UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DISEÑO Y DESARROLLO DE UN PROTOTIPO FÍSICO DE DRON CUADRICÓPTERO

AUTOR: CLAUDIA MARTÍNEZ ABRIL

TUTOR: FRANCISCO EUGENIO ALBERT GIL

COTUTORA: MARÍA NURIA ALEIXOS BORRÁS

Curso Académico: 2018-19

AGRADECIMIENTOS

“A mis padres por apoyarme incondicionalmente”

“A mis tutores y al técnico de laboratorio por estar pendientes de mi trabajo y por la ayuda ofrecida”

“A Ángela y Sandra por animarme y apoyarme durante estos cuatro años de grado”

“A Ángela, Elena, Lorena, Marta, Nayara y Paula por compartir conmigo todos los buenos y no tan buenos momentos de estos cuatro años”

Diseño y desarrollo de un prototipo físico de dron cuadricóptero

RESUMEN

Este proyecto consiste en el diseño y desarrollo de un dron cuadricóptero controlado mediante un sistema de transmisión-recepción.

Para llevarlo a cabo se han seguido las siguientes tareas:

- Planteamiento del diseño: diseño en 3D y electrónica necesaria.
- Modelado 3D de las piezas.
- Simulación del comportamiento mecánico del conjunto.
- Impresión 3D de las piezas y montaje del prototipo.
- Programación de la electrónica escogida.

Palabras clave

Diseño, CAD 3D, Impresión 3D, Programación de microcontroladores

RESUM

Aquest projecte consisteix en el disseny i desenvolupament de un dron cuadricòpter controlat mitjançant un sistema de transmissió - recepció.

Per dur-lo a terme s'han seguit les següents tasques:

- Plantejament del disseny: disseny en 3D i electrònica necessària.
- Modelat 3D de les peces.
- Simulació del comportament mecànic del conjunt.
- Impressió 3D de les peces y muntatge del prototip.
- Programació de l'electrònica escollida.

Paraules clau

Disseny, CAD 3D, Impressió 3D, Programació de microcontroladors

ABSTRACT

This project consists in the design and development of a quadrocopter drone controlled by a transmission – reception system.

In order to carry it out the following tasks have been followed:

- Design approach: 3D design and necessary electronics.
- 3D modeling of the pieces.
- Simulation of the mechanical behavior of the set.
- 3D printing of the parts and prototype assembly.
- Programming of the chosen electronics.

Key words

Design, CAD 3D, 3D printing, Microcontroller programming

Diseño y desarrollo de un prototipo físico de dron cuadricóptero

DOCUMENTOS CONTENIDOS EN EL TFG

- Memoria
- Presupuesto
- Planos
- Anexo de programación

ÍNDICE DE LA MEMORIA

ÍNDICE DEL PRESUPUESTO

ÍNDICE DE FIGURAS DE LA MEMORIA

ÍNDICE DE TABLAS DE LA MEMORIA

ÍNDICE DE TABLAS DEL PRESUPUESTO

MEMORIA

Diseño y desarrollo de un prototipo físico de dron cuadricóptero

Capítulo 1- Introducción al proyecto

1.1- Objetivo del proyecto

El objetivo de este proyecto es el diseño y desarrollo de un dron cuadricóptero controlado mediante un microcontrolador ATmega328. Los puntos de este objetivo son:

· Diseñar una estructura simple imprimible en 3D.

· Estudiar las fuerzas máximas que es capaz de soportar. De esta forma, se estudiarán los puntos críticos de la estructura y se podrá aplicar reformas para mejorar el diseño.

· Construir un prototipo virtual mediante el modelado 3D y el ensamblaje de las piezas.

· Construir el prototipo físico mediante impresión 3D de las piezas, montaje de éstas y los circuitos necesarios.

· Controlar el movimiento del dron mediante un sistema de transmisión y recepción con módulos de radio frecuencia dirigidos por un Arduino.

1.2- Motivación del proyecto

La principal motivación para la realización de este TFG es crear un proyecto desde el principio empezando por su diseño, hasta su montaje y control final. Se aplicarán conocimientos de asignaturas estudiadas durante el grado como Ingeniería Gráfica, Circuitos, Tecnología Eléctrica, Ingeniería Informática e Impresión 3D.

Capítulo 2- Componentes y métodos

2.1- Aspectos teóricos de los drones

2.1.1- Breve historia de los drones

Se denomina "vehículo aéreo no tripulado" (VANT) o comúnmente "dron" a una aeronave que, volando sin tripulación, es capaz de mantener un vuelo controlado y sostenido de manera autónoma y propulsada por un motor eléctrico. Su diseño presenta una amplia variedad de formas, tamaños, configuraciones y características.

A pesar de que en la actualidad existen VANT tanto de uso civil como comercial, sus primeros usos fueron en el ámbito militar y se denominaban "Vehículos Aéreos de Combate No Tripulados". Los primeros modelos fueron desarrollados entre los años 1914 y 1918, durante la I Guerra Mundial.

Posteriormente, las aplicaciones de los drones aumentaron y con ellas el número de consumidores en ámbitos más allá del terreno militar. Actualmente se pueden encontrar infinidad de aplicaciones, de entre las que destacan las siguientes: labores en la lucha contra incendios, investigaciones arqueológicas, búsqueda de personas o diversión y ocio entre otras.

La normativa de estos dispositivos es muy estricta, y en España, el 4 de julio de 2014 se aprobó una nueva ley temporal de regulación de los usos de drones con peso menor a 150kg, entre los que se encuentran grabación, vigilancia y monitorización, revisión de infraestructuras y obtención de mapas.

2.1.2- Clasificación de los drones

Podemos clasificar los drones, a priori, en tres grandes grupos: según su forma, control y uso.

Clasificándolos según su forma y diseño, existen drones de ala fija y multirrotores. Los primeros presentan una forma y desplazamiento similares a las aeronaves convencionales y por tanto son más aerodinámicos que los segundos. Los multirrotores en cambio, pueden contener tres, cuatro, seis y hasta ocho rotores con los cuales se sostienen en el aire y obtienen la propulsión necesaria para desplazarse. Este segundo tipo es el dron más común por simpleza y estabilidad proporcionada por la ubicación centrada de los motores respecto del centro de gravedad del conjunto.

Como obviamente ambos tipos son muy diferentes, las aplicaciones que se le dan a éstos son también distintas. Los drones de ala fija se destinan a labores que necesitan cubrir áreas extensas, como trabajos de cartografía o teledetección, labores de vigilancia etc., ya que presentan gran autonomía y aerodinámica, así como eficiencia y velocidad de vuelo, además de un impacto sonoro mínimo. Por otra parte, la posibilidad de los multirrotores de volar en una posición estática, hacen que éstos resulten más aptos para trabajos de inspección en diferentes sectores.

Si los clasificamos esta vez, según su método de control, se pueden diferenciar los drones autónomos, los controlados remotamente y los supervisados o monitorizados. Los drones autónomos presentan la capacidad de seguir una ruta programada previamente, por lo que son aptos para labores de transporte de correspondencia o mercancía. Por otra parte, los drones controlados remotamente, son los más antiguos y extendidos mundialmente a día de hoy. La persona que controla el dispositivo, contiene un control remoto de éste. Por último, los drones monitorizados o supervisados, se encuentran a mitad camino entre los dos tipos anteriores ya que presentan la capacidad de trabajar de manera autónoma pero debe existir un técnico de supervisión encargado de controlar y supervisar su feedback post vuelo o bien programar previamente su ruta.

Finalmente, el último tipo de clasificación, según la utilidad que se le quiera dar al dron, podemos encontrar drones militares y civiles. Los segundos, pueden ser destinados tanto como para un uso comercial, ocio y diversión como para tareas del gobierno como bien pueden ser rescates o trabajos de reconocimiento, por ejemplo.

El dron que se va a contruir es un multirrotor cuadricóptero, controlado remotamente para diversión.

2.1.3- Conceptos generales de los drones

La posición del dron viene definida en todo momento por los ángulos de navegación “Pitch”, “Roll” y “Yaw” de modo que si se inclina sobre alguno de estos tres ejes, se desplazará en la dirección deseada. Si el dron es inclinado sobre el eje Pitch, el dron podrá avanzar o retroceder según el sentido del giro, inclinarse sobre el eje Roll hará que se desplace hacia la derecha e izquierda y rotar sobre el eje Yaw se traduce en girar sobre su propio eje vertical. A estos movimientos, también se añade la capacidad de ganar o perder altura.

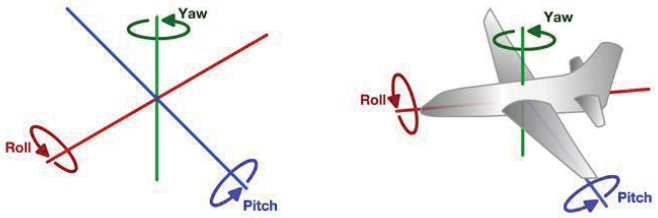


Figura 1. Ejes principales del dron.

[Fuente: https://elchals.wordpress.com/2015/11/30/dron-pepito-mi-primer-arduino-quadcopter/]

Existen dos tipos de configuración de la estructura: la configuración ‘x’ y la configuración ‘+’. La diferencia básica entre ambas es que a la hora de realizar un movimiento determinado, los motores implicados serán diferentes en cada una. En la primera de ellas, para avanzar (inclinarse sobre el eje Pitch) intervendrán los cuatro motores ya que será necesario desacelerar los motores delanteros y acelerar los traseros. En cambio, para la configuración tipo ‘+’, solamente se dispone de un motor delantero y uno trasero (quedando dos laterales) los cuales se deberán desacelerar y acelerar respectivamente (con los laterales a velocidad constante). Los motores que se quedan en la posición lateral, serán necesarios cuando se requiera desplazar hacia la derecha e izquierda (inclinarse en el eje Roll). El tipo de configuración es independiente de la rotación en el eje Yaw, de manera que, si el giro es horario, se aceleran los motores que giran en sentido antihorario y viceversa.

Figura 2. Dron en configuración ‘x’.

[Fuente: https://airamericadrone.com/wp-content/uploads/2017/01/DRON-X23-White-2.jpg]

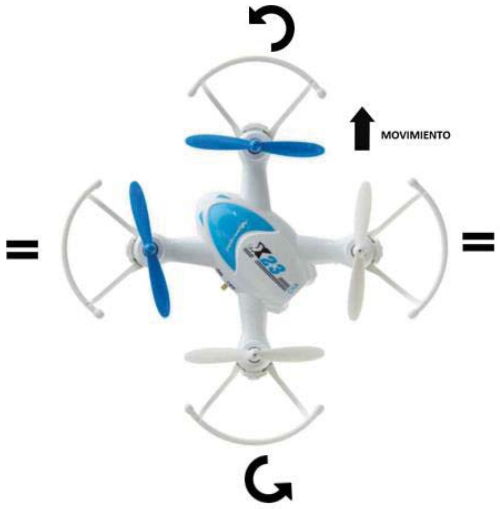


Figura 3. Dron en configuración ‘+’.

[Fuente: https://airamericadrone.com/wp-content/uploads/2017/01/DRON-X23-White-2.jpg]

La configuración que se ha elegido para el diseño es la ‘x’, ya que presenta mayor estabilidad debido a que, para realizar determinados movimientos (desplazarse en los ejes roll y pitch) se emplean todos los motores en lugar de dos (configuración ‘+’) y de esta manera, se reparte el trabajo a realizar.

2.2- Modelado 3D

2.2.1- Autodesk Inventor

Autodesk Inventor es un programa de diseño mecánico en 3D, con modelado paramétrico que ofrece la capacidad de diseñar piezas (archivos .IPT), realizar sus planos (archivos .IDW), ensamblajes (archivos .IAM) y la perspectiva del montaje (archivos .IPN). Inventor es utilizado en ingeniería de diseño para la producción y perfección de piezas y productos.

En Inventor, un proyecto está compuesto por "piezas", las cuales son creadas mediante bocetos y características que las definen. Las distintas piezas del proyecto se conectan para producir "ensamblajes" mediante la aplicación de restricciones entre las mismas.

Autodesk Inventor ofrece simulación de elementos finitos y análisis dinámicos, herramientas necesarias para la realización de piezas de diferentes materiales, etc.

2.2.2- Teoría de elementos finitos

El método de los elementos finitos es una técnica matemática de cálculo numérico que sirve para predecir cómo reaccionará un producto ante las fuerzas, la vibración, el calor, el flujo de fluidos y otros efectos físicos del mundo real en estructuras de ingeniería, con la finalidad de mejorar la calidad de los productos y proyectos.

Autodesk Inventor, como muchas otras aplicaciones de análisis, está basado en el método de elementos finitos y brinda un completo conjunto de herramientas para su estudio. Es importante considerar que este método es siempre una aproximación a la realidad y que presentará errores causados por el proceso de discretización. Estos errores estarán presentes en los bordes curvos o en los componentes geométricamente complejos, pero se debe tener en cuenta que es posible reducirlos aumentando el número de elementos o el orden del polinomio de interpolación.

La geometría de la pieza es sometida a cargas y restricciones y posteriormente se subdivide en más partes denominadas "elementos", que el conjunto de todos ellos representan el dominio continuo del problema. El método sustituye un número infinito de variables desconocidas, por un número limitado de elementos (elementos finitos) de comportamiento definido. Los elementos pueden presentar diversas formas como triangular o cuadrangular dependiendo del tipo y tamaño del problema.

Denominamos "nodos" o "puntos nodales" a las conexiones entre elementos y "malla" al conjunto de elementos y nodos. La cantidad de nodos y elementos, así como su tamaño y forma, definen la precisión de los resultados del análisis, siendo más preciso cuanto mayor sea el número de elementos en una malla y menor sea su tamaño.

2.2.3- Impresión 3D

La impresión 3D, también denominada "fabricación por adición", es un proceso de colocación de material en capas según un modelo digital para así crear objetos físicos. Este proceso permite crear desde piezas sencillas hasta piezas de elevado nivel técnico.

Los inicios de la impresión 3D provienen de la invención de la impresora de inyección de tinta, en 1976. Más tarde, en 1984, se produjo el cambio de impresión con tinta a impresión con materiales. Son muchas las industrias que han desarrollado a lo largo de las últimas décadas la impresión 3D, creándose una gran variedad de aplicaciones de esta tecnología.

Algunos de los métodos más empleados en la impresión 3D son:

- Modelado por deposición fundida (FDM): también es denominado “fabricación de filamento fusionado”. Es el método más sencillo de entre los ya existentes y consta de tres elementos básicos: una placa de impresión donde se imprime la pieza (también denominada “cama”), el filamento utilizado como material de impresión y una cabeza que deposite este material denominada “extrusor”. El funcionamiento es muy sencillo, el filamento es succionado por el extrusor y éste deposita el material capa por capa sobre la cama de impresión siguiendo un conjunto de instrucciones digitales. El material de impresión es termoplástico y hay infinidad de variedades (el más común en este tipo de impresión es el ABS).

- Estereolitografía (SLA): este método emplea luz UV para curar o endurecer resina, capa por capa hasta obtener la pieza deseada.

- Sinterización selectiva del láser (ELS): este método emplea un láser para fusionar materiales en polvo, capa por capa hasta obtener la pieza deseada.

2.2.4- FDM y tipos de impresoras 3D FDM

Se ha empleado el método FDM para la impresión de nuestras piezas, al ser el método de impresión más común en impresoras 3D de escritorio.

Cuando se diferencia entre tipos de impresoras 3D, normalmente se hace referencia a las diferentes tecnologías 3D que existen, pero si se habla de los tipos de impresoras de impresión FDM, existen distintos tipos de máquinas que ofrecen distintos resultados:

- Impresoras 3D cartesianas: son el tipo de impresora FDM más común y extendido del mercado actualmente. Utilizan un sistema de coordenadas cartesianas (eje X, Y y Z) de donde obtienen el nombre, para determinar qué movimientos realizar en las tres dimensiones y la localización del cabezal (extrusor). Controlar la posición del cabezal en todo momento es posible gracias a unos motores paso a paso. La cama de impresión puede moverse en uno de los ejes, siendo obviamente el extrusor quien realiza los movimientos restantes.

Figura 4. Impresora Cartesiana.
[Fuente: http://atlas3dstudio.com/2016/04/14/como-funciona-la-impresion-3d/]

- Impresoras 3D polares: la principal diferencia respecto a las anteriores son sus coordenadas polares ya que describen puntos en una retícula circular en lugar de cuadrada (determinados por los ejes X, Y y Z), determinados por ángulo y longitud. Con lo cual, la cama de impresión realiza movimientos giratorios y el extrusor se mueve en el eje Z y de izquierda a derecha. Éstas impresoras presentan una ventaja a destacar con respecto a las anteriores, consigue un mayor volumen de producción en un menor espacio.

Figura 5. Impresora polar.
[Fuente: http://tecnoimpre3d.com/alta/]

- Impresoras 3D Delta: este modelo de impresora contiene 6 ejes y también trabajan con coordenadas cartesianas. Como particularidad presentan una cama de impresión circular fija que, combinada con el extrusor, fijado por encima de ella con una configuración triangular, obtenemos piezas con una velocidad de impresión superior. También es cierto que presentan un inconveniente y es que no son tan precisas como las cartesianas.

Figura 6. Impresora delta.

[Fuente: http://chupatintas.es/impresoras-3d/1099-impresora-3d-delta-multicolor.html]

- Brazos robóticos: este tipo de impresión se encuentra todavía en desarrollo. Hoy en día son utilizados en líneas de montaje de automóviles y no en piezas con filamentos de extrusión. Pero, como se ha comentado al principio, se encuentra en desarrollo ya que, al no requerir de cama de impresión fija, tiene mayor movilidad al ser el extrusor extremadamente flexible y, por tanto, abre una gran cantidad de nuevas posibilidades de diseños más complejos.

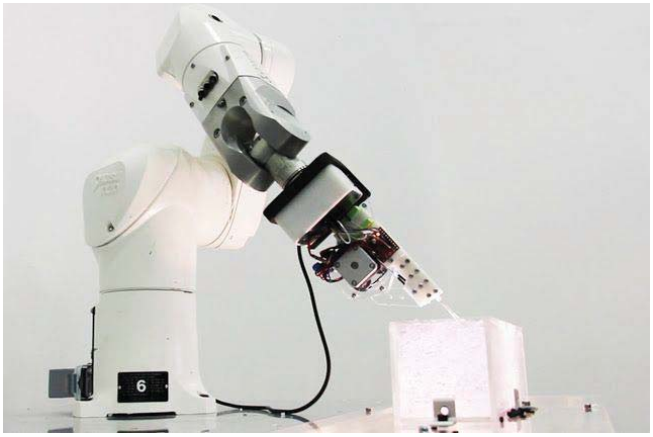


Figura 7. Brazo robótico.
[Fuente: https://www.pinterest.es/pin/292804413258976054/]

2.2.5- Slic3r

A la hora de imprimir una pieza en 3D, una vez se haya diseñado a priori en un programa de diseño 3D (como pueden ser Inventor, SolidWorks, FreeCAD, OpenSCAD, SketchUp, AutoCAD, etc.), cada software con su propio formato, se debe exportar este documento a un formato STL que, queda determinada la geometría de la pieza en 3D sin considerar colores o texturas y demás información irrelevante para la tarea de impresión.

Slic3r es el software que se ha empleado para la realización de la etapa intermedia entre el diseño y la impresión, denominada “slicing” o “fileteado”. Este programa realiza un barrido de la pieza capa por capa, es decir, convierte un archivo STL (modelo 3D) en instrucciones de impresión para la impresora (archivos GCODE). El proceso que realiza es cortar las piezas en rebanadas horizontales (capas) y generar códigos que contienen toda la información necesaria para el proceso de impresión (temperatura, posiciones, velocidades, etc.).

En resumen, este programa permite por tanto cargar uno o más modelos STL y modificarlos al cambiar de posición, rotarlos, duplicarlos, escalarlos, etc. Además permite cambiar parámetros de impresión (velocidad, temperatura, relleno de la pieza, material de soporte, etc.), de filamento (estos dependen del tipo de filamento elegido), o de la impresora (los cuales dependen del modelo de impresora y pueden ser el número de extrusores, tamaño de la superficie de impresión, etc.) y previsualizar la impresión por capas de la pieza.

2.2.6- Impresora JGaurora

La impresora que se ha utilizado para producir las piezas del proyecto es la impresora JGaurora A5 cuyo volumen de impresión es de 305 x 305 x 325 mm.

Este modelo de impresora contiene dos características difíciles de encontrar en otros modelos. La primera de ellas es que es capaz de detectar el descentramiento del filamento y por tanto

emitir un sonido de error y detener la impresión. La segunda de ellas es la capacidad de reanudar el trabajo de impresión exactamente por donde se quedó, después de un corte de luz.

Figura 8. Impresora JGaurora.

[Fuente: http://www.jgaurora3d.com/cpzxCD/info_12.aspx?itemid=444]

2.3- Equipo electrónico

2.3.1- Placa Arduino

Arduino es una plataforma de desarrollo basada en una placa electrónica de hardware libre que incorpora un microcontrolador reprogramable y una serie de pines que permiten establecer conexiones entre el microcontrolador y diferentes sensores y actuadores fácilmente.

Arduino es una PCB ("Printed Circuit Board") con un determinado diseño de circuitería interna, con lo cual el usuario final no debe preocuparse por las conexiones eléctricas que necesita éste para funcionar y directamente puede dedicarse a desarrollar las diferentes aplicaciones electrónicas para las que necesite el dispositivo.

Actualmente, existen diferentes modelos de placas Arduino oficiales, que realizan prácticamente el mismo trabajo, pero cada una está pensada para un propósito diferente y características variadas. Claro está, que al pertenecer a la misma familia, comparten la mayoría de sus características de software como es la arquitectura, librerías y documentación.

Arduino puede instalarse y ejecutar en todos los sistemas operativos (Windows, Mac OS y Linux) por lo tanto su entorno de programación es multiplataforma y su lenguaje de programación está basado en C++.

Las placas Arduino más destacadas son las siguientes:

- ARDUINO UNO:

  Esta placa está basada en el microcontrolador ATmega328, cuenta con 14 entradas/salidas digitales, 6 de las cuales se pueden utilizar como salidas PWM (modulación por ancho de pulsos) y otras 6 son entradas analógicas. Contiene además, conector USB, conector de alimentación, cabecera ICSP (*In-Circuit Serial Programming*) y botón de reseteado.

- ARDUINO MEGA:

  Esta placa cuenta con el microcontrolador Atmega2560 pero con más memoria para programa, más RAM y pines que el resto de las placas y además, es la más potente de entre las placas con microcontrolador de 8 bits.

- ARDUINO NANO:

  Es la placa más pequeña de entre las existentes y también está basada en el Atmega328. Requiere de un cable USB Mini-B para alimentarse y contiene las mismas propiedades eléctricas que la placa Arduino UNO a diferencia de contener 2 pines analógicos más.

- ARDUINO PRO:

  Esta placa está basada, al igual que las anteriores, en el Atmega328. Al igual que Arduino UNO posee 14 pines de entradas/salidas de las cuales 6 se pueden utilizar como salidas PWM. Esta placa, se ha diseñado para ser instalada de manera semipermanente en los dispositivos.

2.3.2- Placa Arduino Nano

La placa escogida para el proyecto es Arduino Nano ya que es la más idónea para poder ser introducida en espacios reducidos.

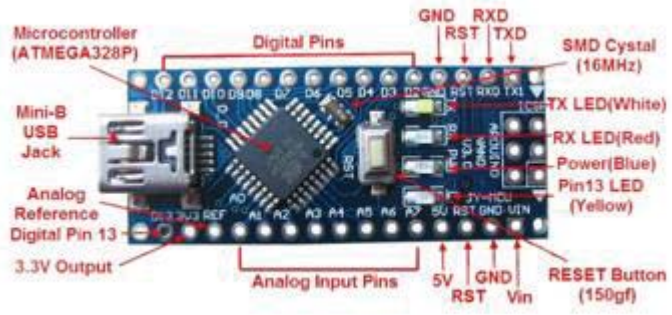


Figura 9. Pines del Arduino NANO.
[Fuente: http://cursoarduino.proserquisa.com/2016/10/19/tutorial-33-arduino-nano/]

Como se ha comentado anteriormente, Arduino Nano posee las mismas características que Arduino Uno. Ambos contienen el microcontrolador ATMEGA328, pero Arduino Nano en una versión más pequeña. El ATMEGA328 posee Memoria Flash 32KB, SRAM 2KB, EEPROM 1 KB, voltaje de operación 5V y Reloj 16 MHZ. Además la placa contiene 6 pines macho para comunicarse con otras tarjetas externas que contengan una comunicación ICSP.

Arduino Nano contiene 30 pines, entre ellos, se destacan los siguientes:

PINES ARDUINO	CARACTERÍSTICAS
Vin	Es utilizado para alimentar al Arduino
RX (D0)	Opera a 5V y pueden proporcionar o recibir un máximo de 40 mA. Es utilizado para transmitir y recibir datos de comunicación serial TTL.
TX (D1)	Opera a 5V y pueden proporcionar o recibir un máximo de 40 mA. Es utilizado para transmitir y recibir datos de comunicación serial TTL.
D2 – D9	Opera a 5V y pueden proporcionar o recibir un máximo de 40 mA.
D10 – D12	Opera a 5V y pueden proporcionar o recibir un máximo de 40 mA. Permite la comunicación SPI la cual es necesaria para el módulo NRF24L01 del cual se hablará más adelante.
D13	Opera a 5V y pueden proporcionar o recibir un máximo de 40 mA. Permite la comunicación SPI la cual es necesaria para el módulo NRF24L01 del cual se hablará más adelante. Este pin contiene una función especial y es encender un led que se encuentra conectado a él mismo.
A0 – A3	Proporciona entrada analógica con resolución de 10 bits. Este pin también puede ser empleado como entrada o salida digital.
A4 – A5	Proporciona entrada analógica con resolución de 10 bits. Este pin permite la comunicación i2C, necesaria para la recepción de datos de los sensores que se emplearán en el proyecto. Este pin también puede ser empleado como entrada o salida digital.
A6 – A7	Proporciona entrada analógica con resolución de 10 bits.
RST	Su función es reiniciar la placa y para ello se debe conectar GND a este pin.
AREF	Este pin es utilizado como voltaje de referencia para las entradas analógicas. El rango de medición de los pines de entrada analógica por defecto es de 0 a 5V y con este pin, se podría disminuir dicho rango.

Tabla 1. Pines relevantes de Arduino NANO.

2.3.3- Motores

2.3.3.1- Tipos de motores

Existen dos tipos de motores: motores con escobillas (brushed) y motores sin escobillas (brushless). La elección de un tipo u otro depende principalmente del tipo de aplicación que se vaya a desarrollar con ellos, su tamaño y requisitos de potencia.

Los motores brushed, como india su nombre, utilizan escobillas conectadas a un colector para realizar el cambio de polaridad en el rotor y conmutar mecánicamente la corriente de las bobinas del motor. Por otra parte, los motores brushless no incorporan colector ni escobillas y es un controlador de motor el que realiza la conmutación de las bobinas eléctricamente.

Presentan mayor vida útil los motores brushless ya que al no contener escobillas, no hay rozamientos que generen desgaste ni ruido y no es necesario un mantenimiento continuo. También presentan mayor eficiencia, ya que en los motores brushed se genera una gran pérdida de calor y potencia, mayor rendimiento, al ofrecer mejor relación entre potencia proporcionada y tamaño, siendo idóneos para aplicaciones con espacio reducido y mejor relación entre velocidad y par motor. Por todas estas razones los motores brushless son los empleados en la construcción de drones.

El motor brushless está compuesto por una parte móvil (el rotor), donde se encuentran los imanes permanentes, y una parte fija (estator) sobre la cual van dispuestos los bobinados de hilo conductor. La corriente eléctrica es conducida por los bobinados del estator generando un campo electromagnético que interacciona con el campo magnético creado por los imanes permanentes del rotor, haciendo que aparezca una fuerza que hace girar al rotor y por lo tanto, al eje del motor. El elemento que controlará el giro del rotor sea cual sea su posición será el variador electrónico.

2.3.3.2- Nomenclatura

El tamaño de los motores sin escobillas se indica con un número de 4 dígitos, AABB. El diámetro del estator es indicado por los dos primeros, AA y su altura por los dos segundos, BB, ambas en milímetros. Obtendremos más rpm cuanto mayor sea la altura y mayor par motor cuanto mayor sea el diámetro.

2.3.3.3- Parámetros que determinan a un motor

Como se ha explicado anteriormente, los motores brushless o sin escobillas son los seleccionados para la construcción de los multirrotores.

Se debe de tener en cuenta una serie de parámetros, para elegir el motor más adecuado de acuerdo al proyecto que se quiera llevar a cabo.

- Peso total estimado del conjunto del dron. Es necesario conocerlo para poder calcular aproximadamente cuánto empuje se necesitará entre los cuatro motores para elevarlo. El empuje generado por los motores debe ser como mínimo, el doble del peso del dron, para así asegurarse de que con el acelerador a mitad camino, presente suficiente empuje para elevar todo el peso. Si esto no se cumple, el dron tendrá dificultades para despegar y no responderá bien a las señales de control indicadas. De esta manera, presentando un peso de 2 Kg en conjunto, el empuje necesario mínimo para su correcto funcionamiento es de 4 Kg, es decir, 1 Kg por cada motor. Si el proyecto estuviese

enfocado en la construcción de un dron de carreras, el ratio de empuje/peso deberá ser elevado para tener un comportamiento ágil y dinámico, aunque hay que tener en cuenta que si es demasiado elevado puede llegar a ser muy difícil de controlar. Si por el contrario, estuviese enfocado a la construcción de un dron de fotografía aérea el ratio empuje/peso deberá ser menor. Esto no sólo proporciona mayor control si no que deja un margen libre para posibles futuras cargas como cámaras o baterías más pesadas para obtener mayor tiempo de vuelo.

- Kv, constante de velocidad. Este parámetro indica las miles de rpm del motor por cada voltio, es decir el teórico incremento de rpm del motor cuando el voltaje sube un voltio sin ninguna carga. Una vez las hélices son conectadas, las rpm caen drásticamente a causa de la resistencia del aire. Una constante de velocidad Kv elevada proporcionará mayor velocidad de giro del motor pero también requerirá un tamaño de hélices más pequeño. Por el contrario, un valor de Kv bajo, generará un par motor (fuerza que hace girar la hélice) mayor por lo tanto se podrá utilizar un tamaño de hélices mayor. Si por el contrario se aplicasen hélices de elevado tamaño con motores con Kv elevados, el motor giraría más rápido como si tuviese acopladas hélices de menor tamaño pero como requeriría mayor par motor, generaría más calor extrayendo más corriente pudiendo quemar el motor.

- Peso y tamaño de los motores. Cuanto mayor sea el tamaño del estator y del dron, mayor debe ser el tamaño de las hélices, mayor par motor y menor Kv requerirá. Si se aplicara un motor con un par motor bajo y unas hélices pesadas, disminuiría considerablemente la eficiencia de éste.

- Eficiencia (g/w) del motor. La eficiencia se calcula como el empuje del motor en gramos dividido entre la potencia en vatios con el 100% de aceleración. Cuanto más empuje se genera, más corriente es necesaria, de manera que los motores idóneos son los que presentan elevado empuje y bajo consumo de corriente. Con motores ineficientes, se malgasta energía, tiempo de vuelo y las baterías sufren un desplome de voltaje.

- Tiempo de respuesta. El tiempo de respuesta está afectado por el peso y el ángulo de las palas de las hélices escogidas. Depende al igual del par motor. Los motores con más par motor presentan un tiempo de respuesta más rápido.

- Temperatura. La temperatura es un factor importante a tener en cuenta en los motores brushless ya que se debilita el campo magnético de los imanes que presentan y desmagnetizan a medida que aumenta la temperatura, por lo tanto reduce su vida útil. Es por esto por lo que aplicar hélices de elevado tamaño con KVs elevados, provocará un aumento de la temperatura y consecuentemente una reducción de la vida útil de los motores. Es importante por esto, que los motores presenten estructuras que ayuden al enfriamiento de los mismos, con el fin de alargar su vida útil.

Otras características que si están presentes en el diseño de los motores, afectarán al rendimiento son:

- Ejes huecos: ahorra peso, permite usar materiales más resistentes y por lo tanto, mejora su vida útil.
- Tipos de imanes: cuanto más fuerte sea el campo magnético, más potencia efectiva podrá generarse y consecuentemente mayor par motor y tiempo de respuesta menor.

- Gap o hueco entre rotor y estator: las fuerzas magnéticas se degradan con el cuadrado de la distancia y al reducir este hueco, aumenta la potencia del motor, mejorando el par motor y el tiempo de respuesta. Como inconveniente al reducir este hueco es que aumenta la corriente consumida y disminuye la eficiencia.
- Forma de los imanes: si se utilizan imanes permanentes en forma de arco, éstos estarán más cerca del estator y el gap será más pequeño.
- Vueltas de hilo de cobre en las bobinas del estator y grosor del hilo de cobre: la máxima corriente que el motor puede llegar a consumir el determinada por el número de vueltas de hilo de cobre en cada bobina del estator mientras que la cantidad de corriente que el motor es capaz de manejar antes de recalentarse está determinado por el grosor del hilo. Los fabricantes juegan con el grosor del hilo de cobre ya que al aumentarlo la potencia aumenta sin disminuir par ni eficiencia. El bobinado también exige más espacio físico y requiere mayor tamaño de estator. Es por esto que cada vez el tamaño de los motores son mayores pero más potentes.
- Motores horarios y anti horarios: se encuentran los motores etiquetados como horarios (CW o clockwise) y los anti horarios (CCW o counterclockwise). Esto es importante a tener en cuenta en los motores con escobillas ya que si se colocan erróneamente se degastarán a una mayor velocidad. Sin embargo los motores brushless CW y CCW son idénticos y la única diferencia que presentan es que el eje está roscado en un sentido u otro.

2.3.3.4- Cálculos para la elección del motor

Para elegir correctamente los motores del dron, debemos calcular primero, el peso total estimado que los motores deberán ser capaces de levantar.

COMPONENTES	PESO UNITARIO	PESO TOTAL
Arduino NANO (placa controladora de vuelo y receptor)	2x4g	8g
NRF240L1	3g	3g
ESCs	4x27g	108g
Motores	4x33g	132g
Cuerpo del dron (piezas impresas 3D)	258,73g	258,73g
Tornillos	12g	12g
Hélices	4x4g	16g
IMU	3g	3g
Regulador de voltaje 3.3V	Despreciable	Despreciable
Pcb (placa controladora de vuelo)	4,5g	4,5g
Batería LiPo	172g	172g
Conector LiPo	2g	2g
Interruptor	4g	4g
Cableado y conectores	10g (aprox)	10g
Pines macho, condensadores, interruptores	6g	6g
	TOTAL ESTIMADO	**739,23g**

Tabla 2. Peso estimado de los componentes a elevar por el dron.

Una vez se ha estimado el peso total del dron, se puede calcular aproximadamente cuánto empuje necesitan generar la combinación de motores y hélices para poder volar. Como se ha comentado anteriormente, el empuje generado debería ser como mínimo, el doble del peso del dron, de manera que se asegure tener suficiente. Se elige un empuje/peso de 2.5, ya que no se trata de un dron de carreras (se elegiría un ratio empuje/peso mucho mayor) sino uno orientado hacia la fotografía aérea. Una vez elegido el empuje/peso del dron, se procede a calcular el empuje que tendrá que proporcionar el dron y cada uno de los motores para elevar el peso, de la siguiente manera:

$$Empuje = \frac{empuje}{peso} \cdot \ peso\ estimado = 2.5\ x\ 739.23 = 1848.075\ g \quad [1]$$

$$Empuje\ por\ motor = \frac{empuje}{nº\ motores} = \frac{1848{,}075}{4} = 462.0187\ g \quad [2]$$

El siguiente paso a realizar es elegir el KV del motor. Se elegirá un KV de 2300 típico en motores de fotografía, ya que no es un valor ni muy elevado (5000 por ejemplo, utilizado para drones de carreras), ni excesivamente bajo.

Una vez elegido, se debe proceder a calcular los vatios por gramo necesarios (W/g), es decir, la inversa de la eficiencia:

$$\frac{W}{g} = 0.17\ x\ \frac{Kv}{1000} + 0.09 = 0.17\ x\ \frac{2300}{1000} + 0.09 = 0.481 \quad [3]$$

Por último, se calcula la potencia necesaria por cada motor, esto es, los vatios por gramo necesarios por el empuje de un motor:

$$0.481\ \frac{W}{g}\ x\ 462.0187 g = 222.231\ W \quad [4]$$

Se debe seleccionar un motor que proporcione una potencia mayor o como mínimo 222,231W. Finalmente, se elige un motor RS2205 2300KV el cual proporciona 280 W.

Figura 10. Motores escogidos.
[Fuente: https://www.ebay.es/itm/4Pcs-Set-RS2205-2300KV-CW-CCW-Brushless-Motor-Para-FPV-Racing-RC-Quadcopter/173501795686?_trkparms=aid%3D111001%26algo%3DREC.SEED%26ao%3D1%26asc%3D20160908105057%26meid%3D29184c9a9e774401bababd15e163173d%26pid%3D100675%26rk%3D1%26rkt%3D15%26sd%3D173501795686%26itm%3D173501795686&_trksid=p2481888.c100675.m4236&_trkparms=pageci%3Ad09b7116-9042-11e9-83eb-74dbd1802204%7Cparentrq%3A60af3bcc16b0ad4a8609d2efffda3c99%7Ciid%3A1]

2.3.4- Hélices

Las hélices son un dispositivo mecánico que contienen palas o álabes con forma curva, sobresaliendo del plano en el que gira y de esta forma se obtiene una diferencia de velocidades entre el fluido de una cara y otra. Esta diferencia de velocidades conlleva una diferencia de presiones, según el principio de Bernoulli, la cual crea una fuerza (fuerza propulsora de una aeronave) perpendicular al plano de rotación de las palas hacia la zona de menos presión.

Puede encontrarse multitud de hélices en cuanto a tamaños y materiales. Son más comunes las compuestas de nylon o ABS pero también se encuentran de fibra de carbono por ejemplo, siendo más caras que las anteriores.

2.3.4.1- Nomenclatura de las hélices

Dos ejemplos de nomenclatura de las hélices son los siguientes: “hélice de 6x4” o “60|40”. La longitud de la hélice en pulgadas viene determinada por el primer dígito (6’’) y el paso de la hélice por el segundo. El paso de la hélice es la distancia que recorre en una vuelta completa, por lo que cuanto más paso contenga una hélice, más velocidad adquirirá el dron y más corriente consumirá.

2.3.4.2- Elección de las hélices

Tal y como se ha comentado en apartados anteriores, cuanto mayor es el KV del motor, más pequeña debe ser la hélice que se acopla a él. El fabricante del motor indica las hélices recomendadas para cada motor.

MODELO	KV (rpm/V)	V	HÉLICES	A	EMPUJE (g)	W	EFICIENCIA (g/W)	CELDAS LIPO	PESO (g)
DX2205	2300	11.1	5045	19.2	660	213	3.1	2 – 4S	28
		14.8		27.6	950	408	2.3		
	2600	11.1	4045	18.5	530	205	2.6		
		14.8		23.2	710	343	2.1		

Tabla 3. Relación motor – hélices según el fabricante.

Por lo tanto, se eligen unas hélices 5045 de acuerdo con los motores elegidos previamente (RS2205 2300kV).

Figura 11. Hélices escogidas.

[Fuente: https://www.ebay.es/itm/Black-DALprop-V2-Regular-2-Pairs-J5045-CW-CCW-Propeller/152736183234?hash=item238fc933c2:g:eGgAAOSwuk1Z24C~]

2.3.5- Controladores electrónicos de velocidad (ESC: Electronic Speed Control)

Un controlador electrónico de velocidad (Electronic Speed Controller) es un sistema capaz de definir la velocidad de giro de un motor mediante la generación de pulsos compatibles con este tipo de motores.

Los ESCs controlan motores eléctricos mediante Modulación por Ancho de Pulso (PWM). El controlador de vuelo envía una señal PWM a los ESCs con variaciones de 1000 a 2000 microsegundos (con 1000 microsegundos, el motor se para, con 1500 microsegundos el motor funciona a la mitad de potencia y con 2000 microsegundos, el motor funciona a máxima velocidad).

El sistema BEC (variadores lineales) está incorporado en la mayoría de los ESCs, cuya función es reducir el voltaje de la batería a la que están conectados los ESCs a 5 V, para que ésta pueda ser utilizada en el receptor y controlador de vuelo, ya que la batería es demasiado potente como para conectarla directamente a estos circuitos. Por lo tanto, con estos variadores lineales, desaparece la necesidad de utilizar una segunda batería para estos circuitos y por tanto, se reducen costes y peso. Si el ESC contiene BEC, contendrá un conector con tres cables (negro, rojo y blanco) utilizados como tierra, salida de 5V y entrada de la señal de pulso y además se aprecia un mayor volumen en el cuerpo de éste. En caso contrario, solamente contendrá el cable de tierra y el de señal.

2.3.5.1- Elección de los ESC

Se debe elegir unos ESC que la corriente que proporcionan se encuentre por encima de la demanda del conjunto motor-hélice de nuestro dron. Si se seleccionan unos ESC que ofrezcan una corriente inferior a la demandada, serán dañados. Cuanta más corriente sea capaz de soportar, mayor será su precio.

El fabricante del motor elegido previamente, recomienda también los ESC a utilizar y son los de 30 A, ya que estos motores consumen 19 A y unos ESC de 20 A podrían quedarse justos de entrega.

Figura 12. ESCs escogidos.

[Fuente: https://www.amazon.es/gp/product/B00X5SLS4Q/ref=ppx_yo_dt_b_asin_image_o02_s00?ie=UTF8&psc=1]

2.3.6- Batería

La batería es un elemento muy importante ya que se encarga de alimentar a todos los componentes eléctricos. Se pueden diferenciar los siguientes tipos:

- Ni-Cd (níquel-cadmio): son las más antiguas, no toleran bien las cargas rápidas y tienen efecto memoria (efecto que reduce la capacidad de las baterías con cargas incompletas).
- Ni-MH (níquel-metal-hidruro): siguen presentando efecto memoria y tienen incluso menor vida útil que las anteriores en cuanto a número de cargas.
- Ion-Litio: contienen casi el doble de capacidad que las anteriores, son mucho más ligeras, no poseen efecto memoria y se descargan más lentamente. Su principal inconveniente es que si se dañan o perforan son inflamables (reaccionan sus componentes con el oxígeno del aire).
- Li-Po (polímero de litio): son las más modernas y utilizadas, tienen una carga más lenta que las antiguas de Ni-Cd o Ni-MH, pero se pueden fabricar con formas más diversas que las de Ion-Litio por lo que se optimiza el espacio del fuselaje dedicado a las baterías. Tampoco tienen efecto memoria ni son inflamables.

Las baterías LiPo tienen la particularidad de disponer de un mayor voltaje por celda que otros tipos, pudiendo llegar a los 4.2 V por celda cuando están completamente cargadas. El número "S" que contiene la batería hace referencia al número de celdas, es decir, al número de pequeñas baterías que en conjunto componen la batería completa. Una batería 3S (3 celdas) está compuesta de 3 subbaterías conectadas en serie. Las más comunes son las de 3 y 4 celdas, las primeras son más baratas y ligeras y las segundas proporcionan mayor velocidad y potencia.

Los miliamperios por hora (mAh) que contenga una batería indica la capacidad de carga. Mayor capacidad no conlleva precisamente mayor tiempo de vuelo ya que una batería de mayor capacidad supone un mayor tamaño y por tanto mayor peso. Debe encontrarse el equilibrio entre ambos parámetros para adecuarlo al tipo de dron.

Por último, el factor de descarga C es un término importante indicador de la velocidad de descarga de la batería de manera continua y segura, esto es, la corriente que suministra de esta

manera en una hora. Si la batería es de, por ejemplo, 25C y 2200 mAh, es capaz de suministrar 55 amperios en una hora.

Finalmente, se elige para el proyecto una batería LiPo de 2200 mAh 25C y 3S.

Figura 13. Batería escogida.

[Fuente: https://www.amazon.es/gp/product/B00T2KB46I/ref=ppx_yo_dt_b_asin_title_o02_s00?ie=UTF8&psc=1]

2.3.7- Módulo NRF24L01

Las conexiones mediante WIFI o mediante BlueTooth, presentan limitaciones considerables en la distancia de uso, alcanzan poco más de 20 metros en condiciones óptimas.

Los dispositivos de radio que contienen sistema en chip de 2.4 GHz de bajo costo, basados en el chip de Nordic semiconductor NRF24, establecen comunicaciones RF (Radio Frecuencia) entre dos o más puntos a diferentes velocidades entre 250 Kbps, 1 Mbps y 2 Mbps y permite la conexión simultánea con hasta 6 dispositivos.

Se resaltan las siguientes ventajas de los módulos: son baratos, la velocidad de transmisión y recepción es configurable, presentan muy bajo consumo en Stand By, y por último que aporta una alcance mínimo de 20 metros hasta un máximo de 80, pero la versión de alta potencia que incorpora una antena externa, ofrece un alcance máximo de 700 a 1000 metros.

Es importante recalcar y tener en mente a la hora de la conexión de componentes electrónicos que, estos módulos, funcionan a 3.3 V y no están capacitados para soportar una tensión mayor.

Para poder emitir y recibir con estos módulos se necesitan dos placas Arduino que recibirán el papel de transmisor y receptor, donde los programas serán distintos pero tendrán características comunes. El módulo de alta potencia irá acoplado al Arduino transmisor y al receptor se le acoplará el módulo básico.

Aunque los módulos consumen muy poco en uso o en Stand By, pueden absorber en el arranque más de lo que la fuente a la que se encuentra conectado puede proporcionar, lo que impide que arranque correctamente. Es por ello que se recomienda el uso de condensadores de 100 pF entre GND y 3.3 V del NRF24L01.

Figura 14. Módulo NRF24L01 con amplificador de potencia (PA) + antena.

[Fuente: https://www.amazon.es/gp/product/B07P95X6HM/ref=ppx_yo_dt_b_asin_title_o00_s00?ie=UTF8&psc=1]

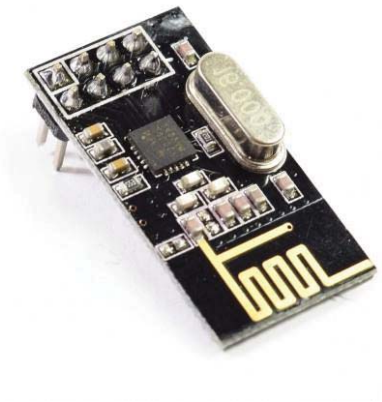


Figura 15. Módulo NRF24L01.

[Fuente: https://www.amazon.es/piezas-versi%C3%B3n-nRF24L01-transceptor-RBTMKR/dp/B01AQ2HCRU/ref=sr_1_1?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&keywords=nrf24&qid=1560695581&s=gateway&sr=8-1]

2.3.8- Regulador de voltaje

La finalidad de un regulador de voltaje es mantener el voltaje en un circuito, alrededor de un valor deseado. Los reguladores de voltaje son componentes electrónicos muy utilizados ya que las fuentes de alimentación pueden producir corrientes que dañen alguno de los restantes componentes en el circuito y con la presencia de éstos, este problema desaparece. El regulador de voltaje recibe una tensión de voltaje variable en la entrada y mantiene en la salida una tensión constante.

El AMS1117 es un regulador de tensión que permite convertir una tensión de entrada de entre 4.8 y 12 VDC a una tensión de salida de 3.3 VDC.

La placa Arudino ya contiene un regulador de voltaje de 3.3V, pero éste no puede proporcionar la suficiente corriente para que el módulo de radio NRF24L01 funcione de manera correcta. Este problema se soluciona incorporando un regulador al circuito que le proporcione la alimentación adecuada. Esto deberemos hacerlo tanto en el módulo del transmisor como en el del receptor.

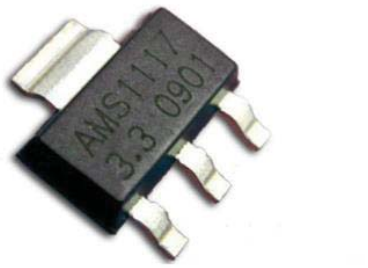


Figura 16. Regulador de voltaje 3.3 V AMS 1117.

[Fuente: https://www.amazon.es/Voltaje-Regulador-AMS1117-Despu%C3%A9s-prototipos/dp/B01N1Z95KL/ref=sr_1_fkmr0_2?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&keywords=regulador+ams1117&qid=1560695647&s=gateway&sr=8-2-fkmr0]

2.3.9- Unidad de medición inercial (IMU)

Una IMU es un dispositivo electrónico utilizado normalmente para aviones y drones, entre otros usos, que proporciona datos sobre la velocidad, orientación y fuerzas gravitacionales de un elemento a partir de acelerómetros y giróscopos. Mediante los acelerómetros se detecta la aceleración actual y mediante los giróscopos, los cambios rotacionales, en los ejes roll (alabeo), pitch (cabeceo) y yaw (guiñada).

El MPU6050 es el sensor más utilizado para la elaboración de drones caseros y contiene 6 grados de libertad. Éste combina un acelerómetro de 3 ejes con el que medir su aceleración y un giróscopo de 3 ejes con el que medir su velocidad angular de tal forma que uno compensa las limitaciones del otro.
Esto último es así, ya que los acelerómetros no presentan deriva (drift) ni a medio ni a largo plazo, pero sí son influenciables por sus movimientos propios y el ruido y es por ello que no son fiables a corto plazo. Los giroscopios, sin embargo, se adaptan a movimientos cortos o bruscos y el inconveniente que presentan es que, se acumulan errores y ruido al obtener el ángulo por integración respecto al tiempo, por lo tanto a medio y largo plazo acaban presentando deriva. Que uno compense las limitaciones del otro quiere decir que se debe utilizar el giroscopio para medidas rápidas e ir corrigiendo la deriva de estos mediante el acelerómetro.

Con el acelerómetro se calcularán los ángulos de inclinación de los ejes roll y pitch. Para que sea detectada una aceleración, debe haber una fuerza que la produzca, si el acelerómetro se encontrase quieto, la gravedad sería la única aceleración que detectará. Si suponemos un sistema donde el sensor se encuentra inclinado un cierto ángulo en su eje roll, entonces la fuerza de la gravedad se repartirá en los dos ejes restantes del sensor, por lo tanto, el ángulo formado entre las dos componentes de la fuerza de la gravedad en estos ejes será igual al ángulo de inclinación respecto del eje roll.

$$Ángulo\ de\ inclinación\ eje\ X = \tan^{-1}\left(\frac{aceleración\ eje\ Y}{aceleración\ eje\ Z}\right) \quad [5]$$

$$Ángulo\ de\ inclinación\ eje\ Y = \tan^{-1}\left(\frac{aceleración\ eje\ X}{aceleración\ eje\ Z}\right) \quad [6]$$

Como el sensor se mueve en tres ejes, se debe tener en cuenta el tercero e incluir la componente de la gravedad en él, siendo este tercer eje aquel respecto del que se quiere medir la inclinación. Mediante las fórmulas de Euler se tiene:

$$AngleY = atan\left(\frac{X}{\sqrt{Y^2 + Z^2}}\right)$$

$$AngleX = atan\left(\frac{Y}{\sqrt{X^2 + Z^2}}\right)$$

Figura 17. Fórmulas de Euler ángulo de inclinación.

Para poder aprovecharse de las ventajas de ambos a corto y largo plazo se deberá combinar ambos. Para realizar esto, existen varios filtros posibles siendo el más utilizado el filtro de Kalman. Este filtro es desarrollado en 1960 por Rudolf E. Kalman, considerado como uno de los grandes descubrimientos del siglo XX por sus implicaciones en el proceso de filtrado en sensores. Este filtro en su versión general implementa cálculos complejos que suponen excesivos para Arduino (en resumen, realiza una estimación del valor futuro de la medición y lo compara con el valor real mediante un análisis estadístico para compensar el error en futuras mediciones). Y es por esto por lo que se emplea un filtro complementario, que puede considerarse una simplificación del primero ya que prescinde del análisis estadístico:

$$\theta = A \times (\theta prev + \theta gyro) + B \times \theta accel \quad [7]$$

Donde A y B son constantes que inicialmente pueden tomarse 0.98 y 0.02 respectivamente.
Éste filtro se comporta como un filtro paso alto para la medición del giroscopio y como filtro paso bajo para la del acelerómetro. A y B deben sumar siempre la unidad.

Como se ha explicado, con la combinación de acelerómetro y giroscopio solamente se obtendría los ángulos de inclinación de los ejes roll y pitch, y no del eje Z o yaw. Para poder obtenerlo, se debería emplear un magnetómetro y dado que el sensor MPU6050 no dispone de este componente, se utilizará uno que lo contenga para poder controlar igualmente este eje.

El sensor MPU9250 contiene 9 grados de libertad ya que combina acelerómetro, giróscopo y también el magnetómetro necesario para calcular el ángulo de inclinación del eje restante.

El magnetómetro permite medir la intensidad del campo magnético terrestre en cada uno de sus ejes y con estas medidas, siempre que se disponga de los ángulos de inclinación de los ejes roll y pitch, se podrá calcular la orientación del sistema en el eje yaw. Para ello utilizaremos los datos proporcionados tanto por el magnetómetro como por el giróscopo y las fusionaremos utilizando un filtro complementario de la misma forma que para los ejes roll y pitch calculados mediante el acelerómetro y el giróscopo.

Debemos proyectar en el plano horizontal, las componentes magnéticas medidas (mx, my, mz) y una vez obtenidas, calcular el ángulo de inclinación yaw a partir de relaciones trigonométricas de la siguiente manera:

$xh = mx \cos\theta + my \sin\theta \sin\phi + mz \cos\phi \sin\theta$ [8]
$yh = -my \cos\phi + mz \sin\theta$ [9]

Donde ϕ es el ángulo de inclinación del eje roll y θ del pitch. Con ello:

$$Ángulo\ de\ inclinación\ eje\ Z\ medido\ por\ el\ magnetómetro = \tan^{-1}\left(\frac{yh}{xh}\right) \quad [10]$$

Y de nuevo aplicamos el filtro que unirá esta vez los datos del magnetómetro con los del giroscopio.

$\theta = \mathrm{A}\ \mathrm{x}\ (\theta\mathrm{gyro}) + B\ \mathrm{x}\ \theta\mathrm{mag}$ [11]

La conexión que utiliza el módulo es por i2C (bus serie de datos utilizado principalmente para la comunicación entre, por ejemplo, un controlador y circuitos periféricos integrados), lo que le permite trabajar con la mayoría de microcontroladores. El módulo, al contener un regulador de voltaje para alimentarse con tensiones de entre 2.4 y 3.6V, se puede alimentar con los 5V del Arduino directamente.

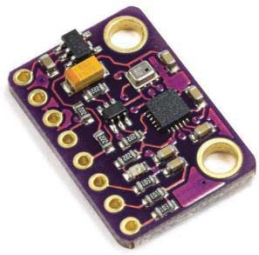


Figura 18. Módulo MPU9250.

[Fuente: https://www.ebay.es/itm/MPU9250-BMP280-10DOF-Acceleration-Gyroscope-Compass-Nine-Shaft-Sensor-GY-91/183421722372?hash=item2ab4c95f04:g:iCkAAOSwM~Rbkt0u]

2.4- Equipo de programación

2.4.1- Arduino IDE

Arduino IDE (Integrated Development Environment) es el entorno de programación para los Arduinos NANO que van a utilizarse. Una vez creado el código a aplicar, se subirá al Arduino mediante cable USB y la placa podrá comenzar a trabajar de manera autónoma. La descarga del programa está disponible gratuitamente en su propia página web.

Una vez adquirido el programa, se comienza a escribir el código en el cuadro del editor de texto. En el área de mensajes se observa la información necesaria sobre si la consola está compilando, cargando, fallos y errores que puedan producirse tanto en el código como en el propio programa.

Desde el menú es posible acceder a funciones como carga de librerías, archivos, configuración, herramientas, etc. y en los botones de acceso rápido encontramos:

1.  para verificar si el programa está correctamente escrito y puede compilar.
2.  para subir el programa a la placa una vez compilado.
3.  para crear un programa nuevo.
4.  para abrir un programa.
5.  para guardar el programa.
6.  para abrir una ventana de comunicación con la placa en la que podemos seguir las respuestas del Arduino al código, siempre que la conexión USB esté conectada.

Cuando el código está listo y compilado, configuramos el IDE para que pueda establecerse la comunicación con la placa, para ello necesitamos la conexión USB y seleccionar en *menú*, *Herramientas* y después *Tarjeta* el modelo de tarjeta que estemos tratando.

Los programas en Arduino IDE son llamados "sketch" (boceto) y presentan dos funciones especiales que forman parte de cada sketch: *setup()* y *loop().* La primera únicamente es ejecutada una vez (al inicio del programa) por lo tanto es un buen lugar para las tareas de configuración de modos de pin o inicializar bibliotecas. La segunda, será ejecutada una y otra vez (en bucle) y por ello es la parte principal de los sketches. Es necesario incluir ambas funciones en el sketch aunque no sean utilizadas.

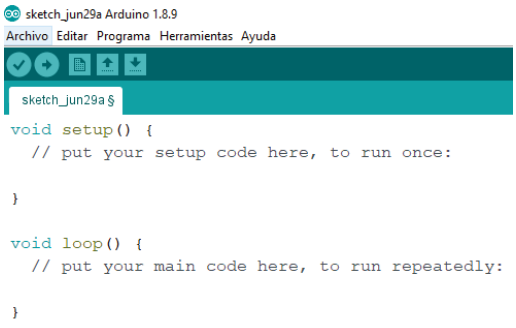


Figura 19. Sketch de Arduino.

Capítulo 3- Diseño del dron

3.1- Desarrollo electrónico

Los tres elementos que conforman el proyecto son transmisor, receptor y controlador de vuelo.

3.1.1- Transmisor

El elemento más importarte del transmisor es un Arduino Nano. En primer lugar, hay que suministrar voltaje de la batería por la entrada 'Vin' y también a un regulador de voltaje 3.3V. Es necesario el regulador de voltaje porque el módulo NRF24 consume demasiada corriente (115 mA) y Arduino no puede suministrarle esa cantidad (aunque lleve ya incluido un regulador de 3.3V, proporciona 50mA, lo cual no es suficiente). Se debe conectar la salida negativa de la batería al pin GND del Arduino y la salida positiva al interruptor de encendido/apagado del mando. El otro pin del interruptor irá directamente al pin Vin del Arduino y a la entrada del regulador de voltaje 3.3 V, de esta manera, cada vez que se encienda el interruptor estará alimentado el Arduino y el regulador de voltaje tendrá 3.3 voltios en su salida.

A continuación, hay que conectar el módulo de radio NRF24 de potencia amplificada. Este módulo amplificado es necesario porque el transmisor siempre necesita más potencia para obtener más alcance. No se puede conectar directamente 5 voltios al módulo de radio, ya que esto lo quemaría, por lo tanto, se debe proporcionar los 3.3V de la salida del regulador de voltaje. Las conexiones entre Arduino y el módulo de radio son las siguientes:

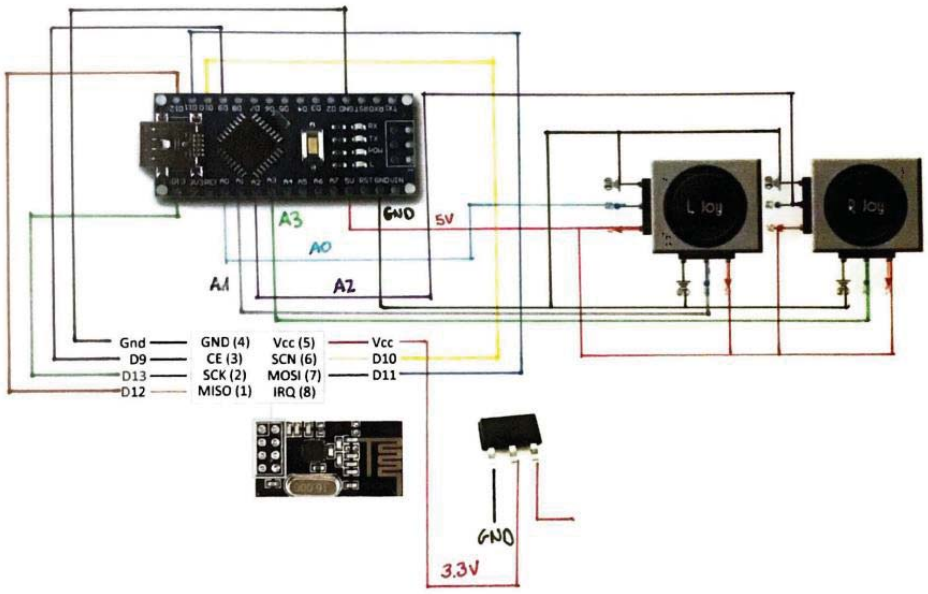


Figura 20. Conexiones transmisor.

Se utilizarán los pines analógicos A0, A1, A2 y A3 para conectar los joysticks (cada joystick contiene dos potenciómetros) del mando de control. Se suministra 5 voltios y tierra a ambos, de esta manera se tiene 0 voltios en las entradas analógicas cuando el potenciómetro se encuentre en la posición más baja y 5 voltios cuando esté en el valor más alto. El transmisor pasa los valores indicados con los dos joysticks desde un rango de 0 a 1023 (ya que Arduino presenta un ADC de 10 bits) a un rango de 0 a 255 (ya que únicamente puede enviar 8 bits).

La gran mayoría de los reguladores de voltaje necesitan un condensador en la salida para estabilizar el circuito, ya que son muy sensibles a las variaciones que se puedan producir y esto genera caídas de voltaje y corrientes imprevistas. Por todo esto, se debe conectar junto al regulador, un condensador de 10uF denominado “condensador de desacoplamiento”.

3.1.2- Receptor

Se emplea de nuevo un Arduino Nano y el mismo módulo de radio NRF24 que utiliza el transmisor para recibir la señal enviada por éste. Estos módulos funcionan tanto de transmisores como receptores. En este caso, se utiliza sin amplificador de potencia o antena. El montaje entre Arduino y módulo de radio es exactamente igual que en el transmisor (también con el regulador de 3.3V y su condensador de desacoplamiento de 10uF).

3.1.3- Controlador de vuelo

El controlador de vuelo contiene un Arduino NANO, el cual es el cerebro del dron. Los sistemas de control de vuelo de drones son muchos y variados. Se pueden encontrar desde sistemas de piloto automático habilitados con GPS hasta sistemas de estabilización básicos que utilizan un hardware de control de radio de nivel de aficionado.

Los sistemas de control de vuelo actuales presentan muchos sensores disponibles: GPS, sensores de presión barométrica, sensores de velocidad del aire, etc. Los principales contribuyentes a los cálculos de vuelo son los giroscopios junto con los acelerómetros y magnetómetros. Se utiliza el módulo MPU9250 el cual contiene los tres sensores y se comunica con el Arduino mediante comunicación i2C.

La comunicación i2C (circuito inter-integrado) es un bus serie de datos diseñado como un bus maestro-esclavo. La transferencia de datos es siempre iniciada por un maestro y el esclavo reacciona.

Los sensores devuelven valores al microcontrolador con los cuales se calculan los ángulos de inclinación de cada uno de los tres ejes en cada momento. Conociendo estos ángulos, se le puede dar una potencia diferente a cada uno de los motores para estabilizar y contrarrestar las fuerzas no deseadas.

Con la configuración elegida del dron en X, se tienen dos motores en la parte frontal y dos en la parte posterior. A cada ESC (controlador electrónico de velocidad) de cada motor le corresponde un pin de salida del Arduino (D4, D5, D6 y D7) según la siguiente imagen:

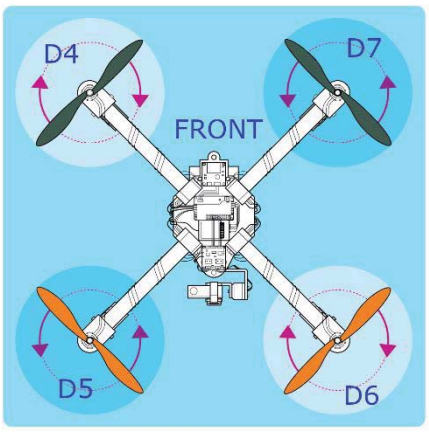


Figura 21. Correspondencia de los motores con los pines del Arduino.

Para ello, los cables de señal (blancos) de los 4 ESC van conectados a éstos cuatro pines, y los cables negros a tierra.

Por otra parte, se debe suministrar desde Arduino, 5V y tierra a la IMU (MPU9250). También conectar los pines SDA y SCL del módulo a los pines analógicos A4 y A5 del Arduino respectivamente para realizar la comunicación i2C.

Como en un único Arduino se tienen pines suficientes, se implementará el receptor y el controlador de vuelo en el mismo, ahorrándose la conexión mediante cableado entre dos Arduinos y la parte del código de transmisión de datos entre uno y otro, y así, utilizar directamente los datos recibidos desde el transmisor para realizar los cálculos necesarios.

Se empleará la salida de 5V de uno de los BEC para alimentar el Arduino y la de otro para alimentar el módulo de radio, pero esta última deberá pasar a priori por el regulador de voltaje para proporcionarle el adecuado (3,3 V).

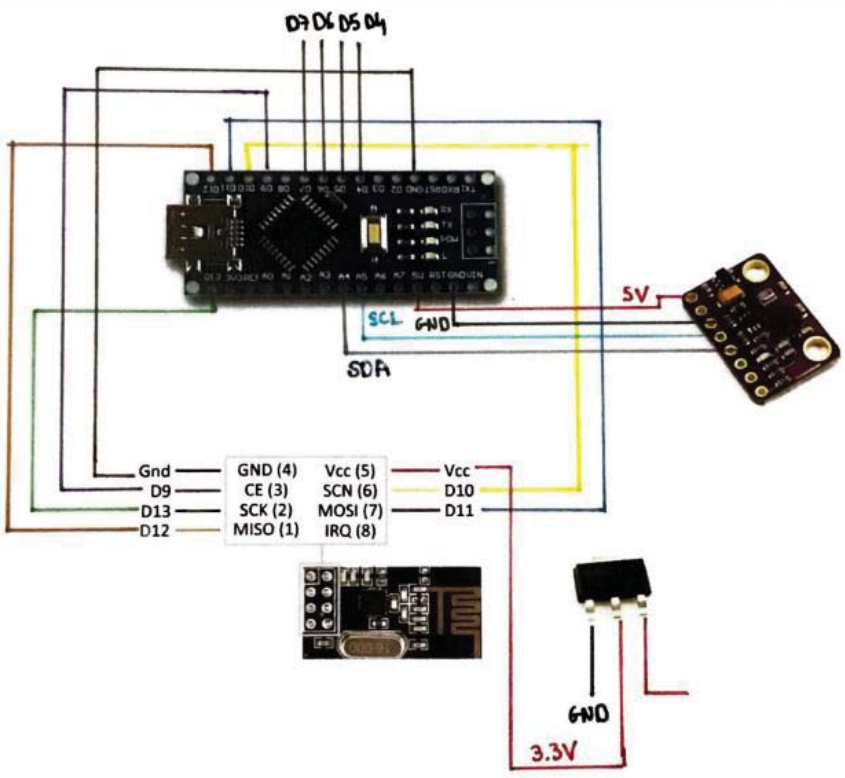


Figura 22. Conexiones receptor + controlador de vuelo.

3.1.4- Montaje final

Hay que asegurarse de que los motores giren en la dirección correcta.

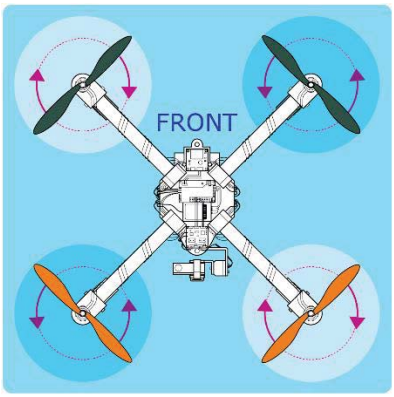


Figura 23. Dirección de giro de los motores.

Se sueldan los tres cables de los ESC a los motores. Hay que tener en cuenta que se debe de configurar el rango de PWM de los ESCs de 1000 a 2000 microsegundos mediante el código diseñado para ello (adjuntado en el anexo de programación). Una vez configurados, se unen, junto con los motores, a los brazos y se conectan a la batería mediante un conector LiPo.

3.2- Diseño piezas en Inventor

A continuación, se va a explicar cada una de las piezas utilizadas para la construcción del dron.

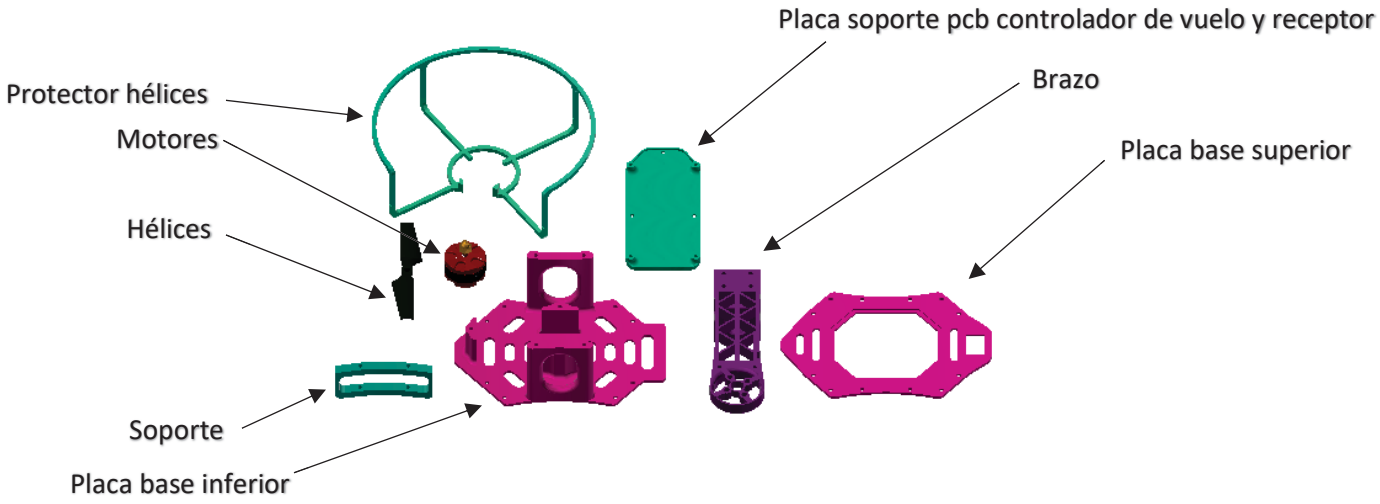


Figura 24. Conjunto de piezas diseñadas para el dron.

3.2.1- Brazo

En sus extremos se ubican los motores del dron atornillados a él. Estos motores van unidos al controlador de vuelo mediante los controladores electrónicos de velocidad (ESC), los cuales, recorren los brazos por su parte inferior hasta llegar a las placas base pasando sus cables por el hueco diseñado para ello. Los brazos se atornillan a ambas placas, superior e inferior.

Se han diseñado de la manera más aerodinámica y ligera posible, introduciendo el mayor número de huecos que la estructura permite. También se ha preparado en ellos el alojamiento

del protector de las hélices para introducirlo a presión. Para evitar el caso de que la estructura recibiese un impacto y alguno de los cuatro protectores rompiera quedándose parte de ellos dentro del brazo, se han añadido unos agujeros encima de sus alojamientos, en la parte superior del brazo, para poder retirar esa parte de la pieza con facilidad.

El primer diseño pensado para el alojamiento del motor, no fue práctico por el hecho de que los motores no presentaban suficiente espacio para su refrigeración, con lo cual, se elevaba la temperatura de éstos y finalmente se detenían.

Al modificar el alojamiento del motor, el principal objetivo fue el espacio dedicado al motor de manera que pudiese refrigerarse y en el nuevo diseño, no se han vuelto a presentar problemas de este tipo.

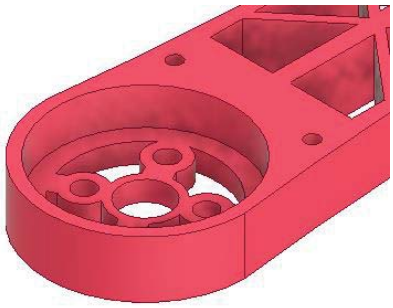


Figura 25. Alojamiento del motor del diseño inicial.

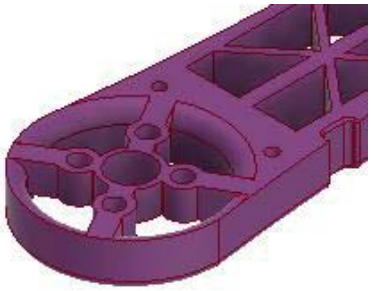


Figura 26. Alojamiento del motor del diseño final.

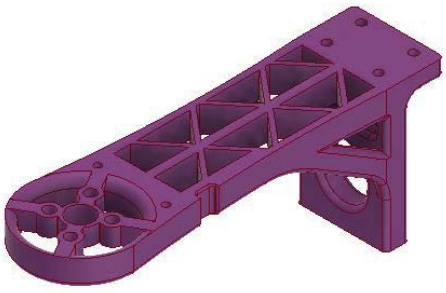


Figura 27. Vista 1 del brazo.

Figura 28. Vista 2 del brazo.

3.2.2- Placa base superior

Se coloca en el centro de la estructura, sobre la placa inferior y entre ellas se encuentra toda la electrónica necesaria para el vuelo. A ella van unidos los cuatro brazos. Se ha diseñado con un gran hueco central, para posteriormente, añadirle un plástico trasparente y que se pueda observar su contenido interior.

La placa contiene un orificio cuadrado para alojar el interruptor de encendido y apagado del dron.

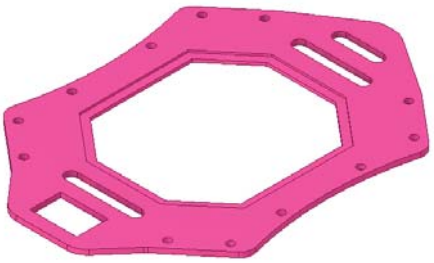


Figura 29. Placa superior.

3.2.3- Placa base inferior

En esta placa se han añadido dos paredes laterales con el fin de realizar la unión con la superior más robusta. Los cuatro brazos también se unen a esta placa por su parte inferior. En su centro, se ha establecido el alojamiento de la batería y encima de ésta irá una plaquita atornillada a la que se le atornillará a su vez la pcb (printed circuit board) del receptor y controlador de vuelo.

En su parte inferior, se encuentran dos agujeros a los que irán atornillados los dos soportes del tren de aterrizaje.

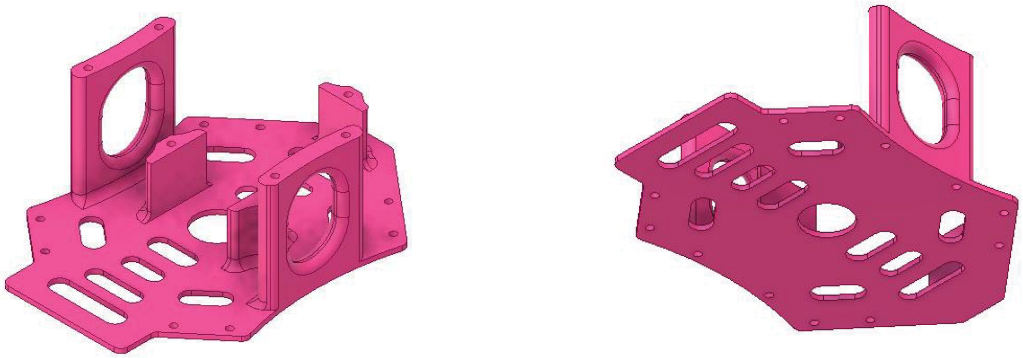


Figura 30. Vista superior e inferior placa inferior.

3.2.4- Soportes

Se trata del tren de aterrizaje, donde el dron se apoya al tomar tierra. Se colocarán dos de ellos, atornillados a la placa base inferior.

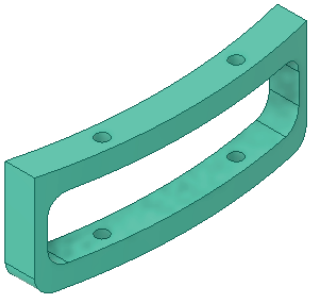


Figura 31. Soporte.

3.2.5- Placa soporte pcb controlador de vuelo y receptor

Como se ha comentado anteriormente, se ha diseñado una placa para alojar la pcb del controlador de vuelo y receptor, la cual irá atornillada a ella y ésta a su vez a la placa inferior.

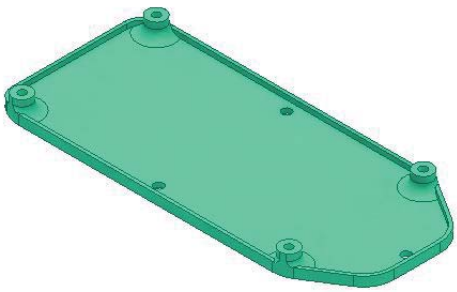


Figura 32. Placa soporte pcb controlador de vuelo y receptor.

3.2.6- Motores

Se han diseñado unos motores similares a los realmente elegidos para poder realizar el ensamblaje completo, los planos de conjunto y despiece.

Figura 33. Vista superior e inferior del motor.

3.2.7- Hélices

Se han diseñado unas hélices similares a las realmente elegidas para poder realizar el ensamblaje completo, los planos de conjunto y despiece.

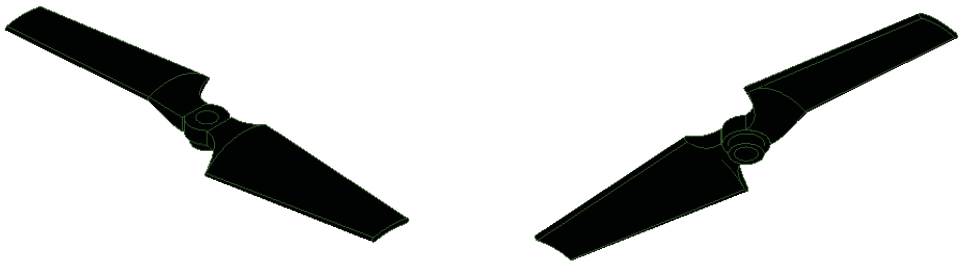


Figura 34. Vista superior e inferior de la hélice.

3.2.8- Protector hélices inferior

Se ha diseñado un protector para las hélices de manera que las rodee para evitar su rotura en futuros golpes del dron. Se ha dividido su diseño en dos partes (inferior y superior). Ésta pieza irá unida al brazo a presión en el alojamiento preparado para ello.

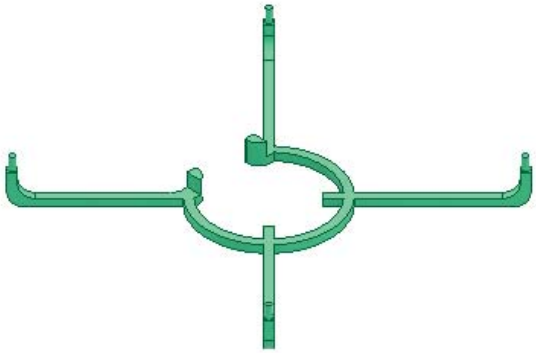


Figura 35. Protector hélices inferior.

3.2.9- Pieza superior protector hélices

La parte superior del protector va unida a presión a la superior.

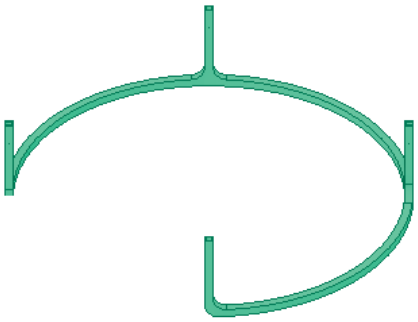


Figura 36. Protector hélices superior.

A continuación, se va a explicar cada una de las piezas utilizadas para la construcción del mando transmisor.

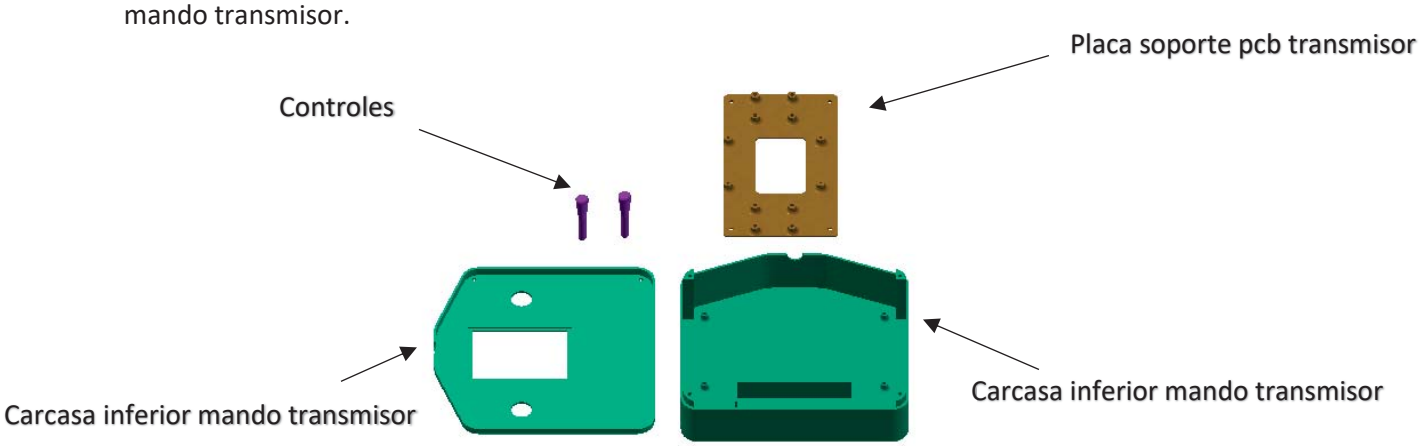


Figura 37. Conjunto de piezas diseñadas para el mando.

3.2.10- Carcasa inferior mando transmisor

En el interior de esta pieza se situará la electrónica necesaria para realizar el transmisor (joysticks, módulo de radio, Arduino, etc.). A ella irá atornillada una plaquita fina a la cual se le atornillarán la pcb del transmisor y los joysticks. También se ha realizado una extrusión rectangular para alojar al portapilas que alimentará el circuito, un agujero lateral donde estará situado el interruptor de encendido y apagado del mando y un hueco para que la antena del módulo de radio sobresalga del mando.

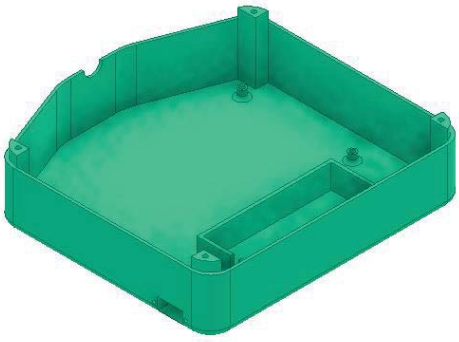


Figura 38. Carcasa inferior mando transmisor.

3.2.11- Carcasa superior mando transmisor

Es la parte superior de la pieza anterior la cual va atornillada a ella. Se le han realizado dos agujeros simétricos por donde controlar los joysticks y una extrusión negativa rectangular para poder observar la electrónica del interior.

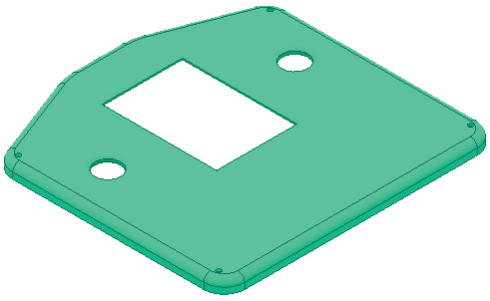


Figura 39. Carcasa superior mando transmisor.

3.2.12- Placa soporte pcb transmisor

Se ha diseñado una plaquita para alojar la pcb del transmisor y los joysticks, que irán atornillados a ella y ésta a su vez a la parte inferior del mando.

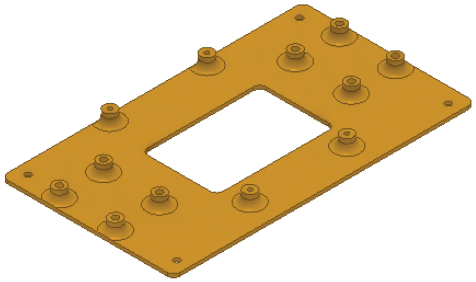


Figura 40. Placa soporte pcb transmisor.

3.2.13- Controles

Se trata de las piezas que irán colocadas en los joysticks y sobresaldrán del mando para poder controlarlos desde fuera del conjunto.

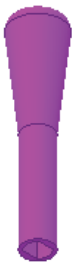


Figura 41. Controles.

3.3- Ensamblaje de piezas en Inventor

Para realizar el ensamblaje de las piezas se ha empleado dos tipos de restricciones:

- Restricción de coincidencia eje con eje: se ha empleado para todas aquellas piezas que van ensambladas de manera concéntrica (hélices a los motores, motores a los brazos, los brazos a ambas placas, tornillos a las piezas, carcasa inferior del mando transmisor a carcasa superior, etc.).
- Restricción de coincidencia plano con plano: se ha empleado para ensamblar piezas, cara con cara (los brazos a ambas placas y éstas entre sí, tornillos a las piezas, ambas carcasas del mando transmisor, etc.).

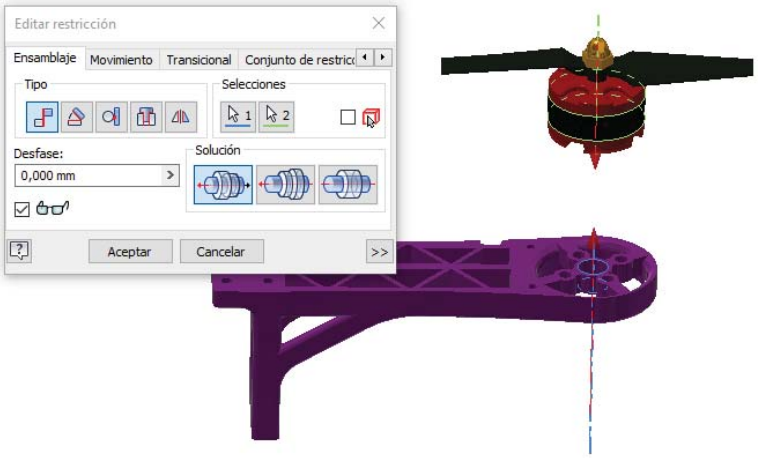


Figura 42. Restricción de coincidencia eje con eje.

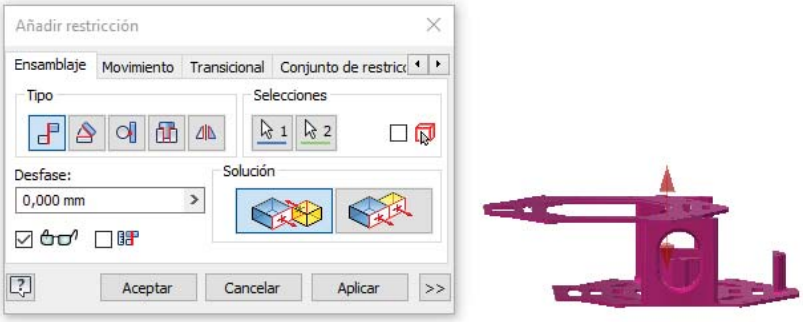


Figura 43. Restricción de coincidencia plano con plano.

Figura 44. Vista 1 ensamblaje final del dron.

Figura 45. Vista 2 ensamblaje final del dron.

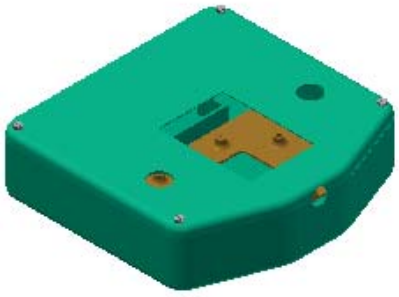


Figura 46. Vista 1 ensamblaje final del mando transmisor.

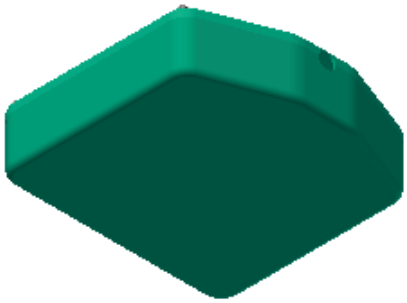


Figura 47. Vista 2 ensamblaje final del mando transmisor.

3.4- Simulación elementos finitos

Para validar el diseño, se procede a realizar un estudio de fuerzas en Autodesk Inventor.

Para comenzar, se crea un nuevo “estudio estático” de la pieza que se quiera analizar, se fijan las aristas que vayan fijas en el ensamblaje y se añade una fuerza de 100 N simulando, por ejemplo, la fuerza de impacto que recibiría la pieza en caída libre.

Para aumentar la precisión de la malla triangular se configura la malla modificando el “tamaño medio del elemento” reduciéndolo a 0,04. Modificando esta propiedad se modifica el tamaño del elemento en relación con el tamaño del modelo, los triángulos que configuran la malla son más pequeños y por tanto la malla es más densa (el valor máximo que se puede asignar es 1). Una malla más densa, requerirá mayor tiempo de análisis. A continuación, se modifica el “ángulo máximo de giro” a 30 grados. Este parámetro permite controlar el número de elementos a lo largo de un arco de 90º. Un valor de 60º creará un mínimo de dos elementos para rellenar un arco de 90º, mientras que un valor de 30º creará al menos tres elementos para el mismo arco. Se recomienda que este parámetro esté entre esos dos valores (30º y 60º) ya que un valor menor, generará una malla excesivamente densa cuando el modelo contenga agujeros.

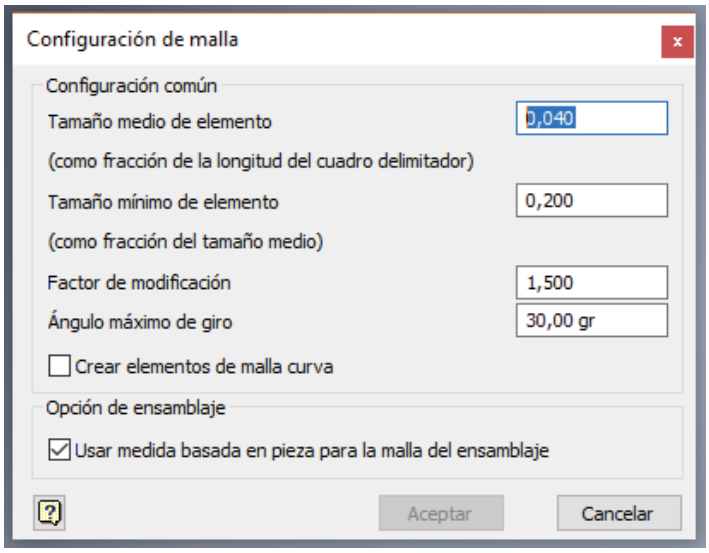


Figura 48. Configuración de malla.

También se deben modificar las propiedades de la configuración de convergencia. Los "criterios de parada (%)" se disminuyen a un 1%. Este parámetro se emplea para la convergencia entre dos refinados consecutivos, el cual se suspende cuando la diferencia entre los dos últimos resultados es inferior al valor (%) especificado. El refinado podrá detenerse cuando se alcance el número máximo de refinados, aunque se cumplan los criterios de parada. Por otra parte, el "número máximo de refinados" se aumenta a 5, y especifica el número máximo de ciclos de refinado h para la convergencia. Si se cumplen los criterios de parada, los refinados podrían detenerse antes de alcanzar el número. Los valores superiores a 5 podrían resultar en singularidades de tensión y conlleva mucho tiempo en analizarlo.

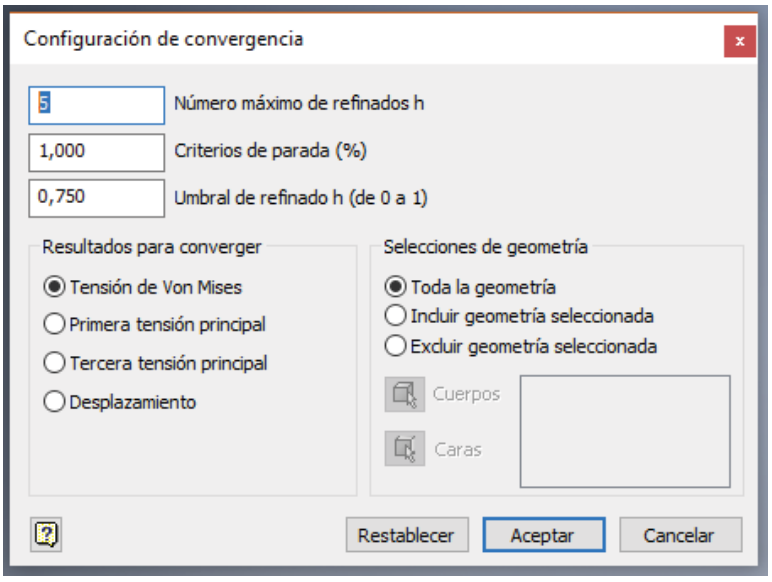


Figura 49. Configuración de convergencia.

Una vez realizadas estas modificaciones, para visualizar la malla de la pieza se selecciona “vista malla” de dicha pieza, se simula y se observa los resultados obtenidos estableciendo como tensión máxima el límite elástico del plástico ABS, 41 Mpa.

3.4.1- Análisis de datos y mejora de la estructura del brazo

Se comienza analizando el brazo en su diseño inicial.

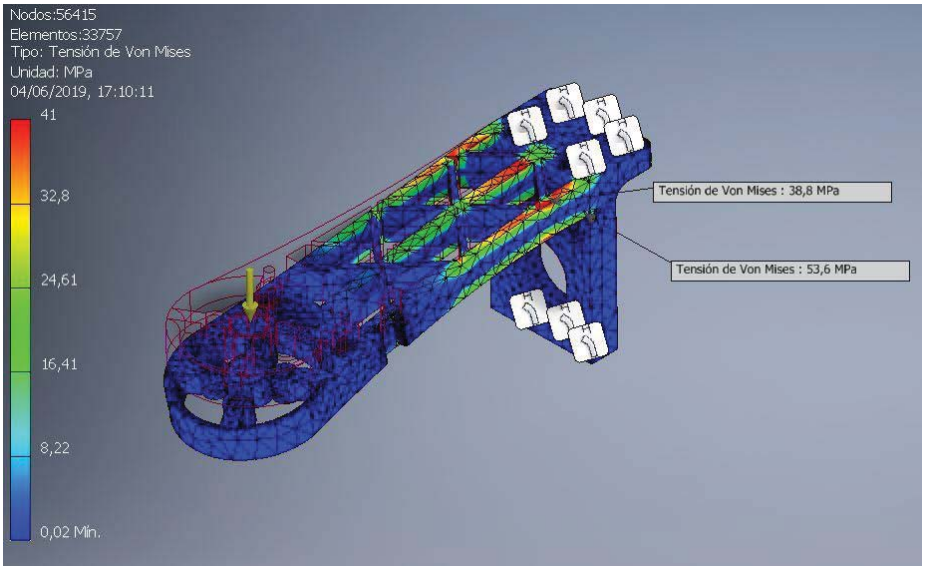


Figura 50. Análisis de tensión en el brazo.

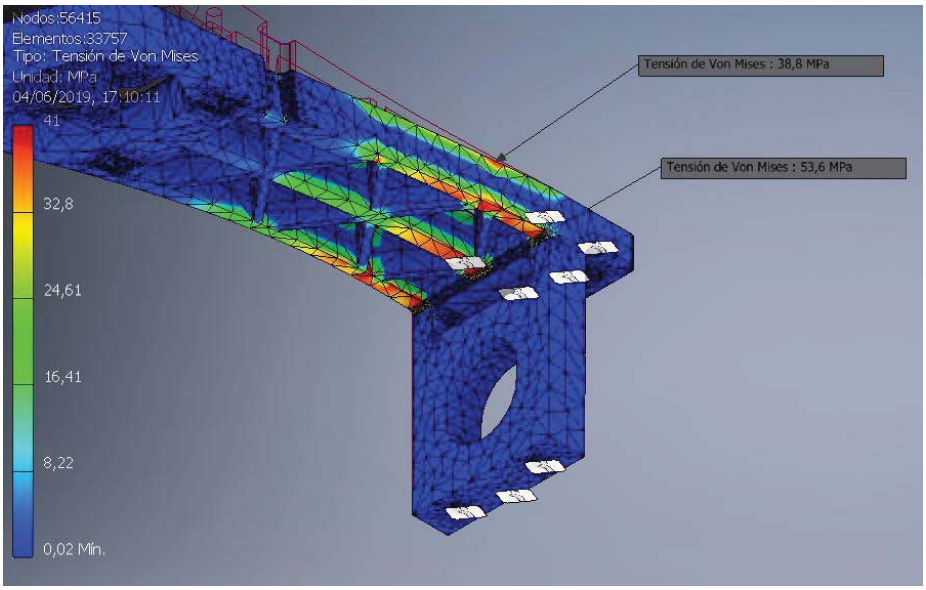


Figura 51. Detalle del análisis de tensión en el brazo.

Como se puede observar, la estructura supera el límite elástico del material en el brazo, por tanto, se aplican varias reformas para intentar evitarlo. La zona más conflictiva es la cara inferior del brazo y las esquinas.

Se plantea introducir varios nervios en la parte inferior de la pieza y realizar empalmes en las esquinas problemáticas y de esta forma, evitar puntos de concentración de tensiones. Se decide por tanto, aplicar dos nervios en los extremos de la pieza en lugar de uno central para respetar el futuro espacio del ESC en ella.

Una vez planteadas las reformas, se aplican a la estructura y se realiza un nuevo análisis de elementos finitos. Se vuelve a observar el resultado respecto al límite elástico del ABS como referencia y se comprueba que la respuesta ha mejorado muy notablemente, pero se sigue alcanzando.

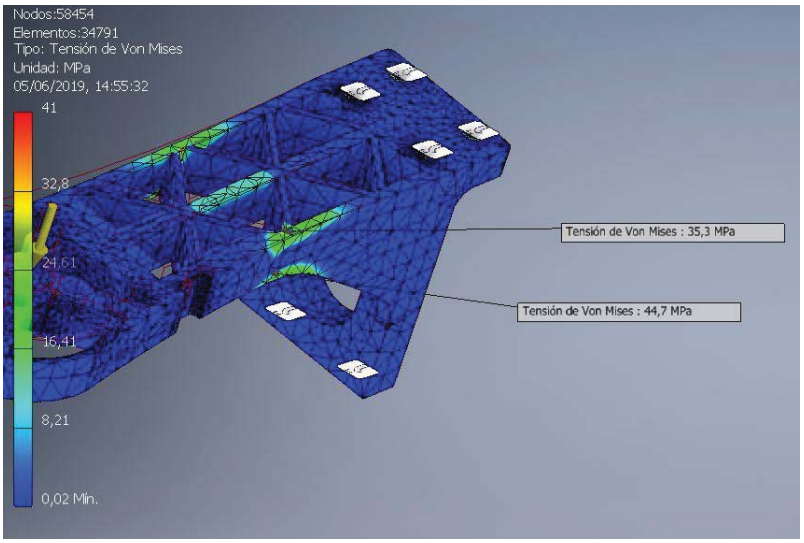


Figura 52. Detalle 1 del análisis de tensión en el brazo con empalmes.

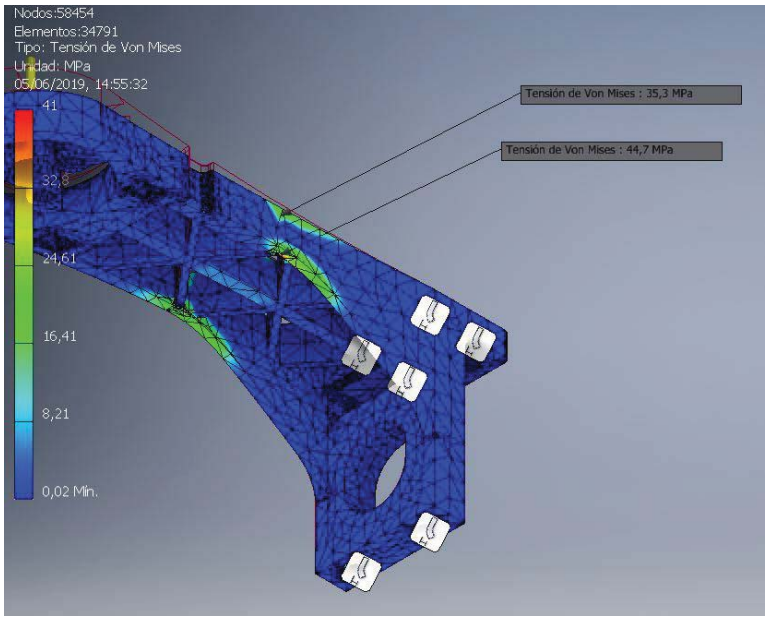


Figura 53. Detalle 2 del análisis de tensión en el brazo con empalmes.

Finalmente, se aplican empalmes en toda la pieza y se realizan dos extrusiones negativas en los nervios para reducir peso de la estructura.

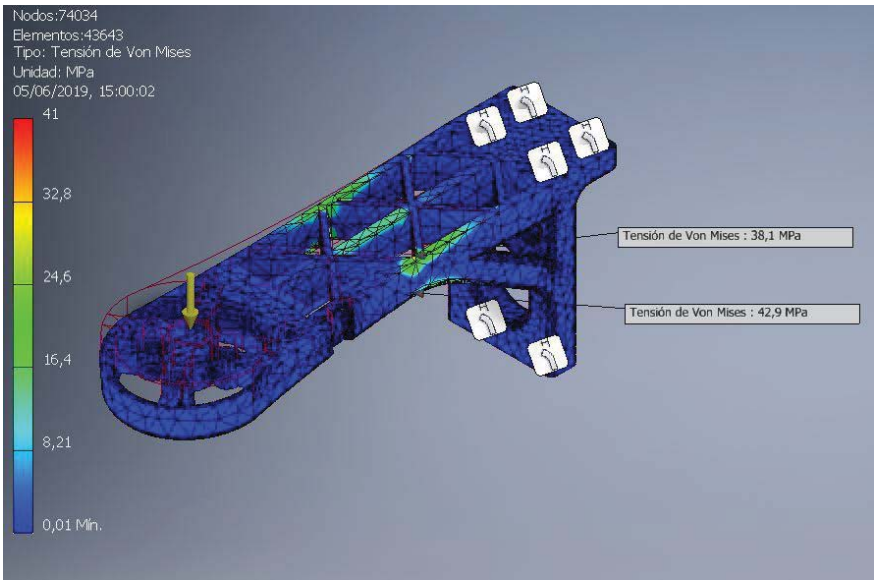


Figura 54. Vista 1 análisis de tensión con las mejoras implementadas.

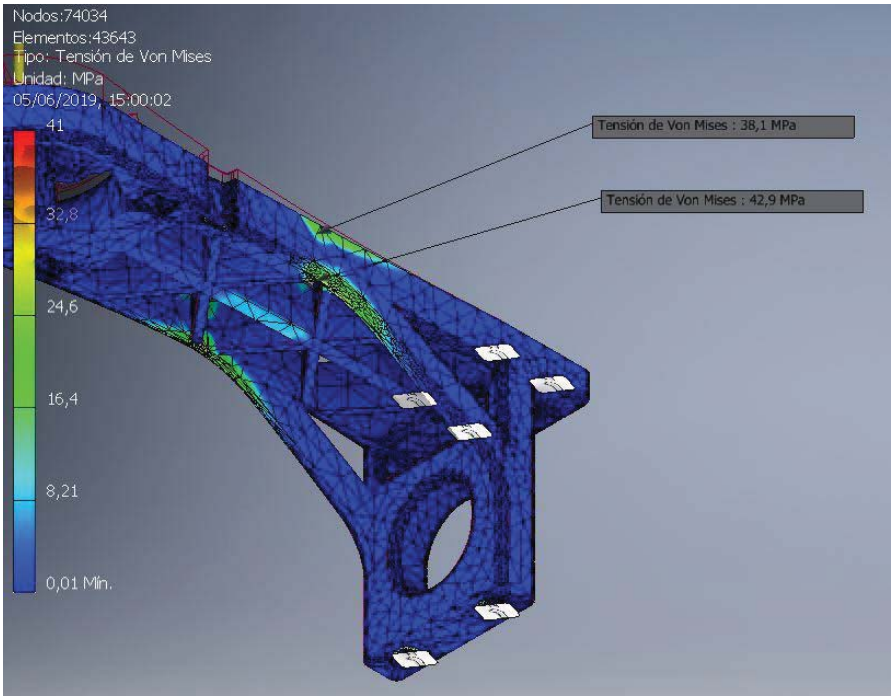


Figura 55. Vista 2 análisis de tensión con las mejoras implementadas.

3.4.2- Análisis de datos y mejora de la estructura del soporte

Se procede a analizar también los soportes.

Se aplica la fuerza en uno de sus extremos simulando un golpe recibido en esta zona y se observa que la tensión en ese punto es muy elevada.

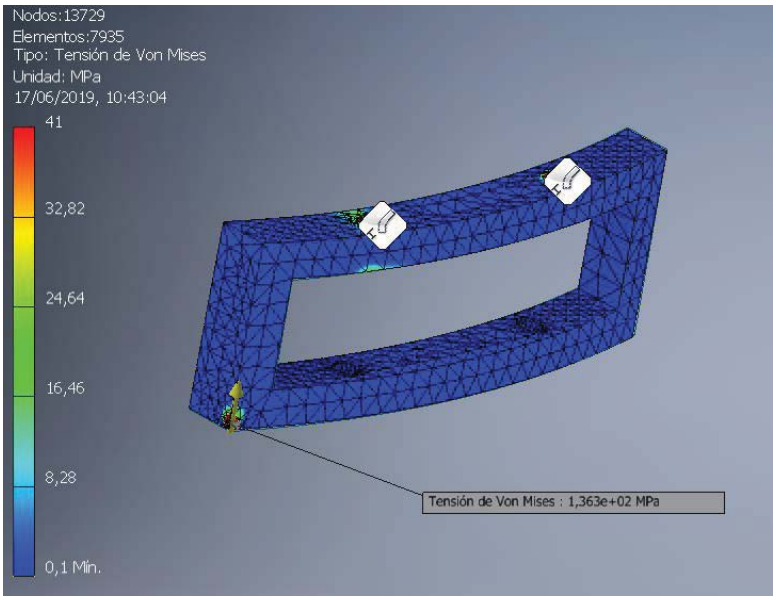


Figura 56. Análisis de tensión en el soporte sin empalmes.

Se decide aplicar empalmes por toda la pieza y con ello se observa una mejora muy notable en toda la estructura.

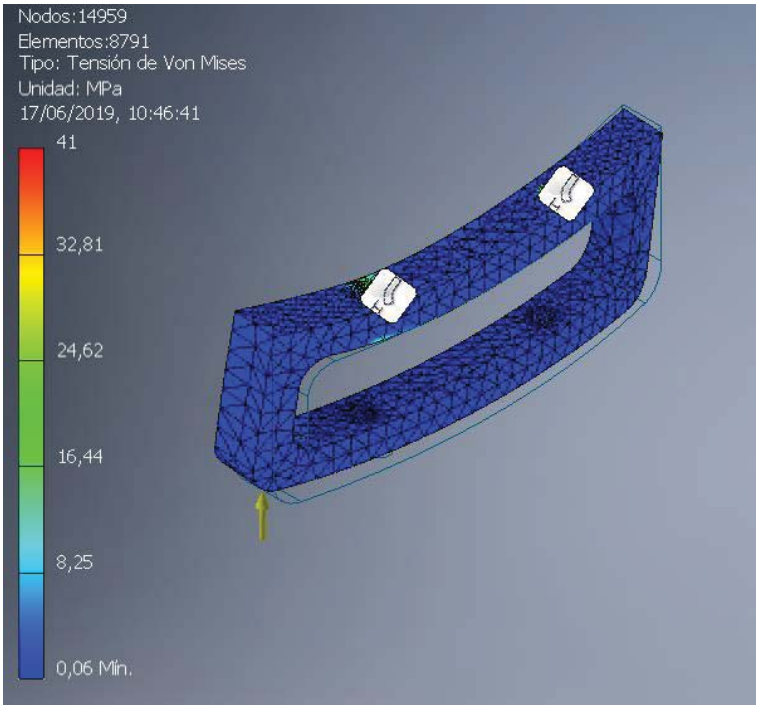


Figura 57. Análisis de tensión en el soporte con empalmes.

Capítulo 4 – Impresión de piezas

4.1- Parámetros relevantes de la impresión

Los parámetros a destacar sobre el proceso de impresión son los siguientes:

- Altura de capa: uno de los parámetros más importantes, la configuración de éste cambia el aspecto de las piezas. La altura de capa es equivalente a la distancia que el extrusor realiza en sentido positivo en el eje Z al cambiar de una capa a otra. Se ha utilizado una altura de capa de 0.1mm.

- Grosor del material de impresión (PLA): los grosores de filamento que se utilizan son de 3 mm y 1.75 mm. Utilizar un filamento de 3 mm puede ser un inconveniente al requerir una baja velocidad de extrusión y como resultado quedar restos de material en la pieza impresa. Por ello, se emplea un grosor de material de impresión de 1.75mm.

- Temperatura de la placa de impresión: es la temperatura que alcanza la cama de impresión y su rango es de 30 a 70 ºC. Se ha configurado este parámetro a 60º C.

- Temperatura del filamento: el rango de temperatura de impresión se encuentra entre 190 y 220 ºC. Este parámetro se ha configurado a 210º C.

- Velocidad de impresión: se trata de un parámetro muy importante. Cuanto menor sea la velocidad de impresión, mejor acabado se obtendrá. Se ha empleado una velocidad de impresión de 45 mm/seg.

- Densidad de relleno: se trata de un parámetro que define la solidez de la pieza. Para determinar el porcentaje de relleno se debe tener en cuenta las futuras fuerzas que deberá soportar la pieza una vez impresa. A mayor porcentaje de relleno, mayor densidad se obtiene y por tanto mayor resistencia, siendo el 100% una pieza totalmente maciza. Este parámetro se ha configurado al 15%.

4.2- Tiempo de impresión

El tiempo de impresión de cada pieza se detalla a continuación:

PIEZA	TIEMPO (por unidad)	UNIDADES	DURACIÓN
Brazo	5h 50min	4	23h 20 min
Placa base superior	1h 32min	1	1h 32min
Placa base inferior	9h 4min	1	9h 4min
Soporte	1h 34min	2	3h 8min
Placa soporte pcb controlador de vuelo y receptor	1h 26 min	1	1h 26 min
Protector hélices inferior	1h 11min	4	4h 44min
Protector hélices superior	1h 50min	4	7h 20min
Carcasa inferior mando transmisor	15h 32min	1	15h 32min
Carcasa superior mando transmisor	4h 35min	1	4h 35min
Placa soporte pcb transmisor	2h 22min	1	2h 22min
Controles	50min	2	1h 40min
			DURACIÓN TOTAL
			74h 43min

Tabla 4. Tiempo de impresión de las piezas.

Por lo tanto, el tiempo dedicado a la impresión 3D de las piezas es de 74 horas y 43 minutos.

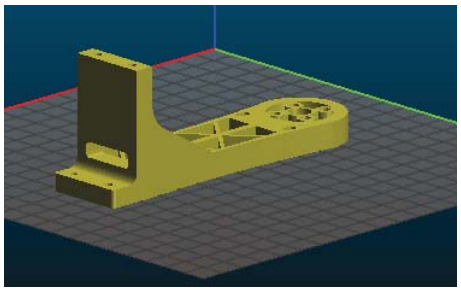


Figura 58. Brazo en Slic3r.

Figura 59. Brazo impreso.

Capítulo 5 – Conclusiones

Se concluye que se ha podido realizar un dron cuadricóptero controlado mediante un sistema de transmisión-recepción con microcontroladores Arduino Nano.

El peso final del dron es de 716 g, menor que el peso inicial estimado, por lo tanto, se verifica que los componentes han sido correctamente seleccionados.

5.1- Problemas y dificultades

A lo largo de la construcción del dron, se han presentado varios problemas con elevada dificultad de detectar y solucionar.

Uno de los principales problemas surgidos, ha sido la presencia de imperfecciones en los motores que se han comprado. Además de lo ya comentado anteriormente sobre el problema de admisión de aire y el diseño de la estructura, se ha detectado otro problema más, evidenciado por el arranque de los motores. El problema trata de que el motor no arranca correctamente por el hecho de presentar sus agujeros roscados no paralelos entre sí y, por lo tanto, al unirlos a la estructura, se crea una ligera pero notoria presión en el motor que impide su correcto arranque. Para solucionar este problema, se ha aumentado el diámetro de los agujeros de la estructura, de tal forma que el motor presente la holgura suficiente para facilitar el arranque y además se ha colocado entre los motores y la estructura un material elástico cuya función sea amortiguar el movimiento de éstos.

El segundo de los problemas relevantes que han surgido ha sido la dificultad de conseguir el correcto funcionamiento de los módulos de radio NRF24. Este problema ha sido debido a la incorrecta soldadura del regulador de voltaje del mando transmisor, produciéndose un cortocircuito en él y consecuentemente, inhabilitando tanto el regulador como el módulo de radio, ya que se le han aplicado a éste más de 3.3V.

Por último, se destaca el último problema presentado en el proyecto. Se trata de la poca imperfección y sensibilidad que presentan los joysticks del mando transmisor, ya que prácticamente se pasa del mínimo valor al valor medio directamente, y de éste al máximo valor, resultando muy complicado el correcto manejo del dron, ya que para su arranque se requiere elevar el 'throttle' muy poco a poco para no perder el control de los motores.

5.2- Ampliaciones y mejoras

Al tener problemas en los motores y los joysticks, se propone el cambio de éstos para eliminar imperfecciones en el proyecto y conseguir un resultado mejorado.

También se propone introducir el módulo 'ESP-32S Developer' como sustituto del Arduino y módulo de radio NRF24 ya que cubre las mismas necesidades, tanto en el transmisor como en el receptor y controlador de vuelo. De esta manera, se ahorraría dinero, peso del conjunto y tiempo de conexionado y soldadura.

Como posibles ampliaciones, se destaca la incorporación de una cámara que capte imágenes y éstas puedan ser vistas desde un dispositivo móvil, recibidas mediante comunicación Bluetooth.

Figura 60. Foto final del dron cuadricóptero.

REFERENCIAS BIBLIOGRÁFICAS

- Apuntes de la asignatura de Ingeniería Gráfica.
- Apuntes de la asignatura de Impresión 3D.
- Apuntes de la asignatura de Ingeniería Informática.
- Apuntes de la asignatura de Métodos Matemáticos.
- Apuntes de la asignatura de Tecnología Automática.
- Conceptos generales drones:

  https://wwww.rctecnic.com/blog/107_que-es-un-drone--tipos-nombres-y-componentes.html
- Autodesk Inventor:

  https://www.autodesk.es/products/inventor/overview
- Wasim Younis. (2012). Inventor y su simulación con ejercicios prácticos.
- Slic3r:

  https://slic3r.org/
- Arduino:

  https://www.arduino.cc/
- Módulo NRF24:

  https://www.luisllamas.es/comunicacion-inalambrica-a-2-4ghz-con-arduino-y-nrf24l01/
- IMU:

  https://www.luisllamas.es/usar-arduino-con-los-imu-de-9dof-mpu-9150-y-mpu-9250/

PRESUPUESTO

Diseño y desarrollo de un prototipo físico de dron cuadricóptero

Capítulo 1.- Contenido del presupuesto

El presupuesto se va a realizar diferenciando entre costes del jornal de la mano de obra y costes del material empleado para la realización del proyecto.

Dentro de los costes de mano de obra encontramos el salario de una Graduada en Ingeniería Industrial y se tiene en consideración las horas dedicadas al proyecto según la implicación y dificultad de la tarea.

Por otra parte, dentro de los costes de material se incluyen todos los productos, software y hardware empleados.

1.1.- Cuadro de precios

1.1.2- Cuadro de precios 1: Mano de obra

Se ha dedicado 12 semanas para la realización de este Trabajo de Fin de Grado, con una dedicación de 5 horas diarias, 5 días a la semana, un total de 300 horas. La distribución de las tareas ha sido la siguiente:

Nº de semana	Actividad
1-3	Planteamiento de diseño
4-5	Modelado
5-6	Generación de planos
6-7	Validación del diseño
8-9-10	Montaje del prototipo
10-11-12	Programación de la electrónica
3,5,6,7,10,12	Elaboración de la documentación final

Tabla 1. Programación de las tareas.

Se ha requerido el trabajo de una graduada en Ingeniería en Tecnologías Industriales para el siguiente listado de tareas:

- Planteamiento del diseño: previamente al modelado 3D se ha realizado un diseño conceptual del producto, donde se han barajado diferentes opciones estructurales del mismo como los diferentes dispositivos electrónicos a emplear para su control.
- Modelado: diseño y producción 3D de cada una de las partes que forman el proyecto así como la elaboración del ensamblaje, donde se puede observar el prototipo virtual.
- Generación de planos: planos de cada una de las piezas que posteriormente se han impreso en 3D así como planos de conjunto y despiece.
- Validación del diseño: se ha realizado un estudio de elementos finitos, en ciertas piezas susceptibles de recibir un futuro impacto, con el cual se ha podido mejorar el diseño de la pieza para una respuesta correcta frente a diferentes esfuerzos.
- Montaje del prototipo: incluye tanto el montaje de las piezas impresas en 3D como el montaje electrónico.
- Programación de la electrónica: programación en Arduino IDE de todas las partes electrónicas que componen el proyecto (transmisor y receptor + controlador de vuelo).
- Elaboración de la documentación final: ha sido redactado de forma paralela al resto de actividades.

Para realizar correctamente los costes de este apartado, se establece el tiempo empleado para realizar cada una de estas actividades. El tiempo es medido en horas.

Unidad de Obra	Unidad	Descripción	Rendimiento	Precio	Importe (€)
1.1		**Planteamiento del diseño**			
	h	Graduada en Ingeniería Industrial	60	15,00	900,00
	%	Costes directos complementarios	2	900,00	18,00
		Coste total			918,00
1.2		**Modelado**			
	h	Graduada en Ingeniería Industrial	35	15,00	525,00
	%	Costes directos complementarios	2	525,00	10,5
		Coste total			535,5
1.3		**Generación de planos**			
	h	Graduada en Ingeniería Industrial	20	15,00	300,00
	%	Costes directos complementarios	2	300,00	6
		Coste total			306,00
1.4		**Validación del diseño**			
	h	Graduada en Ingeniería Industrial	30	15,00	450,00
	%	Costes directos complementarios	2	450,00	9
		Coste total			459,00
1.5		**Montaje del prototipo**			
	h	Graduada en Ingeniería Industrial	60	15,00	900,00
	%	Costes directos complementarios	2	900,00	18
		Coste total			918,00
1.6		**Programación de la electrónica**			
	h	Graduada en Ingeniería Industrial	55	15,00	825,00
	%	Costes directos complementarios	2	825,00	16,5
		Coste total			841,5
1.7		**Elaboración de la documentación final**			
	h	Graduada en Ingeniería Industrial	40	15,00	600,00

	%	Costes directos complementarios	2	600,00	12
		Coste total			612,00
	-	**TOTAL MANO DE OBRA**	-	-	4590,00

Tabla 2. Costes de la mano de obra.

Por lo tanto, el coste de la mano de obra es de un total de 4590,00 €.

1.1.2.- Cuadro de precios 2: Materiales y herramientas

El material que se ha empleado ha sido el siguiente:

Herramienta o material	Precio (€/unidad)
Impresión 3D	18 €/ud
Arduino NANO	3,74 €/ud
Cable USB mini	4,00 €/ud
Motor RS2205 2300Kv	6,745 €/ud
ESC 30 A	6,00 €/ud
Módulo NRF24L01 + Antena	9,06 €/ud
Módulo NRF24L01	2,30 €/ud
MPU9250	8,75 €/ud
Pcb 50x70	0,30 €/ud
Regulador de voltaje 3.3V	0,36 €/ud
Batería LiPo 2200mAh 25C 3S	24,99 €/ud
Cargador batería LiPo 3S	6,99 €/ud
Portapilas	1,36 €/ud
Pila recargable 3,6V	5,20 €/ud
Cargador pilas recargables 3,6V	4,25 €/ud
Soldador	7,90 €/ud
Estaño	2,69 €/ud
Hélices	0,8125 €/ud
Cableado	0,435 €/m
Interruptor deslizante	1,21 €/ud
Condensadores 10uF	0,34 €/ud
Conectores LiPo macho	0,55€/ud
Joysticks	5,295 €/ud
Reprografía	0,5 €/ud

Tabla 3. Precio de los materiales.

Para tener en cuenta los diferentes sistemas de software que han sido utilizados, se debe considerar la amortización de estos:

- Autodesk Inventor Professional 2019: el precio viene dado por el coste de la suscripción anual a Inventor Professional por lo tanto se fija la amortización en un año de suscripción.
- Microsoft Office Professional 2016: ha sido empleado para la redacción del proyecto. Se fija la amortización en un año de suscripción.
- Windows 10: licencia que contiene el ordenador. Se fija la amortización en un año de suscripción.

Software	Precio licencia (€/año)	Amortización (h/año)	Precio amortizado (€/hora)
Autodesk Inventor Professional 2019	2613,60	1400	1,86685
Microsoft Office Professional 2013	99,00	1400	0,07071
Windows 10	50,00	1400	0,03571

Tabla 4. Precio a amortizar de Software.

Se puede estimar que las licencias anuales se emplean aproximadamente 1400 horas al año.

Por último, respecto a los sistemas hardware empleados:

- Ordenador portátil: para la realización de este proyecto se ha empleado un LENOVO-PC. El precio de este ordenador portátil en el momento de su compra fue de 199,00 euros. Se estima el uso del ordenador en 8 horas en una jornada laboral, que considerando 251 días laborales, es un total de 2008 horas al año.
- Impresora JGaurora: el precio de la impresora en el momento de su adquisición fue de 299,00 euros. Para imprimir el trabajo se han empleado 74 horas y 43 minutos. Se considera una amortización de 800 horas al año.

Hardware	Precio total (€)	Amortización (h/año)	Precio amortizado (€/hora)
Ordenador portátil	199,00	2008	0,0991
Impresora JGaurora	299,00	800	0,3737

Tabla 5. Precio a amortizar de Hardware.

Finalmente, se obtiene el presupuesto de todos los materiales y herramientas:

Unidad de Obra	Unidad	Descripción	Cantidad	Precio unitario	Importe (€)
2.1		**Impresión 3D**			
	bobina	Impresión 3D	1,00	18 €/bobina	18,00
	h	Amortización impresora 3D	74,72	0,3737 €/h	27,9266
		Coste total			45,9266
2.2		**Montaje dron**			
	Ud	Arduino NANO	3,00	3,74 €/ud	11,22
	Ud	Motor RS2205 2300Kv	4,00	6,745 €/ud	26,98
	Ud	ESC 30 A	4,00	6,00 €/ud	24,00
	Ud	Módulo NRF24L01 + Antena	1,00	9,06 €/ud	9,06
	Ud	Módulo NRF24L01	1,00	2,30 €/ud	2,30
	Ud	MPU9250	1,00	8,75 €/ud	8,75
	Ud	Pcb 50x70	2,00	0,30 €/ud	0,60

	Ud	Regulador de voltaje 3.3V	2,00	0,36 €/ud	0,72
	Ud	Batería LiPo 2200mAh 25C 3S	1,00	24,99 €/ud	24,99
	Ud	Cargador batería LiPo 3S	1,00	6,99 €/ud	6,99
	Ud	Portapilas	1,00	1,36 €/ud	1,36
	Ud	Pila recargable 3.6V	2,00	5,20 €/ud	10,40
	Ud	Soldador	1,00	7,90 €/ud	7,90
	Ud	Estaño	1,00	2,69 €/ud	2,69
	Ud	Hélices	4,00	0,8125 €/ud	3,25
	m	Cableado	4,00	0,435 €/m	1,74
	Ud	Interruptor deslizante	2,00	1,21 €/ud	2,42
	Ud	Condensadores 10uF	2,00	0,34 €/ud	0,68
	Ud	Conectores LiPo macho	1,00	0,55 €/ud	0,55
	Ud	Joysticks	2,00	5,295 €/ud	10,59
	h	Amortización Autodesk Inventor	85	1,8668 €/h	158,6822
	h	Amortización Windows 10	200	0,0357 €/h	7,142
	h	Amortización ordenador	200	0,0991 €/h	19,8207
		Coste total			342,8349
2.3		**Redacción del documento**			
	h	Amortización Microsoft office	40	0,0707 €/h	2,8284
	h	Amortización Windows 10	40	0,0357 €/h	1,4284
	h	Amortización ordenador	40	0,1140 €/h	4,5617
	Ud	Reprografía	127	0,5 €/ud	63,5
		Coste total			72,3185
		Subtotal	**-**	**-**	**461,08**

Tabla 6. Costes de materiales y herramientas.

Capítulo 2.- Presupuesto general

Partiendo de los costes subtotales obtenidos, se añade un 2% de costes indirectos y un 21% de IVA.

Descripción	**Coste (€)**
Mano de obra	4590,00
Materiales y herramientas	461,08
Subtotal	5051,08
Costes indirectos	2%
CI	101,0216
Coste sin IVA	5152,1016
IVA	21%
IVA	1081,9413
Coste total del proyecto	**6234,04**

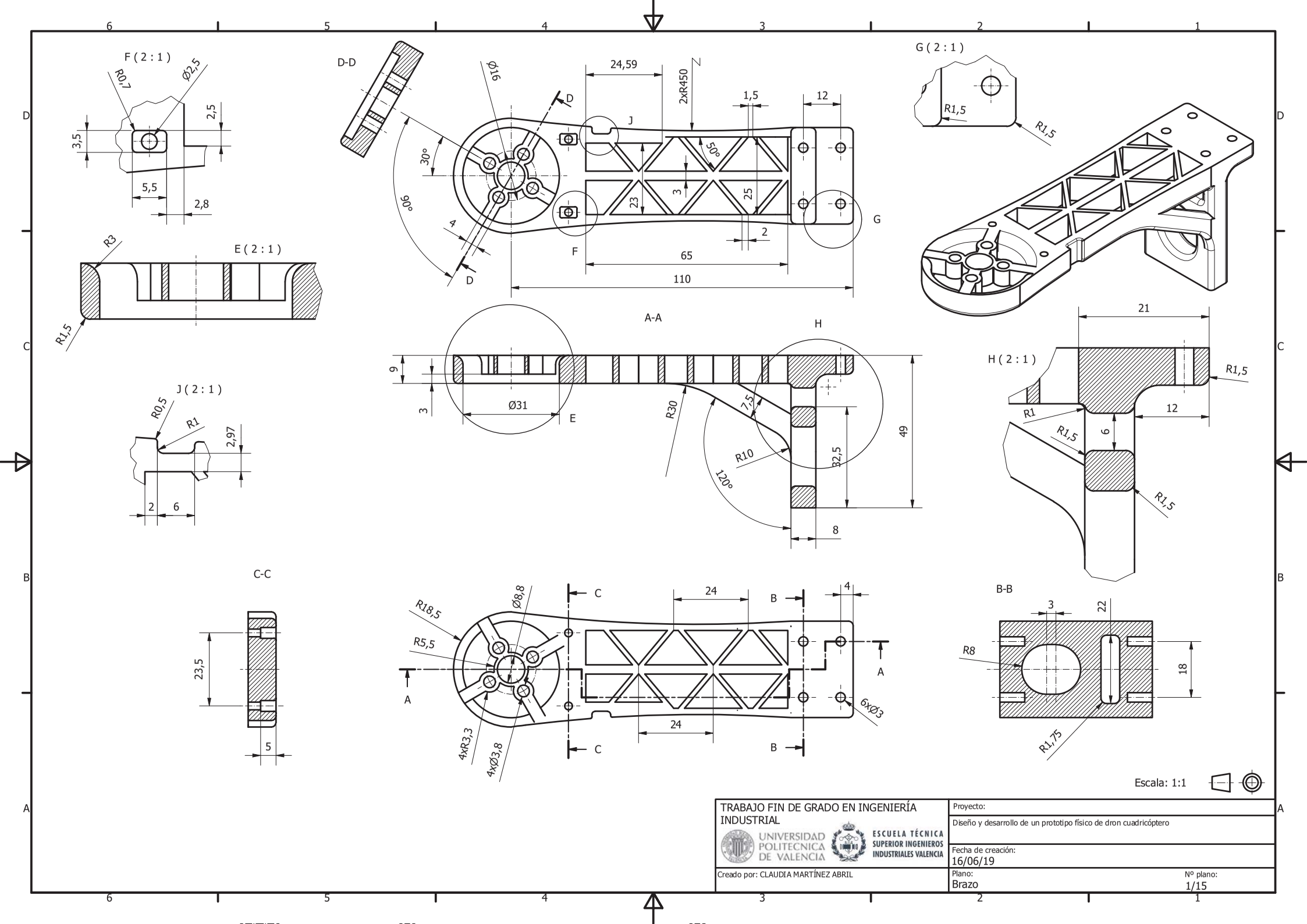
Tabla 7. Costes totales del proyecto.

Con lo cual, el coste total de este TFG es de:

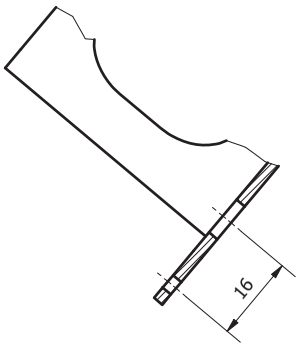
SEIS MIL DOSCIENTOS TRENTA Y CUATRO CON CUATRO CÉNTIMOS.

PLANOS

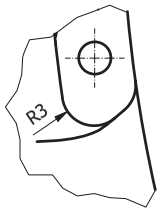
Diseño y desarrollo de un prototipo físico de dron cuadricóptero

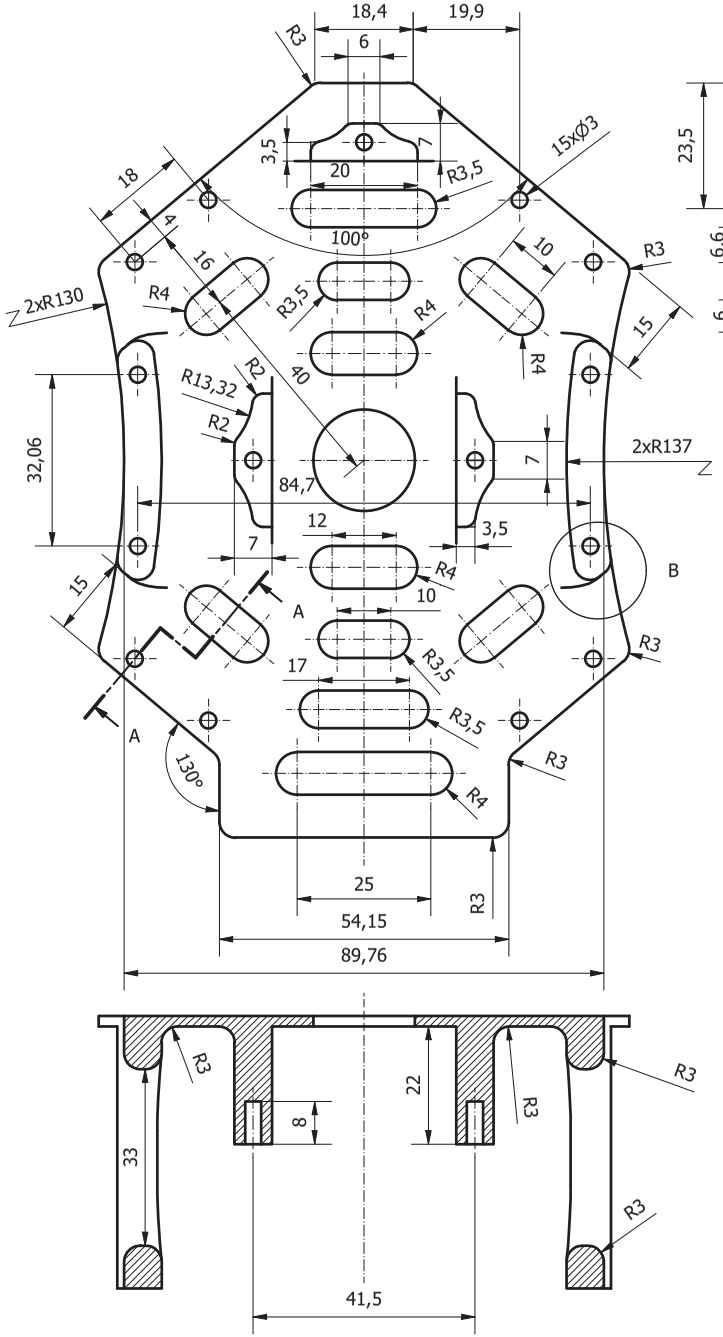

F (2 : 1)
R0,7
Ø2,5
2,5
3,5
5,5
2,8
E (2 : 1)
R3
R1,5
J (2 : 1)
R0,5
R1
2,97
2
6
C-C
23,5
5
D-D
Ø16
D
24,59
2xR450
1,5
12
J
30°
50°
90°
4
23
3
25
2
F
G
65
110
A-A
H
9
3
Ø31
E
R30
7,5
R10
120°
82,5
49
8
G (2 : 1)
R1,5
R1,5
H (2 : 1)
21
R1,5
R1
12
R1,5
6
R1,5
B-B
3
22
R8
18
R1,75
R18,5
R5,5
Ø8,8
C
24
B
4
A
6xØ3
4xR3,3
4xØ3,8
Escala: 1:1
TRABAJO FIN DE GRADO EN INGENIERÍA INDUSTRIAL
UNIVERSIDAD POLITECNICA DE VALENCIA
ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA
Proyecto:
Diseño y desarrollo de un prototipo físico de dron cuadricóptero
Fecha de creación:
16/06/19
Creado por: CLAUDIA MARTÍNEZ ABRIL
Plano:
Brazo
Nº plano:
1/15

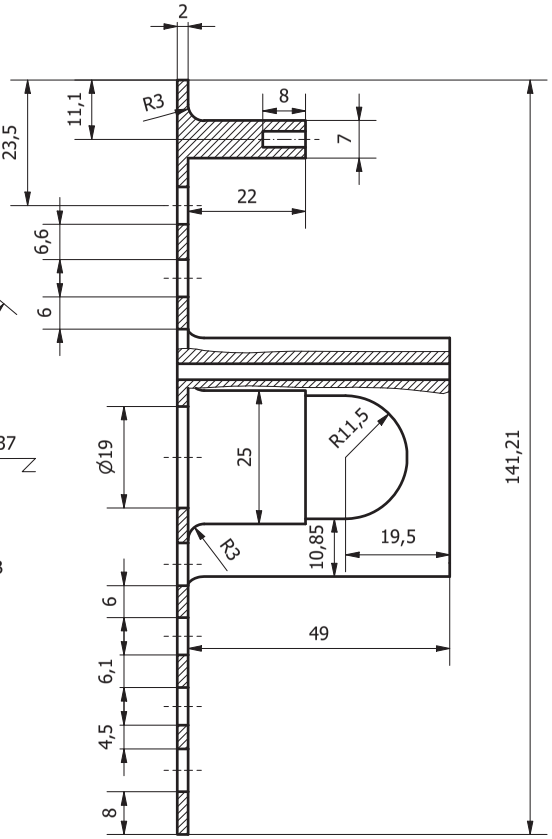
A-A (1 : 1)



B (2 : 1)

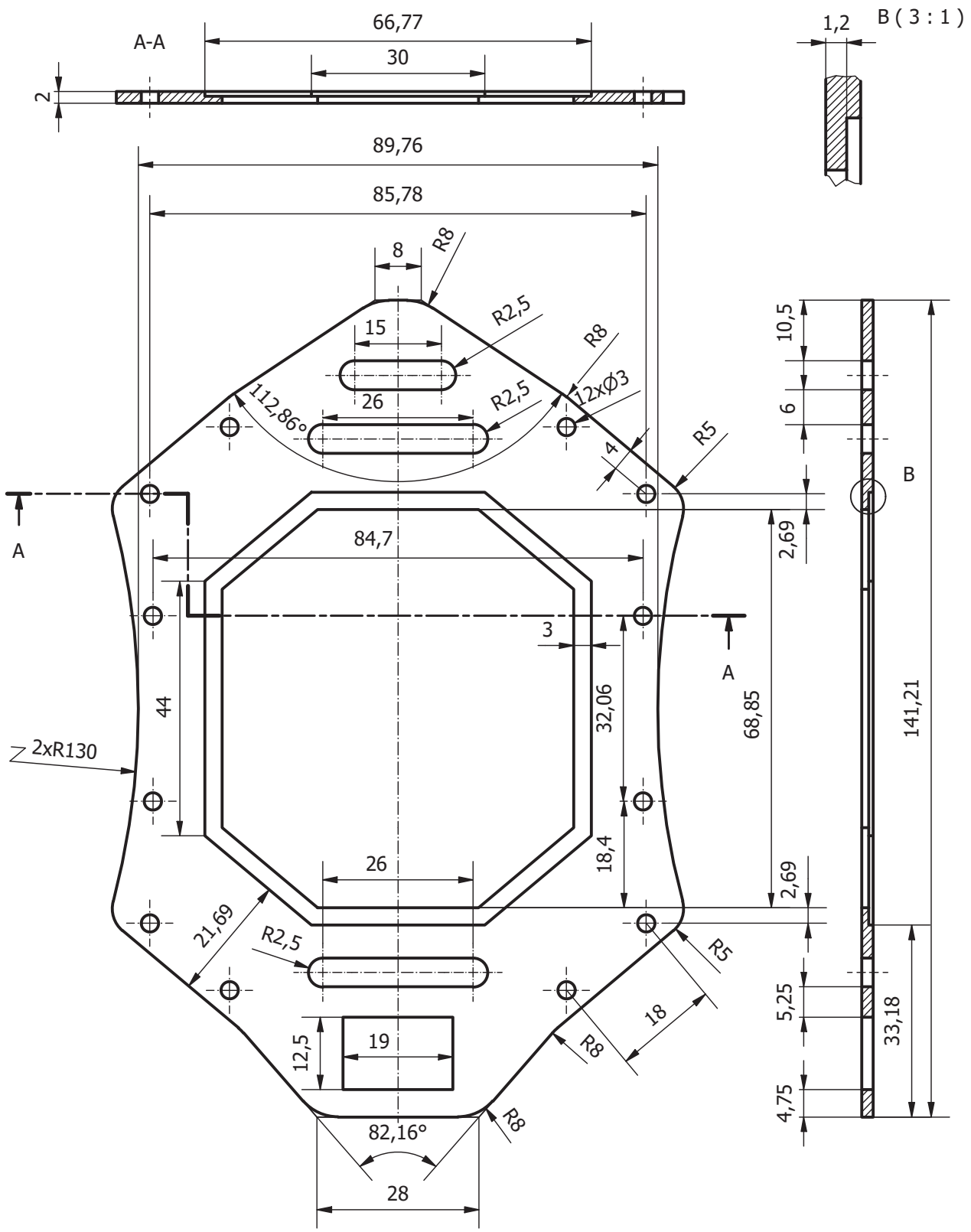






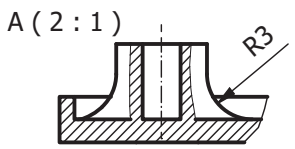
Escala: 1:1

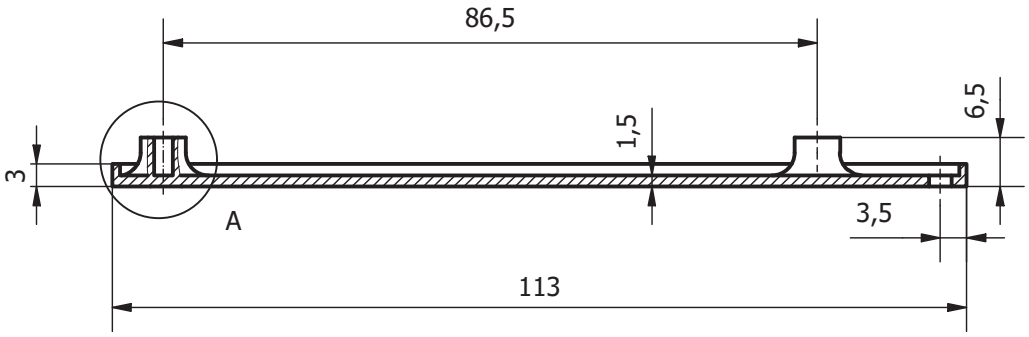
TRABAJO FIN DE GRADO EN INGENIERÍA INDUSTRIAL UNIVERSIDAD POLITECNICA DE VALENCIA · ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA	Proyecto: Diseño y desarrollo de un prototipo físico de dron cuadricóptero
	Fecha de creación: 16/06/19
Creado por: CLAUDIA MARTÍNEZ ABRIL	Plano: Placa inferior Nº plano: 2/15

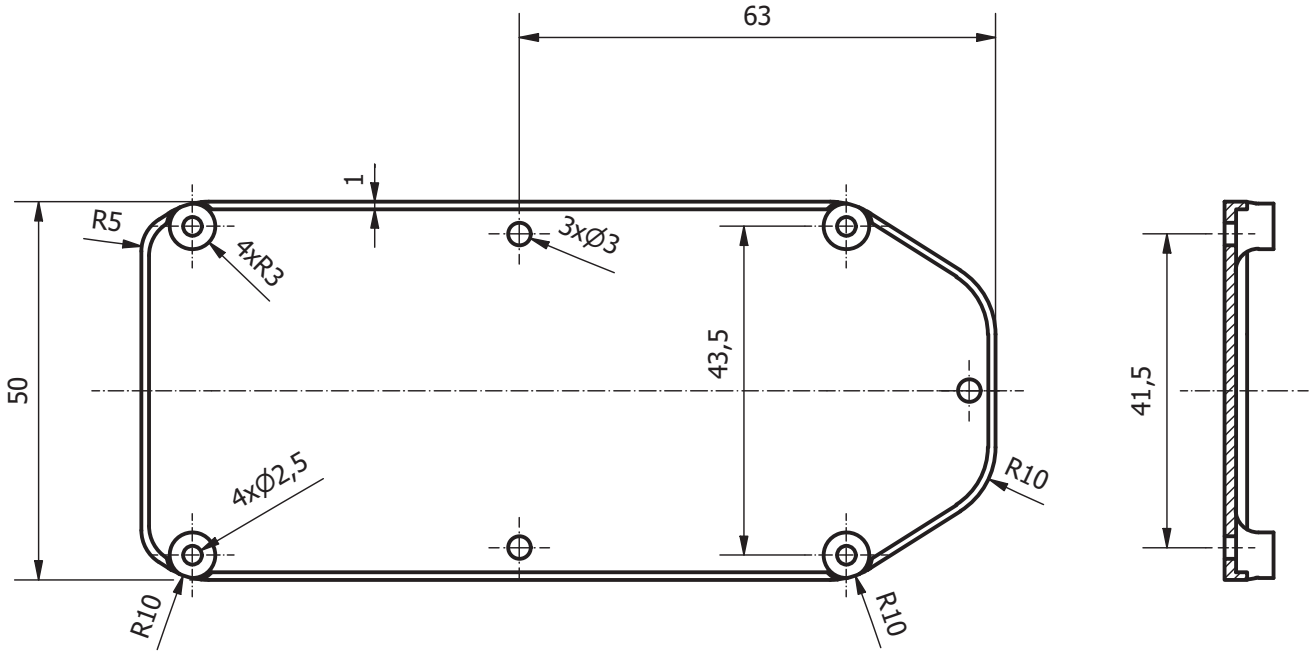


Escala: 1:1

TRABAJO FIN DE GRADO EN INGENIERÍA INDUSTRIAL UNIVERSIDAD POLITECNICA DE VALENCIA ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA	Proyecto: Diseño y desarrollo de un prototipo físico de dron cuadricóptero
	Fecha de creación: 16/06/19
Creado por: CLAUDIA MARTÍNEZ ABRIL	Plano: Placa superior Nº plano: 3/15

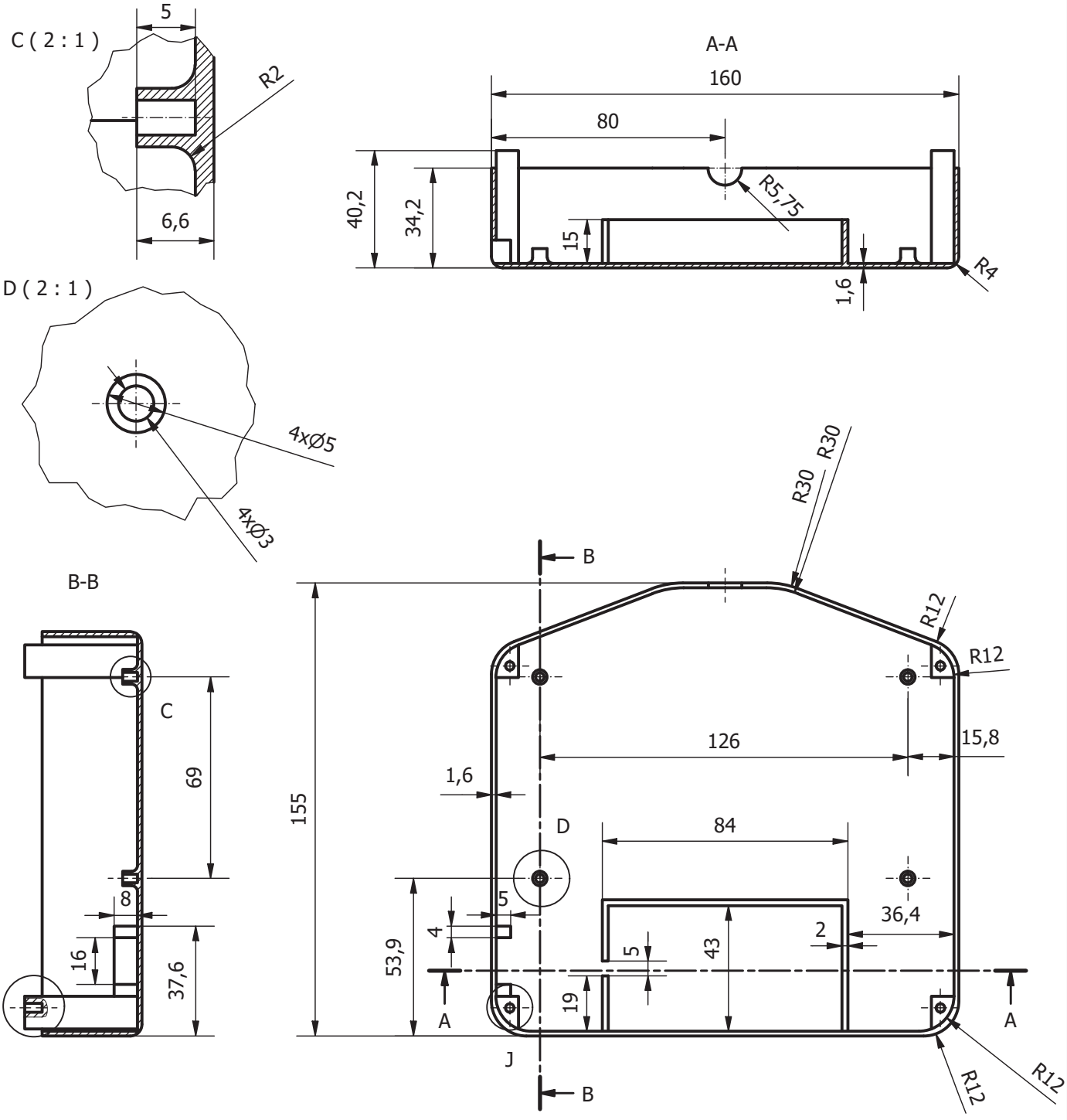






Escala: 1:1

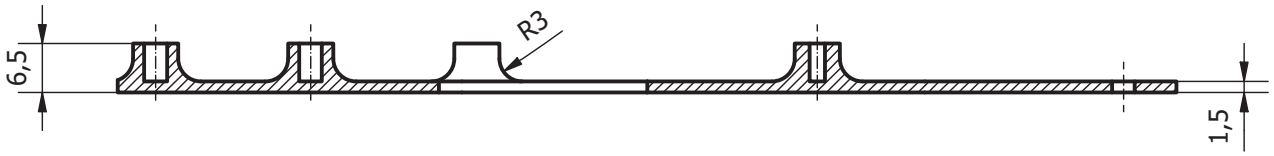
TRABAJO FIN DE GRADO EN INGENIERÍA INDUSTRIAL UNIVERSIDAD POLITECNICA DE VALENCIA · ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA	Proyecto:
	Diseño y desarrollo de un prototipo físico de dron cuadricóptero
	Fecha de creación: 16/06/19
Creado por: CLAUDIA MARTÍNEZ ABRIL	Plano: Soporte pcb Nº plano: 4/15

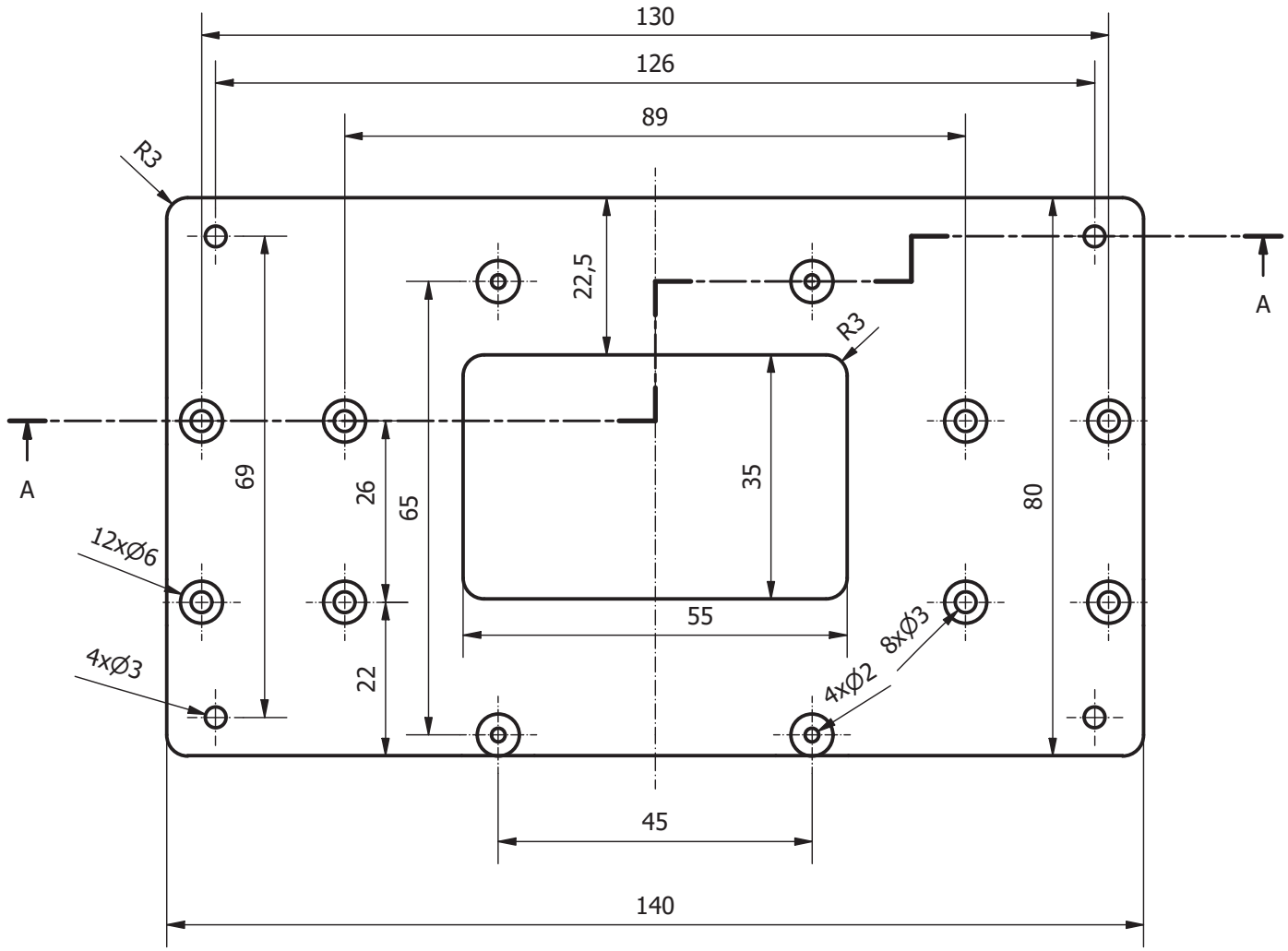


Escala: 1:2

TRABAJO FIN DE GRADO EN INGENIERÍA INDUSTRIAL UNIVERSIDAD POLITECNICA DE VALENCIA · ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA	Proyecto: Diseño y desarrollo de un prototipo físico de dron cuadricóptero
	Fecha de creación: 16/06/19
Creado por: CLAUDIA MARTÍNEZ ABRIL	Plano: Carcasa inferior Nº plano: 5/15

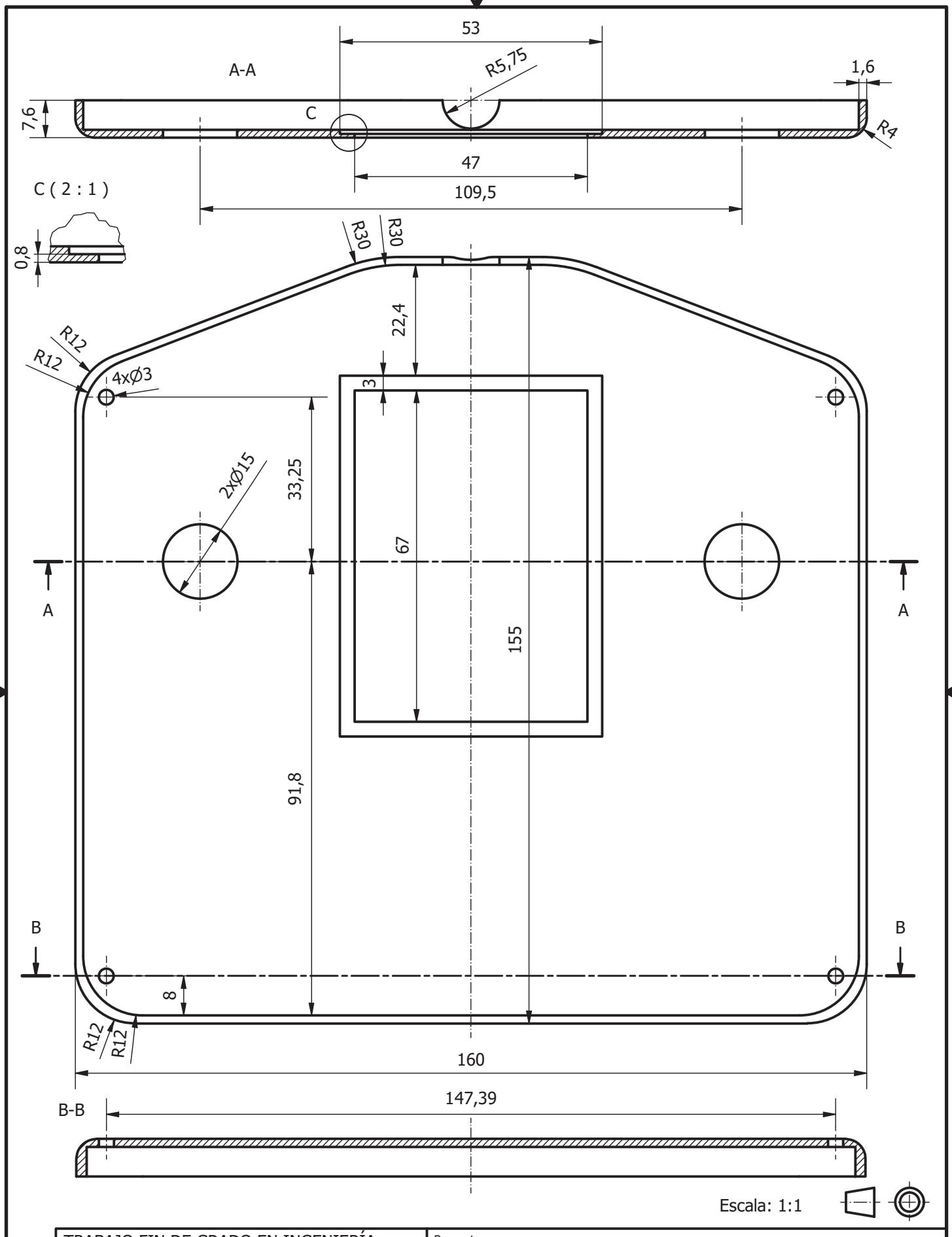
A-A





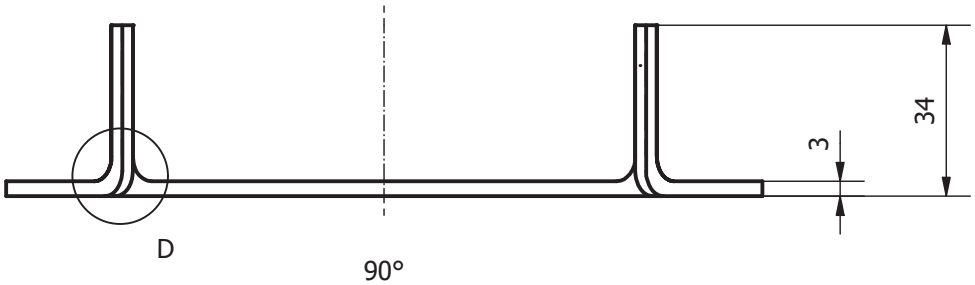
Escala: 1:1

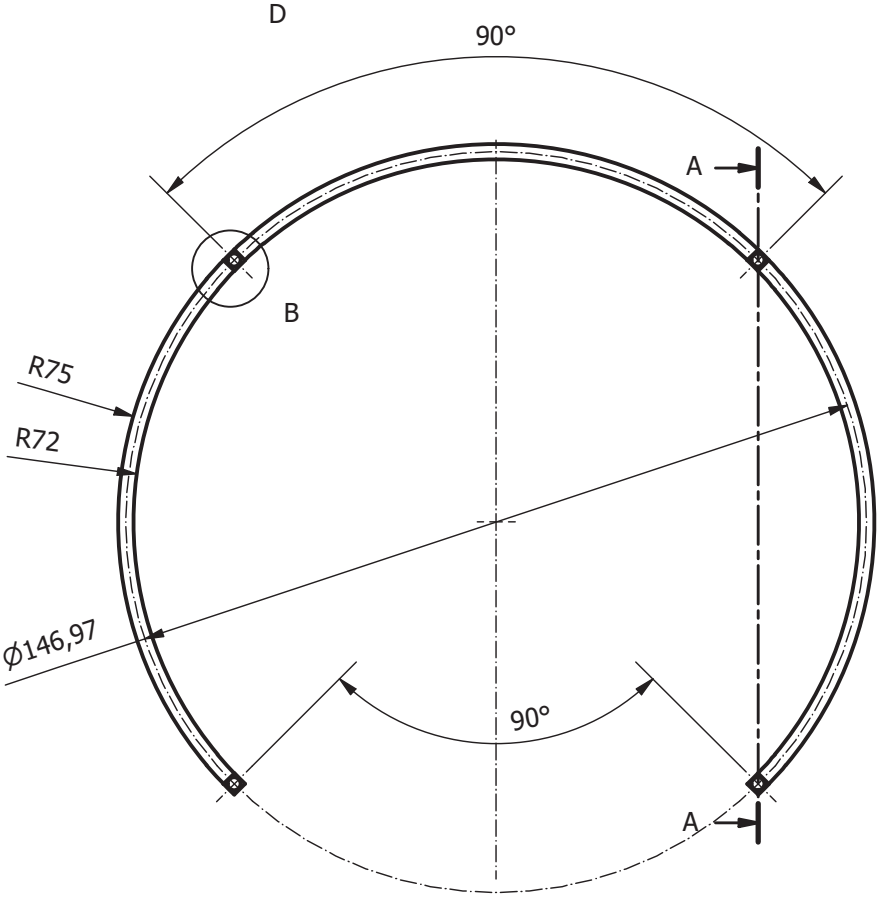
TRABAJO FIN DE GRADO EN INGENIERÍA INDUSTRIAL UNIVERSIDAD POLITECNICA DE VALENCIA · ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA	Proyecto: Diseño y desarrollo de un prototipo físico de dron cuadricóptero
	Fecha de creación: 16/06/19
Creado por: CLAUDIA MARTÍNEZ ABRIL	Plano: Soporte pcb Nº plano: 6/15



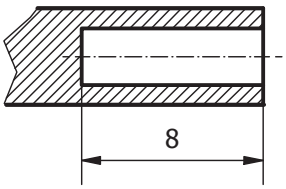
Escala: 1:1

TRABAJO FIN DE GRADO EN INGENIERÍA INDUSTRIAL UNIVERSIDAD POLITECNICA DE VALENCIA ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA	Proyecto: Diseño y desarrollo de un prototipo físico de dron cuadricóptero
	Fecha de creación: 16/06/19
Creado por: CLAUDIA MARTÍNEZ ABRIL	Plano: Carcasa superior Nº plano: 7/15

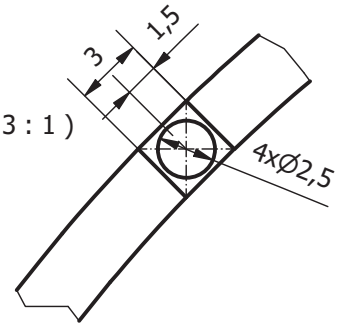




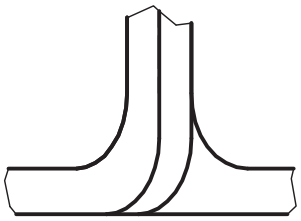
C (3 : 1)



B (3 : 1)

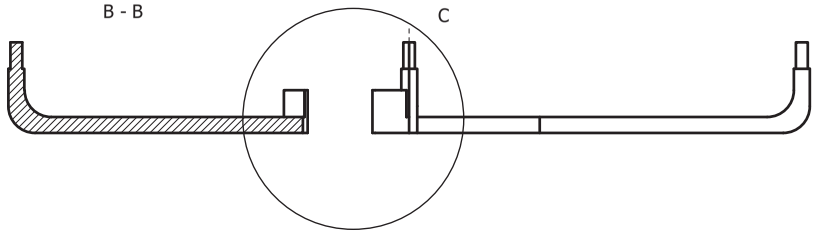


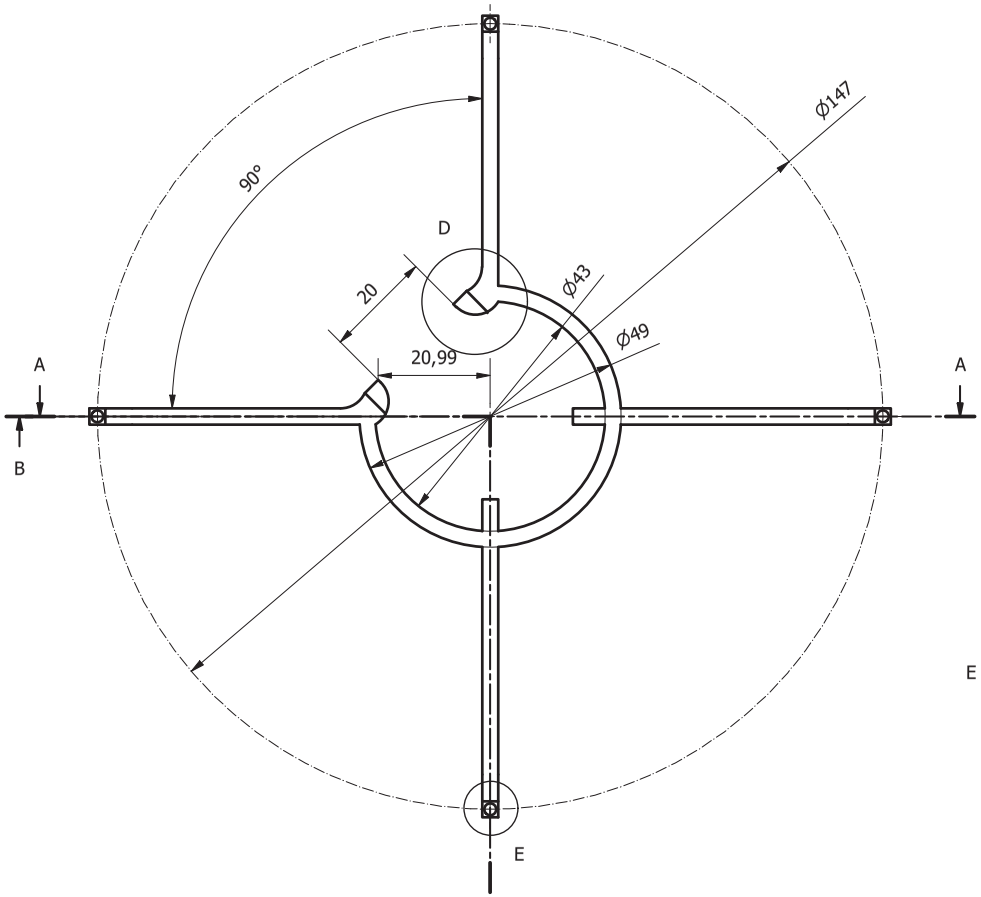
D (2 : 1)

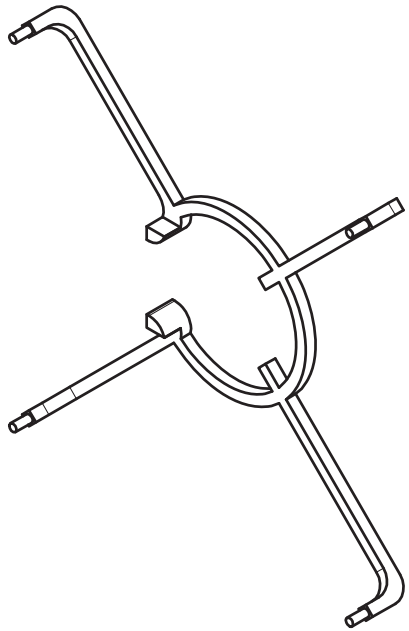


Radios y redondeos no acotados R=5 Escala: 1:2

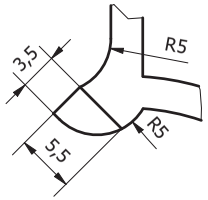
TRABAJO FIN DE GRADO EN INGENIERÍA INDUSTRIAL UNIVERSIDAD POLITECNICA DE VALENCIA · ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA	Proyecto: Diseño y desarrollo de un prototipo físico de dron cuadricóptero
	Fecha de creación: 16/06/19
Creado por: CLAUDIA MARTÍNEZ ABRIL	Plano: Protector superior Nº plano: 8/15



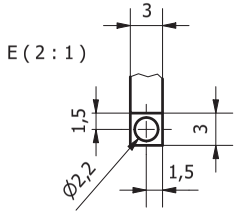




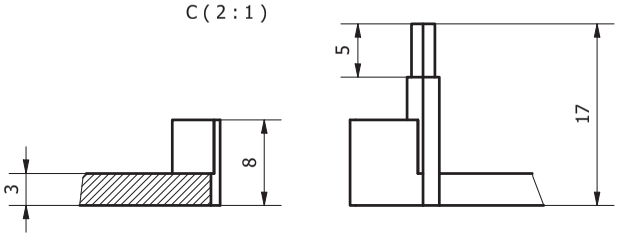
D (2 : 1)



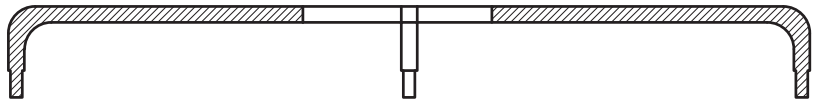
E (2 : 1)



C (2 : 1)

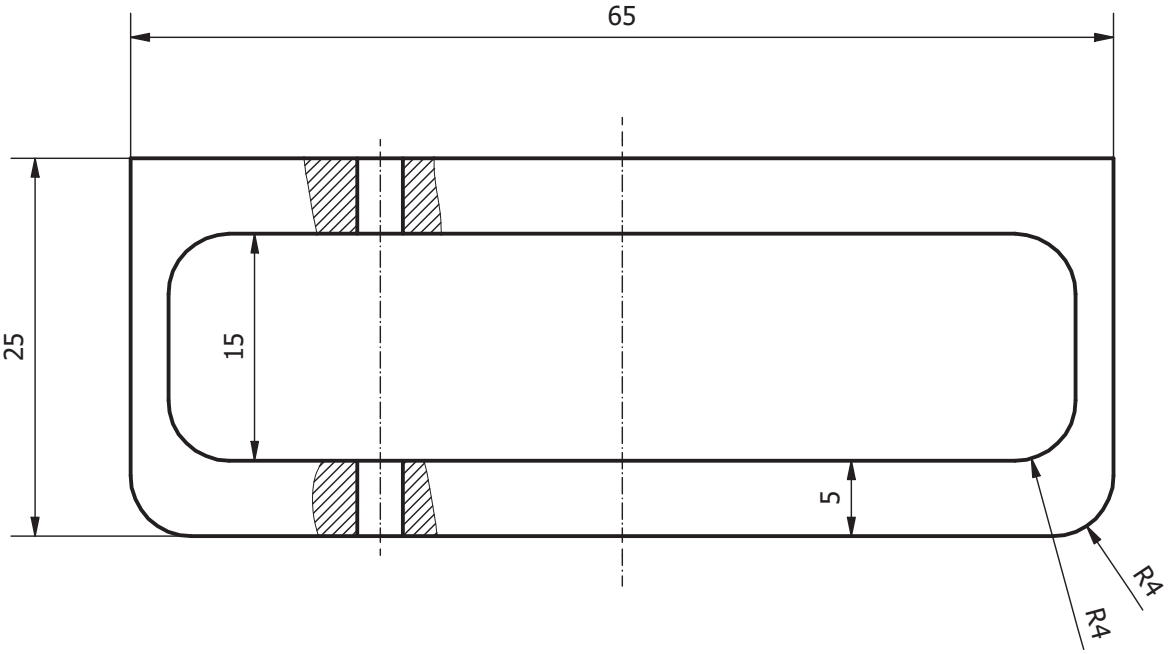


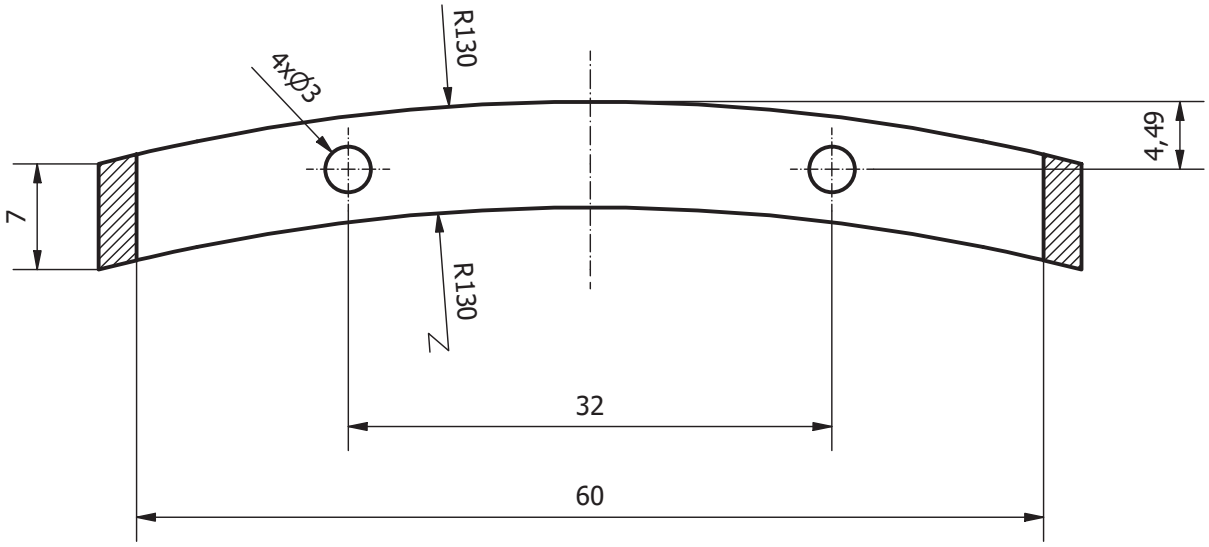
A - A



Escala: 1:1

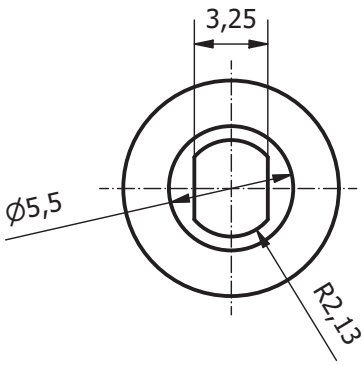
TRABAJO FIN DE GRADO EN INGENIERÍA INDUSTRIAL UNIVERSIDAD POLITECNICA DE VALENCIA · ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA	Proyecto: Diseño y desarrollo de un prototipo físico de dron cuadricóptero
	Fecha de creación: 16/06/19
Creado por: CLAUDIA MARTÍNEZ ABRIL	Plano: Protector inferior Nº plano: 9/15

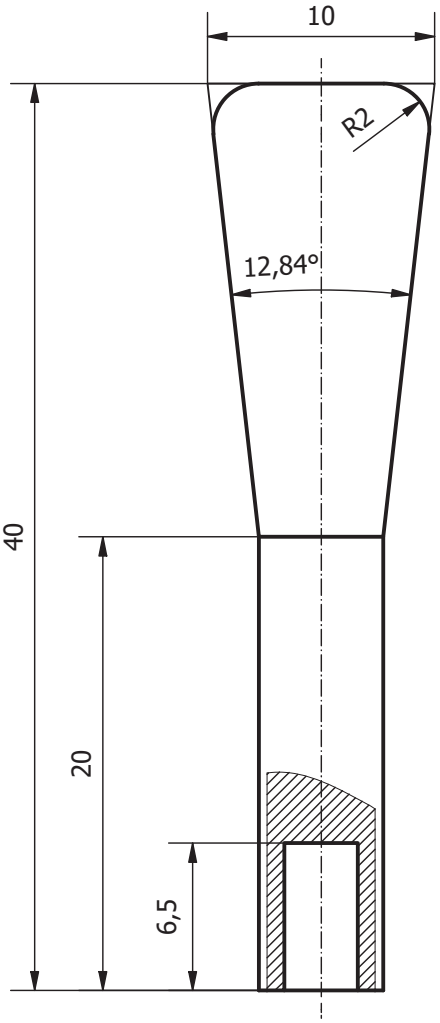




Escala: 2:1

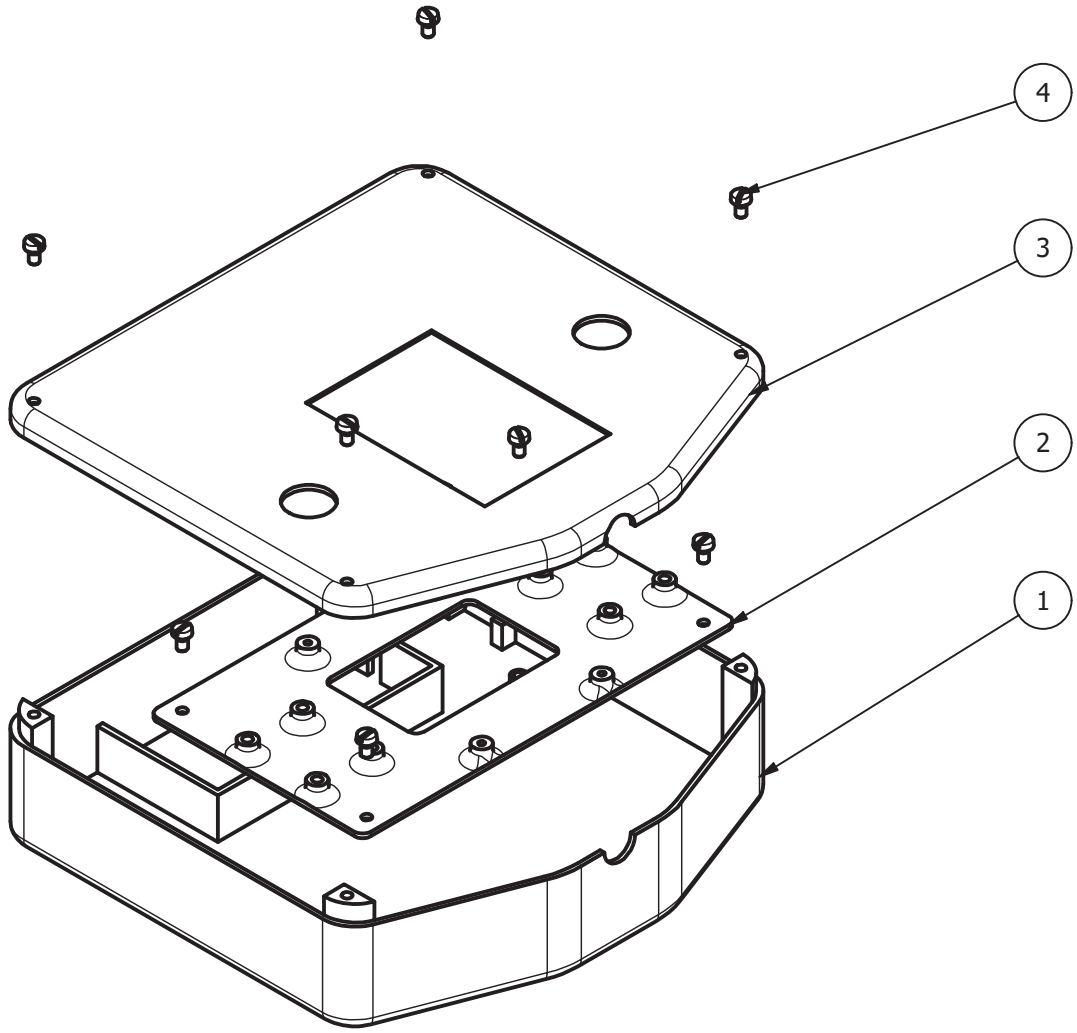
TRABAJO FIN DE GRADO EN INGENIERÍA INDUSTRIAL UNIVERSIDAD POLITECNICA DE VALENCIA ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA	Proyecto: Diseño y desarrollo de un prototipo físico de dron cuadricóptero
	Fecha de creación: 16/06/19
Creado por: CLAUDIA MARTÍNEZ ABRIL	Plano: Soporte Nº plano: 10/15





Escala: 3:1

TRABAJO FIN DE GRADO EN INGENIERÍA INDUSTRIAL UNIVERSIDAD POLITECNICA DE VALENCIA ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA	Proyecto: Diseño y desarrollo de un prototipo físico de dron cuadricóptero
	Fecha de creación: 16/06/19
Creado por: CLAUDIA MARTÍNEZ ABRIL	Plano: Controlador Nº plano: 11/15



Escala: 1:2

LISTA DE PIEZAS			
ELEMENTO	CTDAD	Nº DE PIEZA	DESCRIPCIÓN
1	1	carcasa inferior	
3	1	soporte pcb	
2	1	carcasa superior	
4	8	CNS 4355 - M 3 x 5	Tornillos de cabeza cilíndrica ranurada

TRABAJO FIN DE GRADO EN INGENIERÍA INDUSTRIAL

UNIVERSIDAD POLITECNICA DE VALENCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

Proyecto:

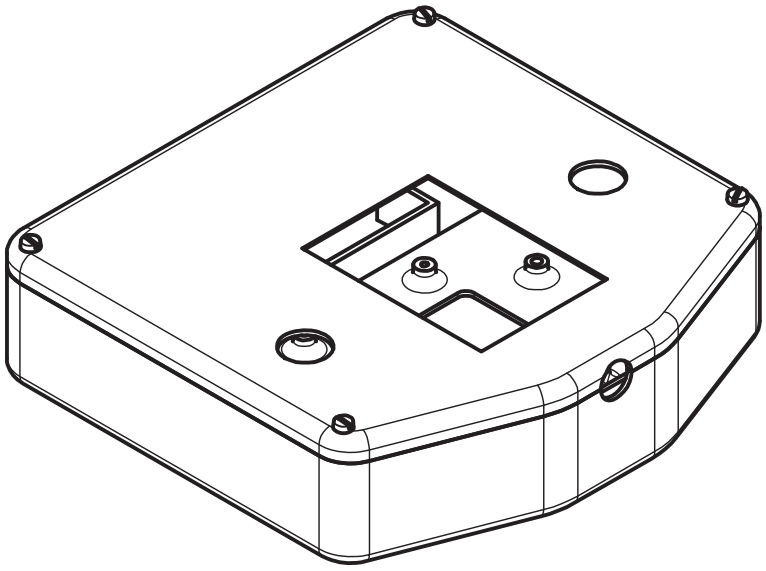
Diseño y desarrollo de un prototipo físico de dron cuadricóptero

Fecha de creación:
16/06/19

Creado por: CLAUDIA MARTÍNEZ ABRIL

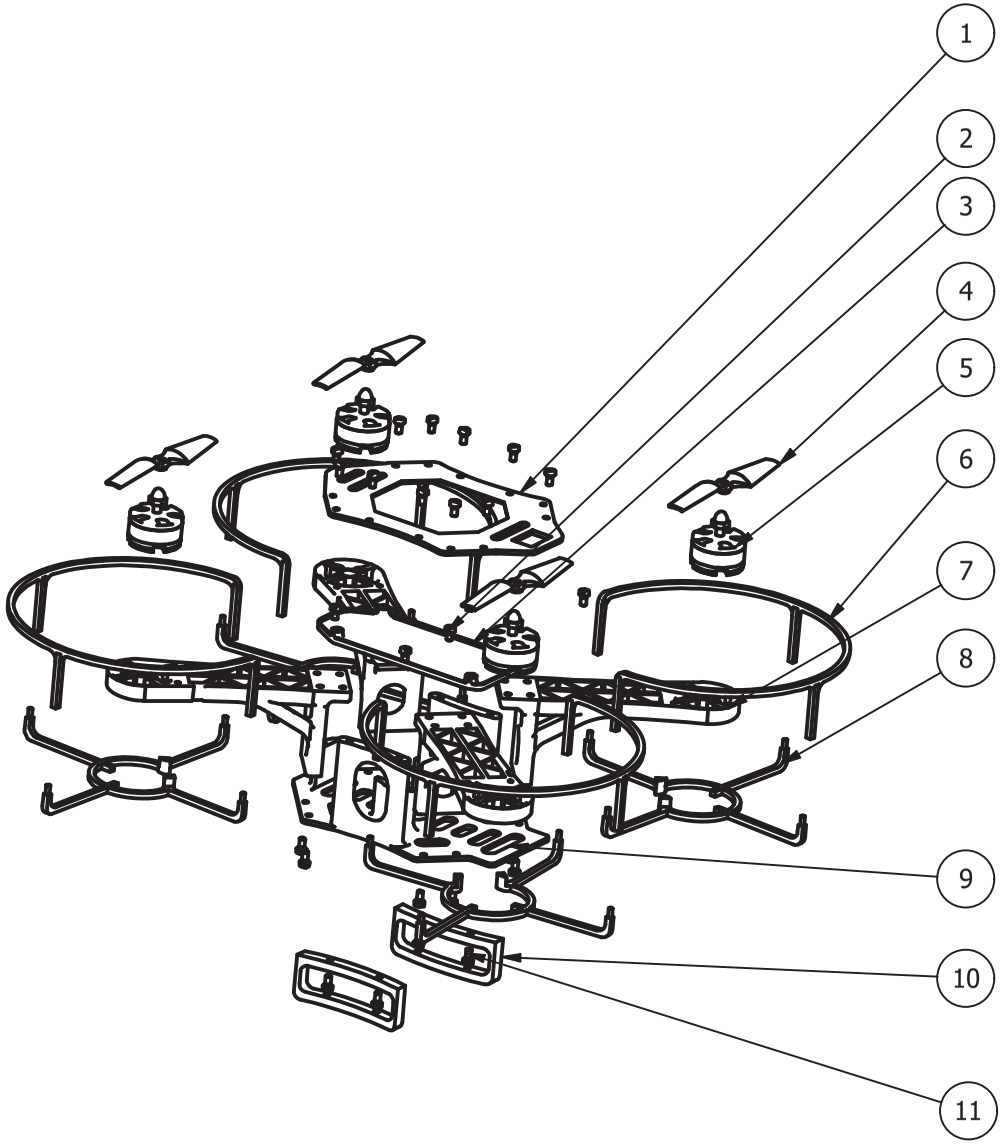
Plano:
Plano de despiece del mando

Nº plano:
12/15



Escala: 1:2

TRABAJO FIN DE GRADO EN INGENIERÍA INDUSTRIAL UNIVERSIDAD POLITECNICA DE VALENCIA ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA	Proyecto: Diseño y desarrollo de un prototipo físico de dron cuadricóptero
	Fecha de creación: 16/06/19
Creado por: CLAUDIA MARTÍNEZ ABRIL	Plano: Plano de conjunto del mando Nº plano: 13/15



Escala: 1:4

LISTA DE PIEZAS			
ELEMENTO	CTDAD	Nº DE PIEZA	DESCRIPCIÓN
1	1	Placa base superior	
2	3	CNS 4355 - M 3 x 5	Tornillos de cabeza cilíndrica ranurada
3	1	Placa soporte pcb conntrolador de vuelo y receptor	
4	4	Hélice	
5	4	Motor	
6	4	Protector hélices superior	
7	4	Brazo	
8	4	Protector hélices inferior	
9	1	Placa base inferior	
10	2	Soporte	
11	24	CNS 4355 - M 3 x 6	Tornillos de cabeza cilíndrica ranurada

TRABAJO FIN DE GRADO EN INGENIERÍA INDUSTRIAL

UNIVERSIDAD POLITECNICA DE VALENCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

Creado por: CLAUDIA MARTÍNEZ ABRIL

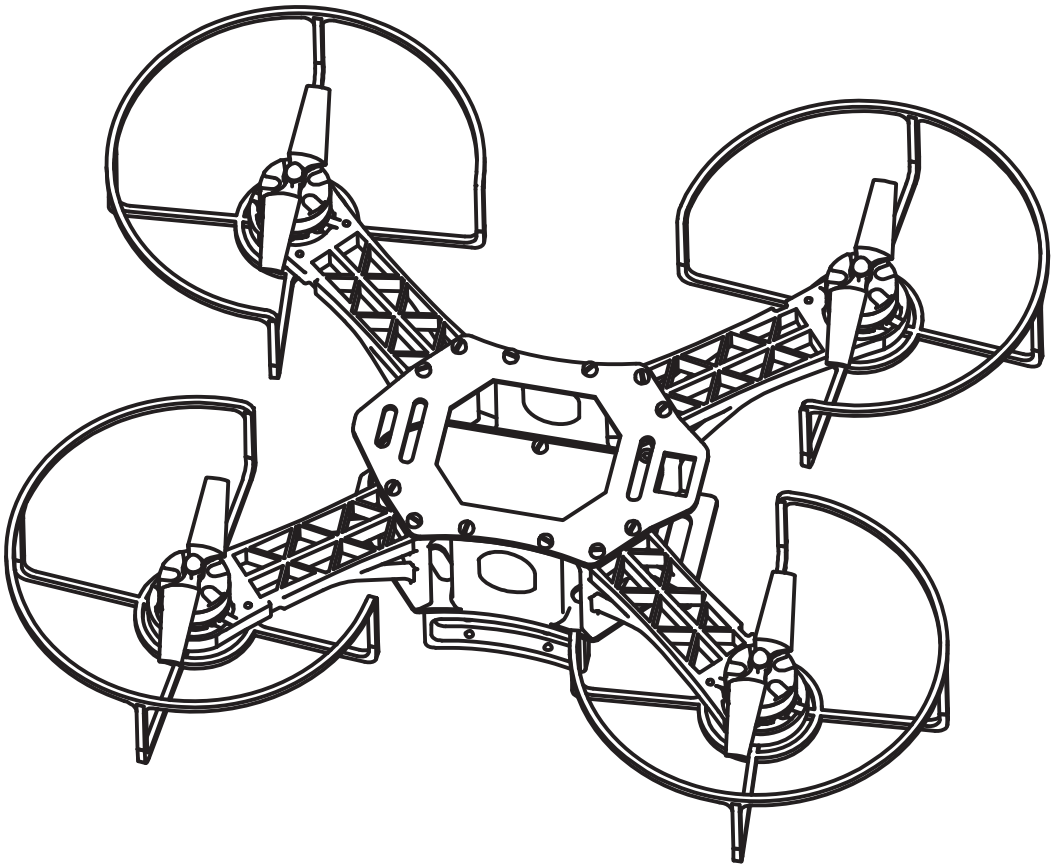
Proyecto:

Diseño y desarrollo de un prototipo físico de dron cuadricóptero

Fecha de creación:
16/06/19

Plano:
Plano de despiece del dron

Nº plano:
14/15



Escala: 1:3

TRABAJO FIN DE GRADO EN INGENIERÍA INDUSTRIAL UNIVERSIDAD POLITECNICA DE VALENCIA ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA	Proyecto: Diseño y desarrollo de un prototipo físico de dron cuadricóptero
	Fecha de creación: 16/06/19
Creado por: CLAUDIA MARTÍNEZ ABRIL	Plano: Plano de conjunto del dron Nº plano: 15/15

ANEXO: PROGRAMACIÓN

ÍNDICE PROGRAMACIÓN

ÍNDICE DE FIGURAS DEL ANEXO

Capítulo 1- Librerías y funciones utilizadas.

1.1- Wire

Para hacer posible la comunicación i2C, el bus i2C requiere únicamente dos cables, uno para la señal de reloj (CLK) y otro para el envío de datos (SDA). Este bus tiene una arquitectura de tipo maestro-esclavo. El dispositivo maestro inicia la comunicación con los esclavos y puede mandar o recibir datos de los esclavos. Los esclavos no pueden iniciar la comunicación (el maestro tiene que preguntarles).

Para usar el bus i2C en Arduino, el IDE Standard proporciona la librería <Wire.h>, que contiene las funciones necesarias para controlar el hardware integrado.

La funciones de esta librería que se van a emplear son:

- begin(): inicia la biblioteca Wire y se une al bus i2C como maestro o esclavo. Si no se pasa nada como parámetro, se une al bus como maestro.

- beginTransmission(): inicia una transmisión i2C al dispositivo esclavo con la dirección pasada.

- write(): escribe datos en un dispositivo esclavo en respuesta a una solicitud de un maestro. Se debe pasar como parámetro un valor o una cadena de valores. write() devolverá el número de bytes escritos, pero la lectura de ese número es opcional.

- endTransmission(): finaliza la transmisión a un dispositivo esclavo que comenzó con beginTransmission().

- requestFrom(): utilizado por el maestro para solicitar bytes a un dispositivo esclavo. Los bytes se pueden recuperar con las funciones available() y read().

- read(): lee un byte que se transmitió desde un dispositivo esclavo a uno maestro después de una llamada a requestFrom() o se transmitió de un maestro a un esclavo.

1.2- Serial

Se utiliza para la comunicación entre la placa Arduino y un PC u otros dispositivos. Todas las placas Arduino tienen al menos un puerto serie.

Las funciones de esta librería que se van a emplear son:

- begin(): establece la velocidad de datos en bits por segundo (baudios) para la transmisión de datos en serie.

- print(): imprime datos en el puerto serie como texto ASCII legible. Los números se imprimen utilizando un carácter ASCII para cada dígito. Los números en coma flotante se imprimen de forma similar como dígitos ASCII, con dos decimales por defecto. Los bytes se envían como un solo carácter. Los caracteres y las cadenas se envían tal como están. La función devuelve el número de bytes escritos, pero la lectura de este número es opcional.

- printIn(): imprime los datos en el puerto serie como texto ASCII legible para las personas, seguido de un carácter de retorno de carro (ASCII 13, o '\ r') y un carácter de nueva línea (ASCII 10, o '\ n').

1.1- Servo

Esta biblioteca permite que una placa Arduino controle servomotores RC. La biblioteca Servo admite hasta 12 motores en la mayoría de las placas Arduino.

Las funciones de esta librería que se van a emplear son:

- attach(): adjunta la librería Servo a un pin. Su sintaxis puede ser de dos formas, *servo.attach(*pin*)* o *servo.attach(*pin, min, max*)*.
  'servo' es una variable de tipo Servo, 'pin' es el número del pin al que está unido el servo, 'min' es el ancho de pulso, en microsegundos, correspondiente al ángulo mínimo en el servo (0 grados) y 'max' el ancho de pulso, en microsegundos, correspondiente al ángulo máximo en el servo (180 grados).

- writeMicroseconds(): escribe un valor en microsegundos al servo, controlando el eje en consecuencia. En los servos estándar, con un valor de 1000 el motor se encontraría parado, con 2000 a su máxima velocidad y consecuentemente 1500 a mitad camino.

1.2- Time

'Time' es una biblioteca que proporciona la funcionalidad de cronometraje para Arduino.

Las funciones de esta librería que se van a emplear son:

- delay(): pausa el programa la cantidad de tiempo (en milisegundos) especificada como parámetro.

- millis(): devuelve el número de milisegundos transcurridos desde que la placa Arduino comenzó a ejecutar el programa actual. Este número se desbordará (devolverá cero) después de aproximadamente 50 días.

- micros(): devuelve el número de microsegundos transcurridos desde que la placa Arduino comenzó a ejecutar el programa actual. Este número se desbordará (devolverá cero) después de aproximadamente 70 minutos.

1.3- Math

La biblioteca incluye una gran cantidad de funciones matemáticas útiles para manipular números de coma flotante.

Las funciones de esta librería que se van a emplear en los códigos son:

- map(value, fromLow, fromHigh, toLow, toHigh): re-mapea un número de un rango a otro, es decir, el valor **fromLow** se asignaría a **toLow**, el valor **fromHigh** a **toHigh** y los valores intermedios del primer rango a valores intermedios del segundo. La función usa matemática de enteros, por lo que no generará fracciones. Los residuos fraccionarios se

truncan y no se redondean ni promedian. Su sintaxis es *'map(*value, fromLow, fromHigh, toLow, toHigh*)',* siendo 'value' el número a mapear, 'fromLow' el límite inferior del rango actual del valor, 'fromHigh' el límite superior del rango actual del valor, 'toLow' el límite inferior del rango objetivo del valor y 'toHigh' el límite superior del rango objetivo del valor. La función devuelve el valor mapeado.

- atan(x): calcula el arcotangente (en radianes) de un número siendo *x* dicho número.

- sqrt(x): calcula la raíz cuadrada de un número. *x* es dicho número.

- pow(base, exponent): calcula el valor de un número elevado a una potencia. *base* es el número y *exponent* es la potencia a la que se eleva la base.

- constrain(x, a, b): restringe un número dentro de un rango. 'x' es el número a restringir, 'a' el extremo inferior del rango y 'b' el extremo superior. La función devuelve 'x' si el valor está entre 'a' y 'b', 'a' si el valor es menor que 'a' y 'b' si el valor es mayor que 'b'.

1.4- SPI

El bus de interfaz de periféricos serie o bus SPI es un estándar de comunicaciones para controlar casi cualquier dispositivo electrónico digital que acepte un flujo de bits serie regulado por un reloj (comunicación sincrónica). Incluye una línea de reloj, dato entrante, dato saliente y un pin de *chip select*, que conecta o desconecta la operación del dispositivo con el que uno desea comunicarse.

La comunicación de datos entre maestro y esclavo se realiza en dos líneas independientes, una del maestro a los esclavos y otra de los esclavos al maestro, por tanto, el maestro puede enviar y recibir datos simultáneamente.

El bus SPI requiere un mínimo de 3 líneas:

- MOSI (Master Out Slave In): para la comunicación del maestro al esclavo.
- MISO (Master In Slave Out): para la comunicación del esclavo al maestro.
- SCK (Clock): señal de reloj enviada por el maestro.

Además, se requiere una línea adicional SS (Slave Select) para cada dispositivo esclavo conectado, para seleccionar el dispositivo con el que se va a realizar la comunicación. Si el pin asociado a esta línea se encuentra a nivel bajo, se produce la comunicación con el maestro y cuando se encuentra a nivel alto, lo ignora.

Para Arduino Nano, los pines SPI son los siguientes:

SS	**MOSI**	**MISO**	**SCK**
D10	D11	D12	D13

Para usar el puerto SPI en Arduino el IDE Standard proporciona la librería <<SPI.h>> que contiene las funciones necesarias para controlar el hardware integrado de SPI. No obstante, al ser trama de datos específica de cada dispositivo, lo más frecuente es que no se utilicen directamente estas funciones y que el uso del bus SPI se realice de forma indirecta a través de la librería del componente.

1.5- RF24

Para realizar el control del módulo NRF24L01 (transmisor y receptor) se empleará la librería RF24. Se creará una instancia de la radio, indicándole los pines de control y el tubo de escritura a utilizar, que es uno de los diferentes canales en lo que la radio puede operar.

RF24 (uint8_t_cepin, uint8_t_cspin) crea una nueva instancia (objeto) de este dispositivo. La instancia se crea con los pines de comunicación SPI pero es necesario especificar los pines de control que están conectados al módulo.

_cepin: pin del Arduino conectado al pin Chip Enable (CE) del módulo.

_cspin: pin del Arduino conectado al pin Chip Select (CS) del móudlo.

Las funciones que se van a utilizar son las siguientes:

- begin (): inicializa el objeto creado.

- setAutoAck (bool): habilita o deshabilita paquetes de auto-reconocimiento. Habilita si el booleano pasado es *true* y deshabilita si es *false*.

- setDataRate (speed): ajusta la velocidad de transmisión de datos. Reducir la velocidad de datos puede mejorar significativamente el rango que puede alcanzar. A una velocidad de 2MBps, la sensibilidad del receptor cae a -82 dBm (decibelios metro) y si se baja a 250KBps es casi 10 veces más sensible que con la velocidad anterior, con lo cual puede decodificar una señal que es 10 veces más débil. La velocidad de 250 Kbps es más que suficiente (el módulo ofrece su mayor sensibilidad).

- setPALevel (level): establece el nivel del amplificador de potencia (PA) en uno de los siguientes niveles: MIN (-18dBm), LOW (-12dBm), MED (-6dBm) o HIGH (0dBm).

- openWritingPipe (address): abre un canal para escribir. Las direcciones son valores hexadecimales de 40 bits.

- write(const void * buf, uint8_t len): escribe en el canal de escritura abierto. Se debe llamar a openWritingPipe() anteriormente para establecer el destino de dónde escribir. *buf* es el puntero a los datos a enviar y *len* el número de bytes a enviar.

- openReadingPipe(uint8_t número, uint64_t dirección): abre un tubo de lectura. Se puede abrir hasta 6 canales de lectura a la vez. Posteriormente, se debe llamar a la función *startListening(). número* es la tubería a abrir y *dirección* es la dirección de 40 bits de la tubería a abrir.

- startListening (): empieza a escuchar en los canales abiertos de lectura.

- stopListening (): finaliza la escucha en los canales abiertos de lectura.

- read(void* buf, uint8_t len): lee la carga útil y devuelve la última carga útil recibida. *buf* es el puntero a un búfer donde se deben escribir los datos leídos y *len* es el número máximo de bytes para leer en el búfer.

- available(): prueba si hay bytes disponibles para leer. Devuelve verdadero si hay carga útil disponible, falso si no la hay.

1.6- Analog I/O

- analogRead(pin): lee el valor del *pin* analógico especificado. Devuelve la lectura analógica en el pin.

Capítulo 2- Código de calibración de los ESCs.

http://ardupilot.org/copter/docs/esc-calibration.html

Para la calibración de los ESCs se emplea el código adjuntado y se introduce, en la constante "MOTOR_PIN", el pin digital al que esté conectado el cable blanco del BEC (el de señal) del ESC que queremos calibrar.

Ejecutando el código, se envía la velocidad máxima (2000) y mínima (1000) al motor. Los ESCs deben emitir unos sonidos (tantos pitidos como celdas contenga la batería que se esté empleando) para comunicar que los valores han sido capturados y que la calibración se ha completado. A continuación, si se introduce por teclado valores entre 1000 y 2000, los motores comenzarán a girar (si se envía 1000 el motor se detendrá y a 2000 girará a máxima velocidad).

Se debe realizar estos pasos para cada uno de los cuatro ESCs y quedarán configurados para el rango deseado.

```
#include <Servo.h>

#define MAX_VALOR 2000
#define MIN_VALOR 1000
#define MOTOR_PIN 9
int DELAY = 1000;

Servo motor;

void setup() {
  Serial.begin(9600);
  delay(1500);
  Serial.println("Empezando el programa...");
  delay(1000);
  Serial.println("Este programa va a calibrar los ESC");

  motor.attach(MOTOR_PIN);
```

```
  Serial.print("Valor máximo del rango: ");
  Serial.print(MAX_VALOR);
  Serial.print("\n");
  Serial.println("Enciende la batería, espera los sonidos de los ESC ");
  Serial.println("y pulsa cualquier tecla");
  motor.writeMicroseconds(MAX_VALOR);

  while (!Serial.available());
  Serial.read();

  Serial.println("\n");
  Serial.println("\n");
  Serial.print("Enviando el valor mínimo: ");
  Serial.print(MIN_VALOR);
  Serial.print("\n");
  motor.writeMicroseconds(MIN_VALOR);


  Serial.println("ESCs calibrados");
  Serial.print("\n");
  Serial.println("Ahora, envía valores entre 1000 y 2000");
  Serial.println("y los motores comenzarán a rotar.");
  Serial.print("\n");
  Serial.println("1000 parará el motor y 2000 será la máxima velocidad");

}

void loop() {

  if (Serial.available() > 0)
  {
    int DELAY = Serial.parseInt();
    if (DELAY > 999)
    {

      motor.writeMicroseconds(DELAY);
      float VEL = (DELAY-1000)/10;
      Serial.print("\n");
      Serial.println("VELOCIDAD DEL MOTOR:");
      Serial.print("  ");
      Serial.print(VEL);
      Serial.print("%");
    }
  }
}
```

Capítulo 3- Código transmisor

Se importan las bibliotecas necesarias. A continuación, se crea el canal para el transmisor de radio, que deberá ser el mismo que el utilizado por el receptor posteriormente, para que la transmisión de datos pueda tener lugar. También se definen los pines de selección y activación de chip para la comunicación SPI, que son los pines CE y CSN del módulo, conectados a los pines D9 y D10 del Arduino.

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <RF24_config.h>

RF24 radio(9, 10);
const byte direccion [6] = "00001";
```

Figura 1. Parte 1 del código del transmisor.

A continuación, se crea una variable de tipo “struct” para almacenar los valores provenientes de los potenciómetros de los joysticks, que serán enviados posteriormente al receptor. Se crea una variable con esta estructura y este será el paquete de datos que será enviado. Con la función resetDatos() se inician los valores de cada canal a 127, valor medio de 8 bits (256).

```
struct Datos {
  byte throttle;
  byte yaw;
  byte pitch;
  byte roll;
};

Datos data;

void resetDatos()
{
  data.throttle = 127;
  data.yaw = 127;
  data.pitch = 127;
  data.roll = 127;
}
```

Figura 2. Parte 2 del código del transmisor.

A continuación, en la función setup() se inicia la transmisión de radio:

```
void setup()
{
  radio.begin();
  radio.setAutoAck(false);
  radio.setDataRate(RF24_250KBPS);
  radio.openWritingPipe(direccion);
  radio.setPALevel(RF24_PA_MIN);

  radio.stopListening();
  resetData();


}
```

Figura 3. Parte 3 del código del transmisor.

Por último, en el bucle loop() son leídos cada uno de los cuatro valores analógicos de cada potenciómetro y pasados a un rango de entre 0 y 255, ya que solo es posible enviar 8 bits. Se guardan estos valores en la estructura creada anteriormente "data" y se envía por el canal de escritura utilizando la función "radio.write".

```
void loop()
{
  data.throttle = map( analogRead(A1), 0, 1023, 0, 255);
  data.yaw      = map( analogRead(A0), 0, 1023, 0, 255);
  data.pitch    = map( analogRead(A2), 0, 1023, 0, 255);
  data.roll     = map( analogRead(A3), 0, 1023, 0, 255);

  radio.write(&data, sizeof(Datos));
}
```

Figura 4. Parte 4 del código del transmisor.

Capítulo 4- Código receptor + Controlador de vuelo

Se importan las bibliotecas necesarias incluyendo esta vez la biblioteca "Servo.h" para controlar los motores y por lo tanto, se crean 4 variables de tipo "Servo" (L_F_motor, L_B_motor, R_F_motor y R_B_motor).

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24_config.h>
#include <RF24.h>
#include <Servo.h>
#include <Wire.h>

Servo L_F_motor;
Servo L_B_motor;
Servo R_F_motor;
Servo R_B_motor;
```

Figura 5. Inicio código Receptor + Controlador de vuelo.

4.1- Receptor

Este código recibe 4 canales provenientes del transmisor (throttle, yaw, pitch y roll), por lo tanto, se crea una estructura para almacenarlos. Es necesario emplear el canal empleado anteriormente para el transmisor de radio, para poder recibir los datos. También se vuelven a definir los pines de selección y activación de chip para la comunicación SPI.

```
int input_THROTTLE = 0;
int input_YAW = 0;
int input_PITCH = 0;
int input_ROLL = 0;

struct Datos {
byte throttle;
byte yaw;
byte pitch;
byte roll;
};

Datos data;

const byte direccion [6] = "00001";
RF24 radio(9, 10);
```

Figura 6. Parte 1 del código del receptor.

Seguidamente, se define el valor inicial de cada dato de llegada en la función resetDatos(). Los potenciómetros estarán a mitad posición, por lo tanto 127 es el valor medio de 8 bits.

```
void resetDatos()
{
data.throttle = 127;
data.yaw = 127;
data.pitch = 127;
data.roll = 127;
}
```

Figura 7. Parte 2 del código del receptor.

A continuación, en la función setup(), el módulo de radio NRF24 es configurado y la comunicación comienza.

```
void setup()
{
  resetDatos();
  radio.begin();
  radio.setAutoAck(false);
  radio.setDataRate(RF24_250KBPS);
  radio.openReadingPipe(1,direccion);

  radio.startListening();
```

Figura 8. Parte 3 del código del receptor.

En la siguiente parte del código, se leen los datos recibidos del transmisor con la función "recibirDatos ()" y éstos son guardados en la estructura.

```
void recibirDatos()
{
while ( radio.available() ) {
radio.read(&data, sizeof(Datos));
lastRecvTime = millis();
}
}
```

Figura 9. Parte 4 del código del receptor.

Por último, en la función loop(), la cual se va a estar ejecutando continuamente, se llama a la función "recibirDatos ()" para leer los datos recibidos desde el transmisor y si se pierde la conexión, ejecuta la función resetDatos() e inicia los valores. Son recibidos 8 bits por cada canal, lo que se traduce en un valor decimal máximo de 255. Se deben pasar estos valores iniciales de 0 a 255 a un rango de entre 1000 y 2000 microsegundos ya que es el rango con el que trabajan los motores.

```
void loop()
{
recibirDatos();
unsigned long now = millis();
//Comprobamos si se ha perdido la señal
if ( now - lastRecvTime > 1000 ) {
//Si se ha perdido la señal
resetData();
}
//recibimos valores del transmisor
input_THROTTLE = map(data.throttle, 0, 255, 1000, 2000);
input_YAW = map(data.yaw,      0, 255, 1000, 2000);
input_PITCH = map(data.pitch,    0, 255, 1000, 2000);
input_ROLL = map(data.roll,     0, 255, 1000, 2000);
```

Figura 10. Parte 5 del código del receptor.

4.2.- Lectura IMU

El código se divide en tres partes: lectura del giroscopio, lectura del acelerómetro y lectura del magnetómetro.

Para hacer posible la comunicación i2C entre la IMU y el Arduino, se descarga la librería <Wire.h>. Se comienza por la lectura del giroscopio.

En el ciclo de configuración, empieza la comunicación con la IMU y su dirección de esclavo es seleccionada. Revisando la hoja de datos del módulo MPU9250 se observa que la dirección de esclavo del dispositivo es ‘0x68’.

If the FIFO buffer is empty, reading this register will return the last byte that was previously read from the FIFO until new data is available. The user should check *FIFO_COUNT* to ensure that the FIFO buffer is not read when empty.

4.38 Register 117 – Who Am I

Name: WHOAMI

Serial IF: Read Only

Reset value: 0x68

BIT	NAME	FUNCTION
[7:0]	WHOAMI	Register to indicate to user which device is being accessed.

This register is used to verify the identity of the device. The contents of *WHO_AM_I* is an 8-bit device ID. The default value of the register is 0x71.

4.39 Registers 119, 120, 122, 123, 125, 126 Accelerometer Offset Registers

For MPU-9250 mode:

Name: XA_OFFS_H

Address: 119

Serial IF: R/W

Reset value: 0x00

Figura 11. Hoja de datos del módulo MPU9250.

Para iniciar el dispositivo, se debe enviar un 0 al registro 0x6B.

InvenSense	MPU-9250 Register Map and Descriptions	Document Number: RM-MPU-9250A-00 Revision: 1.6 Release Date: 01/07/2015

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
66	102	I2C_SLV3_DO	R/W	I2C_SLV3_DO[7:0]							
67	103	I2C_MST_DELAY_CTRL	R/W	DELAY_ES_SHADOW	-	-	I2C_SLV4_DLY_EN	I2C_SLV3_DLY_EN	I2C_SLV2_DLY_EN	I2C_SLV1_DLY_EN	I2C_SLV0_DLY_EN
68	104	SIGNAL_PATH_RESET	R/W	-	-	-	-	-	GYRO_RST	ACCEL_RST	TEMP_RST
69	105	MOT_DETECT_CTRL	R/W	ACCEL_INTEL_EN	ACCEL_INTEL_MODE	-		-		-	
6A	106	USER_CTRL	R/W	-	FIFO_EN	I2C_MST_EN	I2C_IF_DIS	-	FIFO_RST	I2C_MST_RST	SIG_COND_RST
6B	107	PWR_MGMT_1	R/W	H_RESET	SLEEP	CYCLE	GYRO_STANDBY	PD_PTAT	CLKSEL[2:0]		
6C	108	PWR_MGMT_2	R/W	-		DIS_XA	DIS_YA	DIS_ZA	DIS_XG	DIS_YG	DIS_ZG
72	114	FIFO_COUNTH	R/W	-			FIFO_CNT[12:8]				
73	115	FIFO_COUNTL	R/W	FIFO_CNT[7:0]							
74	116	FIFO_R_W	R/W	D[7:0]							
75	117	WHO_AM_I	R	WHOAMI[7:0]							
77	119	XA_OFFSET_H	R/W	XA_OFFS [14:7]							
78	120	XA_OFFSET_L	R/W	XA_OFFS [6:0]							-
7A	122	YA_OFFSET_H	R/W	YA_OFFS [14:7]							
7B	123	YA_OFFSET_L	R/W	YA_OFFS [6:0]							-
7D	125	ZA_OFFSET_H	R/W	ZA_OFFS [14:7]							
7E	126	ZA_OFFSET_L	R/W	ZA_OFFS [6:0]							-

Table 1 MPU-9250 mode register map for Gyroscope and Accelerometer

Note: Register Names ending in _H and _L contain the high and low bytes, respectively, of an internal register value.

In the detailed register tables that follow, register names are in capital letters, while register values are in capital letters and italicized. For example, the ACCEL_XOUT_H register (Register 59) contains the 8 most significant bits, *ACCEL_XOUT*[15:8], of the 16-bit X-Axis accelerometer measurement, *ACCEL_XOUT*.

The reset value is 0x00 for all registers other than the registers below.

- Register 107 (0x01) Power Management 1
- Register 117 (0x71) WHO_AM_I

Figura 12. Hoja de datos del módulo MPU9250.

Seguidamente, los datos del módulo son leídos. Para ello, se consultan los registros de lectura del módulo y se observa que los registros del 43 al 48 contienen los datos con los valores del giroscopio para los ejes X, Y y Z. Cada valor de giro o aceleración se divide en dos registros de 8 bits, por lo tanto, para obtener el dato de un solo eje, se debe leer dos registros consecutivos (H y L) y así obtener el valor total.

43	67	GYRO_XOUT_H	R	GYRO_XOUT_H[15:8]
44	68	GYRO_XOUT_L	R	GYRO_XOUT_L[7:0]
45	69	GYRO_YOUT_H	R	GYRO_YOUT_H[15:8]
46	70	GYRO_YOUT_L	R	GYRO_YOUT_L[7:0]
47	71	GYRO_ZOUT_H	R	GYRO_ZOUT_H[15:8]
48	72	GYRO_ZOUT_L	R	GYRO_ZOUT_L[7:0]

Figura 13. Hoja de datos del módulo MPU9250.

Para facilitar el trabajo con los registros, se crean dos funciones, una de lectura y otra de escritura de éstos.

```
void lecturaRegistro(uint8_t direccion, uint8_t registro, uint8_t nbytes, uint8_t* datos)
{
  Wire.beginTransmission(direccion);
  Wire.write(registro);
  Wire.endTransmission();

  Wire.requestFrom(direccion, nbytes);
  uint8_t x = 0;
  while (Wire.available())
     datos[x++] = Wire.read();
}
```

Figura 14. Función de lectura de registro.

```
void escribirRegistro(uint8_t direccion, uint8_t registro, uint8_t datos)
{
  Wire.beginTransmission(direccion);
  Wire.write(registro);
  Wire.write(datos);
  Wire.endTransmission();
}
```

Figura 15. Función de escritura de registro.

```
void setup()
{
Wire.begin();
escribirRegistro(0x68, 0x6B, 0x00);

//Gyro
escribirRegistro(0x68, 0x1B, 0x18);
```

Figura 16. Parte 1 del código de lectura del giroscopio.

Se escribe en el registro 1B (registro de configuración del giroscopio) el valor '0x18' para obtener el rango de +2000 dps (degrees per second o º/seg) del giroscopio.

1B	27	GYRO_CONFIG	R/W	XGYRO_Cten	YGYRO_Cten	ZGYRO_Cten	GYRO_FS_SEL [1:0]	-	FCHOICE_B[1:0]

Figura 17. Hoja de datos del módulo MPU9250.

4.6 Register 27 – Gyroscope Configuration

CONFIDENTIAL & PROPRIETARY 13 of 55

InvenSense | MPU-9250 Register Map and Descriptions | Document Number: RM-MPU-9250A-00 Revision: 1.6 Release Date: 01/07/2015

Serial IF: R/W

Reset value: 0x00

BIT	NAME	FUNCTION
[7]	XGYRO_Cten	X Gyro self-test
[6]	YGYRO_Cten	Y Gyro self-test
[5]	ZGYRO_Cten	Z Gyro self-test
[4:3]	GYRO_FS_SEL[1:0]	Gyro Full Scale Select: 00 = +250dps 01= +500 dps 10 = +1000 dps 11 = +2000 dps

Figura 18. Hoja de datos del módulo MPU9250.

En el bucle vacío, se arranca la transmisión una vez más con la misma dirección de esclavo y se escribe la primera dirección de registros de giro, '0x43'. Seguidamente se pide los 6 siguientes registros utilizando la función de solicitud y serán devueltos 6 registros empezando por el 0x43. Estos 6 registros son exactamente los valores altos y bajos de los datos de giros X, Y y Z. Para obtener los valores totales de giro se realiza una operación de OR entre el registro bajo y alto desplazado 8 bits. Se guardan los valores de giro en las variables "Gyr_rawX", "Gyr_rawY" y "Gyr_rawZ", pero hay que tener en cuenta que no son los datos reales del giroscopio, ya que se requiere un valor en grados por segundo. Si se consulta una vez más la hoja de datos del módulo, se puede observar que es necesario dividir el valor entre 16.4 para obtener valores reales en grados por segundo, ya que se ha seleccionado el rango de 2000 º/seg anteriormente. Se realiza esta operación y se almacenan los datos finales.

InvenSense Sensing Everything	MPU-9250 Product Specification	Document Number: PS-MPU-9250A-01 Revision: 1.1 Release Date: 06/20/2016

3 Electrical Characteristics

3.1 Gyroscope Specifications

Typical Operating Circuit of section 4.2, VDD = 2.5V, VDDIO = 2.5V, T_A=25°C, unless otherwise noted.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
Full-Scale Range	FS_SEL=0		±250		°/s
	FS_SEL=1		±500		°/s
	FS_SEL=2		±1000		°/s
	FS_SEL=3		±2000		°/s
Gyroscope ADC Word Length			16		bits
Sensitivity Scale Factor	FS_SEL=0		131		LSB/(°/s)
	FS_SEL=1		65.5		LSB/(°/s)
	FS_SEL=2		32.8		LSB/(°/s)
	FS_SEL=3		16.4		LSB/(°/s)

Figura 19. Hoja de datos del módulo MPU9250.

Al ver estos datos, se observa que hay un pequeño error de calibración, por lo tanto, se debe calcular dicho error (“Gyro_raw_error_x”, “Gyro_raw_error_y” y “Gyro_raw_error_z”) en la función setup(), calculando la media de unas 200 muestras iniciales y restándolo a los datos obtenidos.

Estos valores son datos de velocidad y se requiere únicamente valores de ángulos, entonces, para conseguirlos, son multiplicados por el tiempo transcurrido.

```
if(gyro_error==0)
{
  for(int i=0; i<200; i++)
  {
    Wire.beginTransmission(0x68);
    Wire.write(0x43);
    Wire.endTransmission(false);
    Wire.requestFrom(0x68,6,true);
    gx=Wire.read()<<8|Wire.read();
    gy=Wire.read()<<8|Wire.read();
    gz=Wire.read()<<8|Wire.read();
    Gyro_raw_error_x = Gyro_raw_error_x + (gx/16.4);
    Gyro_raw_error_y = Gyro_raw_error_y + (gy/16.4);
    Gyro_raw_error_z = Gyro_raw_error_z + (gz/16.4);
    if(i==199)
    {
      Gyro_raw_error_x = Gyro_raw_error_x/200;
      Gyro_raw_error_y = Gyro_raw_error_y/200;
      Gyro_raw_error_z = Gyro_raw_error_z/200;
      gyro_error=1;
    }
  }
```

Figura 20. Parte 2 del código de lectura del giroscopio.

```
void loop()
{
  timePrev = time;
  time = millis();
  elapsedTime = (time - timePrev) / 1000;

  Wire.beginTransmission(0x68);
  Wire.write(0x43);
  Wire.endTransmission(false);
  Wire.requestFrom(0x68,6,true);
  gx=Wire.read()<<8|Wire.read();
  gy=Wire.read()<<8|Wire.read();
  gz=Wire.read()<<8|Wire.read();

  gx = (gx/16.4) - Gyro_raw_error_x;
  gy = (gy/16.4) - Gyro_raw_error_y;
  gz = (gz/16.4) - Gyro_raw_error_z;

  Gyro_angle_x = gx*elapsedTime;
  Gyro_angle_y = gy*elapsedTime;
  Gyro_angle_z = gz*elapsedTime;
```

Figura 21. Parte 3 del código de lectura del giroscopio.

Para leer los datos del acelerómetro, se consulta los registros de lectura del módulo y se observa que los registros del 3B al 40 contienen los datos con los valores del acelerómetro para los ejes X, Y y Z.

3B	59	ACCEL_XOUT_H	R	ACCEL_XOUT_H[15:8]
3C	60	ACCEL_XOUT_L	R	ACCEL_XOUT_L[7:0]
3D	61	ACCEL_YOUT_H	R	ACCEL_YOUT_H[15:8]
3E	62	ACCEL_YOUT_L	R	ACCEL_YOUT_L[7:0]
3F	63	ACCEL_ZOUT_H	R	ACCEL_ZOUT_H[15:8]
40	64	ACCEL_ZOUT_L	R	ACCEL_ZOUT_L[7:0]

Figura 22. Hoja de datos del módulo MPU9250.

```
void setup()
{
Wire.begin();
escribirRegistro(0x68, 0x6B, 0x00);

//Acc
escribirRegistro(0x68, 0x1C, 0x18);
```

Figura 23. Parte 1 del código de lectura del acelerómetro.

Se escribe primero en la función setup(), en el registro 1C (registro de configuración del acelerómetro), el valor ‘0x18’ para obtener el rango de +-16g del acelerómetro.

1C	28	ACCEL_CONFIG	R/W	ax_st_en	ay_st_en	az_st_en	ACCEL_FS_SEL[1:0]	-

Figura 24. Hoja de datos del módulo MPU9250.

4.7 Register 28 – Accelerometer Configuration

Serial IF: R/W

Reset value: 0x00

BIT	NAME	FUNCTION
[7]	ax_st_en	X Accel self-test
[6]	ay_st_en	Y Accel self-test
[5]	az_st_en	Z Accel self-test
[4:3]	ACCEL_FS_SEL[1:0]	Accel Full Scale Select: ±2g (00), ±4g (01), ±8g (10), ±16g (11)
[2:0]	-	Reserved

Figura 25. Hoja de datos del módulo MPU9250.

De nuevo, en la función loop(), se escribe la primera dirección de registros del acelerómetro, ‘0x3B’. Seguidamente se piden los 6 siguientes registros utilizando la función de solicitud y 6 registros serán devueltos, empezando por el 0x3B. Se desplaza 8 bits los valores altos, se obtienen los datos completos y se guardan en las variables “Acc_rawX”, “Acc_rawY” y “Acc_rawZ”. Para obtener las aceleraciones reales expresadas en “g” se divide el valor de estas variables entre ‘2048.0’ si se selecciona el rango de +/-16g tal y como podemos ver en las hojas de datos del módulo.

3.2 Accelerometer Specifications

Typical Operating Circuit of section 4.2, VDD = 2.5V, VDDIO = 2.5V, T_A=25°C, unless otherwise noted.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
Full-Scale Range	AFS_SEL=0		±2		*g*
	AFS_SEL=1		±4		*g*
	AFS_SEL=2		±8		*g*
	AFS_SEL=3		±16		*g*
ADC Word Length	Output in two's complement format		16		bits
Sensitivity Scale Factor	AFS_SEL=0		16,384		LSB/*g*
	AFS_SEL=1		8,192		LSB/*g*
	AFS_SEL=2		4,096		LSB/*g*
	AFS_SEL=3		2,048		LSB/*g*

Figura 26. Hoja de datos del módulo MPU9250.

De nuevo, el error de estos datos es calculado para posteriormente aplicarlo a los datos finales.

```
if(acc_error==0)
{
  for(int a=0; a<200; a++)
  {
    Wire.beginTransmission(0x68);
    Wire.write(0x3B);
    Wire.endTransmission(false);
    Wire.requestFrom(0x68,6,true);
    ax=(Wire.read()<<8|Wire.read())/2048.0;
    ay=(Wire.read()<<8|Wire.read())/2048.0;
    az=(Wire.read()<<8|Wire.read())/2048.0;
    Acc_angle_error_x = Acc_angle_error_x + ((atan((ay)/sqrt(pow((ax),2) + pow((az),2)))*rad_to_deg));
    Acc_angle_error_y = Acc_angle_error_y + ((atan(-1*(ax)/sqrt(pow((ay),2) + pow((az),2)))*rad_to_deg));
    if(a==199)
    {
      Acc_angle_error_x = Acc_angle_error_x/200;
      Acc_angle_error_y = Acc_angle_error_y/200;
      acc_error=1;
    }
  }
```

Figura 27. Parte 2 del código de lectura del acelerómetro.

```
void loop()
{
  Wire.beginTransmission(0x68);
  Wire.write(0x3B);
  Wire.endTransmission(false);
  Wire.requestFrom(0x68,6,true);

  ax=(Wire.read()<<8|Wire.read())/2048.0;
  ay=(Wire.read()<<8|Wire.read())/2048.0;
  az=(Wire.read()<<8|Wire.read())/2048.0;

 /*Euler*/
 Acc_angle_x = (atan((ay)/sqrt(pow((ax),2) + pow((az),2)))*rad_to_deg) - Acc_angle_error_x;
 Acc_angle_y = (atan(-1*(ax)/sqrt(pow((ay),2) + pow((az),2)))*rad_to_deg) - Acc_angle_error_y;
```

Figura 28. Parte 3 del código de lectura del acelerómetro.

Una vez se tienen los datos de aceleración, se utilizan las fórmulas de ángulos de Euler con las que obtener el ángulo de inclinación utilizando los datos de las aceleraciones de los tres ejes.

$$AngleY = atan\left(\frac{X}{\sqrt{Y^2 + Z^2}}\right)$$

$$AngleX = atan\left(\frac{Y}{\sqrt{X^2 + Z^2}}\right)$$

Figura 29. Fórmulas de Euler ángulo de inclinación.

Una vez incluidas estas operaciones en el código y ya se tiene el ángulo de inclinación de los ejes X e Y, obtenidos con los valores de la aceleración. Finalmente, para tener mejor solución, se aplica el filtro complementario de Kalman con el ángulo obtenido con los datos del acelerómetro y los datos del giroscopio.

```
Total_angle_x = 0.98 *(Total_angle_x + Gyro_angle_x) + 0.02*Acc_angle_x;

Total_angle_y = 0.98 *(Total_angle_y + Gyro_angle_y) + 0.02*Acc_angle_y;
```

Figura 30. Filtro y ángulo total.

Por último, se realiza la lectura del magnetómetro. En la hoja de datos del módulo se observa que la dirección esclavo es ‘0x0C’.

Pass-Through mode is also used to access the AK8963 magnetometer directly from the host. In this configuration the slave address for the AK8963 is 0X0C or 12 decimal.

Figura 31. Hoja de datos del módulo MPU9250.

También puede extraerse que los datos del magnetómetro se encuentran en los registros del '0x03' al '0x08'.

HXL	03H	READ	Measurement data	8	X-axis data
HXH	04H			8	
HYL	05H			8	Y-axis data
HYH	06H			8	
HZL	07H			8	Z-axis data
HZH	08H			8	

Figura 32. Hoja de datos del módulo MPU9250.

En el registro '0x0A' es escrito el valor '0x01' para elegir el 'MODE 0' del magnetómetro.

```
void setup()
{
//Mag
escribirRegistro(0x0C, 0x0A, 0x01);
```

Figura 33. Parte 1 del código de lectura del magnetómetro.

En el bucle vacío, los datos que están contenidos desde el registro '0x03' son guardados y multiplicados por 0.6 para obtener los datos en µT.

3.3 Magnetometer Specifications

Typical Operating Circuit of section 4.2, VDD = 2.5V, VDDIO = 2.5V, T_A=25°C, unless otherwise noted.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
MAGNETOMETER SENSITIVITY					
Full-Scale Range			±4800		µT
ADC Word Length			14		bits
Sensitivity Scale Factor			0.6		µT / LSB
ZERO-FIELD OUTPUT					
Initial Calibration Tolerance			±500		LSB

Figura 34. Hoja de datos del módulo MPU9250.

A continuación se realizan las operaciones necesarias para obtener la inclinación en el eje Z medida mediante el magnetómetro.

```
void loop()
{
  uint8_t Mag[7];
  lecturaRegistro(0x0C, 0x03, 7, Mag);
  // Convertir registros magnetometro
  int16_t mx = (Mag[3] << 8 | Mag[2])*0.6;
  int16_t my = (Mag[1] << 8 | Mag[0])*0.6;
  int16_t mz = (Mag[5] << 8 | Mag[4])*0.6;
   //////CÁLCULO EJE YAW//////////
   double roll_m = Acc_angle_y / rad_to_deg;
   double pitch_m = Acc_angle_x / rad_to_deg;

   double sinroll_m = sin(roll_m);
   double cosroll_m = cos(roll_m);
   double sinpitch_m = sin(pitch_m);
   double cospitch_m = cos(pitch_m);

   double yh = (mz * sinroll_m) - (my * cosroll_m);
   double xh = (mx * cospitch_m) + (my * (sinpitch_m * sinroll_m)) + (mz * (sinpitch_m *cosroll_m));

   double Mag_angle_z = (atan2(yh, xh) * rad_to_deg);
```

Figura 35. Parte 2 del código de lectura del magnetómetro.

Finalmente, el filtro que une esta vez los datos del giroscopio con los del magnetómetro es aplicado para calcular el ángulo de inclinación total del eje Z.

```
Total_angle_z = 0.96 *(Gyro_angle_z) + 0.04*Mag_angle_z;
```

Figura 36. Parte 3 del código de lectura del magnetómetro.

4.3.- Código PID

El código debe seguir una secuencia en bucle. Primero, debe leer los datos suministrados por el sensor MPU9250 para calcular los ángulos de inclinación en los tres ejes. Seguidamente, se realizan los cálculos necesarios para obtener dichos ángulos. A continuación, a partir de estas lecturas y las consignas recibidas desde el mando transmisor, los PID (uno por cada eje) calculan el error y cuánto acelerar o decelerar cada motor para eliminar dicho error.

Si se contempla como ejemplo el eje Roll del dron, lo primero que debe hacer el código es comparar la referencia que llega desde el mando, con los datos proporcionados por el sensor y con ello, calculando la diferencia entre las dos señales se obtiene el error del eje Roll.

Si desde el transmisor el valor recibido es de 0º de *roll* y el sensor MPU9250 informa de que el valor de inclinación sobre dicho eje es de 10º, el error en el eje es de 10º. El objetivo del PID es hacer que este error, y el de los ejes restantes, sea siempre 0 y para ello se debe actuar sobre los motores hasta que la lectura del sensor sea de 0º y por lo tanto, el error también.

El papel del PID es recibir los errores de los tres ejes, y generar una salida de pulso (µs) en función de las constantes *Kp, Ki* y *Kd* que se hayan establecido previamente según la respuesta que se quiera obtener (más agresiva con valores más elevados o suave con valores bajos).

El código del PID comienza obteniendo el ángulo de inclinación deseado, que será el que se decida enviar desde el transmisor. Este valor debe estar dentro de unos límites de ángulos y

para ello se mapean los valores de un rango de [1000 ; 2000] a un rango de [-10 ; +10] grados con lo cual, cuando el joystick correspondiente se encuentre en las posiciones máximas, se obtendrá un ángulo de giro deseado de +10 o -10. Además, se realiza un ajuste de éstos debido al error producido por los joysticks cuando se encuentran en su valor medio.

```
roll_desired_angle = map(input_ROLL,1000,2000,-10,10);
if(roll_desired_angle == -1) roll_desired_angle = 0;
pitch_desired_angle = map(input_PITCH,1000,2000,-10,10);
if(pitch_desired_angle == -1) pitch_desired_angle = 0;
yaw_desired_angle = map(input_YAW,1000,2000,-90,90);
if(yaw_desired_angle >= -15 && yaw_desired_angle <= 30) yaw_desired_angle = 0;
```

Figura 37. Parte 1 del código del PID.

Seguidamente, se calcula el error de los tres ejes. Este error es la diferencia entre el ángulo real y el deseado (el signo negativo en el primer término es para cuadrar los ejes con los del mando transmisor).

```
roll_error = Total_angle_y - roll_desired_angle;
pitch_error = - Total_angle_x - pitch_desired_angle;
yaw_error = - Total_angle_z - yaw_desired_angle;
```

Figura 38. Parte 2 del código del PID.

Una vez calculado el error, se aplica el algoritmo del PID para todos los ejes.

```
roll_pid_p = roll_kp*roll_error;
pitch_pid_p = pitch_kp*pitch_error;
yaw_pid_p = yaw_kp*yaw_error;

roll_pid_i = roll_pid_i+(roll_ki*roll_error);
roll_pid_i = constrain(roll_pid_i, -400, 400);
pitch_pid_i = pitch_pid_i+(pitch_ki*pitch_error);
pitch_pid_i = constrain(pitch_pid_i, -400, 400);

yaw_pid_i = yaw_pid_i+(yaw_ki*yaw_error);
yaw_pid_i = constrain(yaw_pid_i, -400, 400);

roll_pid_d = roll_kd*((roll_error - roll_previous_error)/elapsedTime);
pitch_pid_d = pitch_kd*((pitch_error - pitch_previous_error)/elapsedTime);
yaw_pid_d = yaw_kd*((yaw_error - yaw_previous_error)/elapsedTime);

roll_PID = roll_pid_p + roll_pid_i + roll_pid_d;
pitch_PID = pitch_pid_p + pitch_pid_i + pitch_pid_d;
yaw_PID = yaw_pid_p + yaw_pid_i + yaw_pid_d;
```

Figura 39. Parte 3 del código del PID.

La parte Kp del PID es proporcional al error y simplemente se multiplican ambos términos.

La parte Ki del PID es proporcional al error que se acumula en cada ciclo. El error actual se multiplica por el término Ki, pero en cada nuevo ciclo de control se suma el valor obtenido en el ciclo anterior. Se consigue de esta forma, que el error en régimen permanente sea de 0.

La parte Kd del PID es proporcional a la diferencia de error entre ciclos consecutivos. Sirve para suavizar la respuesta del control.

```
////////////////////////////////PID FOR ROLL////////////////////////////
float roll_PID, pwm_L_F, pwm_L_B, pwm_R_F, pwm_R_B, roll_error, roll_previous_error;
float roll_pid_p = 0;
float roll_pid_i = 0;
float roll_pid_d = 0;
////////////////////////////////ROLL PID CONSTANTS////////////////////
double roll_kp = 2.5;
double roll_ki = 0.0006;
double roll_kd = 1;
float roll_desired_angle = 0;

////////////////////////////////PID FOR PITCH////////////////////////////
float pitch_PID, pitch_error, pitch_previous_error;
float pitch_pid_p = 0;
float pitch_pid_i = 0;
float pitch_pid_d = 0;
////////////////////////////////PITCH PID CONSTANTS////////////////////
double pitch_kp = 2.5;
double pitch_ki = 0.0006;
double pitch_kd = 1;
float pitch_desired_angle = 0;


////////////////////////////////PID FOR YAW////////////////////////////
float yaw_PID, yaw_error, yaw_previous_error;
float yaw_pid_p = 0;
float yaw_pid_i = 0;
float yaw_pid_d = 0;
////////////////////////////////YAW PID CONSTANTS////////////////////
double yaw_kp = 0.27;
double yaw_ki = 0.00006;
double yaw_kd = 0.10;
float yaw_desired_angle = 0;
```

Figura 40. Parte 4 del código del PID.

Una vez calculado el PID, se restringen sus valores a un rango entre -400 y +400, lo cual proporciona un margen de 800 para no saturar y llegar al valor de 1000 dado por el margen existente en el rango [1000 ; 2000] de los motores.

```
roll_pid_i = roll_pid_i+(roll_ki*roll_error);
roll_pid_i = constrain(roll_pid_i, -400, 400);
pitch_pid_i = pitch_pid_i+(pitch_ki*pitch_error);
pitch_pid_i = constrain(pitch_pid_i, -400, 400);
yaw_pid_i = yaw_pid_i+(yaw_ki*yaw_error);
yaw_pid_i = constrain(yaw_pid_i, -400, 400);
```

Figura 41. Parte 5 del código del PID.

A continuación, se calcula el ancho de pulso de las señales PWM para los ESCs, que será el valor de aceleración recibido, más el valor del PID y un extra de 115 microsegundos para de esta manera, asegurarse de que las hélices se encuentren girando al inicio, esto reducirá las

turbulencias al levantarse desde el suelo y asegurará que todas las hélices estén girando al mismo tiempo.

```
pwm_R_F = 115 + input_THROTTLE + roll_PID + pitch_PID - yaw_PID;
pwm_R_B = 115 + input_THROTTLE + roll_PID - pitch_PID + yaw_PID;
pwm_L_B = 115 + input_THROTTLE - roll_PID - pitch_PID - yaw_PID;
pwm_L_F = 115 + input_THROTTLE - roll_PID + pitch_PID + yaw_PID;
```

Figura 42. Parte 6 del código del PID.

El siguiente paso es enviar estas señales PWM a los ESCs, y para ello se utiliza la escritura de microsegundos del servo “writeMicroseconds()”. Las señales PWM estarán conectadas a los pines D4, D5, D6 y D7 (pines de los cuatro motores).

```
L_F_motor.attach(4);
L_B_motor.attach(5);
R_F_motor.attach(7);
R_B_motor.attach(6);

L_F_motor.writeMicroseconds(1000);
L_B_motor.writeMicroseconds(1000);
R_F_motor.writeMicroseconds(1000);
R_B_motor.writeMicroseconds(1000);
```

Figura 43. Parte 7 del código del PID.

Finalmente, se añade esta parte del código para desactivar los motores cuando el ‘Throttle’ y el ‘Yaw’ se encuentren en su valor más bajo durante aproximadamente 3 segundos y activarlos cuando el acelerador se encuentre a 0 y el Yaw al máximo para iniciar el giro de los motores.

```
if(motores_activados)
{
L_F_motor.writeMicroseconds(pwm_L_F);
L_B_motor.writeMicroseconds(pwm_L_B);
R_F_motor.writeMicroseconds(pwm_R_F);
R_B_motor.writeMicroseconds(pwm_R_B);
}
if(!motores_activados)
{
L_F_motor.writeMicroseconds(1000);
L_B_motor.writeMicroseconds(1000);
R_F_motor.writeMicroseconds(1000);
R_B_motor.writeMicroseconds(1000);
}
```

```
if(input_THROTTLE < (1150) && input_YAW > (1800) && motores_activados==0)
{
  if(activados_count==20)
  {
    motores_activados=1;
    activados_count=0;
  }
  activados_count=activados_count+1;
  delay(5);
}

 if(input_THROTTLE < (1150) && input_YAW < (1150) && motores_activados==1)
{
  if(desactivados_count==20)
  {
    motores_activados=0;
    desactivados_count=0;
  }
  desactivados_count=desactivados_count+1;
  delay(5);
}
```

Figura 44. Parte 8 del código del PID.

4.4.- Código completo

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24_config.h>
#include <RF24.h>
#include <Servo.h>
#include <Wire.h>

Servo L_F_motor;
Servo L_B_motor;
Servo R_F_motor;
Servo R_B_motor;

int input_THROTTLE = 0;
int input_YAW = 0;
int input_PITCH = 0;
int input_ROLL = 0;
```

```
struct Datos {
byte throttle;
byte yaw;
byte pitch;
byte roll;
};

Datos data;

const byte direccion [6] = "00001";
RF24 radio(9, 10);

float rad_to_deg = 180/3.141592654;

//Gyro Variables
float elapsedTime, time, timePrev;
int gyro_error = 0;
float gx, gy, gz;
float Gyro_angle_x, Gyro_angle_y, Gyro_angle_z;
float Gyro_raw_error_x, Gyro_raw_error_y, Gyro_raw_error_z;


//Acc Variables
int acc_error = 0;
float ax, ay, az;
float Acc_angle_x, Acc_angle_y;
float Acc_angle_error_x, Acc_angle_error_y;

//Mag Variables
int mag_error=0;
float Mag_angle_x, Mag_angle_y;
float Mag_angle_error_x, Mag_angle_error_y;

float Total_angle_x, Total_angle_y, Total_angle_z;

//Más variables para el código
int i;
int motores_activados = 0;
long activados_count = 0;
long desactivados_count = 0;



////////////////////////////////PID FOR ROLL////////////////////////////
float roll_PID, pwm_L_F, pwm_L_B, pwm_R_F, pwm_R_B, roll_error, roll_previous_error;
float roll_pid_p = 0;
float roll_pid_i = 0;
float roll_pid_d = 0;
////////////////////////////////ROLL PID CONSTANTS/////////////////////
double roll_kp = 2.5;
double roll_ki = 0.0006;
double roll_kd = 1;
float roll_desired_angle = 0;

////////////////////////////////PID FOR PITCH////////////////////////////
float pitch_PID, pitch_error, pitch_previous_error;
float pitch_pid_p = 0;
float pitch_pid_i = 0;
float pitch_pid_d = 0;
////////////////////////////////PITCH PID CONSTANTS/////////////////////
double pitch_kp = 2.5;
double pitch_ki = 0.0006;
double pitch_kd = 1;
float pitch_desired_angle = 0;


////////////////////////////////PID FOR YAW////////////////////////////
float yaw_PID, yaw_error, yaw_previous_error;
float yaw_pid_p = 0;
float yaw_pid_i = 0;
float yaw_pid_d = 0;
////////////////////////////////YAW PID CONSTANTS/////////////////////
double yaw_kp = 0.27;
double yaw_ki = 0.00006;
double yaw_kd = 0.10;
float yaw_desired_angle = 0;
```

```
//Funcion auxiliar lectura
void lecturaRegistro(uint8_t direccion, uint8_t registro, uint8_t nbytes, uint8_t* datos)
{
  Wire.beginTransmission(direccion);
  Wire.write(registro);
  Wire.endTransmission();

  Wire.requestFrom(direccion, nbytes);
  uint8_t x = 0;
  while (Wire.available())
     datos[x++] = Wire.read();
}


// Funcion auxiliar de escritura
void escribirRegistro(uint8_t direccion, uint8_t registro, uint8_t datos)
{
  Wire.beginTransmission(direccion);
  Wire.write(registro);
  Wire.write(datos);
  Wire.endTransmission();
}



void resetDatos()
{
data.throttle = 127;
data.yaw = 127;
data.pitch = 127;
data.roll = 127;
}

void setup()
{
  L_F_motor.attach(4);
  L_B_motor.attach(5);
  R_F_motor.attach(7);
  R_B_motor.attach(6);

  L_F_motor.writeMicroseconds(1000);
  L_B_motor.writeMicroseconds(1000);
  R_F_motor.writeMicroseconds(1000);
  R_B_motor.writeMicroseconds(1000);


//COMUNICACIÓN CON EL TRANSMISOR
  resetDatos();
  radio.begin();
  radio.setAutoAck(false);
  radio.setDataRate(RF24_250KBPS);
  radio.openReadingPipe(1,direccion);

  radio.startListening();

//COMUNICACIÓN CON LA IMU
Wire.begin();
escribirRegistro(0x68, 0x6B, 0x00);

//Gyro
escribirRegistro(0x68, 0x1B, 0x18);


//Acc
escribirRegistro(0x68, 0x1C, 0x18);

//Mag
escribirRegistro(0x68, 0x37, 0x02);
escribirRegistro(0x0C, 0x0A, 0x01);



//COMUNICACIÓN ARDUINO - PC
Serial.begin(9600);
delay(1000);
time = millis();
```

```
//ERRORES

 //Calculamos el error de los datos del Gyro antes de iniciar el ciclo. Relizamos la media de 200 valores
  if(gyro_error==0)
  {
    for(int i=0; i<200; i++)
    {
      Wire.beginTransmission(0x68);
      Wire.write(0x43);
      Wire.endTransmission(false);
      Wire.requestFrom(0x68,6,true);
      gx=Wire.read()<<8|Wire.read();
      gy=Wire.read()<<8|Wire.read();
      gz=Wire.read()<<8|Wire.read();
      Gyro_raw_error_x = Gyro_raw_error_x + (gx/16.4);
      Gyro_raw_error_y = Gyro_raw_error_y + (gy/16.4);
      Gyro_raw_error_z = Gyro_raw_error_z + (gz/16.4);

      if(i==199)
      {
        Gyro_raw_error_x = Gyro_raw_error_x/200;
        Gyro_raw_error_y = Gyro_raw_error_y/200;
        Gyro_raw_error_z = Gyro_raw_error_z/200;
        gyro_error=1;
      }
    }
  }//fin del cálculo de error de Gyro


 //Calculamos el error de los datos del Acc antes de iniciar el ciclo. Relizamos la media de 200 valores
  if(acc_error==0)
  {
    for(int a=0; a<200; a++)
    {

      Wire.beginTransmission(0x68);
      Wire.write(0x3B);
      Wire.endTransmission(false);
      Wire.requestFrom(0x68,6,true);
      ax=(Wire.read()<<8|Wire.read())/2048.0;
      ay=(Wire.read()<<8|Wire.read())/2048.0;
      az=(Wire.read()<<8|Wire.read())/2048.0;
      Acc_angle_error_x = Acc_angle_error_x + ((atan((ay)/sqrt(pow((ax),2) + pow((az),2)))*rad_to_deg));
      Acc_angle_error_y = Acc_angle_error_y + ((atan(-1*(ax)/sqrt(pow((ay),2) + pow((az),2)))*rad_to_deg));
      if(a==199)
      {
        Acc_angle_error_x = Acc_angle_error_x/200;
        Acc_angle_error_y = Acc_angle_error_y/200;
        acc_error=1;
      }
    }
  }//fin del cálculo de error del Acc

}//fin lazo setup()

unsigned long lastRecvTime = 0;

void recibirDatos()
{
while ( radio.available() ) {
radio.read(&data, sizeof(Datos));
lastRecvTime = millis();
}
}
```

```
void loop()
{
//RECIBIMOS DEL TRANSMISOR
recibirDatos();
unsigned long now = millis();
//Comprobamos si se ha perdido la señal
if ( now - lastRecvTime > 1000 ) {
//Si se ha perdido la señal
resetDatos();
}

//recibimos valores del mando
input_THROTTLE = map(data.throttle, 0, 255, 1000, 2000);
input_YAW = map(data.yaw,      0, 255, 1000, 2000);
input_PITCH = map(data.pitch,    0, 255, 1000, 2000);
input_ROLL = map(data.roll,     0, 255, 1000, 2000);


//////////////////////////////I M U/////////////////////////////////////
  timePrev = time;  // guardamos el tiempo anterior antes de leer el actual
  time = millis();  // lectura del tiempo actual
  elapsedTime = (time - timePrev) / 1000; //dividimos entre 1000 para obtener segundos


//LECTURA IMU
 ////////////////////////////////////Gyro read/////////////////////////////////////
  Wire.beginTransmission(0x68);
  Wire.write(0x43);
  Wire.endTransmission(false);
  Wire.requestFrom(0x68,6,true);
  gx=Wire.read()<<8|Wire.read();
  gy=Wire.read()<<8|Wire.read();
  gz=Wire.read()<<8|Wire.read();


  /*Obtenemos los datos de Gyro en grados/segundos*/
  gx = (gx/16.4) - Gyro_raw_error_x;
  gy = (gy/16.4) - Gyro_raw_error_y;
  gz = (gz/16.4) - Gyro_raw_error_z;
  /*Integramos para obtener los datos únicamente en grados */
  Gyro_angle_x = gx*elapsedTime;
  Gyro_angle_y = gy*elapsedTime;
  Gyro_angle_z = Gyro_angle_z + gz*elapsedTime;

  ////////////////////////////////////Acc read/////////////////////////////////////
  Wire.beginTransmission(0x68);
  Wire.write(0x3B);
  Wire.endTransmission(false);
  Wire.requestFrom(0x68,6,true);

  ax=(Wire.read()<<8|Wire.read())/2048.0;
  ay=(Wire.read()<<8|Wire.read())/2048.0;
  az=(Wire.read()<<8|Wire.read())/2048.0;


 /*fórmulas de Euler*/
 Acc_angle_x = (atan((ay)/sqrt(pow((ax),2) + pow((az),2)))*rad_to_deg) - Acc_angle_error_x;
 Acc_angle_y = (atan(-1*(ax)/sqrt(pow((ay),2) + pow((az),2)))*rad_to_deg) - Acc_angle_error_y;

 ///////////LECTURA MAGNETÓMETRO////////
  uint8_t ST1;
  escribirRegistro(0x0C,0x0A,0x01);

  do
  {
     lecturaRegistro(0x0C, 0x02, 1, &ST1);
  } while (!(ST1 & 0x01));

  uint8_t Mag[7];
  lecturaRegistro(0x0C, 0x03, 7, Mag);
```

```
// Convertir registros magnetometro
int16_t mx = (Mag[3] << 8 | Mag[2])*0.6;
int16_t my = (Mag[1] << 8 | Mag[0])*0.6;
int16_t mz = (Mag[5] << 8 | Mag[4])*0.6;


//////CÁLCULO EJE YAW//////////
double roll_m = Acc_angle_y / rad_to_deg;
double pitch_m = Acc_angle_x / rad_to_deg;

double sinroll_m = sin(roll_m);
double cosroll_m = cos(roll_m);
double sinpitch_m = sin(pitch_m);
double cospitch_m = cos(pitch_m);

double yh = (mz * sinroll_m) - (my * cosroll_m);
double xh = (mx * cospitch_m) + (my * (sinpitch_m * sinroll_m)) + (mz * (sinpitch_m *cosroll_m));

double Mag_angle_z = (atan2(yh, xh) * rad_to_deg);

////////////////////////////////////Total angle and filter/////////////////////////////////////
Total_angle_x = 0.98 *(Total_angle_x + Gyro_angle_x) + 0.02*Acc_angle_x;
Total_angle_y = 0.98 *(Total_angle_y + Gyro_angle_y) + 0.02*Acc_angle_y;
Total_angle_z = 0.96 *(Gyro_angle_z) + 0.04*Mag_angle_z;



//////////////////////////P I D////////////////////////////////
roll_desired_angle = map(input_ROLL,1000,2000,-10,10);
if(roll_desired_angle == -1) roll_desired_angle = 0;
pitch_desired_angle = map(input_PITCH,1000,2000,-10,10);
if(pitch_desired_angle == -1) pitch_desired_angle = 0;
yaw_desired_angle = map(input_YAW,1000,2000,-90,90);
if(yaw_desired_angle >= -15 && yaw_desired_angle <= 30) yaw_desired_angle = 0;


roll_error = Total_angle_y - roll_desired_angle;
pitch_error = - Total_angle_x - pitch_desired_angle;
yaw_error = - Total_angle_z - yaw_desired_angle;

roll_pid_p = roll_kp*roll_error;
pitch_pid_p = pitch_kp*pitch_error;
yaw_pid_p = yaw_kp*yaw_error;

roll_pid_i = roll_pid_i+(roll_ki*roll_error);
roll_pid_i = constrain(roll_pid_i, -400, 400);
pitch_pid_i = pitch_pid_i+(pitch_ki*pitch_error);
pitch_pid_i = constrain(pitch_pid_i, -400, 400);
yaw_pid_i = yaw_pid_i+(yaw_ki*yaw_error);
yaw_pid_i = constrain(yaw_pid_i, -400, 400);

roll_pid_d = roll_kd*((roll_error - roll_previous_error)/elapsedTime);
pitch_pid_d = pitch_kd*((pitch_error - pitch_previous_error)/elapsedTime);
yaw_pid_d = yaw_kd*((yaw_error - yaw_previous_error)/elapsedTime);


roll_PID = roll_pid_p + roll_pid_i + roll_pid_d;
pitch_PID = pitch_pid_p + pitch_pid_i + pitch_pid_d;
yaw_PID = yaw_pid_p + yaw_pid_i + yaw_pid_d;

roll_PID = constrain(roll_PID, -400, 400);
pitch_PID = constrain(pitch_PID, -400, 400);
yaw_PID = constrain(yaw_PID, -400, 400);

/*Calculamos el ancho de pulso*/
pwm_R_F /*2*/ = 115 + input_THROTTLE + roll_PID + pitch_PID - yaw_PID;
pwm_R_B /*4*/ = 115 + input_THROTTLE + roll_PID - pitch_PID + yaw_PID;
pwm_L_B /*3*/ = 115 + input_THROTTLE - roll_PID - pitch_PID - yaw_PID;
pwm_L_F /*1*/ = 115 + input_THROTTLE - roll_PID + pitch_PID + yaw_PID;
```

```
/*De nuevo restringimos los valores para asegurarnos de no sobrepasar
 los límites.*/
//Right front
if(pwm_R_F < 1100)
{
  pwm_R_F= 1100;
}
if(pwm_R_F > 2000)
{
  pwm_R_F=2000;
}

//Left front
if(pwm_L_F < 1100)
{
  pwm_L_F= 1100;
}
if(pwm_L_F > 2000)
{
  pwm_L_F=2000;


}

//Right back
if(pwm_R_B < 1100)
{
  pwm_R_B= 1100;
}
if(pwm_R_B > 2000)
{
  pwm_R_B=2000;
}

//Left back
if(pwm_L_B < 1100)
{
  pwm_L_B= 1100;
}
if(pwm_L_B > 2000)
{
  pwm_L_B=2000;
}

roll_previous_error = roll_error;
pitch_previous_error = pitch_error;
yaw_previous_error = yaw_error;

  if(motores_activados)
  {
  L_F_motor.writeMicroseconds(pwm_L_F);
  L_B_motor.writeMicroseconds(pwm_L_B);
  R_F_motor.writeMicroseconds(pwm_R_F);
  R_B_motor.writeMicroseconds(pwm_R_B);
  }
  if(!motores_activados)
  {
  L_F_motor.writeMicroseconds(1000);
  L_B_motor.writeMicroseconds(1000);
  R_F_motor.writeMicroseconds(1000);
  R_B_motor.writeMicroseconds(1000);
  }
```

```
if(input_THROTTLE < (1150) && input_YAW > (1800) && motores_activados==0)
{
  if(activados_count==20)
  {
    motores_activados=1;
    activados_count=0;
  }
  activados_count=activados_count+1;
  delay(5);
}

 if(input_THROTTLE < (1150) && input_YAW < (1150) && motores_activados==1)
{
  if(desactivados_count==20)
  {
    motores_activados=0;
    desactivados_count=0;
  }
  desactivados_count=desactivados_count+1;
  delay(5);
}
```