ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Mecánica y Ciencias de la Producción

Diseño de un purificador de aire portable asistido por inteligencia artificial.

PROYECTO INTEGRADOR

Previo la obtención del Título de:

Ingenieros en Mecatrónica

Presentado por:

Miguel Alejandro Bailón Silva

Guido Alejandro Ramírez Espinoza

GUAYAQUIL - ECUADOR

Año: 2021

DEDICATORIA

El presente proyecto va dedicado a mi madre, Patricia Aracely Silva Párraga, quien se ha esforzado por muchos años para que no nos falte nada a mis hermanos y a mí.

A la memoria de mi padre, Milton Fernando Bailón Alvarado, que estaría muy orgulloso de cada uno de sus hijos.

A mis hermanos, Milton Bailón y Enrique Bailón, quienes me han aconsejado a lo largo de mi vida.

A mis familiares cercanos que siempre me han apoyado con palabras de aliento.

A mis amigos más cercanos: Raúl, Christian, Guido, Luis, Gabriel y Daniela.

Miguel Bailón

DEDICATORIA

El presente proyecto lo dedico a mis padres, Guido Manuel Ramírez y Susana Espinoza, quienes me han brindado su apoyo durante todos mis años como estudiante y me han inculcado la perseverancia y optimismo a lo largo de mi vida.

A mi novia, Daniella Ramos, quien me ha alentado y ayudado durante las largas madrugadas que he dedicado a este proyecto.

A mis amigos más cercanos: Miguel, George, Luis y Gabriela, quienes nunca me han negado ayuda y son partícipes de los mejores recuerdos que me llevaré de esta etapa.

Guido Ramírez

AGRADECIMIENTOS

Agradezco de todo corazón a mi madre por demostrarme que, con perseverancia y muchas ganas, se puede salir adelante en la vida.

A mi mentor, Jonathan Cagua, que me ha enseñado a desarrollarme en el ámbito profesional y personal.

Al futuro ingeniero Carlos Jiménez, por su paciencia y disposición para guiarme en el campo de la inteligencia artificial.

Al Ing. Eduardo Castillo, MSc. Bryan Puruncajas y los profesores que aportaron con su conocimiento.

A mi amigo Guido Ramírez por la perseverancia de inicio a fin en el desarrollo de nuestro proyecto y a todas las personas que aportaron de distintas maneras.

Miguel Bailón

AGRADECIMIENTOS

Agradezco a mis padres, quienes me han dado sugerencias y consejos para la realización de este proyecto.

A mi tío, Vicente Romero, que con sus bastos años de experiencia en diseño de máquinas me ha ayudado a orientarme durante este trabajo.

Al Ph.D. Eduardo Castillo, M.Sc. Livingston Castro, y todos los profesores que amablemente pusieron a disposición su conocimiento para la guía de este proyecto,

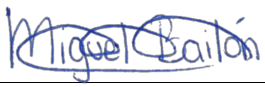
A los M.Sc. Efraín Terán y Bryan Puruncajas por su excelente aporte como profesor y tutor, respectivamente, de esta materia integradora.

A mi amigo y compañero Miguel Bailón y demás amistades forjadas a lo largo de la carrera que han colocado su granito de arena para saber todo lo que sé hoy.

Guido Ramírez

DECLARACIÓN EXPRESA

“Los derechos de titularidad y explotación, nos corresponde conforme al reglamento de propiedad intelectual de la institución; *Guido Alejandro Ramírez Espinoza* y *Miguel Alejandro Bailón Silva* damos nuestro consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"



Miguel Alejandro Bailón Silva

Guido Alejandro Ramírez Espinoza

EVALUADORES

Firmado electrónicamente por:
EFRAIN ANDRES TERAN

Efraín Terán, M.Sc.

PROFESOR DE LA MATERIA

Firmado electrónicamente por:
BRYAN JOAO PURUNCAJAS MAZA

Bryan Puruncajas, M.Sc.

PROFESOR TUTOR

RESUMEN

En el mundo, cerca de 4 millones de persona mueren al año debido a una baja calidad de aire en espacios cerrados. Además, una persona en promedio pasa el 90% de su día dentro de edificaciones o interiores. En dichos espacios se pueden encontrar distintas causas de contaminación, por combustión de combustibles y el uso de agentes químicos, lo que provoca partículas suspendidas en el aire y volátiles orgánicos; con efectos nocivos para nuestra salud. Debido a ello, se propone el desarrollo de un purificador de aire portable y personal asistido por inteligencia artificial para detectar las causas de contaminación en espacios cerrados.

Para el desarrollo de los objetivos planteados, se utilizó *software* de diseño para el modelamiento de las partes mecánicas y lineamientos de entidades especializadas como EPA y ASHRAE para la correcta selección de actuadores y etapa de filtrado. Para el diseño del *firmware* del sistema embebido se implementó el paradigma de programación reactiva y patrón de diseño de objetos activos. El entrenamiento del modelo de inteligencia artificial se realizó con los datos obtenidos de los sensores de calidad de aire y condiciones ambientales.

Se logró desarrollar exitosamente el diseño mecánico, *firmware* e inteligencia artificial de un prototipo inicial de un purificador de aire portable. El filtro posee una capacidad teórica de filtrado de aire del 99.97% y realiza 6 cambios de aire por hora del espacio personal del usuario. El sistema predice si existen contaminantes químicos o de combustión en el entorno con una precisión del 87%. Además, gracias al minucioso diseño de *firmware*, se logró un sistema embebido confiable y seguro.

Palabras Clave: calidad, contaminación, purificador, portable, predice, EFR32.

ABSTRACT

In the world, about 4 million people die every year due to poor indoor air quality. In addition, an average person spends 90% of their day inside buildings or indoors. In these spaces we can find different causes of pollution, due to the combustion of fuels and the use of chemical agents, which causes particles suspended in the air and organic volatiles, with harmful effects on our health. Due to this, the development of a portable and personal air purifier assisted by artificial intelligence is proposed to detect the causes of pollution in closed spaces.

For the development of the proposed objectives, design software was used for the modeling of the mechanical parts and guidelines of specialized entities such as EPA and ASHRAE for the adequate selection of actuators and filtering stage. For the firmware's design of the embedded system, the reactive programming paradigm and the active object design pattern were implemented. The training of the artificial intelligence model was carried out with the data obtained from the air quality sensors and environmental conditions.

The mechanical design, firmware, and artificial intelligence of an initial prototype of a portable air purifier was successfully developed. The filter has a theoretical air filtering capacity of 99.97% and performs 6 air changes per hour of the user's personal space. The system predicts whether there is chemical or combustion pollutants in the environment with an accuracy of 87%. Furthermore, thanks to careful firmware design, a reliable and secure embedded system was achieved.

Keywords: *quality, pollution, purifier, portable, predict, EFR32.*

ÍNDICE GENERAL

ABREVIATURAS

ASHRAE	American Society of Heating, Refrigerating and Air-Conditioning Engineers
OMS	Organización Mundial de la Salud
EPI	Environmental Performance Index
EPA	Environmental Protection Agency
HEPA	High Efficiency Particulate Air
MCU	Microcontroller unit
PCB	Printed Circuit Board
UART	Universal Asynchronous Receiver-Transmitter
I^2C	Inter-Integrated Circuit
LED	Light-emitting diode
RTOS	Real Time Operating System
IDE	Integrated Development Environment
IA	Inteligencia Artificial
HEPA	High Efficiency Particle Arresting

SIMBOLOGÍA

PM	Particulate Matter
NO_x	Óxido de Nitrógeno
VOC	Volatic Organic Compound
tVOC	Total Volatic Organic Compount
eCO_2	Equivalentes a dióxido de carbono
SO_2	Dióxido de Sulfuro
O_3	Ozono
CO	Monóxido de Carbono
µm	Micrómetro
m	Metro
ft	Pies
cfm	Cubic Feet per Minute / Pies cúbicos por minuto.
°C	Grados centígrados
ml	mililitro
W	Watts
V	Voltaje
RMT	Remote Control
VCD	Voltaje Corriente Directa
VCA	Voltaje Corriente Alterna
Pa	Pascales
inH_2O	Pulgadas de columna de agua.

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1

1. INTRODUCCIÓN

Según la Organización Mundial de la Salud (OMS), la contaminación del aire es un problema global que genera un riesgo a la salud ambiental del mundo. Además, representa alrededor de 8 millones de muertes al año, en donde el 47.5% de esas muertes, 3.8 millones, es a causa de contaminación del aire en espacios cerrados [1].

En el territorio ecuatoriano, las únicas ciudades que monitorean la calidad de aire son Quito, ubicada en la provincia de Pichincha, y Cuenca, Azuay [2]. Ambas ciudades utilizan estaciones de supervisión para determinar la concentración de contaminantes en los espacios abiertos, obviando los espacios cerrados, y evaluando que los niveles censados se encuentren por debajo de los niveles máximos establecidos por las normas ecuatorianas de calidad del aire [3].

Las personas se encuentran en un espacio cerrado alrededor de 90% del día, principalmente en la casa o en el espacio de trabajo, por lo que se encuentran expuestos a la contaminación de este tipo de ambiente [4]. Como consecuencia, se genera un riesgo para la salud humana debido a que la contaminación de espacios internos incrementa las tasas de morbilidad y mortalidad [5].

Dado a la vital importancia que tiene la calidad del aire en el bienestar y salud humana, el proyecto tiene como enfoque diseñar un purificador de aire personal, portable, de bajo costo e inteligente. De esta manera, se busca mejorar la calidad del aire en cualquier espacio cerrado y personal en donde se encuentre el portador y que el dispositivo actúe de acuerdo con la contaminación presente.

1.1 Descripción del problema

La OMS estima que durante la última década cerca del 91% de la población humana vive en un entorno donde la calidad de aire no es buena. Según sus lineamientos de calidad de aire se debe mantener anualmente $5\frac{\mu g}{m^3}$ de material particulado (*PM,* por sus siglas en inglés) de 2.5 µm de diámetro o menor. El material particulado, PM_x ,es un contaminante del aire formado mediante una mezcla de partículas sólidas y liquidas suspendidas en el aire y el subíndice *x* determina el tamaño de la partícula en µm.

En la figura 1.1 se puede observar que las zonas con mayor concentración de $PM_{2.5}$ en el planeta se concentra en la región asiática y medio oriente. Sin embargo, en otros países se observa que el nivel de *PM* supera los lineamientos dados por la OMS. La contribución de material particulado en el ambiente proviene de la contaminación de aire, la cual puede ser de origen exterior e interior.

La contaminación de aire exterior se debe mayormente a la actividad industrial humana, cuyas principales emisiones son de CO, SO_2 y NO_x. Su contraparte, contaminación proveniente de interiores (hogares, oficinas, entre otros), es debido al uso de combustibles sólidos, humo de tabaco, contaminantes de origen biológico y emisión de gases por debajo de la superficie de edificaciones.

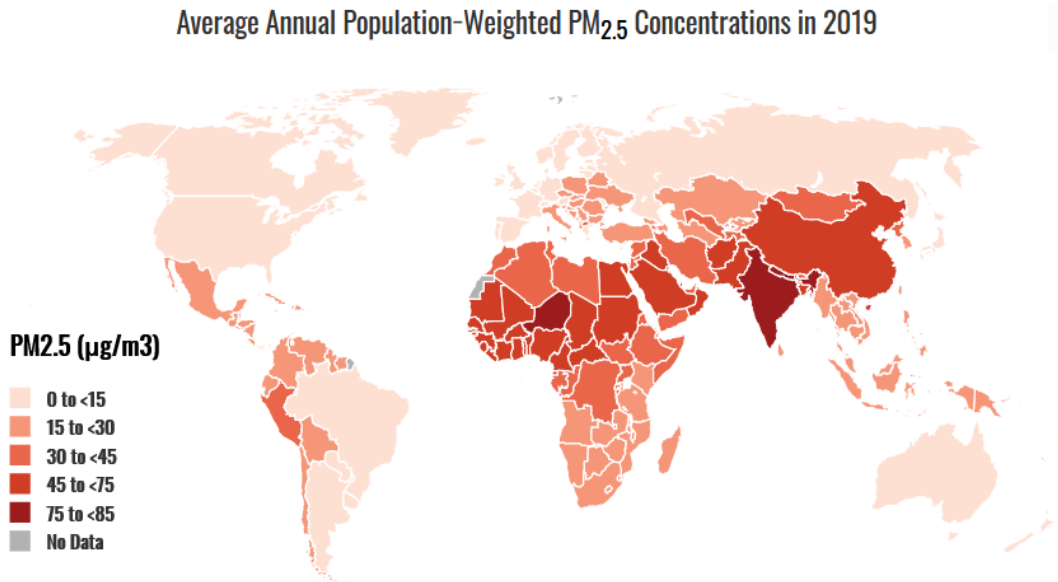


Figura 1.1 Concentración de $PM_{2.5}$ anuales en el mundo [6]

En la tabla 1.1, se muestra las fuentes de contaminación de interiores junto a sus contaminantes. Se destaca que el uso de combustibles sólidos y la combustión de tabaco aportan *PM*, CO y NO_x, tal como lo hacen fuentes de contaminación de aire exteriores. La OMS estima que en ciertas regiones del mundo más del 50% de la contaminación proviene de las zonas interiores. Cabe destacar que la exposición a *PM* y compuestos nocivos en el aire cuyo origen es de agentes exteriores también se da en ambientes interiores. Un ejemplo es China, donde entre el 66% y el 87% de las veces se cumple este caso [7].

Tabla 1.1 Fuentes de contaminación de interiores y sus principales contaminantes.

Materia particulada	$\boldsymbol{CO\ y\ NO_x}$	**Orgánicos volátiles**	**Biológicos**	**Pesticidas**	**Radón**
Combustible sólido, combustión de tabaco.	Combustible y tabaco.	Muebles, productos del hogar, combustible sólido y tabaco.	Muebles, Ductos de ventilación, zonas, áreas húmedas.	Productos del hogar, polvo del exterior.	Suelo debajo de la edificación.

Actualmente, la información provista de los sistemas de monitoreo de calidad de aire de las ciudades de Quito y Cuenca abarca los niveles de contaminación en espacios abiertos de las urbes. Dicha información se podría utilizar como referencia por la estrecha relación entre contaminación interna y externa y la filtración que existe del ambiente externo al cerrado.

El enfoque de este proyecto es disminuir la exposición individual de *PM* y agentes contaminantes en zonas interiores con el uso de filtros HEPA por su gran capacidad de filtrado de partículas entre 0.3 a 10.0 µm con un 99.97% de éxito [7]. Es esencial la correcta selección de actuadores mecánicos que permitan el abastecimiento de aire purificado dirigido para el espacio personal humano o en el mejor de los casos para un área interior de trabajo que no sobrepase los 15 m^2.

Además, se hará énfasis en el uso de sensores dedicados en la medición de *PM* y otros agentes contaminantes para que, aplicando inteligencia artificial (IA), el sistema pueda tomar decisiones de manera autónoma con base en los niveles de polución en el

entorno personal. Por último, es necesario un diseño mecánico adecuado de la forma del purificador, por su efecto en el flujo del aire, y tamaño, para la mayor portabilidad posible.

1.2 Justificación del problema

Según el EPI, la tendencia en más de la mitad de los países del mundo es de reducir ciertos tipos de contaminantes del aire. Sin embargo, naciones altamente desarrolladas como China, India e Indonesia, aportan anualmente 3.6%, 4.8% y 7.3% respectivamente al crecimiento de la concentración de polución en el aire, como se observa en las figuras 1.2 y 1.3. La contaminación en estos países es generada por la gran actividad industrial, incremento vehicular y otras actividades humanas, opacando al esfuerzo del resto de gobiernos [8].

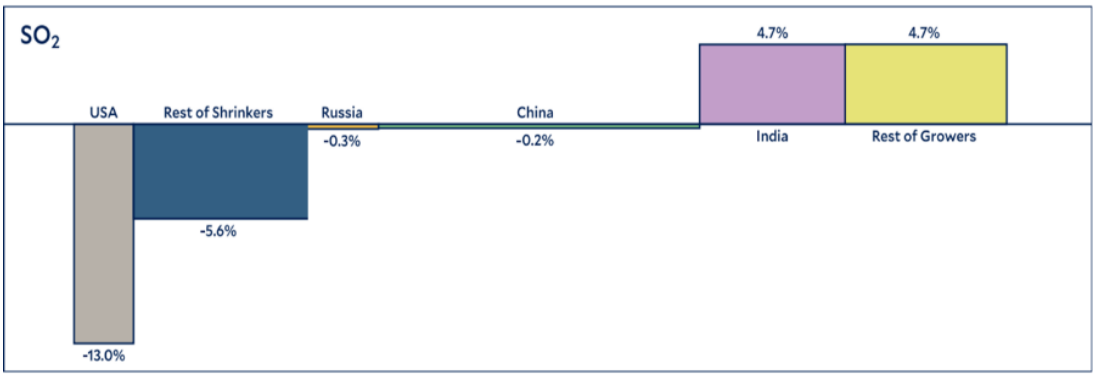


Figura 1.2 Comparación internacional de disminución del SO_2 en el mundo [8]

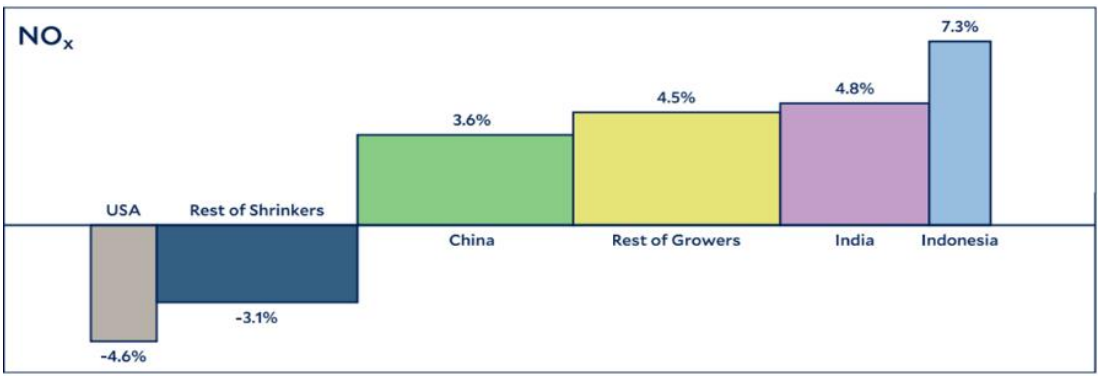


Figura 1.3 Comparación internacional de disminución del NO_x en el mundo [8]

Por tanto, la contaminación del aire es un problema de interés empresarial, educativo y de organizaciones de la salud. El motivo es el gran impacto generado sobre la salud, comodidad, bienestar y productividad del ser humano [3]. No obstante, las actividades

humanas cotidianas dentro de espacios cerrados favorecen al incremento y concentración de los contaminantes a través de distintas fuentes como pesticidas, solventes, agentes limpiadores, entre otros [1].

En el Ecuador, la relación entre la contaminación del aire y el impacto sobre la salud humana no está clara debido a pocos estudios realizados. No obstante, las cifras de muertes, según el Ministerio de Salud Pública, causadas por la polución en el aire son de alrededor de 1770 individuos anualmente [9].

Históricamente, el monitoreo de calidad de aire es costoso debido a la colocación de diversas estaciones y las complejas redes de recolección de datos en zonas trascendentales de la ciudad. Sin embargo, con el avance tecnológico la tendencia está cambiando con el uso de sistemas de supervisión a través de sensores que proveen datos a altas velocidades y en tiempo real. El constante desarrollo y lanzamiento de nuevas tecnologías dedicadas al monitoreo traen consigo costos menores y dispositivos fáciles de usar que dan la oportunidad para que incluso una persona natural pueda conocer la calidad de aire que lo rodea [10].

Existen Objetivos de Desarrollo Sostenible (ODS) enfocados en el medio ambiente y acciones frente al cambio climático. Aplicado a la temática de nuestro proyecto, la IA ha sido de gran beneficio por su capacidad de analizar a larga escala bases de datos y, de manera posterior, desarrollar acciones que apunten a preservar el ambiente. La inteligencia artificial es útil aplicado en el medio ambiente para la medición y almacenamiento de datos para la posterior toma de decisiones en base a modelos de identificación previamente programados [11].

1.3 Objetivos

1.3.1 Objetivo general

- Diseñar un purificador de aire portable e inteligente para mejorar la calidad del aire de un individuo en espacios cerrados.

1.3.2 Objetivos específicos

1. Diseñar la estructura mecánica de un purificador de aire portable junto a la selección del actuador y filtro para el correcto abaste de aire filtrado por el sistema.
2. Crear el *firmware* del purificador del aire portable utilizando el kit de desarrollo *Thunderboard*™ EFR32BG22, basado en la arquitectura ARM, aplicando programación reactiva y el patrón de diseño de objetos activos.
3. Desarrollar las librerías de la red de sensores de calidad de aire y el modelo de inteligencia artificial para la predicción del tipo de contaminante en el entorno mediante el procesamiento y validación de los datos adquiridos.

1.4 Marco teórico

1.4.1 Tipos de contaminantes presentes en el aire

La polución del aire puede tener su origen de zonas exteriores o interiores. La contaminación exterior penetra directamente y contribuye a la contaminación en interiores (hogares). En cuanto al segundo origen, éste posee un mayor impacto en la salud del ser humano debido al factor de exposición de PM_x [12].

De acuerdo con el factor de exposición de contaminantes, periodo de tiempo en el que una persona se encuentra expuesta a la polución del aire, se ha determinado que una porción de contaminación interior puede causar un daño mayor que la polución que se encuentra puertas afuera.

1.4.2 Tipos de riesgos a la salud humana

La contaminación del aire puede afectar a sistemas importantes del cuerpo humano, desde infecciones hasta daño a pulmones, corazón, e incluso el cerebro. Estos efectos adversos pueden ser tanto agudos como crónicos e incluso llevar a la muerte [13].

La exposición a $PM_{2.5}$ se le ha asociado con una lista de impactos para la salud y su respectiva abreviación en inglés: Enfermedad de Obstrucción pulmonar crónica (COPD), Enfermedad Aguda de las Vías Respiratorias (ALRI), Enfermedad Cerebrovascular (CEV), Enfermedad Isquémica del corazón (IHD) y cáncer de pulmón (LC) [14].

En la tabla 1.2 se aprecia el incremento de muertes prematura atribuidas a la exposición de material particulado. El número de muertes en el mundo aumentaría en un 33% para el 2050, dando un total de 3 millones de muertes prematuras adicionales. Por otra parte, se observa que la muerte prematura debido a enfermedades asociadas con la contaminación del aire se duplica.

Tabla 1.2 Muertes prematuras relacionadas al $PM_{2.5}$ y O_3 para la población mayor a 5 años [14]

Región	Año	Población ($x10^6$)	Mortalidad atribuible a la contaminación del aire (muertes x 10^3)						
			PM 2.5					O3	Total
			ALRI < 5 años	IHD > 30 años	CEV > 30 años	COPD > 30 años	LC > 30 años	COPD > 30 años	
África	2050	1807	158	185	262	38	5	13	660
América		1191	0	75	15	7	11	11	119
Mediterráneo Oriental		1021	66	321	246	37	13	40	723
Europa		886	1	307	156	18	37	11	530
Sureste de Asia		2332	104	865	807	419	48	227	2470
Pacifico Oeste		1861	16	413	1120	309	155	57	2070
El mundo		9098	346	2166	2604	828	270	358	6572

1.4.3 Niveles de contaminación del aire permisibles

1.4.3.1 Según la OMS

Las directrices propuestas por la OMS acerca de la Calidad del Aire, actualizadas en el presente año, son aplicadas a nivel global. A través de las directrices, se busca disminuir alrededor de un 15% de las muertes causadas por la contaminación ambiental. Los lineamientos establecidos se basan en evaluaciones realizadas por expertos en base a distintos contaminantes, presentados en la tabla 1.3.

Tabla 1.3 Tipo de contaminante, unidad, tiempo promedio de exposición y nivel de concentración máximo recomendado según los lineamientos de la OMS 2021 [15]

Contaminante	Unidad	Tiempo promedio	Nivel de concentración máximo recomendado
$PM_{2.5}$	µg/m^3	Anual	5
		24 horas	15
PM_{10}	µg/m^3	Anual	15
		24 horas	45
O_3	µg/m^3	Promedio 6 meses	60
		8 horas	100
NO_2	µg/m^3	Anual	10
		24 horas	25
		1 hora	200
SO_2	µg/m^3	24 horas	40
		10 minutos	500
CO	µg/m^3	24 horas	4
		8 horas	10
		1 hora	35
		15 minutos	100

1.4.3.2 Según la norma ecuatoriana de calidad de aire

La norma ecuatoriana de calidad de aire difiere con los niveles de concentración máximo recomendado según los lineamientos de la OMS, 2021. No obstante, la OMS establece los lineamientos internacionales para disminuir los perjuicios provocados por la contaminación del aire, tanto del entorno externo como en espacios cerrados. Es por ello por lo que se utilizara como guía el estándar propuesto por la OMS.

Tabla 1.4 Tipo de contaminante, unidad, tiempo promedio de exposición y nivel de concentración máximo recomendado según los lineamientos de la Norma Ecuatoriana de calidad del aire [16]

Contaminante	Unidad	Tiempo promedio	Nivel de concentración máximo recomendado
$PM_{2.5}$	µg/m^3	Anual	15
		24 horas	50
PM_{10}	µg/m^3	Anual	50
		24 horas	100
O_3	µg/m^3	8 horas	100
NO_2	µg/m^3	Anual	40
		1 hora	200
SO_2	µg/m^3	24 horas	125
		10 minutos	500
CO	µg/m^3	8 horas	10
		1 hora	30000

1.4.4 Tipos de filtros de aire en el mercado

Para fines de este proyecto se tomará en consideración los tipos de filtro comúnmente utilizados en el área de ambientación y filtrado de aire. De este tipo destacan los filtros HEPA y Carbón Activado.

HEPA

Según EPA, los filtros de alta eficiencia de aire particulado o HEPA por sus siglas en inglés, es un tipo de filtro mecánico que teóricamente es capaz de cumplir su función con un 99.97% de éxito. Los filtros HEPA son ampliamente utilizados porque son capaz de retener polvo, polen, moho, bacterias y cualquier partícula con tamaño mínimo de 0,3 micrones. Cabe destacar que este tipo de filtros requiere limpieza periódica y un posterior remplazo una vez su vida útil haya sido alcanzada.

Se considera un filtro por excelencia por su alta capacidad de filtrado. Por otra parte, los filtros HEPA no son capaces de retener partículas de un diámetro inferior al mencionado; partículas como gases, olores y moho, no son retenidos por el filtro. Además, un problema común de los filtros HEPA es que su tiempo de vida es reducido debido a la retención de partículas con diámetros grandes. En la tabla 1.3 se puede observar la clasificación del filtro HEPA por ASHRAE, categorizándolo en la escala MERV.

Tabla 1.3 Clasificación filtros HEPA en categoría MERV [17]

Categoría MERV	Eficiencia según tamaño promedio de partícula.
1-4	3.0 – 10.0 menos del 20%
6	3.0 – 10,0 49.9%
8	3.0 – 10.0 84.9%
10	1.0 – 3.0 50% - 64.9%, 3.0 – 10.0 85% o mayor
12	1.0 – 3.0 80% - 89.9%, 3.0 – 10.0 90% o mayor
14	0.3 – 1.0 75% - 84%, 1.0 – 3.0 90% o mayor
16	0.3 – 1.0 75% o mayor

Filtros de carbón activado

Los filtros de carbón activado son eficientes al momento de remover olores. Estos son capaces de retener compuestos volátiles orgánicos, gases y vapores. Actúan como una esponja para gases dañinos y vapores.

1.4.5 Monitoreo de la calidad del aire

En la figura 1.4 se ilustra de como el uso de estaciones de monitoreo y redes complejas de compilación de datos no requieren de tantos despliegues de equipo. Sin embargo, se requiere de una cantidad elevada de datos de calidad y su costo es mayor. Es observable como el monitoreo personal presenta un costo relativo menor y requiere de menos datos de calidad para el monitoreo de contaminación de aire. Esta última estrategia nos lleva al concepto de ciencia ciudadana, es decir, convencer a las personas en realizar observaciones individuales y recolectar datos. De esta forma no solo se logra una red de monitoreo relativamente barata, sino la conciencia colectiva de la polución presente en el entorno diario.

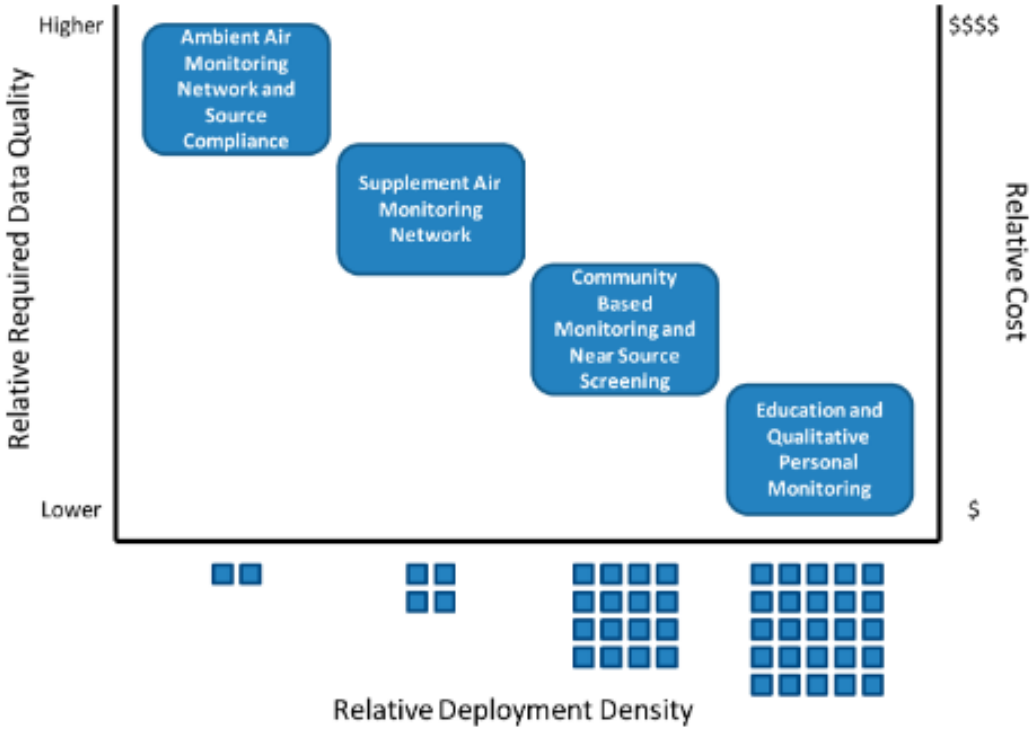


Figura 1.4 Requerimientos de datos, densidad de despliegue y costo por aplicación [10]

Los sensores de calidad de aire con aplicación de IA nos llevarán a un sistema capaz de tomar decisiones al respecto a un ambiente que sea considerado poco

saludable, en cuanto a polución se refiere, a través del uso de actuadores especializados en la purificación del aire. Una gran densidad de dispositivos personales desplegados para el monitoreo de aire conlleva a la conciencia colectiva de nuestro entorno y nos lleva a tomar acciones sobre él.

1.4.6 Estado del arte

Antes, al hablar de sistemas de monitoreo de calidad de aire se esperaba el uso de sistemas caros, estáticos y complejos. No obstante, con el desarrollo de la tecnología en informática y electrónica, dan como resultado sensores de gran utilidad y tamaños reducidos.

Como se mencionó en secciones anteriores, el paradigma del monitoreo de aire busca suplir la necesidad de encontrarse con un ambiente con bajos niveles de contaminación con tecnología de bajo costo, fácil de usar y alta portabilidad de los sensores para la adquisición de datos en tiempo real.

W-Air

W-Air, es un proyecto sueco que consiste en un dispositivo vestible, capaz de monitorear la contaminación del aire alrededor de una persona. El proyecto se enfocó en la integración de sensores de gas a bajo costo, acoplable a distintas prendas como bandas, cinturones, etc.; e integración de sensores ambientales para mejorar la usabilidad en ambientes relacionados a aplicaciones fisiológicas.

La propuesta de W-air nace de un dispositivo personal para un monitoreo multivariable de agentes contaminantes tomando en cuenta la interferencia humana. W-air da como resultado un dispositivo con un esquema de calibración para la detección de niveles de O_3 en ambientes exteriores y CO_2 en interiores. El método de calibración se basa en una red neuronal que usa los datos adquiridos por dos sensores de óxidos metálicos gaseosos y un sensor de temperatura para estimar correctamente la concentración de los compuestos mencionados [18]. En la figura 1.5 se observa la placa W-Air montado en una banda para muñecas.

Figura 1.5 Uso de W-Air como banda para muñeca [18]

UMI

UMI es el nombre del purificador de aire portable propuesto por el departamento de electrónica de la Universidad Técnica de Sofia. El proyecto surge como una solución para mejorar la calidad del aire en ambientes cerrados como oficinas y salas de estar. A través de indicadores, eficiencia de recolección de partículas de paso simple y el rendimiento basado en la capacidad del purificador en reducir los PM, se determinó la buena capacidad de purificación del producto.

El proyecto se conformó a través de distintas etapas: diseño mecánico, configuración de los filtros, diseño electrónico, sensores, comunicación, microprocesador, *display* e intercomunicaciones. Para el diseño mecánico, se siguió los lineamientos provistos por la compañía Radiation Protection Systems Company, para maximizar el flujo del aire y su filtrado. En la figura 1.6 se observa el diseño mecánico.

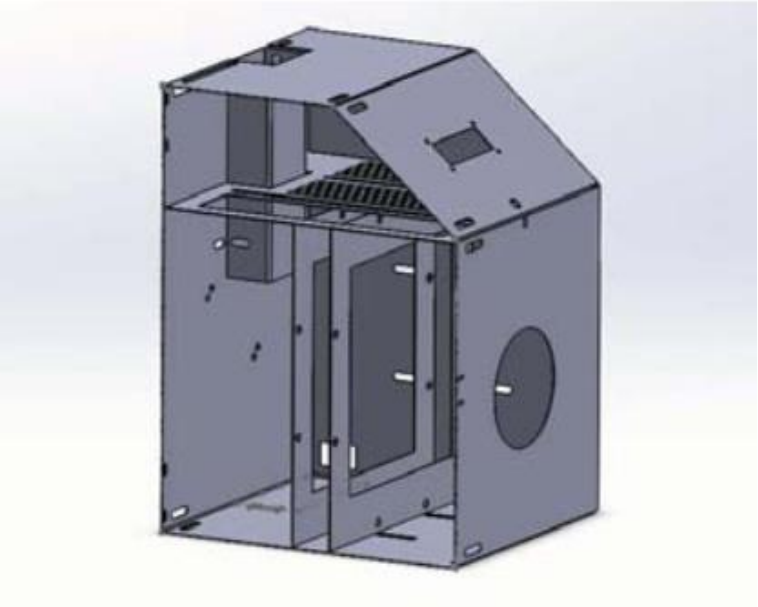


Figura 1.6 Diseño mecánico de UMI [19]

En la etapa de filtrado, se usó una configuración de tres etapas compuestas por el prefiltro, filtro de carbón activado y filtro HEPA. Su propósito es de capturar las partículas de gran tamaño, visibles al ojo humano, en la etapa de prefiltro. Seguido, el filtro de carbón activado se utilizó para capturar malos olores, VOC y humo. Finalmente, en la última etapa el filtro HEPA captura las partículas con un diámetro mayor o igual a 3 µm; todo esto se puede observar en la figura 1.7.

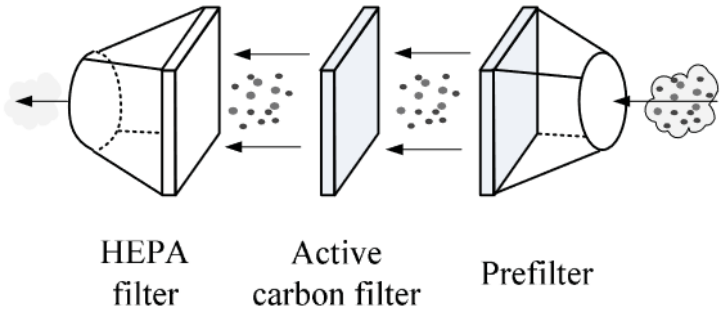


Figura 1.7 Sistema de filtrado de UMI [19]

Por último, el sistema electrónico se compone del microcontrolador Arduino UNO que captura los datos de temperatura y humedad del sensor DHT y utiliza, como sensor principal, para medición de contaminación del aire, el SDS011 que tiene un rango de medición de 0 – 999,9 µg/m3. Finalmente, el microcontrolador muestrea las mediciones

de los sensores y transmite la información hacia la pantalla LCD [19]. Esto se refleja en el diagrama de bloques en la figura 1.8.

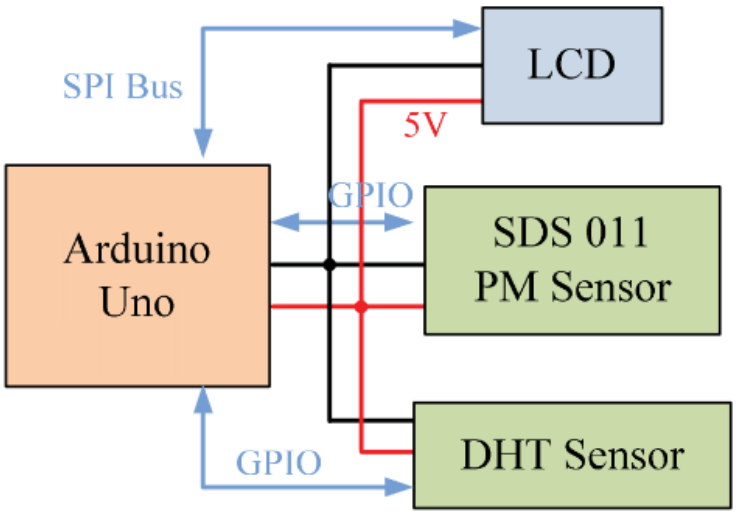


Figura 1.8 Diagrama de bloques del sistema electrónico de UMI [19]

En base a los anteriores proyectos que abordan el monitoreo y filtrado de aire respectivamente se propone una alternativa a nuestra problemática. Se unirá el monitoreo de calidad de aire empleando inteligencia artificial y el diseño de una estructura purificadora de aire. Nuestro proyecto se diferencia en: dar énfasis en un producto altamente portable, siguiendo el paradigma actual de monitoreo; y como método de clasificación a distintos escenarios, un modelo de IA previamente entrenado en base a sensores destinados a la medición de calidad de aire.

CAPÍTULO 2

2. METODOLOGÍA

A lo largo del capítulo dos se detalla el procedimiento para llevar a cabo el diseño mecánico y programación del *firmware* de un purificador de aire. Para el desarrollo de la alternativa seleccionada se tuvo en consideración los requerimientos y limitaciones del cliente: tamaño estructural, espacio de acción, estilos de programación; además, la selección de elementos adecuados: actuadores, filtros y juntas mecánicas. Cabe destacar, que se solicitó, sin influir en la alternativa, la inclusión del *firmware* diseñado para la recolección de datos y posterior carga del modelo de inteligencia artificial.

2.1 Selección de la alternativa de solución

A partir de la problemática planteada en el capítulo anterior, se proponen tres alternativas para el diseño mecánico del purificador de aire, junto a la posible selección de su extractor (ventilador) y filtros (HEPA).

- **Alternativa 1:**

  Diseño de un purificador de aire tipo extractor grande. Usando un ventilador axial y un filtro HEPA MERV 19 rectangular.

- **Alternativa 2:**

  Diseño de un purificador de aire tipo torre mediano usando un ventilador centrifugo y dos filtros HEPA MERV13 rectangulares.

- **Alternativa 3:**

  Diseño de un purificador de aire pequeno usando un ventilador axial y un filtro HEPA MERV13 en forma de cartucho.

Para la selección de la alternativa fue necesario establecer criterios de selección y su prioridad, los cuales pueden ser encontrados en la Tabla 2.1. Se da a conocer una breve explicación de cada criterio:

- **Costo:** El costo abarca la selección del extractor, filtro HEPA y cotización para la realización de la estructura.

- **Portabilidad:** La portabilidad abarca el tamaño y facilidad de traslado de ubicación.
- **Facilidad de uso:** Accesibilidad del diseño para acceder al filtro y poder cambiarlo periódicamente,
- **Volumen purificado:** Capacidad máxima del actuador para purificar cierto volumen de aire en cierto tiempo.
- **Distribución de espacio:** Espacio suficiente para la colocación de componentes electrónicos dentro de la estructura.
- **Consumo energético:** Potencia eléctrica consumida por los actuadores.

Tabla 2.1 Criterios de selección para alternativas

Criterios de selección			
Peso	**Criterio**	**Rango de importancia**	**% de decisión**
3,00	**Costo**	1	20,69
3,00	**Portabilidad**	2	20,69
2,00	**Facilidad de uso**	3	13,79
3,00	**Volumen purificado**	4	20,69
2,00	**Distribución de espacio**	5	11,76
1,50	**Consumo energético**	6	10,34
14,50	**Total**		**100,00**

Posterior al análisis de las alternativas en base a los criterios mencionados, se optó por escoger la alternativa 3 como la más adecuada para cumplir los requerimientos del cliente y dar solución a la problemática.

Esta solución permite alcanzar un costo viable por las características y proporción de los actuadores y filtros necesarios; y se asegura un bajo consumo energético, facilidad de uso y portabilidad.

Por otra parte, se sacrifica un poco la cantidad de volumen purificado y distribución de espacio en el diseño de tal manera para poder optimizar la mayoría de los parámetros. Los resultados se pueden apreciar en la Tabla 2.2.

Tabla 2.2 Resultados de las alternativas según los criterios de selección

	Criterio							Resultados	
Peso	3	3	2	3	2	1,5			
Opciones	**Costo**	**Portabilidad**	**Facilidad de uso**	**Volumen purificado**	**Distribución de espacio**	**Consumo energético**	**Puntaje sin peso**	**Puntaje con peso**	**Prioridad**
Alternativa 1	**0,5**	**1**	**1**	**3**	**3**	**1**	**9,5**	**23**	**3**
Alternativa 2	**1,5**	**2**	**1,5**	**2**	**2,5**	**2**	**11,5**	**27,5**	**2**
Alternativa 3	**3**	**3**	**2**	**1**	**2**	**3**	**14**	**33,5**	**1**

2.2 Requerimientos y limitaciones del diseño

Se necesitó múltiples reuniones previas con el cliente para definir los requerimientos y limitaciones incluidas en el diseño mecánico y de *firmware*. En la Tabla 2.3 se describen los requerimientos acordados con el usuario.

Tabla 2.3 Requerimientos del diseño mecánico

Filtrado	$PM_{2.5}$ o mayores, polvo, humo corriente, humo de tabaco, bacterias y algunos virus.
Área de acción	Mínimo que se filtre el volumen que ocupa el espacio personal de una persona.
Portabilidad	El diseño debe ser fácilmente transportado de una habitación a otra.
Alimentación	Se debe hacer uso de baterías dentro del espacio designado para las placas para energizar temporalmente el sistema.
Costos	Costo de fabricación rentables para ofrecer un precio competitivo a productos similares.
Espacio	Se solicitó dejar espacios dentro del diseño estructural reservados para el cliente.

Tabla 2.4 Requerimientos de *firmware* e inteligencia artificial

Lenguaje de programación	La programación del proyecto debe estar escrito en C.
Kit de desarrollo	Las pruebas deben ser realizadas en el kit de desarrollo *Thunderboard*^{TM} EFR32BG22. Cuyo microcontrolador, MCU, es el Cortex-M33 de 32 bits, basado en la arquitectura ARM.
Modularidad	El proyecto tiene que aplicar: programación reactiva y el patrón de diseño de objetos activos.
Sensores	Los sensores por utilizar serán provistos por el cliente: SDS011, sensor de PM, y SGP30, sensor de tVOC y eCO2.
Inteligencia artificial	El modelo de inteligencia artificial para reconocimiento de contaminantes deberá ser embebido en el kit de desarrollo.
Documentación	La documentación, en inglés, de todas las funciones públicas de las librerías y componentes creados.

2.3 **Diseño conceptual**

En la figura 2.1 se muestran los espacios que debe considerar el diseño mecánico del purificador de aire. Se ha contemplado un espacio para colocar las placas electrónicas. El mayor espacio es destinado al filtro HEPA, el cual también corresponde a la entrada de aire sin filtrar del sistema. Se destina un espacio considerable al ventilador axial y por último a la salida del aire filtrado.

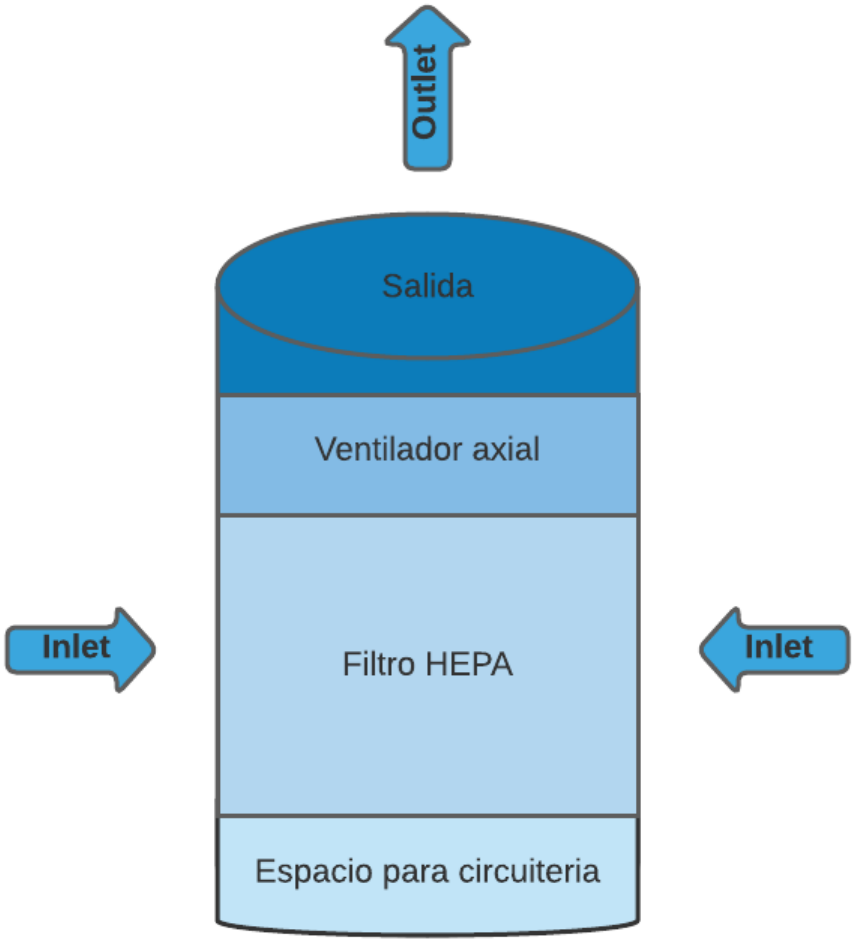


Figura 2.1 Particiones internas del purificador de aire portable

Para el desarrollo de la sección del filtro HEPA y la salida se tuvo en cuenta la forma de los orificios por los cuales entra el fluido, debido a que esto influye en la posterior selección del extractor. El espacio de circuitería debe contar con dimensiones y acoples mecánicos para la colocación y sujeción de una placa y componentes electrónicos adicionales. Como se mencionó anteriormente, la sección destinada al alojamiento del ventilador axial está en función de los requerimientos de presión del filtro HEPA, entradas y salidas de fluido.

Para el monitoreo de calidad de aire, el kit de desarrollo se comunicó con el sensor de PM, SDS011, a través del protocolo de comunicación UART. El sensor de tVOC y eCO_2 , SGP30, y el de temperatura y humedad, Si7021, se comunican utilizando la interfaz I^2C.

Además, se indicó visualmente la predicción realizada por el modelo de IA al encender y apagar uno de los tres LEDs mediante una señal digital a través del GPIO correspondiente. Cada LED representa el contaminante identificado en el ambiente: normal, agente químico y combustión.

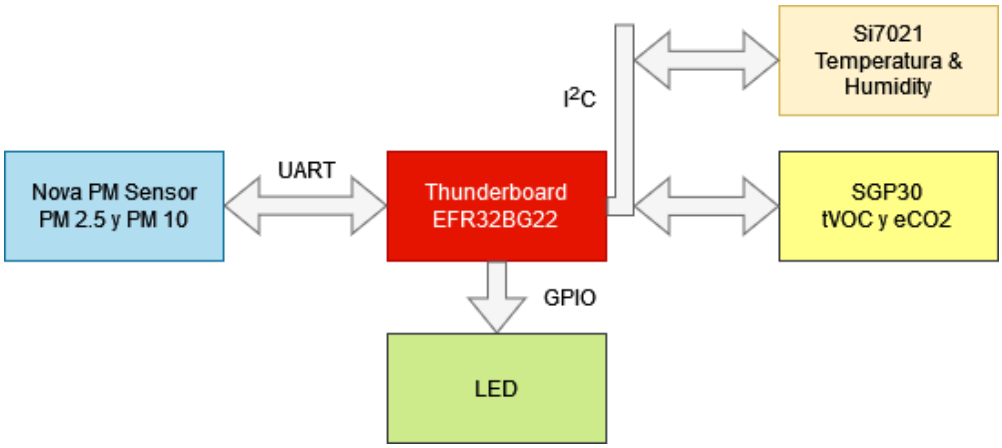


Figura 2.2 Diagrama de bloques de la interacción del kit de desarrollo y los periféricos

Para la comunicación con los sensores SGP30 y SDS011 se crearon las librerías para iniciar la comunicación, transmisión y recepción de los paquetes de datos de manera segura al verificar el CRC. Para el sensor Si7021 y control de los GPIO se utilizaron las librerías de Silicon Labs.

Para la programación se implementó el flujo de trabajo a través de datos asíncronos denominado programación reactiva. Por lo que se requiere cumplir una serie de parámetros enlistados a continuación [20]:

- Una fuente de eventos.
- Un sumidero de eventos.
- Un mecanismo para añadir suscriptores y sus fuentes de eventos.
- Al momento de llegar datos a las fuentes, los suscriptores son notificados.

En la figura 2.3 se puede observar el flujo que se implementará para cumplir con las características de la programación reactiva.

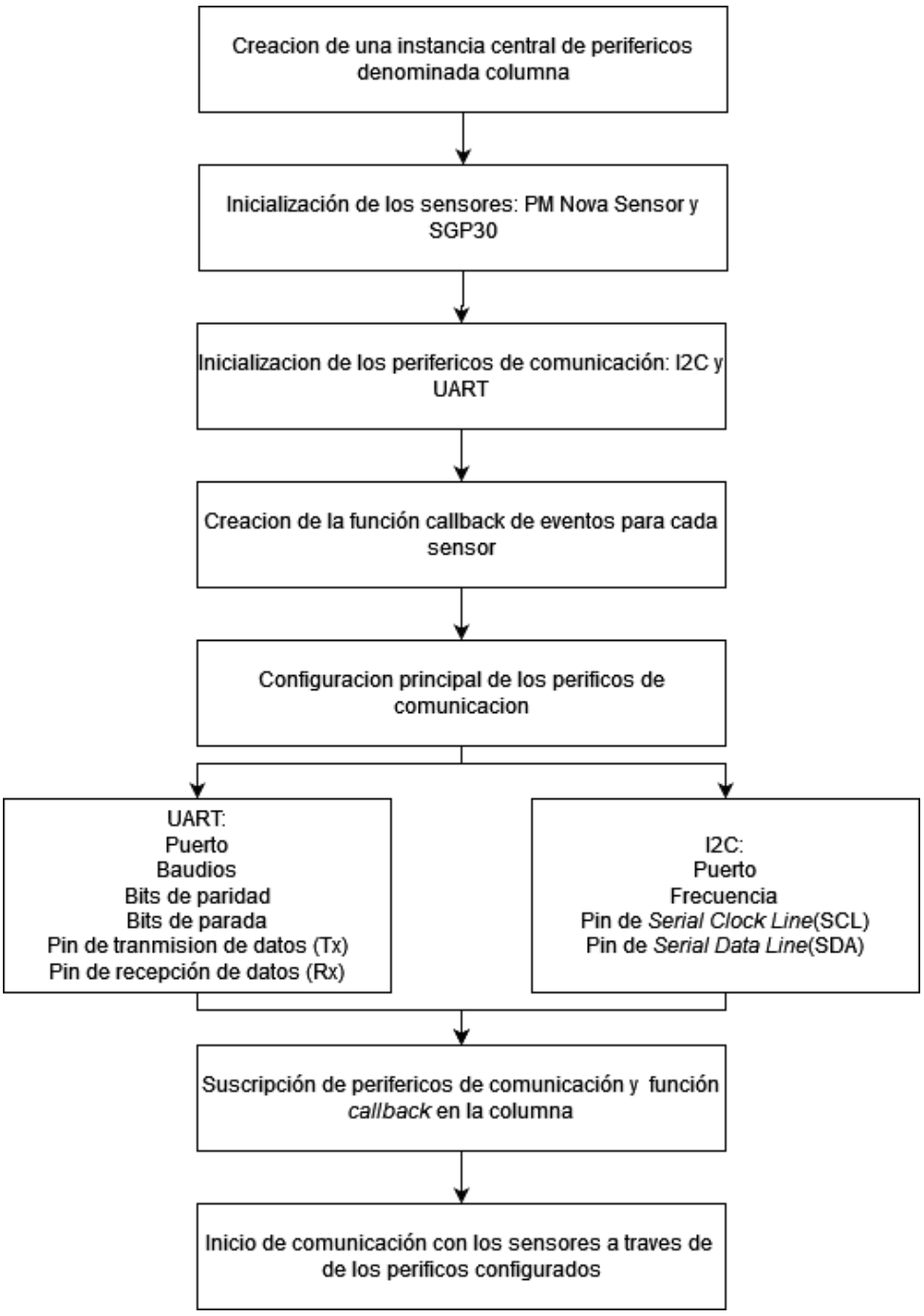


Figura 2.3 Diagrama de flujo de inicio y configuración de los sensores y sus periféricos de comunicación

Adicionalmente se implementó el patrón de diseño de objectos activos que restringe el uso especifico de hilos, denominados también subprocesos o tareas, en el RTOS del sistema embebido. Su correcta ejecución permite incluir el paradigma impulsado por eventos en donde los datos se mantienen privados, aislados, enlazados con el subproceso, como se observa en la figura 2.4, y la comunicación entre tareas se da por eventos asíncronos y se organizan los hilos a través de “publicación de eventos” [21].

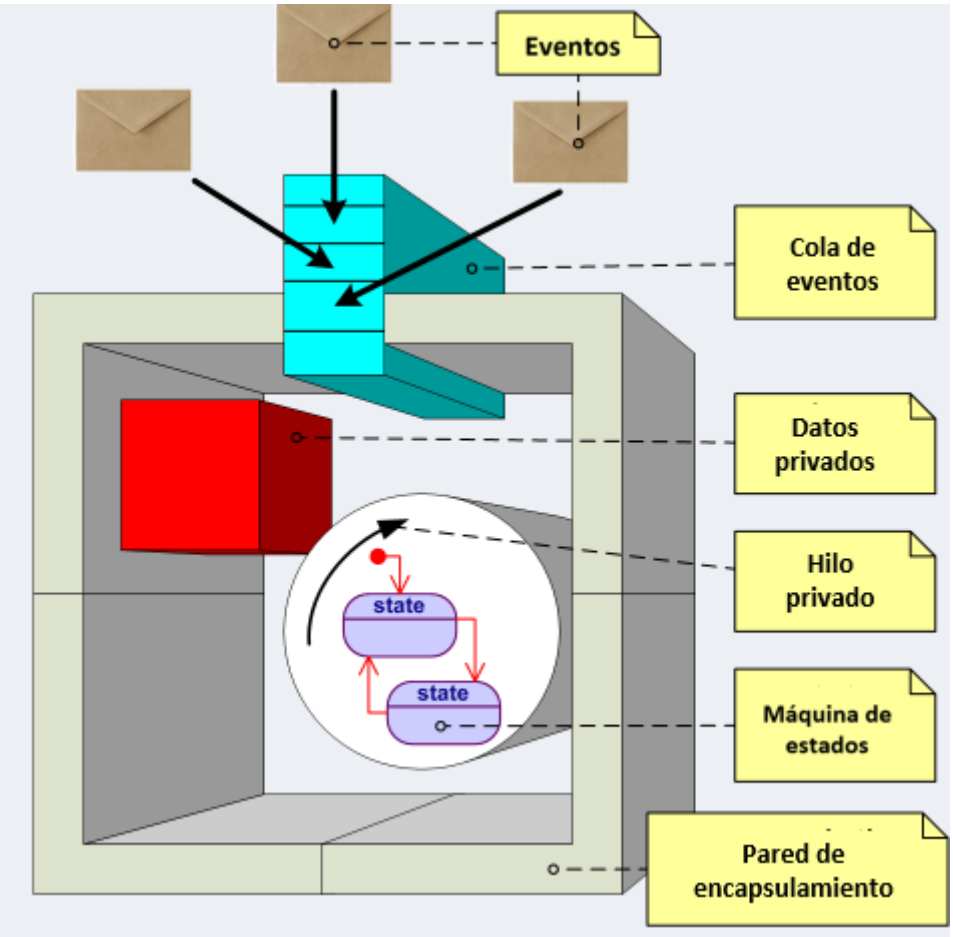


Figura 2.4 Visualización de un componente activo, sus partes y comunicación [22]

2.4 Parámetros de diseño mecánico

Como se mencionó en los requerimientos y limitaciones de la sección 2.2, se requiere purificar al menos el área de espacio personal del usuario. Para el diseño mecánico y selección de actuadores de un purificador de aire se identificó el área de aire que se desea purificar.

El espacio personal de una persona ronda entre los 40 a 120 *cm* de diámetro [23]. Partiendo de ese hecho, se obtuvieron los siguientes datos en la Tabla 2.5 El volumen calculado se basa en una habitación no mayor a 8 pies de altura, tal como indica la EPA [24] .

Tabla 2.5 Parámetros de área y volumen de espacio a purificar

Parámetro	Medidas
Área espacio personal	50 [ft^2]
Volumen	320 [ft^3]

En la tabla 2.6 se observa la cantidad de tasa de aire limpio enviado en pies cúbicos por minuto o cfm de acuerdo con el área de la habitación que se requiere purificar. Con estos datos, se interpoló para el área de nuestro interés de 50 ft^2. Se necesitaría 32 cfm para abastecer el espacio personal de una persona.

Tabla 2.6 Lineamientos de EPA para condicionamiento de aire [24]

Tamaño del purificador de aire para remoción de partículas						
Área de la habitación [ft^2]	100	200	300	400	500	600
CADR [*cfm]*	65	130	195	260	325	390

Además de conocer el CADR mínimo necesario, se consideró que cada espacio necesita una renovación de aire por hora para determinar que efectivamente se está brindando aire purificado.

Para aquello es necesario basarse en los lineamientos de la ASHRAE; la cual indica que, para espacios interiores como oficinas o escuelas, se necesita de 5 a 6 cambios de aire por hora o ACH por sus siglas en inglés [17].

Para verificar que nuestro CADR cumple con los ACH sugeridos basta se calculó los cambios de volumen en minutos y de manera consecuente, se obtienen los valores por hora. Estos cálculos se resumen en la Tabla 2.7.

Tabla 2.7 Cambios de aire

Parámetro	Valor
Tiempo de cambio de volumen [min]	10
Cambios de volumen por hora [u]	6

2.4.1 Selección de filtro HEPA

La mayoría de los proveedores de filtros HEPA para purificadores prefieren un servicio personalizable. Para nuestro caso, basado en el estado de arte mostrado en capítulos anteriores, se necesita que el filtro HEPA tenga un prefiltro y un filtro activado de carbono para aumentar su duración y eficiencia. Además, es necesario que el filtro venga en presentación de cartucho debido a la estructura que se planea diseñar.

Shenzen Lvchuang Co. Ltd nos cotizó un filtro HEPA con los requerimientos mencionados además de pruebas realizadas y características en la Tabla 2.8.

Tabla 2.8 Producto filtro HEPA MERV 13

Parámetros del producto *Shenzhen Lvchuang* MERV13		
Marca	Línea verde de Shenzhen	
Parámetro	Unidad	Standard
Clasificación	NA	MERV13
Eficiencia de filtración	%	90
Resistencia a la humedad	%RH	≤90
Resistencia instantánea a la temperatura	°C	≤100
Resistente a la temperatura	°C	≤80
Tamaño de partícula	um	≥0.5/0.8
Volumen de aire nominal	m^3/h	1000
Resistencia inicial	m/s	20-28pa
Resistencia final	m/s	120pa
Caída de presión de ventilación	$300m^3/h$	28pa
Material		PE+PPT

Se optó por un filtro HEPA de clasificación MERV 13 por la ASHRAE debido a su eficiencia moderada frente a partículas menores a 0.3 *μm*, tamaño promedio de algunos virus comunes en el ambiente, manteniendo una caída de presión y costo aceptable. Además, posee una resistencia inicial de 28 Pa en el filtro HEPA.

Para la incorporación de filtros HEPA en purificadores de aire portables, la resistencia inicial es una base esencial, debido a que cuando la vida útil del filtro llega a su fin, esta resistencia se duplica [25].

Esto conlleva a tomar en cuenta un factor de seguridad del doble de la resistencia inicial, sin embargo, se tiene dos capas más de filtración, prefiltro y carbón activado. En base a esto, y con un factor de seguridad de 2.5, nuestra resistencia final a vencer por el extractor queda expresado en la Tabla 2.8. En dicha tabla, se muestra también el consumo aproximado que debe tener el actuador; este valor es calculado en base a la resistencia final y al caudal necesario mencionado una sección atrás.

Tabla 2.9 Resistencias del filtro HEPA

Parámetro	Valor	
Resistencia inicial	28 *[Pa]*	0.11 *[inH_2O]*
Resistencia con factor de seguridad (2.5)	70 *[Pa]*	0.28 *[inH_2O]*
Potencia necesaria	1.1 [*W*]	

2.4.2 Selección de ventilador

La selección del ventilador marca una restricción en el tamaño del diseño mecánico debido a que debe ser capaz de acoplarse dentro. El ventilador seleccionado tiene una presión estática superior a la resistencia final que se espera vencer debido al filtro HEPA, un flujo de aire mayor al previsto y una potencia superior a la necesaria mostrada en la sección anterior.

El ventilador seleccionado corresponde a un tipo axial, esto debido a que la resistencia exigida por el sistema no es significativa, por ende, es ideal para nuestro

objetivo [26]. En la Tabla 2.10 se muestra un resumen de las características del ventilador seleccionado.

Tabla 2.10 Características del ventilador axial

Característica	Descripción
Voltaje nominal	12 *VDC*
Corriente de entrada	0.84 *A*
Poder de entrada	10.08 *W*
Máximo flujo de aire	50.40 cfm
Máxima presión estática	0.926 inH_2O
Cables	Negro: Negativo Rojo: Positivo Azul: Salida de Tacómetro Amarillo: Control de velocidad (PWM)

Varias especificaciones técnicas son mostradas en el apéndice B. Es importante recalcar que el flujo de aire supera por mucho el mencionado en secciones anteriores. Esto toma relevancia en futuras secciones.

2.4.3 Selección de forma de la entrada y salida del purificador

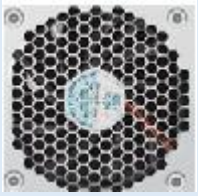
Para la entrada de fluido de nuestro sistema se debe tener en consideración que se trata de una placa agujerada por la cual pasará un fluido. La caída de presión de una placa agujerada depende del porcentaje de espacios abiertos en esta. A mayor porcentaje de espacio agujerado, menos caída de presión.

La estimación mencionada se basa en ensayos de laboratorio donde se llega a concluir que con perforaciones del 68% del área, se tienen caídas de presiones de 8 Pa y 7.5 Pa al 80%. En base a lo anterior, se decidió realizar el diseño de la entrada de fluido con un porcentaje de perforaciones entre 68% a 80% [27].

Para la salida, se tendría una caída de presión baja si se sigue los lineamientos anteriores. Sin embargo, se debe considerar la velocidad de salida del sistema a través del patrón o malla para asegurarnos un correcto abaste del volumen que se quiere remplazar. En la Tabla 2.11 se puede ver la razón de velocidad final basado en la velocidad nominal de un ventilador a través en diferentes patrones de mallado.

Se puede observar que el más adecuado para tener una velocidad de salida aproximada a un sistema sin restricciones es el tipo parrilla o alambre, con un 84% de ratio de velocidad.

Tabla 2.11 Respuesta de tipos de malla en un ventilador axial [28]

Estilo de malla	Figura	Razón de extracción de aire	Velocidad	Razón de velocidad
Sin malla		100%	3.1 m/s	100%
Alambre		71%	2.6 m/s	84%
Panel de abejas		72%	2.4 m/s	77%
Cuadricular		60%	2.2 m/s	71%
Perforaciones redondas		29%	1.1 m/s	35%

Conociendo que nuestra salida disminuirá en un 14% la velocidad del actuador, se debe asegurar que nuestro actuador tenga un flujo mayor para compensar esta perdida.

2.4.4 Consideraciones adicionales

Este apartado es para mencionar consideraciones de espacio dentro del diseño debido a la inclusión de componentes electrónicos adicionales para la alimentación autónoma del actuador y los sensores (figura 2.5 y 2.6). En el apéndice B se puede observar las medidas de los sensores que, de manera relevante, requieren un espacio propio en el diseño.

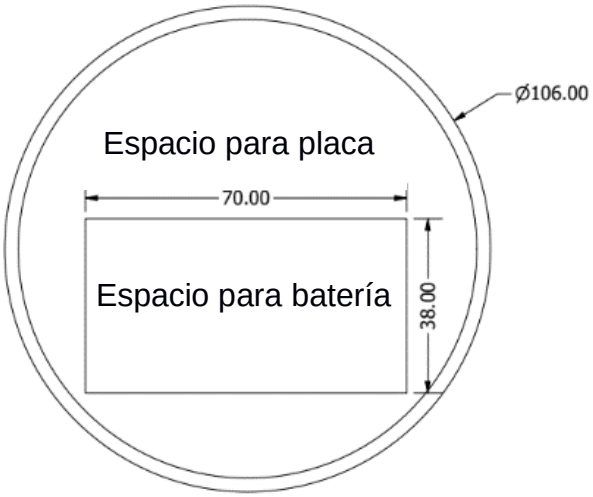


Figura 2.5 Vista superior de *Espacio para circuitería*

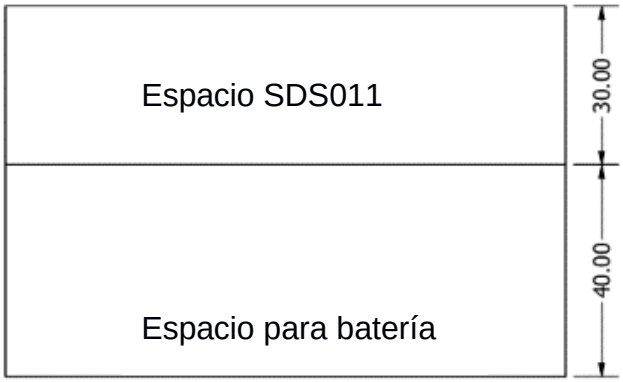


Figura 2.6 Vista frontal de *Espacio para circuitería*

2.5 Selección del método de Inteligencia artificial y diseño de *firmware*

2.5.1 Área de estudio

El área de estudio, la distribución de los elementos y las dimensiones del cuarto se pueden observar en la figura 2.7.

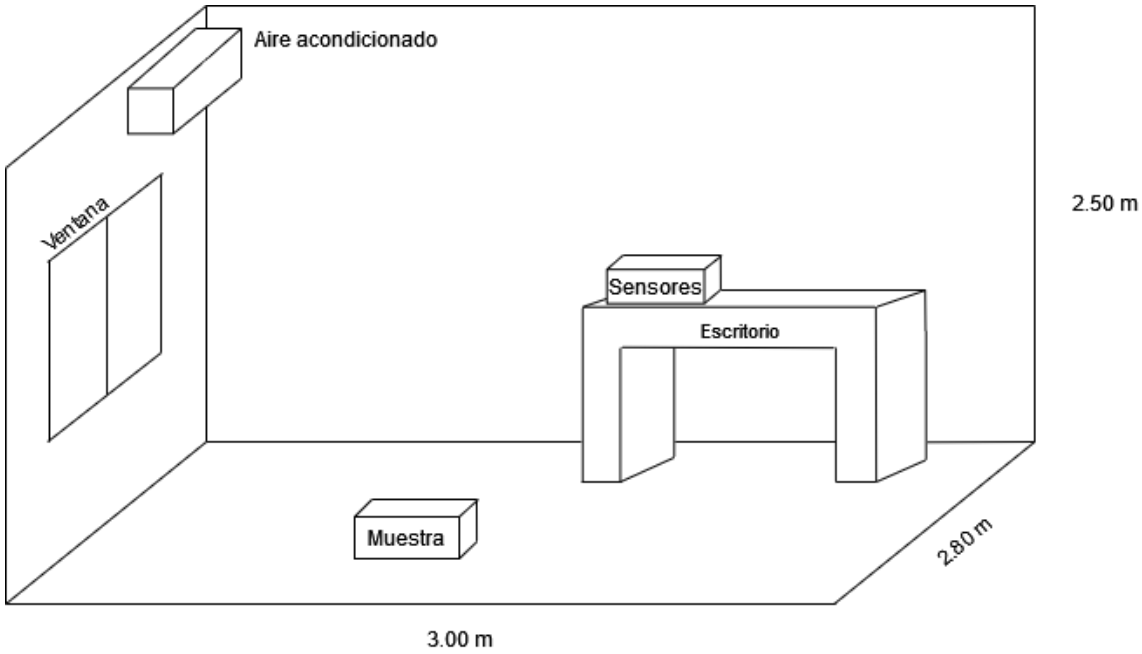


Figura 2.7 Disposición de los sensores y la muestra en el cuarto

Durante cada muestreo, el aire acondicionado estaba encendido, las ventanas cerradas y la muestra se encontraba a 1 m de distancia de los sensores. Cada tipo de muestra corresponde a un tipo de contaminación en específico como se muestra en la tabla 2.12.

Tabla 2.12 Tipo de contaminación y muestra

Tipo de contaminación	Muestra
Aire ambiente	Aire ambiente
Agente químico	Alcohol
Combustión	Cautín con pasta de soldar

2.5.2 Recolección de datos

Se definió 1 jornada como la recopilación continua de datos durante 8 horas, desde las 9:00 a.m. hasta las 5:00 p.m., con un tiempo de muestreo de 1 segundo. En

total, se realizaron 6 jornadas de muestreo, 2 por cada tipo de muestra. De esta manera se obtuvieron un total de 172800 lecturas, 57600 por cada tipo de contaminación.

En el caso del aire ambiente se muestreó el ambiente sin fuentes de contaminación cercanas. Para el muestreo del agente químico, se colocó en un recipiente 55 *ml* de alcohol antibacterial en el centro de la habitación. Finalmente, para la fuente de combustión, cada 30 minutos se calentaba el cautín y se colocaba su punta caliente en la pasta de soldar.

2.5.3 Algoritmo de inteligencia artificial

Se seleccionó el algoritmo de aprendizaje supervisado de regresión logística multiclase, *softmax*, para realizar el entrenamiento del modelo [29]. Se puede realizar una rápida implementación de este tipo de modelo con la librería de TensorFlow Keras.

En la figura 2.8 se observa que cada una de las señales de $\bar{X}$ es procesada por cada neurona Σ_i. Después, la función *softmax* calcula la probabilidad que el resultado de Σ_i pertenece a una respectiva clase y devuelve un arreglo de probabilidades de todas las clases, que suman 1. Finalmente, la predicción de la clase corresponderá al valor más alto del arreglo, $\bar{Y}$.

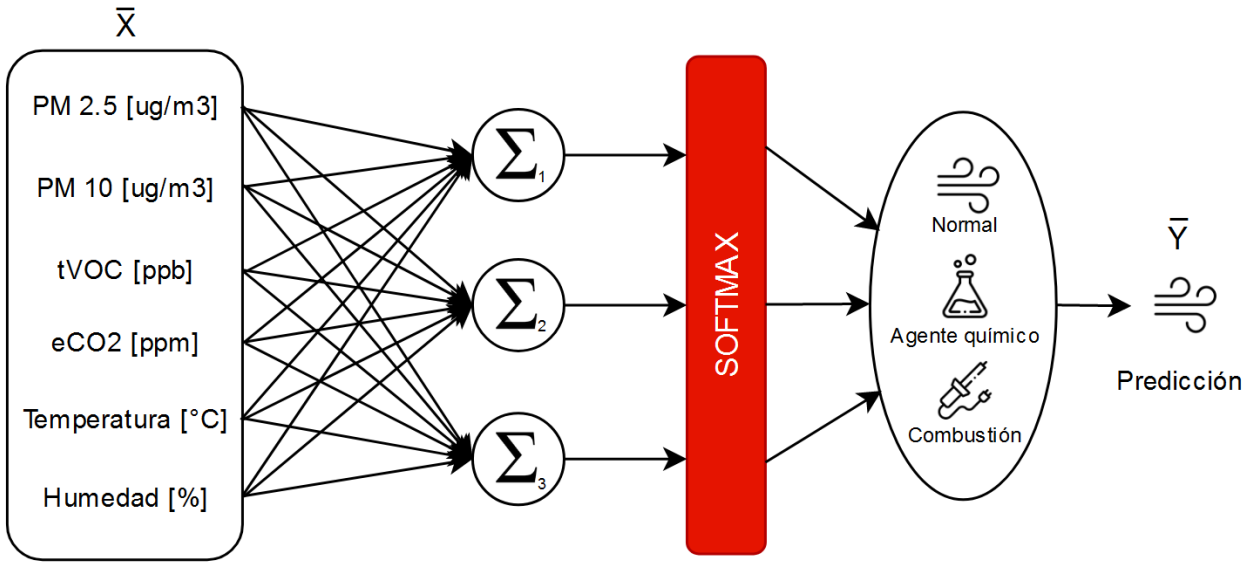


Figura 2.8 Diseño del modelo de inteligencia artificial para clasificación multiclase

2.5.4 Entrenamiento y conversión del modelo de inteligencia artificial

Antes del entrenar el modelo, los datos fueron preprocesados para eliminar valores repetidos. Después, son normalizados mediante la ecuación 2.1 por cada tipo de variable.

$$X_n = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{2.1}$$

Una vez preprocesados los datos, se configuró los hiper parámetros a utilizar para el entrenamiento: función de perdida, optimizador y razón de aprendizaje. Adicionalmente, se implementaron técnicas de *feature engineering*: *one-hot encoding*, lotes y épocas.

Posterior al entrenamiento, se convirtió el modelo TensorFlow a un modelo TensorFlow Lite. Después, es convertido a un modelo cuantizado a entero, cuyos datos de entrada y salida son del tipo *int8* en vez de *float32*, lo que mejora la velocidad de predicción a costo de precisión.

Para calcular la precisión del modelo clasificador multiclase se realizó una matriz de confusión, mostrado en la figura 2.9, por cada tipo de clase. Su precisión por clase se calculó mediante la ecuación 2.2 y se escogió el valor mínimo como la precisión general del modelo.

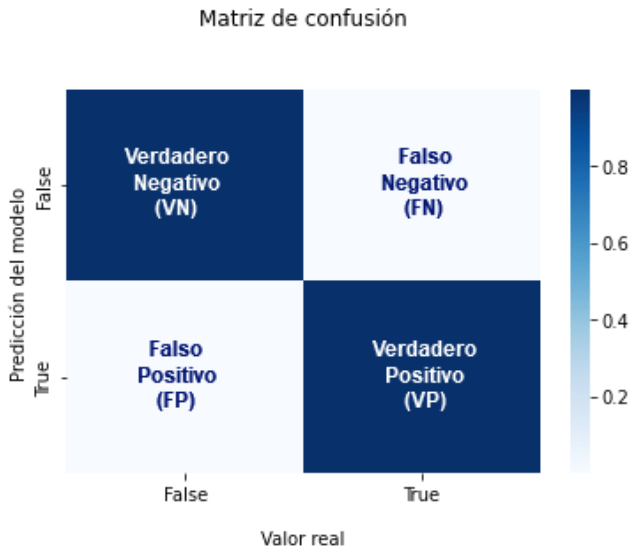


Figura 2.9 Plantilla de matriz de confusión

$$Precisión\ [\%] = \frac{VP + VN}{VP + FP + VN + FN} \cdot 100 \qquad (2.2)$$

Mediante la herramienta SLC del IDE de Silicon Labs se convirtió el modelo a un arreglo en C. El arreglo generado es utilizado para realizar predicciones utilizando las librerías de TensorFlow Lite para microcontroladores. Finalmente, se evaluó la precisión del modelo embebido en el kit al recopilar los datos predichos y compararlos con el valor real durante un periodo de 3 horas. En cada hora estuvo presente un tipo de contaminante distinto.

2.5.5 El kit de desarrollo e IDE

El kit de desarrollo *Thunderboard*™ utiliza el microcontrolador EFR32BG22 basado en la arquitectura ARM del Cortex-M33. El perfil de arquitectura del Cortex-M33 es M, *Microcontroller*, y abarca las siguientes ventajas propias de su familia: bajo consumo de energía, reducido tamaño y con alta eficiencia energética. Este tipo de perfil de arquitectura es altamente utilizado para tecnologías IoT, *Internet of Things,* dispositivos embebidos, entre otros [30].

Las características propias del kit otorgan de un almacenamiento de hasta 512 kB de memoria flash y 32 kB de memoria RAM. Además, posee los periféricos GPIO, I^2C,

UART, entre otros y una serie de sensores integrados tal como el Si7021 de temperatura y humedad relativa [31]. Incluso, es capaz de integrar aplicaciones con Machine Learning.

2.5.6 Sistema operativo para sistemas embebidos

RTOS es un sistema operativo dedicado a organizar las tareas y crea la ilusión que todas las tareas se ejecutan simultáneamente. Además, cada tarea se puede interpretar como una función *main* que se ejecuta secuencialmente [32].

Solo una sola tarea puede ser ejecutada a la vez, por lo tanto, para que las tareas se ejecuten "simultáneamente", una tarea se bloquea para darte tiempo al CPU de ejecutar otros hilos, como se observa en la figura 2.10. Es debido a estas ventajas del RTOS, que se lo ha utilizado como técnica de programación de tareas para el sistema embebido.

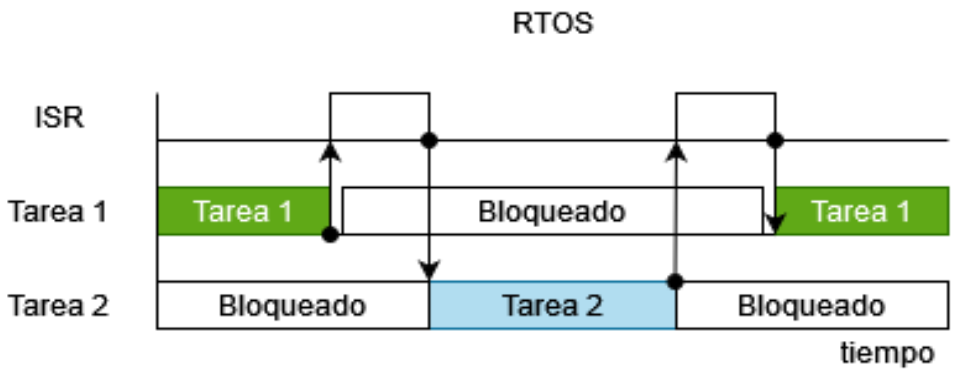


Figura 2.10 Línea de tiempo de una aplicación con implementación de un RTOS

2.5.7 Periféricos del microcontrolador y sensores

En la figura 2.11 se pueden observar las conexiones físicas realizadas entre el kit de desarrollo, los sensores SDS011, SGP30, y los LEDs. El kit integra las conexiones con el sensor Si7021 dentro del PCB.

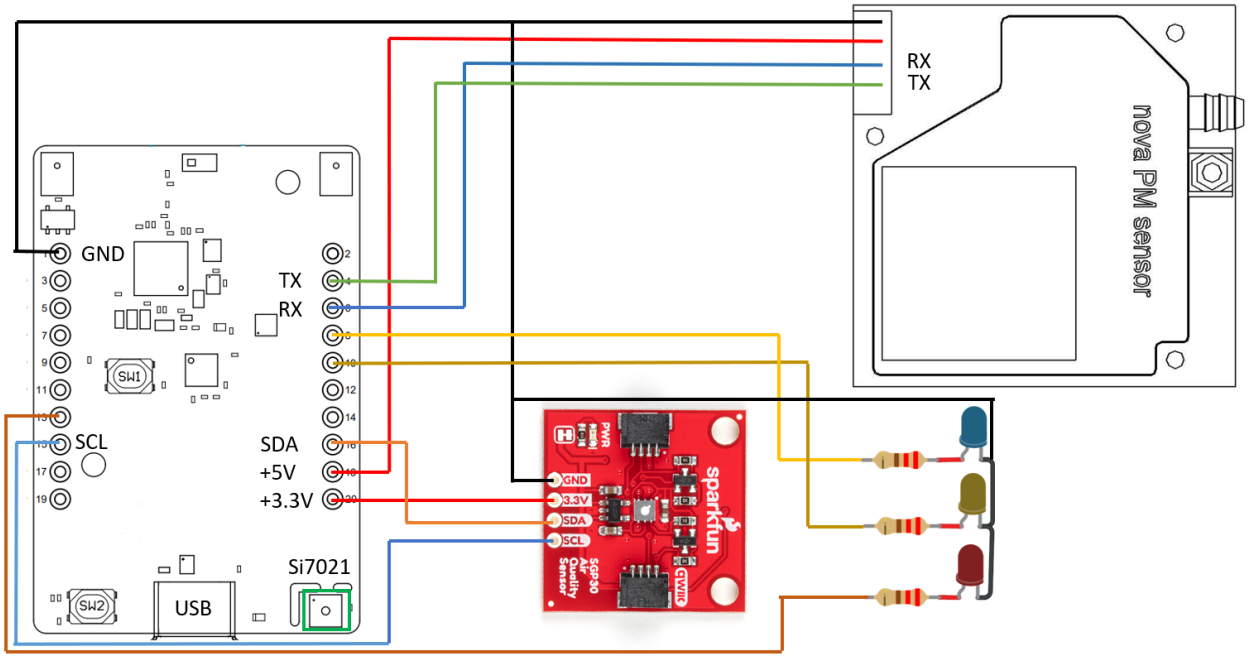


Figura 2.11 Diagrama esquemático de conexiones entre el kit de desarrollo y los sensores de calidad de aire [31], [33], [34]

Para iniciar la comunicación y recopilar datos con el sensor de PM se configuró correctamente el protocolo de comunicación UART, según el apéndice C, y se conectaron los pines de transmisión, Tx, y recepción Rx, de manera invertida con el kit de desarrollo.

Para la comunicación con los sensores SGP30 y Si7021 se configuró la frecuencia del reloj de SCL a 100 kHz debido a que es la máxima frecuencia que soporta el sensor Si7021, a diferencia de los 400 kHz del SGP30. Los pines de SCL y SDA del kit se conectaron correspondientes a las líneas SCL y SDA del sensor SGP30.

A través del maestro, el kit de desarrollo, se enviaron los comandos necesarios de escritura y lectura para leer los datos del fabricante, iniciar el sensor para realizar lecturas, leer y/o establecer los valores referenciales de lectura y, finalmente, obtener la concentración de contaminación en el ambiente, como se observa en la figura 2.12.

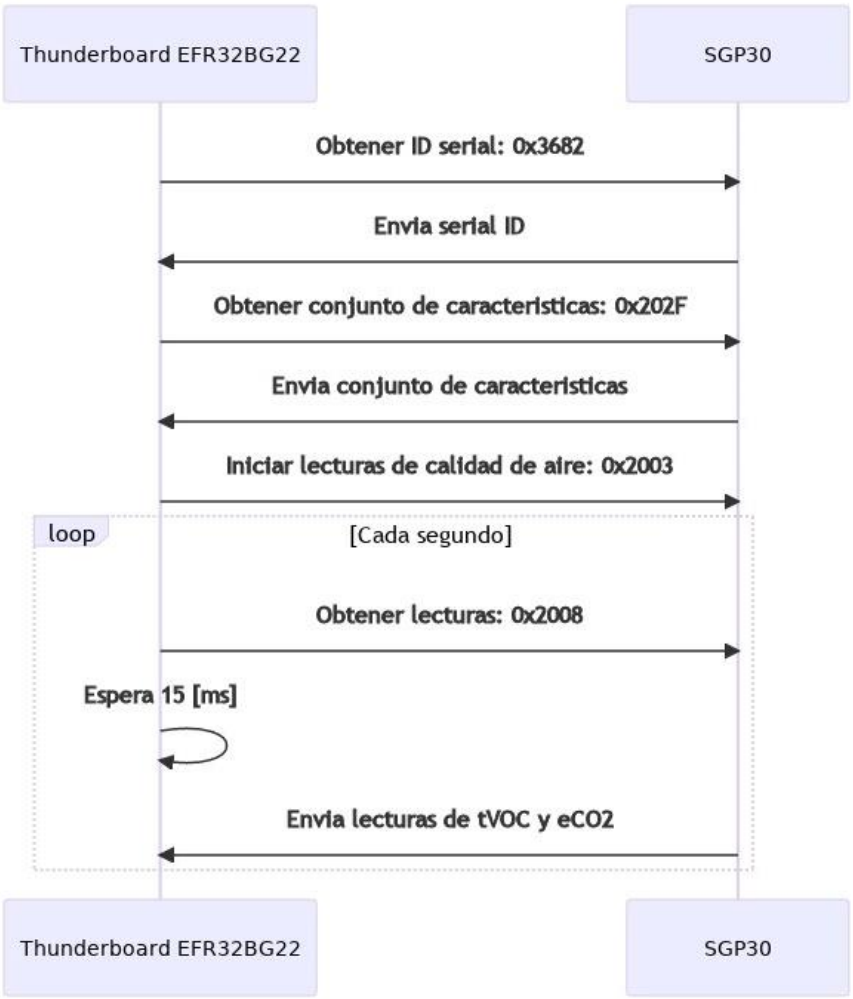


Figura 2.12 Diagrama de secuencia para la configuración del sensor SGP30

En la tabla 2.13 se detalla la duración del ciclo de encendido y apagado para cada LED que representa el tipo de contaminante identificado por el modelo de inteligencia artificial.

Tabla 2.13 Encendido del LED según el tipo de contaminación

No	Predicción	Encendido [ms]	Apagado [ms]	LED
1	Ambiente normal	500	500	Azul
2	Agente de limpieza			Amarillo
3	Combustión			Rojo

En la figura 2.13 se muestran las conexiones realizadas del kit de desarrollo con los distintos sensores y LEDs indicadores, Mismo utilizado para la recopilación de datos.

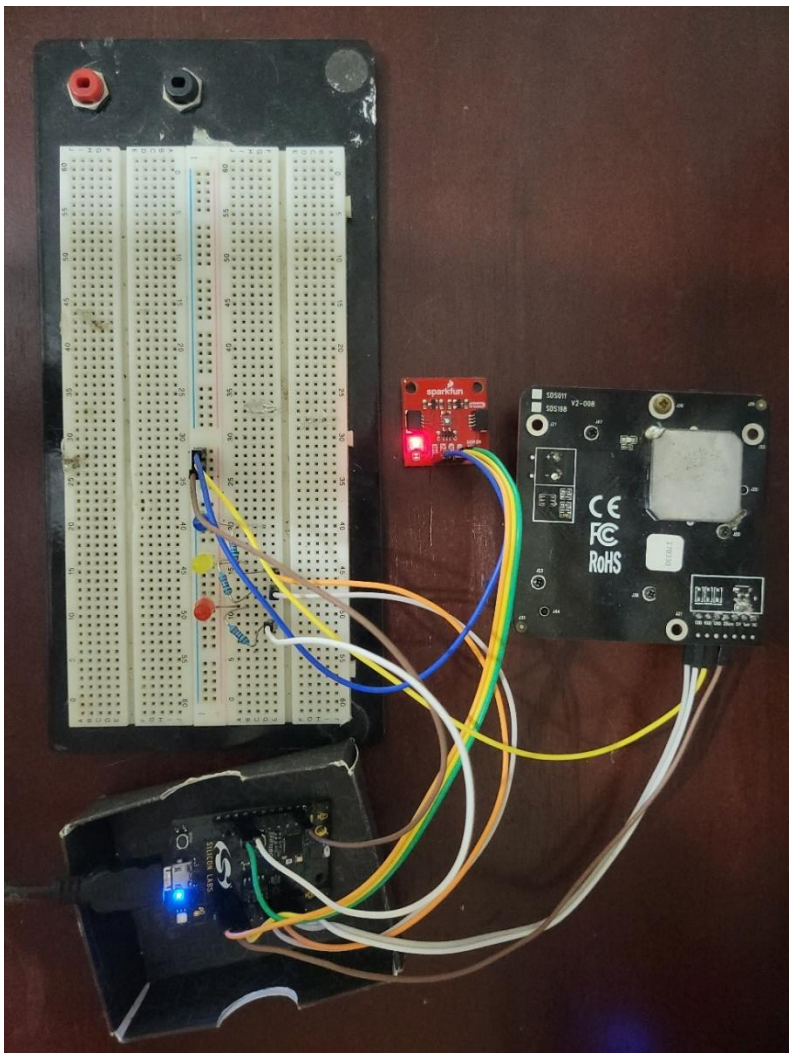


Figura 2.13 Circuitería realizada entre el kit, sensores y LEDs

2.5.8 Gráfico de dependencias

En la figura 2.14 se puede observar que el programa main.c ejecuta la aplicación principal, app.c, que accede al archivo de cabecera del sensor SDS011, SGP30 o de IA para inicializar su respectiva instancia e iniciar las lecturas y predicciones. Al estructurar de esta manera el proyecto, se logra eliminar redundancia entre archivos de cabecera en el código.

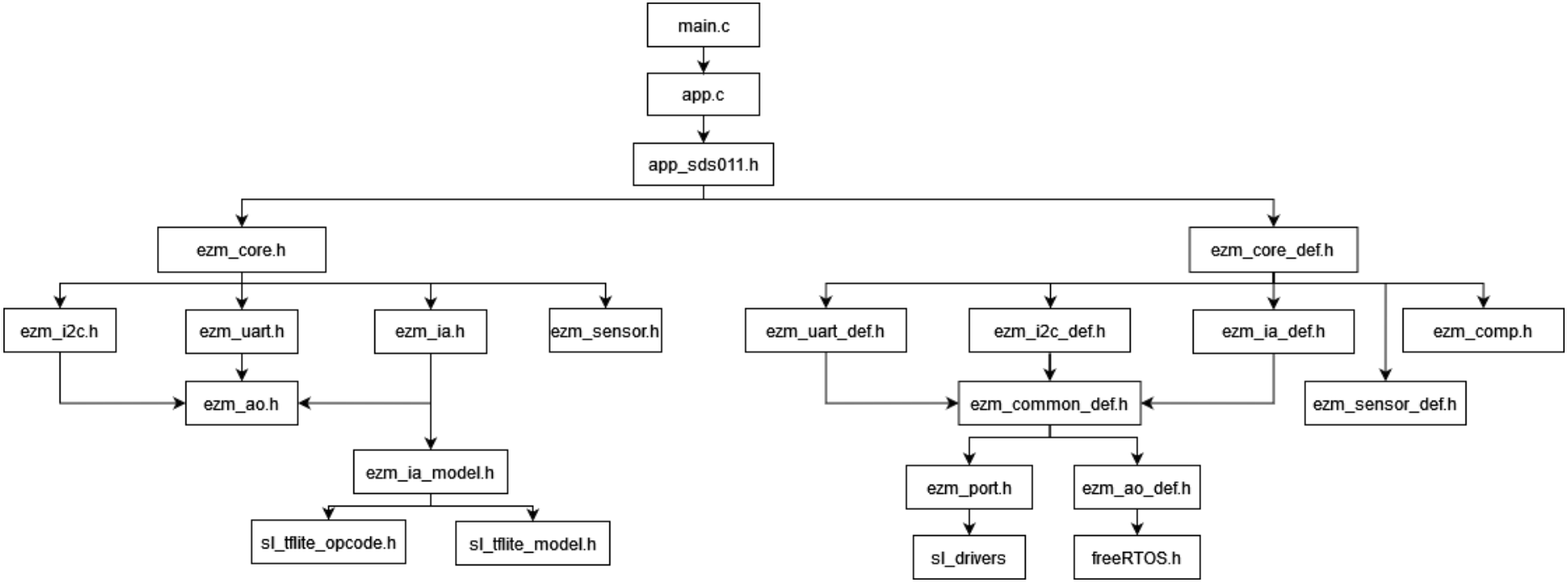


Figura 2.14 Gráfico de dependencias del proyecto

Del lado izquierdo está el núcleo del programa, ezm_core.h, archivo de cabecera que tiene acceso a todas las funciones públicas de los periféricos del sensor, IA, objetos activos y columna del sistema. Del lado derecho, se encuentran las definiciones de estructuras y enumeraciones de cada uno de los periféricos, objetos activos, columna del sistema, acceso a las librerías generales de Silicon Labs, GPIOs disponibles y al RTOS del sistema, freeRTOS.

2.5.9 Diseño de la estructura de la columna

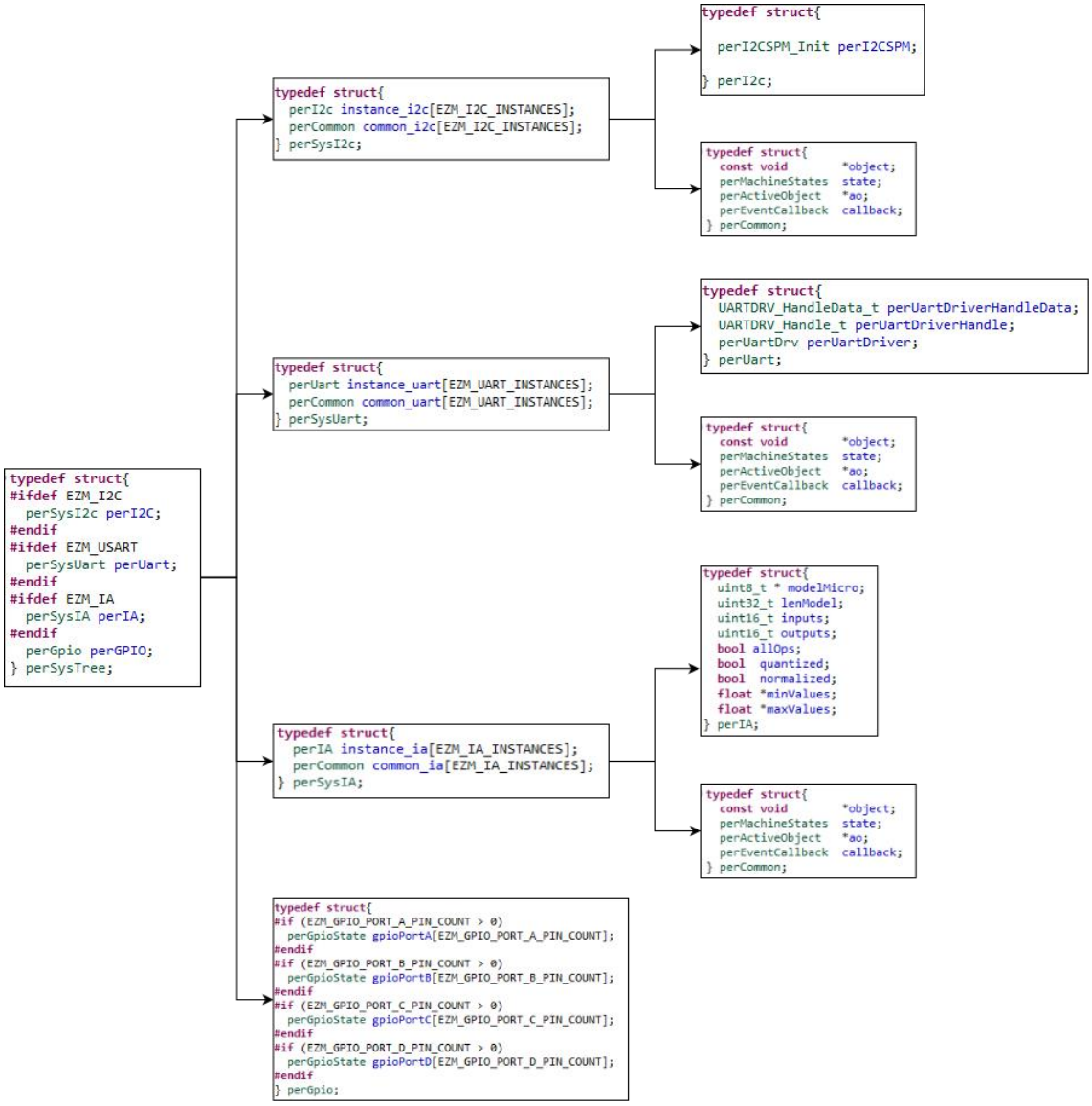


Figura 2.15 Estructura de la columna principal

En la figura 2.15 se observa la estructura de la columna principal, en ella se almacenan las instancias de I^2C, UART, IA y GPIO. Se observa que dentro de las

estructuras de los periféricos de comunicación hay igual cantidad de instancias al número de periféricos de comunicación y pines GPIO de acuerdo con los que soporta el kit.

Además, cada instancia de los periféricos de comunicación integra características comunes. Instancias del periférico creado, estado actual de la máquina de estado, como se observa en la figura 2.16, el objeto activo para la comunicación entre tareas y una función *callback* en donde los eventos de escritura y lectura serán enviados.

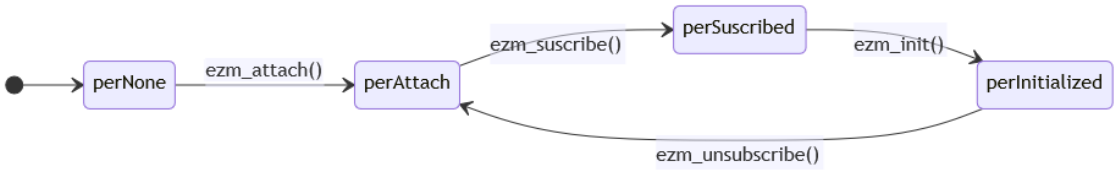


Figura 2.16 Máquina de estados de los objetos en la columna.

Por último, la estructura principal de cada periférico está designada como perI2c y perUart, respectivamente, almacena la configuración principal para la comunicación y los pines utilizados para la conexión física. Además, el estado de los pines es actualizado de disponible a ocupado en la instancia perGpio.

2.5.10 Diseño de los objetos activos

En la figura 2.17 se muestran los pasos para crear un objeto activo, el cual consiste en crear una cola, en donde los eventos serán receptados, una hilo, tarea o subproceso de freeRTOS, sistema operativo seleccionado. Después, se añade una función de retorno en donde serán enviados todos los eventos a ser procesados y finalmente se lo asigna a un periférico de comunicación.

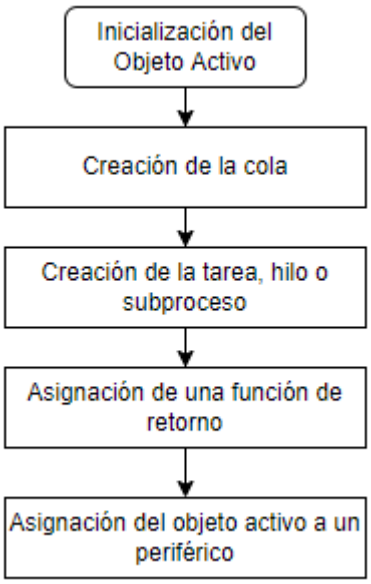


Figura 2.17 Diagrama de flujo para la creación del objeto activo

En la figura 2.18 se muestra la detonación de un evento o mensaje en la función de recepción de datos de los periféricos de comunicación con los sensores. Se consiguió aplicar el patrón de diseño de objetos activas al enviar el evento hacia la cola del objeto activo, enviado hacia la función de retorno y procesar el evento en la función. Además, no se procesará ningún otro evento hasta que la función de retorno termine su ejecución.

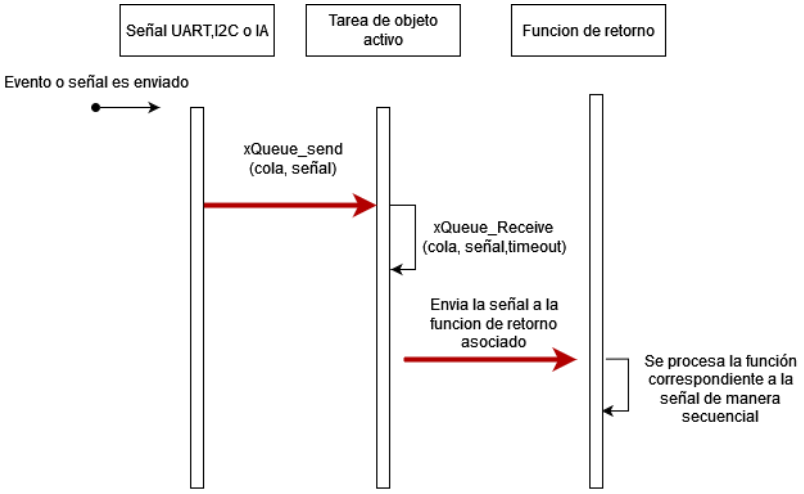


Figura 2.18 Diagrama de interacción entre la recepción de datos, la tarea del objeto activo y la función de retorno

2.5.11 Documentación

El software Doxygen se utilizó para generar un archivo HTML que facilite la lectura de la documentación que incluye una breve descripción del funcionamiento de las funciones públicas y de los parámetros a ingresar.

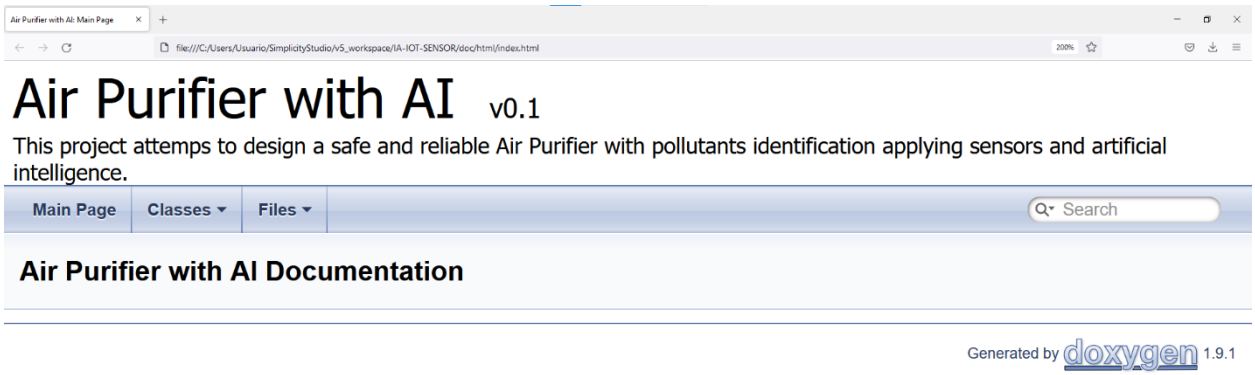


Figura 2.19 Ejemplo del archivo HTML generado por Doxygen de la documentación del proyecto

CAPÍTULO 3

3. RESULTADO Y ANÁLISIS

Se presentará en este capítulo el diseño mecánico en 3D de la estructura del purificador de aire portable junto a sus partes. Para el entrenamiento del modelo de IA se mostrarán los hiper parámetros usados y el calculó del porcentaje de predicciones correctas. Respecto al diseño de *firmware* se detallará la interacción de cada uno de los componentes creados con la aplicación.

3.1 Diseño mecánico

El diseño 3D se muestra en la figura 3.1 y 3.2. Fue realizado en el software Inventor, el cual, además, nos permitió realizar una animación del ensamble para el cliente. El concepto general del diseño se divide en 4 partes. Todos los planos de las secciones serán mostrados en el apéndice A.

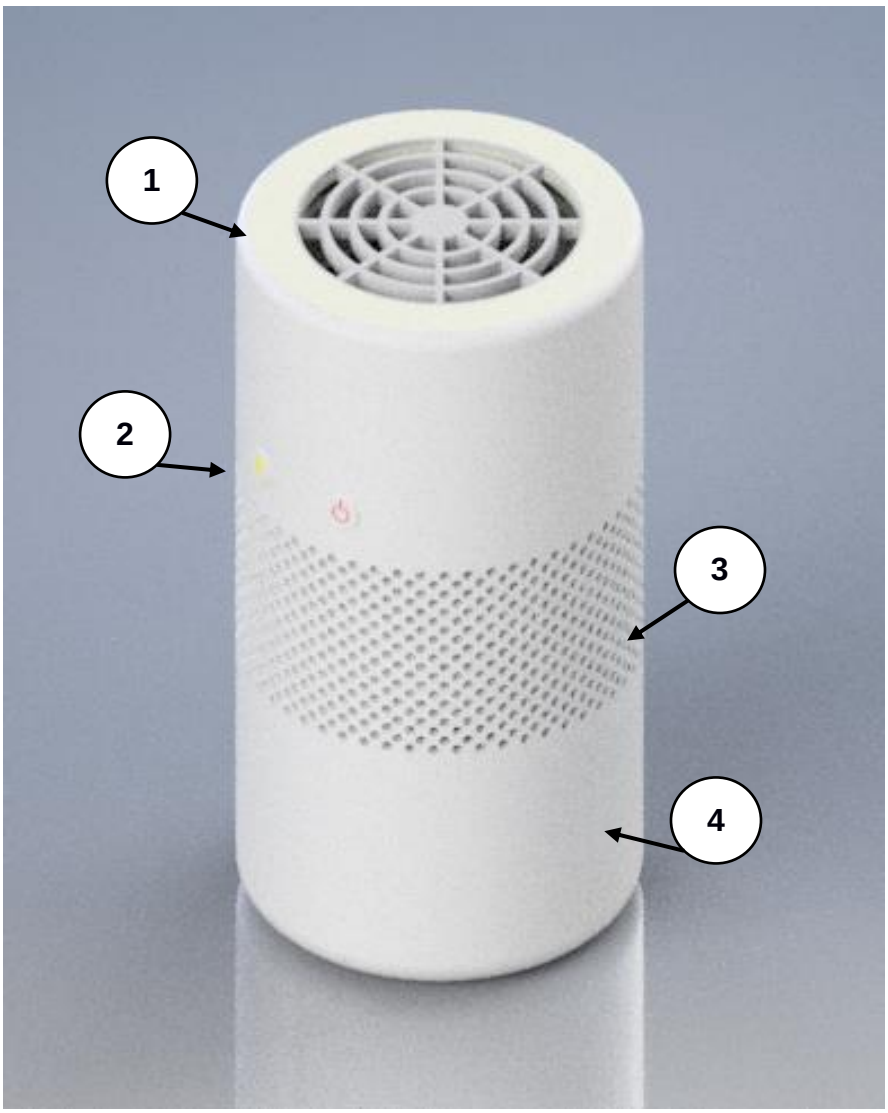


Figura 3.1 Ensamble 3D de la estructura mecánica (1) Salida de aire purificado. (2) Interfaz manual de usuario. (3) Entrada de aire sin tratar. (4) Base removible.

Figura 3.2 Vista explosionada de todos los componentes incluidos en el diseño mecánico

3.1.1 Salida de aire purificado

La salida del purificador posee un soporte para los leds WS2812, una elección como indicador visual del tipo de contaminante que detecte el modelo de inteligencia artificial entrenado. Se escoge estos leds debido a que pueden ser controlado por pulsos a través del periférico RMT, logrando varias secuencias de colores. La iluminación brindada por los leds será visible debido al anillo de material transparente. Nótese que entre los leds y el anillo translucido debe existir una distancia para lograr un efecto visual agradable.

Es importante mencionar que, entre la sección de salida de aire purificado y entrada de aire sin tratar, se acopló el ventilador de CD en las protuberancias con agujero en la parte inferior. Los detalles mencionados se observan en la figura 3.3 y 3.4.

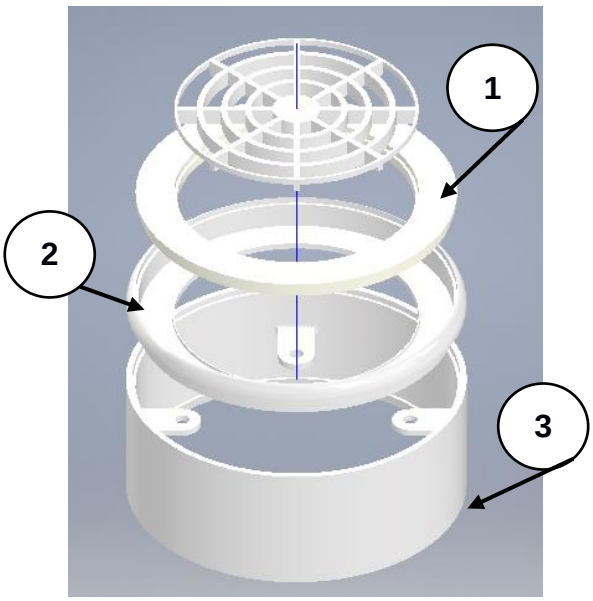


Figura 3.3 Vista explosionada de la salida de aire purificado (1) Anillo de resina. (2) Soporte para leds. (3) Junta entre la salida y entrada de aire.

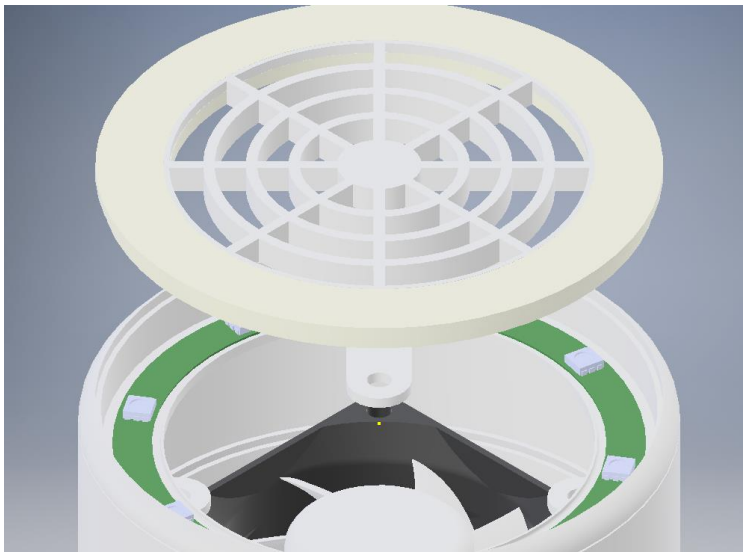


Figura 3.4 Vista explosionada con leds WS2812 incorporados dentro de su soporte

3.1.2 Entrada de aire sin tratar

En la figura 3.5 se enumeran los componentes que conforman esta sección de la estructura mecánica. Se nota la existencia de un soporte para el filtro HEPA seleccionado. Se destaca la existencia de un acople entre estos conjuntos de piezas y la base removible. Estas se unen por medio de una sección enroscada entre ambos diseños. Además, se puede notar en la vista explosionada, los botones de interacción con el usuario que serán mencionados más adelante.

En la figura 3.6 se hace un acercamiento a la parte superior del diseño. Este va acoplado con la salida de aire purificado por medio de una junta mecánica deslizante, en dicha unión, se acopla el actuador necesario para el desplazamiento de aire purificado.

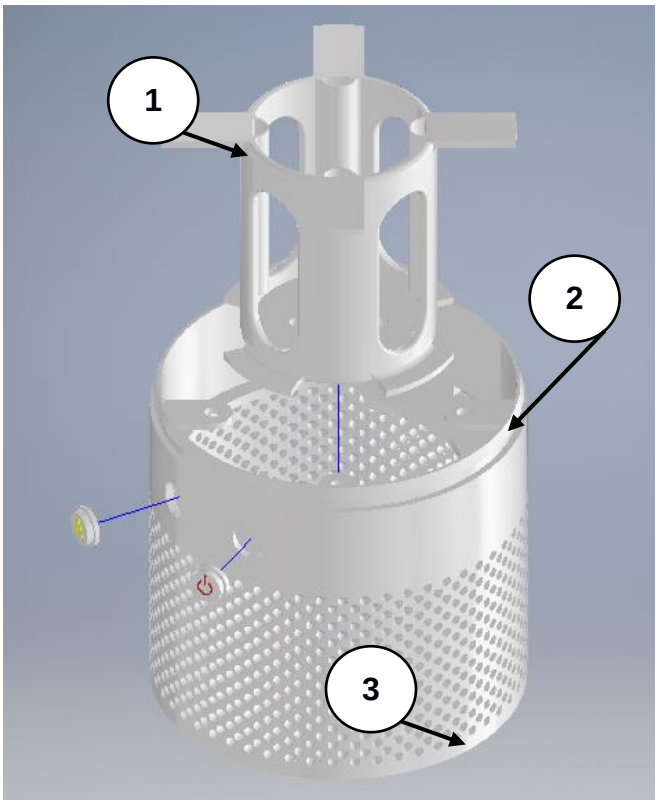


Figura 3.5 Vista explosionada de la entrada de aire sin tratar (1) Soporte del filtro HEPA. (2) Acople entre secciones de entrada y salida de aire. (3) Acople entre la entrada de aire y base removible.

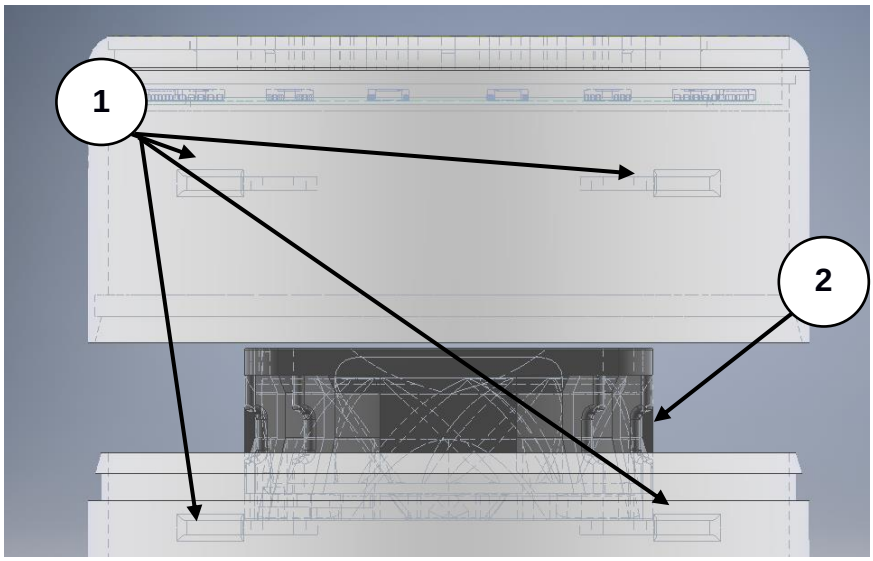


Figura 3.6 Acercamiento del acople mecánico entre la salida y entrada de aire con ventilador incorporado (1) Soportes para ventilador. (2) Ventilador.

3.1.3 Interfaz manual de usuario

El usuario será capaz de interactuar con el purificador por medio de dos botones, encendido y modo automático. El encendido es un inicio del purificador general a la máxima velocidad posible por el actuador. El modo automático hará uso de los sensores y modulará el volumen de aire desplazado dependiendo de la existencia y concentración de un contaminante.

Por otro lado, se agregó una ranura para colocar el puerto de carga USB hembra tipo C, modelo UJ31-CH-3-SMT-TR, para transmitir voltaje al sistema a través de un cargador USB. Se escogió este dispositivo debido a que posee varios pines de salidas con diferentes voltajes: 12 VCD, 5 VCD, etc. Ambas características mencionadas se observan en la figura 3.7.

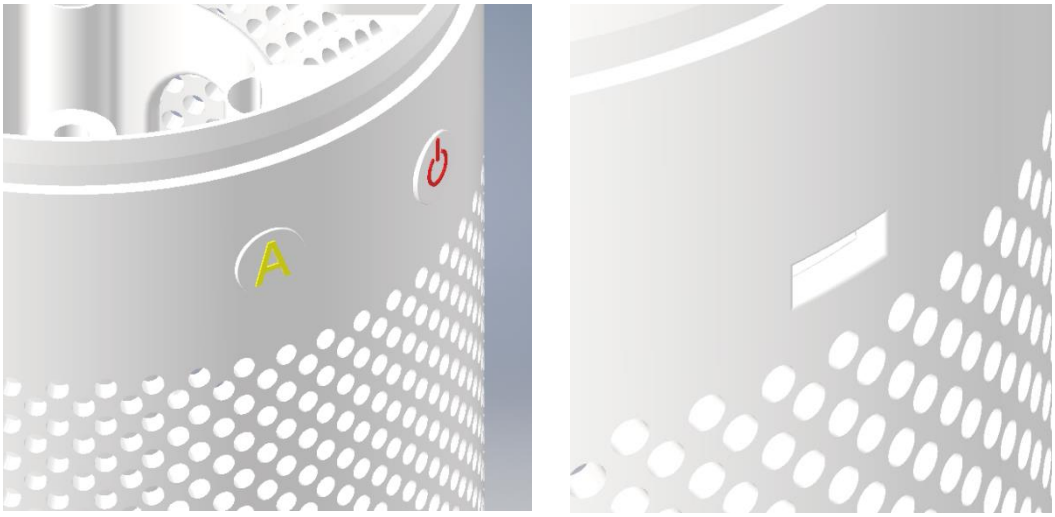


Figura 3.7 Acercamiento interfaz manual de usuario Botones disponibles al usuario (izq.) y puerto de carga (der.) en la sección de entrada de aire sin tratar.

3.1.4 Base removible

En las figuras 3.8 se observa la apariencia de la base removible del diseño. Existen soportes para el sensor SDS011, un espacio dedicado para la batería de litio recargable y, además, espacio para juntas roscadas entre la placa que contendrá el microprocesador y los sensores de calidad de aire, tal como se ve en la figura 3.9. Las constantes muestras de aire entrarán por las pequeñas ranuras en la base.

La tapa de la base posee una sección roscada para acoplarse con la sección de entrada de aire. Esto fue diseñado con el propósito de que el usuario final pueda acceder al soporte de filtro HEPA y remplazarlo una vez llegue al final de su vida útil, tal como se muestra en la figura 3.10.

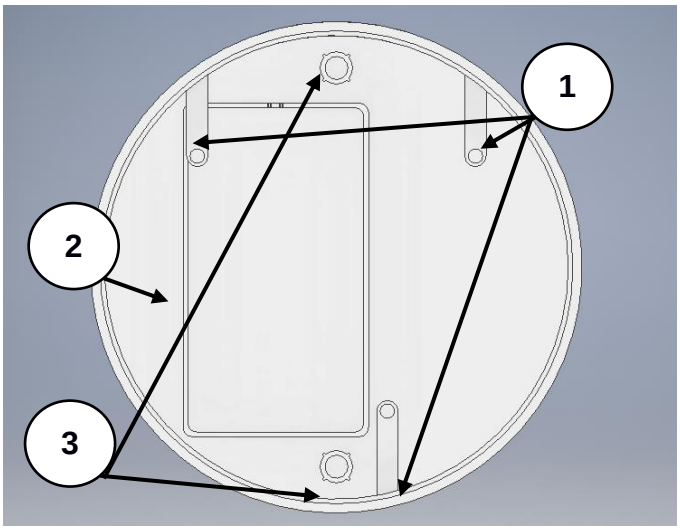


Figura 3.8 Vista superior de la base removible (1) Soporte para sensor SDS011. (2) Espacio para batería de litio. (3) Soporte para placa principal.

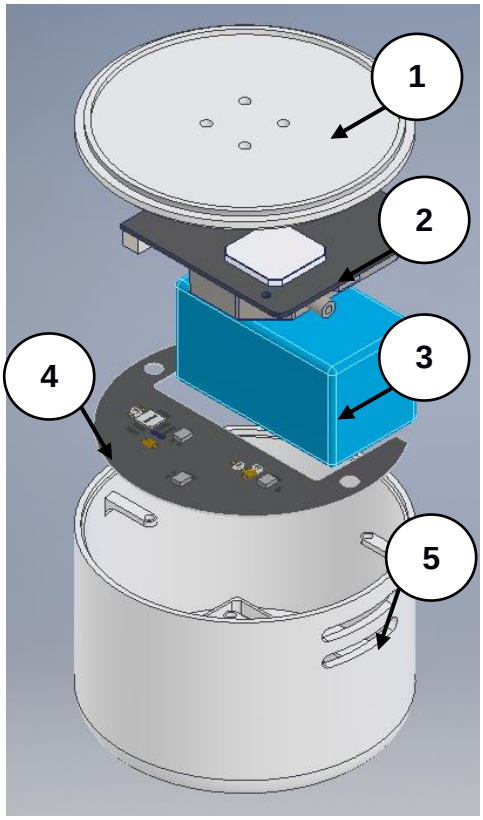


Figura 3.9 Vista explosionada de la base removible junto a sus componentes internos (1) Tapa de la base removible con sección enroscada. (2) Sensor SDS011. (3) Batería de litio. (4) Placa principal. (5) Ranuras para entrada de muestras de aire.

Figura 3.10 Demostración cambio de filtro HEPA una vez removida la base

3.2 Proceso de manufactura para diseño mecánico

El proceso de manufactura y selección del material se basa en la indagación de fabricación de productos embebidos similares y la consulta a diferentes proveedores de servicios de manufactura. Se toma en cuenta que el proyecto se encuentra en una etapa inicial de prototipado y no se espera la calidad de producción final de un producto.

3.2.1 Proceso

El proceso cotizado para la manufactura del producto fue por impresión. La impresión tiene un valor por pieza, y, al terminar el producto final impreso tendrá un costo individual de fabricación mayor. Sin embargo, es mucho más fácil recurrir a este tipo de manufactura debido a que la complejidad y detalle de piezas es menos limitante [35]. A pesar de que el costo por producto final impreso es mayor, es más fácil recuperar la inversión inicial de estos por *n* número de productos.

Los métodos utilizados de impresión son por modelo de disposición fundida o FDM por sus siglas en inglés y estereolitografía o SLA. La decisión es fuertemente influenciada

por la etapa en la que se encuentra el proyecto. La selección de manufactura por impresión nos brinda piezas funcionales en tiempos de fabricación relativamente cortos para poder llevar a cabo pruebas a futuro.

3.2.2 Material

La propuesta inicial de posibles materiales para la estructura mecánica del proyecto fue plástico ABS para la mayoría de las partes diseñadas. Este material es fuerte y versátil con una buena resistencia a impactos y químicos. Es barato y recomendado para partes funcionales y prototipados. Por otra parte, para el anillo transparente que cubrirá los leds es de resina transparente. Este es un material resistente e ideal para acabados parecidos a cristales.

Figura 3.11 Resina transparente [36]

Figura 3.12 Plástico ABS [36]

3.3 Inteligencia artificial para clasificación multiclase

3.3.1 Datos recolectados

La red de sensores de calidad de aire recopiló datos de los contaminantes del aire y condiciones ambientales: $PM_{2.5}$, PM_{10}, tVOC, eCO_2, temperatura y humedad, obteniendo la totalidad de datos previstos en la sección 2.5.2. En las siguientes figuras, se muestran las señales obtenidas durante los 6 días de muestreo.

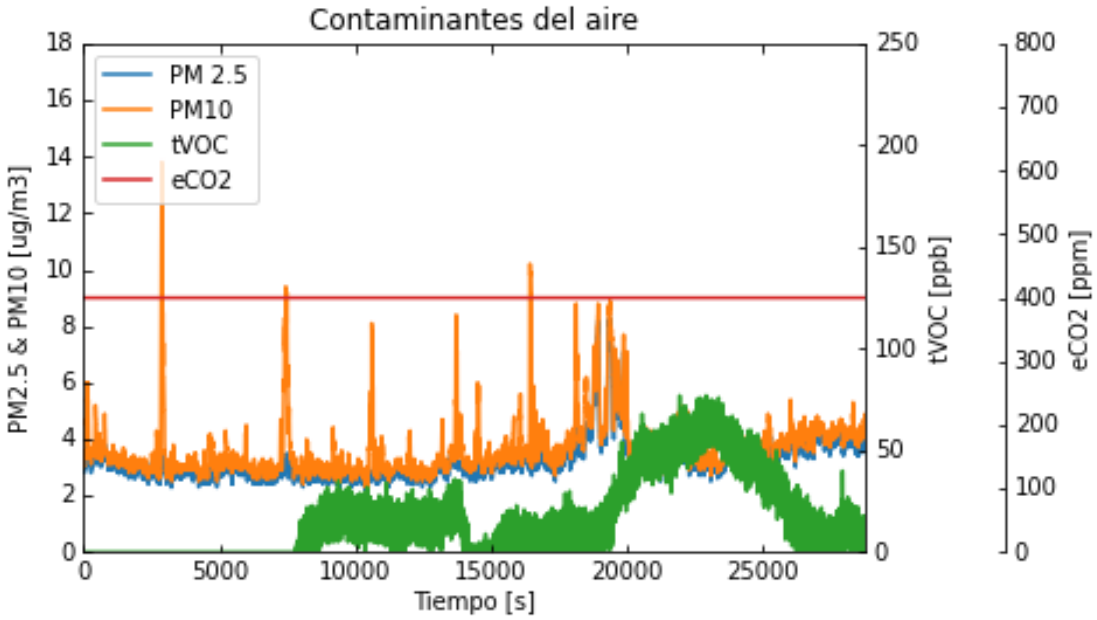


Figura 3.13 Día 1: Muestra del aire ambiente

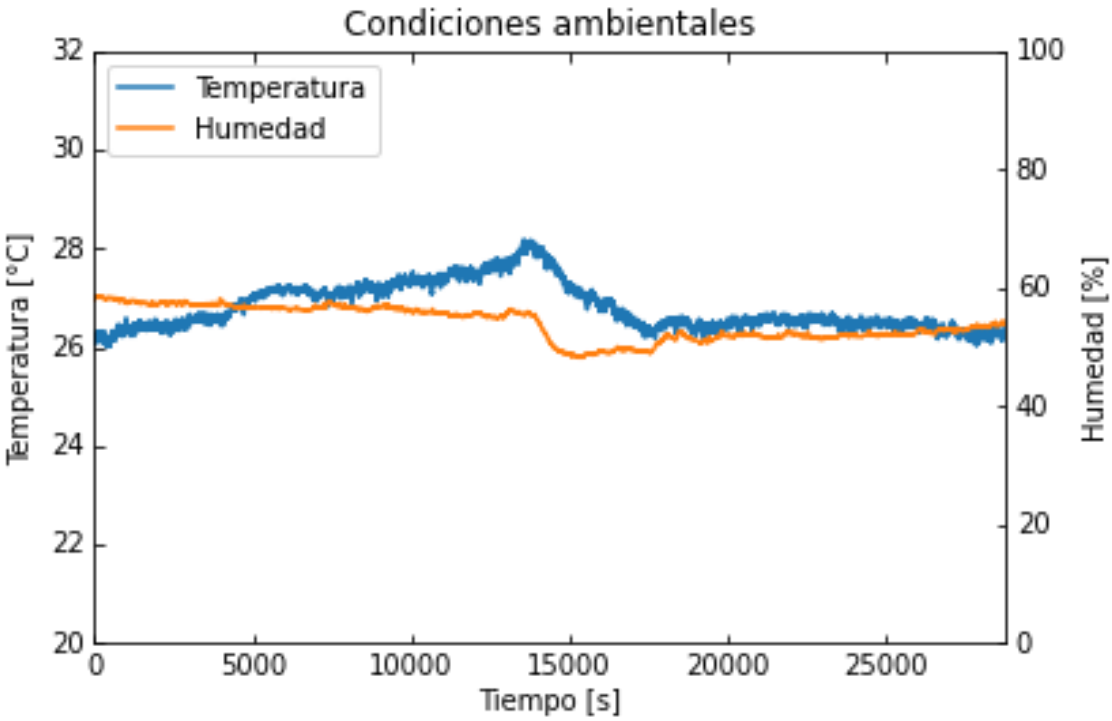


Figura 3.14 Día 1: Condiciones ambientales

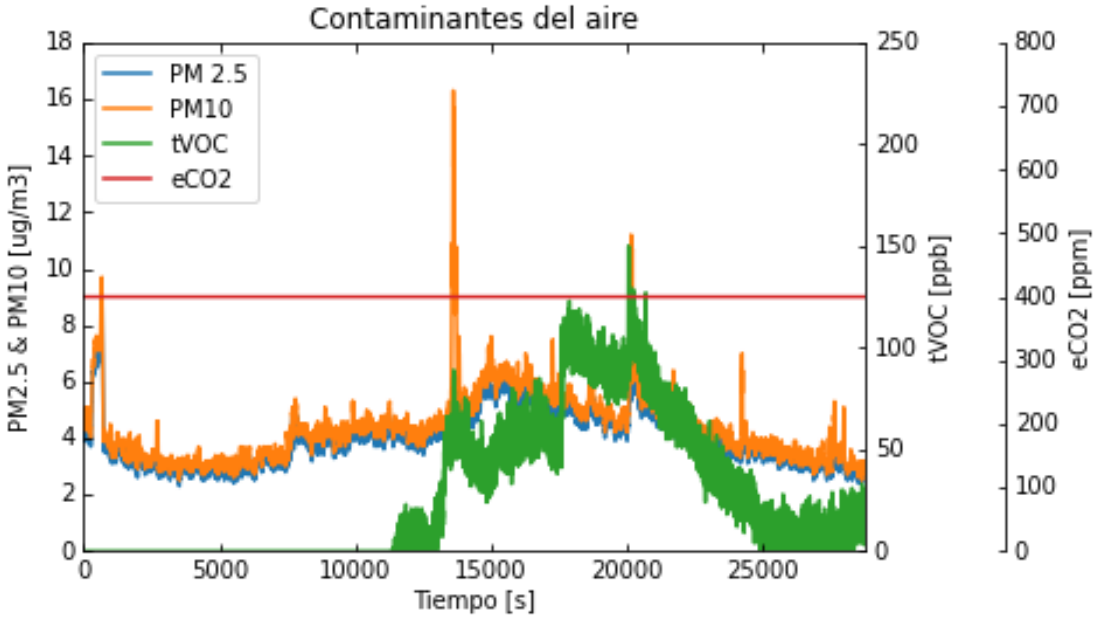


Figura 3.15 Día 2: Muestra del aire ambiente

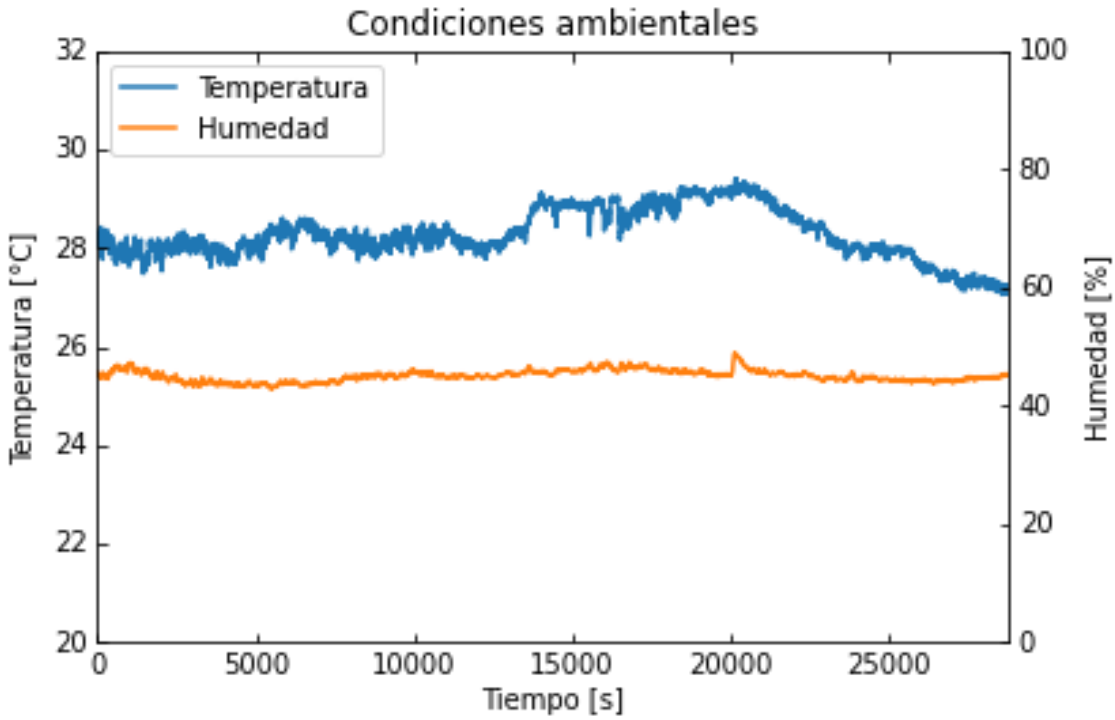


Figura 3.16 Día 2: Condiciones ambientales

En la tabla 3.1 se observan bajos niveles de $PM_{2.5}$, PM_{10}, permisibles según los lineamientos de la OMS, tVOC y eCO_2. Además, la temperatura y humedad se mantienen en niveles constantes entre 26.81 - 28.29 °C, y 45.06 - 54.02 % respectivamente.

Tabla 3.1 Datos estadísticos de los días 1 y 2

Muestra	Aire ambiente			
Día	1		2	
Variable	**Promedio**	**Desviación estándar**	**Promedio**	**Desviación estándar**
$PM_{2.5}$ [ug/m3]	3.19	0.72	3.98	0.96
PM_{10} [ug/m3]	3.65	1.05	4.38	1.24
tVOC [ppb]	14.73	18.26	26.52	33.32
eCO_2 [ppb]	400.00	0.00	400.00	0.00
Temperatura [°C]	26.81	0.47	28.29	0.51
Humedad [%]	54.02	2.81	45.06	0.94

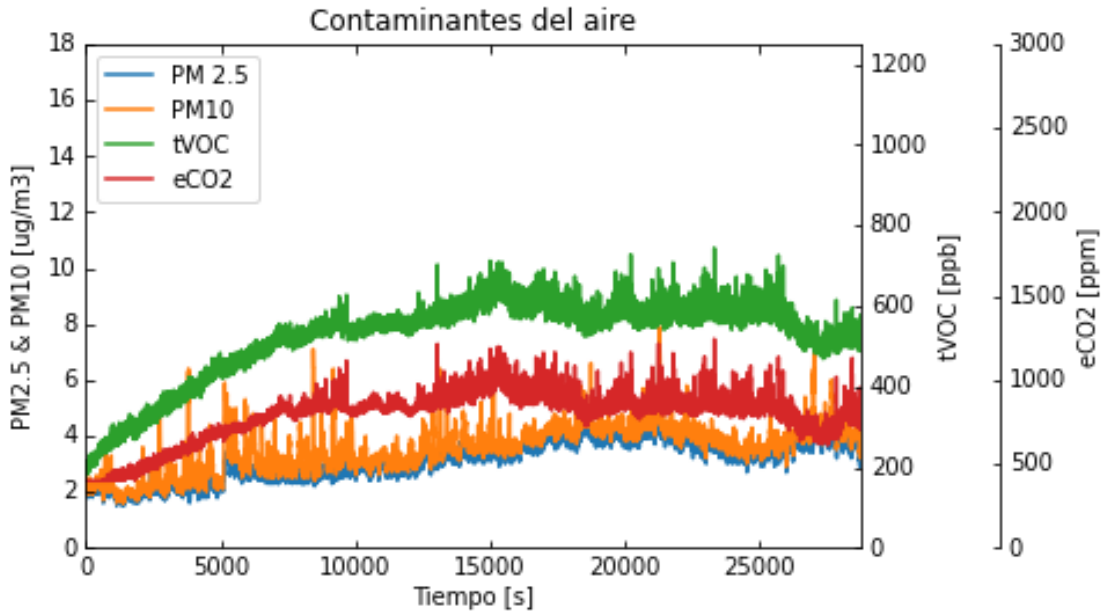


Figura 3.17 Día 3: Muestra de agente químico

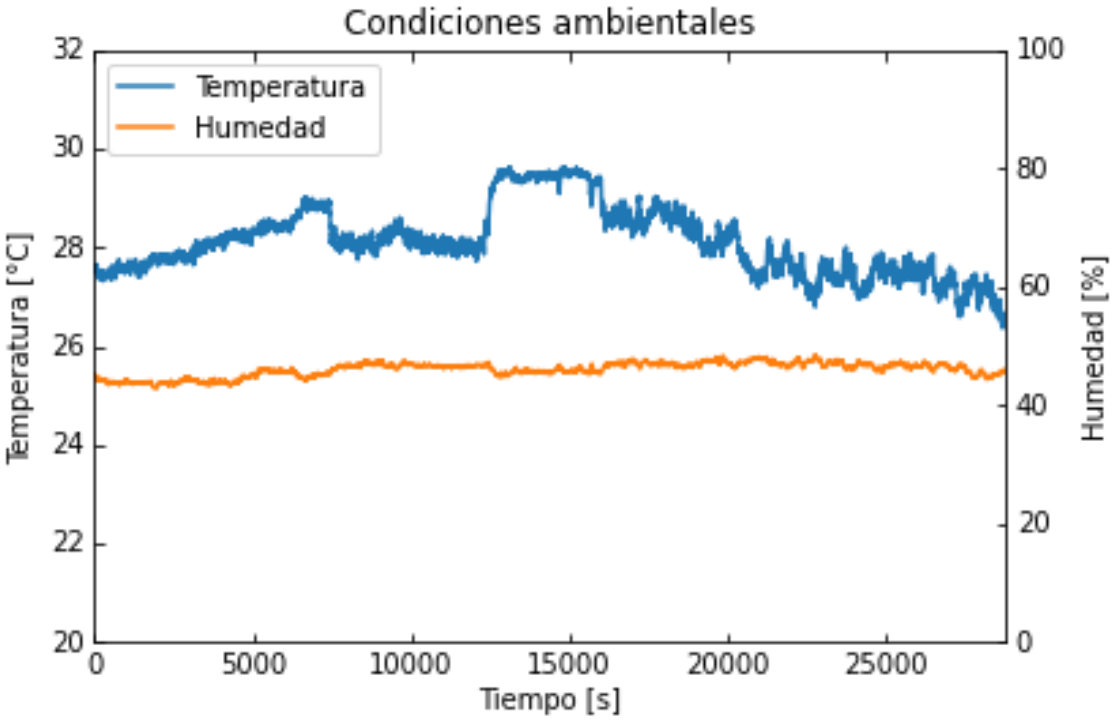


Figura 3.18 Día 3: Condiciones ambientales

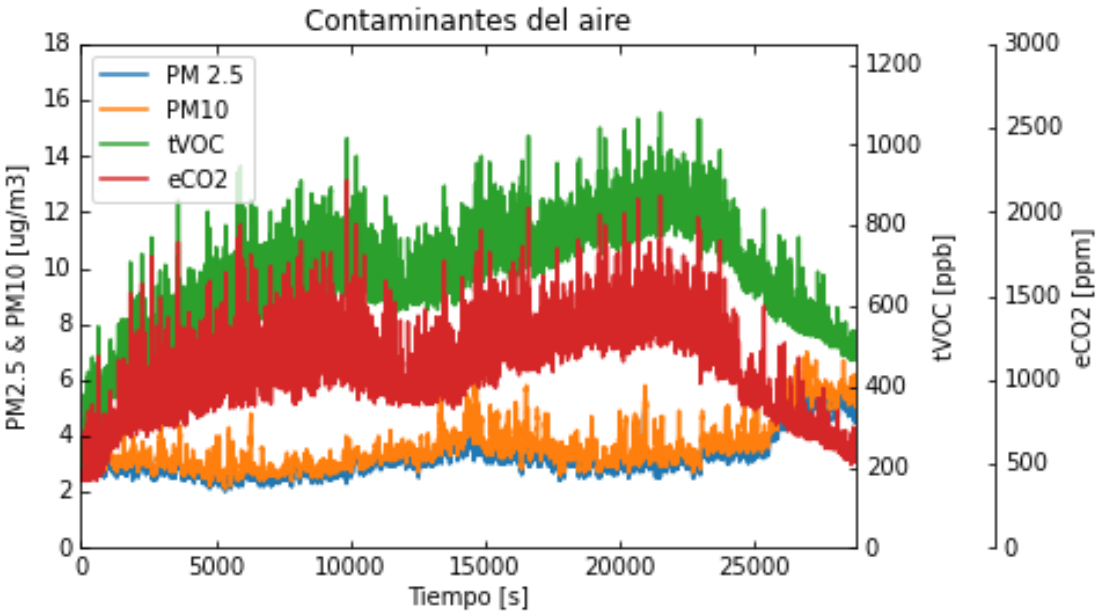


Figura 3.19 Día 4: Muestra de agente químico

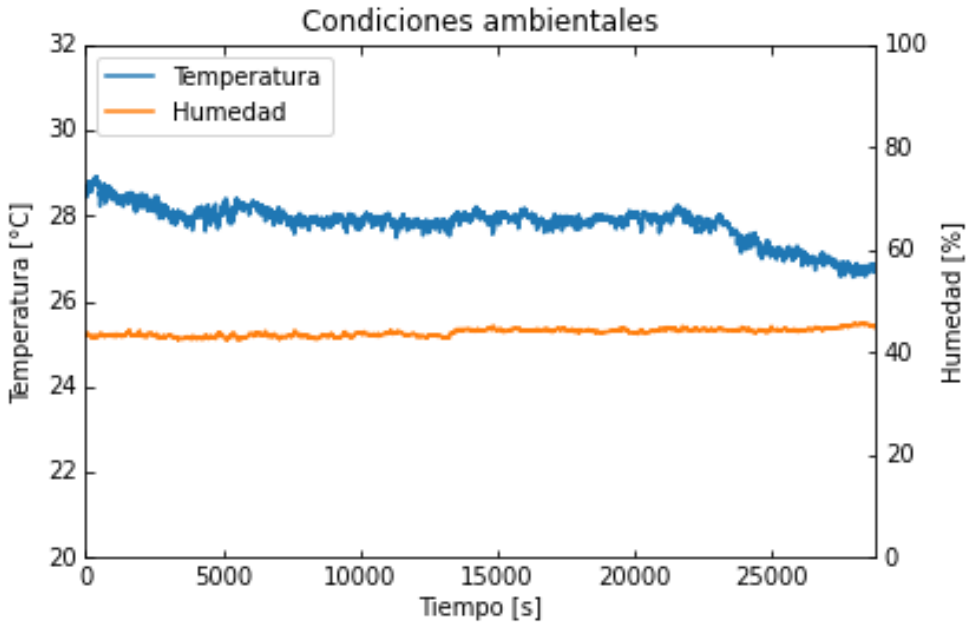


Figura 3.20 Día 4: Condiciones ambientales

En la tabla 3.2 se observan bajos niveles de $PM_{2.5}$ y PM_{10}, permisibles según los lineamientos de la OMS. En cambio, los niveles de concentración de tVOC y eCO_2 aumentaron por la presencia del alcohol a valores promedios de 520.57 - 671.62 ppb y 785.70 - 1068.30 ppm respectivamente. Además, la temperatura y humedad se

mantienen en niveles constantes entre 27.83 - 28.11 °C, y 43.90 - 46.07 % respectivamente.

Este alto nivel de tVOC puede crear irritación y malestar a aquellas personas expuestas [37]. Adicionalmente, los niveles de eCO_2 encontrados, 400 – 1000 ppm, son típicos de lugares con un buen intercambio de aire. Sin embargo, se encuentra en el umbral de la siguiente categoría, 1000 – 2000 ppm, en donde existen efectos de somnolencia y aire deficiente [38].

Tabla 3.2 Datos estadísticos de los días 3 y 4

Muestra	Alcohol antibacterial			
Día	3		4	
Variable	**Promedio**	**Desviación estándar**	**Promedio**	**Desviación estándar**
$PM_{2.5}$ [ug/m3]	3.12	0.74	3.19	0.69
PM_{10} [ug/m3]	3.59	0.86	3.61	0.85
tVOC [ppb]	520.57	100.65	671.62	132.53
eCO_2 [ppb]	785.70	150.82	1,068.30	265.68
Temperatura [°C]	28.11	0.69	27.83	0.43
Humedad [%]	46.07	1.20	43.90	0.68

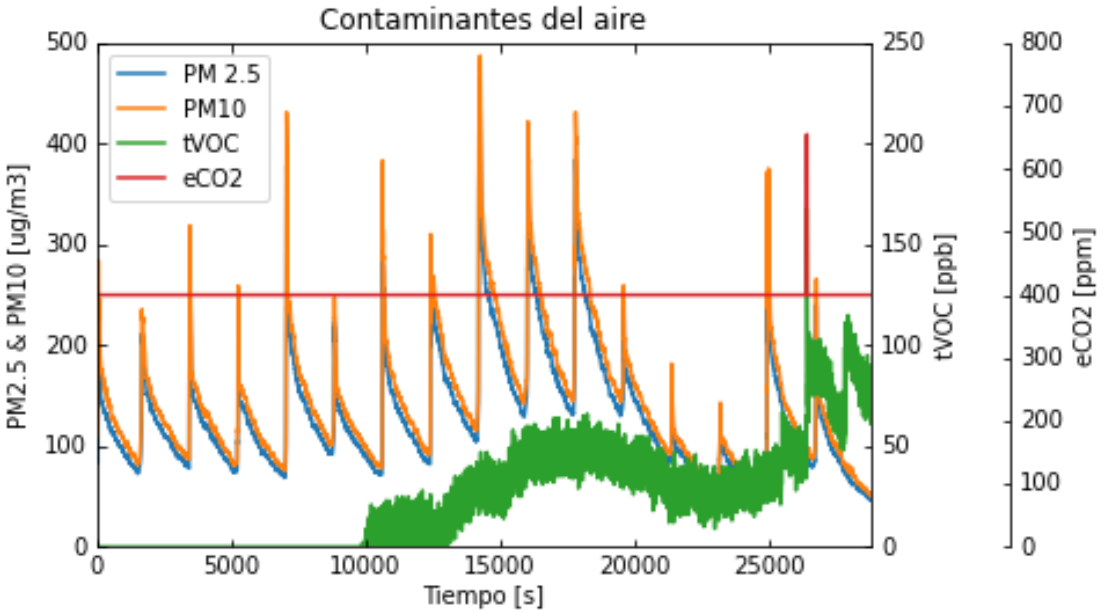


Figura 3.21 Día 5: Muestra de combustión

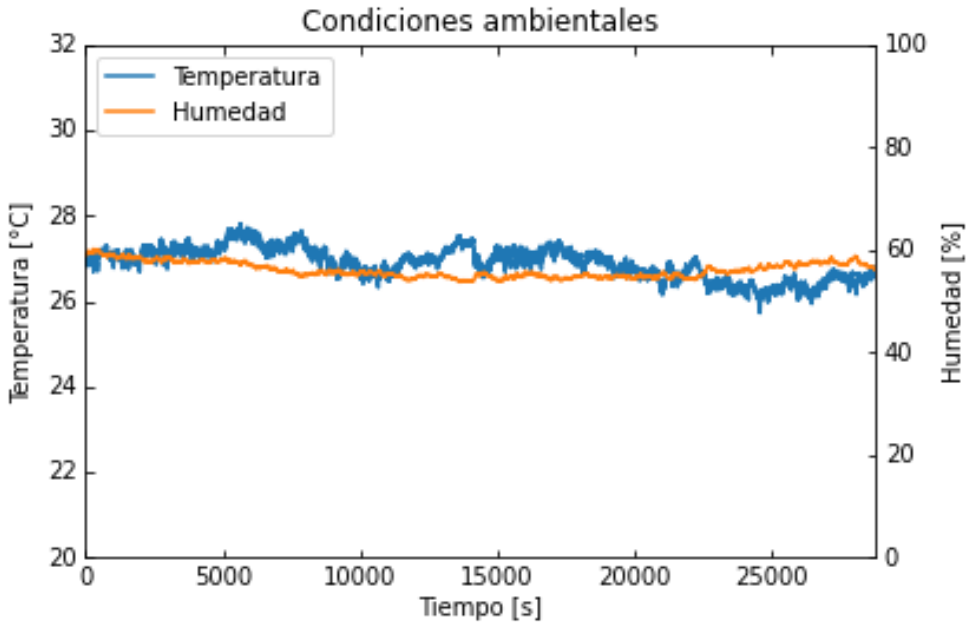


Figura 3.22 Día 5: Condiciones ambientales

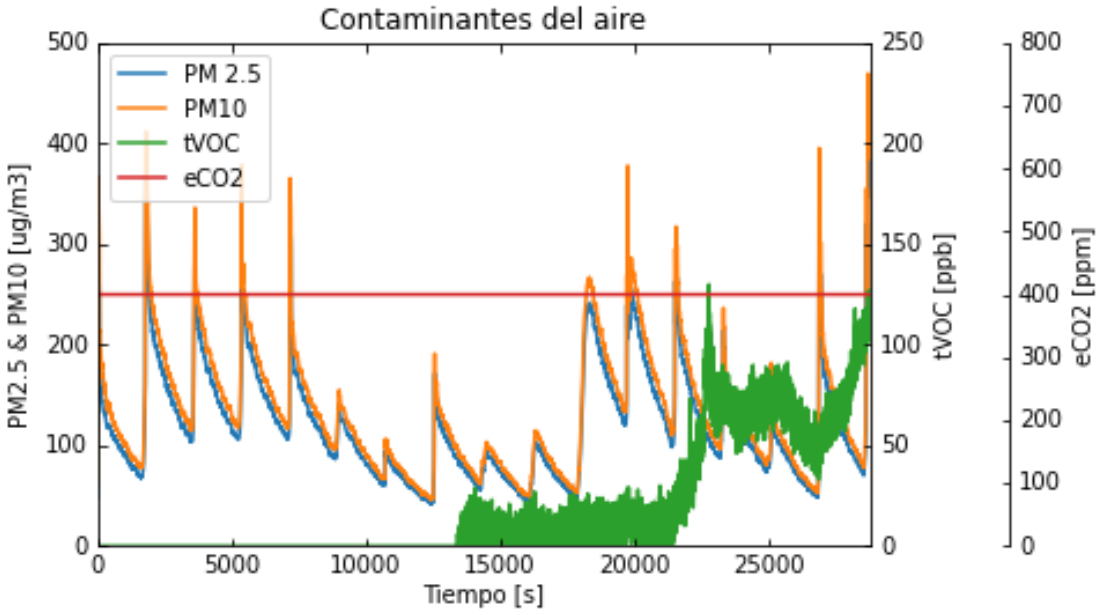


Figura 3.23 Día 6: Muestra de combustión

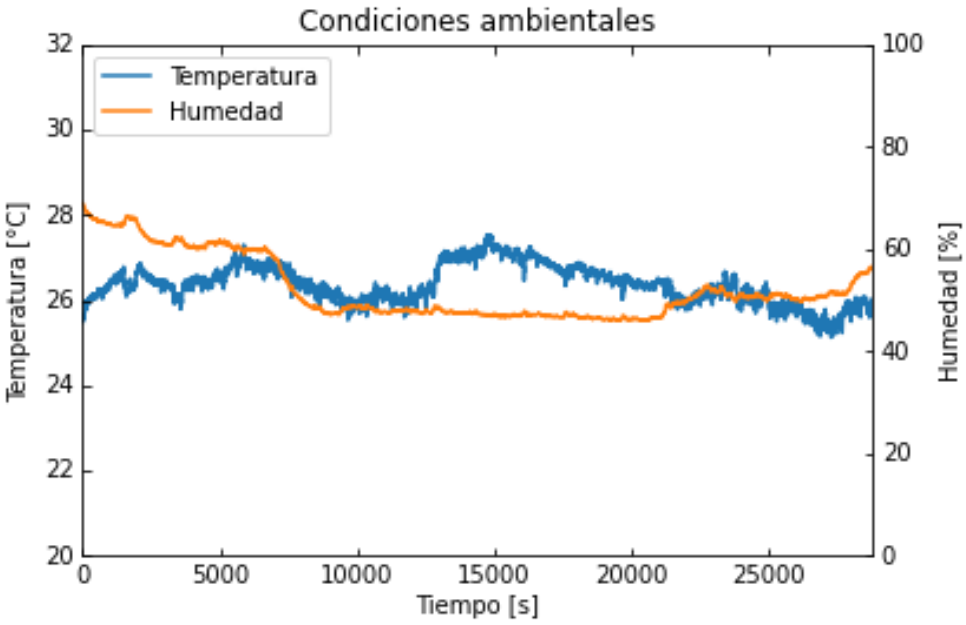


Figura 3.24 Día 6: Condiciones ambientales

En la tabla 3.3 se observan altos niveles de $PM_{2.5}$ y PM_{10} ,debido a la combustión de la pasta de soldar, entre 120.54 – 128.41 y 134.02 – 142.83 ug/m3 respectivamente. La concentración de tVOC y eCO_2 se encuentran en niveles bajos. Además, la temperatura y humedad se mantiene en niveles constantes entre 26.37 - 26.89 °C, y 52.22 - 56.11 % respectivamente.

Los altos niveles de $PM_{2.5}$ y PM_{10} sobrepasan los lineamientos de la OMS. Considerándolo un ambiente no sano debido a la posibilidad de generar infecciones, hasta daño a los pulmones, corazón y/o cerebro a largo plazo.

Tabla 3.3 Datos estadísticos de los días 5 y 6

Muestra	Combustión de la pasta de soldar			
Día	5		6	
Variable	**Promedio**	**Desviación estándar**	**Promedio**	**Desviación estándar**
$PM_{2.5}$ [ug/m3]	128.41	57.60	120.54	56.02
PM_{10} [ug/m3]	142.83	64.54	134.02	63.17
tVOC [ppb]	23.61	24.89	18.37	29.02
eCO_2 [ppb]	400.02	2.45	400.00	0.00
Temperatura [°C]	26,89	0.37	26.37	0.45
Humedad [%]	56.11	1.47	52.22	0.62

3.3.2 Híper parámetros del modelo

En la tabla 3.4 se muestran los híper parámetros utilizados y la cantidad de datos ingresados para el entrenamiento y prueba del modelo de IA.

Tabla 3.4 Total de datos e híper parámetros seleccionados

Total de datos	172800.00
Datos preprocesados y normalizados	166997.00
Datos para el entrenamiento (80%)	133597.00
Datos para pruebas (20%)	33400.00
Función de perdida	*Categorical Cross Entropy*
Optimizador	*RMSprop*
Razón de entrenamiento	1.00 e-4
Error (e)	e < 0.01
Épocas	175.00
Lotes	128.00

En las figuras 3.25 y 3.26 se observa que el modelo alcanzó una alta precisión con un error muy bajo. Estos valores son de 99.99% de precisión para entrenamiento y pruebas, mientras el error obtenido es menor a 0.01. De esta manera, se generó un buen modelo clasificador multiclase. Posteriormente, se lo convirtió en un modelo de TensorFlow Lite cuantizado a enteros.

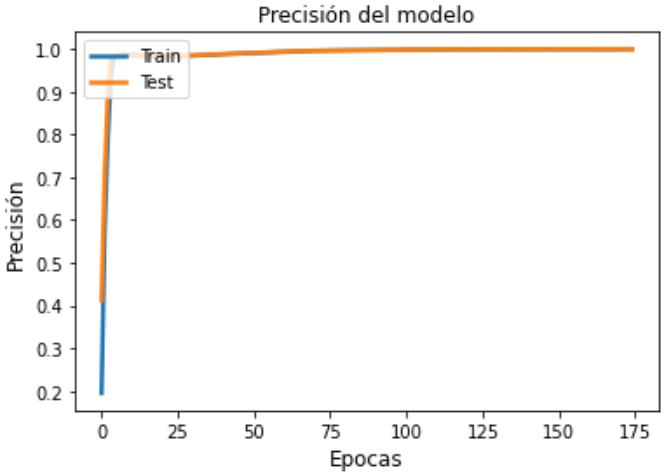


Figura 3.25 Precisión del modelo a través de las épocas

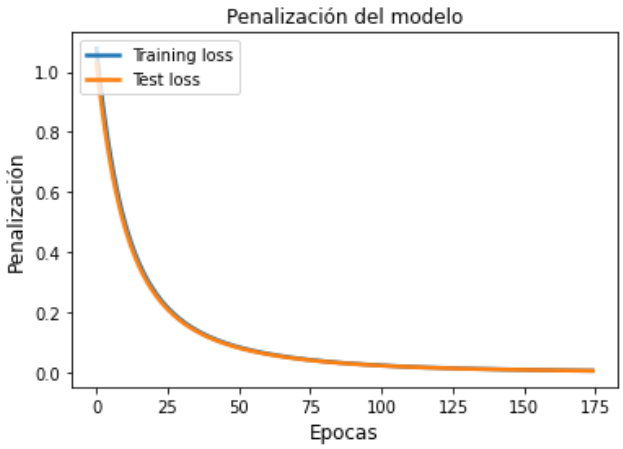


Figura 3.26 Penalización o pérdida del modelo a través de las épocas

En la tabla 3.5 se visualizan los pesos y *bias* de cada neurona del modelo obtenidos mediante Netron, visualizador de modelos de IA, como se observa en el apéndice D [39].

Tabla 3.5 Pesos y bias del modelo de IA

Neurona y	**Pesos W x,y**						**Bias bi**
y \ x	**1**	**2**	**3**	**4**	**5**	**6**	
1	-117	-126	-102	-116	28	29	1196
2	-112	-114	106	114	-1	-92	-1426
3	127	125	-51	-100	-34	7	-1351

A través de Netron se obtuvo la visualización del modelo, tal como se muestra en la figura 3.27. Los datos de entrada, concentración de contaminantes y condiciones ambientales, ingresan a la operación “Fully Connected”. Después, su resultado es computado en la función “Softmax” y en la salida se obtiene el resultado de cada predicción a realizar.

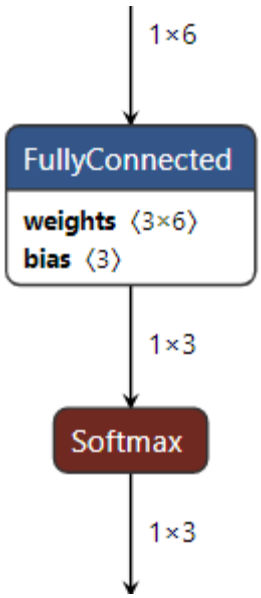


Figura 3.27 Visualización del modelo convertido en formato tflite y cuantizado a enteros mediante la aplicación Netron

3.3.3 Precisión del modelo

En esta primera parte, se muestran las matrices de confusiones obtenidas a partir de los datos de pruebas y el cálculo realizado para obtener la precisión del modelo mediante la ecuación 2.2.

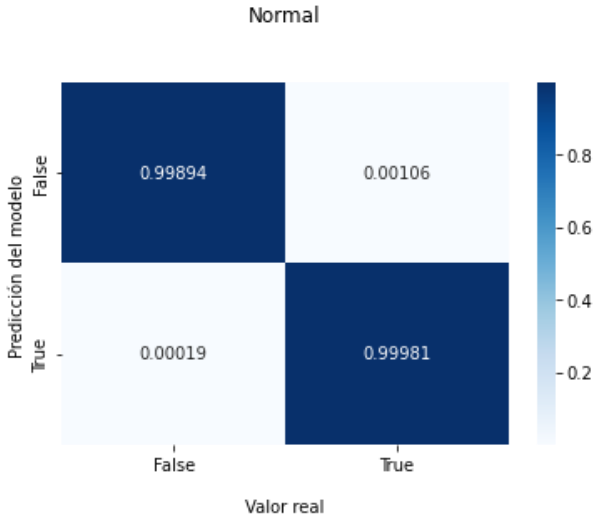


Figura 3.28 Matriz de confusión para la clase: Normal

$$Precisión_N = \frac{0.99981 + 0.99894}{0.99981 + 0.00019 + 0.99894 + 0.00106} \cdot 100 = 99.937\%$$

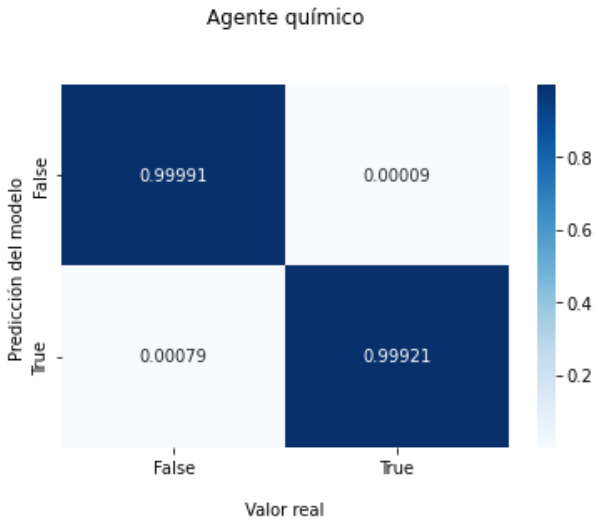


Figura 3.29 Matriz de confusión para la clase: Agente químico

$$Precisión_{AQ} = \frac{0.99921 + 0.99991}{0.99921 + 0.00079 + 0.99991 + 0.00009} \cdot 100 = 99.956\%$$

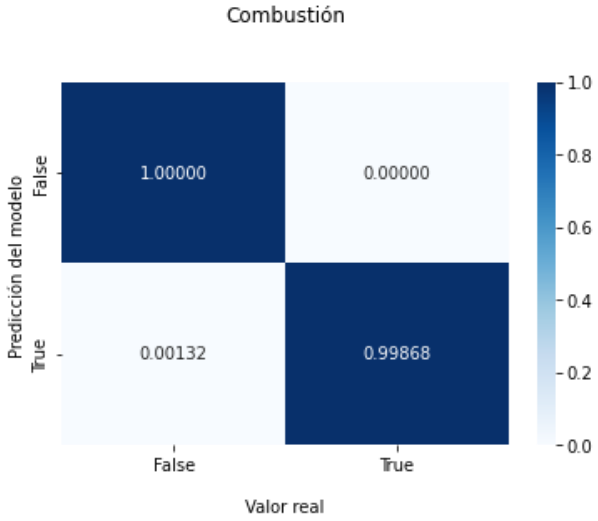


Figura 3.30 Matriz de confusión para la clase: Combustión

$$Precisión_C = \frac{0.99868 + 1.00000}{0.99868 + 0.00132 + 1.00000 + 0.00000} \cdot 100 = 99.934\%$$

Después, se muestran las matrices de confusión de las predicciones realizadas por el modelo de IA embebido y así obtener la precisión real del modelo.

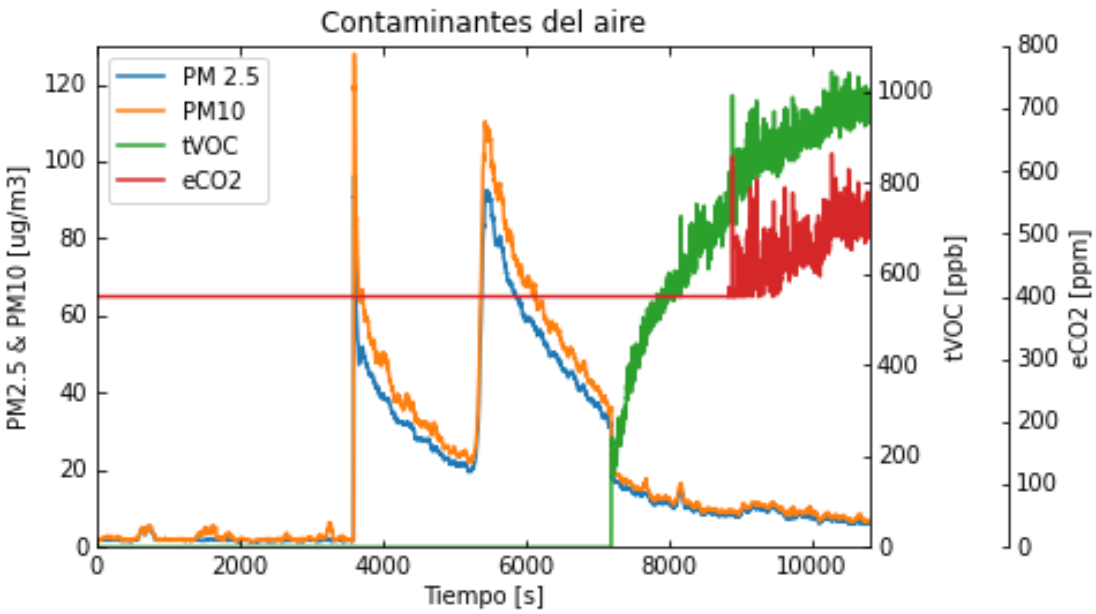


Figura 3.31 Datos recolectados de la prueba de 3 horas con el kit de desarrollo

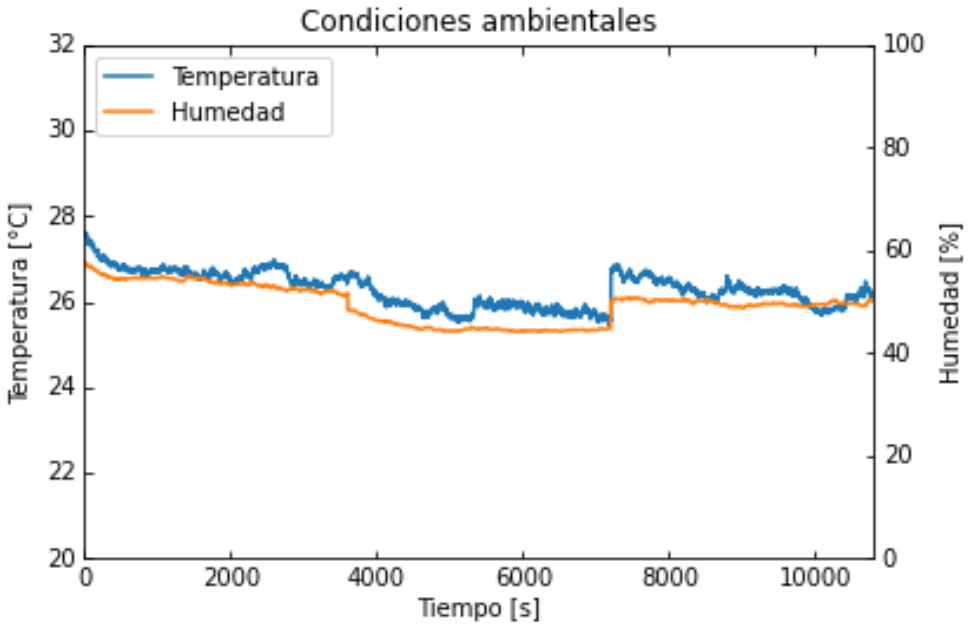


Figura 3.32 Condiciones ambientales de la prueba de 3 horas con el kit de desarrollo

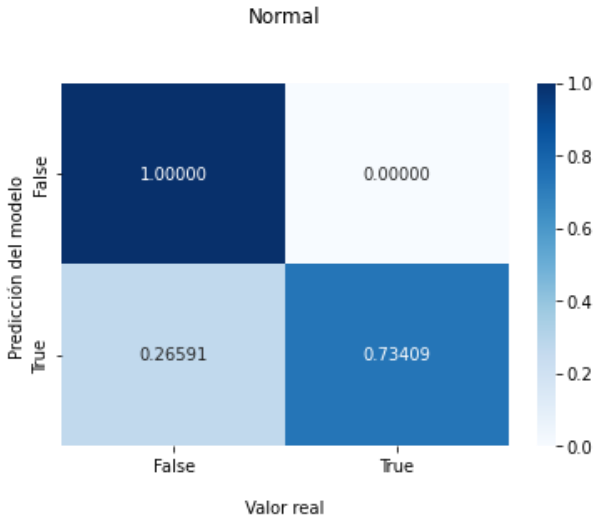


Figura 3.33 Matriz de confusión para la clase: Normal

$$Precisión_N = \frac{0.73409 + 1.00000}{0.73409 + 0.26581 + 1.00000 + 0.00000} \cdot 100 = 86.937\%$$

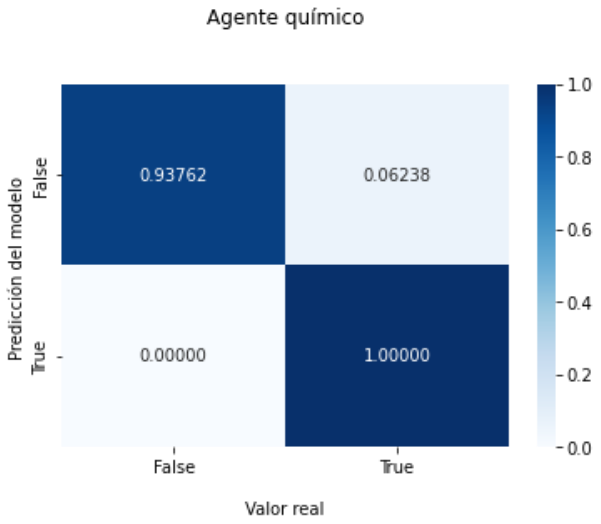


Figura 3.34 Matriz de confusión para la clase: Agente químico

$$Precisión_{AQ} = \frac{1.00000 + 0.93762}{1.00000 + 0.00000 + 0.93762 + 0.06238} \cdot 100 = 96.881\%$$

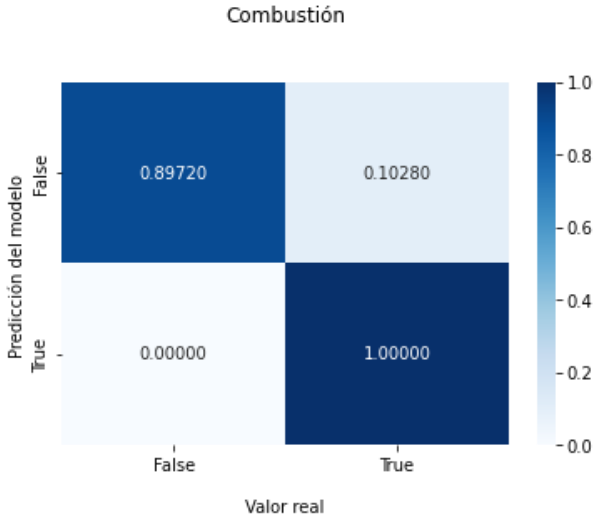


Figura 3.35 Matriz de confusión para la clase: Combustión

$$Precisión_{C} = \frac{1.00000 + 0.89720}{1.0000 + 0.00000 + 0.89720 + 0.10280} \cdot 100 = 94.860\%$$

A partir de las precisiones calculadas, se observa que la precisión del modelo es de 99.934% y disminuyó a 86.937% al ser embebido. Es de esperarse que su

rendimiento para predecir disminuya al ser convertido al formato de Tensor Flow Lite, cuantizado a entero y, para ser usado en el kit, convertido en un arreglo en C.

3.4 Diseño de *firmware*

En la siguiente sección se describe el algoritmo programado que leyó los sensores y predijo el tipo de contaminante en el ambiente a través de las librerías creadas que implementan el paradigma de programación reactiva y el patrón de diseño de objetos activos.

3.4.1 Funcionalidad del algoritmo

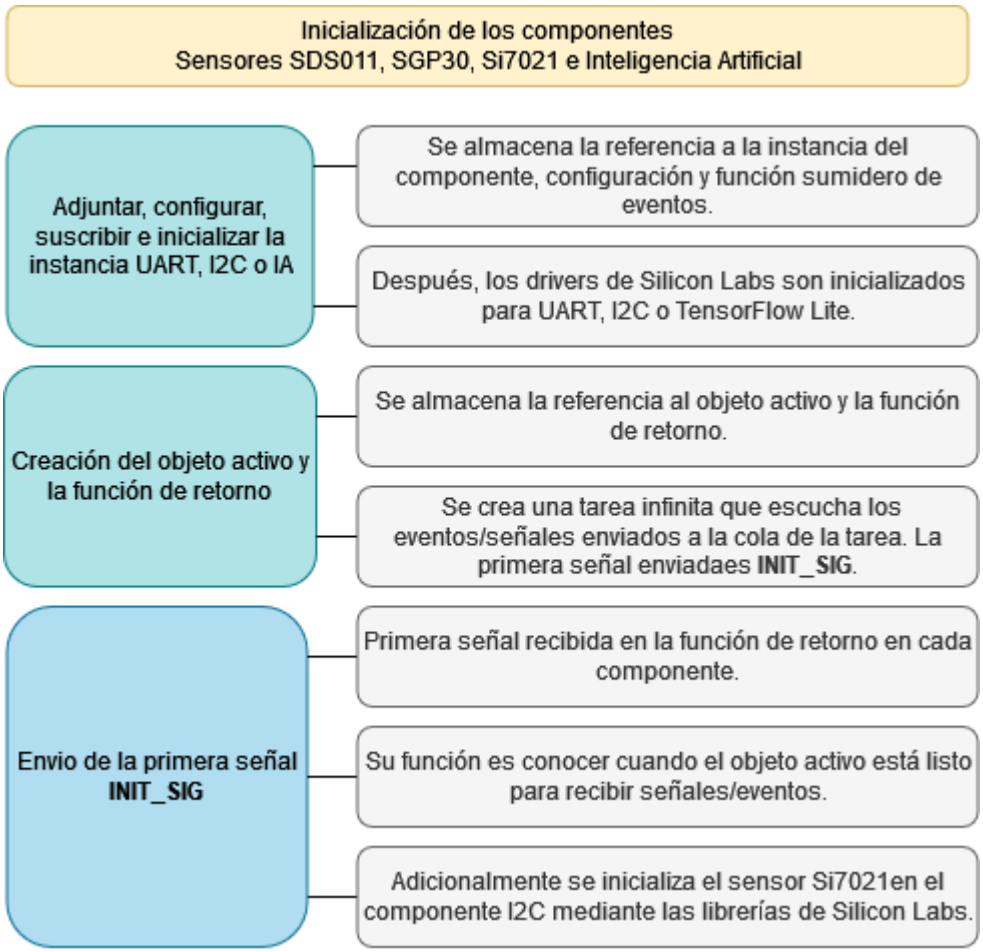


Figura 3.36 Inicialización de los componentes de la programación

En la figura 3.36, se describe secuencialmente como iniciar cada componente: UART, I^2C o IA. La instancia de cada componente y la función sumidero de eventos

fueron suscritos en la columna. Después, se enlazó un objeto activo junto a la función de retorno a cada componente.

En las figuras 3.37, 3.39 y 3.41 se describe la señal enviada a la cola de eventos del componente correspondiente, una breve descripción de lo que ocurre en su ejecución y las funciones internas invocadas en cada paso. Además, según el componente, se notifica a la función *callback* mediante la fuente de eventos suscrita internamente, si la lectura, escritura, o predicción fue realizada con éxito o es fallida. Luego, por medio de la función *post_event_queue* se envía reiteradamente las señales para la lectura de los sensores y predicción del tipo de ambiente. Revisar el apéndice E para más información del código.

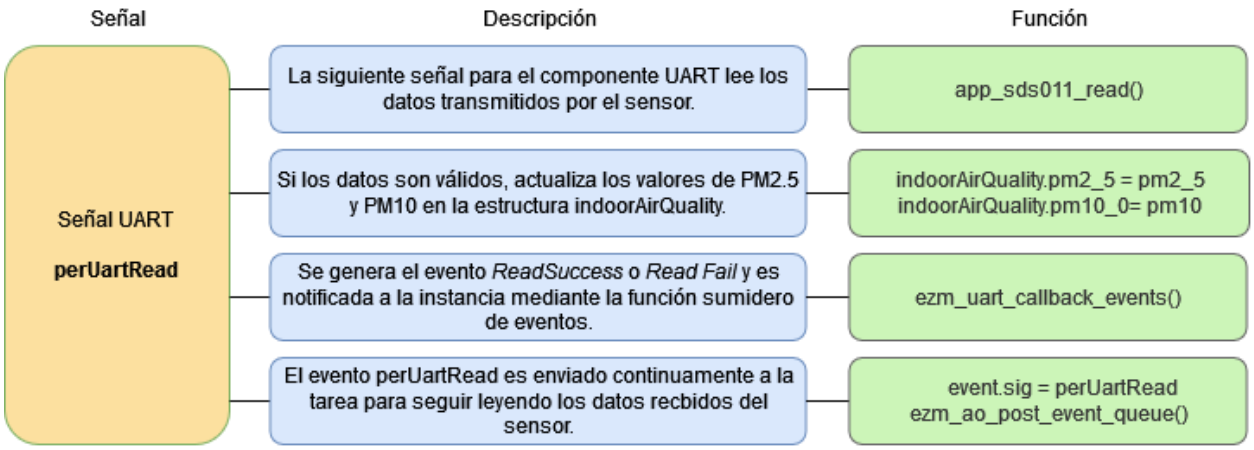


Figura 3.37 Funcionamiento del algoritmo del componente UART

El componente UART utilizó la función de la librería creada app_sds011, *sds011_read*, para leer el sensor de PM y validar sus datos.

```
[I] Air Quality Monitoring Started!
[I] Active Object Started
[I] Active Object Read
[I] Read Success
[I] PM 2.5: 9.2, PM10: 10.1
[I] Active Object Read
[I] Read Success
[I] PM 2.5: 9.2, PM10: 10.1
[I] Active Object Read
[I] Read Success
[I] PM 2.5: 9.1, PM10: 10.0
[I] Active Object Read
[I] Read Success
[I] PM 2.5: 9.1, PM10: 10.0
[I] Active Object Read
[I] Read Success
[I] PM 2.5: 9.1, PM10: 10.0
[I] Active Object Read
[I] Read Success
[I] PM 2.5: 9.1, PM10: 10.0
[I] Active Object Read
[I] Read Success
[I] PM 2.5: 9.1, PM10: 10.0
[I] Active Object Read
[I] Read Success
[I] PM 2.5: 9.1, PM10: 10.0
[I] Active Object Read
[I] 005b,0064,0000,0190,000073c9,0001002a,0
[I] Encender LED Azul: Ambiente Normal
```

Figura 3.38 Impresión de logs únicamente de UART y predicción

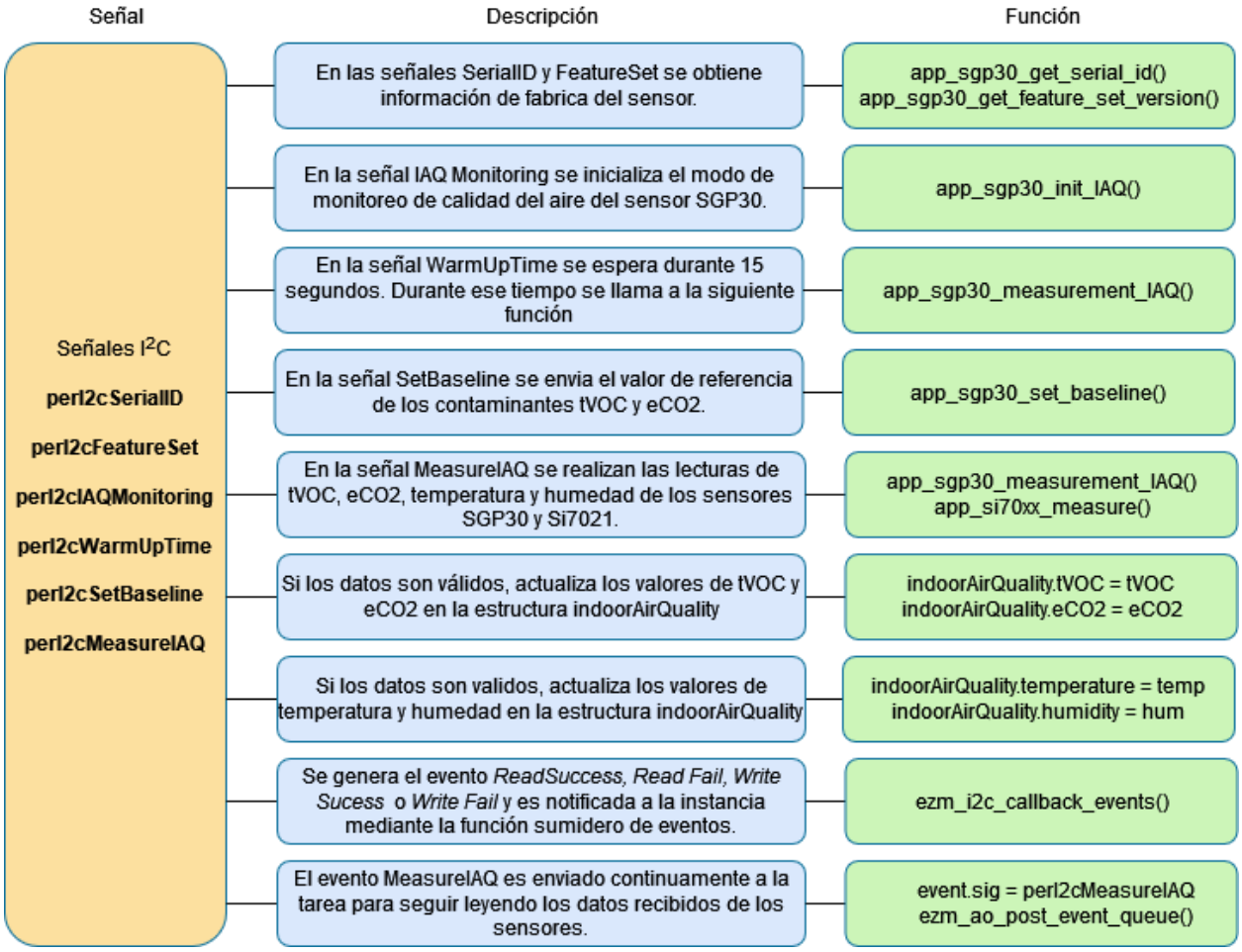


Figura 3.39 Funcionamiento del algoritmo del componente I²C

El componente I²C, invocó las funciones de la librería creada app_sgp30, para la obtención de parámetros, lectura y escritura del sensor SGP30, y de la librería de Silicon Labs para leer el sensor Si7021.

Las señales enviadas ejecutaban las funciones de acuerdo con la descripción correspondiente y así se obtuvo los datos del fabricante, se esperó durante 15 segundos hasta obtener datos válidos y se envió la base de referencia usada en la medición de contaminantes del aire.

```
[I] Air Quality Monitoring Started!
[I] Active Object i2c Started
[I] Active Object: Get Serial ID
[I] Write Success
[I] Read Success
[I] Active Object: Get Feature Set
[I] Write Success
[I] Read Success
[I] Active Object: Start IAQ Monitoring
[I] Write Success
[I] Waiting for 15 seconds warm up...
[I] Reading zero values before start...
[I] Write Success
[I] Read Success
[I] Reading zero values before start...
[I] Write Success
[I] Read Success
[I] Reading zero values before start...
[I] Write Success
[I] Read Success
[I] Active Object: Set Baseline
[I] Write Success
[I] Active Object: Get Baseline
[I] Write Success
[I] Read Success
[I] Active Object: Read from sensors
[I] Write Success
[I] Read Success
[I] tVOC: 0, eCO2: 400, Temp: 30.39, Hum: 60.80
[I] 003d,0043,0000,0190,000073c9,0000ed82,0
[I] Encender LED Azul: Ambiente Normal
```

Figura 3.40 Impresión de logs únicamente de I^2C y predicción. (Se omitió segundos de calibración)

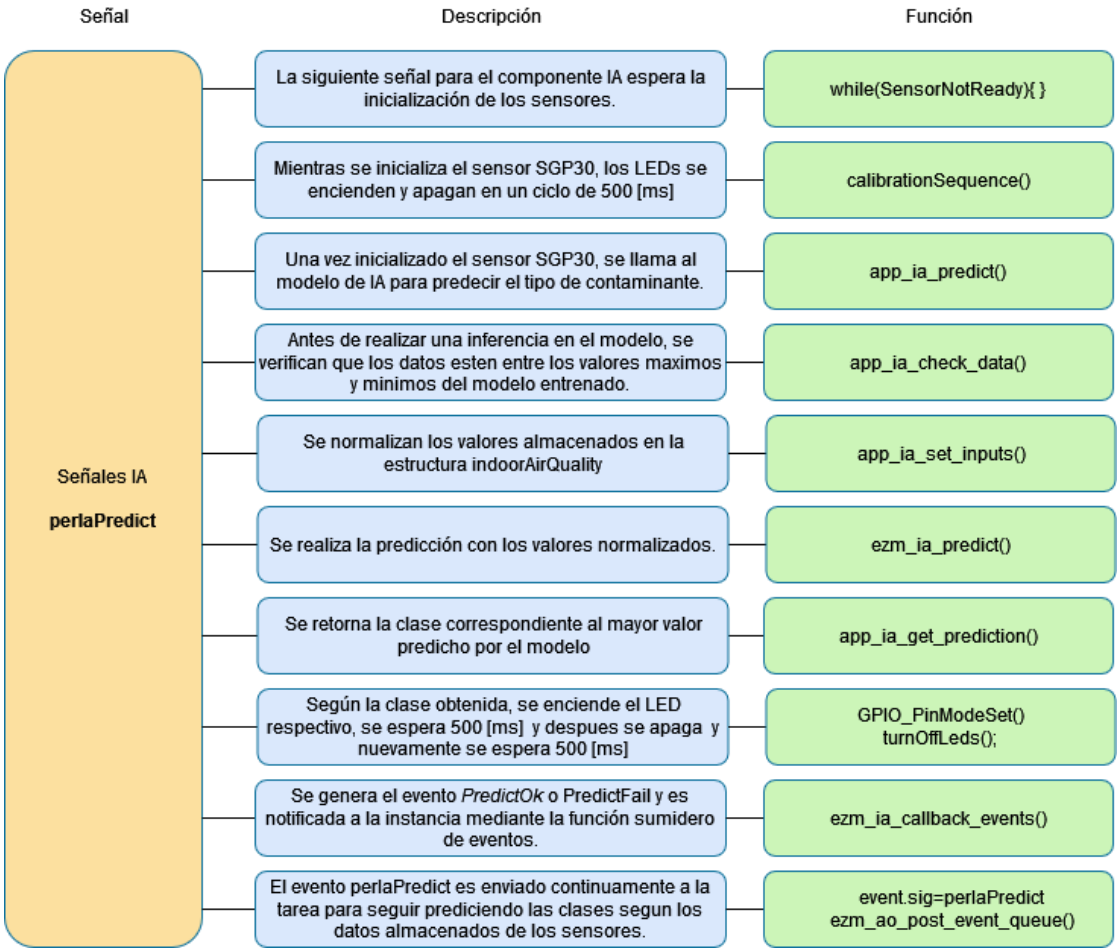


Figura 3.41 Funcionamiento del algoritmo del componente IA

El componente IA, sigue una serie de pasos antes de invocar la función de la librería creada app_ia, *app_ia_predict*, para predecir el tipo de contaminante en el ambiente. En la señal *perIaPredict*, se verificó que los sensores estén listos para medir el nivel de contaminación en el aire y las condiciones ambientales.

Una vez los sensores envían mediciones validas, se invocó a la función *app_ia_predict*, en donde se validó los rangos de los datos, fueron normalizados y cuantizados para ingresar al modelo de IA. Después, el resultado de la predicción se utilizó para parpadear el LED correspondiente a la clase predicha.

```
[I] Air Quality Monitoring Started!
[I] Init Signal Machine Learning
[I] Calibrating...
[I] Calibrating...
[I] Calibrating...
[I] Calibrating...
[I] Calibrating...
[I] Calibrating...
[I] Calibrating...
[I] Calibrating...
[I] Calibrating...
[I] Calibrating...
[I] Calibrating...
[I] Calibrating...
[I] Calibrating...
[I] Calibrating...
[I] Calibrating...
[I] Calibrating...
[I] Predict: OK
[I] PM 2.5: 4.7, PM10: 5.1 [I] tVOC: 0, eCO2: 400, Temp: 28.19, Hum: 51.22
[I] 002f,0033,0000,0190,00006e21,0000c814,0
[I] Encender LED Azul: Ambiente Normal
[I] Predict: OK
[I] PM 2.5: 4.7, PM10: 5.1 [I] tVOC: 0, eCO2: 400, Temp: 28.20, Hum: 51.23
[I] 002f,0033,0000,0190,00006e2c,0000c81c,0
[I] Encender LED Azul: Ambiente Normal
[I] Predict: OK
[I] PM 2.5: 4.7, PM10: 5.1 [I] tVOC: 0, eCO2: 400, Temp: 28.19, Hum: 51.21
[I] 002f,0033,0000,0190,00006e21,0000c80c,0
[I] Encender LED Azul: Ambiente Normal
```

Figura 3.42 Impresión de logs únicamente de IA y predicciones.

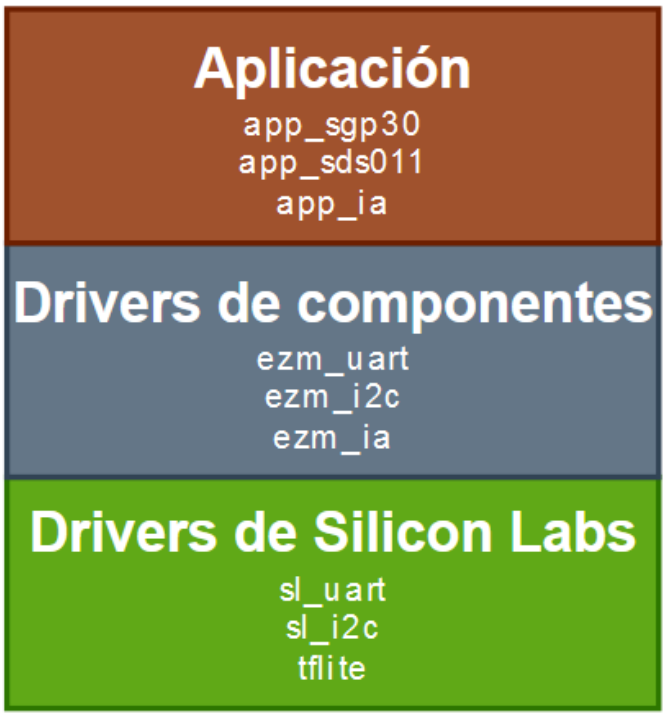


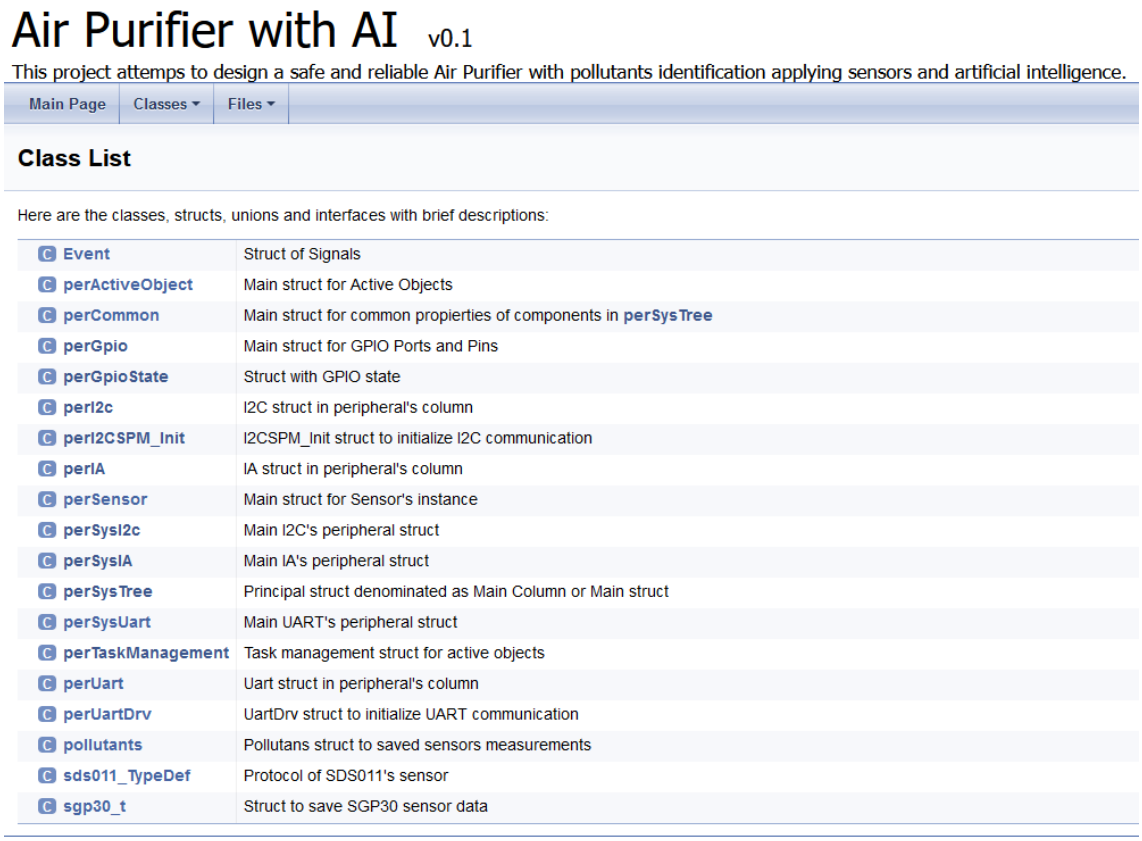
Figura 3.43 Capas del sistema embebido

En la figura 3.43 se puede observar las capas creadas para el *firmware*: aplicación y drivers de componentes. En la capa de aplicación se llamó a las funciones para inicialización, configuración, suscripción e inicialización de los componentes. Además de adjuntar las funciones sumidero y de retorno a la instancia del componente correspondiente.

En la capa de componente de drivers se implementaron todas las funciones necesarias para la configuración inicial de la aplicación, adjuntar funciones *callback* e instancias de los componentes en la columna principal. En esta capa se incluyeron los drivers de Silicon Labs para los componentes de UART, I2C e Inteligencia artificial, TensorFlow Lite.

3.4.2 Documentación

Las figuras a continuación muestran las principales clases y archivos documentados que conforman el código fuente. Para mayor detalle de la documentación, revisar el apéndice F.



Air Purifier with AI v0.1

This project attemps to design a safe and reliable Air Purifier with pollutants identification applying sensors and artificial intelligence.

Main Page | Classes | Files

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Event	Struct of Signals
perActiveObject	Main struct for Active Objects
perCommon	Main struct for common propierties of components in perSysTree
perGpio	Main struct for GPIO Ports and Pins
perGpioState	Struct with GPIO state
perI2c	I2C struct in peripheral's column
perI2CSPM_Init	I2CSPM_Init struct to initialize I2C communication
perIA	IA struct in peripheral's column
perSensor	Main struct for Sensor's instance
perSysI2c	Main I2C's peripheral struct
perSysIA	Main IA's peripheral struct
perSysTree	Principal struct denominated as Main Column or Main struct
perSysUart	Main UART's peripheral struct
perTaskManagement	Task management struct for active objects
perUart	Uart struct in peripheral's column
perUartDrv	UartDrv struct to initialize UART communication
pollutants	Pollutans struct to saved sensors measurements
sds011_TypeDef	Protocol of SDS011's sensor
sgp30_t	Struct to save SGP30 sensor data

Figura 3.44 Descripción de las estructuras principales

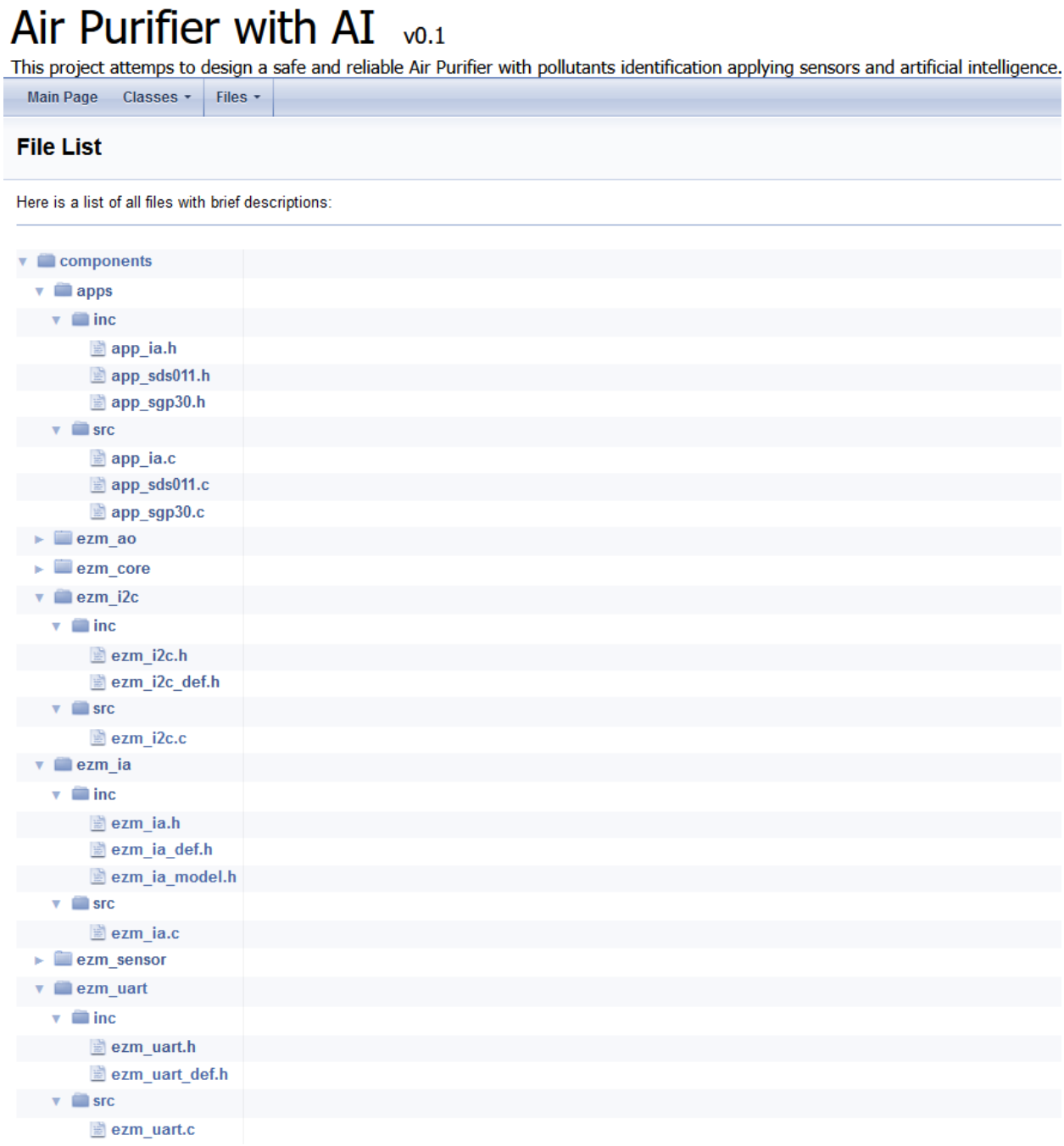


Figura 3.45 Detalle de los archivos de cabecera y fuente de los componentes y librerías.

3.5 Análisis de costo

En esta sección se mencionará y analizará los costos aproximados para el estado actual del proyecto descrito en secciones anteriores.

3.5.1 Costos de ingeniería

En la tabla 3.6, se describen los rubros correspondientes al costo de viáticos e ingeniería en razón a la elaboración del diseño mecánico y programación.

Tabla 3.6 Costo de ingeniería y viáticos

Componente	Pago	Precio unit. [USD]
Viáticos	Único	$300
Diseño de la estructura mecánica	Único	$1800.00
Diseño del *firmware* e inteligencia artificial	Único	$1800.00
Total		**$3900.00**

3.5.2 Diseño mecánico

Para esta etapa de prototipado del proyecto se tiene un precio unitario de $87.57 con respecto a la parte mecánica, tal como se muestra en la tabla 3.7. Deber tomarse en cuenta que los rubros cotizados son para la realización de un prototipo.

Tabla 3.7 Costos de los componentes del diseño mecánico

Componente	Modelo	Precio unit. [USD]
Impresión de la estructura mecánica.	N/A	$61.15
Ventilador axial	AFB0612DH-TP11	$12.43
Batería 12	DNK-LTB3S1PC18EH-R	$12.50
Filtro HEPA	Prefiltro + Filtro carbón activado	$1.49
Total		**$87.57**

3.5.3 Componentes electrónicos seleccionados

Por otro lado, se observa que los precios de los componentes electrónicos de manera unitaria alcanzan los $34.52 como se indica en la tabla 3.8.

Tabla 3.8 Costos de los componentes electrónicos

Componente	Modelo	Precio unit. [USD]
Puerto de carga USB	UJ31-CH-3-SMT-TR	$1.51
Leds [En paquetes de 10]	WS2812	$3.95
Sensor de PM 2.5 y PM 10 [ug/m3]	SDS011	$15.00
Sensor de tVOC [ppb] y eCO2[ppm]	SGP30	$7.46
Sensor de temperatura [°C] y humedad [%]	Si7021	$4.32
Microcontrolador	EFR32BG22	$2.28
Total		**$34.52**

El costo unitario por prototipo es de $122,09, sin embargo, los costos pueden disminuir a medida que las piezas son adquiridas en grandes cantidades de cientos o miles de unidades. Además, el costo único del diseño mecánico y de *firmware* se calculó en base a las 200 horas trabajadas con un precio de $18 por hora, y, adicionalmente, viáticos por $300.

CAPÍTULO 4

4. CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

Se logró diseñar una estructura mecánica de un purificador de aire portable junto a la selección de actuadores y filtros para el correcto abaste de aire filtrado por el sistema. El uso de software especializado en diseño mecánico, Inventor, permitió crear piezas para formar la estructura mecánica que sostiene los elementos mencionados, además de varios componentes adicionales que son la base para un prototipo inicial del proyecto.

La selección de filtros y actuadores es el resultado de la investigación acerca de las normativas que brindan distintas entidades dedicadas al acondicionamiento y purificación de aire enfocada en sistemas portables. No está de más decir que, la previa lectura de los principios físicos que rigen estos sistemas fue de gran importancia para la seguridad de que los filtros y ventiladores seleccionados sean capaz de abastecer las necesidades planteadas a lo largo del capítulo.

El diseño mecánico propuesto en cuanto a capacidad de desplazamiento de aire y resistencia máxima a fluido que soporta posee un factor de seguridad acorde a la teoría acerca de la resistencia de filtros HEPA a lo largo de su vida útil. No se ahondó en profundidad en el estudio de pérdidas en el trayecto del fluido en su totalidad debido a que se posee velocidades relativamente bajas.

Se diseñó el *firmware* requerido para el purificador de aire portable que cumple con las características de la programación reactiva al crear una fuente de eventos que indicaba cuando la lectura y/o escritura del sensor o la predicción era procesada de manera exitosa o fallida y, mediante la fuente de eventos suscrita, se notifica a la instancia del componente al enviar el evento a la función sumidero suscrita.

Adicionalmente, se implementó de manera correcta el patrón de diseño de objetos activos dentro de la columna al crear la pared de encapsulamiento de la tarea y recibir

únicamente los señales o eventos de las librerías a través de la cola de eventos. Los datos se mantenían privados e íntegros mientras las funciones eran ejecutadas dentro del caso del evento y al culminar el tiempo de ejecución de las funciones, la cola de eventos enviaba la siguiente señal para leer las mediciones de los sensores y predecir el tipo de contaminante.

Las librerías creadas permitieron crear funciones para leer el sensor SDS011, escribir, leer y configurar el sensor SGP30 y, además, implementar la funcionalidad de predicción en el componente de IA. Adicionalmente, todos los datos adquiridos mediante los sensores eran validados, preprocesados y enviados al modelo de IA para predecir el tipo de contaminante, de esta manera se mantuvo la integridad de los datos y la veracidad de la predicción realizada.

El modelo de inteligencia artificial utilizado ha sido el adecuado para la clasificación multiclase de tipos de contaminantes en el ambiente debido a que ha presentado una precisión del 99,934%. No obstante, la precisión del modelo embebido disminuyo drásticamente a 86.937%. Este bajo rendimiento es provocado por la conversión a arreglo en C. Además, también influyó que las condiciones ambientales utilizadas para el entrenamiento del modelo de IA, como se puede observar en las tablas 3.1, 3.2 y 3.3, tuvo un mayor rango para los días 1 y 2 en comparación a los días 3, 4, 5 y 6. De esta manera, predijo erróneamente el 26% de los datos al clasificarlos como aire ambiente, como se observa en la figura 3.31.

4.2 Recomendaciones

El diseño mecánico queda sujeto a cambios debido a que la cotización y selección de los elementos seleccionados no es la definitiva. Se debe siempre tener en mente que pueden existir distintas formas de realizar los objetivos planteados en este documento. La selección de mejores sensores, actuadores y filtro puede cambiar drásticamente las dimensiones y ensamble del diseño mostrado.

Queda pendiente la implementación y ensamble del prototipo físico, así como la compra de los componentes mencionados a lo largo de este documento. Las pruebas en

conjunto de la parte mecánica, *firmware* y electrónica deben realizarse de manera sistemática y exhaustiva para dar paso a un prototipo final a través de la prueba y error.

Se debe recordar que la elaboración de este proyecto corresponde a un prototipo inicial funcional del producto. Por ende, se puede reducir costos en la etapa de producción final. Esto debido a que un diseño cuidadoso y elaborado de piezas mecánicas las hace aptas para procesos de manufactura que, si bien son de inversiones altas, las piezas unitarias son mucho más baratas. De forma similar, el precio de componentes por unidad va reduciendo a medida que el volumen del pedido aumenta.

Queda pendiente el estudio detallado del fluido a lo largo del diseño mecánico, de tal forma que se pueda analizar las pérdidas totales considerando además los elementos que causan grandes resistencias dentro del diseño tales como filtros, cambios de áreas bruscos, etc. Este estudio puede ser realizado por software especializado como ANSYS CFD; siempre tomando en cuenta las buenas y correctas prácticas para realizar un mallado pertinente y tener resultados cercanos a la realidad. Además, claro está, basarnos previamente en un cálculo teórico como referencia para simulaciones.

Se recomienda optimizar la memoria del microcontrolador debido al alto consumo de este por las librerías propias e implementadas que impediría añadir más características funcionales como conexión inalámbrica por Bluetooth Low Energy que integra el kit de desarrollo.

Se recomienda utilizar otro microcontrolador de Silicon Labs, por la facilidad de migración del código fuente, con conectividad inalámbrica para enviar datos a través de Bluetooth Low Energy y WiFi lo que permitirá integrar la visualización de la contaminación del aire y la predicción en tiempo real a través de una aplicación móvil o web.

Se recomienda integrar sensores adicionales, de tamaño reducido, para medir la concentración de gases nocivos como lo son el O_3, NO_2, y CO en el ambiente. De esta manera se puede monitorear con mayor precisión la calidad del aire del ambiente. Además, se pueden recopilar datos con los nuevos sensores, preprocesarlos, normalizarlos y entrenar nuevamente el algoritmo de IA con las nuevas entradas.

Se recomienda entrenar el modelo de inteligencia artificial para la clasificación multiclase de tipos de contaminantes utilizando técnicas de Deep Learning y de preprocesamiento de datos tal como el análisis de componentes principales, por sus siglas en inglés, PCA, dado que en otros estudios se han realizado estos modelos de IA y obtenido un mayor porcentaje de predicción al ser embebidos en comparación al resultado obtenido de nuestro modelo. Adicionalmente, se debería disminuir el peso de las variables de temperatura y humedad en el algoritmo y así evitar que las pequeñas variaciones de las condiciones ambientales afecten a la precisión y predicción del modelo.

BIBLIOGRAFÍA

[1] A. Nasr and L. Abdallah, “Using robots to improve indoor air quality and reduce COVID-19 exposure,” *Journal of Applied Research and Technology*, vol. 19, no. 3, pp. 227–237, Aug. 2021, doi: 10.22201/icat.24486736e.2021.19.3.1694.

[2] V. Díaz Suárez *et al.*, “FICHA TÉCNICA Investigación Análisis y Monitoreo Secretaría de Ambiente.” [Online]. Available: www.quitoambiente.gob.ec,

[3] ALCALDÍA DE CUENCA and EMOV, “INFORME DE CALIDAD AIRE CUENCA 2020,” 2020.

[4] V. van Tran, D. Park, and Y. C. Lee, “Indoor air pollution, related human diseases, and recent trends in the control and improvement of indoor air quality,” *International Journal of Environmental Research and Public Health*, vol. 17, no. 8. MDPI AG, Apr. 02, 2020. doi: 10.3390/ijerph17082927.

[5] I. Manisalidis, E. Stavropoulou, A. Stavropoulos, and E. Bezirtzoglou, “Environmental and Health Impacts of Air Pollution: A Review,” *Frontiers in Public Health*, vol. 8. Frontiers Media S.A., Feb. 20, 2020. doi: 10.3389/fpubh.2020.00014.

[6] “State of Global Air,” 2020.

[7] S. D. Lowther *et al.*, “How efficiently can HEPA purifiers remove priority fine and ultrafine particles from indoor air?,” *Environment International*, vol. 144, Nov. 2020, doi: 10.1016/j.envint.2020.106001.

[8] Z. A. Wendling, J. W. Emerson, A. de Sherbinin, and D. C. Esty, “Environmental Performance Index,” New Haven, CT, 2020. Accessed: Oct. 20, 2021. [Online]. Available: https://epi.yale.edu/

[9] UNICEF, “EL AIRE QUE RESPIRAMOS,” Quito, 2020. [Online]. Available: www.unicef.org/ecuador

[10] E. G. Snyder *et al.*, “The changing paradigm of air pollution monitoring,” *Environmental Science and Technology*, vol. 47, no. 20, pp. 11369–11377, Oct. 2013, doi: 10.1021/es4022602.

[11] R. Vinuesa *et al.*, “The role of artificial intelligence in achieving the Sustainable Development Goals,” *Nature Communications*, vol. 11, no. 1. Nature Research, Dec. 01, 2020. doi: 10.1038/s41467-019-14108-y.

[12] K. R. Smith and S. Mehta, “The burden of disease from indoor air pollution in developing countries: comparison of estimates,” 2003. [Online]. Available: http://www.urbanfischer.de/journals/intjhyg

[13] B. Mette, K. Thomas, V. Ole, P. Janet, and C. Nordic, “The invisible killer - health effects of air pollution,” 2009.

[14] J. Lelieveld, J. S. Evans, M. Fnais, D. Giannadaki, and A. Pozzer, “The contribution of outdoor air pollution sources to premature mortality on a global scale,” *Nature*, vol. 525, no. 7569, pp. 367–371, Sep. 2015, doi: 10.1038/nature15371.

[15] World Health Organization, “WHO global air quality guidelines. Particulate matter (PM2.5 and PM10), ozone, nitrogen dioxide, sulfur dioxide and carbon monoxide.,” Ginebra, 2021.

[16] Ministerio del Ambiente, “NORMA ECUATORIANA DE CALIDAD DEL AIRE,” Jun. 2011.

[17] ASHRAE, *Ventilation for Acceptable Indoor Air Quality*. Atlanta, 2013.

[18] B. Maag, Z. Zhou, and L. Thiele, “W-Air,” *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 1, pp. 1–25, Mar. 2018, doi: 10.1145/3191756.

[19] Tekhnicheski universitet--Sofiia and Institute of Electrical and Electronics Engineers, *2019 XXVIII International Scientific Conference Electronics (ET) : proceedings : September 12-14, 2019, Sozopol, Bulgaria*.

[20] Peter. Abraham and Praseed. Pai, *C++ Reactive Programming : Design Concurrent and Asynchronous Applications Using the Rxcpp Library and Modern C++17.* Packt Publishing Ltd, 2018.

[21] M. Samek, “Modern Embedded Software Goes Beyond the RTOS,” *Modern Embedded Software Goes Beyond the RTOS*, 2020. https://www.embeddedonlineconference.com/session/Modern_Embedded_Software_Goes_Beyond_the_RTOS (accessed Jan. 07, 2022).

[22] M. Samek, “State Machines & Tools for Embedded Systems,” *Active Objects (Actor)*, 2020. https://www.state-machine.com/active-object (accessed Jan. 07, 2022).

[23] R. Welsch, C. von Castell, and H. Hecht, “The anisotropy of personal space.,” *PLoS One*, vol. 14, no. 6, 2019, doi: 10.1371/journal.pone.0217587.

[24] “Guide to Air Cleaners in the Home Portable Air Cleaners and Furnace or HVAC Filters in the Home,” Jul. 2018. Accessed: Jan. 08, 2022. [Online]. Available: www.epa.gov/laq

[25] Z. Xu, *Air purifier: Property, assessment and applications*. Springer Singapore, 2018. doi: 10.1007/978-981-13-2532-8.

[26] “ACGIH: Industrial Ventilation,” Ohio, 1998.

[27] T. North, “The Physics of Airflow through a Perforated Plate.” Accessed: Jan. 08, 2022. [Online]. Available: www.bicsi.org

[28] SilverStone, “How do different fan mesh patterns affect fan and chassis airflow ?”

[29] B. Andriy Burkov, “The Hundred-Page Machine Learning,” 2019.

[30] ARM, “Why Arm? CPU ARCHITECTURE.” https://www.arm.com/why-arm/architecture/cpu (accessed Jan. 07, 2022).

[31] Silicon Labs, “UG464: Thunderboard™ EFR32BG22 User’s Guide,” Jun. 2021.

[32] Beningo Jacob, “Embedded RELATED,” *From Baremetal to RTOS: A review of scheduling techniques*, Jun. 08, 2016. https://www.embeddedrelated.com/showarticle/969.php (accessed Jan. 07, 2022).

[33] sparkfun, “sparkfun,” *SparkFun Air Quality Sensor - SGP30 (Qwiic)*. https://www.sparkfun.com/products/16531 (accessed Jan. 08, 2022).

[34] Ltd. Nova Fitness Co., “Laser PM2.5 Sensor specification Product model: SDS011 Version: V1.3,” Oct. 2015.

[35] K. L. Alvarez C, R. F. Lagos C, and M. Aizpun, “Influencia del porcentaje de relleno en la resistencia mecánica en impresión 3D, por medio del método de Modelado por Deposición Fundida (FDM),” *Ingeniare. Revista chilena de ingeniería*, vol. 24, no. Especial, pp. 17–24, Aug. 2016, doi: 10.4067/S0718-33052016000500003.

[36] Craftcloud All3DP, “Material Guide,” *https://craftcloud3d.com/material-guide*. https://craftcloud3d.com/material-guide (accessed Feb. 27, 2022).

[37] Inc. ENVIRONMENTAL Analytical Service, “Total Volatile Organic Compounds.” https://easlab.com/iaqref.htm (accessed Jan. 31, 2022).

[38] WISCONSIN DEPARTMENT of HEALTH SERVICES, “Carbon Dioxide.” https://www.dhs.wisconsin.gov/chemical/carbondioxide.htm (accessed Jan. 31, 2022).

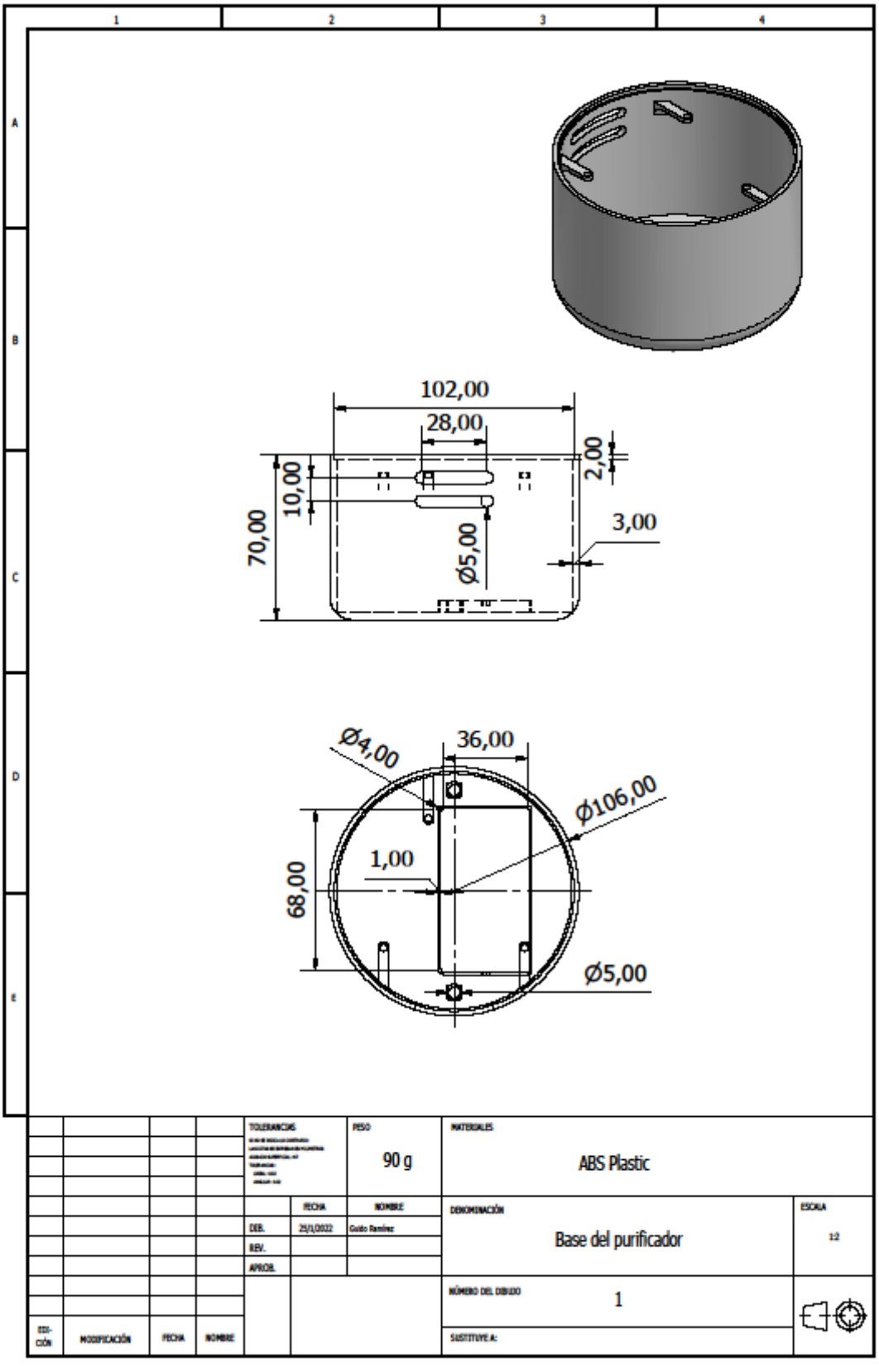
[39] L. Roeder, “NETRON,” Dec. 26, 2011. https://github.com/lutzroeder/netron (accessed Jan. 26, 2022).

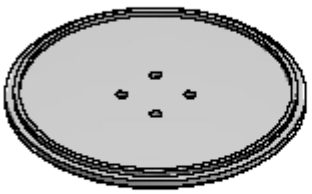
[40] A. England and C. Jekel, “SGP 30 Dimensions,” *SGP30 dimensions*. https://cdn.sparkfun.com/assets/learn_tutorials/1/1/7/4/SGP30_Dimensions.png (accessed Jan. 08, 2022).

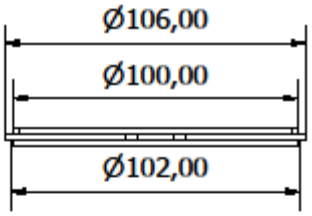
APÉNDICES

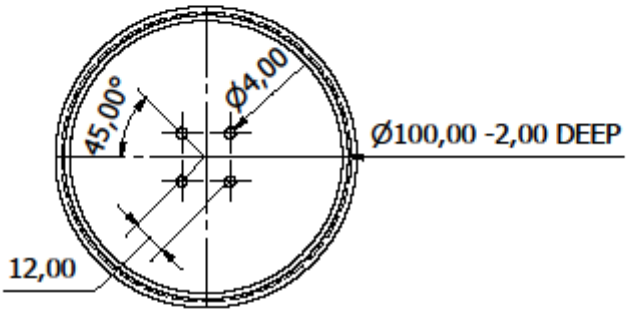
APÉNDICE A

Planos mecánicos

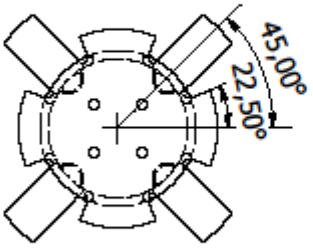


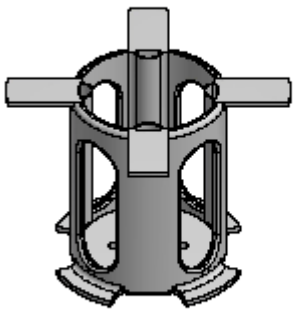


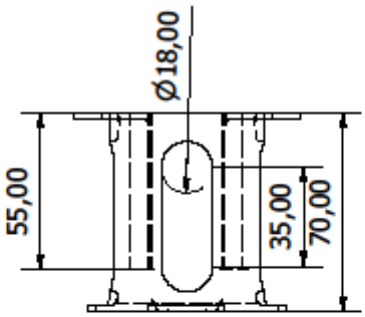


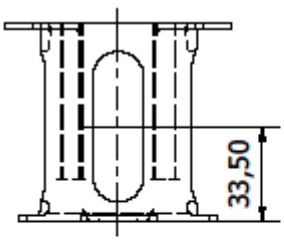


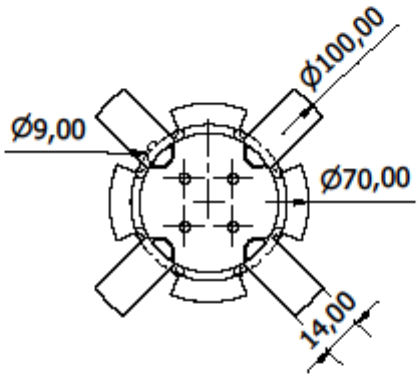
				TOLERANCIAS:		PESO: 25 g	MATERIALES: ABS Plastic
					FECHA	NOMBRE	DENOMINACIÓN: Tapa de la base del purificador
				DIB.	25/1/2022	Guido Ramírez	ESCALA: 1:2
				REV.			
				APROB.			NÚMERO DEL DIBUJO: 2
EDICIÓN	MODIFICACIÓN	FECHA	NOMBRE				SUSTITUYE A:



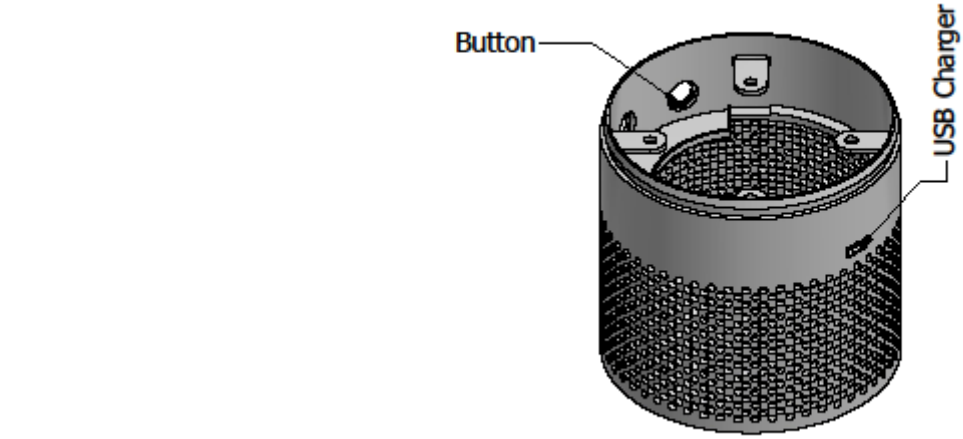


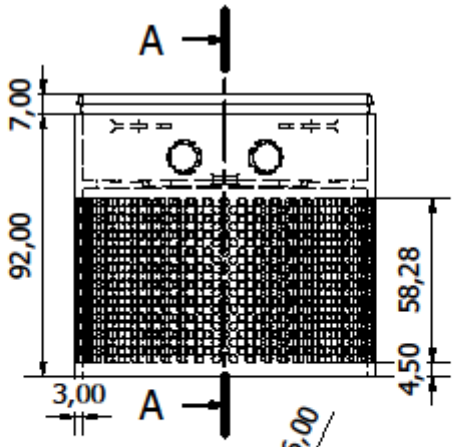




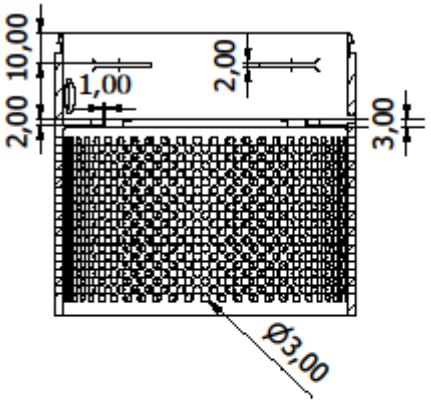


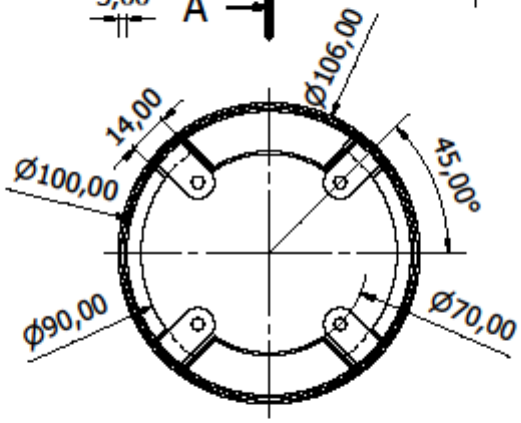
				TOLERANCIAS:	PESO: 37 g	MATERIALES: ABS Plastic	
					FECHA	NOMBRE	
				DIB.	26/1/2022	Guido Ramírez	DENOMINACIÓN: Soporte para filtro HEPA — ESCALA: 1:2
				REV.			
				APROB.			
EDICIÓN	MODIFICACIÓN	FECHA	NOMBRE			NÚMERO DEL DIBUJO: 3	
						SUSTITUYE A:	





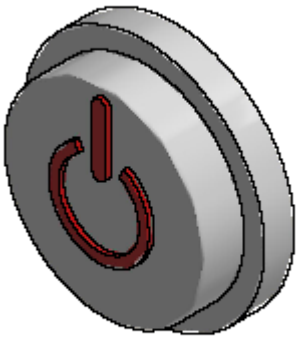
A-A (1 : 2)

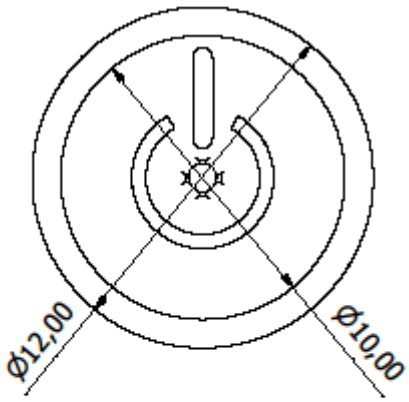


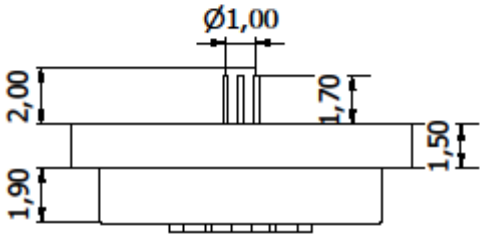


				TOLERANCIAS		PESO	MATERIALES
						79 g	ABS plastic
					FECHA	NOMBRE	DENOMINACIÓN
				DIB.	26/1/2022	Guido Ramírez	Entrada del purificador
				REV.			
				APROB.			
							NÚMERO DEL DIBUJO
							4
EDICIÓN	MODIFICACIÓN	FECHA	NOMBRE				SUSTITUYE A:

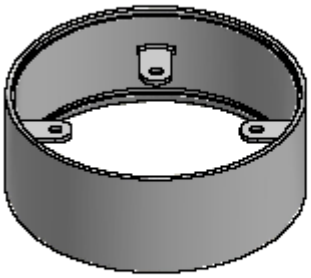
ESCALA: 1:2



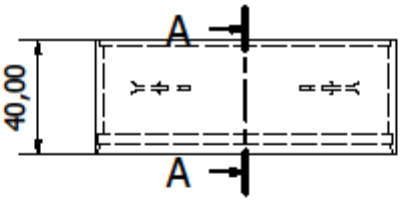


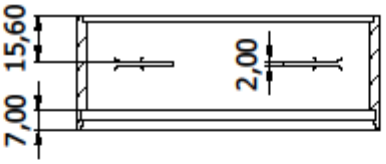


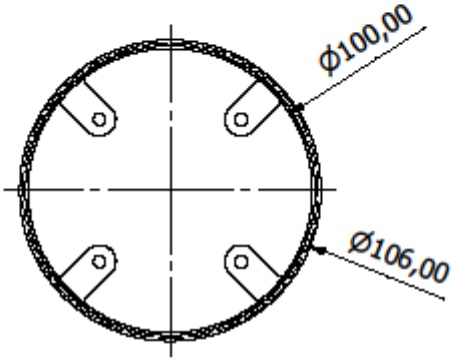
				TOLERANCIAS:		PESO	MATERIALES	
						N/A	ABS Plastic	
					FECHA	NOMBRE	DENOMINACIÓN	ESCALA
				DIB.	26/1/2022	Guido Ramírez	Botón	5:1
				REV.				
				APROB.				
							NÚMERO DEL DIBUJO	
EDICIÓN	MODIFICACIÓN	FECHA	NOMBRE				5	
							SUSTITUYE A:	



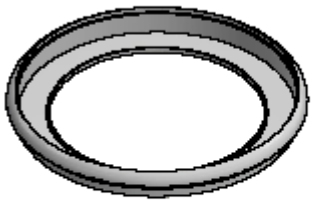
A-A (1 : 2)

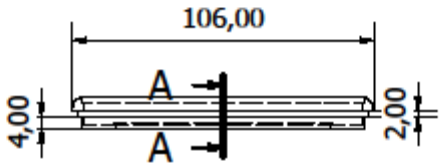


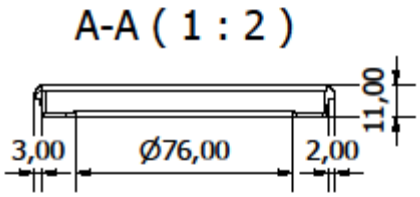


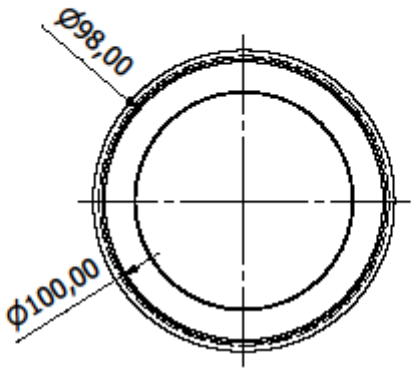


				TOLERANCIAS		PESO	MATERIALES		
						38 g	ABS plastic		
					FECHA	NOMBRE	DENOMINACIÓN		ESCALA
				DIB.	26/1/2022	Guido Ramírez	Salida del purificador		1:2
				REV.					
				APROB.					
							NÚMERO DEL DIBUJO	6	
EDICIÓN	MODIFICACIÓN	FECHA	NOMBRE				SUSTITUYE A:		



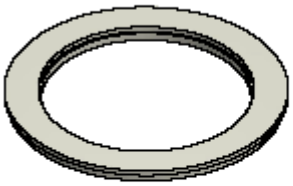


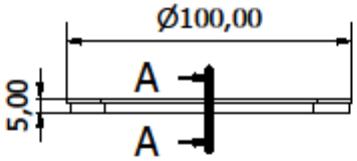


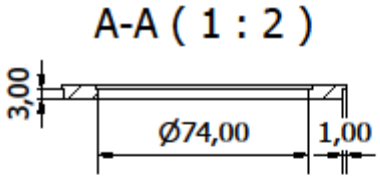


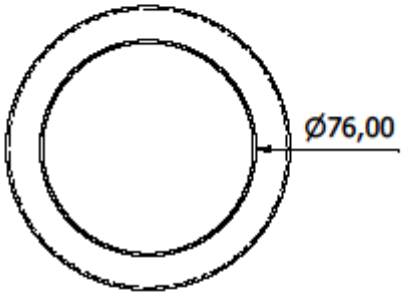
				TOLERANCIAS		PESO	MATERIALES
						11 g	ABS Plastic
					FECHA	NOMBRE	DENOMINACIÓN
				DIB.	26/1/2022	Guido Ramírez	Soporte leds de salida.
				REV.			
				APROB.			NÚMERO DEL DIBUJO
							7
EDICIÓN	MODIFICACIÓN	FECHA	NOMBRE				SUSTITUYE A:

ESCALA
1:2

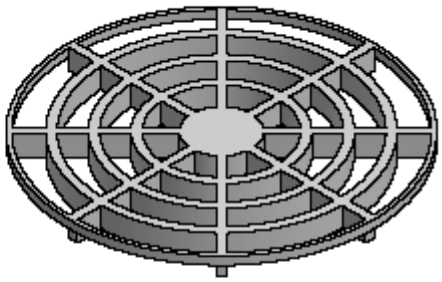


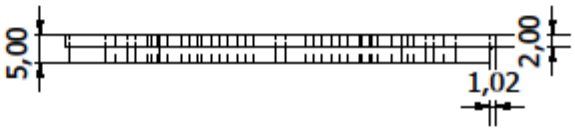


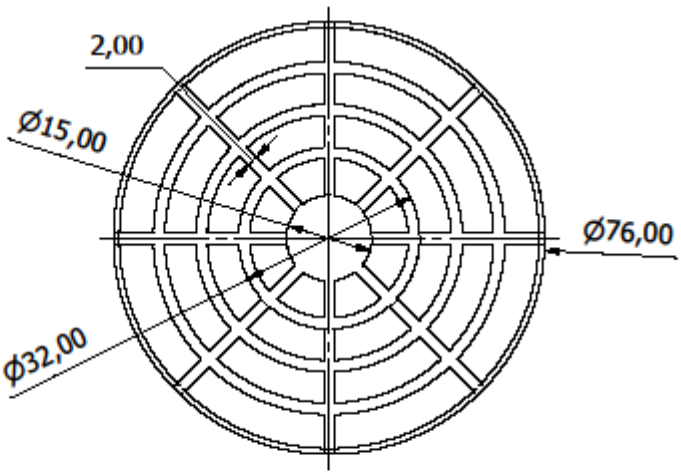




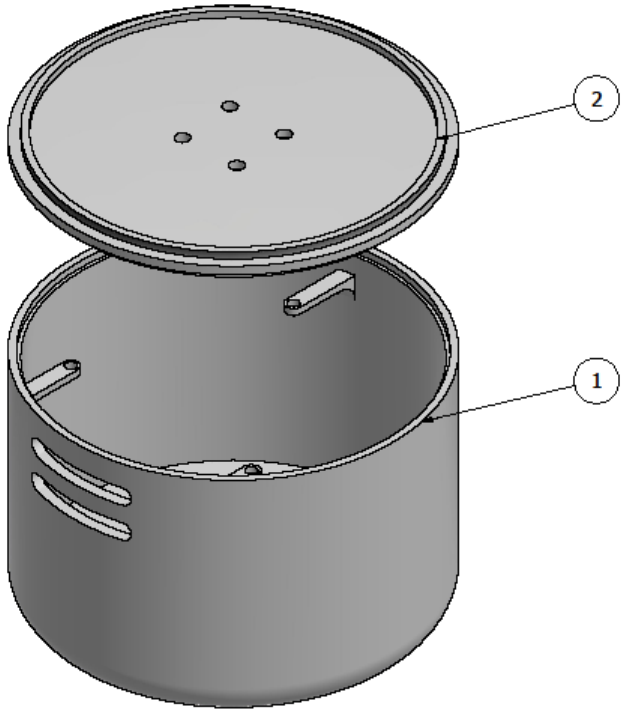
				TOLERANCIAS		PESO	MATERIALES	
						6 g	RC	
					FECHA	NOMBRE	DENOMINACIÓN	ESCALA
				DIB.	26/1/2022	Guido Ramírez	Cobertor led	1:2
				REV.				
				APROB.				
							NÚMERO DEL DIBUJO	
							8	
EDICIÓN	MODIFICACIÓN	FECHA	NOMBRE				SUSTITUYE A:	







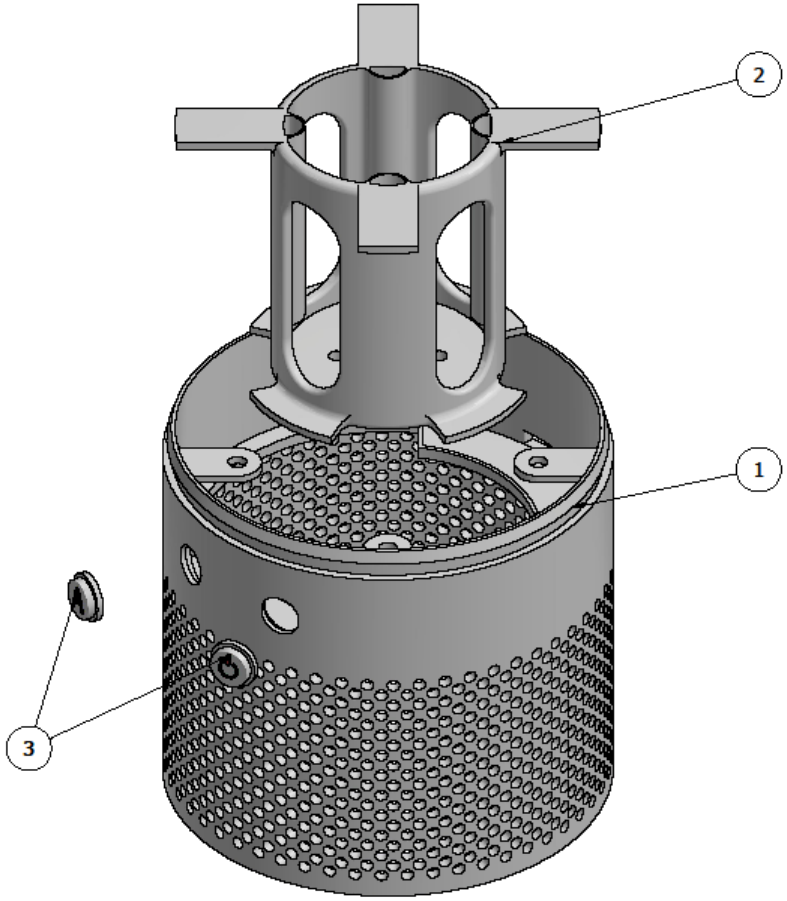
				TOLERANCIAS		PESO	MATERIALES
						8 g	ABS Plastic
					FECHA	NOMBRE	DENOMINACIÓN / ESCALA
				DIB.	26/01/2022	Guido Ramírez	Malla de salida / 1:1
				REV.			
				APROB.			NÚMERO DEL DIBUJO: 9
							SUSTITUYE A:
ESCIÓN	MODIFICACIÓN	FECHA	NOMBRE				



PARTS LIST			
ITEM	QTY	PART NUMBER	DESCRIPTION
1	1	Base	Plástico ABS
2	1	Tapa Base	Plástico ABS

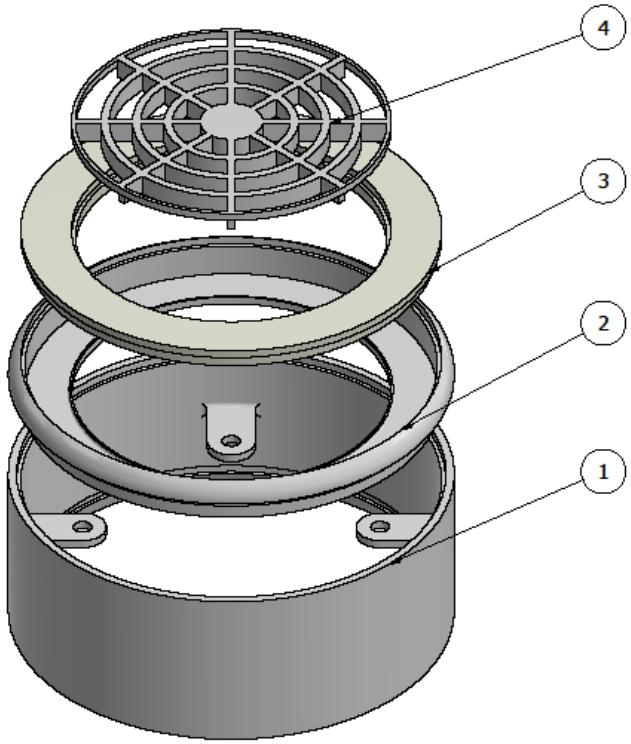
				TOLERANCIAS		PESO N/A	MATERIALES
					FECHA	NOMBRE	DENOMINACIÓN: Base removible
				DIB.	28/1/2022	Guido Ramírez	
				REV.			
				APROB.			
							NÚMERO DEL DIBUJO: 10
ESCRIPCIÓN	MODIFICACIÓN	FECHA	NOMBRE				SUSTITUYE A:

ESCALA: 1:1



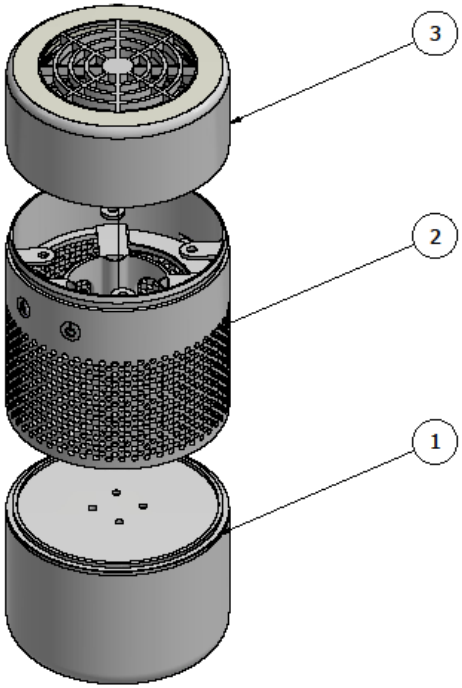
PARTS LIST			
ITEM	QTY	PART NUMBER	DESCRIPTION
1	1	Cuerpo	Plástico ABS
2	1	Soporte filtro	Plástico ABS
3	2	Botón	Plástico ABS

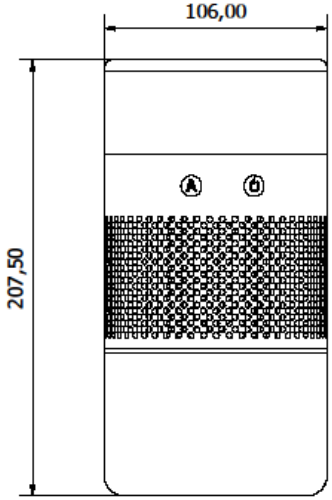
TOLERANCIAS	PESO: N/A	MATERIALES	
DIB. 28/1/2022 Guido Ramírez	DENOMINACIÓN: Entrada de aire	ESCALA: 1:1	
REV.			
APROB.			
EDICIÓN / MODIFICACIÓN / FECHA / NOMBRE	NÚMERO DEL DIBUJO: 11		
	SUSTITUYE A:		



PARTS LIST			
ITEM	QTY	PART NUMBER	DESCRIPTION
1	1	Salida	Plastico ABS
2	1	Soporte leds	Plastico ABS
3	1	Anillo	Policarbonato
4	1	Malla salida	Plastico ABS

TOLERANCIAS	PESO: N/A	MATERIALES	
	FECHA	NOMBRE	DENOMINACIÓN: Salida de aire purificado
DIB.	28/1/2022	Guido Ramírez	ESCALA: 1:1
REV.			NÚMERO DEL DIBUJO: 12
APROB.			SUSTITUYE A:
EDICIÓN	MODIFICACIÓN	FECHA	NOMBRE





PARTS LIST			
ITEM	QTY	PART NUMBER	DESCRIPTION
1	1	Base removible	
2	1	Entrada de aire	
3	1	Salida de aire	

				TOLERANCIAS	PESO	MATERIALES	
					294 g		
					FECHA	NOMBRE	
				DIB.	28/1/2022	Guido Ramírez	
				REV.			
				APROB.			
EDICIÓN	MODIFICACIÓN	FECHA	NOMBRE				

DENOMINACIÓN	Purificador	ESCALA 1:2
NÚMERO DEL DIBUJO	13	
SUSTITUYE A:		

APÉNDICE B

Dimensiones físicas de componentes electrónicos

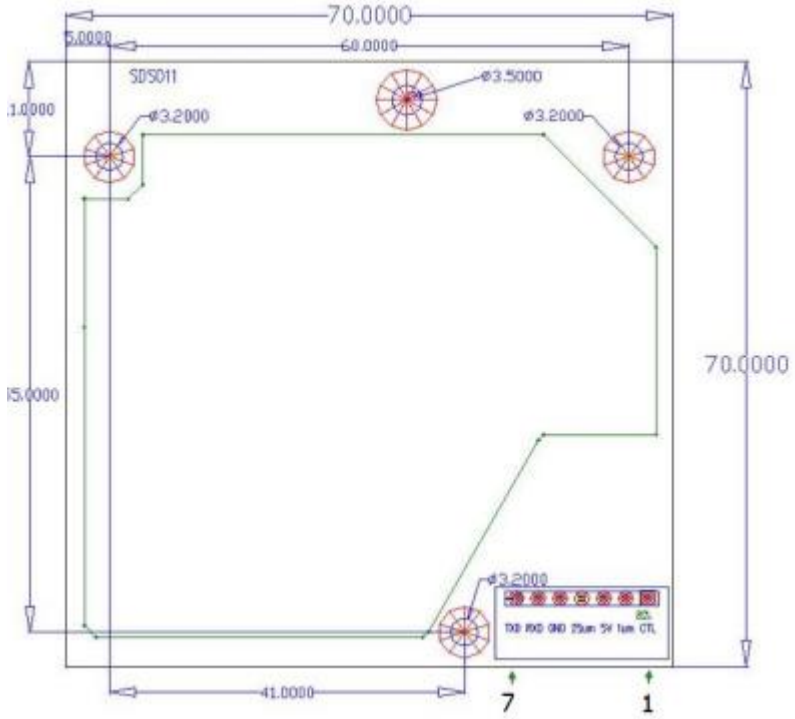


Figura B.1 Vista frontal, dimensiones del sensor de PM, SDS011

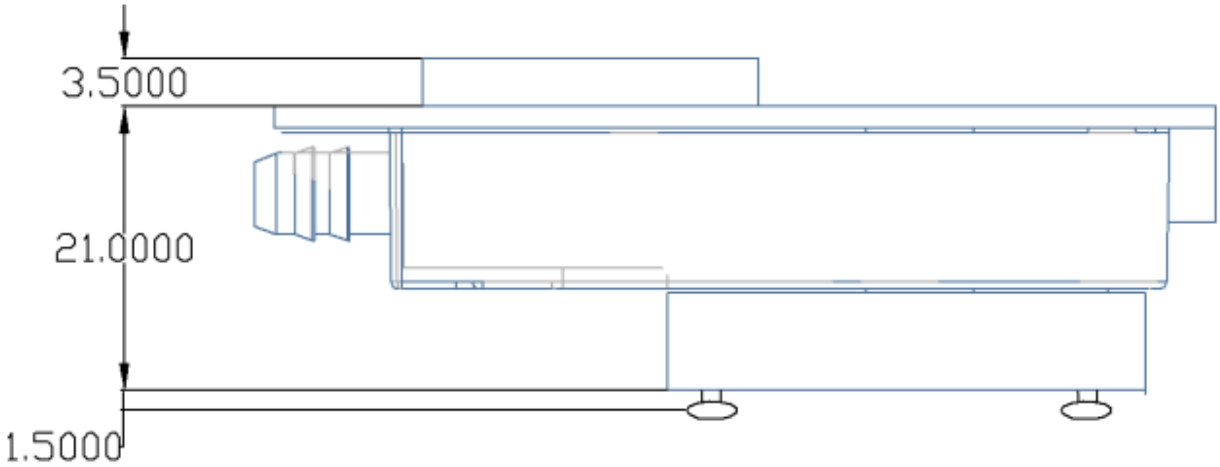


Figura B.2 Vista lateral, dimensiones del sensor de PM, SDS011

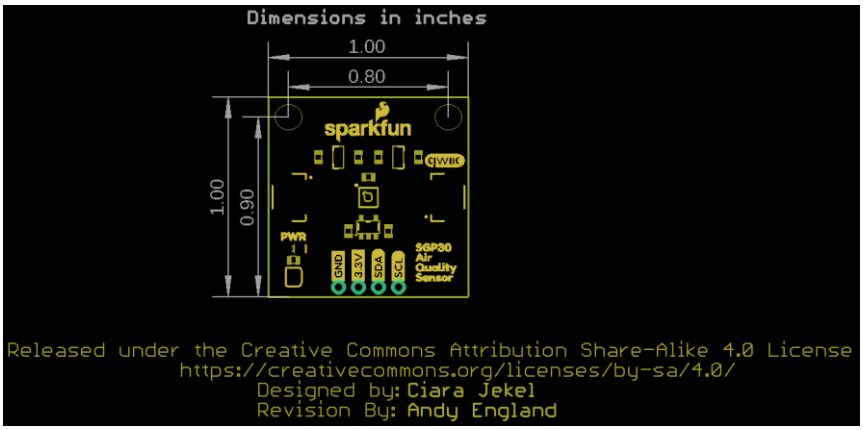


Figura B.3 Vista frontal, dimensiones del sensor de tVOC y eCO2, SGP30 en la PCB de sparkfun electronics [40]

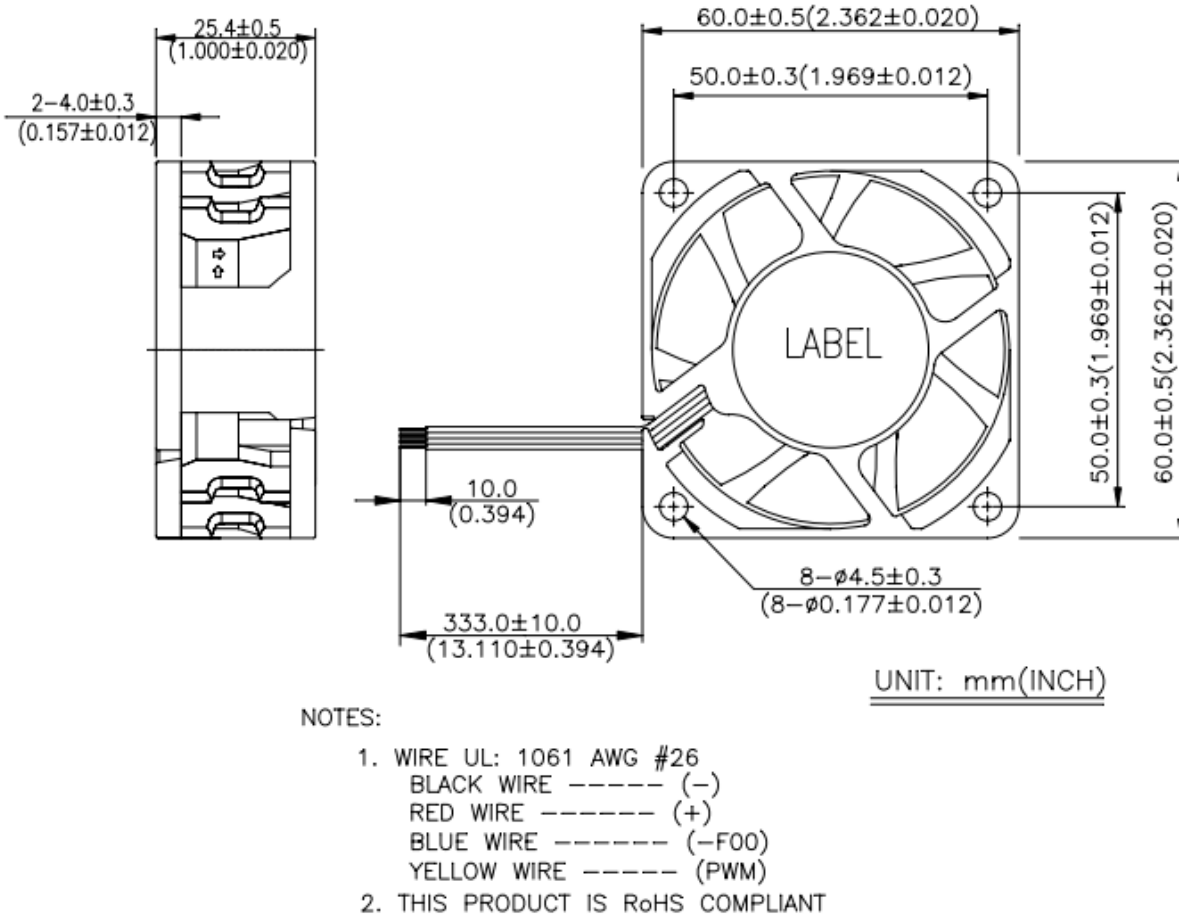


Figura B.4 Dimensiones de ventilador axial seleccionado del distribuidor *Digikey*

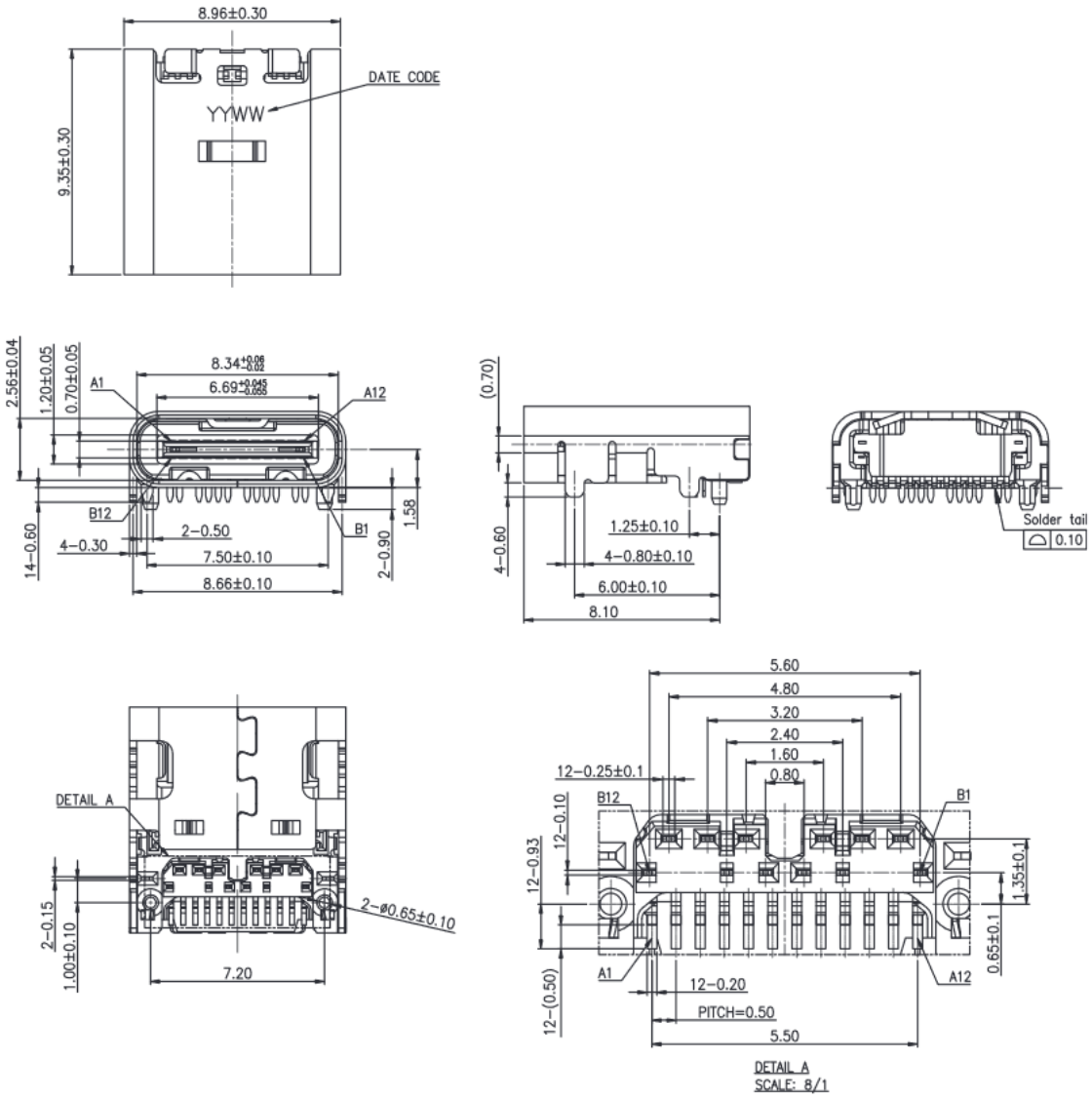


Figura B.5 Dimensiones puerto de carga USB de *cuidevices*

APÉNDICE C

Protocolo de comunicación del sensor SDS011

The UART communication protocol

Bit rate ： 9600

Data bit ： 8

Parity bit： NO

Stop bit ： 1

Data Packet frequency: 1Hz

The number of bytes	Name	Content
0	Message header	AA
1	Commander No.	C0
2	DATA 1	PM2.5 Low byte
3	DATA 2	PM2.5 High byte
4	DATA 3	PM10 Low byte
5	DATA 4	PM10 High byte
6	DATA 5	ID byte 1
7	DATA 6	ID byte 2
8	Check-sum	Check-sum
9	Message tail	AB

Check-sum: Check-sum=DATA1+DATA2+...+DATA6 。

Figura C.1 Configuración del sensor de PM, SDS011 y significado de cada byte de la trama [34]

APÉNDICE D

Pesos y bias de las neuronas

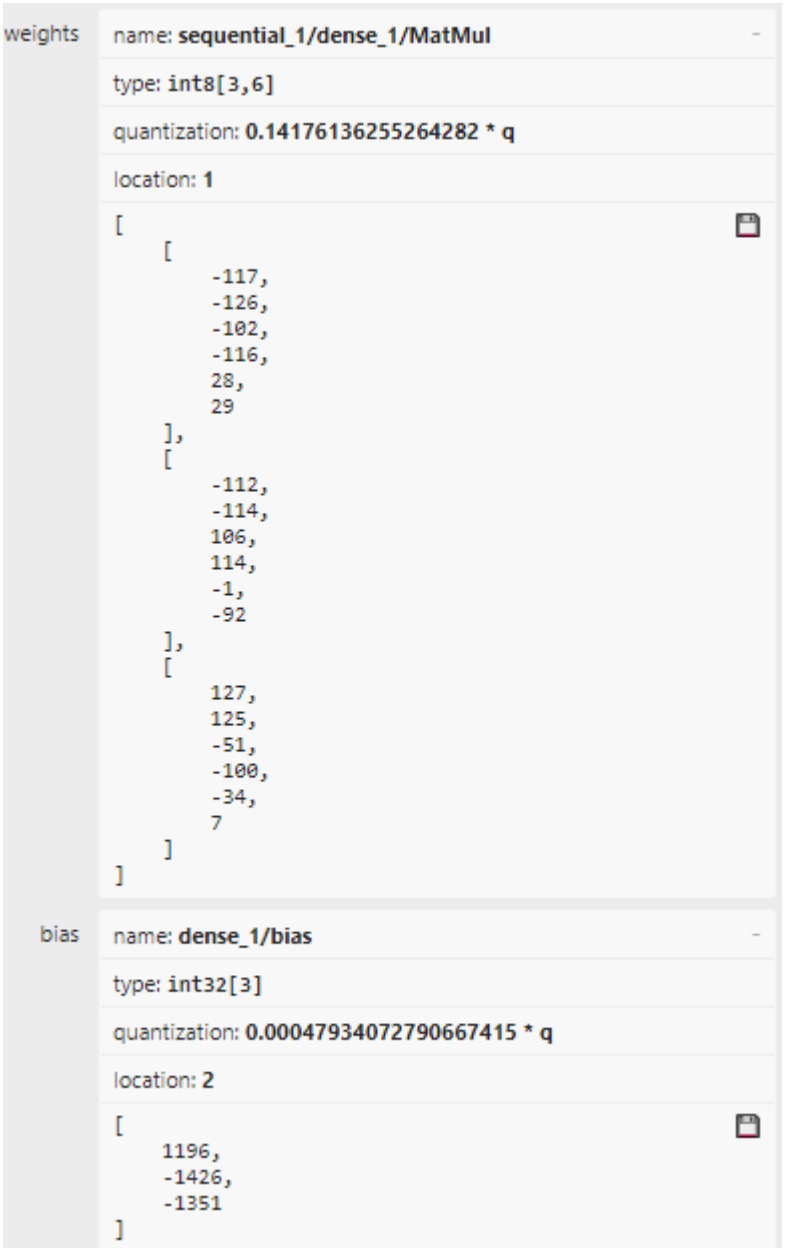


Figura D.1 Pesos y bias de cada neurona creada para el regresor logístico multiclase

APÉNDICE E

Codificación de la función de retorno y sumidero

```c
static void ezm_uart_callback_events(void *ctx, perUartEvents ev){
  (void )&ctx;
  switch(ev){
    case perUartReadSuccess:
      app_log_info("Read Success\n");
      break;
    case perUartReadFail:
      app_log_info("Read Fail\n");
      break;
    case perUartWriteSucces:
      app_log_info("Write Success\n");
      break;
    case perUartWriteFail:
      app_log_info("Write Fail\n");
      break;
    default:
      break;
  }
}
```

Figura E.1 Implementación de la función sumidero en UART

```c
static void ezm_uart_ao_callback(void *ao,Event *e){
  static Event event = {0};
  int err = 0;
  uint16_t pm2_5 = 0;
  uint16_t pm10 = 0;
  switch(e->sig){
    case INIT_SIG:
      app_log_info("Active Object Started\n");
      event.sig = perUartRead;
      ezm_ao_post_event_queue(ao, &event);
      break;
    case perUartRead:
      app_log_info("Active Object Read\n");
      err = app_sds011_read(&ezmUart0,&pm2_5,&pm10,&sds011.rawBufferSize);
      if(err == perSensorErrOk){
          indoorAirQuality.pm2_5 = pm2_5;
          indoorAirQuality.pm10_0= pm10;
          app_log_info("PM 2.5: %.1f, PM10: %.1f\n",(float)(pm2_5/10.0),(float)(pm10/10.0));
      }
      vTaskDelay(pdMS_TO_TICKS(1000));
      ezm_ao_post_event_queue(ao, &event);
      break;
    default:
      break;
  }
}
```

Figura E.2 Implementación de la función de retorno en UART

```c
static void ezm_i2c_callback_events(void *ctx, perI2cEvents ev){
  (void )&ctx;
  switch(ev){
    case perI2cReadSuccess:
      app_log_info("Read Success\n");
      break;
    case perI2ctReadFail:
      app_log_info("Read Fail\n");
      break;
    case perI2ctWriteSucces:
      app_log_info("Write Success\n");
      break;
    case perI2ctWriteFail:
      app_log_info("Write Fail\n");
      break;
    default:
      break;
  }
}
```

Figura E.3 Implementación de la función sumidero en I^2C

```
static void ezm_i2c_ao_callback(void *ao,Event *e){
  static Event event = {0};  int err = 0;
  int32_t temperature = 0;  uint32_t humidity = 0; uint16_t tVOC_Baseline = 0x879D;  uint16_t eCO2_Baseline = 0x8785;
  switch(e->sig){
    case INIT_SIG:
      app_log_info("Active Object i2c Started\n");
      event.sig = perI2cSerialID;     ezm_ao_post_event_queue(ao, &event);
      app_si70xx_init();
      break;
    case perI2cSerialID:
      app_log_info("Active Object: Get Serial ID\n");
      app_sgp30_get_serial_id(&ezmI2C1,&sgp30Instance);
      event.sig = perI2cFeatureSet;   ezm_ao_post_event_queue(ao, &event);
      break;
    case perI2cFeatureSet:
      app_log_info("Active Object: Get Feature Set\n");
      app_sgp30_get_feature_set_version(&ezmI2C1,&sgp30Instance);
      event.sig = perI2cIAQMonitoring;ezm_ao_post_event_queue(ao, &event);
      break;
    case perI2cIAQMonitoring:
      app_log_info("Active Object: Start IAQ Monitoring\n");
      app_sgp30_init_IAQ(&ezmI2C1,&sgp30Instance);
      event.sig = perI2cWarmUpTime;   ezm_ao_post_event_queue(ao, &event);
      break;
    case perI2cWarmUpTime:
      app_log_info("Waiting for 15 seconds warm up...\n");
      for(int i = 0; i< 15; i++){
        app_log_info("Reading zero values before start...\n");
        app_sgp30_measurement_IAQ(&ezmI2C1,&sgp30Instance);
        vTaskDelay(pdMS_TO_TICKS(1000));
      }
      event.sig = perI2cSetBaseline; ezm_ao_post_event_queue(ao, &event);
      break;
    case perI2cSetBaseline:
      app_log_info("Active Object: Set Baseline\n");
      app_sgp30_set_baseline(&ezmI2C1, &sgp30Instance, tVOC_Baseline, eCO2_Baseline);
      event.sig = perI2cGetBaseline; ezm_ao_post_event_queue(ao, &event);
      break;
    case perI2cMeasureIAQ:
      app_log_info("Active Object: Read from sensors\n");
      err = app_sgp30_measurement_IAQ(&ezmI2C1,&sgp30Instance);
      if(err == perSensorErrOk){
          app_si70xx_measure(&humidity, &temperature);
          indoorAirQuality.tVOC = sgp30Instance.tVOC;
          indoorAirQuality.eCO2 = sgp30Instance.eCO2;
          indoorAirQuality.temperature = temperature;
          indoorAirQuality.humidity = humidity;
          app_log_info("tVOC: %d, eCO2: %d, Temp: %.2f, Hum: %.2f\n",
                       sgp30Instance.tVOC,sgp30Instance.eCO2,(float)(temperature/1000.0),(float)(humidity/1000.0));
      }
      vTaskDelay(pdMS_TO_TICKS(1000));
      event.sig = perI2cMeasureIAQ; ezm_ao_post_event_queue(ao, &event);
      break;
    case perI2cGetBaseline:
      app_log_info("Active Object: Get Baseline\n");
      app_sgp30_get_baseline(&ezmI2C1, &sgp30Instance);
      event.sig = perI2cMeasureIAQ; ezm_ao_post_event_queue(ao, &event);
      break;
  }
}
```

Figura E.4 Implementación de la función de retorno en I²C

```
static void ezm_ia_callback_events(void *ctx,perIaEvents ev){
  (void)&ctx;
  switch(ev){
    case perIaPredictOk:
      app_log_info("Predict: OK\n");
      break;
    case perIaPredictFail:
      app_log_info("Predict: FAIL\n");
      break;
  }
}
```

Figura E.5 Implementación de la función sumidero en IA

```c
static void ezm_ia_ao_callback(void *ao, Event *e){
  static Event event = {0};
  int err = 0;
  switch(e->sig){
    case INIT_SIG:
      app_log_info("Init Signal Machine Learning\n");
      event.sig = perIaPredict;
      ezm_ao_post_event_queue(ao,&event);

      break;
    case perIaPredict:
      while(indoorAirQuality.eCO2 == 0){
          app_log_info("Calibrating...\n");
          calibrationSequence();
      }
      err = app_ia_predict();
      if(err >= CLASS_0){
          switch(err){
            case CLASS_0:
              app_log_info("Encender LED Azul: Ambiente Normal\n");
              GPIO_PinModeSet(portLEDNormal, pinLEDNormal, gpioModePushPull, 1);
              break;
            case CLASS_1:
              app_log_info("Encender LED Amarillo: Agente Químico\n");
              GPIO_PinModeSet(portLEDAQ, pinLEDAQ, gpioModePushPull, 1);
              break;
            case CLASS_2:
              app_log_info("Encender LED Rojo: Combustión\n");
              GPIO_PinModeSet(portLEDComb, pinLEDComb, gpioModePushPull, 1);
              break;
            default:
              break;
          }
      }
      vTaskDelay(pdMS_TO_TICKS(500));
      turnOffLeds();
      vTaskDelay(pdMS_TO_TICKS(500));
      event.sig=perIaPredict;
      ezm_ao_post_event_queue(ao, &event);
      break;
    default:
      break;
  }
}
```

Figura E.6 Implementación de la función de retorno en IA

APÉNDICE F

Documentación en inglés del *firmware* generado

Function Documentation

◆ app_sds011_check_crc()

```
int app_sds011_check_crc ( uint8_t *  buffer,
                           uint16_t  len
                         )
```

This functions verifies the integrity of data by SDS011's CRC.

Parameters

buffer Pointer to buffer received by the sensor.

len Len of the buffer.

Returns

int Ok = 0 Err < 0

◆ app_sds011_init()

```
int app_sds011_init ( perSysTree *  per )
```

This function initializes SDS011 library.

Parameters

per Pointer to main struct with principal components.

Returns

int Ok = 0 Err < 0

◆ app_sds011_read()

```
int app_sds011_read ( const instanceUART *  instance,
                      uint16_t *            pm2_5,
                      uint16_t *            pm10,
                      uint8_t *             len
                    )
```

This function reads SDS011 data.

Parameters

instance Pointer to uartInstance subscribed in UART's column.

pm2_5 Pointer to store PM 2.5 value.

pm10 Pointer to store PM 10 value.

len Len of expected data.

Returns

int Ok = 0 Err < 0

Figura F.1 Detalle de las funciones públicas del sensor SDS011.

Function Documentation

◆ app_sgp30_get_baseline()

```
int app_sgp30_get_baseline ( const instanceI2C * instance,
                             sgp30_t *          sensorInstance
                           )
```

This function obtains the baseline of SGP30 sensor.

Parameters

instance	Pointer to instance subscribed in I2C's peripheral.
sensorInstance	Pointer to sensorInstance.

Returns

int Ok = 0 Err < 0

◆ app_sgp30_get_feature_set_version()

```
int app_sgp30_get_feature_set_version ( const instanceI2C * instance,
                                        sgp30_t *          sensorInstance
                                      )
```

This function obtain SGP30's Manufacturer feature set.

Parameters

instance	Pointer to instance subscribed in I2C's peripheral.
sensorInstance	Pointer to sensorInstance.

Returns

int Ok = 0 Err < 0

◆ app_sgp30_get_serial_id()

```
int app_sgp30_get_serial_id ( const instanceI2C * instance,
                              sgp30_t *          sensorInstance
                            )
```

This functions obtain SGP30's Manufacturer ID.

Parameters

instance	Pointer to instance subscribed in I2C's peripheral.
sensorInstance	Pointer to sensorInstance

Returns

int Ok = 0 Err < 0

Figura F.2 Parte 1: Detalle de las funciones públicas del sensor SGP30.

◆ app_sgp30_init()

int app_sgp30_init (**perSysTree** * per)

This function initializes SGP30 library.

Parameters

per Pointer to main struct with principal components.

Returns

int Ok = 0 Err < 0

◆ app_sgp30_send_command()

```
int app_sgp30_send_command ( const instanceI2C *  instance,
                             sgp30_t *           sensorInstance,
                             TickType_t          delay,
                             uint8_t *           writeBuffer,
                             uint8_t             lenWriteBuffer,
                             uint8_t *           readBuffer,
                             uint8_t             lenReadBuffer
                            )
```

This function sends a command directly to SGP30 sensor.

Parameters

instance	Pointer to instance subscribed in I2C's peripherals.
sensorInstance	Pointer to sensorInstance.
delay	Delay time for reading sensor.
writeBuffer	Pointer to write buffer.
lenWriteBuffer	Len of writeBuffer.
readBuffer	Pointer to read buffer.
lenReadBuffer	Len of readBuffer.

Returns

int Ok = 0 Err < 0

◆ app_sgp30_get_crc()

int app_sgp30_get_crc (uint8_t * data)

This function calculates the CRC of the data to be sent or data recieved.

Parameters

data Pointer to data to be used to calculate CRC.

Returns

int CRC calculated.

Figura F.3 Parte 2: Detalle de las funciones públicas del sensor SGP30.

◆ app_sgp30_measurement_IAQ()

```
int app_sgp30_measurement_IAQ ( const instanceI2C *  instance,
                                sgp30_t *            sensorInstance
                              )
```

This function measures tVOC and eCO2.

Parameters

instance	Pointer to instance subscribed in I2C's peripheral.
sensorInstance	Pointer to sensorInstance.

Returns

int Ok = 0 Err < 0

◆ app_sgp30_init_IAQ()

```
int app_sgp30_init_IAQ ( const instanceI2C *  instance,
                         sgp30_t *            sensorInstance
                       )
```

This function initializes Indoor Air Quality measurements.

Parameters

instance	Pointer to instance subscribed in I2C's peripheral.
sensorInstance	Pointer to sensorInstance.

Returns

int Ok = 0 Err < 0

◆ app_sgp30_verify_crc()

```
int app_sgp30_verify_crc ( uint8_t *  readBuffer,
                           uint8_t    lenReadBuffer
                         )
```

This function verify sent/recieved data to SGP30 sensor.

Parameters

readBuffer	
lenReadBuffer	

Returns

int Ok = 0 Err < 0

Figura F.4 Parte 3: Detalle de las funciones públicas del sensor SGP30.

◆ app_sgp30_set_baseline()

```
int app_sgp30_set_baseline ( const instanceI2C *  instance,
                             sgp30_t *           sensorInstance,
                             uint16_t            tVOCBaseline,
                             uint16_t            eCO2Baseline
                           )
```

This function set the baseline for SGP30 sensor measurements.

Parameters

instance	Pointer to instance subscribed in I2C's peripheral.
sensorInstance	Pointer to sensorInstance.
tVOCBaseline	tVOC baseline.
eCO2Baseline	eCO2 baseline.

Returns

int Ok = 0 Err < 0

◆ app_si70xx_init()

```
int app_si70xx_init ( )
```

This function initializes Si7021 sensor library.

Returns

int Ok = SL_STATUS_INITIALIZATION Err != SL_STATUS_INITIALIZATION

◆ app_si70xx_measure()

```
int app_si70xx_measure ( uint32_t *  hum,
                         int32_t *   temp
                       )
```

This function measure temperature and humidity from Si7021 sensor.

Parameters

hum	Pointer to save humidity value.
temp	Pointer to save temperature value.

Returns

int Ok = SL_STATUS_OK Err != SL_STATUS_OK

Figura F.5 Parte 4: Detalle de las funciones públicas del sensor SGP30 y Si7021.

Function Documentation

◆ app_ia_init()

int app_ia_init (**perSysTree** * per)

This function initializes IA library.

Parameters

per Pointer to main struct with principal components.

Returns

int Ok = 0 Err < 0

Figura F.6 Detalle de la función pública del modelo de Inteligencia Artificial.

Function Documentation

◆ ezm_uart_ao_attach_cb()

```
int ezm_uart_ao_attach_cb ( const void *  uartInstance,
                            void *        callback
                          )
```

This API attach the callback function to uartInstance & active object.

Parameters

uartInstance Pointer to uartInstance subscribed in UART's peripheral.

callback Callback function that receive events/signals generated.

Returns

int Ok = 0 Err < 0

◆ ezm_uart_ao_init()

```
int ezm_uart_ao_init ( const void *  uartInstance,
                       void *        activeObject,
                       const char *  taskName,
                       uint16_t      stack,
                       uint8_t       priority,
                       uint16_t      xQueueSize,
                       uint8_t       xQueueItems
                     )
```

This API attach the active object to uartInstance and initialize it.

Parameters

uartInstance Pointer to uartIntance subscribed in UART's peripheral.

activeObject Pointer to active object struct to be attached in UART's peripheral.

taskName Task's name for active object's RTOS task.

stack Stack for active object's RTOS task.

priority Priority for active object's RTOS task.

xQueueSize Queue size for active object's RTOS task.

xQueueItems Queue items for active object's RTOS task.

Returns

int Ok = 0 Err < 0

◆ ezm_uart_attach()

```
int ezm_uart_attach ( void *  peripheralUart )
```

This API attach UART's peripheral and initialize its values to zero and pointers to NULL.

Parameters

peripheralUart Main structure of UART's peripheral.

Returns

int Ok = 0 Err < 0

Figura F.7 Parte 1: Documentación del componente UART

◆ ezm_uart_init()

```
int ezm_uart_init ( const void *  uartInstance )
```

This API initializes UART communication and install UART drivers of Silicon Labs.

Parameters

uartInstance Pointer to uartInstance subscribed in UART's peripheral.

Returns

int Ok = 0 Err < 0

◆ ezm_uart_rxB()

```
int ezm_uart_rxB ( const void *  uartInstance,
                   uint8_t *     buffer,
                   uint16_t      lenBuffer
                 )
```

This API reads any data from RX line.

Parameters

uartInstance Pointer to uartInstance subscribed in UART's peripheral.

buffer Pointer to buffer where it is going to be saved data.

lenBuffer Len of the expected data.

Returns

int Ok = 0 Err < 0

◆ ezm_uart_set_config()

```
int ezm_uart_set_config ( void *  uartDrv,
                          uint8_t port,
                          int     baudRate,
                          int     stopBits,
                          int     parity,
                          int     oversampling,
                          int     flowControl,
                          void *  rxQueue,
                          void *  txQueue
                        )
```

This API sets UART configuration in pointer uartDrv.

Parameters

uartDrv Pointer to uartDrv.

port UART's port number.

baudRate Baud rate.

stopBits Number of stop bits.

parity Parity configuration.

oversampling Oversampling mode.

flowControl Flow control mode.

rxQueue Receive operation queue.

txQueue Transmit operation queue.

Returns

int Ok = 0 Err < 0

Figura F.8 Parte 2: Documentación del componente UART

◆ ezm_uart_set_gpio_config()

```
int ezm_uart_set_gpio_config ( void *   peripherals,
                               void *   uartDrv,
                               uint8_t  gpioPortTx,
                               uint8_t  gpioPinTx,
                               uint8_t  gpioPortRx,
                               uint8_t  gpioPinRx,
                               uint8_t  gpioPortCts,
                               int      gpioPinCts,
                               uint8_t  gpioPortRts,
                               int      gpioPinRts
                             )
```

This API set the UART's GPIO to be used in pointer uartDrv instance.

Parameters

- **peripherals** Pointer to main struct with principal components.
- **uartDrv** Pointer to uartDrv instance.
- **gpioPortTx** Port associated with Tx Port.
- **gpioPinTx** Pin associated with Tx Pin.
- **gpioPortRx** Port associated with Rx Port.
- **gpioPinRx** Pin associated with Rx Pin.
- **gpioPortCts** Port associated with Cts Port.
- **gpioPinCts** Pin associated with Cts Pin (If FlowControl is disabled, -1 must be used as parameter).
- **gpioPortRts** Port associated with Rts Port.
- **gpioPinRts** Pin associated with Rts Pin (If FlowControl is disabled, -1 must be used as parameter).

Returns

int Ok = 0 Err < 0

◆ ezm_uart_subscribe()

```
int ezm_uart_subscribe ( const void *  uartInstance,
                         void *        uartDrv,
                         void *        uartCallback
                       )
```

This API subscribe in UART's peripheral the instance, uartDrv and callback function.

Parameters

- **uartInstance** Pointer to uartInstance subscribed in UART's peripheral.
- **uartDrv** Pointer to uartDrv instance.
- **uartCallback** Pointer to function of perEventCallback prototype.

Returns

int Ok = 0 Err < 0

◆ ezm_uart_unsubscribe()

```
int ezm_uart_unsubscribe ( const void *  uartInstance,
                           void *        peripherals
                         )
```

This API delete all the objects reference and variables subscribed with uartInstance.

Parameters

- **uartInstance** Pointer to uartInstance subscribed in UART's peripheral.
- **peripherals** Pointer to main struct with principal components.

Returns

int Ok = 0 Err < 0

Figura F.9 Parte 3: Documentación del componente UART

Function Documentation

◆ ezm_i2c_ao_attach_cb()

```
int ezm_i2c_ao_attach_cb ( const void *  i2cInstance,
                           void *        callback
                         )
```

This API attach the callback function to i2cInstance & active object.

Parameters

i2cInstance Pointer to i2cInstance subscribed in I2C's peripheral.

callback Callback function that receive events/signals generated.

Returns

int Ok = 0 Err < 0

◆ ezm_i2c_ao_init()

```
int ezm_i2c_ao_init ( const void *  i2cInstance,
                      void *        activeObject,
                      const char *  taskName,
                      uint16_t      stack,
                      uint8_t       priority,
                      uint16_t      xQueueSize,
                      uint8_t       xQueueItems
                    )
```

This API attach the active object to i2cInstance and initialize it.

Parameters

i2cInstance Pointer to i2cInstance subscribed in I2C's peripheral.

activeObject Pointer to active object struct to be attached in I2C's peripheral.

taskName Task's name for active object's RTOS task.

stack Stack for active object's RTOS task.

priority Priority for active object's RTOS task.

xQueueSize Queue size for active object's RTOS task.

xQueueItems Queue items for active object's RTOS task.

Returns

int Ok = 0 Err < 0

◆ ezm_i2c_attach()

```
int ezm_i2c_attach ( void *  peripheralI2c )
```

This API attach I2C's peripheral and initialize its values to zero and pointers to NULL.

Parameters

peripheralI2c Main structure of I2C's peripheral.

Returns

int Ok = 0 Err < 0

◆ ezm_i2c_init()

```
int ezm_i2c_init ( const void *  i2cInstance )
```

This API initializes I2C communication and install I2C drivers of Silicon Labs.

Parameters

i2cInstance Pointer to i2cInstance subscribed in I2C's peripheral.

Returns

int Ok = 0 Err < 0

Figura F.10 Parte 1: Documentación del componente I²C

◆ ezm_i2c_read()

```
int ezm_i2c_read ( const void *  i2cInstance,
                   uint16_t      slaveAddr,
                   uint8_t       targetAddr,
                   uint8_t *     rxBuffer,
                   uint8_t       lenRxBuffer
                 )
```

This API reads from the slave the data stored in rxBuffer.

Parameters

i2cInstance	Pointer to i2cInstance subscribed in I2C's peripheral.
slaveAddr	I2C Slave's address.
targetAddr	Must be zero.
rxBuffer	Pointer to an rxBuffer with len of lenRxBuffer.
lenRxBuffer	Len of expected data.

Returns

int Ok = 0 Err < 0

◆ ezm_i2c_set_config()

```
int ezm_i2c_set_config ( void *    i2cDrv,
                         uint8_t   port,
                         uint32_t  i2cRefFreq,
                         uint32_t  i2cMaxFreq,
                         uint8_t   i2cClhr
                       )
```

This API sets I2C configuration in pointer uartDrv and checks its integrity.

Parameters

i2cDrv	Pointer to i2cDrv.
port	I2C's port number.
i2cRefFreq	I2C Reference Frequency.
i2cMaxFreq	I2C Maximum Frequency.
i2cClhr	I2C Clock HLR.

Returns

int Ok = 0 Err < 0

◆ ezm_i2c_set_gpio_config()

```
int ezm_i2c_set_gpio_config ( void *   peripherals,
                              void *   i2cDrv,
                              uint8_t  gpioPortScl,
                              uint8_t  gpioPinScl,
                              uint8_t  gpioPortSda,
                              uint8_t  gpioPinSda
                            )
```

This API set the I2C's GPIO to be used in pointer i2cDrv instance.

Parameters

peripherals	Pointer to main struct with principal components.
i2cDrv	Pointer to i2cDrv instance.
gpioPortScl	Port associated with SCL Port.
gpioPinScl	Pin asscoiated with SCL Pin.
gpioPortSda	Port associated with SDA Port.
gpioPinSda	Pin associated with SDA Pin.

Returns

int Ok = 0 Err < 0

Figura F.11 Parte 2: Documentación del componente I²C

◆ ezm_i2c_subscribe()

```
int ezm_i2c_subscribe ( const void *  i2cInstance,
                        void *        i2cDrv,
                        void *        i2cCallback
                      )
```

This API subscribe in IA's peripheral the instance, i2cDrv and callback function.

Parameters

i2cInstance Pointer to i2cInstance subscribed in I2C's peripheral.

i2cDrv Pointer to i2cDrv instance.

i2cCallback Pointer to function of perEventCalback prototype.

Returns

int Ok = 0 Err < 0

◆ ezm_i2c_unsubscribe()

```
int ezm_i2c_unsubscribe ( const void *  i2cInstance,
                          void *        peripherals
                        )
```

This API delete all the objects reference and variables subscribed with iaInstance.

Parameters

i2cInstance Pointer to i2cInstance subscribed in I2C's peripheral.

peripherals Pointer to main struct with principal components.

Returns

int Ok = 0 Err < 0

◆ ezm_i2c_write()

```
int ezm_i2c_write ( const void *  i2cInstance,
                    uint16_t      slaveAddr,
                    uint8_t       targetAddr,
                    uint8_t *     txBuffer,
                    uint8_t       lenTxBuffer
                  )
```

This API writes to the slave the stored data in txBuffer.

Parameters

i2cInstance i2cInstance Pointer to i2cInstance subscribed in I2C's peripheral.

slaveAddr I2C Slave's address.

targetAddr Must be zero.

txBuffer Pointer to an array txBuffer with len of lenTxBuffer.

lenTxBuffer Len of writting data.

Returns

int Ok = 0 Err < 0

Figura F.12 Parte 3: Documentación del componente I²C

Function Documentation

◆ ezm_ia_ao_attach_cb()

```
int ezm_ia_ao_attach_cb ( const void *  iaInstance,
                          void *        callback
                        )
```

This API attach the callback function to iaInstance & active object.

Parameters

iaInstance Pointer to iaInstance subscribed in IA's peripheral.

callback Callback function that receive events generated.

Returns

int Ok = 0 Err < 0

◆ ezm_ia_ao_init()

```
int ezm_ia_ao_init ( const void *  iaInstance,
                     void *        activeObject,
                     const char *  taskName,
                     uint16_t      stack,
                     uint8_t       priority,
                     uint16_t      xQueueSize,
                     uint8_t       xQueueItems
                   )
```

This API attach the active object to iaInstance and initialize it.

Parameters

iaInstance Pointer to iaInstance subscribed in IA's peripheral.

activeObject Pointer to active object struct to be attached in IA's peripheral.

taskName Task's name for active object's RTOS task.

stack Stack for active object's RTOS task.

priority Priority for active object's RTOS task.

xQueueSize Queue size for active object's RTOS task.

xQueueItems Queue items for active object's RTOS task.

Returns

int Ok = 0 Err < 0

◆ ezm_ia_attach()

```
int ezm_ia_attach ( void *  peripheralIa )
```

This API attach IA's peripheral and initialize its values to zero and pointers to NULL.

Parameters

peripheralIa Main structure of IA's peripheral.

Returns

int Ok = 0 Err < 0

◆ ezm_ia_init()

```
int ezm_ia_init ( const void *  iaInstance )
```

This API initializes IA communication with Tensor Flow Lite drivers.

Parameters

iaInstance Pointer to iaInstance subscribed in IA's peripheral.

Returns

int Ok = 0 Err < 0

Figura F.13 Parte 1: Documentación del componente IA

◆ ezm_ia_predict()

```
int ezm_ia_predict ( const void *  iaInstance,
                     float *       inputs,
                     float *       outputs
                   )
```

This API start running inferences in the AI's model and return the prediction to pointer outputs.

Parameters

- **iaInstance** Pointer to iaInstance subscribed in IA's peripheral.
- **inputs** Pointer to an array with Model's input.
- **outputs** Pointer to an array to saved Model's output.

Returns

int Ok = 0 Err < 0

◆ ezm_ia_set_config()

```
int ezm_ia_set_config ( void *          perIa,
                        void *          model,
                        const uint32_t  lenModel,
                        uint16_t        inputs,
                        uint16_t        outputs,
                        bool            quant,
                        bool            norm,
                        bool            allOps,
                        float *         minValue,
                        float *         maxValue
                      )
```

This API sets IA configuration in pointer perIa and checks if the model is quantized and/or normalized.

Parameters

- **perIa** Pointer to perIa Instance.
- **model** Pointer to Model's C array.
- **lenModel** Len of the AI Model.
- **inputs** Model's number of inputs.
- **outputs** Model's number of outputs.
- **quant** Model is quantized.
- **norm** Model is normalized.
- **allOps** Tensor Flow Lite will have all operations available.
- **minValue** If it is normalized, a float pointer to an array with minimum values must be added. Otherwise, and empty array must be used.
- **maxValue** If it is normalized, a float pointer to an array with maximum values must be added. Otherwise, and empty array must be used.

Returns

int Ok = 0 Err < 0

◆ ezm_ia_subscribe()

```
int ezm_ia_subscribe ( const void *  iaInstance,
                       void *        perIa,
                       void *        iaCallback
                     )
```

This API subscribe in IA's peripheral the instance, perIa and callback function.

Parameters

- **iaInstance** Pointer to iaInstance subscribed in IA's peripheral.
- **perIa** Pointer to **perIA** instance.
- **iaCallback** Pointer to function of perEventCallback protype.

Returns

int Ok = 0 Err < 0

◆ ezm_ia_unsubscribe()

```
int ezm_ia_unsubscribe ( const void *  iaInstance )
```

This API delete all the objects reference and variables subscribed with iaInstance.

Parameters

- **iaInstance** Pointer to iaInstance subscribed in IA's peripheral.

Returns

int Ok = 0 Err < 0

Figura F.14 Parte 2: Documentación del componente IA