Escuela Superior de
Ingeniería y Tecnología
Sección Ing. Industrial

Grado en Ingeniería Electrónica Industrial y Automática

Desarrollo de simulador de vuelo basado en herramientas *OpenSource*

Trabajo Fin de Grado

Autor:

Augusto Samuel Hernández Martín

Tutores:

Alejandro José Ayala Alfonso

Beatriz Rodríguez Mendoza

Julio 2019

A mis padres, familiares
y amigos,
por creer siempre
en mi

Agradecimientos

Gracias a mis tutores Alejandro Ayala y Beatriz Rodríguez por la dedicación y el apoyo prestado durante este proyecto. Agradecer la libertad concedida, fruto de la confianza depositada en mí. Gracias a Iván Rodríguez y Delfín Díaz por sus consejos en la realización de las PCB y su colaboración para la producción de los paneles de la cabina. Agradecer también el apoyo prestado por Fernando Rosa y Oswaldo Bernabé, por sus consejos en la toma de decisiones para la mejora del diseño del proyecto.

A mis familiares, en especial a mis padres y mi hermana Silvia. A Priscila, gracias por tus aportaciones, por escucharme día a día y aguantar mis quejas sobre los problemas sobrevenidos así como tu ayuda para solucionarlos. Sin tu apoyo este proyecto no habría recibido tantas horas de dedicación.

No podría olvidarme de mis amigos. Gracias a Davinia y mis compañeros de universidad: Carlo, Camilo, Iván, Jorge y Josué; por los buenos momentos vividos durante estos 4 años.

Índice

Resumen

El objetivo del presente Trabajo de Fin de Grado es el desarrollo de una cabina de vuelo totalmente funcional de aviones ligeros monomotor como la Cessna 172.

Su funcionamiento se basa en el empleo de microcontroladores Atmel similares a los que emplean los dispositivos Arduino Nano y Arduino Mega para, con ellos, comunicarse con el ordenador y el resto de dispositivos que conforman la cabina.

Se emplea el *software* de simulación de Microsoft, el *Flight Simulator X*, y un programa intermedio llamado *Link2FS* que realiza la comunicación con los microcontroladores ATmega por puerto *serial*. Con ello, junto con los dispositivos mecánicos fabricados mediante impresora 3D, se consigue replicar de forma totalmente fiel los paneles del avión.

Abstract

The aim of this Final Degree Project is focused on designing and building an airplane cockpit similar to Cessna 172, based on Atmel microcontroller.

Atmel microcontroller are used to communicate with the computer through serial port and with the rest of the cockpit devices through different communication buses (*SPI*,I^2C, etc.).

Microsoft simulation software (*Flight Simulator X*) is used together with a software called *Link2FS* that makes the communication with these microcontrollers by serial port. With this, together with the mechanical devices designed and produced with a 3D printer, the airplane panel can be replicated in a faithful way.

MEMORIA

Capítulo I. Introducción general. Objetivos

I.1 Introducción general

Desde siglos atrás el hombre ha soñado con volar. Esta historia se remonta a los días en que éste observó y deseó imitar el comportamiento de las aves. Así, basándose en la antigua mitología griega, se observan pinturas de humanos *mitad hombre-mitad ave* capaces de elevarse en el aire.

La historia moderna de la aviación comienza en el siglo XVIII cuando el hombre empezó a experimentar con los primeros globos aerostáticos, para posteriormente construir los primeros dirigibles y, finalmente, el primer prototipo de avión funcional en el siglo XX a manos de los hermanos Wright. Estos dos famosos ingenieros fueron capaces de sentar las bases de la aviación moderna. Con su primer avión, el Flyer I (*Figura I.1*), desarrollaron los mecanismos de viraje, es decir, el control que permitía a éste poder realizar giros en el aire. Su principal problema se debía a su poca capacidad para realizar una maniobra vital: el despegue, pues necesitaba un gran impulso externo, generalmente proporcionado por una catapulta.

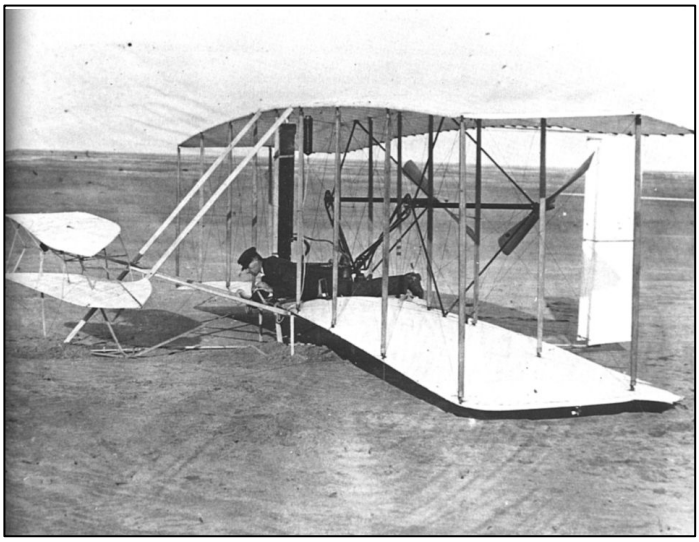


Figura I.1. Flyer I.

Con el paso de los años, los avances técnicos en electrónica, mecánica y ciencias de los materiales, se hizo posible la construcción de modelos mucho más robustos y ligeros, con motores más potentes, capaces de transportar un número importante tanto de personas como de mercancía. Bajo esta premisa surgieron diversos fabricantes: desde las conocidas Boeing, AirBus o ATR Aircraft, hasta las

destinadas a ocio o transporte privado como Cessna (*Figura I.2*) o Bombardier, respectivamente.

Figura I.2. Avioneta Cessna 172.

El elevado número de instrumentos así como la sofisticación de los aviones hace necesaria una formación cada vez más completa en el manejo de los mismos. Por ello, se requieren muchas horas de entrenamiento antes de que un piloto sea capaz de volar uno de estos aparatos. Además, el cambio de un avión a otro, a pesar de ser del mismo fabricante, obliga a una nueva especialización. Bajo este contexto, con el objetivo de dotar de mayor seguridad al transporte aéreo, se desarrollaron por parte de las entidades gubernamentales y los organismos de aviación una serie de permisos denominados habilitaciones, con los cuales se demuestra que un piloto es apto para manejar un modelo concreto de aeronave.

Estos cambios requirieron por parte de los fabricantes el desarrollo de equipos de entrenamiento a los que se les conoce como simuladores de vuelo. Estos dispositivos permiten poner a prueba las habilidades del piloto, sometiéndolo a situaciones extremas, sin poner en riesgo su vida ni la de los demás. Este desarrollo siempre ha ido de la mano de los avances en informática.

En este campo, cada fabricante desarrollaba su propio simulador, coexistiendo en una academia varios de estos equipos. A modo de ejemplo se tiene el simulador TL39 3-Do (*Figura I.3*) que se encuentra en Moscú y es un sistema *Full Motion Flight Simulator* o el simulador de vuelo *Canarias* desarrollado por *Global Training Aviation* en colaboración con el productor de las aeronaves ATR-Aircraft.

Figura I.3. Simulador TL39 3-Do.

Cada uno de estos equipos, como se ha comentado, emplea un *software* específico. Es por ello que empresas como Microsoft lanzaron un *software* capaz de unificar distintos modelos de aviones pertenecientes a diferentes compañías. Así surgió *Microsoft Flight Simulator* [1], capaz de ser ejecutado en un PC y al alcance de cualquier academia. A este programa se unieron otros como el desarrollado por *Laminar Research* conocido como *X-Plane* [2] o el *Prepar3D* [3]. Estos últimos cuentan con la aprobación de la Administración Federal de Aviación de EEUU y las Direcciones de Aeronáutica Civil de países como Francia o Italia.

Estos *softwares* se hicieron extensibles a usuarios aficionados a la aviación, surgiendo así la conocida plataforma para simulación de vuelos llamada IVAO [4]. Este entorno libre permite a usuarios de diferentes países interactuar realizando vuelos simultáneos de forma *online*. Funciona como una academia de pilotos, en la que los usuarios según van adquiriendo horas de vuelo -con la consiguiente experiencia- subirán de rango, pudiendo pilotar aparatos más complejos en lugares de mayor tránsito. A esto le acompaña la posibilidad de ejercitarse como controlador aéreo bajo esta plataforma, dotando a la misma de total realismo.

I.2 Objetivos

El objetivo del presente proyecto se ha centrado en el desarrollo del conjunto de instrumentos indicadores para una cabina de simulación aérea. Se ha tomado como referencia la correspondiente a una Cessna 172 (*Figura I.4*), de la cual se han desarrollado los siguientes instrumentos:

- Altímetro.
- Indicador de velocidad aérea.
- Indicador de revoluciones del motor.

- Horizonte artificial.
- Indicador de rumbo.
- Coordinador de giro.
- Medidor de nivel de combustible.
- Indicador de presión de aceite.
- Reloj de temperatura del motor.
- Panel de radio, comunicaciones y transpondedor.
- Panel de piloto automático.
- Mandos de luces y control de la aviónica principal.
- Conjunto de palancas de gases y acelerador.
- Módulo de control de Flaps y tren de aterrizaje.

Figura I.4. Cabina real de una Cessna 172.

Se ha prescindido del volante (*yoke*) y los pedales de control de timón debido a la elevada complejidad que conlleva su desarrollo. Se emplean dos soluciones comerciales de la marca *Saitek* [5] para completar una cabina totalmente funcional. No obstante, en un futuro se prevé implementarlos.

Los instrumentos son controlados por varios microcontroladores de la familia ATmega [6] conectados entre sí por un bus I^2C (*Figura I.5*). Como se observa en el

diagrama de bloques de dicha figura, hay un dispositivo al que llamamos *maestro* que se conecta al ordenador, donde se ejecuta el *software* de simulación (FSX y *Link2FS*), mediante conexión *serial* USB. Se implementa haciendo uso de un ATmega 2560 por ser el de mayor potencia (se explicará con mayor detalle en el *Capítulo II*) que recibe todos los parámetros de vuelo.

A este *maestro*, se conectan dos microcontroladores ATmega 328 [7], a los que denominaremos *Esclavo I* y *Esclavo II*. El primero de ellos se encarga de controlar los motores *stepper* que simulan los *gauges*[1] de altitud y rumbo explicados en el *Capítulo IV*. El *Esclavo II* se emplea para simular los indicadores relacionados con el aterrizaje instrumental.

Por otro lado, el ATmega 2560 funciona de modo bidireccional, obteniendo los datos del ordenador -como ya se comentó- y leyendo las modificaciones realizadas desde los paneles de radio o de piloto automático que éste controla para ser enviadas al FSX.

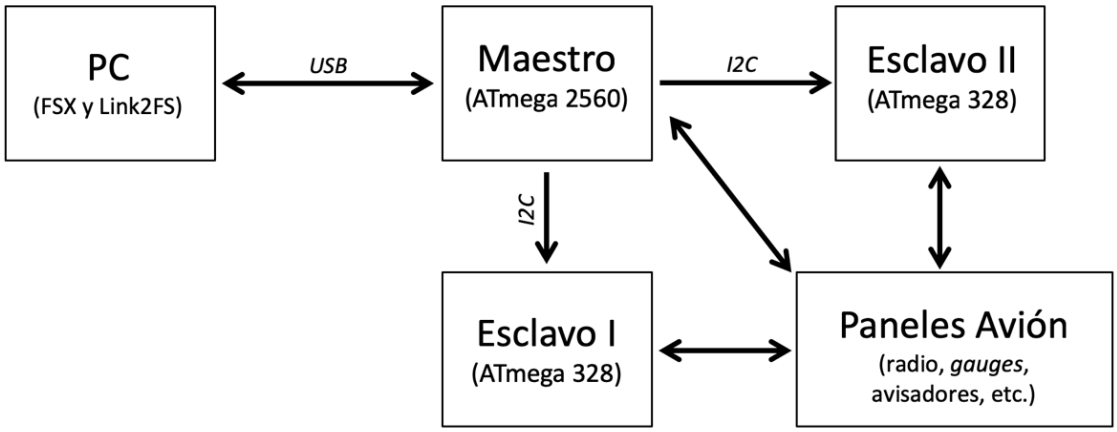


Figura I.5. Diagrama de bloques general del dispositivo.

El usuario podrá emplear el simulador de un modo totalmente funcional, conociendo el estado del avión para actuar sobre los equipos de control en tiempo real.

[1] ***Gauges:*** Se trata de un medidor analógico que consta de una aguja que gira marcando un valor en una escala graduada. Su funcionamiento es similar al del reloj analógico.

I.3 Estructura general del trabajo

La presente memoria se desarrollará en 7 capítulos.

El primero de ellos ofrece una visión general de este proyecto, los objetivos del mismo y la forma en que se estructura.

En el *Capítulo II*, se describen los dispositivos de entrada y salida empleados en el proyecto, tanto pulsadores, interruptores, motores, servomotores, etc. así como los microcontroladores de Atmel [6] empleados con sus características principales.

En el tercero, se detallarán las herramientas empleadas, comenzando por el entorno de desarrollo de Arduino y explicando posteriormente los distintos programas de simulación aérea y comunicación empleados (FSX y *Link2FS*). Se describe también, la impresora 3D Hephestos Prusa i3 utilizada para producir los prototipos de piezas mecánicas y marcos embellecedores.

El *Capítulo IV* se centra en analizar cada uno de los instrumentos del simulador, describiendo su parte mecánica y electrónica así como la intercomunicación entre los diferentes dispositivos con el *maestro*.

El capítulo quinto explica el algoritmo de recepción de los datos por parte del dispositivo *maestro* y su comunicación con los *dispositivos esclavos*, es decir, se detalla la forma en que son recibidos y enviados los datos entre los microcontroladores y el PC.

El *Capítulo VI* muestra de forma rápida el proceso de montaje de la estructura de la cabina y la disposición de los instrumentos, comparándola con la avioneta Cessna tomada como ejemplo.

El último capítulo, explica el modo de funcionamiento de los paneles, para ser usados por cualquier usuario final. Además, se demuestra el correcto funcionamiento del sistema.

Se concluye con las aportaciones de este proyecto, las opciones de mejora y el presupuesto.

Capítulo II. Microcontroladores y dispositivos E/S

En el presente capítulo se describirán las características generales y los datos técnicos de interés de los distintos dispositivos empleados, ya sean los microcontroladores de Atmel o los utilizados para la comunicación con el usuario (pulsadores, pantallas LCD, indicadores luminosos, etc.).

II.1 Microcontroladores

Los microcontroladores son dispositivos que combinan en un único chip los componentes principales de un ordenador convencional, es decir, una unidad de procesamiento o procesador (CPU), una serie de puertos tanto de entrada como de salida y memorias de almacenamiento [8].

En los últimos años, éstos han cobrado mayor importancia tanto en aplicaciones industriales como en proyectos personales gracias al desarrollo de plataformas *OpenSource* como Arduino y a la cultura *maker*[2].

Deben ser programados empleando un lenguaje que puede ser de bajo nivel (ensamblador) o bien de alto nivel como el basado en *C++* que brinda la plataforma Arduino con su IDE. La ventaja de este último, es la mayor facilidad que proporciona al programador de comprender el código del programa y la mejora de la portabilidad del mismo al resultar más sencilla su adaptación a otro microcontrolador.

En este proyecto se ha empleado un microcontrolador ATmega 2560 (incluido en la placa Arduino Mega) [9] y dos ATmega 328 (incluido en el Arduino Nano) [7]. El primero ejerce las funciones de *maestro*, comunicándose con el ordenador y con los otros microcontroladores y dispositivos de entrada/salida (E/S).

Para las fases de prueba y diseño se han empleado placas de Arduino UNO y Mega que contenían los microcontroladores y aportaban facilidades para la conexión. Durante la fase final de construcción del prototipo, se prescindió de estas placas de gran tamaño para fabricar varias PCB cada una con el microcontrolador empleado junto con los circuitos necesarios para la conexión directa de los instrumentos.

---

[2] ***Cultura maker:*** Es un movimiento contemporáneo basado en el *do it yourself* ("hágalo usted mismo"). Pretende crear soluciones no profesionales realizadas por los usuarios para problemas relativamente complejos.

II.1.1 ATmega 2560

El ATmega 2560 es un microcontrolador del fabricante Atmel [6] de montaje superficial (SMD[3]) que emplea la arquitectura AVR [10], es decir, basada en el modelo Harvard de computadores, donde las memorias de instrucciones y almacenamiento se encuentran físicamente separadas (*Figura II.1*). Emplea un conjunto de 32 registros de 8 bits, siendo uno de los dispositivos más potentes de la familia RISC [11] de Atmel, por lo que se empleó como dispositivo principal al que se conectaron el resto de componentes.

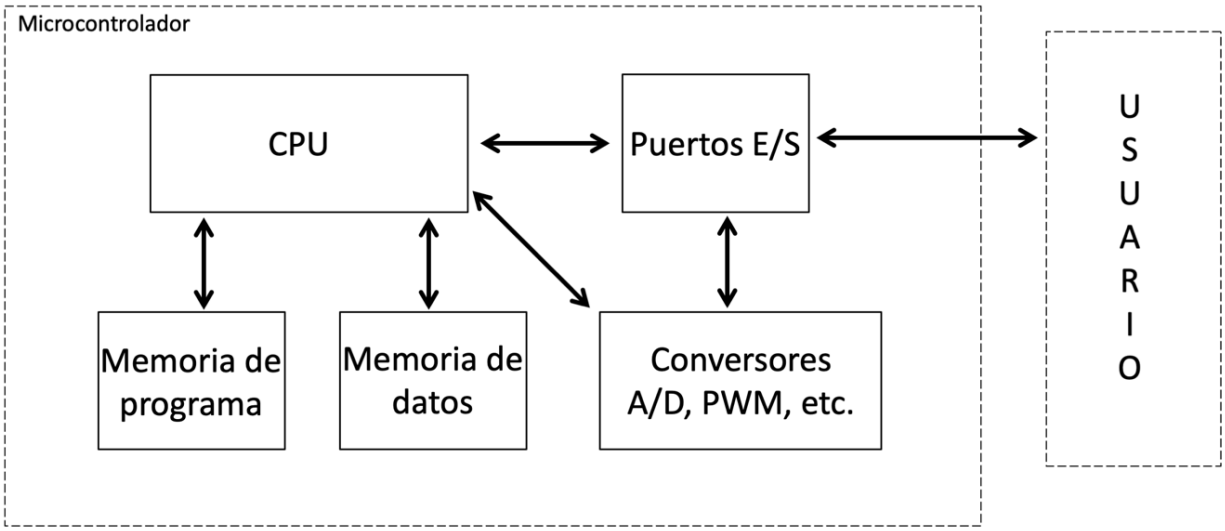


Figura II.1. Diagrama de bloques de funcionamiento del microcontrolador.

II.1.1.1 Patillaje del microcontrolador

Este microcontrolador cuenta con 100 pines entre los que se encuentran los de alimentación (denominados VCC), tierra (GND), 11 puertos de entrada-salida, la conexión del cuarzo para el oscilador y los empleados para comunicaciones *seriales*.

En la *Figura II.2*, extraída de su *datasheet,* se observa el patillaje del ATmega 2560.

[3] ***SMD:*** Tecnología de montaje superficial de componentes. Mediante este método no se realizan taladros en la placa de circuito impreso.

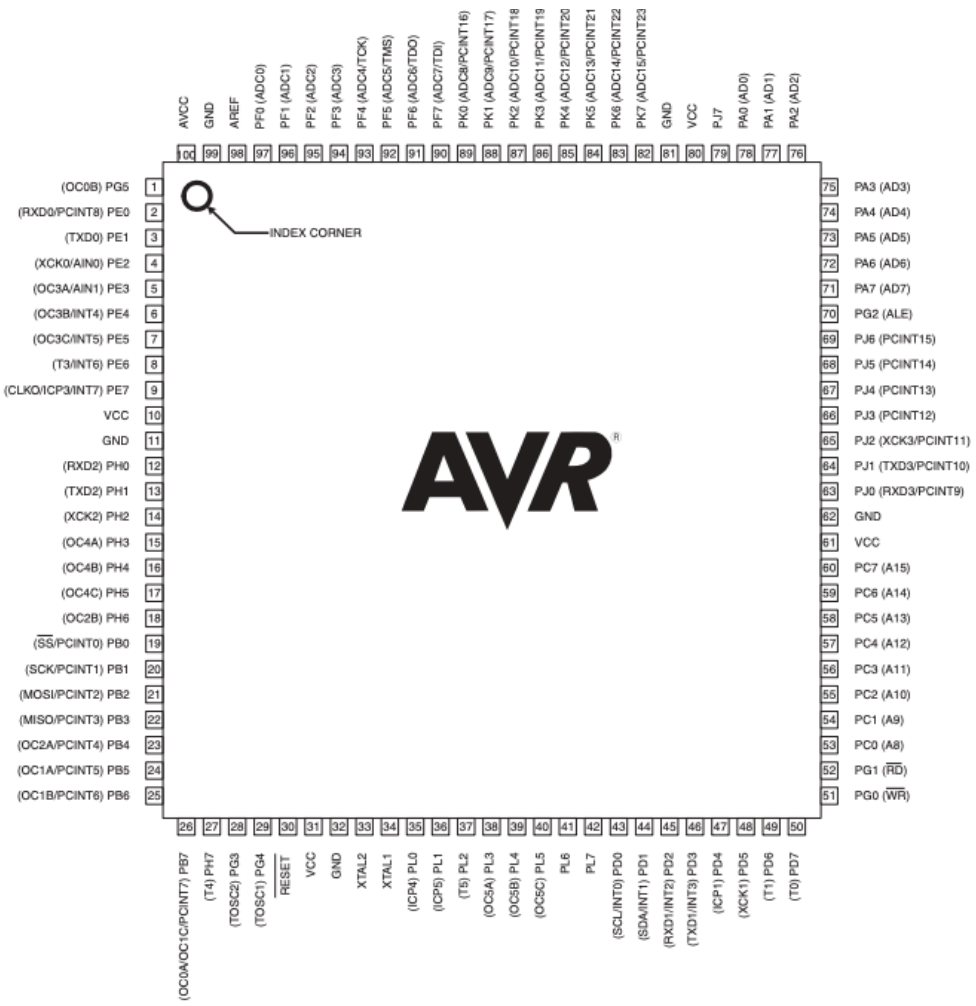


Figura II.2. Pines del ATmega 2560.

II.1.1.2 Puertos de Entrada/Salida

Este microcontrolador cuenta con 54 pines configurables como entrada o salida según las necesidades. De éstos, 15 se pueden emplear para salidas de señal PWM, las cuales se usan para controlar los servomotores de este proyecto.

Por otro lado, presenta 16 pines de entrada analógica multiplexadas para ser llevadas a un conversor analógico/digital (ADC) con 10 bits de resolución. Además, cuenta con una serie de pines vinculados a interrupciones *hardware* del sistema, pero que no han sido usadas en este proyecto.

Se ha hecho uso de los terminales destinados a la comunicación I^2C, y que será detallada en el *Capítulo II*, localizados en los pines 20 (SDA) y 21 (SCL) y los vinculados a la comunicación *serial* (Rx y Tx).

II.1.1.3 Memoria del microcontrolador

El ATmega 2560 dispone de tres tipos de memoria: Flash, SRAM y EEPROM.

- ***Memoria Flash:*** Con capacidad de 256 KB, se emplea para almacenar el programa del microcontrolador.
- ***Memoria SRAM***: En ella el microcontrolador guarda los datos que necesita para la realización de las operaciones. Se trata de una memoria dinámica y que se formatea al eliminar la tensión. En este dispositivo su capacidad es de 4 KB.
- ***Memoria EEPROM:*** De tipo no volátil, en ella que se almacenan aquellas variables o constantes que se pretende que permanezcan sin cambio incluso al hacer un *reset*.

II.1.2 ATmega 328

Este microcontrolador, al igual que el 2560, pertenece a la familia AVR. Se presenta en versiones SMD y *through-hole*[4]. Tienen unas características más modestas que el anterior microcontrolador, pero destaca por su reducido tamaño y bajo consumo.

Este dispositivo viene instalado en el Arduino Nano (en versión SMD), que es el dispositivo que se empleó durante las pruebas y posteriormente se recurrió a la versión con encapsulado 28 DIP[5] para la placa de circuito impreso definitiva.

En la *Figura II.4* se puede observar las diferencias entre ambos encapsulados.

Figura II.3. ATmega 328 en formato SMD y en formato DIP de 28 pines.

[4] ***Through-hole:*** Tecnología empleada en el diseño de circuitos impresos que consiste en la realización de taladros en la placa para la conexión de los componentes eléctricos.

[5] ***DIP:*** Encapsulado plástico de componentes. Presenta patillas de conexión a cada lado posibilitando la conexión a una *ProtoBoard*.

II.1.2.1 Puertos de Entrada/Salida

Este dispositivo (*Figura II.6*) presenta un total de 14 pines de entrada/salida (E/S) digitales, de los cuales 6 proporcionan señales PWM de 8 bits. Además, posee 6 pines destinados a entradas analógicas con conexión al ADC[6] de 10 bits similares a los del ATmega 2560, así como dos pines para conexión *serial* al bus I^2C.

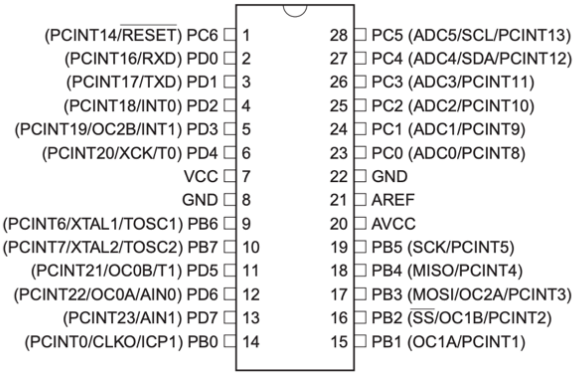


Figura II.4. Pines del ATmega 328P.

II.1.2.2 Memoria del microcontrolador

Presenta los mismos tipos de memoria y funcionamiento que su homólogo el ATmega 2560, pero con las siguientes capacidades:

- ***Flash:*** 32 KB.
- ***SRAM:*** 2 KB.
- ***EEPROM:*** 1 KB.

II.2 Motores paso a paso

Son motores destinados a movimientos muy precisos que se controlan mediante pequeños impulsos eléctricos. Para un motor con 4 bobinas existen tres formas de controlarlo: mediante una secuencia normal, una secuencia de paso completo o de medio paso.

- ***Secuencia normal***: Se activan dos bobinas en cada momento, teniendo un movimiento con alto par.
- ***Secuencia de paso completo***: Se activa una única bobina cada vez. El movimiento es más suave y el par motor es menor.

[6] ***ADC:*** Conversor Analógico Digital. Dispositivo que permite leer un valor analógico de tensión en un dispositivo que presenta únicamente entradas digitales.

- ***Secuencia de medio paso:*** Se activan las bobinas en una secuencia alterna de dos y una. Se consigue un movimiento más suave a cambio de reducir más el par.

En el presente proyecto se emplean motores modelo 28BYJ-48 [12] que disponen de bobinas con una resistencia de 50 Ω unidas a un terminal común de alimentación de 5 voltios (*Figura II.7*). Se emplea la secuencia normal para el control de los mismos al ser la que aporta el mayor par.

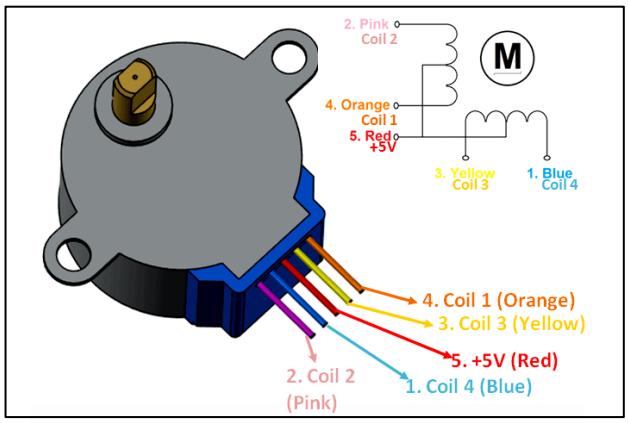


Figura II.5. Motor modelo 28BYJ-48.

Presentan las siguientes características (*Tabla II.1*):

Tabla II.1. Características del 28BYJ-48.

N de pasos por vuelta (en paso completa)	32
Reductora	1/64
Par motor	34 N/m
Corriente nominal	55 mA

Para dar una vuelta completa se precisan 32 excitaciones de las bobinas, pero como presenta una reductora 1/64, finalmente se obtendrá:

$$1\ vuelta \cdot \frac{32\ pasos}{1\ vuelta} \cdot \frac{64\ vueltas\ reductora}{1 vuelta\ motor} = 2048 \frac{pasos}{vuelta} \cong 0{,}18\ ^{o}/paso \qquad (II.1)$$

Esto permite tener un posicionamiento relativamente preciso, ya que es posible realizar movimientos incrementales de 0,18 grados.

Para controlar este motor desde el microcontrolador se precisa un dispositivo que suministre la corriente necesaria. Para ello se emplea un *puente en H*, que es un amplificador formado por un par Darlington [13] de transistores a los que se le conecta a su base la salida del pin digital del microcontrolador con el que se les pondrá en corte o en conducción. Para cada motor se emplea el integrado ULN2003 [14] que combina siete amplificadores con transistores NPN en la misma pastilla, permitiendo con cada integrado alimentar tres bobinas.

A continuación, se muestra el diagrama de bloques de este dispositivo (*Figura II.8*) donde se puede observar que su funcionamiento es con lógica negada, de modo que un 0 lógico a su entrada activa la salida. Sin embargo, como el terminal común de dichos *steppers* es 5V, cargar un 1 a la salida del microcontrolador, supondrá poner a tierra ese pin del motor, permitiendo la activación de dicha bobina.

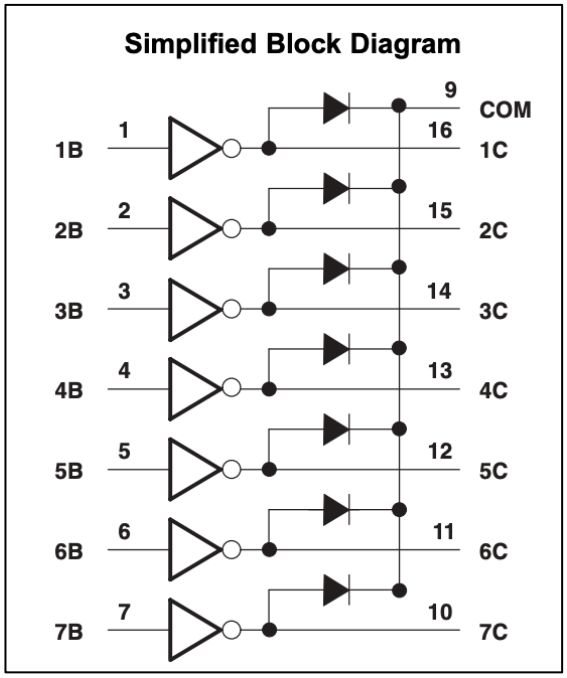


Figura II.6. Diagrama de bloques del ULN2003.

El problema del uso de motores paso a paso es la dificultad para conocer su posición inicial, dado que no presentan ningún sistema de realimentación propio. Para ello, se recurre a un sensor óptico con el que poder realizar un primer

movimiento de *homing*[7] hasta una posición conocida para, posteriormente, realizar los movimientos bajo la hipótesis de que no se pierden pasos.

El dispositivo empleado fue el TCST1103 [15] (*Figura II.9*), que combina un diodo emisor de luz con un fototransistor. Su funcionamiento se basa en emplear la luz del diodo para poner en conducción en fototransistor, de este modo si se intercala un dispositivo con una abertura, se permitirá el paso de la luz una única vez por cada vuelta del motor, conociendo de este modo cuando está en dicha posición.

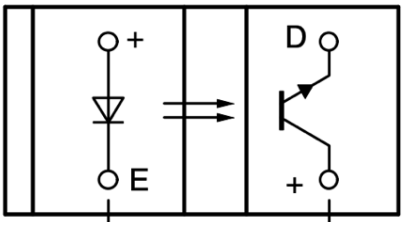


Figura II.7. Vista esquemática del sensor óptico.

II.3 Servomotores

Un servomotor es un motor de corriente continua que cuenta con un lazo de realimentación (*Figura II.10*), lo que permite su control tanto en posición como en velocidad. Para indicar la posición o velocidad deseada se emplea la modulación por anchura de pulsos con una señal PWM proporcionada por el microcontrolador.

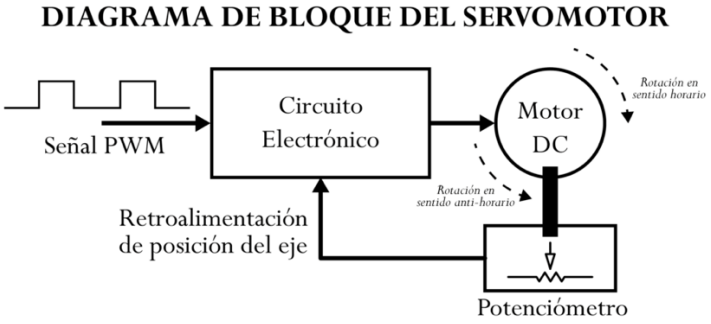


Figura II.8. Diagrama de bloques de un servomotor.

Su principal inconveniente radica en la imposibilidad de realizar un movimiento continuo, puesto que el potenciómetro que incorpora en su eje limita el rango de operación (en grados).

[7] ***Homing:*** Primer movimiento inicial que realiza un mecanismo móvil, con el objetivo de ir a una posición conocida a partir de la cual comenzar sus movimientos.

En este proyecto se emplearon los servos de referencia SG-90 [16] (*Figura II.11*) que están destinados a aplicaciones de radio-control. Sus características principales se detallan a continuación (*Tabla II.2*):

Tabla II.2. Características principales SG-90.

Par motor	1,2 kg/cm
Velocidad máxima	83 rpm
Rango de operación	0-180º

Figura II.9. Servomotor modelo SG-90.

Tanto los servos como los motores paso a paso, se emplean para mover las agujas de los diferentes instrumentos del avión, como si de *gauges* analógicos se tratase.

II.4 *Displays*

Los *Displays* son dispositivos que permitirán mostrar información del avión al usuario. En concreto, se empleó un *Display* LCD para mostrar los parámetros del piloto automático, y *Displays* de 7 segmentos para mostrar los valores de las frecuencias de la radio y transpondedor.

La pantalla LCD empleada ha sido de 16x2 (*Figura II.12*), que proporciona 16 caracteres de ancho por 2 filas de datos.

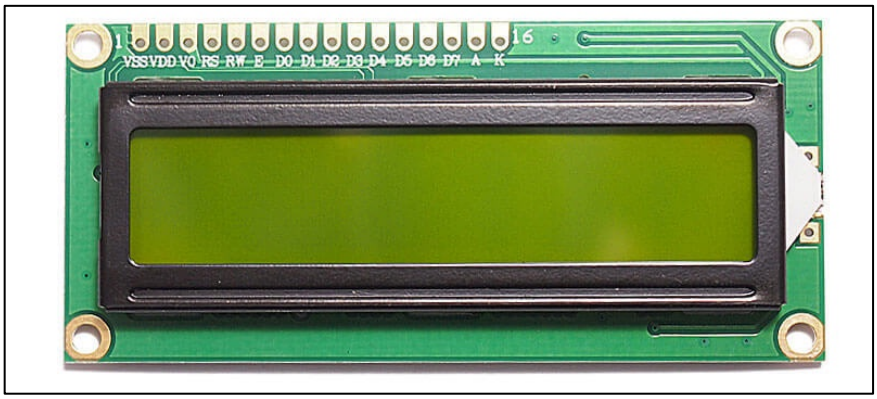


Figura II.10. Pantalla LCD de 16x2.

Éste dispone de una serie de pines de datos y control que se conectan al microcontrolador (*RS,EN,d0,d1,d2,*etc.). Sin embargo, para realizar su control mediante una comunicación paralela, se hace necesario un mínimo de 7 cables por cada pantalla. En su lugar, se decidió emplear una comunicación serie I^2C y hacer uso de expansores de 8 bits (PCF8574) [17] con el objetivo de reducir al mínimo el número de pines empleados en el microcontrolador.

Este integrado se encarga de transformar los 8 bits recibidos en modo *serial* por el bus a una palabra de 8 bits que se transmite de forma paralela a la pantalla. Se usarán también para multiplexar las salidas digitales de los indicadores led, dado que disponen de *Latches*, es decir, mantienen el estado hasta recibir un nuevo mensaje.

Los *Displays* 7 segmentos empleados han sido todos de cátodo común y se han usado en diferentes empaquetados:

- *Displays* 7 segmentos de 4 dígitos (3641-AS) [18], que comparten en su empaquetado los pines a-g y el del punto decimal y 4 pines para la activación de cada dígito (*Figura II.11*).
- *Display* 7 segmentos de 1 dígito (3161-AS) [19], que están formados por sus pines para los segmentos y un pin de cátodo.

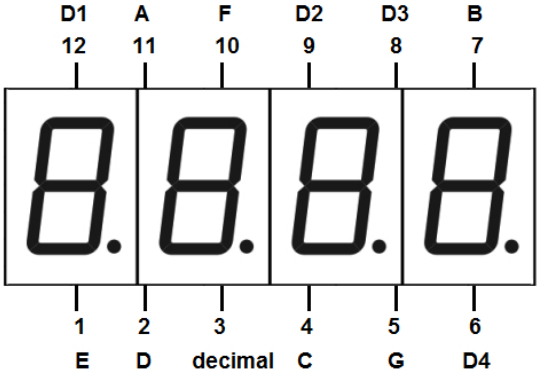


Figura II.11. Pines del display 7 segmentos de 4 dígitos.

Para el control de estos dispositivos, al igual que con *Displays* LCD, se ha hecho uso de un multiplexor aunque en este caso se ha empleado el MAX7219 [20] destinado a controlar éstos mediante el bus *SPI*.

II.5 Interruptores y pulsadores

De forma genérica, un interruptor o un pulsador es un dispositivo que permite el paso de corriente cuando es accionado. El primero permite mantener el flujo constante de corriente y el segundo solo durante el tiempo de accionamiento.

Según se desee obtener un *1* o un *0 lógico* al activar uno de éstos se debe proceder a conectar empleando una resistencia de *pull-down* o *pull-up*. Si se desea obtener un *1 lógico*, se recurre a la configuración de entrada con resistencia *pull-down* que consiste en conectar el pulsador a *Vcc* y una resistencia en paralelo en la entrada conectada a tierra (*A, Figura II.12*). En la otra configuración (*pull-up*), se conecta el

interruptor a tierra y se obtiene un *0 lógico* cuando el pulsador es accionado (*B, Figura II.12*). En este segundo caso se trabaja con una lógica invertida.

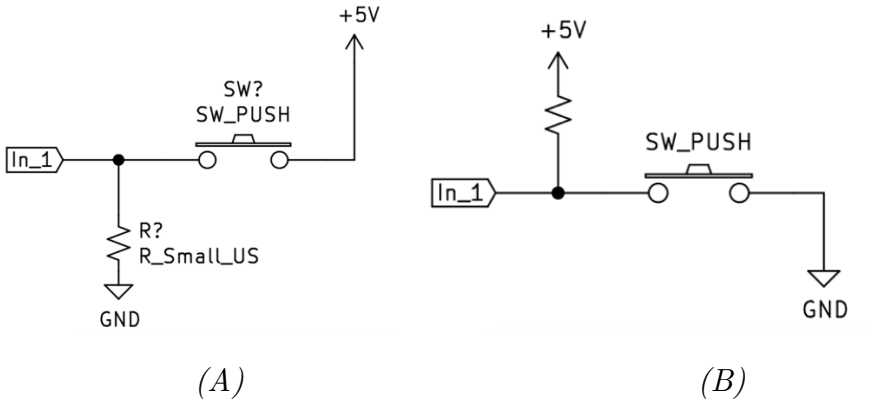


Figura II.12. Entrada en configuración de pull-down(A) y de pull-up(B).

En este proyecto se han empleado interruptores y pulsadores de diferentes tipos: desde pulsadores sencillos, retroiluminados, etc. hasta interruptores *tipo palanca* o basculantes (*Figura II.13*).

Figura II.13. Pulsador sencillo, interruptor de tipo palanca y basculante.

II.6 *Encoder* rotativo

Un *encoder* o codificador rotativo (*Figura II.14*) es un dispositivo de entrada que permite medir la velocidad, posición y el sentido de giro de un eje. Los *encoders* empleados para modificar las frecuencias de la radio y otros controles, han sido de tipo cuadratura.

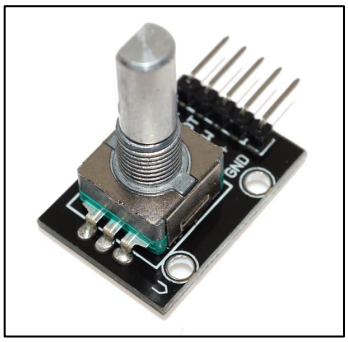


Figura II.14. Encoder rotatorio KY040.

Cuentan con un conjunto de pistas metálicas con divisiones, de modo que con el giro del eje se generan una serie de pulsos. Comparando el desfase entre los dos canales existentes en la generación de pulsos (*Figura II.17*), es posible determinar la dirección de giro del mismo [21] y, en base a esto, poder incrementar o decrementar los valores de la frecuencia de la radio o la altitud del piloto automático.

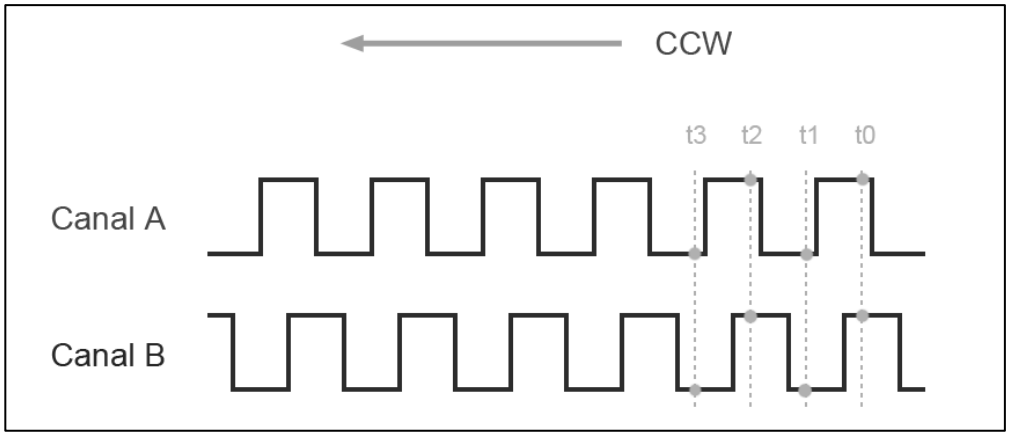


Figura II.15. Señales de un encoder de cuadratura.

Dado que en este caso únicamente interesa conocer el sentido de giro, se ha optado por un modelo económico (*KY-040*) [22] que proporciona una resolución de 20 pulsos por vuelta, de modo que es posible conocer una posición con una precisión de 18°, suficiente para determinar el sentido de giro y, con ello, aumentar o reducir una variable.

II.7 Potenciómetros

Los potenciómetros son resistencias variables empleadas para medir posición o giro de un mecanismo. Se basan en la relación proporcional existente entre un valor de resistencia y la longitud.

$$R\ [\Omega] = \rho \left[\frac{\Omega mm^2}{mm}\right] \cdot \frac{L\ [mm]}{s[mm^2]} \qquad \text{(II. 2)}$$

En este proyecto se utilizaron para conocer la posición de cada una de las palancas de gases. Para ello, se necesitan potenciómetros de movimiento lineal con una buena precisión (*Figura II.16*).

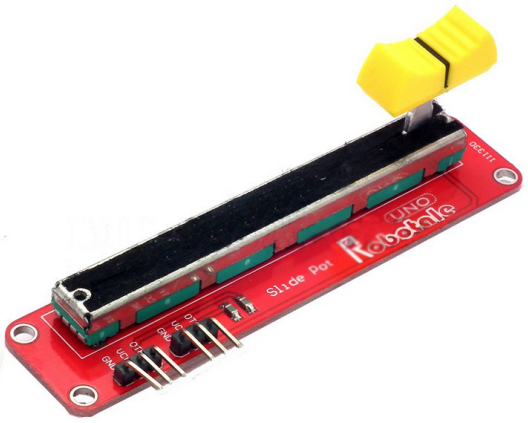


Figura II.16. Potenciómetro de movimiento lineal empleado.

En los microcontroladores, se conectan estos dispositivos a las entradas previstas del ADC. Para realizar una lectura de la tensión en una posición del potenciómetro, se conecta un extremo del mismo a $+5V$ y el otro a tierra, con la *patilla* intermedia a la entrada del microcontrolador. Escalando la misma es posible conocer su posición (en porcentaje).

$$Lectura\ máxima\text{: } 5V = 2^{10} = 1024$$

$$Lectura\ mínima\text{: } 0V = 0$$

Como el ADC del microcontrolador es de 10 bits, se tiene una cuenta máxima de 1023. A partir de esto es posible calcular la resolución:

$$\Delta Lectura\ (\%) = \frac{100\ \% - 0\ \%}{1024 - 0} \cong 0{,}098\ \%/unidad \qquad (\text{II.}3)$$

$$LSB = \frac{V_{máx.} - V_{mín.}}{2^n} = \frac{5\ V}{2^{10}} \cong 0{,}005\ V \qquad (\text{II.}4)$$

Como se puede observar, el incremento de una unidad en la lectura de la entrada del ADC (unos 5 mV) se corresponde a un incremento del 0,098% en la posición.

Capítulo III. Herramientas de desarrollo

Durante este tercer capítulo se explican todas las herramientas empleadas para poder ejecutar el proyecto, tanto las de desarrollo *software* como las mecánicas. Por un lado se explica el entorno de Arduino, el programa de simulación (FSX), el de comunicación con el microcontrolador (*Link2FS*), diseño de PCB así como el de piezas mecánicas. Por otro lado, se comenta el empleo de la placa Arduino UNO durante la fase de pruebas y la impresora Prusa i3 para la producción de piezas en 3D. Por último, se explican los protocolos y buses de comunicación empleados (I^2C, *SPI* y *serial*).

III.1 Herramientas de desarrollo *Software*

Para la programación de los microcontroladores se recurre al entorno unificado de desarrollo de Arduino [23], que permite programar en *C++*. El ordenador ejecuta el *software* de simulación de Microsoft así como un programa intermediario entre el *maestro* y dicho simulador.

III.1.1 IDE de Arduino

La plataforma *OpenSource* de Arduino [23] proporciona un IDE de desarrollo (*Figura III.1*), es decir, un entorno de desarrollo unificado desde el que se puede realizar la escritura y compilación del código fuente para cualquiera de sus microcontroladores. Además, permite realizar la carga del mismo en la memoria *Flash* y cuenta con un monitor *serial* desde el que ver los mensajes que se envían por dicho puerto.

Figura III.1. IDE de Arduino.

Carece de funciones propias de depuración, lo que obliga a escribir pequeños fragmentos de código que posibiliten conocer si una determinada instrucción se ha ejecutado. Para esto se recurre a la activación de ciertas salidas a las que se conectan LED*s* o a la escritura de mensajes por el puerto serie (mediante `void Serial.print(char mensaje)`) que muestre lo que queremos conocer.

III.1.2 FSX

Flight Simulator X [1] (*Figura III.2*) es un programa de simulación aérea desarrollado por Microsoft en el año 2006 y actualizado en el 2008 con los *Service Pack 1* y *2*. Mejora la versión del programa lanzada en 2004 (*FS04*). Permite, en su versión básica, volar 18 aviones entre los que se incluye el modelo en el que se basa este proyecto: la avioneta *Cessna 172*.

Figura III.2. Imagen capturada de una Cessna 172 en FSX.

Una de las ventajas de este programa, frente a otros como *Prepar3D* [3], es la gran cantidad de usuarios (denominados comunidad) que tiene detrás, con una plataforma de entrenamiento como es IVAO [4].

Los requisitos mínimos [24] para ejecutarlo son:

- Procesador de 2 GHz o superior.
- 1 GB de RAM o más.
- Tarjeta de vídeo compatible con *DirectX* 9.

III.1.3 Link2FS

Link2FS [25] es un *software* gratuito desarrollado en *C++* por *Jimspace* para comunicar *Flight Simulator X* con cualquier placa Arduino mediante un puerto USB. La primera versión fue publicada en el año 2011 y, con los años, ha ido incorporando mejoras hasta permitir controlar completamente un avión desde cualquier placa e incluso, emplear hasta tres de forma simultánea. En la *Figura III.3* se puede apreciar la interfaz principal del programa en la que se selecciona el puerto *serial* a emplear.

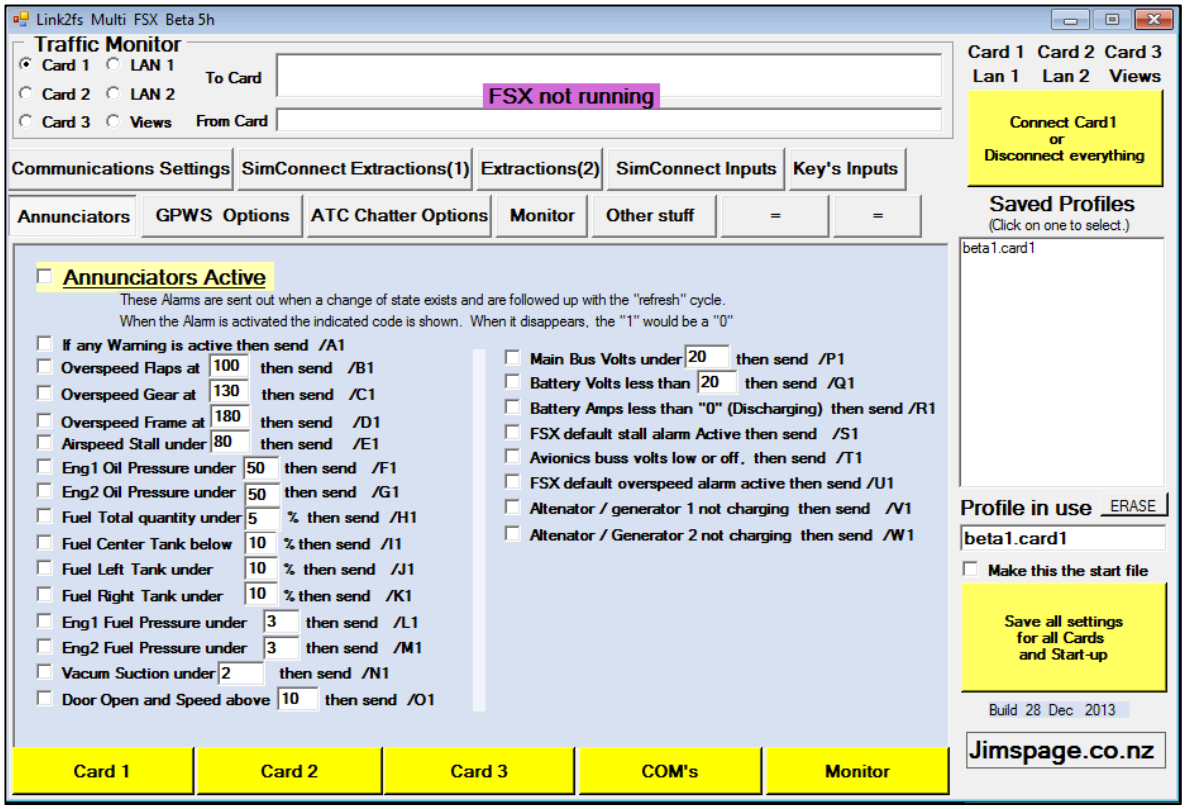


Figura III.3. Interfaz de Link2FS.

Proporciona todos los parámetros de la aeronave en vuelo mediante cadenas de caracteres que pueden ser leídas o escritas por puerto serie con cualquier microcontrolador. Esto será explicado con mayor detalle en el *Capítulo V* pero, a modo general, indicar que envía tramas con dos caracteres de identificación seguido de un conjunto de números de longitud variable dependiendo del parámetro a remitir.

Un ejemplo de la trama puede ser:

=A128.300

En este caso, la trama lleva el indicador correspondiente a la frecuencia principal de la radio (=*A*) y, por tanto, se obtiene que la misma estaría sintonizada en 128.30 MHz.

También lee los valores del puerto serie remitidos por el microcontrolador ATmega 2560 y se encarga de transmitir dicha información al FSX. Además permite configurar códigos leídos por *serial* a atajos de teclado definidos por el usuario.

Si por el *serial*, por ejemplo, el microcontrolador envía la siguiente cadena:

A427763

Este *software* interpreta que el identificador *A42* corresponde a la configuración del transpondedor y modificará el valor del *transponder* del avión a 7763.

III.2 Herramientas de desarrollo *Hardware*

Para el desarrollo del presente proyecto fueron necesarias numerosas piezas mecánicas que debían ser construidas a medida. Para ello, se recurrió a *Fusion 360* y a la impresora 3D Prusa Hephestos i3. Para diseñar los circuitos impresos fue empleado el *software* libre *KiCAD* [26].

III.2.1 Fusion 360

Fusion 360 (*Figura III.4*) es un *software* de la empresa Autodesk [27] que emplea una metodología basada en operaciones de moldeado a partir de dibujos, es decir, permite el diseño de piezas en 3D a partir de *bocetos* hechos en el plano y a los que se aplican operaciones básicas de movimiento como revolución, extrusión, chaflán, etc. Este diseño, de tipo paramétrico, permite modificar una determinada cota y que toda la pieza se ajuste a la misma. Una vez diseñada la pieza se exporta a formato *.stl* para ser reproducida con cualquier máquina de control numérico. Además, permite generar los planos de las piezas y esquemas de montaje, así como *renders*[8] y animaciones.

[8] ***Render:*** Proceso empleado para generar una imagen a partir de un modelo. Permite obtener una representación tridimensional de un objeto diseñado por computador.

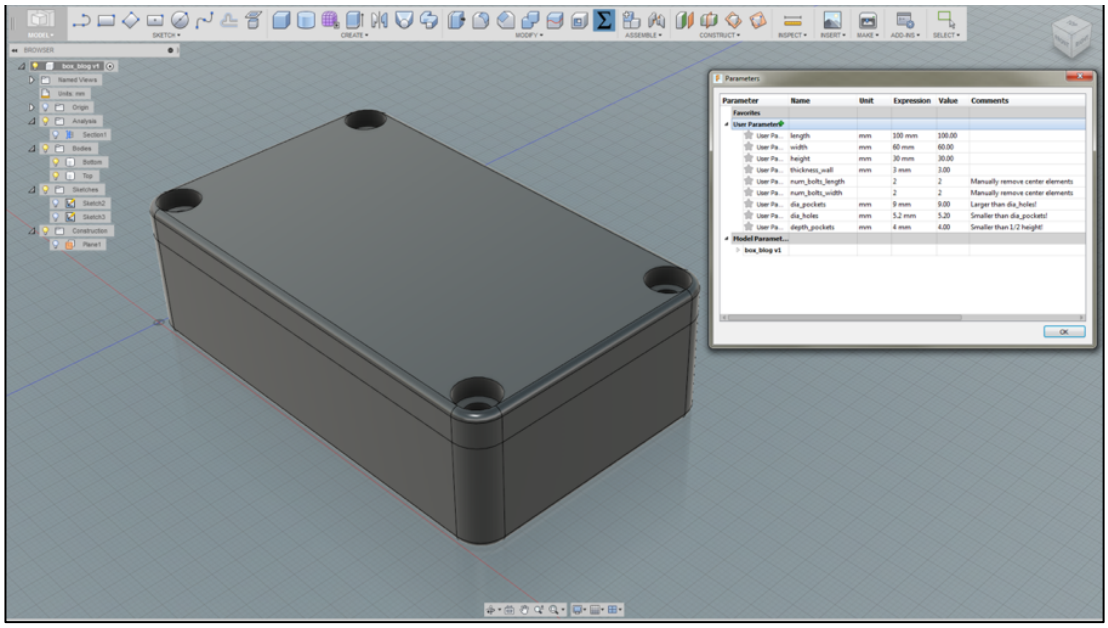


Figura III.4. Entorno de trabajo del Fusion 360.

III.2.2 Impresora 3D Prusa

Las piezas anteriormente diseñadas y exportadas en formato *.stl* deben ser impresas para proceder a la construcción de los prototipos. Para ello se ha empleado una impresora 3D modelo Prusa i3 de Hephestos (*Figura III.5*).

Esta impresora de tipo cartesiano, con movimiento en tres ejes ortogonales (*x,y,z*), cuenta con una resolución de hasta 60 micras, pudiendo imprimir piezas de unas dimensiones inferiores a *215 mm* de ancho, *210 mm* de largo y *180 mm* de alto.

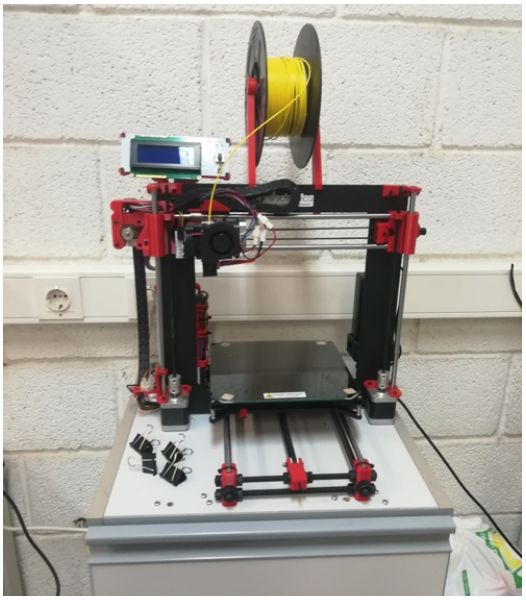


Figura III.5. Impresora 3D modelo Hephestos i3 empleada.

A partir del archivo *.stl* generado por el *software* CAD se debe obtener el código de movimientos de la máquina, el llamado *gcode*. Para ello, se emplea el *software Repetier* [28] que se comunica con la CNC y permite enviar la secuencia de movimientos a la misma o exportarlo a una tarjeta SD.

El principio de funcionamiento de estas impresoras se basa en la adición y superposición de material fundido sobre las capas inferiores existentes. Para cada una de las mismas la impresora va soltando material plástico fundido en las coordenadas exactas. El límite de grosor de las capas viene dado por el diámetro de salida del plástico fundido, que depende a su vez del diámetro del extrusor.

A pesar de las buenas características de la Prusa i3 (detalladas en *Tabla III.1*), se tuvieron ciertos problemas con el diseño de engranajes, al necesitar una resolución mayor. Esto limitó en exceso los diseños.

Tabla III.1. Características de la impresora Hephestos i3.

Velocidad de impresión	Hasta 100 mm/s
Temperatura de trabajo	15-25 ºC
Resolución máxima	60 micras
Diámetro del filamento	1.75 mm
Diámetro boquilla extrusor	0.4 mm
Controlador electrónico	BQ Zum Mega 3D
Alimentación	12 V 100W

III.2.3 KiCAD

Todo el desarrollo de los circuitos, tanto planos como circuitos impresos, se ha llevado a cabo mediante el *software* libre KiCAD [26] (*Figura III.7*).

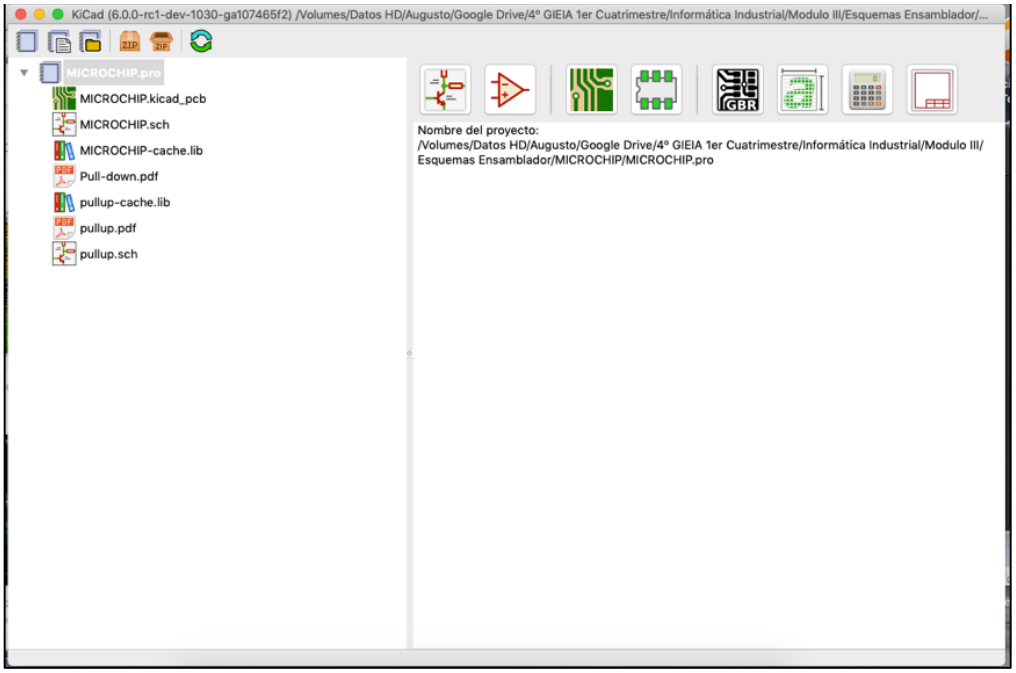


Figura III.6. Entorno de desarrollo de KiCAD.

Este programa desarrollado en 1992 y aprobado en 2013 por el CERN, permite realizar todas las etapas de diseño:

1. **Captura esquemática**: Fase en la que se diseña el esquema del circuito sin entrar en detalles de encapsulados de los componentes o dimensiones de los conectores.
2. **Diseño de PCB**: Donde se define el tamaño del circuito, los encapsulados y conectores a emplear y se dibujan las pistas. En esta etapa se le asigna a cada componente un *footprint*, es decir, un patrón de dibujo en la placa que permita soldarlo o taladrar la misma si es necesario.
3. **Modificación de símbolos y *footprints***: Si no se desea emplear los símbolos que proporciona el editor CAD por defecto o los desarrollados por la comunidad, el usuario puede diseñar símbolos o *máscaras* que se adapten a sus necesidades.
4. **Generación de archivos *GERBER***: Una vez se finaliza el diseño de la placa, se debe exportar a un formato que sea aceptado por las fresadoras CNC y que además permita la impresión de los negativos para la fabricación mediante procesos fotosensibles.

Para el desarrollo de todas estas etapas se cuenta con 4 módulos principales que se describen seguidamente (*Tabla III.2*):

Tabla III.2. Descripción de los módulos de KiCAD.

Módulos de KiCAD	
KiCAD	Es el administrador de proyectos y desde el que se pueden acceder a los demás módulos.
eeschema	Permite diseñar los esquemáticos que servirán de base para la PCB.
pcbnew	Mediante esta herramienta se realiza el diseño del circuito impreso cargando los componentes y las relaciones definidas en el esquemático.
gerberview	Genera el *GERBER* y permite su visualización.

III.3 Bus I^2C

Tal y como se ha comentado en capítulos anteriores, los diferentes microcontroladores se encuentran conectados a un bus que emplea la comunicación I^2C (*Figura III.8*) desarrollada por Philips en 1992 [29].

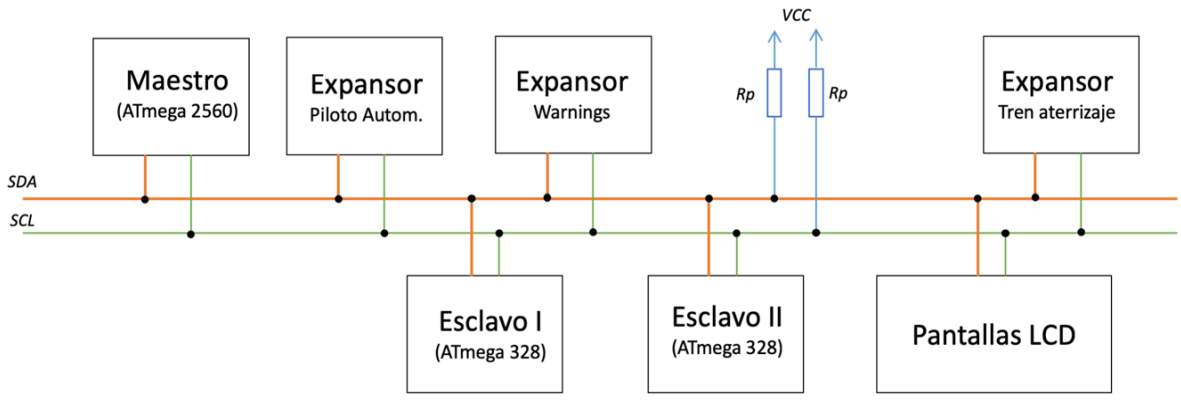


Figura III.7. Estructura del bus I2C empleado.

El *I²C* o *Inter-Integrated Circuit* es un bus serie de datos síncrono que emplea 2 cables para la comunicación (SDA para el envío de datos y SCL para la señal de reloj) y una tierra común. Su comunicación es *half-duplex*, por lo que se permite que todos los dispositivos (tanto *esclavos* como *maestro*) *hablen* y *escuchen,* pero no de manera simultánea y enviando únicamente datos de un byte.

Físicamente, estas dos líneas son de tipo *drenador abierto*[9] [13] por lo que se necesitan dos resistencias de *pull-up* para poner el bus a nivel alto. Los dispositivos envían los datos mediante niveles bajos.

Cada uno de los dispositivos conectados al bus dispone de una dirección única que lo identifica de modo inequívoco dentro de la red. El *maestro* será el encargado de indicar la dirección del dispositivo con el que quiere *hablar* antes de enviar el mensaje.

La comunicación se inicia con un patrón de *start* que es enviado por el *maestro*; esto pone en espera a los *esclavos*. A continuación se envía el byte de identificación que cuenta con 7 bits para la dirección física del *esclavo* y un bit (el LSB) que indica si se desea *escuchar* (*high*) o *hablar* (*low*).

Para una dirección *0x12*, se enviaría el siguiente byte:

- 00100101 para escuchar de ese dispositivo.
- 00100100 para enviar al *esclavo*.

Este dato recorre todos los *esclavos* y el que tenga esa dirección en memoria, responde con un bit de *acknowleage (ACK)*. Una vez reconocida la transmisión se comienza el envío de datos en paquetes de un byte y se recibe un nuevo bit de *ACK* que confirma la recepción. Tras finalizar la transmisión, el *maestro* envía un patrón de bits que indican la parada (los conocidos como bits de *stop*).

De forma genérica el patrón de envío de mensajes es:

Start	*D7*	*D6*	*D5*	*D4*	*D3*	*D2*	*D1*	*R/W*	*ACK*	*Bytes de datos*	*ACK*	*Stop*

[9] ***Salida de tipo drenador abierto***: Se trata de la etapa final de salida de un circuito formada únicamente por un transistor de canal N. De este modo, es posible obtener un 0 lógico pero se necesita una polarización para obtener una salida lógica de 1.

Por tanto, para enviar un byte que contenga el carácter ASCII *'B'* (cuyo número es 66 en decimal y 0x42 en hexadecimal) al dispositivo con dirección 0x06, se enviaría la siguiente trama:

11	*0000110*	*0*	*ACK*	*01000010*	*ACK*	*11*
Start	*ID(0x06)*	*Write*	*ACK*	*Datos (0x42)*	*ACK*	*Stop*

En *Arduino* esta comunicación está vinculada a una serie de pines físicos del dispositivo, que se detallan en la *Tabla III.3*, y es posible emplearla gracias a la librería `wire.h` .

Tabla III.3. Pines I²C en microcontroladores Atmel 328 y 2560.

ATmega 328P	SDA: A4 SCL: A5
ATmega 2560	SDA: 20 SCL: 21

Teóricamente, al poder emplear 7 bits para la dirección, se podrían conectar hasta 128 dispositivos en el bus. Sin embargo, este número se ve reducido debido a la capacidad máxima (en pF) que admite el bus y las direcciones de uso reservado.

III.3.1 Cálculo de las resistencias de polarización

Para obtener el rango de valores de resistencias para polarizar el bus se recurre a la documentación de *Texas Instrument* [30, p. 2] para dispositivos *I²C.*

De ella se extrae que la resistencia mínima depende de la tensión de alimentación (V_{cc}), la máxima tensión de salida del nivel bajo (V_{OL}) y la máxima corriente de reposo (I_{OL}):

$$R_p(\min) = \frac{V_{cc} - V_{OL}(\max)}{I_{OL}} \quad \text{(III. 1)}$$

Que en este caso, teniendo en cuenta los datos del *datasheet* [30] y que todos los dispositivos se alimentan a 5V, se obtiene:

$$R_p(\min) = \frac{5 - 0{,}4}{3\ mA} = 1533{,}33\ \Omega \quad \text{(III. 2)}$$

Para la resistencia máxima, se debe contemplar el tiempo que tarda la señal en pasar de un nivel bajo a alto. Este valor de tiempo es una función de la resistencia y la capacidad del bus. Si la resistencia fuera extremadamente alta, nunca se pondría

a nivel alto. Por tanto, el estudio de este parámetro consiste en el análisis de la respuesta de un circuito RC, es decir:

$$v(t) = V_{cc} \cdot \left(1 - e^{-\frac{t}{RC}}\right) \qquad \text{(III. 3)}$$

Si se toma un nivel de tensión alta como todo aquel valor superior al 70% de la tensión de alimentación se tiene:

$$V_{IH} = 0{,}7 \cdot V_{cc} = V_{cc} \cdot \left(1 - e^{-\frac{t_1}{R_p C_b}}\right) \qquad \text{(III. 4)}$$

Donde R_p es la resistencia de pull-down y C_b la capacidad del bus.

Nuevamente, tomando la ecuación anterior pero para una tensión de nivel bajo (V_{IL}) del 30% de la alimentación, se deduce:

$$V_{IL} = 0{,}3 \cdot V_{cc} = V_{cc} \cdot \left(1 - e^{-\frac{t_2}{R_p C_b}}\right) \qquad \text{(III. 5)}$$

Si se escribe el tiempo de subida (t_r) como la diferencia entre t_1 y t_2 se tiene:

$$t_r = t_2 - t_1 = 0{,}8473 \cdot R_p \cdot C_b \qquad \text{(III. 6)}$$

Lo que despejando la resistencia deja la siguiente expresión.

$$R_p = \frac{t_r}{0{,}8473 \cdot C_b} \qquad \text{(III. 7)}$$

Si se desea una respuesta normal del bus, se puede admitir un tiempo de hasta 1 µs con una capacidad de hasta 200 pF. Por tanto, la resistencia máxima es:

$$R_p(\text{max}) = \frac{1 \cdot 10^{-6} s}{0{,}8473 \cdot 200 \cdot 10^{-12}\ F} = 5901{,}1\ \Omega \qquad \text{(III. 8)}$$

Dentro del rango calculado de 1,5 kΩ y 5,9 kΩ, se decide emplear un valor de 4,7 kΩ, lo que dota al bus de una velocidad de transición de 796 ns, como se muestra a continuación.

$$t_r = R_p \cdot 0{,}8473 \cdot C_b = 4700\ \Omega \cdot 0{,}8473 \cdot 200 \cdot 10^{-12} F = 7{,}96 \cdot 10^{-7} seg. \qquad \text{(III. 9)}$$

Es decir, que la velocidad es de 796 nanosegundos, que es inferior al valor de 1 µs establecido como límite teórico.

Gráficamente, esto se comprueba en la siguientes dos curvas representadas, donde se ve el valor máximo (azul) y el mínimo (rojo) en la *Figura III.8 y III.9.*

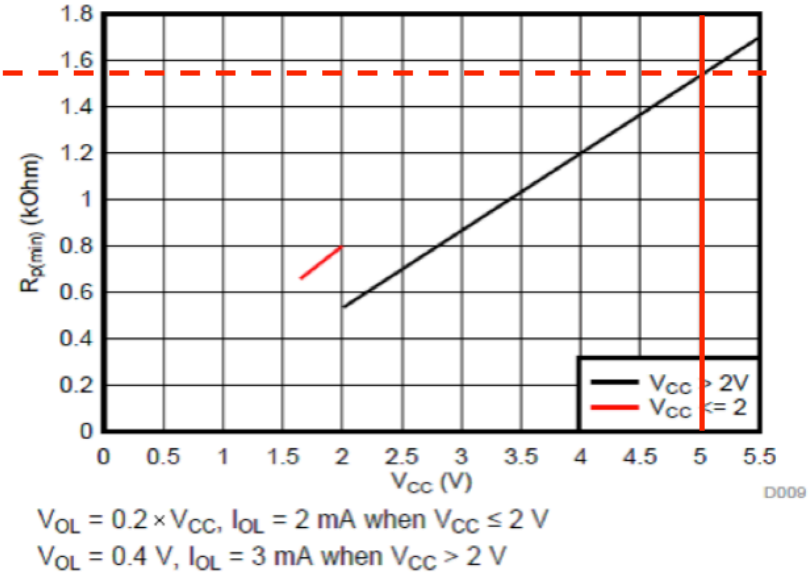


Figura III.8. Cálculo de R_p mínima para el bus I^2C.

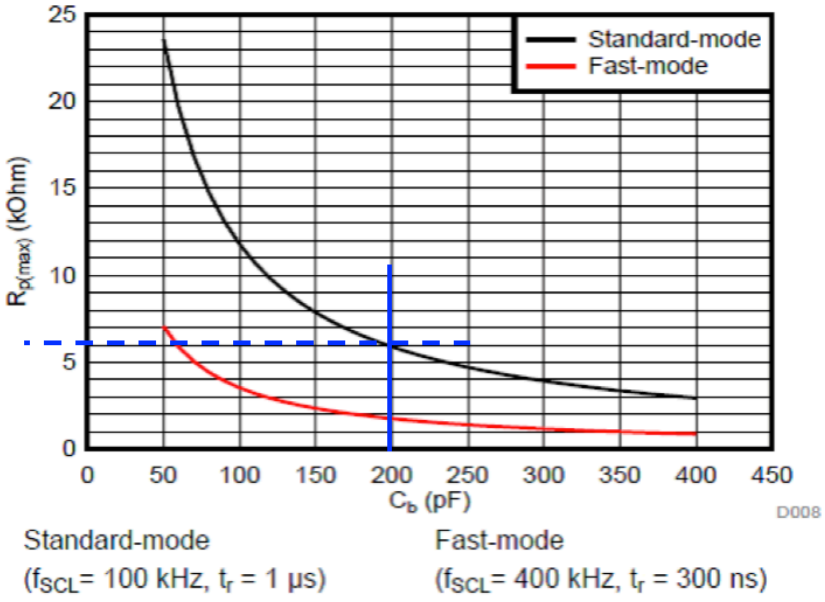


Figura III.9. Cálculo de la Resistencia máxima de pull-down.

III.4 Comunicación *serial* USB y UART/USART

Para comunicar el *dispositivo maestro*, que emplea el microcontrolador ATmega 2560, se usa el puerto *serial* que tiene vinculado a sus pines físicos 2 y 1 correspondientes a las entradas 0 y 1 del Puerto E.

Es una comunicación *full-duplex* (en ambos sentidos) que solo se puede dar entre dos dispositivos (a diferencia del bus *I²C* o *SPI* que permite múltiples dispositivos). Los datos necesitan ser empaquetados y pasados de paralelo a serie. Durante el empaquetado se le añaden una serie de bits que proporcionen cierta información sobre el envío; estos son los correspondientes a *start*, bit de paridad (par o impar) y a *stop*. Para realizar esta función se emplea una UART.

La UART (*Universal Asyncronos Receiver and Transmitter*) es un dispositivo que convierte todos los datos paralelos recibidos a *seriales* añadiéndole dichos bits. En su estructura interna cuenta con un generador de paridad, registros de desplazamiento y un oscilador variable para generar la velocidad de transmisión

La comunicación *serial* de Arduino funciona mediante lógica TTL y se necesita un convertidor de dicha lógica a USB, que son detectados por el PC como puertos *serial*es virtuales (COM). En este proyecto dicha conversión es realizada por el integrado CH340G [31] que necesita de un oscilador de cuarzo de 12 MHz para su funcionamiento. En la *Figura III.9* se muestra de forma esquemática el funcionamiento del conversor.

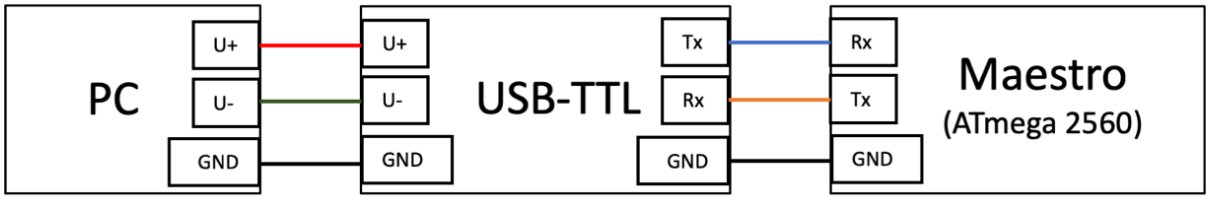


Figura III.10. Diagrama de bloques de conexión entre PC y Maestro.

III.5 Comunicación *SPI*

El *Serial Peripheral Interface* es un bus de datos paralelo desarrollado por Motorola. Presenta una arquitectura de *maestro-esclavo* similar a la explicada para el bus I^2C, pero en este caso se presenta una comunicación *Full-Duplex*.

Para ello, los cables de envío desde el *maestro* y *esclavo* son diferentes, siendo el denominado MOSI el encargado de transmitir los datos del *maestro* al *esclavo* y el MISO el empleado para la acción contraria.

Se trata de un bus síncrono, por lo que necesita una señal de reloj que es proporcionada por el microcontrolador mediante la línea CLK/SCK.

Por último, se necesita una línea de datos para cada *esclavo* con la que elegir con quien interactuar en cada momento, pues a diferencia del I^2C, no hay un valor de dirección de recepción que viaje junto con el mensaje por todos los receptores, sino que existe un cable por cada dispositivo. De esto se encarga el valor transmitido por el cable SS. Este pin se encuentra activo cuando no se desea realizar una comunicación, y en el momento en que se vaya a establecer la misma se pone a *low* la línea correspondiente.

En la configuración adoptada (*Figura III.9*) para la interconexión de los integrados MAX7219 [20] (que se explicará más detenidamente en el *Capítulo IV*), se envía el mensaje en serie a todos los *esclavos* empleando el pin *Dout* del primero para conectarlo al pin MOSI del segundo y de éste al tercero. Evitando así tener que ocupar un pin de SS para cada *esclavo*. Además, como estos dispositivos no envían datos al microcontrolador, la línea MISO no se emplea.

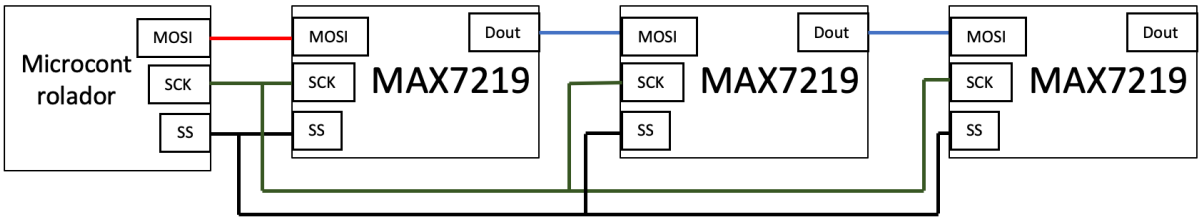


Figura III.11. Diagrama del bus SPI implementado.

Capítulo IV. Instrumentos de vuelo diseñados

En este capítulo se describen cada uno de los instrumentos simulados, explicando su funcionamiento a nivel *software* y *hardware*. Algunos de estos son controlados directamente por el *dispositivo maestro*, otros están conectados al bus I^2C y los restantes son manejados por un microcontrolador *esclavo* conectado a dicho bus. En total se diseñaron 18 de estos dispositivos todos dependientes de los mensajes que recibe el *maestro* conectado al ordenador.

IV.1 Aspectos generales del dispositivo

A grandes rasgos es posible clasificar los instrumentos en tres grupos de acuerdo al modo de funcionamiento:

- **Instrumentos *booleanos* de control**, entre los que se incluyen todos los pulsadores, interruptores e indicadores luminosos (LEDs de *warning* de aceite, presión, interruptores de luces exteriores, etc.).
- **Instrumentos *analógicos*** que serán simulados empleando motores paso a paso o servomotores (su selección ha dependido del rango de operación, en grados, del indicador) que mueven los mecanismos de desmultiplicación o las agujas directamente. Simulan los indicadores *tipo reloj* usados para las revoluciones del motor, nivel de combustible, altitud, velocidad vertical, aérea, rumbo, etc.
- **Pantallas digitales de información** para las radiofrecuencias, los valores del piloto automático programados (altitud, rumbo, velocidad, etc.) o el valor del *transponder*.

En la *Figura IV.1* se muestra el aspecto general del dispositivo en mayor profundidad que en el *Capítulo I*. El ATmega 2560 se conecta al PC de la forma explicada en el *Capítulo III*. A éste se conecta un panel de interruptores para el control de las luces, los calefactores de los tubos de pitot así como ciertos interruptores de control general de alimentación. El módulo de control de radio y piloto automático se instalan de forma similar; éstos incluyen pulsadores para activar el control de rumbo, la altitud, velocidad, modo de aproximación, etc. Todas las entradas se multiplexan y conectan al microcontrolador.

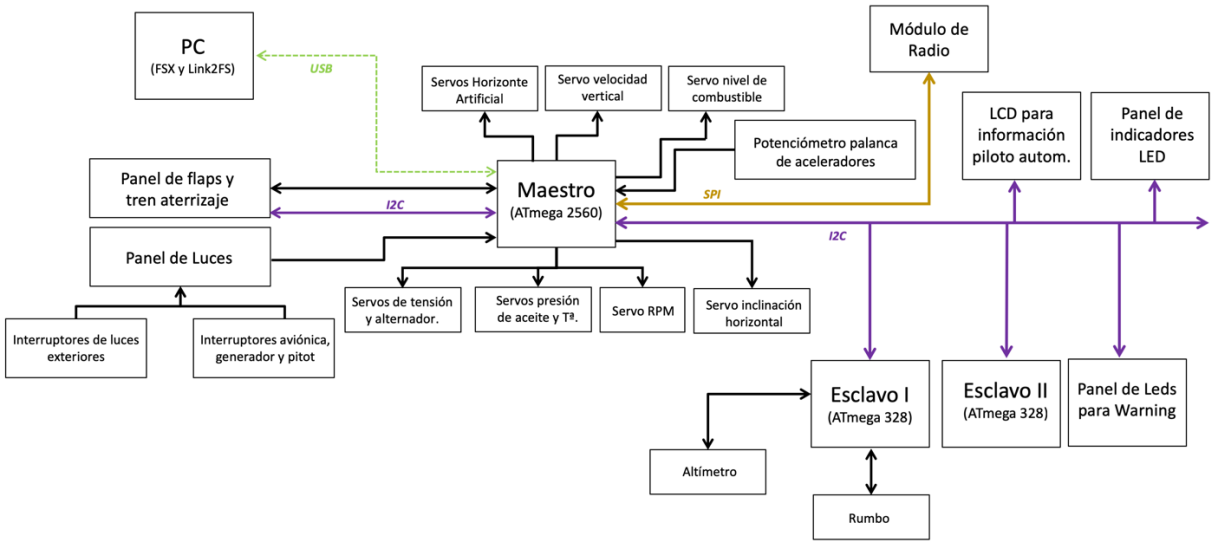


Figura IV.1. Diagrama de bloques detallado del dispositivo.(Puede verse en tamaño A4 en los Anexos).

El *Display 16x2* usado para los datos del piloto automático se conecta al bus I^2C mediante un expansor de 8 bits. Otros expansores similares (PCF8574) [17] se emplean para multiplexar las salidas de los avisos, de modo que no se ocupen pines digitales del ATmega. A este bus se unen también dos ATmega 328 que se encargan de mover los motores paso a paso para simular los *gauges* de rumbo (HDG) y altitud.

Para concluir, el bus *SPI* se utiliza para conectar el módulo de radio, que consta de varios integrados con los que controlar pantallas de 7 segmentos y una serie de pulsadores y *encoders* conectados al *maestro* mediante cable paralelo.

IV.2 Panel de comunicación vía radio

Las comunicaciones entre los pilotos y el personal de tierra resulta indispensable en cualquier vuelo. El piloto debe de tener en todo momento sintonizada la frecuencia correspondiente a la zona en la que está volando.

Cada panel de radio, también llamado COMM por su siglas en ingles de *communication*, cuenta con dos frecuencias: la principal/sintonizada y la de espera/*stand-by*. Esto permite el paso rápido de una comunicación a otra, es decir, cuando un piloto por ejemplo se está comunicando con personal del ATC[10] y va a finalizar la comunicación con éste porque entra en otro espacio aéreo, se le comunica la nueva frecuencia para que la sintonice; una vez sale de dicho espacio aéreo,

[10] ***ATC:*** *Air Traffic Control.* Es el servicio proporcionado por el personal de tierra para guiar al avión durante su ruta.

intercambia las frecuencias de escucha y espera, quedando automáticamente conectado con el nuevo controlador.

La avioneta Cessna cuenta con dos módulos COMM. Cada uno funciona de forma independiente, permitiendo al equipo de vuelo alternarlos en caso de fallo sin perderse la comunicación.

El personal de Control de Tráfico Aéreo necesita además identificar cada uno de los aviones. De esto se encarga un código octal único de 4 dígitos que es indicado al comandante desde el momento en que comienza su rodadura en pista y que no debe modificarse, salvo avería o secuestro de la aeronave. A esto se le conoce como transpondedor y se envía por VHF[11] al igual que la comunicación de radio.

Durante este proyecto se desarrolló únicamente el panel COM1 junto con el módulo de transpondedor, dado que el otro consiste en reproducirlo de forma idéntica.

IV.2.1 Diseño mecánico

Desde el punto de vista mecánico se trata de un dispositivo relativamente sencillo compuesto por los siguientes elementos:

- 5 conjuntos de *Displays* 7 segmentos de 4 dígitos y 4 de 1 dígito con los que se forman los indicadores de la frecuencia de radio sintonizada (en MHz con dos decimales), la de espera, las dos frecuencias del navegador (espera y activa) y el código de *transponder*.
- Un *encoder* para sintonizar la frecuencia de la radio en *stand-by*. Se emplea el botón que incorpora para elegir la escala de la radio a modificar (entre kHz o MHz).
- Un pulsador que permite el intercambio entre la frecuencia principal y la de espera, llamado *Swap button*, y otros similares para alternar el NAV así como para activar o desactivar comunicaciones, DME[12] y ADF[13].
- Un segundo *encoder* para sintonizar las frecuencias del navegador. Funciona de forma similar al codificador de las comunicaciones.
- Otro *encoder* para modificar el dígito seleccionado del XPDR[14]. Se emplea cada pulsación del mismo para seleccionar un nuevo dígito a modificar.

---

[11] ***VHF:*** *Very High Frequency*. Banda del espectro electromagnético que ocupa entre los 30 MHz y los 300 MHz.
[12] ***DME:*** Sistema de medición de distancia entre un objeto y una estación emisora. Permite reemplazar el sistema de radiobalizas y tener una medición de la distancia del avión a la pista durante el aterrizaje.
[13] ***ADF***: Dispositivo capaz de detectar la dirección en la que vienen las señales, lo que permite orientar al avión respecto a una determinada pista de aterrizaje
[14] ***XPDR:*** Transpondedor.

El módulo original de Cessna 172 se muestra a continuación (*Figura IV.2*).

Figura IV.2. Panel original de radio de la Cessna 172.

El módulo desarrollado consta en la primera fila con los pulsadores de las funciones, en la segunda dos grupos de *displays* de 5 dígitos para la frecuencia de Radio *activa* y *Stand-By*. Debajo se sitúa su homólogo para las frecuencias de la asistencia a la navegación y aterrizaje guiado (ILS[15]). En la última fila se muestra el código de transpondedor (*Figura IV.3*).

Figura IV.3. Módulo de comunicaciones diseñado.

Al lado de cada conjunto se instaló el *swap button* correspondiente.

[15] ***ILS:*** Sistema de aproximación y aterrizaje guiado de un avión establecido como estándar en todo el mundo.

IV.2.2 Diseño electrónico

Con el objetivo de poder controlar todos los *displays* desde el mismo *maestro* sin ocupar una excesiva cantidad de pines, se decidió emplear la comunicación *SPI* combinada con unos decodificadores y multiplexores 7 segmentos de la familia MAX, concretamente el MAX 7219 [20].

Dado que cada integrado puede controlar hasta 8 *displays*, se han precisado tres de estos conectados en cascada. El primero de ellos (denominado U1 en la *Figura IV.4*) controla los dígitos del 1 al 4 del NAV principal y los 4 dígitos del XPDR. Éste se conecta a los pines MOSI, SCK y SS[16] del microcontrolador. La salida DOUT transfiere los dígitos sobrantes del mensaje *serial* al segundo integrado (U4 en el esquema) que controla los 4 dígitos menos significativos de la radio principal y del NAV en espera. De igual forma, el último de los integrados (U5) maneja los dígitos del COM en espera y los valores de las centenas de MHz de todos los grupos.

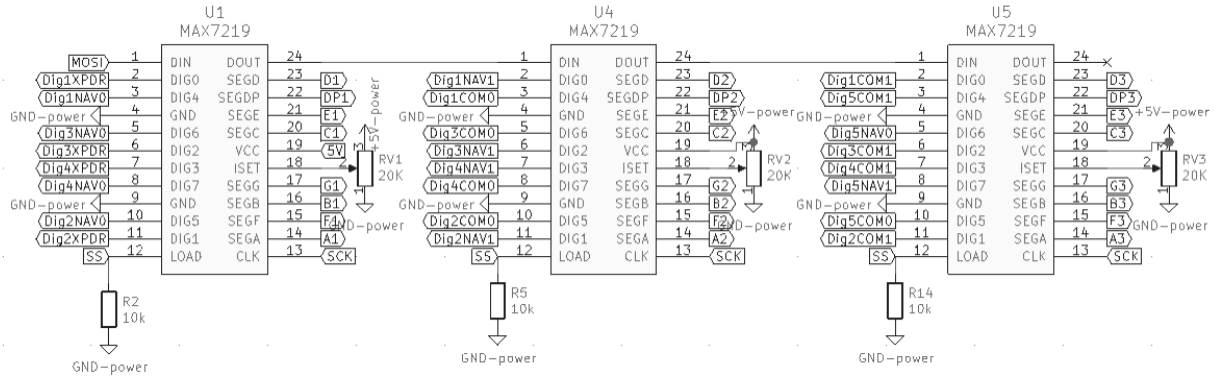


Figura IV.4. Esquema de conexión de los MAX7219 al bus SPI.

Estos integrados, se encuentran en un circuito impreso (*Figura IV.5*) que aloja además, los pulsadores, *encoder*s y los propios *displays*, conectándose a la placa principal del *maestro*, donde se localizan los multiplexores que se explicarán más adelante, mediante cable IDC[17] y el bus *SPI* ya citado.

[16] ***MOSI, SCK,SS***: Tres líneas de comunicación empleadas para *SPI*.
[17] ***IDC***: Tipo de conector para cable paralelo muy usado en informática.

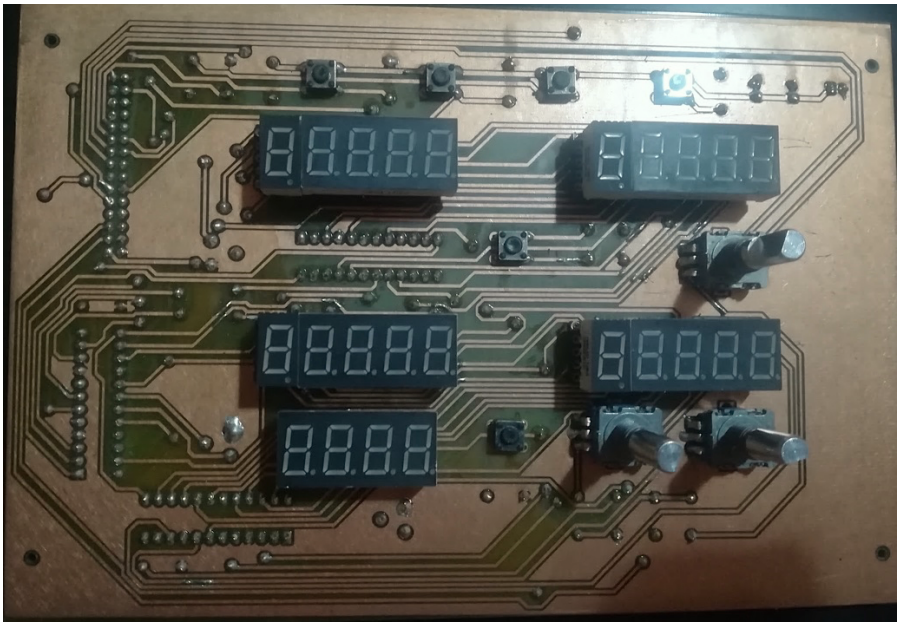


Figura IV.5. PCB fabricada del módulo de radio.

Para controlar el conjunto de *displays* se creó una librería basada en la librería `SPI.h` del propio Arduino. Se ha denominado `MAX7219.h` y consta de los siguientes métodos:

`MAX7219()`: Es el constructor de la clase.

`init(const int decodemode,const int test, const int intensidad,const int encendido_apagado)`: Este método inicializa el conjunto de los tres integrados. El primer parámetro indica el tipo de dato a recibir, si solo números o además otra información. En el presente caso para indicar que únicamente se envían números se carga el valor más alto (0xFF) en el registro correspondiente, que según el fabricante en su *datasheet* [20] es el 0x9. El parámetro `test` es un entero que indica si va a funcionar en modo normal o de prueba y su valor se carga en el registro 0xF. El siguiente parámetro permite ajustar el brillo de los *displays*, enviando y cargando un número entre 1 y 15 en el registro 0xA para regularlo. Por último, con un 1 en el atributo `encendido_apagado` se carga un valor *verdadero* en el registro 0x9 consiguiendo encender los dispositivos.

El último de las métodos empleados es el `enviaString(String s)` con el que se manda el conjunto de números a reproducir en la radio. Para su ejecución se llama a otros métodos de la librería que permiten enviar cada caracter por el bus.

Para tres integrados el *string* del mensaje debe tener una dimensión de 24 caracteres y debido a la distribución de las conexiones se codifica de la siguiente forma.

E. NAV0	D. NAV0	C. NAV0	B. NAV0	XPDR0	XPDR1	XPDR2	XPDR3

E. COM0	D. COM0	C. COM0	B. COM0	E. NAV1	D.NAV1	C.NAV1	B.NAV1

A. NAV0	A. NAV1	A. COM0	A. COM1	E. COM1	D. COM1	C. COM1	B. COM1

Donde D representa centenas de KHz, E decenas de KHz, C unidades de MHz, B decenas de MHz y A millares de MHz. De este modo, para una frecuencia cualquiera el número a mostrar sería:

ABC.DE (MHz)

A modo de ejemplo, si los valores proporcionados por el simulador (resaltados en colores diferentes para su fácil identificación en el mensaje) fuesen los siguientes:

- Frecuencia de radio activa: **129.20**
- Frecuencia de radio en espera: **145.26**
- Frecuencia NAV activa: **136.40**
- Frecuencia NAV en espera: **109.35**
- Transpondedor: **2574**

El mensaje a cargar mediante `enviaString` sería:

0 4 6 3 4 7 5 2 0 2 9 2 1 1 1 1 5 3 9 0

A nivel de *software*, el mensaje se consigue escribir creando un primer *string* con 24 caracteres y según se van obteniendo los valores del simulador, almacenados en *strings* de tamaño 5, se accede a cada uno como si se tratara de un vector, y se vuelca ese valor en la posición correspondiente del *string* creado anteriormente.

Por otro lado, todos los pulsadores se encuentran conectados al *maestro* mediante multiplexores. Se ha decidido emplear 6 multiplexores de 8 bits conectados en cascada a otro de 8 con el objetivo de poder multiplexar hasta 48 entradas. En esta primera versión del proyecto se emplearon 38 entradas, quedando libre 10 adicionales, que se han dejado fáciles de conectar en la PCB mediante un *pinHeader*[18] de 2x5.

Las entradas correspondientes al panel de radio se conectan al multiplexor U1 y U4 con una resistencia de *pull-down* de valor 1 kΩ en cada uno. Sus salidas se enlazan a las entradas *A4 (AM4)* y *A0* (*AM0*) del multiplexor principal (*U7*), que a su vez va unido a la entrada *D28* (*Figura IV.6*). Mediante 6 pines de salida de Arduino (*D22, D23, D24* para los *MUX*[19] secundarios y *D25, D26,* D27 para el *MUX* principal) se selecciona la entrada a leer.

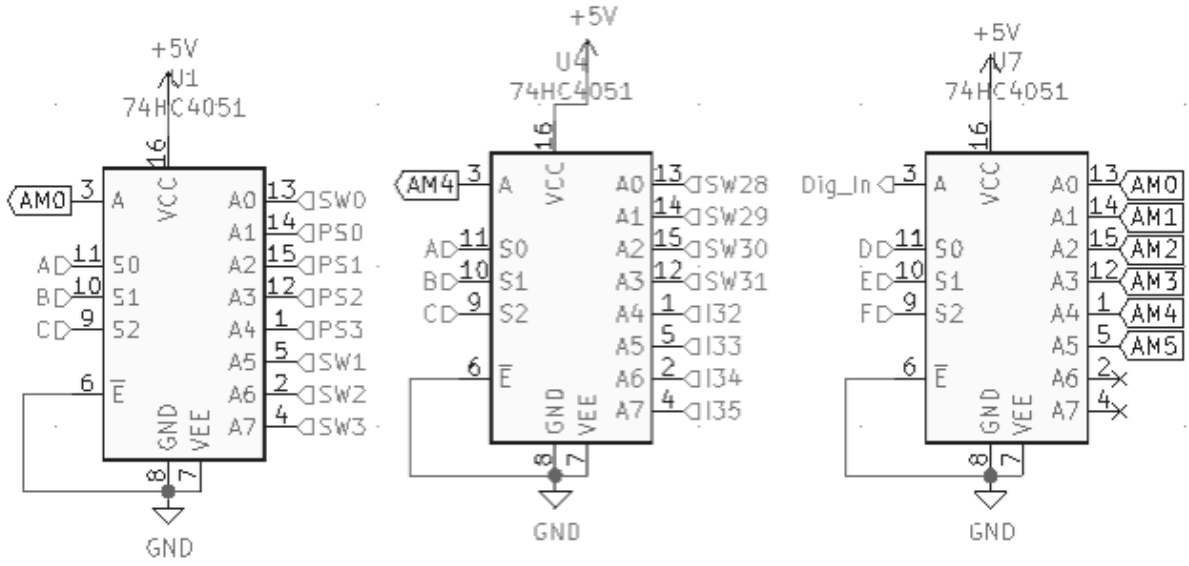


Figura IV.6. Conexionado de los multiplexores, interruptores y pulsadores.

A continuación, en la *Tabla IV.1*, se muestran las combinaciones para la lectura de cada entrada.

[18] ***PinHeader***: tiras de pines macho para conectores de tipo *dupond*.
[19] ***MUX:*** Multiplexor.

Tabla IV.1. Valores de los pines en función de la entrada.

Nombre del pin	**Nº pin**	**MSB principal**	**Bit 1 principal**	**LSB principal**	**MSB Mux i**	**Bit 1 Mux i**	**LSB Mux i**
Cambiar Escala frecuencia radio	PS2	0	0	0	0	1	1
Swap Radio	PS3	0	0	0	1	0	0
Cambiar dígito XPDR	PS1	0	0	0	0	1	0
Swap NAV	I33	1	1	0	0	0	0
Cambiar escala NAV	I32	1	0	0	1	0	0
ADF	SW31	1	0	0	1	1	0
DME	SW31	1	0	0	0	1	0
NAV1	SW29	1	0	0	0	0	1
COM1	SW28	1	0	0	0	0	0

El sentido de giro de los *encoders* conectados es leído empleando `getGiro()` (definido en una librería `encoder.h` propia) (*Figura IV.7*) que guarda la posición actual y la compara con la anterior para determinar la dirección de giro. Devuelve un 1 si el movimiento es horario y -1 si es antihorario. Sus pulsadores cambian el valor de la escala seleccionada devolviendo un 0 para los kHz y 1 para los MHz.

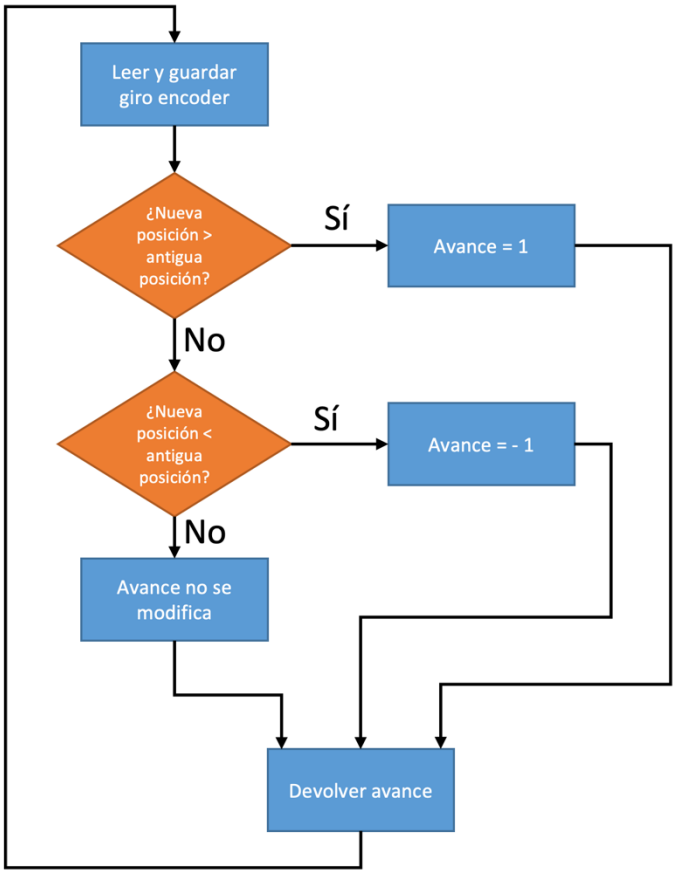


Figura IV.7. Algoritmo de determinación de sentido de giro.

IV.3 Panel de piloto automático

El piloto automático es el dispositivo encargado de controlar de forma autónoma los parámetros de vuelo tales como la altitud, dirección, velocidad aérea y de ascenso. Éste envía la información a un módulo central que actúa sobre la mezcla de aire-combustible del motor, las revoluciones y los alerones para conseguir el ángulo de giro o de ascenso a la velocidad deseada.

Algunos incluyen la opción de sintonizar las radio-balizas de la pista de aterrizaje permitiendo realizar un aterrizaje mediante el ILS. Se activa mediante el botón de APR[20]. Las aeronaves que cuentan con ese dispositivo permiten lo que se conoce en aviación como vuelo instrumental, pudiendo volar con clima adverso y baja visibilidad.

[20] ***APR:*** Aproximación. Maniobra de acercamiento del avión a la pista de aterrizaje.

IV.3.1 Diseño mecánico

Este aparato cuenta con un *display* LCD de 2 filas con 16 caracteres cuyo conexionado se realiza mediante el bus I^2C (para minimizar el número de hilos) y empleando las funciones de la librería `LiquidCristal_PCF8574.h` [32] para su escritura. En la pantalla se ha instalado un PCF8574 para traspasar el mensaje *serial* recibido por el bus a un mensaje de bytes paralelos transmitido al monitor.

El dispositivo original para aviones ligeros presenta el aspecto de la *Figura IV.8.*

Figura IV.8. Panel del piloto automático de una Cessna.

Los botones **UP-DN** se han sustituido por un *encoder* configurado de forma similar a los del módulo de radio, con el que poder incrementar o decrementar el valor de la altitud y la velocidad vertical, alternando con cada pulsación del botón incorporado. El resto de pulsadores se han simulado mediante interruptores retroiluminados con LEDs rojos que se encienden para indicar la activación de dicha función.

El primero de los botones de la izquierda (**AP**), controla el encendido de todo el módulo, es decir, activa o desactiva el piloto automático. A su derecha le sigue el **HDG**, que se encarga de activar la dirección de rumbo especificada en la brújula magnética.

El **NAV** gestiona el seguimiento de la radioseñal VOR1 emitida desde tierra. En cada aeropuerto se localiza una torreta que emite en muy alta frecuencia (VHF) una señal modulada con dos senos: uno de ellos con su fase fija y el otro con su fase variable según la dirección en la que se emite. La estación gira a una velocidad angular constante emitiendo en todas las direcciones con un desfase diferente. El piloto automático, recibe esta señal y conociendo su desfase puede determinar el ángulo de separación respecto de la emisora, para así trazar la ruta de navegación [33].

El **APR** enciende el modo aproximación siguiendo la ruta marcada por el ILS de la pista (como se comenta en la introducción de este apartado). Se dispone de un ILS diferente por cada una, por lo que debe ser sintonizada en su frecuencia correcta mediante el módulo de navegación (que se ha implementado dentro del módulo de

comunicaciones ya explicado). Una de las dos señales guía al piloto horizontalmente y la otra verticalmente, hasta el momento en que se disponga de suficiente visibilidad para aterrizar.

El botón **ALT** activa el control de altitud, que permite, como su nombre indica, mantener una altura fijada. La señal es recibida por el altímetro que se explicará en unas secciones más adelante de este mismo capítulo.

Todos estos controles se han dispuesto en el módulo diseñado a medida que muestra la siguiente figura.

Figura IV.9. Módulo de piloto automático fabricado.

Se conectan al *maestro* mediante los multiplexores siguiendo un esquema similar al de la radio. En este caso se emplean las entradas desde la *A2* hasta la *A6* multiplexor U3 *(Figura IV.10)*. El *encoder KY-040* se conecta con sus dos terminales de señal a los pines *D34* y *D35* del microcontrolador, así como su botón (en configuración *pull-up*) unido al pin *A1* del multiplexor U5.

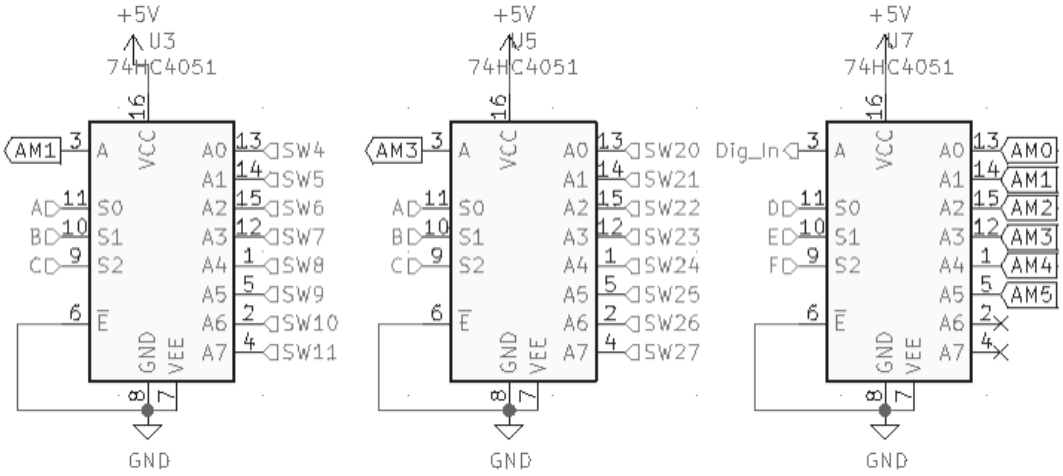


Figura IV.10. Esquema de conexión de los interruptores del AP a los multiplexores.

Se selecciona en cada momento la entrada a leer siguiendo los valores de la siguiente tabla.

Tabla IV.2. Selección de las entradas del piloto automático.

Nombre del pin	**Nº pin**	**MSB principal**	**Bit 1 principal**	**LSB principal**	**MSB MUX i**	**Bit 1 MUX i**	**LSB MUX i**
Cambiar VS y ALT	SW21	0	1	1	0	0	1
AP	SW6	0	0	1	0	1	0
APR	SW7	0	0	1	0	1	1
HDG	SW8	0	0	1	1	0	0
NAV	SW9	0	0	1	1	0	1
ALT	SW10	0	0	1	1	1	0

Las salidas de este módulo, es decir, los LEDs que contienen los propios interruptores, se conectan a otro expansor I^2C. En este caso, se envía un byte con un valor de *on/off* en cada bit correspondiente a un indicador y se conectan siguiendo las indicaciones del fabricante del componente [17, p. 16] (*Figura IV.11*). De este modo, en paralelo con la resistencia limitadora del LED se coloca una resistencia de 100 kΩ para mantener polarizados los transistores FET del integrado. En esta configuración se trabaja con lógica negada.

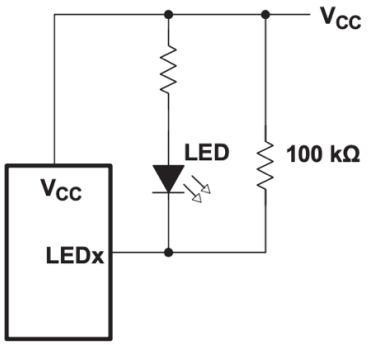


Figura IV.11. Conexión de los LEDs al PCF8574.

Este expansor presenta la dirección hexadecimal 0x38, obtenida uniendo sus 3 pines de selección a *Vcc*. Las etiquetas *LED+* y *LED-* (*Figura IV.12*) hacen referencia a los pines donde se ha colocado cada puerto para la conexión de los LEDs.

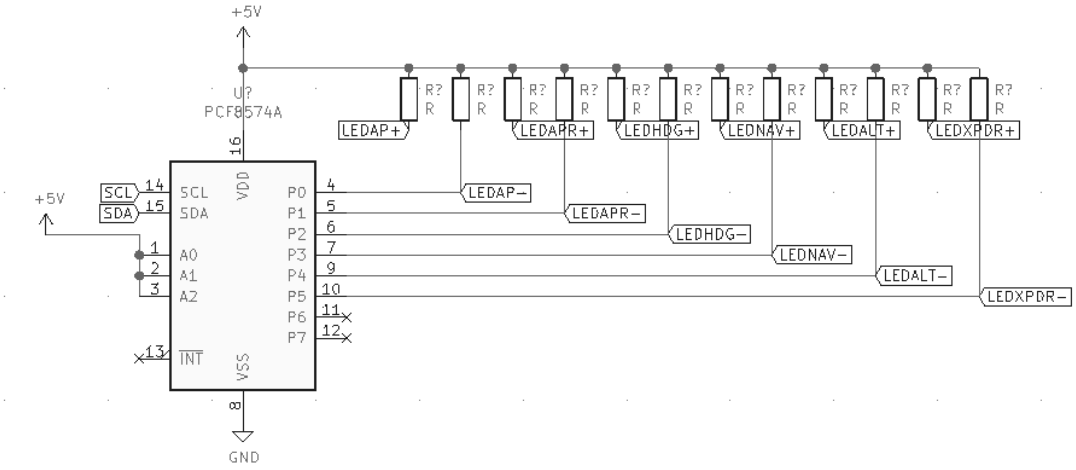


Figura IV.12. Conexión de los LEDs del AP al PCF8574 (expansor).

El byte a enviar contiene los estados de la forma mostrada en *Tabla IV.3*, donde las *X* representan términos *don´t care*.

Tabla IV.3. Significado de los bits del mensaje enviado por I²C.

B7 (MSB)	**B6**	**B5**	**B4**	**B3**	**B2**	**B1**	**B0 (LSB)**
X	*X*	LED XPDR	LED ALT	LED NAV	LED HDG	LED APR	LED AP

Para unificar todo lo relativo a este módulo, se ha diseñado otra PCB (*Figura IV.13*) que incluye dicho expansor, las resistencias limitadoras y de polarización de los LEDs, además de las de *pull-down* de los 5 interruptores. Todas estas señales se cablean al dispositivo *maestro* mediante otro conductor paralelo y el bus *I²C*.

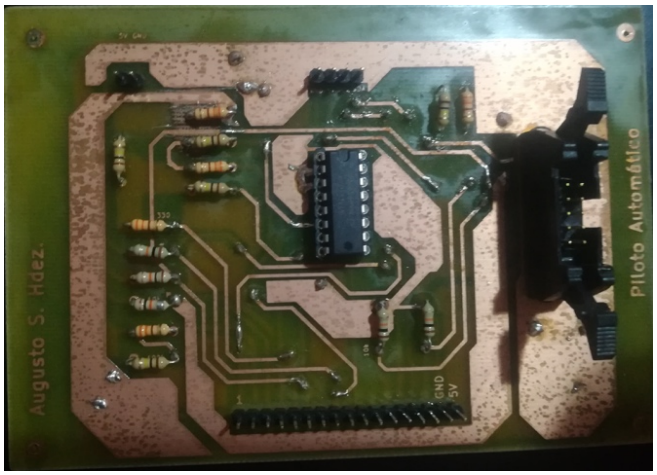


Figura IV.13. PCB del piloto automático.

IV.4 Panel de *switches*

Este modelo de avioneta presenta un conjunto de interruptores (*Figura IV.14*) en la parte inferior izquierda de la cabina desde la que se controlan todas las luces y sistemas auxiliares del avión. Se incluye también una llave de arranque (magneto).

Figura IV.14. Conjunto de switches de una Cessna 172.

IV.4.1 Diseño mecánico

El módulo está formado por un conjunto de *toggle-switches*[21] que se conectan de forma multiplexada al dispositivo *maestro* (ATmega 2560). A continuación se describe la funcionalidad de cada interruptor:

- ***Fuel Pump switch***: Este interruptor permite activar la bomba que suministra el combustible a su único motor tetracilíndrico.
- ***BCN light switch***: Activa las luces *Beacon*. Se trata de indicadores de color rojo que se sitúan en la cola del avión y tienen como propósito señalizar a otros aviones de su existencia para evitar una colisión. Trabajan conjuntamente con las luces estroboscópicas [34].
- ***Land switch:*** Conecta las luces empleadas durante el despegue y aterrizaje para alumbrar la pista y dotar de visibilidad al piloto. Son las de mayor intensidad y se sitúan en las alas [34].
- ***Taxi light switch:*** Controla el encendido de las luces situadas en el tren de rodaje delantero cuyo propósito es iluminar las líneas de guía de maniobra localizadas en la pista de rodaje.

[21] ***Toggle-switch:*** interruptor/conmutador de tipo palanca.

- ***NAV light switch:*** Interruptor para activar las luces de uso obligatorio durante la navegación. Cada zona del avión presenta una luz de color diferente, permitiendo a otros aparatos identificar el sentido y la dirección de vuelo del mismo.
- ***Strobe light switch:*** Con éste se encienden las luces estroboscópicas situadas en las alas. Parpadean a intervalos de tres segundo y forman parte del sistema anticolisiones (al igual que las *Beacon lights*).
- ***Calefactor de los tubos de pitot:*** Los tubos de pitot son los dispositivos encargados de medir la velocidad relativa del avión respecto del aire. El problema de los mismos es su alta probabilidad de congelación por lo que con este botón se procede a activar un calefactor que evita que se hielen y falseen la medida. [35]

Figura IV.15. Panel de interruptores fabricado.

En los extremos de dicha botonera se sitúan los interruptores de los sistemas generales del avión:

- ***Master Alt switch:*** Interruptor que activa el sistema del alternador permitiendo alimentar todos los dispositivos del avión mediante la energía del motor y cargar la batería con el excedente.
- ***Master Bat switch:*** Con éste se conecta la batería como fuente de alimentación de la electrónica del avión.
- ***Avionic Master switch:*** Al ser encendido provee de energía a la aviónica[22].

[22] ***Aviónica:*** Conjunto de aparatos electrónicos empleados en la navegación aérea. [51]

Cada cilindro del motor cuenta con dos juegos de bujías permitiendo una mejor combustión y un funcionamiento seguro en caso de fallo de una. El arranque se realiza mediante un magneto compuesto por las siguientes posiciones:

- ***R***: Activa un juego de bujías del motor.
- ***L:*** Activa el segundo juego de bujías.
- ***Both:*** En esta posición ambos conjuntos están operativos.
- ***Start:*** Acciona el motor de arranque que provoca el encendido. Una vez arrancado debe volver y permanecer en la posición *both* para alimentar ambos sistemas de encendido.

Esta llave se simuló empleando un interruptor rotativo acoplado a un eje realizado en PLA[23] que se conectó a una cerradura *tipo taquilla* (*Figura IV.16*).

Figura IV.16. Mecanismo de llave para el magneto de arranque.

[23] ***PLA:*** Tipo de material plástico empleado para la fabricación de piezas mediante impresora 3D.

IV.4.2 Diseño electrónico

Todos los dispositivos se comunican con el microcontrolador *maestro* mediante un esquema de multiplexores (*Figura IV.17*) como los anteriores.

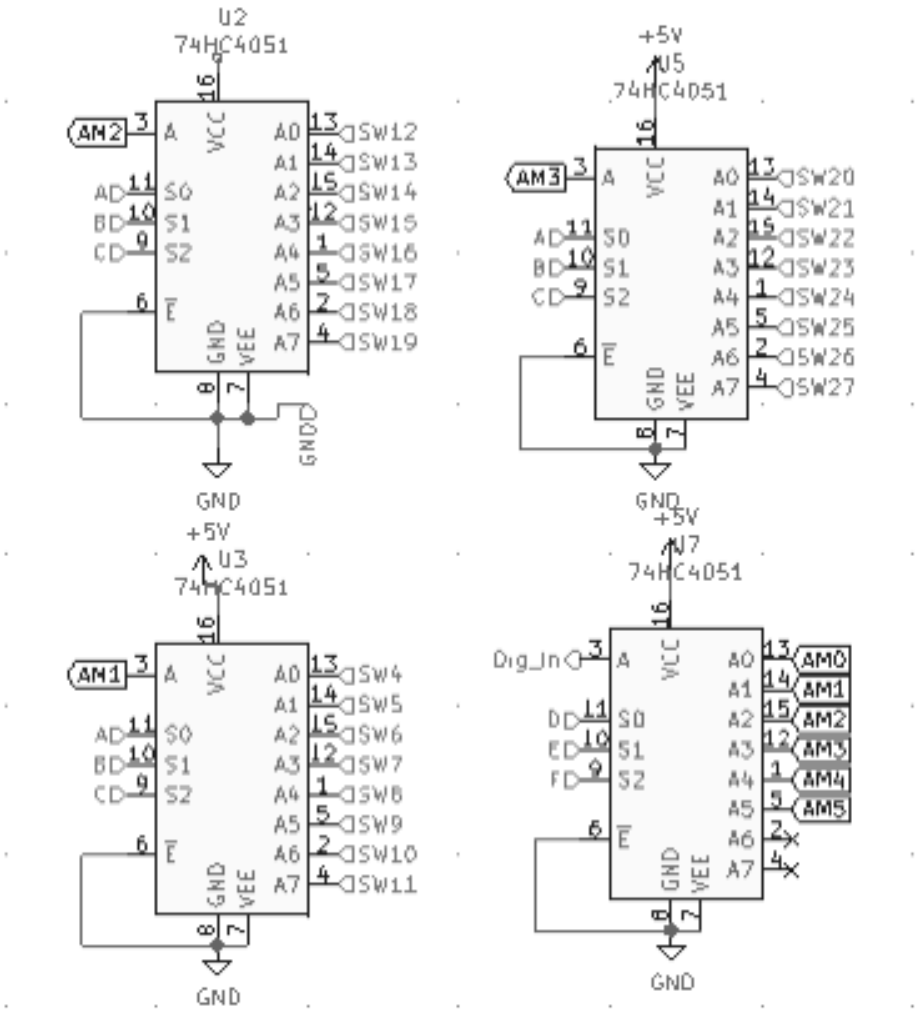


Figura IV.17. Conexiones del módulo de luces.

Como ya se ha explicado su modo de funcionamiento y conexión, se indica únicamente las secuencias necesarias para su lectura (*Tabla IV.4*).

Tabla IV.4. Tabla de verdad para entradas del panel de luces.

Nombre del pin	Nº pin	MSB principal	Bit 1 principal	LSB principal	MSB MUX i	Bit 1 MUX i	LSB MUX i
Fuel pump	SW11	0	0	1	1	1	1
BCN	SW12	0	1	0	0	0	0
LAND	SW13	0	1	0	0	0	1
TAXI	SW14	0	1	0	0	1	0
NAV	SW15	0	1	0	0	1	1
STROBE	SW16	0	1	0	1	0	0
PITOT	SW17	0	1	0	1	0	1
ALT	SW18	0	1	0	1	1	0
BAT	SW19	0	1	0	1	1	1
Aviónica	SW20	0	1	1	0	0	0
R (motor)	SW23	0	1	1	0	1	1
L(motor)	SW24	0	1	1	1	0	0
Both(motor)	SW25	0	1	1	1	0	1
Start (motor)	SW26	0	1	1	1	1	0

IV.5 Panel de tren de aterrizaje y flaps

Los flaps son dispositivos situados en el borde del ala que regulan la sustentación del avión, mejorándola, según el ángulo de inclinación del mismo. Su apertura facilita el despegue y aterrizaje a bajas velocidades.

Por otra parte, el control de tren de aterrizaje permite extenderlo o retraerlo. En la Cessna 172 es fijo, por lo que este botón no es necesario. Sin embargo, se ha decidido instalarlo con el objetivo de permitir reutilizar el simulador para otros aviones ligeros de tren retráctil como la *Beechcraft Baron 58* [36].

Para este módulo se han empleado dos interruptores rotativos, cada uno de ellos acoplados a una palanca impresa en 3D (*Figura IV.18*). Dicho mecanismo permite obtener de forma sencilla dos conmutadores verticales.

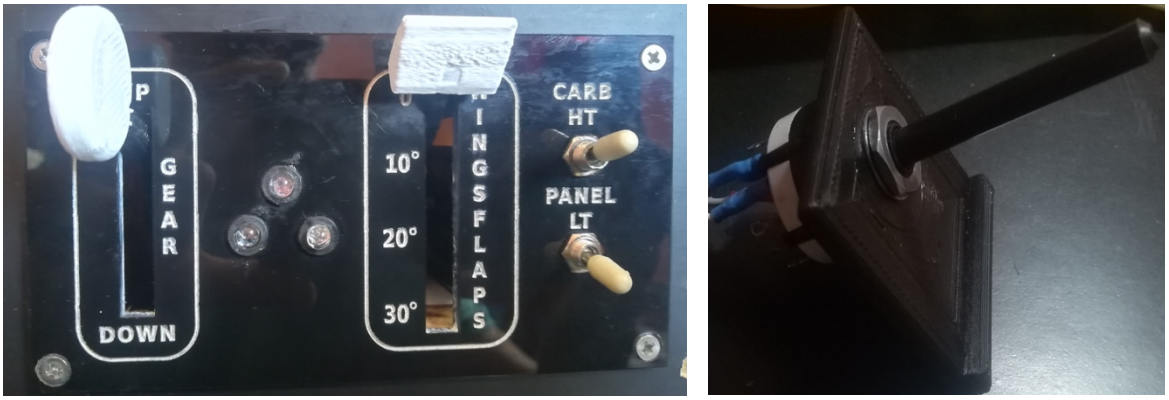


Figura IV.18. (A) Panel de tren diseñado. (B) Acople del eje al interruptor rotativo.

Para los indicadores del estado del tren se ha hecho uso del mismo tipo de expansor que en el módulo de piloto automático, empleando en esta ocasión LEDs bicolor (de ánodo común) cuyos cátodos se conectan a las salidas del PCF8574 (*Figura IV.19*). Este circuito se localiza en otra PCB que fabricada para albergar estas conexiones.

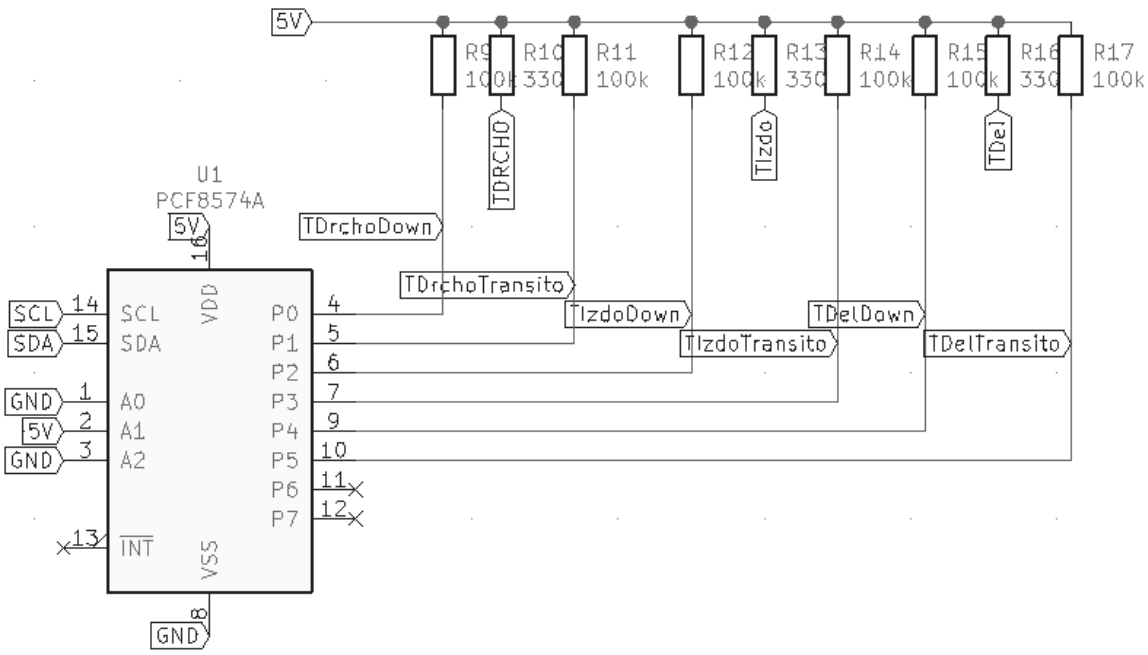


Figura IV.19. Conexiones de los indicadores del tren de aterrizaje.

La secuencia para la lectura de los conmutadores conectados a los multiplexores U1 y U3 se muestra a continuación.

Tabla IV.5. Tabla de lectura de conmutadores.

Nombre del pin	**Nº pin**	**MSB principal**	**Bit 1 principal**	**LSB principal**	**MSB MUX i**	**Bit 1 MUX i**	**LSB MUX i**
Gear Up	SW0	0	0	0	0	0	0
Gear Down	PS0	0	0	0	0	0	1
Flaps0	SW1	0	0	0	1	0	1
Flaps10	SW2	0	0	0	1	1	0
Flaps20	SW3	0	0	0	1	1	1
Flaps30	SW4	0	0	1	0	0	0

IV.6 Módulo de avisos (*Warning*)

La cabina en cuestión consta de un pequeño panel de indicadores luminosos con avisos relativos a baja presión de aceite, bajo nivel de carburante en algún depósito o pérdida de vacío en la cabina.

Figura IV.20. Panel de avisos de Cessna 172 construido con Leds.

Se han simulado empleando un panel de metacrilato con las letras grabadas y unos LEDs en la parte posterior de modo que se iluminen los avisos, como se observa en la *Figura IV.20*. Éstos se han conectado desde el circuito impreso diseñado (*Figura IV.22*) al bus I^2C mediante otro expansor (*Figura IV.21*). En este caso, los pines A1 y A2 se conectan a tierra y el A0 a 5V obteniendo *0x39* como su dirección.

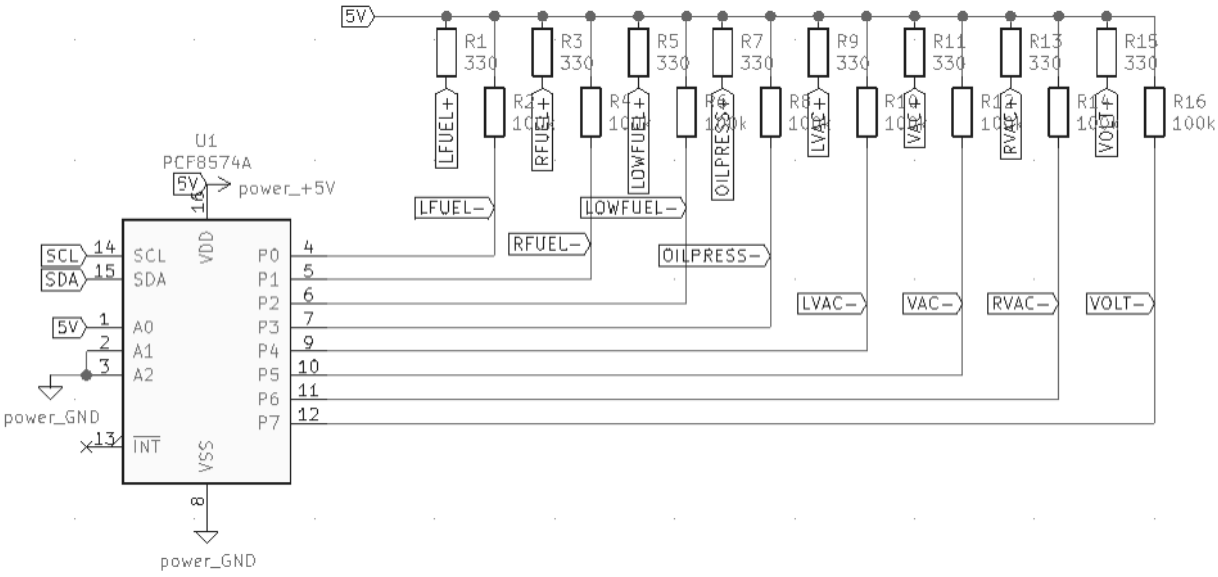


Figura IV.21. Esquema de conexiones del módulo de Warning.

El mensaje a transmitir por el bus contiene la siguiente información.

Tabla IV.6. Significado de los bits del mensaje al módulo Warning.

B7 (MSB)	**B6**	**B5**	**B4**	**B3**	**B2**	**B1**	**B0 (LSB)**
LED VOLT	LED VAC	LED VAC RIGHT	LED VAC LEFT	LED OIL PRESS	LED FUEL LOW	LED FUEL RIGHT	LED FUEL LEFT

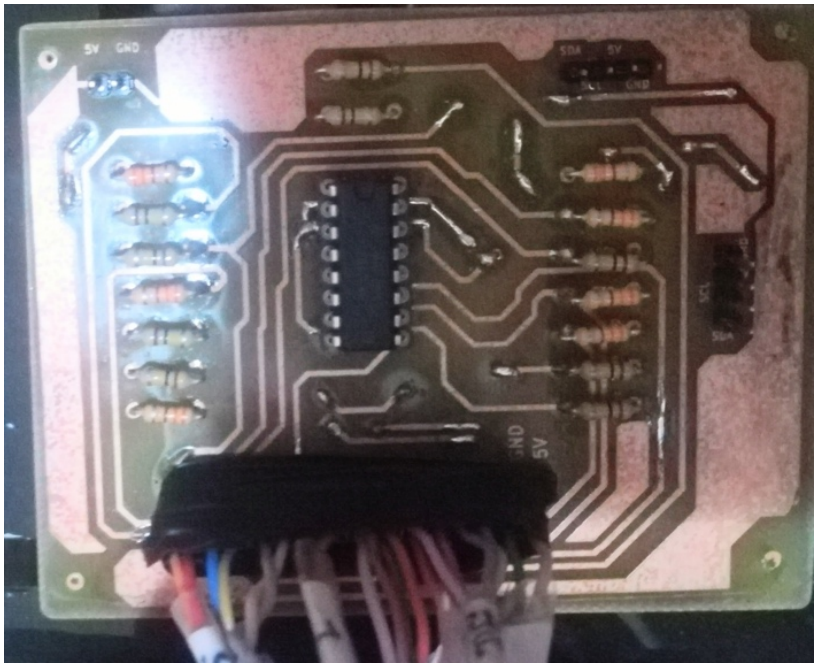


Figura IV.22. PCB fabricada para avisadores.

IV.7 Panel de acelerador, gases y *trim*

Esta avioneta, que como se ha comentado, está formada por un motor de funcionamiento similar a los de los vehículos de carburación, consta de tres mandos para el control de la potencia del mismo.

- ***Mando de gases***: Se trata del mando situado en la parte central del módulo (*Figura IV.23*). Controla la potencia del motor.
- ***Mando de mezcla:*** Situado a la derecha, es de color rojo y permite regular la relación aire-combustible de la mezcla. En la realidad lleva un botón en la parte posterior que permite su bloqueo, el cual por simplicidad en el diseño se ha eliminado.

- ***Mando de calefacción del carburador:*** Se sitúa a la izquierda del panel. Su función es la de calentar el carburador para evitar que se forme hielo durante los trayectos.

Figura IV.23. Panel de gases construido.

IV.7.1 Diseño mecánico

Para su diseño se han impreso unos cojinetes en PLA (*Figura IV.24*) que permiten la unión del eje roscado de 6 mm empleado con el potenciómetro lineal de 10KΩ conectado al microcontrolador. Al extremo de estas varillas se han insertado los tiradores con una forma lo más similar posible a los originales, respetando sus colores y dimensiones.

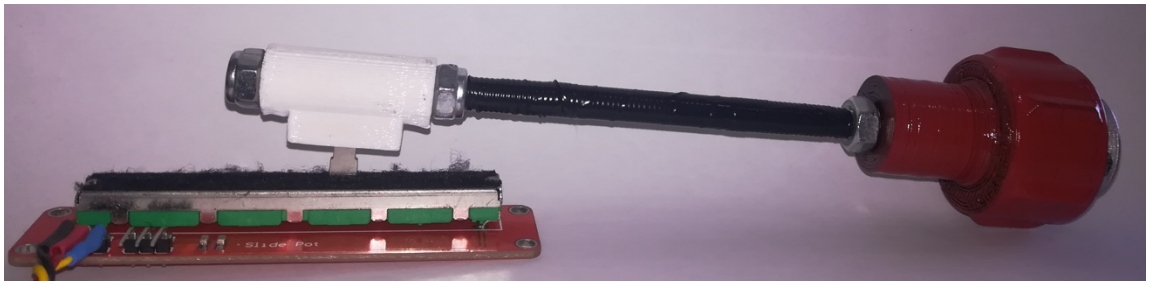


Figura IV.24. Cojinete para la unión del eje del acelerador y el potenciómetro.

Mediante el deslizamiento de estos tres ejes, se cambia el valor de la tensión leída para modificar el valor correspondiente en el simulador, como se detallará en el *Diseño electrónico.*

Por último, para el ajuste de *trim*[24], se emplea el mismo esquema de funcionamiento, pero dado que el movimiento de esta pieza es en forma de revolución, se decidió emplear un *encoder* rotativo al que se le acopla en su eje una rueda impresa en 3D (*Figura IV.25*).

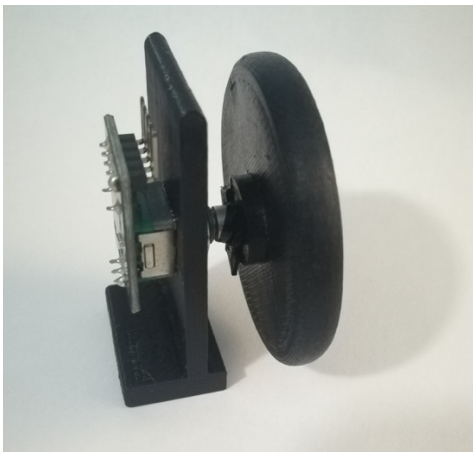


Figura IV.25. Mecanismo de acoplamiento para ajuste de trim.

IV.7.2 Diseño electrónico

Para este módulo se ha recurrido al uso de tres potenciómetros lineales y un codificador con movimiento circular, que se conectan al conector J2 de la placa diseñada (*Figura IV.26*).

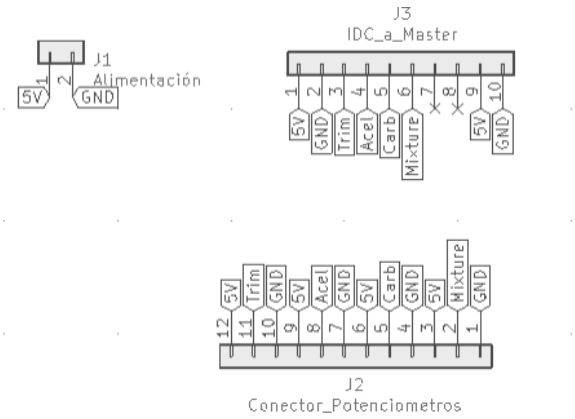


Figura IV.26. Esquema de conectores para los potenciómetros.

El *maestro* recibe los valores de las tensiones y procede a escribir un mensaje en el puerto *serial* por cada variación de 5%, que son los incrementos que puede

[24] ***Trim***: Es un sistema de compensación de morro. Permite liberar al piloto de tener que ejercer una presión constante en el yoke al tomar un determinado valor de inclinación como la posición de reposo.

realizar el simulador por cada pulsación de una tecla, asignándola a una combinación de teclado como podría ser *'Ctrl + A'* [37].

Ésta es escrita por el *software* intérprete cada vez que recibe por puerto *serial* el mensaje `MezclaUp`. Por tanto, el método de la clase `Gases` desarrollado para escribir estas variaciones queda de la siguiente forma:

```
void modifica(){
  actualiza(); %Se toma el nuevo valor de la entrada analogica
  _num_iteraciones = (_valor_actual_an-_valor_anterior_an)/5;
  if(_num_iteraciones > 0){
      for (i<0; i< _num_iteraciones ; i++){
          escribeSerial(_mensaje_incremento);
      } }
  else{
      for (i<0; i< -_num_iteraciones ; i++){
          escribeSerial(_mensaje_decremento);
      } } }
```

IV.8 Anemómetro (*Indicated AirSpeed*)

El IAS [38] es el indicador de velocidad del avión. Se trata, junto con los siguientes instrumentos que se explicarán, de uno de los 6 elementos básicos de todo avión tanto para navegación visual como instrumental.

Este dispositivo (*Figura IV.27*), mide la velocidad basándose en un sistema de tubos de pitot [35], de modo que ofrece la velocidad del aire respecto del avión sin aplicar ningún tipo de corrección. El de esta avioneta tiene un rango de operación de 0 a 200 nudos. No obstante, la velocidad máxima permitida de este avión está en torno a los 180 nudos.

Figura IV.27.Aspecto del anemómetro fabricado y el real.

IV.8.1 Diseño mecánico

Para el movimiento de la aguja de este instrumento, así como de los sucesivos, se ha decidido emplear servomotores del tipo SG-90 [16] explicados en el *Capítulo 2*. Debido a que la aguja de este instrumento debe ser capaz de realizar un giro de 360º y estos servos tienen como rotación máxima la mitad, se decidió diseñar un conjunto de engranajes para la transmisión del movimiento con una relación igual o superior a 2.

El engranaje primario (A, *Figura IV.28*), va acoplado de forma solidaria al servomotor, que se encuentra situado en un lateral respecto al eje de la aguja indicadora. Este engranaje consta de 24 dientes y una muesca para la colocación del servo. El engranaje secundario (B, *Figura IV.28*), unido solidariamente al eje de la aguja está compuesto por 9 dientes, de modo que la relación de transformación es de 2,67; permitiendo dar una vuelta completa con menos de 180º.

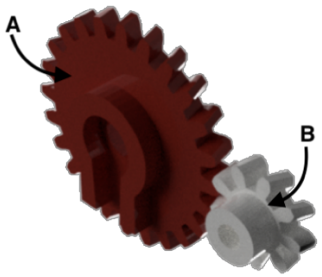


Figura IV.28. Mecanismo de transmisión por engranajes para servomotor.

Además de los dos engranajes anteriormente citados, se emplea: una base para sujetar el servo (A, *Figura IV.29*) y que sirve de eje para el engranaje primario, un separador (B, *Figura IV.29*) y un marco embellecedor (C, *Figura IV.29*) que se instaló por delante del panel. Estos componentes se observan en la siguiente vista explosionada del *gauge*.

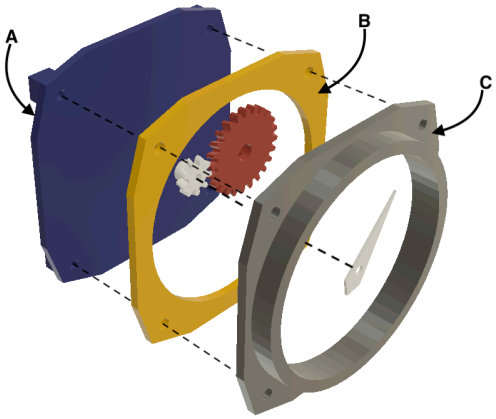


Figura IV.29.Disposición de los elementos que conforman el anemómetro.

IV.8.2 Diseño electrónico

Este indicador únicamente está compuesto por el servomotor antes indicado. Para su control se ha hecho uso de la librería `servo.h` propia de Arduino conectándolo a un pin con salida PWM y se han escalado los valores de velocidad (del rango de 0 a 200 nudos) a un giro de 340º de la aguja, que aplicando la relación de transformación se corresponde con:

$$\frac{340\ º}{2{,}67} = 127.3\ \approx 127º \quad \text{(IV. 1)}$$

Por tanto, empleando la función `map` escalamos el valor de la siguiente forma:

```
Angulo_servo = map(valor_IAS, 0,200,0,127);
```

IV.9 *Turn Coordinator*

El coordinador de giro consta de un indicador de viraje y un indicador de coordinación de viraje [38]. Permite controlar el avión sin referencias visuales, es decir, de forma instrumental indicando el ángulo y velocidad angular de giro hacia alguno de los lados.

Está compuesto por una aguja en forma de avión que de forma visual indica la inclinación de éste y una *bola* (*Figura IV.30*) que permite conocer si el movimiento del avión es coordinado, esto es, si su eje longitudinal está alineado con la dirección de giro deseada y realizada mediante el movimiento de los alerones. Para ello, se actúa también sobre el timón de cola de modo que el avión realice un movimiento suave.

Figura IV.30. Aspecto del coordinador de giro fabricado y real.

IV.9.1 Diseño mecánico

Durante el diseño de este instrumento no ha sido necesario emplear ningún mecanismo de transformación de movimiento dado que el avión bajo ninguna de las circunstancias puede realizar un giro que alcance 90º hacia ninguna dirección, por lo que con menos de 180º de giro total del servomotor es suficiente.

Por consiguiente, se ha dispuesto un servomotor SG-90 con su eje de giro alineado y unido al centro de la aguja en forma de avión (*Figura IV.31*). Para el movimiento de la *bola* se ha instalado un péndulo acoplado a un servomotor paralelo al anterior, de forma que su movimiento se transforme en un movimiento oscilante en el extremo donde se localiza la *bola*.

Figura IV.31. Ensamblaje del Turn-Coordinator.

Todos estos mecanismos se han fabricado en PLA en la impresora 3D del laboratorio.

IV.9.2 Diseño electrónico

Como en el dispositivo anterior, se ha ocupado un pin digital con salida PWM para cada servo y haciendo uso de la misma función `map` se han escalado los valores recibidos a ángulos de giro del servo.

La posición de 0 grados del avión, es decir la de alineación del eje longitudinal, se debe corresponder con la mitad del recorrido del servo, esto es con 90º. De este modo, las funciones para calcular los grados quedan de la siguiente forma:

```
angulo_bola = map (angulo_timon_cola,-25,25,30,150);
angulo_turn_coord = map (angulo_giro,-25,25,30,150);
```

IV.10 *Gauges* de doble aguja

Estos tres grupos de indicadores, por tratarse de indicadores cuya construcción es idéntica, se explicarán de forma conjunta. Se trata de avisadores que no están relacionados con la información de navegación, sino que son útiles para conocer el estado de los motores y la cabina.

El primero de estos (*Figura IV.32*) indica el nivel de combustible (en porcentaje) de los depósitos situados en cada una de las alas, el segundo la temperatura del aceite del motor y el nivel de presión de dicho circuito y el último, la presión de vacío de la cabina así como el consumo de corriente de los instrumentos.

Se trata de aparatos de un tamaño menor, unas 2 pulgadas aproximadamente.

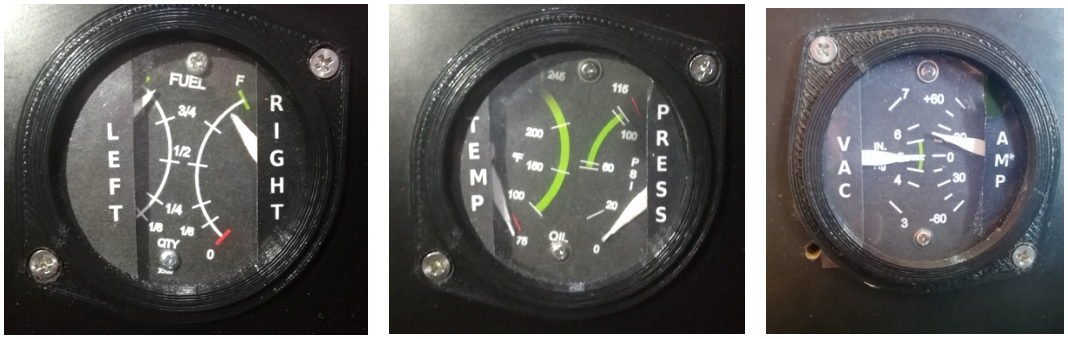


Figura IV.32. Aspecto de los gauges duales fabricados.

IV.10.1 Diseño mecánico

Para la fabricación de cada uno, se ha diseñado una base para sostener a dos servomotores (B, *Figura IV.33*), uno a cada lado, a los que se le acoplan directamente los ejes de las agujas (con los soportes mostrados en A, *Figura IV.33*). Nuevamente, como el recorrido de las mismas es de 120º, inferior a los 180º del servo, no se necesita un mecanismo de multiplicación por engranajes.

Se ha dispuesto un servo en posición vertical para la aguja de la derecha y el otro en posición invertida para la aguja de la izquierda, con el objetivo de que ambos puedan moverla correctamente.

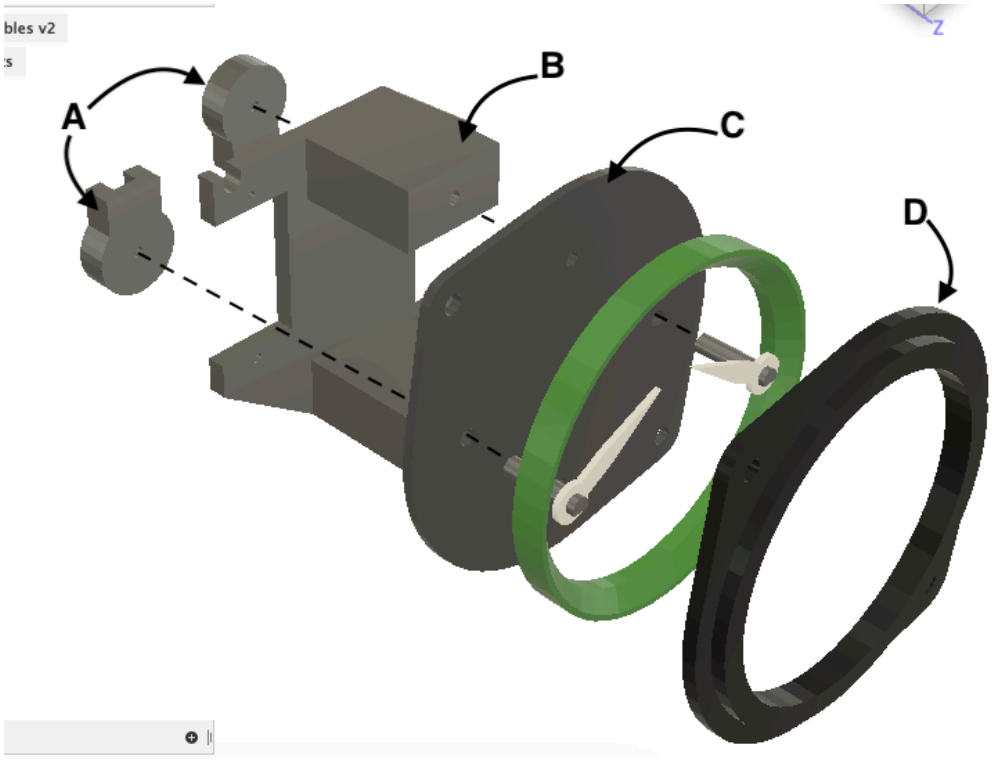


Figura IV.33. Esquema de ensamblaje de los gauges duales.

IV.10.2 Diseño electrónico

Cada servomotor emplea un pin digital del microcontrolador y su control se realiza igual que con los anteriores, aunque en este caso para el indicador situado a la izquierda, es decir, nivel de combustible del ala izquierda, temperatura de motor y nivel de vacío, se deben escalar los ángulos de forma contraria. Por ello, se corresponde el ángulo mayor del servo con el mínimo valor a indicar por el instrumento.

Para el caso por ejemplo del nivel de combustible el escalado quedaría de la siguiente forma:

```
angulo_servo_combust = map(nivel_combustible, 0,100, 150,30);
```

IV.11 Horizonte Artificial

Otro de los seis instrumentos básicos de navegación es el horizonte artificial (*Figura IV.34*). Permite a la tripulación conocer los ángulos de alabeo y cabeceo del avión, es decir, si el mismo está girando o si está ascendiendo o descendiendo, todo ello mostrado en un indicador móvil con dos grados de libertad en el movimiento [38]. Para facilitar su lectura, la mitad superior del indicador se encuentra pintada en color cielo y la mitad inferior en marrón, de esta forma la intersección de las dos marca el horizonte, sirviendo de guía para la posición horizontal del avión.

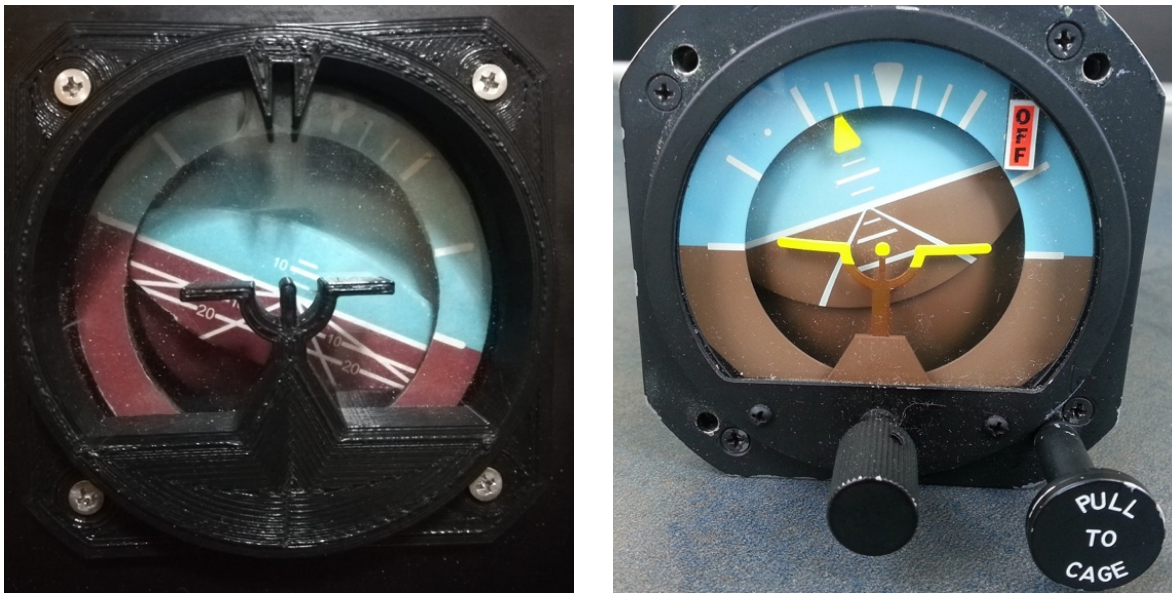


Figura IV.34. Horizonte artificial simulado y real.

IV.11.1 Diseño mecánico

Con el objetivo de dotar de esos dos grados de libertad se ha hecho uso de dos servomotores como los de los demás instrumentos.

Para la parte encargada de señalizar el movimiento del eje transversal del avión (el alabeo), se ha dispuesto uno en la parte trasera de la estructura diseñada (A en la *Figura IV.35*). A éste se le conectó una plataforma con su extremo en forma circular (B en la *Figura IV.35*). Esta plataforma lleva instalada el segundo servo, que mueve el óvalo (C en la *Figura IV.35*) simbolizando el cabeceo del avión. De forma fija se instala alrededor un aro (D, *Figura IV.35*) con las marcas de referencia de alineación del eje transversal.

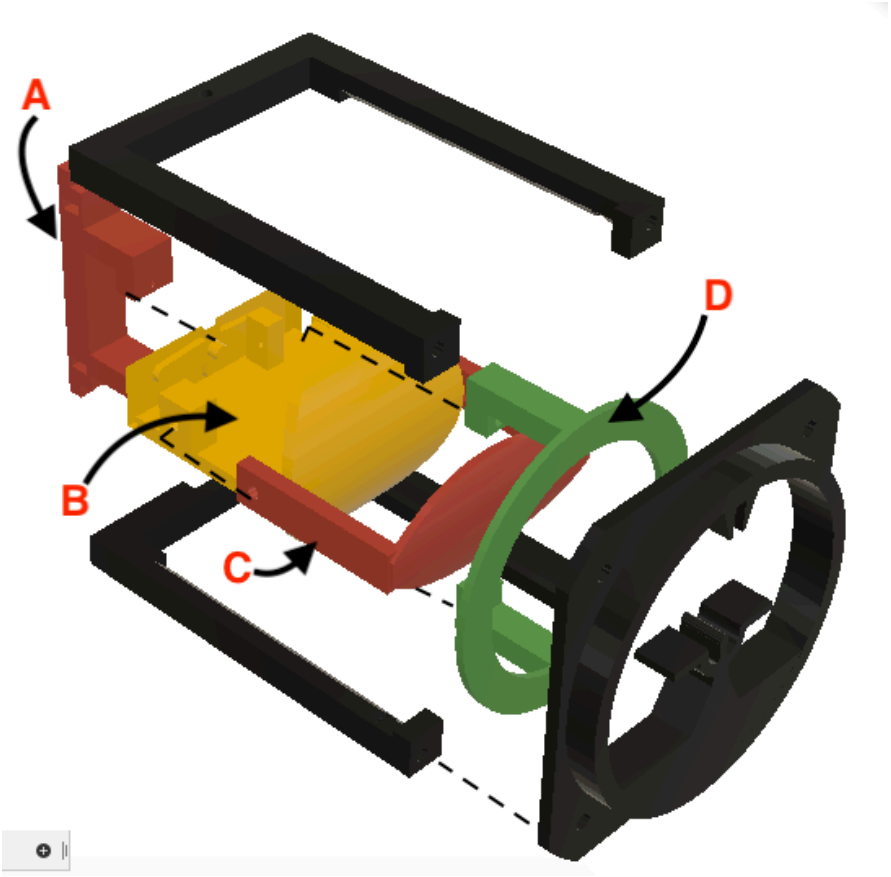


Figura IV.35. Disposición de las piezas del horizonte artificial.

IV.11.2 Diseño electrónico

Al estar constituido por servomotores su control es similar al descrito en los aparatos anteriores; solo cambia el escalado de los valores.

En las avionetas la máxima inclinación del morro es de aproximadamente 45º en ambos sentidos y de 30º en el giro (respecto al eje longitudinal del avión). Por tanto, se han dispuesto unos límites de 55º en ambos sentidos para el cabeceo (*pitch*) y 40º para el alabeo (*roll*) con el fin de tener cierto margen de libertad.

La posición de reposo de los servomotores se corresponde con el ángulo de 90º, por lo que los ángulos límite de giro de estos son de 35º, 145º para el *pitch* y de 50º y 130º para el *roll*, quedando las funciones de escalado de la siguiente forma:

```
angulo_ascenso = map(valor_pitch, -55, +55, 35, 145);
angulo_giro= map(valor_roll,-40, +40, 50, 130);
```

IV.12 Indicador de RPM y de velocidad vertical (variómetro)

Estos dos instrumentos, al presentar una idéntica implementación, se explican de forma conjunta.

Como sus nombres indican, uno de ellos es el encargado de mostrar las revoluciones del motor de la avioneta (A, *Figura IV.36*) y el otro indica la velocidad a la que se encuentra ascendiendo/descendiendo (en pies/minuto) (B, *Figura IV.36*). Dado que es un avión monomotor, solo se implementó un indicador de RPM. No obstante para cabinas de otros modelos ligeros como Baron 58 [36] o Cessna 421 [39] es suficiente con replicar este indicador y producir dos de estos, dado que sus rangos de operación son similares.

Figura IV.36. Instrumentos de RPM (A) y de velocidad vertical (B) fabricados.

Ambos instrumentos tienen un ángulo de giro mayor a los 180º. Por ello nuevamente se hace necesario implementar un mecanismo de transformación (*Figura IV.37*) con el fin de desmultiplicar el movimiento. Se recurrió al mismo tipo de engranajes diseñados para el IAS y se programó un algoritmo similar.

Figura IV.37. Despiece de los gauges de RPM y VS.

IV.13 Altímetro

El altímetro permite medir la altitud a la que se encuentra el avión. Su funcionamiento se basa en un cálculo de la presión que ejerce la columna de aire que se encuentra encima del aeroplano. En base al valor de esta presión es posible determinar la altura, por lo que se necesita precisar el valor de presión que hay en tierra para mediante diferencia poder determinar la altura [38]. Consta de un regulador que se simula con un *encoder* para introducir dicha información.

La altitud se muestra haciendo uso de tres agujas con un mecanismo similar a la hora mostrada por un reloj. La mayor de las agujas indica las decenas de miles de pies , la aguja mediana indica los centenares de pies y la aguja pequeña indica los miles de pies. De esta forma, en la imagen de la *Figura IV.38*, se puede observar como la altitud marcada es de menos de unos 2000 pies para el indicador de la izquierda (el fabricado) y 720 pies para el de la derecha.

Figura IV.38. Altímetro fabricado y altímetro real de una avioneta.

IV.13.1 Diseño mecánico

Se ha decidido emplear un motor paso a paso 28-BYJ-48 [12] por tener mayor par y ser capaz de mover una mayor masa que los servomotores SG-90 [16] empleados en los otros instrumentos.

En primer lugar, para conseguir un movimiento de tres agujas a diferentes velocidades, se ha decidido emplear dos conjuntos de engranajes concéntricos (*Figura IV.39*), tomando como referencia el trabajo realizado por Álvaro Alea en [40]. La aguja de las centenas de pies va unida de forma solidaria al movimiento propio del motor mediante un eje de 3 mm. A este eje, se acopla un engranaje de 9 dientes (Engranaje B en la *Figura IV.39*) que transmite el movimiento a otro engranaje auxiliar (C) con 28 dientes. Éste a su vez, está formado en su otra cara por un engranaje menor , de 9 dientes que se acopla con otro de 28 (D) al cual se coloca el eje hueco de 4 mm que mueve la aguja de los millares de pies.

Dado que la relación *9/28 = 0,32* se tiene:

$$1\ vuelta\ motor = 0{,}32\ vuelta\ engranaje\ B \cdot \frac{0{,}32\ vueltas\ C}{1 vuelta\ de\ B} \approx 0{,}1\ vueltas\ C \quad \text{(IV. 2)}$$

De esta forma cuando se haya dado una vuelta completa en el eje principal, se habrá dado una décima parte de vuelta en el mecanismo que transfiere este movimiento a la aguja de los millares de pies. Este mismo principio se usa con el último de los engranajes manteniendo nuevamente una relación de aproximadamente *1:10*, obteniendo por una vuelta completa de la aguja de los miles de pies, una décima parte de vuelta del indicador de decenas de millar, es decir:

$$1\ vuelta\ engranaje\ C \frac{0{,}32\ vueltas\ de\ D}{1 vuelta\ de\ C} \frac{0{,}32\ vueltas\ de\ E}{1\ vuelta\ de\ D} \approx 0{,}1\ vueltas\ E \quad \text{(IV. 3)}$$

Además, en un lateral del instrumento se ha colocado un eje (F, *Figura IV.39*) unido a un *encoder* que permite ajustar el valor de la presión de referencia. Este ajuste, afecta al disco graduado (G en la *Figura IV.39*), que ve modificada su posición mediante un pequeño engranaje de 9 dientes que transmite el movimiento desde el eje del *encoder* al disco.

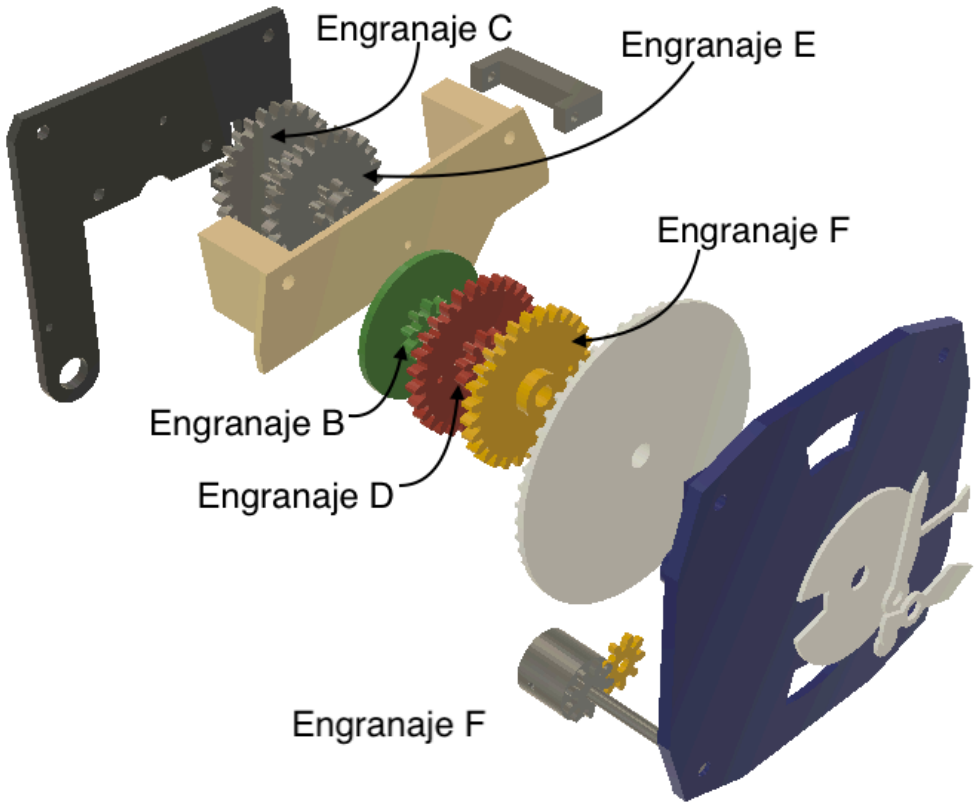


Figura IV.39. Esquema de ensamblaje del Altímetro fabricado.

Al tratarse de un motor paso a paso, se realiza un ajuste de cero pies mediante la alineación de un taladro realizado en cada uno de los engranajes. Cuando estos estén alineados y los agujeros se encuentren en la posición vertical, el sensor óptico se activará detectándose la posición de cero grados.

IV.13.2 Diseño electrónico

Para manipular este motor, se emplea un microcontrolador ATmega 328P como *esclavo* en el bus I^2C. Éste recibe el mensaje de la altitud y en base al valor de altitud anterior, calcula la diferencia. Se convierte dicho valor a pasos del motor, teniendo en cuenta que 1000 pies se corresponden con una vuelta del motor, es decir, con 1024 pasos. Dependiendo del signo de esta diferencia se decide el sentido de giro, definiendo el orden de excitación de las bobinas. De este modo, si por ejemplo, la altitud anterior es de 3960 pies y se recibe que la nueva altitud del avión es de 4230 pies se tiene:

$$(4230 - 3960) pies \cdot \frac{1024\ pasos}{1000\ pies} = 276\ \ pasos \quad \text{(IV. 4)}$$

Por tanto, para el movimiento antes descrito se necesitan dar 276 pasos. Esto permite tener incrementos de altitud de aproximadamente 1 pie por cada paso. Se trata de una resolución similar a la mínima altitud que recibe el *maestro* desde el PC, pues ésta es también de 1 pie.

La polarización de las bobinas siguiendo la secuencia de pasos completos se realiza con un bucle `for` desde `1` hasta el `número de pasos calculados/4`, y en la que en cada iteración se recorre la matriz de excitación de las bobinas según la secuencia, es decir:

```
for(i=0; i<pasos_calculados/4 ; i++){
  for(j=0;j<4; j++){
      digitalWrite(Pin1Stepper,valorPaso[j][0]);
      digitalWrite(Pin2Stepper,valorPaso[j][1]);
      digitalWrite(Pin3Stepper,valorPaso[j][2]);
      digitalWrite(Pin4Stepper,valorPaso[j][3]);
  }
}
```

En caso de tener que realizar un giro en sentido antihorario, únicamente cambia el orden de energización de las bobinas, siendo ahora [3]-[2]-[1]-[0].

Para ello se ha desarrollado una librería que almacena el valor de altitud actual, recibe el nuevo valor, calcula la diferencia en pasos y decide la secuencia de pasos a ejecutar. A modo simplificado el funcionamiento es el indicado en la *Figura IV.40*.

La matriz de excitación empleada es la siguiente:

```
int valorPaso[4][4]={
  {1, 0, 0, 0},
  {0, 1, 0, 0},
  {0, 0, 1, 0},
  {0, 0, 0, 1}
};
```

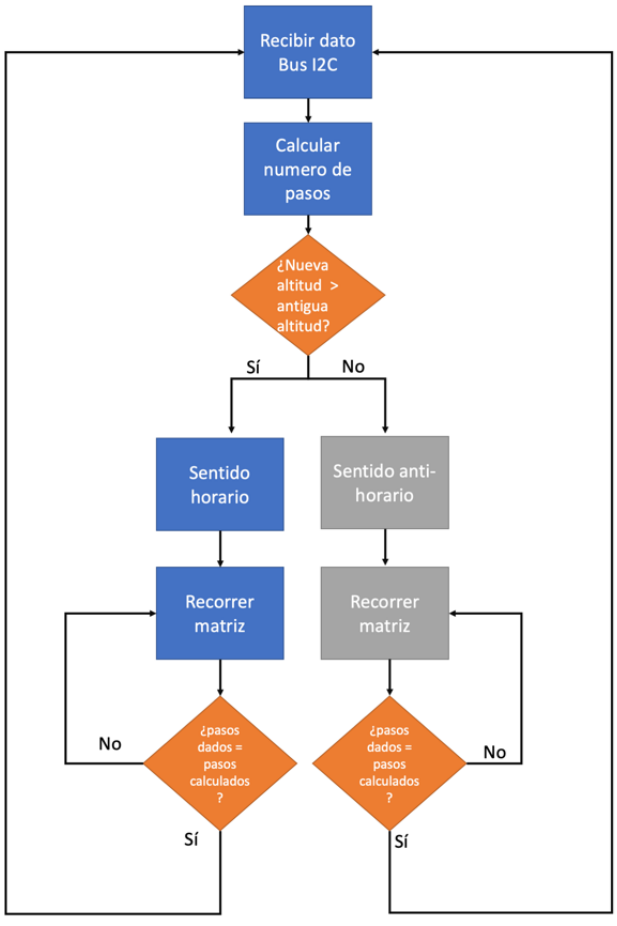


Figura IV.40. Algoritmo de control del altímetro.

Dicho motor se conecta a la placa de circuito impreso que alberga el microcontrolador del Arduino Nano (A1, *Figura IV.41*) y un puente en H (U3, *Figura IV.41*). Se ocupan las salidas *D12, D13, A6* y *A7* para polarizar los transistores y el pin *D8* para leer la entrada del sensor óptico de *homing*.

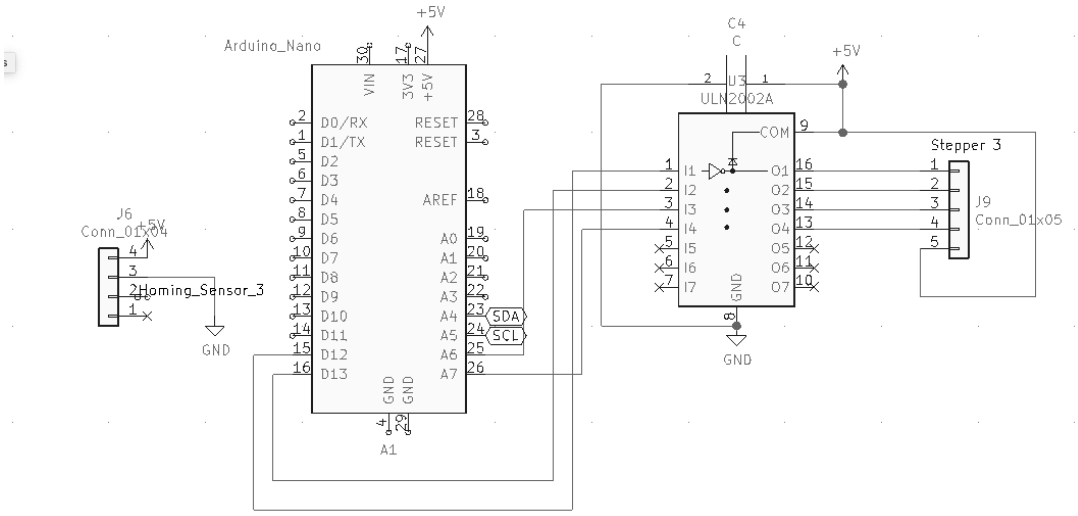


Figura IV.41. Esquema de conexión del puente en H.

IV.14 Indicador de rumbo o HDG

El indicador de rumbo (*Figura IV.42*) es un instrumento destinado a la navegación instrumental. Tiene como propósito indicar el giro direccional de la aeronave, esto es, la orientación de la misma respecto de los ejes cardinales.

Figura IV.42. Indicador de rumbo fabricado frente al real de la Cessna.

IV.14.1 Diseño mecánico

Se dispone de dos motores paso a paso: uno de ellos controla el movimiento del indicador de rumbo fijado en el piloto automático y el otro el valor de rumbo actual del avión.

El primero lleva unido de forma solidaria el *engranaje A* de la figura, que transmite el movimiento al *engranaje B* y con ello se consigue hacer girar el *needle*[25]. Este valor de rumbo se indica al piloto automático mediante un *encoder* situado en la esquina inferior derecha y unido mediante un eje a la perilla blanca (*Figura IV.43*).

El segundo de los motores se une al disco A mediante el eje A de la *Figura IV.43* y gracias a la acción de dos rodamientos situados entre la base, el engranaje y el disco, permite el movimiento de éste sin afectar el recorrido del *indicador A*.

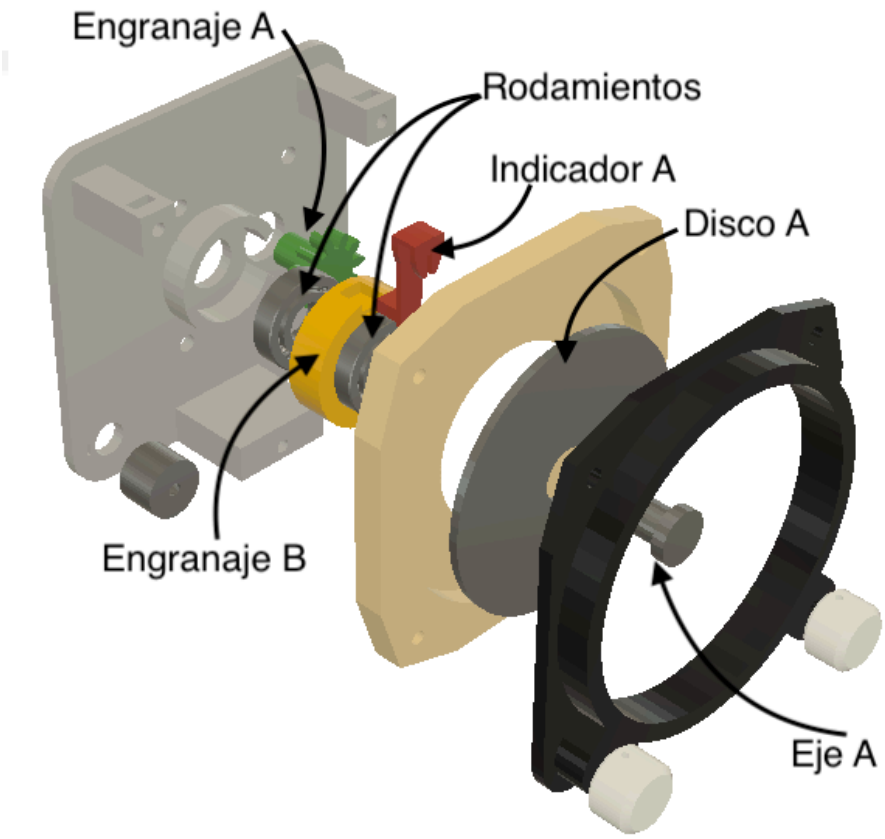


Figura IV.43. Esquema de montaje del HDG producido.

[25] ***Needle***: aguja indicadora.

IV.14.2 Diseño electrónico

Su control se realiza mediante el mismo *esclavo* conectado al bus I^2C que gestiona el altímetro. Por tanto, la función de excitación de las bobinas es la ya mencionada para dicho aparato. La dificultad radica en la sincronización entre la posición del *indicador A* y el *disco A*, puesto que se puede introducir un valor en el piloto automático, haciendo girar el indicador, a la vez que el avión está girando con disco también en movimiento.

Cuando el avión haya alcanzado la posición de referencia fijada, el *needle* debe quedar alineado con la vertical del instrumento y debe apuntar hacia el valor determinado en el piloto automático.

Para conseguirlo, se mueve el indicador de rumbo deseado a la posición que se introduzca en el piloto automático, y por cada movimiento del *eje A* hacia una dirección, se mueve la misma cantidad el *engranaje B* hacia el lado contrario. Dado que los dos engranajes *A* y *B*, se han diseñado con el mismo número de dientes para garantizar una relación 1:1 en la transmisión del movimiento, se tiene una resolución mínima (con movimientos de 4 en 4 pasos) de:

$$4\ pasos \cdot \frac{360\ º}{2048\ pasos} = 0{,}703\ grados/paso \qquad \text{(IV. 5)}$$

Que es mayor que la resolución mínima de 1 grado del dato recibido por el simulador, por lo que se demuestra que es suficiente con el control mediante pasos completos. Como el movimiento se realiza de 4 en 4 pasos, y se desea calcular el número de iteraciones del bucle para conseguir el movimiento, se calcula la inversa del resultado de la ecuación anterior, obteniendo que se deben dar 1,42 conjuntos de 4 pasos para un movimiento de 1 grado. Como esto no es posible, se redondea en el último momento el número de iteraciones del bucle.

En primer lugar, una vez obtenido el valor nuevo del rumbo, se procede a mover el *needle* y el disco graduado, como se indica en el siguiente fragmento de código perteneciente al método `mueveHDG` de la clase `HDG` diseñada.

```
num_pasos_HDG = long((HDG_nuevo_leido - HDG_actual)·1.42);
if( num_pasos_HDG > 0){
  for(i=0; i< num_pasos_HDG ; i++){
      for(j=0; j < 4 ; j++){
          digitalWrite(Pin1StepperHDG,valorPaso[j][0]);
          digitalWrite(Pin2StepperHDG,valorPaso[j][1]);
          digitalWrite(Pin3StepperHDG,valorPaso[j][2]);
          digitalWrite(Pin4StepperHDG,valorPaso[j][3]);
          //Movimiento needle. Como el motor se situa en direccion con
         //traria, se excitan las bobinas al revés.
```

```
            digitalWrite(Pin4StepperNeedle,valorPaso[j][3]);
            digitalWrite(Pin3StepperNeedle,valorPaso[j][2]);
            digitalWrite(Pin2StepperNeedle,valorPaso[j][1]);
            digitalWrite(Pin1StepperNeedle,valorPaso[j][0]);
      }  }  }
else{
// Secuencia de dirección contraria  }
```

Una vez se ha realizado dicho movimiento, se procede a actualizar el valor de la posición de rumbo actual y a leer el nuevo valor de rumbo introducido en el piloto automático para indicarlo mediante el *needle*. Para ello, el procedimiento es igual pero se debe comprobar en este caso la diferencia entre el rumbo de consigna del piloto automático y el rumbo actual, es decir.

```
HDG_actual = HDG_nuevo; //Se actualiza el valor
num_pasos_needle= long((HDG_AP - HDG_actual) · 1.42);
if(pasos > 0){
  for(i=0; i<num_pasos_needle; i++){
      for(j=0; j<4; j++){
          //Se excitan las bobinas
      }  }  }
else{
      //Se realiza pero en sentido contrario, cambiando el orden
//de excitación  }
```

Tanto éste como el instrumento anterior son controlados por el *Esclavo número I* del diagrama general. Su aspecto se muestra en la *Figura IV.44*.

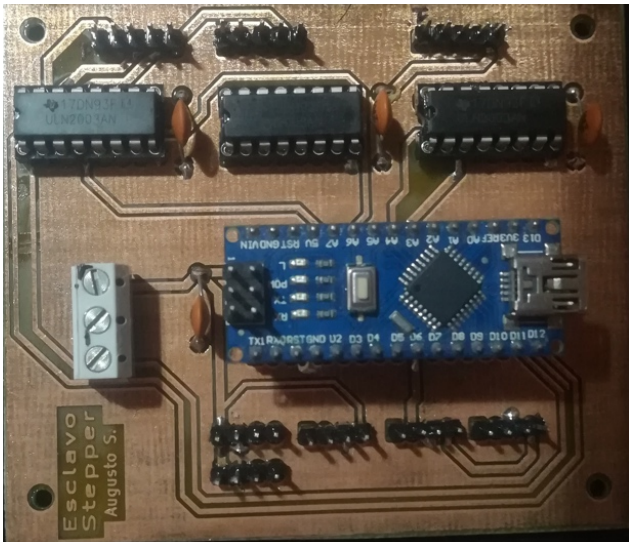


Figura IV.44. PCB del Esclavo que gestiona los steppers.

Capítulo V. Dispositivos *Esclavos* y *Maestro*

En el presente capítulo se describe el funcionamiento del dispositivo *maestro*, su comunicación con el ordenador mediante *serial* y con los dispositivos *esclavos*, tanto mediante I^2C como *SPI*, así como la recepción e interpretación de los mensajes por parte de éstos.

V.1 Mensajes enviados por el *software Link2FS*

Para la recepción de los datos del *software* FSX, se recurrió (como se mencionó en el *Capítulo I*) al programa *Link2FS*. Éste envía los parámetros mediante comunicación *serial* empleando paquetes de tamaño variable, en los que los dos primeros caracteres sirven para identificar el contenido. Cada caracter ASCII se envía como un byte.

Los mensajes carecen de información acerca de la longitud del mismo y su finalización. Por ello, se ha hecho necesario que el *software* lea un número fijado de caracteres que es tomado según el indicador obtenido en la identificación del mensaje. Este parámetro se pasa a la función `leeSerial()`. Así, se puede almacenar en cada variable el contenido correspondiente.

A continuación se describen los mensajes que se envían hacia el microcontrolador, explicando el significado de su contenido. Para ello se emplea una tabla en la que se indica en primer lugar los dos primeros caracteres correspondientes al identificador o cabecera, y posteriormente el formato y tamaño teniendo en cuenta la longitud de la cabecera.

V.1.1 Mensajes del módulo de comunicaciones

Para este módulo es necesario conocer las frecuencias de radio sintonizada, en espera, así como las del vuelo guiado y valor del transpondedor como se describe en la *Tabla V.1.*

Tabla V.1. Mensajes enviados para el módulo de radio.

1er Car. ID	2º Car. ID	Mensaje	Longitud	Significado
=	*A*	*abc.def*	9	Frecuencia de radio principal sintonizada en MHz con tres decimales.
=	*B*	*abc.def*	9	Frecuencia de radio en espera, enviada en MHz con tres cifras decimales.
=	*E*	*abc.de*	8	Frecuencia de navegación principal sintonizada. Se envía con dos decimales únicamente.
=	*F*	*abc.de*	8	Frecuencia de navegación en espera.
=	*J*	*xyza*	6	Valor del transponder de la forma XYZA
=	*M*	x	3	Indica el estado del módulo COM1. Si está activo el mensaje es =M1; para el caso de desactivado se envía =M0.
=	*P*	x	3	Indica el estado de módulo NAV1. Su funcionamiento es igual al mensaje para estado de COM1.
=	*R*	x	3	Indica el estado booleano del DME de la misma forma que el mensaje encabezado por =M.
=	*S*	x	3	Permite indicar si ADF se encuentra activo o no. El contenido del mensaje, nuevamente es un 0 o 1 según su estado.

De este modo, si se tiene una frecuencia sintonizada de 129.647 MHz para el valor de radio en espera, y un código de *transponder* de valor 2451, el mensaje es:

=B129.647

=J2451

V.1.2 Mensajes del piloto automático

Se necesita conocer el valor de la altitud, velocidad y rumbo indicadas al piloto automático, así como el estado de las diferentes funciones (modo aproximación, mantenimiento de altitud activo, control de velocidad, etc.). Todos estos mensajes se detallan a continuación.

Tabla V.2. Mensajes enviados para el piloto automático.

1er Car. ID	2º Car. ID	Mensaje	Longitud	Significado
=	*b*	*cdefg*	7	Valor de altitud introducida en el piloto automático. Es la entrada de consigna para esta función.
=	*c*	*+abcd*	7	Valor de velocidad vertical. Dado que puede ser positivo o negativo según ascienda o descienda, se envía un carácter de signo para indicarlo.
=	*d*	*efg*	5	Ángulo de rumbo seleccionado. Es el valor tomado como referencia para el giro del avión y se transmite al *esclavo* que controla el *gauge* HDG.
=	*a*	X	3	Según alterne su valor a 1 o 0 se indica si el piloto automático está encendido y activo o no, respectivamente.
=	*i*	X	3	Su valor booleano indica si se encuentra activa la selección de rumbo fijada. Con esta opción el avión girará de forma autónoma hacia la dirección indicada por el mensaje con cabecera =d.
=	*k*	X	3	Controlador de altitud activo o no. Su mensaje es booleano.
=	*m*	X	3	Permite conocer si el modo de aproximación de la aeronave está activado.
=	*o*	X	3	Indica el estado del control NAV, su activación (mensaje a 1) significa que el avión se podrá guiar por las balizas sintonizadas mediante el módulo de comunicaciones.
=	*s*	X	3	Indica si el control de velocidad está activo. En caso de recibir un 1 el avión mantendrá la velocidad fijada por el piloto mientras sea posible.

El microcontrolador recibe dichos mensajes y en caso de detectar una variación con respecto al anterior mensaje que tenía almacenado, lo modifica. Para los valores booleanos, se escribe en el bit correspondiente del registro creado para ser enviado nuevamente por el bus I^2C hacia el expansor instalado en el módulo correspondiente (comentado en el *Capítulo IV*). En el caso de los valores enteros

recibidos, los muestra en la pantalla LCD dispuesta y el mensaje del valor de rumbo indicado además, se envía mediante el bus serie.

Así, por ejemplo si el piloto automático y el control de altitud están activos con una altura fijada de 2500 pies y un descenso de 200 pies/minutos, se reciben los siguientes mensajes, respectivamente:

$$=a1$$

$$=k1$$

$$=b02500$$

$$=c\text{-}0200$$

V.1.3 Mensajes del panel de avisos (Warning)

Este modelo de avioneta cuenta con un panel de avisos muy sencillo, como se comentó en el *Capítulo IV.6*. Las advertencias (combustible, presión, voltaje, etc.) se toman leyendo los mensajes *seriales* que cuentan con información *booleana*.

En la pestaña de configuración de *Link2FS* se ajusta el umbral de activación de los mensajes, en la que además se especifica la cabecera de los mismos, como se muestra en la *Figura V.1*.

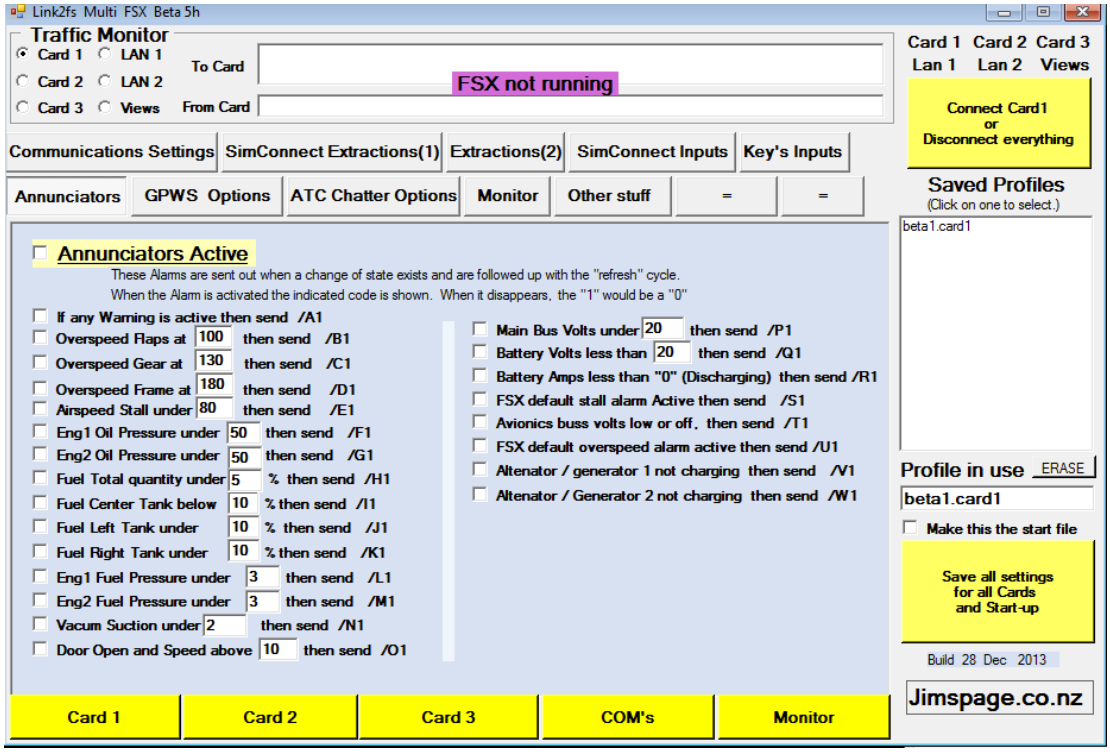


Figura V.1. Captura de pantalla del Link2FS para el ajuste de los avisos.

Los mensajes a leer se detallan en la siguiente tabla.

Tabla V.3. Mensajes enviados para el panel de advertencias.

1er Car. ID	2º Car. ID	Mensaje	Longitud	Significado
/	*F*	X	3	Aviso de advertencia de presión de aceite de motor demasiado baja. Se envía un 1 cuando el valor desciende de 20 PSI.
/	*H*	X	3	Nivel de carburante total inferior al 10%. Se envía cuando ambos tanques tienen una cantidad inferior a la décima parte de su capacidad. En ese caso el mensaje es /H1.
/	*K*	X	3	Nivel de carburante en el depósito del ala derecha inferior al 5%.
/	*J*	X	3	Si su valor es 1, el nivel de carburante del depósito del ala derecha es inferior al 5%.
/	*N*	X	3	Mensaje con valor booleano para indicar un nivel de vacío en la cabina demasiado bajo.
/	*P*	X	3	Indica si la tensión de la batería ha bajado de 19 V.

Toda esta información, tal y como se explicó en el *Capítulo IV*, es enviada desde el *maestro* hacía el expansor con dirección *0x39* para que éste realice el multiplexado de los leds indicadores.

V.1.4 Mensajes empleados por los instrumentos analógicos

Al igual que el resto de paneles, para cada uno de los *gauges* se hizo necesario obtener la información que permitiese el movimiento de la aguja hacia el valor a indicar. Todos éstos, al tratarse de instrumentos con un movimiento continuo, recibían mensajes con información en formato de numérico (aunque el mensaje *serial* se mandara en formato de caracter ASCII con un byte para cada cifra). Por ello, una vez leído cada uno de los mensajes, se procedió a eliminar los dos bytes del encabezado y a convertir a formato numérico la cadena de caracteres. Este procedimiento, se explica con detalle en el *Apartado V.2* del presente capítulo.

En la *Tabla V.4* se detallan los mensajes recibidos y como son empleados por estos indicadores.

Tabla V.4. Mensajes enviados para los gauges.

1er ID	2º ID	Mensaje	Instrumento	Significado
<	*P*	*abc*	Anemómetro	Valor de velocidad aérea en nudos con una resolución de 1 nudo.
<	*R*	*±abc.d*	Coordinador de giro y horizonte	Indica el valor de alabeo del avión con una cifra decimal. Se envía el signo y el punto de separación dentro del mensaje.
<	*X*	*abc*	Indicador de combustible izquierdo	Se envía el porcentaje de carburante en el depósito izquierdo con resolución de un 1%.
<	*Z*	*abc*	Indicador de combustible derecho	Porcentaje de carburante del depósito derecho.
¿	*M*	*±def*	Temperatura de gases	Se recibe el valor de la temperatura (en ºC) con tres cifras enteras y sin decimales.
<	*t*	*ab*	Presión del motor.	Se recibe la presión en PSI con dos cifras enteras y sin decimales.
<	*x*	*ab*	Voltímetro	Tensión de la batería.
¿	*J*	*±abc*	Amperímetro	Corriente (en Amperios) aportada por el alternador del motor o consumida por la batería. El signo permite diferenciar si es aportada o consumida.
<	*Q*	*±abc.d*	Horizonte artificial	Valor de cabeceo del avión con una cifra decimal separada por punto y su signo según el avión se encuentre ascendiendo o descendiendo.
<	*T*	*abcde*	Tacómetro	Se reciben las RPM del motor con 5 cifras sin decimales.
<	*L*	*±abcde*	Variómetro	Velocidad vertical en pies/minuto con 5 cifras, sin decimales.
<	*D*	*abcde*	Altímetro	Valor de altitud del avión en pies, con 5 cifras y sin decimales.
<	*J*	*abc*	HDG	Orientación del avión en grados, es decir, el valor de la brújula magnética con 3 cifras sin decimales.

Por ejemplo, si se recibe el mensaje *<T07320* se sabe que es la información referente a las revoluciones del motor, por tanto se procede a eliminar la cabecera y almacenarla en una variable de forma que se obtenga únicamente en valor 7320.

V.2 Funcionamiento del *maestro*

El *software* del microcontrolador ATmega 2560 se puede dividir en dos bloques:

- Bloque de **lectura de datos** de puerto *serial*, en el que se leen caracter a caracter todos los mensajes y se comparan con los mensajes definidos en las tablas anteriores para clasificar la información recibida.
- Bloque de **escritura de datos**, mediante el cual tras leer el estado de los controles reales de la cabina y los dispositivos *esclavos*, se procede a escribir los mensajes por *serial* según las tablas que se detallarán en el *Apartado V.2.3.*

Además, se encarga de comunicar a los dispositivos eslavos la información necesaria para mostrar los avisos pertinentes (ya sea luces de *warning*, frecuencias o datos de navegación).

V.2.1 Conexiones diseñadas

Para agrupar todas las conexiones entrantes y salientes al microcontrolador ATmega 2560 se ha optado por diseñar y fabricar una placa de circuito a la que se le conecta un microcontrolador ATmega 2560 *embebido*[26] [41] que contiene un convertidor de USB-TTL [42] [43] y en la que se han dispuesto los multiplexores para los instrumentos, los conectores IDC y *pinHeaders* explicados en el *Capítulo IV*.

Dicha PCB realizada a doble cara presenta el aspecto que se muestra en la *Figura V.2* y *Figura V.3*.

[26] ***Embebido:*** ATmega 2560 embebido es un microcontrolador compatible con Arduino que monta el chip del Arduino Mega pero con un tamaño reducido.

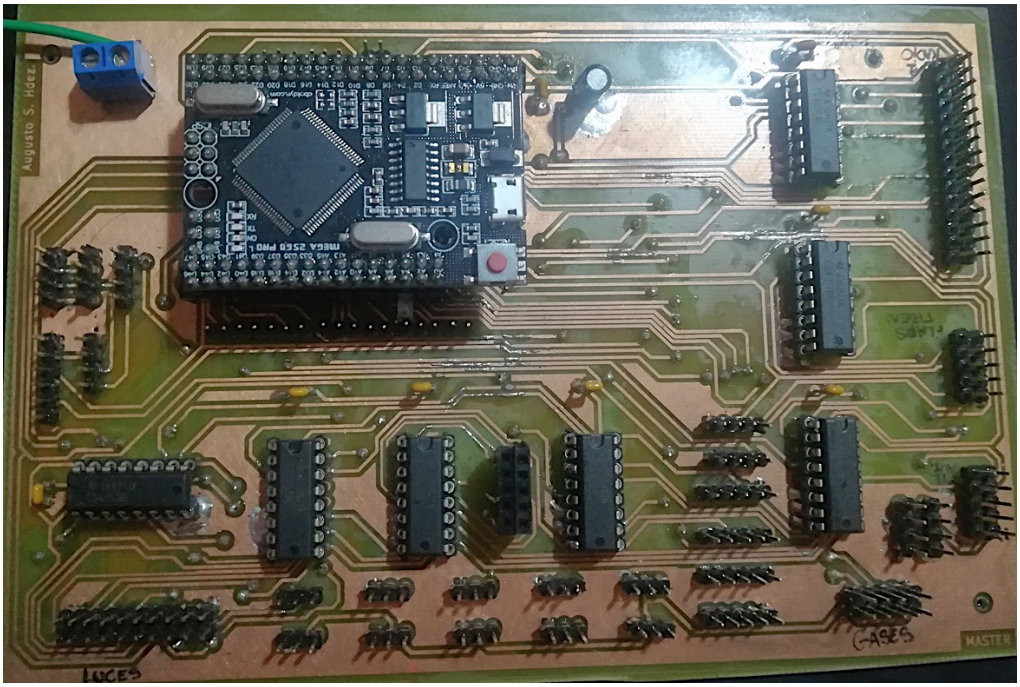


Figura V.2. PCB implementada para el dispositivo maestro (cara superior).

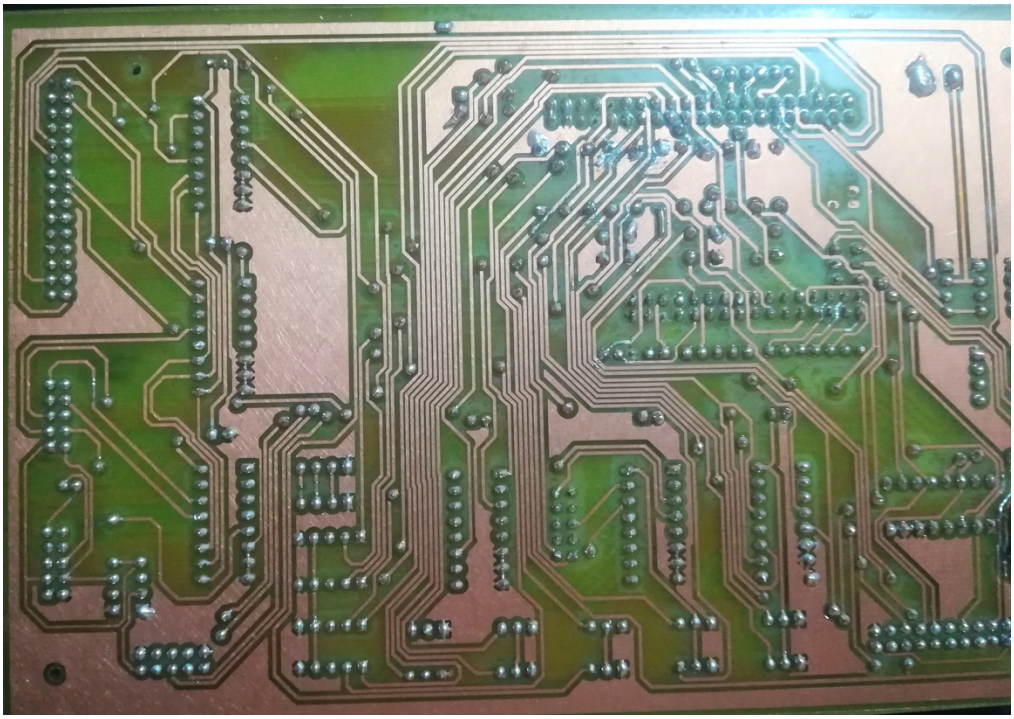


Figura V.3. PCB implementada para el dispositivo maestro (cara inferior).

V.2.2 Recepción de los datos desde el simulador

El microcontrolador empleado como *maestro* cuenta con un *buffer*[27] de 64 bytes de tipo LIFO[28] que se ha ampliado modificando la librería `HardwareSerial.h` [44] hasta los 128 bytes, por ello para la lectura de los mensajes se consulta el mismo mediante la función `Serial.available()` [45] en busca de caracteres sin leer. Mientras esta función devuelva un valor verdadero se leen los dos primeros caracteres uno a uno para clasificar el mensaje (comparándolo con las *Tablas V.1-V.4*) y posteriormente se lee como un único *string* el resto hasta la finalización de la línea.

Para emplear este algoritmo se debe suponer que los mensajes siempre se reciben en su totalidad, sin daños y con los caracteres de forma ordenada. Esto es fácilmente asumible gracias al empleo del *buffer* y a que el único programa que escribe por el puerto *serial* en dirección al microcontrolador es el *Link2FS*.

De forma general, en la *Figura V.4* se muestra el diagrama de bloques del algoritmo implementado para la lectura y clasificación de los mensajes.

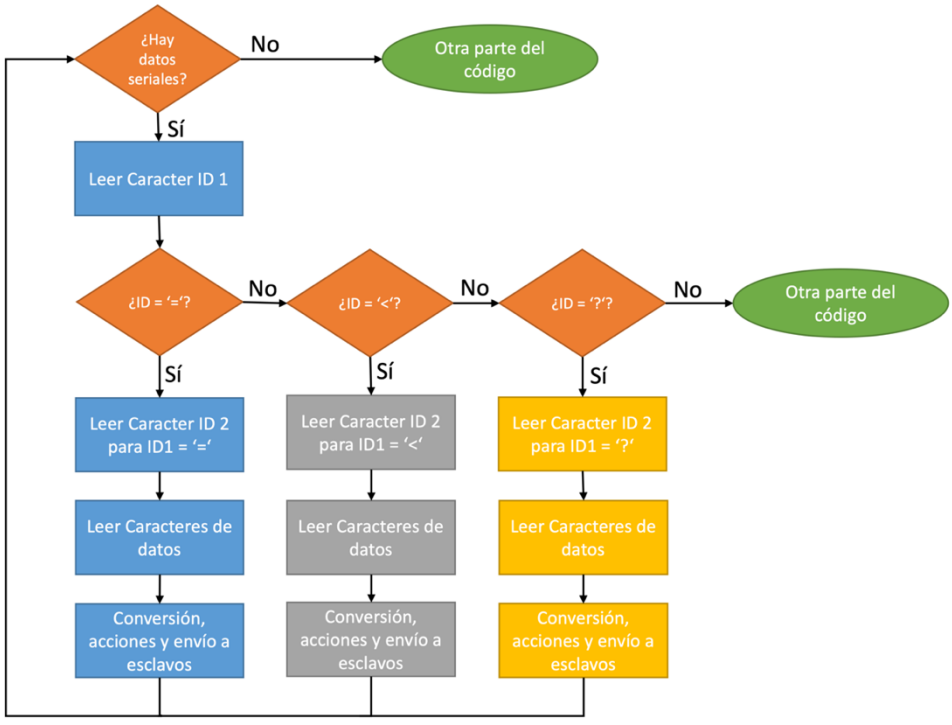


Figura V.4. Diagrama de bloques del algoritmo de lectura del puerto serial.

[27] ***Buffer***: Memoria de almacenamiento temporal que permite transferir información entre unidades funcionales con características de transferencia diferentes.

[28] ***LIFO***: *Last Input First Output*, tipo de pila de datos en la que el ultimo dato almacenado es el primero en ser leído.

Esto se realiza en código *C++* mediante el empleo de un bucle `while` y un método de selección `Switch`. En cada uno de ellos se llama a la función `leeCaracter()` definida en la librería propia `Funciones.h` que devuelve el byte leído en formato ASCII siempre que haya información en el *buffer*.

Por tanto, el código de forma genérica para la lectura quedó de la siguiente manera:

```
if( Serial.available() > 0 ){
      LecturaIdentificador = leeCaracter();
      if (LecturaIdentificador == '=') {
        LecturaIdentificador = leeCaracter();
        switch (LecturaIdentificador) {
          case 'A':{
             //Lectura de datos como string
          }
         //Así con el resto de casos
         }
       }
}
```

V.2.3 Envío de datos al simulador

Link2FS además de enviar los mensajes hacia el microcontrolador es capaz de leer los que éste envía. De este modo la comunicación en la otra dirección se realiza empleando el mismo programa y un algoritmo muy similar que consulta antes de enviar si hay espacio en el *buffer* (mediante `Serial.availableForWrite()`) [45] y comienza a transmitir.

En este caso, primero se leen los estados de todas las entradas siguiendo el procedimiento detallado en el *Capítulo IV*, para posteriormente, enviar el mensaje adecuado a dicho estado. En primera instancia se planteó enviar la información de forma repetitiva en cada ciclo de ejecución. Se observó que esto no era óptimo, dado que los mensajes eran siempre recibidos desde el primer envío. Por tanto, se procedió a enviarlos únicamente cuando ocurriera un cambio en alguna de las entradas, actualizando así solo los valores necesarios. Esta solución carga de más memoria al microcontrolador *maestro*, dado que debe almacenar constantemente los estados de todos los controles, sin embargo, libera de carga al intérprete y reduce el número de mensajes del bus, disminuyendo las posibilidades de congestionarlo.

Los mensajes a enviar son muy similares a los que se recibían, pero con solo un caracter de identificación y a continuación el mensaje. Se detallarán durante este capítulo clasificándolos según el módulo del simulador al que pertenecen.

V.2.3.1 Envío de datos del módulo de comunicaciones

El panel de radio debe enviar las acciones de elevar o reducir las frecuencias según lo indicado con los *encoders* y las acciones de cambiar o activar dicha comunicación. De este modo los mensajes son los que se indican en la *Tabla V.5.*

Tabla V.5. Mensajes a enviar para el módulo de comunicaciones.

Mensaje	Significado
A14	Incrementar en 1 MHz la frecuencia NAV en espera.
A13	Decrementar en 1 MHz la frecuencia NAV en espera.
A16	Incrementar NAV en 25 kHz.
A15	Reducir frecuencia NAV en 25 KHz.
A18	Alternar entre frecuencias NAV.
A45	Conmutar el estado del panel COM1.
A48	Conmutar el encendido de la comunicación NAV.
A50	Conmutar el encendido de la función DME.
A52	Conmutar el encendido de ADF.
A01	Reducir en 1 MHz la frecuencia de la radio COM1.
A02	Incrementar en 1 MHz la frecuencia de COM1.
A03	Reducir en 25 kHz COM1.
A04	Incrementar en 25 kHz la frecuencia de COM1.

V.2.3.2 Envío de datos por parte del piloto automático

Para el control del piloto automático se debe conocer la altitud, rumbo, velocidad vertical de consigna y el estado de control de dichas funciones. Esto se resume en la siguiente tabla.

Tabla V.6. Mensajes enviados por el módulo de Piloto Automático.

Mensaje	Significado
A57	Reducir en 1º el rumbo fijado.
A58	Aumentar en 1º el rumbo fijado.
B01	Alternar entre piloto automático activo o inactivo.
B04	Conmutar el estado del control de rumbo.
B05	Conmutar el encendido del controlador de altitud.
B10	Conmutar el encendido del asistente de navegación.
B11	Incrementar altitud de consigna en 100 pies.
B12	Reducir altitud de consigna en 100 pies.
B13	Incrementar en 10 pies/min la velocidad vertical.
B14	Decrementar la velocidad vertical en 10 pies/min.
B99	Activar el modo aproximación.
BXX	Reducir en 1 mmHg la presión de referencia para la altitud.
BXX	Incrementar en 1 mmHg la presión de referencia.

Como se emplea únicamente un mensaje para alternar, basta con enviar el código cuando se detecta el flanco de subida y bajada de la entrada del interruptor. Estos mensajes no pueden enviarse más de una vez dado que provocarían un cambio accidental del estado del instrumento. Para ello, se crea en la librería propia `Switch.h` un método `void conmuta(String cadena)`, que llama a otro método encargado de comparar el estado actual y anterior de interruptor (`bool cambiaEstado()`) y en caso de cambio enviar el *string* correspondiente por *serial*.

V.2.3.3 Envío de datos por parte del panel de interruptores

Este panel debe indicar el encendido de las luces, aviónica, generadores y arranque del motor. La mayoría están formados por mensajes diferentes para el encendido y el apagado (a diferencia de los paneles descritos en *V.2.3.1* y *V.2.3.2*), por lo que se empleó otra función muy similar pero que envía un mensaje distinto en cada caso. Se incluyó en la misma librería con el nombre `void biestado (String activo, String inactivo)`.

Los mensajes para ambos casos se detallan en la *Tabla V.7*, la *X* indica que el mensaje debe contener un 1 para activar la función y un 0 para desactivarla. Los mensajes del magneto son solo de activación.

Tabla V.7. Mensajes enviados por el panel de interruptores.

Mensaje		Significado
A43x		Interruptor de aviónica
C41x		Gestión de las luces de navegación.
C42x		Control de las luces Beacon.
C43x		Control de luces de tierra.
C44x		Gestión de las luces de taxi.
C45x		Control de las luces estroboscópicas.
E2x		Interruptor del alternador
f02x		Interruptor de la bomba de combustible
E01		Magneto del motor en posición de off
E02		Magneto en posición R
E03		Magneto en posición L
E04		Magneto en posición de ambos (L y R)
E05		Activación del motor de arranque
E18	***E17***	Control de batería, el primero activa el uso de la misma y el otro la desactiva

V.2.3.4 Envío de datos por parte del panel del tren de aterrizaje y Flaps

Para el panel de Flaps se tiene un mensaje para cada posición. Se envían haciendo uso del mensaje C17 seguido del porcentaje de apertura de los flaps. Como esta avioneta tiene 4 posibles posiciones (0º, 10º, 20º, 30º), se envían los porcentajes de 0, 33, 66 y 100% extendido. De este modo los mensajes a enviar son los siguientes (*Tabla V.8*).

Tabla V.8. Mensajes enviados para las posiciones de los Flaps.

Mensaje	Significado
C17000	Flaps totalmente retraídos
C17033	Flaps en posición de 10 grados
C17066	Flaps extendidos a 20 grados
C17100	Flaps totalmente extendidos

Para el tren de aterrizaje, en cambio, se envía el mensaje *C01* para indicar su retracción y *C02* para extenderse.

V.2.4 Comunicación con los microcontroladores esclavos

Los motores *steppers* son controlados por los microcontroladores *esclavos* ATmega 328P. Por ello se hace necesaria una comunicación entre el *maestro* y éstos. Dicha comunicación se realiza en el mismo bus I^2C de modo bidireccional.

En esta primera versión del proyecto, los instrumentos controlados son el altímetro y el indicador de rumbo (HDG). Por tanto, se deben enviar los valores de la altitud del avión, el rumbo actual y el rumbo fijado por el piloto automático (para mover los dos motores del HDG y el del altímetro). Dado que los mensajes a enviar solo pueden tener un byte de tamaño, se hace necesario fragmentar el contenido en dos bytes que es el tamaño del número entero en complemento a dos almacenado. Para ello una vez creado un elemento de la clase I^2C propia (`I2C.h`), se emplea un método que fragmenta el dato y lo envía en el siguiente orden:

1. **Identificador** del mensaje
2. **Byte más alto** del registro
3. **Byte más bajo** del registro

Quedando de la siguiente manera:

```
void enviaEsclavo (int  id, int  valor){
  byte MSB = valor >> 8;
  byte LSB = valor;
  Wire.beginTransmission( _direccion );
  Wire.write( id );
  Wire.write( MSB ); //Se manda 1º el byte más significativo
  Wire.write( LSB ); //Se manda el byte menos significativo
  Wire.endTransmission(); //Acaba transmision
}
```

De este modo el receptor solo necesita conocer los identificadores y proceder a reconstruir el mensaje mediante un nuevo desplazamiento de bits y una *operación or* a nivel de bit.

Los identificadores de los mensajes son los siguientes:

- ***ID = 1*** para el mensaje de altitud.
- ***ID = 2*** para el mensaje de rumbo actual.
- ***ID = 3*** para el valor de rumbo de consigna del piloto automático.

Si se lee por *serial* una altitud 2700 pies, que en complemento a dos es *0000101010001100* los bytes a emitir son:

- **ID:** 00000001
- **Byte alto:** 00001010
- **Byte bajo:** 10001100

Antes de enviar nuevamente otro valor, con el objetivo de no saturar al *esclavo* recibiendo constantemente mensajes, se espera la recepción de valor que indica la finalización del movimiento de los motores. Este mensaje tiene tamaño de un solo byte y sus posibles valores son 0 (para movimiento no finalizado) y 1 (para movimiento finalizado). Es decir, el *maestro* envía un mensaje de tres byte indicando una medida del parámetro de vuelo, por ejemplo, la altitud. El *esclavo* recibe el mensaje y comienza a realizar los movimientos pertinentes, indicándole al *maestro* que está ocupado aún con el movimiento (está enviando un valor de 0); una vez finaliza el movimiento, en la próxima pregunta por parte del *maestro*, se recibe la indicación de que está libre y se puede comenzar a enviar la nueva lectura de los parámetros de la avioneta.

V.3 Funcionamiento de los dispositivos *esclavos*

El *esclavo* encargado de controlar estos motores tiene como dirección en el bus *0x04* y mediante el método `onReceive()` [46] se llama a la función `recibeDato()` que procesa la información para generar el valor entero original. Esto genera una interrupción a nivel *software* dentro del programa, paralizando la ejecución del código principal.

Mientras quede información por leer en el bus se almacena siguiendo el mismo orden anterior: primero ID, luego byte alto y por último byte bajo. El byte más alto se desplaza 8 posiciones a la izquierda y se le aplica una *o lógica* con el byte bajo. De esta forma se genera un número de 16 bits correspondiente al entero original en complemento a 2.

Esta función en *C++* es la siguiente:

```
void recibeDato(int cuantos){
  byte MSB, LSB;
  while(1 < Wire.available()){
    id = Wire.read(); // recibir 1 byte de id
    MSB = Wire.read(); //Leer el MSB
    LSB = Wire.read(); //Leer el LSB
  }
  lecturaBus = (MSB << 8) | LSB; //Desplazar 8 posiciones el
     //byte primero, y le hago la O bit-a-bit al segundo byte
}
```

Retomando el ejemplo del *apartado V.2.4*. Se leen dichos mensajes, se procede a almacenar el identificador en la variable `int id`; se continúa leyendo el byte más alto y desplazándolo, para obtener:

0000101000000000

Numero al que se le aplica una *o lógica* con la siguiente lectura *10001100*, obteniendo finalmente el valor original de 2700 pies.

Para el envío del mensaje de estado (ocupado o libre) que se comentaba al final *apartado V.2.4*, con el que evitar una interrupción *software* por cada ciclo de ejecución del *maestro*, se recurre al método `onRequest()` de la clase `Wire.h` que llama a la función `enviaPeticion()` cuando se solicita un dato por parte de éste. De ese modo, esta última función manda el contenido de la variable `esclavo_ocupado` que puede tener los valores antes descritos de 1 o 0. Con esto se evita que el *maestro* esté enviando valores diferentes de altitud o rumbo durante cada ejecución sin haberse acabado el movimiento anterior.

El algoritmo completo de funcionamiento y comunicación se muestra mediante un diagrama de flujo en la *Figura V.5.* En éste se observa como una vez se recibe el dato se pasa al estado ocupado hasta haber finalizado el movimiento, comunicando dicho estado al *maestro* cuando es solicitado.

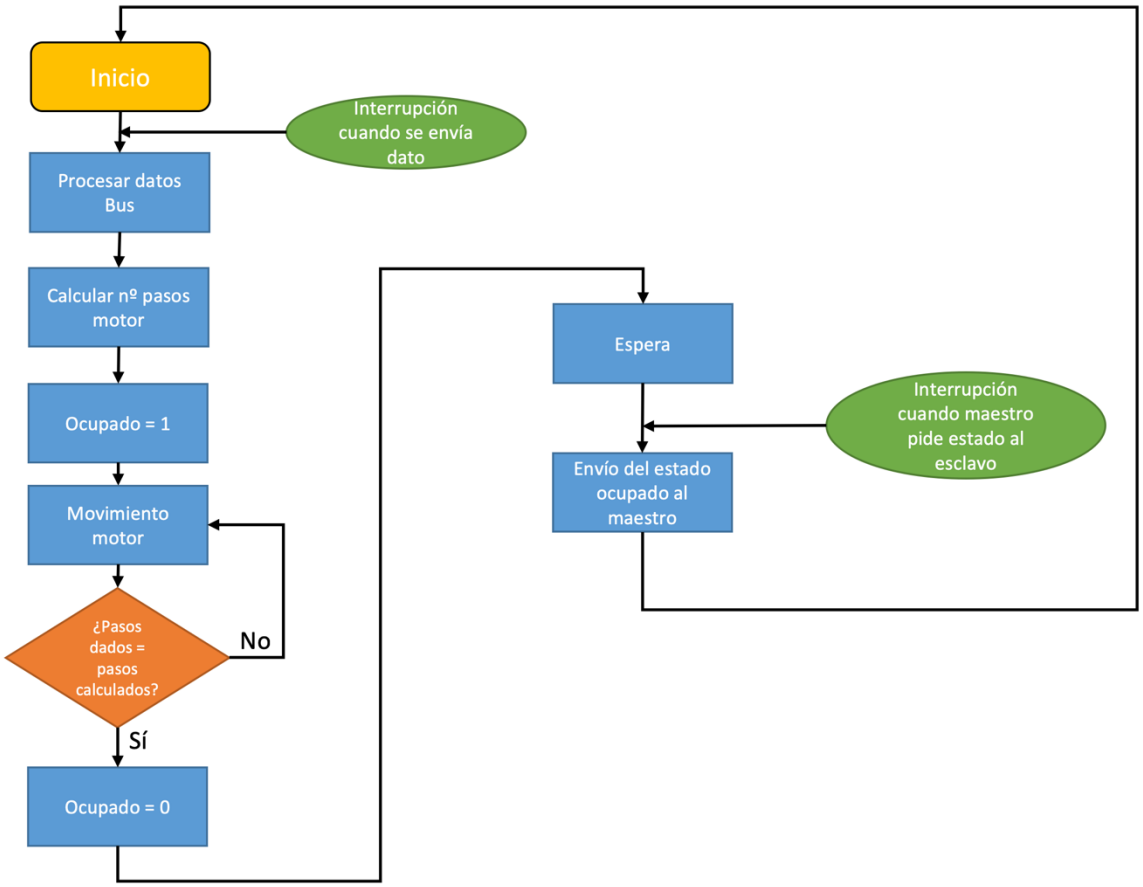


Figura V.5. Flujograma de los dispositivos esclavo basados en ATmega 328P.

Capítulo VI. Ensamblaje y acabado final

Mediante el presente capítulo se pretende mostrar cómo se ha realizado el proceso de montaje de la cabina, haciendo especial hincapié en la colocación de los componentes para respetar la distribución original de la aeronave.

VI.1 Diseño de la estructura

Para la colocación de todos los componentes se ha desarrollado un mueble con una forma similar al salpicadero de la avioneta. Se ha decidido emplear *madera DM*[29] de 10 mm para la realización de toda la estructura por ser un material sencillo de trabajar, ligero y con buen acabado una vez pintado.

Antes de comenzar con la fabricación artesanal, se procedió a diseñar mediante *software* el ensamblaje (*Figura VI.1*) y las dimensiones de las piezas. Una vez comprobado que todos los componentes pueden distribuirse de forma correcta, se procede al corte de la madera según dichas dimensiones.

Figura VI.1. Diseño de la cabina en Fusion 360.

[29] ***Madera DM***: Panel formado por residuos de madera unidos mediante algún tipo de resina a modo de aglutinante.

El hueco situado en la mitad inferior derecha permite la instalación de un ordenador de sobremesa que alberga los programas necesarios. Encima de éste se sitúan los módulos de control de tren de aterrizaje, flaps y gases.

En la región central, se sitúa el *yoke* comercial de Saitek (*A, Figura VI.2*) y la lámina de madera que alberga los relojes, controles de luces, panel de radio y de piloto automático. En el marco externo del mueble se ha decidido instalar una tira de leds de color blanco con los que iluminar los instrumentos. Su activación es controlada mediante el interruptor *panel light* situado a la derecha de la palanca del tren de aterrizaje.

Figura VI.2. Volante comercial de Saitek.

Debido a que el mueble se encontraba totalmente cerrado, se montó un ventilador de ordenador en la parte trasera derecha con el que garantizar un correcto flujo de aire para las PCB y la fuente de alimentación.

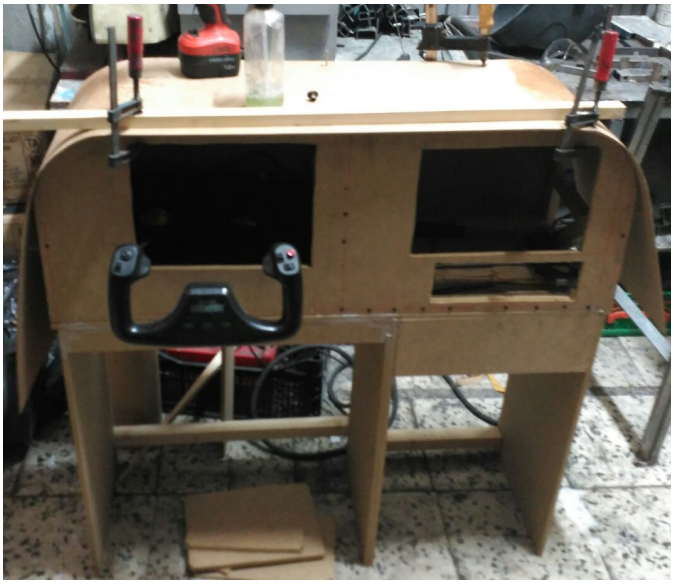


Figura VI.3. Salpicadero en proceso de fabricación.

En la *Figura VI.3* se observa la forma del salpicadero durante su construcción. Para la tapa superior se empleó madera de 3 mm de grosor, que se fue humedeciendo y doblando hasta darle la forma curva en ambos laterales. Una vez conformada la estructura y el salpicadero, se lacó en color negro.

VI.2 Disposición de los instrumentos

Para la elección del lugar de instalación de los instrumentos se tuvo muy en cuenta la configuración recomendada por cualquier fabricante de aviones: lo que se conoce como la *T básica* [47, p. 78]. Esta recomendación permite a los pilotos identificar rápidamente los instrumentos básicos en cualquier modelo de avión.

Comenzando por la esquina superior izquierda, se debe colocar el anemómetro, a su derecha (justo en el centro de la posición del piloto) el horizonte artificial. En la parte inferior de éste, el indicador de rumbo y a la derecha el altímetro (*Figura VI.4*).

Figura VI.4. Disposición de los instrumentos en forma de T básica.

En los huecos existentes, se dispusieron los demás elementos, de una forma lo más parecida a la cabina de referencia de la Cessna. Así, debajo del anemómetro se situó el coordinador de giro; a la izquierda los indicadores de motores: relojes de presión, temperatura, niveles de combustible, tensión e intensidad consumida. Por el otro lado, a la derecha del indicador de rumbo se instaló el variómetro y el tacómetro de su único motor. Dicha disposición se muestra a continuación.

Figura VI.5. Disposición de los instrumentos en el salpicadero fabricado.

En la mitad derecha del salpicadero se colocaron (citando desde la parte superior a la inferior): el conjunto de luces de advertencia, panel de radio, panel de piloto automático y justo a la altura del asiento del piloto los módulos de tren de aterrizaje y flaps (a la izquierda) así como los controles de gases (a la derecha de este último).

En *la Figura VI.6* se muestra la configuración adoptada, comparándola con la disposición real. Por razones de espacio, los paneles se instalaron más juntos y se añadieron botones extras (como el control de tren de aterrizaje) para hacerla utilizable con otros modelos de aeronaves.

Figura VI.6. Parte derecha del salpicadero simulado (A) comparada con el real (B).

El panel de radio es el que más diferencias de tamaño presenta. Esto se debe, como se indicó en el *Capítulo IV*, a la decisión de implementar solo las funciones básicas y únicamente un módulo de cada sistema de comunicaciones. De esta forma, la cabina solo cuenta con un conjunto emisor para la radio y el navegador y carece de sistemas ADF y DME. Si bien, con esto es suficiente para el entrenamiento de cualquier piloto, pues el funcionamiento de todos los instrumentos de comunicación es similar.

VI.3 Distribución de las PCB (*maestro* y *esclavo*s)

Debido a las dimensiones del salpicadero (108 cm de ancho, 141 cm de alto y 35 cm de largo), resultó crucial acercar el circuito impreso de cada módulo lo máximo posible al instrumento controlado para disminuir la longitud de los cables hacia los interruptores y pantallas. De esta forma, si se observa la cabina desde una vista superior, las PCBs se han dispuesto como se muestra en la siguiente figura.

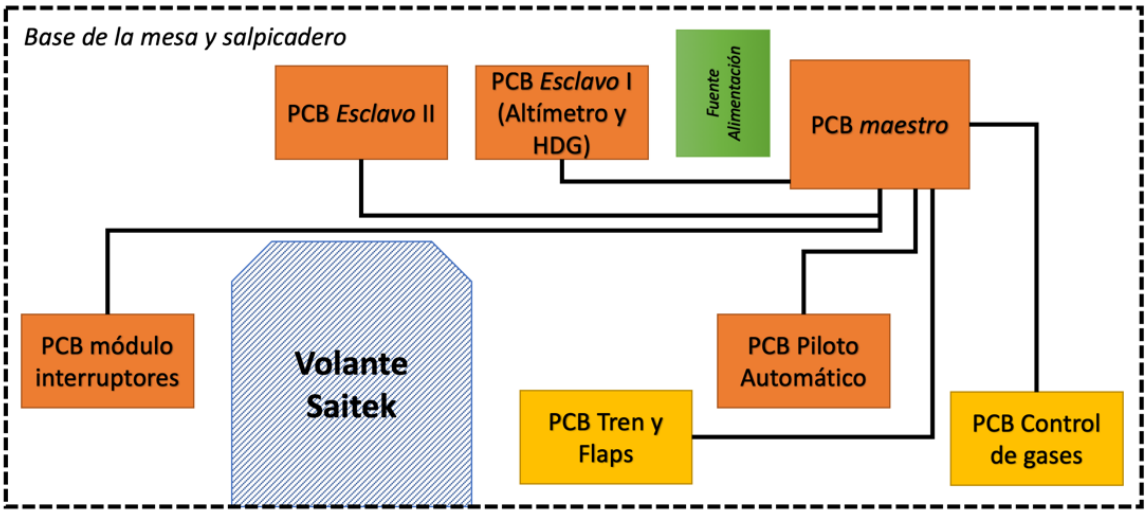


Figura VI.7. Distribución de las PCBs en el salpicadero producido.

Se emplearon dos niveles para su colocación. Las indicadas en naranja se han colocado a una altura de 65 cm en la base que sujeta el volante; las amarillas se sitúan 13 cm más abajo y permiten así estar en el mismo nivel que los potenciómetros lineales y los interruptores rotativos de tren y flaps. Además, el *esclavo* que controla las luces de *warning* así como el de las comunicaciones se han instalado en la cara trasera del frontal del salpicadero (*Figura VI.8*) en una disposición vertical.

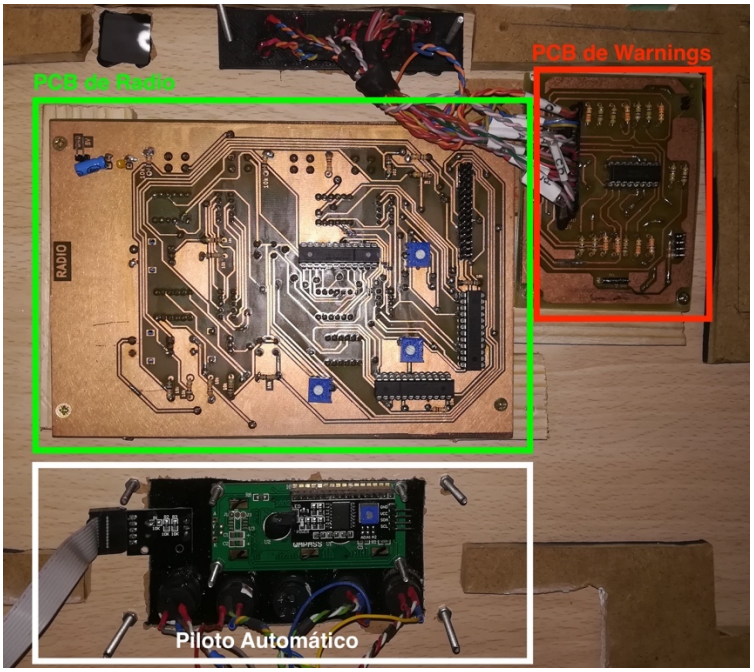
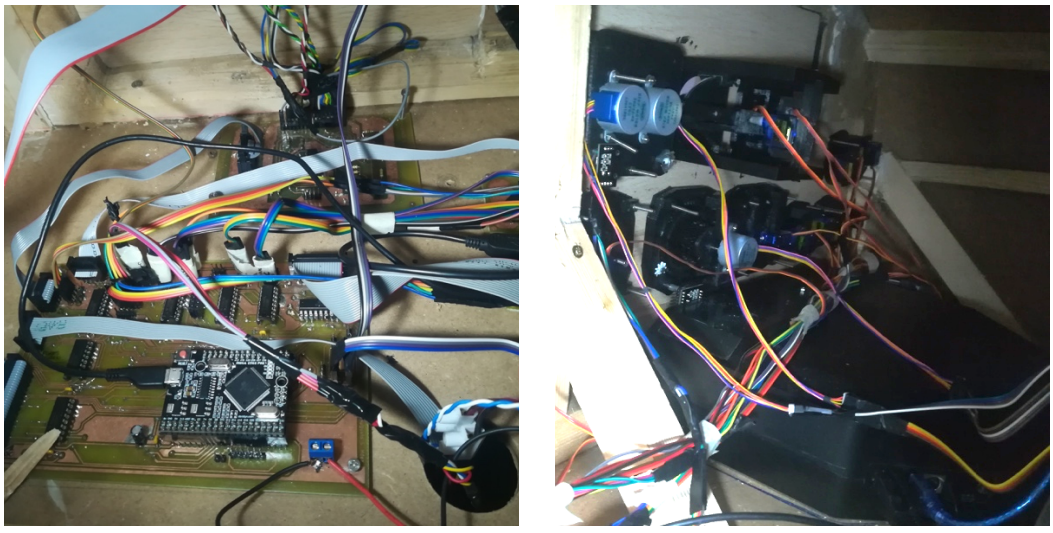


Figura VI.8. PCB situadas en posición vertical en el salpicadero.

Finalmente, el simulador terminado con todos sus instrumentos, el ordenador y la pantalla presenta el siguiente aspecto (*Figura VI.9- Figura VI.11*).

Figura VI.9. Aspecto de la cabina finalizada.

Figura VI.10. Vista de la cabina finalizada.



(A) *(B)*

Figura VI.11. Instalación de cables alrededor del maestro(A) y vista posterior del tablero de instrumentos (B).

Capítulo VII. Modos de uso y Resultados

Durante este último capítulo se explica de forma breve el modo de funcionamiento la cabina, detallando las diferentes funciones de los botones y *encoders* así como los ajustes a realizar en el *software* intérprete.

Para concluir se muestran los resultados finales de la cabina, demostrando que se ha probado de manera satisfactoria.

VII.1 Modos de uso de la cabina

Si bien el funcionamiento resulta claro para cualquier usuario que esté familiarizado con la conducción de aviones ligeros, se explicarán las diferencias que presenta con respecto a la cabina real, como puede ser la sustitución de los *encoders dobles*[30] por su homólogo de un solo eje con un pulsador incorporado o la forma de ajuste del transpondedor.

VII.1.1 Ajustes de las frecuencias de radio y transponder

El módulo de comunicaciones cuenta con tres codificadores rotativos explicados en el *Capítulo II* y *IV*.

El primero, correspondiente al *módulo COM* ajusta la frecuencia de la radio en espera. Su botón incorporado permite elegir si el ajuste se realiza en la escala de los KHz o los MHz. En la siguiente imagen se muestra el módulo con una frecuencia principal sintonizada de 134.47 MHz y un valor de transponder de 4056.

Figura VII.1. Módulo de radio realizado.

[30] ***Encoder doble:*** Se trata de un codificador rotativo formado por dos perillas que giran de forma independiente.

Por tanto, si se pulsa una vez, se editan los valores decimales en una escala de 100 KHz. Cada giro en sentido horario incrementa ésta en 50 KHz, y en el contrario, la reduce en igual cantidad. Si se pulsa su botón, se alterna la escala de edición. En este caso, un giro incrementa o decrementa la frecuencia en 1 MHz.

Una vez establecida la frecuencia deseada, se procede a activarla y sintonizarla en la radio mediante una pulsación del botón central. Tras esto, la frecuencia principal pasa a situarse en la derecha para ser editada, y la de edición para a ser sintonizada.

El segundo *encoder*, situado en la misma posición pero en la parte de Navegación (señalizado con NAV), funciona de forma idéntica aunque afectando a la frecuencia de la radio-ayuda a la navegación. Se alterna entre una y otra mediante el botón central.

A su izquierda se sitúa el editor de transpondedor. En este caso, cada pulsación del botón selecciona un dígito diferente para modificar, desde derecha a izquierda. Así pulsando una vez se selecciona el primer dígito y mediante el giro se aumenta el valor de éste entre 0 y 7. Si se supera el valor de 7 y se sigue girando se reinicia la cuenta en 0. Una vez modificado, se pulsa nuevamente para editar el segundo, tercero y cuarto. Al finalizar se pulsa nuevamente para fijar el valor y que sea visible por el ATC.

Los botones superiores activan, mediante una pulsación, las comunicaciones de radio, navegador, ADF y DME.

VII.1.2 Uso del piloto automático

Las funciones con las que cuenta esta avioneta son las de mantener la altitud, el rumbo y la velocidad de ascenso fijada. Cada una de ellas se activa mediante el interruptor correspondiente. Para que cualquiera de éstos surta efecto, el interruptor de piloto automático debe estar activado, es decir, se debe pulsar el primer botón de la izquierda (A, *Figura VII.2*) y observar que el piloto luminoso esté encendido.

Figura VII.2. Uso del control de rumbo en el P.A.

En caso de haber activado el control de rumbo mediante el botón HDG, se debe ajustar el ángulo al que se desea que se dirija el avión mediante la perilla situada en la parte derecha de la brújula magnética (A, *Figura VII.3*) incrementando un grado por cada giro horario y viceversa. Si el rumbo está activo, el *display* pertinente mostrará en la esquina superior derecha las siglas HDG y el piloto luminoso de su botón se encenderá (B y C, *Figura VII.2*). Cuando éste se encuentre desactivado, la dirección introducida mediante la perilla se muestra en dicha esquina de la pantalla.

Figura VII.3. Perilla de selección de rumbo.

El botón de activación del modo de aproximación (APR), funciona de forma similar. En este caso, cuando está activo, además del piloto luminoso se indica en la pantalla mediante las siglas APR impresas en la esquina inferior izquierda (*Figura VII.4*). Una vez es desactivado, tanto la luz como el mensaje en pantalla se eliminan.

Figura VII.4. Panel de AP con el modo de aproximación activado.

El *encoder* situado a la derecha de la pantalla (A, *Figura VII.5*) sirve para ajustar la altitud de referencia y la velocidad de ascenso deseada. Ambos ajustes solo

tienen efecto cuando el interruptor de ALT (B, *Figura VII.5*) se encuentra activo (con su correspondiente LED encendido). Dicho componente funciona de forma similar a lo explicado para la radio. Una pulsación activa el ajuste de altitud, modificándola mediante su giro; una segunda conmuta al ajuste de velocidad de ascenso. Si es pulsado nuevamente, se retorna al ajuste de altitud.

Los valores modificados se muestran en la pantalla, señalizados mediante las siglas ALT para la altitud y VS para la velocidad vertical (C y D, *Figura VII.5*).

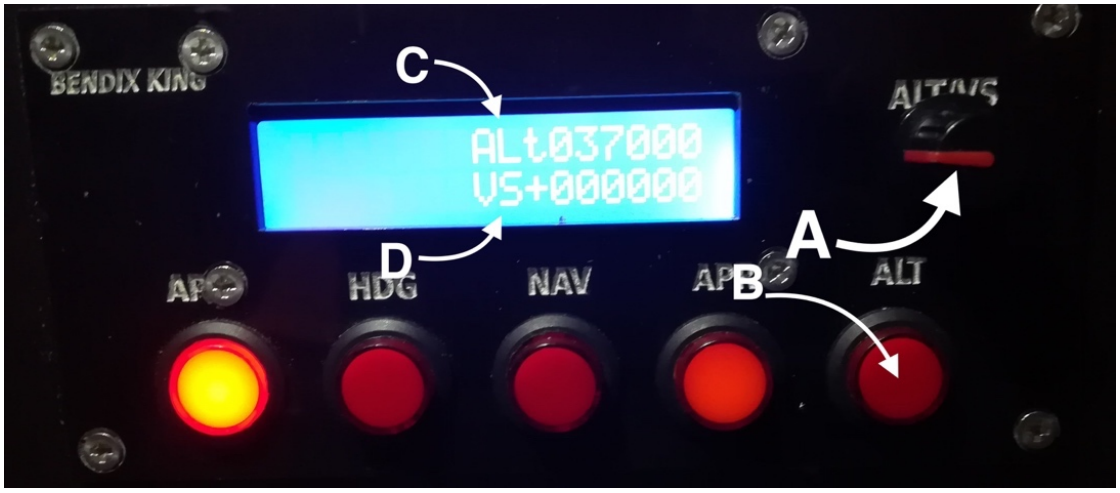


Figura VII.5. Ajuste de los parámetros de referencia del piloto automático.

VII.1.3 Uso del tren de aterrizaje y flaps

Las avionetas ligeras cuenta con 4 configuraciones de flaps: totalmente retraídos, a 10º, a 20º y a 30º. Cada una de estas configuraciones se selecciona mediante la colocación de la palanca de selección en la posición correspondiente (*A, Figura VII.6*).

La posición de 10º grados es la recomendada para el despegue de la aeronave [48, p. 3], mientras que un valor de flaps totalmente extendidos es lo aconsejable instantes antes de tocar tierra durante el aterrizaje para mejorar la sustentación durante el descenso [48, p. 14].

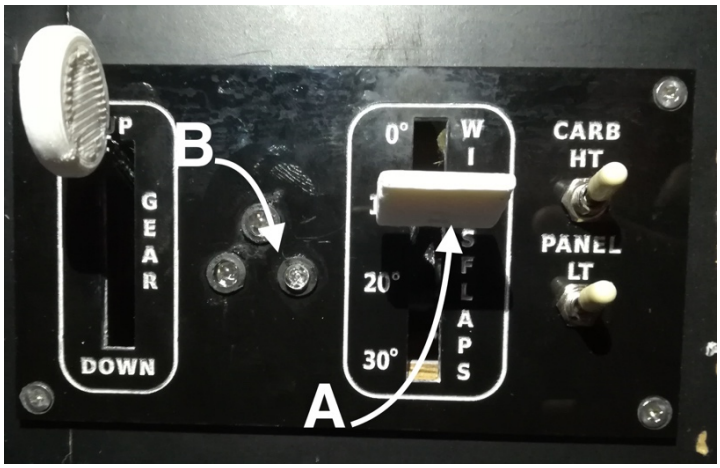


Figura VII.6. Palancas e indicadores del módulo de tren y flaps.

Por otro lado, es posible conocer el estado de cada rueda del tren de aterrizaje mediante el color del indicador correspondiente (B, *Figura VII.6*). El color amarillo señaliza tren en tránsito, es decir, se encuentra cambiando su posición. Una vez alcanza el color rojo, se encuentra totalmente extendido. Si el indicador se encuentra apagado significa que dicha rueda ya se encuentra recogida.

VII.1.4 Uso del magneto y otros conmutadores e indicadores

En el módulo de luces se localiza la llave de arranque del avión (A, *FiguraVII.7*) que cuenta con 5 posiciones. Para un correcto arranque se deben recorrer todas las posiciones. La primera es la de desactivación del motor, luego el encendido de los dos conjuntos de bobinas por separados, posteriormente la activación de ambos y por último el arranque del motor.

Figura VII.7. Magneto situado en el mando de luces.

La iluminación de la cabina se activa mediante el interruptor llamado *Panel LT* situado a la izquierda de las palancas de control de motores.

A la derecha de estos controles (*Figura VII.8*) se encuentra el ajuste de *Trim*: un giro de esta rueda hacia adentro, eleva el morro del avión en la posición de reposo y un giro hacia la otra dirección lo reduce. Dicha configuración se realiza para compensar la modificación del centro de gravedad de la aeronave según la carga que lleve.

Figura VII.8. Rueda de ajuste de Trim situada en el panel de gases.

VII.2 Configuración de *Link2FS*

Dado que la información del FSX es obtenida por el puerto *serial* mediante este programa, resulta vital realizar un buen ajuste del mismo. Éste cuenta con una serie de ventanas de configuración, que se detallan a continuación.

En la primera pestaña (*Figura VII.9*), *Communications Settings*, se debe elegir el puerto COM al que se ha conectado el *maestro* y seleccionar un tiempo de refresco (*Refresh time*) de 10 segundos y de ciclo (*Cycle time*) de 200 M/s.

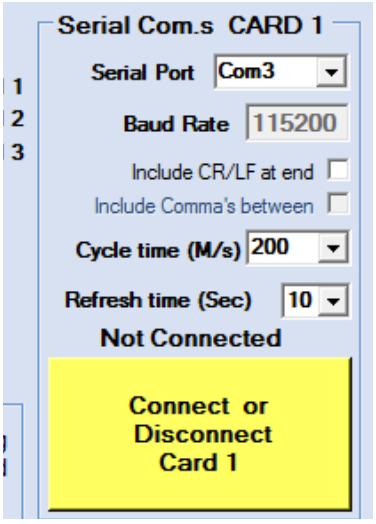


Figura VII.9. Configuración de la pestaña de comunicaciones.

La segunda pestaña debe tener seleccionada la extracción de los datos como se indica en la *Figura VII.10*.

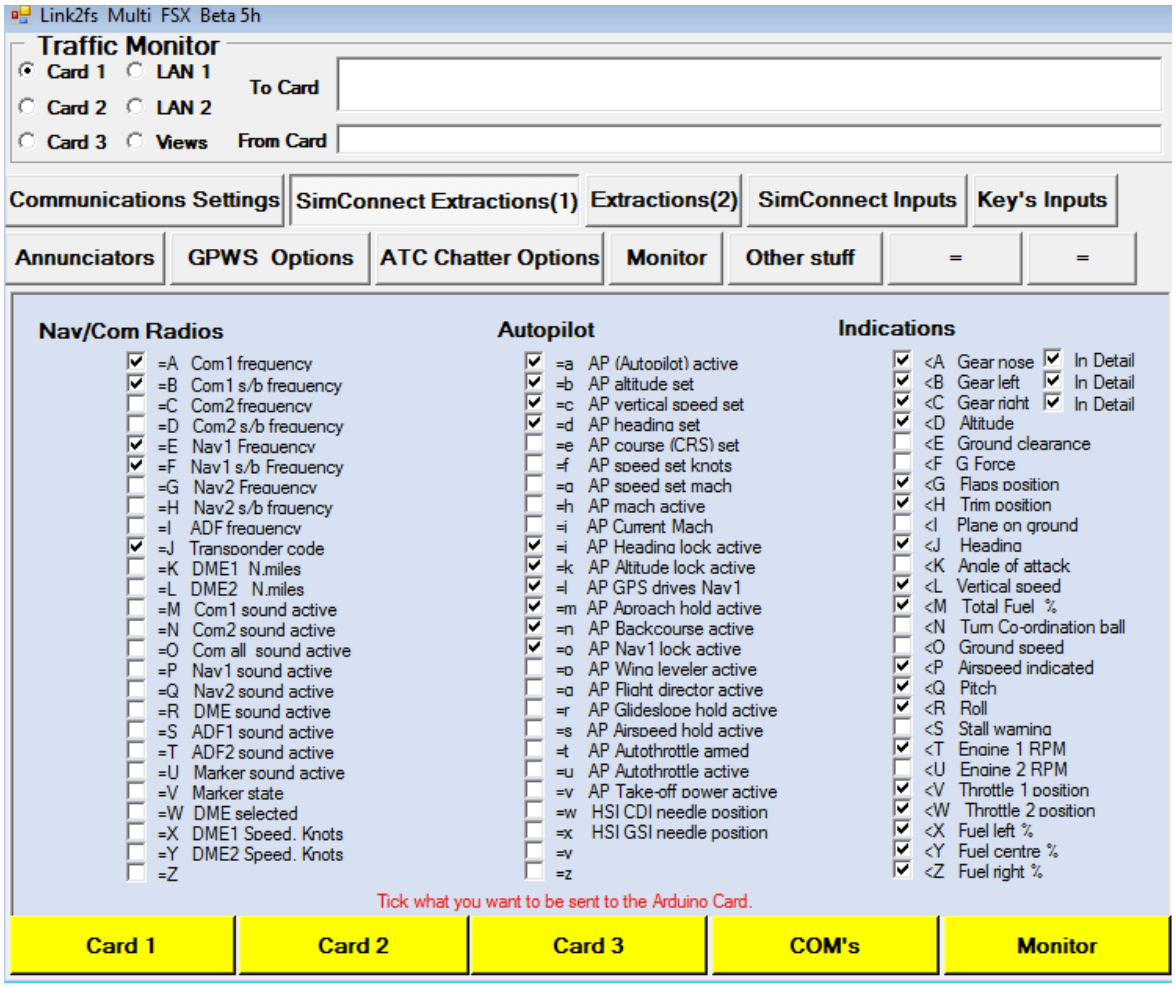


Figura VII.10. Selección de datos a extraer en Link2FS.

La pestaña *Extractions* aporta el resto de datos del avión que no se han seleccionado anteriormente. Su configuración debe ser como muestra la *Figura VII.11*.

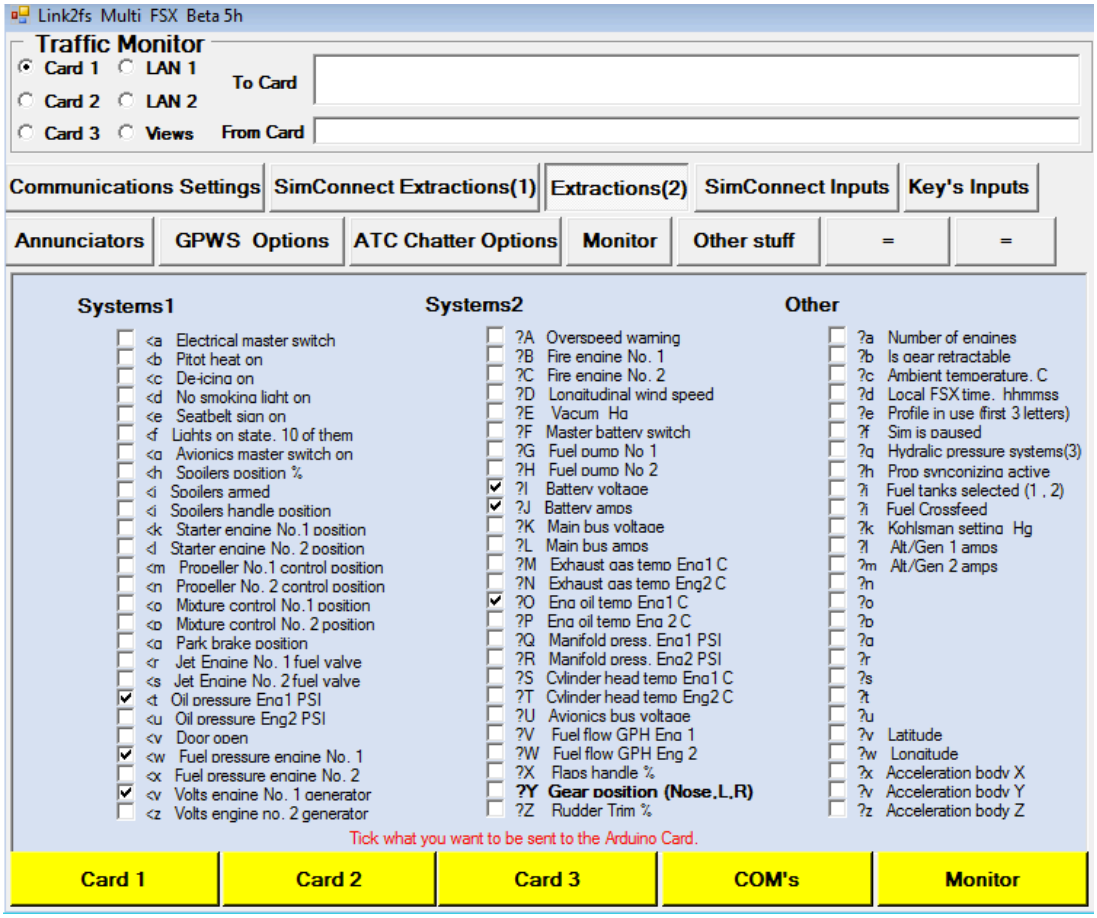


Figura VII.11. Datos a obtener en Extractions (Link2FS).

Los mensajes de aviso del avión se obtienen mediante la pestaña *Annunciators* (*Figura VII.12*), por lo que se deben activar los siguientes:

- ***/F1*** con un valor umbral de 50 PSI.
- ***/H1*** con un valor de 5%. Si se desea un aviso de nivel de combustible con un valor mayor, es suficiente con modificar este valor.
- ***/J1*** y ***/K1*** para los niveles de combustible de las alas. El umbral de activación se ha fijado en un 10% aunque su valor es modificable por el usuario.
- ***/N1*** para encender el aviso de problema de vacío.
- ***/P1*** para indicar un problema con la tensión cuando se reduzca por debajo de 20V su tensión.

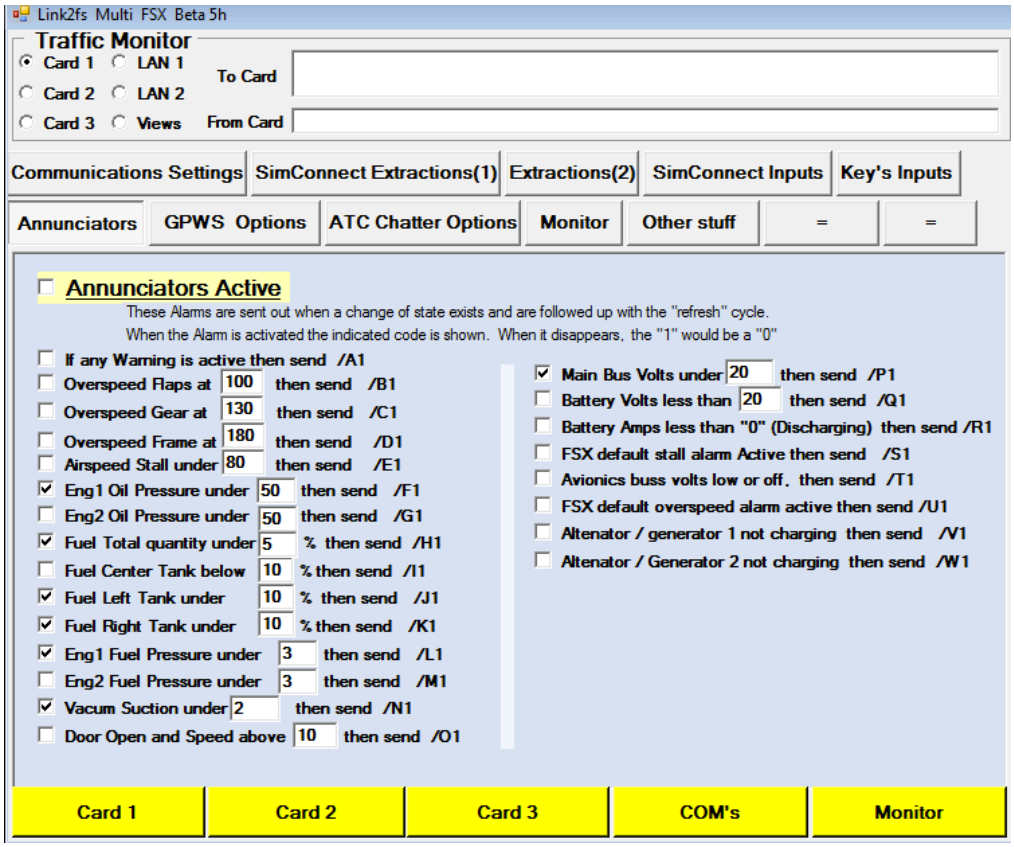


Figura VII.12. Ajuste de umbrales de mensajes de advertencia.

VII.3 Resultados

El simulador de vuelo fue probado de manera satisfactoria, comprobando que todos los instrumentos analógicos simulados mostraban en cada momento los valores reales del avión.

Se observó pequeñas diferencias en ciertos valores intermedios del recorrido de las agujas en los mecanismos que empleaban engranajes para la transmisión, especialmente en el *Altímetro* y el *Tacómetro*. Esto se debía a que los engranajes impresos contaban con imperfecciones que oponían demasiada resistencia al movimiento en ciertos ángulos dado que presentaban demasiado "juego" o bloqueaban el movimiento.

El comportamiento de todos los instrumentos analógicos se puede observar en la *Figura VII.12* donde se muestra una altura de 2000 pies, una velocidad de 200 nudos, con su motor a máxima potencia (3000 rpm) y un rumbo del piloto automático de 25º.

Figura VII.13. Gauges analógicos funcionando.

Los paneles anunciadores, muestran el comportamiento deseado. En determinadas ocasiones, por la distancia existente entre el metacrilato de la radio y la PCB que alberga los botones, es necesario retraer manualmente el botón impreso en PLA que acciona el pulsador.

En la *Figura VII.13* se pude observar la salida del panel de radio comparándola con los valores de FSX.

Figura VII.14. Panel de radio creado sincronizado con el simulador.

La *Figura VII.14* muestra el comportamiento del panel del piloto automático.

Figura VII.15. Panel del P.A. funcionando y sincronizado con FSX.

Para concluir, se ha probado el panel de luces, también de forma satisfactoria, observando que el avión replica de forma correcta el estado de todos los interruptores y captura la posición de la llave de arranque (*Figura VII.15*)

Figura VII.16. Panel de luces sincronizado con el simulador.

Aportaciones y Conclusiones

El presente trabajo ha tenido como objetivo el diseño e implementación de un simulador de vuelo que, mediante microcontroladores, ha permitido:

1. Diseñar y construir (tanto mecánica como electrónicamente) los diferentes indicadores analógicos de tipo reloj, que muestran los parámetros básicos de vuelo de cualquier avión ligero, gestionados mediante un microcontrolador ATmega 2560 que realiza las funciones de *maestro*.
2. Diseñar y construir los paneles de avisos y controles de radio, piloto automático, luces y motores, gestionados mediante dispositivos *esclavos* formados por ATmega 328P y expansores PCF8574. Estos paneles se comunican con el usuario mediante pantallas (LCD o 7 segmentos) e indicadores luminosos.
3. Diseñar un bus I^2C para interconectar todos los dispositivos *esclavos*, siendo todo el sistema gestionado por el ATmega 2560 mediante librerías propias.
4. Diseñar la arquitectura de un bus *SPI* con el que comunicar el módulo de radio con el *maestro* haciendo uso de la librería desarrollada.
5. Construir un mueble y salpicadero donde albergar todos los instrumentos y cableado para dar a la cabina un aspecto realista.

El funcionamiento del conjunto ha sido el esperado. No obstante, existen posibilidades de mejora como:

1. Diseñar y construir un *yoke* con el que controlar el avión, para así no recurrir a una solución comercial.
2. Construir la pedalera inferior con objeto de controlar el timón de cola del avión.
3. Desarrollar un archivo de configuración, mediante el cual sea pueda establecer el programa de simulación a emplear y así modificar de forma automática todos los mensajes enviados por el puerto *serial*. Esto permitiría emplear este sistema electrónico con cualquier *software* comercial y no estaría limitado, como en la actualidad, al uso de FSX.
4. Mejorar el método fabricación de los engranajes, o bien, adaptarlo a medidas comerciales para reducir las holguras y obtener un mejor comportamiento de los *gauges*.

Conclusions

The present Final Degree Project has had as objective the design and implementation of a flight simulator that, by means microcontroller, has allowed:

1. To design and build the different analog gauges to show the basic flight parameters. Everything was managed by the master, the ATmega 2560 microcontroller.
2. To design and build the warnings panels and radio controls, autopilot, lights and engines, managed by the ATmega 328P slave microcontrollers and expanders PCF8574. These panels communicate with the user using screens (7 segments and LCD) and light indicators.
3. To design and I^2C bus to communicate all the slaves devices using own libraries.
4. To design the architecture of an *SPI* bus to communicate the radio module with the master making use of its own developed library.
5. To build a cockpit where to house all the instrument and the wiring to give the cockpit a realistic look.

The simulator operation has been as expected. However, there are improvement possibilities such as:

1. To design a yoke to control the airplane and so as not to use a commercial solution.
2. To build the lower pedalboard to control the aircraft tail rudder.
3. To develop a configuration file, through which you can establish the simulation program to use and automatically modify all messages sent by the serial port. This would allow to use this electronic system with any commercial *software* and would not be limited, as at present, to the FSX use.
4. To improve the gear manufacturing method, or adapt the gears to commercial measures to reduce gaps and obtain a better gauges performance.

Presupuesto

Costes Materiales

PRESUPUESTO MATERIAL			
Descripción	Cantidad	Coste unitario (€/u)	Coste total
Resistencias 10k	30	0,14 €	4,20 €
Resistencias 1k	34	0,14 €	4,76 €
Resistencias 470 Ohm	2	0,14 €	0,28 €
Resistencias 330 Ohm	20	0,14 €	2,80 €
Leds Monocolor	18	0,16 €	2,88 €
Leds RGB	3	0,25 €	0,75 €
Interruptores rotativos 12 vías	6	4,30 €	25,80 €
Encoders	6	2,50 €	15,00 €
Condensadores 100 nF	16	0,23 €	3,68 €
Condensadores 1 uF	4	0,23 €	0,92 €
PinHeader 2.54 mm	15	0,65 €	9,75 €
Zócalo PCB 16 pines	15	0,24 €	3,60 €
Zócalo PCB 28 pines	3	0,22 €	0,66 €
Tiras de zócalo	10	0,85 €	8,50 €
C.I. PCF8574	5	1,25 €	6,25 €
C.I. CD40XXBE	7	0,44 €	3,08 €
Expansor I^2C LCD	1	1,95 €	1,95 €
LCD 16x2	1	9,74 €	9,74 €
Display 7 seg. 4 dígitos (3461AS)	5	3,38 €	16,90 €
Display 7 seg. 1 dígito (3161AS)	4	2,35 €	9,40 €
Pulsador PCB	6	0,75 €	4,50 €
Toggle *Switch*	10	2,22 €	22,20 €
Embellecedor interruptor	10	0,41 €	4,10 €
Interruptor rectangular	3	3,90 €	11,70 €
Interruptor retroiluminado	5	3,60 €	18,00 €
Servo motor SG-90	12	3,50 €	42,00 €
Motor 28-BYJ-48	3	2,85 €	8,55 €
C.I. ULN2003	3	0,57 €	1,71 €
ATMega 2560 Embebido	1	12,24 €	12,24 €
ATMega 328P	2	5,99 €	11,98 €
Conectores Dupond (Pack 20)	5	4,50 €	22,50 €

Cable 0.5 mm^2 (m)	5	0,60 €	3,00 €
Cable 1.5 mm^2 (m)	1	1,10 €	1,10 €
Fuente alimentación 5 V	1	6,80 €	6,80 €
Placa Cobre fotos. 2 caras (16x10x0,16 cm)	5	6,27 €	31,35 €
Placa Cobre fotos. 2 caras (30x20x0,16 cm)	2	19,51 €	39,02 €
Conector IDC 20 pines PCB	6	2,73 €	16,38 €
Conector IDC 10 pines PCB	6	2,73 €	16,38 €
Conector IDC 30 pines PCB	2	2,73 €	5,46 €
Conector IDC Cable	14	2,73 €	38,22 €
Cable paralelo (m)	6	6,50 €	39,00 €
Conectores Phoenix 2.54 mm	6	1,74 €	10,44 €
C.I. MAX 7219	3	10,28 €	30,84 €
Potenciómetro PCB 10 k	5	1,08 €	5,40 €
Potenciómetro lineal 10 k	3	3,85 €	11,55 €
Potenciómetro rotativo 10k	2	1,42 €	2,84 €
Madera DM 10 mm (m^2)	5	11,00 €	55,00 €
Madera DM 3 mm (m^2)	2	7,00 €	14,00 €
Filamento PLA negro (m)	40	0,24 €	9,60 €
Filamento PLA blanco (m)	10	0,24 €	2,40 €
Filamento PLA amarillo (m)	3	0,24 €	0,72 €
Varilla roscada 6 mm	1	5,69 €	5,69 €
Tornillo M3x150 mm	30	0,12 €	3,60 €
Tornillo M3x100 mm	30	0,10 €	3,00 €
Tornillo M3x120 mm	30	0,11 €	3,30 €
Tornillo M4x15 mm	50	0,08 €	4,00 €
Varilla eje 3 mm (m)	1	5,50 €	5,50 €
Varilla hueca eje 4 mm (m)	1	6,20 €	6,20 €
Varilla hueca eje 5 mm (m)	1	6,20 €	6,20 €
Rodamiento 20x08 mm	2	5,50 €	11,00 €
Sensor de barrera fotoeléctrico	3	4,00 €	12,00 €
Diodo emisor IR	1	2,50 €	2,50 €
FotoTransistor	1	2,85 €	2,85 €
Metacrilato negro 2mm (m^2)	0,12	180,00 €	21,60 €
Metacrilato transparente 2 mm (m^2)	0,15	102,00 €	15,30 €

TOTAL MATERIALES	732,62 €

Precios extraídos de la *base de datos de RS* [49]

Costes de Mano de Obra

PRESUPUESTO MANO DE OBRA			
Concepto	Cantidad (h)	Coste unitario (€/h)	Coste total
Tiempo análisis y diseño	90	28,00 €	2.520,00 €
Tiempo de implementación	360	20,00 €	7.200,00 €
Tiempo de documentación	90	18,00 €	1620,00 €

Total Coste Mano de Obra	11.340,00 €

Las horas de mano de obra se han calculado teniendo en cuenta un trabajo diario de 3 horas durante 5 días semanales en el período de octubre de 2018 a mayo de 2019, es decir un total de 36 semanas a 15 h/semanales.

Coste Total

COSTE TOTAL	
Total Coste Materiales (MT)	732,62 €
Total Coste Mano Obra (MO)	11.340,00 €
Gastos Generales 6% (MT+MO)	724,36 €
Beneficio Industrial 13% (MT+MO)	1.569,44 €

Coste Total Proyecto	14.366,42 €

Glosario

ADC	*Analog Digital Converter*	**FSX**	*Flight Simulator X*
ASCII	*American Standard Code for Information Interchange*	**GERBER**	*Formato estándar de intercambio de ficheros de PCB*
ATC	*Air Traffic Control*	**GND**	Ground
ATR	*Fabricante de aviones perteneciente al grupo AirBus*	**HDG**	*Heading*
AVR	*Familia de microcontroladores de ATmega*	**IAS**	*Indicated Air Speed*
BCN	*Beacon Light*	**IDC**	*Insulation-Displacement Connector*
CLK	*Clock*	**IDE**	*Integrated Development Environment*
CPU	*Central Processing Unit*	**I²C**	*Inter-Integrated Circuit*
DIP	*Dual In Package*	**ILS**	*Instrument Landing System*
DME	*Distance Measuring Equipmente*	**IVAO**	*International Virtual Aviation Organisation*
EEPROM	*Electrically Erasable Programmable Read-Only Memory*	**LCD**	*Liquid Cristal Display*
EN	*Enable*	**LED**	*Light-Emitting Diode*
FET	*Fiel-Effect Transistor*	**LIFO**	*Last Input First Output*

LSB	*Least Significant Bit*	**MOSI**	*Master Output Slave Inpt*
L2FS	*Link To Flight Simulator*	**MSB**	*Most Significant Bit*
NAV	*Navigation*	**SPI**	*Serial Peripheral Interface*
NPN	*Tipo de transistor de unión bipolar*	**SMD**	*Surface Mounted Device*
PC	*Personal Computer*	**SRAM**	*Static Random Access Memory*
PCB	*Printed Circuit Board*	**SS**	*Slave Selector*
PLA	*Poliácido láctico*	**STL**	*Standard Triangle Language*
PWM	*Pulse-Width Modulation*	**TTL**	*Transistor to Transistor Logic*
RISC	*Reduced Instruction Set Computer*	**UART**	*Universal Asynchronous Receiver-Transmitter*
RPM	*Revolution per Minute*	**USB**	*Universal Serial Bus*
SCL	*Serial Clock Line*	**VOR**	*VHF Onmidirectional Radio*
SCK	*Serial Clock*	**VS**	*Vertical Speed*
SDA	*Serial Data*	**XPDR**	*Transponder*
MISO	*Master Input Slave Output*		

Bibliografía

[1] Microsoft, [En línea]. Available: https://bit.ly/2FUMsDf. [Último acceso: 28 Enero 2019].

[2] Laminar Research, [En línea]. Available: https://www.x-plane.com/. [Último acceso: 28 Enero 2019].

[3] Lockheed Martin, [En línea]. Available: https://www.prepar3d.com/.

[4] International Virtual Aviation, [En línea]. Available: https://www.ivao.aero/. [Último acceso: 28 Enero 2019].

[5] Logitech/Saitek, [En línea]. Available: https://bit.ly/2sUkcIB. [Último acceso: 29 Enero 2019].

[6] Microchip, «Microchip,» [En línea]. Available: https://www.microchip.com/. [Último acceso: 30 Enero 2019].

[7] Arduino, «Arduino,» [En línea]. Available: https://store.arduino.cc/arduino-nano. [Último acceso: 30 Enero 2019].

[8] I. Dogan, Programación de microcontroladores PIC, Barcelona: Marcombo, 2007.

[9] Arduino, «Arduino,» [En línea]. Available: https://store.arduino.cc/arduino-mega-2560-rev3. [Último acceso: 30 Enero 2019].

[10] WikiDot, «WikiDot,» [En línea]. Available: http://sistdig.wikidot.com/wiki:arquitectura. [Último acceso: 3 Enero 2019].

[11] U. d. Valencia, «ocw.uv.es,» 2011. [En línea]. Available: http://xurl.es/d6b1i. [Último acceso: 8 Enero 2019].

[12] D. 28BYJ-48, «roborcraft.ru,» s.f.. [En línea]. Available: http://robocraft.ru/files/datasheet/28BYJ-48.pdf.

[13] A. P. Malvino, «Seguidores de Emisor. Conexiones Darlington,» de *Principios de Electrónica*, McGraw Hill, 2000, p. 1126.

[14] D. ULN2003A, «Texas Instrument,» [En línea]. Available: http://www.ti.com/lit/ds/symlink/uln2003a.pdf. [Último acceso: 12 Enero 2019].

[15] D. TCST1103, «vishay.com,» [En línea]. Available: https://www.vishay.com/docs/83764/tcst1103.pdf. [Último acceso: 21 Enero 2019].

[16] D. SG-90, «ee.ic.ac,» [En línea]. Available: http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf . [Último acceso: 28 Enero 2019].

[17] PCF8574, «Texas Instrument,» [En línea]. Available: http://www.ti.com/lit/ds/symlink/pcf8574.pdf. [Último acceso: 29 Octubre 2018].

[18] D. 3641-AS, «peterparts.com,» [En línea]. Available: https://www.peterparts.com/CatalogPages/57/1007.pdf. [Último acceso: 25 Enero 2019].

[19] D. 3161-AS, «kitronik.co.uk,» [En línea]. Available: https://www.kitronik.co.uk/blog/7-segment-display-datasheet/. [Último acceso: 25 Enero 2019].

[20] D. MAX7219, «Sparkfun,» [En línea]. Available: https://www.sparkfun.com/datasheets/Components/General/COM-09622-MAX7219-MAX7221.pdf. [Último acceso: 25 Enero 2019].

[21] L. Llamas, «Ingeniería, informática y diseño,» [En línea]. Available: https://bit.ly/2Ws0nGd. [Último acceso: 30 Enero 2019].

[22] D. KY-040, «eeshop,» [En línea]. Available: http://eeshop.unl.edu/pdf/KEYES%20Rotary%20encoder%20module%20KY-040.pdf. [Último acceso: 3 Enero 2019].

[23] Arduino, «Arduino,» [En línea]. Available: https://www.arduino.cc/en/main/software.

[24] Microsoft, «Requisitos mínimos FSX,» s.f.. [En línea]. Available: https://support.microsoft.com/es-es/help/925724/flight-simulator-x-minimum-system-requirements.

[25] JimSpace, «Link2FS,» [En línea]. Available: http://www.jimspage.co.nz/intro.html. [Último acceso: 2 Febrero 2019].

[26] KiCAD org, [En línea]. Available: http://kicad-pcb.org/. [Último acceso: 5 Febrero 2019].

[27] AutoDesk, «Autodesk,» [En línea]. Available: https://www.autodesk.com/products/fusion-360/overview. [Último acceso: 4 Febrero 2019].

[28] Repetier, 28 Enero 2019. [En línea]. Available: https://www.repetier.com/.

[29] I2C-Bus.ORG. [En línea]. Available: https://www.i2c-bus.org/specification/. [Último acceso: 18 Diciembre 2018].

[30] C. d. Rp, «Texas Instrument,» s.f.. [En línea]. Available: http://www.ti.com/lit/an/slva689/slva689.pdf.

[31] Sparkfun, «sparkfun.com,» [En línea]. Available: https://cdn.sparkfun.com/datasheets/Dev/Arduino/Other/CH340DS1.PDF. [Último acceso: 28 Diciembre 2018].

[32] Matherthel, «GitHub.com,» [En línea]. Available: https://github.com/mathertel/LiquidCrystal_PCF8574. [Último acceso: 21 Octubre 2018].

[33] «AviacionD,» [En línea]. Available: http://aviaciond.com/sistema-de-vuelo-automatico-airbus-a340/. [Último acceso: 3 Febrero 2019].

[34] L. i. d. v. y. s. visto, «Aertecsolution.com,» [En línea]. Available: https://aertecsolutions.com/?wpfb_dl=1156. [Último acceso: Febrero 2019].

[35] P. t. i. airplanes, «abdcienciade,» 19 Junio 2009. [En línea]. Available: https://abcienciade.files.wordpress.com/2009/06/pitot-tubes-in-airplanes.pdf. [Último acceso: 10 Febrero 2019].

[36] T. Aviation, «beechcraft.com,» [En línea]. Available: https://beechcraft.txtav.com/en/baron-g58. [Último acceso: 9 Febrero 2019].

[37] M. H., «Instructables,» s.f.. [En línea]. Available: https://www.instructables.com/id/Arduino-LeonardoMicro-as-Game-ControllerJoystick/.

[38] J. J. Alcubilla, Comunicación y sistemas de información de las aeronaves, Madrid: Paraninfo, 2017.

[39] «WikiWand,» [En línea]. Available: https://www.wikiwand.com/es/Cessna_421. [Último acceso: 17 Marzo 2019].

[40] Á. Alea, «El Blog de Alea,» [En línea]. Available: http://aleascockpits.blogspot.com/. [Último acceso: 20 Diciembre 2018].

[41] «aylampmechatronics,» [En línea]. Available: https://naylampmechatronics.com/arduino-tarjetas/402-arduino-mega-2560-embebido.html. [Último acceso: 4 Marzo 2019].

[42] M. B. V., TTL circuitos integrados digitales, Madrid: Paraninfo, 1991.

[43] «Sparkfun,» [En línea]. Available: https://cdn.sparkfun.com/datasheets/Dev/Arduino/Other/CH340DS1.PDF. [Último acceso: 21 Diciembre 2018].

[44] HardwareSerial.cpp, «GitHub,» s.f.. [En línea]. Available: https://github.com/arduino/ArduinoCore-avr/tree/master/cores/arduino. [Último acceso: 10 Febrero 2019].

[45] Arduino, «Arduino CC. Funciones Serial,» [En línea]. Available: https://www.arduino.cc/reference/en/language/functions/communication/serial/. [Último acceso: 19 Noviembre 2018].

[46] Arduino, «Arduino.cc,» [En línea]. Available: https://www.arduino.cc/en/Reference/WireOnReceive. [Último acceso: 4 Marzo 2019].

[47] I. a. l. aeronáutica, «diazdesantos.com,» s.f.. [En línea]. Available: https://www.editdiazdesantos.com/wwwdat/pdf/9788479789374.pdf.

[48] M. d. v. C. 172, «logqslbyc,» s.f.. [En línea]. Available: http://www.logqslbyc.com/c_172/C_172.pdf.

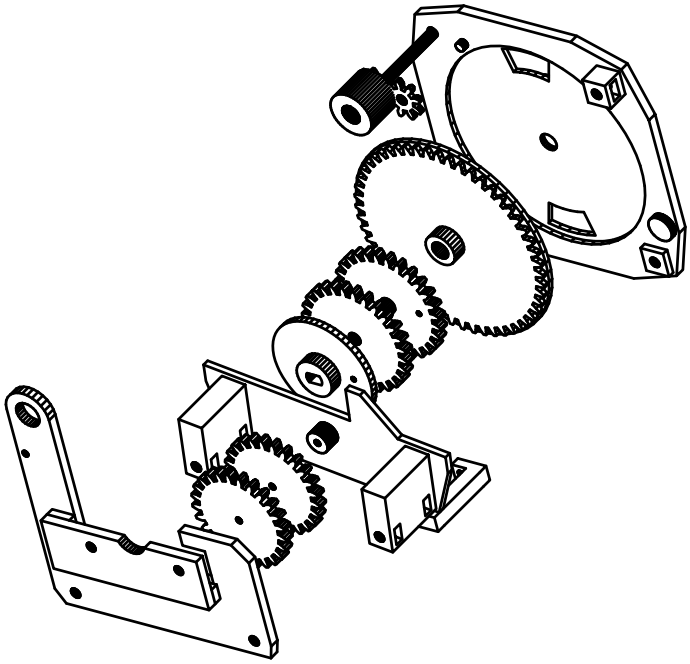
[49] R. Componentes, «RS,» s.f.. [En línea]. Available: https://es.rs-online.com/web/. [Último acceso: 28 Junio 2019].

[50] Raikoo, «Raikoo,» [En línea]. Available: https://bit.ly/2BcLEph. [Último acceso: 2 Febrero 2019].

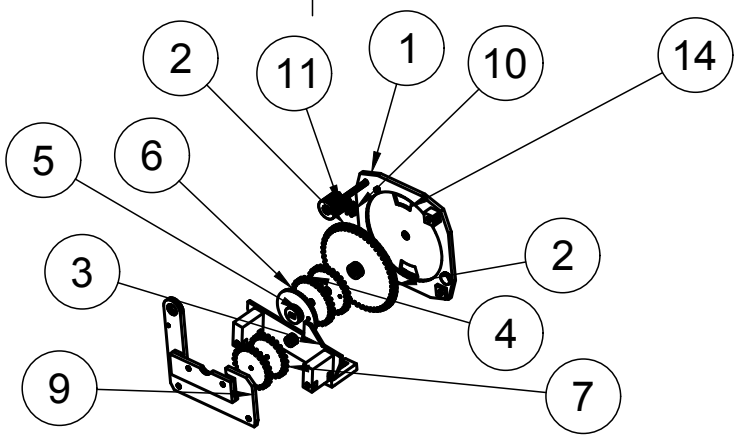
[51] J. A. Mazon, Navegación aérea, Madrid: Paraninfo, 2008.

ANEXOS

Plano de piezas mecánicas



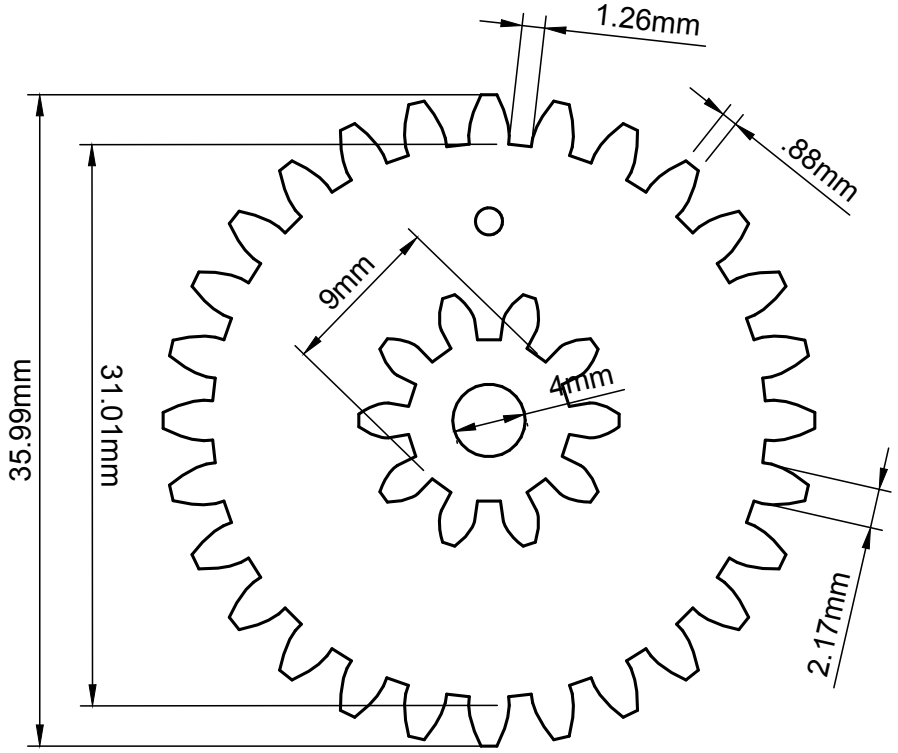
Grado Ing. Electrónica Ind. y A.	Creado por Augusto Samuel Hernández Martín	Fecha 29/11/2018
	Nombre Archivo Altímetro	Estado Documento Definitivo
	Titulo Ensamblaje de altímetro Simulador de Vuelo TFG	Escala 2:3
		Rev. 1.0 / Hoja 1/1

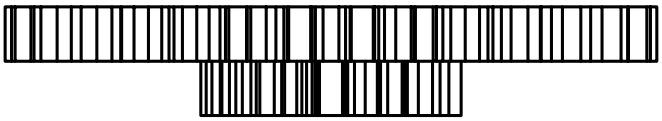


Parts List

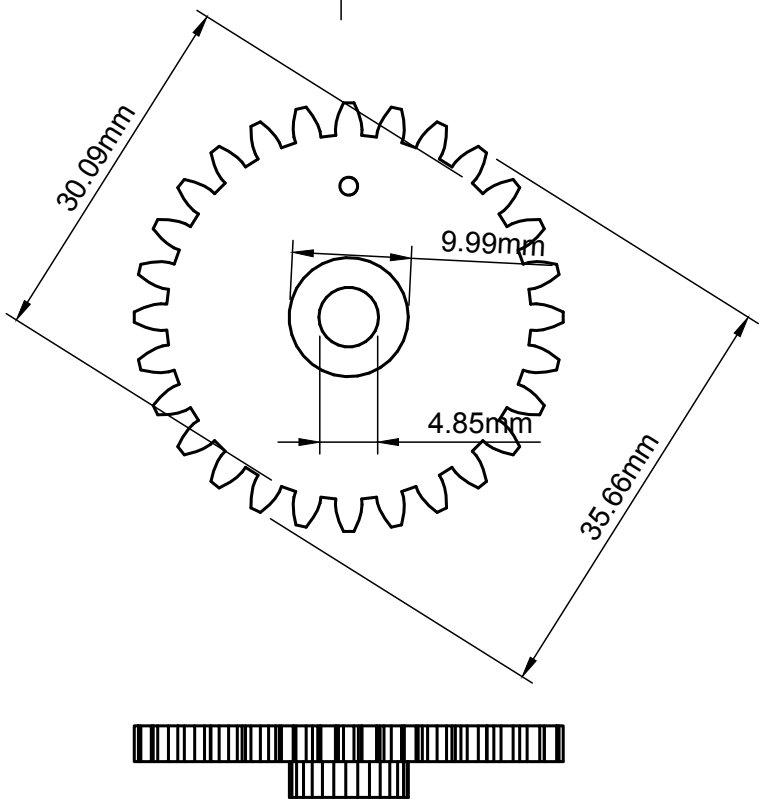
Item	Qty	Part Number	Description	Material
1	1	Fondo		Steel
2	1	Disco dentado		Steel
3	1	Parte superior		Steel
4	1	Engranaje 3		Steel
5	1	Disco con engranaje acoplable a stepper		Steel
6	1	Engranaje 2		Steel
7	2	Engranaje auxiliar		Steel
8	1	Arandela gruesa		Steel
9	1	Soporte superior 2		Steel
10	1	Engranaje pequeño		Steel
11	1	Engranaje con eje		Steel
12	1	Pata		Steel
13	1	Aguja 1		Steel
14	1	Aguja 10 mil pies		Steel
15	1	Aguja 2		Steel

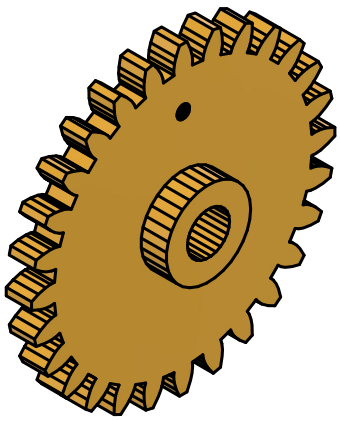
Grado: Ing. Electrónica Ind. y A.	Creado por: Augusto Samuel Hernández Martín	Fecha: 29/11/2018
	Nombre Archivo: Altímetro	Estado Documento: Definitivo
	Titulo: Partes Altímetro Simulador de Vuelo TFG	Escala: 1:4
		Rev.: 1.0 / Hoja: 1/1



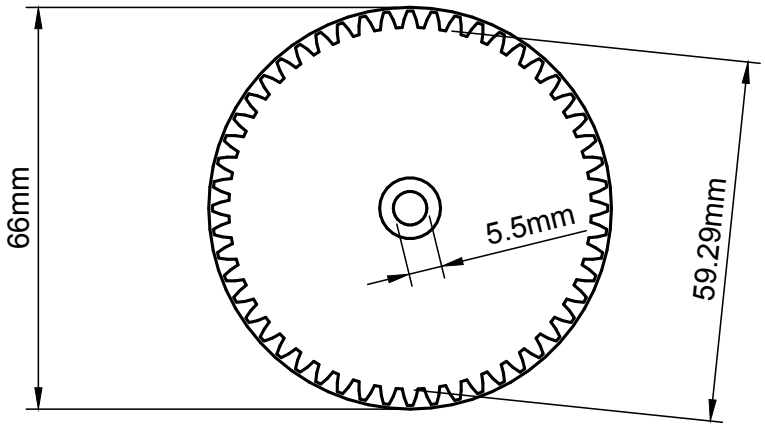


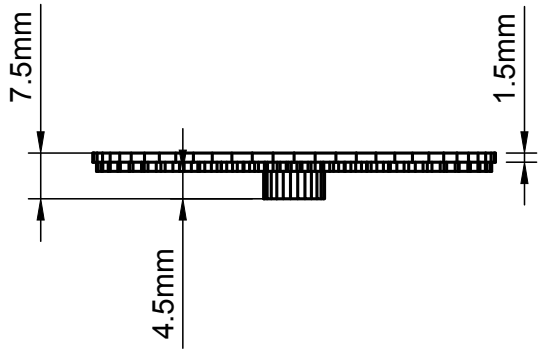
Grado Ing. Electrónica Ind. y A.	Creado por Augusto Samuel Hernández Martín	Fecha 29/11/2018
	Nombre Archivo Engranaje 2	Estado Documento Definitivo
	Titulo Engranaje 2 Simulador de Vuelo TFG	Escala 2:1
		Rev. 1.0 / Hoja 1/1

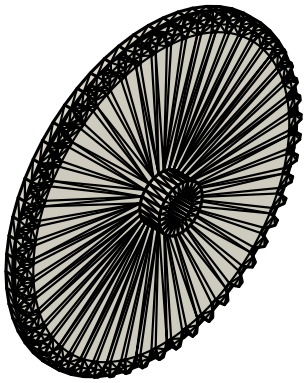




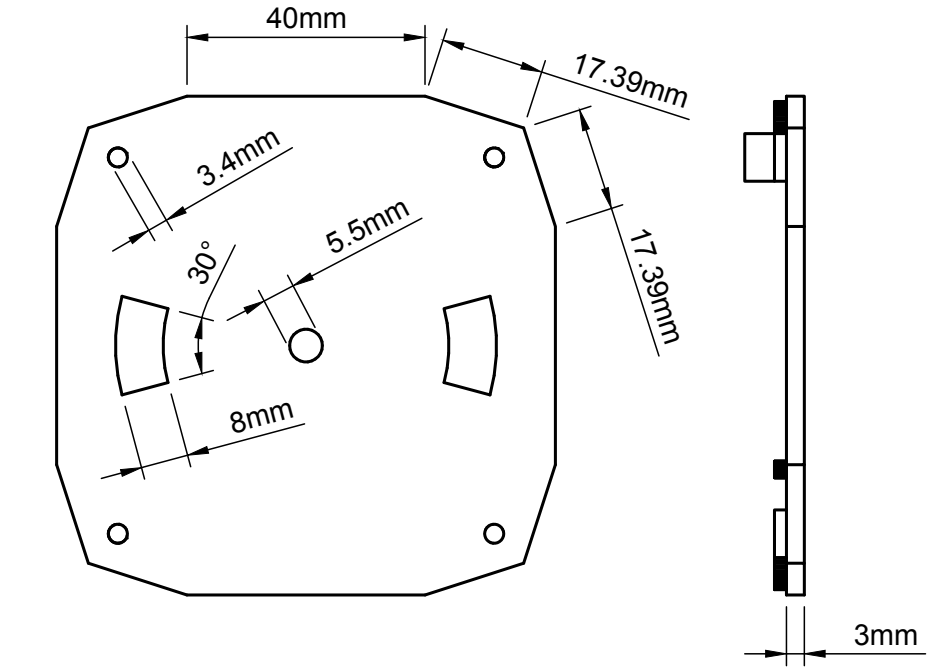
Grado	Creado por	Fecha
Ing. Electrónica Ind. y A.	Augusto Samuel Hernández Martín	29/11/2018
	Nombre Archivo: Engranaje 3	Estado Documento: Definitivo
	Titulo: Engranaje 3 Simulador de Vuelo TFG	Escala: 2:1
		Rev. 1.0 / Hoja 1/1

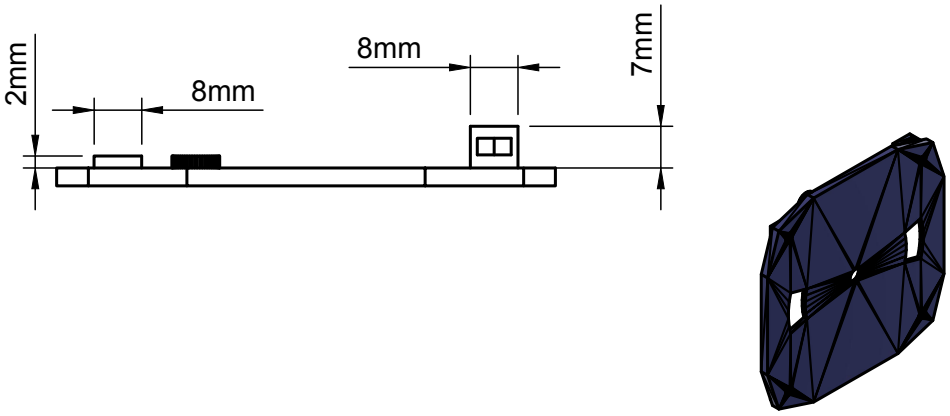




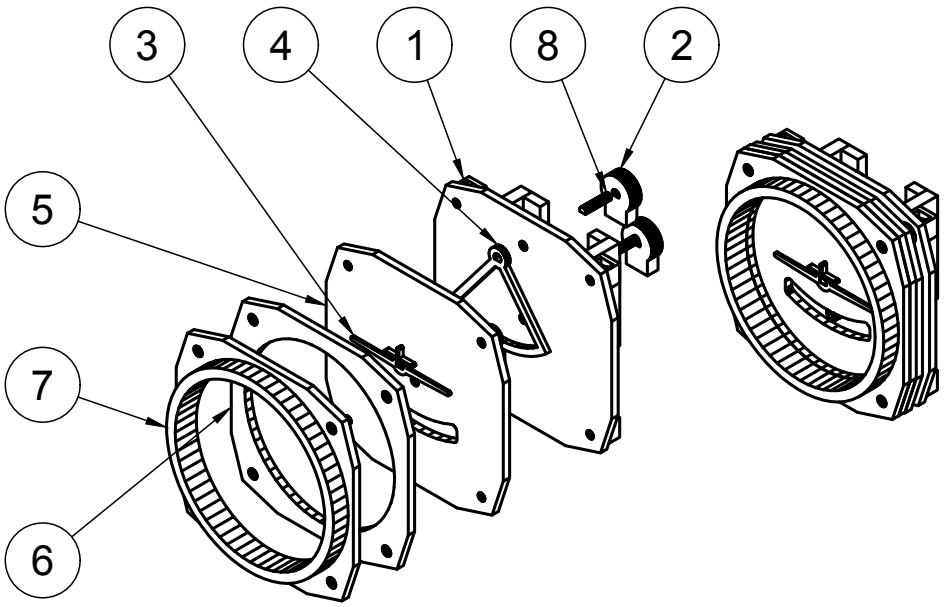


Grado Ing. Electrónica Ind. y A.	Creado por Augusto Samuel Hernández Martín	Fecha 29/11/2018
	Nombre Archivo Disco dentado Altímetro	Estado Documento Definitivo
	Titulo Disco dentado Simulador de Vuelo TFG	Escala 1:1
		Rev. 1.0 / Hoja 1/1





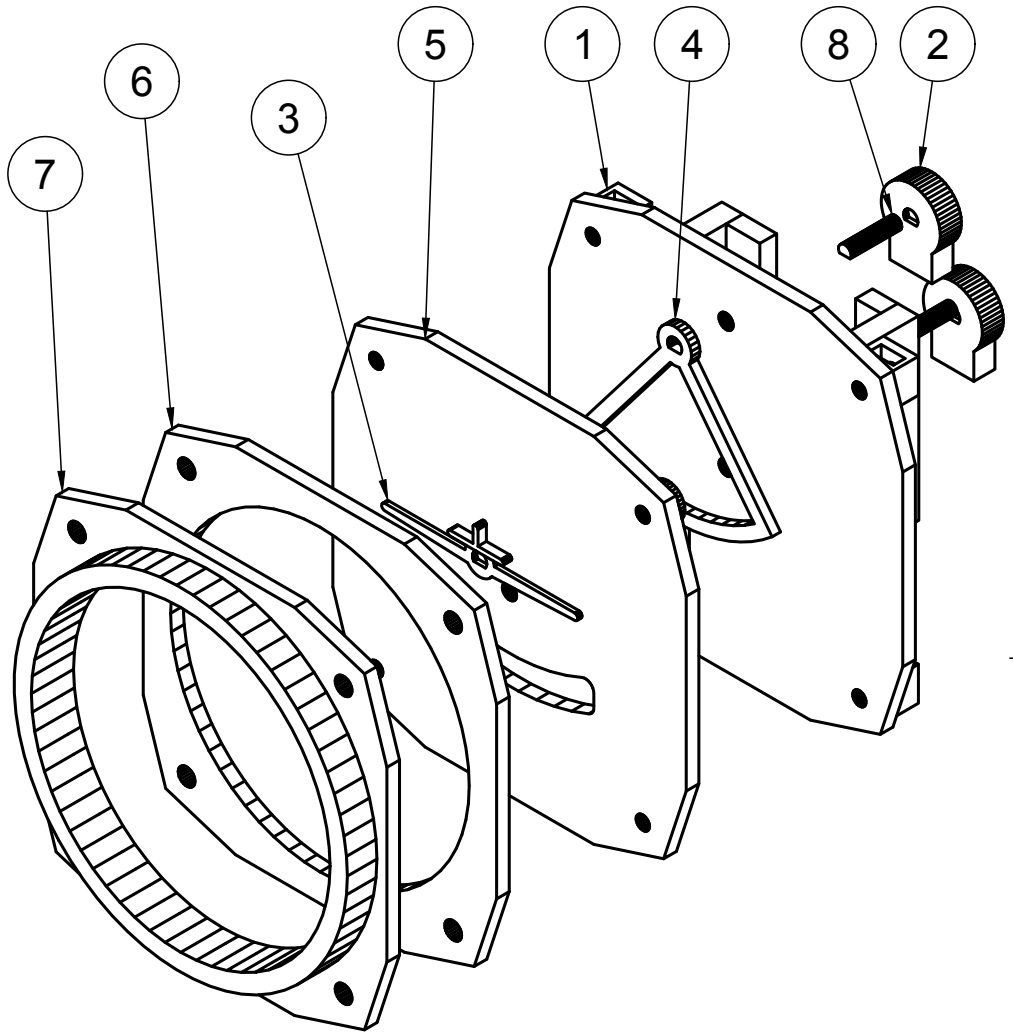
Grado Ing. Electrónica Ind. y A.	Creado por Augusto Samuel Hernández Martín	Fecha 29/11/2018
	Nombre Archivo Fondo Altímetro	Estado Documento Definitivo
	Titulo Fondo Simulador de Vuelo TFG	Escala 1:1 Rev. 1.0 / Hoja 1/1



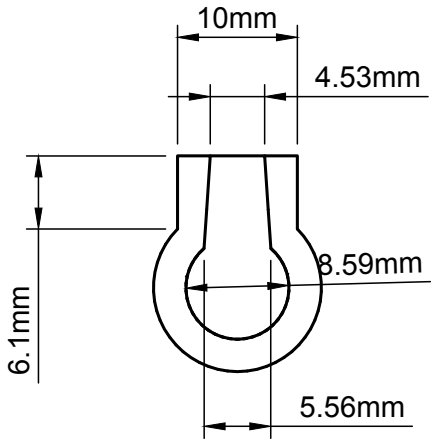
Item	Qty	Part Number	Description	Material
8	2	Eje Turn Coord		Steel
7	1	Bisel		Steel
6	1	Separador		Steel
5	1	Separador 2 de Turn Coord		Steel
4	1	Bola		Steel
3	1	Aguja forma de avion		Steel
2	2	Acople servo y eje Turn Coord		Steel
1	1	Soporte para 2 servos (Turn Coord)		Steel

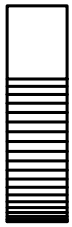
Parts List

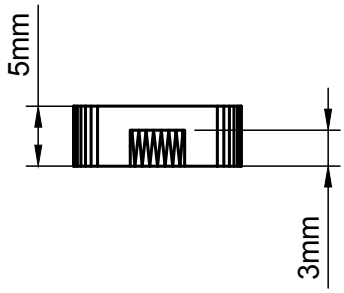
Grado: Ing. Electrónica Ind. y A.	Creado por: Augusto Samuel Hernández Martín	Fecha: 29/11/2018
	Nombre Archivo: Piezas de Coordinador de giro	Estado Documento: Definitivo
	Titulo: Turn Coordinator Ensamblaje Simulador de Vuelo TFG	Escala: 1:2
		Rev.: 1.0 / Hoja: 1/1

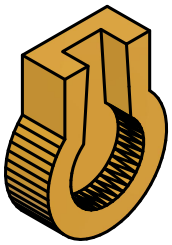


Grado Ing. Electrónica Ind. y A.	Creado por Augusto Samuel Hernández Martín	Fecha 29/11/2018
	Nombre Archivo Coordinador de Giro	Estado Documento Definitivo
	Titulo Turn Coordinator Ensamblaje Simulador de Vuelo TFG	Escala 1:1
		Rev. 1.0 / Hoja 1/1

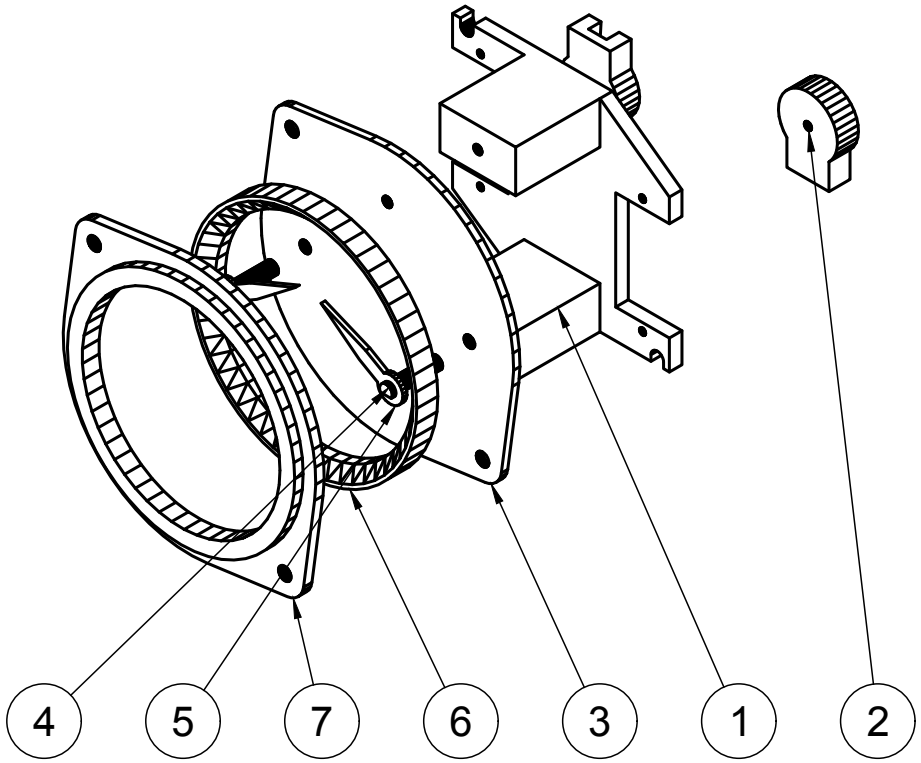








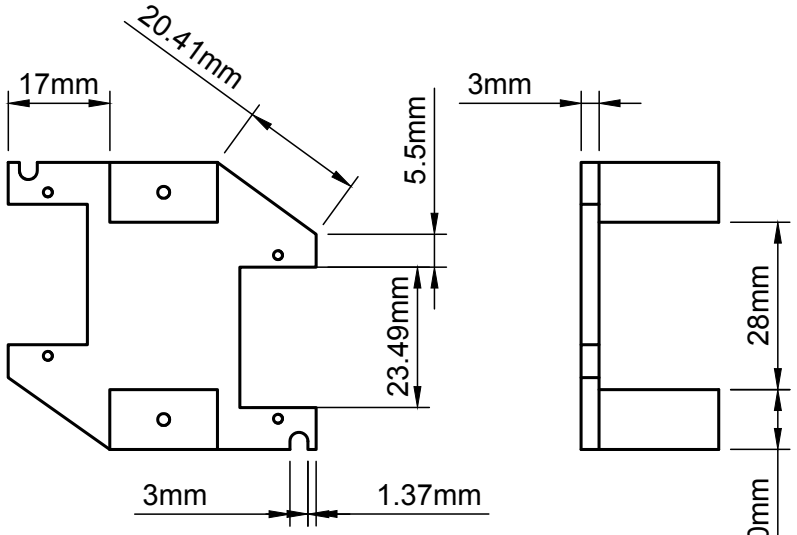
Grado: Ing. Electrónica Ind. y A.	Creado por: Augusto Samuel Hernández Martín	Fecha: 29/11/2018
	Nombre Archivo: Acople servomotores	Estado Documento: Definitivo
	Titulo: Acople servo y eje Turn Coord Simulador de Vuelo TFG	Escala: 2:1 Rev.: 1.0 Hoja: 1/1

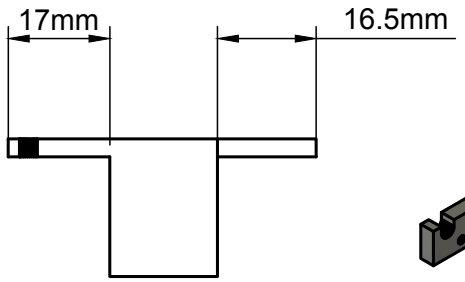


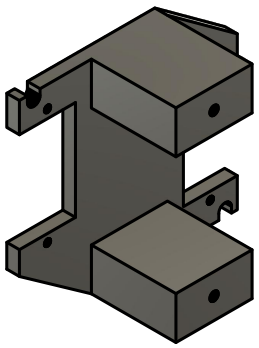
Item	Qty	Part Number	Description	Material
7	1	Bisel		Steel
6	1	Aro separador		Steel
5	2	Aguja		Steel
4	2	Eje		Steel
3	1	Base trasera		Steel
2	2	Acople eje con servo		Steel
1	1	Soporte de servos		Steel

Parts List

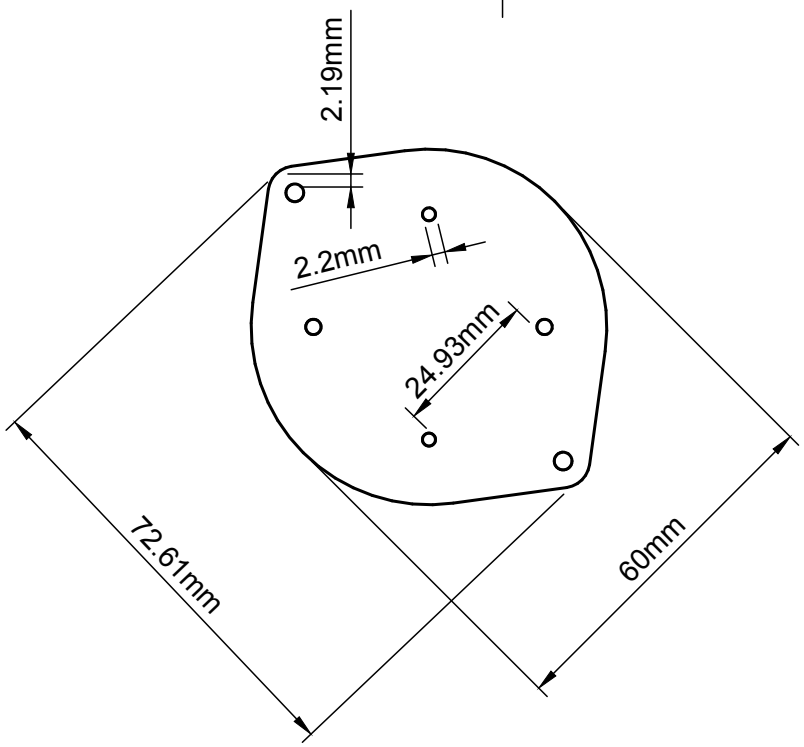
Grado: Ing. Electrónica Ind. y A.	Creado por: Augusto Samuel Hernández Martín	Fecha: 29/11/2018
	Nombre Archivo: Indicadores duales	Estado Documento: Definitivo
	Titulo: Indicadores dobles Simulador de Vuelo TFG	Escala: 1:1
		Rev.: 1.0 / Hoja: 1/1



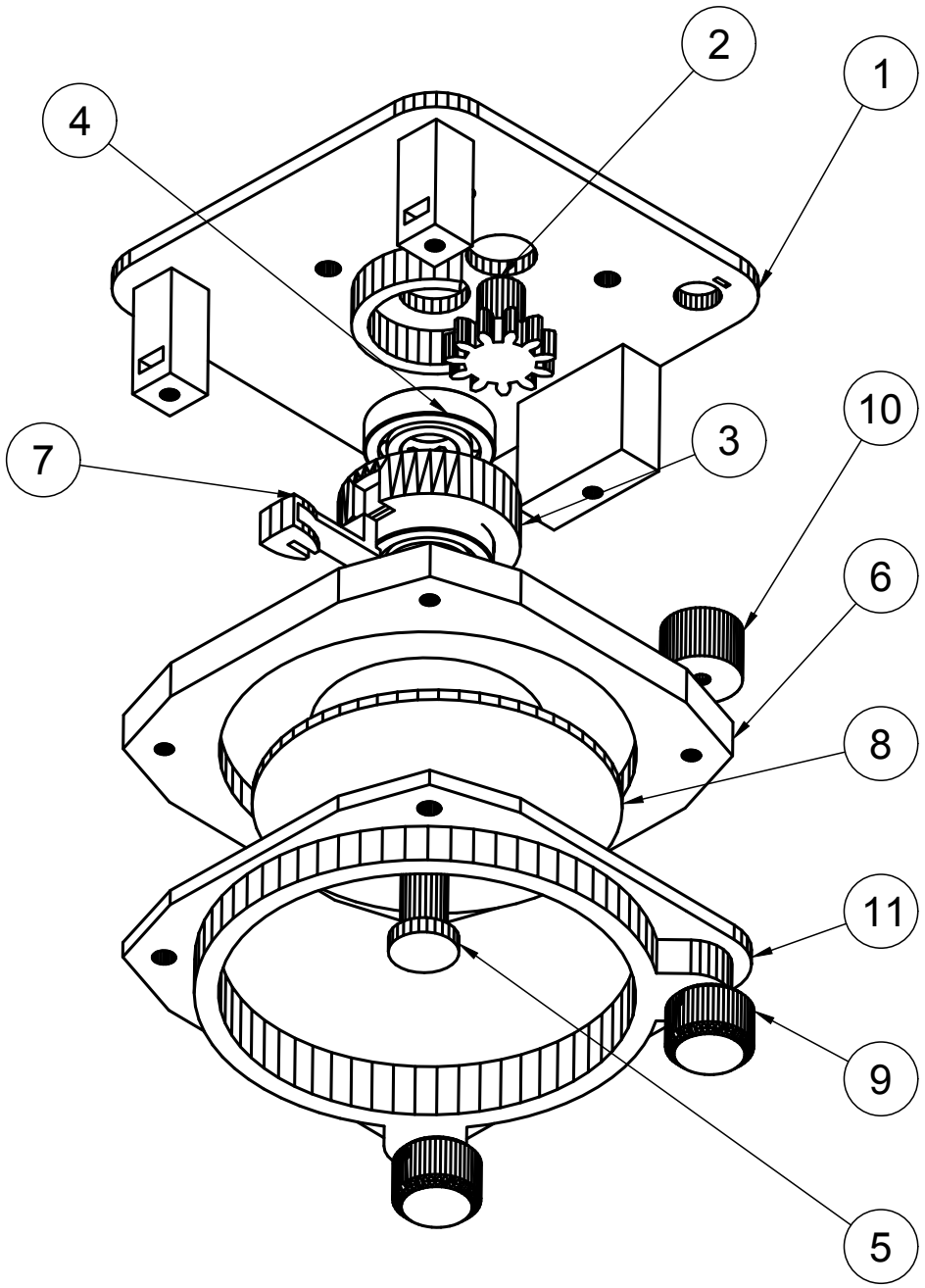




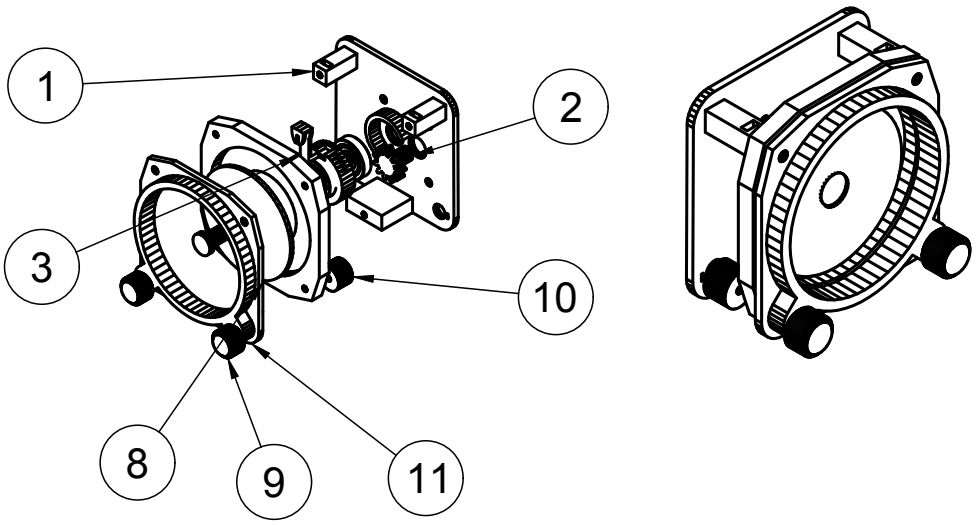
Grado Ing. Electrónica Ind. y A.	Creado por Augusto Samuel Hernández Martín	Fecha 29/11/2018
	Nombre Archivo Soporte servos dobles	Estado Documento Definitivo
	Titulo Soporte de servos Simulador de Vuelo TFG	Escala 1:1
		Rev. 1.0 / Hoja 1/1



Grado Ing. Electrónica Ind. y A.	Creado por Augusto Samuel Hernández Martín	Fecha 29/11/2018
	Nombre Archivo Base trasera indicadores dobles	Estado Documento Definitivo
	Titulo Base trasera Simulador de Vuelo TFG	Escala 1:1
		Rev. 1.0 / Hoja 1/1



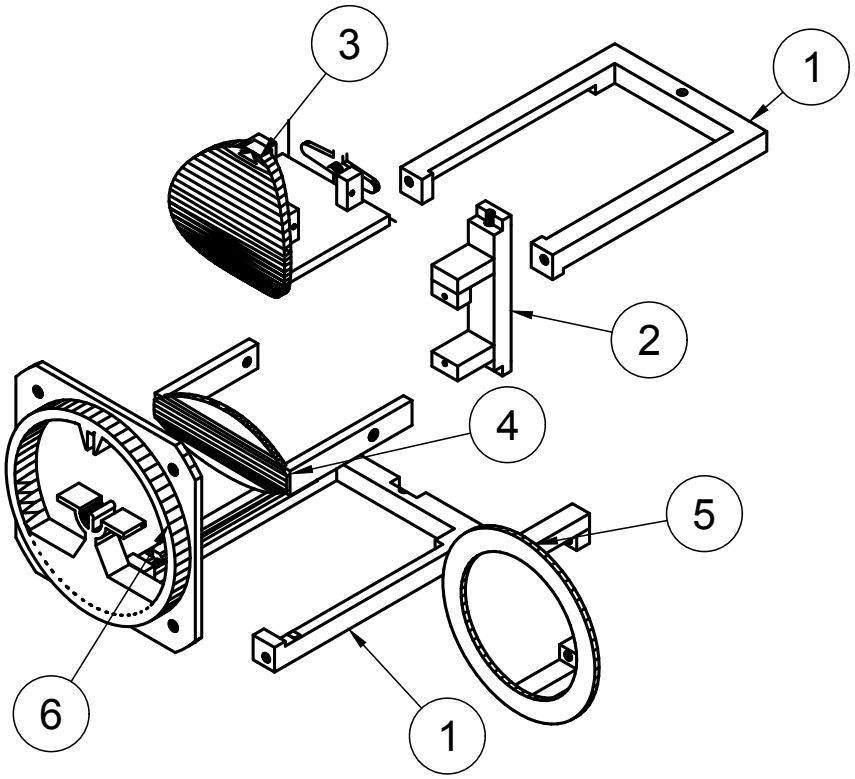
Grado Ing. Electrónica Ind. y A.	Creado por Augusto Samuel Hernández Martín	Fecha 29/11/2018
	Nombre Archivo Indicador de rumbo	Estado Documento Definitivo
	Titulo HDG ensamblaje Simulador de Vuelo TFG	Escala 1:1
		Rev. 1.0 / Hoja 1/2



Item	Qty	Part Number	Description	Material
11	1	Bisel HDG		Steel
10	2	Acople para eje de encoder		Steel
9	2	Boton encoder		Steel
8	1	Rosa		Steel
7	1	Aguja needle		Steel
6	1	Base 2		Steel
5	1	Rosa 2		Steel
4	2	60355K504		Steel
3	1	Engranaje primario		Steel
2	1	Engranaje secundario		Steel
1	1	Base trasera		Steel

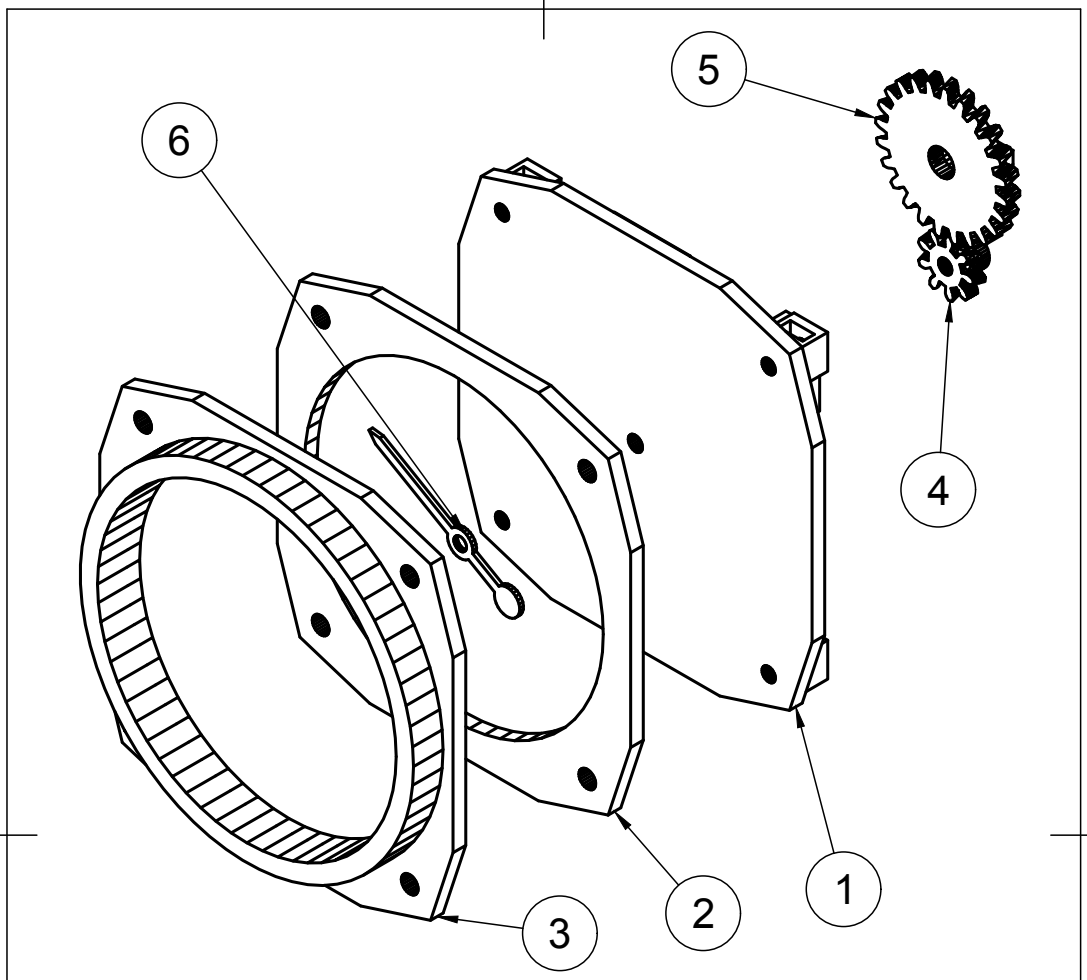
Parts List

Grado: Ing. Electrónica Ind. y A.	Creado por: Augusto Samuel Hernández Martín	Fecha: 29/11/2018
	Nombre Archivo: Indicador de rumbo	Estado Documento: Definitivo
	Titulo: HDG ensamblaje Simulador de Vuelo TFG	Escala: 1:3
		Rev.: 1.0 / Hoja: 2/2



Parts List			
Item	Qty	Part Number	Description
1	2	Pata	Soporte superior e inferior
2	1	Soporte Servo	Sujeción de Servomotor
3	1	Mascara 2	Máscara de movimiento alabeo
4	1	Mascara 1	Máscara de movimiento cabeceo
5	1	Aro	Marco con referencia inclinación
6	1	Frente Horizonte	Carcasa exterior del instrumento

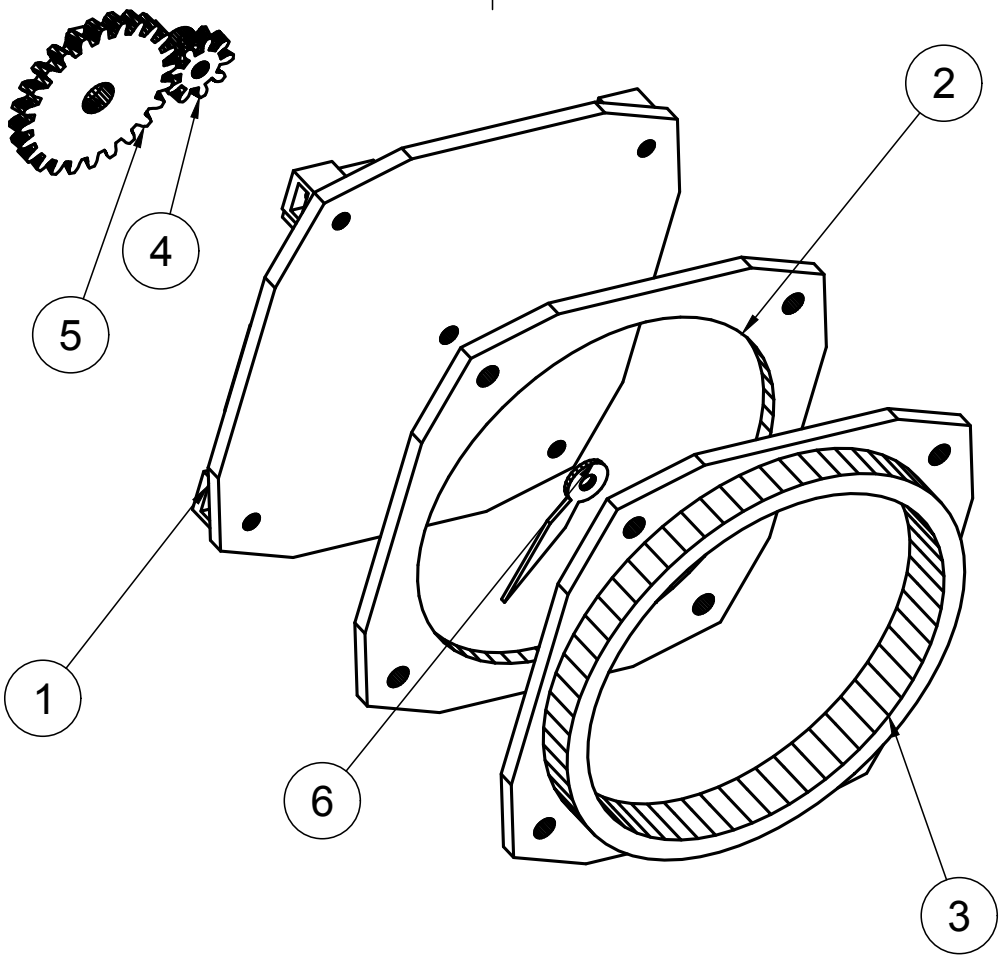
Grado: Ing. Electrónica Ind. y A.	Creado por: Augusto Samuel Hernández Martín	Fecha: 29/11/2018
	Nombre Archivo: Frente_Horizonte	Estado Documento: Definitivo
	Titulo: Ensamblaje Horizonte Art. Simulador de Vuelo TFG	Escala: 1:2
		Rev.: 1.0 / Hoja: 1/1



Item	Qty	Part Number	Description	Material
6	1	Aguja de VS		Steel
5	1	Engranaje Mayor		Steel
4	1	Engranaje Menor		Steel
3	1	Bisel		Steel
2	1	Separador		Steel
1	1	Soporte Servo		Steel

Parts List

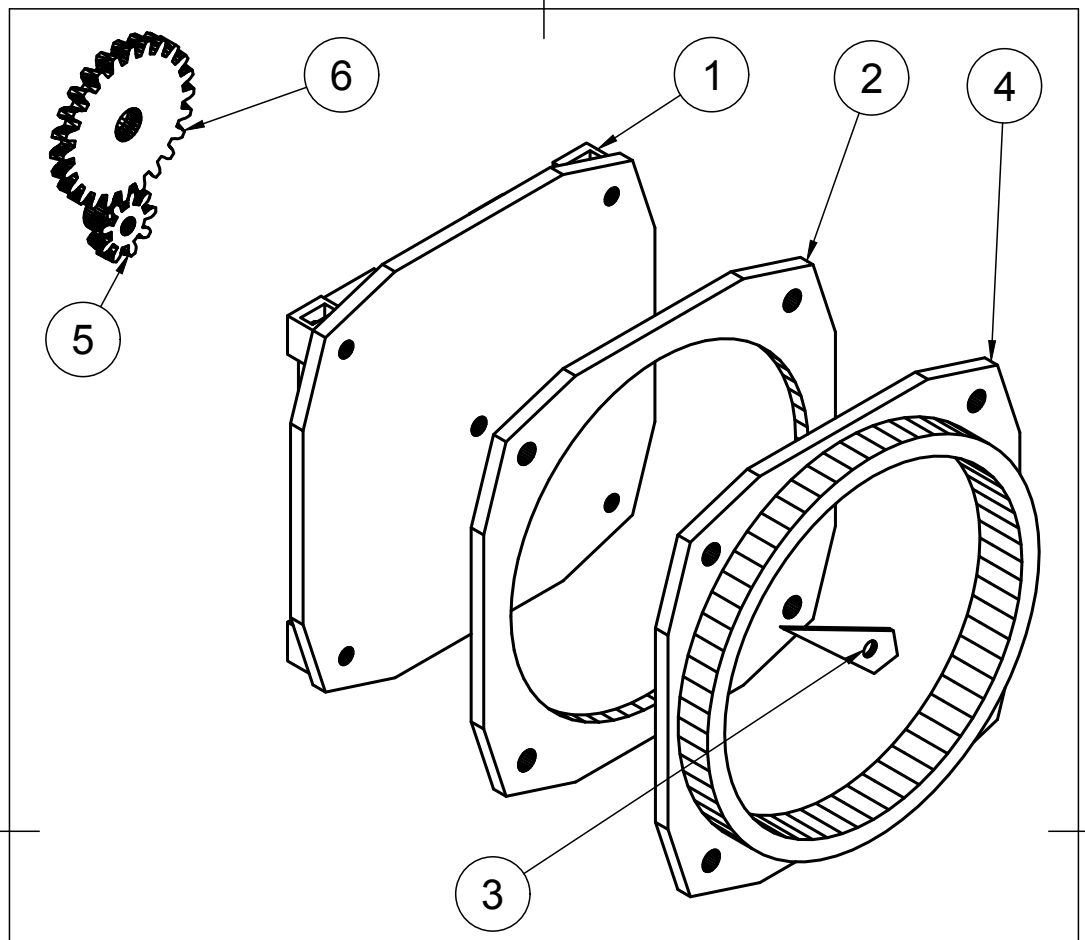
Grado: Ing. Electrónica Ind. y A.	Creado por: Augusto Samuel Hernández Martín	Fecha: 29/11/2018
	Nombre Archivo: AirSpeed	Estado Documento: Definitivo
	Titulo: IAS Ensamblaje Simulador de Vuelo TFG	Escala: 1:1
		Rev.: 1.0 / Hoja: 1/1



Item	Qty	Part Number	Description	Material
6	1	Aguja de RPM		Steel
5	1	Engranaje Mayor		Steel
4	1	Engranaje Menor		Steel
3	1	Bisel		Steel
2	1	Separador		Steel
1	1	Soporte Servo		Steel

Parts List

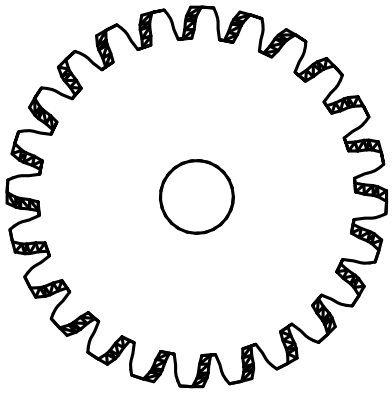
Grado: Ing. Electrónica Ind. y A.	Creado por: Augusto Samuel Hernández Martín	Fecha: 29/11/2018
	Nombre Archivo: Indicador RPM	Estado Documento: Definitivo
	Titulo: RPM Ensamblaje Simulador de Vuelo TFG	Escala: 1:1
		Rev.: 1.0 / Hoja: 1/1

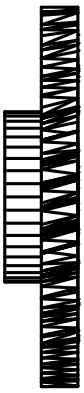


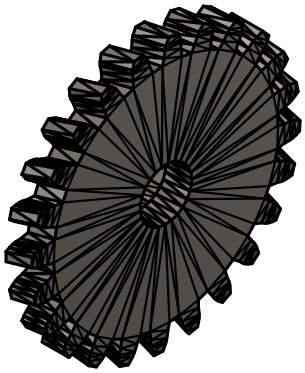
Item	Qty	Part Number	Description	Material
6	1	Engranaje Mayor		Steel
5	1	Engranaje Menor		Steel
4	1	Bisel		Steel
3	1	Aguja IAS		Steel
2	1	Separador		Steel
1	1	Soporte Servo		Steel

Parts List

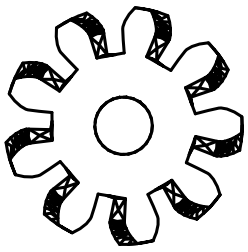
Grado: Ing. Electrónica Ind. y A.	Creado por: Augusto Samuel Hernández Martín	Fecha: 29/11/2018
	Nombre Archivo: Velocidad Vertical	Estado Documento: Definitivo
	Titulo: VS Ensamblaje Simulador de Vuelo TFG	Escala: 1:1
		Rev.: 1.0 / Hoja: 1/1

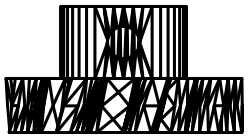


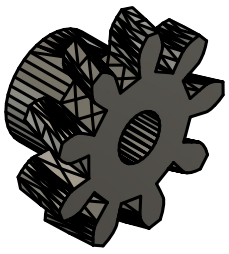




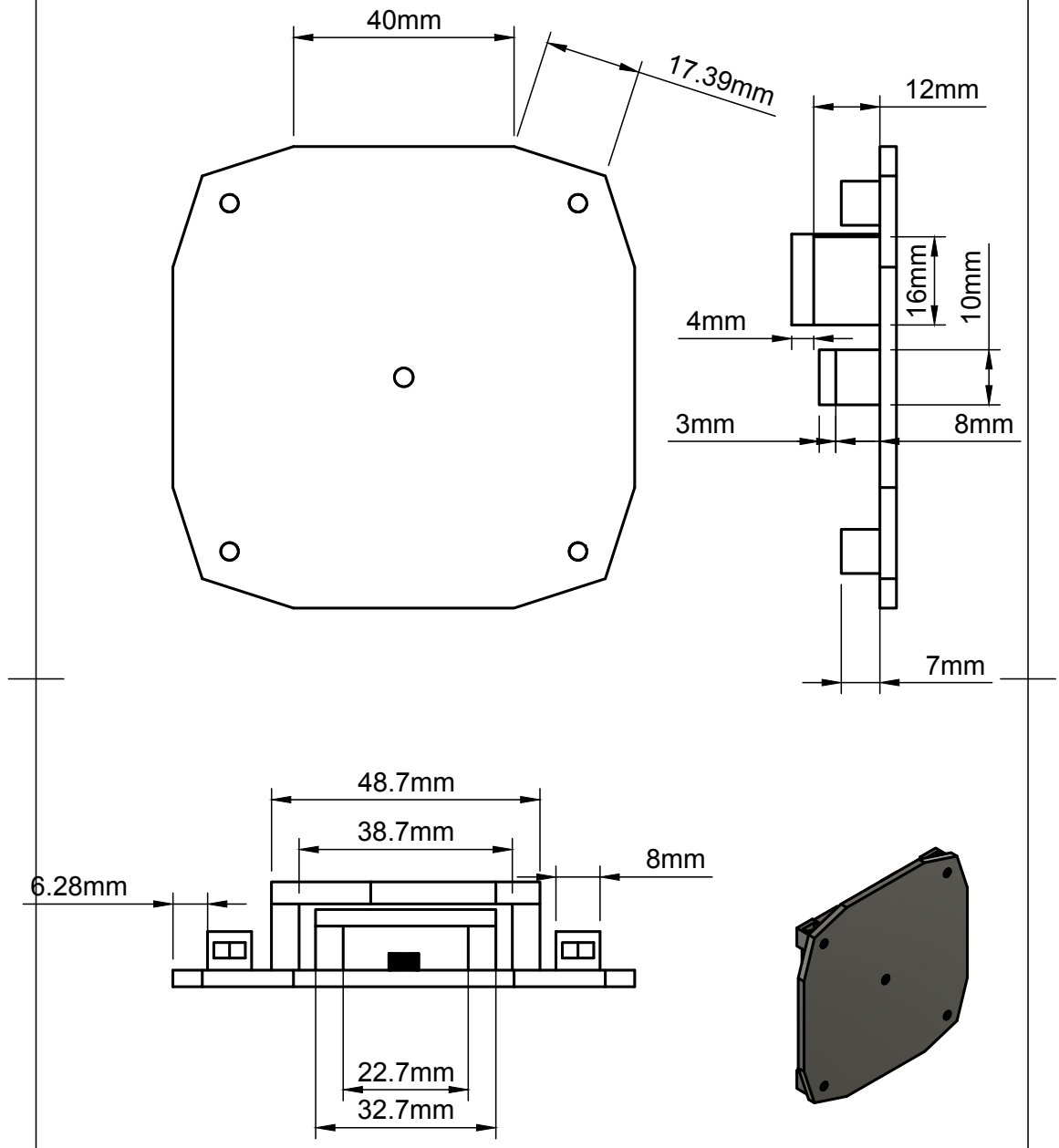
Grado Ing. Electrónica Ind. y A.	Creado por Augusto Samuel Hernández Martín	Fecha 29/11/2018
	Nombre Archivo Engranaje mayor	Estado Documento Definitivo
	Titulo Engranaje Mayor Simulador de Vuelo TFG	Escala 1:1
		Rev. 1.0 / Hoja 1/1



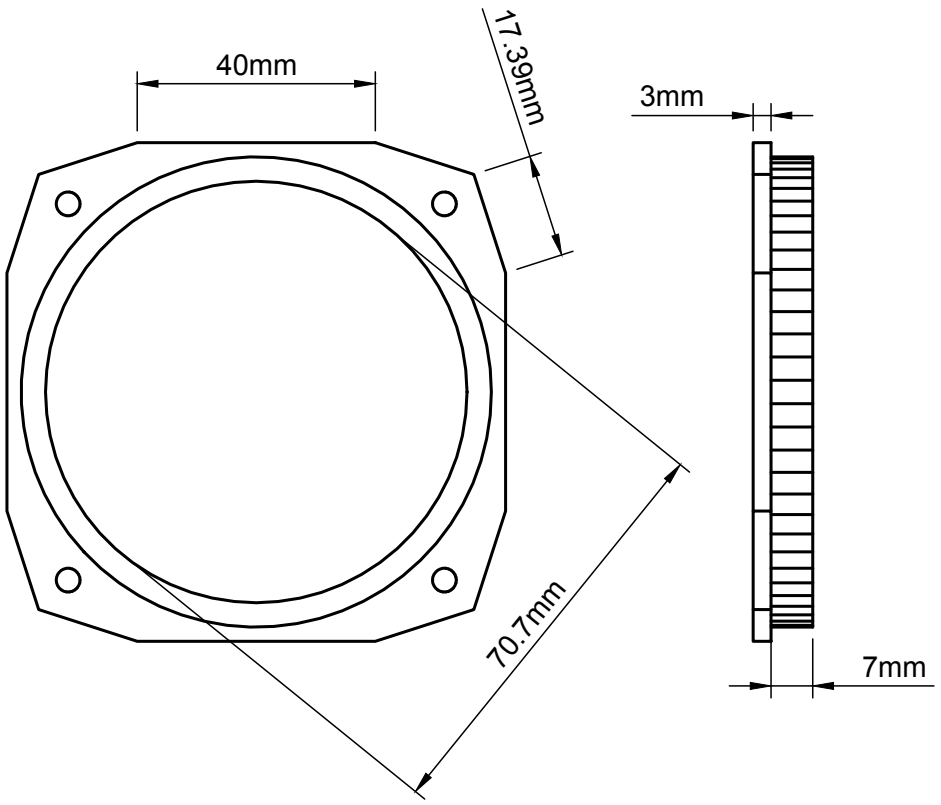


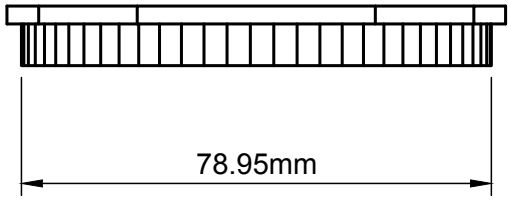


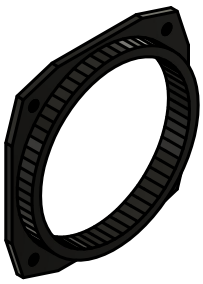
Grado	Creado por	Fecha
Ing. Electrónica Ind. y A.	Augusto Samuel Hernández Martín	29/11/2018
	Nombre Archivo: Engranaje menor	Estado Documento: Definitivo
	Titulo: Engranaje Menor Simulador de Vuelo TFG	Escala: 3:1
		Rev.: 1.0 / Hoja: 1/1



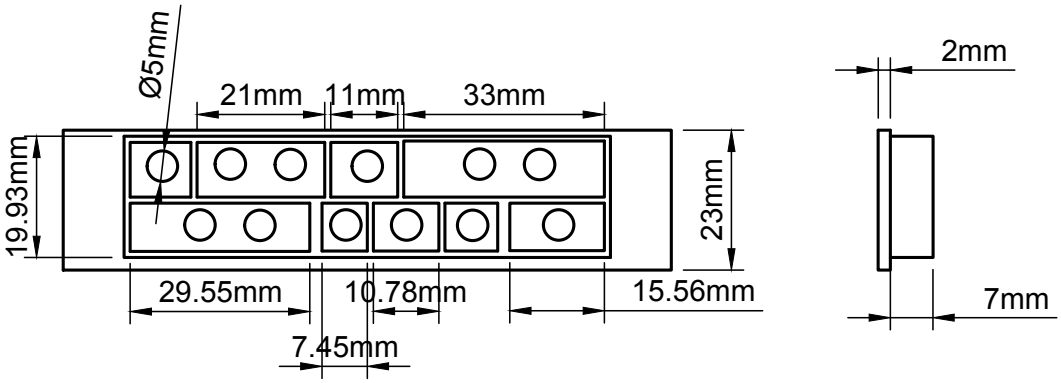
Grado	Creado por	Fecha
Ing. Electrónica Ind. y A.	Augusto Samuel Hernández Martín	29/11/2018
	Nombre Archivo: Sopore de servos con engranaje	Estado Documento: Definitivo
	Titulo: Soporte Servo Simulador de Vuelo TFG	Escala: 1:1
		Rev. 1.0 / Hoja 1/1

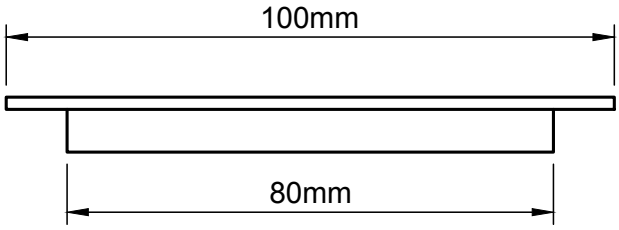


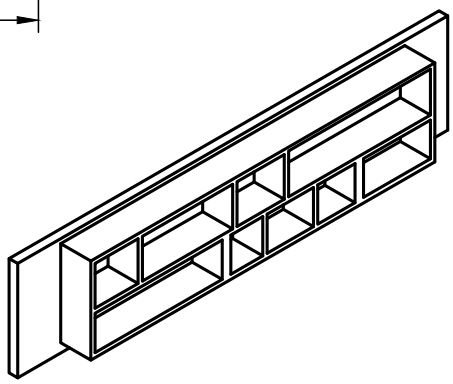




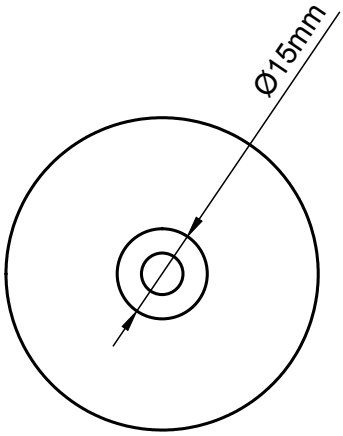
Grado Ing. Electrónica Ind. y A.	Creado por Augusto Samuel Hernández Martín	Fecha 29/11/2018
	Nombre Archivo Bisel insturmentos	Estado Documento Definitivo
	Titulo Bisel Simulador de Vuelo TFG	Escala 1:1 Rev. 1.0 / Hoja 1/1

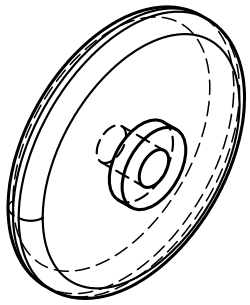


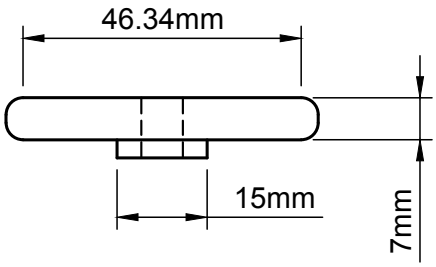




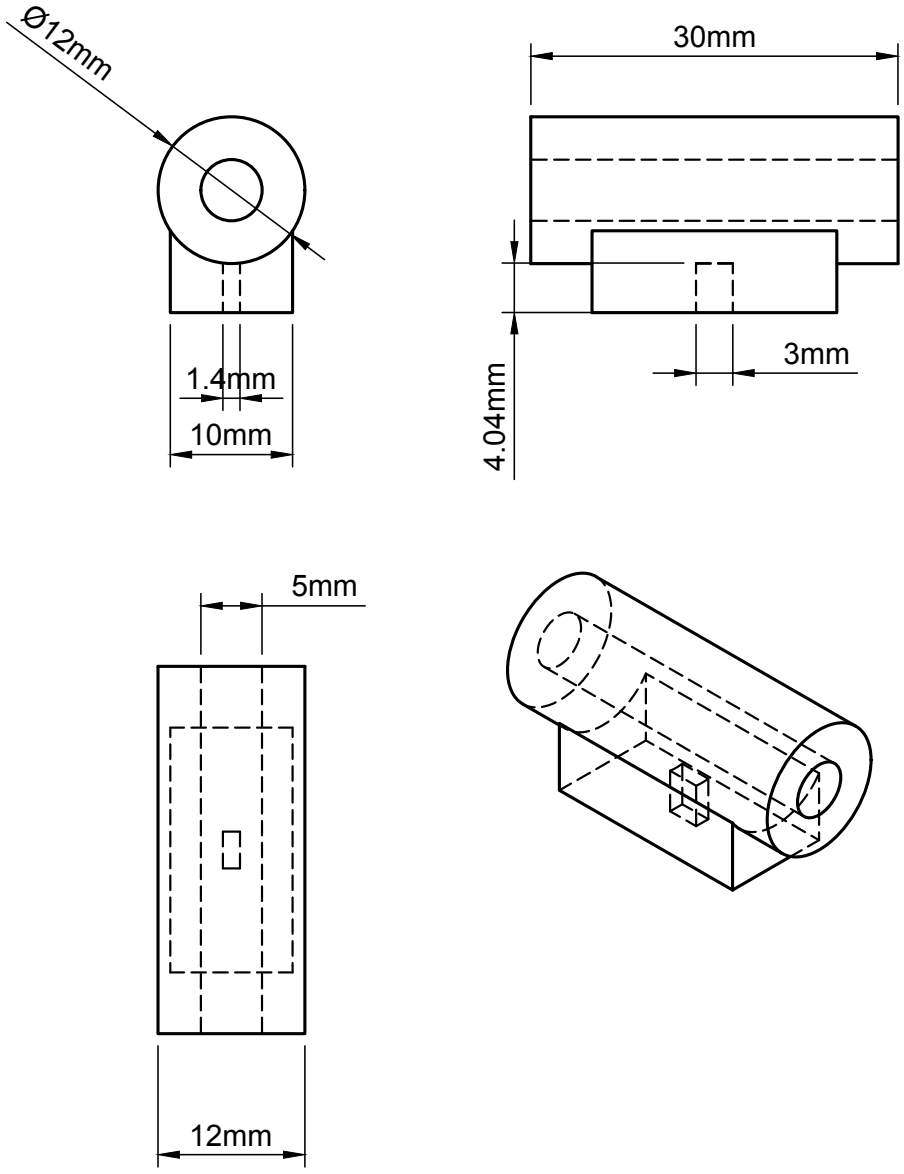
Grado: Ing. Electrónica Ind. y A.	Creado por: Augusto Samuel Hernández Martín	Fecha: 29/11/2018
	Nombre Archivo: Soporte de Leds de avisos	Estado Documento: Definitivo
	Titulo: Avisos Simulador de Vuelo TFG	Escala: 1:1
		Rev.: 1.0 / Hoja: 1/1



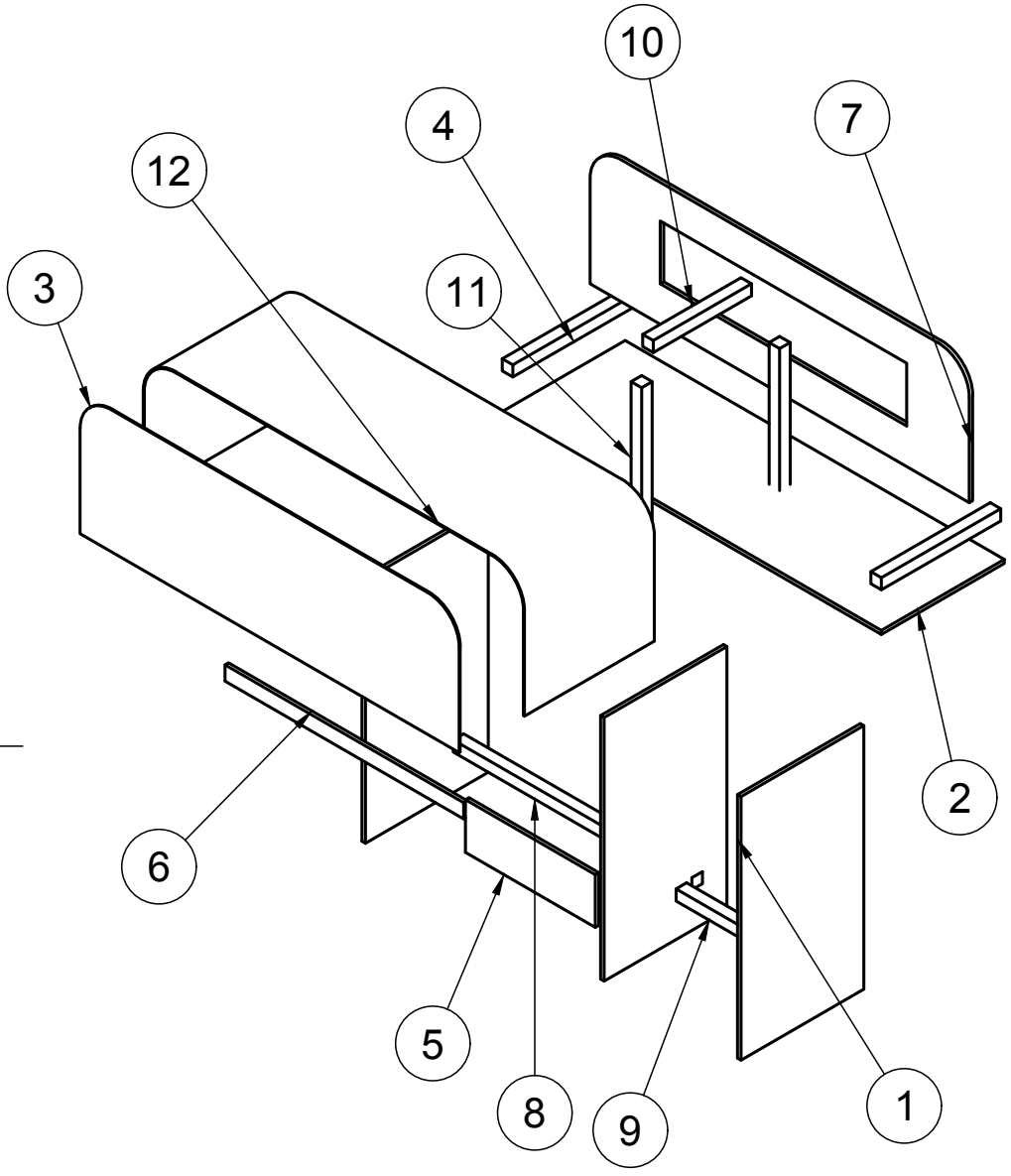




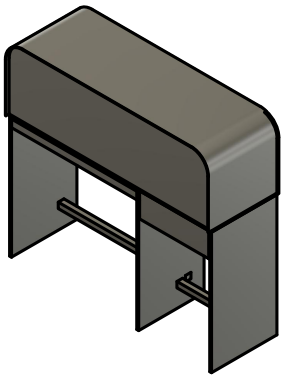
Grado	Creado por	Fecha
Ing. Electrónica Ind. y A.	Augusto Samuel Hernández Martín	29/11/2018
	Nombre Archivo: Rueda Trim	Estado Documento: Definitivo
	Titulo: Rueda_Trim Simulador de Vuelo TFG	Escala: 1:1
		Rev. 1.0 / Hoja 1/1

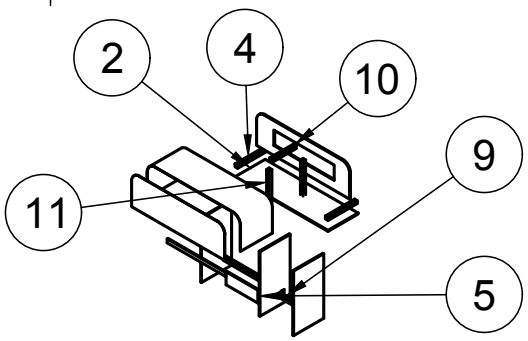


Grado Ing. Electrónica Ind. y A.	Creado por Augusto Samuel Hernández Martín	Fecha 29/11/2018
	Nombre Archivo Usillo Potenciometro Gases	Estado Documento Definitivo
	Titulo Union de eje con Potenciometro Simulador de Vuelo TFG	Escala 2:1 Rev. 1.0 / Hoja 1/1



Grado Ing. Electrónica Ind. y A.	Creado por Augusto Samuel Hernández Martín	Fecha 10/06/2019
	Nombre Archivo Cabina	Estado Documento Definitivo
	Titulo Ensamblaje Cabina Simulador de Vuelo TFG	Escala 1:12
		Rev. 1.0 / Hoja 1/2

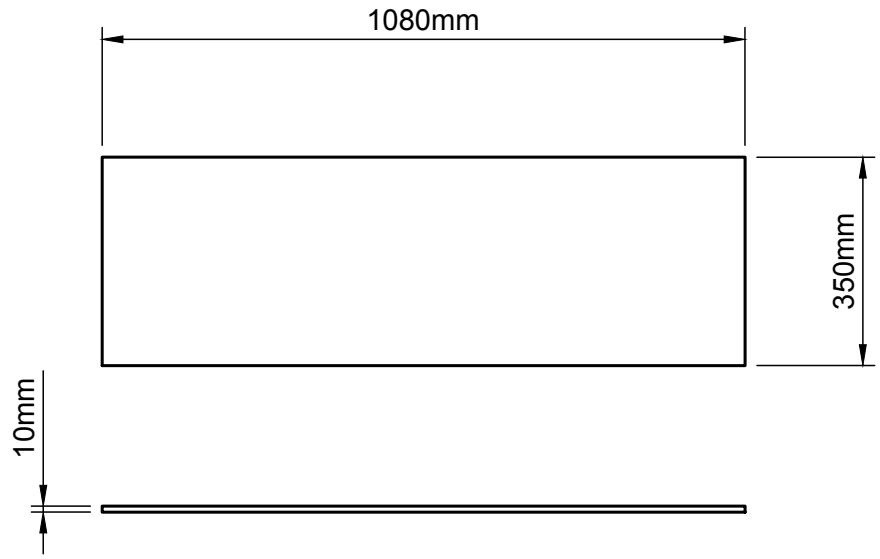


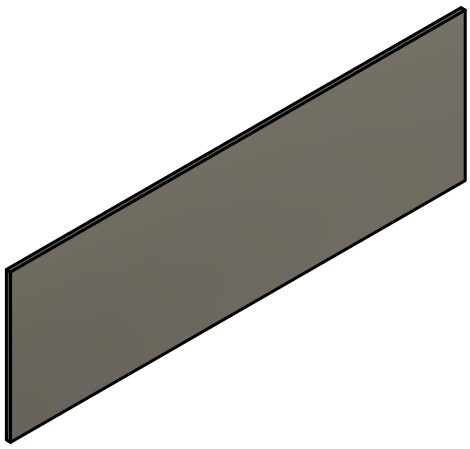


Item	Qty	Part Number	Description	Material
12	1	Tapa superior salpicadero		Steel
11	2	Refuerzo central salpicadero		Steel
10	1	Refuerzo superior central salpicadero		Steel
9	1	Refuerzo inferior drcho		Steel
8	1	Refuerzo inferior izdo		Steel
7	1	Trasero Salpicadero		Steel
6	1	Frontal inferior izdo		Steel
5	1	Frontal inferior drcho		Steel
4	2	Refuerzo superiores laterales salpicadero		Steel
3	1	Frontal salpicadero		Steel
2	1	Base salpicadero		Steel
1	3	Soportes laterales		Steel

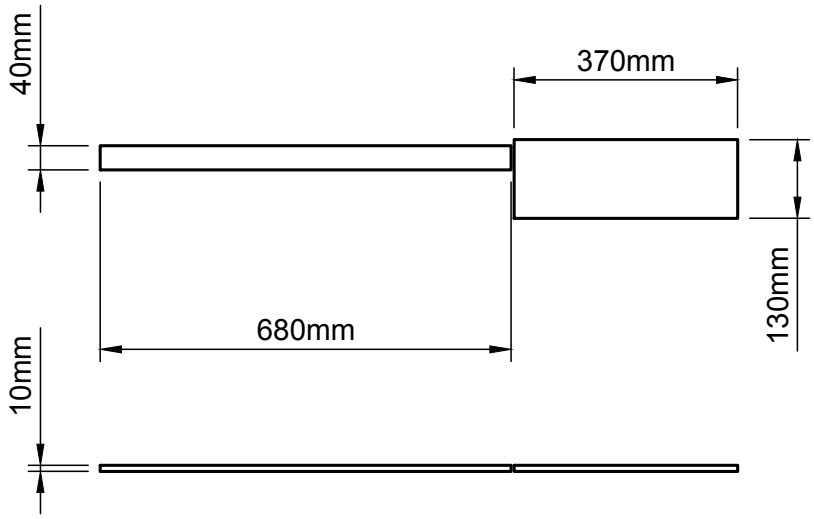
Parts List

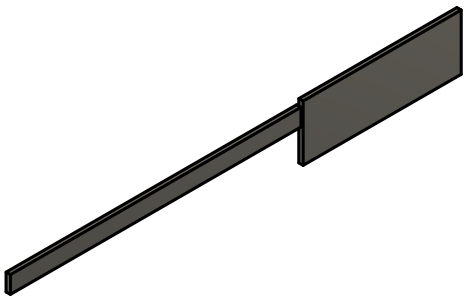
Grado: Ing. Electrónica Ind. y A.	Creado por: Augusto Samuel Hernández Martín	Fecha: 10/06/2019
	Nombre Archivo: Cabina	Estado Documento: Definitivo
	Titulo: Ensamblaje Cabina Simulador de Vuelo TFG	Escala: 1:50
		Rev.: 1.0 / Hoja: 2/2



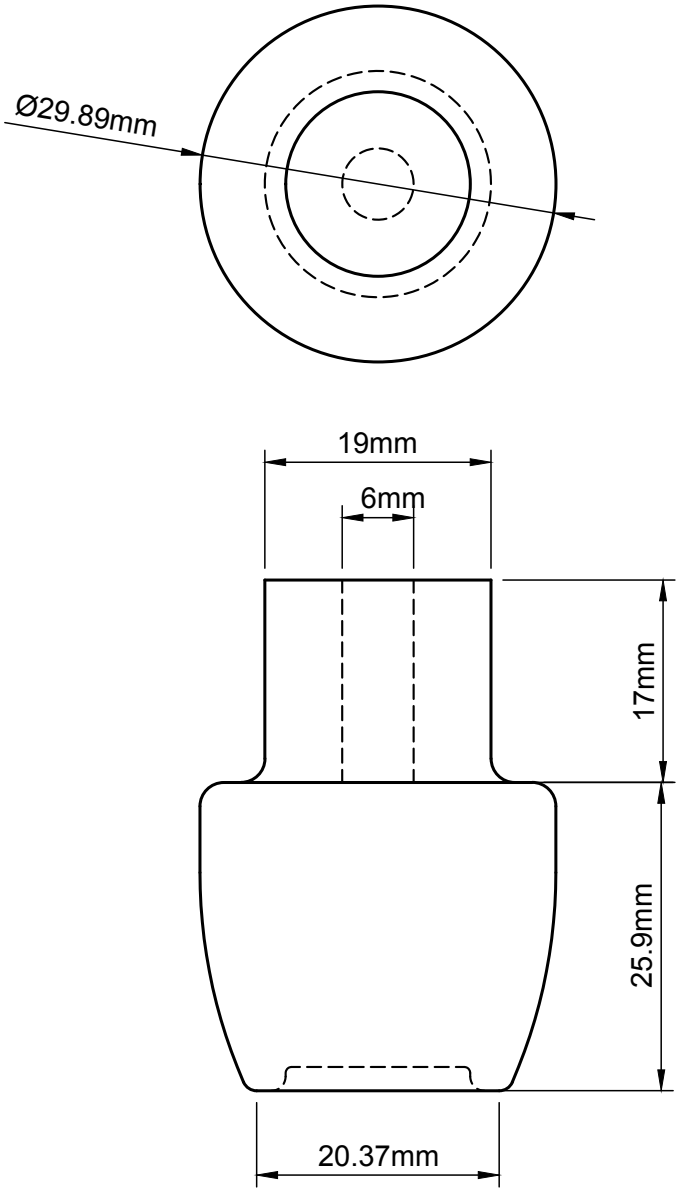


Grado Ing. Electrónica Ind. y A.	Creado por Augusto Samuel Hernández Martín	Fecha 11/06/2019
	Nombre Archivo Base salpicadero	Estado Documento Definitivo
	Titulo Ensamblaje Cabina Simulador de Vuelo TFG	Escala 1:10
		Rev. 1.0 / Hoja 1/1

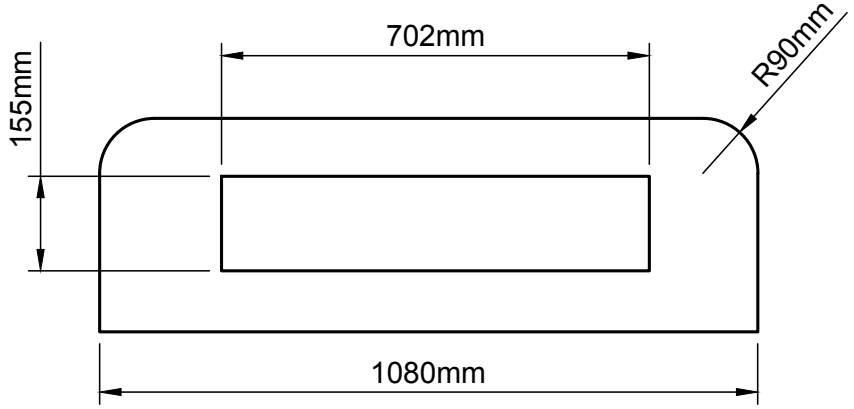


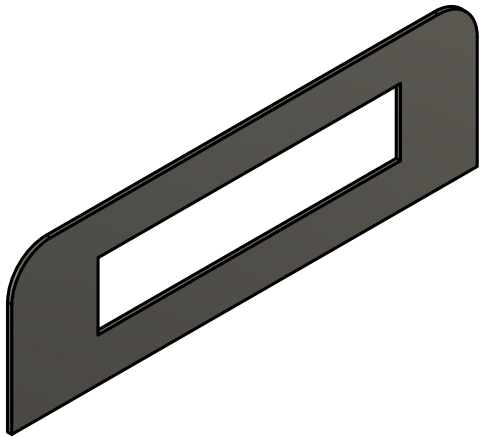


Grado Ing. Electrónica Ind. y A.	Creado por Augusto Samuel Hernández Martín	Fecha 11/06/2019
	Nombre Archivo Frontales inferiores	Estado Documento Definitivo
	Titulo Ensamblaje Cabina Simulador de Vuelo TFG	Escala 1:10
		Rev. 1.0 / Hoja 1/1

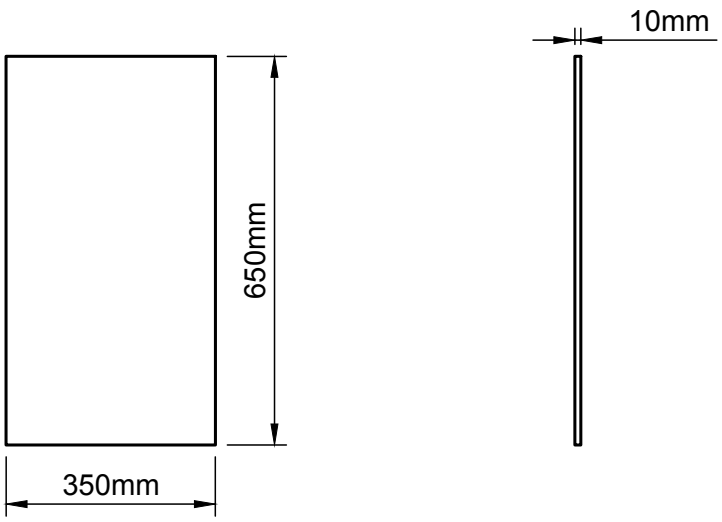


Grado Ing. Electrónica Ind. y A.	Creado por Augusto Samuel Hernández Martín	Fecha 27/12/2018
	Nombre Archivo Tirador de gases	Estado Documento Definitivo
	Titulo Palanca de mezcla Simulador de Vuelo TFG	Escala 2:1
		Rev. 1.0 / Hoja 1/1

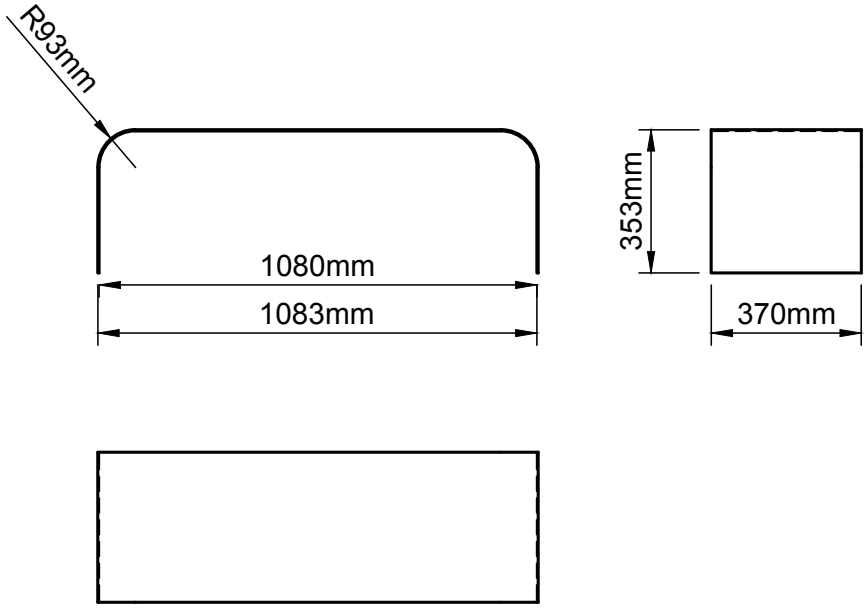


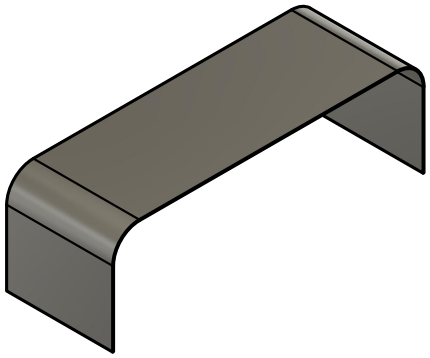


Grado Ing. Electrónica Ind. y A.	Creado por Augusto Samuel Hernández Martín	Fecha 11/06/2019
	Nombre Archivo Frente_Horizonte	Estado Documento Definitivo
	Titulo Ensamblaje Cabina Simulador de Vuelo TFG	Escala 1:10
		Rev. 1.0 / Hoja 1/1

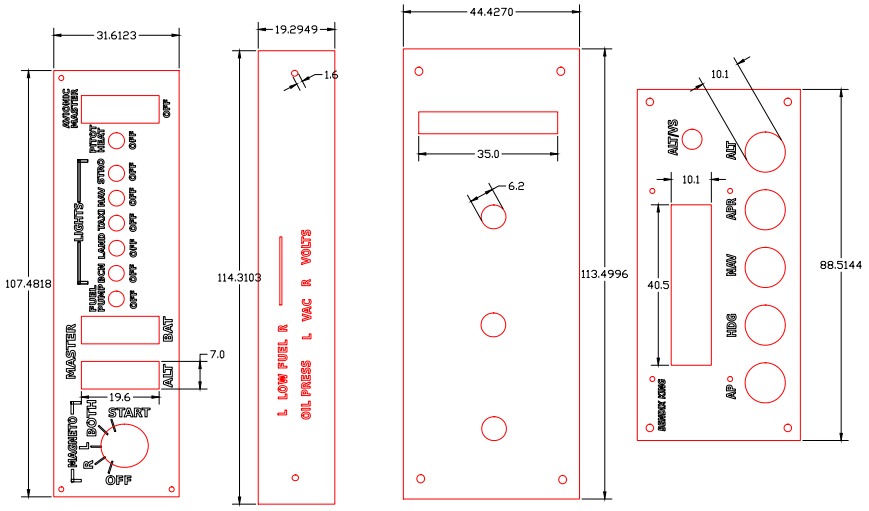


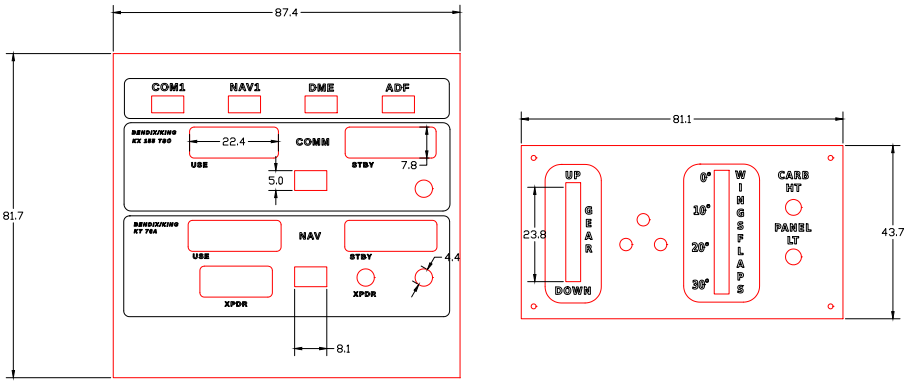
Grado Ing. Electrónica Ind. y A.	Creado por Augusto Samuel Hernández Martín	Fecha 11/06/2019
	Nombre Archivo Soportes laterales	Estado Documento Definitivo
	Titulo Ensamblaje Cabina Simulador de Vuelo TFG	Escala 1:10
		Rev. 1.0 / Hoja 1/1



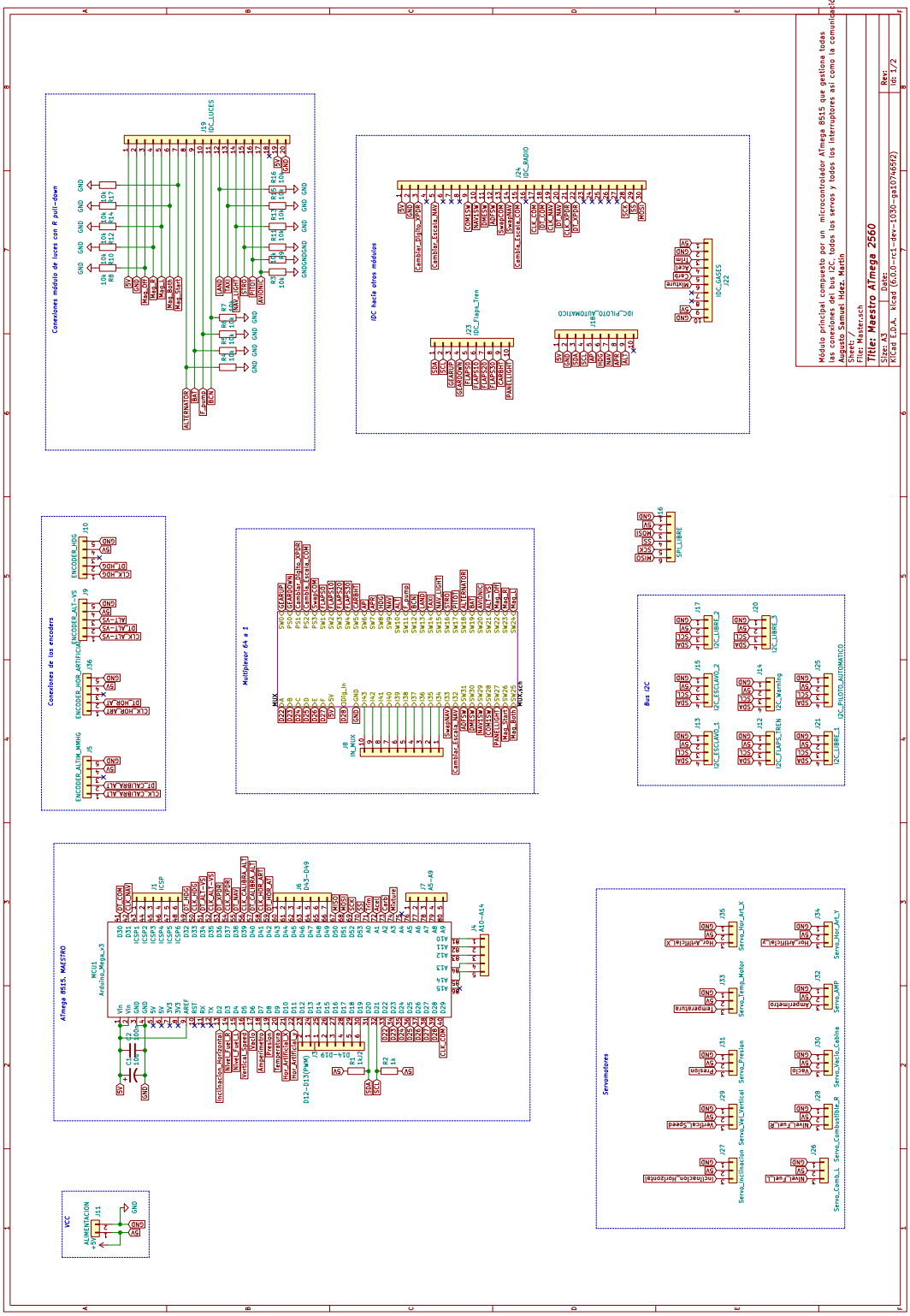


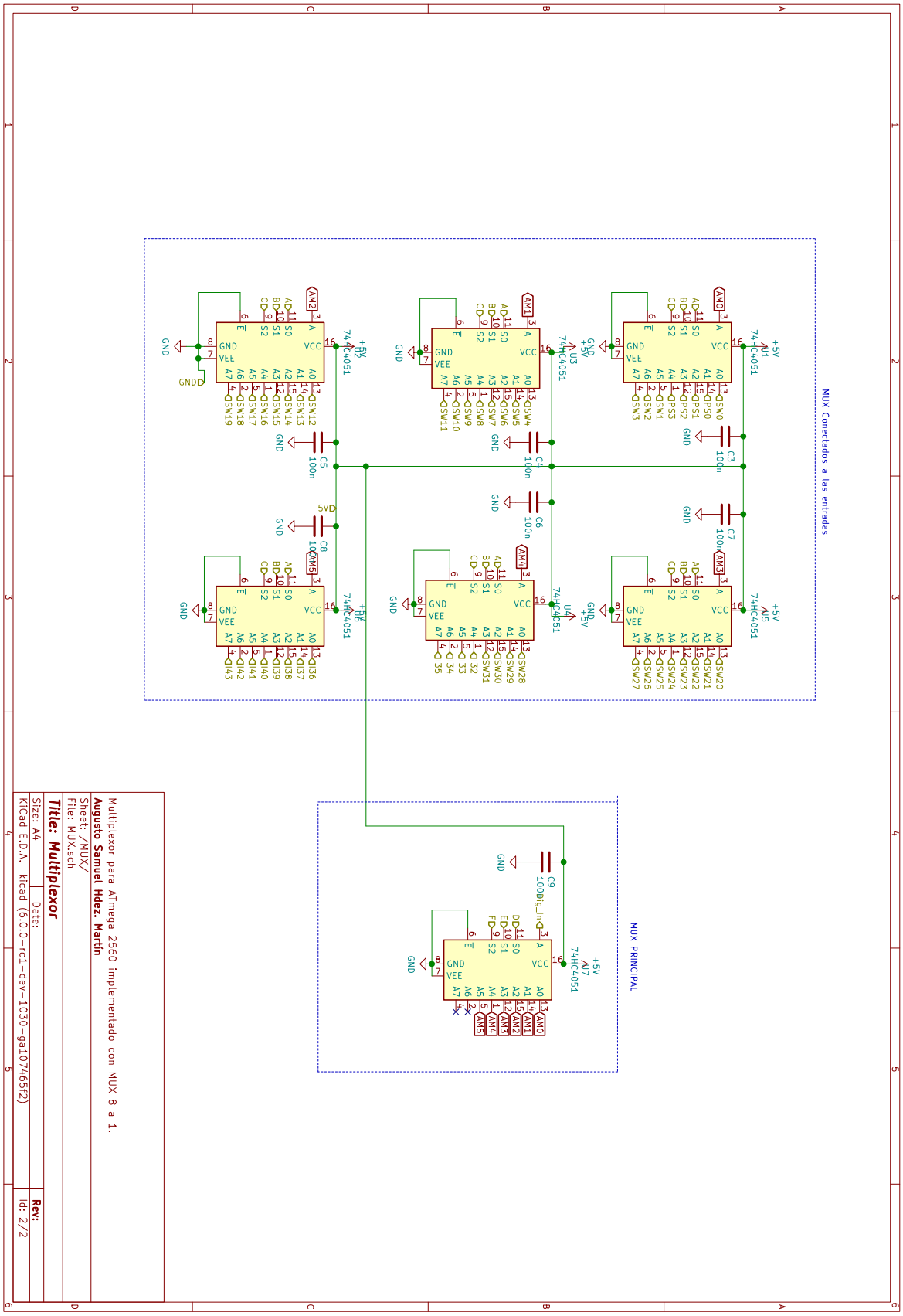
Grado Ing. Electrónica Ind. y A.	Creado por Augusto Samuel Hernández Martín	Fecha 11/06/2019
	Nombre Archivo Tapa superior salpicadero	Estado Documento Definitivo
	Titulo Ensamblaje Cabina Simulador de Vuelo TFG	Escala 1:15
		Rev. 1.0 / Hoja 1/1

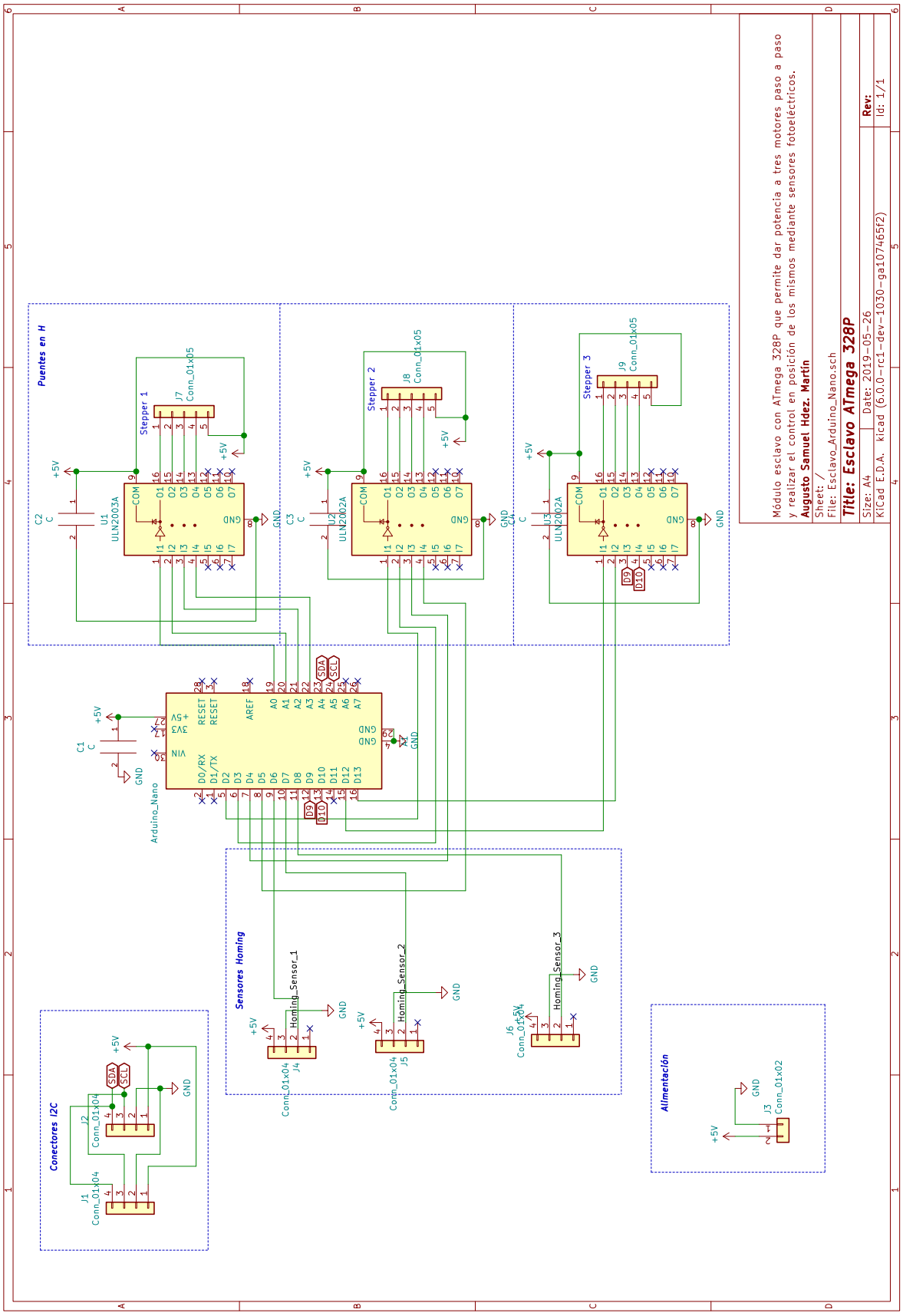

31.6123
19.2949
44.4270
107.4818
114.3103
113.4996
88.5144
35.0
6.2
1.6
10.1
40.5
7.0
19.6
MASTER
BAT
ALT
START
BOTH
OFF
L LOW FUEL R
OIL PRESS L VAC R VOLTS
ACTIVS
ALT
APR
NAV
HDG
AP

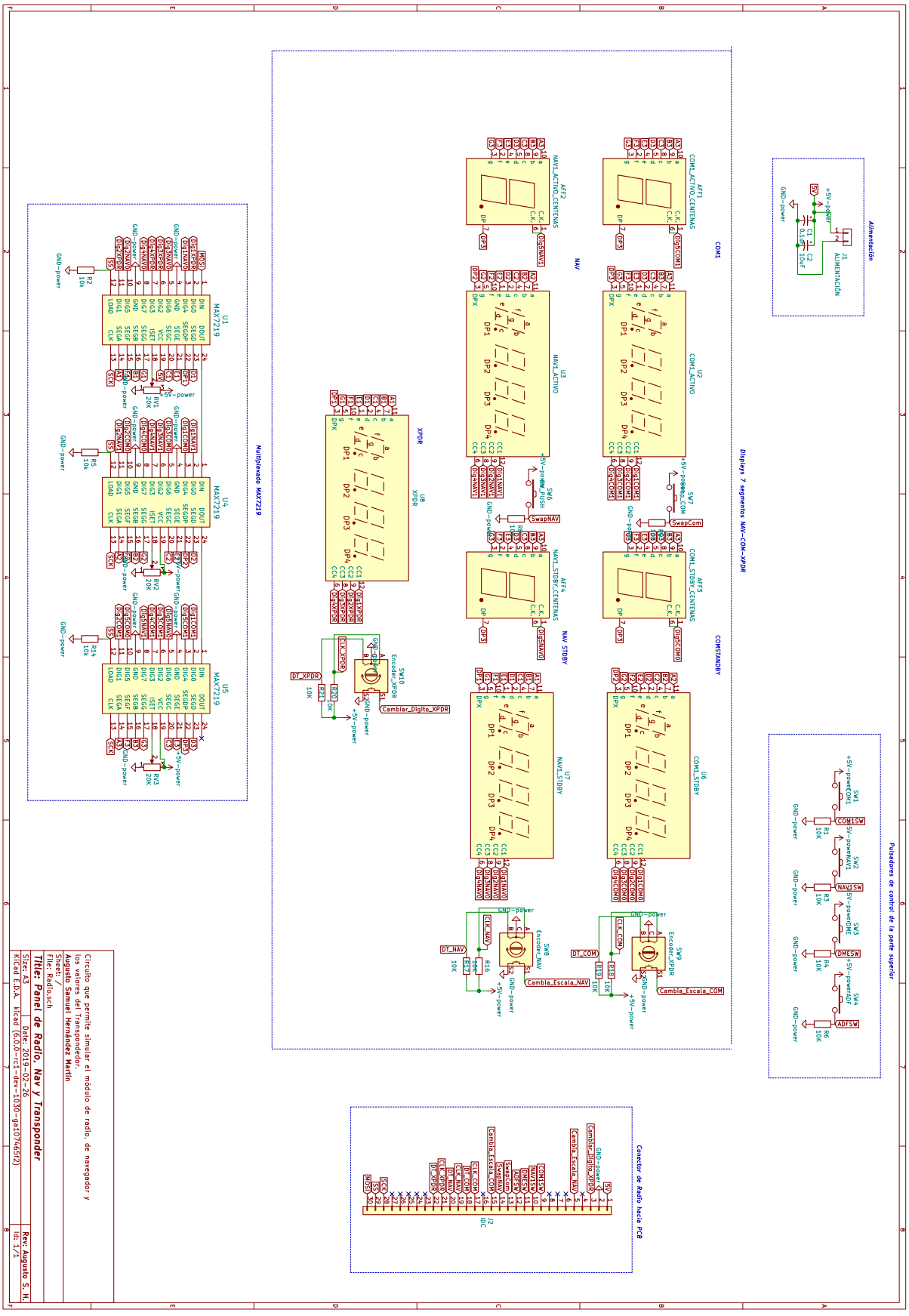

87.4
81.7
COM1
NAV1
DME
ADF
22.4
COMM
5.0
7.8
USE
STBY
NAV
4.4
XPDR
8.1
81.1
23.8
43.7
UP
DOWN
GEAR
0°
10°
20°
30°
WINGFLAPS
CARB HT
PANEL LT

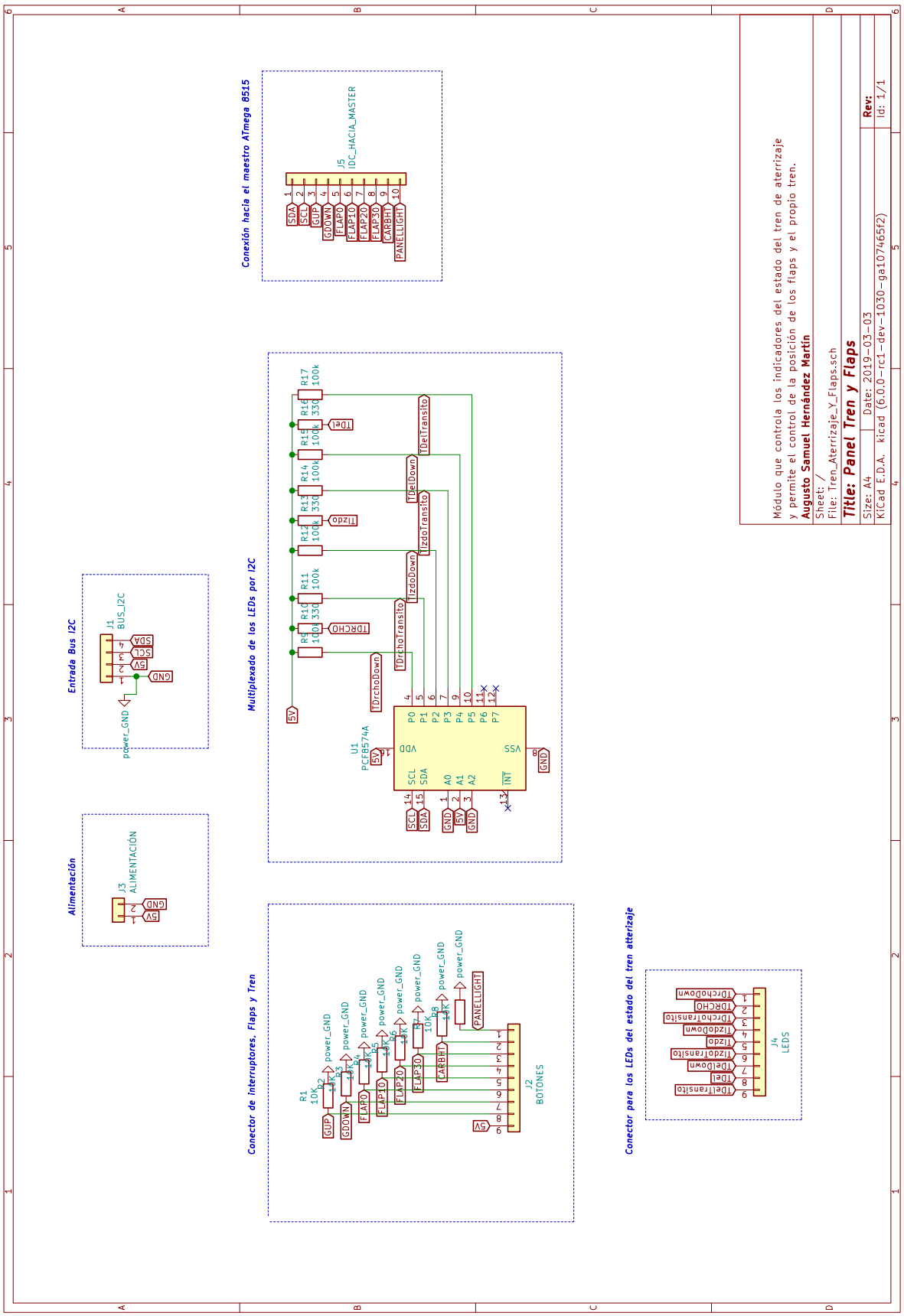
Esquemas electrónicos

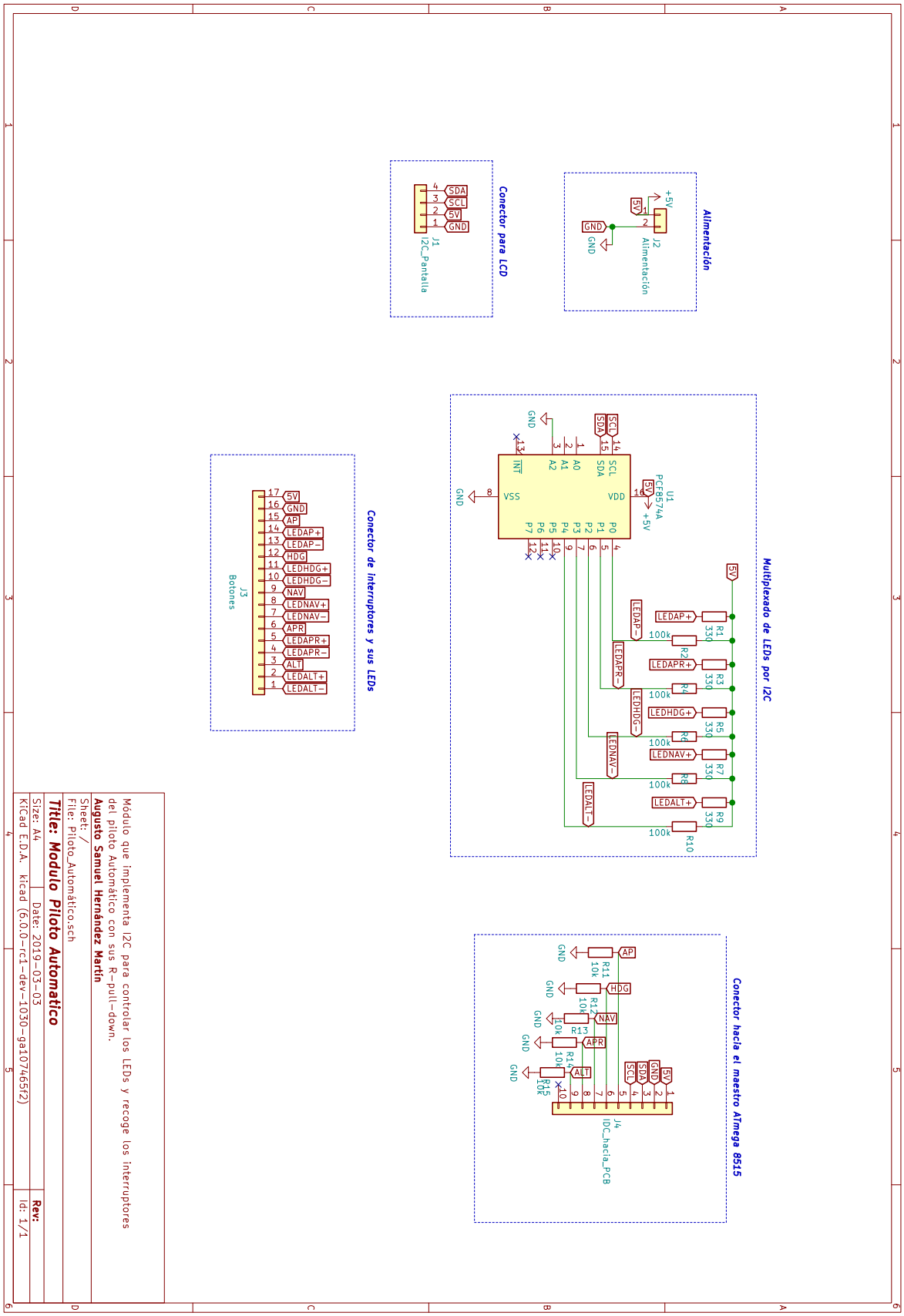


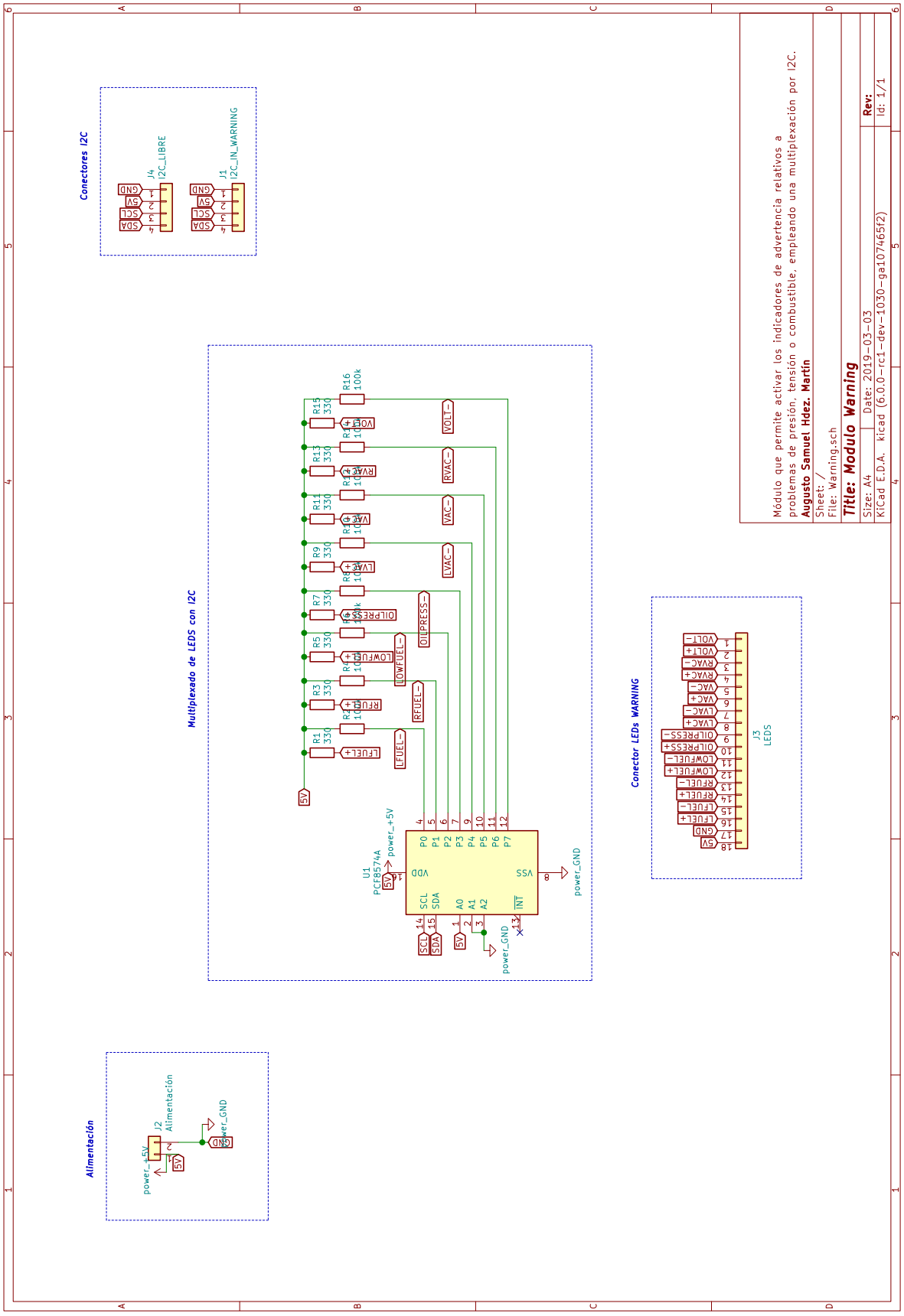

MUX Conectados a las entradas
MUX PRINCIPAL
Multiplexor para ATmega 2560 implementado con MUX 8 a 1.
Augusto Samuel Hdez. Martín
Sheet: /MUX/
File: MUX.sch
Title: Multiplexor
Size: A4
Date:
KiCad E.D.A. kicad (6.0.0-rc1-dev-1030-ga107465f2)
Rev:
Id: 2/2

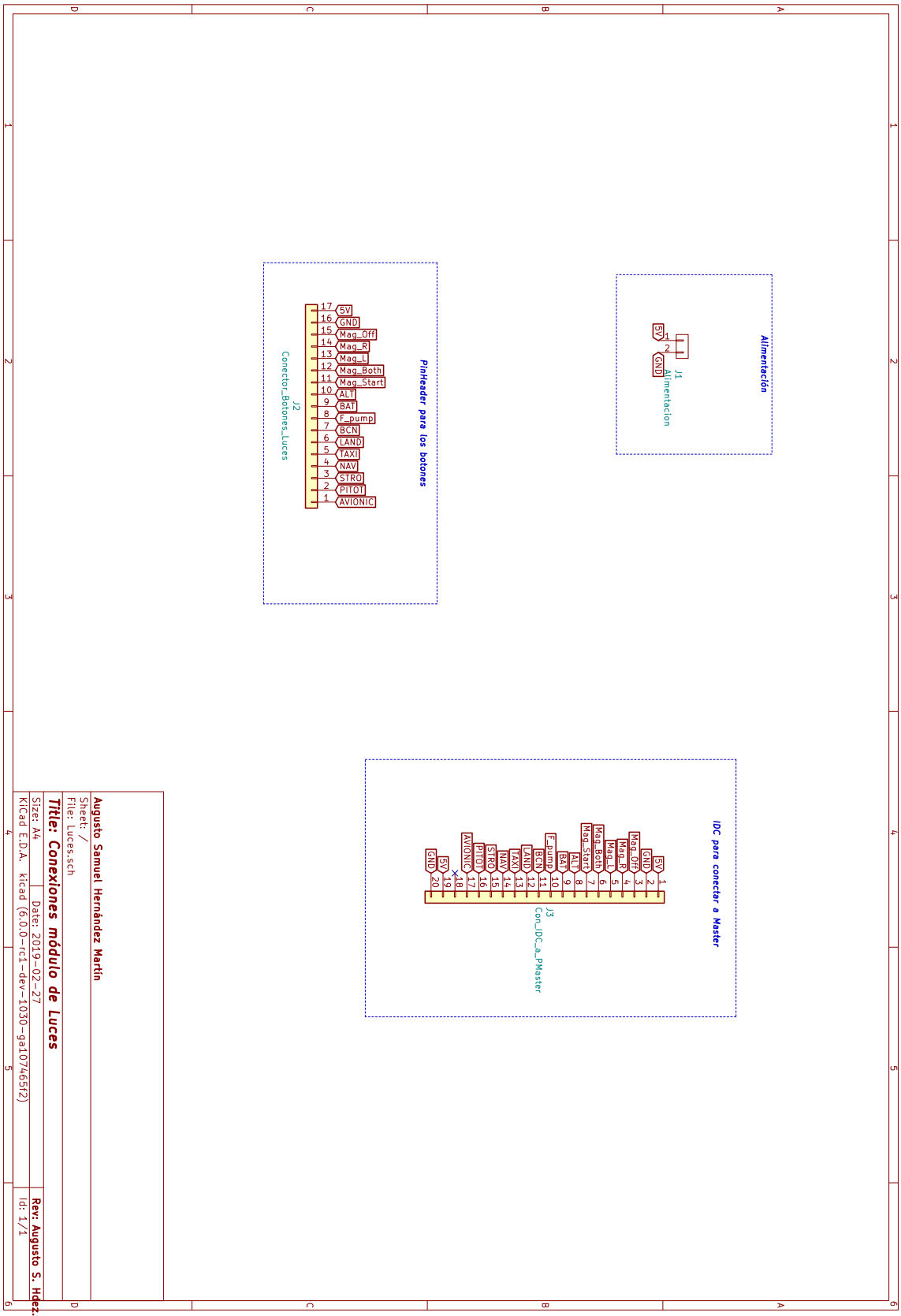

Módulo esclavo con ATmega 328P que permite dar potencia a tres motores paso a paso y realizar el control en posición de los mismos mediante sensores fotoeléctricos.
Augusto Samuel Hdez. Martín
Sheet: /
File: Esclavo_Arduino_Nano.sch
Title: Esclavo ATmega 328P
Size: A4
Date: 2019-05-26
Rev:
KiCad E.D.A. kicad (6.0.0-rc1-dev-1030-ga1074f5f2)
Id: 1/1
Puentes en H
Stepper 1
Stepper 2
Stepper 3
ULN2003A
ULN2002A
Arduino_Nano
Sensores Homing
Homing_Sensor_1
Homing_Sensor_2
Homing_Sensor_3
Conectores I2C
Alimentación
+5V
GND
SDA
SCL

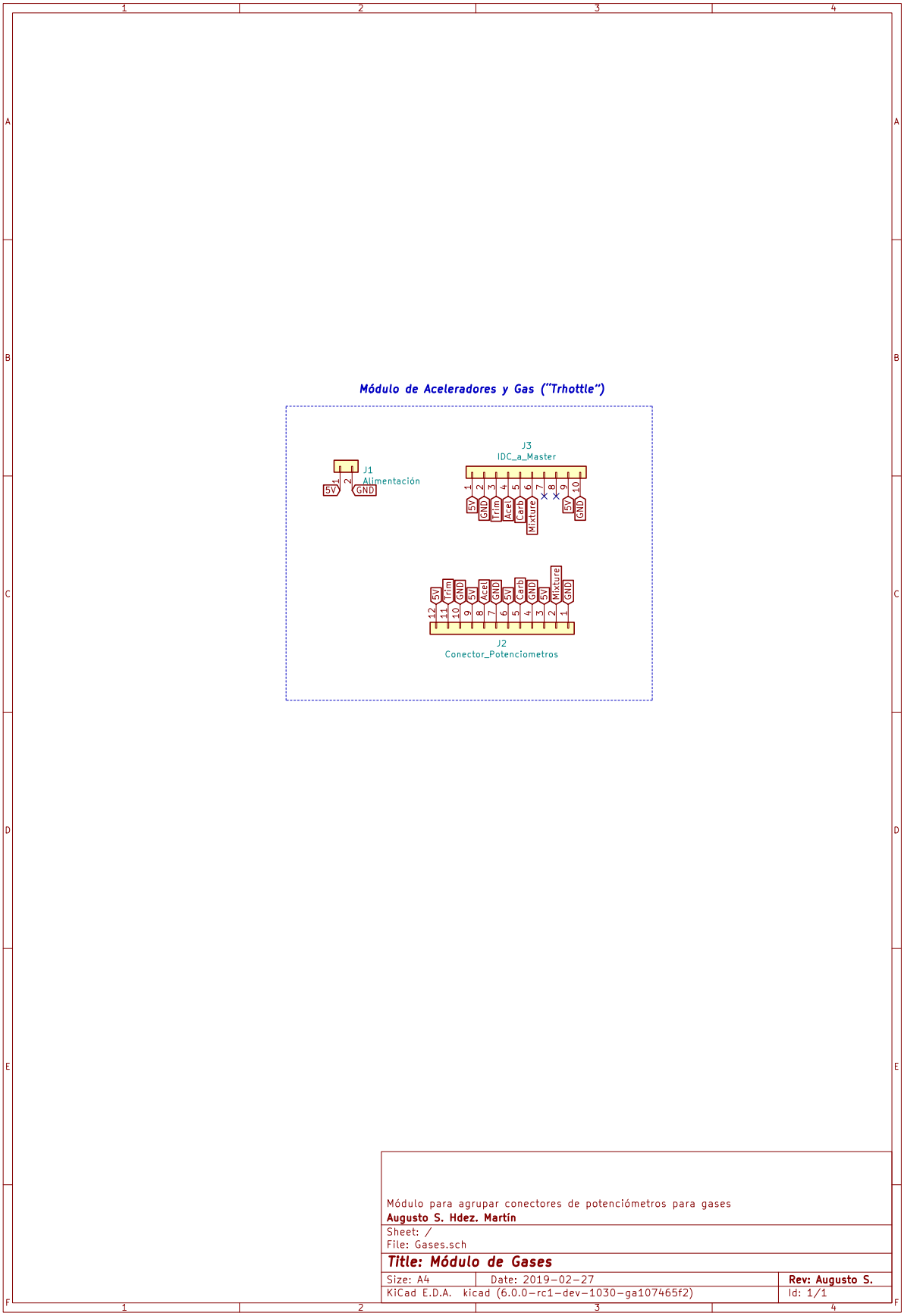

Alimentación
Displays 7 segmentos NAV-COM-XPDR
Pulsadores de control de la parte superior
Multiplexado MAX7219
Conector de Radio hacia PCB
COM1
NAV
COM1STANDBY
NAV STDBY
XPDR
Circuito que permite simular el módulo de radio de navegador y los valores del Transpondedor.
Augusto Samuel Hernández Martín
Sheet: /
File: Radio.sch
Title: Panel de Radio, Nav y Transponder
Size: A3
Date: 2019-02-26
KiCad E.D.A. kicad (6.0.0-rc1-dev-1030-ga107f65f2)
Rev: Augusto S. H.
Id: 1/1


Conexión hacia el maestro ATmega 8515
J5
IDC_HACIA_MASTER
SDA
SCL
GUP
GDOWN
FLAP0
FLAP10
FLAP20
FLAP30
CARBHT
PANELLIGHT
Multiplexado de los LEDs por I2C
U1
PCF8574A
VDD
VSS
SCL
SDA
A0
A1
A2
INT
P0
P1
P2
P3
P4
P5
P6
P7
5V
GND
TDrchoDown
TDrchoTransito
TDelDown
TDelTransito
TIzdoDown
TIzdoTransito
TDRCHO
TIzdo
TDel
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15
R16
R17
10K
100k
330
Entrada Bus I2C
J1
BUS_I2C
power_GND
Alimentación
J3
ALIMENTACIÓN
Conector de interruptores, Flaps y Tren
J2
BOTONES
power_GND
Conector para los LEDs del estado del tren atterizaje
J4
LEDS
Módulo que controla los indicadores del estado del tren de aterrizaje
y permite el control de la posición de los flaps y el propio tren.
Augusto Samuel Hernández Martín
Sheet: /
File: Tren_Aterrizaje_Y_Flaps.sch
Title: Panel Tren y Flaps
Size: A4
Date: 2019-03-03
Rev:
KiCad E.D.A. kicad (6.0.0-rc1-dev-1030-ga1074e65f2)
Id: 1/1

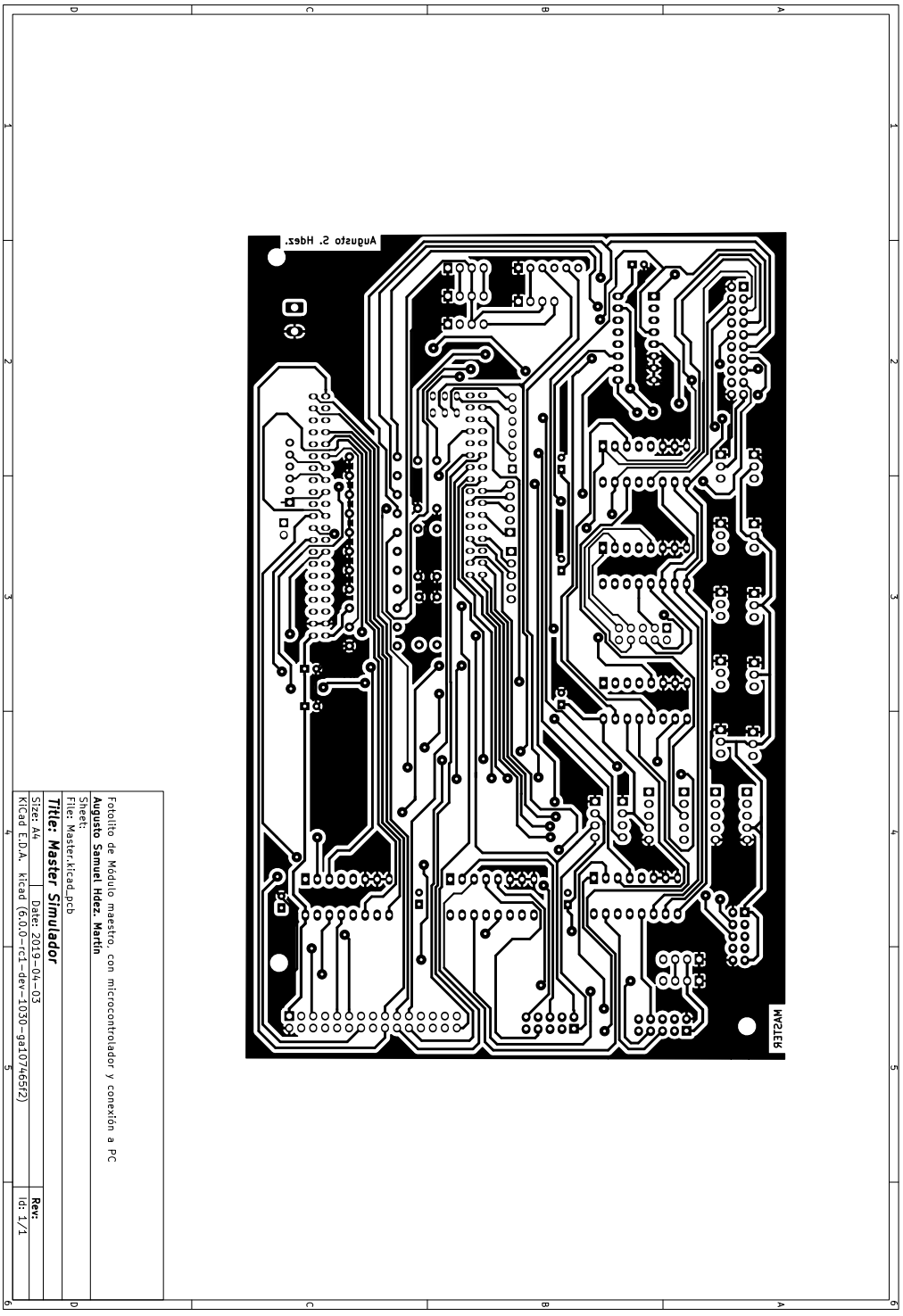

Alimentación
Conector para LCD
Conector de interruptores y sus LEDs
Multiplexado de LEDs por I2C
Conector hacia el maestro ATmega 8515
Módulo que implementa I2C para controlar los LEDs y recoge los interruptores del piloto Automático con sus R-pull-down.
Augusto Samuel Hernández Martín
Sheet: /
File: Piloto_Automático.sch
Title: Modulo Piloto Automatico
Size: A4
Date: 2019-05-03
Rev:
KiCad E.D.A. kicad (6.0.0-rc1-dev-1030-ga10746572)
Id: 1/1

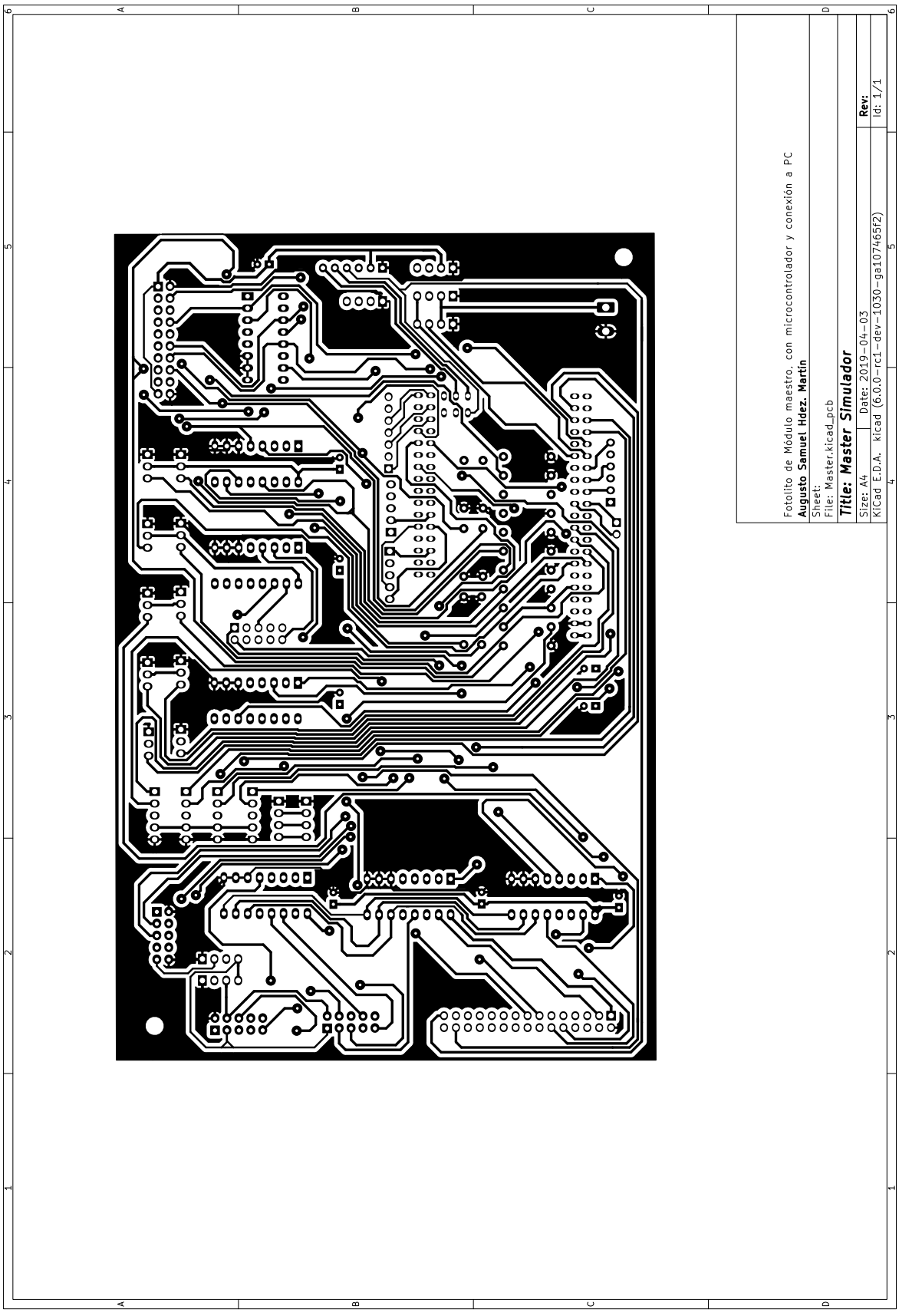

Conectores I2C
J4
I2C_LIBRE
J1
I2C_IN_WARNING
Multiplexado de LEDS con I2C
U1
PCF8574A
Conector LEDs WARNING
J3
LEDS
Alimentación
J2
Alimentación
Módulo que permite activar los indicadores de advertencia relativos a problemas de presión, tensión o combustible, empleando una multiplexación por I2C.
Augusto Samuel Hdez. Martín
Sheet: /
File: Warning.sch
Title: Modulo Warning
Size: A4
Date: 2019-03-03
Rev:
KiCad E.D.A. kicad (6.0.0-rc1-dev-1030-ga1074e65f2)
Id: 1/1

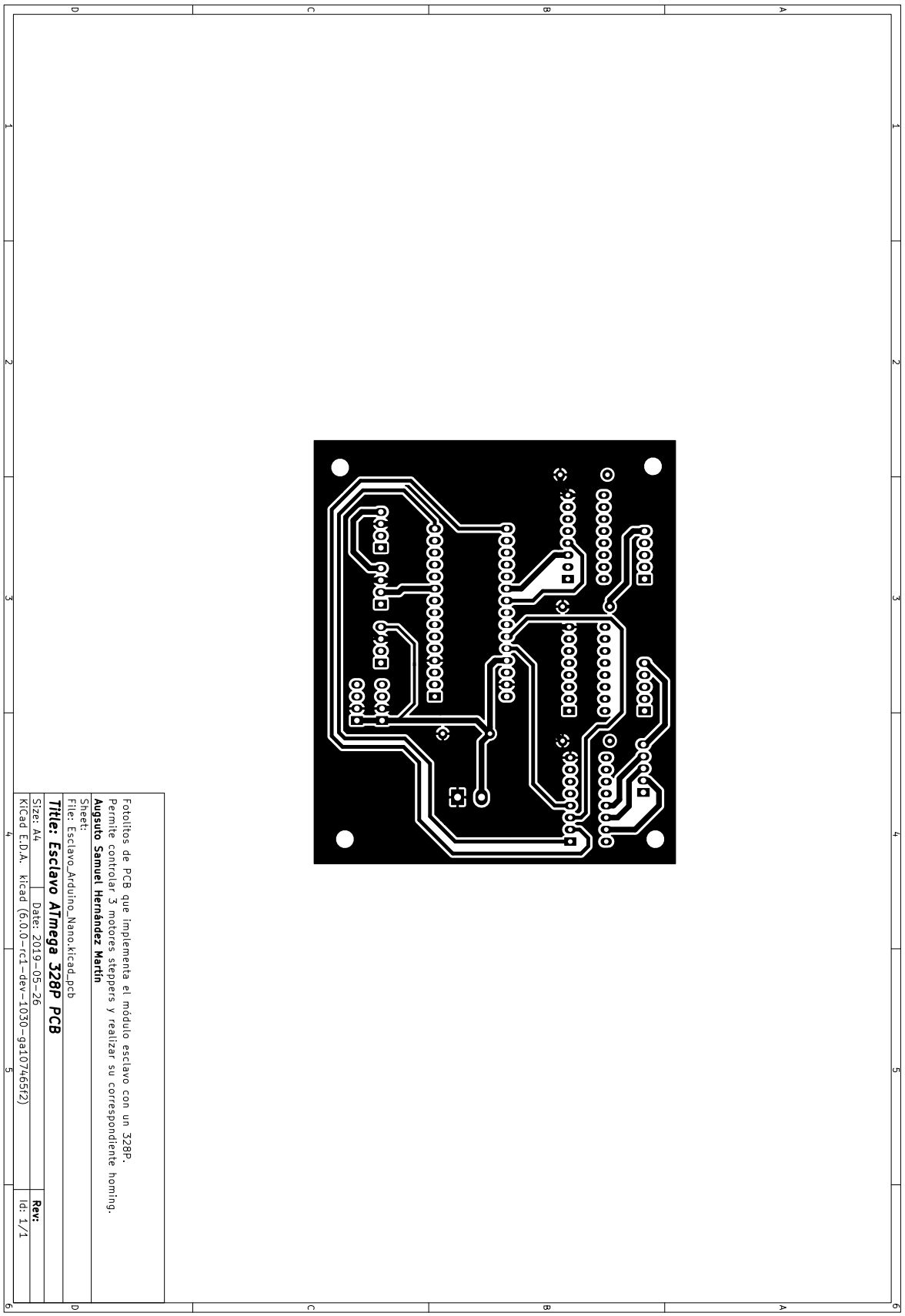

Alimentación
5V
GND
J1
Alimentacion
Pinheader para los botones
J2
Conector_Botones_Luces
17 5V
16 GND
15 Mag_Off
14 Mag_R
13 Mag_L
12 Mag_Both
11 Mag_Start
10 ALT
9 BAT
8 F_pump
7 BCN
6 LAND
5 TAXI
4 NAV
3 STRO
2 PITOT
1 AVIONIC
IDC para conectar a Master
J3
Con_IDC_a_PMaster
5V 1
GND 2
Mag_Off 3
Mag_R 4
Mag_L 5
Mag_Both 6
Mag_Start 7
ALT 8
BAT 9
F_pump 10
BCN 11
LAND 12
TAXI 13
NAV 14
STRO 15
PITOT 16
AVIONIC 17
5V 19
GND 20
Augusto Samuel Hernández Martín
Sheet: /
File: Luces.sch
Title: Conexiones módulo de Luces
Size: A4
Date: 2019-02-27
KiCad E.D.A. kicad (6.0.0-rc1-dev-1030-ga1074652)
Rev: Augusto S. Hdez. Martín
Id: 1/1

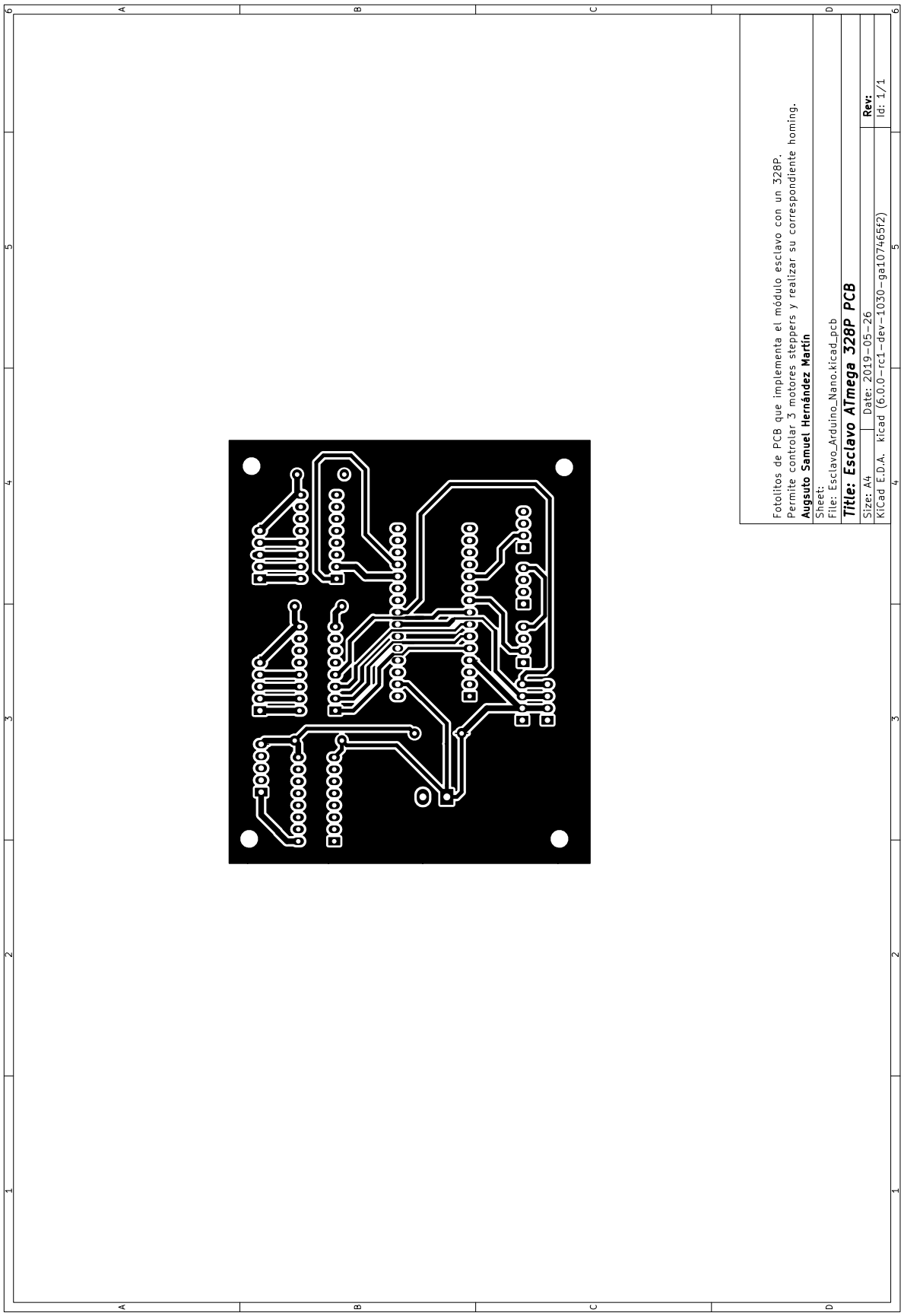

Módulo de Aceleradores y Gas ("Trhottle")
J1
Alimentación
5V
GND
J3
IDC_a_Master
5V
GND
Trim
Acel
Carb
Mixture
5V
GND
J2
Conector_Potenciometros
Módulo para agrupar conectores de potenciómetros para gases
Augusto S. Hdez. Martín
Sheet: /
File: Gases.sch
Title: Módulo de Gases
Size: A4
Date: 2019-02-27
Rev: Augusto S.
KiCad E.D.A. kicad (6.0.0-rc1-dev-1030-ga107465f2)
Id: 1/1

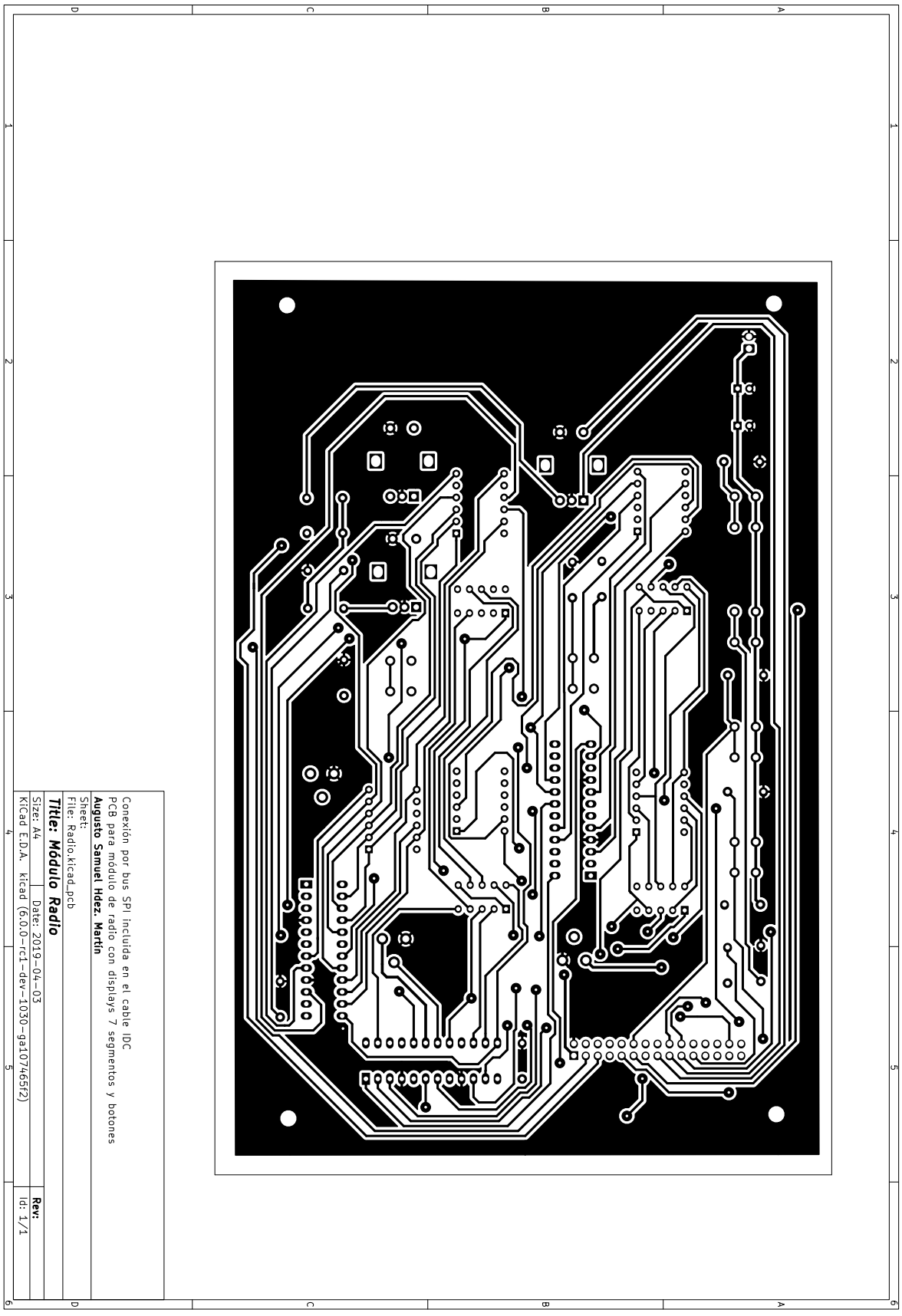
Fotolitos para PCB

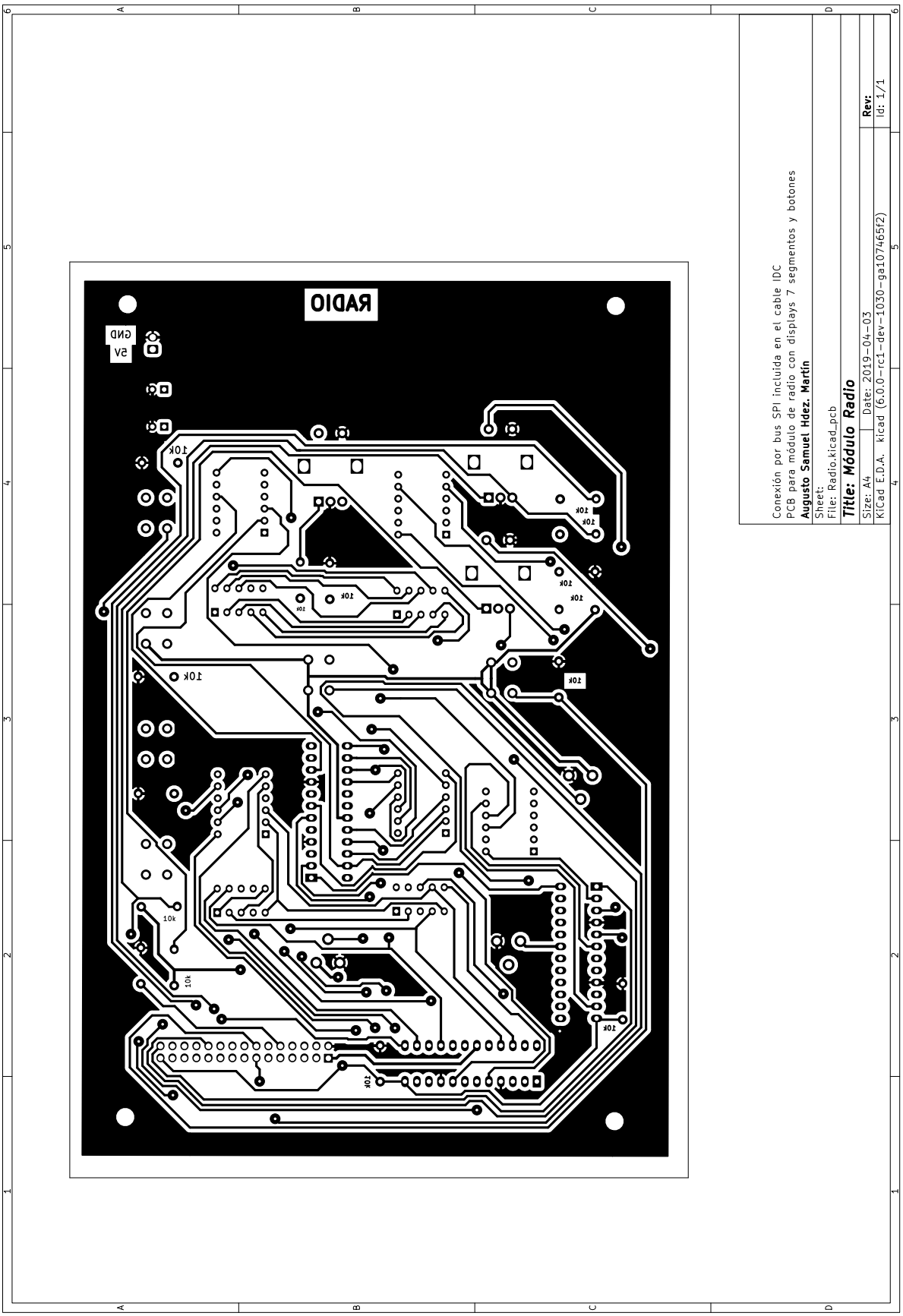


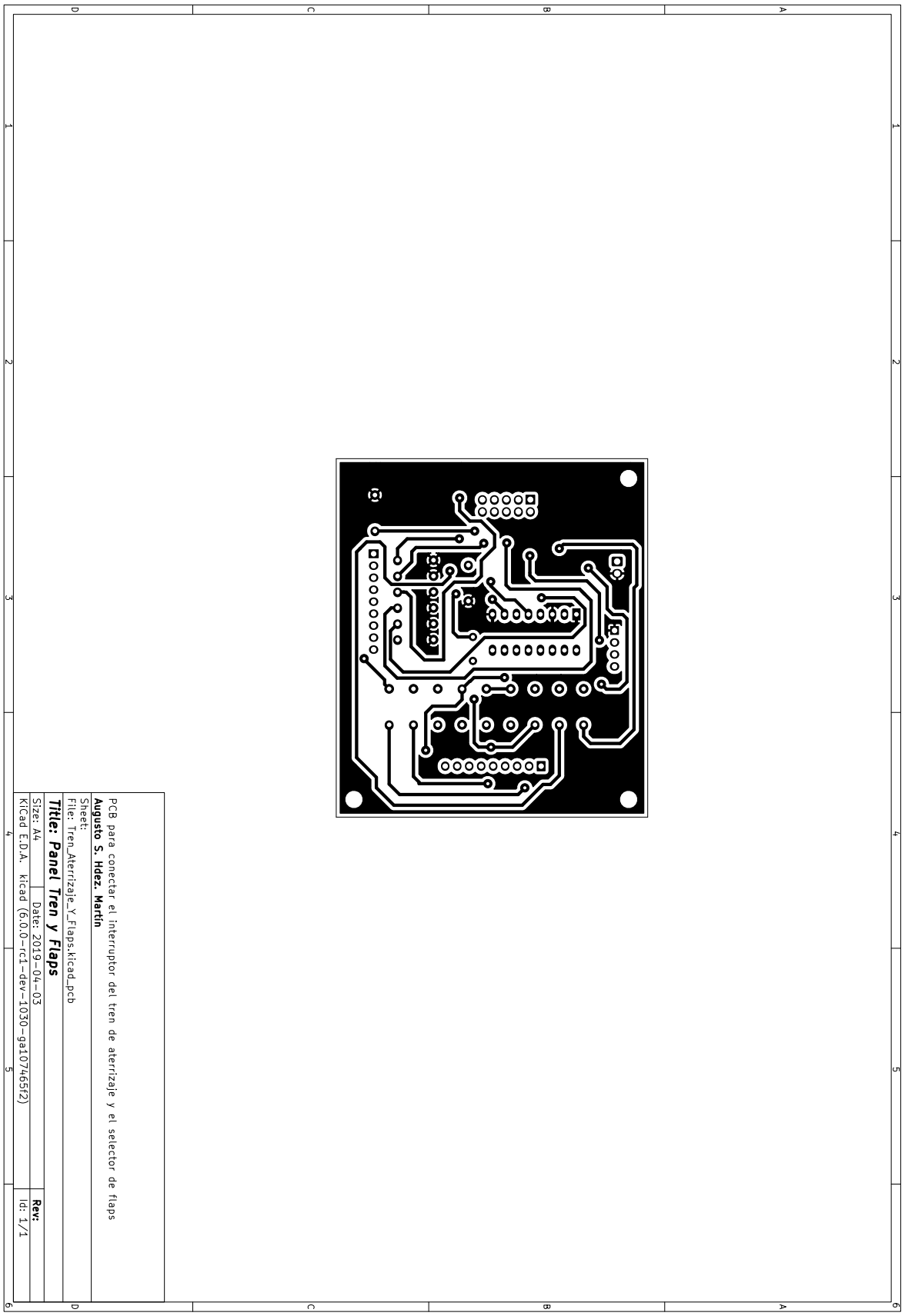

Fotolito de Módulo maestro, con microcontrolador y conexión a PC
Augusto Samuel Hdez. Martín
Sheet:
File: Master.kicad_pcb
Title: Master Simulador
Size: A4
Date: 2019-04-03
Rev:
KiCad E.D.A. kicad (6.0.0-rc1-dev-1030-ga107465f2)
Id: 1/1

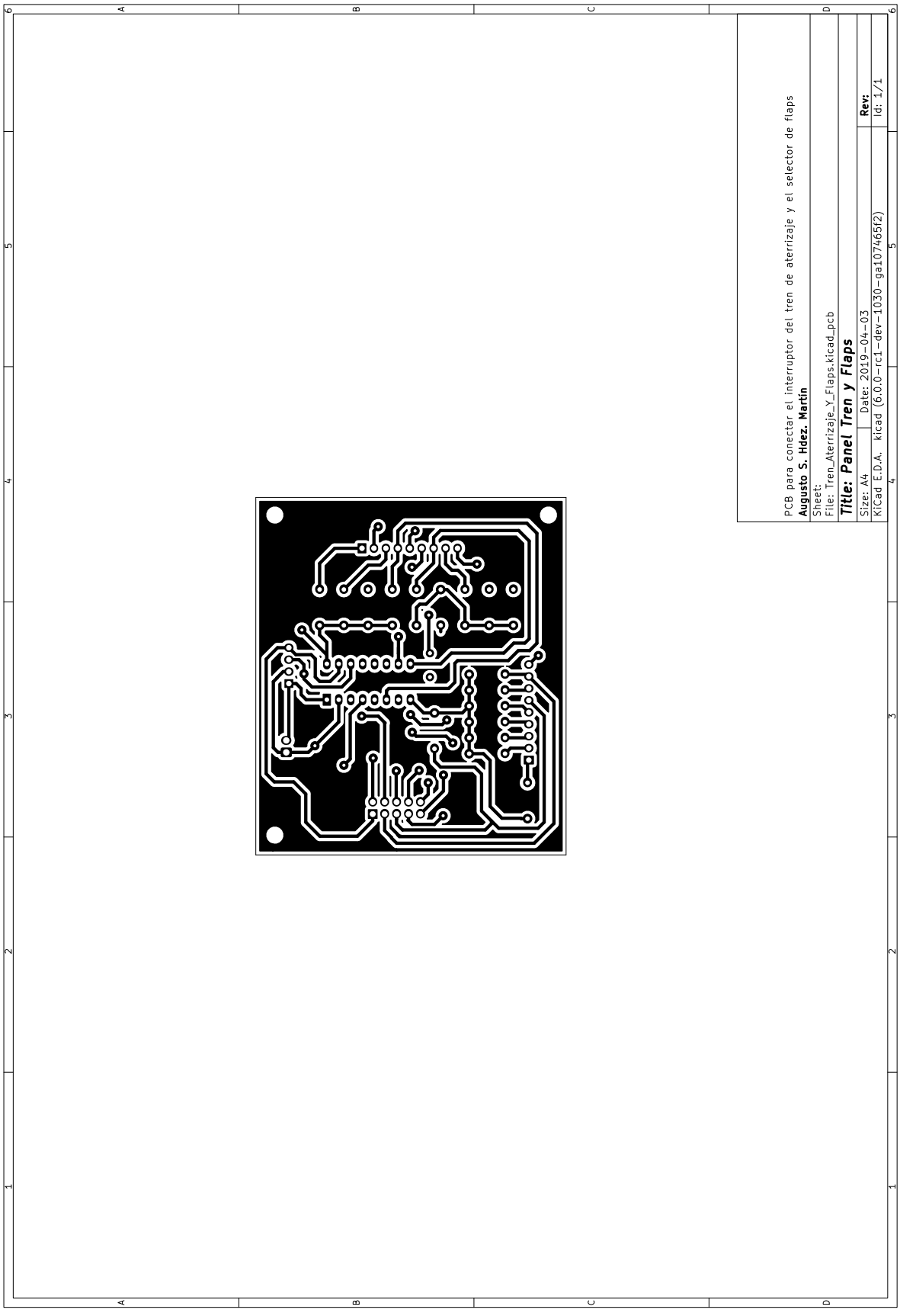

Fotolitos de PCB que implementa el módulo esclavo con un 328P.
Permite controlar 3 motores steppers y realizar su correspondiente homing.
Augsuto Samuel Hernández Martín
Sheet:
File: Esclavo_Arduino_Nano.kicad_pcb
Title: Esclavo ATmega 328P PCB
Size: A4
Date: 2019-05-26
Rev:
KiCad E.D.A. kicad (6.0.0-rc1-dev-1030-ga10746572)
Id: 1/1

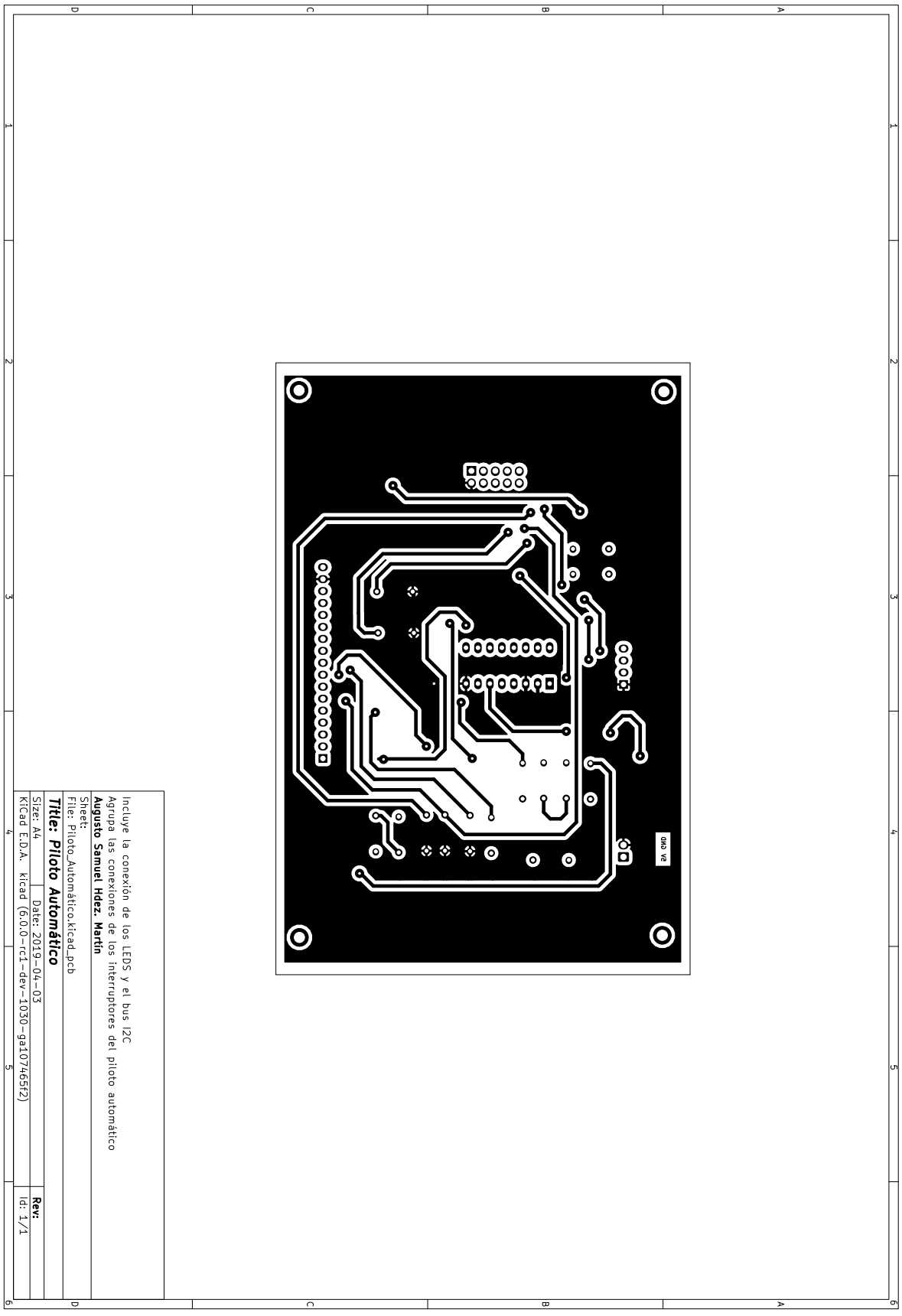

Fotolitos de PCB que implementa el módulo esclavo con un 328P.
Permite controlar 3 motores steppers y realizar su correspondiente homing.
Augusto Samuel Hernández Martín
Sheet:
File: Esclavo_Arduino_Nano.kicad_pcb
Title: Esclavo ATmega 328P PCB
Size: A4
Date: 2019-05-26
Rev:
KiCad E.D.A. kicad (6.0.0-rc1-dev-1030-ga10746512)
Id: 1/1

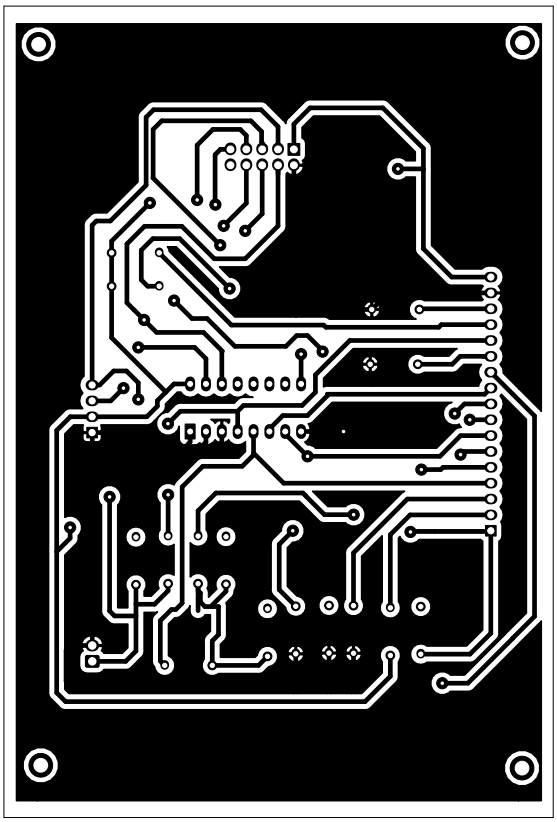

Conexión por bus SPI incluida en el cable IDC
PCB para módulo de radio con displays 7 segmentos y botones
Augusto Samuel Hdez. Martín
Sheet:
File: Radio.kicad_pcb
Title: Módulo Radio
Size: A4
Date: 2019-04-03
Rev:
KiCad E.D.A. kicad (6.0.0-rc1-dev-1030-ga107465f2)
Id: 1/1


RADIO
5A
GND
10K
Conexión por bus SPI incluida en el cable IDC
PCB para módulo de radio con displays 7 segmentos y botones
Augusto Samuel Hdez. Martín
Sheet:
File: Radio.kicad_pcb
Title: Módulo Radio
Size: A4
Date: 2019-04-03
Rev:
KiCad E.D.A. kicad (6.0.0-rc1-dev-1030-ga107465f2)
Id: 1/1


PCB para conectar el interruptor del tren de aterrizaje y el selector de flaps
Augusto S. Hdez. Martín
Sheet:
File: Tren_Aterrizaje_Y_Flaps.kicad_pcb
Title: Panel Tren y Flaps
Size: A4
Date: 2019-04-03
Rev:
KiCad E.D.A. kicad (6.0.0-rc1-dev-1030-ga10746572)
Id: 1/1


PCB para conectar el interruptor del tren de aterrizaje y el selector de flaps
Augusto S. Hdez. Martín
Sheet:
File: Tren_Aterrizaje_Y_Flaps.kicad_pcb
Title: Panel Tren y Flaps
Size: A4
Date: 2019-04-03
Rev:
KiCad E.D.A. kicad (6.0.0-rc1-dev-1030-ga1074f65f2)
Id: 1/1

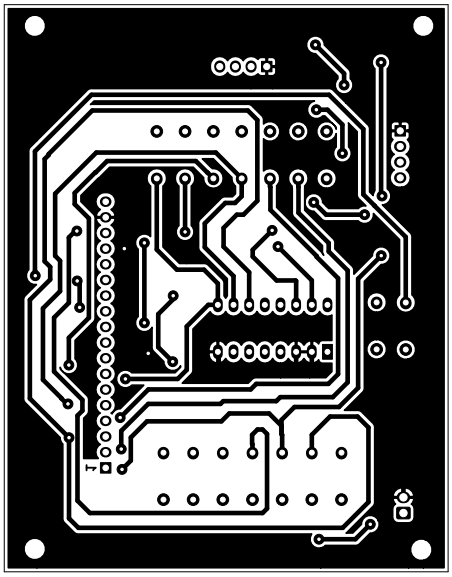

GND
VIN
Incluye la conexión de los LEDS y el bus I2C
Agrupa las conexiones de los interruptores del piloto automático
Augusto Samuel Hdez. Martín
Sheet:
File: Piloto_Automático.kicad_pcb
Title: Piloto Automático
Size: A4
Date: 2019-04-03
Rev:
KiCad E.D.A. kicad (6.0.0-rc1-dev-1030-ga107465f2)
Id: 1/1



Incluye la conexión de los LEDS y el bus I2C
Agrupa las conexiones de los interruptores del piloto automático
Augusto Samuel Hdez. Martín
Sheet:
File: Piloto_Automático.kicad_pcb

Title: Piloto Automático

Size: A4	Date: 2019-04-03	**Rev:**
KiCad E.D.A. kicad (6.0.0-rc1-dev-1030-ga107465f2)		Id: 1/1



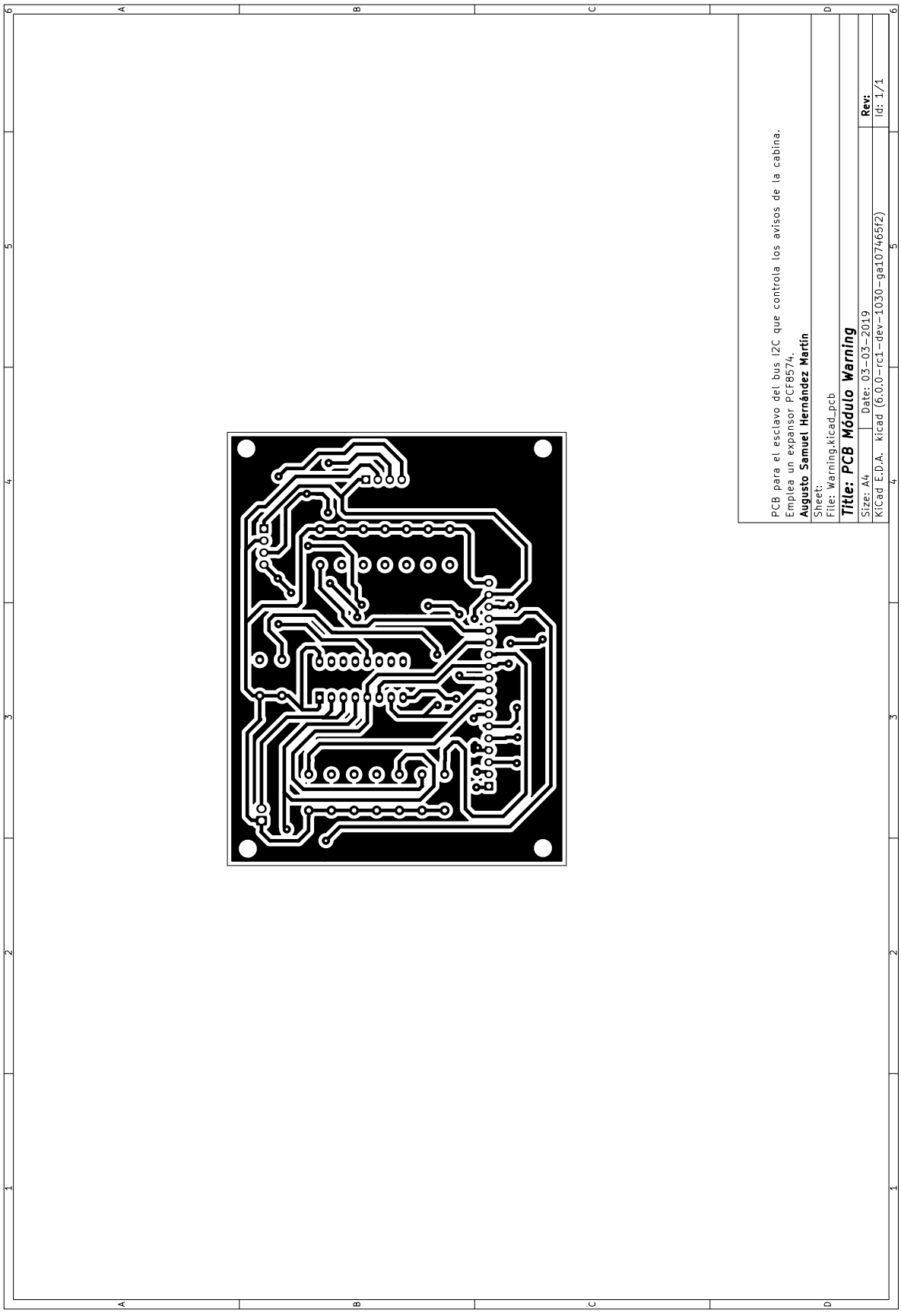
PCB para el esclavo del bus I2C que controla los avisos de la cabina.
Emplea un expansor PCF8574.

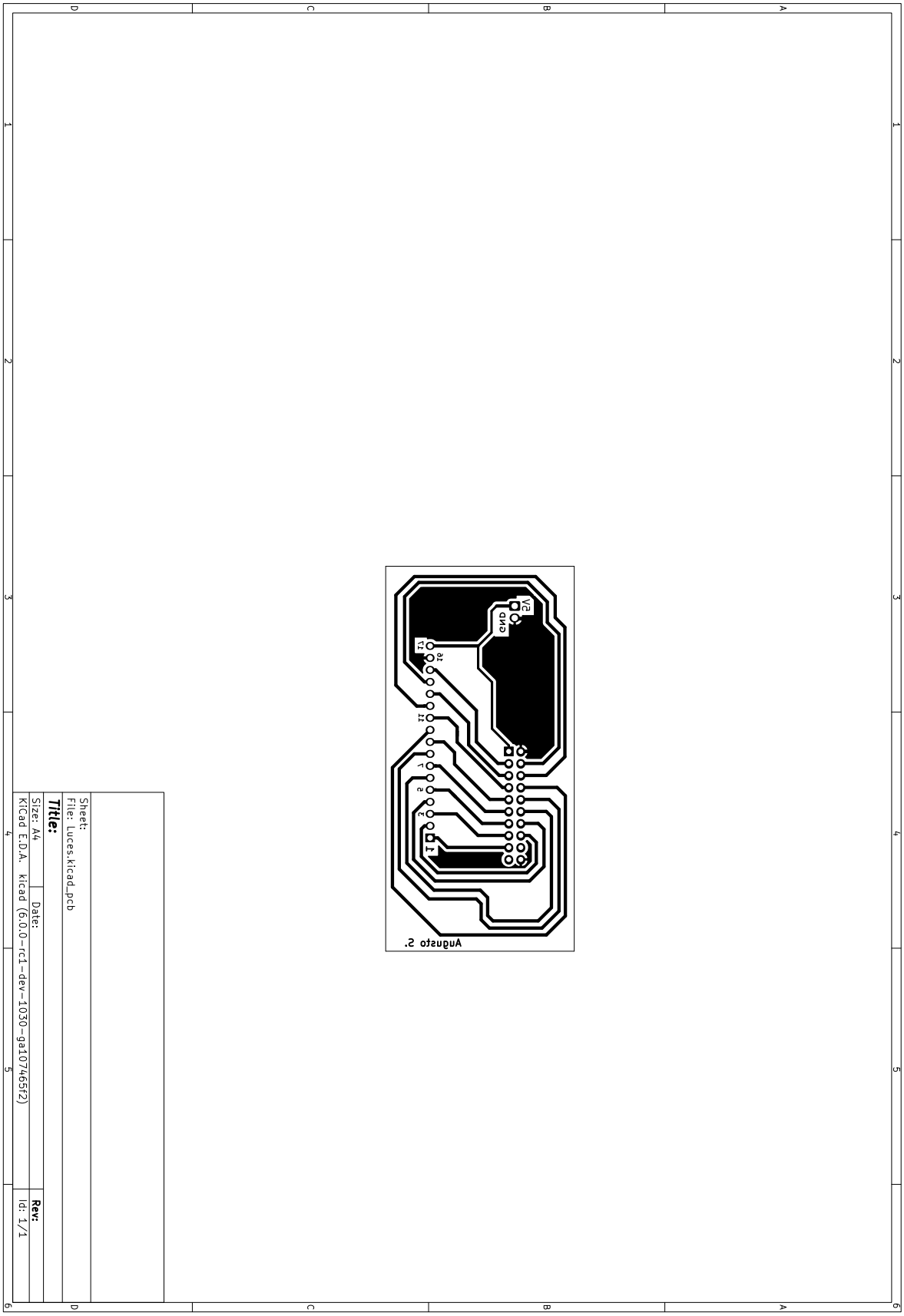
Augusto Samuel Hernández Martín

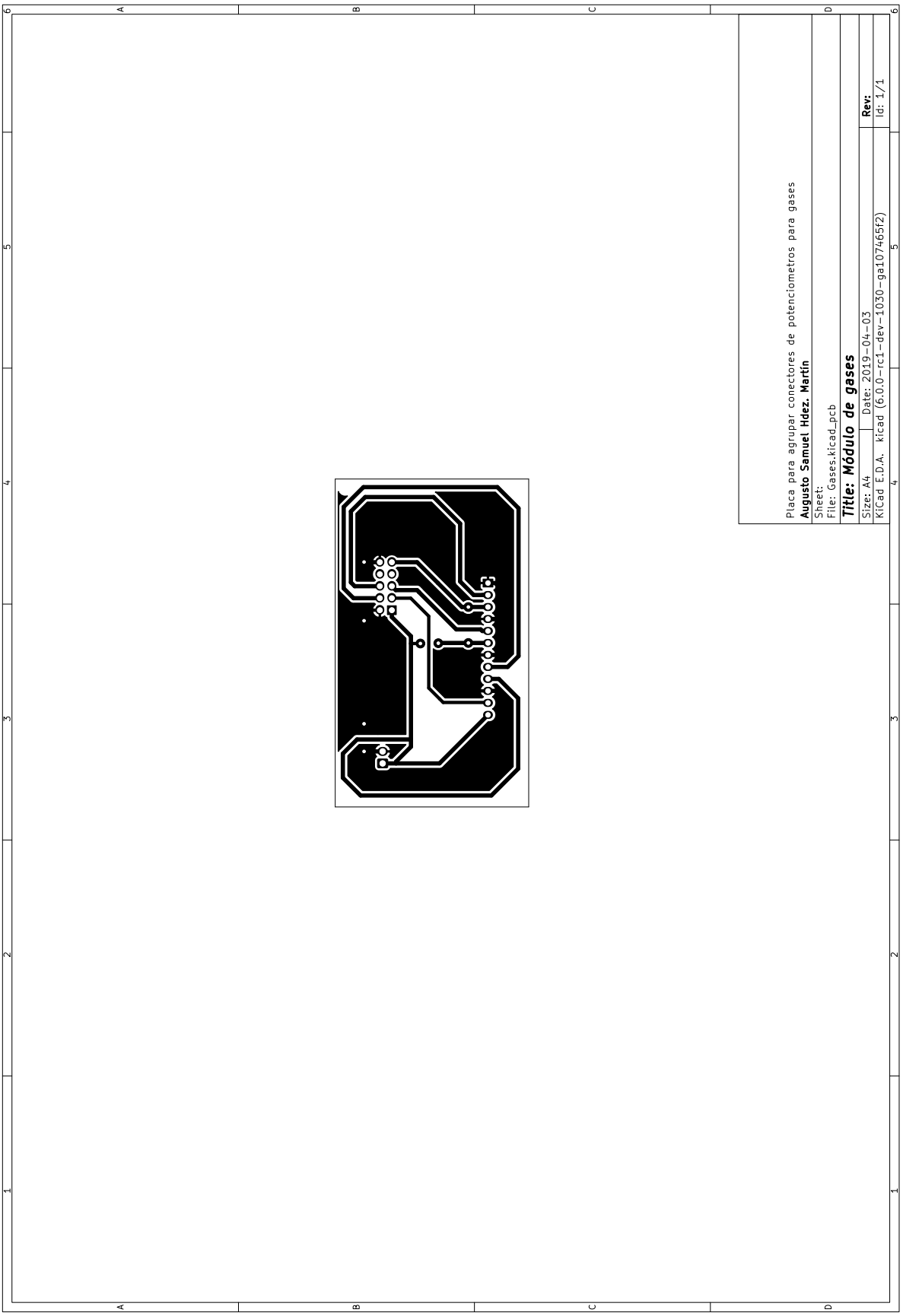
Sheet:
File: Warning.kicad_pcb

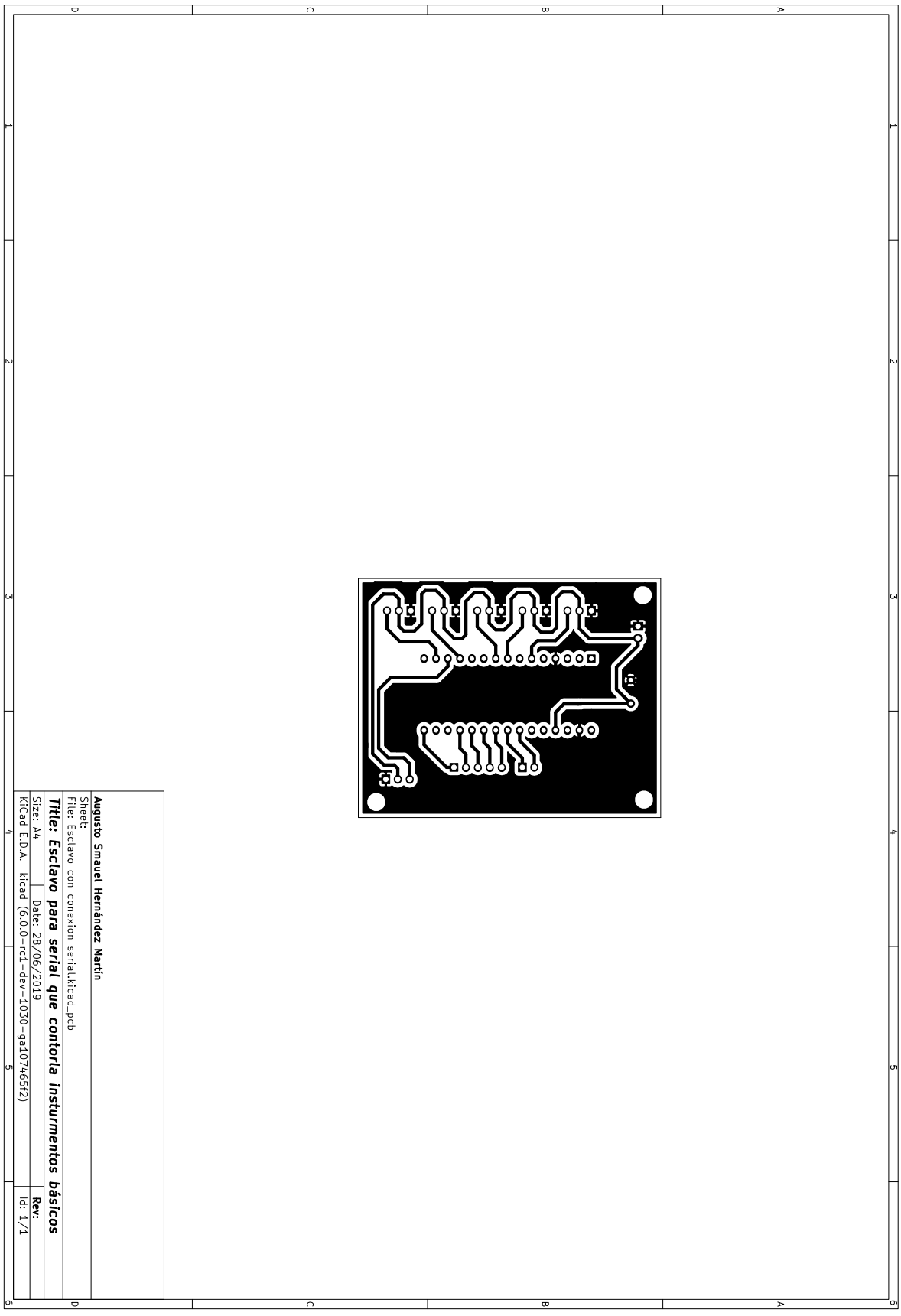
Title: PCB Módulo Warning

Size: A4	Date: 03-03-2019	**Rev:**
KiCad E.D.A. kicad (6.0.0-rc1-dev-1030-ga107465f2)		Id: 1/1


PCB para el esclavo del bus I2C que controla los avisos de la cabina.
Emplea un expansor PCF8574.
Augusto Samuel Hernández Martín
Sheet:
File: Warning.kicad_pcb
Title: PCB Módulo Warning
Size: A4
Date: 03-03-2019
Rev:
KiCad E.D.A. kicad (6.0.0-rc1-dev-1030-ga1074651f2)
Id: 1/1


Sheet:
File: Luces.kicad_pcb
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. kicad (6.0.0-rc1-dev-1030-ga107465f2)
Id: 1/1


Placa para agrupar conectores de potenciometros para gases
Augusto Samuel Hdez. Martín
Sheet:
File: Gases.kicad_pcb
Title: Módulo de gases
Size: A4
Date: 2019-04-03
Rev:
KiCad E.D.A. kicad (6.0.0-rc1-dev-1030-ga10746512)
Id: 1/1


Augusto Smauel Hernández Martín
Sheet:
File: Esclavo con conexion serial.kicad_pcb
Title: Esclavo para serial que contorla insturmentos básicos
Size: A4
Date: 28/06/2019
Rev:
KiCad E.D.A. kicad (6.0.0-rc1-dev-1030-ga107465f2)
Id: 1/1

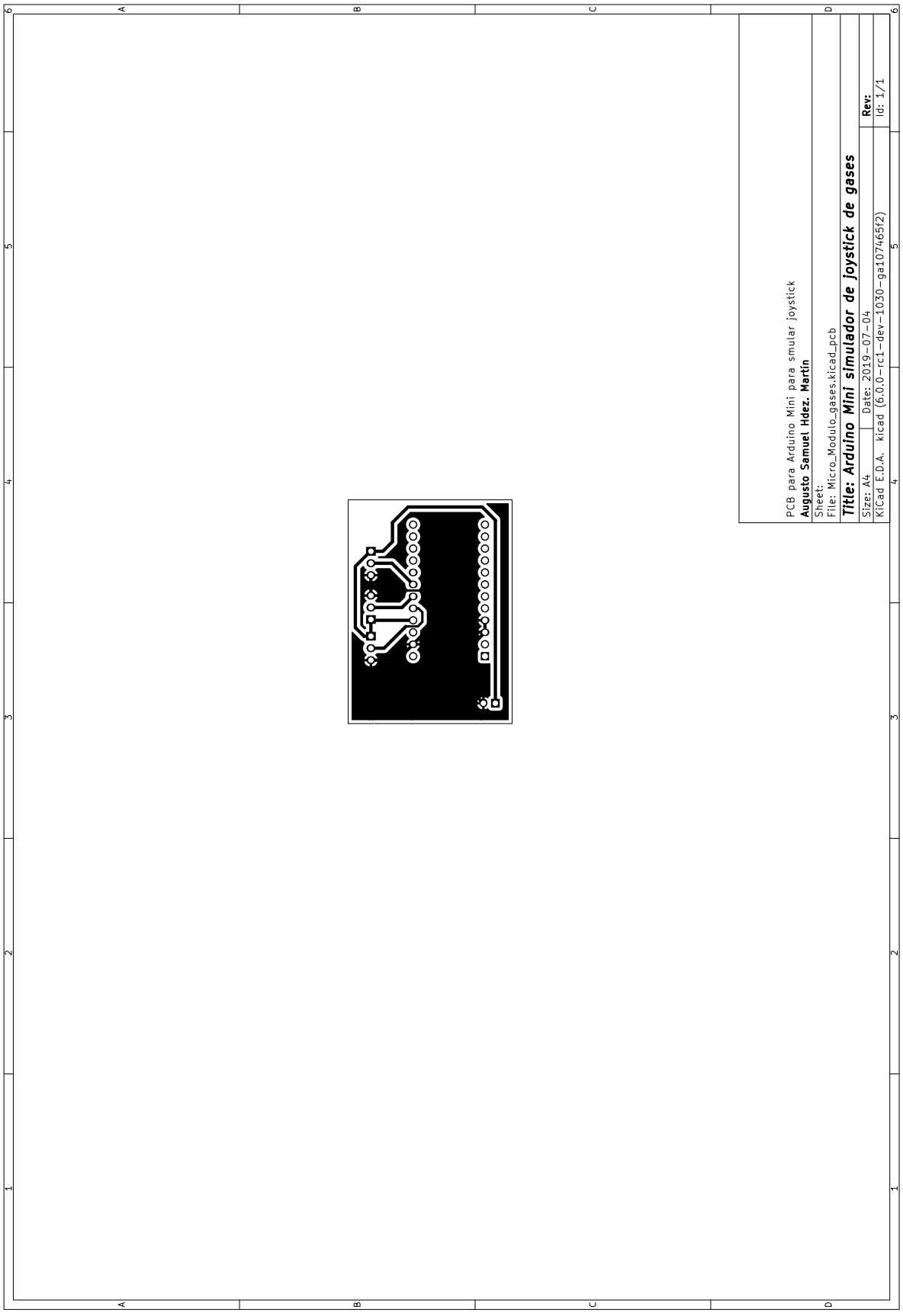
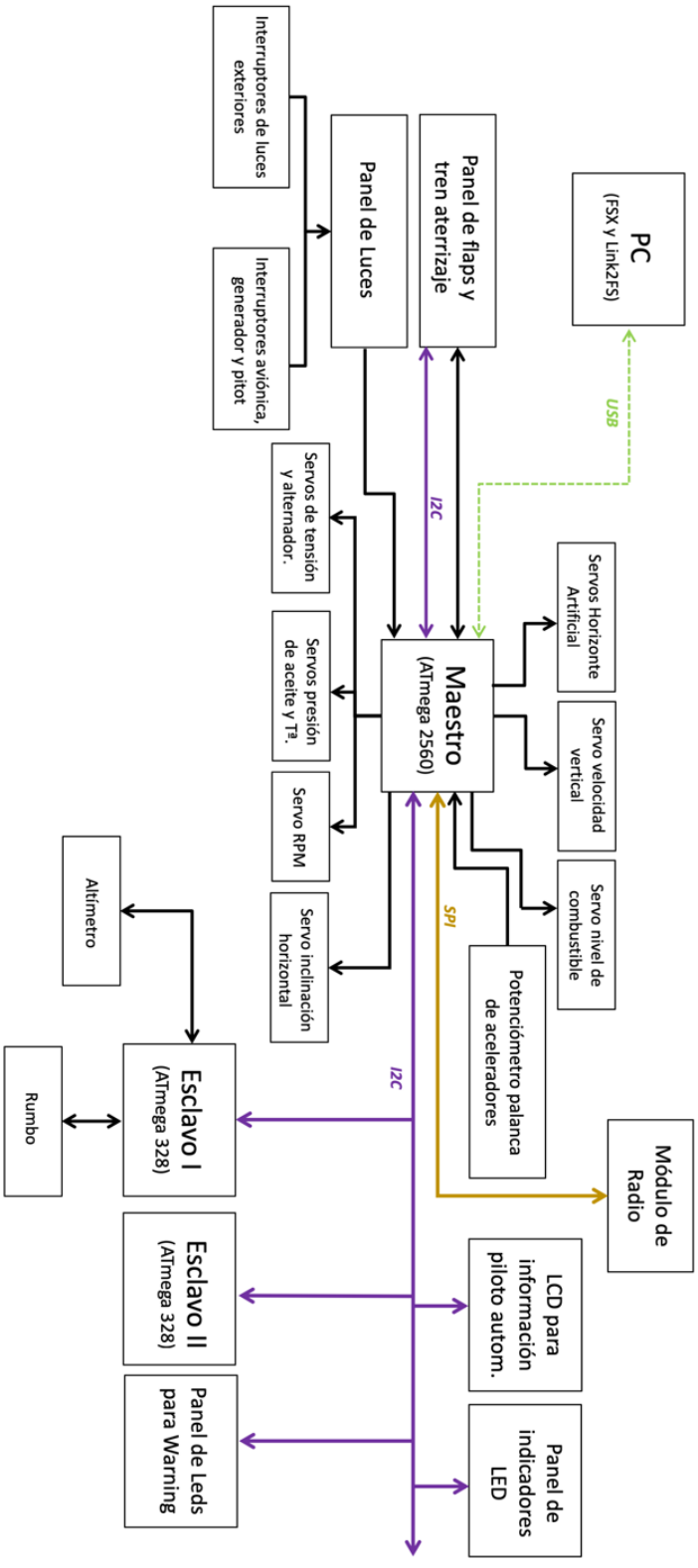
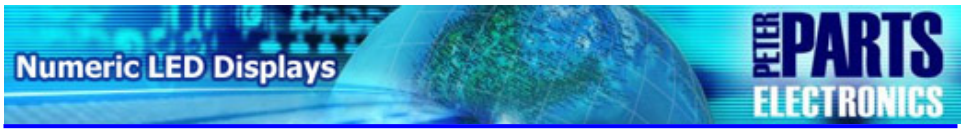

PCB para Arduino Mini para smular joystick
Augusto Samuel Hdez. Martín
Sheet:
File: Micro_Modulo_gases.kicad_pcb
Title: Arduino Mini simulador de joystick de gases
Size: A4
Date: 2019-07-04
Rev:
KiCad E.D.A. kicad (6.0.0-rc1-dev-1030-ga1074465f2)
Id: 1/1

Diagrama de Bloque del dispositivo



Datasheets



Part No.: NED-3641/42AS-11

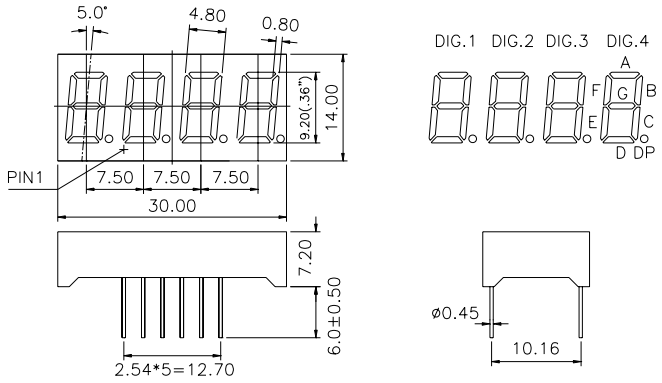
Features:

- 0.36 INCH DIGIT HEIGHT.
- LOW CURRENT OPERATION.
- EASY MOUNTING ON P.C. BOARDS OR SOCKETS.
- I.C. COMPATIBLE.
- MECHANICALLY RUGGED.
- STANDARD: BLACK FACE,WHITE SEGMENT.
- RoHS COMPLIANT

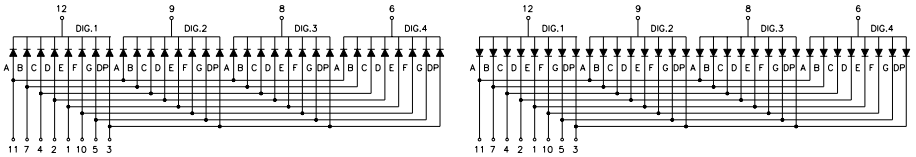
Descriptions:

The Super Bright Red source color devices are made with Gallium Aluminum Arsenide Red Light Emitting Diode

Package Dimensions & Internal Circuit Diagram



NEQ-3641AS **NEQ-3641AS**



Notes:

1. All dimensions are in millimeters(inches),Tolerance is ±0.25（0.01”）unless otherwise noted.
2. Specifications are subject to change without notice.

TEXAS INSTRUMENTS

PCF8574

SCPS068J – JULY 2001 – REVISED MARCH 2015

PCF8574 Remote 8-Bit I/O Expander for I²C Bus

1 Features

- Low Standby-Current Consumption of 10 µA Max
- I²C to Parallel-Port Expander
- Open-Drain Interrupt Output
- Compatible With Most Microcontrollers
- Latched Outputs With High-Current Drive Capability for Directly Driving LEDs
- Latch-Up Performance Exceeds 100 mA Per JESD 78, Class II

2 Applications

- Telecom Shelters: Filter Units
- Servers
- Routers (Telecom Switching Equipment)
- Personal Computers
- Personal Electronics
- Industrial Automation
- Products with GPIO-Limited Processors

3 Description

This 8-bit input/output (I/O) expander for the two-line bidirectional bus (I²C) is designed for 2.5-V to 6-V V_{CC} operation.

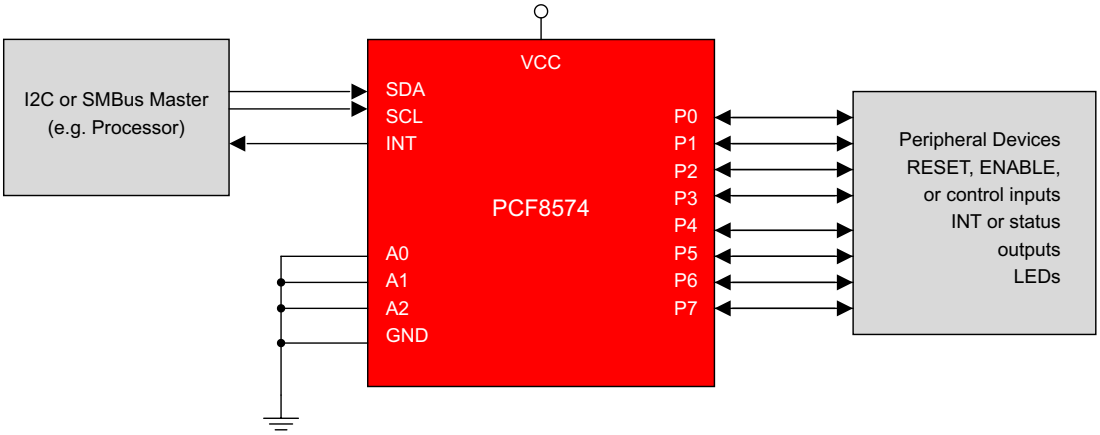
The PCF8574 device provides general-purpose remote I/O expansion for most microcontroller families by way of the I²C interface [serial clock (SCL), serial data (SDA)].

The device features an 8-bit quasi-bidirectional I/O port (P0–P7), including latched outputs with high-current drive capability for directly driving LEDs. Each quasi-bidirectional I/O can be used as an input or output without the use of a data-direction control signal. At power on, the I/Os are high. In this mode, only a current source to V_{CC} is active.

Device Information(1)

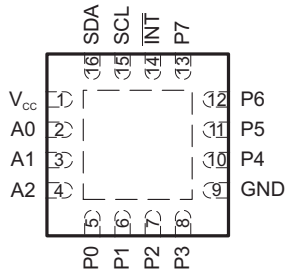
PART NUMBER	PACKAGE (PIN)	BODY SIZE (NOM)
PCF8574	TVSOP (20)	5.00 mm × 4.40 mm
	SOIC (16)	10.30 mm × 7.50 mm
	PDIP (16)	19.30 mm × 6.35 mm
	TSSOP (20)	6.50 mm × 4.40 mm
	QFN (16)	3.00 mm × 3.00 mm
	VQFN (20)	4.50 mm × 3.50 mm

(1) For all available packages, see the orderable addendum at the end of the data sheet.

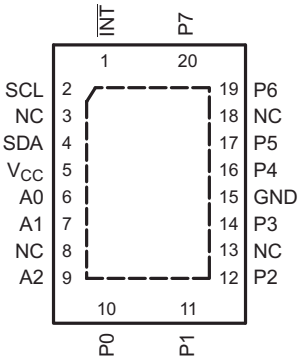


5 Pin Configuration and Functions

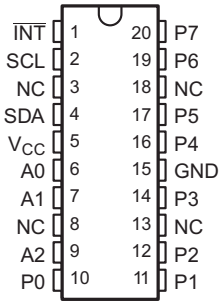
RGT PACKAGE
(TOP VIEW)



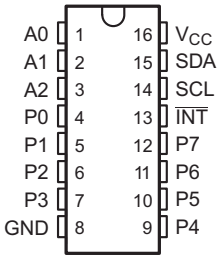
RGY PACKAGE
(TOP VIEW)



DGV OR PW PACKAGE
(TOP VIEW)



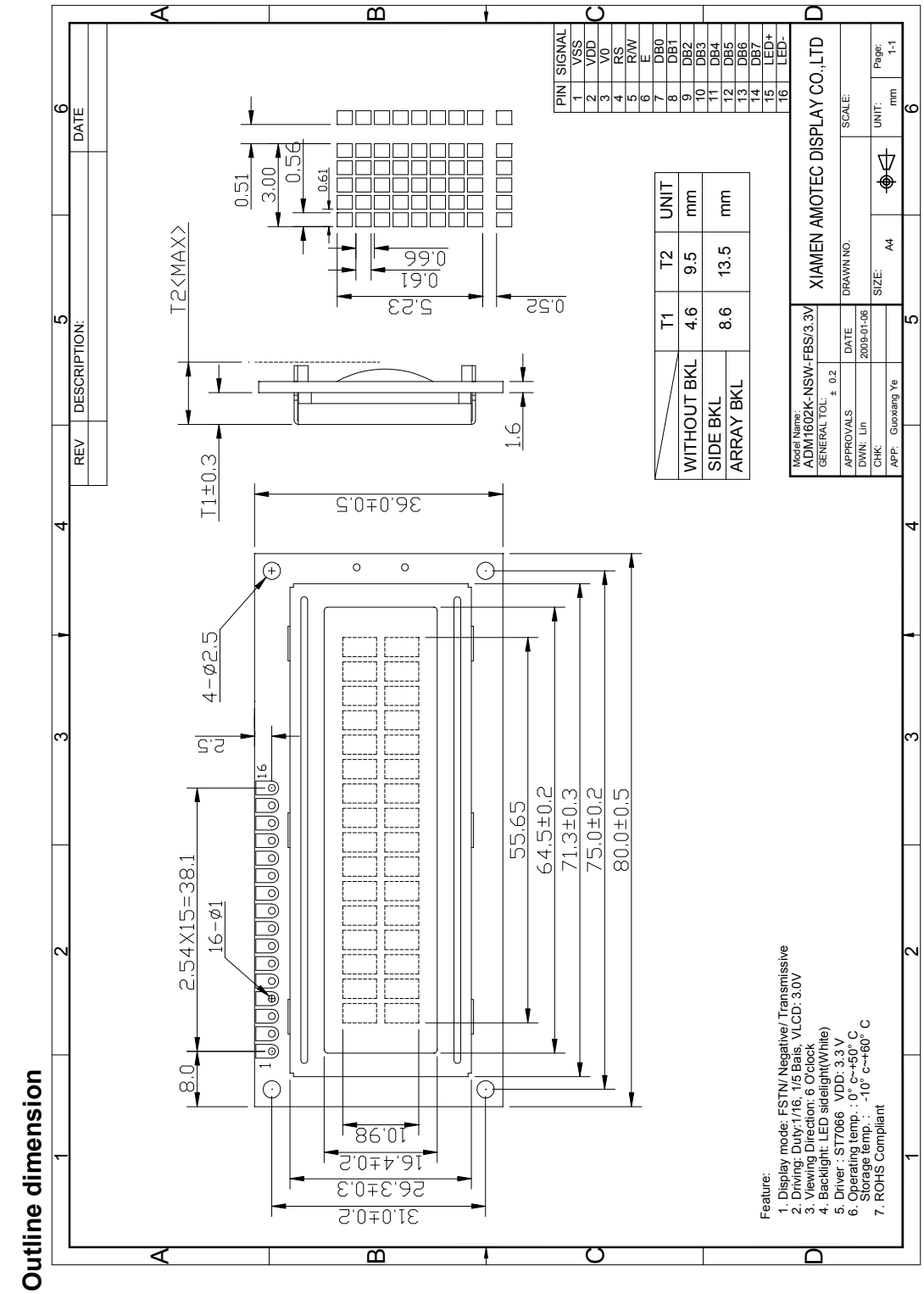
DW OR N PACKAGE
(TOP VIEW)



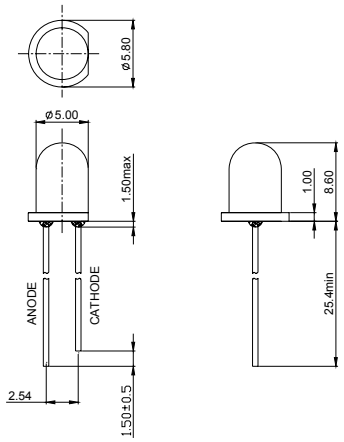
Pin Functions

PIN					TYPE	DESCRIPTION
NAME	RGT	RGY	DGV or PW	DW or N		
A [0..2]	2, 3, 4	6, 7, 9	6, 7, 9	1, 2, 3	I	Address inputs 0 through 2. Connect directly to V_{CC} or ground. Pullup resistors are not needed.
GND	9	15	15	8	—	Ground
$\overline{INT}$	14	1	1	13	O	Interrupt output. Connect to V_{CC} through a pullup resistor.
NC	-	3, 8, 13, 18	3, 8, 13, 18	-	—	Do not connect
P[0..7]	5, 6, 7, 8, 10, 11, 12, 13	10, 11, 12, 14, 16, 17, 19, 20	10, 11, 12, 14, 16, 17, 19, 20	4, 5, 6, 7, 9, 10, 11, 12	I/O	P-port input/output. Push-pull design structure.
SCL	15	2	2	14	I	Serial clock line. Connect to V_{CC} through a pullup resistor
SDA	16	4	4	15	I/O	Serial data line. Connect to V_{CC} through a pullup resistor.
V_{CC}	1	5	5	16	—	Voltage supply

3. Outline dimension



■ Dimension Drawing



Note:
1.All dimensions in mm tolerance is ±0.2mm unless otherwise noted.
2.An epoxy meniscus may extend about 1.5mm down the leads.
3.Burr around bottom of epoxy may be 0.5mm max.

■ Applications:

- Toys
- Lighting
- Traffic light
- Automotive
- Commercial Outdoor Advertising
- Front Panel Indicator

■ Absolute Maximum Ratings(Ta = 25℃)

Items	Symbol	Absolute maximum Rating	Unit
Forward Current(DC)	I_F	50	mA
Peak Forward Current*	I_{FP}	100	mA
Reverse Voltage	V_R	5	V
Power Dissipation	P_D	150	mW
Operation Temperature	T_{opr}	-20 ~ +95	℃
Storage Temperature	T_{stg}	-40 ~ +100	℃
Lead Soldering Temperature	T_{sol}	Max.260°C for 5 sec Max. (3mm from the base of the epoxy bulb)	

*pulse width ≦0.1msec duty ≦1/10

■ Typical Electrical & Optical Characteristics (Ta = 25℃)

Items	Symbol	Condition	Min.	Typ.	Max.	Unit
Forward Voltage	V_F	I_F = 20mA	1.8	---	2.4	V
Reverse Current	I_R	V_R = 5V	---	---	10	μA
Dominant Wavelength	λ_D	I_F = 20mA	586	---	596	nm
Luminous Intensity	I_V	I_F = 20mA	1700	---	3500	mcd
50% Power Angle	$2\theta_{1/2}$	I_F = 20mA	---	30	---	deg

■ Ranks Combination (IF = 20mA)

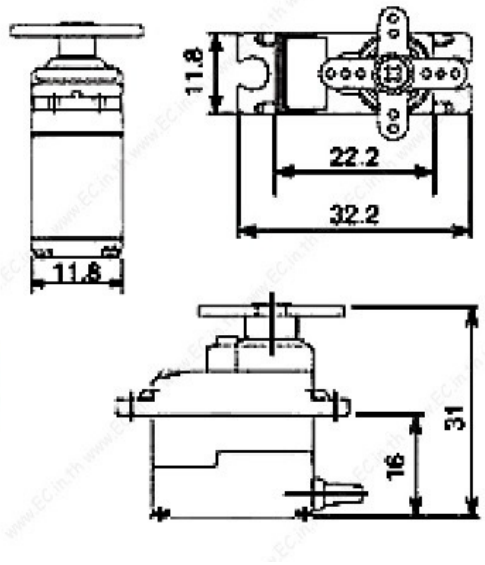
Rank	0C	0D	0E	0F	0G
Dominant Wavelength (nm)	586-588	588-590	590-592	592-594	594-596
Rank	0Q	1R	2R	0S	---
Luminous Intensity (mcd)	1700-2000	2000-2500	2500-3000	3000-3500	---

Rank	0G	0H	0J	0K	0L	0M
Forward Voltage(V)	1.8-1.9	1.9-2.0	2.0-2.1	2.1-2.2	2.2-2.3	2.3-2.4

Important Notes:

1) All ranks will be included per delivery.
2) Tolerance of measurement of luminous intensity is ±15%.
3) Tolerance of measurement of dominant wavelength is ±1nm.
4) Tolerance of measurement of forward voltage is ±0.05 V.
5）Pb content＜1000PPM.

SG90 9 g Micro Servo



Tiny and lightweight with high output power. Servo can rotate approximately 180 degrees (90 in each direction), and works just like the standard kinds but *smaller*. You can use any servo code, hardware or library to control these servos. Good for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. It comes with a 3 horns (arms) and hardware.

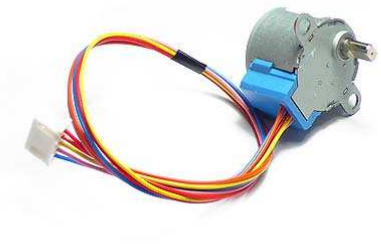
Specifications

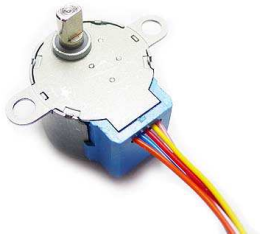
- Weight: 9 g
- Dimension: 22.2 x 11.8 x 31 mm approx.
- Stall torque: 1.8 kgf·cm
- Operating speed: 0.1 s/60 degree
- Operating voltage: 4.8 V (~5V)
- Dead band width: 10 µs
- Temperature range: 0 ºC – 55 ºC

Kiatronics
electronic design and manufacture

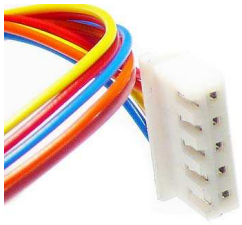
28BYJ-48 – 5V Stepper Motor

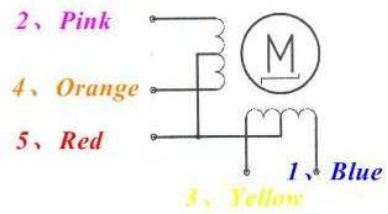
The 28BYJ-48 is a small stepper motor suitable for a large range of applications.

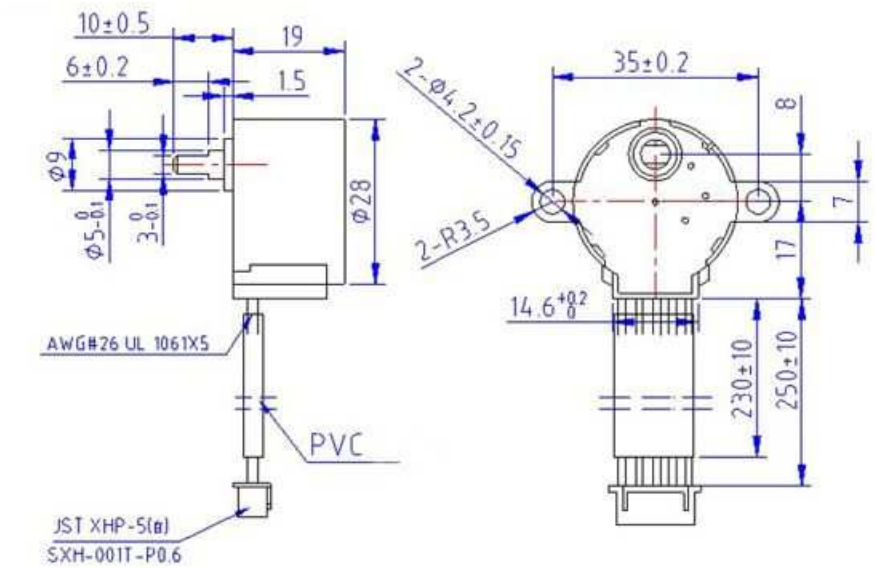




Rated voltage :	5VDC
Number of Phase	4
Speed Variation Ratio	1/64
Stride Angle	5.625°/64
Frequency	100Hz
DC resistance	50Ω±7%(25℃)
Idle In-traction Frequency	> 600Hz
Idle Out-traction Frequency	> 1000Hz
In-traction Torque	>34.3mN.m(120Hz)
Self-positioning Torque	>34.3mN.m
Friction torque	600-1200 gf.cm
Pull in torque	300 gf.cm
Insulated resistance	>10MΩ(500V)
Insulated electricity power	600VAC/1mA/1s
Insulation grade	A
Rise in Temperature	<40K(120Hz)
Noise	<35dB(120Hz,No load,10cm)
Model	28BYJ-48 – 5V







Lead-free Green

ULN2002A/ ULN2003A/ ULN2004A

HIGH VOLTAGE, HIGH CURRENT DARLINGTON TRANSISTOR ARRAYS

Description

The ULN2002A, ULN2003A and ULN2004A are high voltage, high current Darlington arrays each containing seven open collector common emitter pairs. Each pair is rated at 500mA. Suppression diodes are included for inductive load driving, the inputs and outputs are pinned in opposition to simplify board layout.

Device options are designed to be compatible with common logic families:

ULN2002A (14-25V PMOS)
ULN2003A (5V TTL, CMOS)
ULN2004A (6-15V CMOS, PMOS)

These devices are capable of driving a wide range of loads including solenoids, relays, DC motors, LED displays, filament lamps, thermal print-heads and high-power buffers.

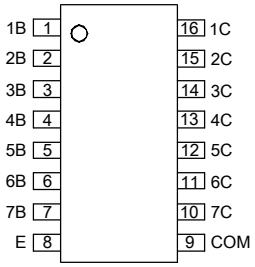
The ULN2002A, ULN2003A and ULN2004A are available in both a small outline 16-pin package (SO-16) and PDIP-16 package.

Features

- 500mA Rated Collector Current (Single Output)
- High Voltage Outputs: 50V
- Output Clamp Diodes
- Inputs Compatible with Popular Logic Types
- Relay Driver Applications
- "Green" Molding Compound (No Br, Sb)
- **Totally Lead-Free & Fully RoHS Compliant (Notes 1 & 2)**
- **Halogen and Antimony Free. "Green" Device (Note 3)**

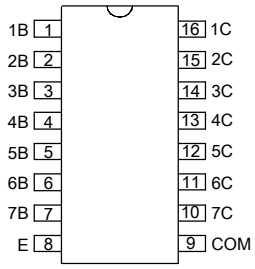
Pin Assignments

(Top View)



SO-16

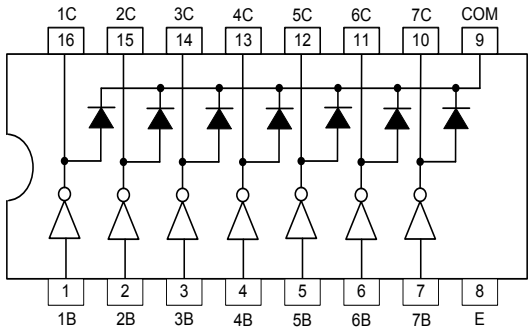
(Top View)



PDIP-16

Notes:
1. No purposely added lead. Fully EU Directive 2002/95/EC (RoHS) & 2011/65/EU (RoHS 2) compliant.
2. See http://www.diodes.com/quality/lead_free.html for more information about Diodes Incorporated's definitions of Halogen- and Antimony-free, "Green" and Lead-free.
3. Halogen- and Antimony-free "Green" products are defined as those which contain <900ppm bromine, <900ppm chlorine (<1500ppm total Br + Cl) and <1000ppm antimony compounds.

Connection Diagram



TEXAS INSTRUMENTS

CD4051B, CD4052B, CD4053B

SCHS047I – AUGUST 1998 – REVISED SEPTEMBER 2017

CD405xB CMOS Single 8-Channel Analog Multiplexer/Demultiplexer With Logic-Level Conversion

1 Features

- Wide Range of Digital and Analog Signal Levels
  - Digital: 3 V to 20 V
  - Analog: ≤20 V_{P-P}
- Low ON Resistance,125 Ω (Typical) Over 15 V_{P-P} Signal Input Range for $V_{DD} – V_{EE}$ = 18 V
- High OFF Resistance, Channel Leakage of ±100 pA (Typical) at $V_{DD} – V_{EE}$ = 18 V
- Logic-Level Conversion for Digital Addressing Signals of 3 V to 20 V ($V_{DD} – V_{SS}$ = 3 V to 20 V) to Switch Analog Signals to 20 V_{P-P} ($V_{DD} – V_{EE}$ = 20 V) Matched Switch Characteristics, r_{ON} = 5 Ω (Typical) for $V_{DD} – V_{EE}$ = 15 V Very Low Quiescent Power Dissipation Under All Digital-Control Input and Supply Conditions, 0.2 µW (Typical) at $V_{DD} – V_{SS} = V_{DD} – V_{EE}$ = 10 V
- Binary Address Decoding on Chip
- 5 V, 10 V, and 15 V Parametric Ratings
- 100% Tested for Quiescent Current at 20 V
- Maximum Input Current of 1 µA at 18 V Over Full Package Temperature Range, 100 nA at 18 V and 25°C
- Break-Before-Make Switching Eliminates Channel Overlap

2 Applications

- Analog and Digital Multiplexing and Demultiplexing
- A/D and D/A Conversion
- Signal Gating
- Factory Automation
- Televisions
- Appliances
- Consumer Audio
- Programmable Logic Circuits
- Sensors

3 Description

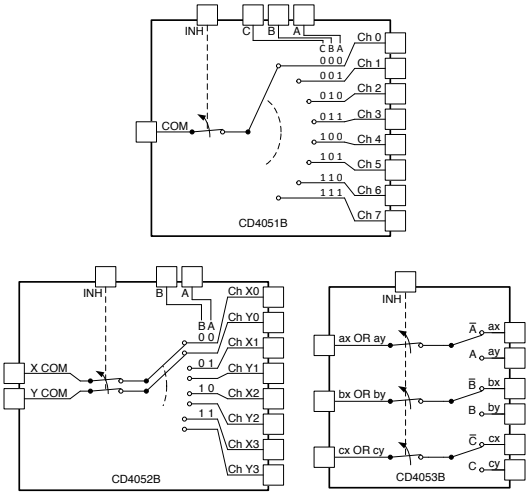
The CD405xB analog multiplexers and demuliplexers are digitally-controlled analog switches having low ON impedance and very low OFF leakage current. These multiplexer circuits dissipate extremely low quiescent power over the full $V_{DD} – V_{SS}$ and $V_{DD} – V_{EE}$ supply-voltage ranges, independent of the logic state of the control signals.

Device Information(1)

PART NUMBER	PACKAGE	BODY SIZE (NOM)
CD405xB	CDIP (16)	19.50 mm × 6.92 mm
	PDIP (16)	19.30 mm × 6.35 mm
	SOIC (16)	9.90 mm × 3.91 mm
	SOP (16)	10.30 mm × 5.30 mm
	TSSOP (16)	5.00 mm × 4.40 mm

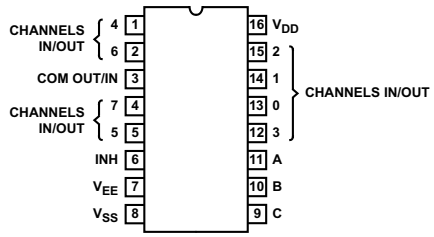
(1) For all available packages, see the orderable addendum at the end of the data sheet.

Functional Diagrams of CD405xB

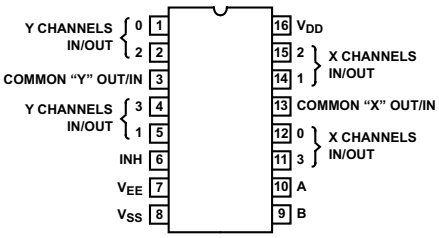


5 Pin Configuration and Functions

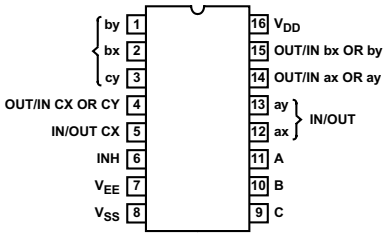
CD4051B E, M, NS, and PW Package
16-Pin PDIP, CDIP, SOIC, SOP, and TSSOP
(Top View)



CD4052B E, M, NS, and PW Package
16-Pin PDIP, CDIP, SOP, and TSSOP
(Top View)



CD4053B E, M, NS, and PW Package
16-Pin PDIP, CDIP, SOP, and TSSOP
(Top View)



Pin Functions CD4051B

PIN		I/O	DESCRIPTION
NO.	NAME		
1	CH 4 IN/OUT	I/O	Channel 4 in/out
2	CH 6 IN/OUT	I/O	Channel 6 in/out
3	COM OUT/IN	I/O	Common out/in
4	CH 7 IN/OUT	I/O	Channel 7 in/out
5	CH 5 IN/OUT	I/O	Channel 5 in/out
6	INH	I	Disables all channels. See Table 1.
7	V_{EE}	—	Negative power input
8	V_{SS}	—	Ground
9	C	I	Channel select C. See Table 1.
10	B	I	Channel select B. See Table 1.
11	A	I	Channel select A. See Table 1.
12	CH 3 IN/OUT	I/O	Channel 3 in/out
13	CH 0 IN/OUT	I/O	Channel 0 in/out
14	CH 1 IN/OUT	I/O	Channel 1 in/out
15	CH 2 IN/OUT	I/O	Channel 2 in/out
16	V_{DD}	—	Positive power input

maxim integrated™

MAX7219/MAX7221

Serially Interfaced, 8-Digit LED Display Drivers

General Description

The MAX7219/MAX7221 are compact, serial input/output common-cathode display drivers that interface microprocessors (µPs) to 7-segment numeric LED displays of up to 8 digits, bar-graph displays, or 64 individual LEDs. Included on-chip are a BCD code-B decoder, multiplex scan circuitry, segment and digit drivers, and an 8x8 static RAM that stores each digit. Only one external resistor is required to set the segment current for all LEDs. The MAX7221 is compatible with SPI™, QSPI™, and MICROWIRE™, and has slew-rate-limited segment drivers to reduce EMI.

A convenient 4-wire serial interface connects to all common µPs. Individual digits may be addressed and updated without rewriting the entire display. The MAX7219/MAX7221 also allow the user to select code-B decoding or no-decode for each digit.

The devices include a 150µA low-power shutdown mode, analog and digital brightness control, a scan-limit register that allows the user to display from 1 to 8 digits, and a test mode that forces all LEDs on.

For applications requiring 3V operation or segment blinking, refer to the MAX6951 data sheet.

Applications

Bar-Graph Displays
Industrial Controllers
Panel Meters
LED Matrix Displays

Features

- **10MHz Serial Interface**
- **Individual LED Segment Control**
- **Decode/No-Decode Digit Selection**
- **150µA Low-Power Shutdown (Data Retained)**
- **Digital and Analog Brightness Control**
- **Display Blanked on Power-Up**
- **Drive Common-Cathode LED Display**
- **Slew-Rate Limited Segment Drivers for Lower EMI (MAX7221)**
- **SPI, QSPI, MICROWIRE Serial Interface (MAX7221)**
- **24-Pin DIP and SO Packages**

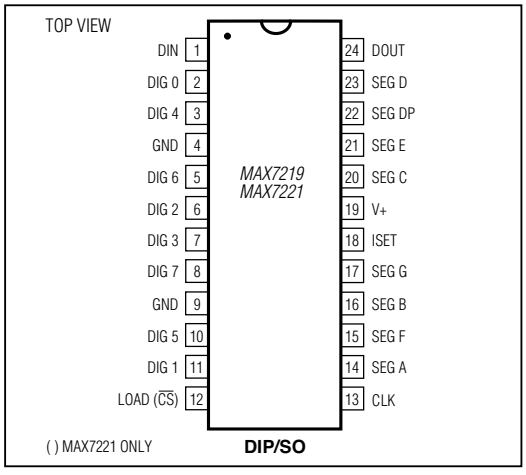
Ordering Information

PART	TEMP RANGE	PIN-PACKAGE
MAX7219CNG	0°C to +70°C	24 Narrow Plastic DIP
MAX7219CWG	0°C to +70°C	24 Wide SO
MAX7219C/D	0°C to +70°C	Dice*
MAX7219ENG	-40°C to +85°C	24 Narrow Plastic DIP
MAX7219EWG	-40°C to +85°C	24 Wide SO
MAX7219ERG	-40°C to +85°C	24 Narrow CERDIP

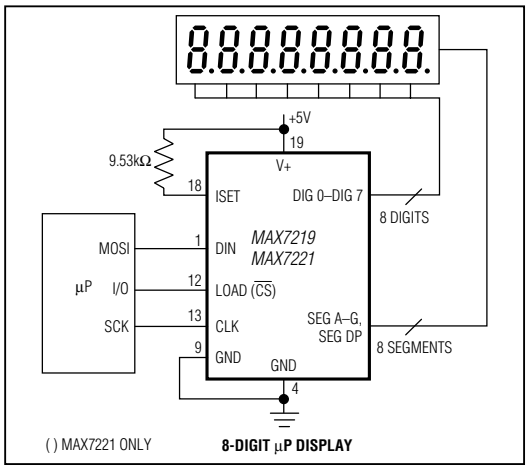
Ordering Information continued at end of data sheet.

**Dice are specified at T_A = +25°C.*

Pin Configuration



Typical Application Circuit



SPI and QSPI are trademarks of Motorola Inc. MICROWIRE is a trademark of National Semiconductor Corp.

For pricing, delivery, and ordering information, please contact Maxim Direct at 1-888-629-4642, or visit Maxim's website at www.maximintegrated.com.

19-4452; Rev 4; 7/03

Henry's Bench

a place for the electronic hobbyist

Menu

KEYES KY-040 ARDUINO ROTARY ENCODER USER MANUAL

Contents [show]

The Keyes KY-040 Rotary Encoder

The Keyes KY-040 rotary encoder is a rotary input device (*as in knob*) that provides an indication of how much the knob has been rotated AND what direction it is rotating in.

Its a great device for stepper and servo motor control. You could also use it to control devices like digital potentiometers.

Rotary Encoder Basics

A rotary encoder has a fixed number of positions per revolution. These positions are easily felt as small "clicks" you turn the encoder.
The Keyes module that I have has thirty of these positions.

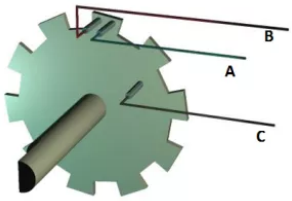
On one side of the switch, there are three pins. They are normally referred to as A, B and C. In the case of the KY-040, they are oriented as shown.

Inside the encoder there are two switches. Once switch connects pin A to pin C and the other switch connects pin B to C.

In each encoder position, both switches are either opened or closed. Each click causes these switches to change states as follows:

- **If both switches were closed**, turning the encoder either clockwise or counterclockwise one position will cause both switches to open
- **If both switches are open**, turning the encoder either clockwise or counterclockwise one position will cause both switches to close.

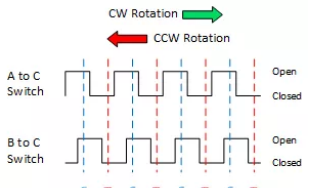
The illustration below is representative of how the switch is constructed.



As you can see, the angular position of the A terminal and the B terminal is such that:

- **Rotating the switch clockwise** will cause the switch connecting A and C to change states first.
- **Rotating the switch counterclockwise** will cause the switch connecting B and C to change states first.

If we were to represent the opening an closing of the switches as wave forms, it would look something like this.



WELCOME!!

Privacy Policy

About Me

ON EBAY...

ON AMAZON...

TOP POSTS & PAGES


The ACS712 Current Sensor with an Arduino


ACS712 Current Sensor User Manual


MAX31855 Arduino K Thermocouple Sensor: Manual and Tutorial


Keyes KY-040 Arduino Rotary Encoder User Manual


U8glib Arduino OLED Tutorial 1: Hello World on Steroids

FOLLOW ME ON TWITTER

Tweets Follow

Capn Fatz 1 Jun
@CaptainFatz
Simple and straightforward.
henrysbench.capnfatz.com/henrys-bench/a...

Capn Fatz 31 May
@CaptainFatz
Playing Music on CNC
crazymakerprojects.capnfatz.com/playing-music-...
pic.twitter.com/M2oLcFsmfc

Capn Fatz 28 May
@CaptainFatz
Arduino Goofy Eyes
crazymakerprojects.capnfatz.com/arduino-goofy-...
pic.twitter.com/MMWpRKD55i

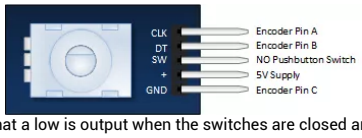
Essentially, determining which switch changed states first is how the direction of rotation of rotating is determined.

If A changed states first, the switch is rotating in a clockwise direction.

If B changed states first, the switch is rotating in a counter clockwise direction.

KY-040 Pin Outs

The pin outs for this rotary encoder are identified in the illustration below.



The module is designed so that a low is output when the switches are closed and a high when the switches are open.

The low is generated by placing a ground at Pin C and passing it to the CLK and DT pins when switches are closed.

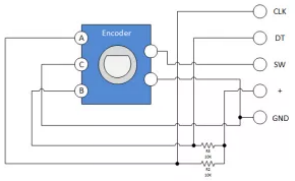
The high is generated with a 5V supply input and pullup resistors, such that CLK and DT are both high when switches are open.

Not previously mentioned is the existence of of push button switch that is integral to the encoder. If you push on the shaft, a normally open switch will close. The feature is useful if you want to change switch function. For example, you may wish to have the ability to between coarse and fine adjustments.

Keyes Rotary Encoder Schematic

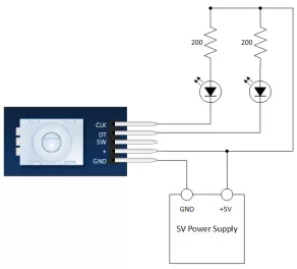
A schematic for this module is provided below. R2 and R3 in the schematic are pull up resistors.

Click on it for a bigger view.



Keyes KY-040 Evaluation Circuit

Successfully implementing the Rotary Encoder into any project requires a clear understanding of everything that has been discussed thus far. If you're still a little fuzzy, you may wish to throw together the evaluation circuit illustrated below:



VERY SLOWLY rotate then encoder shaft both clockwise and counterclockwise. Notice which LEDs change state first with rotation.

KY-040 Arduino Tutorial

Module Connection to the Arduino

Pretty straight forward... All you need to do is connect four wires to the module.

Capn Fatz 27 May
@CaptainFatz
1000W Battery Powered LED Flashlight
crazymakerprojects.capnfatz.com/1000w-battery-...
pic.twitter.com/SZKGSfAIUB

Capn Fatz 25 May
@CaptainFatz
Arduino Coca Cola Can Piano
crazymakerprojects.capnfatz.com/arduino-coca-c...
pic.twitter.com/jdcxXjW16N

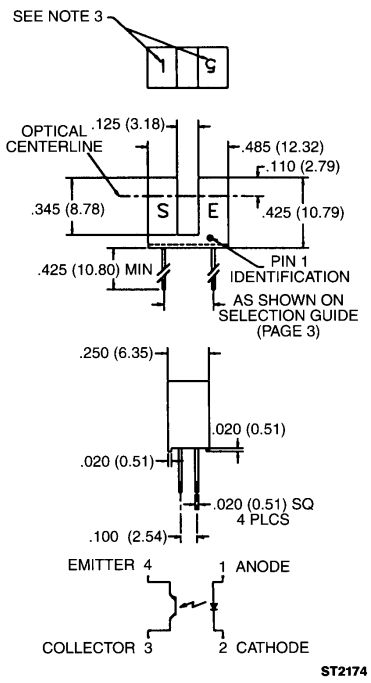
Tweet to @CaptainFatz

FAIRCHILD
SEMICONDUCTOR™

SLOTTED OPTICAL SWITCH

QVA SERIES

PACKAGE DIMENSIONS



NOTES:
1. DIMENSIONS ARE IN INCHES (mm).
2. TOLERANCE IS ±.010 (.25) UNLESS OTHERWISE SPECIFIED.
3. NUMBER INDICATES APERTURE SIZE. (5 = .050", 1 = .010")

DESCRIPTION

The QVA series of switches is designed to allow the user maximum flexibility in applications. Each switch consists of an infrared emitting diode facing an NPN phototransistor across a .125" (3.18 mm) gap. A unique housing design provides a smooth external surface to prevent dust and dirt buildup while molded internal apertures give precise positioning and also provide protection from ambient light interference.

FEATURES

- Ambient light and dust protection.
- Lead spacing available at .220", .300", or .320".
- .010" and .050" apertures.

Atmel

ATmega328P

8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash

DATASHEET

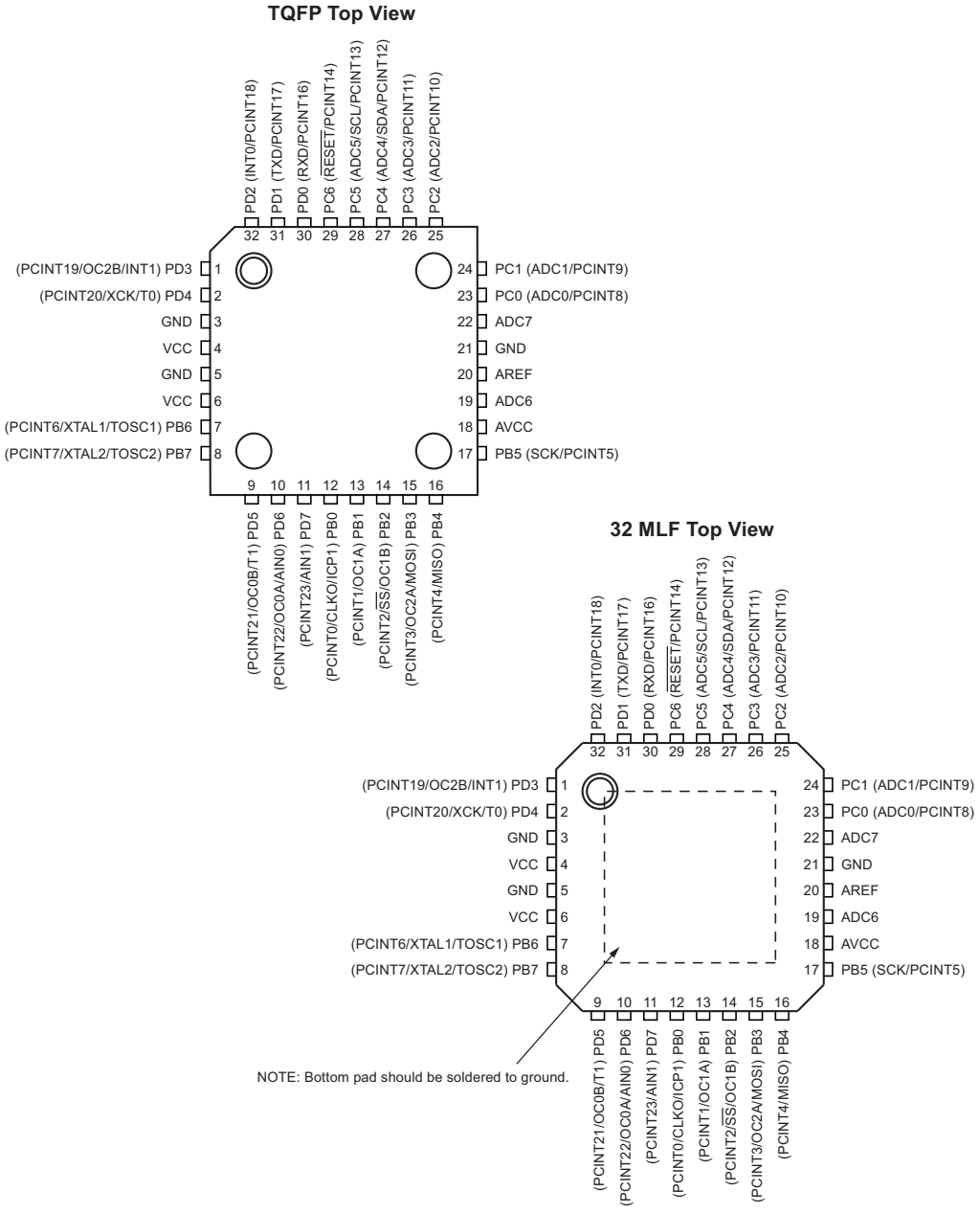
Features

- High performance, low power AVR® 8-bit microcontroller
- Advanced RISC architecture
  - 131 powerful instructions – most single clock cycle execution
  - 32 × 8 general purpose working registers
  - Fully static operation
  - Up to 16MIPS throughput at 16MHz
  - On-chip 2-cycle multiplier
- High endurance non-volatile memory segments
  - 32K bytes of in-system self-programmable flash program memory
  - 1Kbytes EEPROM
  - 2Kbytes internal SRAM
  - Write/erase cycles: 10,000 flash/100,000 EEPROM
  - Optional boot code section with independent lock bits
    - In-system programming by on-chip boot program
    - True read-while-write operation
  - Programming lock for software security
- Peripheral features
  - Two 8-bit Timer/Counters with separate prescaler and compare mode
  - One 16-bit Timer/Counter with separate prescaler, compare mode, and capture mode
  - Real time counter with separate oscillator
  - Six PWM channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    - Temperature measurement
  - Programmable serial USART
  - Master/slave SPI serial interface
  - Byte-oriented 2-wire serial interface (Phillips I^2C compatible)
  - Programmable watchdog timer with separate on-chip oscillator
  - On-chip analog comparator
  - Interrupt and wake-up on pin change
- Special microcontroller features
  - Power-on reset and programmable brown-out detection
  - Internal calibrated oscillator
  - External and internal interrupt sources
  - Six sleep modes: Idle, ADC noise reduction, power-save, power-down, standby, and extended standby

7810D-AVR-01/15

1. Pin Configurations

Figure 1-1. Pinout



Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V

8-bit Atmel Microcontroller with 16/32/64KB In-System Programmable Flash

DATASHEET

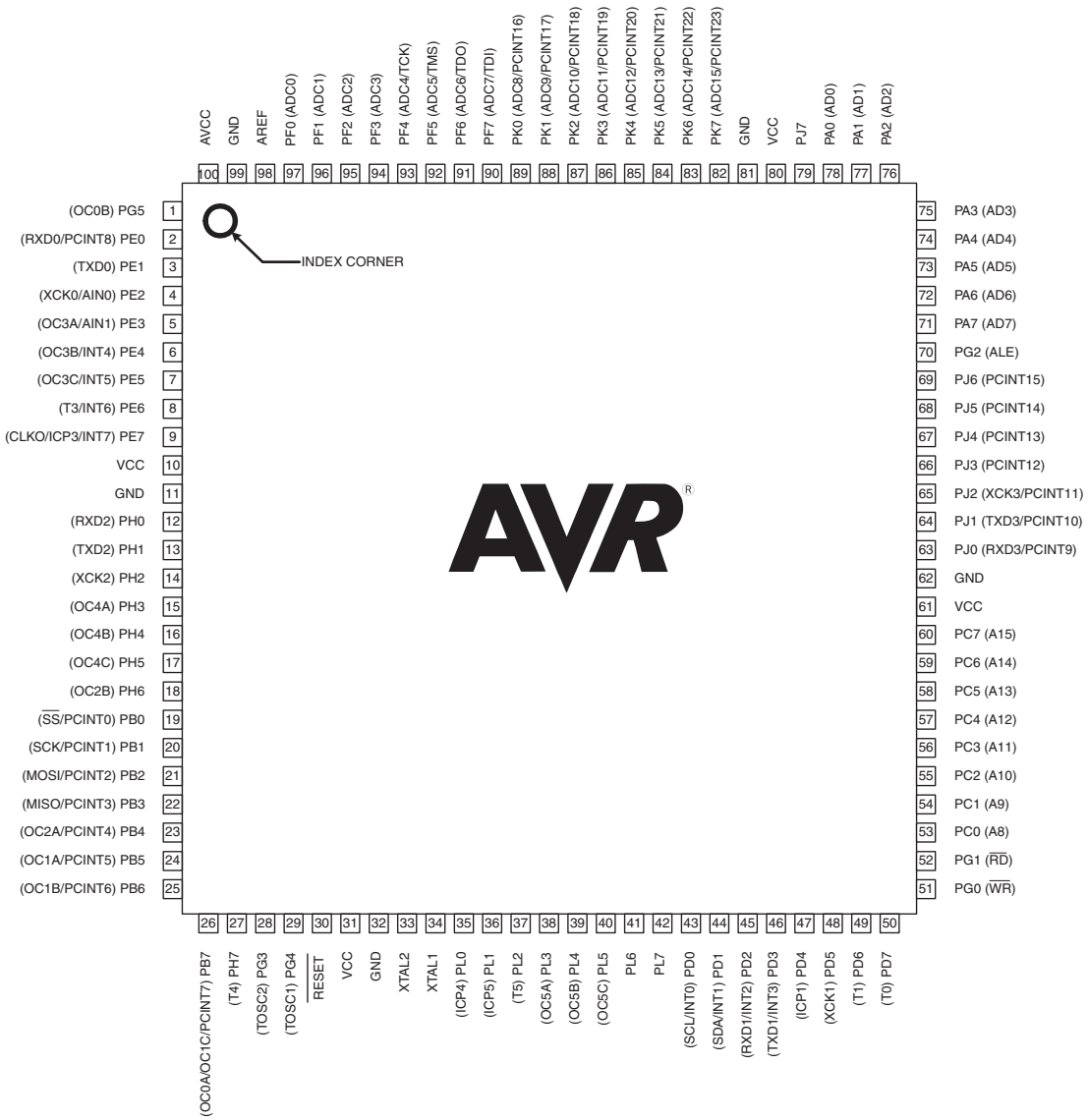
Features

- High Performance, Low Power Atmel® AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 135 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 × 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16MHz
  - On-Chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 64K/128K/256KBytes of In-System Self-Programmable Flash
  - 4Kbytes EEPROM
  - 8Kbytes Internal SRAM
  - Write/Erase Cycles:10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/ 100 years at 25°C
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
    - Endurance: Up to 64Kbytes Optional External Memory Space
- Atmel® QTouch® library support
  - Capacitive touch buttons, sliders and wheels
  - QTouch and QMatrix acquisition
  - Up to 64 sense channels
- JTAG (IEEE® std. 1149.1 compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - Four 16-bit Timer/Counter with Separate Prescaler, Compare- and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Four 8-bit PWM Channels
  - Six/Twelve PWM Channels with Programmable Resolution from 2 to 16 Bits (ATmega1281/2561, ATmega640/1280/2560)
  - Output Compare Modulator
  - 8/16-channel, 10-bit ADC (ATmega1281/2561, ATmega640/1280/2560)
  - Two/Four Programmable Serial USART (ATmega1281/2561, ATmega640/1280/2560)
  - Master/Slave SPI Serial Interface
  - Byte Oriented 2-wire Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 54/86 Programmable I/O Lines (ATmega1281/2561, ATmega640/1280/2560)
  - 64-pad QFN/MLF, 64-lead TQFP (ATmega1281/2561)
  - 100-lead TQFP, 100-ball CBGA (ATmega640/1280/2560)
  - RoHS/Fully Green
- Temperature Range:
  - -40°C to 85°C Industrial
- Ultra-Low Power Consumption
  - Active Mode: 1MHz, 1.8V: 500µA
  - Power-down Mode: 0.1µA at 1.8V
- Speed Grade:
  - ATmega640V/ATmega1280V/ATmega1281V:
    - 0 - 4MHz @ 1.8V - 5.5V, 0 - 8MHz @ 2.7V - 5.5V
  - ATmega2560V/ATmega2561V:
    - 0 - 2MHz @ 1.8V - 5.5V, 0 - 8MHz @ 2.7V - 5.5V
  - ATmega640/ATmega1280/ATmega1281:
    - 0 - 8MHz @ 2.7V - 5.5V, 0 - 16MHz @ 4.5V - 5.5V
  - ATmega2560/ATmega2561:
    - 0 - 16MHz @ 4.5V - 5.5V

2549Q–AVR–02/2014

1. Pin Configurations

Figure 1-1. TQFP-pinout ATmega640/1280/2560



Códigos de los microcontroladores

```
/*AUTOR:Augusto Samuel Hernández Martín
 *fECHA: 22 Julio de 2019
 *
 *DESCRIPCION:Programa que realiza la comunicacion serial con fSX mediante el SW Link2fS que proporciona los
parametros del avion
 *y permite enviarle instrucciones al simulador
 *
 *Se realiza la comunicación con los diferentes esclavos mediante un bus I2C y con los MAX7219 mediante SPI
 */
#define ENCODER_DO_NOT_USE_INTERRUPTS
//*************************************************************************************************************
********************
//                                          INCLUSION DE LIBRERIAS
//*************************************************************************************************************
********************

#include<Encoder.h> //Para usar los encoders
#include <Switch.h> //Libreria propia para grestionar los antirrebotes y los interruptores y pulsadores
#include <I2C.h> //Libreria propia para la comunicaicón con los PCF8574 y con los 2 Arduinos Nano de esclavos
#include <LCD_I2C.h> //Libreria para conectarse al I2C y manejar un LCD. Modificada
#include <Funciones.h> //Libreria propia con las funciones para escirbir y leer por serial
#include <MAX7219.h> //Libreria propia para manejar los displays 7 segmentos del modulo de radio
#include <Servo.h>  //Libreria para los servos
#include <Wire.h>//Libreria para I2C necesaria
#include <SPI.h> //Libreria para usar el bus SPI

//*************************************************************************************************************
********************
//                                          DEfINICION DE CONSTANTES
//*************************************************************************************************************
********************

#define t true //Con el objetivo de minimizar el nombre de la constante
#define f false

//CONSTANTES PARA CALCULOS EN ALGUNOS INSTRUMENtOS
//------Hort. Art. Eje Y -----------
#define HorArtMinY 140 //Angulo minimo dle servo
#define HorArtMaxY 40 //Angulo maximo del servo
#define pitchMax 60 //Valor maximo leido
#define pitchMin -60 //Valor minimo leido

//--------HorArt Eje X y Turn Coord------------
#define HorArtMinX 20 //Como en Hor.Art.Eje Y
#define HorArtMaxX 150
#define rollMax 45
#define rollMin -45
#define TCAnguloMax 95 //Angulo maximo del servo de Turn Coordinator
#define TCAnguloMin 0 //Angulo minimo del servo de Turn Coordinator

//--------Constantes para RPM----------
#define RPMMax 2400 //Valor maximo de RPM
#define RPMMin 0 //Valor minimo de RPM
#define AnguloRPMMax 108 //Angulo maximo del servo de RPM
#define AnguloRPMMin 8 //Angulo minimo del servo de RPM

//---------Constantes VS----------------
#define VSMax 2000
#define VSMin -2000
#define AnguloVSMin 21
#define AnguloVSMax 162

//--------Constantes del IAS-------------
#define IASMax 200
#define IASMin 40
#define AnguloIASMin 15
#define AnguloIASMax 125

//-------Constantes para indicadores de Fuel---------
#define FuelLMin 0
#define FuelLMax 100
```

```
#define AnguloFuelLMin 30
#define AnguloFuelLMax 140

#define FuelRMin 0
#define FuelRMax 100
#define AnguloFuelRMin 180
#define AnguloFuelRMax 60

//------Constantes para Temepratura--------
#define TempMin 75
#define TempMax 250
#define AnugloTempMin 0
#define AnguloTempMax 120

//-------Constantes para presion -----------
#define PressMin 0
#define PressMax 115
#define AnguloPressMin 155
#define AnguloPressMax 35


// IDENtIfICADORES DE ESCLAVOS Y MENSAJES EN I2C
#define idAltimetro 1 //Se envia un 1 antes del mensaje del bus para indicar que se recibirá la altitud
#define idHDG 2 //Se envia un 2 antes del mensaje para HDG
#define idConsignaHDG 3

//DIRECCIONES fÍSICAS DE LOS ESCLAVOS I2C
#define DireccionLEDAutoP 0x38 //OK
#define DireccionLEDtren 0x3A//OK
#define DireccionLEDWarning 0x39 //OK
#define DireccionPantallaAP 0x27 //OK
#define idEsclavo1 0x04

//PINES MUXA,B,C,D,E,f
#define MuxBit0 22 //La entrada del bit 0 de todos los MUX esta conectada a 22
#define MuxBit1 23 //Todos los bit 1 a 23
#define MuxBit2 24 //Todos los bit 2 a 24

//Pines MUXG Lee todas las entradas de los mux que tiene anteriormente
#define MuxGBit0 25 //Las entradas del MUX que selecciona que MUX lee estan en 25, 26 y 27
#define MuxGBit1 26
#define MuxGBit2 27
#define MuxInSignal 28 //Salida de todas las lecturas

//PINES tREN AtERRIZAJE Y fLAPS
#define Recogertren 28 //Se une a MUX A y se lee con secuencia 000000
#define Extendertren 28 //000001
#define flaps0 28 //000101
#define flaps10 28 //000110
#define flaps20 28//000111
#define flaps30 28//001000


//PINES RADIO
#define COM_Dt 30 //DT encoder Radio
#define COM_CLK 29 //CLK Encoder Radio
#define CambiarEscalaCom 28 //Intercambia escala de 1Mhz o 25Khz. 000011
#define SwapCOM 28 //Intercambia frecuencia en emision y stdby. 000100
#define NAV_Dt 38 //DT encoder NAV
#define NAV_CLK 31 //CLK Encoder NAV
#define CambiarEscalaNAV 28 //100100 Permite seleccionar la escala de 1Mhz o 25kHz
#define SwapNAV 28 //Intercambia NAV Stand-By y Activa. Se lee con 100101
#define COM1_On 28 //100000
#define NAV1_On 28 //100001
#define DME_On 28 //100010
#define ADf_On 28 //100011


//PINES TRANSPONDER
#define Dt_XPDR 36
#define CLK_XPDR 37
```

```
#define CambiaDigitoXPDR 28 //Se lee con 000010. Con cada pulsación elige el digito siguiente

//PINES PILOtO AUtOMAtICO
#define APOnOff 28 //Activa y desactiva el AP. 001000
#define APR 28
#define HDG 28 //Activa o desactiva el HDG del AP. 001100
#define NAV 28 //Pin para activar NAV del AP. 001101
#define ALt 28 //Para activar la altitud del AP. 001110
#define AltVs_Dt 34
#define AltVs_CLK 35
#define SwapAltVs 28 //Permite elegir si cambiar Altitud o VS con el encoder en el AP . 011001
#define HDG_Dt  32//Para el encoder de girar el XPDR
#define HDG_CLK 33 // Para girar el XPDR
#define CalibraALt_Dt 40 //Para el encoder que ajusta los mmHg del Altimetro
#define CalibraALt_CLK 39

//PINES PANEL DE LUCES, AVIONICA Y ARRANQUE
#define fuelPump 28//Paara la bomba de combustible. 001111
#define LucesBeacon 28 //Para luces Beacon. 010000
#define LucesLand 28 // Para luces de aterrizaje. 010001
#define Lucestaxi 28 // Para luces de taxi. 010010
#define LucesNav 28// Para las luces de navegacion.010011
#define LucesStrobe 28 // Para las luces estroboscópicas. 010100
#define Pitot 28 //Para activar los calefactores de los tubos de Pitot. 010101
#define Alternador 28 // Para activar el generador de corriente (Alternador) del motor. 010110
#define Bateria 28 // Para conectar la alimentación de la batería. 010101
#define Avionica 28 // Para activar y dar alimentacion a los intrumentos (avionica). 011000
#define Magneto_Off 28 //Apaga el Magneto de arranque. 011010
#define Magneto_R 28 //Enciende un conjunto de bujias del motor. 011011
#define Magneto_L 28 //Enciende el otro conjunto de bujias. 011100
#define Magneto_Both 28 //Enciende ambos conjuntos. 011101
#define Magneto_Start 28 //Arranca el motor. 011110
#define Carb_Heater 28 //Enciende la calefaccion del carburador para arranque en frio. 001001
#define Luces_Panel 28//Enciende la iluminación de los paneles del avión. SIN IMPLEMENtAR. 011111


//PINES PWM PARA SERVOS (EN MEGA DEL 2 AL 13)
#define ServoVerticalSpeed 5 //Pin para el servo de velocidad vertical
#define ServofuelActual_R 3//Pin para servo que muestra el estado del tanque combustible derecho
#define ServofuelActual_L 4//Pin para servo que muestra el estado del tanque combustible izquierdo
#define ServoturnCoordinator 2 //Pin para servo que muestra inclinacion del avion
#define ServoHorizonteArtificialEjeX 10 //Eje X del Hor. Artificial
#define ServoHorizonteArtificialEjeY 11 //Eje Y del Hor. Artificial
#define ServoIAS 7 //Servo que muestra el consumo de corriente de los instrumentos
#define Servotemperatura 6 //Servo que muestra la temperatura del motor
#define ServoPresion 8 //Servo que indica el nivel de vacio del avión
#define ServoRPM 9


//*************************************************************************************************************
********************
//                                          CONSTRUCCIONES DE OBJETOS DE CLASES
//*************************************************************************************************************
********************

//Clase MAX7219
  MAX7219 Radio = MAX7219();


//CLASE I2C PARA EXPANSORES PCf8574
  I2C ModuloAP (DireccionLEDAutoP);
  I2C Modulotren (DireccionLEDtren);
  I2C ModuloWarning (DireccionLEDWarning);
  I2C EsclavoUno (idEsclavo1);


//CLASE LCD I2C
  LiquidCrystal_PCF8574 lcdAutoP(DireccionPantallaAP); //Se crea objeto de clase LCD para I2C

//CLASE SERVO
  Servo gaugeIAS, gaugefuel_L,gaugefuel_R, gaugeturnCoordinator, gaugePresion, gaugeHorizonteX,
```

```
gaugeHorizonteY, gaugeVerticalSpeed, gaugetemperatura, gaugeRPM;

//CLASE ENCODER
  Encoder encoderCOM(COM_Dt,COM_CLK); //Permite seleccionar la frecuencia de radio COM
  Encoder encoderNAV(NAV_Dt, NAV_CLK); //Permite sintonizar la frecuencia NAV
  Encoder encoderXPDR(Dt_XPDR,CLK_XPDR); //Permite modificar el valor de cada digito del transpondedor
  Encoder encoderAltVS(AltVs_Dt, AltVs_CLK); //Permite ajustar la velocidad vertical y altitud
  Encoder encoderHDG(HDG_Dt,HDG_CLK);//Permite ajustar el rumbo del AP
  Encoder encoderCalibrarAltimetro (CalibraALt_Dt, CalibraALt_CLK); //Permite calibrar el Altimetro (mmHg)


//---------------------------------MUXA-----------------------------------------------------------
  Switch InterruptorGearUp (Recogertren,INPUT,MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1, MuxBit0,f, f, f,
f, f, f);
  Switch InterruptorGearDown = Switch(Extendertren, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1,
MuxBit0,f, f, f, f, f, t);
  Switch ButtonCambiarEscalaCOM = Switch(CambiarEscalaCom, INPUT_PULLUP, MuxGBit2, MuxGBit1, MuxGBit0,
MuxBit2, MuxBit1, MuxBit0,f, f, f, f, t, t);
  Switch ButtonSwapCOM = Switch(SwapCOM, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1, MuxBit0,f, f,
t, t, f, f);
  Switch Interruptorflaps0 = Switch(flaps0, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1, MuxBit0,f,
f, f, t, f, t);
  Switch Interruptorflaps10 = Switch(flaps10, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1,
MuxBit0,f, f, f, t, t, f);
  Switch Interruptorflaps20 = Switch(flaps20, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1,
MuxBit0,f, f, f, t, t, t);

//---------------------------------MUXB-----------------------------------------------------------
  Switch Interruptorflaps30 = Switch(flaps30, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1,
MuxBit0,f, f, t, f, f, f);
  Switch InterruptorCarbHt = Switch(Carb_Heater, INPUT,MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1,
MuxBit0,f, f, t, f, f, t );
  Switch InterruptorAP = Switch(APOnOff, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1, MuxBit0,f, f,
t, f, t, f);
  Switch InterruptorAPR = Switch(APR, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1, MuxBit0,f, f, t,
f, t, t);
  Switch InterruptorHDG = Switch(HDG, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1, MuxBit0,t, f, t,
t, f, t); //Realmente va en 000101 pero por problmas en la PCB se pone en 101101 en I43
  Switch InterruptorNAV = Switch(NAV, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1, MuxBit0,f, f, t,
t, f, t);
  Switch InterruptorALt = Switch(ALt, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1, MuxBit0,f, f, t,
t, t, f);
  Switch InterruptorfuelPump = Switch(fuelPump, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1,
MuxBit0,f, f, t, t, t, t);

//---------------------------------MUXC-----------------------------------------------------------
  Switch InterruptorBeaconLight = Switch(LucesBeacon, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1,
MuxBit0,f, t, f, f, f, f);
  Switch InterruptorLandLight = Switch (LucesLand, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1,
MuxBit0,f, t, f, f, f, t);
  Switch InterruptortaxiLight = Switch (Lucestaxi, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1,
MuxBit0,f, t, f, f, t, f);
  Switch InterruptorNavLight = Switch(LucesNav, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1,
MuxBit0,f, t, f, f, t, t);
  Switch InterruptorStrobeLight = Switch(LucesStrobe, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1,
MuxBit0,f, t, f, t, f, f);
  Switch InterruptorPitot = Switch (Pitot, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1, MuxBit0,f,
t, f, t, f, t);
  Switch InterruptorAlternador = Switch(Alternador,INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1,
MuxBit0,f, t, f, t, t, f);
  Switch InterruptorBateria = Switch(Bateria,INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1, MuxBit0,f,
t, f, t, t, t);

//---------------------------------MUXD-----------------------------------------------------------
  Switch InterruptorAvionica = Switch(Avionica,INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1,
MuxBit0,f, t, t, f, f, f);
  Switch ButtonSwapALt_VS = Switch(12, INPUT_PULLUP, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1,
MuxBit0,f, t, t, f, f, t); //MODIFICADO PARA PODER USAR EN MI PLACA POR FALLO EN CTO
  Switch MagnetoOff = Switch(Magneto_Off, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1, MuxBit0,f, t,
t, f, t, f);
  Switch MagnetoR = Switch(Magneto_R, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1, MuxBit0,f, t, t,
```

```
f, t, t);
  Switch MagnetoL = Switch(Magneto_L, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1, MuxBit0,f, t, t,
t, f, f);
  Switch MagnetoBoth = Switch(Magneto_Both, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1, MuxBit0,f,
t, t, t, f, t);
  Switch MagnetoStart = Switch(Magneto_Start, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1,
MuxBit0,f, t, t, t, t, f);
  Switch InterruptorPanelLight = Switch(Luces_Panel, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1,
MuxBit0,f, t, t, t, t, t);

//--------------------------------MUXE-----------------------------------------------------------------

  Switch ButtonDME = Switch(DME_On, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1, MuxBit0,t, f, f, f,
f, t);
  Switch ButtonADF = Switch(ADf_On, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1, MuxBit0,t, f, f, f,
t, f);
  Switch ButtonCambiarEscalaNAV = Switch(CambiarEscalaNAV, INPUT_PULLUP, MuxGBit2, MuxGBit1, MuxGBit0,
MuxBit2, MuxBit1, MuxBit0,t, f, f, t, f, f);
  Switch ButtonSwapNAV = Switch(SwapNAV, INPUT, MuxGBit2, MuxGBit1, MuxGBit0, MuxBit2, MuxBit1, MuxBit0,t, f,
f, t, f, t);



//*************************************************************************************************************
*********************
//                                          DEfINICION VARIABLES GLOBALES
//*************************************************************************************************************
*********************

  char LecturaIdentificador; //Guarda el primer caracter leido del Link2fS ('<','=',...)

  int digitoElegidoXPDR=0; //Guarda valores entre 0 y 4. De 1 a 4 son los digitoos a editar en 0 se activa el
XPDR

//String que guardan las Lecturas del AutoPilot
  String HDGElegida;
  int altitudActual,altitudAnterior;
  int HDGActual,HDGAnterior;
  int HDGConsignaActual, HDGConsignaAnterior;

  String secuenciaRadio="00007777000000011110000"; //Se almacena la secuencia de numeros a enviar para la
radio.

  int VerticalSpeed, airSpeed;
  int pitch, roll;
  int escalafrecuenciaCOM=1 ;
  int escalafrecuenciaNAV=1;
  int ModoAltOVs=1;
  int fuel_L, fuel_R;
  int Amperios, vacio,temperatura;
  int Presion;
  int RPM1;

  long antigua_pos_encoder_XPDR = -999;
  long antigua_pos_encoder_NAV = -999;
  long antigua_pos_encoder_COM = -999;
  long antigua_pos_encoder_ALT = -999;
  long antigua_pos_encoder_HDG = -999;

  bool esclavo_finalizado = true; //Bit que indica si el esclavo ya indico que acabo de mover motores o no.

//Definicion de los Bytes a enviar para multiplexar las salidas de LEDS
  byte panelAutoP = 0b11111111;
  byte lucestren = 0b00000000;
  byte lucesWarning = 0b11111111;

//*************************************************************************************************************
*********************
//                                          COMIENZO DE CONFIGURACION
//*************************************************************************************************************
```

```
**********************

void setup() {

  //Inicializar pines de los MUX
    pinMode (MuxBit0, OUTPUT);
    pinMode (MuxBit1, OUTPUT);
    pinMode (MuxBit2, OUTPUT);
    pinMode (MuxGBit0, OUTPUT);
    pinMode (MuxGBit1, OUTPUT);
    pinMode (MuxGBit2, OUTPUT);
    pinMode(MuxInSignal, INPUT);

  //Inicializacion configuracion para Lecturas por buses
    Serial.begin (115200);//Definimos velocidad del puerto serie
    Wire.begin(); //Se define este Arduino como Maestro


  //-------------------------------------------------CONfIGURACION LCD DEL
AUtOPILOt-------------------------------------------------
    lcdAutoP.begin(16,2);
    lcdAutoP.setBacklight(100);
    lcdAutoP.clear();
    lcdAutoP.imprimeLCD ( 7, 0, "ALt 00000");
    lcdAutoP.imprimeLCD(7, 1, "VS  +0000");

  //-----------------------------------------------------------CONfIGURACION
SERVOS-----------------------------------------------------
    gaugeRPM.attach(ServoRPM);
    gaugefuel_L.attach(ServofuelActual_L);
    gaugefuel_R.attach(ServofuelActual_R);
    gaugeturnCoordinator.attach(ServoturnCoordinator);
    gaugePresion.attach(ServoPresion);
    gaugetemperatura.attach(Servotemperatura);
    gaugeVerticalSpeed.attach(ServoVerticalSpeed);
    gaugeHorizonteX.attach (ServoHorizonteArtificialEjeX);
    gaugeHorizonteY.attach (ServoHorizonteArtificialEjeY);
    gaugeIAS.attach(ServoIAS);


  //-------------------------------------------------SPI
RADIO-----------------------------------------------------------------------
    SPI.begin ();
    //CLASE MAX7219
    Radio.init(0xFF,0,15,1); //Solo numeros, modo normal, máxima iluminación
} //FIN DE SETUP


//*************************************************************************************************************
**********************
//                                          COMIENZO EL LOOP
//*************************************************************************************************************
**********************

void loop() {

 //Lectura PARA DATOS QUE COMIENZAN POR '='

    if( Serial.available() > 0 ){ //Si hay datos para leer en el buffer
      LecturaIdentificador = leeCaracter();
      if (LecturaIdentificador == '=') { //Se lee un '=' como simbolo que es 67 en ASCII
        LecturaIdentificador = leeCaracter(); //Se lee el siguiente caracter
        switch (LecturaIdentificador) { //Para los distintos casos despues del =
          case 'A':{ //Si se lee =A (ASCII 65) es la frecuencia principal de radio COM1
            String frecuenciaCOMActiva = leeString (7);
            secuenciaRadio[20]= frecuenciaCOMActiva[5];
            secuenciaRadio[21]= frecuenciaCOMActiva[4];
            secuenciaRadio[22]= frecuenciaCOMActiva[2];
            secuenciaRadio[23]= frecuenciaCOMActiva[1];
            break;
          }
```

```
          case 'B': {//=B (ASCII 66) frecuencia de StandBy de COM1
            String frecuenciaCOMStandBy = leeString (7); //Se esperan 7 caracteres
            secuenciaRadio[8]= frecuenciaCOMStandBy[5];
            secuenciaRadio[9]= frecuenciaCOMStandBy[4];
            secuenciaRadio[10]= frecuenciaCOMStandBy[2];
            secuenciaRadio[11]= frecuenciaCOMStandBy[1];
            break;
          }
          case 'E':{
            String frecuenciaNAVActiva = leeString (6);
            secuenciaRadio[12]= frecuenciaNAVActiva[5];
            secuenciaRadio[13]= frecuenciaNAVActiva[4];
            secuenciaRadio[14]= frecuenciaNAVActiva[2];
            secuenciaRadio[15]= frecuenciaNAVActiva[1];
            break;
          }

          case 'F': {
            String frecuenciaNAVStandBy = leeString (6); //Se esperan 6 caracteres
            secuenciaRadio[0]= frecuenciaNAVStandBy[5];
            secuenciaRadio[1]= frecuenciaNAVStandBy[4];
            secuenciaRadio[2]= frecuenciaNAVStandBy[2];
            secuenciaRadio[3]= frecuenciaNAVStandBy[1];
            break;
          }
          case 'J': {// =J(ASCII 74) Codigo del transpondedor (XPDR)
            String valorXPDR = leeString (4); //Se esperan 4 caracteres
            secuenciaRadio[4]= valorXPDR[3];
            secuenciaRadio[5]= valorXPDR[2];
            secuenciaRadio[6]= valorXPDR[1];
            secuenciaRadio[7]= valorXPDR[0];
            break;
          }

          case 'a':{ //=a Indica si AP ON o Off
            char estadoAP = leeCaracter(); //Lee el caracter 1 o 0 del estado de AP
            if(estadoAP=='1'){ //El AutoPilot esta activo
              panelAutoP &= (0b11111110); //Poner a cero el led que correpsond epara encenderlo
            }
            else{ //El AutoPilot está inactivo
              panelAutoP=panelAutoP | (0b00000001); //Poner a cero el bit 4 para apagar la luz de AP
            }
            break;
          }

          case 'b':{ // =b Valor de la altitud introducida
            String altitudElegida = "ALt" + leeString (5); //Se esperan 5 caracteres
            //altitudElegida = "ALt " + altitudElegida; //Se coloca ALt por delante del valor para mostrarlo
en pantalla
              lcdAutoP.imprimeLCD ( 7, 0, altitudElegida);//Imprime Altitud Elegida en la posicion 6 de la
primera linea
            break;
          }

          case 'c':{ //=c es la velocidad vertical introducida en AP
            String velocidadVerticalElegida = "VS" + leeString(5); //Lee el simbolo +/- y los 4 digitos de la
velocidad vertical
           //velocidadVerticalElegida= "VS  " + velocidadVerticalElegida; //Se coloca VS por delante del valor
para mostrar en pantall
              lcdAutoP.imprimeLCD ( 7, 1, velocidadVerticalElegida); //Muestra en pantalla la Velocidad
Vertical
            break;
          }

          case 'd':{ // =d es el valor de HDG introducido
            HDGElegida = leeString(3); //Lee los tres valores del HDG elegido en AP
            HDGConsignaActual= HDGElegida.toInt(); //Se convierte a entero el valor de altitud
            if ((HDGConsignaActual != HDGConsignaAnterior)&& (esclavo_finalizado == true)){
              EsclavoUno.enviaEsclavo (idConsignaHDG, HDGConsignaActual); //Se envia por bus I2C
              HDGConsignaAnterior = HDGConsignaActual;
```

```
        }
        break;
      }

      case 'j':{ //=i es la señal de HDG activo o no
        char HDGOnOff= leeCaracter();
        if(HDGOnOff == '1'){
          lcdAutoP.imprimeLCD ( 0, 0, "HDG"); //Escribe HDG en la pantalla
          panelAutoP &= (0b11111011); //Pongo a 0 el Bit de la luz de HDG
        }
        else{
          lcdAutoP.imprimeLCD ( 0, 0, "   "); //Borra HDG de pantalla
          panelAutoP |= (0b00000100); //Pongo a cero el Bit del HDG
          lcdAutoP.imprimeLCD(0,0,HDGConsignaActual);
        }
        break;
      }

      case 'k':{ //=k es la señal de Altitud activa o no
        if(leeCaracter() == '1'){
          panelAutoP &= (0b11101111); //Pongo a 0 el correspondiente
        }
        else{
          panelAutoP |= (0b00010000); //Pongo a cero el correspondiente a ALt LED
        }
        break;
      }
      case 'm':{ //=m es la señal de Aproximacion activo o no
        if(leeCaracter() == '1'){
          lcdAutoP.imprimeLCD ( 0, 1, "APR");
          panelAutoP &= (0b11111101);
        }
        else{
          lcdAutoP.imprimeLCD( 0, 1, "   ");
          panelAutoP |= (0b00000010);
        }
        break;
      }
      case 'o':{ //=o es la señal de NAV activo o no
        if(leeCaracter() == '1'){
          panelAutoP &= (0b11110111); //Pongo xxxxxx0xx
          lcdAutoP.imprimeLCD ( 0, 1, "NAV"); //Escribe NAV en pantalla
        }
        else{
          panelAutoP |= 0b00001000; //Pongo a 1 el Bit correspondiente a NAV para apagarlo
          lcdAutoP.imprimeLCD ( 0, 1, "   "); //Borra NAV en pantalla
        }
        break;
      }
    }
  }//FIN DE Lectura IDENtIfICADOR '='


//Lectura DE DATOS  QUE COMIENZAN CON IDENTIFICADOR '<'

  if (LecturaIdentificador == '<') { //Leemos un '<'
    LecturaIdentificador = leeCaracter(); //Lee siguiente caracter
    switch (LecturaIdentificador) {
      case 'A':{ //<A  Estado de la rueda del tren delantero en %
        String valorleidorueda =  leeString(3);
        int porcentajeRuedaDelantera = valorleidorueda.toInt();
        if( porcentajeRuedaDelantera > 95){ // Mas de 99% de extencion, se enciende LEd de tren abajo
          lucestren &= (0b11111110); //Encender el Bit de tren delantero extendido (Bit 0)
          lucestren |= (0b00000010) ; //Apagar luz de tren delanterotransito (Bit 1)
        }
        else if( porcentajeRuedaDelantera  < 5) { //Menos de 5% tren recogido. Apaga LED
          lucestren |= (0b00000011); //Apago el Bit de tren delantero transito y de extendido (Bit 0 y 1)
        }
        else{ //En transito, enciende LED
          lucestren &= (0b11111101); //enciendo transito (Bit1)
          lucestren |= (0b00000001); //Apago Bit de tren delantero extendido (Bit 0)
```

```
            }
            break;
          }

          case 'B':{ //<B  Estado de la rueda izda. en %
            String valorleidorueda =  leeString(3);
            int porcentajeRuedaIzda = valorleidorueda.toInt();
            if( porcentajeRuedaIzda > 95){ // Mas de 99% de extencion, se enciende LEd de tren abajo y apaga
transito (Bits 4 y 5)
              lucestren &= (0b11101111); //Encendemos bit 4 (Tren abajo)
              lucestren |= (0b00100000); //Apagamos bit 5 (Tren transito)
            }
            else if( porcentajeRuedaIzda  < 5) { //Menos de 5% tren recogido. Apaga LED de transito y
extendido (Bits 4 y 5)
              lucestren |= (0b00110000); //Se apagan bit 4 y 5
            }
            else{ //En transito, enciende LED de transito (Bit 5) y apaga extendido (Bit 4)
              lucestren &= (0b11011111);//Encender bit 5
              lucestren |= (0b00010000); //Apagar bit 4
            }
            break;
          }

          case 'C':{ //<C  Estado de la rueda drcha. en %
            String valorleidorueda =  leeString(3);
            int porcentajeRuedaDrcha = valorleidorueda.toInt();
            if( porcentajeRuedaDrcha > 95){ // Mas de 99% de extencion, se enciende LEd de tren abajo dereho
(Bit 2) y apaga transito (Bit 3)
              lucestren |= (0b00001000); //Ponemos a 1 el que se apaga (Transito, bit 3)
              lucestren &= (0b11111011);//Ponemos a 0 el que se enciende (Extendido, bit 2)
            }
            else if( porcentajeRuedaDrcha < 5) { //Menos de 5% tren recogido. Apaga LED de extendido (Bit 2) y
transito (Bit 3)
              lucestren |= (0b00001100);
            }
            else{ //En transito, enciende LED de transito (Bit 3) y apaga extendido (Bit 2)
              lucestren |= (0b00000100); //Se apaga el bit 2 que es extendido
              lucestren &= (0b11110111); //Se enciende el bit 1 que es transito
            }
            break;
          }

          case 'D':{ // <D Se lee la Altitud Actual
            String altitudActualLeida = leeString(5); //Se esperan 5 caracteres
            altitudActual= altitudActualLeida.toInt(); //Se convierte a entero el valor de altitud
            if ( (altitudActual != altitudAnterior) && (esclavo_finalizado == true)){
              EsclavoUno.enviaEsclavo (idAltimetro, altitudActual); //Se envia por bus I2C
              altitudAnterior = altitudActual;
            }
            break;
          }

          case 'J':{ //<J Valor de HDG del avion
           String HDGActualLeido = leeString(3);//Se esperan 3 caracteres
           HDGActual = HDGActualLeido.toInt(); //Convierte el String a un numero entero con ese valor para HDG
           if ((HDGActual != HDGAnterior) && (esclavo_finalizado == true)){
              EsclavoUno.enviaEsclavo(idHDG, HDGActual); //Se envia el valor de HDG por I2C al esclavo
              HDGAnterior=HDGActual;
           }
           break;
          }

          case 'X':{ //<X  fuel Izquierdo
            String fuelLeido = leeString(3);
            fuel_L = fuelLeido.toInt(); //Convierte a entero el % de fuel restante
            break;
          }

          case 'Z':{ //<Z  fuel Izquierdo
            String fuelLeido = leeString(3);
            fuel_R = fuelLeido.toInt(); //Convierte a entero el % de fuel restante
```

```
        break;
      }

      case 'L':{ //<L Valor de velocidad verrtical
        String totalVerticalSpeedLeido = leeString(6);
        VerticalSpeed = totalVerticalSpeedLeido.toInt();
      }

      case 'P':{ //<P Valor de AirSpeed
        String airSpeedLeido = leeString(3); //Se espera 3 digitos
        airSpeed = airSpeedLeido.toInt(); //Convierte el valor a entero
      }

      case 'Q':{ // <Q Valor del Pitch
        String pitchLeido = leeString(6); //Se esperan 6 digitos +xxx.x
        pitch = pitchLeido.toInt();
        break;
      }

      case 'R':{ //<R roll (inclinacion lateral)
        String rollLeido = leeString(6); //Lee el valor del angulo que es de la forma +040.8
        roll = rollLeido.toInt(); //Convierte a entero solo el numero entero
        break;
      }

      case 'T':{ //<T RPM1
        String RPM = leeString(5);
        RPM1 = RPM.toInt();
        }
        break;


        case 't':{ //<t Presion de aceite
        String PressLeido = leeString(2);
        Presion = PressLeido.toInt(); //Convierte a entero el valor de presion de aceite
        break;
      }
      }


  } //fin del if para '<'

if (LecturaIdentificador == '?') { //Se lee un '?'
      LecturaIdentificador = leeCaracter(); //Lee siguiente caracter
      switch (LecturaIdentificador) {
        case 'L':{ //?L Amperimetro
          String AmperiosLeidos =  leeString(2);
          Amperios = AmperiosLeidos.toInt();
          break;
        }
        case 'E':{ //?E Vacio
          String vacioLeido =  leeString(4);
          vacio = vacioLeido.toInt();
          break;
        }
        case 'O':{ //?O temperatura aceite
          String temperaturaLeida =  leeString(3);
          temperatura = temperaturaLeida.toInt();
          break;
        }
      }
  } //FIN DE LECTURA '?'

if (LecturaIdentificador == '/') { //Leemos un '/'
      LecturaIdentificador = leeCaracter(); //Lee siguiente caracter
      switch (LecturaIdentificador) {
        case 'J':{ // /J LowFuelL AVISO
          if(leeCaracter() == '1'){
            lucesWarning &= 0b11111110;
          }
          else{
```

```
              lucesWarning |= 0b00000001;
            }
            break;
          }
          case 'K':{ // /K LoFuelR AVISO
            if(leeCaracter() == '1'){
              lucesWarning &= 0b11111101;
            }
            else{
              lucesWarning |= 0b00000010;
            }
            break;
          }
          case 'N':{ // /N Aviso de Nivel de vacio bajo
            if(leeCaracter() == '1'){
              lucesWarning &= 0b11101111;
            }
            else{
              lucesWarning |= 0b00010000;
            }
            break;
          }

           case 'P':{ // /P Aviso de tensión baja de Bateria
            if(leeCaracter() == '1'){
              lucesWarning &= 0b01111111;
            }
            else{
              lucesWarning |= 0b10000000;
            }
            break;
          }
        }
      } //FIN DE LECTURA '/'

  } //FIN LECTURAS SERIALES



//*************************************************************************************************************
******************
  //                                    ENVIAR BYTES DE LAS LUCES POR I2C A LOS ESCLAVOS

//*************************************************************************************************************
******************

    ModuloAP.envia(panelAutoP);
    Modulotren.envia(lucestren);
    ModuloWarning.envia(lucesWarning);



//*************************************************************************************************************
******************
  //                                        MODIFICACION DEL DIGITO ELEGIDO DEL XPDR

//*************************************************************************************************************
******************

  //Dependiendo del digito elegido, se lee el estado de subir o bajar frecuencia y escribe por puerto serie el
codigo asociado a ese digito
  ButtonCambiarEscalaCOM.seleccionaMux();

    if (digitalRead(MuxInSignal) == LOW ){ //Se lee el estado del pulsador
      digitoElegidoXPDR = (digitoElegidoXPDR + 1)%5 ; //El numero de digito a modificar se mueve entre 0 y 4
  }

  long nueva_pos_encoder_XPDR = encoderXPDR.read()/4;//+1 horario, -1 antihorario.
  int avanceXPDR = 0;

    if( nueva_pos_encoder_XPDR > antigua_pos_encoder_XPDR){
```

```
      avanceXPDR=1;
      antigua_pos_encoder_XPDR = nueva_pos_encoder_XPDR;
    }
    else if (nueva_pos_encoder_XPDR != antigua_pos_encoder_XPDR){
      antigua_pos_encoder_XPDR = nueva_pos_encoder_XPDR;
      avanceXPDR = -1;
    }

  //Se elige que digito se modifica segun el valor marcado por el encoder

  switch(digitoElegidoXPDR){
      case 1:
         if (avanceXPDR == 1){ //Digito primero y modo avanzar
            escribeSerial("A34");
         }

         else if( avanceXPDR == -1){ //Digito primero y modo reducir valor XPDR
            escribeSerial("A38");
         }
      break;

      case 2:
         if (avanceXPDR == 1){ //Digito primero y modo avanzar
            escribeSerial("A35");
         }

         else if( avanceXPDR == -1){ //Digito primero y modo reducir valor XPDR
            escribeSerial("A39");
         }
      break;

      case 3:
         if (avanceXPDR == 1){ //Digito primero y modo avanzar
            escribeSerial("A36");
         }

         else if( avanceXPDR == -1){ //Digito primero y modo reducir valor XPDR
            escribeSerial("A40");
         }
      break;

      case 4:
         if (avanceXPDR == 1){ //Digito primero y modo avanzar
            escribeSerial("A37");
         }

         else if( avanceXPDR == -1){ //Digito primero y modo reducir valor XPDR
            escribeSerial("A41");
         }
      break;
  }



//*************************************************************************************************************
********************
  //                                              MODULO DE RADIO

//*************************************************************************************************************
********************


  //-----------------------------------------------------ENVIO DE FREC.
SINTONIZADAS-----------------------------------------------------
    Radio.enviaString (secuenciaRadio);



//-----------------------------------------------------COM1-----------------------------------------------------
------------------
```

```
  ButtonCambiarEscalaCOM.seleccionaMux();

  if (digitalRead(MuxInSignal) == LOW ){ //Se lee el estado del pulsador
    escalafrecuenciaCOM = escalafrecuenciaCOM * (-1);
  }

  //OBTENER DIRRECION GIRO ENCODER
  int avancefrecuenciaCOM = 0;
  long nueva_pos_encoder_COM = encoderCOM.read()/4;//+1 horario, -1 antihorario.

  if( nueva_pos_encoder_COM > antigua_pos_encoder_COM){
    avancefrecuenciaCOM=1;
    antigua_pos_encoder_COM = nueva_pos_encoder_COM;
  }
  else if (nueva_pos_encoder_COM != antigua_pos_encoder_COM){
    antigua_pos_encoder_COM = nueva_pos_encoder_COM;
    avancefrecuenciaCOM = -1;
  }


  switch (escalafrecuenciaCOM){
    case 1: {//Escala de 1Mhz
      if (avancefrecuenciaCOM == 1) { //Orden de incrementar en 1 MHZ
        escribeSerial("A01");
        break;

      }
      if (avancefrecuenciaCOM == -1) { //Orden de decrementar en 1 MHZ
        escribeSerial("A02");
      }
      break;
    }
    case -1: {//Escala de 25 KHz
      if (avancefrecuenciaCOM == 1) { //Orden de incrementar en 25 KHZ
        escribeSerial("A03");

        break;
      }
      if (avancefrecuenciaCOM == -1) { //Orden de decrementar en 25 KHZ
        escribeSerial("A04");

        break;
      }
      break;
    }
  }

  ButtonSwapCOM.pulsador("A06"); //Leo el pulsador de cambiar la frecuencia de radio y escribo si se pulsa


//-----------------------------------------------------------------NAV1----------------------------------------

  //SELECCIONAR ESCALA DE LA FRECUENCIA NAV
  ButtonCambiarEscalaNAV.seleccionaMux();
  if (digitalRead(MuxInSignal) == LOW ){ //Se lee el estado del pulsador
    escalafrecuenciaNAV = escalafrecuenciaNAV * (-1);
  }


  //OBTENER DIRRECION GIRO ENCODER
  int avancefrecuenciaNAV = 0;
  long nueva_pos_encoder_NAV = encoderNAV.read()/4;//+1 horario, -1 antihorario.

  if( nueva_pos_encoder_NAV > antigua_pos_encoder_NAV){
    avancefrecuenciaNAV=1;
    antigua_pos_encoder_NAV = nueva_pos_encoder_NAV;
  }
   else if (nueva_pos_encoder_NAV != antigua_pos_encoder_NAV){
    antigua_pos_encoder_NAV = nueva_pos_encoder_NAV;
    avancefrecuenciaNAV = -1;
  }
```

```
  //ENVIAR EL MENSAJE DE CAMBIO EN ESCALA
    switch (escalafrecuenciaNAV){
      case 1: {//Escala de 1Mhz
        if (avancefrecuenciaNAV == 1) { //Orden de incrementar en 1 MHZ
          escribeSerial("A13");
          break;

        }
        if (avancefrecuenciaNAV == -1) { //Orden de decrementar en 1 MHZ
          escribeSerial("A14");
        }
        break;
      }
      case -1: {//Escala de 25 KHz
        if (avancefrecuenciaNAV == 1) { //Orden de incrementar en 25 KHZ
          escribeSerial("A15");

          break;
        }
        if (avancefrecuenciaNAV == -1) { //Orden de decrementar en 25 KHZ
          escribeSerial("A16");

          break;
        }
        break;
      }
    }

    ButtonSwapNAV.pulsador("A18"); //Escribe el codigo si se pulsa

   //-----------------------------------------------SELECCION DE ELEMENTOS A
TRANSMITIR---------------------------------------

    //ButtonDME.pulsador("A48");  //Se activa la comunicación DME (No implementado ahora)


   // ButtonADF.pulsador("A50");   //Se activa la comunicaicón ADF (No implemntado ahora)



//*************************************************************************************************************
*********************
  //                                                PILOTO AUTOMATICO

//*************************************************************************************************************
*********************

  //-----------------------------------------MODIFICAR ALt Y
VS-------------------------------------------------------------------

    ButtonSwapALt_VS.seleccionaMux();
    if (digitalRead(MuxInSignal) == LOW ){ //Se lee el estado del pulsador
      ModoAltOVs = ModoAltOVs * (-1); //PAra obtener dos estados opuesots correspondientes al modo de edicion
de altitud o de VS
    }

    //OBTENER DIRRECION GIRO ENCODER
    int sentido_giro_ALT_VS = 0;
    long nueva_pos_encoder_ALT = encoderAltVS.read()/4;//+1 horario, -1 antihorario.

    if( nueva_pos_encoder_ALT > antigua_pos_encoder_ALT){
      sentido_giro_ALT_VS=1;
      antigua_pos_encoder_ALT = nueva_pos_encoder_ALT;
    }
     else if (nueva_pos_encoder_ALT != antigua_pos_encoder_ALT){
      antigua_pos_encoder_ALT = nueva_pos_encoder_ALT;
      sentido_giro_ALT_VS = -1;
    }

  //ENVIAR EL MENSAJE DE CAMBIO EN ESCALA
```

```
    switch (ModoAltOVs){
      case 1: {//Selecciona ALt
        if (sentido_giro_ALT_VS == 1) { //Sube ALt
          escribeSerial("B11");
          break;

        }
        if (sentido_giro_ALT_VS == -1) { //Baja ALt
          escribeSerial("B12");
        }
        break;
      }
      case -1: {//Selecciona VS
        if (sentido_giro_ALT_VS == 1) { //Sube VS
          escribeSerial("B13");

          break;
        }
        if (sentido_giro_ALT_VS == -1) { //Baja Vs
          escribeSerial("B14");

          break;
        }
        break;
      }
    }



     //--------------------------------------------------AJUSTAR RUMBO
HDG-----------------------------------------------

    //OBTENER DIRRECION GIRO ENCODER
    long nueva_pos_encoder_HDG = encoderHDG.read()/4;//+1 horario, -1 antihorario.

    if( nueva_pos_encoder_HDG > antigua_pos_encoder_HDG){
      escribeSerial("A57");
      antigua_pos_encoder_HDG = nueva_pos_encoder_HDG;
    }
    else if (nueva_pos_encoder_HDG != antigua_pos_encoder_HDG){
      antigua_pos_encoder_HDG = nueva_pos_encoder_HDG;
      escribeSerial("A58");
    }



//*************************************************************************************************************
*********************
  //                                        ACTIVAR FUNCIONES DEL PILOTO AUTOMATICO

//*************************************************************************************************************
*********************


    InterruptorAP.conmuta("B01"); //Enciende y apaga el modulo AP
    InterruptorHDG.conmuta("B04"); //Enciende y apaga el control de rumbo o brujula magnetica
    InterruptorALt.conmuta("B05"); //Enciende y apaga el control de altitud
    InterruptorNAV.conmuta("B10"); //Activa la sintonización de frecuencia de ayuda a la navegación
    InterruptorAPR.conmuta("B09"); //Activa o desactiva el modo de aproximacion



//*************************************************************************************************************
*********************
  //                                                Lectura Switches LUCES, ALTERNADOR Y AVIONICA

//*************************************************************************************************************
*********************


    InterruptorfuelPump.biestado("f020","f021");
```

```
    InterruptorBeaconLight.biestado("C420","C421");
    InterruptorLandLight.biestado("C431","C430");
    InterruptortaxiLight.biestado("C440","C441");
    InterruptorNavLight.biestado("C411","C410");
    InterruptorStrobeLight.biestado("C450","C451");
    InterruptorPitot.biestado("C06","C05");
    InterruptorAlternador.biestado("E21","E20");
    InterruptorBateria.biestado("E18","E17");
    InterruptorAvionica.biestado("A431","A430");



//*************************************************************************************************************
*******************
  //                                                Magneto DE ARRANQUE DE MOTOR

//*************************************************************************************************************
*******************


    MagnetoOff.biestado("E01",""); ///Apagar motor
    MagnetoR.biestado("E02",""); //Encender bujias R
    MagnetoL.biestado("E03",""); //Bujias L
    MagnetoBoth.biestado("E04",""); //Todas las bujias
    MagnetoStart.biestado("E05",""); //Arrancar



//*************************************************************************************************************
*******************
  //                                            Lectura SwitchES TREN ATERRIZAJE Y FLAPS

//*************************************************************************************************************
*******************


    Interruptorflaps0.biestado("C13",""); //Poner a 0°
    Interruptorflaps10.biestado("C17020",""); //Poner a 10°
    Interruptorflaps20.biestado("C17040",""); //A 20°
    Interruptorflaps30.biestado("C16",""); //A 30°

    InterruptorGearUp.biestado("C01","C02"); //Tren arriba o abajo



//*************************************************************************************************************
**************************************************
  //                                                        SE PREGUNTA A ESCLAVO SI ESTA OCUPADO O NO

//*************************************************************************************************************
**************************************************
    Wire.requestFrom(4,1); //Se pide un bite al esclavo 1
    while (Wire.available()) {
       esclavo_finalizado = Wire.read(); // Guardo el valor leido
    }



//*************************************************************************************************************
**************************************************
  //                                                           MODULO GASES Y tRIM

//*************************************************************************************************************
**************************************************

  //LEER ENCODER Y PONER EN TRIM
```

```
//*************************************************************************************************************
********************
  //                                                GAUGES ANALOGICOS CON SERVO

//*************************************************************************************************************
********************


   gaugeVerticalSpeed.write(map(VerticalSpeed, VSMin,VSMax,AnguloVSMin, AnguloVSMax));

   gaugeHorizonteY.write(map(pitch, pitchMin, pitchMax, HorArtMinY, HorArtMaxY));
   gaugeHorizonteX.write(map(roll, rollMin, rollMax, HorArtMinX, HorArtMaxX));

   gaugeRPM.write(map(RPM1, RPMMin, RPMMax, AnguloRPMMin, AnguloRPMMax));

   gaugeturnCoordinator.write(map(roll,rollMin,rollMax,TCAnguloMin,TCAnguloMax));

   gaugeIAS.write(map(airSpeed, 0, 200, 0, 127));

   gaugefuel_L.write(map(fuel_L,FuelLMin,FuelLMax,AnguloFuelLMin,AnguloFuelLMax));
   gaugefuel_R.write(map(fuel_R,FuelRMin,FuelRMax,AnguloFuelRMin, AnguloFuelRMax));

   gaugetemperatura.write(map(temperatura,TempMin,TempMax,AnugloTempMin,AnguloTempMax));

   gaugePresion.write(map(Presion, PressMin, PressMax, AnguloPressMin, AnguloPressMax));

}//FIN DE MAIN
```

```
/*ESCLAVO BUS I2C QUE GESTIONA ALTIMETRO Y HDG. CUANDO COMIENZA A MOVER MOTORES SE PONE EN ESTADO OCUOPADO Y
SI EL MAESTRO PREGUNTA, SE LO IDNCIA ENVIANDO UN BYTE CON UN VALOR DE TRUE
 * CUANDO ACABA DE MOVER MOTORES, S EPONE A LIBRE Y SE PUEDE SABER PORQUE EL BYTE QUE ENVIA ES UN TRUE.
 * LOS DATOS SE RECIBEN CON UN TAMAÑO DE 2 BYTES (TIPO INT). SE DESPLAZAN LOS BITS Y SE GUARDAN EN LA VARIABLE
CORRESPONDIENTE SEGÚN EL ID RECIBIDO TBN COMO UN BYTE.
 * LOS MOTORES SE CONTROLAN CON SECUENCIA DE PASO COMPLETO, LO QUE DA MAS PAR PERO MENOS PRECISION
 * SE EMPLEA UN FOR PARA LLEGAR A LA POSICION DESEADA DEL INDICADOR.
 *
 * AUGUSTO SAMUEL HERNÁNDEZ MARTIN
 * 3 MAYO DE 2019
 * ULL
 */


#include <Wire.h>

#define PinHomeHDG 6
#define PinHomeIAS 7

//pines para Motor de HDG
#define IN1HDG A0
#define IN2HDG A1
#define IN3HDG A2
#define IN4HDG A3

//Pines para Motor de IAS
#define IN1IAS 12
#define IN2IAS 13
#define IN3IAS 9
#define IN4IAS 10

//Pines para Motor de Eje Needle
#define IN1NEEDLEHDG 2
#define IN2NEEDLEHDG 3
#define IN3NEEDLEHDG 4
#define IN4NEEDLEHDG 5

byte id; //Este byte almacena a que correponde el dato
int lecturaBus; //Se almacena el valor leido del bus
int HDG =0;
int HDG_anterior= 0;
int ALT, ALT_anterior;
int HDG_AP = 0;
int HDG_AP_anterior=0;

 int HDG_needle;
 int num_pasos_needle;
 int num_pasos_HDG;
 int num_pasos_ALT;


int valorPaso[4][4]={ //Secuencia de paso completo
       {1, 1, 0, 0},
       {0, 1, 1, 0},
       {0, 0, 1, 1},
       {1, 0, 0, 1} };

int i=0;
int j=0;

bool esclavo_ocupado = true;

void setup(){

//PONEMOS TODOS LOS PINES A OUTPUT
  pinMode (A0, OUTPUT);
  pinMode (A1, OUTPUT);
  pinMode (A2, OUTPUT);
  pinMode (A3, OUTPUT);
  pinMode(2,OUTPUT);
  pinMode(3,OUTPUT);
```

```
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);
  pinMode(12,OUTPUT);
  pinMode(13,OUTPUT);
  pinMode(9,OUTPUT);
  pinMode(10,OUTPUT);

//COLOCAMOS COMO ESCLAVO DEL BUS CON DIR 0x04

 Wire.begin(4); // Icicializamos el bus i2c como esclavo numero 4
 Wire.onReceive(recibeDato); //Se indica que funcion debe ejecutarse al recibir un dato por I2C
 //Serial.begin(9600); // Inicializamos la salida del monitor serie
 Wire.onRequest(enviaPeticion); //Cuando el maestro pide ver como esta el esclavo, se llama a esta funcion

}


void loop(){

  esclavo_ocupado=true; //Dejo el esclavo ocupado para realizar todo

  switch (id){
    case 2: //Se recibe el HDG
      HDG_anterior = HDG; //Guardamos el valor antiguo
      HDG = lecturaBus; //Toamos el nuevo valor
    break;

    case 1: //Se recibe el Altimetro (altitud)
     ALT_anterior = ALT; //Guardamos el antiguo
     ALT = lecturaBus; //Cargamos el nuevo valor de ALT
    break;

    case 3://Se recibe la direccion de rumbo del AP (la consigna)
     HDG_AP_anterior = HDG_AP;
     HDG_AP = lecturaBus;
    break;

  } //Fin del Switch



//-------------------------
//Se mueve cada motor
//-------------------------


//Se mueve HDG y needle a la par

num_pasos_HDG = (HDG - HDG_anterior)*120/360;
      if( num_pasos_HDG > 0){
       for(i=0; i< num_pasos_HDG ; i++){
           for(j=0; j < 4 ; j++){
               digitalWrite(IN1HDG,valorPaso[j][0]);
               digitalWrite(IN2HDG,valorPaso[j][1]);
               digitalWrite(IN3HDG,valorPaso[j][2]);
               digitalWrite(IN4HDG,valorPaso[j][3]);
               digitalWrite(IN1NEEDLEHDG,valorPaso[j][3]);
               digitalWrite(IN2NEEDLEHDG,valorPaso[j][2]);
               digitalWrite(IN3NEEDLEHDG,valorPaso[j][1]);
               digitalWrite(IN4NEEDLEHDG,valorPaso[j][0]);
           }
       }
      }
     else if (num_pasos_HDG < 0){
      for(i=0; i< abs(num_pasos_HDG) ; i++){
           for(j=0; j < 4 ; j++){
               digitalWrite(IN4HDG,valorPaso[j][3]);
               digitalWrite(IN3HDG,valorPaso[j][2]);
               digitalWrite(IN2HDG,valorPaso[j][1]);
               digitalWrite(IN1HDG,valorPaso[j][0]);
               digitalWrite(IN4NEEDLEHDG,valorPaso[j][0]);
```

```
                digitalWrite(IN2NEEDLEHDG,valorPaso[j][1]);
                digitalWrite(IN2NEEDLEHDG,valorPaso[j][2]);
                digitalWrite(IN1NEEDLEHDG,valorPaso[j][3]);
            }
       }

     }


  //Vemos ahora el valor de rumbo del PA
  HDG_needle = (HDG_AP - HDG_AP_anterior) - (HDG - HDG_anterior); //Ahora apunta a otro sitio al haberse
movido con el indicador de rumbo
  num_pasos_needle = HDG_needle * 120/360;

//Movemos needle de HDG
        if( num_pasos_needle > 0){
       for(i=0; i< num_pasos_needle ; i++){
           for(j=0; j < 4 ; j++){
               digitalWrite(IN4NEEDLEHDG,valorPaso[j][0]);
               digitalWrite(IN3NEEDLEHDG,valorPaso[j][1]);
               digitalWrite(IN2NEEDLEHDG,valorPaso[j][2]);
               digitalWrite(IN1NEEDLEHDG,valorPaso[j][3]);
           }
       }
      }
     else if (num_pasos_needle < 0){
      for(i=0; i< abs(num_pasos_needle) ; i++){
           for(j=0; j < 4 ; j++){
               digitalWrite(IN1NEEDLEHDG,valorPaso[j][3]);
               digitalWrite(IN2NEEDLEHDG,valorPaso[j][2]);
               digitalWrite(IN3NEEDLEHDG,valorPaso[j][1]);
               digitalWrite(IN4NEEDLEHDG,valorPaso[j][0]);

           }
       }

     }



//Movemos agujas del altimetro

num_pasos_ALT = (ALT - ALT_anterior)*120/1000; //Se calculan los pasos a dar para la altitud. Una vuelta del
motor son 1000 pies
        if( num_pasos_ALT > 0){
       for(i=0; i< num_pasos_ALT ; i++){
           for(j=0; j < 4 ; j++){
               digitalWrite(IN1IAS,valorPaso[j][0]);
               digitalWrite(IN2IAS,valorPaso[j][1]);
               digitalWrite(IN3IAS,valorPaso[j][2]);
               digitalWrite(IN4IAS,valorPaso[j][3]);
           }
       }
      }
     else if (num_pasos_ALT < 0){
      for(i=0; i< abs(num_pasos_ALT) ; i++){
           for(j=0; j < 4 ; j++){
               digitalWrite(IN1IAS,valorPaso[j][3]);
               digitalWrite(IN2IAS,valorPaso[j][2]);
               digitalWrite(IN3IAS,valorPaso[j][1]);
               digitalWrite(IN4IAS,valorPaso[j][0]);

           }
       }

     }

//SE HA TERMINAOD DE MOVER LOS 3 MOTORES STEPPER

//librero a esclavo. Ahora esta libre para escuchar
  esclavo_ocupado = false;
```

```
}


// funcion que se ejecuta al recibir petición de datos desde un master
void recibeDato(int cuantos){
  byte MSB, LSB;
  while(1 < Wire.available()){
    id = Wire.read(); // recibir 1 byte de id
    MSB = Wire.read(); //Leo el MSB
    LSB = Wire.read(); //Leo el LSB
  }
  lecturaBus = (MSB << 8) | LSB; //Desplazo ocho posiciones el byte primero, y le hago la O bit-a-bit al
segundo byte
}

void enviaPeticion(){
  Wire.write(esclavo_ocupado); //Envia el estado del esclavo cuando se le pide
}
```