   if(ContrBomb == true){
235     bola_prueba=true;
236   }else{
237     bola_prueba=false;
238   }
239
240   if(EstadoBom==true){
241     flow_rate = acumFl_Ins/contFl_Ins;
242   }else{
243     flow_rate = 0;
244   }
245
246   F=(int)(flow_rate);
247   windSpd=acumWind_Ins/contWind_Ins;
248   T=SHT30_ReadT(); H=SHT30_ReadH(); TT_1=MCP9600_1_ReadT(); TTA_1=
      MCP9600_1_ReadTAmb(); TT_2=MCP9600_2_ReadT(); TTA_2=
      MCP9600_2_ReadTAmb(); CA=Read_SGP40(); DirVel=Veleta(); LectSEN55
      ();PM1p0=sen5T[0];PM2p5=sen5T[1];PM4p0=sen5T[2];PM10p0=sen5T[3];
      voc=sen5T[4];nox=sen5T[5]; VB=Voltaje();
249
250   snprintf(buffer,150,BT_Data,T,H,TT_1,TT_2,CA,F,VB,DirVel,windSpd,
      PM1p0,PM2p5,PM4p0,PM10p0,voc,nox);
```

```
  BT_VOC.print(buffer);

  snprintf(buffer,150,BT_Data_prom,promT,promH,promTT_1,promTT_2,
   promCA,promF,promVB,promDirVel,promwindSpd,prom1p0,prom2p5,
   prom4p0,prom10p0,promvoc,promnox);
  BT_VOC.println(buffer);

  acumFl_Ins=0; contFl_Ins=0;
  acumWind_Ins=0; contWind_Ins=0;
}
```

Listing A.4: Función para obtener datos.

1.3.2. Obtención de datos para 30 segundos

```
//*** Obtencion de datos cada 30 segundos ***//
////////////////////////////////////////////////////////////////////
void Datos30seg(){
  DateTime fecha = rtc.now();


  acumT+=T; acumH+=H; acumTT_1+=TT_1; acumTTA_1+=TTA_1; acumTT_2+=
   TT_2; acumTTA_2+=TTA_2; acumCA+=CA; acumWSpd+=windSpd, acum1p0+=
   PM1p0; acum2p5+=PM2p5; acum4p0+=PM4p0; acum10p0+=PM10p0; acumvoc
   +=voc;acumnox+=nox; acumVB+=VB;
  contP++;

  TMax=tempmax(T); TMin=tempmin(T); windSpdmax=VVmax(windSpd);

  SenCos(DirVel);
  IndicesSen[contP-1]=indSen;
  IndicesCos[contP-1]=indCos;

  int anio=fecha.year(),dia=fecha.day(),mes=fecha.month(),hora=fecha
   .hour(),minuto=fecha.minute(),segundo=fecha.second();
  documento(dia, mes, anio, nomDoc, ext30, Doc);
  if(!SD.exists(Doc)){
    writeFile(SD, Doc, "Fecha,Temperatura(C),Humedad(%),Temperatura
   TP_1(C),Temperatura TP_2(C),Calidad Aire,Flujo(mL/m),Dir. Viento(
   Grad),Vel. Viento(m/s),PM 1.0 ug/m3,PM 2.5 ug/m3,PM 4.0 ug/m3,PM
   10.0 ug/m3,VOC,NOx,Voltaje Bat(V)\n");
  }
  snprintf(buffer,150,msj30,anio,mes,dia,hora,minuto,segundo,T,H,
   TT_1,TT_2,CA,F,DirVel,windSpd,PM1p0,PM2p5,PM4p0,PM10p0,voc,nox,VB
   );
  appendFile(SD, Doc, buffer);

  if(EstadoBom==true && ContrBomb==true){
    acumF+=F;
    contF++;
  }
}
```

Listing A.5: Función para guardar datos.

1.3.3. Promedios para 15 minutos

```
288 //*** Obtencion de promedios para 15 minutos ***//
289 ////////////////////////////////////////////////////////////////////
290 void Prom15Min(){
291   //*** Verifica si ya se configuro la bomba para obtener datos de
       flujo ***//
292   //ControlFlujo();
293   DateTime fecha = rtc.now();
294   if(EstadoBom==true){
295     promF=acumF/contF;
296   }else{
297     promF=0;
298   }
299   promT=acumT/contP, promH=acumH/contP, promTT_1=acumTT_1/contP,
       promTTA_1=acumTTA_1/contP, promTT_2=acumTT_2/contP, promTTA_2=
       acumTTA_2/contP, promCA=acumCA/contP, promwindSpd=acumWSpd/contP,
        prom1p0=acum1p0/contP, prom2p5=acum2p5/contP, prom4p0=acum4p0/
       contP, prom10p0=acum10p0/contP, promvoc=acumvoc/contP, promnox=
       acumnox/contP, promVB=acumVB/contP;
300   float SumSen=0,SumCos=0,convGrad=0;
301   for(int g=0; g < contP; g++){
302     SumSen = SumSen + senDeg[IndicesSen[g]];
303     SumCos = SumCos + cosDeg[IndicesCos[g]];
304   }
305   convGrad= (180.0*atan(SumSen/SumCos))/PI;
306   if(convGrad<0){
307     promDirVel=convGrad+360;
308   }else{
309     promDirVel=convGrad;
310   }
311
312   int anio=fecha.year(),dia=fecha.day(),mes=fecha.month(),hora=fecha
       .hour(),minuto=fecha.minute(),segundo=fecha.second();
313   documento(dia, mes, anio, nomDoc, ext15, Doc);
314   if(!SD.exists(Doc)){
315     writeFile(SD, Doc, "Fecha,Temperatura(C),TemperaturaMax(C),
       TemperaturaMin(C),Humedad(%),Temperatura TP_1(C),Temperatura TP_2
       (C),Calidad Aire,Flujo(mL/m),Dir. Viento(Grad),Vel. Viento(m/s),
       Vel. Viento Max(m/s),PM 1.0 ug/m3,PM 2.5 ug/m3,PM 4.0 ug/m3,PM
       10.0 ug/m3,VOC,NOx,Voltaje Bat(V)\n");
316   }
317   snprintf(buffer,150,msj15,anio,mes,dia,hora,minuto,segundo,promT,
       TMax,TMin,promH,promTT_1,promTT_2,promCA,promF,promDirVel,
       promwindSpd,windSpdmax,prom1p0,prom2p5,prom4p0,prom10p0,promvoc,
       promnox,promVB);
318   appendFile(SD, Doc, buffer);
319
320   if(Wifi == true){
321     ThingSpeak.setField(1,promT);ThingSpeak.setField(2,promH);
       ThingSpeak.setField(3,prom2p5);ThingSpeak.setField(4,promvoc);
       ThingSpeak.setField(5,promnox);ThingSpeak.setField(6,promF);
       ThingSpeak.setField(7,promDirVel);ThingSpeak.setField(8,
       promwindSpd);
```

```
322     ThingSpeak.writeFields(channelID,WriteAPIKey);
323   }
324 
325   TMax=0;
326   TMin=100;
327   windSpdmax=0;
328 
329   acumT=0; acumH=0; acumTT_1=0; acumTTA_1=0; acumTT_2=0; acumTTA_2
       =0; acumCA=0; acumF=0; acumWSpd=0; SumSen=0; SumCos=0; acum1p0=0;
       acum2p5=0;acum4p0=0;acum10p0=0;acumvoc=0;acumnox=0; acumVB=0;
330   contP=0; contF=0;
331 }
```

Listing A.6: Función para guardar promedios.

1.3.4. Valores máximos y mínimos de algunas variables y función auxiliar para ángulos de veleta

```
333 //*** Obtención de temperatura máxima ***//
334 ////////////////////////////////////////////////////////////////////
335 float tempmax(float temp){
336   if (temp>TMax){
337     TMax=temp;
338   }
339   return TMax;
340 }
341 
342 //*** Obtención de temperatura mínima ***//
343 ////////////////////////////////////////////////////////////////////
344 float tempmin(float temp){
345   if (temp < TMin ){
346     TMin = temp;
347   }
348  return TMin;
349 }
350 
351 //*** Obtención de velocidad del viento máxima ***//
352 ////////////////////////////////////////////////////////////////////
353 float VVmax(float vv){
354   if (vv>windSpdmax){
355     windSpdmax=vv;
356   }
357   return windSpdmax;
358 }
359 
360 //*** Obtención de indices para el calculo de los Senos y Cosenos
       ***//
361 ////////////////////////////////////////////////////////////////////
362 void SenCos(float angulo){
363   if(angulo == 112.5 || angulo == 67.5){
364     indSen=0;
365   }else if(angulo == 90){
366     indSen=1;
```

```
367   }else if(angulo == 157.5 || angulo == 22.5){
368     indSen=2;
369   }else if(angulo == 135 || angulo == 45){
370     indSen=3;
371   }else if(angulo == 202.5 || angulo == 337.5){
372     indSen=4;
373   }else if(angulo == 180 || angulo == 0){
374     indSen=5;
375   }else if(angulo == 247.5 || angulo == 292.5){
376     indSen=6;
377   }else if(angulo == 225 || angulo == 315){
378     indSen=7;
379   }else{
380     indSen=8;
381   }
382
383   if(angulo == 112.5 || angulo == 247.5){
384     indCos=0;
385   }else if(angulo == 67.5 || angulo == 292.5){
386     indCos=1;
387   }else if(angulo == 90 || angulo == 270){
388     indCos=2;
389   }else if(angulo == 157.5 || angulo == 202.5){
390     indCos=3;
391   }else if(angulo == 135 || angulo == 225){
392     indCos=4;
393   }else if(angulo == 180){
394     indCos=5;
395   }else if(angulo == 22.5 || angulo == 337.5){
396     indCos=6;
397   }else if(angulo == 45 || angulo == 215){
398     indCos=7;
399   }else{
400     indCos=8;
401   }
402 }
```

Listing A.7: Funciones auxiliares para obtener datos.

1.4. Funciones para el control de tiempos

1.4.1. Control cada 15 minutos

```
404 //*** Control para realizar acciones cada 15 minutos  ***//
405 ////////////////////////////////////////////////////////////////////
406 void Ctr15Min(){
407   DateTime fecha = rtc.now();
408   if((fecha.minute() == 0 || fecha.minute() == 15 || fecha.minute()
      == 30 || fecha.minute() == 45) && fecha.second() == 0){
409       if(Ctr15MinBool==true){
410           Prom15Min();
411           Ctr15MinBool=false;
412         }
```

```
413     }
414   if(fecha.minute() == 1 || fecha.minute() == 16 || fecha.minute()
      == 31 || fecha.minute() == 46){
415     Ctr15MinBool=true;
416   }
417 }
```

Listing A.8: Función para controlar acciones cada 15 minutos.

1.4.2. Control cada 5 minutos

```
419 //*** Control para realizar acciones cada 5 minutos, verifica flujo
      y conexión a internet ***//
420 ////////////////////////////////////////////////////////////////////
421 void Ctr5Min(){
422   DateTime fecha = rtc.now();
423   if(fecha.minute()==1 || fecha.minute() == 6 || fecha.minute() ==
      11 || fecha.minute() == 16 || fecha.minute() == 21 || fecha.
      minute() == 26 || fecha.minute() == 31
424   || fecha.minute() == 36 || fecha.minute() == 41 || fecha.minute()
      == 46 || fecha.minute() == 51 || fecha.minute() == 56){
425     if(Ctr5MinBool==true){
426       if(ContrOFFBool == true && (flow_rate > (Setpoint * 1.2) ||
      flow_rate < (Setpoint * 0.8))){
427         PID_error=0.0;PID_p=0.0; PID_i=0.0; PID_d=0.0;previous_error
      =0;PID_value=374;
428         Kp=0.08; Ki=0.05; Kd=0.010;
429       }
430       int j=0;
431       mySgp40.setRhT(/*relativeHumidity = */ SHT30_ReadH(), /*
      temperatureC = */ SHT30_ReadT());
432       if(Wifi == true){
433         //Serial.println(WiFi.status());
434         ThingSpeak.begin(cliente);
435         if(WiFi.status() != WL_CONNECTED){
436           WiFi.reconnect();
437           while(WiFi.status() != WL_CONNECTED){
438             if(j > 100){
439               break;
440             }
441             j++;
442           }
443         }
444       }
445       Ctr5MinBool=false;
446     }
447   }
448   if(fecha.minute()==2 || fecha.minute() == 7 || fecha.minute() ==
      12 || fecha.minute() == 17 || fecha.minute() == 22 || fecha.
      minute() == 27 || fecha.minute() == 32
449   || fecha.minute() == 37 || fecha.minute() == 42 || fecha.minute()
      == 47 || fecha.minute() == 52 || fecha.minute() == 57){
450     Ctr5MinBool=true;
```

```
451   }
452 }
```

Listing A.9: Función para controlar acciones cada 5 minutos.

1.4.3. Control cada 5 segundos

```
454 //*** Control para realizar acciones cada 5 segundos  ***//
455 ////////////////////////////////////////////////////////////////////
456 void Ctr5Seg(){
457   DateTime fecha = rtc.now();
458   //if(/*fecha.second() == 0 ||*/ fecha.second() == 8 || fecha.
      second() == 13 || fecha.second() == 18 || fecha.second() == 23 ||
       fecha.second() == 28 || /*fecha.second() == 30
459   //|| */fecha.second() == 38 || fecha.second() == 43 || fecha.
      second() == 48 || fecha.second() == 53 || fecha.second() == 58){
460   if(fecha.second() == 1 || fecha.second() == 6 || fecha.second() ==
       11 || fecha.second() == 16 || fecha.second() == 21 || fecha.
      second() == 26 || fecha.second() == 31 || fecha.second() == 36 ||
       fecha.second() == 41 || fecha.second() == 46 || fecha.second()
      == 51 || fecha.second() == 56){
461     if(Ctr5SegBool==true){
462       Datos5seg();
463       Ctr5SegBool=false;
464     }
465   }
466   //if(/*fecha.second() == 1 || */fecha.second() == 9 || fecha.
      second() == 14 || fecha.second() == 19 || fecha.second() == 24 ||
       fecha.second() == 29 || /*fecha.second() == 31
467   //|| */fecha.second() == 39 || fecha.second() == 44 || fecha.
      second() == 49 || fecha.second() == 54 || fecha.second() == 59){
468   if(fecha.second() == 2 || fecha.second() == 7 || fecha.second() ==
       12 || fecha.second() == 17 || fecha.second() == 22 || fecha.
      second() == 27 || fecha.second() == 32 || fecha.second() == 37 ||
       fecha.second() == 42 || fecha.second() == 47 || fecha.second()
      == 52 || fecha.second() == 57){
469     Ctr5SegBool=true;
470   }
471 }
472
473 //*** Control para realizar acciones cada 30 segundos ***//
474 ////////////////////////////////////////////////////////////////////
475 void Ctr30Seg(){
476   DateTime fecha = rtc.now();
477   if(fecha.second() == 3 || fecha.second() == 33){
478   //if(fecha.second() == 2 || fecha.second() == 32){
479     if(Ctr30SegBool==true){
480       Datos30seg();
481       Ctr30SegBool=false;
482     }
483   }
484   if(fecha.second() == 4 || fecha.second() == 34){
485   //if(fecha.second() == 3 || fecha.second() == 33){
```

```
    Ctr30SegBool=true;
  }
}
```

Listing A.10: Función para controlar acciones cada 5 segundos.

1.4.4. Control cada 30 segundos

```
//*** Control para realizar acciones cada 30 segundos ***//
////////////////////////////////////////////////////////////////////
void Ctr30Seg(){
  DateTime fecha = rtc.now();
  if(fecha.second() == 3 || fecha.second() == 33){
  //if(fecha.second() == 2 || fecha.second() == 32){
    if(Ctr30SegBool==true){
      Datos30seg();
      Ctr30SegBool=false;
    }
  }
  if(fecha.second() == 4 || fecha.second() == 34){
  //if(fecha.second() == 3 || fecha.second() == 33){
    Ctr30SegBool=true;
  }
}
```

Listing A.11: Función para controlar acciones cada 30 segundos.

1.5. Casos para Bluetooh

```
//*** Casos para el Bluetooth  ***//
////////////////////////////////////////////////////////////////////
void CaseBT(){
  DateTime fecha = rtc.now();     // Funcion que devuelve fecha y
   horario en formato

  Cadena = BT_VOC.readStringUntil('\n');
  Dato = Cadena.substring(0,1);
  var = Dato.charAt(0);
  switch (var){
    case 'A':
      Dato1 = Cadena.substring(2,4); Dato2 = Cadena.substring(5,7);
   Dato3 = Cadena.substring(8,10);
      Dato4 = Cadena.substring(11,13); Dato5 = Cadena.substring
   (14,16);
      diaON = Dato1.toInt(); mesON = Dato2.toInt(); horaON = Dato3.
   toInt(); minON = Dato4.toInt(); interv = Dato5.toInt();
      TPruebaHor = 2*(Dato5.toInt());
      TPruebaMin = 59;
      horaValv = horaON; minutoValv = minON;

      Serial.print("Prueba establecida el (dia/mes): "); Serial.
   print(diaON); Serial.print("/"); Serial.println(mesON);
```

```
525       Serial.print("Hora: "); Serial.print(horaON); Serial.print(":"
    ); Serial.print(minON); Serial.println(" hrs");
526       Serial.print("Flujo a: "); Serial.print(Setpoint); Serial.
    println(" mL/min");
527
528       a=0;
529       b=interv;
530       ContrBombBool = true;
531       ContrBomb = true;
532
533       break;
534     case 'B':
535       Serial.print("Comienza prueba el dia de hoy (dia/mes): ");
    Serial.print(fecha.day()); Serial.print("/");
536       Serial.println(fecha.month());
537       Serial.print("Hora: "); Serial.print(fecha.hour()); Serial.
    print(":"); Serial.print(fecha.minute()); Serial.println(" hrs");
538       horaValv = fecha.hour(); minutoValv = fecha.minute();
539       anioON = fecha.year(); diaON = fecha.day(); mesON = fecha.
    month();
540       horaON = fecha.hour(); minON = fecha.minute();
541
542
543       Serial.print("Flujo a: "); Serial.print(Setpoint); Serial.
    println(" mL/min");
544       Serial.println(" ");
545
546       ContrValv = true;
547       valvBool = true;
548       a=0;
549       b=interv;
550
551       ContrBombBool = true;
552       ContrBomb = true;
553
554       break;
555     case 'C':
556       ContrValv = true;
557       valvBool = true;
558       digitalWrite(electroValA, LOW);
559       digitalWrite(electroValB, LOW);
560       Dato1 = Cadena.substring(2,4);
561       Dato2 = Cadena.substring(5,7);
562       Dato3 = Cadena.substring(8,10);
563       horaValv = Dato1.toInt();
564       minutoValv = Dato2.toInt();
565       interv = Dato3.toInt();
566       a=0;
567       b=interv;
568       BT_VOC.print("Las valvulas conmutaran a las "); BT_VOC.print(
    horaValv); BT_VOC.print(":");
569       BT_VOC.print(minutoValv); BT_VOC.println(" hrs");
570       BT_VOC.print("Con un intervalo de "); BT_VOC.print(interv);
    BT_VOC.println(" hrs");
```

```
571       BT_VOC.println(" ");
572
573       break;
574     case 'D':
575       Dato1 = Cadena.substring(2,4); Dato2 = Cadena.substring(5,7);
    Dato3 = Cadena.substring(8,10);
576       Dato4 = Cadena.substring(11,13);
577       diaOFF = Dato1.toInt(); mesOFF = Dato2.toInt(); horaOFF =
    Dato3.toInt(); minOFF = Dato4.toInt();
578       BT_VOC.print("La prueba se detendra el (dia/mes): "); BT_VOC.
    print(diaOFF);BT_VOC.print("/");BT_VOC.println(mesOFF);
579       BT_VOC.print("Hora: "); BT_VOC.print(horaOFF); BT_VOC.print(":
    "); BT_VOC.print(minOFF); BT_VOC.println(" hrs");
580       BT_VOC.println(" ");
581       ContrOFF = true;
582       ContrOFFBool = true;
583
584       break;
585     case 'F':
586       ContrBomb = false;
587       ContrOFF = false;
588       ContrOFFBool = false;
589
590       ContrValv = false;
591       valvBool = false;
592
593       PararBom();
594       digitalWrite(electroValA, LOW);
595       digitalWrite(electroValB, LOW);
596       digitalWrite (vent, LOW);
597       Dato1 = Cadena.substring(2,6);
598       Setpoint = Dato1.toDouble();
599       Serial.println(" ");
600       Serial.print("Flujo establecido a: "); Serial.print(Setpoint);
     Serial.println(" mL/min");
601       Serial.println(" ");
602
603       break;
604     case 'P':
605       ContrBomb = false;
606       ContrOFF = false;
607       ContrOFFBool = false;
608
609       ContrValv = false;
610       valvBool = false;
611
612       PararBom();
613       Setpoint = 70;
614       digitalWrite(electroValA, LOW);
615       digitalWrite(electroValB, LOW);
616       digitalWrite (vent, LOW);
617
618       Serial.println(" ");
619       Serial.println("Prueba detenida");
```

```
620       Serial.println(" ");
621
622       break;
623     case 'V':{
624       Dato1 = Cadena.substring(2,3);
625       Dato2 = Cadena.substring(4,7);
626       char dat = Dato1.charAt(0);
627
628       anioON = fecha.year(); diaON = fecha.day(); mesON = fecha.
    month();
629       horaON = fecha.hour(); minON = fecha.minute();
630       horaValv = horaON; minutoValv = minON;
631       switch (dat){
632         case '1':
633           digitalWrite(vent, HIGH);
634           digitalWrite(electroValA, HIGH);
635           digitalWrite(electroValB, LOW);
636
637           //***Horas***
638           TPruebaHor = Dato2.toInt();
639           TPruebaMin = 59;
640
641           //***Minutos***
642           //TPruebaMin = Dato2.toInt();
643
644           EncenderBom();
645           OffAutomatico();
646           ContrOFF = true;
647           ContrOFFBool = true;
648
649           break;
650         case '2':
651           digitalWrite(vent, HIGH);
652           digitalWrite(electroValA, LOW);
653           digitalWrite(electroValB, HIGH);
654
655           //***Horas***
656           TPruebaHor = Dato2.toInt();
657           TPruebaMin = 59;
658
659           //***Minutos***
660           //TPruebaMin = Dato2.toInt();
661
662           EncenderBom();
663           OffAutomatico();
664           ContrOFF = true;
665           ContrOFFBool = true;
666
667           break;
668         case '3':
669           interv = Dato2.toInt();
670           //***Horas***
671           TPruebaHor = 2*(Dato2.toInt());
672           TPruebaMin = 59;
```

```
673
674           //***Minutos***
675           //TPruebaMin = 2*(Dato2.toInt());
676
677           a=0;
678           b=interv;
679
680           ContrBombBool = true;
681           ContrBomb = true;
682
683           break;
684         default:
685           // statements
686           break;
687       }
688       break;
689     }
690     case 'K':{
691       Dato1 = Cadena.substring(2,6); Dato2 = Cadena.substring(7,11);
      Dato3 = Cadena.substring(12,16);
692       Kp = Dato1.toDouble();
693       Ki = Dato2.toDouble();
694       Kd = Dato3.toDouble();
695       //myPID.SetTunings(Kp,Ki,Kd);
696       BT_VOC.print(Kp);BT_VOC.print(" ");
697       BT_VOC.print(Ki);BT_VOC.print(" ");
698       BT_VOC.print(Kd);BT_VOC.println(" ");
699       break;
700     }
701     default:
702       // statements
703       break;
704   }
705 }
```

Listing A.12: Función para verificar que hacer cuando llega un comando por bluetooth.

1.6. Control de pruebas programadas

1.6.1. Encendido de prueba

```
490 //*** Encendido de la bomba programada  ***//
491 ////////////////////////////////////////////////////////////////////
492 void BombaON(){
493   if(ContrBomb == true){
494     DateTime fecha = rtc.now();
495     if(fecha.day() == diaON && fecha.month() == mesON && fecha.hour
    () == horaON && fecha.minute() == minON){ // si hora = 14 y
    minutos = 30
496       if(ContrBombBool == true){       //Para evitar ingresar mas de
     una vez
497         ContrBombBool = false;
498         ContrBomb = false;
```

```
499         anioON=fecha.year();
500         ContrValv = true;
501         valvBool = true;
502         digitalWrite (vent, HIGH);
503         EncenderBom();
504         OffAutomatico();
505 
506         ContrOFF = true;
507         ContrOFFBool = true;
508 
509         Serial.println(" ");
510         Serial.println("Comienza Prueba");
511 
512         if(TPruebaHor != 24){
513           Serial.print("La prueba se detendrá a las: "); Serial.
    print(horaOFF); Serial.print(":");
514           Serial.print(minOFF); Serial.println(" hrs");
515           Serial.println(" ");
516         }
517         else{
518           Serial.print("La prueba se detendrá mañana a las: ");
    Serial.print(horaOFF); Serial.print(":");
519           Serial.print(minOFF); Serial.println(" hrs");
520           Serial.println(" ");
521         }
522       }
523     }
524   }
525 }
```

Listing A.13: Función para controlar el encendido de una prueba programada.

1.6.2. Apagado de prueba

```
490 //*** Apagado de la bomba programada  ***//
491 ////////////////////////////////////////////////////////////////////
492 void BombaOFF(){
493   if(ContrOFF == true){
494     DateTime fecha = rtc.now();
495     if(fecha.hour() == horaOFF && fecha.minute() == minOFF){
496       if(ContrOFFBool == true){       //Para evitar ingresar mas de
    una vez
497         ContrOFFBool = false;
498         ContrValv = false;
499         ContrBomb = false;
500         PararBom();
501         Setpoint = 70;
502         digitalWrite(electroValA, LOW);
503         digitalWrite(electroValB, LOW);
504         digitalWrite(vent, LOW);
505 
506         Serial.println(" ");
507         Serial.println("Prueba detenida");
```

```
508         Serial.println(" ");
509       }
510     }
511   }
512 }
```

Listing A.14: Función para controlar el apagado de una prueba programada.

1.7. Control de flujo

```
707 //*** Control de flujo de aire con PID  ***//
708 ////////////////////////////////////////////////////////////////
709 void ControlFlujo(){
710   if(EstadoBom==true){
711       flujoInst = SensorFlujo(Read_MCP3208(0));
712       acumFl_Ins+=flujoInst;
713       contFl_Ins++;
714       //BT_VOC.print(flow_rate);BT_VOC.print("  ");
715 
716       // Cálculo del error PID
717       //PID_error = (flujoInst - (Setpoint+3));
718       PID_error = (flujoInst - (Setpoint));
719 
720       // Ajuste dinámico de los parámetros PID
721       if(PID_error < (Setpoint*0.60) && PID_error > -(Setpoint*0.60)
    ){
722         if(PID_error < (Setpoint*0.15) && PID_error > -(Setpoint
    *0.15)){
723           Kp=0.002; Ki=0.002; Kd=0.002;
724           //Kp=0.001, Ki=0.0, Kd=0.001;
725         }
726         else{
727           Kp=0.004, Ki=0.01, Kd=0.005;
728         }
729         //Kp=0.002, Ki=0.04, Kd=0.002;
730         //Kp=0.002, Ki=0.03, Kd=0.001;
731       }
732       else{
733         Kp=0.07; Ki=0.05; Kd=0.010;
734       }
735 
736       // Cálculo del término proporcional
737       PID_p = Kp * PID_error;
738 
739       // Cálculo del término integral con anti-windup
740       PID_i += Ki * PID_error;
741       if(PID_i > 500) PID_i = 500;
742       if(PID_i < -500) PID_i = -500;
743 
744       // Cálculo del término derivativo con mejor control del tiempo
745       timePrev = Time;
746       Time = millis();
```

```
747       elapsedTime = (Time - timePrev) / 1000.0; // Conversión a
      segundos
748
749       if(elapsedTime > 0) { // Prevenir división por cero
750         PID_d = Kd * ((PID_error - previous_error) / elapsedTime);
751       }else{
752         PID_d = 0;
753       }
754
755       // Suma de los términos PID
756       PID_value = PID_p + PID_i + PID_d;
757
758       // Limitar la salida PID
759       if(PID_value < 0){
760         PID_value = 0;
761       }
762       if(PID_value > 1000){ //374
763         PID_value = 1000;   //374
764       }
765
766       // Ajuste de velocidad de la bomba
767       vel = 1024 - PID_value;
768       EncenderBom();
769
770       // Almacenar el error actual como el error previo
771       previous_error = PID_error;
772
773       //Gráficas del control PID
774       //Serial.print(flow_rate);
775       //Serial.print(",");
776       //Serial.print(flujoInst);
777       //Serial.print(",");
778       //Serial.print(Setpoint);
779       //Serial.print(",");
780       //Serial.println(PID_error);
781   }
782   else{
783     flujoInst = SensorFlujo(Read_MCP3208(0));
784   }
785 }
```

Listing A.15: Función para controlar el flujo de la bomba.

1.8. Setup y loop del código main

```
787 //*** Setup  ***//
788 ////////////////////////////////////////////////////////////////////
789 void setup(){
790   Serial.begin(115200);     // inicializa comunicacion serie a 115200
      bps
791
792   pinMode(electroValA, OUTPUT);
793   pinMode(electroValB, OUTPUT);
```

```
794   pinMode(vent, OUTPUT);
795   digitalWrite(electroValA, LOW);
796   digitalWrite(electroValB, LOW);
797   digitalWrite (vent, LOW);
798
799   if(Wifi == true){
800     WiFi.begin(ssid,password);
801     while(WiFi.status() != WL_CONNECTED){
802     }
803     /*Una vez conectado, se imprimirá una frase y se iniciará la
       conexión a la Plataforma usando el cliente definido anteriormente
       */
804     Serial.println("
       ----------------------------------------------------");
805     Serial.println("¡Conectado al WiFi");
806     Serial.print("Current ESP32 IP: ");
807     Serial.println(WiFi.localIP());
808     Serial.print("Gateway (router) IP: ");
809     Serial.println(WiFi.gatewayIP());
810     Serial.print("Subnet Mask: " );
811     Serial.println(WiFi.subnetMask());
812     Serial.print("Primary DNS: ");
813     Serial.println(WiFi.dnsIP(0));
814     Serial.print("Secondary DNS: ");
815     Serial.println(WiFi.dnsIP(1));
816     Serial.println("
       ----------------------------------------------------");
817     ThingSpeak.begin(cliente);
818   }
819
820   BTVOC_Init();  //Inicialización del Bluetooth
821   OLED_Init();   //Inicialización de la oled (1)
822   Reloj_Init();  //Inicialización del reloj en tiempo real (2)
823   Bomba_Init();  //Inicialización del la bomba (11)
824   SHT30_Init();  //Inicialización del higrómetro (3)
825   MCP9600_Init();//Inicialización de los modulos de termopar (4 y 5)
826   SGP40_Init();  //Inicialización del del sensor de indice de
       calidad del aire (6)
827   SEN55_Init();  //Inicialización del SEN55 (7)
828   SD_Init();     //Inicialización del modulo para la memoria microSD
       (8 y 9)
829   pinMode(ANEMOMETER_PIN, INPUT);
830
831   attachInterrupt(digitalPinToInterrupt(ANEMOMETER_PIN), isr,
       FALLING);
832   Setpoint = 70;
833   interv = 12;
834   TPruebaHor = 24;
835   TPruebaMin = 59;
836
837   TMax=0;
838   TMin=100;
839   BT_VOC.println("Sistema VOC Iniciado");
840
```

```
841   DateTime fecha = rtc.now();
842   documento(fecha.day(), fecha.month(), fecha.year(), nomDoc, ext30,
        Doc);
843   if(!SD.exists(Doc)){
844     writeFile(SD, Doc, "Fecha,Temperatura(C),Humedad(%),Temperatura
      TP_1(C),Temperatura TP_2(C),Calidad Aire,Flujo(mL/m),Dir. Viento(
      Grad),Vel. Viento(m/s),PM 1.0 ug/m3,PM 2.5 ug/m3,PM 4.0 ug/m3,PM
      10.0 ug/m3,VOC,NOx,Voltaje Bat(V)\n");
845   }
846   documento(fecha.day(), fecha.month(), fecha.year(), nomDoc, ext15,
        Doc);
847   if(!SD.exists(Doc)){
848     writeFile(SD, Doc, "Fecha,Temperatura(C),TemperaturaMax(C),
      TemperaturaMin(C),Humedad(%),Temperatura TP_1(C),Temperatura TP_2
      (C),Calidad Aire,Flujo(mL/m),Dir. Viento(Grad),Vel. Viento(m/s),
      Vel. Viento Max(m/s),PM 1.0 ug/m3,PM 2.5 ug/m3,PM 4.0 ug/m3,PM
      10.0 ug/m3,VOC,NOx,Voltaje Bat(V)\n");
849   }
850 }
851
852 //*** Loop  ***//
853 ////////////////////////////////////////////////////////////////////
854 void loop (){
855   _timer = millis();
856   if (_timer > _nextCalc){
857     _nextCalc = _timer + CALC_INTERVAL;
858     windSpdInst = readWindSpd();
859     acumWind_Ins+=windSpdInst;
860     contWind_Ins++;
861   }
862
863   OLED_MENU();
864
865   dirX = mcp3208.readADC(2);; // read X axis value [0..1023]
866   dirY = mcp3208.readADC(3);; // read Y axis value [0..1023]
867
868   //*** Verifica si se manda algo por Bluetooth ***//
869   if(BT_VOC.available()){
870     CaseBT();
871   }
872
873   //*** Verifica si se llega a la hora especificada para encender la
        bomba ***//
874   BombaON();
875
876   //*** Verifica si se llega a la hora especificada para apagar la
      bomba ***//
877   BombaOFF();
878
879   //*** Verifica si ya se configuro el control de las valvulas ***//
880   if(ContrValv == true){
881     //ControlValvulasMin();
882     ControlValvulasHor();
883   }
```

```
884
885   //*** Verifica si ya se configuro la bomba para obtener datos de
        flujo ***//
886   ControlFlujo();
887
888   if(manual == true && click1 == true){
889     if(sel1 == false && sel2 == true){
890       digitalWrite(vent, HIGH);
891       EncenderBom();
892
893       digitalWrite(electroValA, HIGH);
894       digitalWrite(electroValB, LOW);
895     }else if(sel1 == true && sel2 == false){
896       digitalWrite(vent, HIGH);
897       EncenderBom();
898
899       digitalWrite(electroValA, LOW);
900       digitalWrite(electroValB, HIGH);
901     }else if(sel1 == false && sel2 == false){
902       digitalWrite(vent, HIGH);
903       EncenderBom();
904
905       digitalWrite(electroValA, HIGH);
906       digitalWrite(electroValB, HIGH);
907     }else if(sel1 == true && sel2 == true){
908       PararBom();
909       digitalWrite(vent, LOW);
910
911       digitalWrite(electroValA, LOW);
912       digitalWrite(electroValB, LOW);
913     }
914   }
915   //*** Obtención de promedios cada 15 minutos ***//
916   Ctr15Min();
917   Ctr5Min();
918
919   //*** Obtención de datos cada 5 segundos en pantalla***//
920   Ctr5Seg();
921
922   //*** Obtención de datos cada 30 segundos para promedios***//
923   Ctr30Seg();
924
925   if(cuentaSL<=187){
926     cuentaSL++;
927   }else{
928     sleepp=true;
929   }
930
931   //*** Verifica si ya se configuro la bomba para obtener datos de
        flujo ***//
932   ControlFlujo();
933 }
```

Listing A.16: Función para controlar el apagado de una prueba programada.

2. Biblioteca para el anemómetro

2.1. Variables utilizadas

```
//Biblioteca para obtener lecturas del anemometro

#define DEBOUNCE_TIME 10
#define CALC_INTERVAL 1000
#define ANEMOMETER_PIN 35

volatile uint32_t _anemometerCounter = 0;
volatile unsigned long last_micros_an = 0;

unsigned long _nextCalc = 0;
unsigned long _timer = 0;
float  tiemp=0.0,velv=0.0;
```

Listing A.17: Variables usadas para el anemómetro.

2.2. Funciones para el uso y obtención de datos

```
//** Función de interrupción: Cuenta los pulsos del anemómetro **//
////////////////////////////////////////////////////////////////////
void IRAM_ATTR isr() {
  unsigned long currentMicros = micros();
  if (currentMicros - last_micros_an >= DEBOUNCE_TIME * 1000) {
    _anemometerCounter++;
    last_micros_an = currentMicros;
  }
}

//*** Función para calcular la velocidad del viento en m/s ***//
////////////////////////////////////////////////////////////////////
float readWindSpd() {
  noInterrupts();  // Deshabilita interrupciones para lectura segura
  uint32_t counterCopy = _anemometerCounter;
  _anemometerCounter = 0;  // Reinicia el contador
  interrupts();  // Reactiva interrupciones

  float spd = (counterCopy/2.2) * 0.6671;  // Convierte pulsos a
   velocidad en m/s
  return spd;
}
```

Listing A.18: Funciones para obtener datos del anemómetro.

3. Biblioteca para *Bluetooth*

3.1. Variables, bibliotecas y objetos utilizados

```
//Biblioteca para el Bluetooth de la ESP32.

#include "BluetoothSerial.h"

//#define USE_PIN //Uncomment this to use PIN during pairing. The pin is specified on the line below
const char *pin = "VOC"; // Change this to more secure PIN.
String nombre = "Sistema VOC_M1";

#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and enable it
#endif

#if !defined(CONFIG_BT_SPP_ENABLED)
#error Serial Bluetooth not available or not enabled. It is only available for the ESP32 chip.
#endif

BluetoothSerial BT_VOC;
```

Listing A.19: Bibliotecas, variables y objetos utilizados para el bluetooth.

3.2. Funciones para el uso y obtención de datos

```
//*** Inicialización del Bluetooth ***//
////////////////////////////////////////////////////////////////
void BTVOC_Init(){
  BT_VOC.begin(nombre); //Bluetooth device name|
  Serial.println("-----------------------------------------------");
  Serial.printf("El dispositivo con nombre \"%s\" está listo.\nAhora puede conectarse a Bluetooth!\n", nombre.c_str());
  Serial.println("-----------------------------------------------");
  #ifdef USE_PIN
    BT_VOC.setPin(pin);
    Serial.println("Using PIN");
  #endif
}
```

Listing A.20: Funcion para inicializar el bluetooth.

4. Biblioteca para la bomba

4.1. Variables utilizadas

```
//Biblioteca para el control de la bomba

//Propiedades PWM
const int frecuencia = 5000;
```

```
const int canal = 0;
const int resolucion = 10;

//Bomba
#define bom1 33
#define bom2 32
#define PWMB 25
bool EstadoBom = false;
int vel=0;
int interv, TPruebaHor, TPruebaMin;

#define electroValA 26
#define electroValB 27
#define vent 17

//PID
float PID_error=0.0, PID_p=0.0, PID_i=0.0, PID_d=0.0, Time, timePrev
    , previous_error=0, elapsedTime;
int PID_value=0;

double Kp=0.08, Ki=0.05, Kd=0.010;
```

Listing A.21: Variables usadas para el control de la bomba.

4.2. Funciones para el control de la bomba/puente H

```
//*** Inicialización del PWM para la bomba ***//
////////////////////////////////////////////////////////////////////
void Bomba_Init(){
  ledcSetup(canal, frecuencia, resolucion); //configuramos la
    funcionalidad PWM
  ledcAttachPin(PWMB, canal);   //Asociamos el canal al GPIO

  pinMode (bom1, OUTPUT);
  pinMode (bom2, OUTPUT);
}

//*** Encender bomba ***//
////////////////////////////////////////////////////////////////////
void EncenderBom(){
  digitalWrite (bom1, HIGH);
  digitalWrite (bom2, LOW);
  ledcWrite(canal, vel);
  EstadoBom=true;
}

//*** Detener bomba y reinicio de párametros del PID***//
////////////////////////////////////////////////////////////////////
void PararBom(){
  PID_error=0.0;PID_p=0.0; PID_i=0.0; PID_d=0.0;previous_error=0;
    PID_value=0;
  Kp=0.08; Ki=0.05; Kd=0.010;

```

```
  interv = 12;
  TPruebaHor = 24;
  TPruebaMin = 59;
  digitalWrite (bom1, LOW);
  digitalWrite (bom2, LOW);
  ledcWrite(canal, 0);
  EstadoBom=false;
}
```

Listing A.22: Funciones para el control de la bomba.

5. Biblioteca para el ADC

5.1. Variables, bibliotecas y objetos utilizados

```
//Biblioteca para elc control del ADC de 4 canales, para leer la veleta y sensor de flujo

#include "MCP320X.h"

// (Optional) Define model and SPI pins
#define CS_PIN 5     // ESP8266 default SPI pins
#define CLOCK_PIN 18  // Should work with any other GPIO pins, since the library does not formally
#define MOSI_PIN 23   // use SPI, but rather performs pin bit banging to emulate SPI communication.
#define MISO_PIN 19   //
#define MCP3208 8     // (Generally "#define MCP320X X", where X is the last model digit/number of inputs)

float flow_rate = 0;

int lectVel = 0, lectVolt=0;
// These arrays are specific to the ADS wind vane with a 10kOhm fixed resistor in the voltage divider
float dirDeg[] = {112.5,67.5,90,157.5,135,202.5,180,22.5,45,247.5,225,337.5,0,
292.5,315,270};
int sensorMin[] = {240,296,346,411,651,851,991,1301,1651,1901,2301,2451,2701,3056,
3201,3401};
int sensorMax[] = {295,345,410,650,850,990,1300,1650,1900,2300,2450,2700,3055,
3200,3400,3700};

float senDeg[] = {0.9238,1,0.3826,0.7071,-0.3826,0,-0.9238,-0.7071,-1};
float cosDeg[] = {-0.3826,0.3826,0,-0.9238,-0.7071,-1,0.9238,0.7071,1};
int IndicesSen[30];
int IndicesCos[30];
```

```

// Create an instance of MCP320X object.
MCP320X mcp3208(MCP3208, CLOCK_PIN, MOSI_PIN, MISO_PIN, CS_PIN);
```

Listing A.23: Variables, bibliotecas y objetos utilizados para el ADC.

5.2. Funciones para el uso y obtención de datos

```
//*** Lectura del ADC en voltaje ***//
////////////////////////////////////////////////////////////////////
float Read_MCP3208(int x) {
  float lectura;
  lectura=((float)mcp3208.readADC(x)*5.0)/4095.0;
  delay(40);
  return lectura;
}

//*** Conversión del dato binario a voltaje y flujo ***//
////////////////////////////////////////////////////////////////////
float SensorFlujo(float val){
    float y=13.496*(pow(val,3))- 0.3812*(pow(val,2))+ 207.5*(val) -
   100.36;
    if(val<=0.5){
        y=0;
    }
    return y;
}

//*** Obtención del angulo de la veleta ***//
////////////////////////////////////////////////////////////////////
float Veleta(){
    int h = 0;
    float angle;
    lectVel=mcp3208.readADC(1);
    for(h=0; h <= 15; h++) {
        if(lectVel >= sensorMin[h] && lectVel <= sensorMax[h]){
        angle = dirDeg[h];
        break;
        }
    }
    return angle;
}

//*** Lectura del voltaje de entrada del sistema ***//
////////////////////////////////////////////////////////////////////
float Voltaje(){
    float volt;
    lectVolt=mcp3208.readADC(4);
    if(lectVolt == 0){
    Serial.println("\n
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
    Serial.println("Falló en el ADC");
    uint8_t error = 10;
```

```
    while(1){
      BT_VOC.printf("%i\n",error);
      Serial.printf("\nNúmero de error = %i,10\n",error);
      delay(1000);
    }
  }
  volt=(((lectVolt*5.0)/4095.0))/0.340599;
  return volt;
}
```

Listing A.24: Funciones para obtener datos del ADC.

6. Biblioteca para el módulo de termopar

6.1. Variables, bibliotecas y objetos utilizados

```
//Biblioteca para obtener lecturas del termopar
//Address_1 0x67
//Address_2 0x60

#include <Wire.h>
#include <Adafruit_I2CDevice.h>
#include <Adafruit_I2CRegister.h>
#include "Adafruit_MCP9600.h"

#define I2C_ADDRESS_1 (0x67)
#define I2C_ADDRESS_2 (0x60) //Pin ADDR to GND

Adafruit_MCP9600 mcp_1;
Adafruit_MCP9600 mcp_2;
```

Listing A.25: Variables usadas para los módulos del termopares.

6.2. Funciones para el uso y obtención de datos

```
//*** Inicialización del modulo del termopar ***//
////////////////////////////////////////////////////////////////
void MCP9600_Init(){
  //Sensor_1
  if (! mcp_1.begin(I2C_ADDRESS_1)){
    Serial.println("\n
    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
    Serial.println("Módulo termopar_1 falló en inicialización, por
    favor verifica la conexión");
    uint8_t error = 4;
    while(1){
      BT_VOC.printf("%i,4\n",error);
      Serial.printf("\nNúmero de error = %i\n",error);
      delay(1000);
    }
```

```
29    }
30
31    mcp_1.setADCresolution(MCP9600_ADCRESOLUTION_18);
32    mcp_1.setThermocoupleType(MCP9600_TYPE_K);
33    mcp_1.setFilterCoefficient(3);
34    mcp_1.enable(true);
35    Serial.println("-----------------------------------------------");
36    Serial.println("¡Módulo termopar_1 inicializado correctamente!");
37    Serial.println("-----------------------------------------------");
38
39    //Sensor_2
40    if (! mcp_2.begin(I2C_ADDRESS_2)){
41      Serial.println("\n
      !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
42      Serial.println("Módulo termopar_2 falló en inicialización, por
      favor verifica la conexión");
43      uint8_t error = 5;
44      //print_Error(error);
45      while(1){
46        BT_VOC.printf("%i,5\n",error);
47        Serial.printf("\nNúmero de error = %i\n",error);
48        delay(1000);
49      }
50    }
51
52    mcp_2.setADCresolution(MCP9600_ADCRESOLUTION_18);
53    mcp_2.setThermocoupleType(MCP9600_TYPE_K);
54    mcp_2.setFilterCoefficient(3);
55    mcp_2.enable(true);
56    Serial.println("-----------------------------------------------");
57    Serial.println("¡Módulo termopar_2 inicializado correctamente!");
58    Serial.println("-----------------------------------------------");
59  }
60
61  //*** Lectura del dato de temperatura del termopar_1 ***//
62  ////////////////////////////////////////////////////////////////////
63  float MCP9600_1_ReadT(){
64    float t = mcp_1.readThermocouple();
65    if(isnan(t) == true){
66      ContrBomb = false;
67      ContrOFF = false;
68      ContrOFFBool = false;
69
70      ContrValv = false;
71      valvBool = false;
72
73      PararBom();
74      digitalWrite(electroValA, LOW);
75      digitalWrite(electroValB, LOW);
76      digitalWrite (vent, LOW);
77
78      Serial.println(" ");
79      Serial.println("Prueba detenida");
80      Serial.println(" ");
```

```

    Serial.println("\n !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
    Serial.println("Módulo termopar_1 falló en lectura, por favor verifica la conexión");
    uint8_t error = 4;
    //print_Error(error);
    while(1){
      BT_VOC.printf("%i,5\n",error);
      Serial.printf("\nNúmero de error = %i\n",error);
      delay(1000);
    }
  }
  return t;
}

//*** Lectura del dato de temperatura ambiente_1 ***//
//////////////////////////////////////////////////////////////////////
float MCP9600_1_ReadTAmb(){
  float t = mcp_1.readAmbient();
  if(isnan(t) == true){
    ContrBomb = false;
    ContrOFF = false;
    ContrOFFBool = false;

    ContrValv = false;
    valvBool = false;

    PararBom();
    digitalWrite(electroValA, LOW);
    digitalWrite(electroValB, LOW);
    digitalWrite (vent, LOW);

    Serial.println(" ");
    Serial.println("Prueba detenida");
    Serial.println(" ");

    Serial.println("\n !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
    Serial.println("Módulo termopar_1 falló en lectura, por favor verifica la conexión");
    uint8_t error = 4;
    //print_Error(error);
    while(1){
      BT_VOC.printf("%i,5\n",error);
      Serial.printf("\nNúmero de error = %i\n",error);
      delay(1000);
    }
  }
  return t;
}

//*** Lectura del dato de temperatura del termopar_2 ***//
//////////////////////////////////////////////////////////////////////
```

```
float MCP9600_2_ReadT(){
  float t = mcp_2.readThermocouple();
  if(isnan(t) == true){
    ContrBomb = false;
    ContrOFF = false;
    ContrOFFBool = false;

    ContrValv = false;
    valvBool = false;

    PararBom();
    digitalWrite(electroValA, LOW);
    digitalWrite(electroValB, LOW);
    digitalWrite (vent, LOW);

    Serial.println(" ");
    Serial.println("Prueba detenida");
    Serial.println(" ");

    Serial.println("\n
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
    Serial.println("Módulo termopar_2 falló en lectura, por favor
   verifica la conexión");
    uint8_t error = 5;
    //print_Error(error);
    while(1){
      BT_VOC.printf("%i,5\n",error);
      Serial.printf("\nNúmero de error = %i\n",error);
      delay(1000);
    }
  }
  return t;
}

//*** Lectura del dato de temperatura ambiente_2 ***//
///////////////////////////////////////////////////////////////////
float MCP9600_2_ReadTAmb(){
  float t = mcp_2.readAmbient();
  if(isnan(t) == true){
    ContrBomb = false;
    ContrOFF = false;
    ContrOFFBool = false;

    ContrValv = false;
    valvBool = false;

    PararBom();
    digitalWrite(electroValA, LOW);
    digitalWrite(electroValB, LOW);
    digitalWrite (vent, LOW);

    Serial.println(" ");
    Serial.println("Prueba detenida");
    Serial.println(" ");
```

```

    Serial.println("\n!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
    Serial.println("Módulo termopar_2 falló en lectura, por favor verifica la conexión");
    uint8_t error = 5;
    //print_Error(error);
    while(1){
      BT_VOC.printf("%i,5\n",error);
      Serial.printf("\nNúmero de error = %i\n",error);
      delay(1000);
    }
  }
  return t;
}
```

Listing A.26: Funciones para obtener datos de los módulos para termopar.

7. Biblioteca para módulo micro SD

7.1. Variables y bibliotecas utilizadas

```
//Biblioteca para el control del modulo para la microSD

#include "FS.h"
#include "SD.h"
#include "SPI.h"

#define SCK  14
#define MISO  16
#define MOSI  13
#define CS  15

SPIClass spi = SPIClass(HSPI);

char buffer[]=" ";
            //"año-dia-mes hora:minuto:segundo,temperatura,humedad,temperatura T1,temperatura T2,ICA,flujo,dir viento,vel viento,pm1,pm2.5,pm4,pm10,voc,nox,voltaje"
char* msj30="%i-%i-%i %i:%i:%i,%.2f,%.2f,%.2f,%.2f,%i,%i,%.1f,%.2f,%.1f,%.1f,%.1f,%.1f,%.1f,%.1f,%.2f\n";
            //"año-dia-mes hora:minuto:segundo,temperatura,tempMax,tempMin,humedad,temperatura T1,temperatura T2,ICA,flujo,dir viento,vel viento,vel vientoMax,pm1,pm2.5,pm4,pm10,voc,nox,voltaje"
char* msj15="%i-%i-%i %i:%i:%i,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.1f,%.2f,%.1f,%.1f,%.1f,%.1f,%.1f,%.1f,%.2f\n";

char Doc[30];
char* fechaValv =" - %i/%i/%i - %i:%i hrs\n";
const char* nomDoc = "/Datos";
```

```
const char* nomValv = "/Valvulas";
const char* ext30 = "_30s.csv";
const char* ext15 = "_15min.csv";
const char* ext = ".csv";

int anioON, mesON, diaON, horaON, minON, mesOFF, diaOFF, horaOFF,
    minOFF;
```

Listing A.27: Bibliotecas, variables y objetos utilizados para el módulo SD.

7.2. Funciones para manipular los archivos

```
//*** Lectura de un archivo ***//
////////////////////////////////////////////////////////////////////
void readFile(fs::FS &fs, const char * path){
    //Serial.printf("Reading file: %s\n", path);

    File file = fs.open(path);
    if(!file){
        Serial.println("Failed to open file for reading");
        return;
    }

    Serial.print("Read from file: ");
    while(file.available()){
        Serial.write(file.read());
    }
    file.close();
}

//*** Escribir en un archivo ***//
////////////////////////////////////////////////////////////////////
void writeFile(fs::FS &fs, const char * path, const char * message){
    Serial.printf("Writing file: %s\n", path);

    File file = fs.open(path, FILE_WRITE);
    if(!file){
      Serial.println("\n
    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
      Serial.println("Modulo SD falló en agregar datos, por favor
    verifica la conexión");
      uint8_t error = 8;
      //print_Error(error);
      while(1){
        BT_VOC.printf("%i,8\n",error);
        Serial.printf("\nNúmero de error = %i\n",error);
        delay(1000);
      }
      //Serial.println("Failed to open file for writing");
      //return;
    }
    if(file.print(message)){
        //Serial.println("File written");
```

```
    } else {
        //Serial.println("Write failed");
    }
    file.close();
}

//*** Agregar en un archivo ***//
////////////////////////////////////////////////////////////////////
void appendFile(fs::FS &fs, const char * path, const char * message)
   {
    Serial.printf("Appending to file: %s\n", path);

    File file = fs.open(path, FILE_APPEND);
    if(!file){
      Serial.println("\n
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
      Serial.println("Modulo SD falló en agregar datos, por favor
   verifica la conexión");
      uint8_t error = 8;
      //print_Error(error);
      while(1){
        BT_VOC.printf("%i,8\n",error);
        Serial.printf("\nNúmero de error = %i\n",error);
        delay(1000);
      }
      //Serial.println("Failed to open file for appending");
      //return;
    }
    if(file.print(message)){
        //Serial.println("Message appended");
    } else {
        //Serial.println("Append failed");
    }
    file.close();
}

//*** Eliminar un archivo ***//
////////////////////////////////////////////////////////////////////
void deleteFile(fs::FS &fs, const char * path){
    Serial.printf("Deleting file: %s\n", path);
    if(fs.remove(path)){
        Serial.println("File deleted");
    } else {
        Serial.println("Delete failed");
    }
}
```

Listing A.28: Funciones proporcionadas por los ejemplos del IDE para uso de un módulo SD.

7.3. Funciones SD_Init() y documento()

```
//*** Inicializar módulo SD ***//
/////////////////////////////////////////////////////////////////////
void SD_Init(){
  spi.begin(SCK, MISO, MOSI, CS);

  if(!SD.begin(CS,spi,80000000)){
    Serial.println("\n !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
    Serial.println("Modulo SD falló en inicialización, por favor verifica la conexión");
    uint8_t error = 8;
    //print_Error(error);
    while(1){
      BT_VOC.printf("%i,8\n",error);
      Serial.printf("\nNúmero de error = %i\n",error);
      delay(1000);
    }
  }
  uint8_t cardType = SD.cardType();

  if(cardType == CARD_NONE){
    Serial.println("\n !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
    Serial.println("Memoria SD no encontrada");
    uint8_t error = 9;
    //print_Error(error);
    while(1){
      BT_VOC.printf("%i,9\n",error);
      Serial.printf("\nNúmero de error = %i\n",error);
      delay(1000);
    }
  }
  Serial.println(" ----------------------------------------------------");
  Serial.println("¡Modulo SD y Tarjeta SD inicializado correctamente!");
  Serial.println(" ----------------------------------------------------");
}

//*** Asignar nombre deseado a un documento ***//
/////////////////////////////////////////////////////////////////////
void documento(int dia, int mes, int anio, const char* nom, const char* ext, char sal[30]){
  char* Date="_%i-%i-%i";
  memset(sal, 0, sizeof(sal));
  sprintf(buffer,Date,anio,mes,dia);  //"_año-mes-dia"
  strcpy(sal, nom);                   //"/Datos" o "/Valvulas"
  strcat(sal, buffer);                //"/Datos_año-mes-dia" o "/Valvulas_año-mes-dia"
  strcat(sal, ext);                   //"/Datos_año-mes-dia_30s.csv" o "/Datos_año-mes-dia_15m.csv" o "/Valvulas_año-mes-dia.csv"
```

```
}
```

Listing A.29: Funciones creadas para inicializar y crear/llamar documentos con el modulo SD.

8. Biblioteca para el la pantalla OLED

8.1. Variables, bibliotecas y objetos utilizados

```
//Biblioteca para el control del la pantalla oled y el joystick
//Address 0x3D

#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include "Bomba.hpp"
#include "Reloj_RTC.hpp"

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

#define OLED_RESET -1 //4 // Reset pin # (or -1 if sharing Arduino reset pin)
#define SCREEN_ADDRESS 0x3D ///< See datasheet for Address; 0x3D for 128x64, 0x3C for 128x32
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

#define NUMFLAKES     10 // Number of snowflakes in the animation example

#define LOGO_HEIGHT   16
#define LOGO_WIDTH    16

int dirX = 0, dirY = 0;

//RTC
bool ContrBombBool = true, ContrOFFBool = false, valvBool = true, ContrBomb = false, ContrOFF = false, ContrValv = false, bola_wf= false, bola_prueba=false;  // variable de control con valor verdadero
String Cadena, Dato, Dato1, Dato2, Dato3, Dato4, Dato5;

int horaValv, horaM, minutoValv, a, b, FR_An=0, cuentaSL=0;
float flujAct;
float corriente;
char var;

//Datos de sensores
```

```
35 float TMax=0.0,TMin=0.0,T=0.0,H=0.0,acumT=0.0,acumH=0.0,acumF=0.0,
       windSpd=0.0, PM1p0=0.0, PM2p5=0.0, PM4p0=0.0, PM10p0=0.0, voc
       =0.0, nox=0.0, windSpdmax=0.0;
36 float TT_1=0.0,TTA_1=0.0,TT_2=0.0,TTA_2=0.0,VB=0.0,acumTT_1=0.0,
       acumTTA_1=0.0,acumTT_2=0.0,acumTTA_2=0.0,acumVB=0.0,acumCA=0.0,
       DirVel=0.0,acumWSpd=0.0,acum1p0=0.0,acum2p5=0.0,acum4p0=0.0,
       acum10p0=0.0,acumvoc=0.0,acumnox=0.0;
37 uint16_t CA;
38 int contP=0,contF=0,F;
39 float promF;
40 int indSen,indCos;
41
42 int x=23,y=1,id=0,im=0,ih=0,imn=0,ifo=0,xm=7,ym=8;
43 double Setpoint;
44
45 String dia[32], mes[13], hora[25], minuto[61], flujol[11];
46 bool sel=true, sel1=true,sel2=true, click1 = false, menu=true,datos=
       false,datos2=false,datos3=false,datos4=false,datos5=false,datos6=
       false, manual=false, sleepp=false, estbot=false, bolam=false,
       bolah=false, bolad=false, bolames=false, bolaf=false, bolav1=
       false, bolav2=false;
47 String Cadenaa, Datoo;
48 char varr;
49 static const unsigned char PROGMEM escudofi_negro [] = {
50   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
       0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
51   0x00, 0x00, 0x0f, 0x80, 0x15, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18,
       0x00, 0x0f, 0xc0, 0x00, 0x00,
52   0x00, 0x00, 0x08, 0x00, 0x1d, 0xb0, 0x00, 0x00, 0x00, 0x00, 0xfc,
       0x00, 0x3f, 0x00, 0x00, 0x00,
53   0x01, 0xe0, 0x3d, 0xc0, 0x1f, 0xc0, 0xf8, 0x00, 0x07, 0xf8, 0x05,
       0xc0, 0x2e, 0x01, 0xfc, 0x00,
54   0x0c, 0x38, 0x01, 0xc0, 0x3f, 0x01, 0xc2, 0x00, 0x1d, 0xf0, 0x02,
       0xc0, 0x5e, 0x00, 0xf3, 0x00,
55   0x1b, 0xb0, 0x0e, 0xe0, 0x6b, 0x00, 0xd9, 0x80, 0x3e, 0x78, 0x0b,
       0x60, 0x7a, 0x01, 0xed, 0x80,
56   0x37, 0xdc, 0x01, 0x30, 0x1d, 0x07, 0xf7, 0x80, 0x27, 0xe0, 0x00,
       0x00, 0x00, 0x00, 0x5e, 0xc0,
57   0x2d, 0xef, 0xff, 0x07, 0xc0, 0x00, 0xde, 0xc0, 0x3f, 0x7f, 0xfc,
       0x07, 0xf0, 0x00, 0x5e, 0xc0,
58   0x3d, 0xff, 0xf9, 0xc7, 0xce, 0x01, 0x7e, 0xc0, 0x3d, 0xfb, 0xf1,
       0x87, 0xce, 0x01, 0x7b, 0x40,
59   0x2d, 0x5f, 0xe1, 0xc7, 0xcf, 0x07, 0x4b, 0x40, 0x2d, 0x5d, 0xe1,
       0x87, 0xcf, 0x07, 0x2b, 0x40,
60   0x29, 0x7f, 0xc1, 0x80, 0xcf, 0x87, 0xa9, 0x40, 0x3d, 0x5e, 0xe0,
       0x00, 0x3f, 0x8f, 0xfb, 0xc0,
61   0x3d, 0x7f, 0xc0, 0x00, 0xff, 0x8e, 0xeb, 0xc0, 0x3d, 0x7f, 0x80,
       0xe6, 0xdf, 0xff, 0xeb, 0x40,
62   0x3d, 0x77, 0x8f, 0x70, 0xff, 0xff, 0x6b, 0x40, 0x3d, 0x77, 0x80,
       0x70, 0xe0, 0x3f, 0x6a, 0xc0,
63   0x3d, 0x77, 0xa0, 0x70, 0xe6, 0xbf, 0xea, 0xc0, 0x3d, 0x47, 0xbf,
       0x70, 0xe0, 0x7f, 0xea, 0xc0,
64   0x3d, 0x43, 0xff, 0xb0, 0x40, 0x39, 0x6a, 0x80, 0x35, 0xcf, 0x7f,
       0xe0, 0x00, 0x1e, 0x5a, 0x80,
```

```
  0x35, 0x07, 0x5f, 0xe0, 0x00, 0x7c, 0x1a, 0x80, 0x36, 0x07, 0xff,
   0xf7, 0x00, 0x7a, 0x1f, 0x80,
  0x34, 0x03, 0x2f, 0x47, 0xb1, 0xcc, 0x47, 0x80, 0x34, 0x06, 0xa4,
   0x07, 0xb9, 0x4e, 0xc5, 0x00,
  0x14, 0x7a, 0x28, 0x09, 0xfd, 0x43, 0xc5, 0x00, 0x18, 0x1c, 0x40,
   0x07, 0xfd, 0x43, 0xe5, 0x00,
  0x18, 0x74, 0x51, 0x07, 0xe8, 0x27, 0x03, 0x00, 0x10, 0x04, 0x50,
   0xcf, 0x10, 0xa4, 0x03, 0x00,
  0x10, 0x06, 0x52, 0xbc, 0x35, 0xa0, 0x02, 0x00, 0x00, 0x04, 0x56,
   0xfc, 0x14, 0xa0, 0x02, 0x00,
  0x08, 0x00, 0xd6, 0x9c, 0x54, 0xa0, 0x00, 0xc0, 0x01, 0x00, 0xd6,
   0xbc, 0x54, 0xb0, 0x25, 0x40,
  0x01, 0xe8, 0x96, 0xac, 0xd4, 0x90, 0x01, 0x00, 0x30, 0x28, 0x16,
   0xbc, 0xd4, 0x80, 0x06, 0x80,
  0x00, 0x40, 0x00, 0x0e, 0x00, 0x00, 0x08, 0x00, 0x0c, 0x73, 0x80,
   0x02, 0x00, 0x01, 0xa0, 0x80,
  0x11, 0x0c, 0x22, 0x84, 0x00, 0x02, 0x21, 0x00, 0x00, 0xd8, 0x18,
   0x00, 0x03, 0x84, 0x26, 0x00,
  0x00, 0x90, 0x00, 0x40, 0x04, 0x44, 0xa0, 0x00, 0x00, 0xa0, 0x08,
   0x10, 0xe0, 0x48, 0x40, 0x00,
  0x08, 0xc4, 0x06, 0x04, 0x08, 0x41, 0x80, 0x00, 0x02, 0x88, 0x81,
   0x0b, 0x88, 0x52, 0x00, 0x00,
  0x01, 0x04, 0x01, 0x40, 0x28, 0x90, 0x00, 0x00, 0x00, 0x02, 0x60,
   0x30, 0x51, 0x42, 0x3c, 0x00,
  0x01, 0x01, 0x04, 0x90, 0x4a, 0x82, 0x00, 0x00, 0x01, 0x00, 0x81,
   0x03, 0x06, 0x04, 0x02, 0x00,
  0x02, 0xc0, 0x00, 0x11, 0x08, 0x07, 0x04, 0x00, 0x00, 0x1c, 0x91,
   0x99, 0x6e, 0x48, 0xc8, 0x00,
  0x06, 0x00, 0x54, 0x26, 0x70, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x1f, 0xd0, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x61, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0xa3, 0x10, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0xc0, 0x00, 0x00, 0x00
};
```

Listing A.30: Variables usadas para la oled.

8.2. Funciones para escribir/manipular la pantalla oled.

8.2.1. Función de Inicialización y apagado automático

```
//*** Apagado automatico Minutos ***//
////////////////////////////////////////////////////////////////
void OffAutomatico(){
  horaOFF = horaON + TPruebaHor;
  if(horaOFF >= 24){
    horaOFF = horaOFF-24;
  }
  minOFF = minON + TPruebaMin;
  if(minOFF >= 60){
    minOFF = minOFF - 60;
  }
  if(minON == 0){
```

```
96      horaOFF--;
97    }
98  }
99
100 void OLED_Init() {
101   dia[0]="Dia";dia[1]="01 ";dia[2]="02 ";dia[3]="03 ";dia[4]="04 ";
      dia[5]="05 ";dia[6]="06 ";dia[7]="07 ";dia[8]="08 ";dia[9]="09 ";
      dia[10]="10 ";dia[11]="11 ";dia[12]="12 ";dia[13]="13 ";
102   dia[14]="14 ";dia[15]="15 ";dia[16]="16 ";dia[17]="17 ";dia[18]="
      18 ";dia[19]="19 ";dia[20]="20 ";dia[21]="21 ";dia[22]="22 ";dia
      [23]="23 ";dia[24]="24 ";dia[25]="25 ";dia[26]="26 ";
103   dia[27]="27 ";dia[28]="28 ";dia[29]="29 ";dia[30]="30 ";dia[31]="
      31 ";
104   mes[0]="Mes";mes[1]="01 ";mes[2]="02 ";mes[3]="03 ";mes[4]="04 ";
      mes[5]="05 ";mes[6]="06 ";mes[7]="07 ";mes[8]="08 ";mes[9]="09 ";
      mes[10]="10 ";mes[11]="11 ";mes[12]="12 ";
105   hora[0]="Hr ";hora[1]="01 ";hora[2]="02 ";hora[3]="03 ";hora[4]="
      04 ";hora[5]="05 ";hora[6]="06 ";hora[7]="07 ";hora[8]="08 ";hora
      [9]="09 ";hora[10]="10 ";hora[11]="11 ";hora[12]="12 ";
106   hora[13]="13 ";hora[14]="14 ";hora[15]="15 ";hora[16]="16 ";hora
      [17]="17 ";hora[18]="18 ";hora[19]="19 ";hora[20]="20 ";hora[21]=
      "21 ";hora[22]="22 ";hora[23]="23 ";hora[24]="00 ";
107   minuto[0]="Min";minuto[1]="01 ";minuto[2]="02 ";minuto[3]="03 ";
      minuto[4]="04 ";minuto[5]="05 ";minuto[6]="06 ";minuto[7]="07 ";
      minuto[8]="08 ";minuto[9]="09 ";minuto[10]="10 ";
108   minuto[11]="11 ";minuto[12]="12 ";minuto[13]="13 ";minuto[14]="14
      ";minuto[15]="15 ";minuto[16]="16 ";minuto[17]="17 ";minuto[18]="
      18 ";minuto[19]="19 ";minuto[20]="20 ";minuto[21]="21 ";
109   minuto[22]="22 ";minuto[23]="23 ";minuto[24]="24 ";minuto[25]="25
      ";minuto[26]="26 ";minuto[27]="27 ";minuto[28]="28 ";minuto[29]="
      29 ";minuto[30]="30 ";minuto[31]="31 ";
110   minuto[32]="32 ";minuto[33]="33 ";minuto[34]="34 ";minuto[35]="35
      ";minuto[36]="36 ";minuto[37]="37 ";minuto[38]="38 ";minuto[39]="
      39 ";minuto[40]="40 ";minuto[41]="41 ";
111   minuto[42]="42 ";minuto[43]="43 ";minuto[44]="44 ";minuto[45]="45
      ";minuto[46]="46 ";minuto[47]="47 ";minuto[48]="48 ";minuto[49]="
      49 ";minuto[50]="50 ";minuto[51]="51 ";
112   minuto[52]="52 ";minuto[53]="53 ";minuto[54]="54 ";minuto[55]="55
      ";minuto[56]="56 ";minuto[57]="57 ";minuto[58]="58 ";minuto[59]="
      59 ";minuto[60]="00 ";
113   flujol[0]="25";flujol[1]="50";flujol[2]="75";flujol[3]="100";
      flujol[4]="125";flujol[5]="150";flujol[6]="175";flujol[7]="200";
      flujol[8]="225";flujol[9]="250";flujol[10]="275";
114
115
116   // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V
      internally
117   if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
118     Serial.println("\n
      !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
119     Serial.println("Oled falló en inicialización, por favor verifica
      la conexión");
120     uint8_t error = 1;
121     while(1){
```

```
      BT_VOC.printf("%i,1\n",error);
      Serial.printf("\nNúmero de error = %i\n",error);
      delay(1000);
    }
  }

  // Show initial display buffer contents on the screen --
  // the library initializes this with an Adafruit splash screen.
  display.clearDisplay();
  display.drawBitmap(32, 0, escudofi_negro, 59, 63, SSD1306_WHITE);
  display.display();
  delay(3000); // Pause for 2 seconds

  // Clear the buffer
  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(2);
  display.setCursor(22,18);
  display.print("Sistema");
  display.setCursor(46,37);
  display.println("VOC");
  display.display();
  delay(100);

  pinMode(36, INPUT);
  pinMode(39, INPUT);
  pinMode(0, INPUT_PULLUP);
  Serial.println("-----------------------------------------------");
  Serial.println("¡Oled inicializada correctamente!");
  Serial.println("-----------------------------------------------");
}
```

Listing A.31: Funciones inicializar y el apagado automático.

8.2.2. Funciones para manipular datos o moverse por la pantalla oled con el *joystick*.

```
void sumarDia(void){
    if(id==31){
      id=1;
    }else{
      id++;
    }
}
void sumarMes(void){
    if(im==12){
      im=1;
    }else{
      im++;
    }
}
void restarDia(void){
    if(id==1){
```

```
169       id=31;
170     }else{
171       id--;
172     }
173 }
174 void restarMes(void){
175     if(im==1){
176       im=12;
177     }else{
178       im--;
179     }
180 }
181
182 void sumarHr(void){
183     if(ih==24){
184       ih=1;
185     }else{
186       ih++;
187     }
188 }
189 void sumarMin(void){
190     if(imn==60){
191       imn=1;
192     }else{
193       imn++;
194     }
195 }
196 void restarHr(void){
197     if(ih==1){
198       ih=24;
199     }else{
200       ih--;
201     }
202 }
203 void restarMin(void){
204     if(imn==1){
205       imn=60;
206     }else{
207       imn--;
208     }
209 }
210 void sumarF(void){
211     if(ifo==10){
212       ifo=0;
213     }else{
214       ifo++;
215     }
216 }
217 void restarF(void){
218     if(ifo==0){
219       ifo=10;
220     }else{
221       ifo--;
222     }
```

```
223 }
224
225 void movDerecha(void){
226   if(menu==false){
227     if(manual==false){
228       if(x==23){
229       x=x+49;
230       }
231     }
232     if(manual==true){
233       if(x==23 && y==1){
234       x=x+49;
235       }
236     }
237     if(datos==true){
238       if(datos2==true){
239         datos3=true;
240         datos2=false;
241         datos4=false;
242         datos5=false;
243         datos6=false;
244         delay(100);
245       }else if(datos3==true){
246         datos3=false;
247         datos2=false;
248         datos4=true;
249         datos5=false;
250         datos6=false;
251         delay(100);
252       }else if(datos4==true){
253         datos3=false;
254         datos2=false;
255         datos4=false;
256         datos5=true;
257         datos6=false;
258         delay(100);
259       }else if(datos5==true){
260         datos3=false;
261         datos2=false;
262         datos4=false;
263         datos5=false;
264         datos6=true;
265         delay(100);
266       }else if(datos6==true){
267         delay(100);
268       }else{
269         datos2=true;
270         delay(100);
271       }
272     }
273   }
274 }
275
276 void movIzquierda(void){
```

```
277   if(menu==false){
278     if(manual==false){
279       if(x==72){
280       x=x-49;
281       }
282     }
283     if(manual==true){
284       if(x==72 && y==1){
285       x=x-49;
286       }
287     }
288     if(datos==true){
289       if (datos3==true){
290         datos3=false;
291         datos2=true;
292         datos4=false;
293         datos5=false;
294         datos6=false;
295         delay(100);
296       }else if(datos4==true){
297         datos3=true;
298         datos2=false;
299         datos4=false;
300         datos5=false;
301         datos6=false;
302         delay(100);
303       }else if(datos5==true){
304         datos3=false;
305         datos2=false;
306         datos4=true;
307         datos5=false;
308         datos6=false;
309         delay(100);
310       }else if(datos6==true){
311         datos3=false;
312         datos2=false;
313         datos4=false;
314         datos5=true;
315         datos6=false;
316         delay(100);
317       }else{
318         datos2=false;
319         delay(100);
320       }
321     }
322   }
323 }
324
325 void movAbajo(void){
326   if(menu==true){
327     if(ym==31||ym==8){
328       ym=ym+23;
329     }
330   }else if(manual==false){
```

```
331     if(x==23&&y==1&&sel==false){
332       restarDia();
333     }
334     if(x==72&&y==1&&sel==false){
335       restarMes();
336     }
337     if(x==23&&y==21&&sel==false){
338       restarHr();
339     }
340     if(x==72&&y==21&&sel==false){
341       restarMin();
342     }
343       if(y==21&&sel==true){
344        y=y+19;
345        //x=49;
346       }
347       if(y==1&&sel==true){
348       y=y+20;
349       }
350   }else if(manual==true){
351     if(x==72&&y==21&&sel==false){
352       restarF();
353     }
354     if(y==21&&sel==true){
355       y=y+19;
356       x=49;
357     }
358     if(y==1){
359        y=y+20;
360        x=72;
361     }
362   }
363 }
364
365 void movArriba(void){
366   if(menu==true){
367   if(ym==54||ym==31){
368     ym=ym-23;
369   }
370   }else if(manual==false){
371     if(x==23&&y==1&&sel==false){
372       sumarDia();
373     }
374     if(x==72&&y==1&&sel==false){
375       sumarMes();
376     }
377     if(x==23&&y==21&&sel==false){
378       sumarHr();
379     }
380     if(x==72&&y==21&&sel==false){
381       sumarMin();
382     }
383       if(y==21&&sel==true){
384       y=y-20;
```

```
385       }if(y==40&&sel==true){
386         y=21;
387         //x=23;
388       }
389   }else if(manual==true){
390     if(x==72&&y==21&&sel==false){
391       sumarF();
392     }
393     if(y==21&&sel==true){
394       y=y-20;
395       x=23;
396     }
397     if(y==40){
398        y=21;
399        x=72;
400     }
401   }
402 }
403 
404 void selecc(void){
405   if(menu==true){
406     if(ym==8){
407       menu=false;
408       manual=true;
409       datos=false;
410       x=23;
411       y=1;
412     }else if(ym==31){
413       datos=false;
414       manual=false;
415       menu=false;
416       x=23;
417       y=1;
418     }else{
419       datos=true;
420       manual=false;
421       menu=false;
422     }
423   }else if(datos==false&&manual==false){
424     if(x==23&&y==1&&sel==true){
425       id=1;
426       bolad=false;
427     }
428     if(x==72&&y==1&&sel==true){
429       im=1;
430       bolames=false;
431     }
432     if(x==23&&y==21&&sel==true){
433       ih=1;
434       bolah=false;
435     }
436     if(x==72&&y==21&&sel==true){
437       imn=1;
438       bolam=false;
```

```
439     }
440
441     if(x==23&&y==1&&sel==false){
442       bolad=true;
443     }
444     if(x==72&&y==1&&sel==false){
445       bolames=true;
446     }
447     if(x==23&&y==21&&sel==false){
448       bolah=true;
449     }
450     if(x==72&&y==21&&sel==false){
451       bolam=true;
452     }
453
454     if(x==72&&y==40&&sel==true){
455       if(im>0&&imn>0&&id>0&&ih>0){
456         diaON=dia[id].toInt();
457         mesON=mes[im].toInt();
458         horaON=hora[ih].toInt();
459         minON=minuto[imn].toInt();
460         Serial.print("Prueba establecida el (dia/mes): "); Serial.
    print(diaON);Serial.print("/");Serial.println(mesON);
461         Serial.print("Hora: "); Serial.print(horaON); Serial.print("
    :"); Serial.print(minON); Serial.println(" hrs");
462         horaValv = horaON; minutoValv = minON;
463         Serial.print("Flujo a: "); Serial.print(Setpoint); Serial.
    println(" L/min");
464         Serial.println(" ");
465
466         a=0;
467         b=12;
468         ContrBombBool = true;
469         ContrBomb = true;
470
471         sel=false;
472         menu=true;
473       }else{
474         sel=false;
475       }
476     }else if(x==23&&y==40&&sel==true){
477         sel=false;
478         menu=true;
479       }
480      if(sel==true){
481         sel=false;
482       }else{
483         sel=true;
484       }
485     }else if(manual==true){
486           if(x==23 && y==1 && sel1==false){
487             bolav1=false;
488             sel1=true;
489           }else if(x==23 && y==1 && sel1==true){
```

```
490             bolav1=true;
491             sel1=false;
492           }
493           if(x==72 && y==1 && sel2==false){
494             bolav2=false;
495             sel2=true;
496           }else if(x==72 && y==1 && sel2==true){
497             bolav2=true;
498             sel2=false;
499           }
500           if(x==72 && y==21 && sel==true){
501             ifo=0;
502             bolaf=false;
503             sel=false;
504           }else if(x==72 && y==21 && sel==false){
505             bolaf=true;
506             sel=true;
507             Setpoint = flujol[ifo].toDouble();
508             Serial.print("Flujo establecido a: "); Serial.print(
    Setpoint); Serial.println(" mL/min");
509             Serial.println(" ");
510           }
511
512
513           if(x==49 && y==40){
514               sel=true;
515               sel1=true;
516               sel2=true;
517               menu=true;
518               manual=false;
519               bolav1=false;
520               bolav2=false;
521             }
522     }else if(datos==true){
523       menu=true;
524       datos=false;
525     }
526 }
527
528 int dirJS() {
529   int X=0, Y=0, But, direcjs=5;
530   /*X = mcp3208.readADC(2);; // read X axis value [0..1023]
531   Y = mcp3208.readADC(3);; // read Y axis value [0..1023]*/
532   X = dirX;
533   Y = dirY;
534   But = digitalRead(0); // read Button state [0,1]
535   //Serial.print("X: ");Serial.println(X);
536   //Serial.print("Y: ");Serial.println(Y);
537   //if(X <= 2000 && X>= 1000 && Y <= 4095 && Y >= 3100){ //arrba
538   if(Y <= 4095 && Y >= 3100){ //arrba
539     //Serial.print("0");
540     direcjs=0;
541     sleepp=false;
542     cuentaSL=0;
```

```
543   //}else  if(X <= 2000 && X>= 1000 && Y <= 800 && Y >= 0){ //abajo
544   }else  if(Y <= 800 && Y >= 0){ //abajo
545     //Serial.print("1");
546     direcjs=1;
547     sleepp=false;
548     cuentaSL=0;
549   //}else if(X <= 4095 && X>= 3100 && Y <= 2500 && Y >= 1800){//
       derecha
550   }else if(X <= 4095 && X>= 3100){//derecha
551     //Serial.print("2");
552     direcjs=2;
553     sleepp=false;
554     cuentaSL=0;
555   //}else if(X <= 800 && X>= 0 && Y <= 1400 && Y >= 900){
556   }else if(X <= 800 && X>= 0){
557     //Serial.print("3");
558     direcjs=3;
559     sleepp=false;
560     cuentaSL=0;
561   }else if(But==0){
562       click1 = true;
563       if(estbot==false){
564        estbot=true;
565       //Serial.print("4");
566       direcjs=4;
567       sleepp=false;
568       cuentaSL=0;
569       delay(10);
570       }
571    }else if(But==1){
572       click1 = false;
573       direcjs=5;
574       estbot=false;
575     }
576
577   delay(190);
578   return direcjs;
579 }
580
581 void OLED_MENU(){
582   if (sleepp == true){
583     display.clearDisplay();
584     display.display();
585   }else{
586     if(menu==true){
587
588       paginMenu();
589     }else if(datos==true){
590       if(datos2==true){
591         paginDatos2();
592       }else if(datos3==true){
593         paginDatos3();
594       }else if(datos4==true){
595         paginDatos4();
```

```
596       }else if(datos5==true){
597         paginDatos5();
598       }else if(datos6==true){
599         paginDatos6();
600       }else{
601         paginDatos1();
602       }
603     }else if(manual==true){
604       paginaMan();
605     }else{
606       paginaOpc();
607     }
608   }
609
610     varr = dirJS();
611
612     switch (varr) {
613       case 2:
614         if(sel==true){
615         movDerecha();
616         }
617         break;
618      case 3:
619      if(sel==true){
620       movIzquierda();
621      }
622       break;
623
624      case 0:
625       movArriba();
626       break;
627
628      case 1:
629       movAbajo();
630       break;
631
632      case 4:
633       selecc();
634       break;
635     }
636 }
```

Listing A.32: Funciones para cambiar datos de inicio de prueba en pantalla

8.2.3. Funciones para el despliegue de datos

```
403 void paginaMan(void){
404   display.clearDisplay();
405   display.setTextColor(SSD1306_WHITE);
406   display.drawRoundRect(x, y, 40, 22, 5, SSD1306_WHITE);
407   if(bolav1==true){
408     display.drawRoundRect(14, 9, 7, 7, 45, SSD1306_WHITE);
409   }
```

```
410   if(bolav2==true){
411     display.drawRoundRect(113, 9, 7, 7, 45, SSD1306_WHITE);
412   }
413   if(bolaf==true){
414     display.drawRoundRect(113, 30, 7, 7, 45, SSD1306_WHITE);
415   }
416   display.setTextSize(2);
417   display.setCursor(27,6);             // Start at top-left corner
418   display.println("V1  V2");
419   display.setTextSize(2);
420   display.setCursor(27,27);             // Start at top-left corner
421   display.println("F:  "+flujol[ifo]);
422   display.setCursor(51,44);             // Start at top-left corner
423   display.println("<-");
424   display.display();
425 }
426
427 void paginaOpc(void){
428   display.clearDisplay();
429   display.setTextColor(SSD1306_WHITE);
430   display.drawRoundRect(x, y, 40, 22, 5, SSD1306_WHITE);
431   if(bolad==true){
432     display.drawRoundRect(14, 9, 7, 7, 45, SSD1306_WHITE);
433   }
434   if(bolames==true){
435     display.drawRoundRect(113, 9, 7, 7, 45, SSD1306_WHITE);
436   }
437   if(bolam==true){
438     display.drawRoundRect(113, 30, 7, 7, 45, SSD1306_WHITE);
439   }
440   if(bolah==true){
441     display.drawRoundRect(14, 30, 7, 7, 45, SSD1306_WHITE);
442   }
443   display.setTextSize(2);
444   display.setCursor(27,6);             // Start at top-left corner
445   display.println(dia[id]+"/"+mes[im]);
446   display.setTextSize(2);
447   display.setCursor(27,27);             // Start at top-left corner
448   display.println(hora[ih]+":"+minuto[imn]);
449   display.setCursor(27,44);             // Start at top-left corner
450   display.println("<-  OK");
451   display.display();
452 }
453
454 void wifi(){
455   display.drawPixel(36, 58, SSD1306_WHITE);
456   display.drawPixel(37, 57, SSD1306_WHITE);
457   display.drawPixel(38, 56, SSD1306_WHITE);
458   display.drawPixel(39, 56, SSD1306_WHITE);
459   display.drawPixel(40, 56, SSD1306_WHITE);
460   display.drawPixel(41, 56, SSD1306_WHITE);
461   display.drawPixel(42, 56, SSD1306_WHITE);
462   display.drawPixel(43, 56, SSD1306_WHITE);
463   display.drawPixel(44, 56, SSD1306_WHITE);
```

```
464   display.drawPixel(45, 57, SSD1306_WHITE);
465   display.drawPixel(46, 58, SSD1306_WHITE);
466   display.drawPixel(38, 59, SSD1306_WHITE);
467   display.drawPixel(39, 58, SSD1306_WHITE);
468   display.drawPixel(40, 58, SSD1306_WHITE);
469   display.drawPixel(41, 58, SSD1306_WHITE);
470   display.drawPixel(42, 58, SSD1306_WHITE);
471   display.drawPixel(43, 58, SSD1306_WHITE);
472   display.drawPixel(44, 59, SSD1306_WHITE);
473   display.drawPixel(40, 60, SSD1306_WHITE);
474   display.drawPixel(41, 60, SSD1306_WHITE);
475   display.drawPixel(42, 60, SSD1306_WHITE);
476   display.drawPixel(41, 62, SSD1306_WHITE);
477 }
478
479 void prueba(){
480   display.drawPixel(55, 59, SSD1306_WHITE);
481   display.drawPixel(55, 60, SSD1306_WHITE);
482   display.drawPixel(56, 60, SSD1306_WHITE);
483   display.drawPixel(56, 61, SSD1306_WHITE);
484   display.drawPixel(57, 61, SSD1306_WHITE);
485   display.drawPixel(57, 62, SSD1306_WHITE);
486   display.drawPixel(58, 60, SSD1306_WHITE);
487   display.drawPixel(58, 61, SSD1306_WHITE);
488   display.drawPixel(59, 60, SSD1306_WHITE);
489   display.drawPixel(59, 59, SSD1306_WHITE);
490   display.drawPixel(60, 59, SSD1306_WHITE);
491   display.drawPixel(60, 58, SSD1306_WHITE);
492   display.drawPixel(61, 58, SSD1306_WHITE);
493   display.drawPixel(61, 57, SSD1306_WHITE);
494   display.drawPixel(62, 57, SSD1306_WHITE);
495   display.drawPixel(62, 56, SSD1306_WHITE);
496 }
497
498 void paginMenu(){
499   display.clearDisplay();
500   display.setTextColor(SSD1306_WHITE);
501   display.drawRoundRect(xm, ym, 7, 7, 45, SSD1306_WHITE);
502   display.setTextSize(2);
503   display.setCursor(20,4);             // Start at top-left corner
504   display.println("MANUAL");
505   display.setCursor(20,27);             // Start at top-left corner
506   display.println("CONFIG.");
507   display.setCursor(20,50);             // Start at top-left corner
508   display.println("DATOS");
509   display.display();
510 }
511
512 void paginDatos1(){
513   DateTime fecha = rtc.now();
514   display.clearDisplay();
515   display.setTextColor(SSD1306_WHITE);
516   if(bola_wf==true){
517     wifi();
```

```
518   }
519   if(bola_prueba==true){
520     prueba();
521   }
522   display.setTextSize(2);
523   display.setCursor(0,0);             // Start at top-left corner
524   display.print("Temp=");
525   display.print(T,1);
526   display.println("C");
527   display.print("Hum=");
528   display.print(H);
529   display.println("%");
530   display.setTextSize(1);
531   display.println("");
532   display.println("");
533   display.print("\n");
534   display.print(fecha.hour());
535   display.print(":");
536   display.print(fecha.minute());
537   display.print("              1/6");
538   display.display();
539 }
540 void paginDatos2(){
541   DateTime fecha = rtc.now();
542   display.clearDisplay();
543   display.setTextColor(SSD1306_WHITE);
544   if(bola_wf==true){
545     wifi();
546   }
547   if(bola_prueba==true){
548     prueba();
549   }
550   display.setTextSize(2);
551   display.setCursor(0,0);             // Start at top-left corner
552   display.print("TT1=");
553   display.print(TT_1,1);
554   display.println("C");
555   display.print("TT2=");
556   display.print(TT_2,1);
557   display.println("C");
558   display.setTextSize(1);
559   display.println("");
560   display.println("");
561   display.print("\n");
562   display.print(fecha.hour());
563   display.print(":");
564   display.print(fecha.minute());
565   display.print("              2/6");
566   display.display();
567 }
568 void paginDatos3(){
569   DateTime fecha = rtc.now();
570   display.clearDisplay();
571   display.setTextColor(SSD1306_WHITE);
```

```
572   if(bola_wf==true){
573     wifi();
574   }
575   if(bola_prueba==true){
576     prueba();
577   }
578   display.setTextSize(2);
579   display.setCursor(0,0);             // Start at top-left corner
580   display.print("ICA=");
581   display.println(CA);
582   display.print("F=");
583   display.print(F);
584   display.println("mL/m");
585   display.print("VBat=");
586   display.print(VB,1);
587   display.println("V");
588   display.setTextSize(1);
589   display.print("\n");
590   display.print(fecha.hour());
591   display.print(":");
592   display.print(fecha.minute());
593   display.print("              3/6");
594   display.display();
595 }
596 void paginDatos4(){
597   DateTime fecha = rtc.now();
598   display.clearDisplay();
599   display.setTextColor(SSD1306_WHITE);
600   if(bola_wf==true){
601     wifi();
602   }
603   if(bola_prueba==true){
604     prueba();
605   }
606   display.setTextSize(2);
607   display.setCursor(0,0);             // Start at top-left corner
608   display.print("DV=");
609   display.print(DirVel,1);
610   display.println("G");
611   display.print("VV=");
612   display.print(windSpd,1);
613   display.println("m/s");
614   display.print(" ");
615   display.setTextSize(1);
616   display.println("");
617   display.println("\n");
618   display.print(fecha.hour());
619   display.print(":");
620   display.print(fecha.minute());
621   display.print("              4/6");
622   display.display();
623 }
624
625 void paginDatos5(){
```

```
626   DateTime fecha = rtc.now();
627   display.clearDisplay();
628   display.setTextColor(SSD1306_WHITE);
629   if(bola_wf==true){
630     wifi();
631   }
632   if(bola_prueba==true){
633     prueba();
634   }
635   display.setTextSize(2);
636   display.setCursor(0,0);             // Start at top-left corner
637   display.print("PM");
638   display.setTextSize(1);
639   display.print("1.0 ");
640   display.setTextSize(2);
641   display.print(PM1p0,1);
642   display.setTextSize(1);
643   display.print("ug/m3");
644   display.setTextSize(2);
645   display.print("\nPM");
646   display.setTextSize(1);
647   display.print("2.5 ");
648   display.setTextSize(2);
649   display.print(PM2p5,1);
650   display.setTextSize(1);
651   display.print("ug/m3");
652   display.setTextSize(2);
653   display.print("\nPM");
654   display.setTextSize(1);
655   display.print("4.0 ");
656   display.setTextSize(2);
657   display.print(PM4p0,1);
658   display.setTextSize(1);
659   display.println("ug/m3");
660   display.println("\n");
661   display.print(fecha.hour());
662   display.print(":");
663   display.print(fecha.minute());
664   display.print("              5/6");
665   display.display();
666 }
667
668 void paginDatos6(){
669   DateTime fecha = rtc.now();
670   display.clearDisplay();
671   display.setTextColor(SSD1306_WHITE);
672   if(bola_wf==true){
673     wifi();
674   }
675   if(bola_prueba==true){
676     prueba();
677   }
678   display.setTextSize(2);
679   display.setCursor(0,0);             // Start at top-left corner
```

```
680   display.print("PM");
681   display.setTextSize(1);
682   display.print("10  ");
683   display.setTextSize(2);
684   display.print(PM10p0,1);
685   display.setTextSize(1);
686   display.print("ug/m3");
687   display.setTextSize(2);
688   display.print("\nVOC=");
689   display.println(voc,1);
690   display.print("NOx=");
691   display.println(nox,1);
692   display.setTextSize(1);
693   display.print("\n");
694   display.print(fecha.hour());
695   display.print(":");
696   display.print(fecha.minute());
697   display.print("             6/6");
698   display.display();
699 }
```

Listing A.33: Funciones para desplegar datos en pantalla

9. Biblioteca para el reloj en tiempo real

9.1. Variables, bibliotecas y objetos utilizados

```
1 //Biblioteca para el control del reloj en tiempo real
2 //Address 0x68
3
4 #include <Wire.h>   //Incluye libreria para interfaz I2C
5 #include <RTClib.h>   // incluye libreria para el manejo del modulo
    RTC
6
7 RTC_DS3231 rtc;     // crea objeto del tipo RTC_DS3231
```

Listing A.34: Variables, bibliotecas y objetos utilizados para el reloj en tiempo real.

9.2. Funciones para el uso y obtención de datos

```
9  //*** Inicialización del modulo del reloj en tiempo real ***//
10 ////////////////////////////////////////////////////////////////
11 void Reloj_Init(){
12   if (! rtc.begin()){
13     Serial.println("\n
     !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
14     Serial.println("Modulo de reloj RTC no encontrado");  // muestra
       mensaje de error
15     uint8_t error = 2;
16     //print_Error(error);
```

```
    while(1){
      BT_VOC.printf("%i,2\n",error);
      Serial.printf("\nNúmero de error = %i\n",error);
      delay(1000);
    }
  }
  Serial.println("-----------------------------------------------");
  Serial.println("¡Reloj RTC inicializado correctamente!");
  Serial.println("-----------------------------------------------");
}
```

Listing A.35: Funcion para inicializar el reloj en tiempo real.

10. Biblioteca para el sensor de COVs y PM´s

10.1. Variables, bibliotecas y objetos utilizados

```
//Biblioteca para obtener lecturas del sensor de VOC
//Address 0x69

#include <SensirionI2CSen5x.h>
#include <Wire.h>
float sen5T[6];
SensirionI2CSen5x sen5x;

bool sen55_error = true;
```

Listing A.36: Variables, bibliotecas y objetos utilizados para el SEN55.

10.2. Funciones para el uso y obtención de datos

```
//*** Inicialización del SEN55 ***//
////////////////////////////////////////////////////////////////
void SEN55_Init() {
    while (!Serial) {
        delay(100);
    }

    Wire.begin();
    sen5x.begin(Wire);
    uint16_t err;
    char errorMessage[256];
    err = sen5x.deviceReset();
    if (err) {
        errorToString(err, errorMessage, 256);
    }

    err = sen5x.startMeasurement();
    if(err){
```

```
      //Serial.print("Error trying to execute startMeasurement(): ");
      errorToString(err, errorMessage, 256);
      //Serial.println(errorMessage);
      Serial.println("\n!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
      Serial.println("Sen55 falló en inicialización, por favor verifica la conexión");
      uint8_t error = 7;
      //print_Error(error);
      while(1){
        BT_VOC.printf("%i,7\n",error);
        Serial.printf("\nNúmero de error = %i\n",error);
        delay(1000);
      }
    }
    Serial.println("----------------------------------------------");
    Serial.println("¡SEN55 inicializado correctamente!");
    Serial.println("----------------------------------------------");
}

//*** Lectura de datos de SEN55 ***//
////////////////////////////////////////////////////////////////////
void LectSEN55(){
    uint16_t err;
    char errorMessage[256];

    // Read Measurement
    float massConcentrationPm1p0;
    float massConcentrationPm2p5;
    float massConcentrationPm4p0;
    float massConcentrationPm10p0;
    float ambientHumidity;
    float ambientTemperature;
    float vocIndex;
    float noxIndex;

    err = sen5x.readMeasuredValues(
        massConcentrationPm1p0, massConcentrationPm2p5, massConcentrationPm4p0,
        massConcentrationPm10p0, ambientHumidity, ambientTemperature, vocIndex,
        noxIndex);

    if (err){
      ContrBomb = false;
      ContrOFF = false;
      ContrOFFBool = false;

      ContrValv = false;
      valvBool = false;

```

```
      PararBom();
      digitalWrite(electroValA, LOW);
      digitalWrite(electroValB, LOW);
      digitalWrite (vent, LOW);

      Serial.println(" ");
      Serial.println("Prueba detenida");
      Serial.println(" ");

      errorToString(err, errorMessage, 256);
      //Serial.println(errorMessage);
      Serial.println("\n
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
      Serial.println("Sen55 falló en lectura, por favor verifica la
conexión");
      uint8_t error = 7;
      //print_Error(error);
      while(1){
        BT_VOC.printf("%i,7\n",error);
        Serial.printf("\nNúmero de error = %i\n",error);
        delay(1000);
      }
    }
    sen5T[0]= massConcentrationPm1p0;
    sen5T[1]= massConcentrationPm2p5;
    sen5T[2]= massConcentrationPm4p0;
    sen5T[3]= massConcentrationPm10p0;
    sen5T[4]= vocIndex;
    sen5T[5]= noxIndex;
}
```

Listing A.37: Función para inicializar y leer datos del SEN55.

11. Biblioteca para el sensor de calidad del aire

11.1. Variables, bibliotecas y objetos utilizados

```
//Biblioteca para obtener lectura del sensor de calidad del aire
// Address 0x59

#include <DFRobot_SGP40.h>
#include <Wire.h>
DFRobot_SGP40    mySgp40(&Wire);
```

Listing A.38: bibliotecas y objetos utilizados para el SGP40.

11.2. Funciones para el uso y obtención de datos

```
//*** Inicialización del sensor de calidad del aire ***//
////////////////////////////////////////////////////////////////////
```

```
void SGP40_Init(){
  while(mySgp40.begin(/*duration = */10000) !=true){
    Serial.println("\n!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
    Serial.println("sgp40 falló en inicialización, por favor verifica la conexión");
    uint8_t error = 6;
    //print_Error(error);
    while(1){
      BT_VOC.printf("%i,6\n",error);
      Serial.printf("\nNúmero de error = %i\n",error);
      delay(1000);
    }
  }
  Serial.println("-----------------------------------------------");
  Serial.println("¡sgp40 inicializado correctamente!");
  Serial.println("-----------------------------------------------");
}

//*** Lectura del dato de calidad del aire ***//
////////////////////////////////////////////////////////////////////
uint16_t Read_SGP40(){
  uint16_t index = mySgp40.getVoclndex();
  return index;
}
```

Listing A.39: Función para inicializar y leer datos del SGP40.

12. Biblioteca para el termohigrómetro

12.1. Variables, bibliotecas y objetos utilizados

```
//Biblioteca para obtener lecturas del sensor de humedad y temperatura
// Address 0x44

#include <Wire.h>   //Incluye libreria para interfaz I2C
#include "Adafruit_SHT31.h"
Adafruit_SHT31 sht30 = Adafruit_SHT31();
```

Listing A.40: bibliotecas y objetos utilizados para el SHT30.

12.2. Funciones para el uso y obtención de datos

```
//*** Inicialización del sensor de humedad y temperatura ***//
////////////////////////////////////////////////////////////////////
void SHT30_Init(){
  if (! sht30.begin(0x44)) {   // Set to 0x45 for alternate i2c addr
    Serial.println("\n!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
```

```
    Serial.println("Higrómetro falló en inicialización, por favor verifica la conexión");
    uint8_t error = 3;
    //print_Error(error);
    while(1){
      BT_VOC.printf("%i,3\n",error);
      Serial.printf("\nNúmero de error = %i\n",error);
      delay(1000);
    }
  }
  Serial.println("----------------------------------------------");
  Serial.println("¡Higrómetro inicializado correctamente!");
  Serial.println("----------------------------------------------");
}

//*** Lectura del dato de temperatura ***//
////////////////////////////////////////////////////////////////////
float SHT30_ReadT(){
  float t = sht30.readTemperature();
  if(isnan(t) == true){
    ContrBomb = false;
    ContrOFF = false;
    ContrOFFBool = false;

    ContrValv = false;
    valvBool = false;

    PararBom();
    digitalWrite(electroValA, LOW);
    digitalWrite(electroValB, LOW);
    digitalWrite (vent, LOW);

    Serial.println(" ");
    Serial.println("Prueba detenida");
    Serial.println(" ");

    Serial.println("\n!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
    Serial.println("Higrómetro falló en lectura, por favor verifica la conexión");
    uint8_t error = 3;
    //print_Error(error);
    while(1){
      BT_VOC.printf("%i,5\n",error);
      Serial.printf("\nNúmero de error = %i\n",error);
      delay(1000);
    }
  }
  return t;
}

//*** Lectura del dato de humedad ***//
////////////////////////////////////////////////////////////////////
float SHT30_ReadH(){
```

```
  float h = sht30.readHumidity();
  if(isnan(h) == true){
    ContrBomb = false;
    ContrOFF = false;
    ContrOFFBool = false;

    ContrValv = false;
    valvBool = false;

    PararBom();
    digitalWrite(electroValA, LOW);
    digitalWrite(electroValB, LOW);
    digitalWrite (vent, LOW);

    Serial.println(" ");
    Serial.println("Prueba detenida");
    Serial.println(" ");

    Serial.println("\n
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
    Serial.println("Higrómetro falló en lectura, por favor verifica
   la conexión");
    uint8_t error = 3;
    //print_Error(error);
    while(1){
      BT_VOC.printf("%i,5\n",error);
      Serial.printf("\nNúmero de error = %i\n",error);
      delay(1000);
    }
  }
  return h;
}
```

Listing A.41: Funciones para inicializar y leer datos del SHT30.

B Ejemplos de uso

1. Código de ejemplo MCP9600

```
#include <Wire.h>
#include <Adafruit_I2CDevice.h>
#include <Adafruit_I2CRegister.h>
#include "Adafruit_MCP9600.h"

#define I2C_ADDRESS (0x67)

Adafruit_MCP9600 mcp;

void setup()
{
    Serial.begin(115200);
    while (!Serial) {
      delay(10);
    }
    Serial.println("MCP9600 HW test");

    /* Initialise the driver with I2C_ADDRESS and the default I2C
    bus. */
    if (! mcp.begin(I2C_ADDRESS)) {
        Serial.println("Sensor not found. Check wiring!");
        while (1);
    }

  Serial.println("Found MCP9600!");

  mcp.setADCresolution(MCP9600_ADCRESOLUTION_18);
  Serial.print("ADC resolution set to ");
  switch (mcp.getADCresolution()) {
    case MCP9600_ADCRESOLUTION_18:   Serial.print("18"); break;
    case MCP9600_ADCRESOLUTION_16:   Serial.print("16"); break;
    case MCP9600_ADCRESOLUTION_14:   Serial.print("14"); break;
    case MCP9600_ADCRESOLUTION_12:   Serial.print("12"); break;
  }
  Serial.println(" bits");

  mcp.setThermocoupleType(MCP9600_TYPE_K);
  Serial.print("Thermocouple type set to ");
  switch (mcp.getThermocoupleType()) {
```

```
    case MCP9600_TYPE_K:  Serial.print("K"); break;
    case MCP9600_TYPE_J:  Serial.print("J"); break;
    case MCP9600_TYPE_T:  Serial.print("T"); break;
    case MCP9600_TYPE_N:  Serial.print("N"); break;
    case MCP9600_TYPE_S:  Serial.print("S"); break;
    case MCP9600_TYPE_E:  Serial.print("E"); break;
    case MCP9600_TYPE_B:  Serial.print("B"); break;
    case MCP9600_TYPE_R:  Serial.print("R"); break;
  }
  Serial.println(" type");

  mcp.setFilterCoefficient(3);
  Serial.print("Filter coefficient value set to: ");
  Serial.println(mcp.getFilterCoefficient());

  mcp.setAlertTemperature(1, 30);
  Serial.print("Alert #1 temperature set to ");
  Serial.println(mcp.getAlertTemperature(1));
  mcp.configureAlert(1, true, true);  // alert 1 enabled, rising
   temp

  mcp.enable(true);

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

void loop()
{
  Serial.print("Hot Junction: "); Serial.println(mcp.
   readThermocouple());
  Serial.print("Cold Junction: "); Serial.println(mcp.readAmbient())
   ;
  Serial.print("ADC: "); Serial.print(mcp.readADC() * 2); Serial.
   println(" uV");
  delay(1000);
}
```

2. Código de ejemplo SHT30

```
/***************************************************
  This is an example for the SHT31-D Humidity & Temp Sensor

  Designed specifically to work with the SHT31-D sensor from
   Adafruit
  ----> https://www.adafruit.com/products/2857

  These sensors use I2C to communicate, 2 pins are required to
  interface
 ****************************************************/

#include <Arduino.h>
```

```
#include <Wire.h>
#include "Adafruit_SHT31.h"

bool enableHeater = false;
uint8_t loopCnt = 0;

Adafruit_SHT31 sht31 = Adafruit_SHT31();

void setup() {
  Serial.begin(9600);

  while (!Serial)
    delay(10);     // will pause Zero, Leonardo, etc until serial
    console opens

  Serial.println("SHT31 test");
  if (! sht31.begin(0x44)) {   // Set to 0x45 for alternate i2c addr
    Serial.println("Couldn't find SHT31");
    while (1) delay(1);
  }

  Serial.print("Heater Enabled State: ");
  if (sht31.isHeaterEnabled())
    Serial.println("ENABLED");
  else
    Serial.println("DISABLED");
}


void loop() {
  float t = sht31.readTemperature();
  float h = sht31.readHumidity();

  if (! isnan(t)) {  // check if 'is not a number'
    Serial.print("Temp *C = "); Serial.print(t); Serial.print("\t\t"
    );
  } else {
    Serial.println("Failed to read temperature");
  }

  if (! isnan(h)) {  // check if 'is not a number'
    Serial.print("Hum. % = "); Serial.println(h);
  } else {
    Serial.println("Failed to read humidity");
  }

  delay(1000);

  // Toggle heater enabled state every 30 seconds
  // An ~3.0 degC temperature increase can be noted when heater is
    enabled
  if (loopCnt >= 30) {
    enableHeater = !enableHeater;
    sht31.heater(enableHeater);
```

```
    Serial.print("Heater Enabled State: ");
    if (sht31.isHeaterEnabled())
      Serial.println("ENABLED");
    else
      Serial.println("DISABLED");

    loopCnt = 0;
  }
  loopCnt++;
}
```

3. Código de ejemplo SEN0392

```

/*!
 * @file getVocIndex.ino
 * @brief Read the environmental VOC index.  Range: 0-500;
 * @n Experimental phenomena: read environmental VOC index once per
   second and print the value in serial port
 *
 * @copyright  Copyright (c) 2010 DFRobot Co.Ltd (http://www.dfrobot
   .com)
 * @licence     The MIT License (MIT)
 * @author [yangfeng]<feng.yang@dfrobot.com>
 * @version  V1.0
 * @date  2020-12-18
 * @get from https://www.dfrobot.com
 * @url  https://github.com/DFRobot/DFRobot_SGP40
 */
#include <DFRobot_SGP40.h>

/*
 * #include <Wire.h>
 * DFRobot_SGP40    mySgp40(&Wire);

 */

//#include <Wire.h>
//DFRobot_SGP40    mySgp40(&Wire);
DFRobot_SGP40    mySgp40;


void setup() {
  Serial.begin(115200);
  Serial.println("sgp40 is starting, the reading can be taken after
   10 seconds...");

  while(mySgp40.begin(/*duration = */10000) !=true){
    Serial.println("failed to init chip, please check if the chip
   connection is fine");
    delay(1000);
```

```
  }
  Serial.println("-----------------------------------------------");
  Serial.println("sgp40 initialized successfully!");
  Serial.println("-----------------------------------------------");

  //mySgp40.setRhT(/*relativeHumidity = */ 50, /*temperature = */
    20);

}

void loop() {
  /*
   *    0-100  no   need to ventilate,purify
   *    100-200  no   need to ventilate,purify
   *    200-400   ventilate   ,purify
   *    400-500   ventilate   ,purify intensely
   */
  uint16_t index = mySgp40.getVocIndex();

  Serial.print("vocIndex = ");
  Serial.println(index);
  delay(1000);
}
```

4. Código de ejemplo SEN55

```
*
 * I2C-Generator: 0.3.0
 * Yaml Version: 2.1.3
 * Template Version: 0.7.0-112-g190ecaa
 */
/*
 * Copyright (c) 2021, Sensirion AG
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or
   without
 * modification, are permitted provided that the following
   conditions are met:
 *
 * * Redistributions of source code must retain the above copyright
   notice, this
 *   list of conditions and the following disclaimer.
 *
 * * Redistributions in binary form must reproduce the above
   copyright notice,
 *   this list of conditions and the following disclaimer in the
   documentation
 *   and/or other materials provided with the distribution.
 *
 * * Neither the name of Sensirion AG nor the names of its
```

```
21  *   contributors may be used to endorse or promote products derived
        from
22  *   this software without specific prior written permission.
23  *
24  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
      CONTRIBUTORS "AS IS"
25  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
       TO, THE
26  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
      PARTICULAR PURPOSE
27  * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
      CONTRIBUTORS BE
28  * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY,
      OR
29  * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
       OF
30  * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
      BUSINESS
31  * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
      WHETHER IN
32  * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
      OTHERWISE)
33  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
      ADVISED OF THE
34  * POSSIBILITY OF SUCH DAMAGE.
35  */
36
37 #include <Arduino.h>
38 #include <SensirionI2CSen5x.h>
39 #include <Wire.h>
40
41 // The used commands use up to 48 bytes. On some Arduino's the
      default buffer
42 // space is not large enough
43 #define MAXBUF_REQUIREMENT 48
44
45 #if (defined(I2C_BUFFER_LENGTH) &&                 \
46      (I2C_BUFFER_LENGTH >= MAXBUF_REQUIREMENT)) || \
47     (defined(BUFFER_LENGTH) && BUFFER_LENGTH >= MAXBUF_REQUIREMENT)
48 #define USE_PRODUCT_INFO
49 #endif
50
51 SensirionI2CSen5x sen5x;
52
53 void printModuleVersions() {
54     uint16_t error;
55     char errorMessage[256];
56
57     unsigned char productName[32];
58     uint8_t productNameSize = 32;
59
60     error = sen5x.getProductName(productName, productNameSize);
61
62     if (error) {
```

```
63          Serial.print("Error trying to execute getProductName(): ");
64          errorToString(error, errorMessage, 256);
65          Serial.println(errorMessage);
66      } else {
67          Serial.print("ProductName:");
68          Serial.println((char*)productName);
69      }
70
71      uint8_t firmwareMajor;
72      uint8_t firmwareMinor;
73      bool firmwareDebug;
74      uint8_t hardwareMajor;
75      uint8_t hardwareMinor;
76      uint8_t protocolMajor;
77      uint8_t protocolMinor;
78
79      error = sen5x.getVersion(firmwareMajor, firmwareMinor,
    firmwareDebug,
80                               hardwareMajor, hardwareMinor,
    protocolMajor,
81                               protocolMinor);
82      if (error) {
83          Serial.print("Error trying to execute getVersion(): ");
84          errorToString(error, errorMessage, 256);
85          Serial.println(errorMessage);
86      } else {
87          Serial.print("Firmware: ");
88          Serial.print(firmwareMajor);
89          Serial.print(".");
90          Serial.print(firmwareMinor);
91          Serial.print(", ");
92
93          Serial.print("Hardware: ");
94          Serial.print(hardwareMajor);
95          Serial.print(".");
96          Serial.println(hardwareMinor);
97      }
98  }
99
100 void printSerialNumber() {
101     uint16_t error;
102     char errorMessage[256];
103     unsigned char serialNumber[32];
104     uint8_t serialNumberSize = 32;
105
106     error = sen5x.getSerialNumber(serialNumber, serialNumberSize);
107     if (error) {
108         Serial.print("Error trying to execute getSerialNumber(): ");
109         errorToString(error, errorMessage, 256);
110         Serial.println(errorMessage);
111     } else {
112         Serial.print("SerialNumber:");
113         Serial.println((char*)serialNumber);
114     }
```

```
115 }
116
117 void setup() {
118
119     Serial.begin(115200);
120     while (!Serial) {
121         delay(100);
122     }
123
124     Wire.begin();
125
126     sen5x.begin(Wire);
127
128     uint16_t error;
129     char errorMessage[256];
130     error = sen5x.deviceReset();
131     if (error) {
132         Serial.print("Error trying to execute deviceReset(): ");
133         errorToString(error, errorMessage, 256);
134         Serial.println(errorMessage);
135     }
136
137 // Print SEN55 module information if i2c buffers are large enough
138 #ifdef USE_PRODUCT_INFO
139     printSerialNumber();
140     printModuleVersions();
141 #endif
142
143     // set a temperature offset in degrees celsius
144     // Note: supported by SEN54 and SEN55 sensors
145     // By default, the temperature and humidity outputs from the
    sensor
146     // are compensated for the modules self-heating. If the module
    is
147     // designed into a device, the temperature compensation might
    need
148     // to be adapted to incorporate the change in thermal coupling
    and
149     // self-heating of other device components.
150     //
151     // A guide to achieve optimal performance, including references
152     // to mechanical design-in examples can be found in the app note
153     //   SEN5x       Temperature Compensation Instruction  at www.
    sensirion.com.
154     // Please refer to those application notes for further
    information
155     // on the advanced compensation settings used
156     // in ‘setTemperatureOffsetParameters‘, ‘setWarmStartParameter‘
    and
157     // ‘setRhtAccelerationMode‘.
158     //
159     // Adjust tempOffset to account for additional temperature
    offsets
160     // exceeding the SEN module’s self heating.
```

```
161     float tempOffset = 0.0;
162     error = sen5x.setTemperatureOffsetSimple(tempOffset);
163     if (error) {
164         Serial.print("Error trying to execute
    setTemperatureOffsetSimple(): ");
165         errorToString(error, errorMessage, 256);
166         Serial.println(errorMessage);
167     } else {
168         Serial.print("Temperature Offset set to ");
169         Serial.print(tempOffset);
170         Serial.println(" deg. Celsius (SEN54/SEN55 only");
171     }
172 
173     // Start Measurement
174     error = sen5x.startMeasurement();
175     if (error) {
176         Serial.print("Error trying to execute startMeasurement(): ")
    ;
177         errorToString(error, errorMessage, 256);
178         Serial.println(errorMessage);
179     }
180 }
181 
182 void loop() {
183     uint16_t error;
184     char errorMessage[256];
185 
186     delay(1000);
187 
188     // Read Measurement
189     float massConcentrationPm1p0;
190     float massConcentrationPm2p5;
191     float massConcentrationPm4p0;
192     float massConcentrationPm10p0;
193     float ambientHumidity;
194     float ambientTemperature;
195     float vocIndex;
196     float noxIndex;
197 
198     error = sen5x.readMeasuredValues(
199         massConcentrationPm1p0, massConcentrationPm2p5,
    massConcentrationPm4p0,
200         massConcentrationPm10p0, ambientHumidity, ambientTemperature
    , vocIndex,
201         noxIndex);
202 
203     if (error) {
204         Serial.print("Error trying to execute readMeasuredValues():
    ");
205         errorToString(error, errorMessage, 256);
206         Serial.println(errorMessage);
207     } else {
208         Serial.print("MassConcentrationPm1p0:");
209         Serial.print(massConcentrationPm1p0);
```

```
        Serial.print("\t");
        Serial.print("MassConcentrationPm2p5:");
        Serial.print(massConcentrationPm2p5);
        Serial.print("\t");
        Serial.print("MassConcentrationPm4p0:");
        Serial.print(massConcentrationPm4p0);
        Serial.print("\t");
        Serial.print("MassConcentrationPm10p0:");
        Serial.print(massConcentrationPm10p0);
        Serial.print("\t");
        Serial.print("AmbientHumidity:");
        if (isnan(ambientHumidity)) {
            Serial.print("n/a");
        } else {
            Serial.print(ambientHumidity);
        }
        Serial.print("\t");
        Serial.print("AmbientTemperature:");
        if (isnan(ambientTemperature)) {
            Serial.print("n/a");
        } else {
            Serial.print(ambientTemperature);
        }
        Serial.print("\t");
        Serial.print("VocIndex:");
        if (isnan(vocIndex)) {
            Serial.print("n/a");
        } else {
            Serial.print(vocIndex);
        }
        Serial.print("\t");
        Serial.print("NoxIndex:");
        if (isnan(noxIndex)) {
            Serial.println("n/a");
        } else {
            Serial.println(noxIndex);
        }
    }
}
```

5. Código de ejemplo OLED

```
/*************************************************************
 This is an example for our Monochrome OLEDs based on SSD1306
   drivers

 Pick one up today in the adafruit shop!
 ------> http://www.adafruit.com/category/63_98

 This example is for a 128x64 pixel display using I2C to communicate
 3 pins are required to interface (two I2C and one reset).
```

```
9
10  Adafruit invests time and resources providing this open
11  source code, please support Adafruit and open-source
12  hardware by purchasing products from Adafruit!
13
14  Written by Limor Fried/Ladyada for Adafruit Industries,
15  with contributions from the open source community.
16  BSD license, check license.txt for more information
17  All text above, and the splash screen below must be
18  included in any redistribution.
19  *********************************************************************/
20
21 #include <SPI.h>
22 #include <Wire.h>
23 #include <Adafruit_GFX.h>
24 #include <Adafruit_SSD1306.h>
25
26 #define SCREEN_WIDTH 128 // OLED display width, in pixels
27 #define SCREEN_HEIGHT 64 // OLED display height, in pixels
28
29 // Declaration for an SSD1306 display connected to I2C (SDA, SCL
       pins)
30 // The pins for I2C are defined by the Wire-library.
31 // On an arduino UNO:       A4(SDA), A5(SCL)
32 // On an arduino MEGA 2560: 20(SDA), 21(SCL)
33 // On an arduino LEONARDO:   2(SDA),  3(SCL), ...
34 #define OLED_RESET     -1 // Reset pin # (or -1 if sharing Arduino
       reset pin)
35 #define SCREEN_ADDRESS 0x3D ///< See datasheet for Address; 0x3D for
        128x64, 0x3C for 128x32
36 Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
       OLED_RESET);
37
38 #define NUMFLAKES     10 // Number of snowflakes in the animation
       example
39
40 #define LOGO_HEIGHT   16
41 #define LOGO_WIDTH    16
42 static const unsigned char PROGMEM logo_bmp[] =
43 { 0b00000000, 0b11000000,
44   0b00000001, 0b11000000,
45   0b00000001, 0b11000000,
46   0b00000011, 0b11100000,
47   0b11110011, 0b11100000,
48   0b11111110, 0b11111000,
49   0b01111110, 0b11111111,
50   0b00110011, 0b10011111,
51   0b00011111, 0b11111100,
52   0b00001101, 0b01110000,
53   0b00011011, 0b10100000,
54   0b00111111, 0b11100000,
55   0b00111111, 0b11110000,
56   0b01111100, 0b11110000,
57   0b01110000, 0b01110000,
```

```
  0b00000000, 0b00110000 };

void setup() {
  Serial.begin(9600);

  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V
   internally
  if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }

  // Show initial display buffer contents on the screen --
  // the library initializes this with an Adafruit splash screen.
  display.display();
  delay(2000); // Pause for 2 seconds

  // Clear the buffer
  display.clearDisplay();

  // Draw a single pixel in white
  display.drawPixel(10, 10, SSD1306_WHITE);

  // Show the display buffer on the screen. You MUST call display()
   after
  // drawing commands to make them visible on screen!
  display.display();
  delay(2000);
  // display.display() is NOT necessary after every single drawing
   command,
  // unless that's what you want...rather, you can batch up a bunch
   of
  // drawing operations and then update the screen all at once by
   calling
  // display.display(). These examples demonstrate both approaches
   ...

  testdrawline();      // Draw many lines

  testdrawrect();      // Draw rectangles (outlines)

  testfillrect();      // Draw rectangles (filled)

  testdrawcircle();    // Draw circles (outlines)

  testfillcircle();    // Draw circles (filled)

  testdrawroundrect(); // Draw rounded rectangles (outlines)

  testfillroundrect(); // Draw rounded rectangles (filled)

  testdrawtriangle();  // Draw triangles (outlines)

  testfilltriangle();  // Draw triangles (filled)
```

```
106
107   testdrawchar();      // Draw characters of the default font
108
109   testdrawstyles();    // Draw 'stylized' characters
110
111   testscrolltext();    // Draw scrolling text
112
113   testdrawbitmap();    // Draw a small bitmap image
114
115   // Invert and restore display, pausing in-between
116   display.invertDisplay(true);
117   delay(1000);
118   display.invertDisplay(false);
119   delay(1000);
120
121   testanimate(logo_bmp, LOGO_WIDTH, LOGO_HEIGHT); // Animate bitmaps
122 }
123
124 void loop() {
125 }
126
127 void testdrawline() {
128   int16_t i;
129
130   display.clearDisplay(); // Clear display buffer
131
132   for(i=0; i<display.width(); i+=4) {
133     display.drawLine(0, 0, i, display.height()-1, SSD1306_WHITE);
134     display.display(); // Update screen with each newly-drawn line
135     delay(1);
136   }
137   for(i=0; i<display.height(); i+=4) {
138     display.drawLine(0, 0, display.width()-1, i, SSD1306_WHITE);
139     display.display();
140     delay(1);
141   }
142   delay(250);
143
144   display.clearDisplay();
145
146   for(i=0; i<display.width(); i+=4) {
147     display.drawLine(0, display.height()-1, i, 0, SSD1306_WHITE);
148     display.display();
149     delay(1);
150   }
151   for(i=display.height()-1; i>=0; i-=4) {
152     display.drawLine(0, display.height()-1, display.width()-1, i,
      SSD1306_WHITE);
153     display.display();
154     delay(1);
155   }
156   delay(250);
157
158   display.clearDisplay();
```

```
159
160   for(i=display.width()-1; i>=0; i-=4) {
161     display.drawLine(display.width()-1, display.height()-1, i, 0,
      SSD1306_WHITE);
162     display.display();
163     delay(1);
164   }
165   for(i=display.height()-1; i>=0; i-=4) {
166     display.drawLine(display.width()-1, display.height()-1, 0, i,
      SSD1306_WHITE);
167     display.display();
168     delay(1);
169   }
170   delay(250);
171
172   display.clearDisplay();
173
174   for(i=0; i<display.height(); i+=4) {
175     display.drawLine(display.width()-1, 0, 0, i, SSD1306_WHITE);
176     display.display();
177     delay(1);
178   }
179   for(i=0; i<display.width(); i+=4) {
180     display.drawLine(display.width()-1, 0, i, display.height()-1,
      SSD1306_WHITE);
181     display.display();
182     delay(1);
183   }
184
185   delay(2000); // Pause for 2 seconds
186 }
187
188 void testdrawrect(void) {
189   display.clearDisplay();
190
191   for(int16_t i=0; i<display.height()/2; i+=2) {
192     display.drawRect(i, i, display.width()-2*i, display.height()-2*i
      , SSD1306_WHITE);
193     display.display(); // Update screen with each newly-drawn
      rectangle
194     delay(1);
195   }
196
197   delay(2000);
198 }
199
200 void testfillrect(void) {
201   display.clearDisplay();
202
203   for(int16_t i=0; i<display.height()/2; i+=3) {
204     // The INVERSE color is used so rectangles alternate white/black
205     display.fillRect(i, i, display.width()-i*2, display.height()-i
      *2, SSD1306_INVERSE);
206     display.display(); // Update screen with each newly-drawn
```

```
    rectangle
207     delay(1);
208   }
209
210   delay(2000);
211 }
212
213 void testdrawcircle(void) {
214   display.clearDisplay();
215
216   for(int16_t i=0; i<max(display.width(),display.height())/2; i+=2)
      {
217     display.drawCircle(display.width()/2, display.height()/2, i,
      SSD1306_WHITE);
218     display.display();
219     delay(1);
220   }
221
222   delay(2000);
223 }
224
225 void testfillcircle(void) {
226   display.clearDisplay();
227
228   for(int16_t i=max(display.width(),display.height())/2; i>0; i-=3)
      {
229     // The INVERSE color is used so circles alternate white/black
230     display.fillCircle(display.width() / 2, display.height() / 2, i,
      SSD1306_INVERSE);
231     display.display(); // Update screen with each newly-drawn circle
232     delay(1);
233   }
234
235   delay(2000);
236 }
237
238 void testdrawroundrect(void) {
239   display.clearDisplay();
240
241   for(int16_t i=0; i<display.height()/2-2; i+=2) {
242     display.drawRoundRect(i, i, display.width()-2*i, display.height
      ()-2*i,
243       display.height()/4, SSD1306_WHITE);
244     display.display();
245     delay(1);
246   }
247
248   delay(2000);
249 }
250
251 void testfillroundrect(void) {
252   display.clearDisplay();
253
254   for(int16_t i=0; i<display.height()/2-2; i+=2) {
```

```
255     // The INVERSE color is used so round-rects alternate white/
        black
256     display.fillRoundRect(i, i, display.width()-2*i, display.height
        ()-2*i,
257       display.height()/4, SSD1306_INVERSE);
258     display.display();
259     delay(1);
260   }
261
262   delay(2000);
263 }
264
265 void testdrawtriangle(void) {
266   display.clearDisplay();
267
268   for(int16_t i=0; i<max(display.width(),display.height())/2; i+=5)
        {
269     display.drawTriangle(
270       display.width()/2  , display.height()/2-i,
271       display.width()/2-i, display.height()/2+i,
272       display.width()/2+i, display.height()/2+i, SSD1306_WHITE);
273     display.display();
274     delay(1);
275   }
276
277   delay(2000);
278 }
279
280 void testfilltriangle(void) {
281   display.clearDisplay();
282
283   for(int16_t i=max(display.width(),display.height())/2; i>0; i-=5)
        {
284     // The INVERSE color is used so triangles alternate white/black
285     display.fillTriangle(
286       display.width()/2  , display.height()/2-i,
287       display.width()/2-i, display.height()/2+i,
288       display.width()/2+i, display.height()/2+i, SSD1306_INVERSE);
289     display.display();
290     delay(1);
291   }
292
293   delay(2000);
294 }
295
296 void testdrawchar(void) {
297   display.clearDisplay();
298
299   display.setTextSize(1);      // Normal 1:1 pixel scale
300   display.setTextColor(SSD1306_WHITE); // Draw white text
301   display.setCursor(0, 0);     // Start at top-left corner
302   display.cp437(true);         // Use full 256 char 'Code Page 437'
        font
303
```

```
304   // Not all the characters will fit on the display. This is normal.
305   // Library will draw what it can and the rest will be clipped.
306   for(int16_t i=0; i<256; i++) {
307     if(i == '\n') display.write(' ');
308     else          display.write(i);
309   }
310
311   display.display();
312   delay(2000);
313 }
314
315 void testdrawstyles(void) {
316   display.clearDisplay();
317
318   display.setTextSize(1);             // Normal 1:1 pixel scale
319   display.setTextColor(SSD1306_WHITE);        // Draw white text
320   display.setCursor(0,0);             // Start at top-left corner
321   display.println(F("Hello, world!"));
322
323   display.setTextColor(SSD1306_BLACK, SSD1306_WHITE); // Draw '
      inverse' text
324   display.println(3.141592);
325
326   display.setTextSize(2);             // Draw 2X-scale text
327   display.setTextColor(SSD1306_WHITE);
328   display.print(F("0x")); display.println(0xDEADBEEF, HEX);
329
330   display.display();
331   delay(2000);
332 }
333
334 void testscrolltext(void) {
335   display.clearDisplay();
336
337   display.setTextSize(2); // Draw 2X-scale text
338   display.setTextColor(SSD1306_WHITE);
339   display.setCursor(10, 0);
340   display.println(F("scroll"));
341   display.display();      // Show initial text
342   delay(100);
343
344   // Scroll in various directions, pausing in-between:
345   display.startscrollright(0x00, 0x0F);
346   delay(2000);
347   display.stopscroll();
348   delay(1000);
349   display.startscrollleft(0x00, 0x0F);
350   delay(2000);
351   display.stopscroll();
352   delay(1000);
353   display.startscrolldiagright(0x00, 0x07);
354   delay(2000);
355   display.startscrolldiagleft(0x00, 0x07);
356   delay(2000);
```

```
357   display.stopscroll();
358   delay(1000);
359 }
360
361 void testdrawbitmap(void) {
362   display.clearDisplay();
363
364   display.drawBitmap(
365     (display.width()  - LOGO_WIDTH ) / 2,
366     (display.height() - LOGO_HEIGHT) / 2,
367     logo_bmp, LOGO_WIDTH, LOGO_HEIGHT, 1);
368   display.display();
369   delay(1000);
370 }
371
372 #define XPOS   0 // Indexes into the 'icons' array in function below
373 #define YPOS   1
374 #define DELTAY 2
375
376 void testanimate(const uint8_t *bitmap, uint8_t w, uint8_t h) {
377   int8_t f, icons[NUMFLAKES][3];
378
379   // Initialize 'snowflake' positions
380   for(f=0; f< NUMFLAKES; f++) {
381     icons[f][XPOS]   = random(1 - LOGO_WIDTH, display.width());
382     icons[f][YPOS]   = -LOGO_HEIGHT;
383     icons[f][DELTAY] = random(1, 6);
384     Serial.print(F("x: "));
385     Serial.print(icons[f][XPOS], DEC);
386     Serial.print(F(" y: "));
387     Serial.print(icons[f][YPOS], DEC);
388     Serial.print(F(" dy: "));
389     Serial.println(icons[f][DELTAY], DEC);
390   }
391
392   for(;;) { // Loop forever...
393     display.clearDisplay(); // Clear the display buffer
394
395     // Draw each snowflake:
396     for(f=0; f< NUMFLAKES; f++) {
397       display.drawBitmap(icons[f][XPOS], icons[f][YPOS], bitmap, w,
    h, SSD1306_WHITE);
398     }
399
400     display.display(); // Show the display buffer on the screen
401     delay(200);        // Pause for 1/10 second
402
403     // Then update coordinates of each flake...
404     for(f=0; f< NUMFLAKES; f++) {
405       icons[f][YPOS] += icons[f][DELTAY];
406       // If snowflake is off the bottom of the screen...
407       if (icons[f][YPOS] >= display.height()) {
408         // Reinitialize to a random position, just off the top
409         icons[f][XPOS]   = random(1 - LOGO_WIDTH, display.width());
```

```
        icons[f][YPOS]   = -LOGO_HEIGHT;
        icons[f][DELTAY] = random(1, 6);
      }
    }
  }
}
```

6. Código de ejemplo BT

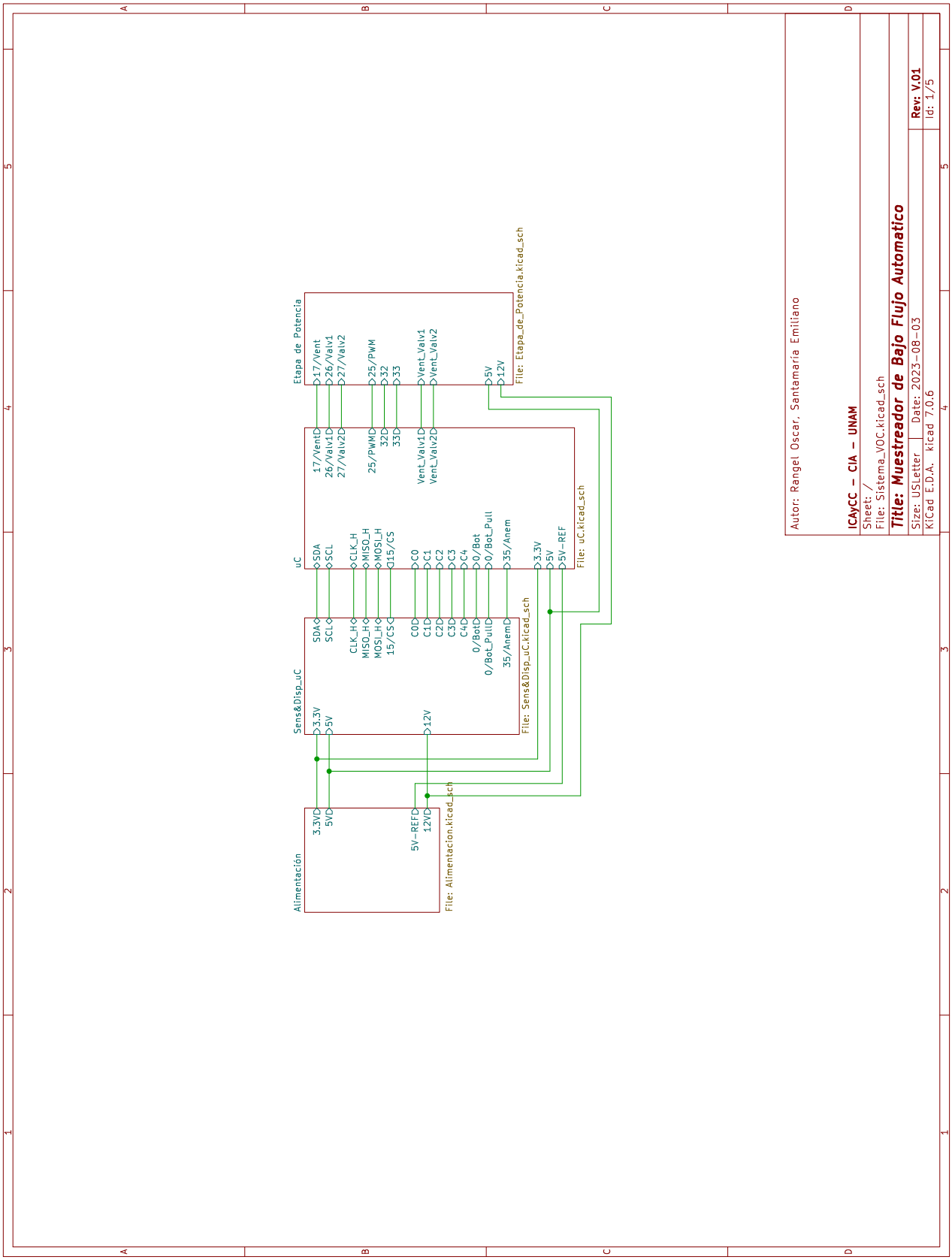
```
//#define USE_PIN // Uncomment this to use PIN during pairing. The pin is specified on the line below
const char *pin = "1234"; // Change this to more secure PIN.

String device_name = "ESP32-BT-Slave";

#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and enable it
#endif

#if !defined(CONFIG_BT_SPP_ENABLED)
#error Serial Bluetooth not available or not enabled. It is only available for the ESP32 chip.
#endif

BluetoothSerial SerialBT;

void setup() {
  Serial.begin(115200);
  SerialBT.begin(device_name); //Bluetooth device name
  Serial.printf("The device with name \"%s\" is started.\nNow you can pair it with Bluetooth!\n", device_name.c_str());
  //Serial.printf("The device with name \"%s\" and MAC address %s is started.\nNow you can pair it with Bluetooth!\n", device_name.c_str(), SerialBT.getMacString()); // Use this after the MAC method is implemented
  #ifdef USE_PIN
    SerialBT.setPin(pin);
    Serial.println("Using PIN");
  #endif
}

void loop() {
  if (Serial.available()) {
    SerialBT.write(Serial.read());
  }
  if (SerialBT.available()) {
    Serial.write(SerialBT.read());
  }
  delay(20);
```

```
35 }
```

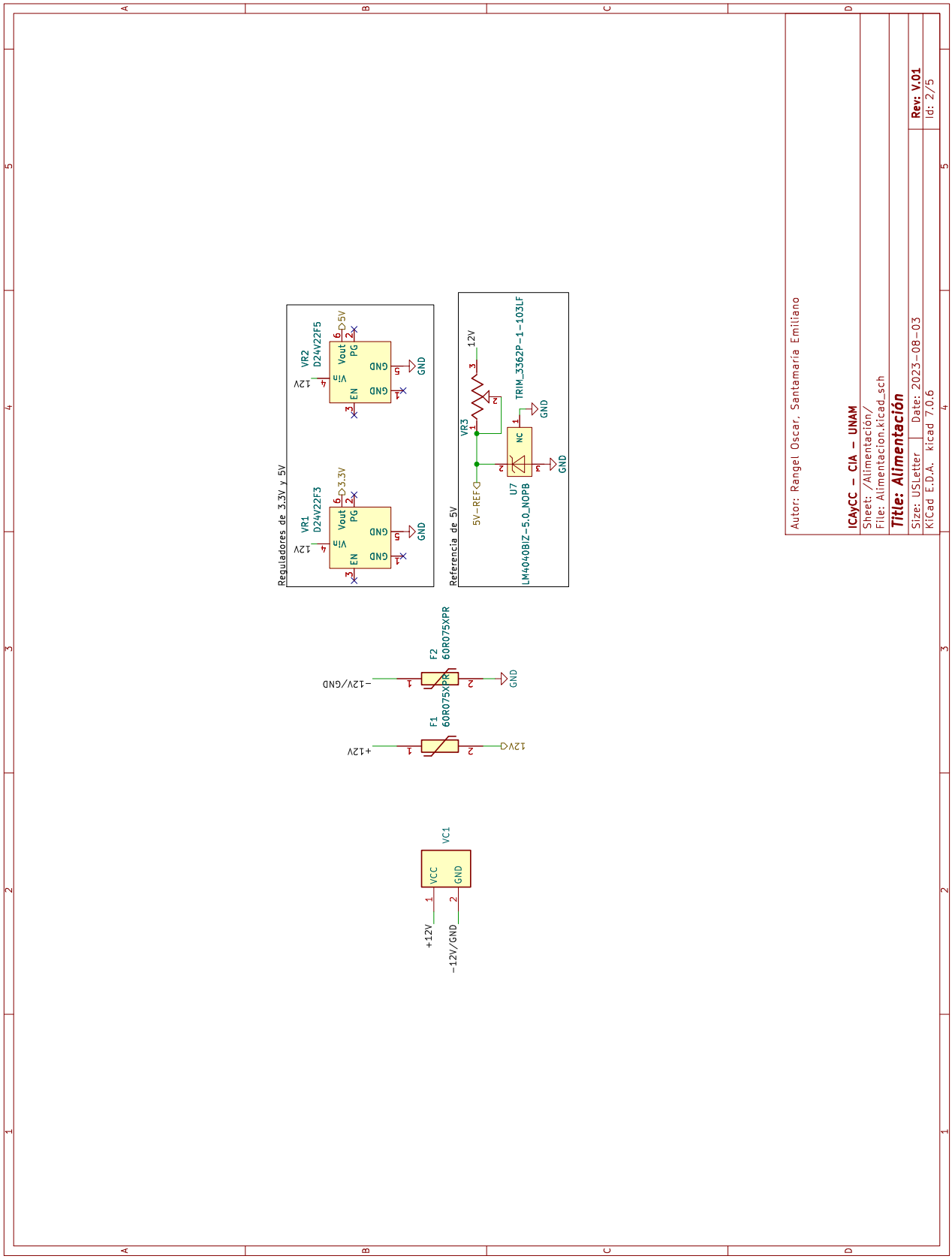
C PCB

1. Esquemático

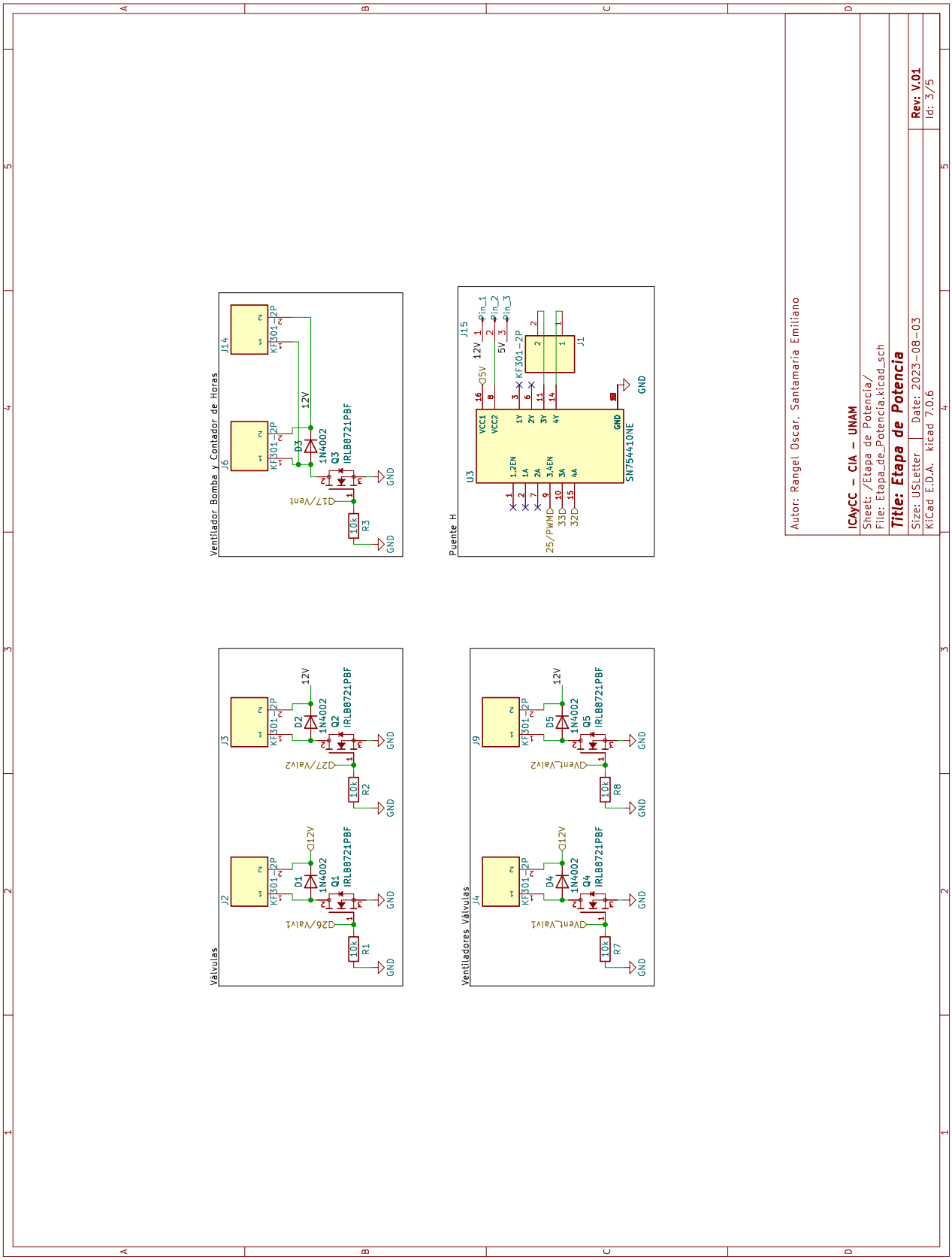
1.1. Muestreador Automático de Bajo Flujo (MABF)



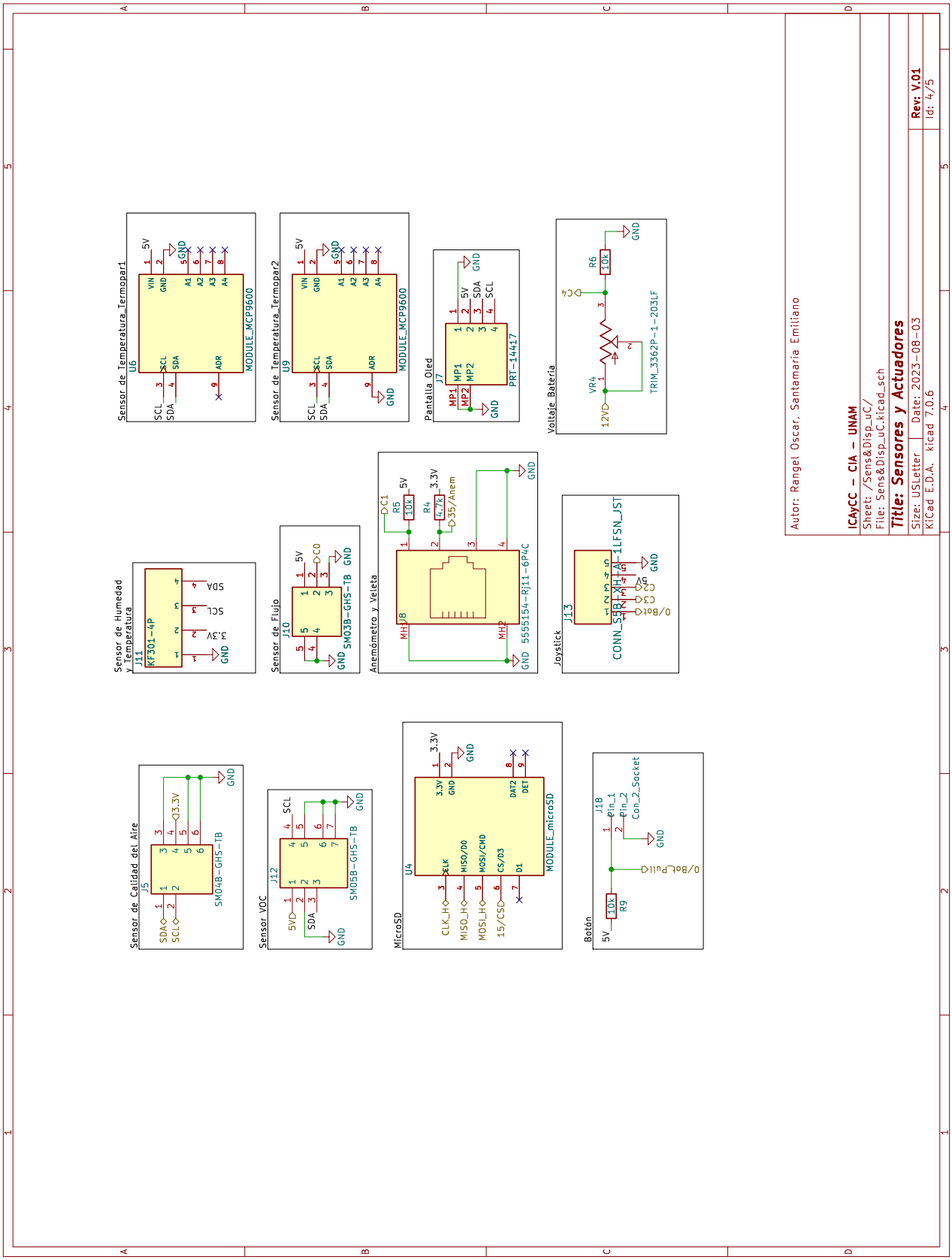
1.2. Alimentación



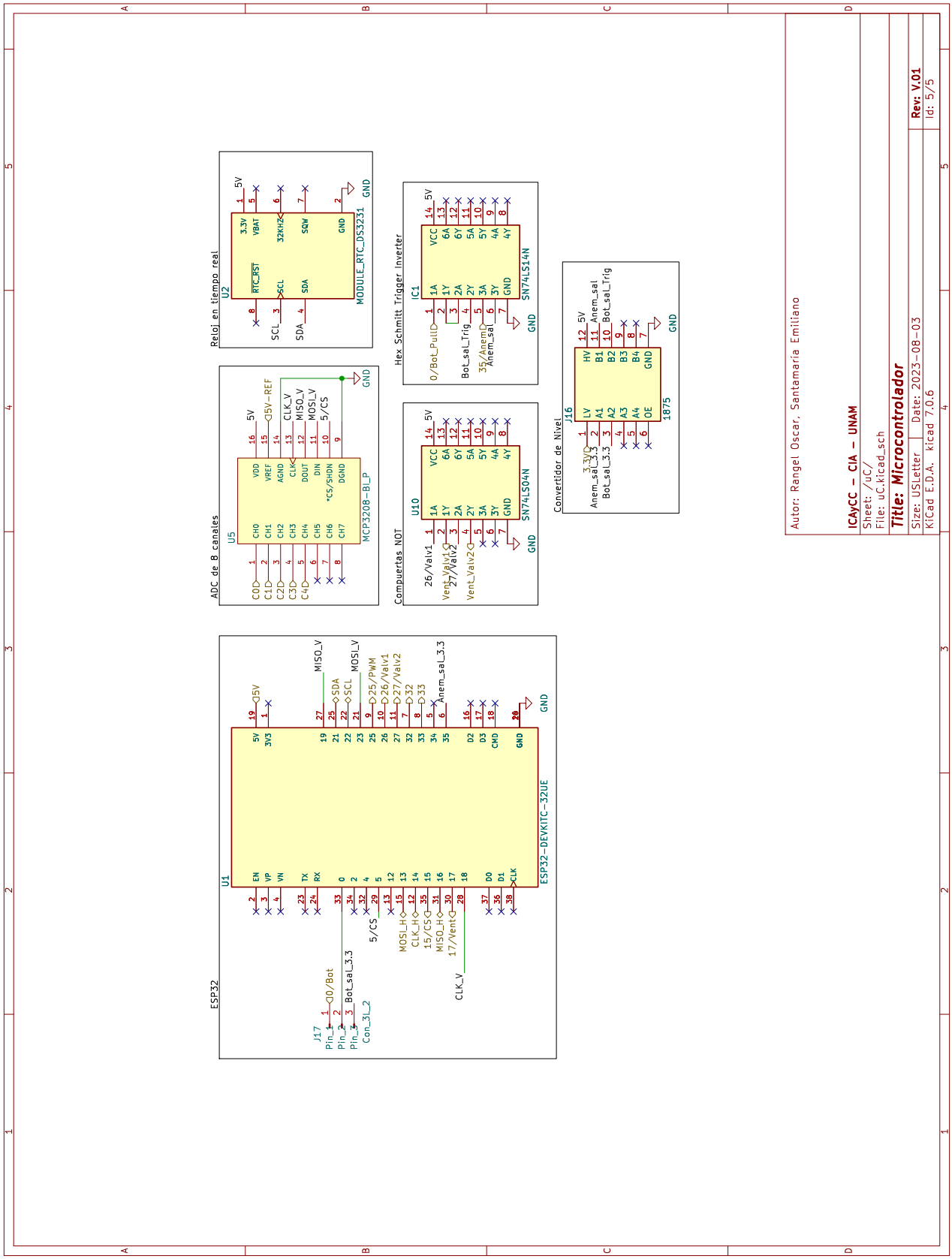
1.3. Etapa de potencia



1.4. Sensores y actuadores



1.5. Microcontrolador



D Manual Técnico

Muestreador Automático de Bajo Flujo (MABF)– Manual (App, Configuración y Uso)

v. 2024-1

Eduardo Emiliano Santamaría de la Rosa, Oscar Ivan Rangel Reyes,

15 de diciembre de 2024

Resumen

Este muestreador de Compuestos Orgánicos Volátiles (COVs) es un dispositivo autónomo equipado con sensores meteorológicos y de calidad del aire, que permiten su monitoreo en tiempo real, ya sea mediante una pantalla integrada o a través de conexiones Bluetooth y WiFi. Su diseño permite programar pruebas con fecha, hora, duración e intensidad de flujo ajustables, adaptándose a diversas necesidades. Gracias a su sistema de alimentación solar y baterías recargables, puede operar de forma continua sin necesidad de conexión directa a la red eléctrica.

Índice

1. Descripción General del MABF

El Muestreador, que se puede ver en la Figura 1, está equipado con varios sensores meteorológicos y módulos específicos. Algunos de estos se conectan externamente a la caja mediante cables y conectores, mientras que otros están integrados directamente en la tarjeta electrónica diseñada la cual se muestra en la Figura 6.

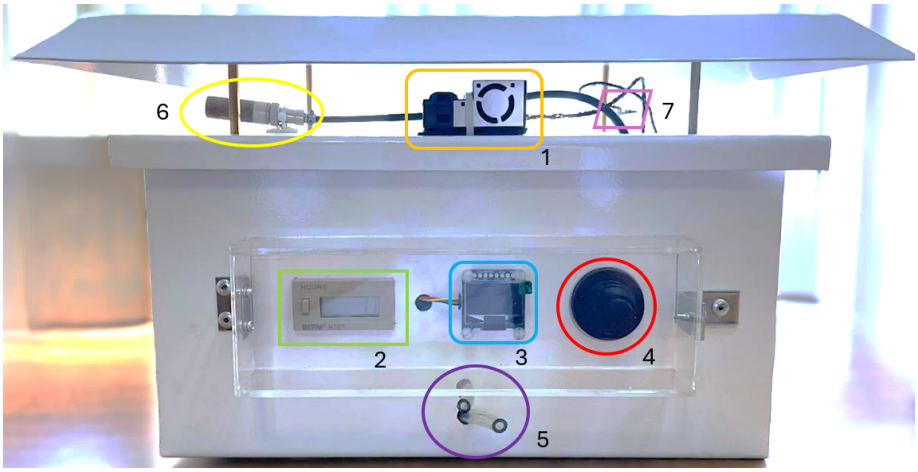


Figura 1: MABF visto de frente.

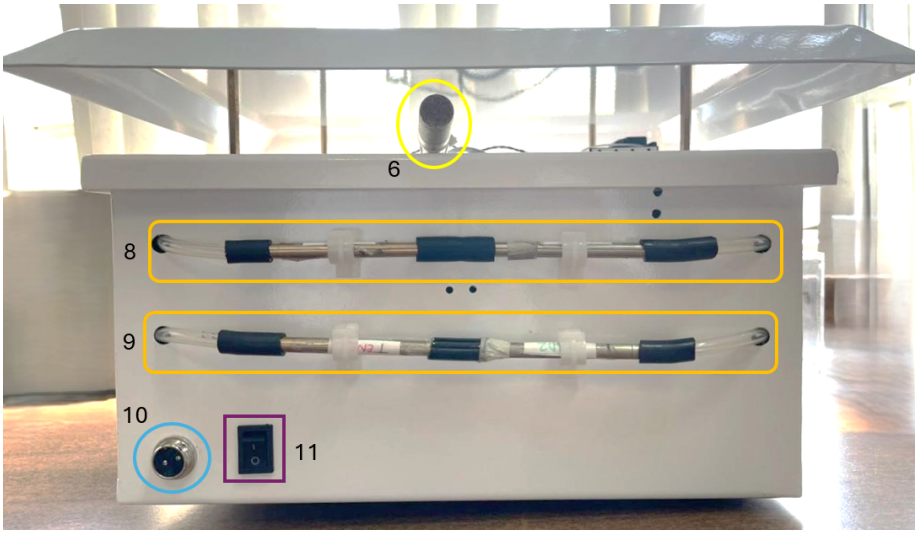


Figura 2: MABF visto de lado derecho.

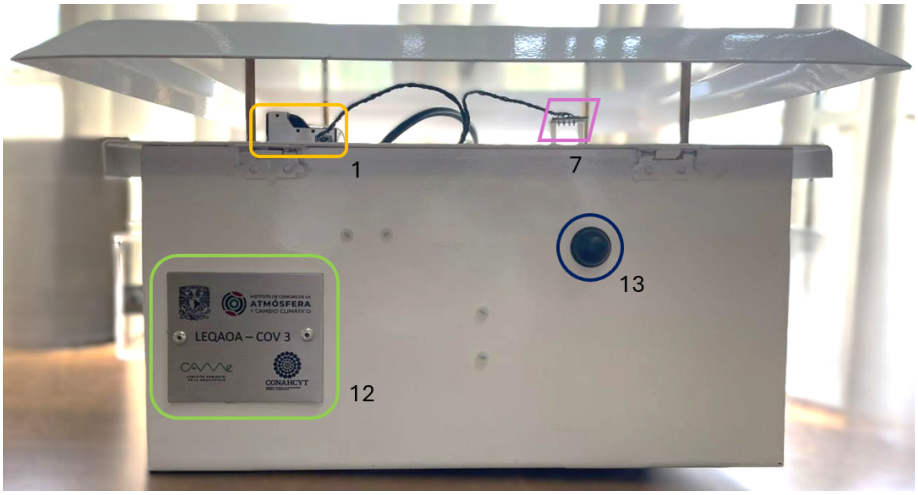


Figura 3: MABF visto de lado izquierdo.

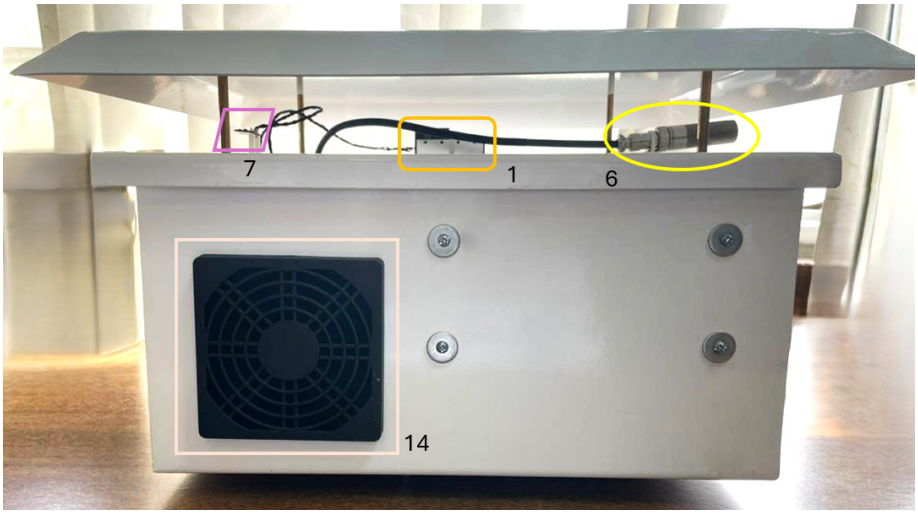


Figura 4: MABF visto de atrás.

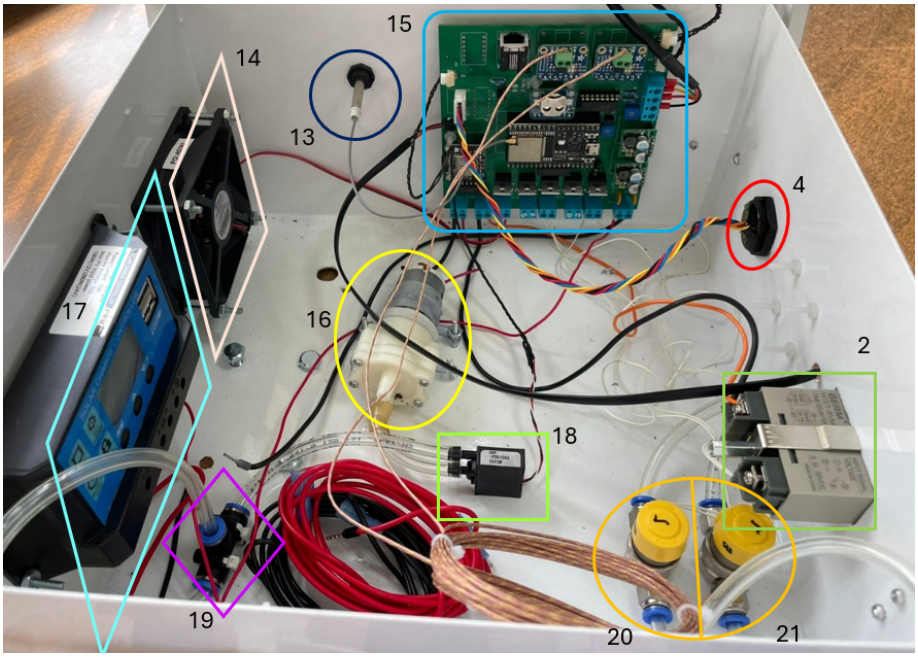


Figura 5: MABF visto desde adentro.

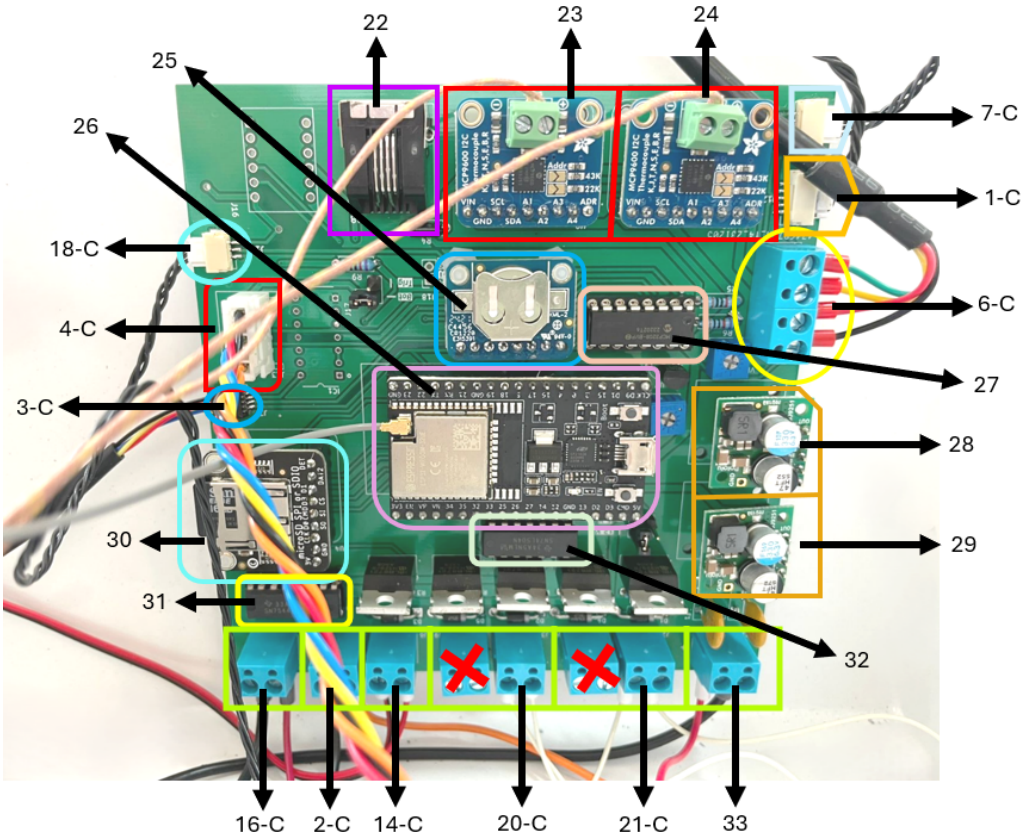


Figura 6: Tarjeta electrónica diseñada.

Cada una de las figuras anteriores contiene una numeración. A continuación, se detalla el significado de cada número asociado a los componentes.

1. SEN55: Sensor de hidrocarburos y material particulado con interfaz I2C.
2. H7Et BVM: Contador eléctrico digital de horas.
3. Display Gráfico OLED: Pantalla monocromática con interfaz I2C.
4. TS6T1S02A: Joystick o palanca de mando para controlar el sistema.
5. Mangueras de entrada: Tomas de aire para la colecta.
6. SHT-30: Sensor de temperatura y humedad (termohigrómetro) resistente a la intemperie con protección de malla.
7. SEN0392: Sensor de calidad del aire SGP40 con interfaz I2C.
8. Tubos de Colecta Diurna: Cartuchos con absorbentes recomendados para capturar muestras de aire durante el día (se activan al inicio de una prueba programada).
9. Tubos de Colecta Nocturna: Cartuchos con absorbentes recomendados para capturar muestras de aire durante la noche (se activan después de los tubos diurnos en una prueba programada).
10. Pines de Aviación GX16-2 macho: Conector para la entrada del panel solar.
11. SWITCH BALANCÍN KCD11: Interruptor para encender y apagar el sistema.
12. Placa de aluminio: Identificador del muestreador correspondiente según el caso.
13. ANT-DB1-WRT-UFL: Antena WiFi/Bluetooth dipolo omnidireccional para exteriores.
14. Ventilador de plástico de 4": Ventilador para enfriar la tarjeta electrónica y el compresor de aire.
15. Tarjeta electrónica diseñada.
16. Bomba de aire: Dispositivo para la colecta de aire en los cartuchos.
17. Controlador de Carga Solar: Dispositivo encargado de gestionar la carga y descarga adecuada de las baterías utilizando energía proveniente del panel solar.
18. D6F-P0010A2: Sensor de flujo de aire analógico.
19. Conector Neumático T: Dispositivo utilizado para dividir una entrada de aire en dos salidas.
20. Electroválvula 2: Válvula eléctrica encargada de controlar el paso de aire a través del cartucho de colecta nocturno.
21. Electroválvula 1: Válvula eléctrica encargada de controlar el paso de aire a través del cartucho de colecta diurno.

22. 5555154-2: Conector RJ11 hembra utilizado para el sistema de viento, conformado por un anemómetro y una veleta.

23. MCP9600: Módulo amplificador con interfaz I2C para conectar el termopar 2 y monitorear la temperatura de algún cartucho.

24. MCP9600: Módulo amplificador con interfaz I2C para conectar el termopar 1 y monitorear la temperatura de algún cartucho.

25. DS3231 Precision RTC: Reloj en tiempo real utilizado para obtener fecha y hora precisa, incluso sin conexión a Internet.

26. ESP32-WROOM-32UE: Microcontrolador utilizado para el funcionamiento del sistema, encargado de controlar los módulos, sensores y actuadores.

27. MCP3208-BI/P: Convertidor analógico a digital (ADC) de 8 canales con interfaz SPI.

28. D24V22F5: Regulador de voltaje utilizado para obtener 5 Volts.

29. D24V22F3: Regulador de voltaje utilizado para obtener 3.3 Volts.

30. Adafruit Micro SD SPI: Módulo para lectura y escritura en una tarjeta micro SD con interfaz SPI, utilizado para guardar datos.

31. SN754410NEE4: Controlador de la bomba, que permite mandar una señal PWM.

32. SN74LS04N: Compuerta digital NOT utilizada para invertir las señales que activan las electroválvulas.

En la Figura 6 se observan varios números acompañados de una “C”. Esto indica el conector o la ubicación en la que debe conectarse el dispositivo o sensor correspondiente con el mismo número dentro de la placa electrónica.

2. Uso de display del muestreador

2.1. Menu principal

El menú principal del display cuenta con tres opciones a elegir (MANUAL, CONFIG. y DATOS), como se muestra en la Figura 7. Utilizando el joystick puedes desplazar la selección (Círculo blanco a la izquierda de cada opción) del menú hacia arriba o abajo. Cuando la selección se encuentre a la izquierda de la opción deseada, dar click con el joystick (El joystick se puede dejar presionado hasta que se produzca un cambio en la pantalla) para ingresar a la misma.

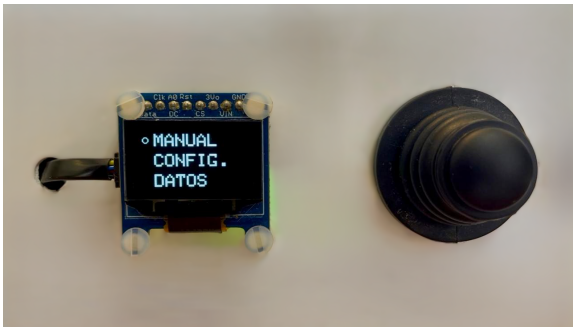


Figura 7: Menú Principal Display y Joystick.

2.2. Página MANUAL Display

En la página MANUAL puedes encender la bomba y las válvulas instantáneamente para realizar pruebas rápidas. Además puedes variar el flujo de la bomba a voluntad.

Figura 8: Selección MANUAL en Display.

Al entrar en esta opción se mostrará una pantalla como la Figura 9. El recuadro blanco que rodea a V1 en la imagen, representa el cursor. Si das click con el cursor rodeando a V1 la válvula 1 se abrirá y se encenderá la bomba programada al flujo predeterminado (50 mL). Lo mismo sucederá si posicionas el cursor en V2 y das click.

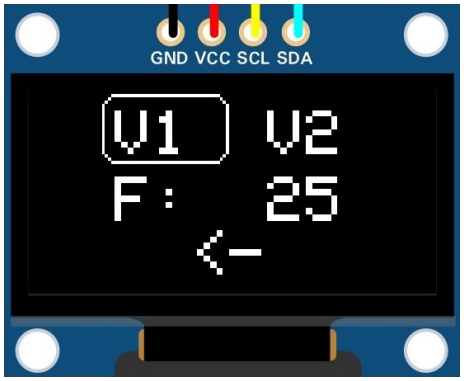


Figura 9: Pantalla MANUAL Display.

En el momento que des click a cualquiera de las dos válvulas (V1 y V2), aparecerá un círculo blanco a un lado del texto, como se muestra en la Figura 10, lo que indicará que la válvula esta abierta. Puedes desplazarte por toda la página con el cursor y las válvulas sin problemas, sólo cambiarán su estado hasta que reciban otro click.

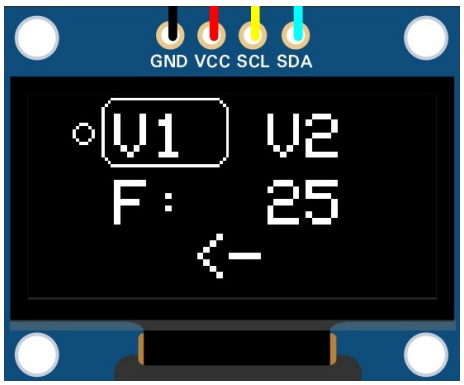


Figura 10: Indicador de Estado de Válvulas Display.

Para cambiar el flujo de la bomba, se debe posicionar el cursor en el número al lado derecho del texto “F:” (Figura 11), dar click y desplazarse hacia arriba o abajo por los valores posibles de flujo. Cuando se tenga el valor de flujo deseado, dar click y aparecerá un círculo blanco, como muestra la Figura 11, indicando que el flujo de la bomba ha sido modificado y que puedes desplazar el cursor por la pantalla nuevamente. Los cambios en el flujo de la bomba se conservarán, incluso en pruebas programadas, a menos que se modifique el mismo en esta pantalla o por otros medios.

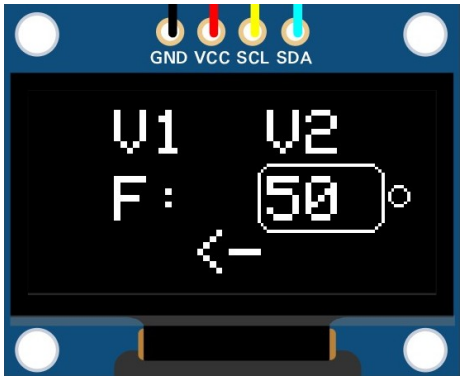


Figura 11: Cambio de Flujo en Display.

Para regresar al menu principal se debe colocar el cursor en la flecha apuntando a la izquierda (Figura 12) y dar click. Las válvulas y las bombas se mantendrán en los valores que se seleccionaron, no será hasta que se vuelva a entrar a esta pantalla que las válvulas se cerrarán y la bomba se apagará.

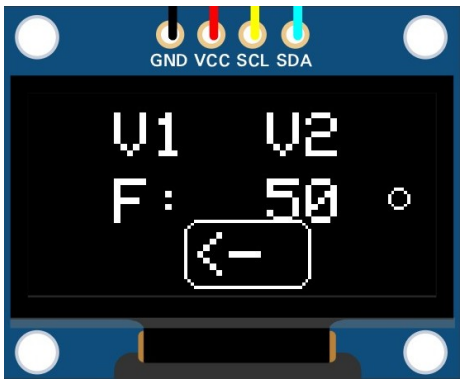


Figura 12: Salida Pantalla MANUAL.

2.3. Página CONFIG. Display

En la página de CONFIG. se podrá programar una prueba de 24 horas (12 horas un cartucho y 12 horas el otro cartucho) colocando el día, mes, hora y minuto a la que se quiera que inicie esta prueba.

Figura 13: Selección MANUAL en Display.

En la página se encuentran cuatro opciones (Día, mes, hora y minuto) como se muestra en la figura 14. La selección de las opciones es el rectángulo blanco que rodea a la palabra DIA en la figura 14. Esta selección la puedes desplazar hacía arriba, abajo, izquierda o derecha para elegir cualquiera de las cuatro opciones.

Figura 14: Pantalla CONFIG. Display.

Para modificar una de las cuatro variables, se posiciona el selector en la variable a modificar y se hace click con el joystick. La variable cambiará de texto a número cuando se haga click sobre ella como se muestra en la figura 15. Cuando suceda esto, podrás cambiar de valor la variable desplazándote con el joystick hacía arriba o hacía abajo.

Figura 15: Selección de Variables.

Cuando se llegue al valor deseado de la variable, se tiene que volver a dar click. Con el click se mostrará un círculo blanco a un lado de la variable modificada (Figura 16). Este círculo indicará que el valor de esa variable se fijó y puedes volverte a desplazar entre variables para modificarlas.

Figura 16: Fijando una Variable.

En el momento que todas las variables hayan sido modificadas y fijadas, como se muestra en la figura 17, ya sólo es necesario desplazarte al letrero que muestra el texto .ºK", dar click y si el display regresa al menú principal la prueba habrá quedado programada correctamente.

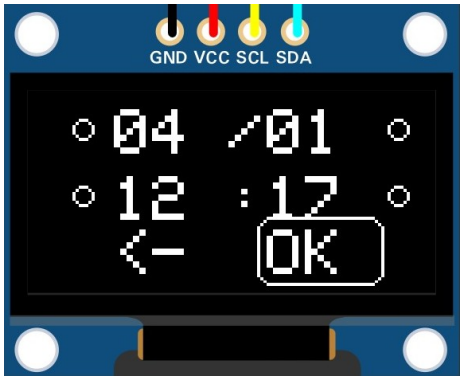


Figura 17: Variables Modificadas y Fijas.

2.4. Página DATOS Display

En esta página podrás observar los datos maestreados en tiempo real.

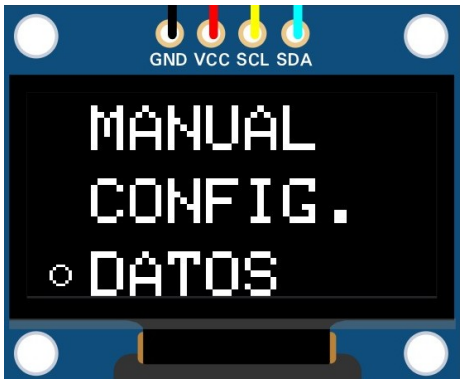
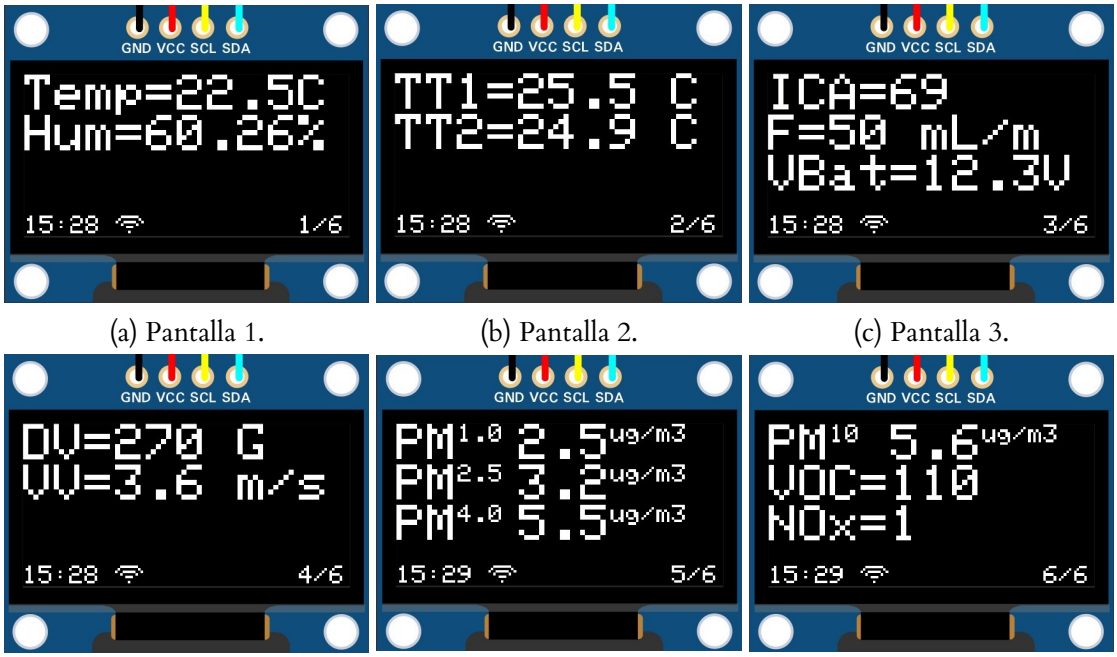


Figura 18: Selección MANUAL en Display.

La pantalla mostrará, en el lado izquierdo, el nombre del dato y, en el lado derecho, el valor en tiempo real de esa variable. En la esquina inferior izquierda se encuentra la hora actual en formato de 24 horas, junto con un ícono de Wi-Fi que indica si el muestreador está conectado a internet. Si este ícono no aparece, significa que se ha perdido dicha conexión. En la esquina inferior derecha se muestra el número de página en la que se encuentra el display.

Podrás desplazarte a través de las seis páginas de datos (Figura 19f) utilizando el joystick hacia la derecha o hacia la izquierda. Cuando alcances los límites de estas páginas (Página 1 o Página 6), no será posible continuar desplazándote: hacia la derecha en la Página 6 o hacia la izquierda en la Página 1. Para regresar al menú principal, simplemente presiona el joystick desde cualquier página.



(a) Pantalla 1. (b) Pantalla 2. (c) Pantalla 3.

(d) Pantalla 4. (e) Pantalla 5. (f) Pantalla 6.

Figura 19: Pantalla DATOS Display

3. App Móvil para el Muestreador Automático – Sistema VOC

Para utilizar esta aplicación, es necesario contar con un dispositivo Android que tenga Bluetooth, preferiblemente un teléfono.

3.1. Instalación

1. En el dispositivo donde se va a instalar, accede al siguiente enlace de Google Drive: `https://drive.google.com/drive/folders/1Mkf7OD1jbQ_ERAKEtO8NOJyJN3hH66qD?usp=drive_link` y descarga el archivo: SistemaVOCMIT.apk o escanea el código QR de la figura 20 y descarga el el archivo: SistemaVOCMIT.apk

Figura 20: Código QR para descargar app.

2. Una vez completada la descarga, abre el archivo .apk descargado.

3. Se abrirá un mensaje que te dirá si deseas instalar la aplicación. Selecciona “Instalar”.
4. Cuando finalice la instalación, basta con seleccionar “Abrir” y serás redirigido a la aplicación.

3.2. Antes de usar la App

Antes de abrir la aplicación móvil y comenzar a usarla, es necesario configurar el Bluetooth. Para ello, abre la configuración de Bluetooth en el celular y busca dispositivos cercanos. Deberá aparecer un dispositivo con el nombre “Sistema VOC_M#”, donde “#” corresponde a un número del 1 al 10 que identifica a cada muestreador. Por ejemplo, si tienes el muestreador 7, el nombre será “Sistema VOC_M7”.

Una vez que el dispositivo aparezca en la lista, selecciónalo. El sistema mostrará un mensaje preguntando si deseas vincularlo; selecciona “Sí o Vincular”. El mensaje será similar al que se muestra en la siguiente imagen 21.

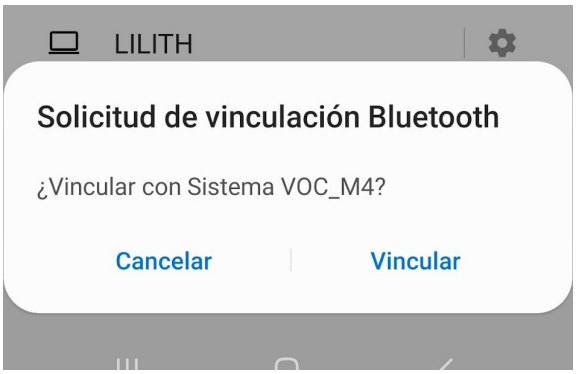


Figura 21: Mensaje mostrado para vincular dispositivo

Ya con el dispositivo vinculado correctamente podemos pasar al uso de la aplicación móvil.

3.3. Uso de la App

Al ingresar a la aplicación, se muestra el menú, que tiene tres botones: MANUAL, PRUEBA y DATOS. Su función es similar a la de los botones que llevan el mismo nombre en el display del muestreador. En la Figura 22 se muestra dicho menú.

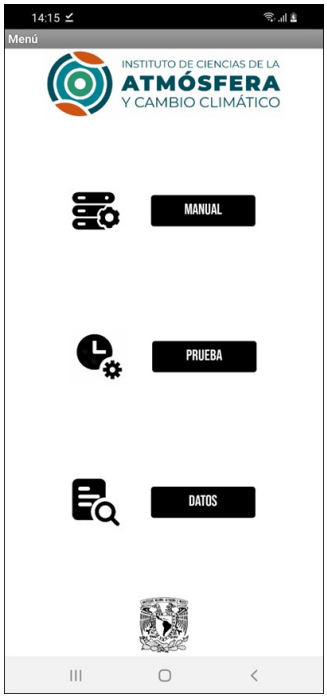


Figura 22: Menú mostrado al ingresar a la App.

Al seleccionar cualquiera de los tres botones, se abrirá una página con el nombre del botón seleccionado. Esta página incluirá dos botones: uno llamado “ESTADO” y otro llamado “SALIR”, tal como se muestra en la figura 23. El nombre esta resaltado en rojo y se muestran los botones mencionados.

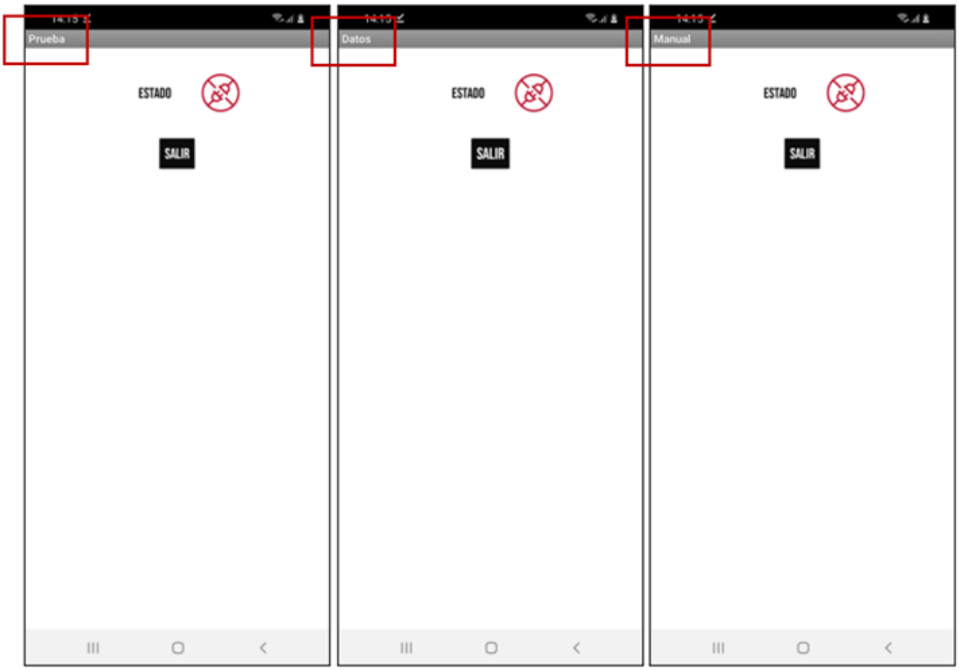


Figura 23: Paginas mostradas al presionar cualquier botón en el menú

Para esto ya que estemos dentro de la página solicitada tendremos que apretar el botón rojo que esta a un lado de la palabra estado el cual es un cable de color rojo que indica des conexión. El cual desplegará una lista de dispositivos bluetooth vinculados al teléfono en uso, en esta lista deberemos buscar el nombre que se vinculo en la sección 3.2 (“Sistema VOC_M#”). Se selecciona el nombre correspondiente y, en caso de que la conexión haya sido exitosa, el letrero cambiará su texto a ÇONECTADO", como se muestra en la figura 24.

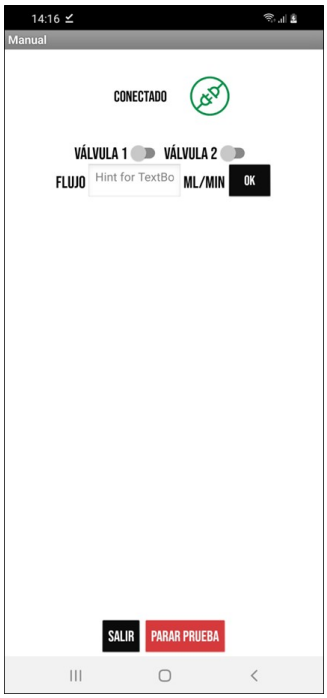


Figura 24: Pagina mostrada al seleccionar el botón MANUAL y conectarse al dispositivo

En caso de que no se haya logrado conectar, el letrero mostrará el texto “ERROR DE CONEXIÓN” y se tendrán que realizar nuevamente los pasos anteriores para la conexión de un dispositivo.

3.4. Página MANUAL App

De manera similar a la página “MANUAL” del display, es posible encender las válvulas, ya sea por separado o juntas, y ajustar el intervalo de tiempo durante el cual permanecerán encendidas las válvulas seleccionadas.

De manera inicial, ambas válvulas se muestran en color gris. Cuando se activa una o ambas, cambian a color verde y aparece un control deslizante (slider) que indica: “Válvula 1” si se selecciona únicamente la válvula 1, “Válvula 2” si se selecciona únicamente la válvula 2, o “Intervalo” si se seleccionan ambas válvulas. Esto se ilustra en la Figura 25.

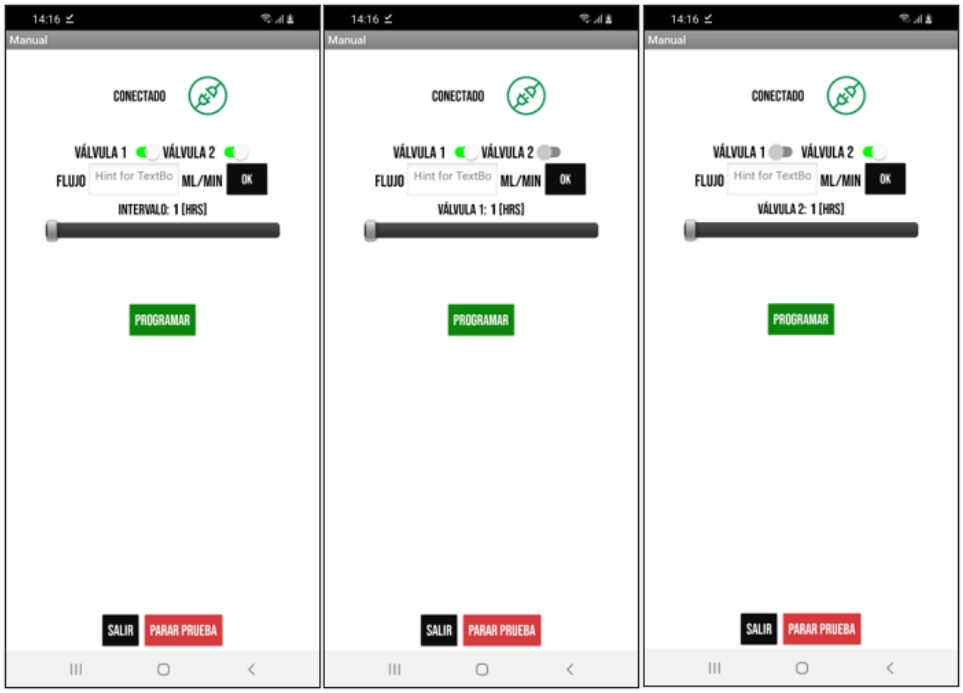


Figura 25: Pagina mostrada al seleccionar las combinaciones entre válvulas

El control deslizante (slider) que aparece se puede mover de forma táctil de manera sencilla. Esto permite modificar el tiempo durante el cual las válvulas seleccionadas permanecerán encendidas. El valor puede ajustarse entre 1 y 12 horas. En la Figura 26 se muestra el slider configurado en 12 horas con ambas válvulas seleccionadas.

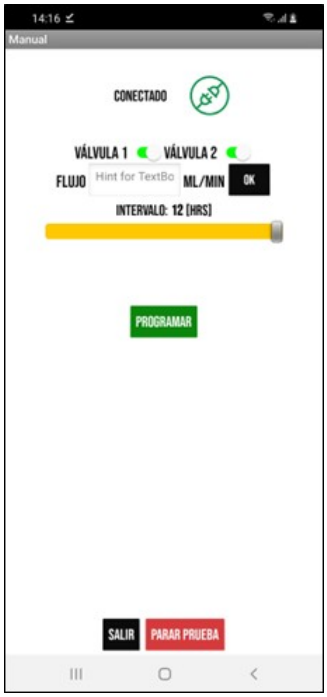


Figura 26: Pagina de ejemplo al seleccionar ambas válvulas con un valor de 12 horas

Por último, basta con presionar el botón “PROGRAMAR” para que los cambios se apliquen en el muestreador y se enciendan las válvulas seleccionadas durante el tiempo configurado en el slider. Cabe destacar que, si se seleccionan ambas válvulas, por ejemplo, con un intervalo de 7 horas, primero se encenderá la válvula 1 durante 7 horas; luego se apagará y se encenderá la válvula 2 también por 7 horas.

Como función adicional, existe un apartado denominado “FLUJO”. Al seleccionar el cuadro de texto (Hint for Text), se desplegará un teclado numérico (Figura 27) que permite modificar el valor del flujo del muestreador, el cual puede ajustarse entre 50 y 150 ml/min. Para cambiar el valor, basta con ingresarlo y presionar el botón “OK” ubicado a la derecha. De forma predeterminada, el flujo está configurado en 50 ml/min, por lo que no es necesario modificarlo a menos que se requiera un ajuste.

Figura 27: Teclado numérico para cambiar el flujo en el Muestreador

Es importante tener en cuenta que, si hay una prueba en curso y se cambia el valor de flujo, la prueba se detendrá. Por esta razón, se recomienda ajustar primero el valor del flujo y luego programar la prueba.

3.5. Página PRUEBA App

Esta página cuenta con la misma función que la página CONFIG. del display, la cual es programar una prueba para un día y hora en específico. Sin embargo, esta página tiene funciones adicionales a las del display.

Como se muestra en la imagen 28, hay distintos botones que indican en qué fecha y a qué hora se quiere programar la prueba. Al elegir el día y hora correctos se presiona el botón inferior que dice “ACEPTAR”.

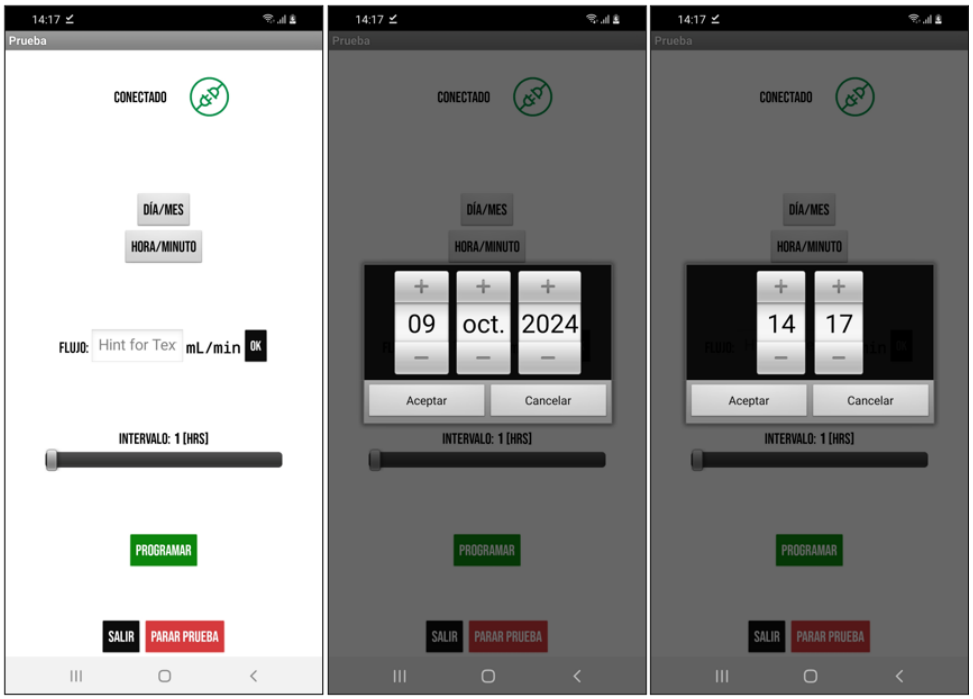


Figura 28: Pagina mostrada al seleccionar el botón PRUEBA y conectarse al dispositivo.

Además de estos botones la página cuenta con un cuadro de texto en el que podrás definir el flujo de la bomba en mililitros sobre minuto, de la misma forma que en la sección 3.4. Por último, la página cuenta con un control deslizante que define el intervalo en el que conmutarán las válvulas. Si se coloca este control deslizante en 12 hrs, la prueba durará 24 hrs con 12 hrs succionando por la válvula 1 y 12 hrs succionando por la válvula 2 (Figura 29).

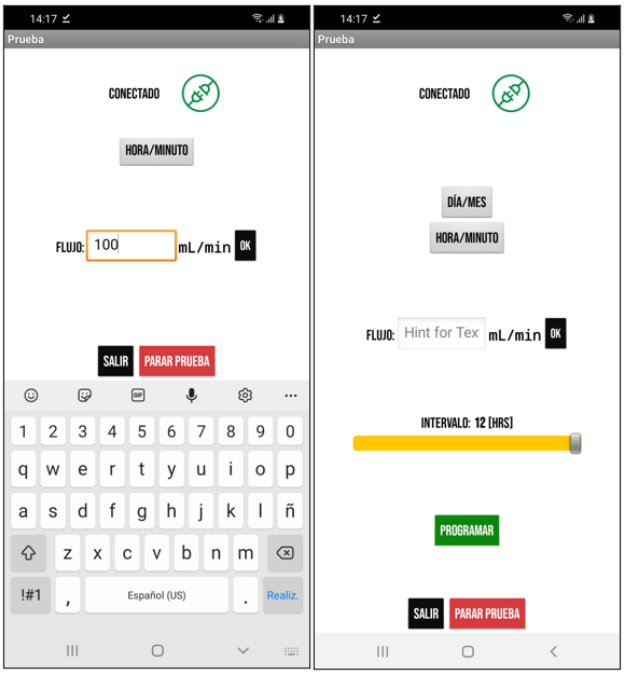


Figura 29: Cambio de Flujo e Intervalo de Válvulas.

Cuando se tenga todo definido para iniciar la prueba, se presiona el botón con el texto “PROGRAMAR” y se mostrará una ventana de texto que confirma el día y la hora de programado de la prueba, como se muestra en la figura 32.

Figura 30: Mensaje mostrado al programar una prueba.

3.6. Página DATOS App

En esta página será posible ver los datos en tiempo real del muestreador y los promedios cada 15 minutos del mismo. Al estar conectado con el muestreador, los datos empezarán a desplegarse en esta página.

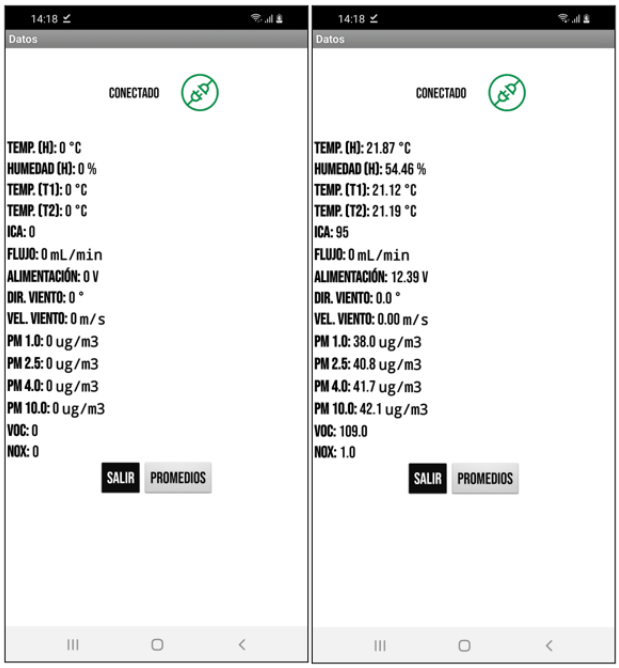


Figura 31: Datos mostrados al conectarse a la página de “Datos”.

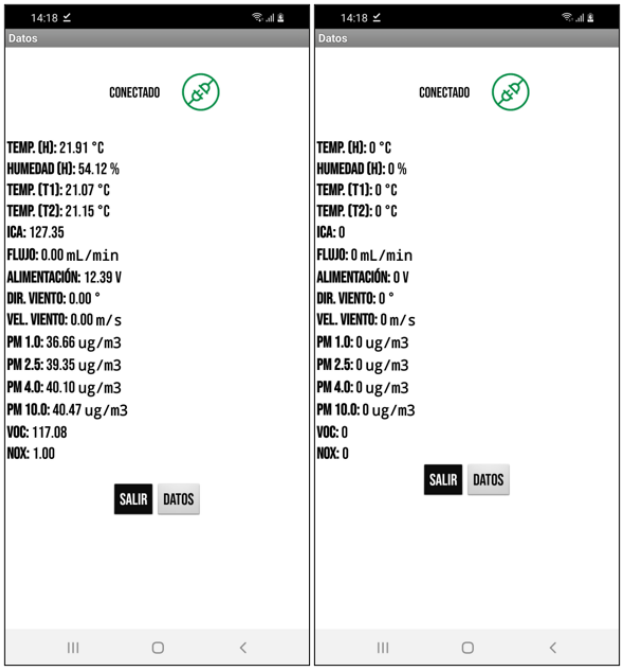


Figura 32: Datos mostrados al conectarse a la página de datos y seleccionar el botón “PROMEDIOS”.

E Cuestionario para voluntarios

7 respuestas

Ver en Hojas de cálculo

Resumen | Pregunta | Individual

Nombre

7 respuestas

- Reyna Aideé Mares Flores
- Noé Facundo Roque Pacheco
- Oscar Martinez Cervantes
- Octavio Ortiz alvarado
- Jorge Luis Hernández Chávez
- Marisol Gordillo
- Yeriana León cruz

Lugar donde usaste el sistema de monitoreo

Copiar gráfico

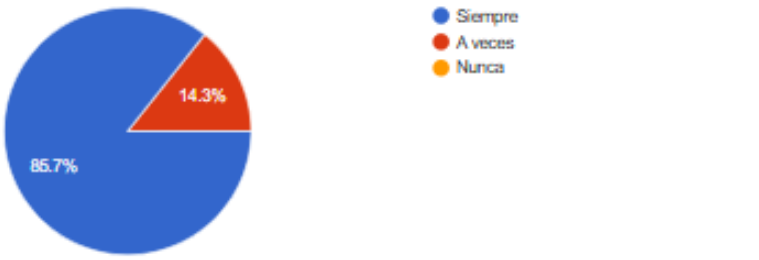
7 respuestas

Funcionamiento del sistema

¿El sistema encendió correctamente cuando lo necesitaste?

Copiar gráfico

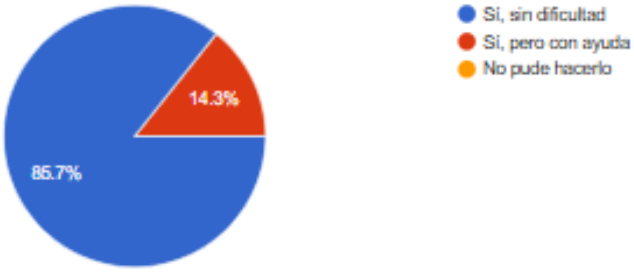
7 respuestas



¿Pudiste programar el inicio de la prueba sin problemas?

Copiar gráfico

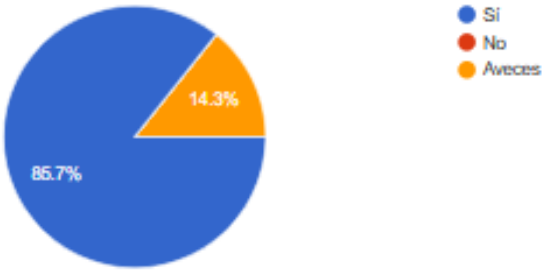
7 respuestas



¿El sistema responde rápidamente a las acciones que realizas en la interfaz?

Copiar gráfico

7 respuestas



Durante los meses que estuvo el sistema instalado. ¿Consideras que soportó climas extremos? Tales como: fuertes lluvias, calor excesivo, vientos, etc. Describe brevemente cuales y en que temporadas

7 respuestas

Considero que si soportó de manera correcta los climas, no se ve desgastado ni dañado.
La única parte que necesita refuerzo son las grapas de soporte donde se insertan los trenes, ya que se rompio una durante el primer mes.

soportó climas extremos

Si resistió, resistió fuertes vientos, aguaceros, granizadas y demás cambios repentinos del clima.

Si soporto climas extremos sin problema

Nula

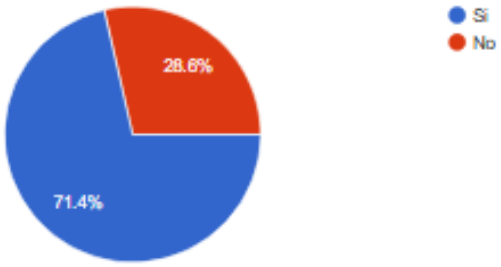
Los vientos fuertes de febrero a mayo

Si soporto fuertes vientos

¿El sistema funciona de manera confiable durante largos periodos de tiempo sin apagarse, dejar de responder o presentar lentitud?

Copiar gráfico

7 respuestas



De las 21 pruebas que se realizaron ¿En cuántas ocasiones consultó a cualquiera de los involucrados para resolver dudas sobre la programación o pruebas de fugas? (Escriba un número aproximado)

7 respuestas

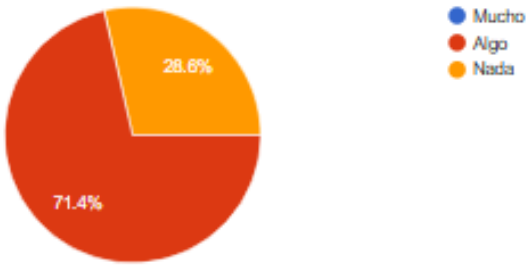
10
1
16
2 o 3 veses
2
Dos
Cómo 3 veces

Manejo de cartuchos y flujo

¿Qué tan difícil fue colocar y retirar los cartuchos?

Copiar gráfico

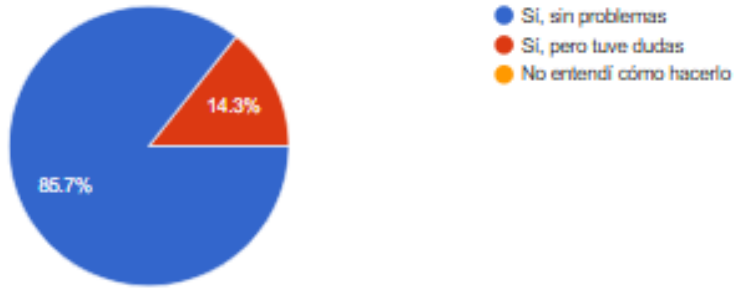
7 respuestas



¿Pudiste verificar el flujo de aire siguiendo las instrucciones que te proporcionaron con el sistema o durante la campaña de capacitación?

Copiar gráfico

7 respuestas

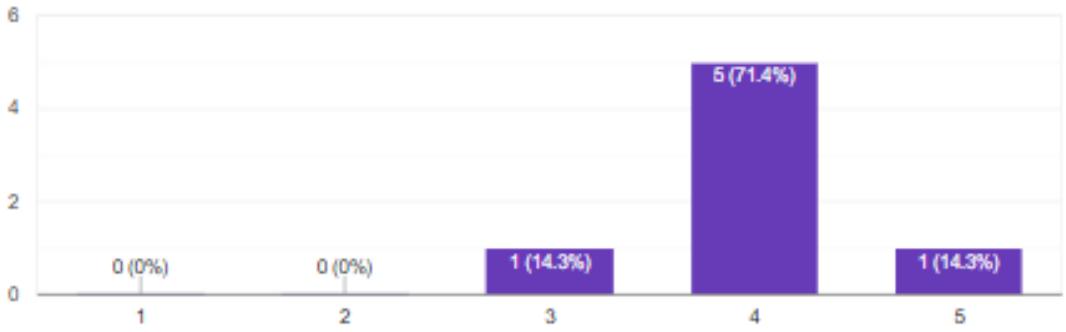


Facilidad de uso

¿Qué tan fácil fue entender cómo usar el sistema la primera vez?

Copiar gráfico

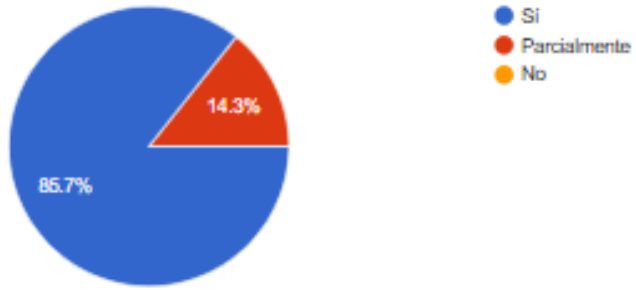
7 respuestas



¿Recibiste instrucciones claras para operar el sistema?

Copiar gráfico

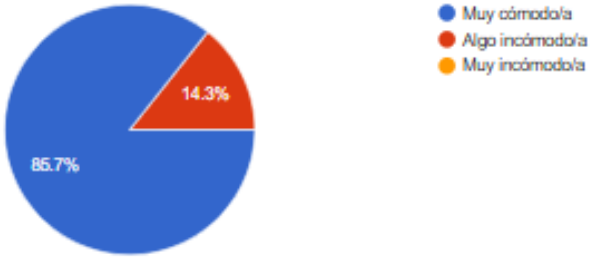
7 respuestas



¿Qué tan cómodo/a te sentiste usando el sistema?

Copiar gráfico

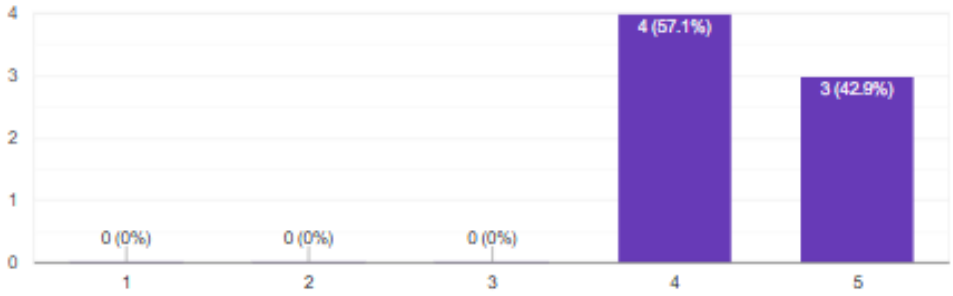
7 respuestas



¿Cómo calificarías la facilidad de uso de la interfaz del sistema, tanto de la pantalla, como de la aplicación del celular? (1-5, donde 1 es muy difícil y 5 es muy fácil)

Copiar gráfico

7 respuestas



Opinión general

¿Consideras que el sistema resulto útil para la comunidad?

7 respuestas

- Si
- Sí, la información generada será de gran utilidad para procesos comunitarios
- necesito saber los resultados finales
- Si, claro, y más claro nos nos va a quedar cuando nos compartan los resultados obtenidos.
- Considero q fue muy útil por q en estos tiempos ya no sabemos cómo está de contaminado el aire q respiramos teniendo muy serca una refinería esperamos q las pruebas q se isieron salga todo bien es mi opinión gracias
- Pues aún no se ,si hay mucha contaminación

¿Qué fue lo más difícil o confuso para ti durante estos meses de muestreo, hablando sobre el uso del aparato de muestreo?

7 respuestas

- Lo más dificil fueron los momentos en los que fallo la caja de muestreo, lo más fácil fue todo lo demás.
- la colocación de trenes, por el desgaste de las mangueras (negras) de conexión
- Comprendí la importancia de las mangueritas para realizar las lecturas correctamente, que si estaban muy duras o si estaban muy flojas.
  En la aplicación se me olvidaba seguido la parte que decia " parar prueba "
- Considero q todo estuvo Fasil por q cuando nos dieron las instrucciones los ingenieros nos explicaron paso por paso cada cosa q se iba a aser y yo en lo personal no se m complicó el aparato fundino bien
- Programar con joistyc
- Q no quedarán fugas
- Que hay veces las mangueritas negras quedaban flojas y había fuga

¿Qué mejorarías en la aplicación del celular para programar pruebas?

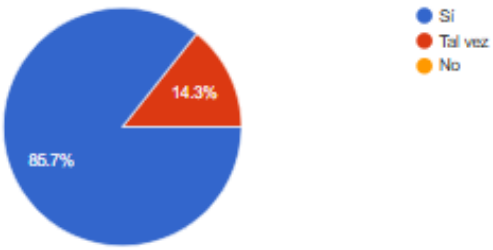
7 respuestas

- No tener que salir de la visualización de la prueba para ver la variación del flujo.
- funcionó correctamente
- No, nada, es muy práctica
- La aplicación no me dio ningún problema estuvo bien
- .
- Eso estuvo bien
- Nada

¿Te sentirías cómodo/a usando el sistema otra vez?

Copiar gráfico

7 respuestas



Bibliografía

[1] V. Soni, P. Singh, V. Shree, and V. Goel, "Effects of vocs on human health," *Air pollution and control*, pp. 119–142, 2018.

[2] G. del Estado de México, "Equipos de medición." https://rama.edomex.gob.mx/equipos_medicion.

[3] J. L. Santos and F. Farahi, *Handbook of Optical Sensors*. Boca Ratón, FL: CRC Press, 2014.

[4] J. P. Moore, *Volatile Organic Compounds*. New York, NY: Environmental Science Press, 2016.

[5] C. Puente and R. Ramaroson, "Medición y análisis de los compuestos orgánicos volátiles en la atmósfera: últimas técnicas, aplicabilidad y resultados a nivel europeo," 2010.

[6] L. Vallecillos, J. Riu, R. M. Marcé, and F. Borrull, "Air monitoring with passive samplers for volatile organic compounds in atmospheres close to petrochemical industrial areas. the case study of tarragona (2019–2021)," *Atmospheric Pollution Research*, vol. 15, 2024.

[7] L. Melymuk, P. Bohlin-Nizzetto, R. Prokes, P. Kukučka, and J. Klánová, "Sampling artifacts in active air sampling of semivolatile organic contaminants: Comparing theoretical and measured artifacts and evaluating implications for monitoring networks," *Environmental Pollution*, vol. 217, pp. 97–106, 2016.

[8] Secretaría de Economía / COTEMARNAT, "Estaciones meteorológicas, climatológicas e hidrológicas – parte 1: especificaciones técnicas que deben cumplir los materiales e instrumentos de medición de las estaciones meteorológicas automáticas y convencionales." https://www.gob.mx/cms/uploads/attachment/file/166835/nmx-aa-166-1-scfi-2013_1_.pdf, 2013. Publicado en el Diario Oficial de la Federación el 4 de septiembre de 2013; vigente desde el 3 de noviembre de 2013.

[9] E. Systems, "Issue #4726: Spi communication issues with multiple devices on esp32." https://github.com/espressif/esp-idf/issues/4726, 2021.

[10] T. Wescott, "Pid without a phd." https://web2.qatar.cmu.edu/~gdicaro/16311-Fall17/slides/PID-without-PhD.pdf, 2016.

[11] Arduino, "Arduino Software (IDE)." https://www.arduino.cc/en/software/.

[12] Arduino Forum, "Arduino Forum - Página principal." https://forum.arduino.cc/.

[13] T. Instruments, "Sn754410nee4 quadruple half-h drivers datasheet," 2021. Disponible en: https://www.ti.com/lit/ds/symlink/sn754410.pdf.

[14] Maxim Integrated, *DS3231 Extremely Accurate I²C-Integrated RTC with TCXO and Crystal*. Maxim Integrated, 2020. Disponible en: https://www.analog.com/media/en/technical-documentation/data-sheets/DS3231.pdf.

[15] Pololu Corporation, *D24V22F5 Step-Down Voltage Regulator Datasheet*. Pololu, 2022. Disponible en: https://www.pololu.com/product/2858.

[16] Pololu Corporation, *D24V22F3 Step-Down Voltage Regulator Datasheet*. Pololu, 2022. Disponible en: https://www.pololu.com/product/2857.

[17] Littelfuse, Inc., *RHEF100-AP Resettable Fuse PTC Datasheet*. Littelfuse, 2020. Disponible en: https://www.littelfuse.com/assetdocs/resettable-ptc-rhef-datasheet?assetguid=5c5a29fe-1611-4d65-b4ce-dc979c44d206.

[18] Espressif Systems, *ESP32-DevKitC-32UE Development Board Datasheet*. Espressif Systems, 2023. Disponible en: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-da_datasheet_en.pdf.

[19] Omron Corporation, *D6F-P MEMS Flow Sensor Datasheet (D6F-P0010A2)*. Omron, 2021. Disponible en: https://www.mouser.mx/datasheet/2/307/en_d6f_series-1128136.pdf.

[20] DFRobot, *SEN0392 SGP40 Air Quality Sensor Module Datasheet*. DFRobot, 2022. Disponible en: https://wiki.dfrobot.com/SGP40_Air_Quality_Sensor_SKU_SEN0392.

[21] Sensirion AG, *SEN55 Environmental Node Sensor Datasheet (SEN55-SDN-T)*. Sensirion, 2023. Disponible en: https://sensirion.com/media/documents/6791EFA0/62A1F68F/Sensirion_Datasheet_Environmental_Node_SEN5x.pdf.

[22] APEM Inc., *TS6T1S02A Miniature Thumbstick (2-Axis, Hall-Effect) Datasheet*. APEM Inc., 2025. Disponible en: https://www.apem.com/idec-apem/en_US/Joysticks/Thumb-Controls/TS/p/TS6T1S02A.

[23] SparkFun Electronics, *SEN-15901 Weather Meter Kit Datasheet*. SparkFun Electronics, 2017. Disponible en: https://cdn.sparkfun.com/assets/d/1/e/0/6/DS-15901-Weather_Meter.pdf.

[24] Sensirion AG, *SHT30-DIS Digital Humidity and Temperature Sensor Datasheet*. Sensirion AG, 2022. Disponible en: https://www.sensirion.com/media/documents/213E6A3B/63A5A569/Datasheet_SHT3x_DIS.pdf.

[25] Adafruit Industries, *MCP9600 I²C Thermocouple Amplifier Breakout Board (Adafruit 4101) Datasheet*. Adafruit Industries, 2023. Disponible en: https://www.adafruit.com/product/4101.

[26] JLCPCB, "Jlcpcb: Your reliable pcb manufacturer." https://jlcpcb.com/, 2024. Accessed: 2024-07-24.

[27] MathWorks, "Thingspeak." https://www.mathworks.com/products/thingspeak.html, 2025. Accedido el 15 de mayo de 2025.

[28] "KiCad eda suite." https://www.kicad.org/, 2024. Software libre para diseño electrónico.