UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE CUENCA

CARRERA DE ELECTRÓNICA Y AUTOMATIZACIÓN

DESARROLLO DE UN SISTEMA DE MONITOREO DE LA CONCENTRACIÓN DE OXIGENACIÓN Y PRESIÓN ATMOSFÉRICA PARA UN VEHÍCULO CON SISTEMA DE ENTRETENIMIENTO BASADO EN ANDROID.

Trabajo de titulación previo a la obtención del
título de Ingeniero en Electrónica

AUTORES: PAUL SEBASTIAN PACURUCU PARRA
PAUL SEBASTIAN TINIZHAÑAY ZUMBA
TUTOR: ING. MARCELO ESTEBAN FLORES VÁZQUEZ, Mst.

Cuenca - Ecuador
2023

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Nosotros, Paul Sebastian Pacurucu Parra con documento de identificación N° 0106740640 y Paul Sebastian Tinizhañay Zumba con documento de identificación N° 0302729652; manifestamos que:

Somos los autores y responsables del presente trabajo; y, autorizamos a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Cuenca, 18 de agosto del 2023

Atentamente,

Paul Sebastian Pacurucu Parra	Paul Sebastian Tinizhañay Zumba
0106740640	0302729652

CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Nosotros, Paul Sebastian Pacurucu Parra con documento de identificación N° 0106740640 y Paul Sebastian Tinizhañay Zumba con documento de identificación N° 0302729652, expresamos nuestra voluntad y por medio del presente documento cedemos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que somos autores del Proyecto técnico: "Desarrollo de un sistema de monitoreo de la concentración de oxigenación y presión atmosférica para un vehículo con sistema de entretenimiento basado en Android.", el cual ha sido desarrollado para optar por el título de: Ingeniero en Electrónica, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribimos este documento en el momento que hacemos la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Cuenca, 18 de agosto del 2023

Atentamente,

Paul Sebastian Pacurucu Parra	Paul Sebastian Tinizhañay Zumba
0106740640	0302729652

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, Marcelo Esteban Flores Vázquez con documento de identificación N° 0102408978, docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: DESARROLLO DE UN SISTEMA DE MONITOREO DE LA CONCENTRACIÓN DE OXIGENACIÓN Y PRESIÓN ATMOSFÉRICA PARA UN VEHÍCULO CON SISTEMA DE ENTRETENIMIENTO BASADO EN ANDROID., realizado por Paul Sebastian Pacurucu Parra con documento de identificación N° 0106740640 y por Paul Sebastian Tinizhañay Zumba con documento de identificación N° 0302729652, obteniendo como resultado final el trabajo de titulación bajo la opción Proyecto técnico que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Cuenca, 18 de agosto del 2023

Atentamente,

Ing. Marcelo Esteban Flores Vázquez, Mst.
0102408978

AGRADECIMIENTOS

Quiero expresar mi más sincero agradecimiento a todas las personas que contribuyeron en este largo proceso que ha llegado a su fin. Agradecer especialmente a mi familia por ser parte fundamental en mi vida, por tener fe y esperanza en mí, ya que, sin su apoyo y aliento, completar esta meta habría sido mucho más complicada de hacer.

Agradezco también a todos mis amigos, por acompañarme todos estos años, por compartir momentos inolvidables, que siempre estarán presentes a lo largo de mi vida.

Paúl Sebastián Pacurucu Parra

Agradezco a mi amada familia por estar presente a lo largo de este camino, su apoyo ha sido mi mayor fortaleza y motivación en cada paso que he dado. A mis padres, por ser los pilares fundamentales de mi vida, siendo ustedes mi mayor fuente de inspiración y mi apoyo incondicional.

A mi hermana, con quien he compartido risas, lágrimas y alegrías, brindándome siempre palabras de aliento y ánimo en los momentos más difíciles.

Finalmente, a mis amigos, quienes estuvieron presentes a lo largo de esta travesía y han dejado grabadas en mi corazón un millón de inolvidables anécdotas.

Gracias a todos ustedes, por siempre creer en mí y alentarme a alcanzar mis sueños.

Paúl Sebastián Tinizhañay Zumba

DEDICATORIAS

A mis padres, que siempre estuvieron para mí, guiándome en cada uno de los pasos que he dado hasta ahora.

A mi hermano, por estar siempre conmigo y apoyarme en los momentos más difíciles que han surgido en los últimos años, ya que, sin su apoyo, este proceso hubiese sido mucho complido de sobrellevar.

A mis abuelos, que siempre me estuvieron animando a seguir progresando sin importar el esfuerzo que conlleve, ya que a mientras más complicado es el proceso, más satisfactorio es el resultado.

Paúl Sebastián Pacurucu Parra

Con cariño y gratitud, dedico este trabajo a Dios, mi fuente de inspiración y fortaleza para concluir este proceso. A mi familia, por ser mi apoyo incondicional a lo largo de mi vida y carrera universitaria. Y a todas las personas especiales que me acompañaron, brindando apoyo y contribuyendo a mi formación como profesional y ser humano.

Paúl Sebastián Tinizhañay Zumba

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El presente informe detalla el diseño e implementación de un proyecto técnico orientado al monitoreo de la concentración de oxigenación y la presión atmosférica mediante la tecnología Bluetooth, haciendo uso de la placa de desarrollo ESP32. Este dispositivo se encargará de la transmisión de los datos adquiridos de cada uno de los sensores hacia una aplicación Android, especialmente adaptada e integrada en el sistema de entretenimiento vehicular, permitiendo así una presentación en tiempo real e interacción optima con los datos recopilados.

El prototipo se enfoca en la medición de dos variables importantes al momento de realizar una conducción todoterreno o por carretera (presión atmosférica y concentración de O2). Para lograr esto, se ha seleccionado 2 tipos de sensores para cada variable y se emplean los protocolos de comunicación SPI e I2C específicos para cada tipo de sensor.

La aplicación constara con una interfaz de usuario intuitiva, sencilla y de fácil acceso diseñada en Flutter con el fin de brindar una experiencia fluida y agradable para los usuarios.

Adicionalmente, se desarrolló un protector plástico el cual servirá de cubierta para los componentes electrónicos empleados garantizando su ubicación adecuada dentro del compartimento del vehículo. Finalmente realizaremos pruebas de campo para evaluar el desempeño y la eficiencia del proyecto en condiciones reales.

INTRODUCCIÓN

La correcta medición de la concentración de oxígeno y la presión atmosférica en tiempo real, para los vehículos utilizados en actividades todoterreno nos permitirá garantizar tanto la seguridad como el óptimo rendimiento del vehículo durante la conducción, posibilitando la adecuada toma de decisiones, pronosticando así posibles accidentes.

El creciente uso de dispositivos de entretenimiento basados en Android para vehículos ha revolucionado la experiencia de conducción, sin embargo, a partir de todas las ventajas que estos sistemas nos brindan la falta de integración y compatibilidad con aplicaciones que nos permitan monitorear dichos valores ha limitado su utilidad y efectividad.

En el presente proyecto se abordará la creación de un aplicativo Android de monitoreo, diseñado específicamente para el sistema de entretenimiento propio del automotor. El cual nos brindara la lectura tanto de los sensores de presión atmosférica como a su vez de la concentración de oxigenación, proporcionando una visualización clara y comprensible para el conductor.

Mediante la implementación de este sistema, lo que se busca es proporcionar una herramienta eficiente y accesible que mejore significativamente la seguridad y experiencia del conductor. Posibilitando la obtención de datos precisos en tiempo real y visualizarlos en forma intuitiva. Esperando obtener un impacto positivo en la gestión eficiente del vehículo y la seguridad de los ocupantes.

ANTECEDENTES DEL PROBLEMA DE ESTUDIO

Los dispositivos para medir tanto la presión atmosférica como la concentración de oxigenación se han utilizado en diversas aplicaciones en el pasado. Estos dispositivos pueden proporcionar información valiosa a los conductores y pasajerosde vehículos todoterreno, especialmente en el ámbito de la conducción off-road, permitiéndoles tomar decisiones informadas y reducir el riesgo de accidentes.

Los conductores y pasajeros de vehículos especiales todoterreno se enfrentana una serie de retos que no existen en la conducción convencional fuera de carretera.Por ejemplo, la conducción en altitud puede provocar una reducción de la cantidad de oxígeno, como de la presión atmosférica en el vehículo, lo que puede afectar al rendimiento del motor y la seguridad las personas a bordo.

Numerosos estudios han examinado los riesgos asociados a la conducciónа gran altitud y han propuesto varias soluciones para aumentar la seguridad de los pasajeros. Sin embargo, a medida que avanzan las tecnologías, se necesitan dispositivos más precisos, con la fiabilidad más alta y sobre todo adaptables para poder monitorear la presión atmosférica y la concentración de oxígeno.

JUSTIFICACIÓN (IMPORTANCIA Y ALCANCES)

Los sistemas de entretenimiento basado en Android para automóviles, por lo general carecen de sensores que permitan medir tanto la concentración de oxigenación como de la presión atmosférica de un vehículo, normalmente se implementan en los dispositivos móviles. Como resultado, varias de estas aplicaciones son diseñadas para equipos móviles, no son compatibles con los sistemas de entretenimiento Android que contienen los vehículos.

El uso de vehículos destinados a la conducción todoterreno ha aumentado considerablemente los últimos años, por ende, ha provocado un incremento de instrumentos de monitoreo que permitan garantizar la seguridad y el rendimiento óptimo del automotor. Al realizar el uso de estas aplicaciones en los dispositivos móviles, por lo general no son muy recomendados, ya que el movimiento y vibracióndel vehículo afectan considerablemente el sistema de sujeción del teléfono móvil. Actualmente, en el mercado existen dispositivos para medir la presión atmosférica y la cantidad de oxígeno para vehículos todoterreno, desafortunadamente muchos de ellos no se adaptan a las necesidades específicas de los conductores, siendo incluso difícilesde manipular.

Por lo tanto, existe la necesidad de un sistema accesible e integrado al sistemade entretenimiento propio del vehículo que mida con precisión estos valores en tiempo real y proporcione una visualización posterior al conductor de una manera clara y comprensible, ajustando así su manejo en consecuencia. Evitando daños en el motor y preservando la seguridad de los ocupantes. El sistema contendrá una interfaz de usuario fácil de usar, esta permitirá a los conductores acceder de una manera rápidaa esta información, mejorando la seguridad en la conducción y la experiencia generaldel usuario.

En conclusión, el desarrollo de un sistema basado en Android diseñado específicamente para el sistema de entretenimiento propio del automotor, permitirámonitorizar los parámetros de la concentración de oxígeno y la presión atmosférica, permitiendo así proporcionar una variedad de beneficios en términos de garantía de seguridad y gestión eficiente del vehículo.

OBJETIVOS

OBJETIVO GENERAL

- Desarrollar un sistema de monitoreo de la concentración de oxígeno y presión atmosférica de un vehículo con sistema de entretenimiento basado en Android

OBJETIVOS ESPECÍFICO

- Realizar un estudio de los sensores de oxígeno y presión atmosférica disponibles, a través de la revisión de los manuales de proveedores de sensores y bibliografía especializada, a fin de seleccionar el dispositivo más adecuado.

- Diseñar y construir un dispositivo electrónico que permita a la aplicación móvil leer los datos proporcionados, los sensores de oxígeno y presión atmosférica, considerando requerimientos de inmunidad a la interferencia electromagnética presente en el vehículo, a fin de obtener un dispositivo fiable.

- Desarrollar una aplicación móvil a ser instalada en un sistema de entretenimiento basado en el sistema operativo Android, que permita establecer la comunicación con el hardware, a desarrollar y visualizar la información proporcionada, mediante el uso de herramientas informáticas especializadas, en áreas de sentar las bases para la implementación.

- Realizar un análisis de los límites de operación mediante la realización de pruebas de funcionamiento a fin de determinar el desempeño y posibles mejoras.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En este capítulo se dará a conocer la fundamentación teórica, la cual constará de toda la información necesaria y requerida. Dándonos así las bases necesarias para cumplir a totalidad y de manera eficiente este proyecto.

1.1 FUNDAMENTOS TEÓRICOS

1.1.1 SENSORES DE OXÍGENO Y PRESIÓN ATMOSFÉRICA

Los sensores diseñados para monitorear los niveles de oxígeno y presión atmosférica son dispositivos capaces de desempeñar un papel fundamental en una amplia gama de aplicaciones en diversas situaciones y entornos.

Los dos tipos de sensores se basan en tecnologías y principios de funcionamiento diferentes. Los sensores de oxígeno normalmente utilizan técnicas como la electroquímica, la óptica o la galvanometría, mientras que los sensores de presión atmosférica emplean tecnologías como la piezoresistividad o la capacitancia para detectar los cambios de presión.

1.1.1.1 SENSOR DE OXÍGENO

El sensor de oxígeno, se le conoce también como sensor de oxígeno disuelto o sensor de oxígeno ambiental, es utilizado normalmente en aplicaciones relacionadas con la calidad del aire, la monitorización de la salud, el control de procesos industriales también en la investigación científica. Lo que hace este sensor

es detectar y medir la concentración de oxígeno existente en un medio, ya sea liquido o gas. De esta forma nos permite obtener información exacta sobre la cantidad de oxígeno disponible en el entorno, lo que nos resulta importante en las diferentes áreas mencionadas anteriormente. A continuación, se presentarán algunos tipos de sensores.

- Sensor de oxígeno óptico
- Sensor de oxígeno galvánico
- Sensor de oxígeno electroquímico

En este proyecto se utilizará el sensor de oxígeno electroquímico y se lo detallará brevemente. Estos son utilizados con mayor frecuencia en el método de difusión, que implica la entrada de gas ambiental a través de un orificio en la cabeza de la celda. Algunas personas suministran muestras de aire o gas al sensor usando una bomba. Para evitar que entren líquidos en la celda, se coloca una membrana de PTFE sobre el orificio. [1]

Puede ser específico para un gas o vapor en particular en el rango de partes por millón. El tipo de sensor, la concentración del gas y su grado lo afectan. No vulnerable a ser envenenado por otros gases.[1]

Figura 1.1 *Sensor Electroquímico*
Fuente: *Advanced micro Instruments*

Uno de sus principales problemas es la mayor sensibilidad que se produce cuando un gas distinto del que se está controlando o detectando tiene el potencial de afectar la lectura dada. Esto hace que el electrodo interno del sensor reaccione incluso si el gas objetivo no está realmente presente en el ambiente.[1]

1.1.1.2 SENSOR DE PRESIÓN ATMOSFÉRICA

El sensor de presión atmosférica también es conocido como barómetro, este sensor tiene como principal objetivo medir la presión del aire. Cabe mencionar que la presión atmosférica es la fuerza ejercida por una columna de aire en un punto especifico y se encuentra directamente relacionada con las condiciones climáticas. Son utilizados normalmente en sistemas de navegación, aplicaciones meteorológicas, y sobre todo en el monitoreo y control de altitud en los aviones. A continuación, se presentarán algunos tiposde sensores.

- Sensor de presión de capacitancia
- Sensor de presión de piezo-eléctrico
- Sensor de presión piezorresistivo.

En los sensores piezorresistivos, su elemento se basa en un puente de Wheatstone basado en silicio. Este se extiende de una manera mínimamente baja cuando la presión cambia la resistencia eléctrica. Este efecto es conocido como efecto piezorresistivo.[2]

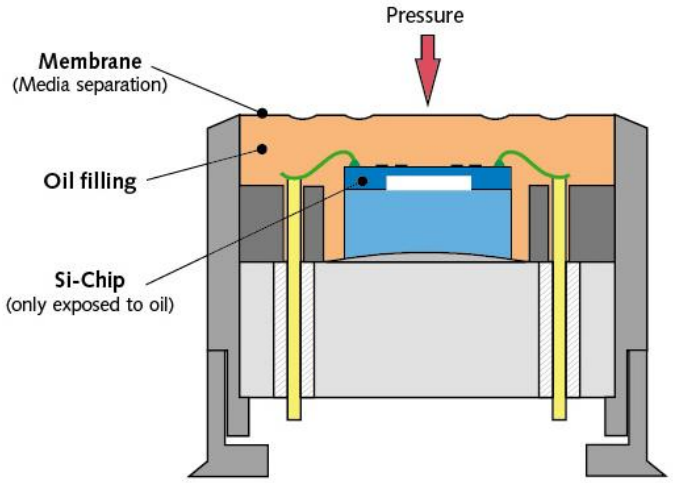

Figura 2.2 *Puente de Wheatstone*
Fuente: *Kistler*

Su funcionamiento es muy sencillo; un chip de silicona en el dispositivo mide la presión a través de una membrana de silicona imperceptible y aceite de silicona. El chip recibe energía a través de un mecanismo de alimentación de aleación de aluminio y vidrio. La liberación del sello de presión se muestra en milivoltios (mV). La señal de presión aumenta a una señal salina V o mA correspondiente a medida que la temperatura la compensa.[2]

Dependiendo del propósito del sensor, los sensores de presión piezorresistivos miden la presión en una variedad de puntos diferentes, incluido el valor absoluto en relación con el vacío, la presión ambiental y diferencialmente en relación con otra presión. Los sensores de presión absoluta, relativa (manómetro) y diferencial pueden ser adecuados según la aplicación.[2]

- **Presión absoluta**

Dado que la presión absoluta toma como referencia la presión absoluta cero, es imposible que tenga un valor negativo.[3]

- **Presión de Manómetro**

La presión manométrica mide la diferencia entre un sistema a la presión atmosférica circundante. Puede ser positiva o negativa, dependiendo de que si el sistema es de vacío.[3]

Los sensores de los manómetros, que se asemejan a un diafragma, se desvían debido a la variación de presión. Para que los sensores de presión utilicen la presión atmosférica como punto de referencia, se requiere ventilación. La presión métrica es independiente de la presión atmosférica. No cambiaría con la altitud porque la presión manométrica siempre tiene en cuenta la presión atmosférica local. Por lo tanto, es apropiado para medir la presión en tubos y sitios donde sea superior a la atmósfera.[3]

- **Presión diferencial**

La presión diferencial consiste en la diferencia entre dos presiones aplicadas. La mayoría de las lecturas de presión son técnicamente diferenciales, esto debido a que la presión se mide con relación al vacío absoluto y a la presión atmosférica.[3]

- **PRESIÓN BAROMÉTRICA**

Es la cantidad de presión que se mide en cualquier punto por encima del nivel del océano.[4]

El barómetro se utiliza para medir los valores de presión barométrica. Evangelista Torricelli creó este dispositivo en 1643. Lo hizo introduciendo mercurio en un tubo que estaba abierto por un extremo y comprobando que el nivel de mercurio bajaba a 760 mm de altura sin importar el diámetro o la forma del tubo. Esto llevó a la

conclusión de que había una fuerza opuesta que impedía que el mercurio cayera por debajo de ese valor.[4]

En el pronóstico del tiempo, se utiliza en la predicción de los cambios climáticos en una región en particular. En ingeniería, determina los puntos de ebullición y fusión de los materiales. Se utiliza en pozos y piezómetros de columna.[5]

La presión barométrica se utiliza en sistemas como los procesos de recubrimiento por plasma atmosférico. Utilizando las presiones atmosféricas, el equipo de plasma produce una capa de nano revestimiento que evita que la humedad y otras impurezas entren en contacto con el sustrato. [5]

Además, la presión barométrica determinará la tasa de corrosión atmosférica, que es causada por el clima normal fluctuante. Esto prepara a los diseñadores e ingenieros con información vital necesaria para desarrollar soluciones resistentes a la corrosión.[5]

1.1.2 SISTEMA OPERATIVO ANDROID

El sistema operativo Android, diseñado principalmente para equipos móviles, en la sociedad actual este sistema se puede instalar no solo en dichos equipos, sino que también en varios otros dispositivos, como tabletas, televisiones, discos multimedia, etc. Se encuentra basado en sistema operativo denominado Linux, ya que se conoce como el núcleo del actual sistema operativo abierto, de multiplataforma y gratis. Nos permite realizar aplicaciones empleando una variedad de Java llamada DalviK la cual nos proporciona la interfaz que se necesita para facilitar el desarrollo de las aplicaciones y su acceso a las distintas funciones del equipo. Una de las características más relevantes se encuentra basada en que es totalmente libre.[6]

En noviembre de 2007 se realizó el primer lanzamiento de Android, a la par con SDK (Software Development Kit), para la gran diversidad de programadores a nivel global puedan realizar sus aplicaciones en este nuevo sistema operativo.[6]

Como se mencionó anteriormente, el corazón de Android es Linux, el cual nos permitirá realizar la gestión de la red Wi-Fi, la pantalla, la cámara, el teclado, etc. Nose tendrá un acceso directo a esta, sino este nivel se utilizarán una gran variedad de librerías que se encuentran en niveles superiores y nos separan del hardware.

Además,la programación del equipo se la realiza utilizando el Framework de dicha aplicación,en donde se gestionan el ciclo de vida y sus recursos, y en la última capa se encuentran las aplicaciones de usuario.[7]

Figura 3.3 *Arquitectura Android*
Fuente: *KeepCoding*

Se debe tener en cuenta algunos conceptos claves antes de empezar a realizar la aplicación, como por ejemplo los servicios, las Actividades, Intens, y proveedores de contenido.

- ***Actividades***

Estas correspondes a las pantallas, es decir, se tendrá la misma cantidad de actividades como pantallas tenga la aplicación creada.

Cada actividad se responsabiliza de mantenerse en su estado, para que de esta forma se puedan integrar en el ciclo de vida de esta. Las interfaces se pueden crear de dos formas, la primera es usando el código Java, y la segunda forma es usar un fichero XML para describirla como una página HTML. Siendo la más factible y sencilla.[7]

- ***Intens***

Los Intens son mecanismos para describir acciones, como por ejemplo enviar un e-mail, o tomar una foto. Este es un mecanismo flexible, ya que, mediante él, se puede crear una actividad para envío de SMS y registrarla de forma que corresponda al Inten adecuado.[7]

- ***Servicios***

Los Servicios son tareas que se ejecutan en segundo plano. Por ejemplo, hacemos sonar un MP3, y a su vez, deseamos poder seguir utilizando el teléfono. Mediante otro servicio, se puede realizar esta acción y podemos usar otras aplicaciones con normalidad.[7]

- ***Proveedores de Contenido***

Los Proveedores almacenan datos compartidos por la mayoría de las aplicaciones. Para acceder a estos datos se lo hace mediante la API para cada proveedor de contenido. [7]

- ***Tiempo de ejecución de Android***

Cada aplicación utiliza sus propias instancias de Android Runtime (ART) para llevar a cabo sus procesos. El ART está diseñado para ejecutar varias máquinas virtuales en dispositivos con poca memoria, un tipo de código basado en bytes creado exclusivamente para Android, para ocupar la menor cantidad de memoria.[8]

Algunas de las funciones clave incluyen la compilación justo a tiempo (JIT) y antes de tiempo (AOT), así como la recolección optimizada de elementos de desecho (GC).[8]

- ***Bibliotecas C/C++ nativas***

Los servicios básicos del sistema operativo Android, como HAL y ART, se construyen utilizando código nativo el cual requiere bibliotecas C y C++ nativas. La plataforma Android ofrece API de banco de trabajo de Java.[8]

1.1.3 PROTOCOLO DE COMUNICACIÓN SPI

SPI es un protocolo de sincronización que funciona en modo dúplex completo para recibir y enviar datos, lo que permite que dos dispositivos se comuniquen al mismo tiempo utilizando canales o líneas diferentes en el mismo cable.[9]

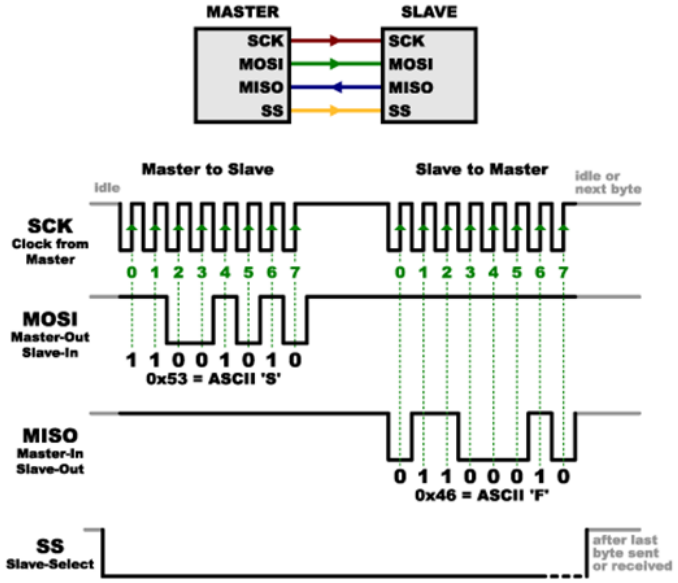

Figura 4.4 *Estructura general y funcionamiento del protocolo SPI.*
Fuente: *Prometec*

Las vías lógicas encargadas del proceso son:

- **MOSI** (Master Out Slave In): Se utiliza principalmente para llevar los bits del maestro al esclavo.
- **MISO** (Master In Slave Out): Se utiliza para llevar los bits que del esclavo hacia el maestro.
- **CLK** (Clock): Envía la señal de clock.
- **SS o CS** (Slave Select): Habilita y selecciona un esclavo.

Protocolo de comunicación del bus SPI

Se habilita el chip y se carga un byte en el buffer de salida. La señal de reloj envía los bits de datos a través de MOSI, mientras que el receptor captura los bits entrantes desde MISO y los almacena en su buffer. Se repite el proceso 8 veces para transmitir un byte completo. La comunicación es full-dúplex, lo que significa que el Master puede enviar y recibir datos al mismo tiempo. La línea SS se usa para despertar y desactivar el dispositivo esclavo seleccionado. [9]

1.1.4 PROTOCOLO DE COMUNICACIÓN I2C

I2C es un protocolo de comunicación y un puerto serie que define el protocolo de transferencia de datos y las conexiones físicas requeridas para la transmisión de bits entre dos dispositivos digitales. El puerto I2C se compone de dos líneas de comunicación llamadas SDA (Serial Data) y SCL (Serial Clock). Este protocolo permite establecer conexiones entre hasta 127 dispositivos, y todos comparten las

mismas líneas de SDA y SCL. Las transferencias de datos pueden realizarse a velocidades de 100, 400 y 1000 kbit/s, ofreciendo flexibilidad en la elección de la velocidad según las necesidades de la aplicación. [10]

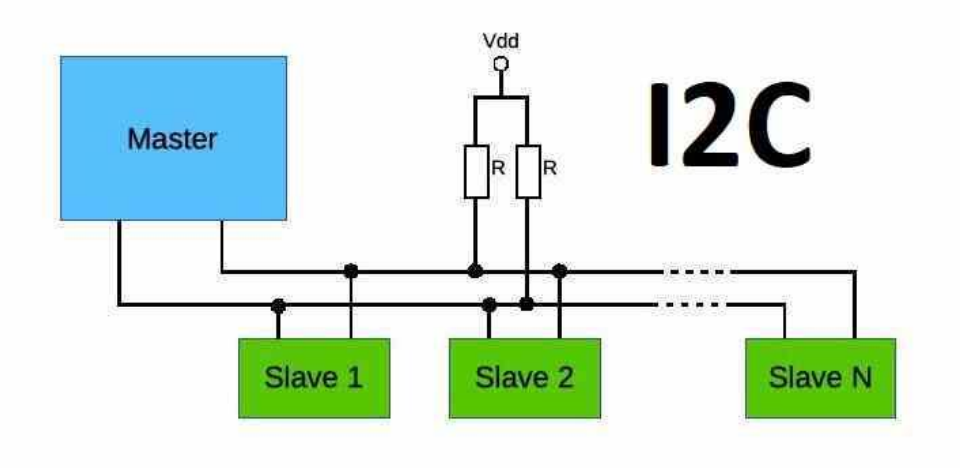

Figura 5.5 *Estructura general del protocolo I2C*

Fuente: Signal Gen

Descripción de las señales

- **SCL** (System Clock): Pulsos de reloj para sincronizar el sistema.
- **SDA** (System Data): Transfiere los datos entre los dispositivos.
- **GND** (Masa): Conexión común de tierra para una comunicación estable.

Protocolo de comunicación del bus I2C

Solo los dispositivos maestros pueden iniciar la comunicación en el bus I2C. La condición de inicio ocurre cuando el maestro pone en bajo la línea de datos (SDA) mientras mantiene en alto la línea de reloj (SCL). Después de la condición de inicio, el maestro transmite un byte que contiene la dirección del dispositivo y un bit que indica si será una operación de lectura o escritura. Si es escritura, se envían los datos al dispositivo esclavo, recibiendo señales de reconocimiento (ACK) en respuesta. La conexión concluye cuando se transmiten todos los datos. El maestro puede liberar el bus generando una condición de parada (stop). [10]

1.1.5 COMPUTADOR DE PLACA UNICA ESP32

La computadora de placa única ESP32 está diseñada con una arquitectura integrada con conectividad inalámbrica Wi-fi y Bluetooh. Evaluando su comportamiento térmico, se identificó varios elementos que afectan su rendimiento y su estabilidad operativa.

El diseño de su PCB influye un papel crucial en el manejo térmico de la ESP32. Considerando los trazos de señales y planos de potencia adecuados para minimizar la resistencia, lo cual ayudara a evitar acumulaciones de calor en ciertas áreas.

Figura 6.6 *Computador de placa unica ESP32*
Fuente: *ElectroStore.*

La ESP32 contiene mecanismos internos de protección térmica para salvaguardar el funcionamiento. Considerando la implementación de umbrales de alarma térmica se puede notificar al sistema sobre condiciones críticas de temperatura.

- ***Placa de desarrollo ESP32 para control industrial***

Cuando se diseña una pantalla que se usará en producción o cuando se coloca en una pantalla "superior", la placa de desarrollo ESP32 es de gran utilidad. Estos rótulos van desde rótulos de "inicio" muy básicos hasta rótulos sofisticados con procesadores secundarios y LCD.[11]

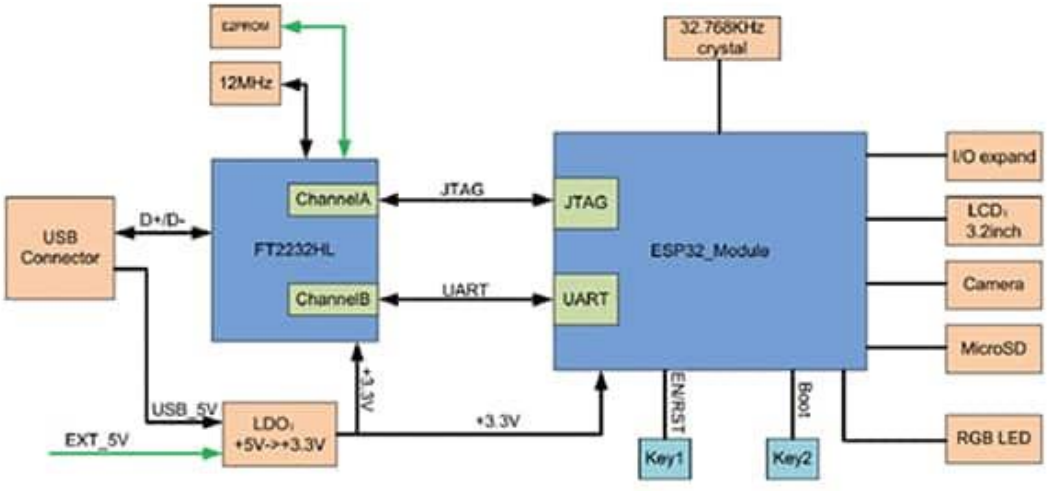

Figura 7.7 *ESP-WROVER-KIT*
Fuente: *Espressif Systems*

1.1.6 JEEP GRAND CHEROKEE WJ

Un vehículo utilitario de tipo SUV diseñado para un rendimiento confiable en una variedad de terrenos y condiciones climáticas. Ideal para la conducción todoterreno. Presenta un estilo robusto y a la vez elegante, sus líneas son aerodinámicas, permitiendo así una mejor eficiencia en combustible y una menor resistencia al viento. El Jeep Grand Cherokee WJ consta de un peso alrededorde 1.970 kg. debido a su robusta estructura y a varias características de seguridad.Llega equipado con un motor V8 de 4.7 litros, generando así 235 caballos de fuerza. Además, posee tracción en todas sus ruedas, lo que nos permite un control mayor de las mismas en terrenos irregulares.[12]

Figura 8.8 *Marco UniFrame Jeep*
Fuente: *Jeep Specs*

CAPÍTULO 2: MARCO METODOLÓGICO

En este capítulo se presentará la metodología aplicada para la elaboración del proyecto, este se encuentra compuesto de 4 etapas las cuales corresponden a: investigación, diseños (contendrá la etapa 2 y la etapa 3) e implementación del sistema de visualización. En cada etapa se describirá como se llevó a cabo el proceso para cumplir con el objetivo relacionado a cada una de las mismas.

2.1 ETAPA 1

El proyecto iniciara con una revisión e investigación del estado del arte relacionado con el sistema de censado de oxígeno y presión atmosférica.

Para esto se llevó a cabo la investigación sobre los diferentes tipos de sensores que se pueden llegan a utilizar según las características establecidas del proyecto.

Los sensores de oxígeno detectan la concentración de oxígeno en un medio establecido. Para lo cual existen varios sensores, con diferentes funcionamientos y aplicaciones. Para lo cual tenemos los siguientes ejemplos.

- Sensor electroquímico.
- Sensor óptico.
- Sensor galvánico.

Los sensores de presión atmosférica son dispositivos que miden la presión ejercida en un medio. Son utilizados en diferentes tecnologías para medir la presión, como, por ejemplo.

- Sensor piezo-eléctricos.
- Sensor de capacitancia.
- Sensor de membrana.

Teniendo en cuenta estos parámetros se llegó a la conclusión que los óptimos para este proyecto son: SENSOR ELECTROQUIMICO O2 SEN0465 y el SENSOR DE PRESION ATMOSFERICA BMP280.

2.1.1 SENSOR ELECTROQUIMICO O2 SEN0465

La serie de sensores de gas está diseñada para la detección de una variedad de gases inflamables, combustibles y tóxicos en el entorno. Estos sensores vienen precalibrados, lo que permite medir con rapidez y precisión la concentración de gases específicos. Ofrecen múltiples modos de salida, incluyendo analógica, I2C, UART y señales de alarma digital, lo que brinda una flexibilidad considerable en diversas aplicaciones. Estas sondas utilizan principios electroquímicos, lo que les confiere una gran estabilidad y sensibilidad, y tienen una vida útil de hasta dos años.[13]

La interfaz Gravity utilizada en estos sensores es sencilla y fácil de usar, lo que permite la construcción sin complicaciones de varios detectores de concentración de gas. La amplia gama de aplicaciones de esta serie de sensores incluye la producción segura, la fabricación industrial y la protección del medio ambiente. Por ende, es especialmente adecuada para entornos como minas de carbón, industrias químicas, laboratorios químicos y gestión medioambiental.[13]

Figura 2.11 *SENSOR SEN0465*
Fuente: *DFROBOT*

- **PRECAUCIONES DE USO**

Para este sensor existen ciertas advertencias con respecto al uso del dispositivo. Mencionaremos algunos a continuación.

Queda terminantemente prohibido abrir la membrana blanca impermeable y transpirable del sensor del módulo, de lo contrario se considerará un daño artificial. Al igual que está prohibido enchufar o desenchufar la sonda con la alimentación

conectada. También se encuentra prohibido soldar directamente las patillas del módulo, pero se pueden soldar las tomas de estas.[13]

El módulo debe evitar el contacto con disolventes orgánicos, pinturas, productos farmacéuticos, aceites y gases de alta concentración, además no debe someterse a golpes o vibraciones excesivos. [13]

El módulo debe calentarse durante más de 5 minutos cuando se enciende por primera vez. Se recomienda calentarlo durante más de 24 horas si no se ha utilizado durante mucho tiempo. No se debe instalar el módulo en entornos con fuerte convección de aire. Ni se debe dejar en gas orgánico de alta concentración durante mucho tiempo.[13]

Los datos devueltos por el puerto serie del módulo son el valor de concentración en tiempo real en el entorno actual.

- **ESPECIFICACIONES**

  - Detección de gas: NH3, O3, CO, H2S, SO2, NO2, O2, H2, PH3, HF, HCL, CL2.
  - Voltaje de trabajo: 3.3 – 5.5VDC.
  - Corriente: < 5mA
  - Señal de salida: I2C, salida UART.
  - Detección de error: ± 10% del voltaje de salida o el 5% del total de la escala.
  - Temperatura: -20 – 50°C.
  - Humedad: 5 -90%RH.
  - Vida útil: > 2 años
  - Tamaño placa: 37 x 32mm.

2.1.2 SENSOR DE PRESIÓN ATMOSFÉRICA BMP280

La función de este sensor es medir la altura en relación con el nivel del mar, se basa en la relación entre la altura y la presión.[14]

El rango del BMP280 es de 300 a 1100 hPa. Basado en la tecnología piezorresistiva de Bosch, con alta robustez EMC, con estabilidad a largo plazo y una alta precisión. Ha sido diseñado para conectarse a un microcontrolador mediante SPI o I2C. En comparación con el sensor BMP180, el BMP280 ofrece una serie de mejoras,

que incluyen: Mejor precisión, mejores filtros digitales, mejor resolución de temperatura y presión y un consumo bajo de energía.[14]

Este sensor se utiliza con frecuencia en sistemas de piloto automático para drones ya que se puede utilizar para calcular la altura con gran precisión.[14]

Figura 22.2 *Sensor de presión BMP280*
Fuente: *Naylamp*

2.2 ETAPA 2

En esta etapa se realizará el diseño del circuito electrónico que se apegue a las necesidades del proyecto y se utilizará la tecnológica adecuada en función de lo realizado en la etapa 1.

Para poder realizar las conexiones de maneras eficientes se debe conocer las características y especificaciones de los dispositivos que se van a utilizar. Para este caso se va a utilizar el módulo ESP32, ya que incluye un módulo Bluetooth para el envío y recepción de datos. Además, se utilizará un módulo DC-DC convert, este módulo nos ayudará a convertir los 12v que entrega el vehículo a los 5v que necesita el ESP32.

2.2.1 COMPUTADOR DE PLACA ÚNICA ESP32

La línea ESP32 es un microcontrolador con bajo consumo de energía a bajo costo y con Wi-Fi dual integrado y Bluetooth, Su consumo bajo de energía, y los numerosos entornos de desarrollo y bibliotecas lo convierten en la opción ideal para los desarrolladores.[15]

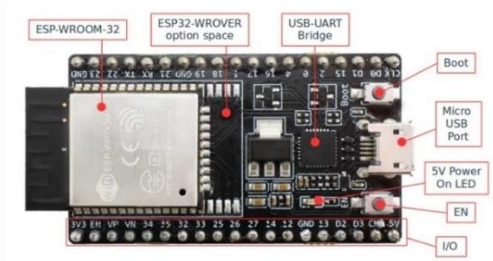

Figura 2.33 FI*Computador de placa unica ESP32*
Fuente: *Mouser Electronics*

Tabla 2.1 Características del Módulo ESP32

Características técnicas ESP32	
Voltaje de operación	3 - 6v
CPU	Microprocesador LX6 de 32 bits, doble núcleo
Memoria	520 KB SRAM
WiFi	802.11 b/g/n
Bluetooth	EDR y BLE, v4.2 BR
ADC SAR	De 12 hasta 18 canales
DAC	2 de 8 bits
GPIO	10 sensores táctiles, detección capacitiva
COMUNICACIÓN SPI	4
INTERFACES I2S	2
INTERFACES I2C	2
COMUNICACIÓN UART	3
CONTROLADOR HOST	SDIO/MMC/ CE-ATA/ eMMC/SD
CONTROLADOR ESCLAVO	SPI/SDIO
INTERFAZ ETHERNET	MAC con DMA, IEEE 1588
INFRARROJO	TX/RX, 8 CANALES
PWM	16 CANALES
SEGURIDAD	ESTÁNDAR IEEE 802.11
OTP	768 para clientes y 1024 bits normal
HARDWARE CRIPTOGRAFICO	, SHA-2,AES, RSA, ECC,RNG

2.3 ETAPA 3

En esta etapa se realizará el diseño de una app móvil mediante el sistema de Android, para esto se efectuará la adquisición de dispositivos establecidos en la etapa 2.

2.3.1 DESARROLLO DE APLICACIÓN MOVIL

Flutter es un marco que nos permite la creación de proyectos. Es gratuito, fue creado por Google en mayo de 2017 y es de código abierto.[16]

Nos permite la creación de una app móvil con una sola pieza de código. El cual es capaz de usar un código base para crear dos aplicaciones diferentes (para Android y iOS). Este es quizás el principal beneficio de Flutter y lo que lo convierte en elemento extremadamente valioso.[16]

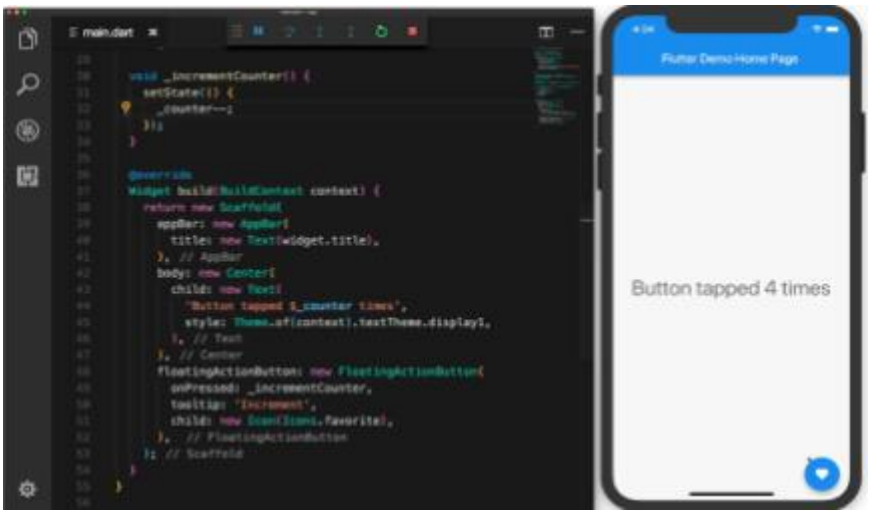

Figura 2.4 4 *Interfaz Flutter*
Fuente: *Flutter*

2.3.2 IDE ARDUINO INTERFAZ DE DESARROLLO DE LA COMUNICACIÓN DEL COMPUTADOR DE PLACA ÚNICA ESP32

El entorno de desarrollo de Arduino IDE es una herramienta que permite programar de forma intuitiva. Su más grande potencial se encuentra en su compatibilidad, a que nos permite programar en la mayoría de los dispositivos existentes en el actual mercado.[17]

El software de Arduino cuenta con varias secciones:

- **Barra de menú:** se encuentran la variedad de menús, las cuales nos permiten acceder a las diferentes acciones disponibles en el entorno.
- **Barra de acceso directo:** se encuentra las acciones de cargar y compilar el código.
- **Área de pestañas**: correspondientes al archivo de proyecto.
- **Área de edición:** se genera el código
- **Área de estado y salida**: se muestran las notificaciones del proyecto.

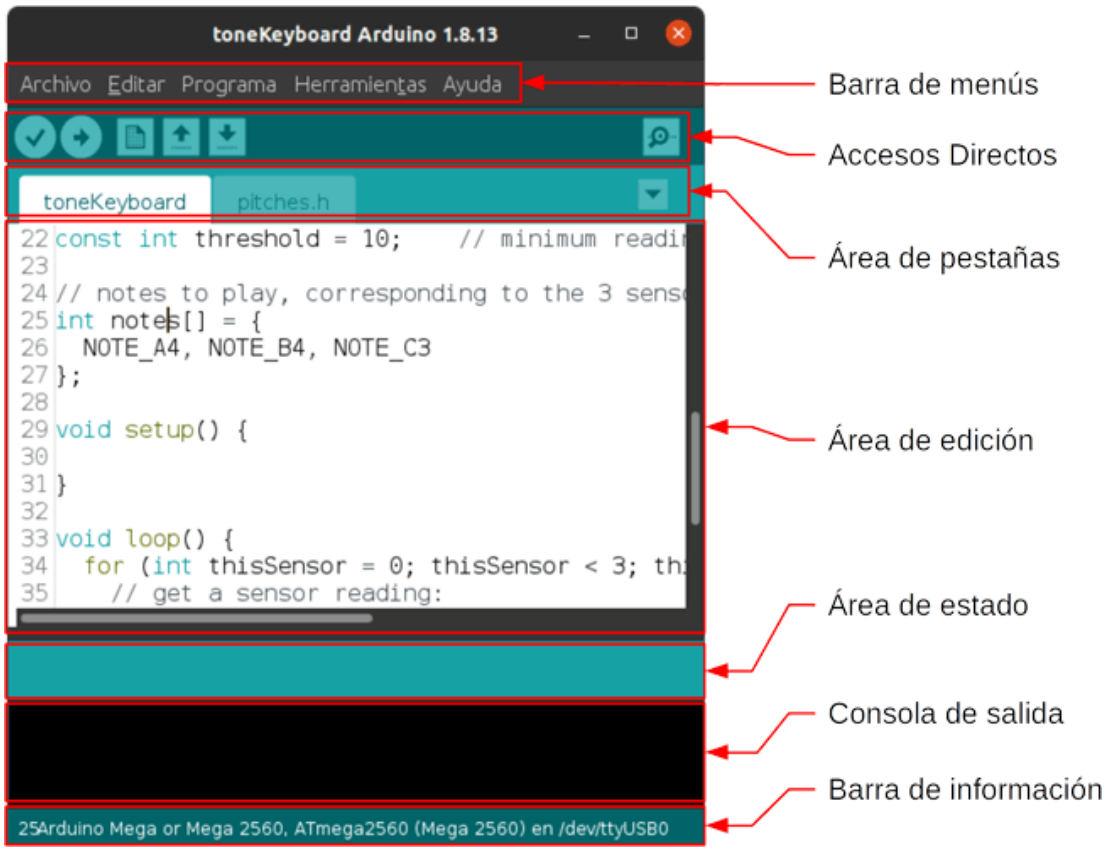

Figura 2.55 *Interfaz Arduino*
Fuente: *Arduino*

2.3.3 DISEÑO PROTECCION DE DISPOSITIVOS

La solución de software de modelado 3D más conocida que ofrece Autodesk es AutoCAD. Fusion 360 es una solución integral al proporcionar herramientas de CAD, así como herramientas CAM y CAE. Debido a que su objetivo es facilitar el desarrollo de productos, es un software más fácil de usar que AutoCAD para aplicaciones de fabricación aditiva.[18]

Fusion 360 se basa en la nube e incorpora herramientas CAM, CAE Y CAD para combinar ingeniería, fabricación y diseño en una única plataforma. Fundamentalmente, al modelar con Fusion 360, no es necesario renunciar al entorno de diseño. permite la creación de modelos 3D técnicos y mecánicos. Los usuarios se quejan con frecuencia de que el software está limitado en términos de modelado orgánico porque no proporciona muchas herramientas para el encubrimiento.[18]

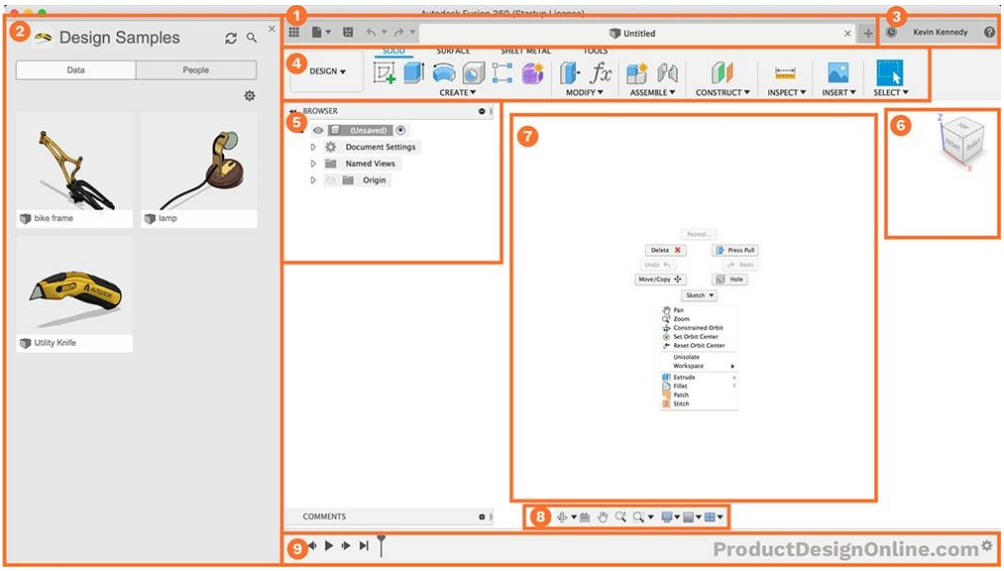

Figura 2.66 *Interfaz Autodesk Fusion 360*
Fuente: *Fusion 360*

2.4 ETAPA 4

En esta etapa se efectuarán las diferentes pruebas de funcionamiento y cobertura del sistema implementado, lo que hará que verifiquemos su correcto funcionamiento.

2.4.1 LECTURA DE DATOS DE LOS SENSORES MEDIANTE LA PLATAFORMA IDE ARDUINO

Comprobamos los datos registrados por los sensores, para esto la data se muestra en el monitor serial.

Figura 2.77 *Lectura de datos monitor serial.*
Fuente: *Autoria*

2.4.2 COMPROBACION DE REGISTRO DE DATOS DE LOS SENSORES MEDIANTE COMUNICACIÓN BLUETOOTH

Se corroboro la correcta recepción de datos en la APP diseñada, además se presenta sus valores en tiempo real, en su respectiva área de presentación.

Como podemos verificar los datos obtenidos con un pequeño porcentaje de error corresponde a los entregados por la web.

Figura 2.88 *Interfaz APP monitoreo*
Fuente: *Autoría*

CAPÍTULO 3: IMPLEMENTACIÓN Y ANÁLISIS DE RESULTADOS

En este capítulo se dará a conocer el proceso detallado de la implementación del proyecto. Dando a conocer los pasos específicos realizados para cumplir con los objetivos planteados en los capítulos anteriores.

3.1 DISEÑO DE LA APLICACIÓN ANDROID

Para la generación de la aplicación Android se va a utilizar la herramienta de programación Visual Studio Code, a la cual le agregamos las extensiones de Flutter, que nos será de gran ayuda para poder realizar la programación en Android.

Para empezar con el desarrollo, primero se crea un nuevo proyecto tipo Flutter, en el cual en la carpeta definida como pubspec.yaml agregamos las librerías correspondientes para el desarrollo de la app. En la sección de dependencias, se agrega la que nos será de ayuda para la comunicación bluetooth, y la otra syncfusion, la cual será la encargada de ayudarnos con la visualización.

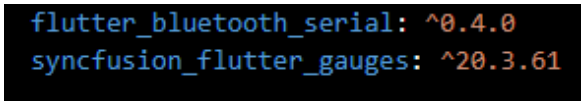

Figura 3.1 Librerías a utilizar
Fuente: VisualStudio

Se ha implementado una variable para definir la paleta de colores global de los widgets, así como para la visualización de los datos recibidos.

Figura 3.2 Creación variable color
Fuente: VisualStudio

En el archivo principal del programa, importamos todas las librerías necesarias. Luego, procedemos a inicializar los widgets y finalmente, ejecutamos la aplicación.

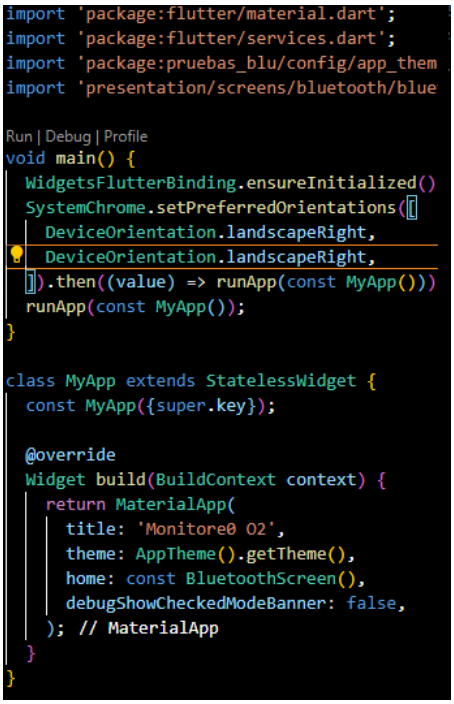

Figura 3.3 Creación main
Fuente: VisualStudio

Definimos las variables requeridas para la comunicación bluetooth, al igual que las variables del proyecto.

```dart
// Define some variables, which will be required later
List<BluetoothDevice> _devicesList = [];
BluetoothDevice? _device;
bool _connected = false;
bool _isButtonUnavailable = false;

// Bluetooth device
// To track whether the device is still connected to Bluetooth
bool get isConnected => connection != null && connection!.isConnected;
bool isDisconnecting = false;

//! Variables del proyecto
double presion = 0.00;
double altitud = 0.00;
double concentracion = 0.0;

@override
void initState() {
  super.initState();
  // Get current state
  FlutterBluetoothSerial.instance.state.then((state) {
    setState(() {
      _bluetoothState = state;
    });
  });
  _deviceState = 0; // neutral
  // If the bluetooth of the device is not enabled,
  // then request permission to turn on bluetooth
  // as the app starts up
  enableBluetooth();

```

Figura 3.4 Variables conexión bluetooth
Fuente: VisualStudio

Proceso de recepción de datos. Para esto se compararán los datos cabecera enviados del módulo ESP32, y dependiendo si la comparación es falsa o verdadera, se mostrarán los datos en los diferentes widgets.

```dart
// Metodo de conexión bluetooth
void _connect() async {
  setState(() {
    _isButtonUnavailable = true;
  });
  if (_device == null) {
    show('Seleccione un dispositivo', Colors.red);
    _isButtonUnavailable = false;
  } else {
    if (!isConnected) {
      await BluetoothConnection.toAddress(_device!.address)
          // ignore: no_leading_underscores_for_local_identifiers
          .then((_connection) {
        log('Connected to the device');
        connection = _connection;
        setState(() {
          _connected = true;
        });
        //**LISTEN DATA
        connection!.input!.listen((Uint8List data) {
          //TODO 1  Se recibe la data del bluetoth
          final text = ascii.decode(data);
          if (text.contains('Presion')) {
            final pressure = text.substring(8);
            // log('Presion: $pressure');
            presion = double.parse(pressure);
            setState(() {});
            // log(presion.toString());
          }
          if (text.contains('Altitud')) {
            final altitude = text.substring(8);
            altitud = double.parse(altitude);
            setState(() {});
          }
          if (text.contains('Concentracion')) {
            final concentracionn = text.substring(14);
            // log('Concentracion: $concentracionn');
            concentracion = double.parse(concentracionn);
            setState(() {});
          }
        }).onDone(() {
          if (isDisconnecting) {
            log('Disconnecting locally!');
          } else {
            log('Disconnected remotely!');
          }
          if (mounted) {
            setState(() {});
          }
        });
      }).catchError((error) {
        log('Cannot connect, exception occurred');
        log(error);
      });
      show('Dispositivo connectado', Colors.green);
      setState(() => _isButtonUnavailable = false);
    }
  }
}
```

Figura 3.5 Recepción de datos
Fuente: VisualStudio

3.2 DESARROLLO DEL PROGRAMA DE LECTURA DE SENSORES

Para habilitar el funcionamiento del ESP32 dentro del entorno de desarrollo (IDE) de Arduino, debemos incorporar una dirección específica para asegurar su completa compatibilidad.

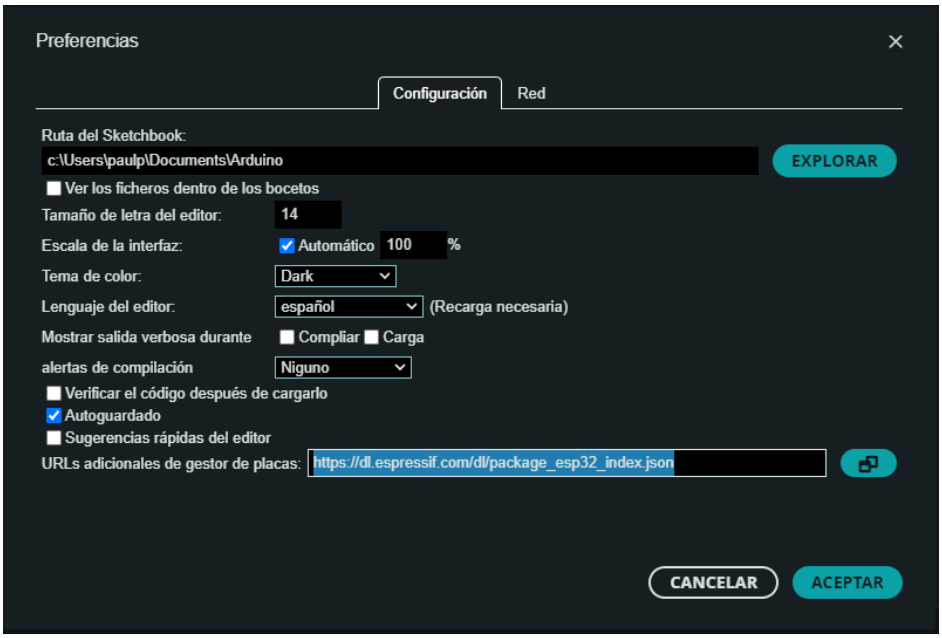

Figura 3.6 Extensión ESP32 para IDE Arduino
Fuente: IDE Arduino

Una vez a que se ha realizado la adición de la extensión correspondiente, procedemos a escoger el dispositivo que se utilizara. En este caso particular, el proyecto emplea el módulo ESP32 Dev Module como la placa de desarrollo seleccionada.

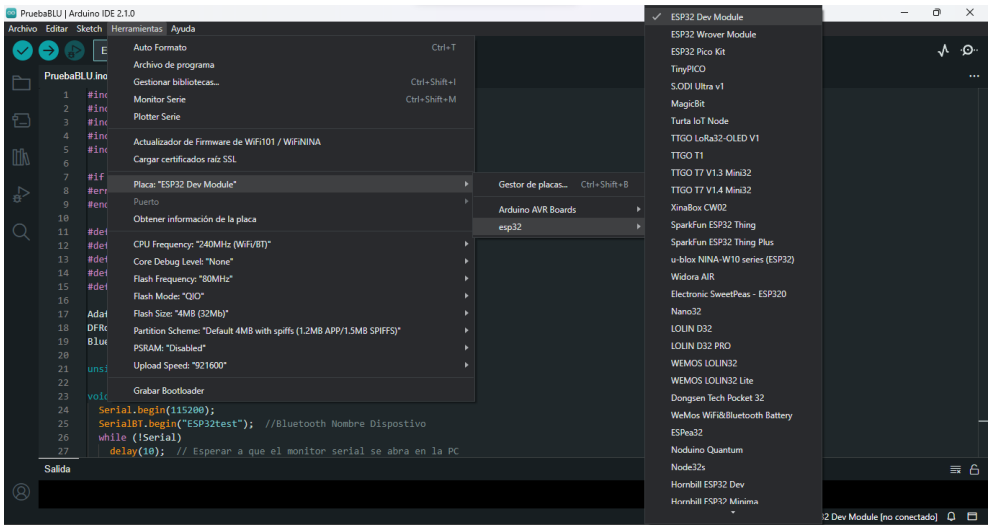

Figura 3.7 Selección de ESP32 Dev Module
Fuente: IDE Arduino

Agregamos las variables y librerías para la comunicación Bluetooth, cada uno cumple con una función importante, las más destacables son “<Adafruit_Sensor.h>”, “Adafruit_BME280.h” las cuales nos servirán para obtener y guardar los datos registrados por el sensor de presión atmosférica. Otro de suma importancia es “DFRobot_MultiGasSensor.h” la cual nos permitirá leer los datos registrados por el sensor de Oxígeno. Al igual la librería que nos ayudara con la comunicación Bluetooth <BluetoothSerial.h>.

```
//#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#include "DFRobot_MultiGasSensor.h"
#include "BluetoothSerial.h"

#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and enable it
#endif

#define BME_SCK 18
#define BME_MISO 19
#define BME_MOSI 23
#define BME_CS 5
#define SEALEVELPRESSURE_HPA (1013.25)

Adafruit_BME280 bme(BME_CS, BME_MOSI, BME_MISO, BME_SCK);
DFRobot_GAS_I2C gas(&Wire, 0x74);
BluetoothSerial SerialBT;

unsigned long delayTime;
```

Figura 3.8 Inclusión de librerías
Fuente: IDE Arduino

Procedemos con la creación de funciones para leer los datos de presión y oxígeno, a la par que enviamos los datos registrados mediante comunicación Bluetooth, la forma de hacerlo, es primero enviando datos de cabecera lo cual se verificara en el archivo de flutter.

```
void loop() {
  printBMEValues();
  printGasValues();
  delay(delayTime);
}

void printBMEValues() {

  String p1 = "Presion ";
  String p2 = String(bme.readPressure() / 100.0F);
  String p3 = p1 + p2;
  SerialBT.println(p3);

  String a1 = "Altitud ";
  String a2 = String(bme.readAltitude(SEALEVELPRESSURE_HPA));
  String a3 = a1 + a2;
  SerialBT.println(a3);
}

void printGasValues() {

  String c1 = "Concentracion ";
  String c2 = String(gas.readGasConcentrationPPM());
  String c3 = c1 + c2;
  SerialBT.println(c3);
}
```

Figura 3.9 Envío de datos

Fuente: IDE Arduino

3.3 DESARROLLO DEL PROTECTOR PLASTICO

Utilizando el software Fusión 360, hemos creado un boceto en la cuadricula de esbozo siguiendo las medidas previamente obtenidas. Esto nos servirá como punto de partida para iniciar la construcción del case plástico del proyecto.

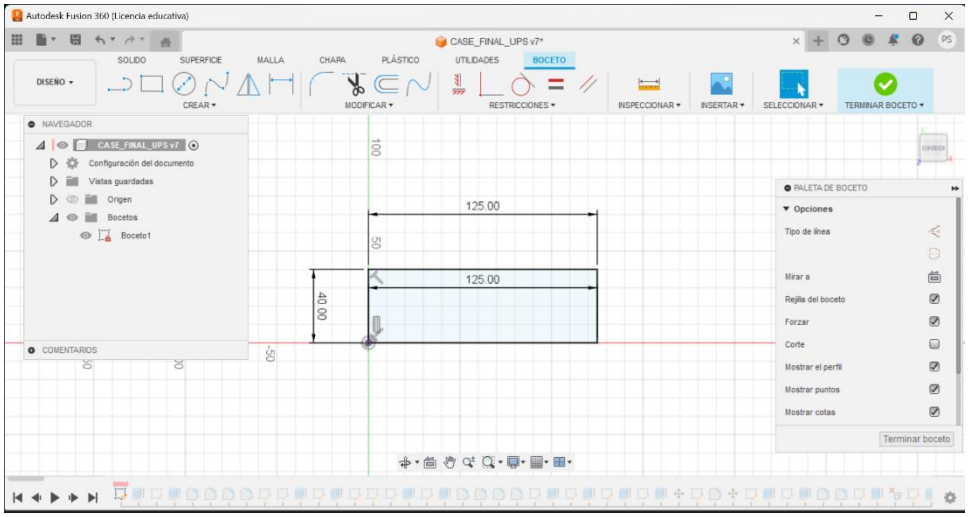

Figura 3.10 Boceto del protector a diseñar 1
Fuente: Autoría

Con el uso de la herramienta Extruir procedemos a agregar una dimensión vertical adecuada a nuestro case.

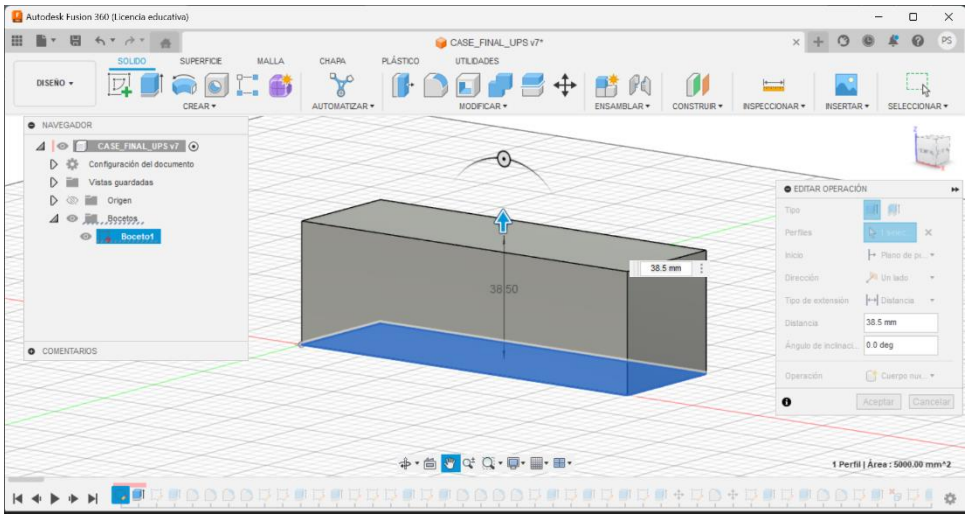

Figura 3.11 Boceto del protector 2
Fuente: Autoría

Una vez obtenida la estructura tridimensional, con la misma herramienta Extruir llevamos a cabo el proceso de agregar los respectivos gabinetes, con las medidas correspondientes para cada componente electrónico, permitiéndonos así una adecuada colocación de los módulos y sensores en su interior.

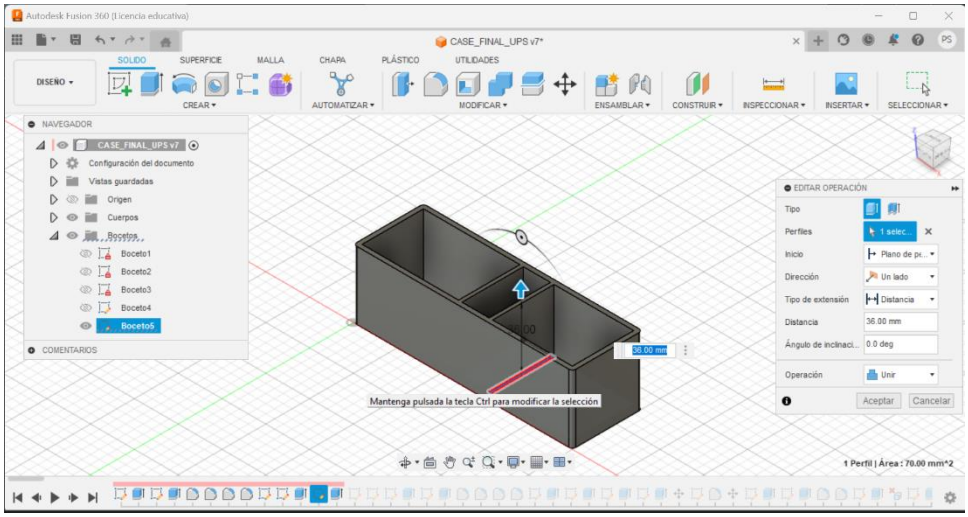

Figura 3.12 Boceto del protector 3
Fuente: Autoría

Procedemos a diseñar la tapa protectora de nuestro case, basándonos en el diseño previamente elaborado. Utilizaremos las especificaciones establecidas anteriormente para garantizar la correcta integridad de los cuerpos.

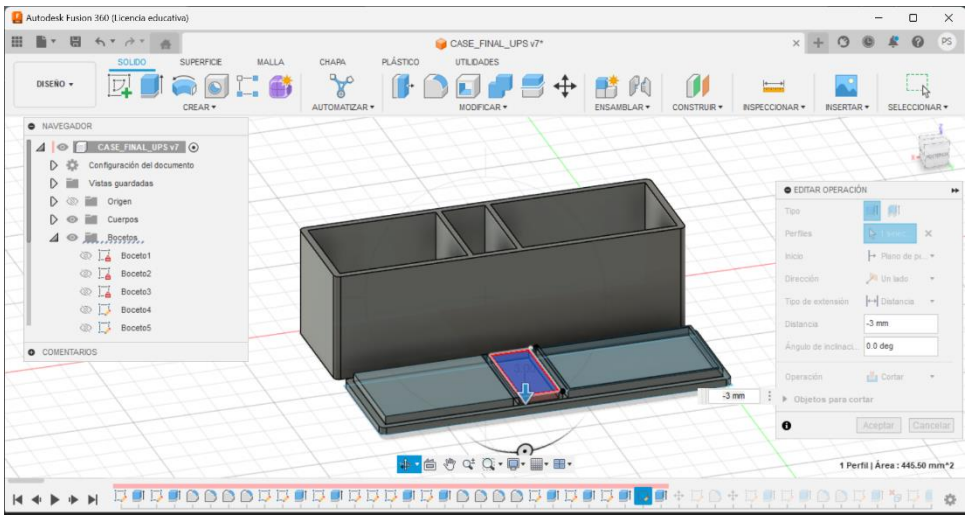

Figura 3.13 Boceto del protector 4
Fuente: Autoría

A continuación, elaboraremos los soportes individuales respectivos para cada componente dentro de nuestro case. Estos soportes serán diseñados de manera precisa para asegurar una óptima sujeción y ubicación de los elementos internos.

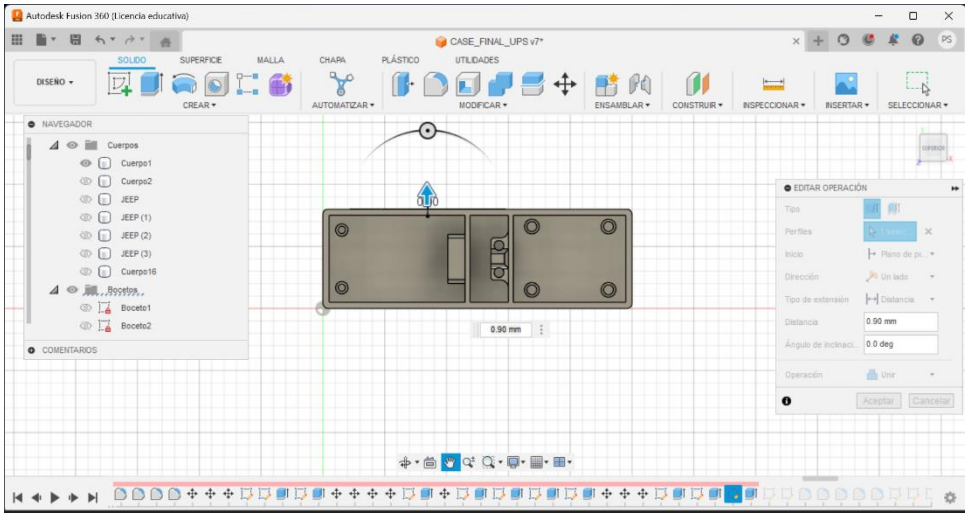

Figura 3.14 Boceto del protector 5
Fuente: Autoría

Finalmente incorporamos las grillas hexagonales y circulares en la tapa como también lo realizamos en una sección lateral del protector para asegurarnos una precisa lectura de los datos provenientes de los sensores.

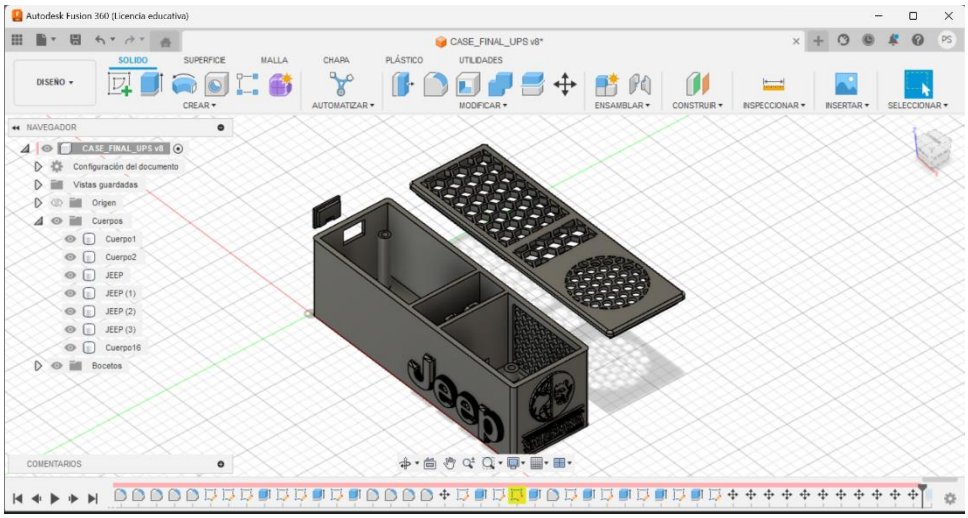

Figura 3.15 Boceto del protector 6
Fuente: Autoría

Todas estas adiciones han sido realizadas de manera cuidadosa, asegurándonos que el diseño del protector plástico resulte estéticamente agradable y funcionalmente adecuado para el propósito del proyecto.

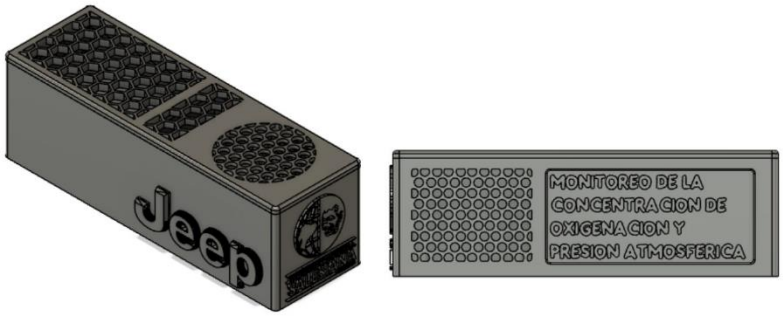

Figura 3.16 Boceto del protector 7
Fuente: Autoría

3.4 CIRCUITO ELECTRÓNICO

El circuito consta de la computadora de placa única ESP32, el cual se encuentra conectado a los sensores, de oxígeno O2 SEN0465 y el sensor de presión atmosférica BMP280 exactamente.

Para la conexión del sensor de presión atmosférica BMP280, se utilizó el protocolo de comunicación SPI.

En la computadora de placa única ESP32 los pines para este tipo de comunicación son el D23, D19, D18, D5, los cuales se comunicarán con los pines SDI, SDO, SCK y CSB respectivamente del sensor.

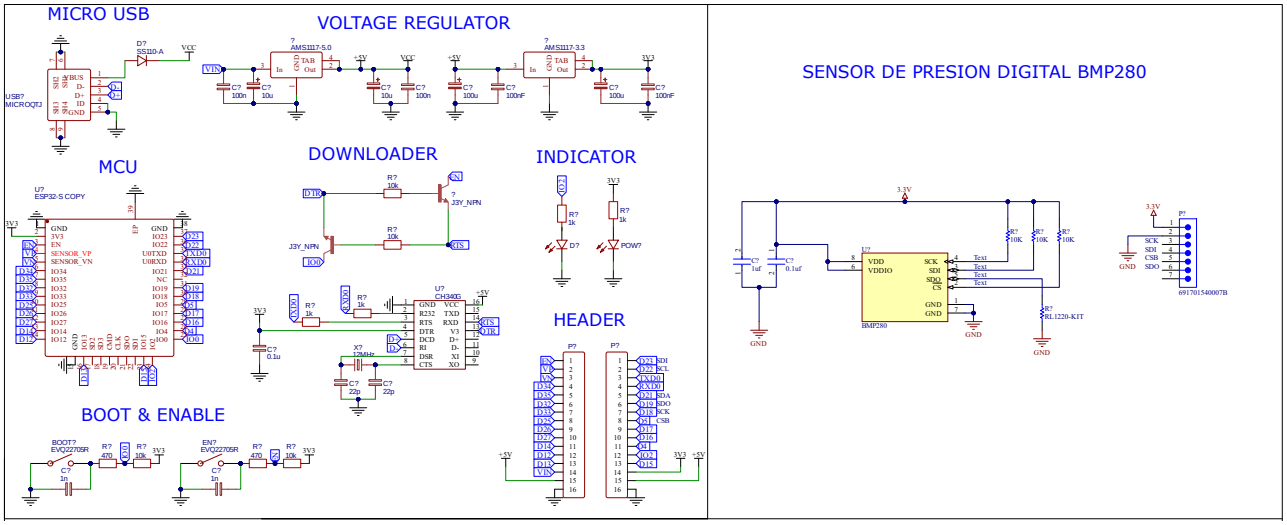

Figura 3.17 Conexión sensor BMP280 Y ESP32
Fuente: Autoría

Para la conexión del sensor de oxígeno SEN0645, se utilizó el protocolo de comunicación I2C.

En la computadora de placa única ESP32 los pines para este tipo de comunicación son el D22, D21, los cuales se comunicarán con los pines VO y ALA, respectivamente del sensor.

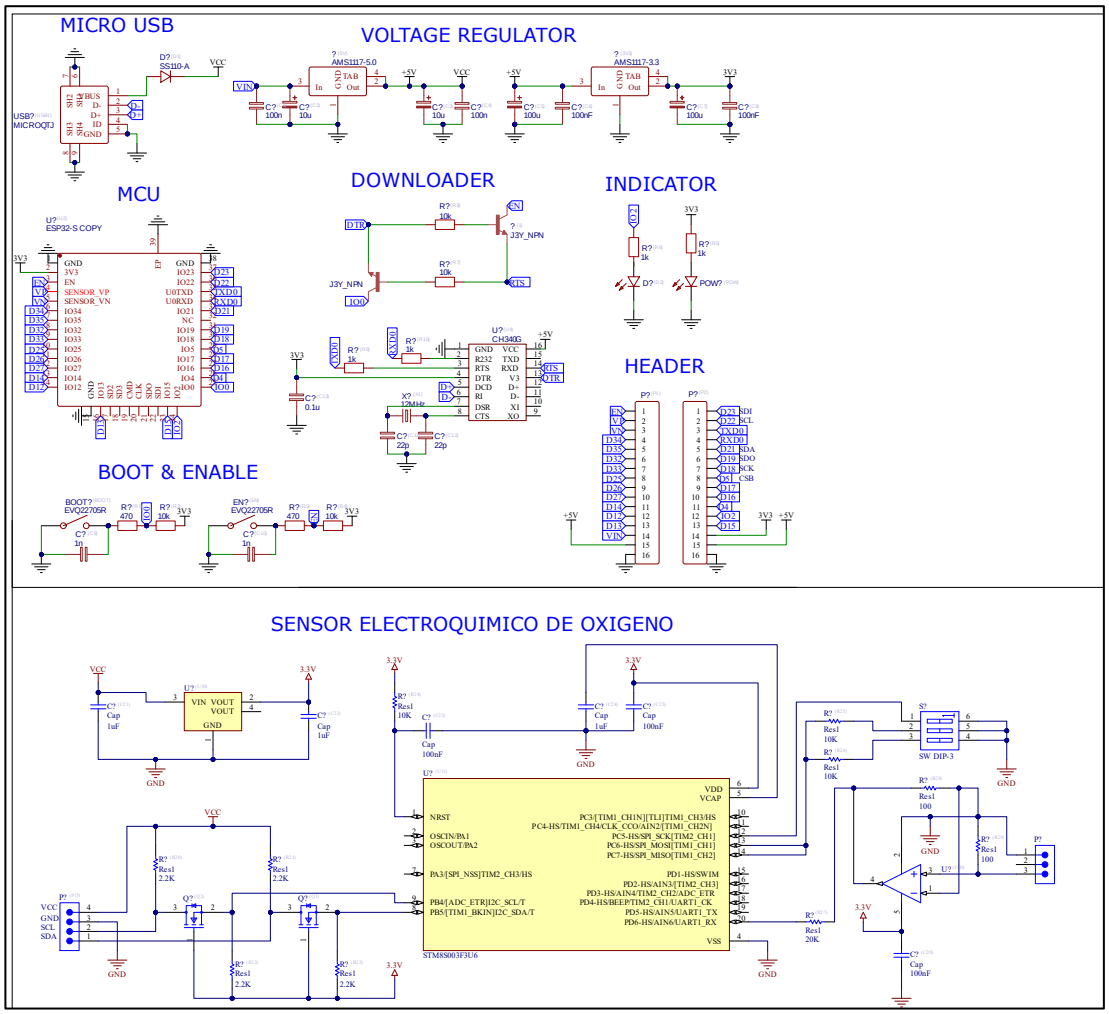

Figura 3.18 Conexión sensor SEN0645 y ESP32
Fuente: Autoría

Al final obtenemos la conexión entre los sensores y computadora de placa única, alimentado con el voltaje del vehículo.

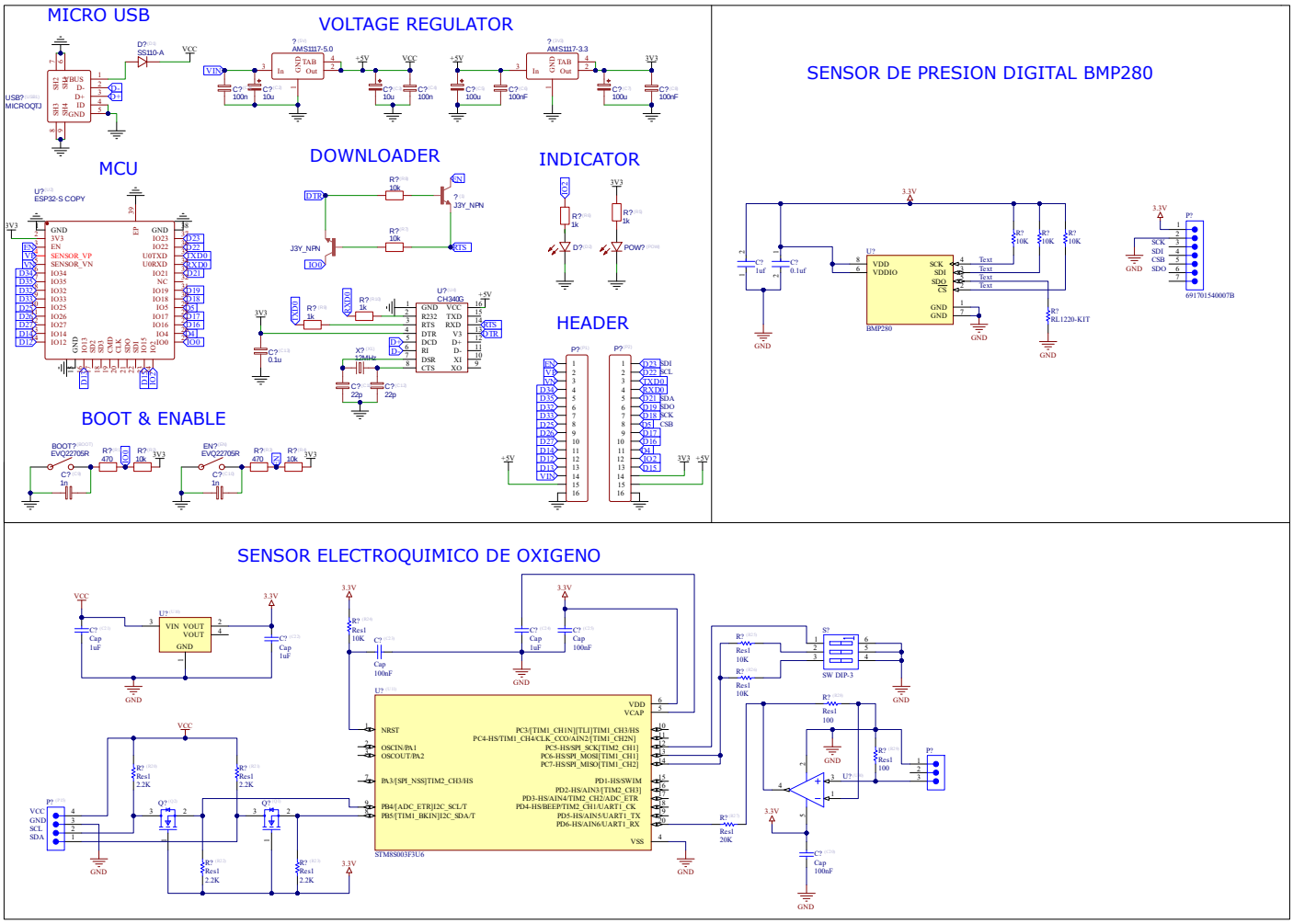

Figura 3.19 Conexión final entre los Sensores y ESP32
Fuente: Autoría

3.5 PROTOCOLOS DE COMUNICACIÓN

3.5.1 PROTOCOLO DE COMUNICACIÓN SPI

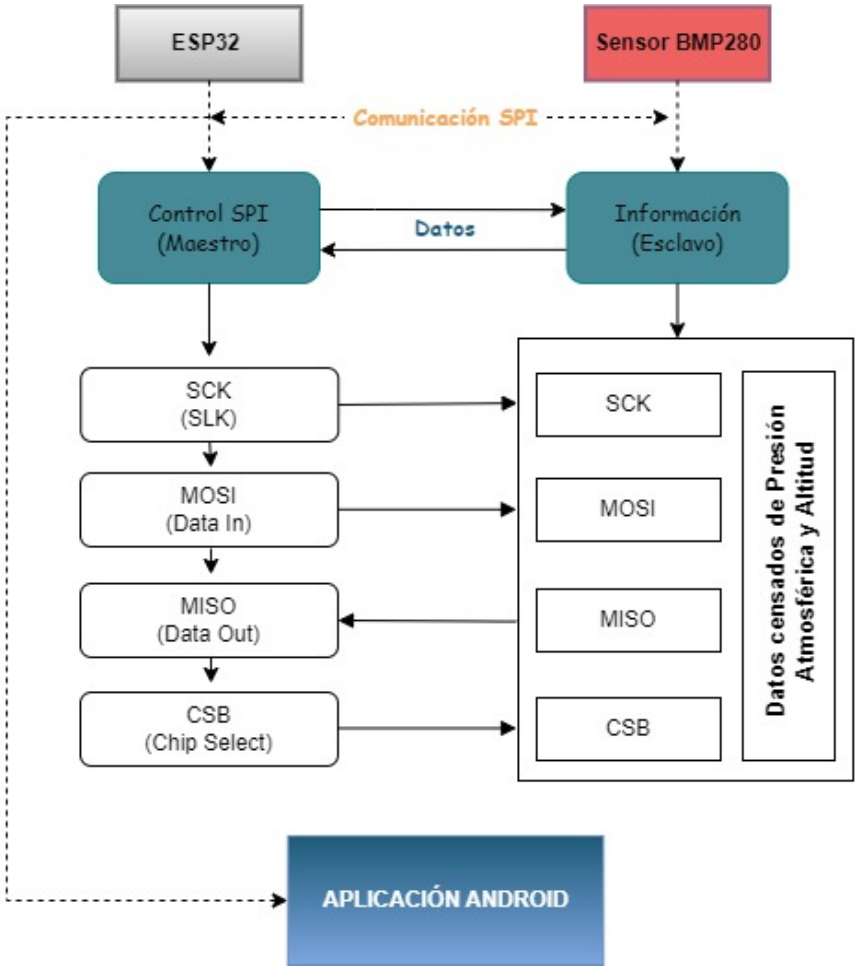

Figura 3.20 Diagrama de bloques para la comunicación SPI (ESP32 y BMP280)
Fuente: Autoría

A continuación, se detalla una explicación de la función de cada uno de los bloques:

ESP32: Actuara como maestro (Control SPI). Se encarga de iniciar y controlar la comunicación con el sensor BMP280.

Sensor BMP280: Es el sensor que actúa como esclavo (Información). Responde a las solicitudes del ESP32 proporcionando datos de presión atmosférica y altitud.

SCK (Serial Clock): Es el pin de reloj en el bus SPI que sincroniza la comunicación entre el maestro y el esclavo. El ESP32 genera este pulso para indicar cuándo los bits de datos deben ser leídos o escritos.

MOSI (Master Out Slave In): Es el pin a través del cual el ESP32 envía comandos y datos al sensor BMP280.

MISO (Master In Slave Out): Es el pin a través del cual el sensor BMP280 envía los datos de respuesta al ESP32.

CSB (Chip Select): Es el pin que habilita o deshabilita la comunicación entre el ESP32 y el sensor BMP280. Cuando el ESP32 pone este pin en estado bajo, selecciona el sensor para la comunicación.

3.5.2 PROTOCOLO DE COMUNICACIÓN I2C

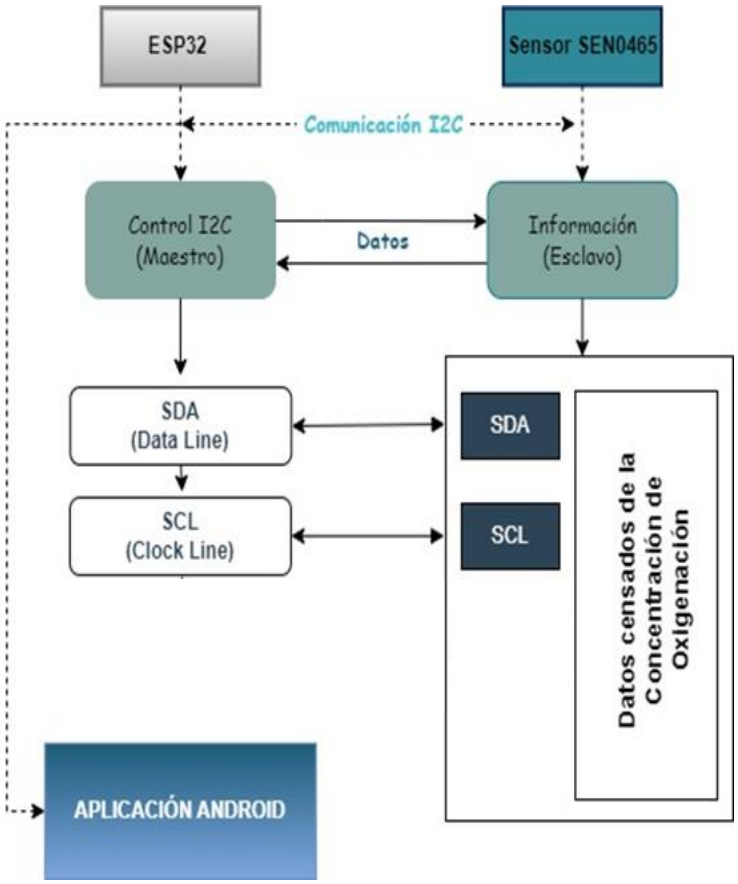

Figura 3.21 Diagrama de bloques para la comunicación I2C (ESP32 y SEN0465)

Fuente: Autoría

A continuación, se detalla una explicación de la función de cada uno de los bloques:

ESP32: (Control I2C) Su función de maestro controlara la comunicación I2C con el sensor de oxígeno SEN0465. Se encarga de enviar comandos y recibir datos del sensor de oxígeno a través del bus I2C.

SEN0465: Es el sensor de oxígeno que actúa como esclavo en la comunicación I2C.

SDA (Serial Data Line): Es el pin de datos en el bus I2C que se utiliza para enviar y recibir datos entre el ESP32 y el sensor de oxígeno.

SCL (Serial Clock Line): Es el pin de reloj en el bus I2C que sincroniza la comunicación entre el maestro y el esclavo.

SCB (Serial Clock): Es el pin que habilita o deshabilita la comunicación entre el ESP32 y el sensor de oxígeno. Cuando el ESP32 pone este pin en estado bajo, selecciona el sensor para la comunicación.

3.6 PRUEBAS COMUNICACIÓN ENTRE EL MÓDULO ESP32 Y APP DE MONITOREO

Para la realización de pruebas de comunicación, se efectuó en dos fases, la primera fase consta en comprobar su funcionamiento mediante comunicaciones simple de envío y lectura de datos, con los dispositivos armados en el protoboard.

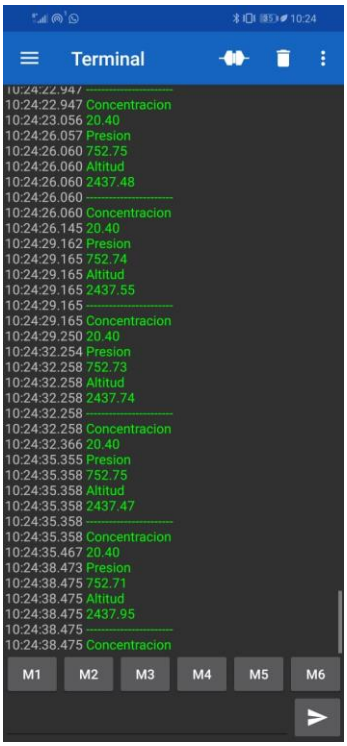

Figura 3.22 Recepción de datos
Fuente: Autoría

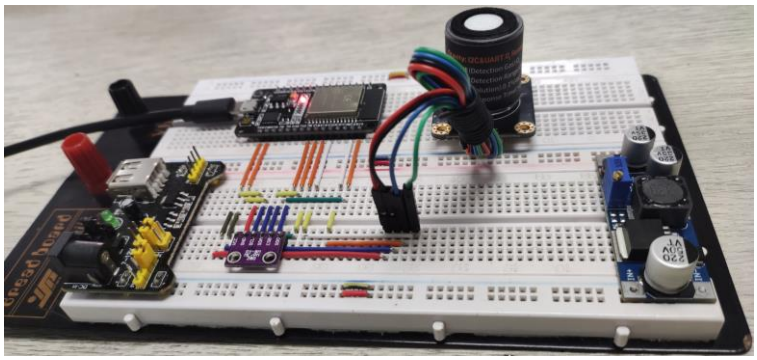

Figura 3.23 Conexión de dispositivos
Fuente: Autoría

La segunda fase de pruebas de comunicación consta en comprobar el funcionamiento de los dispositivos y la recepción de datos, al encontrase dentro del protector plástico y al igual que la aplicación cargada en un automóvil de prueba.

Figura 3.24 Dispositivos en el protector plástico.
Fuente: Autoría

Figura 3.25 Recepción de datos pantalla.
Fuente: Autoría

3.7 ANALISIS Y RESULTADOS

Los resultados obtenidos fueron coherentes con las expectativas preestablecidas, ya que al momento de comprobar los protocolos de comunicaciones se corroboró que no hubo ninguna falla al momento de recibir los datos. Se comprobó el nivel de presión atmosférica y el nivel de oxígeno en la Universidad Politécnica Salesiana.

Figura 3.26 Prueba UPS
Fuente: Autoría

Adicionalmente, se realizaron pruebas en zonas rurales, con un aumento significativo de altitud. Estas pruebas permitieron confirmar que el tanto valor de la presión atmosférica, como el nivel de oxigenación experimentan variaciones dependiendo de la ubicación geográfica del dispositivo de medición.

Figura 3.27 Conexión de dispositivos
Fuente: Autoría

A su vez también se comprobó en diferentes vehículos, comprobando de esta forma que la aplicación móvil se adapte correctamente a las necesidades requeridas de los diferentes usuarios.

Figura 3.28 Implementación de la aplicación móvil en el JEEP Grand Cherokee WJ
Fuente: Autoría

Durante estas pruebas en vehículos, se llevó a cabo una exhaustiva evaluación para asegurar que la aplicación móvil funcionara de manera óptima en diferentes entornos y situaciones. Se comprobó que la interfaz de la aplicación fuera intuitiva y fácil de usar, permitiendo a los usuarios acceder de forma rápida y eficiente a los datos requeridos.

CAPÍTULO 4: CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

El uso del módulo ESP32 DevKit V1, nos permite el desarrollo y control de pruebas entre hardware y software mediante diferentes comunicaciones como es la Bluetooth y Wi-Fi, estas pueden ser utilizadas según las especificaciones y requisitos definidos en el proyecto a realizar; teniendo en cuenta esta información se llevó a cabo diferentes pruebas de comunicación y recepción de datos llegando a la conclusión de que el mejor sistema a utilizar es la comunicación Bluetooth, aplicándolo a nuestro sistema de monitoreo se obtuvo los resultados esperados con una recepción más eficaz en tiempo real de los datos registrados por los sensores .

La implementación de este proyecto contemplo ciertos requisitos específicos, como su método de conexión con los sensores, por consiguiente, al momento de investigar y realizar pruebas se definió el tipo de comunicación correspondiente para cada uno de los sensores, concluimos que el método de comunicación para el Sensor Electroquímico O2SEN0465 será del tipo I2C y para el Sensor de Presión Atmosférica del tipo SPI respectivamente.

La recepción de datos que serán visualizados en la aplicación móvil se lo llevara a cabo de manera inalámbrica, para comprobar la veracidad y eficacia de este

método se realizó diferentes pruebas en diferentes ambientes para corroborar la exactitud del sensor y del código generado, llegando a los resultados esperados y verificando la correcta funcionalidad de estos.

4.2 RECOMENDACIONES

Con la finalidad de una mayor duración de los componentes electrónicos internos se recomienda adicionar al interior del protector plástico una esponja VCI FOAM con el fin de evitar la corrosión, y la exposición a agentes externos que podrían interferir con la funcionalidad del sistema.

Se recomienda adicionar las diferentes características de medición que posee el sensor BMP280 a la programación del entorno IDE de Arduino, cabe mencionar que se debería modificar la interfaz y el método de recepción de datos de la aplicación móvil desarrollada.

Se recomienda agregar un convertidor DC-DC exclusivo al sistema electrónico automotriz del prototipo para adaptarse a las variaciones de voltaje al encender el automóvil y evitar daños por picos de voltaje. Esto mejorará la estabilidad y confiabilidad de la conexión con la alimentación directa del automóvil.

REFERENCIAS BIBLIOGRÁFICAS

[1] G. Pratt, "¿Cómo funcionan los sensores electroquímicos? ," Apr. 22, 2022. https://www.crowcon.com/es/blog/how-do-electrochemical-sensors-work/ (accessed Jul. 09, 2023).

[2] Kistler Group, "Sensor de presión piezorresistivo | Kistler." https://www.kistler.com/INT/es/sensor-de-presion-piezorresistivo/C00000143 (accessed Jul. 09, 2023).

[3] Tameson, "Presión atmosférica, absoluta, manométrica y diferencial | Tameson.es." https://tameson.es/pages/presion-atmosfericap-absolutap-manometrica-y-diferencial (accessed Jul. 09, 2023).

[4] mundocompresor, "Qué es la Presión barométrica - definición mundocompresor.com." https://www.mundocompresor.com/diccionario-tecnico/presion-barometrica (accessed Jul. 09, 2023).

[5] Industriapedia, "Qué es Presión barométrica." https://industriapedia.com/que-es-presion-barometrica/ (accessed Jul. 09, 2023).

[6] R. David, *Desarrollo de aplicaciones para Android I*. Ministerio de Educación, Cultura y Deporte.

[7] A. Garcia, "Android," no. 49, p. 50=53.

[8] Developers, "Arquitectura de la plataforma | Desarrolladores de Android | Android Developers," May 07, 2020. https://developer.android.com/guide/platform?hl=es-419 (accessed Jul. 09, 2023).

[9] "Bus SPI | Aprendiendo Arduino," Nov. 13, 2016. https://aprendiendoarduino.wordpress.com/2016/11/13/bus-spi/ (accessed Jul. 29, 2023).

[10] Robots Didácticos, "Descripción y funcionamiento del Bus I2C | Robots Didácticos," Jun. 24, 2019. https://robots-

argentina.com.ar/didactica/descripcion-y-funcionamiento-del-bus-i2c/ (accessed Jul. 29, 2023).

[11] J. Beningo and Digikey, “Los módulos inalámbricos ESP32 simplifican el diseño IoT | DigiKey,” Jan. 21, 2020. https://www.digikey.com/es/articles/how-to-select-and-use-the-right-esp32-wi-fi-bluetooth-module (accessed Jul. 09, 2023).

[12] JeepSpecs, “Second-Generation ‘WJ / WG’ Jeep Grand Cherokee Guide.” [Online]. Available: https://jeepspecs.com/grand-cherokee/wj-wg-generation/

[13] DFROBOT, “CO, O2, NH3, H2S, NO2, HCL, H2, PH3, SO2, O3, CL2, HF Gas Sensor Wiki - DFRobot.” https://wiki.dfrobot.com/SKU_SEN0465toSEN0476_Gravity_Gas_Sensor_Calibrated_I2C_UART (accessed Jul. 09, 2023).

[14] Naylamp Mechatronics, “Sensor de presión BMP280.” https://naylampmechatronics.com/sensores-posicion-inerciales-gps/358-sensor-de-presion-bmp280.html (accessed Jul. 09, 2023).

[15] R. Mischianti, “DOIT ESP32 DEV KIT v1 high resolution pinout and specs – Renzo Mischianti,” Feb. 17, 2021. https://mischianti.org/2021/02/17/doit-esp32-dev-kit-v1-high-resolution-pinout-and-specs/ (accessed Jul. 09, 2023).

[16] F. Cristancho, “¿Qué es Flutter y para qué sirve? - Talently | Talently Blog,” 2022. https://talently.tech/blog/que-es-flutter/ (accessed Jul. 09, 2023).

[17] J. Guerra, “Arduino IDE en Windows Linux y Mac,” Aug. 20, 2020. https://programarfacil.com/blog/arduino-blog/arduino-ide/ (accessed Jul. 09, 2023).

[18] A. M, “Fusion 360: todo lo que necesitas saber sobre el software 3D - 3Dnatives,” Apr. 29, 2020. https://www.3dnatives.com/es/fusion-360-software-290420202/#! (accessed Jul. 09, 2023).

APÉNDICES

APÉNDICE A: DIMENSIONES DEL PROTECTOR PLÁSTICO PARA LOS DISPOSITIVOS Y SENSORES

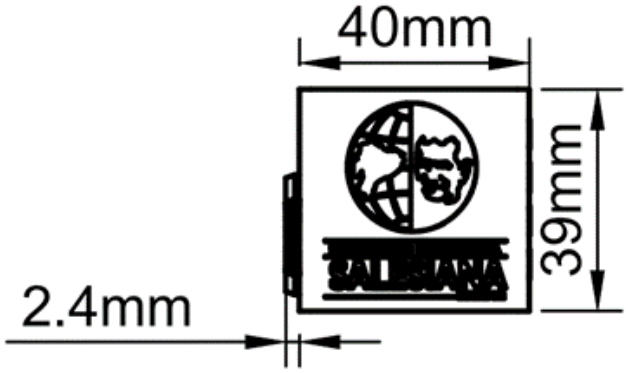

Figura A 1.1Vista Frontal del Protector Plástico

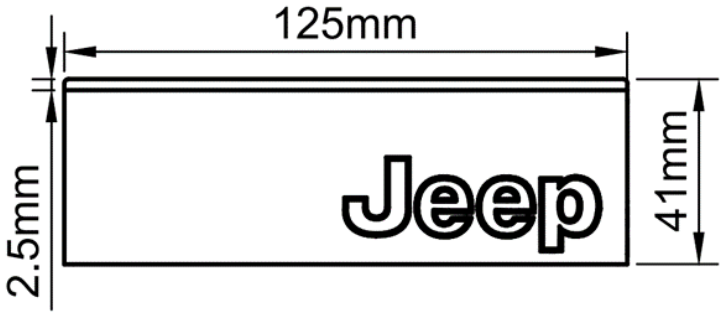

Figura A 1.2 Vista Lateral Izquierda del Protector Plástico

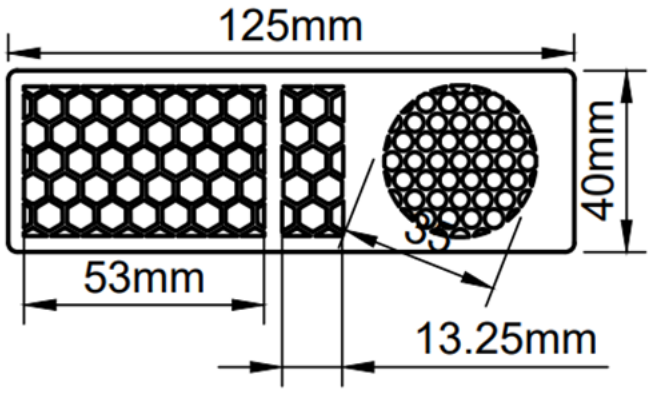

Figura A 1.3 Vista Superior del Protector Plástico

APÉNDICE B: ALGORITMO IMPLEMENTADO PARA LA OBTENCIÓN DE LOS PARÁMETROS

Implementación del código realizado en Arduino para la obtención de los parámetros.

```
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#include "DFRobot_MultiGasSensor.h"
#include "BluetoothSerial.h"

#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and
enable it
#endif

#define BME_SCK 18
#define BME_MISO 19
#define BME_MOSI 23
#define BME_CS 5
#define SEALEVELPRESSURE_HPA (1013.25)

Adafruit_BME280 bme(BME_CS, BME_MOSI, BME_MISO, BME_SCK);
DFRobot_GAS_I2C gas(&Wire, 0x74);
BluetoothSerial SerialBT;

unsigned long delayTime;

void setup() {
 Serial.begin(115200);
 SerialBT.begin("ESP32test"); //Bluetooth Nombre Dispostivo
 while (!Serial)
 delay(10); // Esperar a que el monitor serial se abra en la PC

 if (!bme.begin(0x76)) {
 Serial.println("No se pudo encontrar un sensor BMP280 válido.
Verifique la conexión.");
 while (1)
 ;
 }

 Wire.begin(21, 22); // Especificar los pines SDA (D21) y SCL (D22) del
ESP32

 while (!gas.begin()) {
 Serial.println("No se detectó el dispositivo!");
```

```
 delay(1000);
 }

 Serial.println("No se pudo encontrar un sensor Gravity O2 válido.
Verifique la conexión.");

 gas.changeAcquireMode(gas.PASSIVITY);
 delay(1000);

 gas.setTempCompensation(gas.OFF);

 Serial.println("===== BMP280 y Sensor de Oxígeno Test =====");
 delayTime = 3000;
}

void loop() {
 printBMEValues();
 printGasValues();
 delay(delayTime);
}

void printBMEValues() {

 String p1 = "Presion ";
 String p2 = String(bme.readPressure() / 100.0F);
 String p3 = p1 + p2;
 SerialBT.println(p3);

 String a1 = "Altitud ";
 String a2 = String(bme.readAltitude(SEALEVELPRESSURE_HPA));
 String a3 = a1 + a2;
 SerialBT.println(a3);
}

void printGasValues() {

 String c1 = "Concentracion ";
 String c2 = String(gas.readGasConcentrationPPM());
 String c3 = c1 + c2;
 SerialBT.println(c3);
}
```

APÉNDICE C: PROGRAMACIÓN DE LA APLICACIÓN MÓVIL EN FLUTTER

Implementación de la sección del tema de la APP..

```
import 'package:flutter/material.dart';

class AppTheme {
 final Color primary = Colors.white;

 ThemeData getTheme() => ThemeData(
 useMaterial3: true,
 colorSchemeSeed: primary,
 appBarTheme: AppBarTheme(
 centerTitle: true,
 // color: primary,
 backgroundColor: primary),
 // switchTheme: SwitchThemeData()
 );
}
```

Implementación de la sección de comunicación y diseño.

```
import 'dart:async';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'dart:developer';
import 'dart:convert';
import
'package:flutter_bluetooth_serial/flutter_bluetooth_serial.dart';
import 'package:pruebas_blu/config/app_theme.dart';
import 'package:syncfusion_flutter_gauges/gauges.dart';

class BluetoothScreen extends StatefulWidget {
 const BluetoothScreen({super.key});

 @override
 State<BluetoothScreen> createState() => _BluetoothScreenState();
}

class _BluetoothScreenState extends State<BluetoothScreen> {
 // Initializing the Bluetooth connection state to be unknown
 BluetoothState _bluetoothState = BluetoothState.UNKNOWN;
 // Get the instance of the Bluetooth
 // ignore: prefer_final_fields
 FlutterBluetoothSerial _bluetooth = FlutterBluetoothSerial.instance;
```

```
// Track the Bluetooth connection with the remote device
BluetoothConnection? connection;

// ignore: unused_field
int? _deviceState;

// Define some variables, which will be required later
List<BluetoothDevice> _devicesList = [];
BluetoothDevice? _device;
bool _connected = false;
bool _isButtonUnavailable = false;

// Bluetooth device
// To track whether the device is still connected to Bluetooth
bool get isConnected => connection != null && connection!.isConnected;
bool isDisconnecting = false;

//! Variables del proyecto
double presion = 0.00;
double altitud = 0.00;
double concentracion = 0.0;

@override
void initState() {
super.initState();
// Get current state
FlutterBluetoothSerial.instance.state.then((state) {
setState(() {
_bluetoothState = state;
});
});
_deviceState = 0; // neutral
// If the bluetooth of the device is not enabled,
// then request permission to turn on bluetooth
// as the app starts up
enableBluetooth();

// Listen for further state changes
FlutterBluetoothSerial.instance
.onStateChanged()
.listen((BluetoothState state) {
setState(() {
_bluetoothState = state;
if (_bluetoothState == BluetoothState.STATE_OFF) {
_isButtonUnavailable = true;
}
getPairedDevices();
});
});
```

```
}

@override
void dispose() {
// Avoid memory leak and disconnect
if (isConnected) {
isDisconnecting = true;
connection!.dispose();
connection = null;
}
super.dispose();
}

// Request Bluetooth permission from the user
Future<bool> enableBluetooth() async {
// Retrieving the current Bluetooth state
_bluetoothState = await FlutterBluetoothSerial.instance.state;
// If the bluetooth is off, then turn it on first
// and then retrieve the devices that are paired.
if (_bluetoothState == BluetoothState.STATE_OFF) {
await FlutterBluetoothSerial.instance.requestEnable();
await getPairedDevices();
return true;
} else {
await getPairedDevices();
}
return false;
}

// For retrieving and storing the paired devices in a list.
Future<void> getPairedDevices() async {
List<BluetoothDevice> devices = [];
// To get the list of paired devices
try {
devices = await _bluetooth.getBondedDevices();
} on PlatformException {
log("Error getPairedDevices");
}
// It is an error to call [setState] unless [mounted] is true.
if (!mounted) {
return;
}
// Store the [devices] list in the [_devicesList] for accessing the
list outside this class

setState(() {
_devicesList = devices;
});
}
```

```
@override
Widget build(BuildContext context) {
return Scaffold(
// key: _scaffoldKey,
backgroundColor: Colors.black, //cambio de color ipfsdfsdfsdf
appBar: AppBar(
title: const Text(
"PANTALLA DE MONITOREO",
style: TextStyle(
color: Colors.black,
fontWeight: FontWeight.bold,
fontSize: 24,
),
),
// backgroundColor: Colors.deepPurple,
actions: [
IconButton(
onPressed: () async {
// So, that when new devices are paired
// while the app is running, user can refresh
// the paired devices list.
await getPairedDevices().then((_) {
show('Dispositivos actualizados correctamente.', Colors.green);
});
// ScaffoldMessenger.of(context).showSnackBar(
// SnackBar(content: Text("Hello This is FlutterCampus")));
},
icon: const Icon(
Icons.refresh,
color: Colors.black,
),
// splashColor: Colors.deepPurple,
)
],
),
// ignore: avoid_unnecessary_containers
body: SingleChildScrollView(
child: Column(
mainAxisSize: MainAxisSize.max,
children: [
Visibility(
visible: _isButtonUnavailable &&
_bluetoothState == BluetoothState.STATE_ON,
child: const LinearProgressIndicator(
backgroundColor: Colors.yellow,
valueColor: AlwaysStoppedAnimation<Color>(Colors.red),
),
),
```

```
//**BTN PARA ACTIVAR/DESACTIVAR */
Padding(
padding: const EdgeInsets.all(10),
child: Row(
mainAxisAlignment: MainAxisAlignment.start,
children: <Widget>[
const Expanded(
child: Text(
'Activar Bluetooth',
style: TextStyle(
color: Colors.white,
fontSize: 16,
),
),
),
Switch(
value: _bluetoothState.isEnabled,
onChanged: (bool value) {
// print(value);
future() async {
if (value) {
await FlutterBluetoothSerial.instance.requestEnable();
} else {
await FlutterBluetoothSerial.instance
.requestDisable();
}

await getPairedDevices();
_isButtonUnavailable = false;

if (_connected) {
_disconnect();
}
}

future().then((_) {
setState(() {});
});
},

)
],
),
),
//**DISPOSITIVOS */
Stack(
children: [
Column(
```

```
children: [
Padding(
padding: const EdgeInsets.only(top: 10),
child: Text(
"DISPOSITIVOS VINCULADOS",
style:
TextStyle(fontSize: 24, color: AppTheme().primary),
textAlign: TextAlign.center,
),
),
Padding(
padding: const EdgeInsets.all(8.0),
child: Row(
mainAxisAlignment: MainAxisAlignment.spaceBetween,
children: [
const Text(
'Dispositivo:',
style: TextStyle(fontWeight: FontWeight.bold, color: Colors.white),
),
DropdownButton(
items: _getDeviceItems(),
onChanged: (value) =>
setState(() => _device = value),
value: _devicesList.isNotEmpty ? _device : null,
),
FilledButton(
onPressed: _isButtonUnavailable
? null
: _connected
? _disconnect
: _connect,
child:
Text(_connected ? 'Desconectar' : 'Conectar'),
),
],
),
)
],
)
],
),

Row(
mainAxisAlignment: MainAxisAlignment.spaceEvenly,
children: [
//**Barometro
SfRadialGauge(
// backgroundColor: Colors.white,
title: const GaugeTitle(
```

```
text: 'Barómetro',
textStyle: TextStyle(
color: Colors.white,
fontSize: 25.0,
fontWeight: FontWeight.bold,
fontFamily: 'SourceSansPro'),
),
enableLoadingAnimation: true,
axes: <RadialAxis>[
RadialAxis(
minimum: 0,
maximum: 1080,
// canRotateLabels: true,
// interval: 10,
// maximumLabels: 1080,
showFirstLabel: true,
showLastLabel: true,
// labelOffset: 550,
// showLabels: true,
// showAxisLine: true,
// showTicks: tru,
ranges: <GaugeRange>[
GaugeRange(
startValue: 0,
endValue: 1080,
//color: AppTheme().primary,
color: Colors.white,
// sizeUnit: GaugeSizeUnit.logicalPixel,
// label: 'hpa',
),
],
pointers: <GaugePointer>[
//TODO 2 Presion
NeedlePointer(
value: presion,
enableAnimation: true,
needleColor: AppTheme().primary,
knobStyle: KnobStyle(
knobRadius: 0.06,
borderColor: AppTheme().primary,
color: Colors.white,
borderWidth: 0.035,
),
tailStyle: TailStyle(
color: AppTheme().primary,
width: 4,
length: 0.15,
),
)
```

```
],
annotations: <GaugeAnnotation>[
//TODO 3 Altitud
GaugeAnnotation(
widget: Text(
'${altitud.toString()} m',
style: const TextStyle(
fontSize: 15, fontWeight: FontWeight.bold, color: Colors.white),
),
angle: 90,
positionFactor: 0.5,
),
//TODO 2 Presion
GaugeAnnotation(
widget: Text('${presion.toString()} hPA',
style: const TextStyle(
fontSize: 25, fontWeight: FontWeight.bold, color: Colors.white)),
angle: 90,
positionFactor: 1,
)
],
)
],
),
//**End Barometro
//**Concentración
SfRadialGauge(
title: const GaugeTitle(
text: 'Concentración 02',
textStyle: TextStyle(
color: Colors.white,
fontSize: 25.0,
fontWeight: FontWeight.bold,
fontFamily: 'SourceSansPro'),
),
enableLoadingAnimation: true,
axes: <RadialAxis>[
RadialAxis(
radiusFactor: 0.8,
showAxisLine: false,
// onLabelCreated: _handleLabelCreated,
startAngle: 270,
endAngle: 270,
canRotateLabels: true,
labelsPosition: ElementsPosition.outside,
axisLabelStyle: const GaugeTextStyle(),
majorTickStyle: const MajorTickStyle(
color : Colors.white,//////////////////////////
length: 0.15,
```

```
lengthUnit: GaugeSizeUnit.factor,
thickness: 1,
dashArray: <double>[2, 1],
),
minorTicksPerInterval: 4,
interval: 10,
minorTickStyle: const MinorTickStyle(
color : Colors.white,//////////////////////////
length: 0.06,
thickness: 1,
lengthUnit: GaugeSizeUnit.factor,
/// Dash array not supported in web.
dashArray: <double>[2, 1]),
pointers: <GaugePointer>[
//TODO 4 Concentracion
NeedlePointer(
enableAnimation: true,
animationDuration: 1300,
value: concentracion,
needleColor: AppTheme().primary, ///////color
needleStartWidth: 0,
needleEndWidth: 3,
needleLength: 0.8,
tailStyle: TailStyle(
width: 4,
lengthUnit: GaugeSizeUnit.logicalPixel,
length: 20,
color: AppTheme().primary,
),
knobStyle: KnobStyle(
knobRadius: 8,
sizeUnit: GaugeSizeUnit.logicalPixel,
color: AppTheme().primary,
),
)
],
annotations: <GaugeAnnotation>[
//TODO 4 Concentracion
GaugeAnnotation(
widget: Text(
// concentracion.toString() ,
'${concentracion.toString()} %',
style: const TextStyle(
fontSize: 22,
fontWeight: FontWeight.bold,
color: Colors.white
),
),
angle: 90,
```

```
positionFactor: 0.5,
),
],
)
],
)

//**End Concentración
],
)
],
),
),
// ),
);
}

// Creacion de la lista de dispositivos disponibles
List<DropdownMenuItem<BluetoothDevice>> _getDeviceItems() {
List<DropdownMenuItem<BluetoothDevice>> items = [];
if (_devicesList.isEmpty) {
items.add(const DropdownMenuItem(
child: Text('SIN DISPOSITIVOS',
style: TextStyle(
color: Colors.white,
fontWeight: FontWeight.bold,
fontSize: 12,
)),
));
} else {
for (var device in _devicesList) {
items.add(DropdownMenuItem(
value: device,
child: Text(device.name!,
style: TextStyle(
backgroundColor: Colors.black,
color: Colors.white,
fontWeight: FontWeight.bold,
fontSize: 12,
)),
));
}
}
return items;
}

// Metodo de conexión bluetooth
void _connect() async {
setState(() {
```

```
_isButtonUnavailable = true;
});
if (_device == null) {
show('Seleccione un dispositivo', Colors.red);
_isButtonUnavailable = false;
} else {
if (!isConnected) {
await BluetoothConnection.toAddress(_device!.address)
// ignore: no_leading_underscores_for_local_identifiers
.then((_connection) {
log('Connected to the device');
connection = _connection;
setState(() {
_connected = true;
});
//**LISTEN DATA
connection!.input!.listen((Uint8List data) {
//TODO 1 Se recibe la data del bluetoth
final text = ascii.decode(data);
if (text.contains('Presion')) {
final pressure = text.substring(8);
// log('Presion: $pressure');
presion = double.parse(pressure);
setState(() {});
// log(presion.toString());
}
if (text.contains('Altitud')) {
final altitude = text.substring(8);
// log('Altitud: $altitude');
altitud = double.parse(altitude);
setState(() {});
}
if (text.contains('Concentracion')) {
final concentracionn = text.substring(14);
// log('Concentracion: $concentracionn');
concentracion = double.parse(concentracionn);
setState(() {});
}
}).onDone(() {
if (isDisconnecting) {
log('Disconnecting locally!');
} else {
log('Disconnected remotely!');
}
if (mounted) {
setState(() {});
}
});
}).catchError((error) {
```

```
log('Cannot connect, exception occurred');
log(error);
});
show('Dispositivo connectado', Colors.green);
setState(() => _isButtonUnavailable = false);
}
}
}

// Metodo de desconeccion bluetooth
void _disconnect() async {
// setState(() {
// _isButtonUnavailable = true;
// _deviceState = 0;
// presion = 0.00;
// altitud = 0.00;
// concentracion = 0.0;
// });

await connection!.close();
show('Dispositivo desconectado', Colors.green);
if (!connection!.isConnected) {
setState(() {
_connected = false;
_isButtonUnavailable = false;
_deviceState = 0;
presion = 0.00;
altitud = 0.00;
concentracion = 0.0;
});
}
}

// Method to show a Snackbar
show(String message, Color color) {
ScaffoldMessenger.of(context).clearSnackBars();
final snackbar = SnackBar(
content: Text(
message,
textAlign: TextAlign.center,
),
duration: const Duration(milliseconds: 3000),
backgroundColor: color,
);
ScaffoldMessenger.of(context).showSnackBar(snackbar);
}
}
```

Implementación de la sección main.

```dart
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:pruebas_blu/config/app_theme.dart';
import 'presentation/screens/bluetooth/bluetooth_screen.dart';

void main() {
 WidgetsFlutterBinding.ensureInitialized();
 SystemChrome.setPreferredOrientations([
 DeviceOrientation.landscapeRight,
 DeviceOrientation.landscapeRight,
 ]).then((value) => runApp(const MyApp()));
 runApp(const MyApp());
}

class MyApp extends StatelessWidget {
 const MyApp({super.key});

 @override
 Widget build(BuildContext context) {
 return MaterialApp(
 title: 'Monitore0 O2',
 theme: AppTheme().getTheme(),
 home: const BluetoothScreen(),
 debugShowCheckedModeBanner: false,
 );
 }
}
```