
This is the **published version** of the bachelor thesis:

Chakrane Hedioued, Ilias. *Desarrollo de instrumentos analógicos basados en microcontroladores para simulación de vuelo*. Treball de Final de Grau (Universitat Autònoma de Barcelona), 2026 (Enginyeria Informàtica)

This version is available at <https://ddd.uab.cat/record/334435>

under the terms of the  license.

Desarrollo de instrumentos analógicos basados en microcontroladores para simulación de vuelo

Ilias Chakrane Hedioued

Resumen— La simulación de vuelo requiere, en muchos casos, instrumentos físicos que mejoren la inmersión y faciliten la interacción con la cabina. Sin embargo, las soluciones comerciales suelen ser costosas, poco flexibles y difíciles de adaptar a configuraciones personalizadas. Para dar solución a esta limitación, se aborda el diseño, desarrollo y fabricación de un panel físico modular con cuatro instrumentos analógicos esenciales: altímetro, indicador de velocidad aérea, indicador de actitud e indicador de velocidad vertical. La propuesta se apoya en una arquitectura jerárquica de tres capas: Microsoft Flight Simulator genera los datos de vuelo, un programa en C++ los obtiene mediante la API de SimConnect y los transmite al sistema embebido, encargado de procesar la información y accionar los instrumentos. El prototipo permite validar la viabilidad de la arquitectura propuesta y representar físicamente las principales variables de vuelo mediante módulos analógicos.

Palabras clave— Simulación de vuelo, microcontroladores, instrumentos analógicos, sistema embebido, comunicación modular y diseño de bajo coste.

Abstract— Flight simulation often requires physical instruments that improve immersion and facilitate interaction with the cockpit. However, commercial solutions are usually expensive, inflexible, and difficult to adapt to custom configurations. To address this limitation, this work tackles the design, development, and fabrication of a modular physical panel with four essential analog instruments: altimeter, airspeed indicator, attitude indicator, and vertical speed indicator. The proposed solution is based on a three-layer hierarchical architecture: Microsoft Flight Simulator generates the flight data, a C++ program retrieves it through the SimConnect API and transmits it to the embedded system, which processes the information and drives the instruments. The prototype validates the feasibility of the proposed architecture and enables the main flight variables to be physically represented through analog modules.

Keywords— Flight simulation, microcontrollers, analog instruments, embedded system, modular communication and low-cost design.

1 INTRODUCCIÓN

MICROSOFT Flight Simulator ha facilitado e impulsado el uso de la simulación de vuelo doméstica a un nivel de realismo muy elevado, lo que hace que aumente el interés por interfaces físicas que ayuden a mejorar la inmersión y el entrenamiento. Entre estas interfaces, los instrumentos analógicos continúan siendo relevantes por su lectura rápida y su similitud con aeronaves de aviación general. Sin embargo, gran parte de las solucio-

nes comerciales disponibles tienen un coste muy elevado o presentan poca flexibilidad para la personalización, mantenimiento o escalabilidad [3].

Se propone una alternativa económica y modular, formada por cuatro instrumentos analógicos esenciales: velocidad aérea, velocidad vertical, altitud y actitud, mostrados en Fig. 1, accionados por motores paso a paso y servomotores controlados por microcontroladores [7]. La arquitectura se divide en tres componentes básicos, el simulador, un programa intermedio que obtiene datos mediante una API de SimConnect [1] y un conjunto de placas electrónicas que transforman dichas variables en movimiento físico envueltas en una carcasa impresa en 3D.

El propósito no es competir en acabados industriales de alta gama [6], sino demostrar que, con una arquitectura bien definida y un diseño reproducible, es posible obtener una

• E-mail: iliaschakraneh@gmail.com

• Menció: Ingeniería de Computadores

• Trabajo tutorizado por: Dolores Isabel Rexachs del Rosario. Departamento de Arquitectura de Computadores y Sistemas Operativos.

• Curso 2025/26

experiencia funcional con un presupuesto limitado.

El documento se estructura en diferentes apartados donde se describe de forma progresiva el planteamiento del trabajo. Se presenta la necesidad y el alcance del proyecto, junto con los objetivos del mismo. Después, se expone la metodología que se siguió para su desarrollo. A continuación, se muestra la arquitectura del sistema propuesta, seguida de una descripción general del diseño electrónico y mecánico. Por último, se establece el plan de validación, antes de cerrar con las conclusiones.



Fig. 1: Conjunto de instrumentos del panel: altímetro (arriba izquierda), indicador de velocidad aérea (arriba derecha), indicador de actitud (abajo izquierda) e indicador de velocidad vertical (abajo derecha).

2 NECESIDAD Y ALCANCE DEL PROYECTO

En el mundo de la simulación de vuelo doméstica, falta una solución intermedia entre, por un lado, paneles virtuales basados en pantallas[4, 5, 6], fáciles de implementar pero poco escalables al aumentar el número de indicadores, y, por otro lado, instrumentación analógica, habitualmente también costosos o con una barrera técnica elevada en sus alternativas abiertas [3].

En este contexto, el alcance del trabajo resume el desarrollo de los principales bloques de dicha solución intermedia. En primer lugar, una aplicación software encargada de la lectura de las variables de vuelo desde Microsoft Flight Simulator y su envío al sistema embebido.

A continuación, el trabajo incluye el desarrollo del sistema responsable de procesar la información y separar los datos para enviárselos a cada uno de los cuatro módulos controlados por el sistema embebido.

Asimismo, el proyecto abarca el diseño e implementación del sistema de control de los instrumentos analógicos. De forma complementaria, también se incluye el diseño mecánico del conjunto mediante piezas fabricadas en impresión 3D y ensamblaje final.

Durante la fase final de integración se detectó una limitación mecánica importante en el módulo de altímetro. Por

ello, el alcance final del prototipo se centra en los indicadores de velocidad aérea, velocidad vertical y actitud. El altímetro se mantiene como parte del diseño y del firmware desarrollado, pero no se considera validado a nivel mecánico ni se incluye en las pruebas finales de precisión.

Por último, la validación del sistema mediante pruebas que verifiquen la precisión, estabilidad y respuesta.

3 OBJETIVOS

Una vez definido el alcance del proyecto, se han establecido los objetivos que guiaron el diseño, desarrollo y validación del sistema. Los objetivos nos permitirán evaluar si la solución final es adecuada frente a las necesidades que se han identificado y sus limitaciones.

El objetivo general es diseñar, desarrollar y validar un panel físico modular de instrumentos analógicos para simulación de vuelo, orientado a Microsoft Flight Simulator, que forma una solución funcional, de bajo coste, precisa y escalable.

A partir de este objetivo general, se establecen los siguientes objetivos específicos:

- **Analizar** soluciones comerciales y proyectos existentes del sector para identificar sus limitaciones, costes y oportunidades de mejora.
- **Definir** la arquitectura del sistema modular que permita separar la obtención de los datos, su procesamiento y el control de los instrumentos.
- **Diseñar** una solución modular a nivel hardware y software, donde cada instrumento pueda funcionar como un módulo independiente, fácil de montar, sustituir y ampliar.
- **Implementar** un sistema preciso, estable y legible de las variables de vuelo en los indicadores físicos.
- **Asegurar** un comportamiento robusto ante reinicios, apagados y pérdidas de comunicación.

4 METODOLOGÍA

El desarrollo de este proyecto siguió una metodología incremental, con prototipos funcionales por capas. A diferencia de un desarrollo estrictamente lineal, en este caso las fases no se consideraron cerradas de forma definitiva, sino que se pudo revisar y ajustar limitaciones detectadas en la integración. De este modo, el proceso incorporó retroalimentación continua y permitió volver atrás cuando fue necesario.

En primer lugar se diseñó la aplicación intermedia encargada de obtener los datos de Microsoft Flight Simulator y enviarlos a las placas. Después se realizó una primera prueba con un dispositivo montado en una protoboard, con el objetivo de comprobar el control del instrumento y validar la comunicación master-slave, como se muestra en la Fig. 22 ubicada en el apéndice.

Una vez comprobada la comunicación, se desarrolló el firmware de los distintos dispositivos y se diseñaron las PCBs correspondientes. Durante esta fase fue necesario modificar la aplicación inicial para añadir envíos de mensajes

adicionales y adaptar el protocolo a las necesidades reales de los módulos.

Una vez validados los módulos, se integraron en el panel completo mediante un primer prototipo físico. Por último, se completó la documentación técnica y la presentación final del proyecto.

El diagrama de Gantt muestra la planificación general del proyecto. Aunque se representa de forma secuencial para facilitar la lectura, varias tareas se desarrollaron de manera solapada.

5 ARQUITECTURA DEL SISTEMA

Como se ha mencionado, el proyecto se estructura en una arquitectura de tres módulos. A su vez, dentro del sistema embebido se usará una jerarquía master-slave, donde un microcontrolador maestro distribuye la información a módulos esclavos independientes, uno por instrumento Fig. 1.

Capa de simulación. Microsoft Flight Simulator genera los datos de vuelo y muestra las variables necesarias mediante SimConnect [1](p. ej., velocidad indicada, altitud, actitud y velocidad vertical).

Capa intermedia. Una aplicación en C++ crea la conexión y obtiene todas las variables necesarias, las encapsula y envía al hardware mediante un enlace USB directo a la microcontroladora master.

Capa de control. El sistema embebido se organiza con una Pico 2 [2] master y módulos slave independientes, uno por instrumento, seleccionada por su bajo coste, tamaño reducido, disponibilidad de pines GPIO y compatibilidad con C++. El master recibe los datos de la aplicación y la distribuye a los módulos slave mediante conexiones USB-C usadas como interfaz física estandarizada para transportar alimentación y UART (TX/RX y GND). El master aplica además un criterio de prioridad de refresco, es decir, los instrumentos de dinámica rápida y mayor impacto perceptivo (p. ej., actitud y velocidad vertical) se actualizan con mayor frecuencia o menor tolerancia a latencia que variables de evolución más lenta (p. ej., altitud y velocidad aérea). Cada módulo slave realiza el escalado y suavizado necesarios y controla localmente su actuador, integrados mecánicamente en carcasas impresas en 3D y conectados mediante PCBs a medida.

Esta separación permite aislar funciones, facilitar la depuración y mantener un diseño modular y escalable.

6 APLICACIÓN

Se ha desarrollado una primera versión funcional de la aplicación intermedia del sistema. Esta aplicación está pensada para ejecutarse en Windows y ha sido desarrollada en C++, utilizando SimConnect para la comunicación con Microsoft Flight Simulator y compilada con MSVC [9].

Además de la parte lógica, se ha diseñado una interfaz gráfica desarrollada con Qt 6, Qt Quick y QML [8]. La parte visual no se ha tratado como algo secundario, sino como una parte importante del desarrollo, ya que ayuda a comprobar el comportamiento del programa durante las pruebas, ver Fig. 3.

En concreto, la aplicación muestra el estado de la conexión con el simulador, junto con las variables de vuelo.

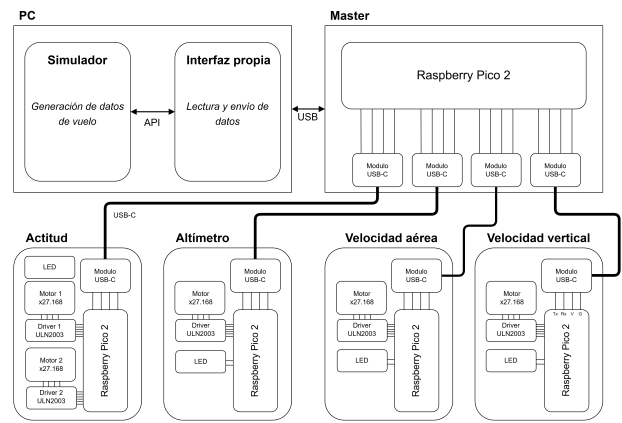


Fig. 2: Arquitectura del sistema basada en tres capas y una jerarquía master-slave.



Fig. 3: Interfaz visual de la aplicación.

La lógica se encapsula en un servicio de conexión que permite a la aplicación detectar automáticamente si Microsoft Flight Simulator está abierto, cerrado o si pierde conexión durante la ejecución. Esto es importante porque el simulador puede cerrarse, reiniciarse o dejar de enviar datos durante una sesión, y la aplicación debe poder recuperarse sin bloquear el sistema. El servicio ejecuta un bucle en un hilo independiente que intenta abrir una sesión con el simulador mediante `SimConnect_Open`. Si la conexión falla, espera 1 segundo y vuelve a intentarlo, si tiene éxito, entra en un ciclo continuo donde procesa todos los mensajes del simulador mediante `SimConnect_CallDispatch`, con una pausa de 100 ms entre iteraciones.

Código 1: Bucle principal de conexión con SimConnect

```
while (!stopRequested) {
    if (FAILED(SimConnect_Open(&hSimConnect, "
        Avionics Software", nullptr, 0, 0, 0))) {
        sleepInterruptible(std::chrono::seconds
            (1));
        continue;
    }

    connected = true;

    while (!stopRequested && connected) {
        hr = SimConnect_CallDispatch(hSimConnect,
            &Impl::dispatchProc, this);
    }
}
```

La recogida de datos de vuelo se realiza cuando

la conexión ya está establecida con Microsoft Flight Simulator mediante una solicitud periódica del objeto `SIMCONNECT_OBJECT_ID_USER`, que representa la aeronave del usuario. Los datos recibidos llegan en mensajes `SIMCONNECT_RECV_SIMOBJECT_DATA`, se extraen en la rutina asociada a `SimConnect_CallDispatch()`, y después se representa en la interfaz.

Para no enviar información irrelevante y evitar ruido, la aplicación recoge las variables principales en una sola línea de texto con el formato `FD:ias,alt,vsi,pitch,roll`. La aplicación tiene un hilo de envío hacia la Raspberry Pi Pico 2 master que trabaja cada 20 ms, pero solo transmite si detecta cambios o si han pasado 500 ms desde el último envío. Esto nos ayuda a reducir el tráfico por el puerto serie y nos permite saber también si la aplicación sigue activa. Esta lógica se recoge en el fragmento de código del Apéndice 20.

7 FIRMWARE

En el firmware se busca separar bien las funciones. La Raspberry Pi Pico 2 master recibe las variables de vuelo del PC y decide a qué instrumento debe ir cada dato, mientras que las Raspberry Pi Pico 2 slave se limitan a controlar su indicador. Gracias a esta separación, los módulos slave no necesitan saber nada de SimConnect ni del formato completo de la aplicación, solo reciben datos de movimiento, órdenes y generan las señales de control, como se muestra en el Apéndice 21.

7.1. Identificación de módulos y enlaces UART

Cada instrumento tiene un identificador fijo dentro del protocolo, la velocidad aérea usa el ID 1, la altitud el ID 2, la velocidad vertical el ID 3 y la actitud el ID 4. Es una solución sencilla y práctica, ya que cada módulo slave anuncia periódicamente su ID y, cuando el master lo recibe, lo asocia al puerto físico donde se ha conectado.

Código 2: Identificadores usados por los instrumentos

```
namespace link {
constexpr uint8_t BROADCAST = 0;
constexpr uint8_t AIRSPEED = 1;
constexpr uint8_t ALTITUDE = 2;
constexpr uint8_t VSI = 3;
constexpr uint8_t ATTITUDE = 4;
}
```

Aunque se utilicen conectores y cables USB-C entre las distintas placas, la comunicación entre las Raspberry Pi Pico 2 no usa el protocolo USB ni las líneas D+ y D- como bus diferencial. Los conductores se aprovechan como cables normales para una UART TTL a 115200 baudios. El enlace USB real solo existe entre el PC y la Raspberry Pi Pico 2 master.

La Raspberry Pi Pico 2 dispone de UART por hardware, pero los bloques UART0 y UART1 son compartidos y no permiten crear cuatro enlaces independientes [11]. Por este motivo implementa cuatro UART mediante PIO (*Programmable I/O*) [15], un subsistema programable capaz de generar señales UART por software/hardware asignando una UART a cada conector. Los módulos *slave*, al necesitar solo

un enlace, utilizan UART0 con GPIO16 como UART TX y GPIO17 como UART RX [10].

7.2. Protocolo de comunicación

La Raspberry Pi Pico 2 master lee las líneas recibidas desde la aplicación, interpreta el paquete `FD`: y guarda las variables de vuelo disponibles. A partir de ahí, no reenvía los valores tal como llegan del simulador, sino que los transforma a órdenes de movimiento. En los indicadores de velocidad aérea y velocidad vertical, basados en los motores X27.168, su rango de movimiento es de 0 a 1800 pasos. En el caso del indicador de actitud, se usan servomotores, controlados mediante PWM y limitados entre 0 y 180 grados, dejando unos márgenes mecánicos para no forzar los topes. Para el altímetro, el firmware mantiene la conversión de altitud y la integración dentro del protocolo. Por tanto, el firmware convierte magnitudes como nudos, pies, pies por minuto o grados en la orden correcta para cada tipo de motor, ver tabla 1.

El envío master-slave no se realiza con texto, sino con una trama binaria fija de 8 bytes para mantener un tamaño reducido y simplificar el parseo [13]. La estructura de la trama contiene una marca de inicio, el tipo de mensaje, el ID del módulo, un número de secuencia, un valor de 16 bits, un CRC simple y una marca de fin.

Código 3: Construcción de una trama master-slave

```
inline uint8_t crc(uint8_t type, uint8_t id,
                  uint8_t seq, uint8_t lo,
                  uint8_t hi) {
    return 0xC3u ^ type ^ id ^ seq ^ lo ^ hi;
}

const uint8_t lo = value & 0xFF;
const uint8_t hi = (value >> 8) & 0xFF;

put_byte(START);
put_byte(type);
put_byte(id);
put_byte(seq);
put_byte(lo);
put_byte(hi);
put_byte(crc(type, id, seq, lo, hi));
put_byte(END);
```

Los mensajes principales son `MSG_ID`, enviado por cada uno de los módulos slave para identificarse, `MSG_SETPPOINT`, usado por la master para indicar el objetivo del actuador, `MSG_SETPPOINT_AUX`, usado en el indicador de actitud para el segundo eje, y `MSG_LIGHTS`, que sirve para encender o apagar la iluminación de todos los instrumentos.

7.3. Funcionamiento de la master

El bucle principal de la master se ha diseñado para que ninguna tarea deje parada al resto. En una ejecución normal lee el USB del PC, comprueba el botón de luces, escucha mensajes de identificadores de los módulos slave y, si la aplicación sigue activa, envía los objetivos de movimiento de los actuadores. Si no llegan datos de la aplicación durante 1,5 s, valor equivalente a unos tres periodos consecutivos de keepalive, la master deja de reenviar setpoints.

Código 4: Estructura del bucle de la master

```

read_usb_telemetry(t);
check_light_button();
poll_slave_ids(ports);

const bool app_ok = t.last_ms &&
    now - t.last_ms <= APP_TIMEOUT_MS;

if (app_ok && now - last_send >= SEND_MS) {
    last_send = now;
    for (auto& p : ports)
        send_to_slave(p);
}

```

Los setpoints se envían cada 40 ms. El orden de envío depende del ID detectado en cada puerto. En el caso de la actitud, se manda además un segundo mensaje con el pitch mediante MSG_SETPOINT_AUX. Las luces tienen prioridad de respuesta porque el botón se comprueba antes del envío periódico de posiciones.

Código 5: Selección del dato enviado a cada slave

```

if (telemetry_setpoint_for(t, p.slave_id, value))
{
    write_frame(tx, p.seq, MSG_SETPOINT,
        p.slave_id, value);
}

if (p.slave_id == link::ATTITUDE &&
    telemetry_aux_setpoint_for(t, p.slave_id,
    value)) {
    write_frame(tx, p.seq, MSG_SETPOINT_AUX,
        p.slave_id, value);
}

```

El control de las luces se gestiona desde un botón conectado a la master. Cuando se detecta una pulsación, el estado cambia y se envía un MSG_LIGHTS con ID BROADCAST. Además, la master repite este mensaje cada 500 ms para que un módulo conectado después de arrancar reciba también el estado correcto.

7.4. Funcionamiento de los slaves

Todos los módulos slave comparten una estructura muy similar. Al encender, anuncian su ID, inicializan su actuador y después esperan tramas del master. En los módulos con X27.168 se realiza un calibrado contra su máximo mecánico, mientras que con los servomotores se fija una posición central inicial y se limita el recorrido por software. El parser comprueba el inicio, el final, el CRC y el ID del mensaje. Si el ID coincide con el del módulo, guarda el nuevo objetivo, si el mensaje es de luces y va dirigido a BROADCAST, actualiza el pin de iluminación.

Código 6: Parseo de mensajes en un slave

```

while (uart_is_readable(UART)) {
    if (!parser.feed(uart_getc(UART), f))
        continue;

    if (f.type == MSG_SETPOINT && f.id ==
        DEVICE_ID)
        X27_set_target(motor, f.value);
    else if (f.type == MSG_LIGHTS &&
        (f.id == BROADCAST || f.id ==
        DEVICE_ID))
        gpio_put(LIGHTS_PIN, f.value != 0);
}

```

El motor X27.168 no dispone de lectura absoluta de posición [12], por lo que cada arranque requiere de una calibración inicial. El firmware mueve la aguja hacia el tope durante más pasos de los que permite el recorrido teórico, fija esa posición como cero y, a partir de ahí, mantiene un modelo interno de la posición. Para evitar saltos bruscos, se realiza un filtrado y el movimiento se limita por velocidad y aceleración [14].

Durante el avance del proyecto se comprobó que el motor X27.168 no era adecuado para todos los indicadores. En el altímetro, el mecanismo requiere un giro continuo, sin limitación de máximo. En el indicador de actitud, además, al necesitar de dos servomotores, esto hace que se añada más peso, por lo que en esos dos módulos se sustituyó el motor paso a paso por servomotores sencillos. Esta solución permite controlar directamente el ángulo mediante PWM, simplifica mucho la programación y facilita centrar el instrumento en caso de pérdida de comunicación.

TABLA 1: ESCALADO PRINCIPAL DE VARIABLES DE VUELO.

Instrumento	Conversión aplicada
Airspeed	0–240 kt a pasos calibrados del X27.168
Altímetro	0–10000 ft, giro continuo de agujas.
VSI	-2000 a +2000 ft/min a pasos calibrados del X27.168.
Attitude	Roll limitado a ± 60 grados y pitch a ± 30 grados, ambos centrados.

En caso de pérdida de comunicación, los módulos slave aplican un timeout definido de 2s. Si durante ese intervalo de tiempo no reciben nuevos setpoints, los módulos con X27.168 apagan las bobinas del motor y mantienen internamente la posición actual. En los módulos con servomotores, el firmware puede volver a una posición centrada. Esto reduce consumo y calentamiento, además de evitar movimientos inesperados.

8 DISEÑO ELECTRÓNICO

El diseño electrónico se ha centrado en dos tipos de placas, la PCB master, mostrada en la Fig. 4 y una PCB base para los indicadores de los módulos slave, mostrada en la Fig. 5. El objetivo principal ha sido simplificar el cableado, dejar conectores modulares, alimentar los instrumentos desde la master y facilitar un montaje repetible.

8.1. PCB master

La PCB master contiene una Raspberry Pi Pico 2, cuatro conectores hacia los módulos slave, un botón de control de luces, un LED de estado de luces, un LED de encendido y

un condensador de filtrado en la alimentación. El LED de encendido depende directamente de VBUS y permite comprobar que la placa recibe alimentación. El LED de luces, en cambio, depende del GPIO14 y muestra el estado enviado a los instrumentos. El botón se conecta a GPIO15 y GND, usando la resistencia *pull-up* interna de la Pico, ver Tabla 2.

TABLA 2: COMPONENTES PRINCIPALES DE LA PCB MASTER.

Elemento	Función
Pico 2	Microcontrolador master y router de telemetría.
J1-J4	Conexión UART a módulos slave alimentación, GND, TX y RX.
Botón	Cambia el estado global de iluminación.
LEDs	Indican alimentación y estado de luces.
C1	Filtra VBUS ante cambios de consumo.

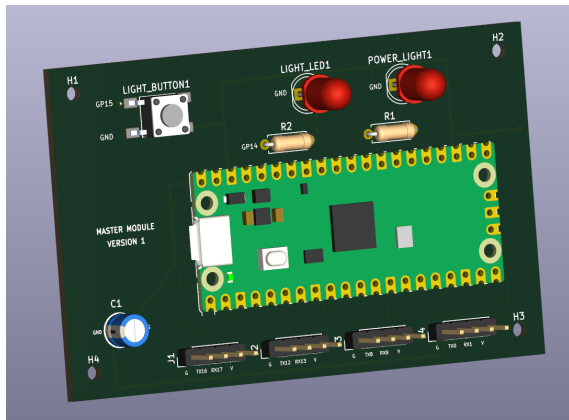


Fig. 4: PCB master.

8.2. PCB base de indicadores

La PCB de los módulos slave sirve como base para todos los indicadores, ya que todos comparten una Pico 2, un motor X27.168, una entrada UART desde la master y un circuito de iluminación. La diferencia principal está en el actuador, velocidad aérea y velocidad vertical usan motores X27.168, mientras que la actitud usan servomotores. El altímetro queda pendiente de rediseño con actuadores independientes. El indicador de actitud mantiene la misma idea, pero necesita dos motores, uno para el roll y otro para el pitch, ver Tabla 3.

Los servomotores no han sido incluidos en la placa porque se incorporaron más adelante en el proyecto, cuando la PCB ya estaba diseñada y orientada al uso del motor

TABLA 3: COMPONENTES PRINCIPALES DE LA PCB SLAVE.

Elemento	Función
Pico 2	Control local del instrumento.
J4	Entrada desde la master: VSYS, GND, RX y TX.
X27.168	Motor paso a paso que mueve la aguja.
LEDs	Iluminación interna del indicador.
P2N2222A	Conmuta los LEDs sin cargarlos sobre un GPIO.
Resistencias	Limitan corriente de LEDs y base del transistor.

X27.168. Las conexiones necesarias se realizaron soldando nuevos cables a los pines correspondientes de la Raspberry Pi Pico 2. Esta decisión evitó rehacer la placa completa y permitió validar rápidamente la solución antes de empezar el diseño 3D, aunque deja como mejora futura una revisión específica de la PCB con conectores dedicados para los servomotores.

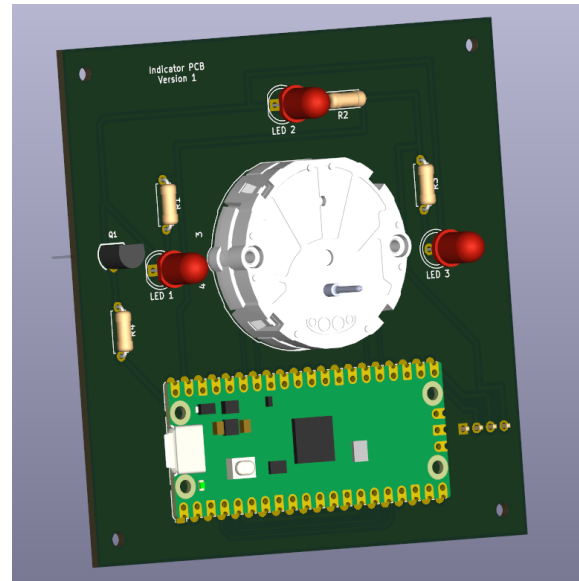


Fig. 5: PCB slave.

Un punto importante del diseño es el cruce entre TX y RX. En la master, el pin usado como TX sale hacia el conductor que llega al RX del módulo slave, y el TX del módulo slave vuelve al RX de la master.

Durante el diseño de las placas también ha sido necesario buscar y adaptar símbolos, huellas y modelos 3D que no estaban incluidos directamente en el programa de diseño. Esta fase no solo aporta estética al PCB sino que permite comprobar dimensiones, orientación de pines y compatibilidad con el montaje final.

9 DISEÑO 3D Y MECÁNICO

El diseño 3D se ha planteado como una parte importante del trabajo, no solo como una carcasa que cubra la electrónica, sino como un elemento que permita que el sistema sea cómodo, modular y fácil de usar. La idea principal ha sido crear una conexión y desconexión sencilla entre módulos, ocultar en la medida de lo posible el cableado, permitir el desmontaje para labores de mantenimiento y lograr un acabado compacto y profesional. De este modo, el diseño responde directamente al objetivo de construir una solución modular, donde cada instrumento pueda montarse, sustituirse o ampliarse de forma independiente.



Fig. 6: Sistema completo con los cuatro módulos indicadores acoplados al módulo master.

El sistema se divide en dos familias de piezas, los módulos de los indicadores y el módulo master. Los indicadores mantienen una estética común para que el panel completo tenga una apariencia simétrica y homogénea, mientras que el módulo master actúa como la base de alimentación, comunicación y control general. En la Fig. 6 se observan los módulos conectados a la master.

9.1. Módulo de indicadores

Todos los indicadores comparten una misma carcasa exterior. Esta decisión reduce el número de piezas distintas, simplifica la fabricación y hace que el sistema tenga una apariencia uniforme. La carcasa protege en su totalidad la electrónica. La parte trasera incorpora una ranura específica para el conector USB-C, de forma que quede encajado y no se mueva al conectar o desconectar el cable. Con ello se logra evitar un esfuerzo mecánico sobre las soldaduras, cables o la placa. Además, el uso de una carcasa común facilita futuras modificaciones de diseño, ya que cualquier mejora puede aplicarse directamente a todos los indicadores sin hacer un rediseño de cada módulo por separado.



Fig. 7: Módulos de vuelo.



Fig. 8: Módulo Velocidad Aérea.



Fig. 9: Módulo Velocidad Vertical.



Fig. 10: Módulo Altímetro.

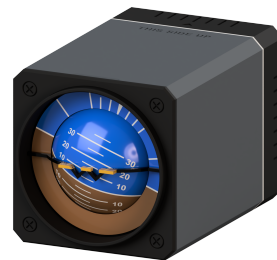


Fig. 11: Módulo Actitud.

En la parte trasera de cada módulo se han previsto cuatro imanes. Estos permiten realizar un acople al módulo master de forma rápida manteniéndolo fijo. La carcasa dispone de un espacio interior para guardar el exceso de cable entre el módulo slave y el master. Así, cuando el módulo se conecta, el cable queda escondido dentro, mejorando la sensación de un sistema integrado.

La parte frontal se cierra con una tapa con el diseño específico de cada instrumento, que oculta el interior y solo deja visibles la esfera y la aguja o el mecanismo correspondiente. Todas las uniones se realizan mediante insertos roscados y tornillos M2. Esta elección permite desmontar y volver a montar las piezas sin dañar el plástico y evita holguras durante el funcionamiento.

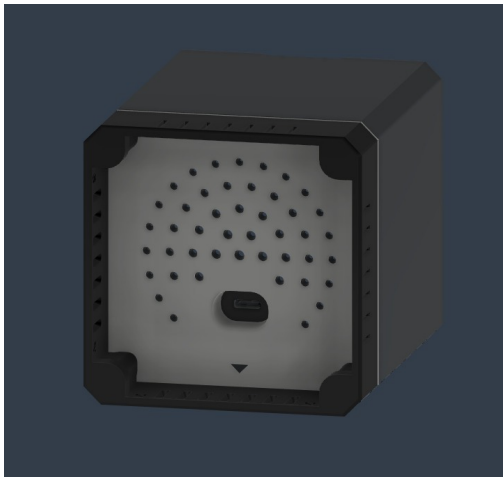


Fig. 12: Módulo parte trasera.

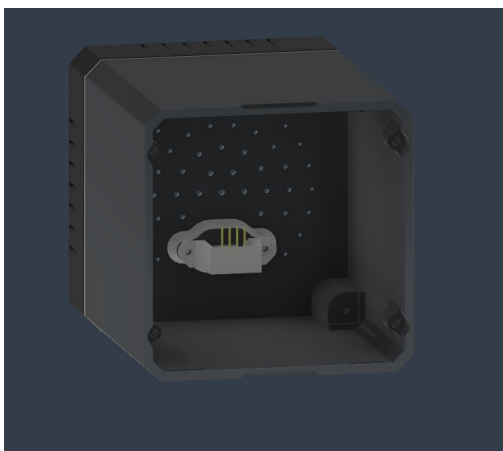


Fig. 13: Módulo parte interna.

La parte específica de cada módulo depende del instrumento. En los indicadores de velocidad aérea y velocidad vertical no se necesita un mecanismo muy complejo, el motor X27.168 se conecta a la aguja mediante una pequeña columna de transmisión. En cambio, el altímetro y el indicador de actitud necesitan soluciones mecánicas propias.

El altímetro fue el módulo que generó la mayor dificultad. El planteamiento inicial consistía en mover varias agujas mediante un único motor. Para ello era necesario implementar una transmisión mecánica entre escalas, donde una vuelta completa de la aguja de cientos produjera una décima de vuelta en la aguja de miles, y una vuelta completa de la aguja de miles produjera una décima de vuelta en la de decenas de miles.

Durante las pruebas se comprobó que la solución propuesta no era viable con los motores disponibles. La reducción que requerían las agujas aumentaba la fricción del conjunto y pedía un par por encima del que puede proporcionar el motor usado. Además, las tolerancias de impresión 3D, el tamaño reducido de los engranajes, el material de impresión y el ajuste entre ejes generaban holguras y rozamientos que impedían que el módulo se moviese correctamente.

La solución más adecuada sería usar dos motores independientes, quitando así, la necesidad de una reducción mecánica.

Debido al tiempo tan ajustado, el altímetro no se completó a nivel mecánico dentro del prototipo final. Sí se desa-

rolló su firmware y se mantuvo su integración lógica dentro del protocolo. El módulo quedó fuera de la validación mecánica final.

El indicador de actitud usa dos servomotores colocados de manera compacta. Un motor se sitúa sobre el otro para permitir que el primero produzca un giro lateral, mientras que el segundo, situado encima del primero de forma lateral, genera el desplazamiento vertical y horizontal de forma simultánea. Este mecanismo permite disponer de dos grados de libertad y deja margen de movimiento suficiente para evitar choques entre piezas, motores y cables.

9.2. Módulo master

El módulo master es mecánicamente más sencillo, pero tiene una función importante. Su carcasa contiene la PCB master, organiza la conexión de los cuatro módulos y deja accesibles elementos de control del sistema. En la parte superior está integrado el botón encargado de activar o desactivar la iluminación, junto con los dos LEDs indicadores que permiten comprobar visualmente el estado de alimentación y de luces.

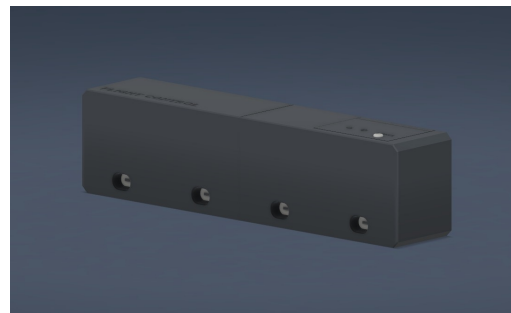


Fig. 14: Módulo master.

La pieza del master también utiliza insertos roscados y tornillos M2 para mantener la PCB y la carcasa firmes. Debido a la longitud del módulo master, el diseño se divide en dos piezas que se imprimen por separado, ya que el tamaño supera el volumen de impresión de muchas impresoras 3D. Una vez montadas, las dos piezas forman una base sobre la que se acoplan magnéticamente todos los módulos.

El resultado global busca que el usuario pueda conectar el sistema con muy pocos pasos, colocar cada instrumento en su posición, ocultar el cable sobrante y mantener un sistema firme y robusto.



Fig. 15: Sistema completo.

10 PLAN DE VALIDACIÓN Y MÉTRICAS

La validación final se realizó mediante pruebas controladas una vez el prototipo ya estaba montado. Para evitar depender de Microsoft Flight Simulator durante la toma de datos, se utilizó un script auxiliar en Python que envía manualmente objetivos de movimiento a la placa por un puerto serie USB.

La validación se centró en los módulos funcionales integrados en el prototipo. El altímetro no se incluye en la validación mecánica final debido a la limitación explicada anteriormente.

Precisión de indicación: Se enviaron valores de referencia y se comparó la posición visual de la aguja con el valor esperado. La lectura se realizó de forma manual observando la escala física del instrumento. Los resultados se muestran en la siguiente tabla 4:

TABLA 4: RESULTADOS DE PRECISIÓN DE INDICACIÓN.

Prueba	Instr.	Objetivo	Lectura	Error abs.
1	IAS	40 kt	50 kt	10 kt
2	IAS	160 kt	162 kt	2 kt
3	VSI	-1000 ft/min	-1000 ft/min	0 ft/min
4	VSI	1000 ft/min	900 ft/min	100 ft/min
5	Roll	30°	30°	0°

Latencia: La latencia se estimó a partir de los tiempos configurados en el firmware, es decir, el módulo master envía datos cada 40 ms, el bucle principal se ejecuta aproximadamente cada 1 ms, la trama interna ocupa 8 bytes a 115200 baudios y los slaves actualizan el control del motor cada 200 μ s. Con estos valores, la latencia esperada queda en torno a 65–75 ms, como se muestra en la tabla 5.

Estos valores son coherentes con la observación durante las pruebas, donde el inicio del movimiento se observó prácticamente instantáneo después de enviar los datos.

TABLA 5: ESTIMACIÓN DE LATENCIA.

Prueba	Salto aplicado	Latencia
1	IAS 0 $\rightarrow$ 120 kt	69 ms
2	IAS 120 $\rightarrow$ 0 kt	70 ms
3	VSI 0 $\rightarrow$ 1500 fpm	70 ms
4	Pitch 0 $\rightarrow$ 20°	68 ms
5	Roll 0 $\rightarrow$ 45°	69 ms

Suavidad: La suavidad se evaluó mediante secuencias de movimiento y una valoración visual de 0 a 3, donde 0 son movimientos deficientes y 3 movimientos suaves. Los resultados se recogen en la tabla 6.

TABLA 6: RESULTADOS DE SUAVIDAD DE MOVIMIENTO.

Prueba	Movimiento	Nota
1	IAS 0–240 kt	3/3
2	VSI -2000–2000 fpm	3/3
3	Pitch -20–20°	2/3
4	Roll -45–45°	2/3
5	Combinado	2/3

Los indicadores accionados por motor X27, como la velocidad aérea y la velocidad vertical, muestran el movimiento más suave. El indicador de actitud tuvo una suavidad inferior, seguramente debido al peso de las piezas 3D que soportan los motores y tolerancias, aunque sin oscilaciones graves ni bloqueos.

Robustez: La robustez se comprobó sometiendo al sistema a valores fuera de rango, vuelta a valores normales, pérdida temporal de datos, desconexiones y reconexiones y el botón de luces. Los resultados se muestran en la tabla 7.

TABLA 7: RESULTADOS DE ROBUSTEZ.

Prueba	Caso evaluado	Resultado
1	Valores fuera de rango	OK
2	Vuelta a valor normal	OK
3	Timeout sin datos	OK
4	Reconexión de slave	OK
5	Botón de luces	OK

En todos los casos el sistema mantuvo un comportamiento estable. Los valores fuera de rango son limitados por código, el sistema volvió correctamente a valores normales y la pérdida temporal no produce movimientos descontrolados.

11 CONCLUSIONES

En este trabajo se ha diseñado y desarrollado una solución modular para visualizar variables de vuelo de Microsoft Flight Simulator mediante instrumentos físicos. El resultado final no pretende competir con sistemas comerciales de alta gama, pero sí demuestra que es posible construir una alternativa funcional, de bajo coste y escalable combinando software y hardware.

Las pruebas realizadas permiten validar la arquitectura general del sistema. La placa master recibe las variables de vuelo, las procesa y las reparte correctamente a los módulos slave, manteniendo cada instrumento aislado a nivel de control. Esta separación ha sido una de las decisiones más acertadas del proyecto, ya que ha permitido desarrollar, probar y corregir cada indicador de forma independiente. Además, facilita futuras ampliaciones, ya que un módulo puede modificarse o sustituirse sin rediseñar todo el sistema.

A nivel de precisión, los resultados han sido adecuados para un prototipo físico de bajo coste. El indicador de velocidad aérea muestra un error máximo de 10 kt en las pruebas realizadas, mientras que el indicador de velocidad vertical mostró un error máximo de 100 ft/min. El indicador de actitud mostró correctamente los valores enviados. Estos resultados demuestran que el sistema representa correctamente las variables principales, aunque también muestran que la calibración y la mecánica son factores muy importantes.

Durante el desarrollo aparecieron limitaciones mecánicas que obligaron a adaptar decisiones iniciales. El caso más relevante fue el altímetro, cuyo planteamiento inicial pretendía accionar varias agujas mediante un único motor. En la práctica el par que requería y la complejidad de la reducción hicieron inviable esta solución con los motores disponibles. Aunque el firmware del altímetro fue desarrollado, el módulo no quedó completado a nivel mecánico. Esta limitación nos confirma la importancia de validar la parte mecánica siempre antes de cerrar completamente el diseño.

Otra limitación que fue corregida a tiempo fue el uso de los motores paso a paso X27.168. Aunque funcionan correctamente para indicadores simples de aguja como la velocidad aérea y la velocidad vertical, se comprobó que no eran adecuados para el altímetro ni la actitud debido al recorrido, el par disponible y la carga mecánica de estos ins-

trumentos. La sustitución por servomotores logró resolver esta limitación.

La metodología seguida ha sido incremental y se ha ido adaptando al avance real del proyecto. Primero se desarrolló la aplicación de PC para obtener las variables de vuelo mediante el SDK de Microsoft Flight Simulator 2020. Después se implementaron los firmwares de los módulos y se validó la comunicación entre la placa master y los módulos slave mediante USB-C. Una vez comprobado el funcionamiento básico, se diseñaron las PCBs y se avanzó el diseño 3D. Los mayores retrasos aparecieron en la aplicación y el firmware, por la necesidad de comprender el SDK y definir la comunicación. El diseño electrónico fue más asumible, mientras que el diseño 3D fue el reto más complejo por los diseños mecánicos y cambios de hardware necesarios. Este proceso ha demostrado la importancia de adaptarse a los imprevistos durante la integración de un sistema físico.

A nivel personal, el trabajo ha permitido integrar conocimientos de programación, electrónica, comunicación entre dispositivos, diseño de PCBs, soldadura, firmware y diseño mecánico. También ha mostrado que en un sistema físico los errores no dependen únicamente del código, sino también del montaje, los conectores, la alimentación, el par disponible, las tolerancias de fabricación y el ensamblaje de las piezas. Esta parte práctica ha sido una de las más exigentes del proyecto, pero también una de las que más aprendizaje ha aportado.

Como trabajo futuro, sería necesario realizar una revisión de la PCB que incluya conectores específicos para servomotores, mejorar el sistema de calibración mediante mecanismos dedicados y refinar algunas piezas 3D para facilitar el montaje y desmontaje. También sería necesario rediseñar el altímetro usando dos motores en vez de uno, lo que permitiría resolver la limitación mecánica. Aun con estas mejoras pendientes, el proyecto deja una base funcional, validada y escalable sobre la que continuar desarrollando un panel de instrumentos más completo y preciso.

USO DE LA IA GENERATIVA

Durante la elaboración de este trabajo se han utilizado herramientas de IA generativa de forma limitada y únicamente como apoyo auxiliar. En concreto, se ha hecho uso de ChatGPT para la revisión de redacción, corrección ortográfica y la mejora del estilo del texto.

Estas herramientas no se han utilizado para generar contenidos evaluables del trabajo, como el planteamiento metodológico, el diseño de la solución, la realización de pruebas, el análisis o interpretación de resultados, ni la elaboración de conclusiones.

REFERENCIAS

- [1] Microsoft Flight Simulator SDK, “Introduction,” *SimConnect Documentation*. <https://docs.flightsimulator.com/html/Introduction/Introduction.htm> [Consultado: 02-Mar-2026].
- [2] Raspberry Pi Ltd., “Raspberry Pi Pico 2 Datasheet.” <https://datasheets.raspberrypi.com/pico/pico-2-datasheet.pdf> [Consultado: 02-Mar-2026].
- [3] Agnalim, “Paneles de simuladores de vuelo - El comienzo con Arduino.” <https://www.agnalim.com/es/post-panel-simulador-vuelo-casero-comienzos.html> [Consultado: 02-Mar-2026].
- [4] SimVimX, “Instrument Panel Program.” https://simvimx.com/panel_constr.html [Consultado: 02-Mar-2026].
- [5] SimVimX, “SimVimPanel Setup Guide.” https://simvimx.com/svp_guide.html [Consultado: 02-Mar-2026].
- [6] Logitech G, “Flight Instrument Panel – LCD Multi-Instrument Controller.” <https://www.logitechg.com/en-eu/shop/p/light-simulator-instrument-panel> [Consultado: 02-Mar-2026].
- [7] PilotMall, “Instrumentos de aeronaves: comprensión de los indicadores de vuelo clave.” <https://www.pilotmall.com/es/blogs/news/the-complete-guide-to-aircraft-instruments-more-than-just-the-6-pack> [Consultado: 05-Mar-2026].
- [8] Qt Documentation, “qmake Manual.” <https://doc.qt.io/qt-6/qmake-manual.html> [Consultado: 23-Mar-2026].
- [9] Microsoft Learn, “Walkthrough: Compile a C++/CLI program on the command line.” <https://learn.microsoft.com/en-us/cpp/build/walkthrough-compiling-a-cpp-cli-program-on-the-command-line?view=msvc-170> [Consultado: 23-Apr-2026].
- [10] Raspberry Pi, “pico-examples: PIO examples.” <https://github.com/raspberrypi/pico-examples/tree/master/pio> [Consultado: 15-May-2026].
- [11] Raspberry Pi Ltd., “Hardware UART.” *Raspberry Pi Pico SDK Documentation*. https://www.raspberrypi.com/documentation/pico-sdk/hardware.html#group_hardware_uart [Consultado: 15-May-2026].
- [12] Juken Swiss Technology Ltd., *X27 Stepper Motor Specification*. http://agr.ucoz.lv/files/SP_X27_e_C.pdf. [Consultado: 15 de mayo de 2026].
- [13] MAVLink, “Packet Serialization.” *MAVLink Guide*. <https://mavlink.io/en/guide/serialization.html> [Consultado: 15-May-2026].
- [14] NIST/SEMATECH, “Single Exponential Smoothing.” *e-Handbook of Statistical Methods*. <https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc431.htm> [Consultado: 15-May-2026].
- [15] Raspberry Pi Ltd., “Hardware PIO.” *Raspberry Pi Pico SDK Documentation*. https://www.raspberrypi.com/documentation/pico-sdk/hardware.html#group_hardware_pio [Consultado: 22-May-2026].

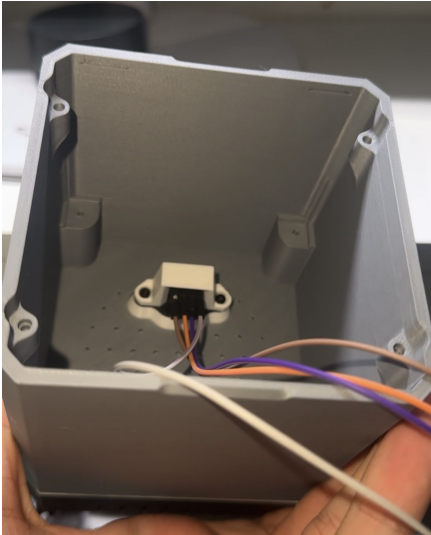
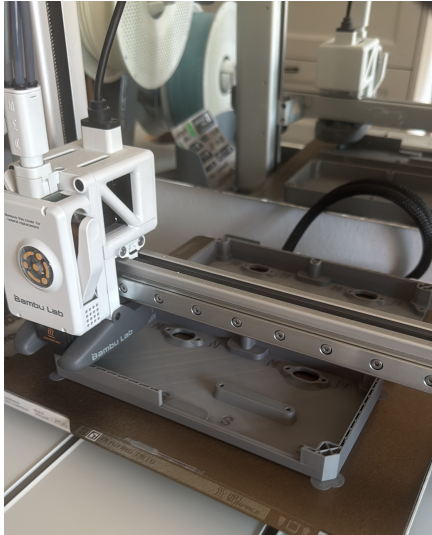
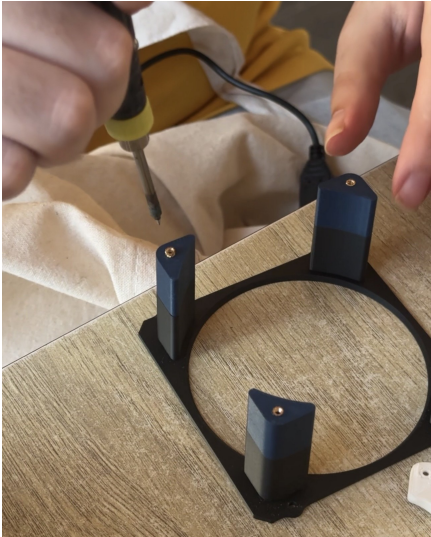
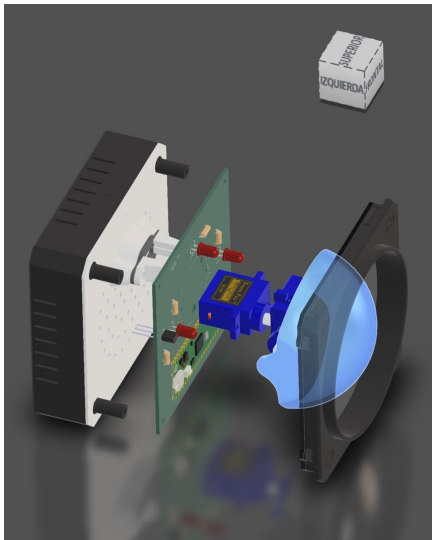
APÉNDICE

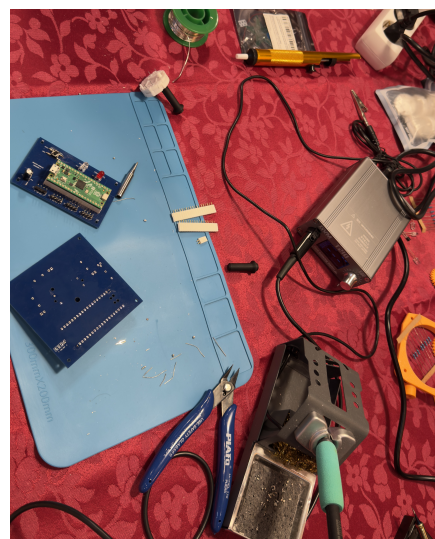
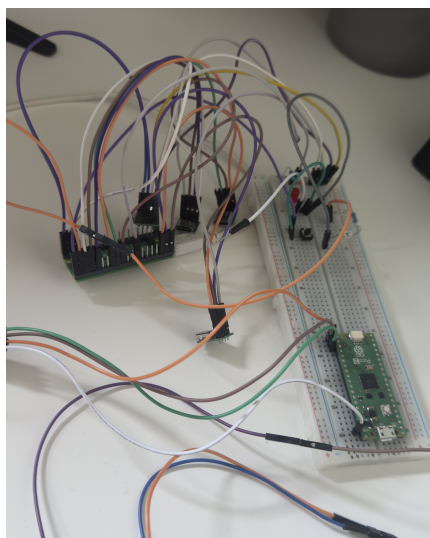
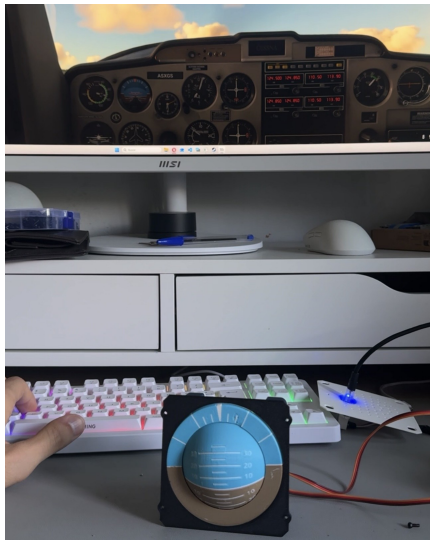
A TABLA DE MATERIALES

TABLA 8: TABLA DE MATERIALES.

Componente	Cantidad	Precio
Raspberry Pi Pico 2	5	39€
Motor X27.168	2	4€
Servomotores	3	2€
Módulo USB-C hembra	8	2€
Cable USB-C	4	10€
LEDs	14	1€
Resistencias	18	1€
Botón	1	0,1€
Transistores	4	0,5€
Condensador	1	0,3€
Jumpwires	41	7€
PCBs	15	31€
Filamento PLA mate 1Kg	2	30€
Total		127,9€

B PROCESO DE FABRICACIÓN Y MONTAJE





C ESQUEMÁTICOS

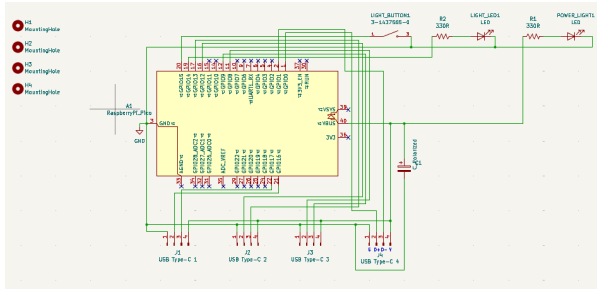


Fig. 16: Esquemático electrónico del módulo master.

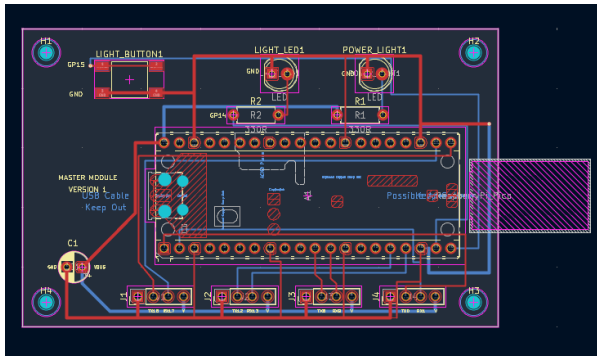


Fig. 17: Diseño de la PCB del módulo master.

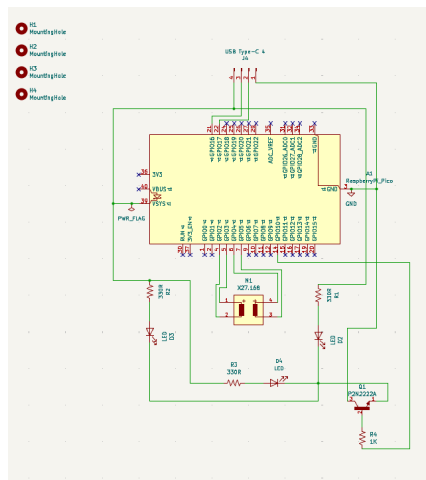


Fig. 18: Esquemático electrónico del módulo slave.

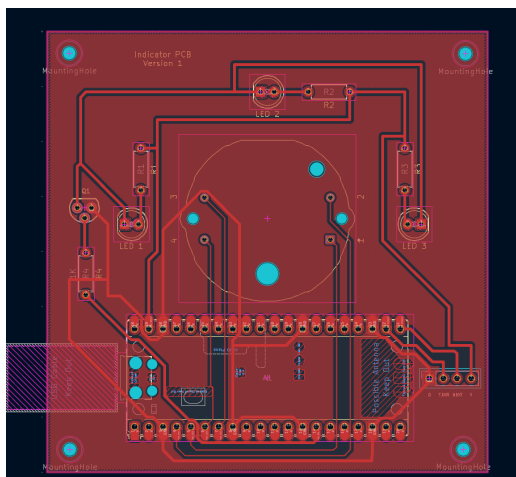


Fig. 19: Diseño de la PCB del módulo slave.

D MAPA DE PINES Y PROTOCOLOS

En esta sección se resumen las conexiones principales entre la placa master y los módulos slave. El enlace PC-master se realiza mediante USB CDC, y los conectores USB-C entre placas se utilizan como interfaz para UART TTL, alimentación y GND.

TABLA 9: PINES PRINCIPALES DEL MÓDULO MASTER.

Función	GPIO	Pin	Uso
Botón luces	GP15	20	Entrada con <i>pull-up</i> ; pulsador a GND.
LED luces	GP14	19	Indica el estado de iluminación.
USB Pico	—	—	Comunicación PC-master por USB CDC.
USB-C1	TX GP16 / RX GP17	21 / 22	UART TTL hacia slave.
USB-C2	TX GP12 / RX GP13	16 / 17	UART TTL hacia slave.
USB-C3	TX GP8 / RX GP9	11 / 12	UART TTL hacia slave.
USB-C4	TX GP0 / RX GP1	1 / 2	UART TTL hacia slave.

TABLA 10: PINES PRINCIPALES DE LOS MÓDULOS SLAVE.

Función	GPIO	Pin	Uso
UART0 TX	GP16	21	Envío hacia la master.
UART0 RX	GP17	22	Recepción desde la master.
Luces	GP10	14	Control del driver de iluminación.
X27 IN1	GP3	5	Entrada del driver del motor.
X27 IN2	GP2	4	Entrada del driver del motor.
X27 IN3	GP5	7	Entrada del driver del motor.
X27 IN4	GP4	6	Entrada del driver del motor.
Servo pitch	GP2	4	PWM en el módulo de actitud.
Servo roll	GP3	5	PWM en el módulo de actitud.

TABLA 11: TRAMA BINARIA MASTER-SLAVE.

Byte	Campo	Descripción
0	START	Inicio de trama: 0xA5.
1	type	Tipo de mensaje.
2	id	Módulo origen o destino.
3	seq	Contador de secuencia.
4	value_lo	Byte bajo del valor <code>int16_t</code> .
5	value_hi	Byte alto del valor <code>int16_t</code> .
6	crc	0xC3 XOR type, id, seq, lo y hi.
7	END	Fin de trama: 0x5A.

TABLA 12: IDS Y TIPOS DE MENSAJE DEL PROTOCOLO.

Campo	Valor	Uso
ID 0	BROADCAST	Mensaje para todos los módulos.
ID 1	AIRSPEED	Indicador de velocidad aérea.
ID 2	ALTITUDE	Altímetro.
ID 3	VSI	Indicador de velocidad vertical.
ID 4	ATTITUDE	Indicador de actitud.
0x01	MSG_ID	Identificación slave-master.
0x10	MSG_SETPOINT	Consigna principal.
0x11	MSG_SETPOINT_AUX	Segunda consigna para actitud.
0x20	MSG_LIGHTS	Control de iluminación.

TABLA 13: CONVERSIÓN PRINCIPAL DE VARIABLES A CONSIGNAS.

Variable	Conversión
IAS	0-240 kt a pasos calibrados del X27.168.
Altitud	0-10000 ft, giro continuo de agujas.
VSI	-2000 a +2000 ft/min, pasos calibrados del X27.168.
Pitch	-30 a +30 grados, centrado en 900.
Roll	-60 a +60 grados, centrado en 900.

E LÓGICA DE ENVÍO SERIE Y KEEPALIVE

```
constexpr DWORD kSendIntervalMs = 20;
constexpr DWORD kKeepAliveMs = 500;

bool changed = ias != lastIas || altitude !=
    lastAltitude ||
    vsi != lastVsi || pitch != lastPitch ||
    roll != lastRoll;

if (changed || nowMs - lastWriteMs >= kKeepAliveMs)
{
    WriteFile(hPort, msg.constData(), msg.size(), &
        written, nullptr);
    lastWriteMs = nowMs;
}

Sleep(kSendIntervalMs);
```

Fig. 20: Envío serie solo cuando hay cambios o keepalive periódico.

F SEPARACIÓN ENTRE TELEMETRÍA Y CONTROL

```
switch (slave_id)
{
case ID_AIRSPEED:
    send_value(airspeed_steps(t.airspeed_kt));
    break;
case ID_VERTICAL_SPEED:
    send_value(vertical_speed_steps(t.vertical_speed_fpm));
    break;
case ID_ATTITUDE:
    send_attitude(t.pitch_deg, t.roll_deg);
    break;
}
```

Fig. 21: La master envía comandos simples para cada slave.

G DIAGRAMA DE GANTT

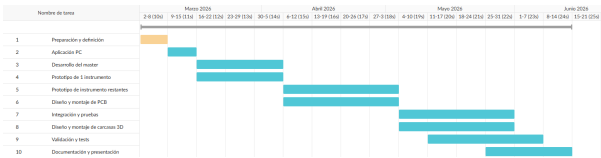


Fig. 22: Diagrama de Gantt