



AeroAlert

Sistema de emergencia para aeronaves

Informe descriptivo

Escuela de Educación Secundaria Técnica
N°7 “Taller Regional Quilmes”





ÍNDICE

<u>INTRODUCCIÓN</u>	4
• INTEGRANTES	4
• DOCENTES RESPONSABLES	5
• FECHA DE INICIO	5
• DURACIÓN	5
• PERSONAS AFECTADAS	5
• OBJETIVO DEL PROYECTO	6
• SEGMENTO DESTINO Y ALCANCE SOCIOECONÓMICO	6
• BENEFICIO	7
• FUNCIONALIDADES	9
• INTRODUCCIÓN AL FUNCIONAMIENTO TÉCNICO	9
• PROTOTIPO	10
 <u>DESCRIPCIÓN TÉCNICA</u>	12
• INTRODUCCIÓN	16
◦ OBJETIVOS Y RECOMENDACIONES	
• MÓDULOS DEL SISTEMA: LINEAMIENTOS GENERALES	17
◦ MÓDULO V.D.B	17
◦ MÓDULO A.E.S	17
◦ MÓDULO C.T.R.T	18

• MÓDULO V.D.B: ESPECIFICACIONES	19
◦ LINEAMIENTOS GENERALES	19
◦ DESCRIPCIÓN DEL HARDWARE	20
▪ COMPONENTES UTILIZADOS Y CRITERIO DE SELECCIÓN ...	21
▪ FUNCIONAMIENTO DEL CIRCUITO ELECTRÓNICO	25
◦ ENSAMBLAJE ESTRUCTURAL	28
▪ CRITERIOS DE DISEÑO E IMPLEMENTACIÓN	28
◦ DESCRIPCIÓN DEL SOFTWARE	35
▪ CONFIGURACIÓN DEL SENSOR	35
▪ MEDICIÓN DE SATURACIÓN DE OXÍGENO EN SANGRE	37
▪ MEDICIÓN DE RITMO CARDÍACO	37
▪ ENVÍO DE INFORMACIÓN	39
▪ CONFIGURACIÓN DEL DISPOSITIVO V.D.B	40
• MÓDULO A.E.S: ESPECIFICACIONES	41
◦ LINEAMIENTOS GENERALES	
◦ DESCRIPCIÓN DEL HARDWARE	42
▪ COMPONENTES UTILIZADOS Y CRITERIO DE SELECCIÓN ...	44
▪ FUNCIONAMIENTO DEL CIRCUITO ELECTRÓNICO	50
◦ ENSAMBLAJE ESTRUCTURAL	54
▪ CRITERIOS DE DISEÑO E IMPLEMENTACIÓN	
◦ DESCRIPCIÓN DEL SOFTWARE	61
▪ PANEL DE SIMULACIÓN VIRTUAL	
▪ PROCESAMIENTO DE DATOS Y SISTEMA DE ALERTAS	66
▪ SISTEMA DE ALERTAS SONORAS	77
▪ SIMULACIÓN CON EL PROGRAMA XPLANE - 11	78



• MÓDULO C.T.R.T: ESPECIFICACIONES	86
○ LINEAMIENTOS GENERALES	
○ DESCRIPCIÓN DEL HARDWARE	87
■ COMPONENTES UTILIZADOS Y CRITERIO DE SELECCIÓN ...	90
■ FUNCIONAMIENTO DEL CIRCUITO ELECTRÓNICO	96
○ ENSAMBLAJE ESTRUCTURAL	97
■ CRITERIOS DE DISEÑO E IMPLEMENTACIÓN	98
○ DESCRIPCIÓN DEL SOFTWARE	108
■ RECEPCIÓN Y TRANSMISIÓN DE DATOS	
■ SISTEMA DE ALERTAS	109
■ INTERACCIÓN CON EL OPERARIO	111
■ INTERFAZ GRÁFICA DEL EQUIPO C.T.R.T	113

Introducción

Integrantes

- ALTINIER, Mauro - DNI: 45.910.250 - Curso: 7mo 1ra Aviónica
- CUCCARO, Juan Manuel - DNI: 46 027 093 - Curso: 7mo 1ra Aviónica
- HUSULAK, Mateo - D.N.I: 46 346 883 - Curso: 7mo 1ra Aviónica
- LOPEZ, Teo - D.N.I: 45 783 894 - Curso: 7mo 1ra Aviónica
- PAGANO, Jonás - D.N.I: 46 623 446 - Curso: 7mo 1ra Aviónica
- ROCA, Leandro - D.N.I: 46 428 655 - Curso: 7mo 1ra Aviónica
- RUIZ, Nicolás - D.N.I: 46 199 098 - Curso: 7mo 1ra Aviónica





Docentes responsables

- BIANCO, Carlos
- CARLASSARA, Fabrizio
- CARRO, Jorge
- MEDINA, Sergio
- PALMIERI, Diego
- SALINAS, Jorge
- SBRUZZI, Juan Francisco

Fecha de inicio

La idea fundacional del proyecto AeroAlert tuvo sus inicios a mediados de septiembre del 2022, tras el análisis de un accidente aéreo. Como consecuencia de tal evento, la entonces idea de proyecto fue evolucionando y tomando forma desde ese momento hasta la fecha, dando inicio a su desarrollo como tal a comienzos de marzo del año 2023.

Duración

El tiempo estimado de duración del proyecto es de 28 semanas, con 16 horas semanales, acumulando por lo tanto un total 448 horas de trabajo entre sus integrantes, así como participantes externos intermitentes entre los que destacan profesores y demás tipos de personal ajeno al proyecto.

Personas Afectadas

A lo largo del desarrollo del proyecto, se cuenta con la participación de los integrantes Mauro ALTINIER, Juan Manuel CUCCARO, Mateo HUSULAK, Teo LOPEZ, Jonás PAGANO, Leandro ROCA y Nicolás RUIZ, así como colaboradores externos intermitentes (tales como profesores) en un promedio de trabajo equivalente a 16 horas semanales.



Objetivo del proyecto

AeroAlert es un sistema de emergencia, el cual tiene como meta prevenir un amplio rango de accidentes aéreos. En pro de tal objetivo, implementa un procesamiento y monitoreo constante de parámetros que involucran tanto a los pilotos de aeronaves como al entorno que los rodea.

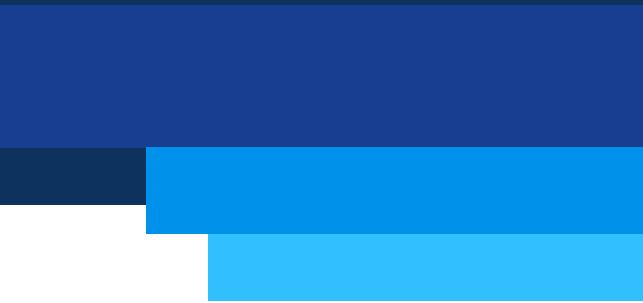
Alrededor del 70% de los accidentes aéreos ocurren debido a fallas humanas, entre las cuales destacan algunas como las imprudencias, el cansancio y el vuelo en condiciones de salud no óptimas, entre otras.

En base a lo previamente dicho, AeroAlert propone el desarrollo de equipos que permitan no solo un monitoreo constante, sino también que provean de una respuesta eficiente ante emergencias que, al día de hoy, no son capaces de resolverse de forma íntegra, y con mínimos o nulos daños, lo cual reduciría enormemente los costos económicos y humanos para las distintas partes implicadas en la industria aeronáutica.

Segmento destino y alcance socioeconómico

Inclusive siendo considerado como el medio de transporte más seguro en todo el mundo, el avión (así como quienes lo pilotan y se transportan en él) no se encuentra exento de sufrir accidentes por desperfectos técnicos y, principalmente, por cuestiones humanas.

Además, y por increíble que resulte en una industria tal como la aeronáutica, existen casos de emergencia que, al día de hoy, no tienen una respuesta completa que minimice el riesgo de accidentes y muertes tanto de pasajeros como de la tripulación y potenciales civiles en tierra. Tan solo se debe pensar qué ocurre si todos los ocupantes de una aeronave quedan inconscientes por hipoxia, o qué pasaría si un piloto desea acabar con su vida estrellando un avión comercial. A pesar de que existan protocolos contra estos casos, ninguno es



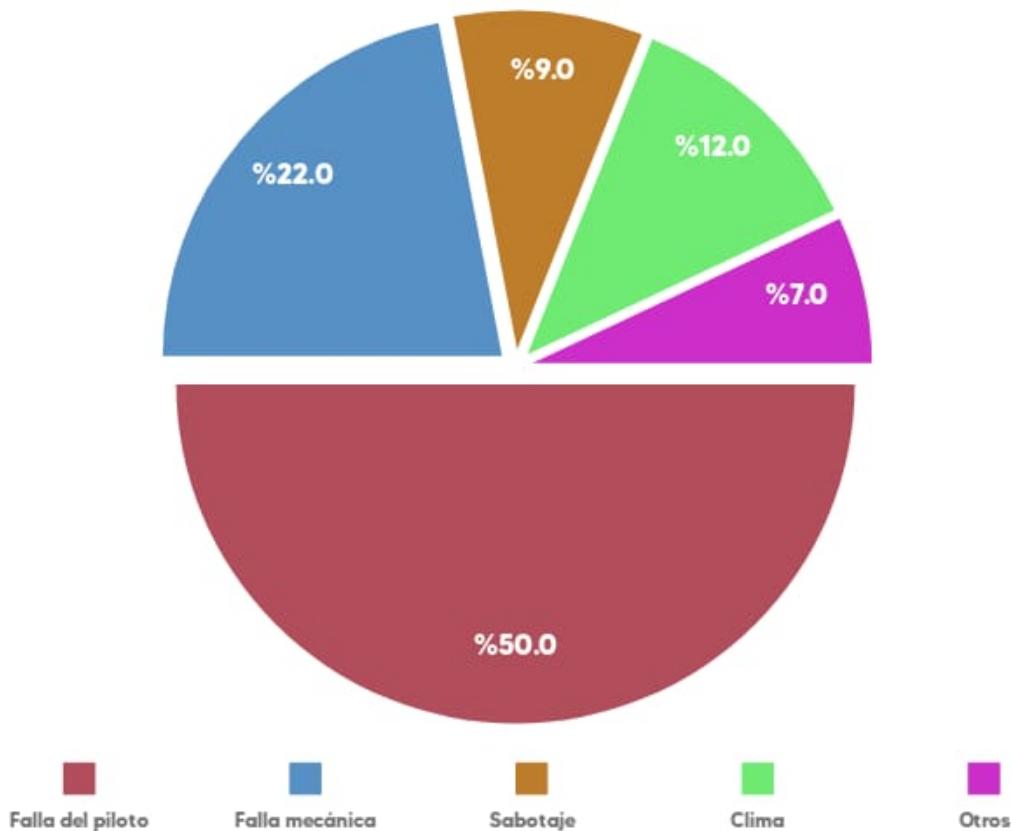
capaz de reducir al mínimo la posibilidad de accidentes y muertes, ya sea a través de un control automático o el monitoreo de la situación por parte del control del tráfico aéreo.

Durante las investigaciones preliminares llevadas a cabo por miembros de este grupo, se han recopilado un total de [27 casos relacionados](#) (de los cuales 12 cuentan con desenlaces fatales) en un lapso de 35 años hasta la actualidad). Esta idea sería capaz de despertar interés en fabricantes de equipos aeronáuticos, aerolíneas y/o agencias reguladoras nacionales e internacionales, que podrían buscar llevar a cabo la idea propuesta de una forma que hoy en día el grupo de trabajo, con sus recursos disponibles, no sería capaz.

Beneficio

En el devenir de las últimas décadas de nuestra historia, el sector aeronáutico ha experimentado notables avances en todas sus dimensiones. Actualmente, los aviones son universalmente reconocidos como el medio de transporte más seguro disponible. A pesar de estos logros, y aunque con infrecuente ocurrencia, los incidentes aéreos continúan portando un potencial destructivo significativo. Dichos eventos trascienden más allá de la pérdida de vidas humanas, tanto de pasajeros como civiles involucrados, impactando así en la reputación de empresas y profesionales de la ingeniería. Inclusive, la supervivencia misma de corporaciones enteras relacionadas a la industria en cuestión puede verse amenazada ante la magnitud de tales sucesos: esto se puede entender observando los estragos que un accidente es capaz de causar en una compañía aérea a través de indemnizaciones, penalizaciones y demás costos asociados a procesos judiciales (los cuales pueden en promedio constituir montos equivalentes a decenas de millones, hasta inclusive cientos de millones de dólares según sea el caso), así como la pérdida de prestigio y confianza por parte de los usuarios de la infraestructura aeronáutica y sus servicios.

Causas principales de los accidentes aéreos



AeroAlert emerge como una herramienta de valor al dirigirse directamente a la prevención de accidentes aéreos, cuya raíz principal continúa siendo el factor humano. Su esencia radica en la capacidad para reducir de manera sustancial el peligro inherente a la operación aérea: por medio del censado constante de las condiciones de salud de los pilotos, sumado a la implementación de equipos que permitan tener un control tanto automático como manual (a través del piloto) y externo (por medio del control del tráfico aéreo) sería capaz de reducirse considerablemente el riesgo de siniestros aéreos, permitiendo la preservación de la vida, así como beneficios tangibles al resguardar la integridad financiera de las compañías aéreas y fortalecer la confianza en la industria en su conjunto.



Funcionalidades

Conforme a su disposición técnica, planteo teórico y el análisis de accidentes aéreos, AeroAlert es un sistema de emergencia planteado con el objetivo de:

- **Preservar tanto vidas humanas como recursos económicos de la industria**
- ***Defender al vuelo de potenciales amenazas***
- ***Mejorar la rapidez de respuesta ante emergencias***
- ***Proporcionar mayor control externo sobre los vuelos***

Introducción al funcionamiento técnico

El objetivo de proyecto requiere la disposición de 2 equipos. El primero de ellos denominado “A.E.S” (***Automatic Emergency System***), tiene como función procesar los datos de los elementos sensores, así como también brindar alertas tanto de índole sonora como lumínica. Este equipo encuentra planificado su emplazamiento en la cabina de la aeronave, debiendo a su vez conectarse a la computadora de vuelo.

El segundo equipo, denominado bajo el acrónimo “C.T.R.T” (***Control Tower Reciever and Transmitter***), cumple el rol de procesar la información enviada por el equipo A.E.S, brindar alertas sonoras y lumínicas al personal en tierra, así como también proveer de instrucciones que permitan efectuar un aterrizaje de emergencia de ser requerido este procedimiento de maniobras.

Respecto a los elementos sensores, el sistema dispone de las “V.D.B” (***Vital Data Bracelet***), el cual le provee al equipo A.E.S información respecto al ritmo cardíaco de los pilotos y su saturación de oxígeno en sangre.

Prototipo

Para su explicación y demostración, el sistema planteado en la idea de proyecto es llevado a cabo en una versión de prototipo, por medio del cual demostrar todas funciones de este, emulando un contexto aeronáutico por medio de un programa simulador de vuelo.

Para lograr dicho cometido, fue necesario el uso de componentes electrónicos, como así también de herramientas, programas e instrumentos, tales como:

Componentes electrónicos:

- Dos placas de desarrollo ESP32 NodeMCU
- Un microcontrolador ESP32 WROOM 32
- Un sensor digital para medición de frecuencia cardíaca y saturación de oxígeno en sangre modelo MAX30102.
- Dos módulos elevadores de tensión (Step - Up) MT3608
- Luminaria ojo de buey
- Relés (Simple Inversor)
- Un display de barras LED
- Transistores BJT NPN modelo BC337
- Interruptores tipo palanca
- Un módulo de carga TP4056 (18650)
- Un regulador fijo positivo LM1117 (3.3V)
- Interruptor con retención Dip Switch
- Interruptor con llave keylock
- Un teclado matricial 4x4
- Una batería de polímero de litio (LiPo) 470 mAh
- Resistores
- Capacitores
- Una placa adaptadora para el microcontrolador ESP32 WROOM 32
- Una computadora personal modelo Dell Latitude 3120



En cuanto a herramientas y programas informáticos, el desarrollo del proyecto ha llevado a sus integrantes a involucrarse en el manejo del siguiente paquete de programas:

Desarrollo de software

La confección y posterior prueba de funcionamiento de programas referentes al debido comportamiento de los diferentes módulos del sistema prototipo planteado, así como su interacción con el software de simulación de vuelo ha requerido emplear una variedad de editores de texto y entornos de desarrollo integrado. Particularmente, ha sido requerida la utilización del editor de texto Visual Studio Code con objetivos de investigación y pruebas de comunicación UDP para con el programa simulador de vuelo, así como el desarrollo del panel de simulación virtual y pruebas de comunicación UART para con el ordenador dedicado al programa simulador de vuelo. El uso dicho editor de texto queda justificado a través de su sencillez, eficiencia y limpia interfaz gráfica que el mismo provee para el desarrollo de software con posterior aplicación en ordenadores.

Por otro lado, se hizo uso del entorno de desarrollo integrado (IDE) Thonny a fin de desarrollar el software MicroPython por medio del cual programar las placas de desarrollo y microcontroladores previamente descritos. Su criterio de utilización se fundamenta en su elevada compatibilidad con el software y firmware de MicroPython, así como su capacidad de cargar software de dicho lenguaje en la memoria del microcontrolador, visualizar los archivos almacenados en la memoria del dispositivo y actualizar el firmware del módulo a conectar (siempre y cuando el mismo sea compatible con el lenguaje en cuestión).

El desarrollo de los distintos códigos fuente asociados al funcionamiento de los módulos pertinentes al proyecto requirió del uso de Git como software de control de versiones, alojando los respectivos archivos en un repositorio dentro de la plataforma web Github.

Diseño de circuitos electrónicos

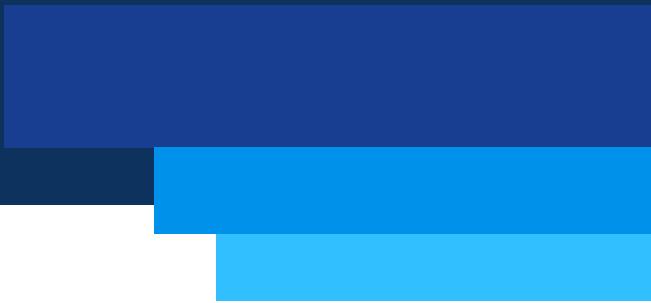
Con el objetivo de confeccionar los diferentes circuitos electrónicos asociados a cada módulo del sistema prototipo, así como su posterior traducción en placa de circuito impreso (PCB), se hizo uso del software de diseño asistido por computadora enfocado a la automatización del diseño electrónico (EDA) KiCAD, aprovechando su simplicidad al momento de su utilización, la rápida curva de aprendizaje que conlleva amigarse con sus distintas funcionalidades y la sencilla capacidad de creación y modificación de componentes (así como librerías) no existentes por defecto en el programa.

Modelado 3D

Se ha empleado la plataforma web de diseño asistido por computadora Onshape con el propósito de modelar las diferentes carcassas y/o estructuras que requieren los módulos prototipo no solo para el alojamiento de sus componentes, sino también para proveer una presentación visual estética de los mismos. Se escoge tal programa a fin de cumplir el objetivo previamente planteado conforme a experiencias previas de utilización por parte de miembros del equipo de trabajo, la rápida curva de aprendizaje y relativa sencillez en su manejo (modelado 3D tipo Sketch - Based), así como su limpia interfaz gráfica y facilitación del trabajo cooperativo.

Diseño de identidad y recursos gráficos

Con el objetivo de diseñar los componentes gráficos correspondientes a la marca del proyecto, tales como logotipos, banners, marcos y portadas referentes a documentación, así como material gráfico apuntado a su uso en redes sociales, se ha hecho uso de diversas plataformas de diseño gráfico.



En particular, el programa de diseño vectorial CorelDraw fue utilizado para diseñar el paquete gráfico referente a la presentación de AeroAlert como marca (logotipos y sus variantes, selección y adaptación de paleta de colores para la posterior presentación como marca), eligiendo tal programa de diseño vectorial por sobre otros bajo razón de su rápida vectorización y adaptación de imágenes, a través de lo cual fue posible la confección de diferentes versiones del logotipo principal según el contexto y organización espacial en el que fuera requerida la aparición del mismo.

Por otro lado, la producción del contenido gráfico destinado al uso en redes sociales, tales como publicaciones, posteos en formato historia, íconos para historias destacadas y la producción de banners (correspondientes tanto al perfil de LinkedIn del proyecto como a la presentación en exposiciones) requirió de la utilización de la plataforma web Canva, orientada tanto al diseño gráfico como a la confección de documentos. La misma herramienta ha sido utilizada a su vez para la redacción de la documentación respectiva al proyecto, aprovechando sus características de gratuidad, autoguardado y facilitación del trabajo cooperativo.

Desarrollo Web

El desarrollo de la plataforma web de presentación del proyecto ha requerido de su correspondiente maquetado y adición de estilos que permitan una fiel representación del paquete gráfico planteado con anterioridad. La tarea en cuestión precisó del trabajo con los lenguajes HTML, CSS y JavaScript, comúnmente asociados al desarrollo web de tipo Front - End. A su vez, se ha utilizado la plataforma web Font Awesome para la adquisición de recursos gráficos externos, con posterior implementación en la página web del proyecto.



Representación de funcionamiento del sistema en el contexto aeronáutico

Con el objetivo de enseñar el funcionamiento del proyecto como prototipo, así como plasmar las diferentes situaciones de emergencia en vuelo hacia las cuales el mismo busca tener alcance, Se ha hecho uso del programa simulador de vuelo X - Plane 11. Su criterio de elección se basa en la relativa sencillez en cuanto a investigación, pruebas y aprendizaje en relación a la manipulación de datos internos, la posibilidad de comunicación (lectura/escritura de valores, instrucciones y comandos) por medio del protocolo de red UDP, así como su utilización en un simulador de vuelo presente en la escuela, sistema en el cual se apuntó a probar el prototipo del proyecto en un inicio.



DESCRIPCIÓN TÉCNICA

Proyecto AeroAlert





Introducción

Objetivos:

El presente capítulo ha sido elaborado con el propósito de proporcionar las ideas y principios de funcionamiento generales de los diferentes módulos que componen al proyecto, enseñando por medio de una sintetizada y a la vez concisa explicación las especificaciones básicas de diseño y cualidades técnicas sobre las cuales se apoyan dichos equipos al momento de su funcionamiento como sistema prototipo.

El capítulo en cuestión cumple el rol de proveer un relevamiento general acerca de la composición interna de los equipos así como de los distintos criterios de diseño, asignables tanto a hardware como software y acabados estructurales.

Recomendaciones:

Se recomienda la lectura del presente capítulo a modo informativo, acercando como parte de la audiencia a todo aquel lector interesado tanto en la idea como en la factibilidad de la misma, con lo cual poder desembocar en un análisis crítico del potencial desarrollo e impacto que podría generar la propuesta de proyecto en el rubro de la aviación, así como en la seguridad de la sociedad civil.



Módulos del Sistema: Lineamientos Generales

Módulo V.D.B:

Acrónimo en idioma inglés de **Vital Data Bracelet** (en español: Pulsera Detectora de Signos Vitales), corresponde al dispositivo pulsera portado tanto por piloto como por copiloto (en caso de existir este último). Su función principal es aquella relacionada al relevamiento de datos asociados a la salud de los conductores de la aeronave (llámese comandante y su potencial compañero de fórmula en caso de existir), con el objetivo de proveer un monitoreo permanente de las variables a medir tanto en cabina como en tierra, de forma tal que exista la posibilidad de generar alertas y/o emergencias tanto de manera presencial (en el medio de transporte físico del vuelo) como remota (por medio del control de tráfico aéreo) en función de las variables de salud relevadas por el módulo descrito previamente, permitiendo la detección temprana de posibles problemas de salud y la toma de medidas adecuadas. La integración de esta tecnología se alinea con el enfoque del proyecto para garantizar la monitorización efectiva de los signos vitales y la rápida respuesta ante cualquier eventualidad.

Módulo A.E.S:

Acrónimo en idioma inglés de **Automatic Emergency System**, es el módulo que se encuentra alojado en cabina, encargado de recibir y procesar la información recopilada por la instrumentación sensorial del sistema (módulo V.D.B). Este equipo desempeña un papel crucial al emitir avisos pertinentes



en cabina, establecer comunicación con el equipo en tierra y, en situaciones críticas, interactuar con el piloto automático nativo de la aeronave. La capacidad de establecer comunicación bidireccional con el equipo en tierra permite la transmisión efectiva de datos relevantes y la recepción de instrucciones actualizadas, lo cual es esencial para la coordinación durante operaciones de emergencia.

Módulo C.T.R.T:

Acrónimo en idioma inglés alusivo a ***Control Tower Reciever and Transmitter*** (en español: Receptor y Transmisor de la Torre de Control), es un módulo alojado en tierra y a disposición del control de tráfico aéreo, cuyo propósito es el de recibir, representar visualmente e informar a los controladores aéreos de emergencias, alertas y/o peticiones. Además de su rol informativo, el módulo C.T.R.T se encarga de generar y transmitir instrucciones al equipo en cabina, específicamente al módulo A.E.S. Su propósito primordial es facilitar la ejecución de aterrizajes de emergencia en caso de que esta acción sea necesaria, garantizando una comunicación efectiva y coordinada entre la torre de control y la aeronave, contribuyendo así a la seguridad y eficiencia en situaciones críticas.



Módulo V.D.B: **Especificaciones**

Lineamientos Generales:

El módulo V.D.B corresponde al dispositivo pulsera portado por el comandante de la aeronave (y su compañero de fórmula, en caso de existir el mismo). El propósito del módulo en cuestión es el de relevar datos asociados a la salud de su portador para con el equipo en cabina, así como la estación en tierra bajo jurisdicción del control del tráfico aéreo, con el objetivo de proveer cierta trazabilidad relacionada a la salud de su usuario, para con ello permitir la detección de anomalías en vuelo capaces de comprometer la maniobrabilidad de la aeronave.

El módulo V.D.B no consta en su versión de prototipo con un alcance considerable en cuanto a la medición de variables asociadas a la salud de su usuario: entendiendo su rol de prototipo y haciendo énfasis en las anomalías barajadas como aquellas de mayor necesidad en su detección de acuerdo a los casos de accidentes aéreos estudiados para con el objetivo del proyecto, se reserva su operación a la detección de variables de salud asociadas a las funciones cardiorrespiratorias del usuario (en particular, la frecuencia cardíaca y la saturación de oxígeno en sangre).



Módulo V.D.B: Descripción del Hardware

A la hora de diagramar el conexionado y funcionamiento electrónico del módulo V.D.B, las principales necesidades y desafíos de diseño a cumplir con el objetivo de garantizar un prototipo eficiente y semejante a diferentes modelos de dispositivos pulsera inteligentes contemporáneos encontradas fueron el requerimiento de que el dispositivo diseñado sea satisfactoriamente portable en la muñeca de su potencial usuario, que sea ligero y de una estética ergonómica, aunque a su vez discreta, y que tenga la capacidad de transmitir la información tomada por un medio inalámbrico.

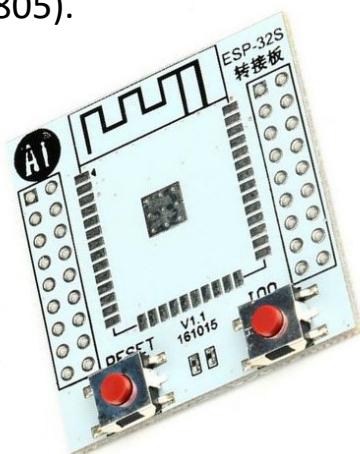
En función de dichos desafíos de diseño, se definieron las principales características de hardware del módulo V.D.B:

- Se opta por la utilización de un microcontrolador ESP32 WROOM con módulo WiFi integrado, descartando la utilización de placas de desarrollo de la misma familia por cuestiones de tamaño y especialidad.
- Se decide implementar el sensor MAX30102, sensor digital capaz de tomar datos de frecuencia cardíaca y saturación de oxígeno en sangre de manera integrada, y comunicarlos con módulos exteriores por medio del protocolo I2C.
- Se decide implementar una batería de polímero de litio (LiPo, con su correspondiente módulo de carga MicroUSB) como fuente de alimentación del dispositivo.

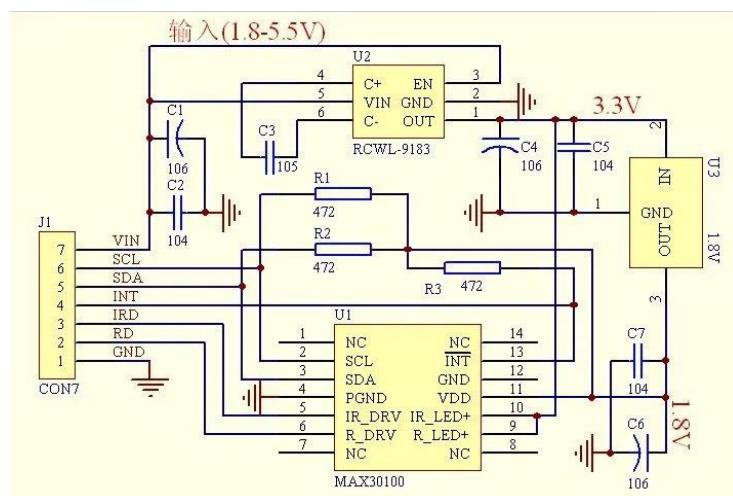
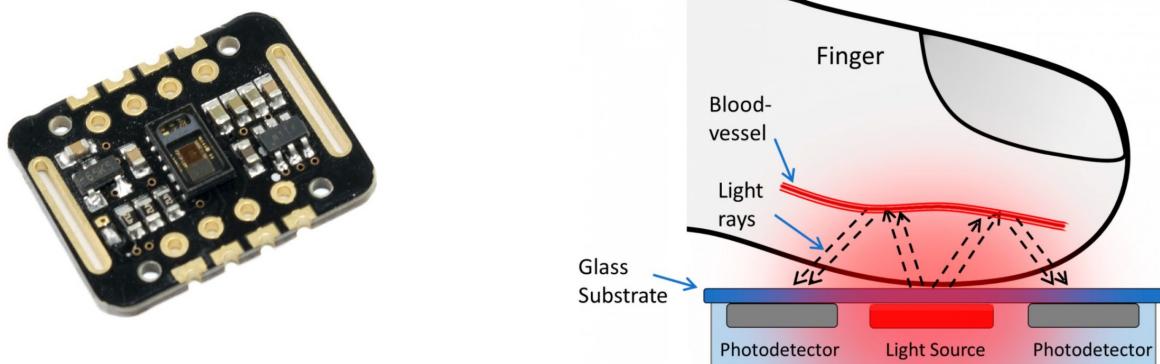
Componentes utilizados y criterio de selección:

En base a la descripción de hardware previamente proveída, así como los criterios de diseño definidos en función de las necesidades encontradas para con el desarrollo del módulo V.D.B., se procedió a seleccionar los componentes consecutivamente descritos para el posterior desarrollo de la placa controladora del dispositivo. A continuación, se detalla el listado específico de componentes integrados en la placa controladora del módulo V.D.B., así como el criterio de selección de cada uno:

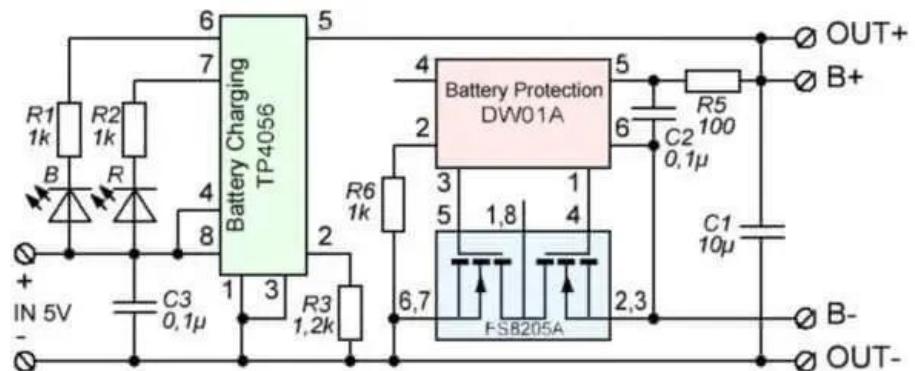
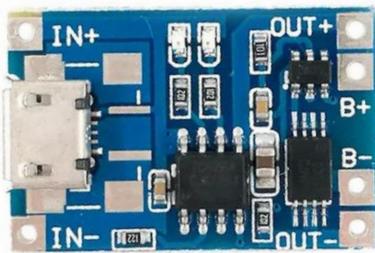
- Microcontrolador ESP32 WROOM 32: Corresponde a la unidad de procesamiento del módulo V.D.B. Su utilización queda definida dada su conveniencia en tamaño y portabilidad, así como la integración de tecnología WiFi en su compacta talla. A su vez, presenta la necesidad de implementar técnicas de soldadura SMD para su adhesión en la placa. Por lo tanto, se procede a la utilización de una placa adaptadora de tecnología SMD a THT (agujero pasante) para la utilización más amena del microcontrolador en pruebas e implementaciones finales. Se destaca que dicha placa adaptadora requiere de la unión por soldadura de pulsadores SMD para reset y flasheo del microcontrolador, así como resistores pull - up de 10KΩ (tecnología SMD, encapsulado 0805).



- Sensor MAX30102: Corresponde a la unidad de adquisición de datos asociados a la salud del usuario para con el módulo V.D.B. El mismo es un sensor de tipo digital (transmisión de datos por medio de protocolo I2C), el cual integra capacidad de medición de frecuencia cardíaca y saturación de oxígeno en sangre, aprovechando el fenómeno físico de reflexión de la luz. Su elección se encuentra justificada en su tamaño compacto, así como la capacidad de medición de más de una variable de salud dentro de un mismo módulo y la relativa sencillez de su implementación. El circuito electrónico equivalente al sensor digital en cuestión, de la misma forma que su principio de funcionamiento, queda definido en las siguientes imágenes.



- Módulo de Carga TP4056 (18650): Corresponde a la etapa de carga de la batería de LiPol. Es capaz de cargar la batería emulando la curva de carga ideal de las baterías de ion de litio, e integra un puerto MicroUSB para la alimentación exterior. La utilización de dicho componente resulta por sugerencia de un docente responsable, destacando su tamaño compacto, relativa universalidad en su alimentación y capacidad de emular la curva de carga. El circuito equivalente del módulo de carga se detalla en las siguientes imágenes:



- Regulador fijo positivo LM1117: Es el componente clave en la etapa de adaptación de voltaje proveniente de la línea de alimentación de la placa (sea que dicho voltaje provenga de la batería o, en su defecto, del módulo de carga). En la versión utilizada, se encarga de regular el voltaje de forma fija hasta el nivel de 3.3V necesario para alimentar correctamente el microcontrolador. Se elige la implementación del mismo en tecnología SMD (encapsulado SOT -223) por cuestiones de tamaño. Se destaca que el mismo debe encontrarse acompañado de capacitores (10 uF) eléctricamente conectados entre la entrada y el plano de masa, así como entre la salida y dicho plano.



- Batería LiPo: Corresponde a la fuente de alimentación del módulo V.D.B. Se elige una batería de LiPo con capacidad de 470 mAh (encapsulado tipo pouch) por cuestiones de tamaño y densidad energética.



- Interruptor con retención: El mismo cumple la función de cortar o permitir la alimentación de la fuente para con el microcontrolador y, por lo tanto, con el resto del circuito electrónico de la plaqueta. La idea de su implementación se reserva para casos de fallas operativas, así como la necesidad de asemejar el prototipo a los dispositivos pulsera contemporáneos. En la plaqueta, se reserva una tira de 2 pines para poder alojar al componente de forma remota (aunque cableada) en un espacio físico desarrollado en la carcasa del módulo V.D.B.

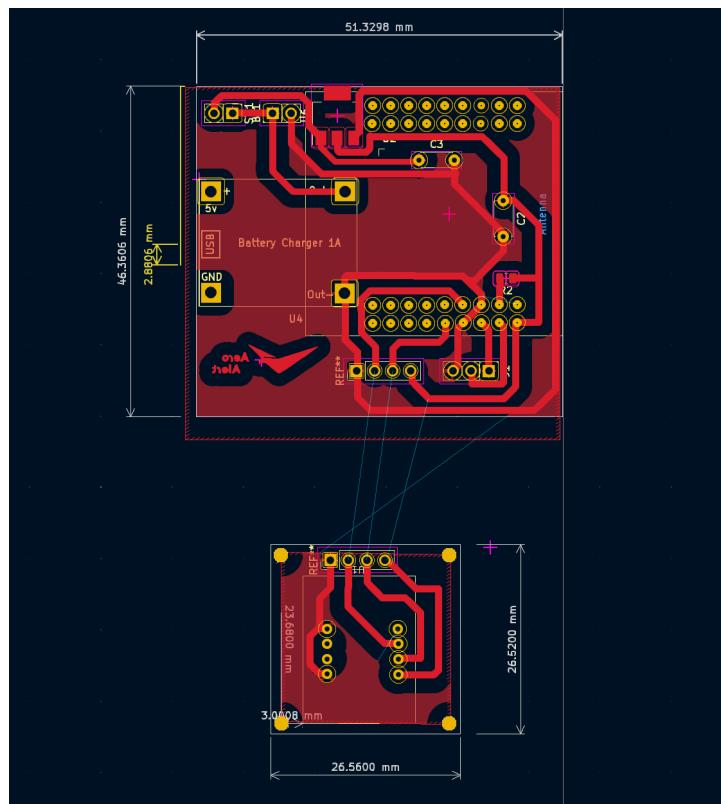
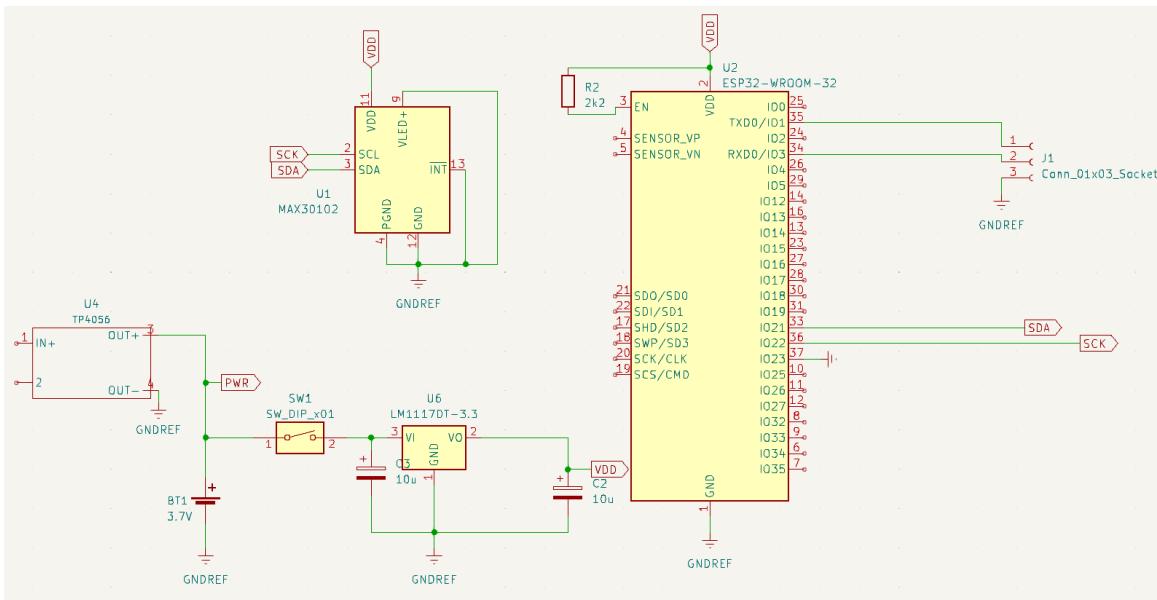


Se destaca la utilización de un adaptador USB a UART modelo PL2303 (por fuera de la plaqueta) con el objetivo de cargar el firmware y los distintos programas que el microcontrolador deberá correr en tiempo real (el microcontrolador no soporta nativamente protocolo USB como demás placas de desarrollo, dificultando así su conexión con ordenadores al momento de cargar programas en memoria). Se reserva una tira de pines en la plaqueta con tal objetivo.



Funcionamiento del Circuito Electrónico:

El circuito electrónico correspondiente al módulo V.D.B., para su mejor comprensión, queda detallado en los siguientes esquemas:

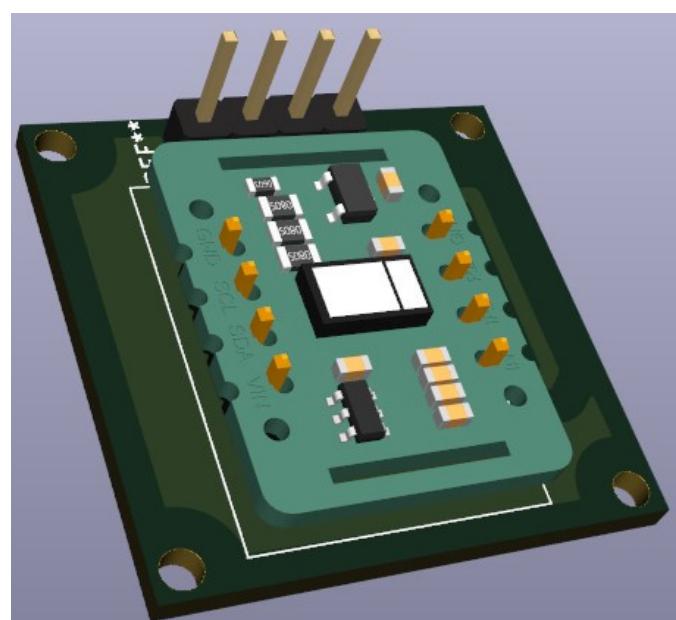
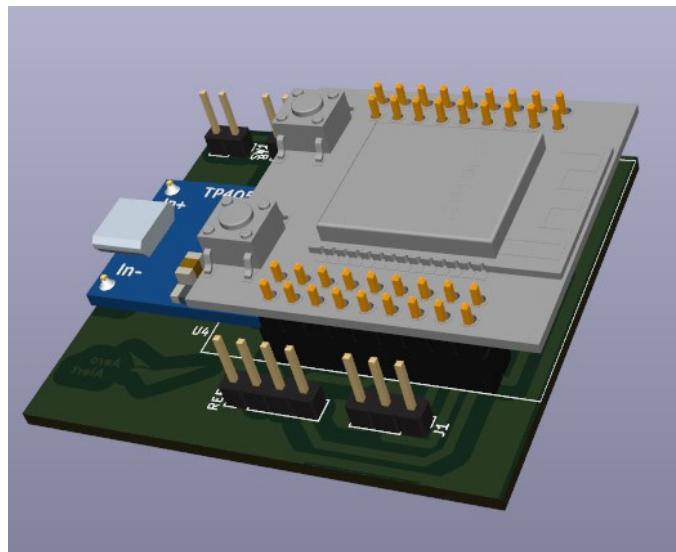




El circuito electrónico del módulo V.D.B. no consta de mayores complejidades en su diseño: Por medio del cargador TP4056 (18650) se provee de alimentación a la batería (para su correspondiente carga, y solamente cuando se requiere cargar la misma, apelando al buen cuidado de la batería en cuanto al proceso de carga). Por medio del interruptor con retención, se determina la alimentación de las etapas posteriores del circuito (aquellas con mayor utilidad real en cuanto al alcance del dispositivo V.D.B.). A la salida del interruptor se encuentra la etapa reguladora, con los correspondientes capacitores requeridos para la obtención de una salida fija de 3.3.V, a través de la cual se provee alimentación tanto al microcontrolador como al sensor MAX30102. Se conecta una resistencia de 2.2k entre la alimentación de 3.3V y el pin EN a modo de limitar la corriente. Se proveen pines de conexión UART (Tx, Rx, GND) para la posterior conexión del adaptador USB a UART siempre que se necesite cargar programas en memoria.

La conexión entre el microcontrolador y el sensor MAX30102 es efectuada por medio del protocolo digital I2C. En el diseño PCB del módulo en cuestión, se decide partir el circuito en 2 plaquetas: una reservada al control de alimentación y al procesamiento de datos, y otra reservada a la adquisición de datos de salud. La decisión de inclinarse por este formato de diseño surge por la necesidad de optimización del espacio (en pruebas iniciales, se desarrollaron diseños en una sola placa, los cuales acarreaban inconvenientes a la hora de poder diseñar a su vez una carcasa compacta para el módulo V.D.B.) En ambas plaquetas, se reserva una tira de 4 pines (VCC, GND, SCL, SDA) para la satisfactoria interacción entre el sensor MAX30102 y el microcontrolador ESP32 WROOM 32.

Las siguientes imágenes corresponden a la renderización de las plaquetas de procesamiento y adquisición de datos previamente descritas. A través de su visualización, se puede notar tanto la disposición espacial de componentes electrónicos como detalles mencionados con anterioridad respecto a la reserva de tiras de pines para propósitos de alimentación y comunicación.





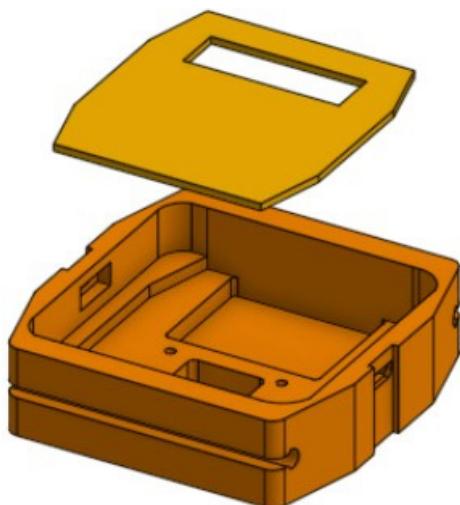
Módulo V.D.B: Ensamblaje estructural

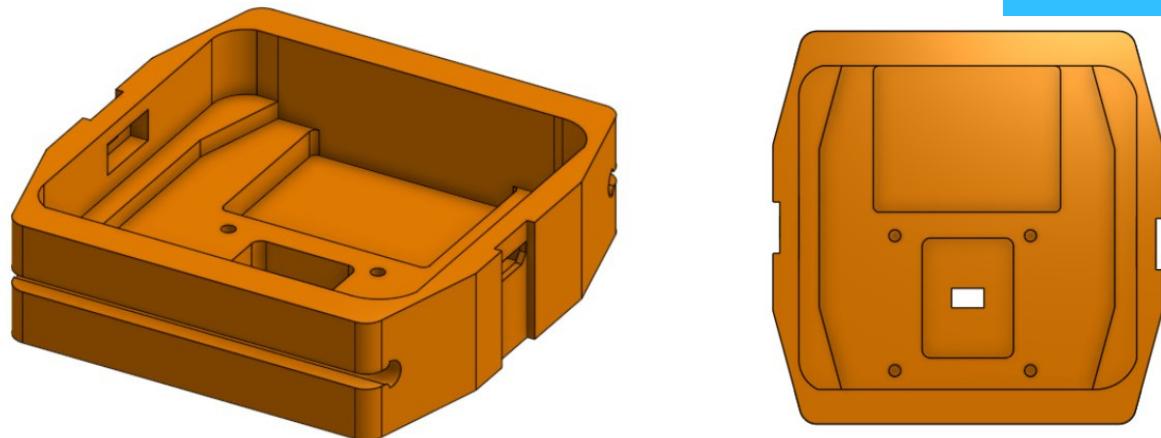
Criterios de diseño e implementación:

Con el objetivo de desarrollar un dispositivo pulsera, haciendo foco en la portabilidad, ergonomía y semejanza con demás dispositivos comerciales de índole similar; el diseño, acabado y ensamblaje estructural debe de seguir la misma línea de pensamiento, tal y como fue planteado, al mismo tiempo, el diseño y disposición del circuito electrónico y sus componentes.

La existencia de un circuito electrónico separado en dos plaquetas (una dedicada a alimentación y procesamiento, otra dedicada a la adquisición de datos) y la necesidad de un método de alojamiento físico para la batería (de manera tal que se encuentre firmemente sujetada en todo momento) fueron dos de las principales características a tener en cuenta para emprender un diseño satisfactorio.

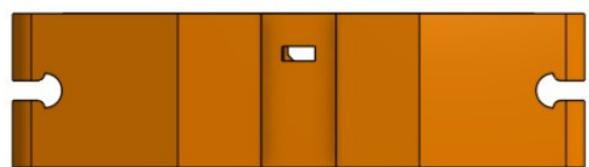
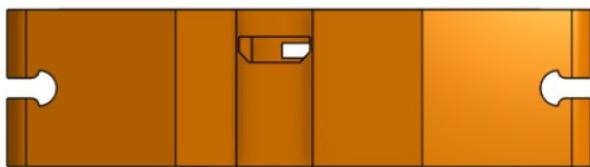
Por medio de la plataforma web Onshape, dedicada al modelado 3D con fines industriales, se procedió a desarrollar el modelo correspondiente a la carcasa del módulo V.D.B. El mismo, con el objetivo de proveer un correcto acabado y ensamblaje, resultó dividido en 2 partes, tal y como demarca la siguiente ilustración:



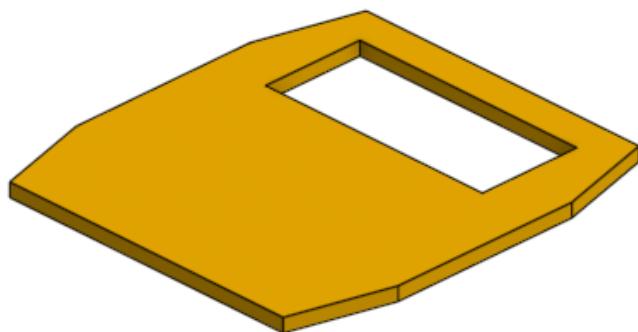


El núcleo estructural de la carcasa del módulo V.D.B encuentra su propósito en alojar tanto a la batería como a las placas de procesamiento y adquisición de datos. En principio, su diseño puede ser entendido como una superficie octogonal con zócalos diseñados para alojar la batería del sistema y la placa de adquisición de datos, rodeados por una pared diseñada con el objetivo de permitir la separación entre la placa de procesamiento y la placa de adquisición de datos, tanto por medios propios como por el encastre generado a través de la tapa intermedia. A su vez, el zócalo diseñado para alojar al sensor MAX30102 cuenta no solo con agujeros para fijación de su placa, sino también con una muesca rectangular para la correcta colocación del LED integrado del sensor, encargado de la adquisición de datos de frecuencia cardíaca y saturación de oxígeno en sangre por medio del principio de reflexión de la luz. Se denota que el zócalo ideado para el alojamiento del sensor MAX30102 tiene el fin de mantener al mismo a la mínima distancia posible de la piel, con la idea de garantizar la correcta adquisición y posterior procesamiento de los datos relevados. El núcleo estructural del módulo en cuestión fue diseñado de manera tal que sea capaz de soportar la conexión MicroUSB requerida por el módulo de carga.

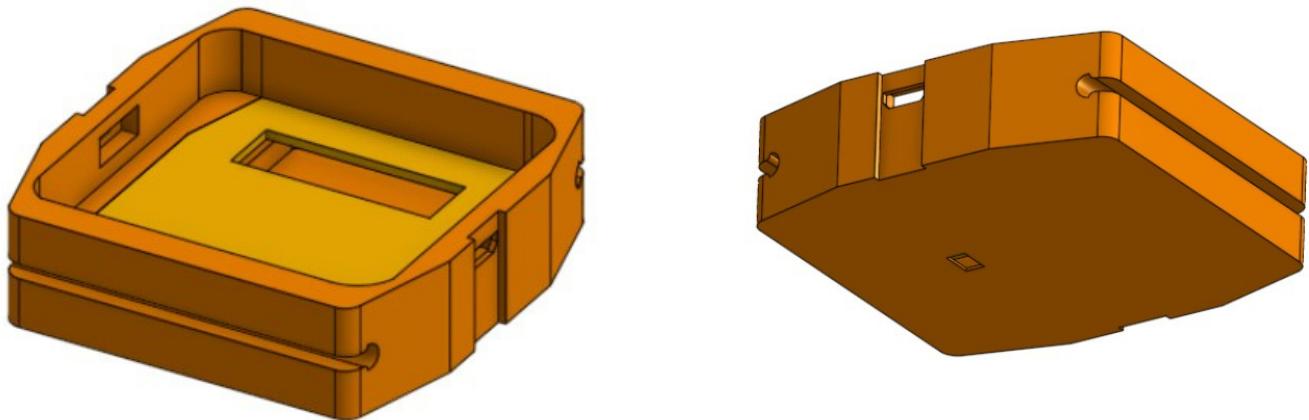
TP4056 (18650), la disposición de un pulsador con retención y el encastre de correas tipo loop, de manera tal que la sujeción del dispositivo a la muñeca del usuario se asemeje a aquella presente en productos comerciales del mismo tipo. Esta parte fue diseñada bajo la idea de converger en un dispositivo abierto en su sección superior, permitiendo mayor libertad al momento de acomodar la placa de procesamiento y sus respectivos componentes. A continuación, se enseñan las vistas de perfil del núcleo estructural, a través de las cuales se puede visualizar de manera más efectiva las muescas diseñadas para el alojamiento del puerto microUSB de carga, el interruptor con retención y las muescas necesarias para el encastre de correas tipo loop.



La tapa intermedia fue diseñada con el objetivo de evitar cualquier potencial contacto indeseado entre las pistas de la placa de procesamiento y la placa de adquisición de datos (dispuestas enfrentadas entre sí. Para satisfacer tal propósito, su diseño copia la forma del hueco presente en el núcleo estructural del módulo. A su vez, se provee de una muesca rectangular en la misma a modo de permitir el pasaje del cableado de comunicación y alimentación necesario para la correcta operación del módulo V.D.B y la debida interacción entre la placa de procesamiento y aquella dedicada a la adquisición de datos.



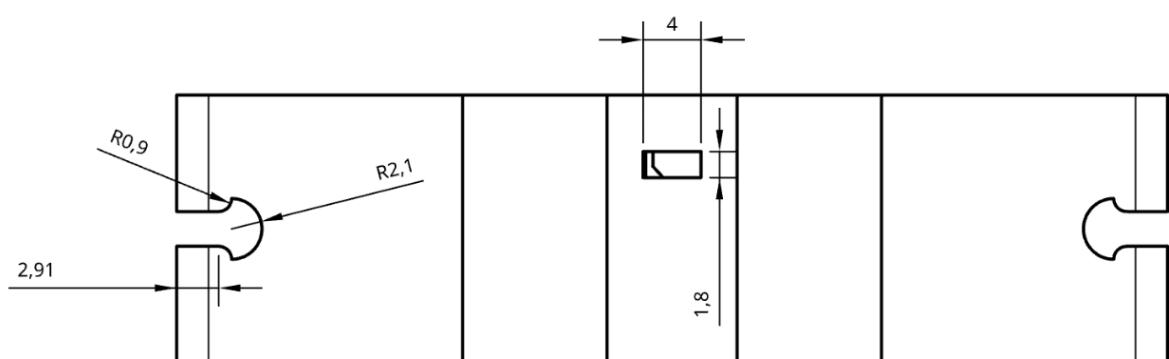
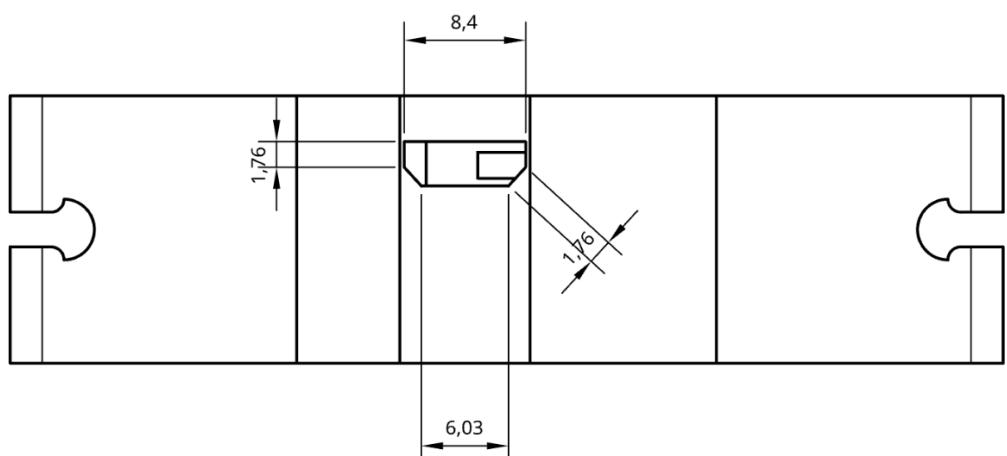
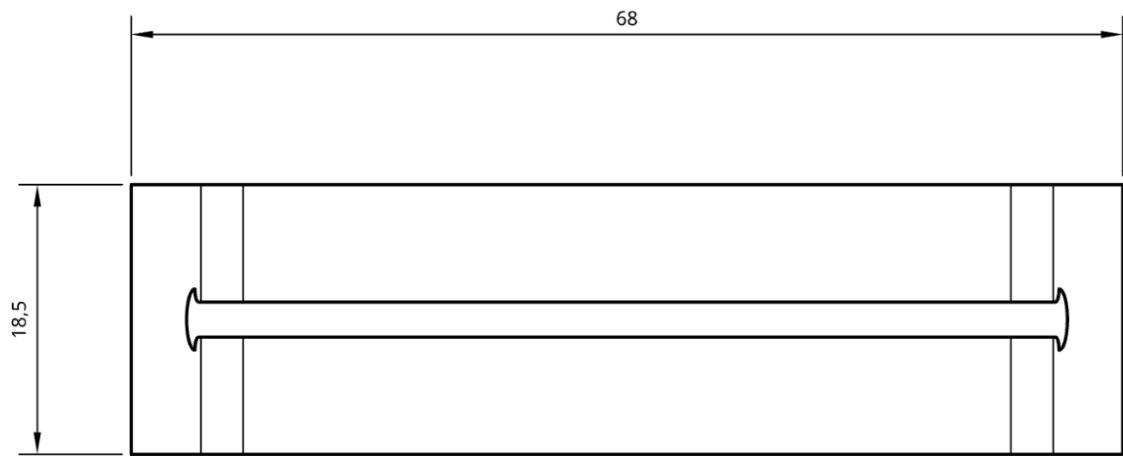
Características del diseño

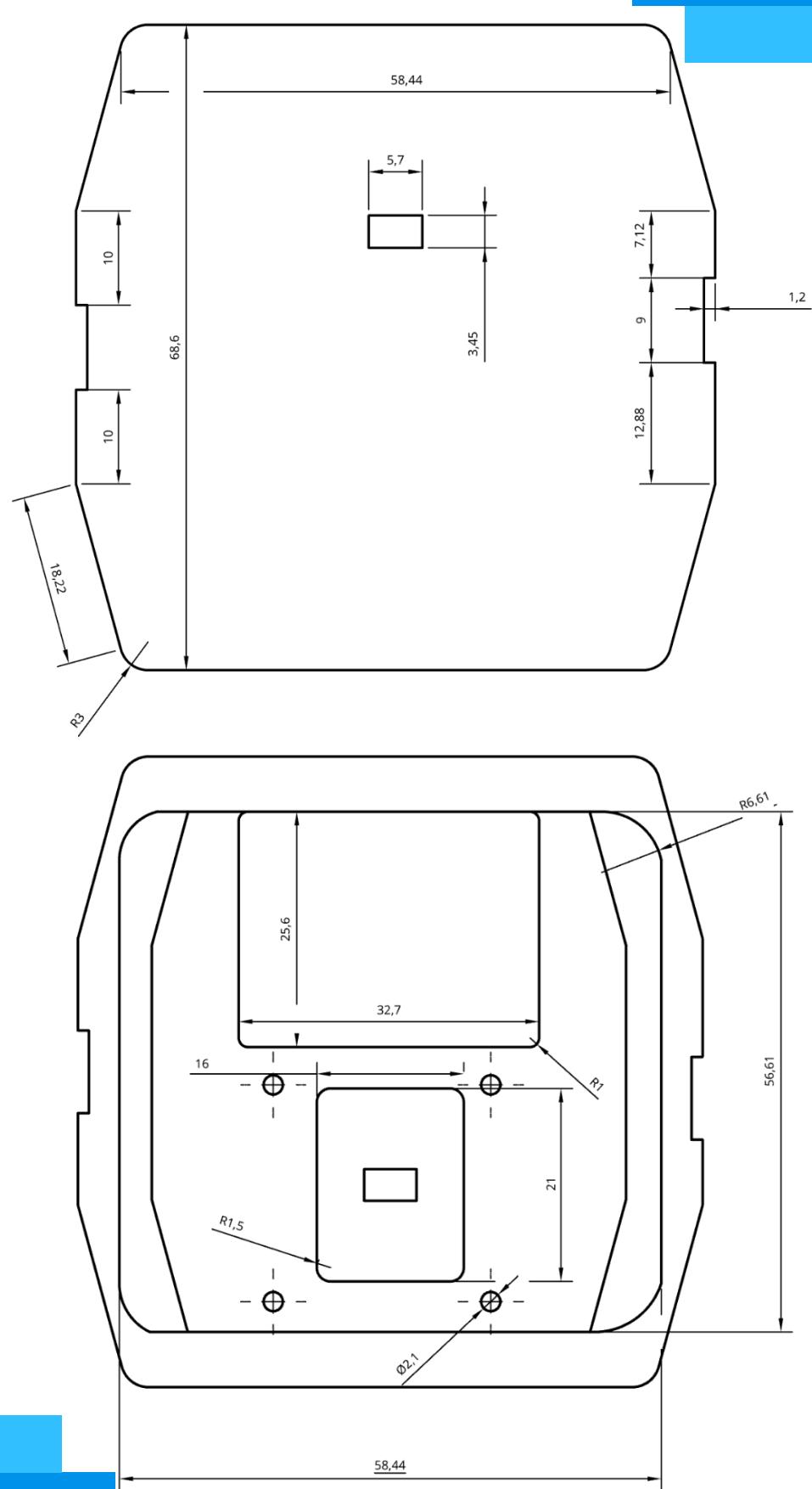


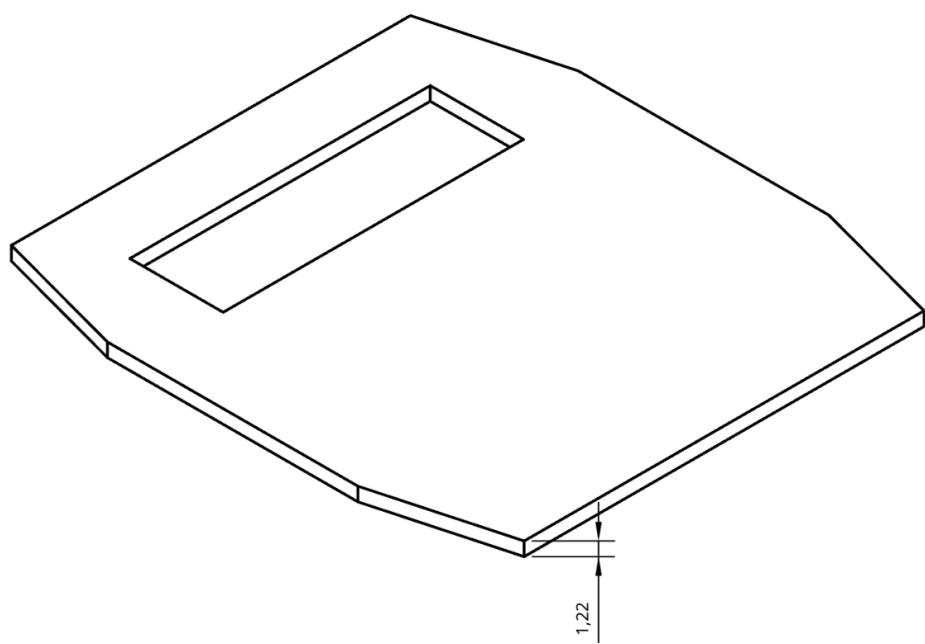
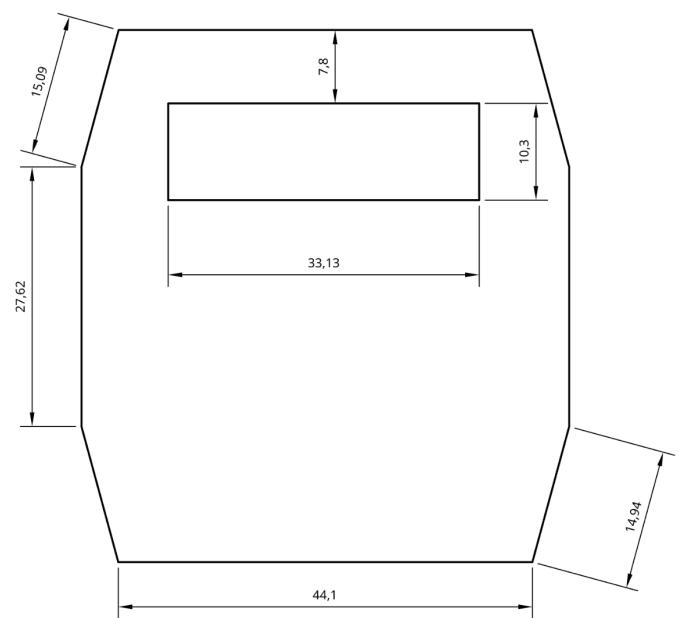
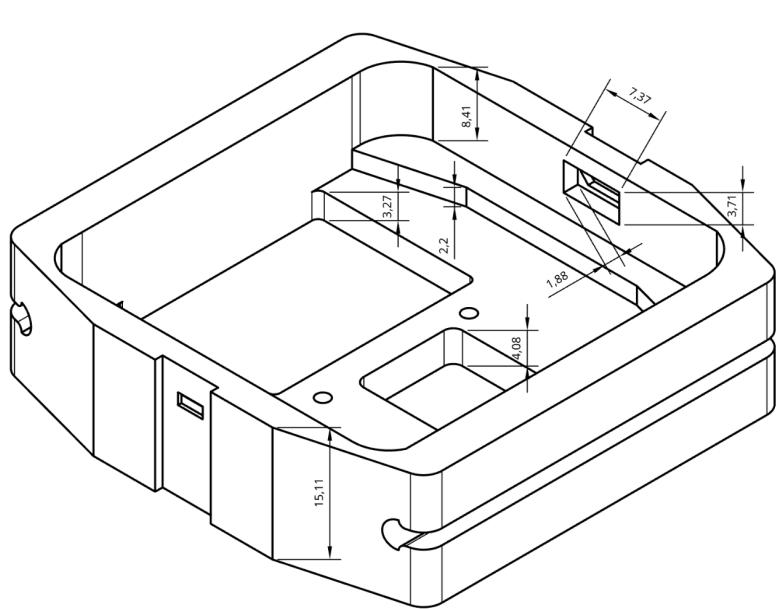
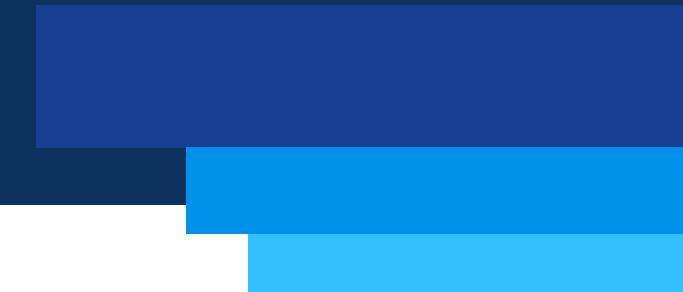
Vistas superior e inferior en perspectiva del ensamblaje de las partes de la carcasa. Se puede denotar tanto la abertura destinada al LED propio del sensor MAX30102, como la muesca de la tapa intermedia dedicada al interconexión de alimentación y comunicación.

Para el posterior desarrollo físico y acabado final del modelado en cuestión, se opta por la fabricación de las piezas por medio de impresión 3D en material plástico PLA, utilizando métodos de impresión convencionales por filamento.

Las siguientes imágenes a presentar corresponden a los planos de las piezas que componen a la carcasa en cuestión, en sus distintas vistas de interés y con las acotaciones necesarias para el correcto entendimiento del diseño. Se destaca que toda medida presente en los planos se encuentra expresada en milímetros (mm).









Módulo V.D.B: Descripción del software

Configuración del Sensor (`pulsometro.py`):

El software relativo al funcionamiento del módulo V.D.B se focaliza en los objetivos de adquirir los datos relevados por parte del sensor digital modelo MAX30102, el procesamiento de estos datos, y la posterior transmisión de tales variables aprovechando el protocolo de red WiFi. Dicho software, presentado parcialmente en el siguiente apartado, se encuentra desarrollado en el lenguaje MicroPython, haciendo uso del editor de texto Visual Studio Code. En el siguiente fragmento, se define un objeto “Pulso” que se configura mediante el protocolo I2C como MAX30102. De ser reconocido, configura el funcionamiento del sensor.

El programa se vale de la utilización de las librerías ***utime*** (medir el tiempo entre pulsos y sleeps), ***max30102*** (uso del sensor MAX30102) y ***machine*** (recepción de datos por I2C y configurar los pines de encendido y led).

A modo de obtener una visión y comprensión completa del software en cuestión, recomendamos visitar nuestro repositorio en GitHub:

github.com/impatrq/aeroalert

```

from machine import SoftI2C, Pin, I2C
from utime import ticks_diff, ticks_us
from max30102 import MAX30102, MAX30105_PULSE_AMP_MEDIUM
import utime

led = Pin(2,Pin.OUT)
class Pulso():
    def __init__(self):
        self.datos=0
        self.datos2=0
        self.datos3=0

    def muestra (self, pin_prendido):
        utime.sleep(1)
        i2c = SoftI2C(sda=Pin(21),
                      scl=Pin(22),
                      freq=400000)
        sensor = MAX30102(i2c=i2c)

        if sensor.i2c_address not in i2c.scan():
            print("Sensor no encontrado")
            return
        elif not (sensor.check_part_id()):
            print("ID de dispositivo I2C no correspondiente a MAX30102")
            return
        else:
            print("Sensor conectado y reconocido")

        print("Configurando el sensor", '\n')
        sensor.setup_sensor()
        sensor.set_sample_rate(400)
        sensor.set_fifo_average(8)
        sensor.set_pulse_width(411)
        #sensor.set_adc_range(16348)

        sensor.set_active_leds_amplitude(MAX30105_PULSE_AMP_MEDIUM)

```

Medición de saturación de oxígeno en sangre (pulsometro.py):

Se genera un bucle en el cual se efectúa la medición de signos vitales solo si el dispositivo pulsera se encuentra encendido (botón), por lo cual recogerá el valor de intensidad de luz infrarroja reflejada, el cual define el nivel de oxígeno en sangre por medio de su multiplicación por una constante (definida como el valor máximo de intensidad de luz infrarroja capaz de ser devuelto por parte del dispositivo sensor).

```
while True:
    utime.sleep(3)
    while pin_prendido.value():
        # The check() method has to be continuously polled, to check if
        # there are new readings into the sensor's FIFO queue. When new
        # readings are available, this function will put them into the storage.
        sensor.check()
        # Check if the storage contains available samples
        if sensor.available():
            # Access the storage FIFO and gather the readings (integers)
            red_reading = sensor.pop_red_from_storage()
            ir_reading = sensor.pop_ir_from_storage()
            valueir = ir_reading
            valuered = red_reading

            Spo2 = valueir * 100/10953
            self.datos2 = Spo2
```

Medición de ritmo cardíaco (pulsometro.py):

La medición de saturación de oxígeno en sangre es posteriormente aprovechada para la adquisición de datos referentes al ritmo cardíaco (pulsaciones por minuto). Para ello, el programa se encarga de guardar los valores (especialmente con los mínimos y máximos, a través de los cuales definir límites superiores e inferiores). Si el valor es relativo a la medición sobre un dedo (>4000) comprueba si supera el límite superior para determinar

que este mismo corresponde al inicio de un pulso, si es menor al límite inferior, el pulso se encuentra en su finalización. Calculando el intervalo de tiempo entre cada inicio de pulso, es posible determinar la frecuencia cardíaca.

```
Spo2 = valueir * 100/10953
self.datos2 = Spo2

history.append(valuered)
# Get the tail, up to MAX_HISTORY length
history = history[-MAX_HISTORY:]

minima, maxima = min(history), max(history)

threshold_on = (minima + maxima * 1.75) // 2.75
threshold_off = (( 1.3 * minima + maxima) // 2.3)
if valuered > 4000:
    if not beat and valuered > threshold_on:
        beat = True
        t_us = ticks_diff(ticks_us(), t_start)
        t_s = t_us/1000000
        f = 1/t_s
        bpm = f * 60
        if bpm < 300:
            t_start = ticks_us()
            beats_history.append(bpm)
            beats_history = beats_history[-MAX_HISTORY:]
            beats = round(sum(beats_history)/len(beats_history) )
            self.datos = beats

    if beat and valuered< threshold_off:
        beat = False
```

Envío de información: (station.py)

Los datos relevados acerca de las variables médicas de interés para el propósito del módulo V.D.B. (el ritmo cardíaco y la saturación de oxígeno en sangre) son transmitidos por medio del protocolo inalámbrico de red conocido como WiFi hacia una placa de desarrollo ESP32 NodeMCU situada en el módulo A.E.S. (descrito posteriormente) para su posterior procesamiento y determinación de casos de anomalías médicas. Este código está realizado en MicroPython, e implementa las librerías **socket** para determinar el puerto al que se va a enviar la información, **time** con el objetivo de enviar información cada cierto tiempo, **json** a modo de codificar el mensaje para mejor comunicación y **network** para conectarse a una red WiFi.

```
def do_connect():
    addr = socket.getaddrinfo('192.168.4.1', 8000)[0][-1]
    server_ip = '192.168.4.1'
    server_port = 8000
    sta_if = network.WLAN(network.STA_IF)

    if not sta_if.isconnected():
        sta_if.active(True)
        sta_if.connect('ESP32', 'aeroalert')
        while not sta_if.isconnected():
            print(".")
            time.sleep(.1)
        print()
    print('network config:', sta_if.ifconfig())

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect((server_ip, server_port))
    print("conecto")

    return client_socket, sta_if

def send_message(bpm, spo2, temp, conectado, client_socket, sta_if):

    data = {'1': bpm, '2': spo2, '3': temp, '4': conectado}
    message = json.dumps(data).encode('utf-8')

    client_socket.send(message)
    print(f'{bpm}, {spo2}, {temp}, {conectado}')

def send_type(client_socket, sta_if):
    message = json.dumps('soy_band').encode('utf-8')
    client_socket.send(message)
```

Configuración del dispositivo V.D.B (main.py):

Por medio de este programa, se coordinan todos los códigos explicados previamente: se configuran los pines, comienzan las mediciones del sensor, y se conecta el V.D.B al A.E.S, transmitiendo las mediciones del sensor MAX30102 a través del protocolo WiFi por medio de un proceso paralelo. El programa hace uso de las librerías **pulsómetro** para crear el objeto sensor (proviene de pulsómetro.py), **Pin** para configurar los pines, **_thread** para iniciar los procesos en paralelo, **utime** para enviar la información periódicamente, **gc** para efectuar un borrado periódico de valores innecesariamente guardados a modo de optimización y **station** para enviar la información al módulo A.E.S.

```
sensor = Pulsómetro()
pin_conectado = Pin(17, mode=Pin.IN)
pin_prendido = Pin(16, mode=Pin.IN)
pin_led = Pin(19, mode=Pin.OUT)
conectado = pin_conectado.value()
client_socket = led_prendido = 0
```

```
def mediciones():
    sensor.muestra(pin_prendido)
```

```
def mostrar():
    utime.sleep(2)
    global client_socket
    while True:
        if pin_prendido.value() == 1:
            print("intentando conectar al AES")
            client_socket, sta_if = station.do_connect()
            print("conectado al AES", client_socket)
            station.send_type(client_socket, sta_if)

        while True:
            while pin_prendido.value():
                if led_prendido == 0:
                    pin_led.value(1)
                    led_prendido = 1
                beats = sensor.datos
                spo2 = sensor.datos2
                temp = sensor.datos3
                conectado = pin_conectado.value()
                print(f'{beats}, {spo2}, {temp}, {conectado}')
                try:
                    station.send_message(beats, spo2, temp,
                                         conectado, client_socket, sta_if)
                    print("enviado")
                except:
                    print("No enviado")
                gc.collect()
                utime.sleep(3)
            if led_prendido == 1:
                pin_led.value(0)
                led_prendido = 0
            utime.sleep(3)
    _thread.start_new_thread(mediciones,())
    utime.sleep(1)
    _thread.start_new_thread(mostrar,())
```

Módulo A.E.S.: Especificaciones

Lineamientos Generales:

El módulo A.E.S. corresponde al equipo dispuesto en la cabina de la aeronave. El propósito del mismo es el de recibir información relevada por la sensórica, procesar dicha información, dar avisos en cabina, comunicarse con el equipo en tierra y en caso de emergencia comunicarse con el piloto automático nativo de la aeronave para efectuar un aterrizaje (de ser este último necesario).

El módulo A.E.S. presenta un panel frontal por medio del cual se efectúa la representación visual y frontal de datos. Dicha representación queda definida bajo un sencillo código de identificaciones luminosas obtenido en función de la información relevada por medio del módulo V.D.B., de las instrucciones enviadas desde el equipo dispuesto en tierra bajo operación del control de tráfico aéreo, o por medio del ingreso manual de instrucciones soportado por el propio módulo a través de su respectivo panel frontal.





Módulo A.E.S.: Descripción del Hardware

A la hora de diagramar el conexionado y funcionamiento electrónico del módulo A.E.S., las principales necesidades y desafíos de diseño a cumplir fueron la capacidad de emular un dispositivo de cabina con representación visual y frontal de datos, así como accionamiento manual por parte del usuario y capacidad de comunicación inalámbrica tanto de las variables de salud adquiridas por el módulo V.D.B. como de aquellos parámetros ingresados por medio del panel frontal del equipo.

En función de dichos desafíos de diseño, se procedió a definir las principales características de hardware del módulo A.E.S.:

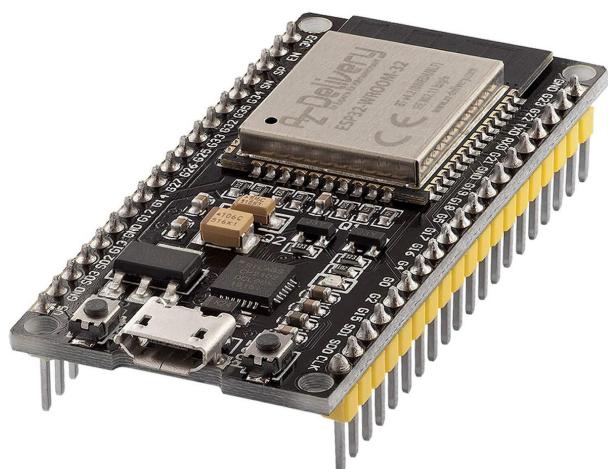
- En esta ocasión, la implementación de una placa de desarrollo ESP32 NodeMCU, aprovechando que la misma integra tecnología WiFi, así como todas las funciones de regulación de voltaje y adaptación de la alimentación, tanto de la propia placa como de los pines GPIO que la conforman. En relación al microcontrolador ESP32 WROOM 32, acarrea una desventaja en cuanto al espacio que la misma ocupa (de hecho, la placa de desarrollo ESP32 NODEMCU utiliza en sí mismo un microcontrolador ESP32 WROOM 32), y al mismo tiempo permite mayor facilidad a la hora de realizar trabajos relacionados a la prueba de componentes y/o circuitos. Las cuestiones de tamaño no resultan mayor impedimento para el desarrollo del hardware del módulo A.E.S., dado que no existe necesidad alguna de desarrollar un dispositivo de ergonomía considerable (más allá del ahorro de material a ser proveído).

- La implementación de luminaria ojo de buey, a modo de asimilar las indicaciones luminosas a desarrollar con aquellas presentes en los sistemas aeronáuticos con representación de información en cabina. Se provee de distintas luminarias ojo de buey (según coloración), con un código de identificación de situaciones por color previamente diagramado, y que deberá de ser respetado para garantizar el correcto funcionamiento del proyecto.
- La implementación de un módulo elevador de tensión (Step - Up) MT3608, con el objetivo de alimentar a la luminaria ojo de buey (la misma opera a una tensión de 12V, mientras que la placa de desarrollo es capaz de ofrecer alimentación hasta un máximo de 3.3V) por medio de la placa de desarrollo. Se destaca que, para el diseño del circuito electrónico del módulo A.E.S., se consideró la implementación de una fuente de alimentación exterior conectada a la placa de desarrollo por medio del puerto MicroUSB integrado en la misma (lo cual no solo facilita su alimentación, sino también su programación).
- La implementación de relés (de tipo simple inversor) con el objetivo de controlar el encendido y apagado de la luminaria ojo de buey según criterios de programación. Se escogen relés como componente conmutador debido no solamente a la simplicidad de su utilización, sino también por su capacidad de controlar una tensión más elevada por medio de una más pequeña.

Componentes utilizados y criterio de selección:

En base a la descripción de hardware previamente proveída, así como los criterios de diseño definidos en función de las necesidades encontradas para con el desarrollo del módulo A.E.S., se procedió a seleccionar los componentes consecutivamente descritos para el posterior desarrollo de la placa controladora del dispositivo. A continuación, se detalla el listado específico de componentes integrados en la placa controladora del módulo A.E.S., así como el criterio de selección de cada uno:

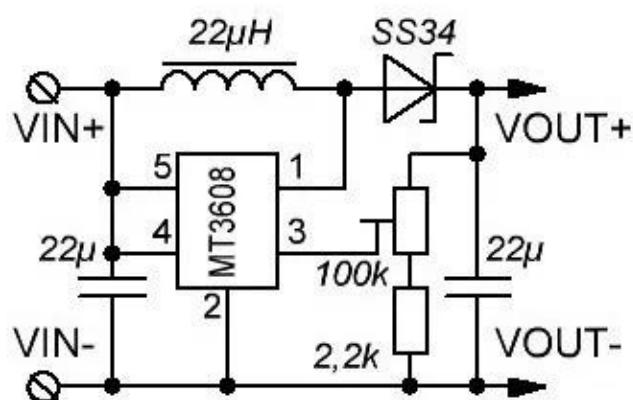
- Placa de desarrollo ESP32 NodeMCU: Corresponde a la unidad de procesamiento del módulo A.E.S. Su utilización queda definida dada la sencillez de su implementación (integración correctamente resuelta de todos los módulos de alimentación, regulación de alimentación y protocolos de comunicación sin necesidad de intervención ni posterior diseño por parte de su usuario), así como la integración de tecnología WiFi, requerida en esta ocasión para la recepción de los datos adquiridos por parte del módulo V.D.B, su procesamiento y posterior comunicación con el equipo en tierra (de ser ello necesario) .



- Luminaria ojo de buey: Corresponde al principal medio de representación visual y frontal de datos que integra el panel en cabina del módulo A.E.S. Su criterio de selección queda definido principalmente por cuestiones estéticas (se consideró su implementación tras denotar una mayor similaridad y profesionalidad con la luminaria existente en los típicos páneles frontales por parte de este tipo de luces que aquella presentada por otros modelos). Dentro del panel frontal, se puede observar la existencia de luminaria ojo de buey tanto color roja como ámbar, definido bajo criterios de representación visual de alertas y/o emergencias en función del protocolo a seguir (Warning , Alert, Test Fail). Se destaca que la tensión de alimentación de la luminaria ojo de buey difiere respecto a aquella que la placa de desarrollo es capaz de ofrecer por medio de sus GPIOs. Por lo tanto, se procedió a seleccionar componentes que permitan adaptar la tensión de alimentación para el correcto uso y control de encendido de la luminaria, así como dispositivos conmutadores que permitan controlar circuitos operativos a tensiones mayores por medio de tensiones menores. Dichos componentes quedan detallados en los ítems consecutivos.



- Módulo elevador de tensión (Step - Up) MT3608: Corresponde al componente adaptador de tensión de trabajo de la placa de desarrollo con la luminaria del panel (se destaca que, al no poseer una fuente de alimentación interna, toda carga existente en el circuito electrónico del módulo A.E.S. es directa o indirectamente alimentada por la tensión que la placa de desarrollo es capaz de ofrecer). El componente en cuestión fue SELECCIÓNado debido a la sencillez de su implementación (evita la necesidad de diseñar un módulo Step - Up por cuenta propia, al mismo tiempo que soluciona dicha problemática en un compacto tamaño, de manera tal que su incidencia en la placa en términos de espacialidad se reduce al mínimo posible) y su capacidad de configuración (su tensión de salida queda definida en función del ajuste del cursor de un preset integrado para tales fines). Por medio de las siguientes imágenes, queda definido el circuito esquemático equivalente del módulo.



- Relevador simple inversor: Este tipo de componente fue seleccionado para obrar como controlador de conmutación de la luminaria ojo de buey (llave accionada por corriente eléctrica proveniente de GPIOs) dada su sencilla interacción tanto con la placa de desarrollo como con la carga a alimentar. A su vez, tampoco requiere de configuraciones específicas para su correcta conmutación. Se destaca que la tensión de conmutación del relevador simple inversor (5 a 12 V) difiere para con aquella capaz de ser proveída por la placa de desarrollo (3.3V). Con dicho objetivo, se coloca un dispositivo transistor en configuración de conmutación, con un correspondiente diodo rectificador en antiparalelo a la inductancia del relevador.



- Interruptor por llave Keylock (On/Off): El componente en cuestión corresponde al interruptor asignado para la activación manual del módulo A.E.S. por medio del panel frontal. Su criterio de selección queda definido por sus características de retención de estado e incapacidad de conmutación de no contar con la correspondiente llave que permita su cambio de estado.



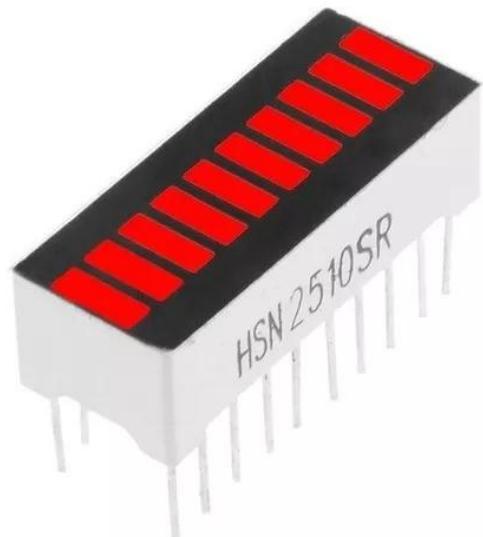
Transistor BJT NPN BC337: El componente en cuestión se encarga de obrar como intermediario a fin de hacer posible el control de conmutación del relevador por medio de la tensión proveída por la placa de desarrollo. La implementación del modelo BC337 en particular no es requisito excluyente para lograr una correcta conmutación (existen diferentes modelos de transistores en el mercado para una aplicación genérica tal como la requerida en esta ocasión), aunque su elección particular se debe principalmente a una serie de pruebas exitosas efectuadas con este modelo. Cabe destacar que se debe de colocar un resistor de valor máximo 10KΩ entre la alimentación del GPIO y la base del componente a fin de que el mismo entre en su zona de saturación.



Interruptor por llave a palanca: El componente en cuestión obra como interruptor asignado para la activación manual del módulo A.E.S, así como del modo de testeo del equipo. Su criterio de selección queda definido por su ergonomía simple y adaptable a cualquier cuestión de conmutación básica, así como su sencilla implementación a nivel electrónico.

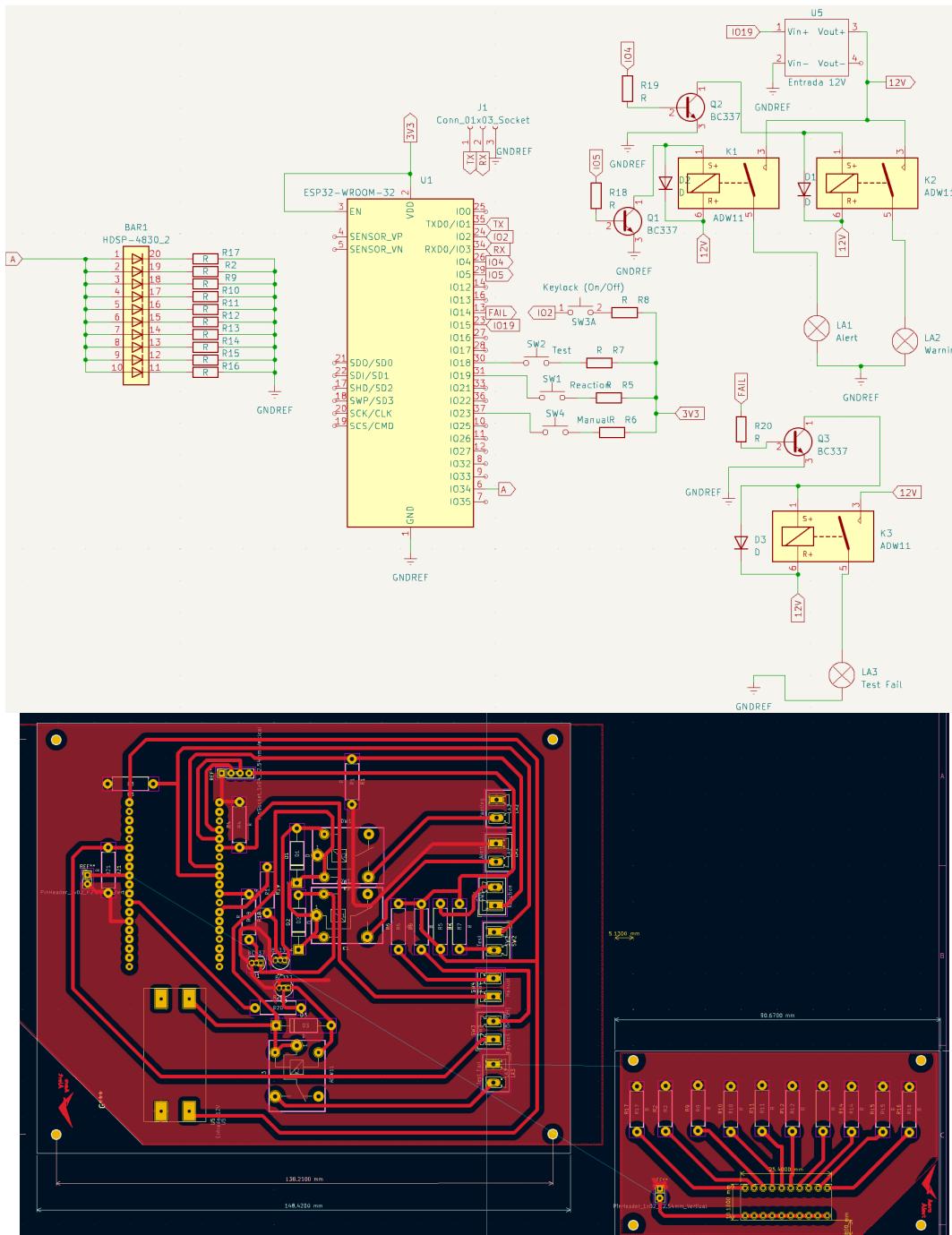


- Display de barras LED: Corresponde al componente definido para obrar como flag de advertencia del panel frontal del módulo A.E.S. Su criterio de selección se reduce a cuestiones de luminosidad del componente y su tamaño, barajado como ideal para la representación de dicho flag en el panel frontal del módulo.



Funcionamiento del Circuito Electrónico:

El circuito electrónico correspondiente al módulo A.E.S., para su mejor comprensión, queda detallado en los siguientes esquemas:





El circuito electrónico del módulo A.E.S. no consta de mayores complejidades en su diseño, aunque si goza de cierta repetitividad en cuanto a sus distintas secciones para luminaria e interruptores: Asumiendo que la alimentación del circuito es recibida por medio de una fuente externa conectada a la placa de desarrollo por medio del puerto MicroUSB integrado, se desprende que cada GPIO configurado a nivel programa como salida digital será capaz de generar una diferencia de potencial máxima de 3.3V entre su terminal y el plano de masa.

Para cada circuito de luminaria ojo de buey (Warning, Alert y Test Fail) se conecta la base de un transistor BC337 junto a un resistor de $10K\Omega$ máximo a la salida directa de cada GPIO asignada a tal función (caso contrario, el transistor no es capaz de alcanzar su zona de saturación). Se denota que el colector de cada transistor se encuentra conectado a uno de los extremos de la inductancia del relevador, así como su respectivo emisor se encuentra referenciado a masa. De esta forma, el rol de dicho transistor en el circuito es muy sencillo de explicar: como el terminal de alimentación del relevador se encuentra conectado a la salida de 12V del módulo Step -Up, el transistor obrará como una llave en el terminal neutro del bobinado. Cuando la base del transistor es energizada por su GPIO a requerimientos de programa, el transistor comienza a conducir y cierra el circuito de alimentación del relevador, produciéndose entonces la conmutación y posterior encendido de la luminaria ojo de buey en cuestión, ya que al igual que el relevador se encuentra siempre alimentado a 12V (conmuta únicamente cuando se cierra el terminal neutro del circuito), el contacto asociado se encuentra siempre energizado en uno de sus terminales, esperando a ser cerrado para proveer alimentación a la luminaria que deba de ser energizada.



Por otro lado, se encuentra el tramo del circuito asignado a los interruptores: cada uno de ellos se conecta entre la alimentación y un GPIO configurado como entrada, con un resistor en serie a modo de limitar la corriente eléctrica. Cada GPIO evaluará a nivel programa el estado lógico de su entrada (“1” si recibe alimentación a través de un contacto cerrado, “0” si no recibe alimentación debido a un contacto abierto), a través de lo cual se procederá posteriormente, a requerimientos de programa, a desarrollar una serie de protocolos que pueden incluir la activación de luminaria y/o la comunicación con el equipo en tierra.

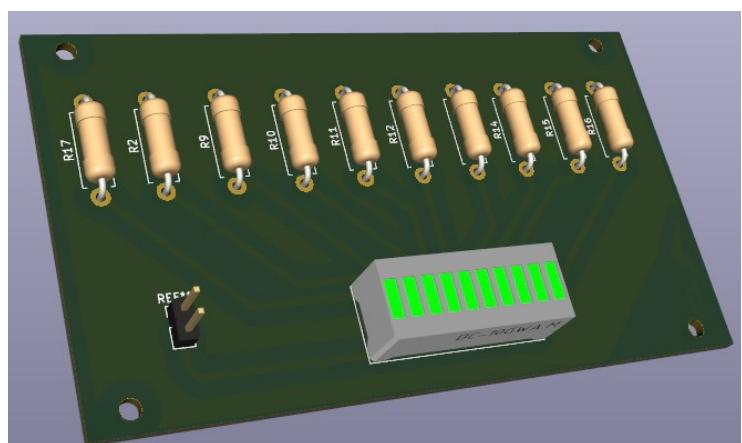
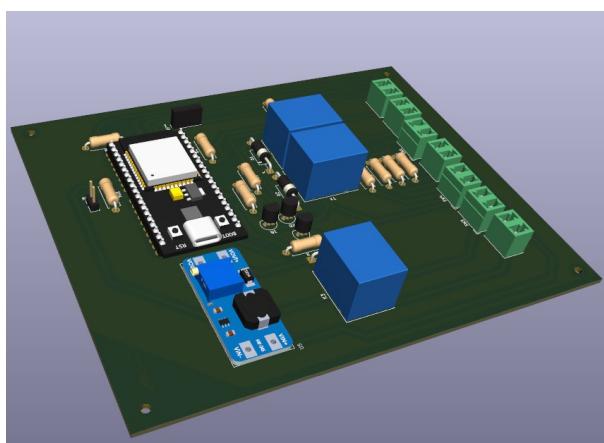
Por otro lado, también es importante describir el tramo del circuito dedicado a la adaptación de voltaje por medio del módulo Step - Up: su entrada positiva se encuentra conectada al pin de alimentación de la placa de desarrollo (3V3), mientras que su entrada negativa se encuentra referenciada a masa.

Posteriormente, su salida positiva es aquella capaz de proveer los 12V necesarios para la operación del apartado de luminarias del circuito según configuración del preset integrado (alimentando tanto al bobinado del relevador, como al contacto asociado), mientras que la salida negativa es ignorada para el interés y objetivo del diseño.

Por último, se puede visualizar en el circuito electrónico un apartado dedicado al funcionamiento de la barra LED (emuladora del flag de advertencia). A modo de eficientizar el uso de GPIOS de la placa de desarrollo, el principal objetivo de este tramo del circuito no solo es el accionamiento de 10 LEDs en paralelo por medio de un solo pin GPIO, sino también ser capaz de proveer la corriente necesaria para el correcto funcionamiento de la barra LED (se destaca que la corriente demandada es de alrededor de 200 mA, superior a la que la placa de desarrollo es capaz de proveer). A modo de solventar tales problemáticas, se coloca un transistor BC337, con su emisor conectado al

GPIO controlador de dicho tramo del circuito en conjunto con un resistor de valor máximo 10KΩ, su colector conectado al pin de alimentación de la placa de desarrollo (3V3) y su emisor referenciado a la entrada de cada LED de la tira. A su vez, cada LED de la barra correspondiente cuenta con su propio resistor a modo de limitar la corriente.

Para este caso, el transistor vuelve a obrar en conmutación: por medio de la energía que el GPIO conectado a la base provea según requerimientos de programa, el transistor entrará en conducción y alimentará a los distintos LEDs que conforman al display de barras, a través de los cuales se podrá visualizar la indicación luminosa del flag de advertencia del módulo A.E.S. Se destaca que el circuito del display de barras LED fue separado en otra placa por fuera de la placa madre del módulo A.E.S por cuestiones de ensamblaje estructural y disposición de componentes en el panel frontal.

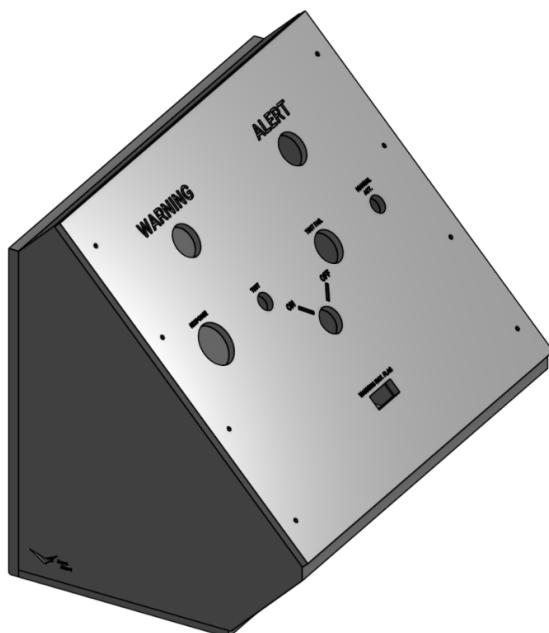


Módulo A.E.S.: Ensamblaje estructural

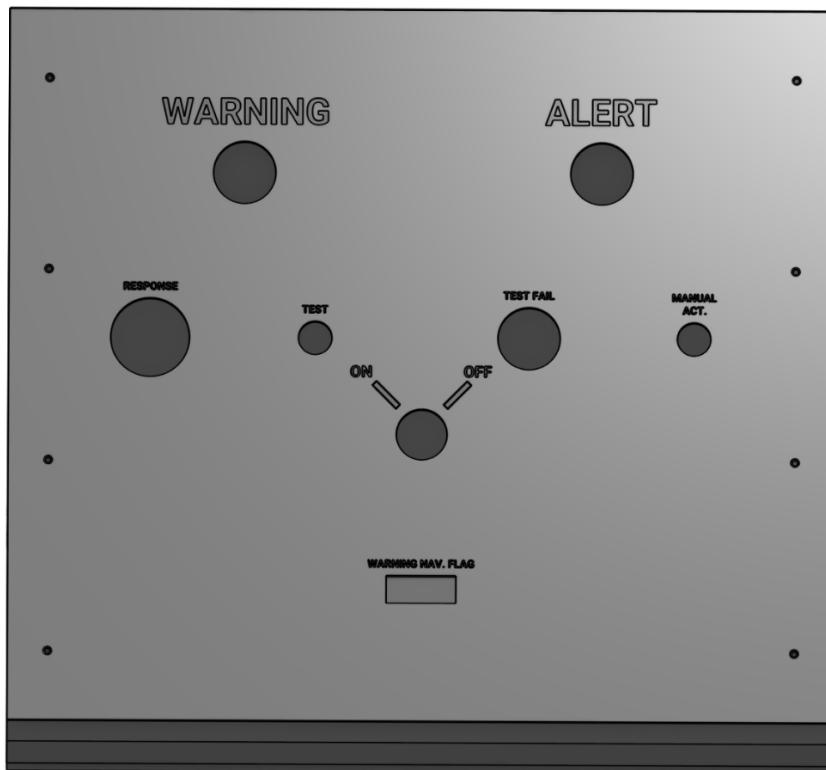
Criterios de diseño e implementación:

Con el objetivo de desarrollar un equipo cuyo propósito es su alojamiento en cabina, y la enseñanza por medios visuales y frontales de alertas y/o emergencias en vuelo, se procedió a diseñar una carcasa para el módulo A.E.S haciendo el principal foco en el desarrollo del panel frontal del equipo. A su vez, los principales desafíos de diseño a enfrentar para el modelado de la correspondiente carcasa consistieron en el espacio físico en el cual alojar la placa madre del módulo (y su principal inconveniente derivado, la conexión de la luminaria e interruptores del panel frontal para con ella), así como proveer un espacio dedicado a la conexión cableada de la alimentación del equipo por medio del puerto MicroUSB que integra la placa de desarrollo.

Haciendo uso de la plataforma web Onshape, dedicada al modelado 3D con fines industriales, se procedió a desarrollar el modelo correspondiente a la carcasa del módulo A.E.S., tal y como demarca la siguiente ilustración:

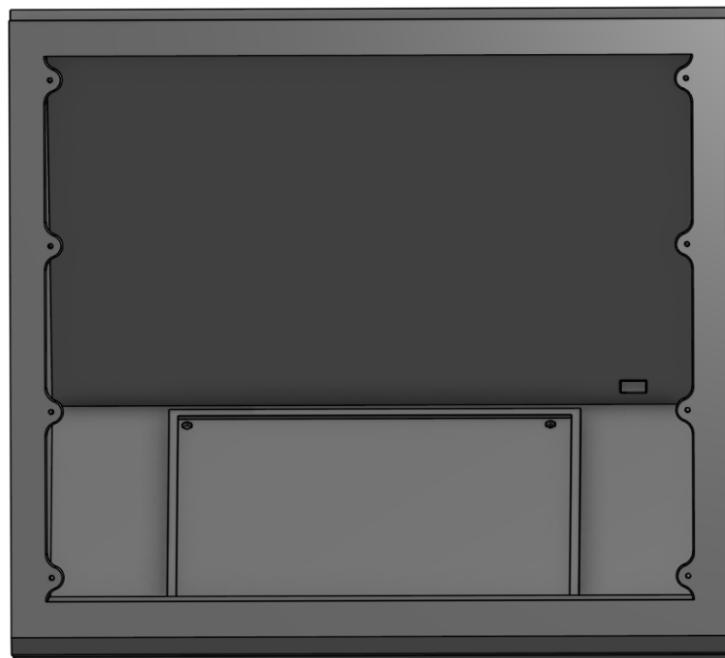


Vista posterior del panel frontal, en el cual se encuentra emplazada la luminaria e interruptores correspondientes para la representación visual y frontal de alertas, así como el ingreso manual de instrucciones básicas para con el módulo A.E.S. (activación manual, modo test, desactivación del módulo A.E.S). A su vez, cada zócalo alusivo a su correspondiente luminaria o interruptor encuentra escrita su función por medio de un grabado dentro del panel frontal. Se toma como base para el diseño de dicho panel el boceto presentado en el apartado de especificaciones.

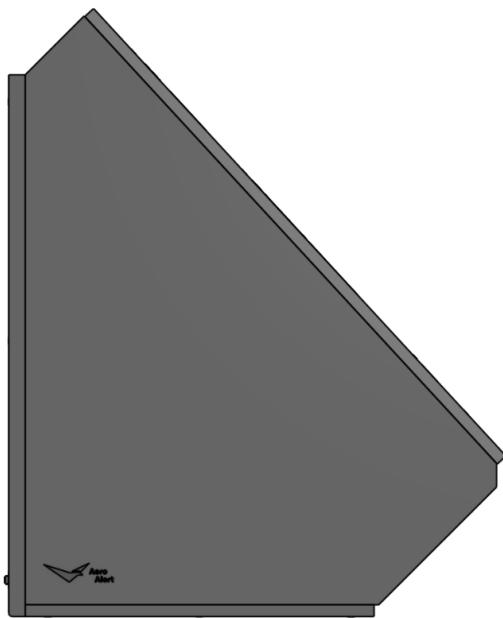


El panel frontal, entendido como una tapa con sus correspondientes agujeros destinados a alojar cada componente del mismo, incluyendo sobre ellos leyendas descriptivas, se encuentra estructuralmente separado del núcleo de la carcasa del módulo. Su unión a la misma es llevada a cabo por medio de

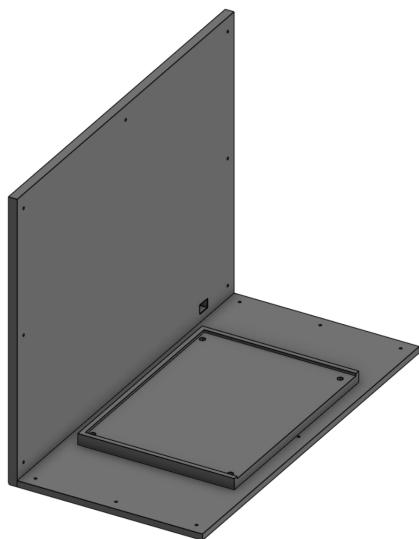
atornillamiento, destinando agujeros tanto en la tapa frontal como en el núcleo estructural para tal finalidad. En la siguiente imagen, se enseña el método de fijación propuesto para la unión entre el llamado núcleo y la tapa frontal (visto desde el núcleo). A su vez, se permite ver parcialmente el alojamiento interno de la placa madre del módulo A.E.S.



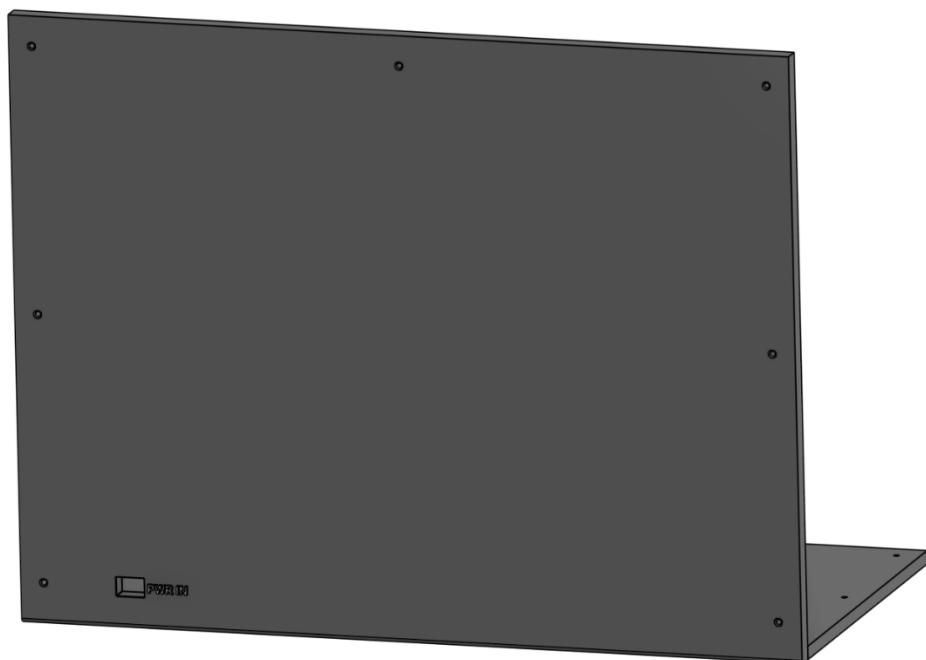
La siguiente imagen enseña una vista lateral del ensamblaje estructural del módulo en cuestión. A través de la misma, se puede visualizar el acabado resultante por la unión entre la tapa frontal y las tapas trasera e inferior para con el núcleo estructural del módulo. En la pared lateral derecha, se puede denotar un grabado con el logo del proyecto.



La siguiente ilustración enseña, con una vista en perspectiva, la disposición de las tapas trasera e inferior del módulo A.E.S. Se puede denotar en la tapa inferior un espacio en relieve destinado al alojamiento y sujeción de la placa madre del módulo A.E.S., con sus respectivos agujeros para atornillamiento de la plaqeta. A su vez, pueden notarse agujeros alrededor de los contornos de ambas tapas (trasera e inferior) con el propósito del posterior atornillamiento de las mismas hacia el núcleo estructural. Tomando como referencia frontal la cara predominantemente visible de la tapa trasera, se puede visualizar una muesca rectangular en el sector inferior derecho, destinado al pasaje del cable de alimentación externa descrito en las especificaciones electrónicas del módulo.

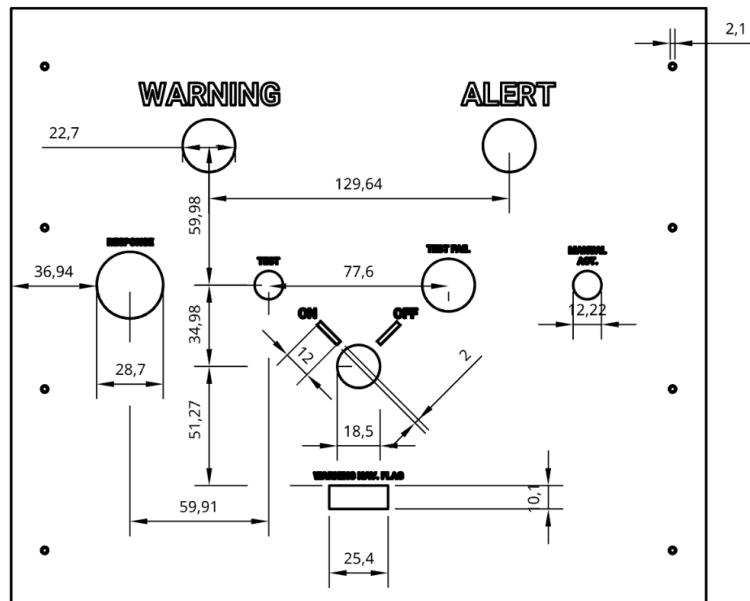
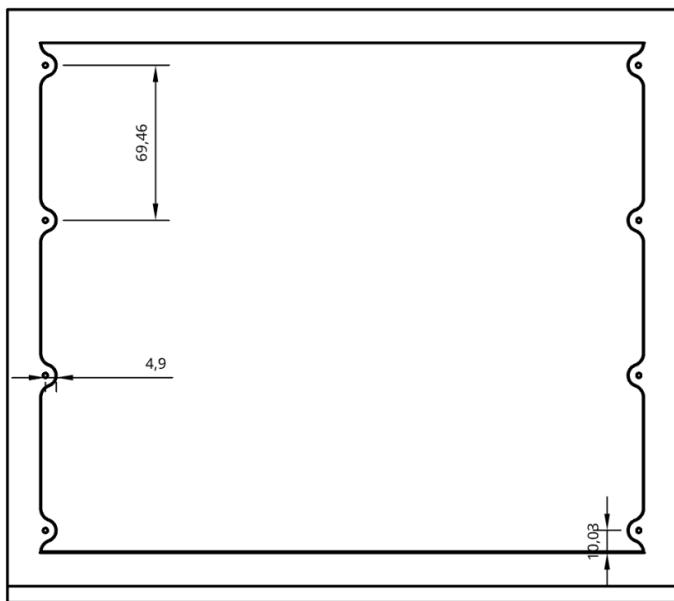
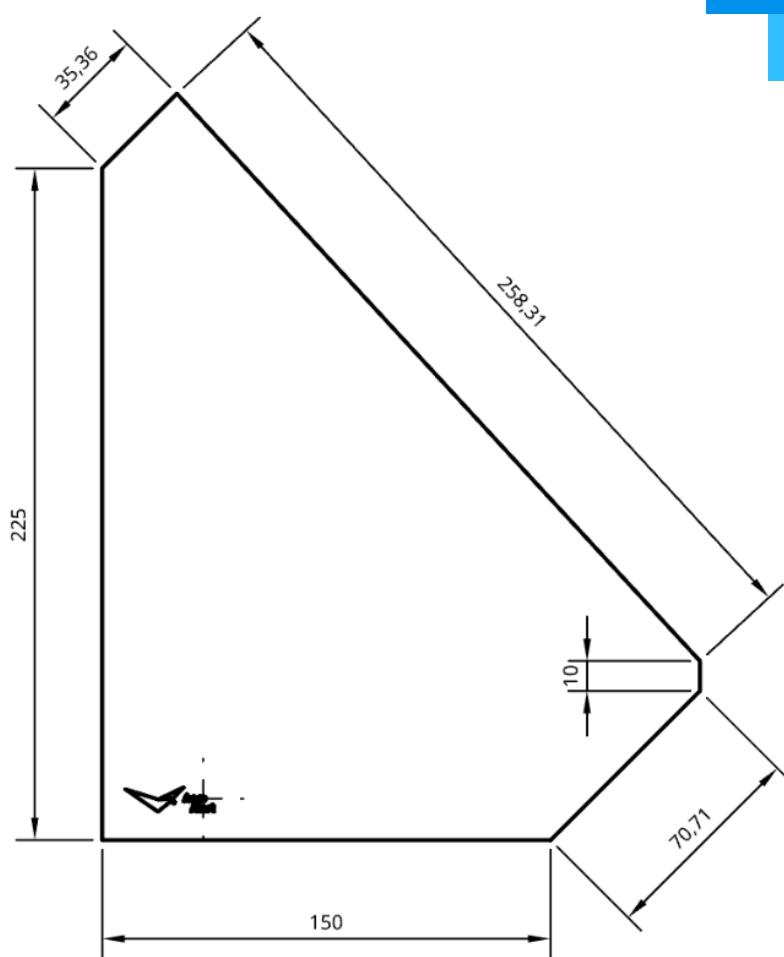


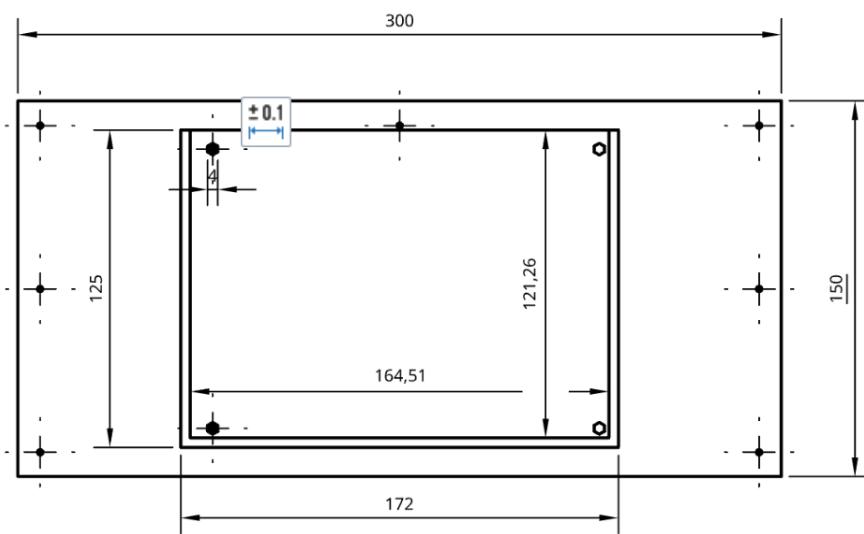
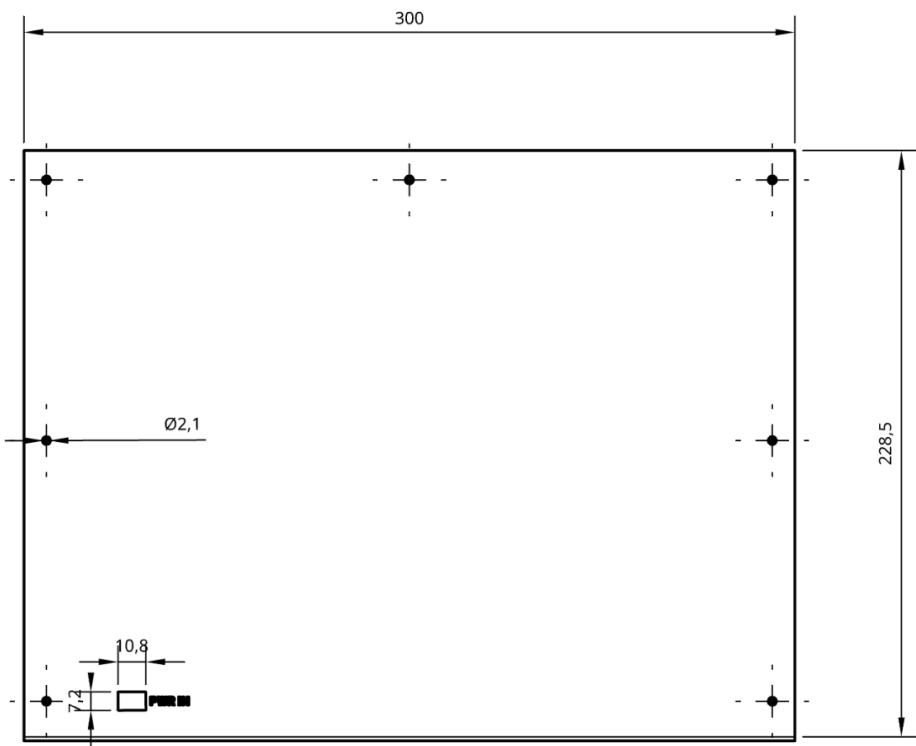
En la posterior imagen se puede obtener una vista predominante de la cara exterior de la tapa trasera. Como fue previamente descrito, se encuentran los agujeros lindantes al contorno de la tapa destinados a la fijación de la misma para con el núcleo estructural del equipo, así como la muesca rectangular presente en el sector inferior de la tapa. En esta vista, se puede denotar un grabado con la leyenda “PWR IN” alusiva a la entrada del cable de alimentación externa.



Para llevar a cabo el ensamblaje físico estructural del módulo A.E.S., se opta por la confección en madera tipo MDF (espesor 9mm). Una vez obtenida la pieza final modelada, se procede a exportar sus vistas exteriores a formato DXF, con el objetivo de prepararlas para el posterior ploteado por corte láser.

Las siguientes imágenes a presentar corresponden a los planos de las piezas que componen a la carcasa del módulo A.E.S, en sus distintas vistas de interés y con las acotaciones necesarias para el correcto entendimiento del diseño. Se destaca que toda medida presente en los planos se encuentra expresada en unidad de milímetros (mm).





Módulo A.E.S.:

Descripción del software

Panel de simulación virtual:

A modo de proveer un sistema prototipo completo, el proyecto en cuestión cuenta con un módulo de adquisición de datos sobre el piloto, un equipo de emplazamiento en cabina y una estación a disposición del control del tráfico aéreo. Sin embargo, dada la incapacidad de pruebas del sistema correspondiente al proyecto en un ámbito aeronáutico fidedigno (y, por lo tanto, la improbabilidad de aparición real de casos de anomalías de salud en base al ritmo cardíaco y saturación de oxígeno en sangre), se procede a confeccionar un panel de simulación virtual por medio del cual controlar, adaptar y seleccionar los distintos casos de interés para con el objetivo del proyecto a la hora de su simulación. A través del lenguaje Python, y haciendo uso de la librería Python Arcade, se llevó a cabo el maquetado y diseño de la Interfaz Gráfica de Usuario (G.U.I) a modo de facilitar posteriores simulaciones.



Las dos principales funciones del software en cuestión son aquellas destinadas a enseñar en pantalla la G.U.I (Graphical User Interface) y su control y la posterior transmisión de los datos ingresados por medio del protocolo WiFi a la placa de desarrollo ESP32 NodeMCU correspondiente al módulo A.E.S. (especificado posteriormente), con el objetivo de su consecutivo procesamiento. Para ello, se implementan las librerías ***arcade*** para activación de las llaves y botones, ***arcade.gui*** para formación de la interfaz, ***UISlider*** a fin crear las barras desplazables, ***UIManager***, ***UIAnchorWidget*** y ***UILabel*** con el propósito de decorar la interfaz y ***UIOnChangeEvent*** para detectar inputs del usuario.

Creación de objeto de tipo botón:

```
global Dicc
class Boton(arcade.gui.UITextureButton):
    def __init__(self, mensa_Al:str, mensa_BA:str):
        super().__init__(
            texture = arcade.load_texture("llave.png"),
            texture_pressed=arcade.load_texture("llave.png"))
        self.variable = True

        self.mensajeA = mensa_Al
        self.mensajeB = mensa_BA
        self.mensaje = self.mensajeA

        self.on_click = self.button_clicked
        self.scale(0.17)

    def On_button_on(self):
        if self.variable == True:
            self.texture = \
                arcade.load_texture("llave2.png")
            self.variable = False
            self.mensaje = self.mensajeB
        elif self.variable == False:
            self.texture = \
                arcade.load_texture("llave.png")
            self.variable = True
            self.mensaje = self.mensajeA
```

Creación de llaves:

```
self.boton4 = Boton("Normal som","Somnolencia")
self.boton4.on_click = self.boton_clicked
box.add(self.boton4)

self.boton1 = Boton("Pulso Alto ","Pulso Bajo ")
box.add(self.boton1)
self.boton1.on_click = self.boton_clicked1

self.boton2 = Boton("Normal ","Muerte ")
box.add(self.boton2)
self.boton2.on_click = self.boton_clicked2

self.boton3 = Boton("Normal hy ","Hipoxia ")
box.add(self.boton3)
self.boton3.on_click = self.boton_clicked3

self.boton5 = Boton("None","None")
box1.add(self.boton5)
self.boton5.on_click = self.boton_clicked4
```

Creación de objeto Barra de la G.U.I (GUI.py):

```
class Barra(UISlider):
    def __init__(self, valor:int, ValorM:int,
                 ValorMin:int, Width:int, Height:int, X:int):
        super().__init__(
            valor = valor,
            width = Width,
            height = Height,
            max_value= ValorM,
            min_value= ValorMin,
            size_hint_min=X
        )
        self.manager = UIManager()
        self.manager.enable()

        self.a = int(0)
        self.label = UILabel(text=f"{self.value:02.0f}")

    @self.event()
    def on_change(event: UIOnChangeEvent):
        self.label.text = f"{self.value:02.0f}"
        self.a = self.value
        self.label.fit_content()
        self.valor()

    def Label(self):
        return self.label

    def valor(self):
        return self.a
```

Creación de barras de la G.U.I:

```
self.barra1 = Barra(50, 100, 0, 300, 50,0)
self.Label1 = self.barra1.Label()

self.barra2 = Barra(75, 150, 0, 300, 50,0)
self.Label2 = self.barra2.Label()
```

Conexión a la Red WiFi del A.E.S. (StationPC.py):

```
1  from wireless import Wireless
2  import socket, json
3
4  # Conectarse a la red wifi llamada "ESP32" contraseña: "aeroalert"
5  def do_connect():
6      ssid = 'ESP32'
7      password = 'aeroalert'
8      wire = Wireless()
9      wire.connect(ssid, password)
10
11     server_ip = '192.168.4.1'
12     server_port = 8000
13     addr = socket.getaddrinfo(server_ip, server_port)[0][-1]
14
15     client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
16     client_socket.connect((server_ip, server_port))
17     print("conecto")
18     return client_socket
19
20 def send_type(tipo, client_socket):
21     message = json.dumps(tipo).encode('utf-8')
22     client_socket.send(message)
23
24 def send_message(client_socket, data):
25     try:
26         message = json.dumps(data).encode('utf-8')
27         client_socket.send(message)
28         print('enviado:', message)
29     except Exception as error:
30         print(f"error: {error} \n {data}")
```

Conexión haciendo uso de station (GUI.py):

```
def wifi():
    global client_socket
    print("intentando conectar")
    client_socket = station.do_connect()
    station.send_type("soy_PC", client_socket)
    print("tipo enviado")

    thread = threading.Thread(target=wifi, args=())
    # Inicia un proceso en paralelo para enviar informacion al SAE
    thread.start()
```

Confección del conjunto de datos a enviar (GUI.py):

```
class Diccionario():
    def __init__(self, Piloto:"0", Hipoxia:"0", Muerte:"0",
                 Somnolencia:"0", Pulso2:"0", Pulso:int, Saturacion:int):
        super().__init__()
        self.Piloto_valor = Piloto
        self.Hipoxia_valor = Hipoxia
        self.Muerte_valor = Muerte
        self.Somnolencia_valor = Somnolencia
        self.Pulso_valor = int(Pulso)
        self.Pulso_estado = Pulso2
        self.Saturacion_valor = int(Saturacion)
        self.Dic = {
            'Piloto' : self.Piloto_valor,
            'Muerte' : self.Muerte_valor,
            'Hipoxia' : self.Hipoxia_valor,
            'Somnolencia' : self.Somnolencia_valor,
            'Pulso' : self.Pulso_estado,
            'Bpm' : self.Pulso_valor,
            'SpO2' : self.Saturacion_valor
        }
        self.dicc = self.Dic
        print(self.Dic)

    def valor (self):
        return self.dicc
```

Repetido para cada botón:

```
def boton_clicked3(self, *_):
    self.boton3.On_button_on()
    self.Dicc = Diccionario("2", self.boton3.valor boton(), self.boton2.valor boton(),
                           self.boton4.valor boton(), self.valor2, self.valor1)
```

Enviar datos cuando se toca el botón 4 (se repite para cada piloto):

```
def button4_on(self):
    if self.variable4 == True:
        self.variable4 = False
    elif self.variable4 == False:
        self.variable4 = True

    dic = self.Dicc.Dic
    station.send_message(client_socket, dic)
```

Procesamiento de datos y sistema de alertas:

Corresponde al código principal de funcionamiento del equipo A.E.S, siendo aquel con interacción más cercana para con el hardware que compone al mismo, y teniendo como función la recepción y procesamiento de los datos enviados por el módulo V.D.B., el muestreo de alertas en función de dicha información y la posterior transmisión de información por medio del protocolo WiFi al equipo C.T.R.T. Para dicha ocasión, se optó por emplear el lenguaje MicroPython por medio del IDE Thonny, adaptado de manera comfortable para la programación orientada a microcontroladores.

El programa en cuestión se vale de la utilización de las librerías **socket** a modo de definir los puertos de recepción de datos, **gc** para evitar el almacenamiento innecesario de registros previos, **time** a fin de ejecutar el programa de manera tal que la placa de desarrollo no se sobrecargue, **json** para codificar y decodificar los mensajes enviados y recibidos, **_thread** para ejecutar funciones en paralelo, **Pin** con el objetivo de configurar los pines para cada luz y llave, **Timer** a modo de controlar el tiempo que se mantienen prendidas las luces y **UART** para enviar los datos necesarios a la PC del simulador en caso de ser necesario llevar a cabo un aterrizaje de emergencia.

```
import usocket as socket
import gc
gc.collect()
import time, json, _thread
import network
from machine import Pin, Timer, UART
```

Configuración de Pines:

```
def definir_pines():
    global pin_luz_ambar, pin_luz_roja, pin_luz_test, pin_flag
    global pin_activacion_manual, pin_test, pin_reaccion, pin_on_off
    global pin_boton_test, pin_boton_reaccion

    pin_luz_ambar = machine.Pin(17, machine.Pin.OUT)
    pin_luz_roja = machine.Pin(4, machine.Pin.OUT)
    pin_luz_test = machine.Pin(14, machine.Pin.OUT)
    pin_flag = machine.Pin(26, machine.Pin.OUT)
    pin_activacion_manual = machine.Pin(21, machine.Pin.IN)
    pin_on_off = machine.Pin(35, machine.Pin.IN)

    pin_test = machine.Pin(18, machine.Pin.IN)
    pin_boton_test = pin_test.value()

    pin_reaccion = machine.Pin(19, machine.Pin.IN)
    pin_boton_reaccion = pin_reaccion.value()

definir_pines()
```

Configuración de WiFi:

```
def conectar_wifi():
    global s, ap
    # Configuracion
    addr = socket.getaddrinfo('192.168.4.1', 8000)[0][-1]
    ssid = 'ESP32'
    password = 'aeroalert'

    ap = network.WLAN(network.AP_IF)
    ap.active(True)
    ap.config(essid=ssid, password=password)
    ap.config(authmode=3)
    while ap.active() == False:
        pass
    print('Connection succesful')
    print(ap.ifconfig())

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind(('', 8000)) #o addr
    s.listen(2)
    print("listening on",addr)
    return s
```

Asignar usuarios a sus funciones:

```
def escuchar_tipos():
    time.sleep(1)
    global s, ap, addr
    print("Escuchando tipos")
    while True:
        conn, addr = s.accept()
        print('Got a type connection from %s' % str(addr))
        message = conn.recv(1024)
        tipo = json.loads(message.decode('utf-8'))

        if tipo == "soy_band":
            _thread.start_new_thread(escuchar_band, (conn, addr))
            print("VDB conectada")

        elif tipo == "soy_rtdc":
            _thread.start_new_thread(escuchar_rtdc, (conn, addr))
            _thread.start_new_thread(enviar_rtdc, (conn, addr))
            print("CTRT conectada")

        elif tipo == "soy_PC":
            _thread.start_new_thread(escuchar_PC, (conn, addr))
            _thread.start_new_thread(enviar_PC, (conn, addr))
            print("PC/X-PLANE conectado")
```

Recepción de datos provenientes del módulo V.D.B:

```
def escuchar_band(conn_band, addr):
    time.sleep(1)
    while True:
        message = conn_band.recv(1024)
        print('From Band %s' % str(addr))

        data = json.loads(message.decode('utf-8'))#.decode('utf-8')
        bpm = data['1']
        spo = data['2']
        temp = data['3']
        conectado = data['4']
        print("bpm={:02} spo={:02}% Temp={:02}°C puesta={:1}".format(bpm, spo, temp,
evaluar_info(bpm, spo, temp, conectado, "band"))
```

Recepción de datos provenientes de C.T.R.T:

```
aterrizar = aterrizar_manual = 0
def escuchar_rtdc(conn_rtdc,addr):
    global aterrizar
    pin_flag.value(0)
    try:
        while True:
            data = conn_rtdc.recv(1024)
            print('From RTDC %s' % str(addr))

            message = json.loads(data.decode('utf-8'))
            print(message)

            if message['mensaje'] == "aterriza":
                print("tiene que aterrizar")
                aterrizar = 1
            if message['mensaje'] == "no aterrizar":
                print("no tiene que aterrizar")
                aterrizar = 0
    except:
        pin_flag.value(1)
        # Recibe los comandos de vuelo enviados por la rtdc
```

Envío de datos y petición de aeropuerto a C.T.R.T:

```
def enviar_rtdc(conn, addr):
    global codigo, solicitar, info_aeropuerto
    alerta_enviada = 0
    solicito_aterrizaje = json.dumps("solicito aterrizaje").encode('utf-8')
    alerta_desactivacion_sae = json.dumps("alerta desactivacion del sae").encode('utf-8')
    sae_activado = json.dumps("Sae activado").encode('utf-8')
    try:
        while True:
            print("enviando a RTDC: ", addr)

            if solicitar == 1:
                conn.send(solicito_aterrizaje)
                solicitar = 0
                info_aeropuerto = conn.recv(1024)

                codigo = actualizar_codigo()
                codigo_enviar = json.dumps(codigo).encode('utf-8')
                conn.send(codigo_enviar)
                if pin_on_off.value() == 1 and alerta_enviada == 0:
                    codigo_enviar = alerta_desactivacion_sae
                    conn.send(codigo_enviar)
                    alerta_enviada = 1
                elif pin_on_off.value() == 0 and alerta_enviada == 1:
                    codigo_enviar = sae_activado
                    conn.send(codigo_enviar)
                    alerta_enviada = 0
                time.sleep(10)
    except:
        print("rtdc perdida")
```

Recepción de datos de la interfaz gráfica en PC:

```
def escuchar_PC(conn_PC, addr):
    global bloqueo_PC
    global dormido1, spo_bajos1, bpm_altos1, muerte1
    estados = {"Piloto1": {"Somnolencia": 0, "Pulso": 0, "Hipoxia": 0,
                           "Muerte": 0, "Spo2": 0, "Bpm": 0},
                "Piloto2": {"Somnolencia": 0, "Pulso": 0, "Hipoxia": 0,
                           "Muerte": 0, "Spo2": 0, "Bpm": 0}}
    bloqueo_PC = 0
    while True:
        time.sleep(3)
        if pin_on_off.value() == 1:
            pass
        else:
            message = conn_PC.recv(1024)
            print()
            print('Got a connection from PC %s' % str(addr))
            info_PC = json.loads(message.decode('utf-8'))
            print(info_PC)

            if info_PC == "1":
                if bloqueo_PC == 0:
                    bloqueo_PC = 1
                elif bloqueo_PC == 1:
                    bloqueo_PC = 0

                if estados["Piloto1"]["Bpm"] != 0 and estados["Spo2"] != 0:
                    bloqueo_PC = 1
                    evaluar_info(estados["Piloto1"]["Bpm"],
                                 estados["Piloto1"]["Spo2"], 15, 1, "PC")
                else:
                    evaluar_info_piloto1(estados["Piloto1"])

                if estados["Piloto1"]["Somnolencia"] == '1':
                    dormido1 = 1
                else:
                    dormido1 = 0

                evaluar_info_piloto2(estados["Piloto2"])

            elif info_PC["Piloto"] == '1':
                info_PC.pop("Piloto")
                estados["Piloto1"] = info_PC
            elif info_PC["Piloto"] == '2':
                info_PC.pop("Piloto")
                estados["Piloto2"] = info_PC
```

Envío de instrucciones para X-Plane a PC:

```
def enviar_PC(conn, addr):
    global aterrizar, aterrizar_manual, solicitar, info_aeropuerto
    enviado = 0
    info_aeropuerto = 0
    aterrizar = json.dumps("ATERRIZAR").encode('utf-8')
    no_aterrizar = json.dumps("NO ATERRIZAR").encode('utf-8')
    while True:
        # pedir permiso para aterrizar antes a rtde
        if aterrizar == 1 and enviado == 0 or aterrizar_manual == 1 and enviado == 0:
            solicitar = 1
            print("aterrizar a PC: ", addr)
            conn.send(aterrizar)
            enviado = 1
        while True:
            if info_aeropuerto != 0:
                conn.send(json.dumps(info_aeropuerto).encode('utf-8'))
                info_aeropuerto = 0
                break
        elif aterrizar_manual == 0 and aterrizar == 0 and enviado == 1:
            print("no aterrizar a PC: ", addr)
            conn.send(no_aterrizar)
            enviado = 0
            time.sleep(5)
```

Envío de instrucciones de alarmas sonoras a PC:

```
if alarma_sonora_1:
    print("prender alarma_sonora_1")
    sonora1_enviada = 1
elif not alarma_sonora_1 and sonora1_enviada:
    print("apagar alarma_sonora_1")
    sonora1_enviada = 0

if alarma_sonora_2:
    print("prender alarma_sonora_2")
    sonora2_enviada = 1
elif not alarma_sonora_2 and sonora2_enviada:
    print("apagar alarma_sonora_2")
    sonora2_enviada = 0
```

Protocolos de alarmas audibles y luces:

```
while True:
    time.sleep(2)
    if pin_on_off.value() == 1:
        pin_luz_ambar.value(0)
        pin_luz_roja.value(0)
        pass
    else:
        codigo = actualizar_codigo()
        print(codigo)
        print()
        # Luz roja titilar cuando activacion manual
        if codigo[14] == 1:
            if prendido == 0:
                aterrizar_manual = 1
                pin_luz_roja.value(1)
                prendido = 1
            elif prendido == 1:
                pin_luz_roja.value(0)
                prendido = 0
            else:
                aterrizar_manual = 0
                if prendido == 1:
                    pin_luz_roja.value(0)
                    prendido = 0

        # Boton tipo switch
        # Si el boton de reaccion cambio de valor no va a val
        if pin_reaccion.value() != tocado:
            tocado = pin_reaccion.value()
            pin_boton_reaccion = 1 # Pone en 1
```

Protocolos de hipoxia:

El mismo fragmento se repite para la detección de ritmo cardíaco anormal, así como para la detección de somnolencia.

```
# Protocolo hipoxia-----
if codigo[6] and codigo[7]:                                # Si ambos tienen hipoxia
    print("2 spo")
    if alarmas_off_spo == 0:                                 # Si las alarmas no estan desactivadas
        pin_luz_roja.value(1)                               # Activa luz alarma (hipoxia)
        alarma_sonora_1 = 1

        # Si el piloto toca el boton de reaccion desactiva las alarmas, no deja que tomen el control
        # Tmb inicia un contador de 60seg que estara sin las alarmas
        if pin boton_reaccion == 1:                         # Si el boton de reaccion fue presionado
            alarmas_off_spo = 1                            # Las alarmas de spo2 se desactivan
            tomar_control = 0                            # Se pone en 0 el pin de tomar el control
            if contador_iniciado_60_spo != 1:             # Si el contador de 60s spo2 no esta iniciado
                t60spo.init(mode=Timer.ONE_SHOT, period=60000, callback=contador60spo) #Lo inicia
                contador_iniciado_60_spo = 1               # Cambia la variable para que la prox sepa que est

            # Inicia contador 30 segs para definir hipoxia peligrosa
            elif contador_iniciado_30_spo != 1:           # Sino si el contador de 30s spo2 no esta iniciado
                t30spo.init(mode=Timer.ONE_SHOT, period=30000, callback=contador30spo)
                contador_iniciado_30_spo = 1               # Pone la variable en 1 cont_init_30spo

            # 30 segs despues de tener hipoxia y no tener reaccion deja que tomen el control
            elif pasaron_30segsspo == 1:                  # Sino si pasaron30segsspo2 esta en 1
                contador_iniciado_30_spo = 0              # Pone en 0 la variable de cont_init_30spo
                if pin boton_reaccion != 1:                # Si el pin de reaccion no es 1
                    tomar_control = 1                     # Pone pin tomar control en 1

            elif alarmas_off_spo == 1:                   # Sino si estan desactivadas las alarmas spo
                pin_luz_roja.value(0)                      # Apaga luz alarma
                pass

        elif codigo[6] or codigo[7]:                  # Sino si 1 tiene spo2 en 1
            print("1 spo")
            pin_luz_roja.value(1)
            alarma_sonora_1 = 0

    elif not codigo[6] and not codigo[7]:          # Sino si ninguno tiene spo2 en 1
        print("no spo")
        pin_luz_roja.value(0)
        alarma_sonora_1 = 0
```

Casos de somnolencia, muerte de pilotos o desconexión de V.D.B:

```
if codigo[4] or codigo[5]:
    if ambar_titilando == 0:
        pin_luz_ambar.value(1)

    if codigo[4] and codigo[5]:      #si
        print("2 dormidos")
        # 2dormidos hacer coso de 30 s
    else:
        print("1 dormido")
else:
    print("0 dormidos")

if pulsera_conectada == 0:
    if ambar_titilando == 0:
        pin_luz_ambar.value(1)
        print("pulsera mal")
    else:
        print("pulsera bien")
#-----

if codigo[12] and codigo[13]:
    print("ambos muertos")
    pin_luz_roja.value(1)
elif codigo[12] or codigo[13]:
    print("uno muerto")
    #pin_luz_roja.value(0)
    pin_luz_ambar.value(1)
    ambar_fija = 1
else:
    print("todos vivos")
    #pin_luz_roja.value(0)
```

Evaluación de datos provenientes del módulo V.D.B para determinación de somnolencia:

```
def evaluar_info(bpm, spo, temp, conectado, de):
    global bpm_bajos1, bpm_altos1, spo_bajos1, dormido1, temp_baja1, temp_alta1, muerte1
    global listabpm, listaspo          #se usa en distintos threads por eso global
    global bloqueo_PC
    global pulsera_conectada

    if bloqueo_PC == 0:           # se evalua la info si es de band sin bloqueo
        #Listas de pulsaciones y oxigeno
        #
        pulsera_conectada = conectado
        listabpm.append(bpm)
        listabpm = listabpm[-48:]
        listaspo.append(spo)
        listaspo = listaspo[-48:] #12 valores cada uno cada 5 segs son 60 segs en total

        #suma de primeros 7 digitos de la lista
        bpm_prom_inicial = sum(listabpm[0:7])
        #suma de los ultimos 8 digitos de la lista
        bpm_prom_actual = sum(listabpm[:-8:-1])
        #diferencia entre ambos
        bpm_dif = bpm_prom_actual - bpm_prom_inicial

        spo_prom_inicial = sum(listabpm[0:7])
        spo_prom_actual = sum(listabpm[:-8:-1])
        spo_dif = spo_prom_actual - spo_prom_inicial

        if spo_dif >= 3 and spo_dif <= 8:
            if bpm_dif >= 15 and bpm_dif <= 35:
                print("tiene menos pulsaciones y oxigeno que hace un ratito, suponemos que esta dormido")
                dormido1 = 1
                bpm_dormido = bpm
                spo_dormido = spo

        if dormido1 == 1:
            bpm_dormido_dif = bpm - bpm_dormido
            spo_dormido_dif = spo - spo_dormido
            if bpm_dormido_dif >= 20:
                if spo_dormido_dif >= 3:
                    print("esta despierto ahora")
                    dormido1 = 0
```

Evaluación de datos provenientes de V.D.B para determinar casos de hipoxia y nivel anormal de ritmo cardíaco :

```
#bpms
if bloqueo_PC == 0 or de == "PC":
    if bpm < 60:
        bpm_bajos1 = 1
        bpm_altos1 = 0
    elif bpm > 140:
        bpm_altos1 = 1
        bpm_bajos1 = 0
    else:
        bpm_bajos1 = 0
        bpm_altos1 = 0

    if bpm == 0:
        muerte1 = 1
    elif bpm != 0:
        muerte1 = 0

#spo
if spo <= 90:
    spo_bajos1 = 1
elif spo > 90:
    spo_bajos1 = 0
```

Evaluación de datos provenientes de V.D.B para determinar sobrecalentamiento o congelamiento del sensor :

```
#segun temperaturas:
if temp < 5:
    temp_baja1 = 1
    temp_alta1 = 0
elif temp > 40:
    temp_alta1 = 1
    temp_baja1 = 0
else:
    temp_baja1 = 0
    temp_alta1 = 0
```

Inicialización de procesos en paralelo para la comunicación con la C.T.R.T, V.D.B, PC, control de luces y alarmas en cabina:

```
s = conectar_wifi()
print("wifi conectado")

_thread.start_new_thread(activar_SAE, ())
print("protocolos activados")
time.sleep(1)

_thread.start_new_thread(escuchar_tipos, ())
print("Comunicación activada")
```

Sistema de alertas sonoras:

El equipo A.E.S provee de alertas sonoras, las cuales son ejecutadas mediante un programa escrito en lenguaje Python, y alojado en la computadora principal de simulación. Por lo tanto, la transmisión de información para con tal ordenador (a través de la cual gestionar el encendido de las diversas alertas sonoras) es efectuada haciendo uso del protocolo UART.

El código encargado de la recepción de información por medio del protocolo UART así como su procesamiento y la consecutiva generación de alertas sonoras se encuentra desarrollado con el lenguaje Python mediante el programa Visual Studio Code. Por otro lado, el código dedicado a la transmisión de información haciendo uso del protocolo UART se encuentra realizado en el lenguaje MicroPython mediante el programa Thonny, con el propósito de ser alojado en la memoria del microcontrolador ESP32 del módulo A.E.S. (y, por lo tanto, ser corrido en la misma unidad). Se hace uso de las librerías **serial** para recepción de mensajes por UART, **playsound** para reproducción de alarmas, **time** a modo de gestionar la espera a repetir el sonido de alarma y **xplane** con el propósito de enviar notificaciones a gestionar para con el programa X-Plane 11.

```

ser = serial.Serial(
    port='/dev/ttyACM0',
    baudrate=115200,
    timeout=0
)
print("connected to: " + ser.portstr)
prendida1 = prendida2 = 0
while True:
    leer = ser.readline()
    read = leer.decode('utf-8')
    if read:
        print(read)
        if read == "apagar alarma_sonora_1":
            prendida1 = 0
        elif read == "prender alarma_sonora_1" or prendida1:
            playsound("alarma_1.mp3")
            prendida1 = 1
            time.sleep(2)
        elif read == "apagar alarma_sonora_2":
            prendida2 = 0
        elif read == "prender alarma_sonora_2" or prendida2:
            playsound("alarma_2.mp3")
            prendida2 = 1
            time.sleep(2)
        elif read == "aterrizar":
            xplane.send(1)
        elif read == "no aterrizar":
            xplane.send(0)

```

Simulación con el programa X-Plane 11

Con el objetivo de lograr una simulación de los diferentes casos para los cuales el proyecto busca generar cobertura, así como la respuesta en vuelo que se busca que la aeronave sea capaz de sufrir en cuanto a su maniobrabilidad, se opta por trabajar con el programa simulador de vuelo Xplane 11.

En relación a la comunicación que se busca que el programa simulador de vuelo genere para con el restante sistema prototipo, existen dos principales programas a modo de solventar tal objetivo: el primero en cuestión se enfoca



en las distintas maniobras y configuraciones de vuelo que debe realizar la aeronave. El mismo esta realizado en el lenguaje de programación LUA (más precisamente, una variante de dicho lenguaje adaptada a simuladores de vuelo, conocida como FlyWithLua) y se encuentra alojado en la computadora principal de simulación. El segundo código se enfoca en la recepción de información proveniente de la placa de desarrollo ESP32 por medio del protocolo UART, en donde el es la placa quien asume el rol de enviar información según el contexto de vuelo a simular. Este código esta desarrollado en MicroPython con el IDE Thonny.

Recepción instrucciones de aterrizaje:

```
prey_sound("alarma.mp3")  
prendida2 = 1  
time.sleep(2)  
elif read == "aterrizar":  
    xplane.send(1)  
elif read == "no aterrizar":  
    xplane.send(0)
```

Código de control de variables de simulación de vuelo (safereturn.lua)

Función para establecer aeronave en emergencia:

```

function safereturn()
    play_sound(autolandone)

    -- squawk 7700 and mode C
    dataref("sqkCode", "sim/cockpit2/radios/actuators/transponder_code", "writable")
    sqkCode = 7700
    dataref("sqkMode", "sim/cockpit2/radios/actuators/transponder_mode", "writable")
    sqkMode = 2

    -- QNH must be accurate
    pilotQNH = XPLMFindDataRef("sim/cockpit2/gauges/actuators/barometer_setting_in_hg_pilot")
    actualQNH = XPLMFindDataRef("sim/weather/barometer_sealevel_inhg")
    XPLMSetDataf(pilotQNH, XPLMGetDataf(actualQNH))

    -- wing level
    command_once("sim/autopilot/wing_leveler")

    -- fd ap on
    command_once("sim/autopilot/fdir_on")
    command_once("sim/autopilot/servos_on")

    -- check we're in GPS mode
    dataref("CDIVLOC", "sim/cockpit2/radios/actuators/HSI_source_select_pilot", "writable")
    CDIVLOC = 2

    dataref("altitude", "sim/cockpit2/autopilot/altitude_readout_preselector", "writable")
    -- DESCEND ON CURRENT HEADING

    -- set altitude - aim for 1200 but will stop at 1200agl. this should get us in on most glideslopes.
    local ln_alt = dataref_table("sim/cockpit/autopilot/altitude", "writable")
    ln_alt[0] = 1200
    local ln_vs = dataref_table("sim/cockpit/autopilot/vertical_velocity", "writable")
    if altitude < 4200 then
        ln_vs[0] = -1500
    elseif altitude > 4200 then
        ln_vs[0] = -1800
    end
    command_once("sim/autopilot/vertical_speed_pre_sel")

    -- AT ON
    dataref("ATON", "sim/cockpit2/autopilot/autothrottle_enabled", "writable")
    ATON = 1

    -- SET DESCENT SPEED 165KIAS
    dataref("SPEEDSET", "sim/cockpit/autopilot/airspeed", "writable")
    SPEEDSET = 165

    function checkAlt()
        alt = XPLMGetDataf(XPLMFindDataRef("sim/flightmodel/position/y_agl"))
        ground = alt * 3.2808399
        if (ground < 1200 and fired == false) then
            fired = true
            command_once("sim/autopilot/altitude_hold")
            phasetwo()
        end
    end

    do_often("checkAlt()")
end

```

Función para generar ruta de aproximación ILS:

```
function phasetwo()
    dataref("FL_CURR_ALT", "sim/cockpit2/autopilot/altitude_readout_preselector")
    inCurr_Alt = math.floor(FL_CURR_ALT)
    -- WHEN DESCENT DONE
    play_sound(autolandtwo)
    -- 128KIAS
    SPEEDSET = 128
    -- put flaps down
    command_once("sim/flight_controls/flaps_down")
    -- gear down
    command_once("sim/flight_controls/landing_gear_down")
    -- check ILS is valid (if not, keep checking) and run distance check
    function checkILS()
        -- get my lat lon
        lat = XPLMGetDataf(XPLMFindDataRef("sim/flightmodel/position/latitude"))
        lon = XPLMGetDataf(XPLMFindDataRef("sim/flightmodel/position/longitude"))
        dataref("altitude", "sim/cockpit2/autopilot/altitude_readout_preselector", "writable")
        local ln_vs = dataref_table("sim/cockpit/autopilot/vertical_velocity", "writable")
        function altitude_check()
            if altitude < 4200 then
                ln_vs[0] = -1500
                if altitude < 3000 then
                    ln_vs[0] = -1000
                elseif altitude < 2000 then
                    ln_vs[0] = -700
                elseif altitude < 1500 then
                    ln_vs[0] = -500
                elseif altitude < 1000 then
                    ln_vs[0] = -150
                end
            elseif altitude > 4200 then
                ln_vs[0] = -1800
            end
        end
        do_often("altitude_check()")
        -- find closest ils and store so we can retrieve freq
        local airport_ILS
        --> --> --> _, airport_ILS, _ = XPLMGetNavAidInfo( XPLMFindNavAid( nil, nil, lat, lon, nil, xplm_Nav_ILS ) )
        -- find closest ILS freq
        local ILS_freq
        one, airLat, airLon, four, ILS_freq, six, seven, _ = XPLMGetNavAidInfo( XPLMFindNavAid( nil, airport_ILS, lat, lon, nil, xplm_Nav_ILS ) )
        if (airport_ILS ~= "NTFND" and firedILS == false) then
            firedILS = true
            -- tune radio
            dataref("NAVONE", "sim/cockpit/radios/nav1_freq_hz", "writable")
            dataref("NAVTWO", "sim/cockpit/radios/nav1_stby_freq_hz", "writable")
            NAVONE = ILS_freq
            NAVTWO = ILS_freq
            -- navigate to the ILS
            XPLMClearFMSEntry(0)
            XPLMSetFMSEntryInfo(0, XPLMFindNavAid( nil, airport_ILS, nil, nil, ILS_freq, xplm_Nav_ILS ), 2000)
            XPLMSetDestinationFMSEntry(0)
            command_once("sim/autopilot/NAV")
            logMsg(airport_ILS)
            -- run distance check
            checkDist()
            if (firedILS == true) then
                command_once("sim/GPS/g1000n1_hdg")
            end
        elseif(airport_ILS == "NTFND") then
            logMsg("ILS Route Not Found")
        end
    end
    do_often("checkILS()")
end
```

Función de aterrizaje:

```
function land()

    -- APPROACH PHASE

    -- switch to radio beacon nav for ILS
    CDIVLOC = 0
    ln_alt = dataref_table("sim/cockpit/autopilot/altitude", "writable")
    ln_alt[0] = 1200
    ln_vs = dataref_table("sim/cockpit/autopilot/vertical_velocity", "writable")
    if dist >= 5 then
        SPEEDSET = 110
        command_once("sim/GPS/g1000n1_apr")
    elseif dist < 5 then
        SPEEDSET = 80
    end
    -- wait for land
    function checkLand()
        altland = XPLMGetDataf(XPLMFindDataRef("sim/flightmodel/position/y_agl"))
        groundland = altland * 3.2808399
        vertfpm = XPLMGetDataf(XPLMFindDataRef("sim/flightmodel/position/vh_ind_fpm"))
        dist = XPLMGetDataf(XPLMFindDataRef("sim/cockpit2/radios/indicators/gps_dme_distance_nm"))
        dataref('trk', 'sim/cockpit2/autopilot/trk_fpa')
        -- all has gone well, cut engine at touchdown
        if (groundland < 25 and firedThree == false) then
            firedThree = true
            rollout()
            logMsg("Great Landing!")
        -- pops chute if near airport but not on GS
        elseif (dist < 1 and groundland > 500 and vertfpm > -50 and firedThree == false) then
            firedThree = true
            caps()
            logMsg("Near airport but not on GS")
        -- if it starts flying away from the airport (downwind ils etc)
        elseif (dist > 12 and firedThree == false) then
            firedThree = true
            caps()
            logMsg("Flying away from airport")
        end
    end
    do_often("checkLand()")
end
```



El accionamiento de las líneas de código escritas en lenguaje LUA es desarrollado por medio de la exportación de las instrucciones implícitas a un comando dentro del programa simulador de vuelo. El control tanto de lectura como escritura de variables y comandos dentro del programa es capaz de ser llevado a cabo por medio del protocolo de red UDP dentro del mismo ordenador. Los fragmentos del siguiente código corresponden al programa que permite tanto la lectura y escritura de variables (datarefs), así como el accionamiento de comandos. El mismo implementa las librerías **socket** para establecer la comunicación UDP y **struct** a fin de traducir valores decimales a arreglos interpretables por la ruta a transmitir al programa y viceversa.

Generación de conexión UDP con el programa simulador de vuelo:

```
class UDPSocket():

    def __init__(self):
        self.socket = socket(AF_INET,SOCK_DGRAM)
        self.localIP = "127.0.0.1"
        self.localPort = 49004
        self.bufferSize = 1024
        self.socket.bind((self.localIP, self.localPort))

    def loop(self):
        print("Server listening")

        #print("Before receiving data")
        bytesAddressPair = self.socket.recvfrom(self.bufferSize)
        #print("After receiving data")

        messages = bytesAddressPair[0]
        address = bytesAddressPair[1]

        #print("Received message:", messages)
        #print("Source address:", address)
        return messages

    def udp_cliente(self, IP, port, message):

        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        print (message)
        sock.sendto(message, (IP, port))
```

Armado y descomposición de paquetes de datos para posterior lectura y escritura:

```
def DREF (self, message, value):
    buf = bytearray(507)
    buf [0:3] = bytes('DREF', 'ascii')
    buf [5:8] = bytearray(struct.pack("f", value))
    buf[9:(9+len(message))] = bytes(message, 'ascii')
    return buf
```

```
def DREFtranslate (self, messages):
    hdgmode = "sim/cockpit2/autopilot/heading_mode"
    gpscourse = "sim/cockpit/radios/gps_course_degtm"
    navcourse = "sim/cockpit/radios/nav1_course_degdm"
    maghdg = "sim/cockpit/autopilot/heading_mag"
    autostate = "sim/cockpit/autopilot/autopilot_state"

    if hdgmode in str(messages):
        buf = bytearray(messages)
        val = struct.unpack('f', buf [5:9])
        value = val[0]
        return [hdgmode, value]
    elif gpscourse in str(messages):
        buf = bytearray(messages)
        val = struct.unpack('f', buf [5:9])
        value = val[0]
        return [gpscourse, value]
    elif navcourse in str(messages):
        buf = bytearray(messages)
        val = struct.unpack('f', buf [5:9])
        value = val[0]
        return [navcourse, value]
    elif maghdg in str(messages):
        buf = bytearray(messages)
        val = struct.unpack('f', buf [5:9])
        value = val[0]
        return [maghdg, value]
    elif autostate in str(messages):
        buf = bytearray(messages)
        val = struct.unpack('f', buf [5:9])
        value = val[0]
        return [autostate, value]
```

Lectura de datarefs y posterior accionamiento de comandos:

```
s = UDPsocket()
i = 1
while True:
    APST = 0
    APHDG = 0
    GPSCR = 0
    NAVCRS = 0
    MHDG = 0
    hdgmode = "sim/cockpit2/autopilot/heading_mode"
    gpscourse = "sim/cockpit/radios/gps_course_degtm"
    navcourse = "sim/cockpit/radios/nav1_course_degm"
    maghdg = "sim/cockpit/autopilot/heading_mag"
    autostate = "sim/cockpit/autopilot/autopilot_mode"
    a = s.loop()
    b = s.DREFtranslate(a)
    if hdgmode == b[0]:
        APHDG = int(b[1])
        #print(APHDG)
    elif gpscourse in b:
        GPSCR = int(b[1])
        #print(GPSCR)
    elif navcourse in b:
        NAVCRS = int(b[1])
        #print(NAVCRS)
    elif maghdg in b:
        MHDG = int(b[1])
        #print(MHDG)
    elif autostate in b:
        APST = int(b[1])
        #print(APST)

    if APST == 2:
        if i == 1:
            a = s.CMND('sim/autopilot/heading')
            s.udp_cliente('127.0.0.1', 49000, a)
            i = 2

        elif NAVCRS - GPSCR < 3:
            if i == 2:
                a = s.CMND('sim/autopilot/approach')
                s.udp_cliente('127.0.0.1', 49000, a)
                i = 3
```

A modo de obtener una visión y comprensión completa del software en cuestión, recomendamos visitar nuestro repositorio en GitHub:
github.com/impatrq/aeroalert



Módulo C.T.R.T.: **Especificaciones**

Lineamientos Generales:

El módulo C.T.R.T. corresponde al equipo dispuesto en tierra, con el objetivo de ser administrado por el control del tráfico aéreo. El propósito de dicho equipo es el de recibir, representar visualmente e informar a los controladores aéreos de emergencias, alertas y/o peticiones. Además es el encargado de generar y transmitir instrucciones al equipo en cabina (módulo A.E.S) con el objetivo de concretar un aterrizaje de emergencia en caso de ser necesario este último.

El módulo C.T.R.T. presenta una pantalla por medio de la cual enseña una interfaz gráfica con información del estado de los vuelos circundantes (aeronave involucrada, estado de salud de los pilotos en función de las variables relevadas por el dispositivo V.D.B a portar por ellos, la actualización de comandos efectuados y un historial de los mismos), así como un panel frontal en el cual se dispone de un par de indicadores luminosos cuyo propósito es el de relevar las situaciones de emergencia y/o alerta que sean capaces de presentar las aeronaves circundantes en el espacio aéreo de cobertura, y un teclado matricial por medio del cual ingresar instrucciones y navegar a través de la interfaz gráfica y la información que ella presenta.



Módulo C.T.R.T: Descripción del Hardware

A la hora de diagramar el conexionado y funcionamiento electrónico del módulo C.T.R.T., las principales necesidades y desafíos de diseño a cumplir fueron la capacidad de emular una estación aeronáutica computarizada, con capacidad de representar datos embebidos en una interfaz gráfica, así como el ingreso de comandos para con el programa simulador de vuelo y el módulo A.E.S. y la capacidad de comunicación inalámbrica tanto para recepción de las variables de salud adquiridas por el módulo V.D.B. como de aquellos parámetros ingresados por medio del panel frontal del equipo A.E.S, así como el envío de comandos que puedan ser posteriormente procesados por parte del módulo A.E.S.

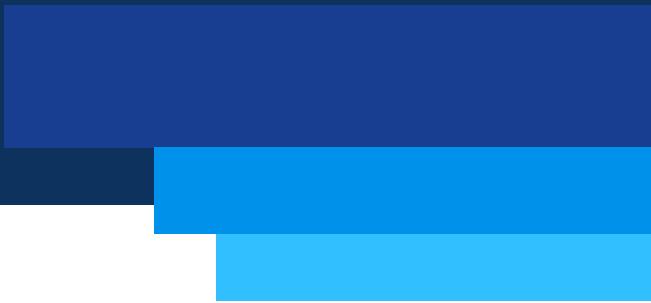
En función de dichos desafíos de diseño, se procedió a definir las principales características de hardware del módulo C.T.R.T.:

- La elección de incorporar la placa de desarrollo ESP32 NodeMCU se fundamenta en la integración efectiva de tecnología WiFi y en las funciones avanzadas de regulación de voltaje y adaptación de la alimentación que ofrece, tanto para la propia placa como para los pines GPIO asociados. En comparación con el microcontrolador ESP32 WROOM 32, el cual es utilizado en la placa de desarrollo ESP32 NodeMCU, se destaca la desventaja de ocupar un espacio mayor. Sin embargo, esta elección se justifica por la conveniencia que brinda al facilitar la ejecución de tareas relacionadas con la prueba de componentes y circuitos, y aunque el tamaño físico de la placa de desarrollo ESP32 NodeMCU podría considerarse una limitación, resulta insustancial para el desarrollo del



hardware del módulo C.T.R.T. Dicha limitación no presenta un obstáculo significativo, ya que el diseño del dispositivo no requiere una consideración sustancial de la ergonomía, más allá de consideraciones pragmáticas relacionadas con el uso eficiente de materiales

- La elección de implementar una luminaria tipo ojo de buey responde a la necesidad de homologar las indicaciones luminosas con los parámetros establecidos en el criterio de codificación luminosa del módulo A.E.S. Est tipo de luminaria proporciona una solución óptima para lograr la uniformidad deseada en las indicaciones luminosas, asegurando al mismo tiempo la coherencia estética con el diseño general entre módulos.
- Se ha optado nuevamente por la incorporación de un módulo elevador de tensión (Step-Up) MT3608 con el propósito de adaptar la tensión a un nivel útil para el posterior encendido controlado de la luminaria ojo de buey, hecho que surge de la disparidad de voltajes entre esta y la placa de desarrollo ESP32 NodeMCU, que ofrece una capacidad de alimentación máxima de 3.3V (frente a los 12V que requiere la luminaria para su encendido). Se destaca que, de igual manera que en el diseño del circuito electrónico del módulo A.E.S., se consideró la implementación de una fuente de alimentación exterior conectada a la placa de desarrollo por medio del puerto MicroUSB integrado en la misma (lo cual no solo facilita su alimentación, sino también su programación).
- Se ha optado por la implementación de relés de tipo simple inversor con la finalidad de gestionar el encendido y apagado de la luminaria tipo ojo de buey, siguiendo criterios programados. La selección de relés como



dispositivos de conmutación es respaldada por su versatilidad y eficacia en el control de cargas eléctricas. Estos relés, al ser de tipo simple inversor, ofrecen la capacidad de cambiar entre dos estados de manera fiable y eficiente. Además, la capacidad de controlar tensiones más altas con señales de entrada más bajas es esencial para la gestión efectiva de la luminaria, proporcionando una solución eficaz y segura.

- La implementación de un teclado matricial, con el objetivo de ingresar comandos tanto al módulo A.E.S. como al programa simulador de vuelo, así como permitir la navegación dentro de la información de aeronaves presente en la interfaz gráfica enseñada por medio de la computadora personal.
- La implementación de una computadora personal (notebook) modelo Dell Latitude 3120 como medio por el cual proveer, a través de su pantalla, una representación de la información de las aeronaves circundantes embebida en una interfaz gráfica, así como el envío de comandos al programa simulador de vuelo por medio del protocolo de comunicación UDP.

Componentes utilizados y criterio de selección:

En base a la descripción de hardware previamente proveída, así como los criterios de diseño definidos en función de las necesidades encontradas para con el desarrollo del módulo C.T.R.T., se procedió a seleccionar los componentes consecutivamente descritos para el posterior desarrollo de la placa controladora del dispositivo. A continuación, se detalla el listado específico de componentes integrados en la placa controladora del módulo C.T.R.T, así como el criterio de selección de cada uno:

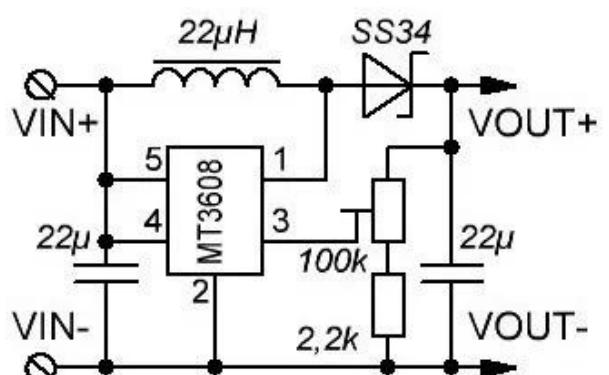
- Placa de desarrollo ESP32 NodeMCU: Corresponde a la unidad de procesamiento del módulo C.T.R.T. Su utilización queda definida dada la sencillez de su implementación, que se caracteriza por la resolución integrada de los módulos de alimentación, regulación de voltaje y protocolos de comunicación, lograda sin requerimientos de intervención ni diseño adicional por parte del usuario. A su vez, la inclusión de tecnología WiFi se alinea con los requisitos del equipo, en el cual se demanda la recepción de datos provenientes del módulo V.D.B (incluyendo su subsiguiente procesamiento) y las instrucciones de accionamiento manual generadas a través del panel frontal del módulo A.E.S, así como el envío de instrucciones para con dicho equipo.



- Luminaria ojo de buey: Corresponde al principal medio de representación luminosa de alertas y emergencias en vuelo que integra el panel frontal del módulo C.T.R.T. Su selección asegura la coherencia estética del panel frontal y la funcionalidad operativa del sistema, cumpliendo con los requisitos visuales y eléctricos necesarios para la representación efectiva de alertas y emergencias. La presencia de luminarias de color rojo y ámbar en el panel frontal responde a la necesidad de representar visualmente alertas y emergencias según el protocolo establecido (Warning, Alert). Es relevante señalar que la tensión de alimentación requerida para la luminaria ojo de buey difiere de la proporcionada por la placa de desarrollo a través de sus GPIOs. En consecuencia, se ha llevado a cabo la selección de componentes que permitan adaptar la tensión de alimentación, garantizando así el correcto funcionamiento y control del encendido de la luminaria, así como dispositivos conmutadores diseñados para controlar circuitos operativos a tensiones superiores mediante voltajes inferiores. Dichos componentes quedan detallados en los ítems consecutivos.



- Módulo elevador de tensión (Step - Up) MT3608: La incorporación del módulo MT3608 en el diseño del sistema responde a la necesidad de adaptar la tensión de trabajo de la placa de desarrollo con la luminaria del panel. Es preciso destacar que, al carecer de una fuente de alimentación interna, cualquier carga presente en el circuito electrónico del módulo C.T.R.T. recibe alimentación directa o indirecta a partir de la tensión suministrada por la placa de desarrollo. El componente en cuestión fue seleccionado debido a la simpleza de su implementación (evita la necesidad de diseñar un módulo Step - Up por cuenta propia, al mismo tiempo que resuelve esta problemática en un formato compacto, minimizando su impacto en la placa en términos de espacio ocupado) y su capacidad de configuración (su tensión de salida queda definida en función del ajuste del cursor de un preset integrado para tales fines, proporcionando flexibilidad en la configuración de tensión de salida de manera precisa y eficiente para adaptarse a los requisitos específicos).



- Relevador simple inversor: Este tipo de componente fue seleccionado para obrar como controlador de conmutación de la luminaria ojo de buey (llave accionada por corriente eléctrica proveniente de GPIOs) dada su sencilla

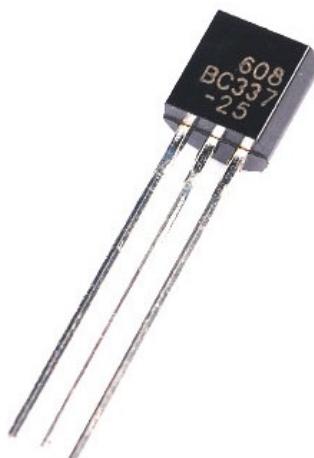
interacción tanto con la placa de desarrollo como con la carga a alimentar. A su vez, tampoco requiere de configuraciones específicas para su correcta conmutación. Se destaca que la tensión de conmutación del relevador simple inversor (5 a 12 V) difiere para con aquella capaz de ser proveída por la placa de desarrollo (3.3V). Con dicho objetivo, se coloca un dispositivo transistor en configuración de conmutación, con un correspondiente diodo rectificador en antiparalelo a la inductancia del relevador. Esta configuración asegura una adaptación eficiente de las tensiones y garantiza un funcionamiento preciso del relevador en el sistema



- Teclado Matricial 4x4: Corresponde al componente por medio del cual ingresar instrucciones en base a cadenas de caracteres estandarizadas para el envío de comandos a las aeronaves capaces de ser visualizadas en la pantalla del módulo C.T.R.T. El criterio de selección de este componente se encuentra definido por su tamaño compacto, de características eficientes y ergonómicas para la introducción de comandos, cumpliendo con los requisitos de tamaño reducido para una integración fluida en el sistema, contribuye significativamente a su usabilidad, asegurando una entrada de datos precisa y simplificada para la transmisión de instrucciones a las aeronaves.



- Transistor BJT NPN BC337: El transistor en cuestión se selecciona para desempeñar el papel de intermediario, posibilitando el control de la conmutación del relevador mediante la tensión proporcionada por la placa de desarrollo. Aunque no se impone la implementación específica del modelo BC337 para lograr una conmutación adecuada (dada la disponibilidad de diversos modelos de transistores en el mercado para aplicaciones genéricas como la requerida en este contexto), su elección particular se basa principalmente en una serie de pruebas exitosas realizadas con este componente. Es importante señalar que se debe incorporar un resistor de valor máximo de 10KΩ entre la alimentación del GPIO y la base del transistor. Esta disposición garantiza que el transistor se comporte en su región de saturación, optimizando así su rendimiento en el proceso de conmutación. La elección consciente de estos componentes contribuye a la fiabilidad y eficiencia del sistema, respaldando las necesidades específicas de control de conmutación.



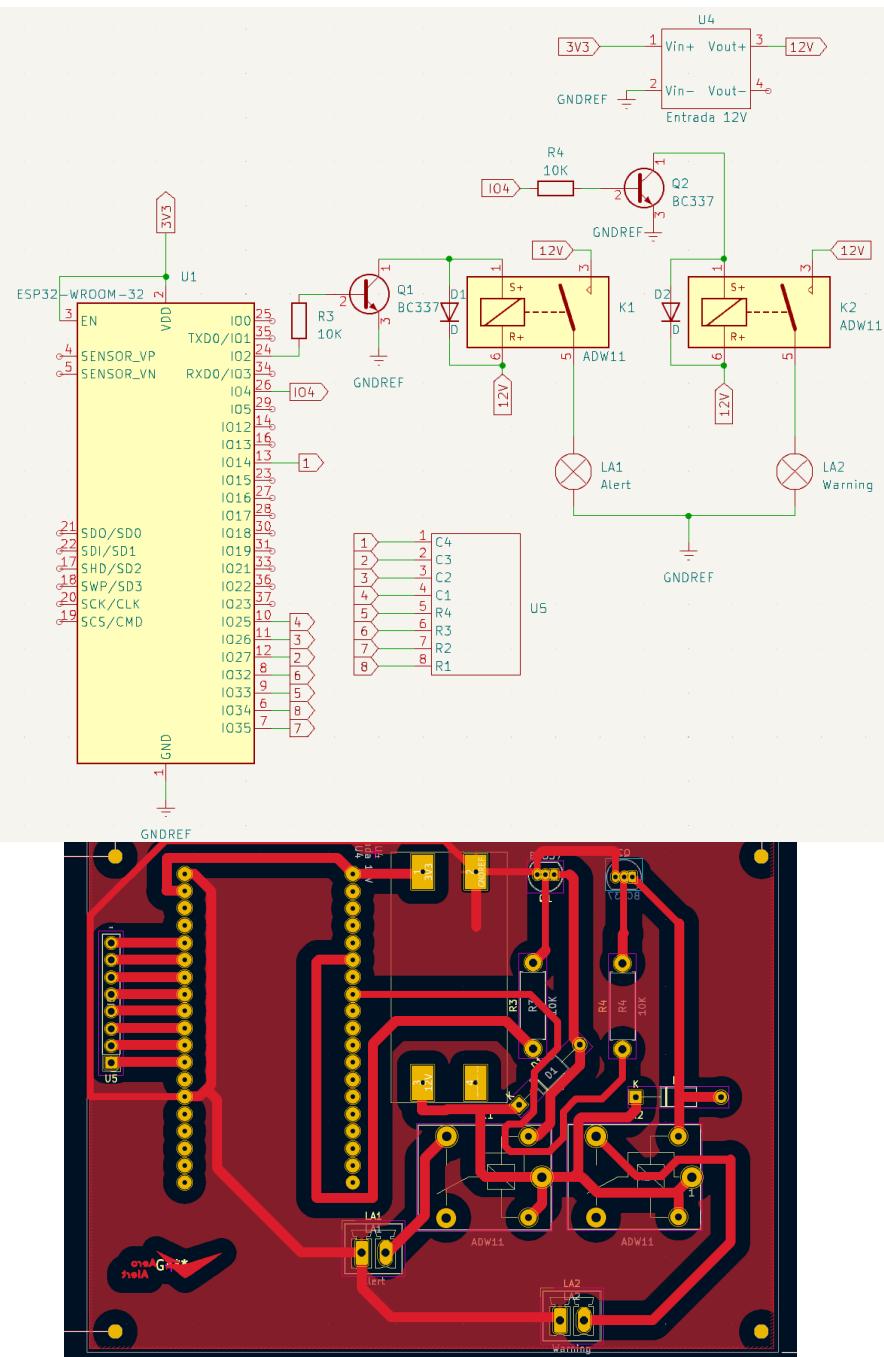
- Computadora Dell Latitude 3120: La utilización de una computadora personal modelo Dell Latitude 3120 es definida dada la necesidad de representar en tiempo real los datos de aeronaves involucradas por medio de una interfaz gráfica a través de la cual sea posible visualizar el estado de salud de los pilotos, la identificación de la aeronave y el historial de comandos enviados a cada aeronave en cuestión (de haber sido necesario efectuar tales acciones). Además, por medio de la batería interna de la computadora personal, la misma será capaz de alimentar a la placa madre del módulo C.T.R.T., aprovechando el puerto MicroUSB integrado de la placa de desarrollo.





Funcionamiento del Circuito Electrónico:

El circuito electrónico correspondiente al módulo C.T.R.T., para su mejor comprensión, queda detallado en los siguientes esquemas:





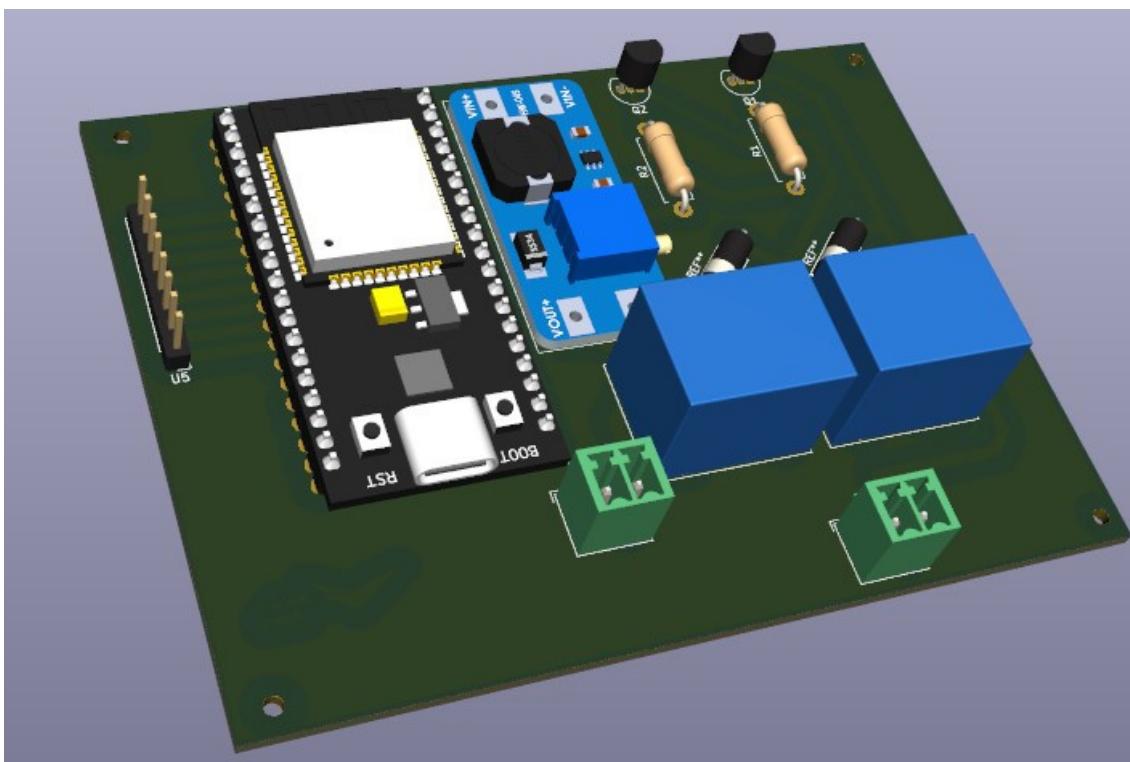
El circuito electrónico del módulo C.T.R.T. presenta un diagrama de diseño sin mayores complejidades: Se asume que la alimentación del circuito se obtiene de la batería interna de la computadora personal, conectada a la placa de desarrollo mediante el puerto MicroUSB integrado. Cada GPIO configurado como salida digital a nivel de programa es capaz de generar una diferencia de potencial máxima de 3.3V entre su terminal y el plano de masa.

Para cada circuito de luminaria (Warning y Alert), se conecta la base de un transistor BC337 junto con un resistor en serie de valor máximo 10KΩ a la salida directa de cada GPIO asignada a dicha función. Se destaca que el colector de cada transistor se conecta a uno de los extremos de la inductancia del relevador, mientras que su emisor se referencia a masa. El transistor actúa como una llave en el terminal neutro del bobinado. Cuando la base del transistor recibe energía desde su respectivo GPIO según los requisitos del programa, el transistor se activa y cierra el circuito de alimentación del relevador, provocando la conmutación y el encendido de la luminaria correspondiente. Dado que el relevador está siempre alimentado a 12V (comuta solamente en caso de que el terminal neutro del circuito sea cerrado), el contacto asociado permanece energizado, listo para cerrarse y suministrar alimentación a la luminaria que debe activarse.

Por otro lado, también es importante describir el subcircuito dedicado a la adaptación de voltaje por medio del módulo Step - Up: su entrada positiva se conecta al pin de alimentación de la placa de desarrollo (3V3), mientras que su entrada negativa se referencia a masa. La salida positiva proporciona los 12V necesarios para la operación de las luminarias según la configuración del preset integrado, alimentando tanto al bobinado del relevador como al contacto asociado. La salida negativa no se considera para los propósitos de diseño.

Por último, se destina una tira de 8 pines dedicada a la conexión del teclado matricial para con la placa de desarrollo.

La siguiente imagen hace alusión al montaje de los componentes electrónicos que integra la placa madre del módulo C.T.R.T, presentando una simulación renderizada de dicha placa de circuito impreso.



Módulo C.T.R.T.: Ensamblaje estructural

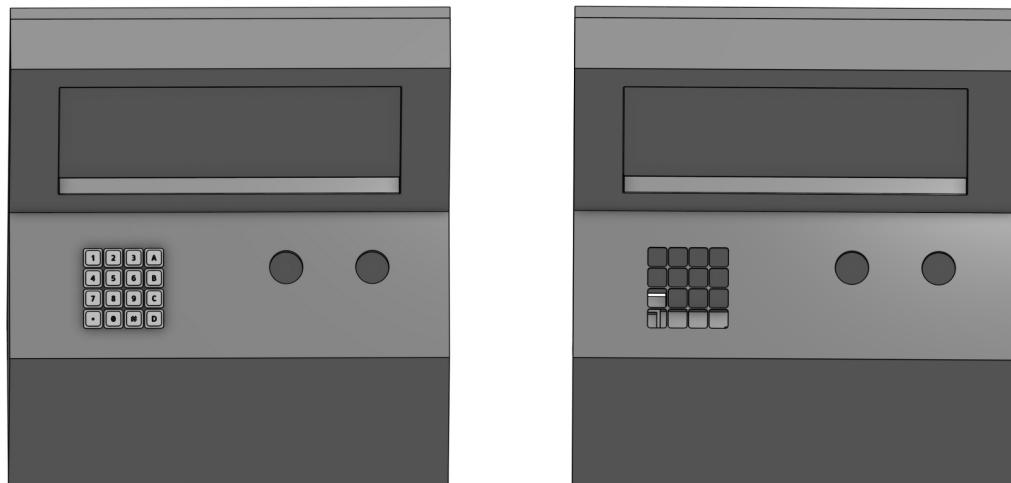
Criterios de diseño e implementación:

Con el objetivo de desarrollar un equipo cuyo propósito es su alojamiento en tierra a disposición del control de tráfico aéreo, y la enseñanza por medios visuales y frontales de alertas y/o emergencias en vuelo, se procedió a diseñar una carcasa para el módulo C.T.R.T. haciendo el principal foco en el desarrollo del panel frontal del equipo. A su vez, los principales desafíos de diseño a enfrentar para el modelado de la correspondiente carcasa consistieron en el espacio físico en el cual alojar la placa madre del módulo (y su principal inconveniente derivado, la conexión de la luminaria e interruptores del panel frontal para con ella), así como proveer un espacio dedicado al alojamiento de la computadora personal y el cableado de la alimentación del equipo por medio del puerto MicroUSB que integra la placa de desarrollo.

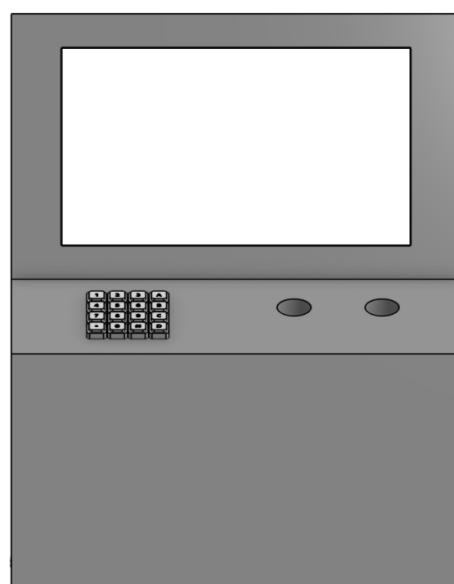
Haciendo uso de la plataforma web Onshape, dedicada al modelado 3D con fines industriales, se procedió a desarrollar el modelo correspondiente a la carcasa del módulo C.T.R.T., tal y como demarca la siguiente ilustración:



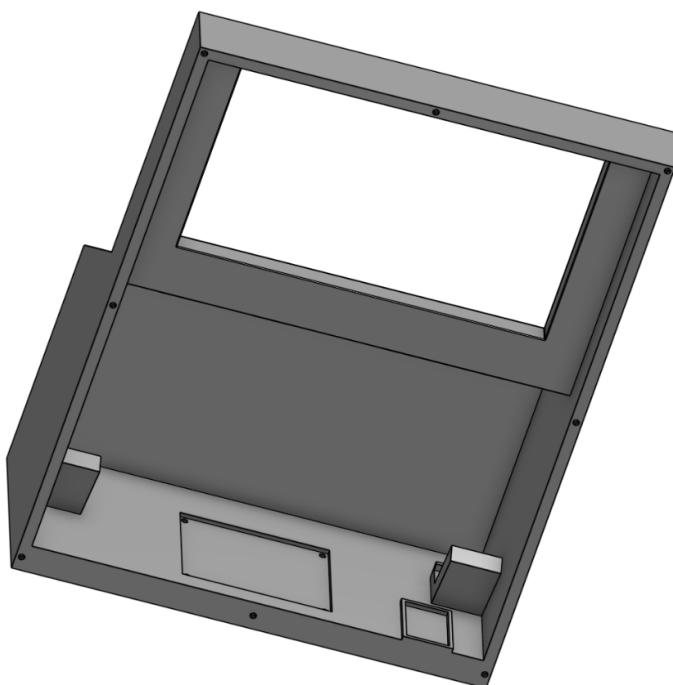
En el panel frontal, se encuentra dispuesta una serie de agujeros con el objetivo de emplazar tanto la luminaria para la representación visual y frontal de alertas y/o emergencias, como el teclado matricial para el ingreso manual de instrucciones hacia el módulo A.E.S. en cabina.



A su vez, se integra un marco en el panel frontal con el objetivo de emplazar correctamente la pantalla de la computadora personal, a modo de obtener una visualización clara y concisa de la información de aeronaves circundantes.

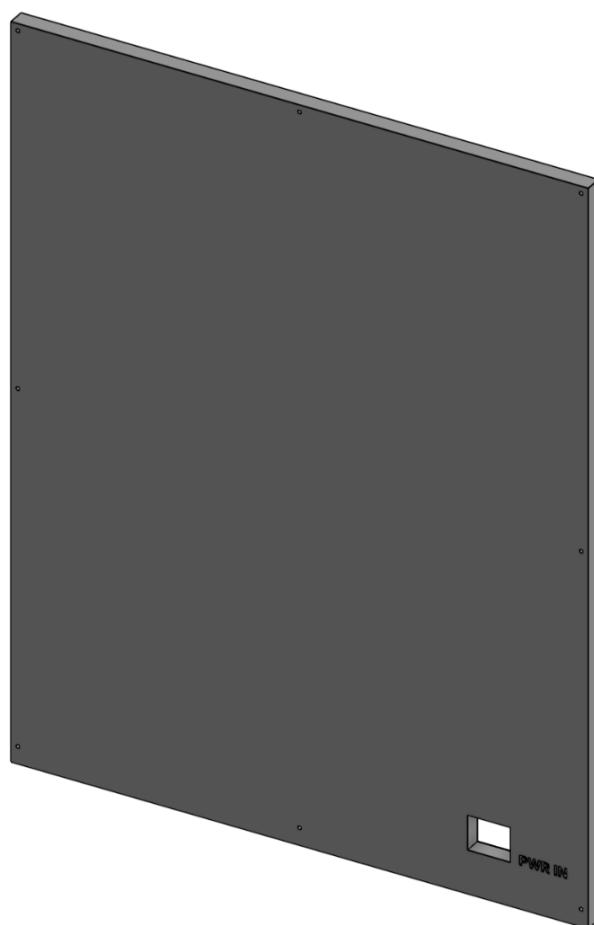


En el interior del núcleo estructural del módulo C.T.R.T., y más precisamente en su superficie inferior, se diagramaron zócalos para el emplazamiento de la placa madre del módulo, así como para sostener el cargador de la computadora personal. Los soportes en forma de “L” cumplen la función de alojar la computadora personal en disposición vertical, con el objetivo generar una correcta alineación entre la pantalla del ordenador y el marco del panel frontal destinado a tal fin.

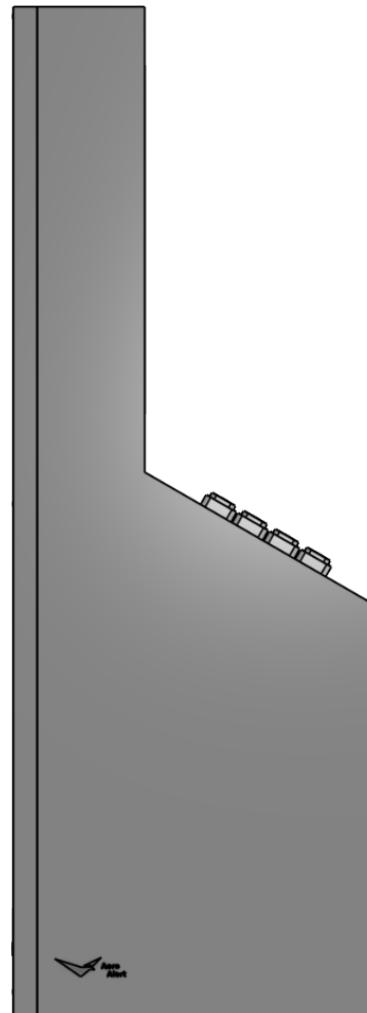


La tapa trasera del módulo C.T.R.T. implementa una serie de agujeros alrededor de su borde exterior, a fin de la posterior fijación de la misma con demás partes de la carcasa del equipo. A su vez, de la misma forma y con el mismo propósito que la tapa trasera del módulo A.E.S, se dispone de una muesca rectangular (con su correspondiente leyenda “PWR IN” grabada a un costado) para el pasaje del cable de alimentación del cargador de la computadora personal. En esta

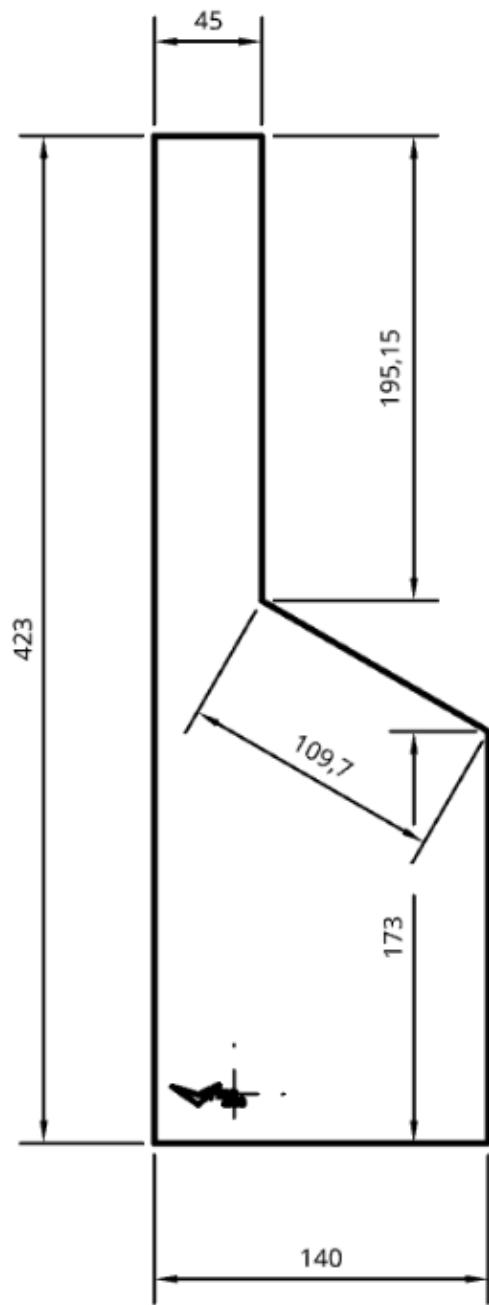
ocasión, dicha muesca se emplaza en el costado opuesto a la muesca análoga del módulo A.E.S, y tal diferencia se explica en el sector en donde se aloja el zócalo correspondiente al cargador de la computadora personal, cuya ubicación en la superficie inferior fue diagramada estratégicamente para coincidir con el mismo perfil del ordenador en donde se aloja el puerto de alimentación.

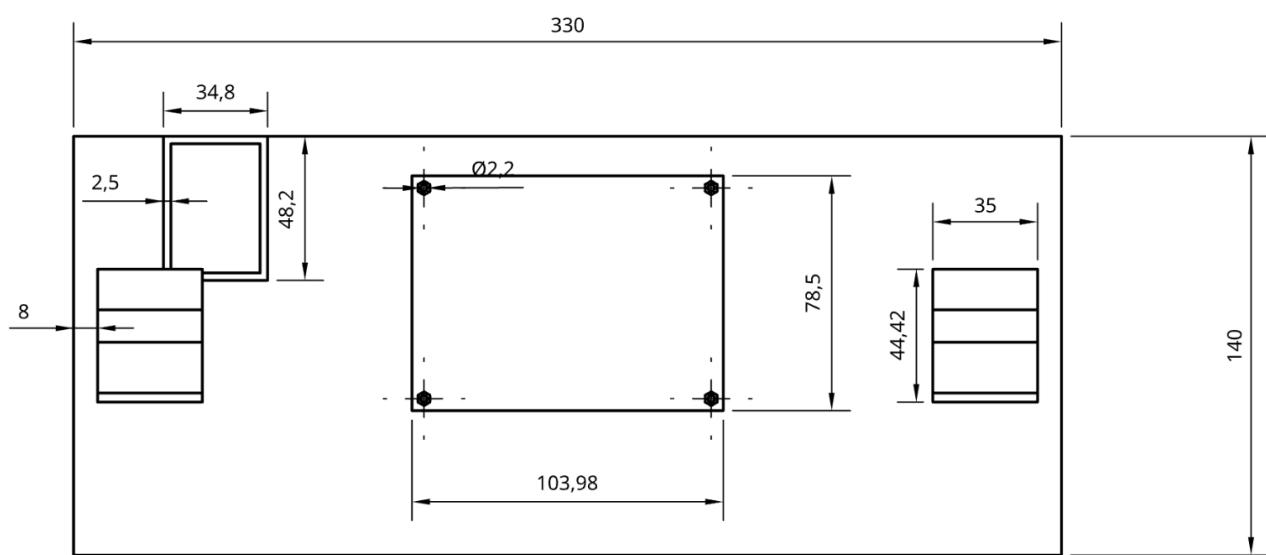
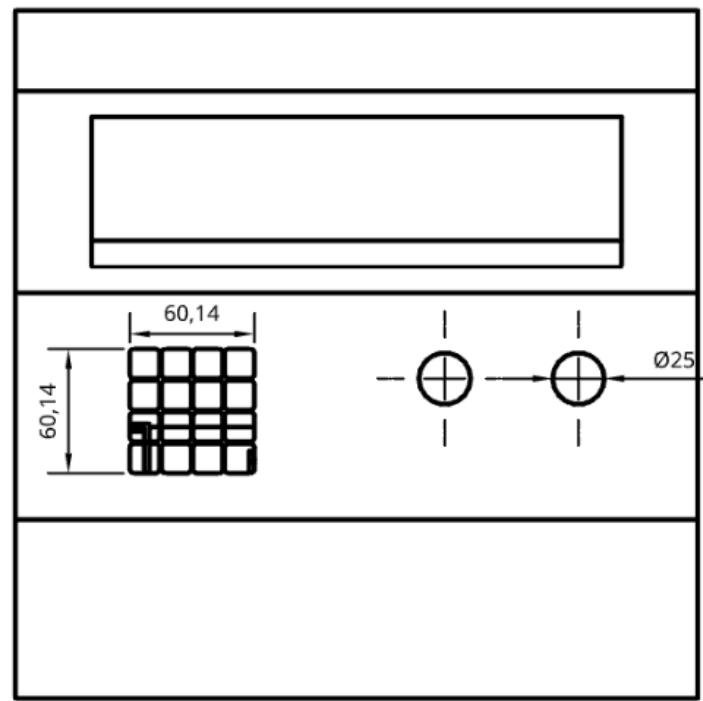
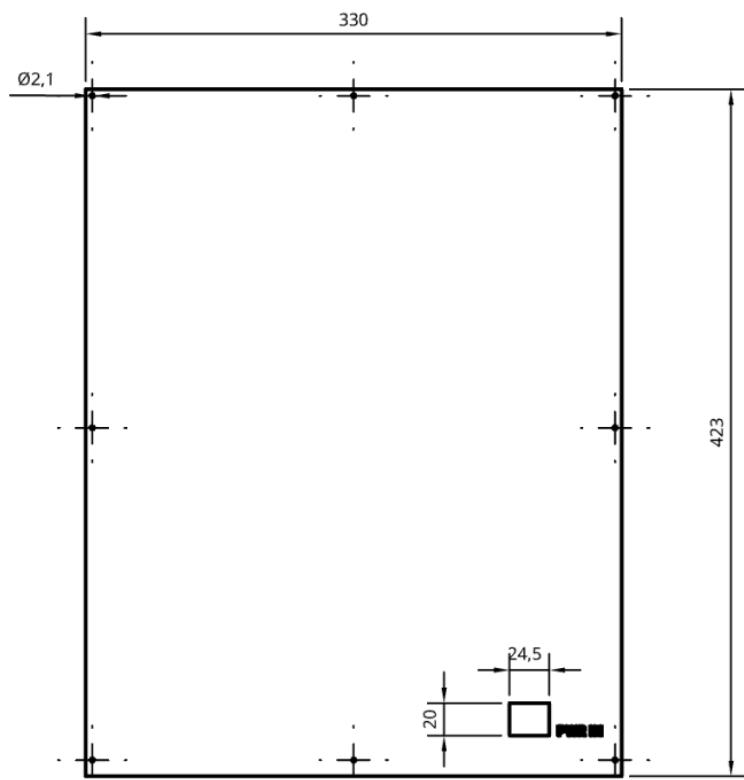


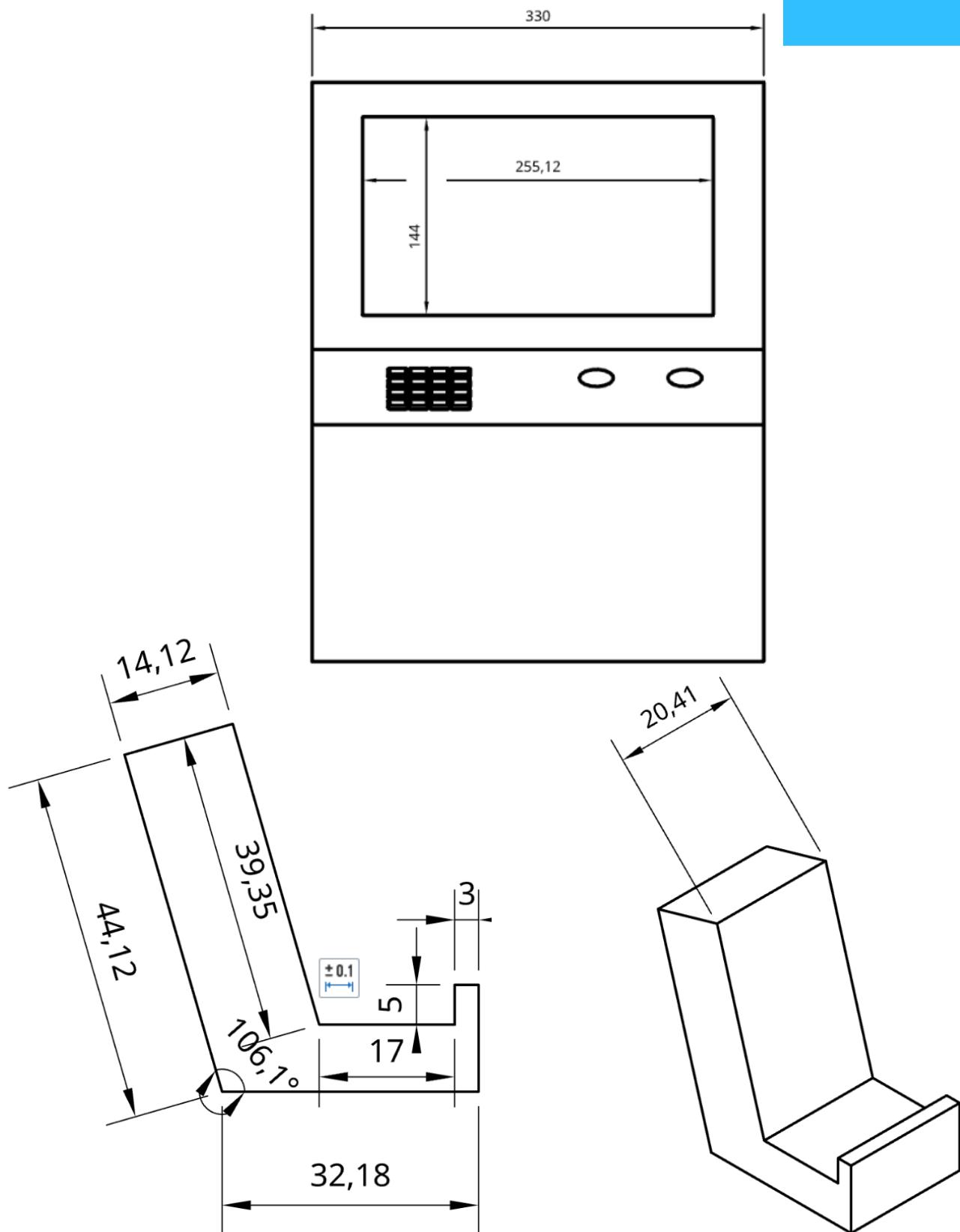
La siguiente imagen enseña una vista lateral del ensamblaje estructural del módulo en cuestión. A través de la misma, se puede visualizar el acabado resultante por la unión entre la tapa trasera y demás secciones de la carcasa del módulo C.T.R.T. En la pared lateral derecha, se puede encontrar un grabado del logo del proyecto en su esquina inferior izquierda (tomando como referencia la orientación de la imagen).

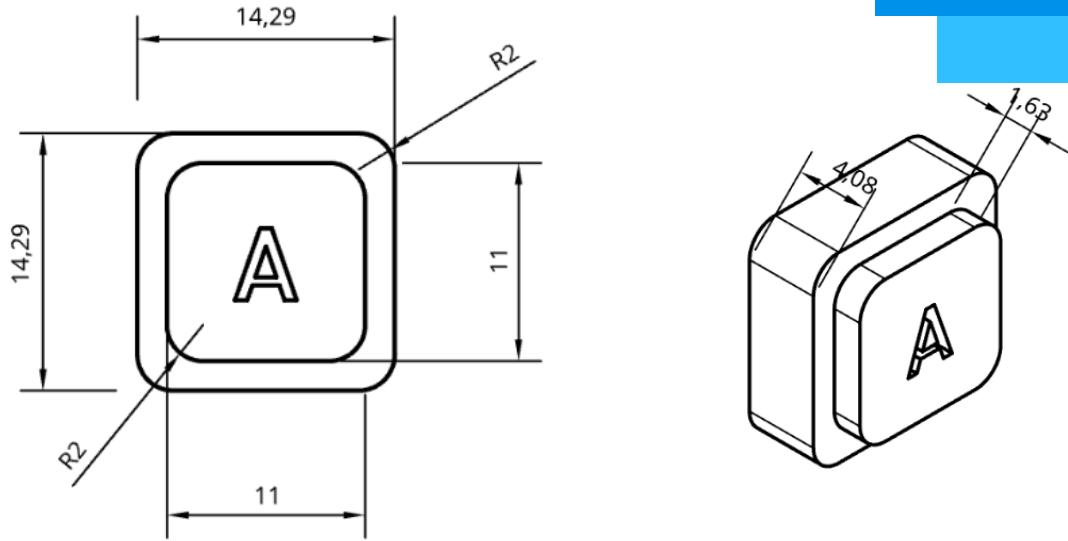


Las siguientes imágenes a presentar corresponden a los planos de las piezas que componen a la carcasa en cuestión, en sus distintas vistas de interés y con las acotaciones necesarias para el correcto entendimiento del diseño. Toda medida presente en los planos se encuentra expresada en unidad de milímetros (mm).

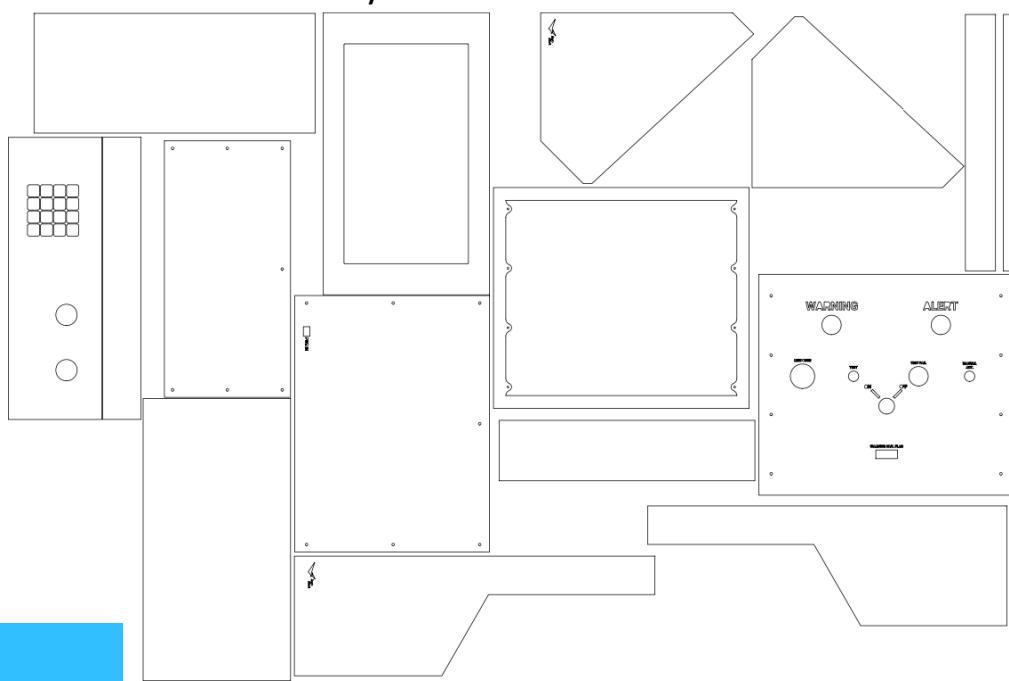








Para llevar a cabo el ensamblaje físico estructural del módulo A.E.S., se opta por la confección en madera tipo MDF (espesor 9mm). Una vez obtenida la pieza final modelada, se procede a exportar sus vistas exteriores a formato DXF, con el objetivo de prepararlas para el posterior ploteado por corte láser, en simultáneo con el desarrollo estructural final del módulo A.E.S. La siguiente imagen enseña el archivo de corte enviado al proveedor para la realización del trabajo de confección de las piezas que posteriormente formarán parte de las carcasas de los módulos A.E.S. y C.T.R.T.





Módulo C.T.R.T:

Descripción del software

Recepción y Trasmisión de datos :

Los datos relevados acerca de las variables médicas y el estado del vuelo (Alerta/Emergencia/Normal) son transmitidos por medio del protocolo WiFi (Wireless Fidelity) hacia la placa de desarrollo ESP32 NodeMCU situada en el módulo C.T.R.T. para su posterior procesamiento. Este código se encuentra escrito en MicroPython, por medio del IDE Thonny.

Dicha placa de desarrollo, de la misma forma que recibe datos por el protocolo de red mencionado, también envía información a una placa situada en el equipo A.E.S. Dicho fragmento de código fue desarrollado en el lenguaje MicroPython, haciendo uso del programa Thonny.

El programa se vale de las librerías **socket** para definir los puertos que se van a usar en la comunicación, **json** a fin de codificar y decodificar los mensajes de la comunicación, **network** con el objetivo de establecer la comunicación por WiFi y **time** para evitar sobrecargas en la placa de desarrollo.

```
def do_connect():
    server_ip = '192.168.4.1'
    server_port = 8000
    addr = socket.getaddrinfo(server_ip, server_port)
    sta_if = network.WLAN(network.STA_IF)
    if not sta_if.isconnected():
        sta_if.active(True)
        sta_if.connect('ESP32','aeroalert')
        while not sta_if.isconnected():
            print(".", end="")
            time.sleep(.1)
        print()
    print('network config:', sta_if.ifconfig())

client_socket = socket.socket(socket.AF_INET,
                               socket.SOCK_STREAM)
client_socket.connect((server_ip, server_port))
print("conecto")
return client_socket, sta_if
```

```
def send_type(client_socket, tipo):
    message = json.dumps(tipo).encode('utf-8')
    client_socket.send(message)

def send_message(client_socket, msg):
    data = {'mensaje': msg}
    message = json.dumps(data).encode('utf-8')
    client_socket.send(message)
    print('Mensaje enviado:', message)

def receive_data(client_socket):
    data = client_socket.recv(1024)
    print('respuesta: ', data)
    return data
```

Sistema de alertas:

Tras la recepción de los datos alusivos al estado de salud de los pilotos, como así también del correspondiente al vuelo, el módulo C.T.R.T es capaz de generar alertas tanto sonoras como lumínicas en función de la información recibida.

En el caso de las alertas audibles, la información procesada determina el envío de instrucciones por medio del protocolo UART a la computadora del equipo C.T.R.T., a fin del accionamiento de las respectivas alarmas.

En relación a las alertas lumínicas, la información procesada determina la configuración en estado lógico “alto” de GPIOs de la placa de desarrollo ESP32

NodeMCU.

```
def notification(cual):
    global pin_luz_ambar, pin_luz_roja
    global emergency, alert
    if cual == "emergency":
        print("emergency")
        pin_luz_roja.value(1)
        pin_luz_ambar.value(0)
        emergency = 1
        alert = 0

    elif cual == "alert":
        print("alert")
        pin_luz_roja.value(0)
        pin_luz_ambar.value(1)
        alert = 1
        emergency = 0

    elif cual == "clean":
        pin_luz_ambar.value(0)
        pin_luz_roja.value(0)
        alert = 0
        emergency = 0
```

```
info_aeropuerto = [{"aeropuerto":"ezeiza", "coordenadas": [23,43]},  
{"aeropuerto":"aeroparque", "coordenadas": [54,22]}]
```

```
data = stationrtdc.receive_data(client_socket)  
print(data)
```

```
if data == "solicito aterrizaje":  
    index_aeropuerto = int(input("ingrese numero de aeropuerto: "))  
    stationrtdc.send_message(info_aeropuerto[index_aeropuerto])  
elif data == "alerta desactivacion del sae":  
    estado_sae = 0  
elif data == "Sae activado":  
    estado_sae = 1  
elif type(data) == list:
```

```
if data["no_reaccion"] or data["manual"] or info["muerte1"] and info["muerte2"]:  
    notification("emergency")  
else:  
    if data["muerte1"] or data["muerte2"]:  
        notification("alert")  
    elif data["spo_bajos1"] or data["spo_bajos2"]:  
        notification("alert")  
    elif data["dormido1"] or data["dormido2"]:  
        notification("alert")  
    elif not data["pulsera_conectada"] or data["pin_on_off"]:  
        notification("alert")  
if estado_sae == 0:  
    notification("alert")  
if sum(data) == 1:  
    notification("clean")
```

Interacción con el operario:

Para el control manual del equipo C.T.R.T. se dispone de un teclado matricial 4x4, cuyos caracteres de entrada son leídos y procesados por la placa de desarrollo presente en la placa madre del equipo C.T.R.T. Por medio del IDE Thonny, la escritura del código en cuestión fue llevada a cabo en lenguaje MicroPython.

```
# Definición de Pines
filas = [16, 4, 0, 2]
columnas = [19, 18, 5, 17]

# Definimos los pines de las filas como salida
pines_Filas = [Pin(pin_nombre, mode=Pin.OUT) for pin_nombre in filas]

# Definimos los pines de las columnas de salida
pines_Columnas = [Pin(pin_nombre, mode=Pin.IN, pull=Pin.PULL_DOWN) for pin_nombre in columnas]

# Función para inicializar el teclado
def inicio():
    for fila in range(0, 4):
        for col in range(0, 4):
            pines_Filas[fila].value(0)

def escanear(fila, columna):
    """ Escaneo del teclado """
    # poner todas las filas en alto
    pines_Filas[fila].value(1)
    key = None
    # verificación al presionar una tecla o evento
    if pines_Columnas[columna].value() == Tecla_Abajo:
        key = Tecla_Abajo
    if pines_Columnas[columna].value() == Tecla_Arriba:
        key = Tecla_Arriba
    pines_Filas[fila].value(0)
    # retorne el estado de la tecla
    return key
```



En dicho código, se puede visualizar la asignación de los pines del teclado matricial 4x4 a sus respectivos caracteres asociados. Posteriormente, se lleva a cabo un escaneo de las filas y de los pines, a modo de verificación de su funcionamiento.

```
print("Iniciando... Presione una tecla: ")
inicio()

# Variable para almacenar la secuencia ingresada
secuencia_actual = ""

while True:
    for fila in range(4):
        for columna in range(4):
            tecla = escanear(fila, columna)
            if tecla == Tecla_Abajo:
                print("Es el número:", teclas[fila][columna])
                secuencia_actual += teclas[fila][columna]
                sleep(0.6)

            if len(secuencia_actual) >= 2:
                if secuencia_actual == "3A":
                    print("AES Activado")
                elif secuencia_actual == "6B":
                    print("AES Desactivado")
                elif secuencia_actual == "9C":
                    print("EMERGENCIA")
                elif secuencia_actual == "*0":
                    print("Iniciar Comunicación con Cabina")
                elif secuencia_actual == "0#":
                    print("Finalizar Comunicación con Cabina")
                else:
                    print("Secuencia desconocida")
                secuencia_actual = "" # Reiniciar la secuencia después de detectar la combinación
            inicio() # Reiniciar la detección de teclas en cada iteración
```

De acuerdo a la secuencia de caracteres ingresada por medio del teclado, se genera una respuesta asociada a combinaciones estándar previamente definidas. A modo de ejemplo, al presionar “3A” la respuesta obtenida es “AES Activado”. Por otro lado, al presionar una combinación por fuera del estándar, la respuesta por parte del código es “Secuencia Desconocida”.

Interfaz gráfica del equipo C.T.R.T

Para la demostración en pantalla de la información alusiva a datos de vuelo por parte del módulo C.T.R.T., se hace uso de una computadora personal, la cual enseña a través de un navegador una página web alojada en un servidor generado por la placa de desarrollo presente en la placa madre del equipo. La placa de desarrollo devuelve una página web con la interfaz gráfica y datos del equipo C.T.R.T. El código funcional a este último propósito se encuentra realizado en HTML, CSS y JavaScript, haciendo uso de Visual Studio Code como editor de texto. Por otro lado, el código alusivo al hosteo en servidor web de dicha página por parte de la placa de desarrollo se encuentra escrito en el lenguaje MicroPython, valiéndose de la utilización del IDE Thonny para ello.

Por medio de la pestaña principal, se presentan tres ejemplos de vuelos (la imagen superior muestra a todos ellos en estado "Normal").

Flight details	SKX5554 A21N	TAM3081 A320	ARG1262 B738
Flight details			
Aircraft Name	State	Date / Time	
SKX5554 A21N	Normal	14/10/2023, 19:04:28	
Flight details			
Aircraft Name	State	Date / Time	
TAM3081 A320	Normal	14/10/2023, 19:04:28	
Flight details			
Aircraft Name	State	Date / Time	
ARG1262 B738	Normal	14/10/2023, 19:04:28	

A su vez, existe una pestaña asignada individualmente a cada vuelo. En la primera sección, se indica el historial de solicitudes ingresadas. Debajo de cada solicitud, se especifica si esta ha sido confirmada o se encuentra pendiente de resolución. Se adjunta la fecha y hora de realización de cada potencial solicitud.

The screenshot shows a user interface for managing flight requests. At the top, there are four tabs: 'Flight details' (selected), 'SKX554 A21N', 'TAM3081 A320', and 'ARG1262 B738'. Below these tabs, there are three separate sections, each containing a bolded 'Request entered' message and a descriptive text below it.

- Request entered**
The code should appear here and next to it the resolution, which means: "AES Activated"
- Request entered**
Here it should appear, Pending - Confirmed
- Request entered**
Previous requests, with date and time

Finalmente, en caso de que la aeronave se encuentre en una situación de emergencia, un cartel comenzará a aparecer en la pestaña principal, notificando de forma el estado en el que se encuentra la aeronave involucrada.

The screenshot shows the same user interface as before, but now the 'SKX554 A21N' tab is selected. In the main content area, there is a table listing three aircraft with their current states and last updated times. A large red rectangular overlay covers the bottom portion of the table, with the word 'EMERGENCY' written in white capital letters across it.

Aircraft Name	State	Date / Time
SKX554 A21N	EMERGENCY	14/10/2023, 19:14:41
TAM3081 A320	Normal	14/10/2023, 19:14:41
ARG1262 B738	Normal	14/10/2023, 19:14:41



Los fragmentos de código presentados a continuación, escritos en los lenguajes HTML, CSS y JavaScript, enseñan tanto el maquetado como los estilos y demás funcionalidades interactivas y relacionadas a la actualización de datos de vuelo de la interfaz de usuario previamente descrita. En particular, las siguientes dos imágenes presentan la organización espacial de dicha interfaz, llevada a cabo por medio de la utilización del lenguaje HTML.

```
<div class="wrap">
    <ul class="tabs">
        <li><a href="#tab1"></span><span class="tab-text">Flight details</span></a></li>
        <li><a href="#tab2"></span><span class="tab-text">SKX5554 A21N</span></a></li>
        <li><a href="#tab3"></span><span class="tab-text">TAM3081 A320</span></a></li>
        <li><a href="#tab4"></span><span class="tab-text">ARG1262 B738</span></a></li>
    </ul>

    <div class="secciones">
        <article id="tab1">
            <div class="container">
                <table>
                    <thead>
                        <tr>
                            <th>Aircraft Name</th>
                            <th>State</th>
                            <th>Date / Time</th>
                        </tr>
                    </thead>
                    <tbody id="tabla-datos">
                        <tr>
                            <td>SKX5554 A21N</td>
                            <td class="estado estado-en-vuelo">Normal</td>
                            <td class="fecha-hora">Date / Time</td>
                        </tr>
                    </tbody>
                </table>
            </div>
        </article>
    </div>

```

En su pantalla principal, la interfaz mencionada se vale de un diagramado intuitivo a fin de presentar una lista de vuelos en tiempo real. Por medio de la sección "Flight Details", se puede revisar el estado actualizado de las aeronaves presentadas simultáneamente. Cada vuelo se muestra con su respectivo estado: "Normal", "Pendiente" o "Emergencia", junto con las fechas y horarios correspondientes.

La imagen a continuación enseña la disposición de elementos con la que cuentan las pestañas individuales de cada aeronave dentro de la interfaz. A través de las mismas, se puede visualizar las solicitudes pendientes y revisar el historial completo de peticiones, incluyendo aquellas que fueron aceptadas, rechazadas o no atendidas. Cada solicitud está marcada con su respectiva fecha y hora.



```
</thead>
<tbody id="tabla-datos">
<tr>
    <td>ARG1262 B738</td>
    <td class="estado estado-en-vuelo">Normal</td>
    <td class="fecha-hora">Date / Time</td>
</tr>
</tbody>
</table>
<div class="alerta-container alerta-chico" id="alerta-alerta">
    ALERT
</div>
<div class="alerta-container alerta-grande" id="alerta-emergencia">
    EMERGENCY
</div>
</div>
</article>
<article id="tab2">
    <div class="info-box">
        <h2>Request entered</h2>
        <p id="infoContent">The code should appear here and next to it the resolu</p>
    </div>
    <div class="info-box">
        <h2>Request entered</h2>
        <p id="infoContent">Here it should appear, Pending - Confirmed</p>
    </div>
    <div class="info-box">
        <h2>Request entered</h2>
        <p id="infoContent">Previous requests, with date and time</p>
    </div>
</article>
```

El siguiente fragmento de código refiere a la adición de estilos conforme la necesidad de diseño de cada cuadrante asociado a información sobre un vuelo particular. Se define su forma, espacio, ancho y color a fin de reflejar la información de manera precisa e intuitiva, así como estéticamente amena para el ágil entendimiento y escaneo visual de la situación. A su vez, se define la estandarización de colores para alertas, ajustando la misma de acuerdo a variaciones y niveles de importancia para garantizar una representación visual efectiva de las notificaciones.



```
.wrap{  
    width: 800px;  
    max-width: 90%;  
    margin: 30px auto;  
}  
  
.tabs {  
    width: 100%;  
    background: #363636;  
    list-style: none;  
    display: flex;  
    justify-content: space-between; /* Distribuir equitativamente los elementos */  
    padding: 0; /* Eliminar el espacio entre los elementos */  
}  
  
.tabs li {  
    flex: 1; /* Hace que cada elemento ocupe el mismo espacio */  
    text-align: center;  
}  
  
.tabs li a {  
    color: #fff;  
    text-decoration: none;  
    font-size: 16px;  
    display: block;  
    padding: 20px 0px;  
}  
  
.active{  
    background: #3F0082;  
}
```

```
/* Estilos para el cartel de alerta */  
.alerta-container {  
    display: none;  
    position: fixed;  
    top: 55%;  
    left: 50%;  
    transform: translate(-50%, -50%);  
    padding: 60px;  
    color: #fff;  
    font-size: 60px;  
    font-weight: bold;  
    text-align: center;  
    z-index: 9999;  
}  
  
.alerta-container.active {  
    display: block;  
}  
  
.alerta-chico {  
    width: 200px;  
    height: 100px;  
    background-color: #ffecb3; /* Amarillo */  
}  
  
.alerta-grande {  
    width: 535px;  
    height: 200px;  
    background-color: rgba(255, 0, 0, 0.8); /* Rojo opaco */  
    animation: blink 1s infinite; /* Animación de parpadeo */  
}
```

El manejo del lenguaje JavaScript fue interiorizado para la confección de la interfaz de usuario del módulo C.T.R.T. ante la necesidad de implementar funcionalidades de muestreo de fecha y hora, así como también la exhibición del flag indicador de emergencia, características de notoria relevancia para proporcionar información actualizada y gestionar las situaciones de emergencia de manera eficaz.

El presente fragmento de código enseña la función para mostrar la fecha y la hora, identificando las secciones y elementos a actualizar con fecha y hora en tiempo real.

Por otro lado, la función presentada consecutivamente se encarga de actualizar de forma automática y periódica la información en una interfaz web de gestión de vuelos en un intervalo de 2 segundos: se detecta el estado del avión (en vuelo, en tierra, en alerta o en emergencia), se muestra la hora actualizada y permite la reproducción de sonidos de alerta según el estado.

```

<script>
    $(document).ready(function(){
        $('ul.tabs li a:first').addClass('active');
        $('.secciones article').hide();
        $('.secciones article:first').show();

        $('ul.tabs li a').click(function(){
            $('ul.tabs li a').removeClass('active');
            $(this).addClass('active');
            $('.secciones article').hide();

            var activeTab = $(this).attr('href');
            $(activeTab).show();
            return false;
        });

        function actualizarFechasHoras() {
            const fechaHoraActual = new Date().toLocaleString();

            // Actualizar celdas de fecha y hora en cada tabla
            $('.fecha-hora').text(fechaHoraActual);
            $('.fecha-hora1').text(fechaHoraActual);
            $('.fecha-hora2').text(fechaHoraActual);
        }

        // Actualizar las fechas y horas iniciales
        actualizarFechasHoras();

        // Actualizar las fechas y horas cada 2 segundos
        setInterval(actualizarFechasHoras, 2000);

        function actualizarDatos() {
            const filaAvion = document.querySelector('#tabla-datos tr');
            const estadoAvion = filaAvion.querySelector('.estado').textContent.toLowerCase().trim();

            const fechaHoraCell = filaAvion.querySelector('.fecha-hora', '.fecha-hora2', '.fecha-hora1');
            const fechaHoraActual = new Date().toLocaleString();
            fechaHoraCell.textContent = fechaHoraActual;

            filaAvion.querySelector('.estado').classList.remove('estado-en-vuelo', 'estado-en-tierra');

            if (estadoAvion === 'en vuelo') {
                filaAvion.querySelector('.estado').classList.add('estado-en-vuelo');
                document.getElementById('alerta-alerta').classList.remove('active');
                document.getElementById('alerta-emergencia').classList.remove('active');
            } else if (estadoAvion === 'en tierra') {
                filaAvion.querySelector('.estado').classList.add('estado-en-tierra');
                document.getElementById('alerta-alerta').classList.remove('active');
                document.getElementById('alerta-emergencia').classList.remove('active');
            } else if (estadoAvion === 'alerta') {
                document.getElementById('alerta-alerta').classList.add('active');
                document.getElementById('alerta-emergencia').classList.remove('active');
                reproducirAlerta();
            } else if (estadoAvion === 'emergencia') {
                document.getElementById('alerta-emergencia').classList.add('active');
                document.getElementById('alerta-alerta').classList.remove('active');
                reproducirAlerta();
            }
        }

        function reproducirAlerta() {
            const audioAlerta = document.getElementById('audio-alerta');
            audioAlerta.play();
        }
    });

```