UNIVERSIDAD DE VALLADOLID

Escuela Técnica Superior de Ingenieros de Telecomunicación

TRABAJO FIN DE GRADO

Grado en Ingeniería de Tecnologías Específicas de Telecomunicación Mención en Sistemas Electrónicos

Diseño y fabricación de un cuadricóptero (dron) auto estabilizado en posición y altura con ruta de vuelo programable.

Autor: Francisco Javier Hernantes Luaces.

Tutor: Martín Jaraíz Maldonado.

2023

TÍTULO	**Diseño y fabricación de un cuadricóptero (dron) auto estabilizado en posición y altura con ruta de vuelo programable**
AUTOR:	**Francisco Javier Hernantes Luaces**
TUTOR:	**Martín Jaraíz Maldonado**
DEPARTAMENTO:	**Electricidad y Electrónica**

TRIBUNAL

PRESIDENTE:	**Jesús Arias Álvarez**
SECRETARIO:	**Jesús M. Hernández Mangas**
VOCAL:	**Martín Jaraíz Maldonado**
SUPLENTE 1:	**Lourdes Pelaz Montes**
SUPLENTE 2:	**Ruth Pinacho Gómez**

FECHA:	
CALIFICACIÓN:	

AGRADECIMIENTOS

A mis padres y a mi hermana, cuyo apoyo durante todos estos años ha sido incondicional y fundamental para conseguir los objetivos que me propuse al principio del grado.

Al tutor del TFG, Martín Jaraíz Maldonado por su ayuda y compromiso durante todos estos meses de trabajo.

RESUMEN DEL PROYECTO

El objetivo de este proyecto consiste en diseñar y fabricar un cuadricóptero (dron) que sea capaz de despegar, auto estabilizarse y aterrizar de forma adecuada utilizando el microcontrolador ESP32 de la empresa Espressif Systems.

El cuadricóptero se basa en una estructura con un soporte central capaz de cargar la batería, el ESP32 y el sistema de medición inercial MPU6050 que nos proporcionará el pitch, el roll y el yaw de del cuadricóptero, los motores se situarán en cada uno de los cuatro ejes que hay alrededor del soporte central y mediante la señal PWM que reciben giraran más rápido o más lento en función del ciclo de trabajo de esta señal PWM. De esta forma, el dron podrá auto estabilizarse variando la velocidad de los motores en función de los movimientos que se quieran hacer enviados desde un mando a distancia que consta de otro microcontrolador ESP32, dos joysticks, una pantalla OLED y un led rojo.

ABSTRACT

The objective of this project is to design and manufacture a quadcopter (drone) that is capable of taking off, self-stabilizing and landing properly using the ESP32 microcontroller from Espressif Systems.

The quadcopter is based on a structure with a central support capable of charging the battery, the ESP32 and the MPU6050 inertial measurement system that will provide us with the pitch, roll and yaw of the quadcopter, the motors will be located in each of the four axes that are around the central support and through the PWM signal they receive will rotate faster or slower depending on the duty cycle of this PWM signal. In this way, the drone will be able to stabilize itself by varying the speed of the motors depending on the movements that it wants to make, sent from a remote control that consists of another ESP32 microcontroller, two joysticks, an OLED screen and a red Led.

PALABRAS CLAVE

Dron, ESP32, MPU6050, PWM, pitch, roll, yaw, PID.

Keywords

Dron, ESP32, MPU6050, PWM, pitch, roll, yaw, PID.

Índice general

1 Introducción

1.1 Introducción

La presente memoria tiene la funcionalidad de detallar el diseño y fabricación de un cuadricóptero(dron) auto estabilizado en posición y altura con ruta de vuelo programable mediante la utilización de un microcontrolador ESP32 programado en el lenguaje C++ con el entorno de programación de Arduino que nos permite realizar todas las funciones que tiene que realizar el dron para auto estabilizarse y poder realizar una ruta de vuelo programable.

Este proyecto pertenece al Departamento de Electricidad y Electrónica de la Escuela técnica superior de Ingenieros de Telecomunicación de la Universidad de Valladolid.

En los siguientes capítulos se explicarán los siguientes conceptos fundamentales:

- Para procesar toda la información se utiliza un microcontrolador ESP32, es un microcontrolador de bajo consumo y bajo coste desarrollado por la empresa Espressif systems, se caracteriza por poseer protocolos de comunicaciones inalámbricos como WiFi y bluetooth y además tener un procesador de doble núcleo llamado Xtensa LX6, este microprocesador es compatible con el lenguaje C++ que se utilizará en este proyecto.
- Un sistema de Medición inercial en concreto el MPU6050, es un dispositivo que permite medir el ángulo de pitch, roll y yaw, consta de 3 acelerómetros que miden la aceleración en el eje x, y, z, también consta de un giroscopio para medir la velocidad angular en cada uno de los ejes anteriores.
- Un sistema de comunicaciones el ESP-NOW, permite comunicar los distintos ESP32 entre sí y enviar información en ambos sentidos entre estos microcontroladores, este protocolo de comunicaciones fue creado por la empresa Espressif systems se caracteriza por su bajo consumo de potencia y por trabajar en la banda de frecuencias de 2.4 GHZ.
- Un controlador PID (controlador proporcional, integral y derivativo) es un mecanismo de control que a través de un lazo de retroalimentación permite corregir el error que tenemos entre la referencia que queremos enviada desde el mando y la salida en ese instante (inclinación del dron en cada eje), mediante un actuador se genera un efecto corrector sobre la salida, de esta forma reducimos el error tendiendo a igualar la salida y la referencia.

1.2 Motivación

La aparición de los drones ha supuesto un gran desarrollo tecnológico en el mundo y en la vida de las personas, con este proyecto se pretende diseñar un dron auto estabilizado con ruta de vuelo programable para introducirse más en este fascinante mundo de los vehículos aéreos no tripulados.

El mercado de los drones ha crecido exponencialmente en los últimos años tanto en los sectores de consumo, comercial como militar. Para entender la importancia que tendrán los drones en el futuro podemos fijarnos en un estudio de Porsche consulting en el cual se estima que en 2035 el negocio de 'air taxi' alcanzará los 32.000 millones de euros, otras fuentes fiables como SENASA (empresa pública dependiente del ministerio de transporte, movilidad y agenda urbana) estiman que para 2030 habrá 340 millones de personas que vivirán en ciudades que podrán acceder a los servicios de movilidad Aérea Urbana no tripulada, además esta fuente estima que se crearán 90000 puestos de trabajo para 2030 en este innovador sector.

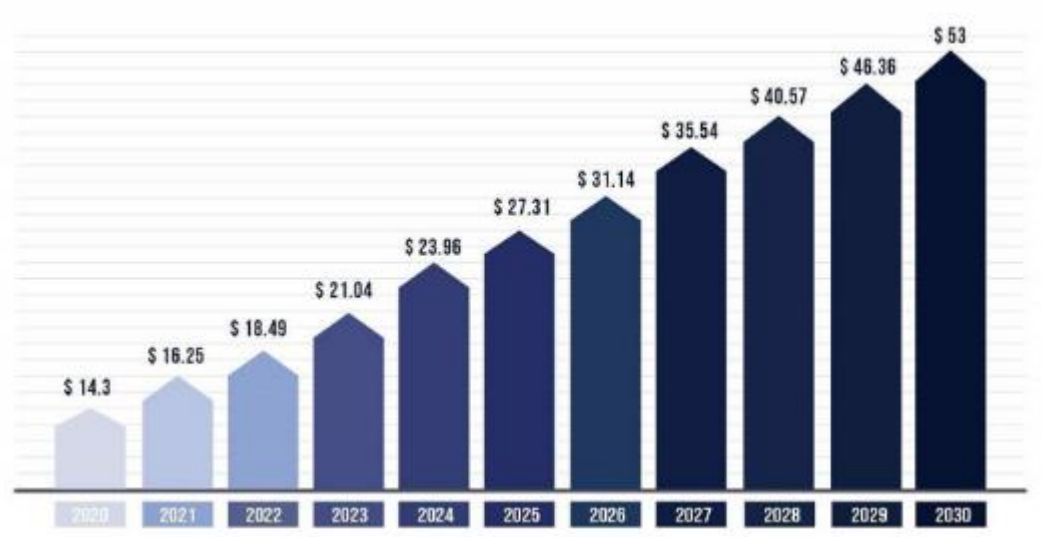


Figura 1. Tamaño del mercado mundial de los drones en millones de dolares (Precedence Research 2023).

En la Unión Europea se estima que el uso comercial de drones en las ciudades empezará a producirse en pequeña escala alrededor de 2025 con la entrega de mercancías mediante drones o el uso de estos vehículos no tripulados para el transporte urbano de personas, además la Agencia Europea de Seguridad Aérea (EASA) estima que la movilidad Aérea Urbana será más segura que la movilidad por carretera, reduciendo en consecuencia el número de accidentes.

ENAIRE (el gestor de navegación aérea de España) está preparándose para la operación segura de transportes de movilidad urbana como los aerotaxis, apoyados en el sistema U-space que es el futuro marco operativo donde los drones podrán operar de forma automatizada y totalmente segura al igual que los aviones convencionales que tenemos en la actualidad, últimamente ya se han realizado pruebas de este tipo donde se analizó como funcionaban estos vehículos dentro de un espacio aéreo controlado, una prueba fue realizada el pasado septiembre en Santiago de Compostela.

En la actualidad existen múltiples ramas de la ciencia que utilizan drones para su beneficio como es la cartografía, que gracias a las cámaras y distintos sensores que llevan los drones nos permite hacer un mapeo muy preciso del terreno con un bajo coste de dinero y de

tiempo. La principal forma de crear modelos 3D de mapas es mediante los sensores LIDAR que emiten pulsos láser en un intervalo de tiempo regular, y calculan el tiempo que tarda el pulso en rebotar, con este tiempo se calcula las coordenadas X, Y y Z. Cuantas más muestras se obtienen más preciso es el mapa que se obtiene.

Una de las operaciones más interesantes que realizan los drones en mi opinión es en buques marítimos, realizan tareas de inspección del tanque de combustible cuando este se vacía para comprobar que no existen fugas de combustible ni cualquier tipo de problema en el tanque, se utiliza los drones para estas tareas ya que para una persona meterse dentro del tanque puede ser incomodo además de que puede haber gases nocivos para la salud de las personas. En la industria marítima también se utilizan para la vigilancia de puertos y evitar el robo de mercancías, también lo utiliza la policía para evitar la pesca furtiva.

Los drones también se han utilizado para transporte de mercancías no muy pesadas, por ejemplo una empresa de transporte marítimo Maersk probó con éxito el transporte de mercancía desde un el puerto hasta un buque que estaba cerca de este puerto, la parte difícil de esta operación es que fue muy difícil encontrar un dron que cumpla la legislación para transportar mercancías de un cierto peso, el mercado actual de drones para el transporte de mercancías es mínimo ya que es muy difícil que el dron pueda mantener la estabilidad si se le pone una elevada mercancía al dron, actualmente empresas de rescate marítimo utilizan drones para entregar chalecos y balsas salvavidas o alguna pieza de repuesto no muy grande a barcos que se quedan estropeados en el mar.

Otra de las principales funciones de los drones es en la industria agraria por ejemplo se utilizan los drones para la fumigación agrícola ya que nos permite utilizar el dron con un tanque y un dispensador para fumigar el campo, los drones nos permiten recorrer una elevada cantidad de terreno en muy poco tiempo y sin necesidad de pisar el terreno como se haría con cualquier tipo de maquinaria o mediante una persona, este uso es muy usado en Castilla y León al ser una comunidad agrícola.

Además de todas estas aplicaciones tiene otras muchas más como son la industria militar por ejemplo en la actual guerra de Ucrania que los drones se utilizan como instrumentos de lanzamiento de granadas o como proyectiles kamikaze contra el objetivo rival, también se utiliza para visualizar desde el aire donde se encuentra la posición del enemigo y tenerlo en todo momento localizado.

Sin embargo uno de los problemas de los drones en la actualidad es la legislación que existe, ya que es muy restrictiva, por ejemplo no se puede sobrepasar 120 metros verticales respecto al suelo ni 50 metros horizontales respecto a la persona que lo maneja y hay que estar a una distancia de al menos 8km de cualquier aeropuerto, además se necesita un carnet que para obtenerlo es necesario pasar un examen, este tipo de leyes se aplica para drones de más de 250 gramos, para drones de menos de 250 gramos existe una normativa más laxa aunque tampoco puedes volar ni cerca de aeropuertos ni parques nacionales.

En conclusión, podemos deducir que la evolución de los drones en la sociedad es cada vez mayor, por todos estos motivos cada vez más gente se interesa en este mundo y en intentar construir uno para ellos que pueda realizar las funciones que desean, gracias al crecimiento exponencial de este sector se ha convertido los drones en una oportunidad laboral muy importante para ingenieros de telecomunicaciones como yo, de ahí mi interés por el mundo de los drones y mi intención de aprender a realizar un dron como es el caso de este proyecto final de carrera que me sirve para aprender como estabilizar un dron en el aire, y a partir de ahí se puede intentar enfocar el dron realizado en cualquiera de los sectores mencionados en este apartado anteriormente.

1.3 Objetivos

El objetivo de este proyecto es el diseño y fabricación de un dron que sea capaz de estabilizarse en el aire gracias a el uso de un microcontrolador ESP32 y un PID que nos mantenga al dron en la posición que nosotros enviemos desde nuestro mando de radiocontrol, para saber cuál es la inclinación y la rotación de nuestro dron se utiliza el sistema de medición inercial MPU6050, habrá que evitar que los motores produzcan muchas interferencias que afecten a las señales que envía el MPU6050 al ESP32 y en consecuencia no se pueda medir adecuadamente la posición del dron y afecte a su estabilidad, por lo que habrá que realizar técnicas para reducir las interferencias que producen estos motores, además será necesario realizar un circuito electrónico para medir el voltaje que tiene la batería en cada momento, el último objetivo fue hacer que el dron tuviese una ruta programable, para ello se propuso que durante un tiempo el dron se inclinase hacia el lado que quisiésemos para que la persona pudiese indicar hacia donde debía ir el dron mediante código.

1.4 Requisitos funcionales del proyecto

En este proyecto, el usuario podrá interactuar con el dron gracias al mando a distancia que poseemos.

Variables de control que se pueden modificar:

- Pitch de referencia para el PID para mover el dron hacia delante o hacia atrás.
- Roll de referencia para el PID para mover el dron hacia la derecha o hacia la izquierda.
- Yaw de referencia para el PID para rotar el dron en sentido horario o antihorario.
- Throttle que consiste en aumentar o disminuir la velocidad de todos los motores.

Se dispone de una pantalla Oled para visualizar información acerca del dron:

- pitch del dron en ese instante.
- roll del dron en ese instante.
- yaw del dron en ese instante.
- Voltaje de la batería en ese instante.

El principal requisito de este proyecto es realizar un dron que funcione en modo estable y que se pueda controlar adecuadamente con un mando a distancia por parte de una persona, además se debe poder programar una ruta de vuelo para el dron por código.

2 Hardware

2.1 ESP32

El ESP32 es un microcontrolador de bajo coste y bajo consumo diseñado por la empresa ESpressif systems, es el sucesor del ESP8266, en nuestro proyecto el ESP32 será el cerebro del dron y se encargará de comunicarse con el otro ESP32 del mando.

Las Especificaciones del ESP32 son:

- Conectividad inalámbrica con una antena integrada:
    - WiFi a una tasa de datos de 150 Mbps.
    - Bluetooth: BLE(bluetooth de bajo consumo) y bluetooth clásico.
- Procesador: microprocesador Tensilica Xtensa de doble núcleo de 32 bits que funciona a 160 o 240 MHz, el reloj funcionara de 80MHz a 240 MHz.
- Memoria:
    - ROM: 448 KB, se utiliza para funciones básicas y para el arranque.
    - SRAM: 520KB, se utiliza para datos e instrucciones.
    - RTC fas SRAM: 8KB, se utiliza para almacenar datos y CPU principal durante el arranque.
    - RTC lento SRAM: 8KB, para acceder al coprocesador durante el modo de suspensión profunda.
    - eFuse: 1Kbit, 256 bits se utilizan para la dirección MAC que cambia según el ESP32 y los 768 bits restantes se reservan para las aplicaciones del cliente
    - Flash: 4MB para almacenar datos y el programa cuando no hay alimentación, de esta forma solo se necesita conectar alimentación para volver a ejecutar el programa.
- Interfaz USB a UART: permite conectar el ordenador al ESP32para cargar el programa en este y aplicarle la alimentación que necesite.
- Botones de BOOT y RESET: sirve para poner la placa en modo intermitente para poder cargar el programa o en modo reinicio para resetear la placa respectivamente.
- Pines de alimentación de 3.3V y 5V y los pines de tierra (GND).
- 48 GPIOs con múltiples funciones, aquí se puede ver todas las funciones que hay:
    - 18 convertidores analógicos a digital.
    - 3 interfaces SPI.
    - 3 interfaces UART.
    - 2 interfaces I2C.
    - 16 canales de salida PWM.
    - 2 convertidores de digital a analógico.
    - 2 interfaces I2S.
    - 10 GPIO de detección capacitiva.

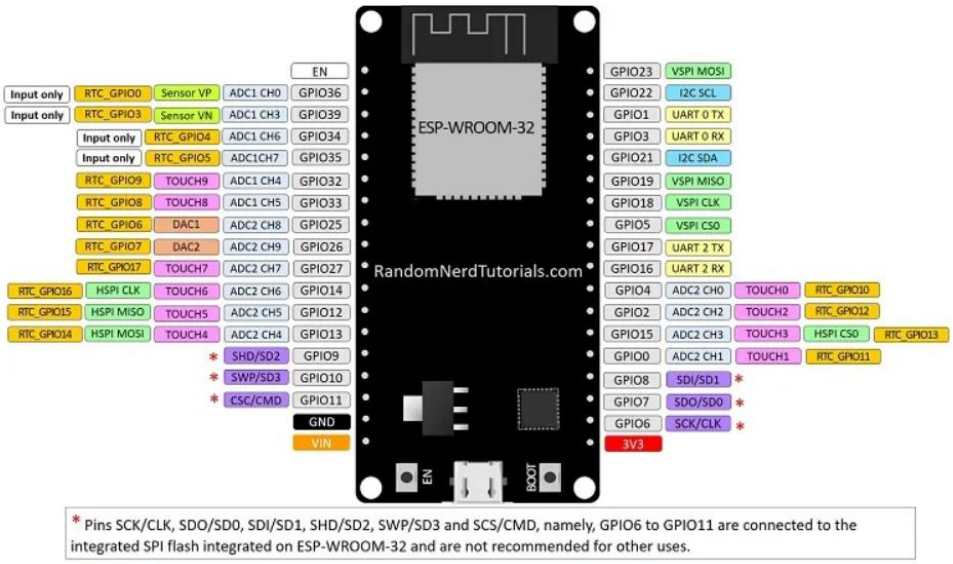


Figura 2. Modelo ESP32 del dron (Randomn Tutorials,2023).

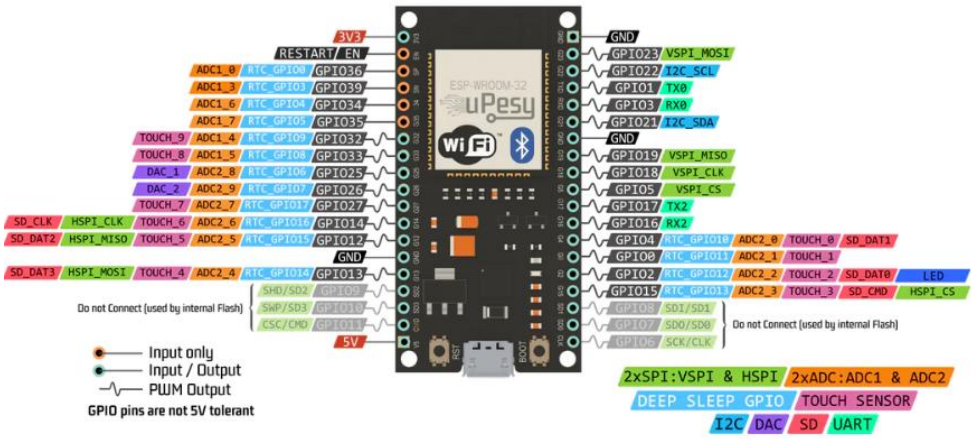


Figura 3. Modelo ESP32 del mando (Upsey,2023).

De todos los GPIOS existen algunos que solo se pueden utilizar como entradas que son los GPIO 34,35,36 y 39. Luego hay otros que no se pueden utilizar ni como entrada ni como salida que son los GPIO del 6 al 11, estos pines están conectados al flash SPI del ESP32, además hay GPIOS que tienen otras características relevantes como el GPIO 0 que tiene una resistencia de pull up automáticamente cuando se configura como entrada, también cabe destacar los GPIO 1 y 3 que son los pines de transmisión y recepción respectivamente de la UART.

GPIO	Input	Output	Notes
0	pulled up	OK	outputs PWM signal at boot, must be LOW to enter flashing mode
1	TX pin	OK	debug output at boot
2	OK	OK	connected to on-board LED, must be left floating or LOW to enter flashing mode
3	OK	RX pin	HIGH at boot
4	OK	OK	
5	OK	OK	outputs PWM signal at boot, strapping pin
6	x	x	connected to the integrated SPI flash
7	x	x	connected to the integrated SPI flash
8	x	x	connected to the integrated SPI flash
9	x	x	connected to the integrated SPI flash
10	x	x	connected to the integrated SPI flash
11	x	x	connected to the integrated SPI flash
12	OK	OK	boot fails if pulled high, strapping pin
13	OK	OK	
14	OK	OK	outputs PWM signal at boot
15	OK	OK	outputs PWM signal at boot, strapping pin
16	OK	OK	
17	OK	OK	
18	OK	OK	
19	OK	OK	
21	OK	OK	
22	OK	OK	
23	OK	OK	
25	OK	OK	
26	OK	OK	
27	OK	OK	
32	OK	OK	
33	OK	OK	
34	OK		input only
35	OK		input only
36	OK		input only
39	OK		input only

Figura 4. Funciones pines ESP32 (Randomn Tutorials,2023).

2.2 Motores brushless 2200KV y ESC(speed controllers) 30A

Estos motores sin escobillas se encargan de girar las hélices del dron para que pueda despegar el dron y moverse en función de las consignas que le enviemos, se van a utilizar 4 motores de 2200KV y 4 ESC (speed controllers) de 30 A, cuando hacemos que

los motores giren a una velocidad media los ESC consumen una corriente de 10-15 amperios, mientras que a una velocidad alta consumen de 20-25 amperios, sin embargo a baja velocidad consumen una corriente de 5 a 10 amperios , la principal restricción es que no se les puede alimentar a más 45 amperios durante más de 123 segundos de funcionamiento, también cabe destacar que se dispone de una salida de 5 voltios que nos permitirá alimentar el ESP32 el cual consumirá una corriente de 80 a 260 mili amperios al alimentarlo a través de la patilla de 5V del ESP32, gracias a estos componentes y a la generación de una señal PWM podemos mover los motores a la velocidad que nosotros queramos.

Figura 5. Motor brushless con ESC (speed controller) (Amazon,2023).

2.3 Sensor MPU6050

El MPU6050 es un sistema de medición inercial que posee giroscopio y acelerómetro y nos permite utilizar ambos para darnos un ángulo exacto de pitch, yaw y roll de nuestro dron utilizando para ello los datos de la aceleración y la rotación tanto en el eje x, y como z que es capaz de medir este sensor.

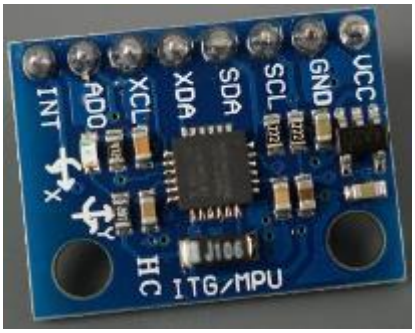


Figura 6. sensor MPU6050 (Randomn Tutorials,2023).

2.4 Mando a distancia

El mando está formado por dos joysticks en el que cada uno tiene 2 ejes, cada eje tiene una salida que va al conversor analógico digital que nos dice en qué posición se encuentra el

joystick, además tenemos una pantalla oled que nos permite visualizar los valores del pitch, yaw, roll y el valor del voltaje de la batería, además se dispone de un led rojo para indicar si el voltaje de la batería es igual o menor de 9V que indica que la batería está descargada.

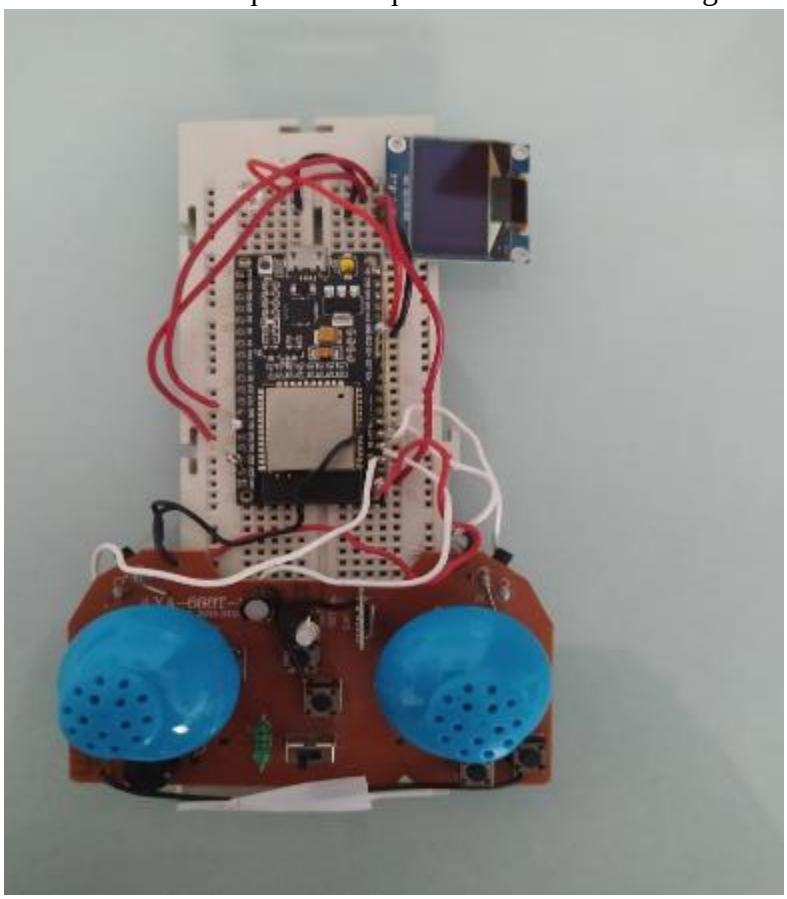


Figura 7. Foto del montaje del mando a distancia.

2.5 Batería Roaring Top 2200mAh

Para alimentar los motores sin escobillas del dron utilizamos una batería de polímero de iones de litio que se caracterizan por estar formado por una pila recargable (célula secundaria), la batería tiene varias células secundarias idénticas en paralelo que nos permite aumentar la capacidad de la corriente de descarga.

Características de la batería:

- Voltaje = 11.1V (3 celdas cada una 3.7V (11.1V/3=3.7V)).
- Velocidad de descarga máxima: 35C.
- Capacidad: 2200mAh.
- Tamaño de la batería: 107.5 * 36.3 * 24.2mm.
- Peso de la batería: 174g.
- Conector XT60 hembra específico para aguantar elevada carga y potencia.

Figura 8. Batería (Amazon,2023).

Esta batería al tener una velocidad de descarga máxima de 35C, por lo que la máxima corriente que puede ofrecer es 77A (35*2.2A=77A).
El tiempo que va a estar la batería funcionando es directamente proporcional a la capacidad de la batería e inversamente proporcional a la corriente a la que se descarga la batería que es la corriente que demandan los motores, en nuestra fórmula haremos un cálculo estimado suponiendo que nuestro dron trabaja con motores a velocidad media de giro y velocidad baja de giro, por lo que cada uno demanda de 10 a 15 amperios de corriente a la batería trabajando a velocidades medias y de 5 a 10 amperios a velocidades bajas.
Velocidad media de giro de los motores:

$$duración\ de\ la\ batería(horas) = \frac{Capacidad\ de\ la\ batería(mAh)}{Corriente\ de\ descarga(Amperios)} = \frac{2200mAh}{(10-15)\ Amperios * 4\ ESC}$$

$$duración\ de\ la\ batería(horas) = 0.036\text{-}0.055$$

$$duración\ de\ la\ batería(minutos) = 2.2\text{-}3.3$$

Velocidadbaja de giro de los motores:

$$duración\ de\ la\ batería(horas) = \frac{Capacidad\ de\ la\ batería(mAh)}{Corriente\ de\ descarga(Amperios)} = \frac{2200mAh}{(5-10)\ Amperios * 4\ ESC}$$

$$duración\ de\ la\ batería(horas) = 0.055\text{-}0.11$$

$$duración\ de\ la\ batería(minutos) = 3.3\text{-}6.6$$

Nuestros motores suelen girar más tiempo a velocidades bajas, por lo que las baterías suelen durar aproximadamente 5 minutos con un funcionamiento bastante aceptable.

2.6 Cargador de la batería Lipo 3s

Para cargar las baterías anteriores es necesario un cargador capaz de cargar baterías en configuración 3S y con voltaje de 11.1V que necesita la batería, por ello se dispone de la batería Haisito B3 en configuración 3S y con corriente de salida de 3*800mA, por lo que carga cada celda secundaria con una corriente de 800mA.
Características de el cargador:

- Corriente de salida máxima: 3 * 800mA.
- Tamaño del cargador: 100* 60 * 35 mm.
- Peso del artículo: 180 g.

Figura 9. Cargador de Batería LIPO (Amazon,2023).

$$tiempo\ de\ carga\ (h) = \frac{Capacidad\ de\ la\ batería(mAh)}{Corriente\ de\ salida\ máxima(mA)} = \frac{2200mAh}{800mA} = 2.75\ horas$$

En nuestro caso tarda en cargar cada batería 2 horas y 45 minutos aproximadamente.

2.7 Placa de distribución de potencia

Para realizar las interconexiones de alimentación entre la batería LIPO y los ESC (Speed controllers) se ha utilizado una placa de distribución de potencia que nos permite alimentar con el voltaje de 11.1 V que nos proporciona la batería para los ESC (Speed controllers), además nos permite conectar los 4 ESC a la vez utilizando esta placa de distribución de potencia.
Características de la placa de distribución de potencia:

- 6 conexiones para los ESC.
- Conector XT60 macho específico para aguantar elevada carga y potencia.
- Tamaño: 14.3* 7.4 * 0.7 cm.
- Peso del artículo: 10 g.

2.8 Hélices tripala

Las hélices que se han elegido para que nuestro dron vuele son las hélices tripala 5045, ya que están hecha de policarbonato que es un material duro y que aguanta ciertos golpes sobre las hélices, además el problema que se tenía con las hélices que viene con los motores es que solo funciona en un sentido de giro, mientras que nosotros necesitamos que nuestras hélices funcionen tanto en sentido horario como en sentido antihorario, con estas hélices disponemos de 2 que giran en sentido horario y otras dos que giran en sentido antihorario, se pueden distinguir ya que sobre las hélices viene serigrafiado el sentido de giro de la hélice para que tenga un buen funcionamiento. Además, las hélices que giran en sentido horario el nombre de la pala viene acompañado de una R, a la hora de funcionamiento para saber si tu pala está bien colocada y funciona correctamente el aire tiene que ir vertical hacia debajo de dron para que así este se pueda elevar.

Figura 10. Hélices tripala (Amazon,2023).

2.9 Componentes pasivos

A continuación, se enumeran todos los componentes pasivos que se han utilizado en nuestro cuadricóptero:

- Para hacer el divisor de tensión para conectar el voltaje de la batería al ESP32:
    - 1 diodo 1N4007.
    - 1 resistencia de 270KΩ.
    - 1 resistencia de 100KΩ.
- 1 diodo led rojo.

3 Esquemas electrónicos

3.1 Mando a distancia

Nuestro mando a distancia cuenta con dos joysticks, el primer joystick tiene un eje para indicar al dron el ángulo de pitch y otro eje para indicar el ángulo roll que debe tener en cada momento, el segundo joystick tiene un eje para cambiar el throttle(velocidad base de giro) que nos sirve para elevar el dron aumentando el throttle o para descender el dron bajando el throttle, el otro eje sirve para cambiar el yaw del dron, esto nos permite rotar el dron sobre su propio eje.

El mando también posee una pantalla para poder visualizar el ángulo de pitch, roll y yaw que tiene el dron en cada momento y el voltaje que tiene en ese instante la batería para controlar cuando se descarga esta y evitar que el dron pierda el control, el último componente de el mando es un led rojo que sirve para que cuando batería esté por debajo de 9V este se encienda y la persona que maneje el dron se dé cuenta que tiene que aterrizar el dron.

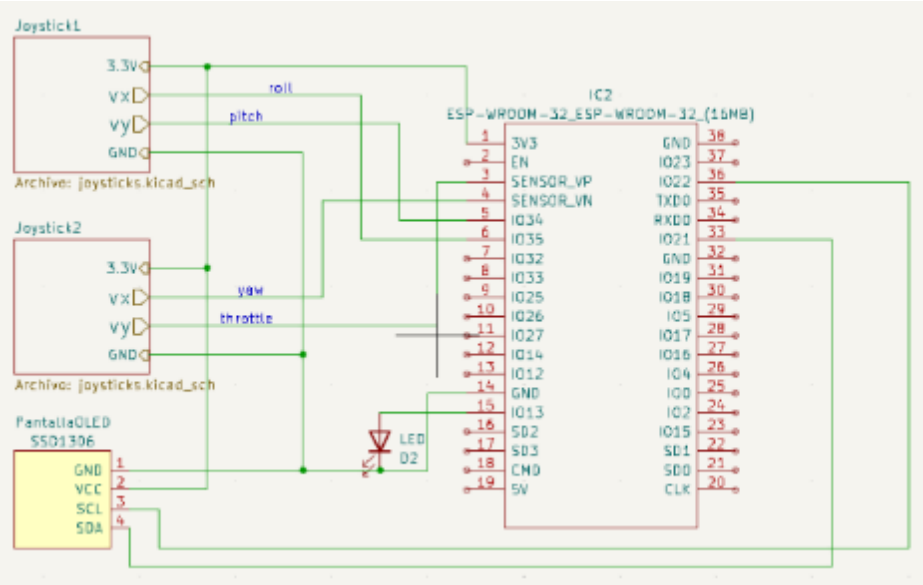


Figura 11. Esquema electrónico mando a distancia (realizado con KiCad).

Conexiones del mando:

- Para el primer joystick:
  - se conecta la alimentación del joystick a la patilla de alimentación de 3,3V del ESP32.
  - Se conecta la salida de voltaje del eje x del joystick que indica el ángulo de roll que queremos que tenga el dron en cada momento al GPIO 35 del ESP32 que actuará como ADC para convertir a digital el voltaje que sale del eje x del potenciómetro.
  - Se conecta la salida de voltaje del eje y del joystick que indica el ángulo de pitch que queremos que tenga el dron en cada momento al GPIO 34 del

ESP32 que actuará como ADC para convertir a digital el voltaje que sale del eje y del joystick.
    - La tierra del joystick se conecta a la tierra del ESP32.
- Para el segundo joystick:
    - se conecta la alimentación del joystick a la patilla de alimentación de 3,3V del ESP32.
    - Se conecta la salida de voltaje del eje x del joystick que indica el yaw que queremos que tenga el dron en cada momento al GPIO 39(SENSOR_VN) del ESP32 que actuará como ADC para convertir a digital el voltaje que sale del eje x del joystick.
    - Se conecta la salida de voltaje del eje y del joystick que indica el throttle que queremos que tenga el dron en cada momento al GPIO 36(Sensor_VP) del ESP32 que actuará como ADC para convertir a digital el voltaje que sale del eje y del joystick.
    - La tierra del joystick se conecta a la tierra del ESP32.
- Para la pantalla OLED:
    - se conecta la alimentación de la pantalla a la patilla de alimentación de 3,3V del ESP32.
    - Se conecta la patilla SCL(línea de la señal de reloj) al GPIO 22 del ESP32.
    - Se conecta la patilla SDA(línea de la señal de datos) al GPIO 21 del ESP32.
    - La tierra de la pantalla se conecta a la tierra del ESP32.
- Para el diodo led:
    - Se conecta el ánodo al GPIO 13 del ESP32, cuando esta patilla se pone a 3,3V se ilumina el led.
    - Se conecta el cátodo a la tierra del ESP32.

3.1.1 Joysticks

Cada joystick está compuesto de dos potenciómetros uno para el eje x y otro para el eje y, el funcionamiento es que en una patilla del extremo se conecta 3,3V y en la otra patilla del extremo se conecta la GND del ESP32, en la patilla de medio saldrá el valor del voltaje de salida de cada eje en función de cómo estemos presionando el joystick, si queremos el voltaje de salida máximo se moverá el joystick para tener la resistencia entre la salida y tierra del divisor de tensión del potenciómetro mucho más grande que la resistencia entre 3,3V y la salida de voltaje de ese eje, si queremos el voltaje de salida de 0 se moverá el joystick para tener la resistencia entre la salida y tierra del divisor de tensión del potenciómetro mucho más pequeña que la resistencia entre 3,3V y la salida de voltaje de ese eje, en cambio si queremos una salida de 1,65V que es el valor medio se mueve el joystick para poner la resistencia entre la salida y tierra del divisor de tensión del potenciómetro de el mismo valor que la resistencia entre 3,3V y la salida de voltaje de ese eje.

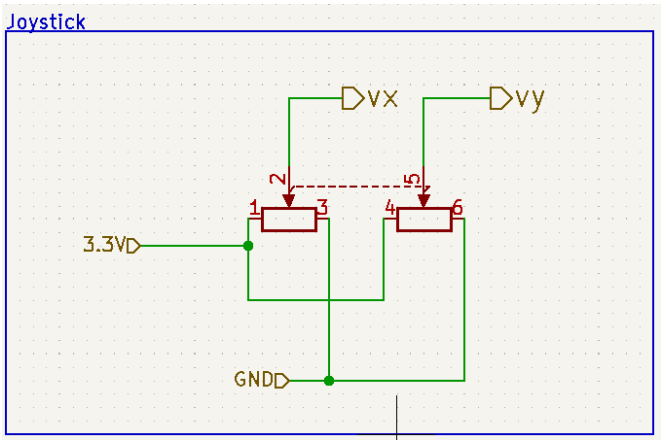


Figura 12. Esquema electrónico joystick (realizado con KiCad).

3.2 Dron

Nuestro dron cuenta con 4 motores brushless, también posee un circuito de potencia que engloba la batería y las salidas de la placa de distribución de potencia, además cuenta con el acelerómetro-giroscopio MPU6050 para medir los ángulos de pitch, roll y yaw, por último disponemos de un divisor de tensión junto con un diodo 1N4007, este circuito está conectado entre los 11,1V que nos proporciona la placa de distribución de potencia y la entrada del ESP32, luego tenemos la salida del divisor de tensión que va a tener un valor de 2,82V cuando la batería está cargada(tiene un voltaje de 11,1V) y 2,28V cuando la batería está cerca de estar descargada(tiene un voltaje de 9V), la función del diodo es evitar que la corriente vaya de cátodo a ánodo ya que si el valor del voltaje que hay en el cátodo es mayor que el voltaje del ánodo el diodo pasará a corte y no conducirá, por lo que siempre la corriente irá del ánodo al cátodo, es decir de la batería al ESP32, ya que el valor del voltaje que hay en el ánodo es mayor que el voltaje del cátodo, por lo que el diodo estará en conducción.

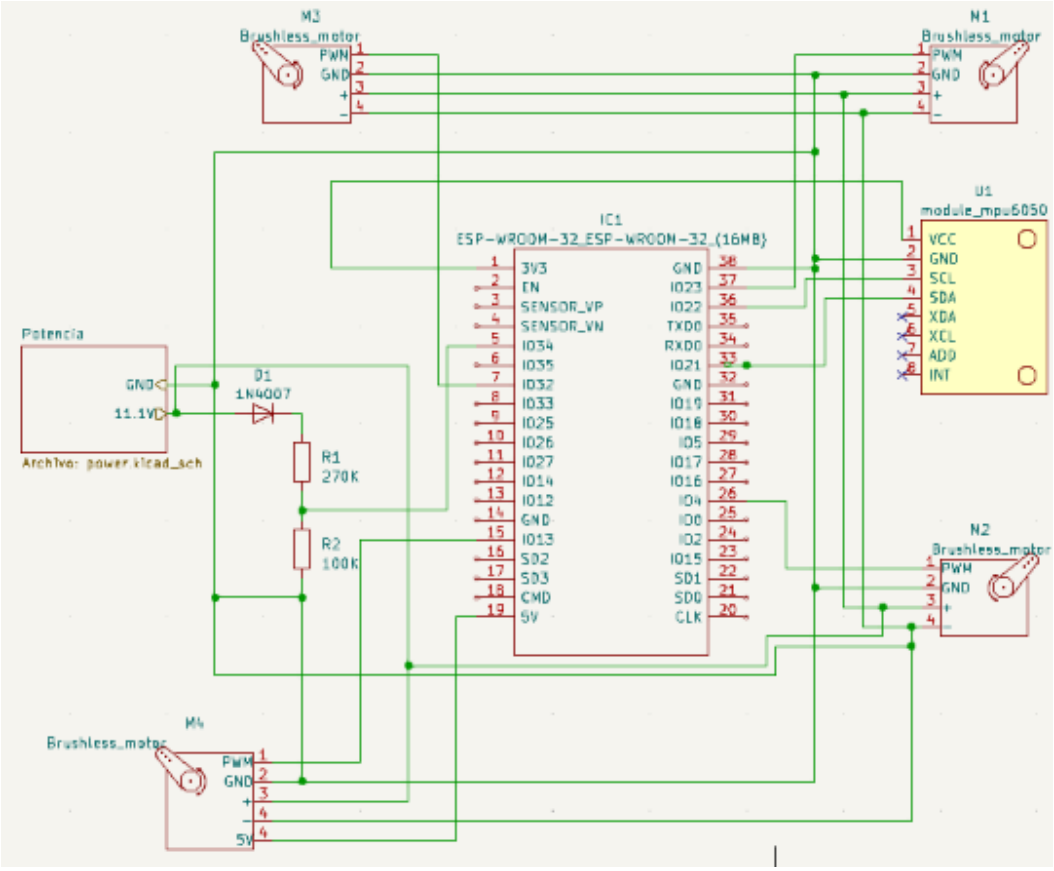


Figura 13. Esquema electrónico dron (realizado con KiCad).

Conexiones del Dron:

- Para la batería (circuito de potencia):
    - La conexión de 11,1V que nos proporciona la batería nos permite alimentar cada uno de los 4 motores.
    - La conexión GND del circuito de distribución de potencia se conecta al GND del ESP32 para que todas las GNDs de todos los dispositivos de nuestro circuito tengan la misma referencia.
- Para el motor Brushless 1:
    - La conexión por la que se introduce la señal PWM de 200Hz y ciclo de trabajo variable se conecta al GPIO 23 del ESP32.
    - La conexión GND de señal se conecta al GND del ESP32.
    - La alimentación positiva del motor se conecta a los 11,1V que nos proporciona el circuito de potencia.
    - La conexión negativa de alimentación se conecta al GND del circuito.
- Para el motor Brushless 2:
    - La conexión por la que se introduce la señal PWM de 200Hz y ciclo de trabajo variable se conecta al GPIO 4 del ESP32.
    - La conexión GND de señal se conecta al GND del ESP32.
    - La alimentación positiva del motor se conecta a los 11,1V que nos proporciona el circuito de potencia.

    - La conexión negativa de alimentación se conecta al GND del circuito
- Para el motor Brushless 3:
    - La conexión por la que se introduce la señal PWM de 200Hz y ciclo de trabajo variable se conecta al GPIO 32 del ESP32.
    - La conexión GND de señal se conecta al GND del ESP32.
    - La alimentación positiva del motor se conecta a los 11,1V que nos proporciona el circuito de potencia.
    - La conexión negativa de alimentación se conecta al GND del circuito.
- Para el motor Brushless 4:
    - La conexión por la que se introduce la señal PWM de 200Hz y ciclo de trabajo variable se conecta al GPIO 13 del ESP32.
    - La conexión GND de señal se conecta al GND del ESP32.
    - La alimentación positiva del motor se conecta a los 11,1V que nos proporciona el circuito de potencia.
    - La salida de 5V se conecta a la entrada de 5 voltios del ESP32 para alimentar a este.
    - La conexión negativa de alimentación se conecta al GND del circuito.
- Para el circuito que nos permite saber el voltaje de la batería en cada instante:
    - El diodo 1N4007 se conecta entre la salida de 11,1V del circuito de potencia y la entrada del divisor de tensión.
    - El divisor de tensión está compuesto por la resistencia en serie del diodo y la resistencia de 270KΩ en conjunto con la resistencia de 100KΩ, la salida de este divisor de tensión va al GPIO 34 del ESP32.
- Para el acelerómetro-giroscopio MPU6050:
    - se conecta la alimentación de la pantalla del MPU6050a la patilla de alimentación de 3,3V del ESP32.
    - Se conecta la patilla SCL(línea de la señal de reloj) al GPIO 22 del ESP32.
    - Se conecta la patilla SDA(línea de la señal de datos) al GPIO 21 del ESP32.
    - La tierra del MPU6050 se conecta a la tierra del ESP32.

Es necesario que todos los GNDs de los distintos dispositivos estén conectados entre sí para que tengan la misma referencia.

4 Protocolos de comunicaciones

4.1 ESP-NOW

El protocolo ESP-NOW es un protocolo de comunicaciones que permite enviar información en formato de tramas entre dos ESP32 de forma inalámbrica, este protocolo de comunicaciones fue creado por la empresa Espressif Systems que es la empresa que creó el ESP32 y el ESP8266.

El protocolo ESP-NOW se caracteriza por ser un protocolo de comunicación rápido ya que puede transmitir a una tasa de envío de datos de 1Mbps por defecto, es capaz de transmitir mensajes pequeños de hasta 250 bytes entre los dispositivos ESP32 que nosotros asociamos, es un protocolo que consume baja potencia y trabaja en la banda de frecuencias de 2,4GHz al igual que el protocolo de comunicaciones que la Wifi alliance llama WiFi 4 ya que trabaja en la banda de frecuencias de 2,4GHz.

Al trabajar en la banda de frecuencias de 2,4 GHz tiene el inconveniente de que ya existen otros protocolos de comunicaciones inalámbricas que tienen mayor velocidad como el Wifi 5 y el Wifi 6 que trabajan respectivamente en la banda de frecuencias de 5 y 6 GHz.

Este protocolo tiene la característica de que la conexión es segura y fiable ya que una vez que dos ESP32 se conectan la conexión es persistente, por lo tanto, cuando uno de los dos dispositivos se reinicie se volverán a conectar instantáneamente entre sí, además se caracteriza por ser un protocolo con muy bajo consumo de potencia.

El ESP-NOW utiliza como protocolo de seguridad el protocolo CCMP (Counter Mode with Cipher Block Chaining Message Authentitacion Code Protocol), este es un protocolo de encriptación usado para redes inalámbricas de área local. CCMP emplea claves de seguridad de 128 bits y un vector de inicialización de 48 bits para reducir la vulnerabilidad a que nuestro protocolo de comunicación sea atacado y robada la información enviada, el componente Counter Mode proporciona privacidad en los datos y el componente Cipher Block Chaining Message Authentication Code proporciona un servicio de autenticación de datos e integridad de envío de datos

La tecnología ESP-NOW admite las siguientes funciones:

- Comunicación unidireccional o bidireccional, y la información puede estar cifrada o no cifrada.
- Los dispositivos ESP32 pueden ser cifrados o no cifrados.
- Se puede transportar una carga útil (mensaje) de hasta 250 bytes.
- Es un protocolo que tiene implementado la función de asentimiento para indicar a la capa de aplicación de los ESP32 implicados en la comunicación que el mensaje que enviaron se recibió correctamente

El protocolo ESP-NOW tiene las siguientes limitaciones:

- El número máximo de pares cifrados en una comunicación es de 10 si se trabaja en el modo estación, es decir que se conecta a una red WiFi generada por un router, y de 6 si

se trabaja en el modo SoftAp, es decir que un ESP32 genera la señal Wi-Fi y los otros ESP32 acceden a esta red WiFi, es decir que un ESP32 va a actuar como router.

- El número de pares que intervienen en la comunicación como máximo puede ser 20 incluyendo los pares cifrados y no cifrados
- El tamaño útil del mensaje como máximo puede ser de 250 bytes.

El formato de las tramas que se envían con el protocolo ESP-NOW es el siguiente:

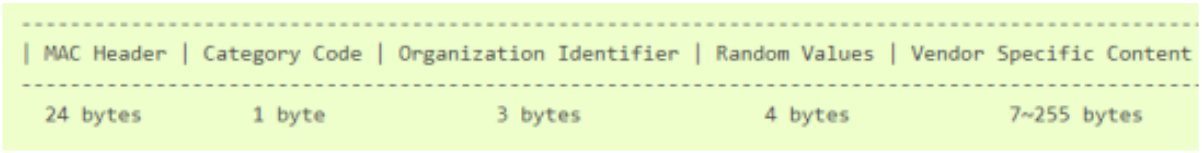


Figura 14. Formato tramas ESP-NOW (facultad de ciencias técnicas de Bitola,2023).

- Mac Header: en esta parte de la trama se envía la dirección MAC del ESP32 al que se envían los datos, tiene una longitud de 24 bytes y esta dirección viene especificada de fábrica para cada ESP32, sin está dirección MAC del receptor es imposible establecer la comunicación.
- Category Code: este campo normalmente tiene un valor de 127 que significa la categoría específica del emisor del mensaje, tiene una longitud de 1 byte.
- Organization identifier: este campo contiene el identificador 0x18fe34 que indica los tres primeros bytes de la dirección MAC del dispositivo que son iguales para todos los ESP32, este criterio fue estandarizado por la empresa ESpressif Systems, este campo ocupa 3 bytes.
- Random Value: este campo se utiliza para prevenir ataques de retransmisión, este campo ocupa 4 bytes.
- Vendor Specific content: contiene los bytes que envían el mensaje útil al receptor, las diferentes partes de este campo son:

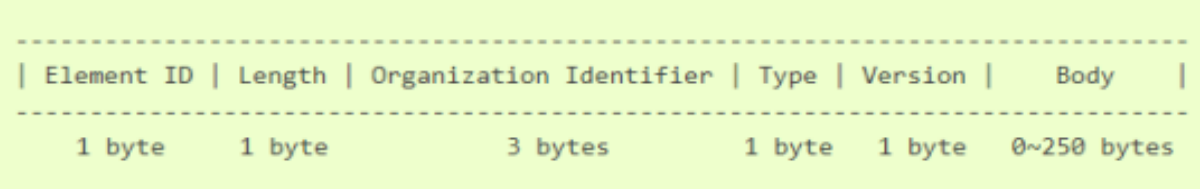


Figura 15. Campos del vendor specific content en las tramas ESP-NOW (facultad de ciencias técnicas de Bitola, 2023).

- Element ID: este campo está por defecto a 1 para indicar la dirección específica del emisor del mensaje, este campo tiene una longitud de 1 byte.
- Length: en este campo de 1 byte de longitud se incluye el tamaño de los campos Organization Identifier, Type, Version y el campo Body.
- Organization identifier: este campo contiene el identificador 0x18fe34 que indica los tres primeros bytes de la dirección MAC del dispositivo que son iguales para todos los ESP32, este criterio fue estandarizado por la empresa Espressif Systems, este campo ocupa 3 bytes.

- Type: este campo de 1 byte siempre transmite un 4 ya que indica que se está utilizando el protocolo ESP-NOW
- Version: este campo de 1 byte de longitud indica la versión del protocolo ESP-NOW que se está utilizando.
- Body: este campo contiene el mensaje útil, es decir la información que va a utilizar el receptor, puede tener una longitud máxima de 250 bytes.

El protocolo de comunicaciones ESP-NOW permite comunicación unidireccional en el que solo un ESP32(master) envía datos a otros ESP32(slaves) y estos no pueden enviar datos al máster, pero también existe comunicación bidireccional en la que todos los ESP32 pueden enviar información entre sí.

Figura 16. Comunicación unidireccional y bidireccional ESP-NOW (Randomn Tutorials,2023).

Otra ventaja de este protocolo es que es capaz de comunicar dos ESP32 hasta una distancia de 190 metros en campo abierto, por lo que no tendremos problemas para conectar nuestro mando y el dron, aunque se encuentren a una distancia alejada.

4.2 I2C

El protocolo I2C es un estándar diseñado por Philips que permite la comunicación bidireccional entre microcontroladores, memorias y otros dispositivos electrónicos con cierta complejidad, tiene una velocidad de 100 Kbps, pero hay ciertos modos como en el ESP32 el modo Fast Mode que transmite a una velocidad de 400 Kbps, I2C más modernos tienen velocidades de hasta 3 Mbps y 4 Mbps.

El bus I2C es un protocolo serie síncrono, esto significa que tiene una línea para transmitir la señal de reloj y otra línea para enviar los datos, estas son las distintas líneas del bus I2C:

- SCL es la línea que transmite los pulsos de la señal de reloj para que el master y el slave estén sincronizados a la hora de enviar los datos, el master es el dispositivo encargado de generar esta señal de reloj.
- SDA es la línea por la que se envían los datos entre el master y el Slave, el master es el que gestiona quién habla en cada momento en el bus I2C.
- VCC es la línea que se encarga de alimentar el dispositivo esclavo.
- GND es la línea que se encarga de interconectar las tierras del maestro y del esclavo.

En la siguiente imagen se puede ver los distintos dispositivos que se pueden conectar al ESP32 a través del bus I2C, en nuestro caso utilizaremos el MPU6050 para el dron con dirección de esclavo 0x68 y la pantalla OLED para el mando con dirección de esclavo 0x3C.

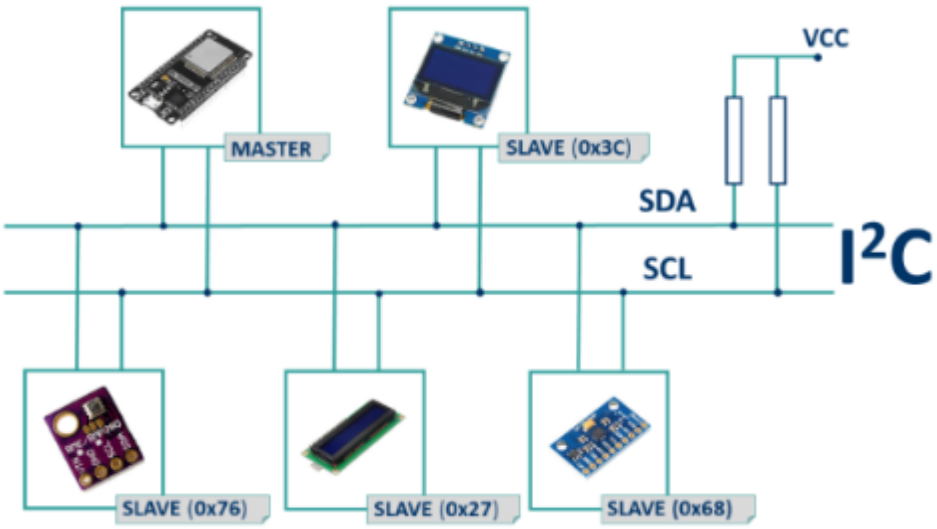


Figura 17. Comunicación maestro esclavo por protocolo I2C (Randomn Tutorials,2023).

Las líneas SDA y SCL están polarizadas en nivel lógico alto cuando estamos en un estado inactivo, para ello es necesario poner unas resistencias de pull-up cuyo valor puede variar entre 1.8 kΩ y 47 kΩ, para elegir qué valor de resistencia elegir hay que tener en cuenta que cuanto más pequeña es la resistencia mayor es el consumo de potencia de los integrados, pero en cambio disminuye la sensibilidad al ruido y mejora la respuesta temporal de nuestro circuito disminuyendo los retardos de las señales a lo largo de la línea.

4.2.1 Funcionamiento

Inicialmente el bus está libre, esto se produce cuando tanto la Línea SCL y SDA están en alta, luego el maestro que es el dispositivo encargado de establecer la comunicación envía la condición de start por lo que pone la línea SDA en nivel bajo y la línea SCL permanece en estado alto.

tras la condición de start se transmite un byte en el que los primeros 7 bits son la dirección del dispositivo esclavo con el que se va a comunicar el maestro y el octavo bit es el que indica si la operación que se va a realizar es de lectura o escritura, si es un 1 se indica que es operación de lectura y si es 0 es operación de escritura.

Si en el bus existe algún dispositivo con esa dirección de esclavo que se envió anteriormente por el bus se envía un ack en nivel bajo que representa que hay un dispositivo esclavo que acepta la solicitud y está disponible para comunicarse.

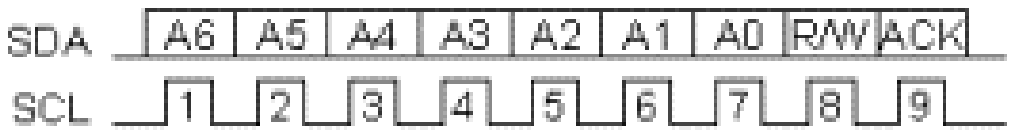


Figura 18. Trama protocolo I2C (Eduardo J.Carletti, 2023).

Si estamos en el modo escritura el maestro envía datos al esclavo de forma síncrona, cada pulso de reloj se envía un bit, cabe destacar que la línea de datos solo puede cambiar su valor cuando la línea de reloj está en baja, el esclavo va a leer todos los bits que se envíen y cada 8 bits(1 byte) envía un asentimiento de que se han recibido bien los datos, si este ack está en nivel bajo significa que el receptor ha recibido el mensaje correctamente y está listo para recibir el siguiente dato de 1 byte .

Cuando estamos en el modo de lectura el receptor envía los datos de forma síncrona con la señal de reloj, al igual que en el modo escritura cada 1 byte que transmite el emisor el receptor debe enviar un bit de ack para que el emisor sepa que todos los datos se han recibido correctamente, si este ack está en nivel bajo significa que el receptor ha recibido el mensaje correctamente y está listo para recibir el siguiente dato de 1 byte.

Cuando se termina la comunicación por que ya se han enviado todos los datos necesarios el máster genera una condición de parada, para indicar esta condición de parada pone la línea de datos y de reloj en nivel alto.

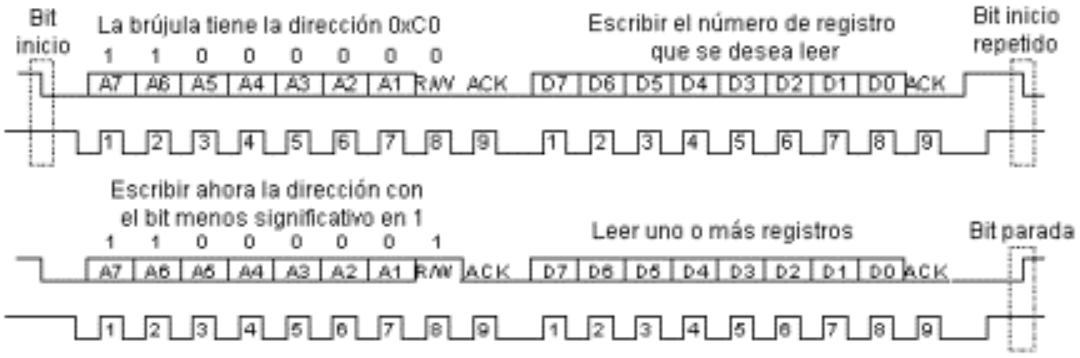


Figura 19. Funcionamiento protocolo I2C (Eduardo J.Carletti, 2023).

5 Diseño mecánico

Para realizar la base del dron se aprovechó material que había por el laboratorio para hacer el dron lo más barato posible, para ello todo lo que es la estructura de este se hizo con placas de cobre que se utilizan de normal para hacer placas de circuito impreso, al final la base central se hizo de una placa de fibra de carbono ya que es más resistente y pesa menos, por arriba de esta base central de fibra de carbono tenemos el ESP32 junto con el sensor MPU6050, mientras que por la parte de abajo tenemos todo el circuito de alimentación que consta de la batería y la placa de distribución de potencia, la cual sirve para alimentar cada uno de los cuatro motores, las medidas de la base central de fibra de carbono son 20cm de largo por 8,5cm de ancho.

En esta foto se puede ver cómo era la estructura más simple del dron en el que ya podía intuir la forma que iba a tener el dron.

Figura 20. Base central del dron.

Para las alas del dron se hizo una estructura rectangular también de cobre en la cual tenía que haber suficiente espacio de ancho para que entrase el motor y de largo para poder atar con cinta aislante los ESC de cada motor a las alas, luego para unirlo a la base central del dron se utilizaron unos tornillos de métrica 3mm de diámetro, al igual que para unir tanto el circuito del ESP32 con el MPU6050 como el circuito de distribución a la base del dron, con todo esto tendríamos la estructura del dron al

completo, en las medidas de la ala la dimensión más importante es el ancho para que pudiesen entrar los motores y el ESC, el ancho de cada hélice es de 4 cm.

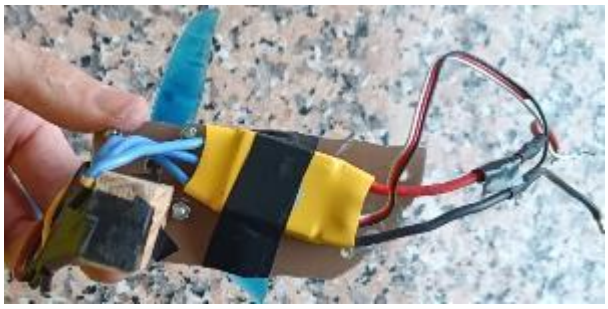


Figura 21. Parte trasera del ala del dron.

Figura 22. Parte delantera del ala del dron.

Con todo unido a la base central del dron la estructura final del dron aparece en la siguiente foto lista ya para volar, para evitar que los cables estén sueltos se puso cinta aislante para mantener fijos los cables a la base del dron.

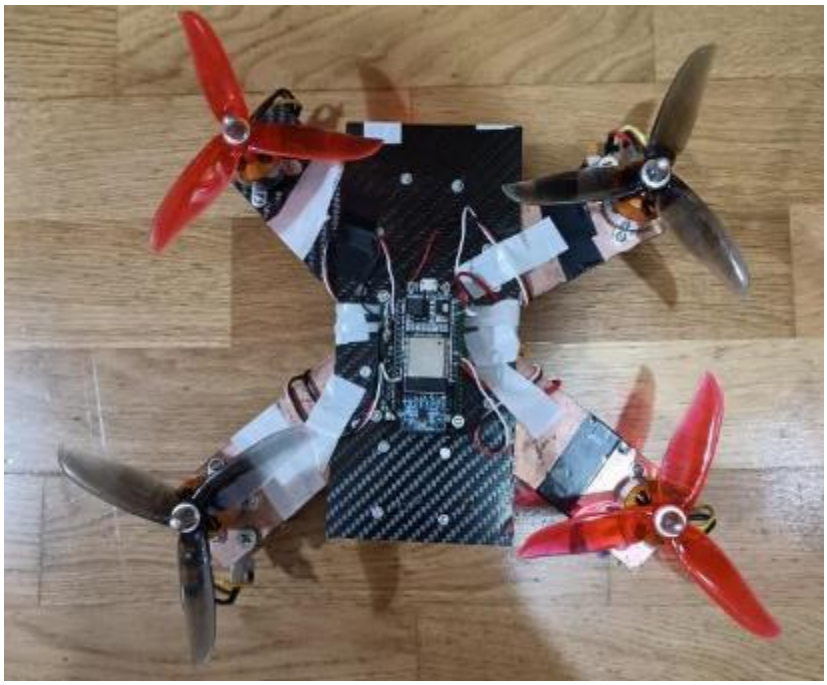


Figura 23. Estructura completa del dron.

6 Funcionamiento

6.1 Motores brushless 2200Kv y ESC(speed controllers) 30A

Los motores brushless constan de un rotor que posee una serie de imanes permanentes y un estator bobinado, gracias al campo magnético generado por los imanes nos permite mover el motor sin necesidad de utilizar las escobillas, de ahí su nombre motor sin escobillas, este motor cuenta de las siguientes partes, el rotor que es la parte móvil, mientras que el estator es la parte fija la cual cuenta con tres bobinados que son las tres fases del motor.

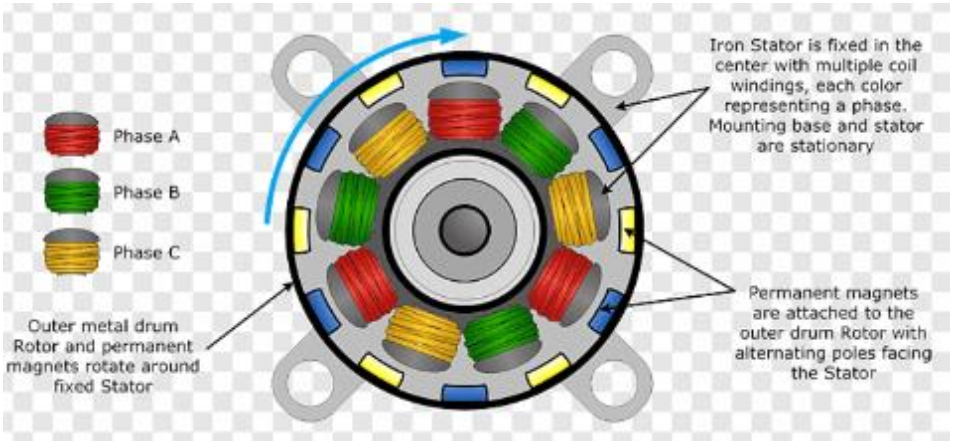


Figura 24. Esquema de motor brushless (pngegg, 2023).

El funcionamiento de los motores brushless consiste en que al energizar las bobinas del estator estas crean un campo magnético en el interior. El rotor posee un campo magnético constante que es capaz de detectarla variación del campo magnético creado por el estator, el rotor lo que hace es que iguala el campo magnético creado por el estator con el propio del rotor, para hacer esto el rotor girara en la dirección en la que cumpla lo dicho anteriormente. Para que el movimiento del rotor sea constante lo que se hace es que se energiza la bobina siguiente a la que se activó anteriormente y esta se deja de alimentar, gracias a que el rotor va a seguir al campo magnético del estator vamos a poder hacer que el motor gire sin necesidad de escobillas.

Para que las bobinas del estator generen el campo magnético necesario para que el motor gire es necesario que nuestro ESP32 genere una señal PWM que envíe al ESC el cual está formado por un puente de transistores en H, estos pulsos van a encender y apagar los transistores del puente en H, con este funcionamiento podemos hacer que la tensión en cada una de las tres bobinas del motor varíe creando un campo magnético giratorio que puede ir cambiando de velocidad en función del ciclo de trabajo de la señal PWM de una frecuencia fija que le enviamos desde el ESP32, los ESC que nosotros vamos a utilizar tienen una corriente máxima permitida de 45A el ESC además dispondrá de un condensador para filtrar las señales de alta frecuencia y eliminar el ruido en la señal PWM que se envía desde el ESP32, gracias al ESC a partir de la señal

PWM que recibe se generan una serie de pulsos desfasados 120 grados de la misma frecuencia que la señal PWM, con todo esto los motores brushless son capaces de girar con la velocidad que le especifiquemos en el ciclo de trabajo de la señal PWM.

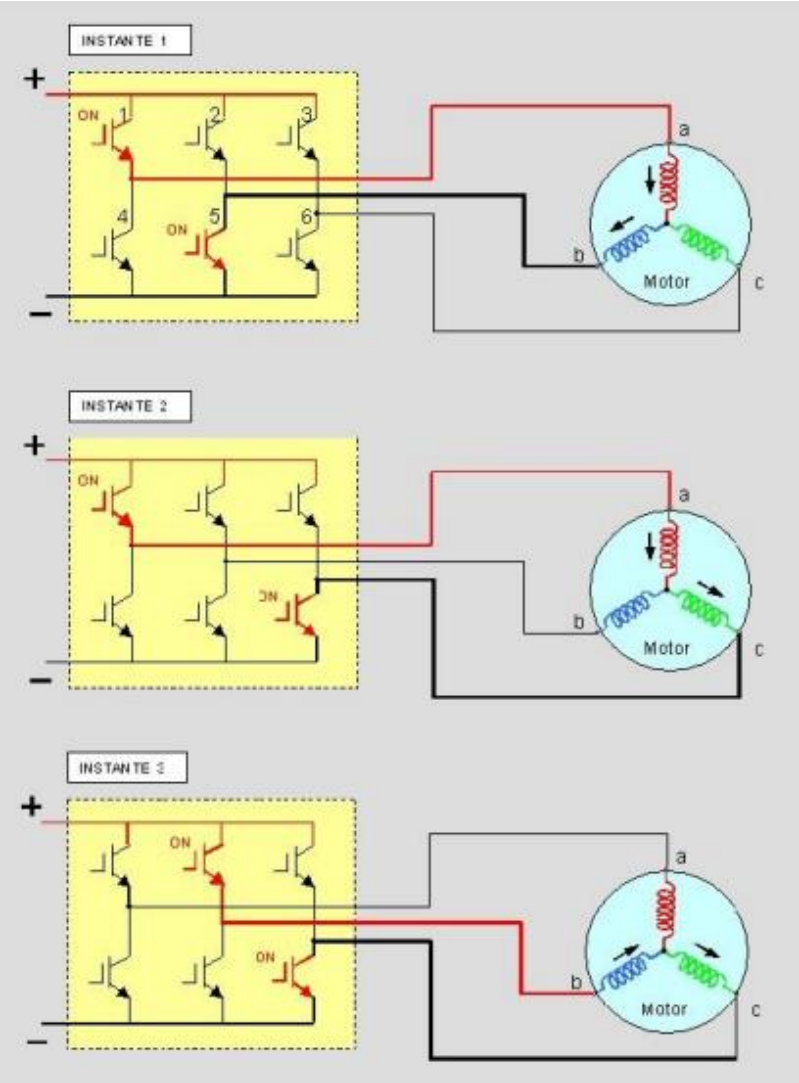


Figura 25. Figura1 polarización motor brushless (Gonzalo Solchaga Pérez de Lazárraga, 2023).

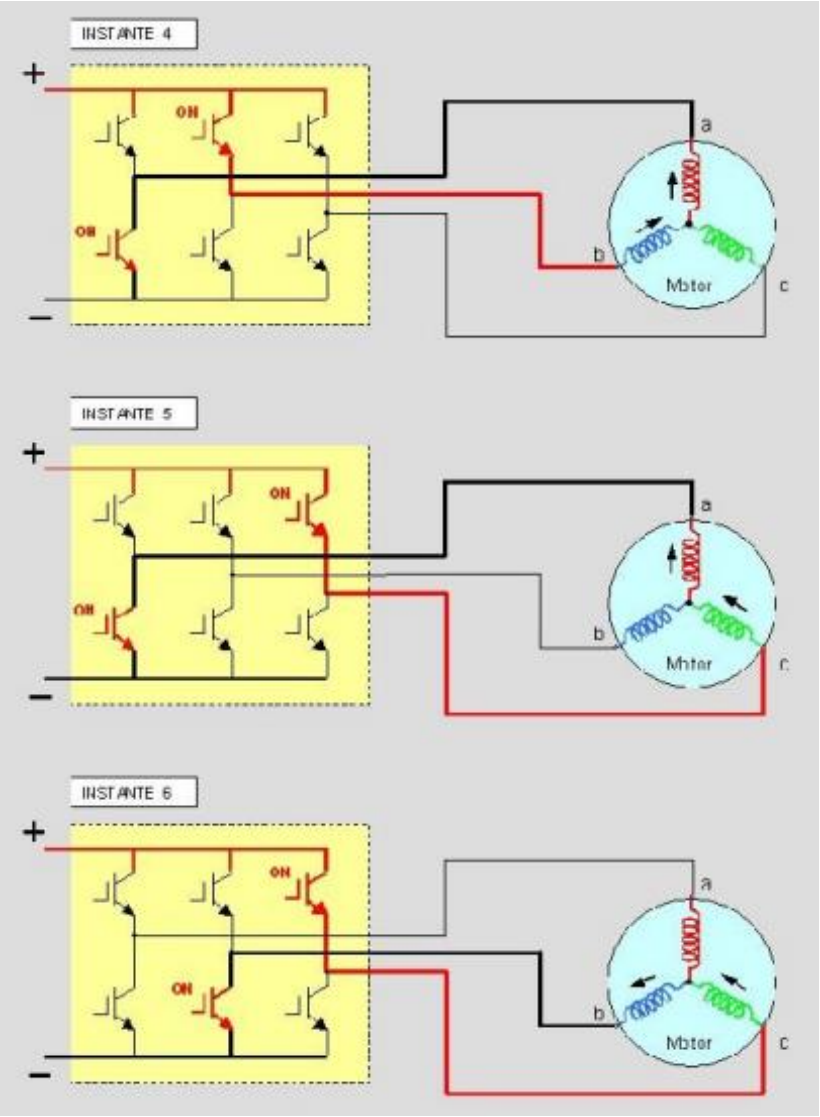


Figura 26. Figura2 polarización motor brushless (Gonzalo Solchaga Pérez de Lazárraga, 2023).

Para caracterizar la velocidad que lleva nuestro motor existe una fórmula que relaciona la velocidad en revoluciones por minuto y el valor medio de la tensión generada en la señal PWM que enviamos desde el ESP32, por eso cuando aumentamos el ciclo de trabajo de la señal PWM aumenta la velocidad de los motores, ya que aumenta el valor medio de la tensión de la señal PWM, la velocidad y la tensión se relacionan gracias a un coeficiente que depende de cada motor, cuyas siglas son kv.

$$velocidad(RPM) = kv(RPM/V) * valor\ medio\ de\ la\ tensión\ de\ entrada(V)$$

En el motor que se ha elegido en este trabajo tiene una kv de 2200RPM/V, ya con este valor y en función del valor medio de nuestra señal PWM podemos calcular la velocidad a la que gira nuestro motor, por ejemplo, si el valor medio de voltaje de nuestra señal PWM es 0,5V la velocidad será de 1100RPM.

6.2 Sensor MPU6050

El sensor MPU6050 es un sistema de medición inercial con 6 grados de libertad ya que combina un acelerómetro de 3 ejes y un giroscopio de 3 ejes, los 3 ejes son el eje x, el eje y y el eje z, este sensor nos servirá para medir la aceleración y la velocidad angular de nuestro dron, y a partir de estos valores obtendremos los ángulos de pitch, roll y yaw del dron, cabe destacar que el MPU6050 se comunica con el ESP32 mediante el protocolo I2C.

Para medir la aceleración el acelerómetro de este sensor utiliza la fórmula de la segunda ley de newton $a = F/M$. El acelerómetro posee un MEMS (Micro Electro Mechanical Systems), que nos permite medir la aceleración de forma similar a un sistema de masa resorte, habrá uno de estos en cada uno de los distintos ejes tanto para el x como el y como el z, cabe destacar que aunque no haya ningún peso sobre el eje z siempre tendremos una aceleración de 9.8 m/ s^2 debido a la fuerza de la gravedad.

Figura 27. Modelo acelerómetro del MPU6050(Naylamp Mechatronics, 2023).

Para medir la velocidad angular (tasa de desplazamiento angular por unidad de tiempo) utilizamos el giroscopio basado en el efecto Coriolis, este efecto se basa en establecer un sistema de referencia fijo y a partir de este sistema de referencia medir la aceleración de nuestro objeto respecto a este sistema de referencia, esta aceleración va a ser siempre perpendicular al eje de rotación del sistema y a las componentes radial y tangencial de la velocidad que posee nuestro cuerpo.

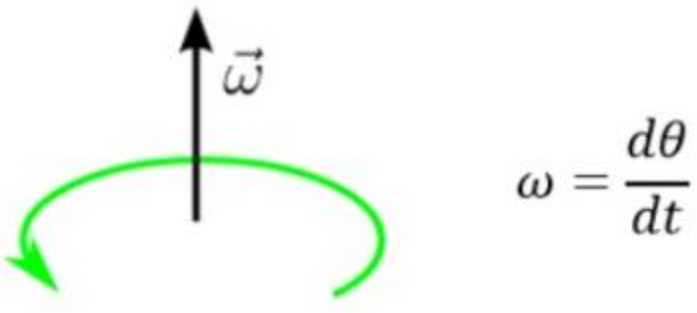


Figura 28. Funcionamiento giroscopio del MPU6050(Naylamp Mechatronics, 2023).

Tal y como se dijo antes el MPU6050 tiene un giroscopio de tres ejes que nos permite medir la velocidad angular y un acelerómetro de tres ejes que nos permite medir la aceleración de nuestro dron, en la siguiente foto se puede observar hacia donde es el signo positivo tanto de la aceleración como de la velocidad angular en cada eje:

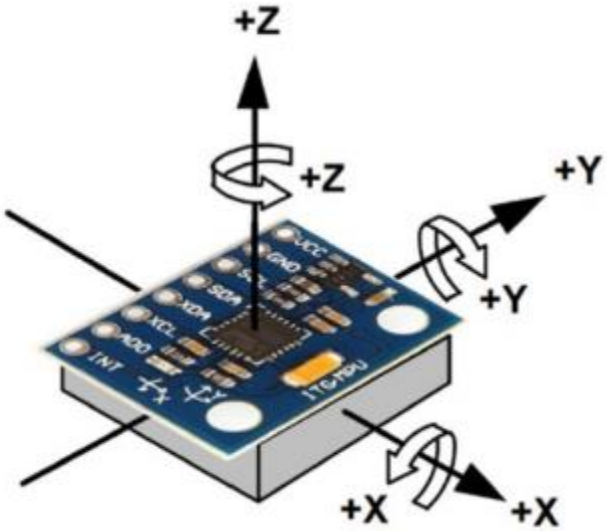


Figura 29. Sentido velocidad angular y aceleración del MPU6050 (Randomn Tutorials,2023).

Para un buen funcionamiento del acelerómetro es necesario configurar el rango de medición del acelerómetro, los rangos de medición son +-2G, +-4G, +-8G y +-16G, en nuestro caso para obtener una buena sensibilidad y resolución a la hora de medir la aceleración se ha escogido el rango de +-8G, para el giroscopio los rangos de medición son +-250dps, +-500dps, +-1000dps y +-2000dps, en nuestro caso se ha elegido para tener una buena resolución el rango de +-500dps, el MPU6050 es un sistema digital, ya que convierte los valores analógicos del giroscopio y del acelerómetro mediante un conversor analógico/digital de 16 bits, en nuestro caso al estar trabajando en el rango del giroscopio de +-500dps cuando tenemos un valor digital de 65536 significa que la velocidad angular es de 500dps, además es necesario configurar un filtro paso bajo digital que nos permite eliminar el ruido que tiene tanto el acelerómetro como el giroscopio del MPU6050, en nuestro caso se ha tenido que poner un filtro paso bajo digital de una frecuencia de corte de 21 Hz, esta es una frecuencia de corte bastante restrictiva que nos permite

eliminar el ruido a frecuencias altas y así evitar las derivas en los valores del giroscopio y del acelerómetro, básicamente el funcionamiento de este filtro paso bajo es de hacer una media de los valores que obtiene el giroscopio y el acelerómetro y así evitar fluctuaciones en el cálculo de la inclinación del pitch, el roll y el yaw producidas por el ruido.

6.3 Mando a distancia

Para el mando distancia se ha utilizado un ESP32 que se va a comunicar con el otro ESP32 del dron para disponer de una comunicación bidireccional en la que el dron envíe el ángulo de inclinación del dron, del pitch y del yaw, mientras que el ESP32 del mando a distancia envía la salida de voltaje de cada uno de los joysticks en el que el eje vertical del joystick sirve para cambiar el valor del throttle, el eje horizontal del joystick izquierdo sirve para cambiar el yaw del dron, el eje vertical del joystick derecho sirve para cambiar el roll del dron y por último el eje horizontal de este dron sirve para cambiar el picth del dron. Todas las salidas de este voltaje van a unos pines de entrada del ESP32 que son los pines 34,35,36 y 39 que mediante el conversor analógico/digital de 12 bits nos convierte el valor analógico de salida que nos da el potenciómetro en función de donde se encuentre el joystick colocado en digital para poder ser interpretado por este ESP32, los convertidores analógicos/digitales convierten el valor analógico de 0 a 3,3V en un valor digital de 0 a 4095, la función de transferencia del ESP32 es prácticamente lineal menos en la zona de 0 a 0,1 V y 3,2 a 3,3V en la que el comportamiento del ADC es no lineal(zona de saturación) tal y como se puede ver en la siguiente foto:

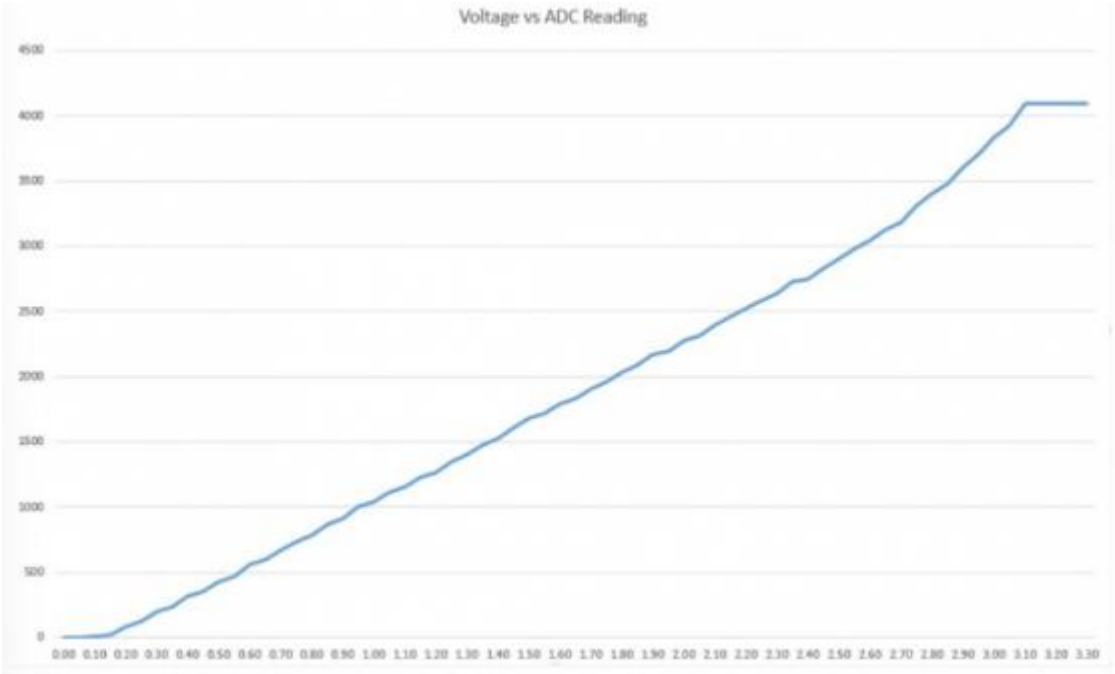


Figura 30. Gráfica valor digital de ADC del ESP32 respecto al voltaje de entrada (Randomn Tutorials,2023).

Además, el mando posee una pantalla OLED que nos permite visualizar por pantalla el ángulo de pitch, roll y yaw del dron en cada instante, además podemos visualizar el voltaje que posee la batería del dron en cada instante para saber si nos queda suficiente batería para que los motores

no se paren, la comunicación entre el ESP32 y la pantalla OLED se realiza mediante el protocolo I2C.
El último componente que tenemos en el mando es un diodo led que se enciende cuando se pone una tensión positiva en el ánodo y tierra en el cátodo, lo que se hace es que cuando el voltaje de la batería baja de un valor de 9 voltios este Led rojo se enciende poniendo un 1 digital en el pin 13 del ESP32, es decir 3,3V en la salida del pin 13 que va conectado al ánodo del led para indicar que la batería está prácticamente descargada.

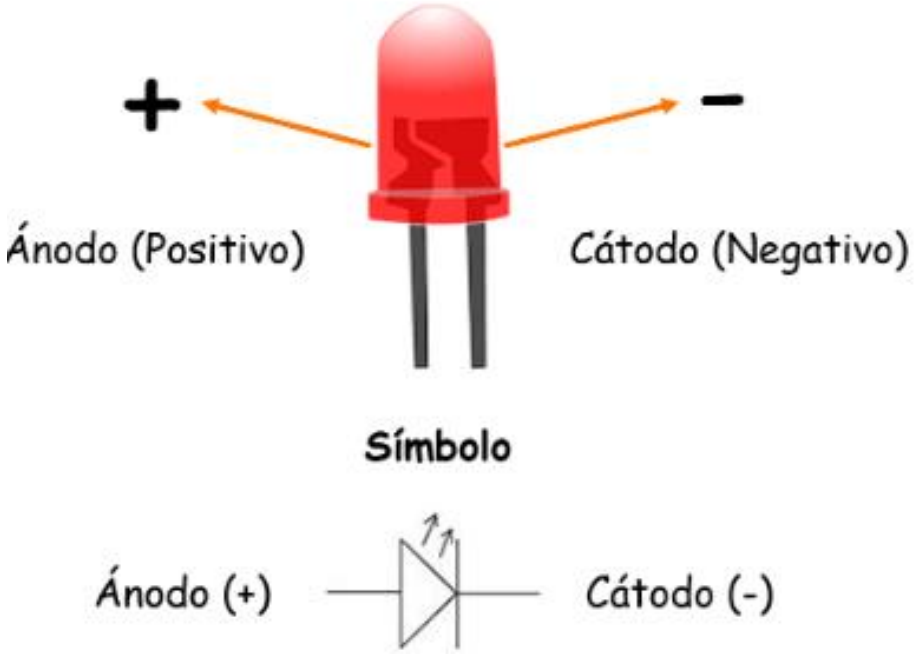


Figura 31. Representación y esquemático diodo led (Wikipedia, 2023).

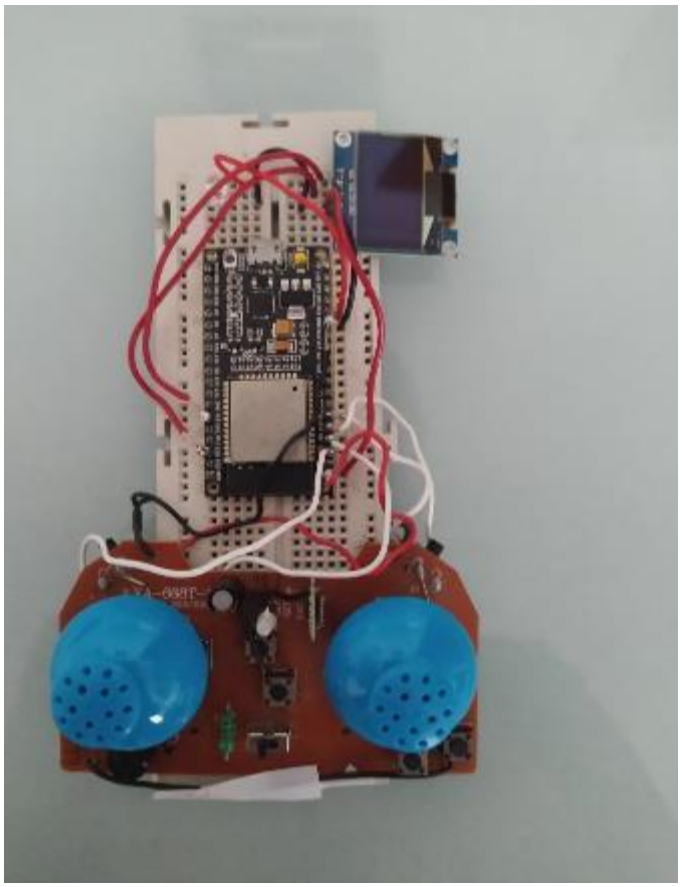


Figura 32. Foto del montaje de el mando a distancia.

6.4 Control PID

Una parte muy importante para mantener el dron estable es la utilización de un PID. Los PIDs nos sirven para indicar cómo de rápido deben girar los motores para conseguir una elevada estabilidad en este.

El PID es un sistema en lazo cerrado con realimentación negativa que mediante una serie de cálculos nos permite conseguir que la variable de entrada de nuestro sistema (salida del sensor de MPU6050) sea lo más parecido posible al valor de referencia del PID para que el error sea 0, siempre estamos comparando el ángulo que mide en cada eje el MPU6050 y el valor de referencia angular que tenga nuestro dron en cada eje, luego este error se pasa por el PID y se aplica a los motores la salida del PID para que el dron se mantenga estable y próximo al valor de referencia que se le envía desde el mando.

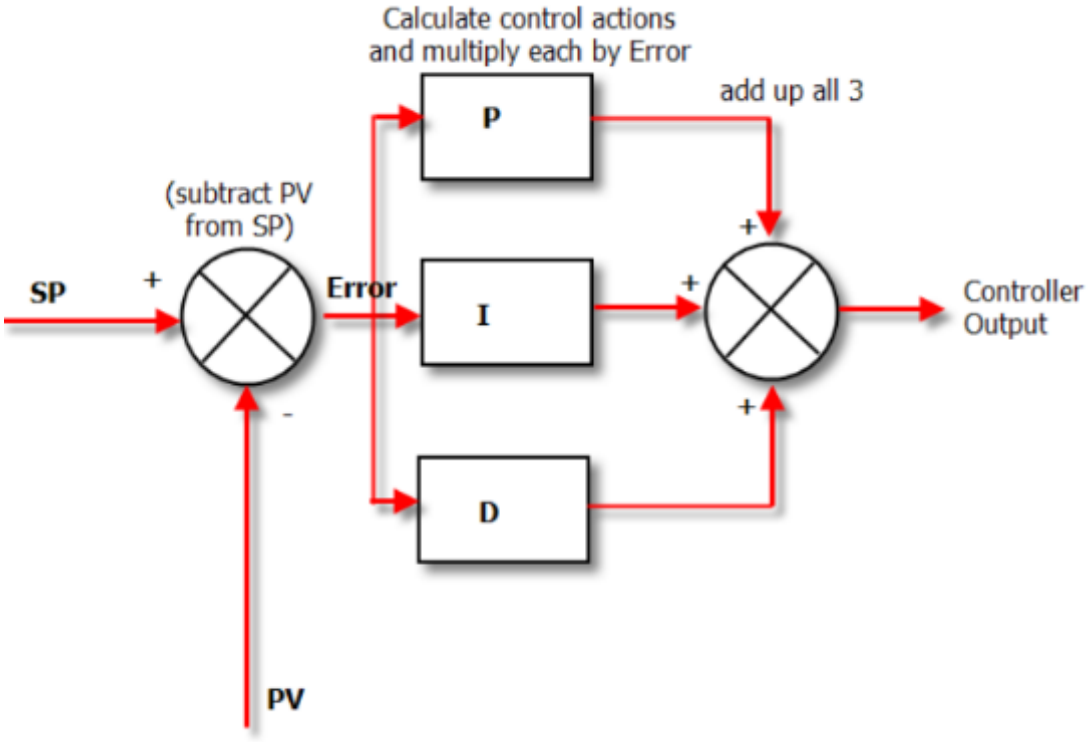


Figura 33. Diagrama del PID (FpvMax, 2023).

Los distintos componentes del PID son la parte proporcional, la parte derivativa y la parte integral:

La parte proporcional consta del producto de la señal de error y la constante proporcional que se va introduciendo, nos sirve para lograr que el error estacionario se aproxime a cero tras unas oscilaciones, cuanto mayor sea el valor del parte proporcional más rápido el error tenderá a cero, pero si ponemos una constante proporcional muy grande se producirá el fenómeno llamado sobre oscilación y nuestro sistema empezará a oscilar cada vez más y más.

$$parteproporcional = k_p\left(salida_{sensor} - consigna_{referencia}\right)$$

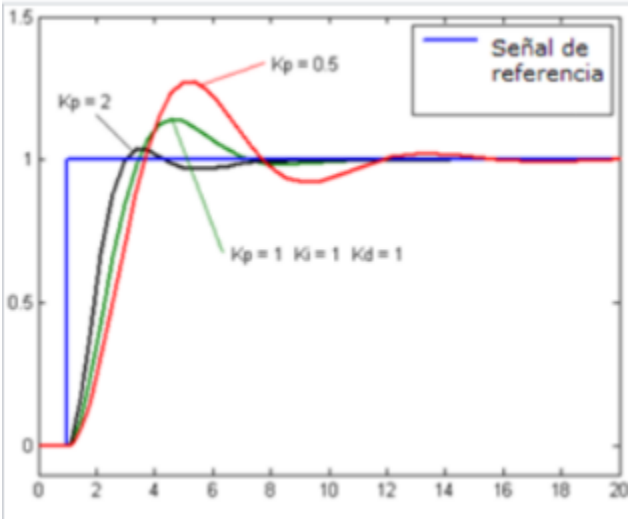


Figura 34. Señal de salida para distintos valores de kp (Wikipedia, 2023).

La parte integral tiene el objetivo de acumular el error que va teniendo nuestro sistema para hacer que el error que tiene nuestro PID sea 0, principalmente es que nos sirve para corregir perturbaciones externas que aparecen en un determinado momento, el control integral actúa cuando hay una diferencia entre el valor de referencia y el valor del sensor que se utilice, se integra este error y se suma a la parte proporcional. Cuando se integra el error se le multiplica por la constante integral, la parte integral tiene la función de promediar el error durante un periodo de tiempo que en nuestro caso va a ser constante de 5 milisegundos, hay que evitar poner la constante integral muy elevada ya que puede producir una elevada oscilación en nuestro sistema.

$$pi = \int_0^t k_i \left(salida_{sensor} - consigna_{referencia}\right) dt$$

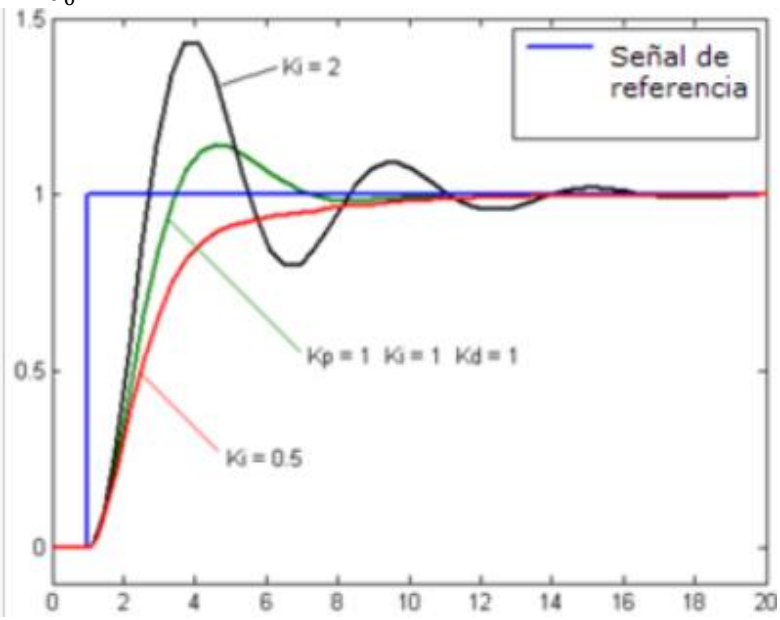


Figura 35. Señal de salida para distintos valores de ki (Wikipedia, 2023).

La acción derivativa solo aparece cuando hay un cambio absoluto del error, cuanto mayor sea este cambio mayor será el efecto de la parte derivativa, la principal función de la parte derivativa es mantener el error al mínimo para corregir proporcionalmente la velocidad para que nuestro dron se mantenga estable. La función matemática consiste en que se deriva el error respecto al tiempo y luego se multiplica por la constante derivativa, en la siguiente figura se puede observar como una mayor constante derivativa produce que la respuesta del sistema sea más suave, pero si nos pasamos demasiado podemos tener una gran inestabilidad en el proceso, mientras que si el valor es muy bajo se producirán muchas oscilaciones en nuestro sistema producido por la parte proporcional e integral.

$$pd = k_d * \frac{d(salida_{sensor} - consigna_{referencia})}{dt}$$

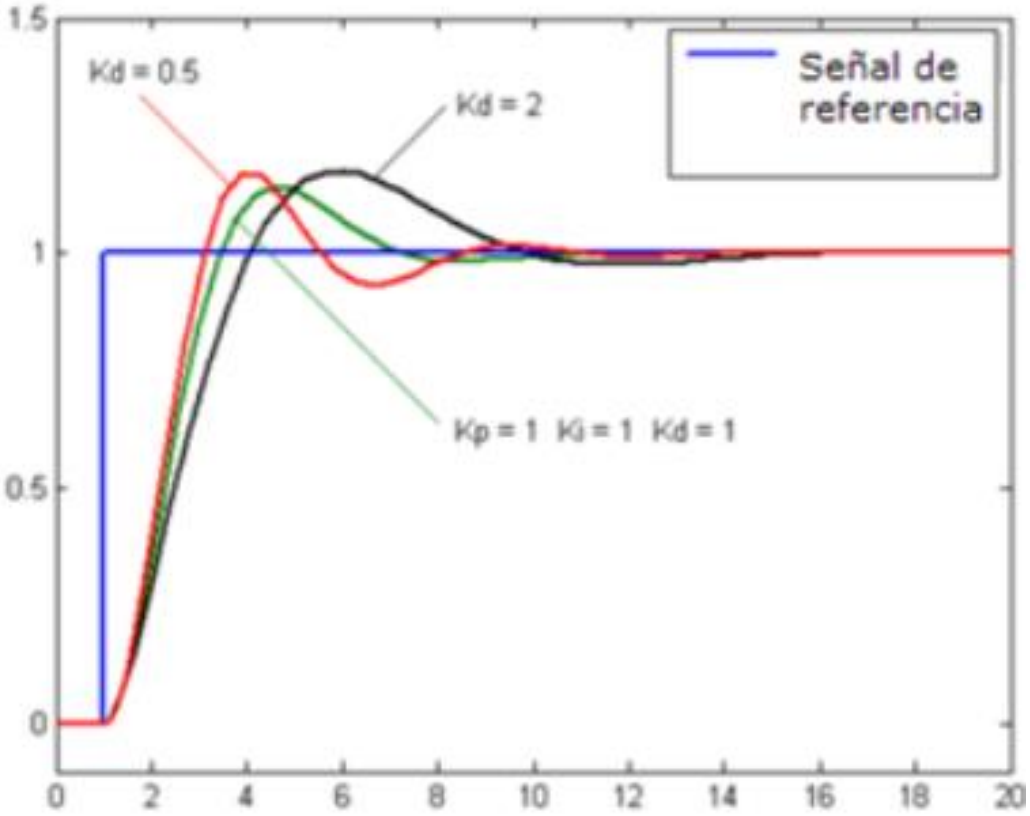


Figura 36. Señal de salida para distintos valores de kd (Wikipedia, 2023).

6.5 Voltaje batería

Para calcular el voltaje que tiene la batería en cada momento y así poder saber si podemos seguir utilizando esta batería para mover los motores o si necesito cargar la batería para poder volver a controlar los motores, de nuestro circuito de potencia sale un voltaje de 11,1V cuando la batería está completamente cargada, luego este valor va bajando y pasa por el diodo 1N4007 que lo que nos permite es que la corriente circule solo en un sentido que es desde el circuito de potencia al ESP32 y evitar que por cualquier motivo la corriente vaya en sentido contrario y pueda fastidiar la batería, luego además tanto el diodo 1N4007 junto con la resistencia de 270KΩ forman un divisor de tensión junto a la resistencia de 100KΩ, cuando el diodo conduce su resistencia es muy pequeña por lo que se considera despreciable respecto a las otras, pero el valor real de voltaje de entrada al ESP32 va a ser menor al que se va a calcular ahora por el efecto de la resistencia del diodo en conducción, para saber el voltaje que entra a la patilla del ESP32 se utiliza la siguiente fórmula:

$$Voltaje_{entradaESP32} = voltaje_{salidabat} * \frac{R1}{R1 + R2 + Z_{diodo}}$$

$$= voltaje_{salidabat} * \frac{100K\Omega}{100K\Omega + 270K\Omega + 0K\Omega}$$

Para un valor de voltaje de salida del circuito de potencia de 11.1 voltios (batería cargada) tenemos un voltaje a la entrada del conversor analógico/digital del pin del ESP32 de 3V y cuando consideramos que la batería ya no puede mover los motores con fluidez es cuando el

voltaje de salida del circuito de potencia es inferior a 9V, para este valor el voltaje de entrada al conversor analógico/digital es de 2,43V.

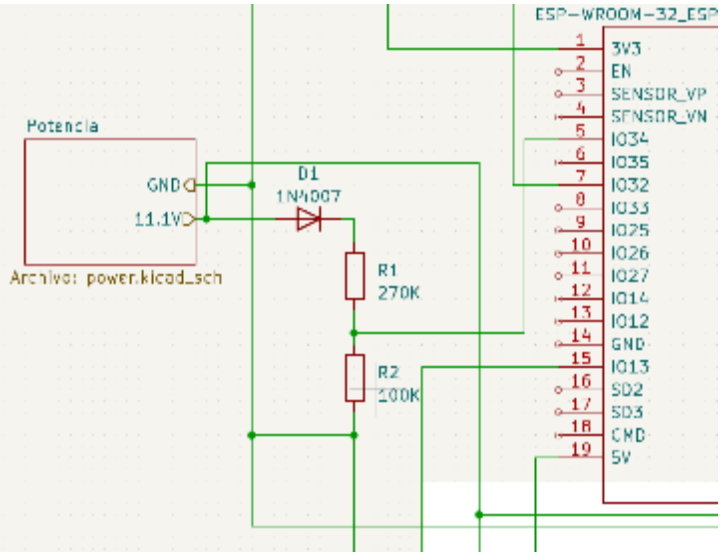


Figura 37. Esquema electrónico para medir el voltaje de la batería.

6.6 Filtro paso bajo digital

A lo largo del proyecto se han utilizado muchos filtros paso bajo digitales, los cuales se han utilizado para reducir el ruido en frecuencias altas, por ejemplo, se ha utilizado para reducir el ruido de la señal que nos devuelve el voltaje de la batería, o para reducir el ruido en alta frecuencia de las señales que nos envía el sensor MPU6050. La fórmula del filtro paso bajo digital es la siguiente:

$$A_n = \alpha M + (1 - \alpha) A_{n-1}$$

Figura 38. Formula filtro paso bajo digital (Luis Lamas, *2023*).

En función del valor de α que se le ponga tendremos un filtro paso bajo de cierta frecuencia de corte, si se pone α pequeña se tendrá menor frecuencia de corte que con un α más alta, es decir más restrictivo, solo que también menos α implica mayor retardo ya que si la frecuencia de corte es más restrictiva entonces el producto R*C es mayor, en consecuencia el retardo que produce el filtrado en nuestra señal es mayor, en las siguientes imágenes se puede observar todo lo explicado en este párrafo.

Con α =0,6 se puede ver como se ha eliminado cierto ruido en alta frecuencia, aunque aún se observa como la señal filtrada tiene cierto ruido, pero lo bueno es que el retardo que introduce la señal filtrada es mínimo.

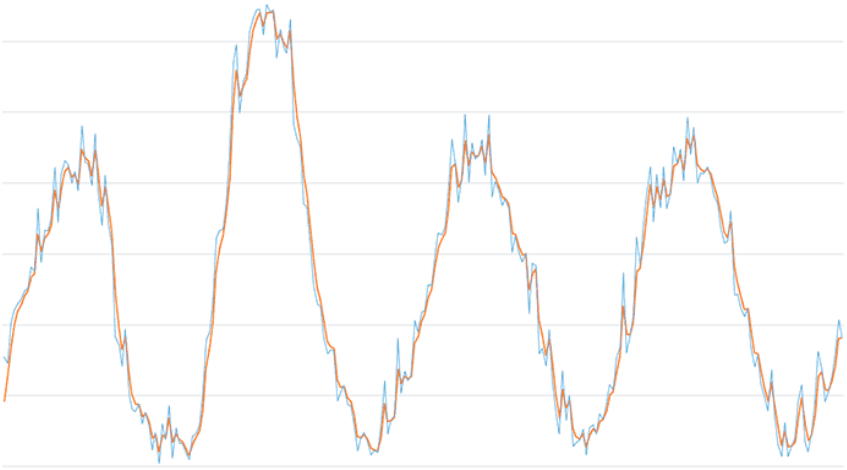


Figura 39. Señal de salida con α =0,6 (Luis Lamas, *2023*).

Con α =0.2 se puede observar cómo se ha eliminado gran parte del ruido en alta frecuencia, pero tenemos un retardo considerable entre la señal de entrada y la señal filtrada.

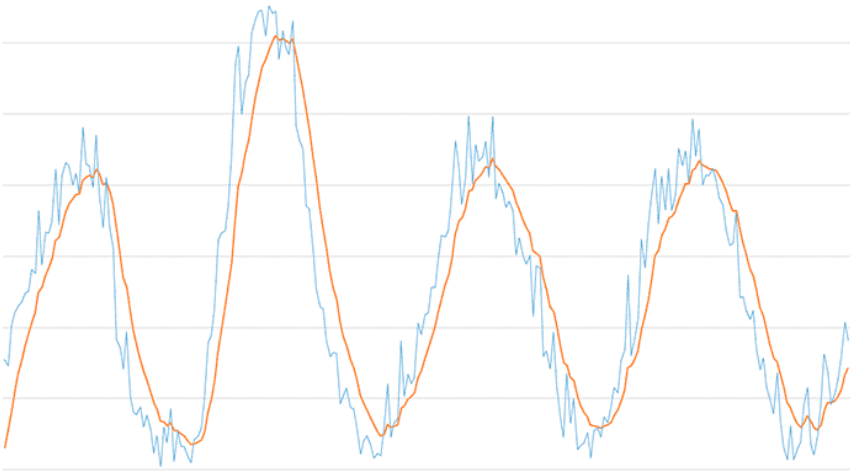


Figura 40. Señal de salida con α =0,2 (Luis Lamas, *2023*).

Con α =0.05 solo tenemos la componente principal de la señal por lo que se ha eliminado todo el ruido en alta frecuencia, pero el retardo que se produce entre la señal de entrada y la señal filtrada es muy considerable.

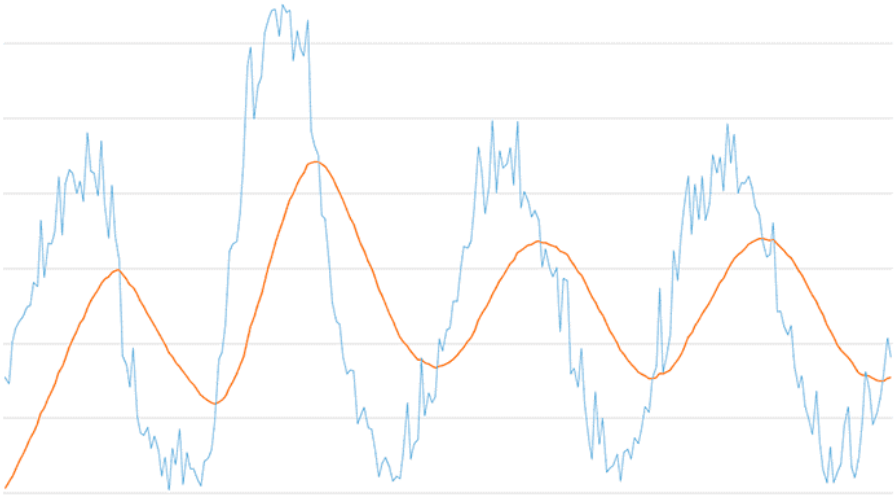


Figura 41. Señal de salida con α =0,05 (Luis Lamas, *2023*).

6.7 Calibración de hélices

Uno de los principales problemas que se tiene a la hora de volar el dron son las vibraciones producidas tanto por los motores como por las hélices por lo que es necesario realizar un calibrado de ambas, para los motores la única calibración que se puede hacer es ajustarlos bien con los tornillos al chasis de dron, pero para las hélices se puede realizar un calibrado muy preciso haciendo que las tres palas de nuestras hélices tripala pesen todas lo mismo y están perfectamente compensadas la una con la otra procurando que se produzcan las mínimas vibraciones sobre el chasis de nuestro dron y que afecten a las medidas del MPU6050.
Para calibrar las hélices se utiliza un demagogo que este es un dispositivo que tiene una barra de metal en la cual se ponen las hélices y en los extremos tenemos un imán que nos permite que a través del campo magnético que genera ese imán la barra de metal quede fija y al poner las hélices en la barra esta hélice girará hacia un lado si esa pala pesa más que las otras, si esto pasa se pone algo de cinta aislante en la pala del lado contrario para así compensar el peso que tiene la otra pala, hay que hacer esto varias veces hasta que la hélice se quede estable en el medio, cuando esto pase significa que nuestra hélice está perfectamente equilibrada y por lo tanto habremos reducido al mínimo las vibraciones producida por las hélices.

Figura 42. Demagogo(Amazon, 2023).

7 Software

7.1 Inicializar sensor MPU6050

Para utilizar el acelerómetro Giroscopio MPU6050 con nuestro ESP32 es necesario instalar tres librerías en arduino llamada “Adafruit_MPU6050.h”, “Adafruit_Sensor.h” y “Wire.h”, las dos primeras sirven para ajustar el rango de funcionamiento del giroscopio y del acelerómetro, además nos permite obtener los parámetros que nos devuelve el sensor MPU6050, es necesario poner la librería Wire.h ya que es la librería que nos va a permitir realizar la comunicación por el protocolo I2C entre el ESP32 y el MPU6050, para que todo pueda funcionar correctamente es necesario asociar el pin SCL del MPU6050 al pin 22 del ESP32 y el pin SDA del MPU6050 al pin 21 del ESP32 tal y como se ve en la siguiente figura.

VCC	Power the sensor (3.3V or 5V)
GND	Common GND
SCL	SCL pin for I2C communication (GPIO 22)
SDA	SDA pin for I2C communication (GPIO 21)
XDA	Used to interface other I2C sensors with the MPU-6050
XCL	Used to interface other I2C sensors with the MPU-6050
ADO	Use this pin to change the I2C address
INT	Interrupt pin – can be used to indicate that new measurement data is available

Figura 43. Conexiones entre el ESP32 y el sensor MPU6050(Randomntutorials, 2023).

El primer paso que tenemos que hacer en nuestro código es comprobar que nuestro ESP32 es capaz de encontrar a nuestro MPU6050, para ello se utiliza la función mpu.begin() que enviará una condición de start al MPU6050 utilizando la dirección de esclavo de este que es la dirección 0x68, esta función nos devuelve un 1 si se ha encontrado el MPU6050 y todas las conexiones son adecuadas y un 0 si no lo ha encontrado, si no se encuentra no se ejecutará el resto del programa tal y como se puede ver en el siguiente extracto de código.

```
330  Serial.println("testeo Adafruit MPU6050 !");
331
332  //comprobamos que el MPU6050 funciona correctamente
333  if (!mpu.begin()) {//si no lo encuentra nos quedamos en el bucle
334    Serial.println("fallo para encontrar MPU6050 chip");
335    while (1) {
336      delay(10);
337    }
338  }
339  Serial.println("MPU6050 Found!");// encontrado
```

Figura 44. Código para inicializar el sensor MPU6050.

El siguiente paso es configurar los rangos de medida tanto del acelerómetro como del giroscopio como se explicó en el capítulo anterior, para asignar el rango de medida del acelerómetro se utiliza la función mpu.setAccelerometerRange() y se le especifica que el rango sea +-8G, es decir que la máxima aceleración que puede detectar es 8*fuerza de la gravedad, mientras que la mínima aceleración que puede detectar es -8*fuerza de la gravedad, esta función mpu.setAccelerometerRange() lo que hace es que escribe en el registro 0x1C un 0x10 que se interpreta como que el rango del acelerometro se ajusta a +-8G.

```
mpu.setAccelerometerRange(MPU6050_RANGE_8_G);//establecer rango del acelerómetro en +-8G.
Serial.print("Accelerometer range set to: ");
switch (mpu.getAccelerometerRange()) {
case MPU6050_RANGE_2_G:
  Serial.println("+-2G");
  break;
case MPU6050_RANGE_4_G:
  Serial.println("+-4G");
  break;
case MPU6050_RANGE_8_G:
  Serial.println("+-8G");
  break;
case MPU6050_RANGE_16_G:
  Serial.println("+-16G");
  break;
}
```

Figura 45. Código para configurar el rango del acelerómetro del sensor MPU6050.

Para asignar el rango de medida del giroscopio se utiliza la función mpu.setGyroRange() y se le especifica que el rango sea +-500dps, es decir que la máxima velocidad angular que puede detectar es 500dps, mientras que la mínima velocidad que puede detectar es -500dps, esta función mpu.setGyroRange() lo que hace es que escribe en el registro 0x1B un 0x08 que se interpreta como que el rango del giroscopio se ajusta a +-500dps.

```
357  mpu.setGyroRange(MPU6050_RANGE_500_DEG);// establecer rango del giroscopio
358  Serial.print("Gyro range set to: ");
359  switch (mpu.getGyroRange()) {
360  case MPU6050_RANGE_250_DEG:
361    Serial.println("+- 250 deg/s");
362    break;
363  case MPU6050_RANGE_500_DEG:
364    Serial.println("+- 500 deg/s");
365    break;
366  case MPU6050_RANGE_1000_DEG:
367    Serial.println("+- 1000 deg/s");
368    break;
369  case MPU6050_RANGE_2000_DEG:
370    Serial.println("+- 2000 deg/s");
371    break;
372  }
```

Figura 46. Código para configurar el rango del giroscopio del sensor MPU6050.

Por último, es necesario implementar un filtro paso bajo digital para evitar que el MPU6050 detecte las vibraciones producidas por los motores y las hélices sobre el chasis del dron, primero vamos a estimar cual es la frecuencia de estas vibraciones, nuestros motores suelen girar a una velocidad de 3000rpm cuando empiezan a girar y a 11000rpm cuando están girando a máxima velocidad.

$$frecuenciavibración(Hz) = \frac{velocidaddegiro(rpm)}{60\ segundos}$$

Con esta fórmula se estima que la frecuencia de las vibraciones producidas por los motores y las hélices con una correcta calibración oscilará entre 50Hz y 184Hz, por ello se ha optado por configurar el filtro paso bajo con una frecuencia de 21 Hz que sea capaz de filtrar estas vibraciones y que sean casi imperceptibles por el MPU6050, lo malo de los filtro es que según más pequeña sea la frecuencia de corte más retardo presenta nuestro sistema tal y como se puede observar en la siguiente ilustración obtenida del datasheet del MPU6050.

DLPF_CFG	Accelerometer (F_s = 1kHz)		Gyroscope		
	Bandwidth (Hz)	Delay (ms)	Bandwidth (Hz)	Delay (ms)	Fs (kHz)
0	260	0	256	0.98	8
1	184	2.0	188	1.9	1
2	94	3.0	98	2.8	1
3	44	4.9	42	4.8	1
4	21	8.5	20	8.3	1
5	10	13.8	10	13.4	1
6	5	19.0	5	18.6	1
7	RESERVED		RESERVED		8

Figura 47. Relación entre frecuencia de corte del filtro y el retraso entre la señal de salida y la de entrada (alldatasheet, 2023).

para configurar el filtro paso bajo digital se utiliza la función mpu.setFilterBandwith() y se le especifica que la frecuencia de corte sea de 21Hz, esta función mpu.setAccelerometerRange() lo que hace es que escribe en el registro 0x1A un 0x04 que se interpreta como que la frecuencia de corte del filtro paso bajo digital es de 21Hz para el acelerómetro y 20Hz para el giroscopio.

```
374   mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);//configurar filtro paso bajo digital.
375   Serial.print("Filter bandwidth set to: ");
376   switch (mpu.getFilterBandwidth()) {
377   case MPU6050_BAND_260_HZ:
378     Serial.println("260 Hz");
379     break;
380   case MPU6050_BAND_184_HZ:
381     Serial.println("184 Hz");
382     break;
383   case MPU6050_BAND_94_HZ:
384     Serial.println("94 Hz");
385     break;
386   case MPU6050_BAND_44_HZ:
387     Serial.println("44 Hz");
388     break;
389   case MPU6050_BAND_21_HZ:
390     Serial.println("21 Hz");
391     break;
392   case MPU6050_BAND_10_HZ:
393     Serial.println("10 Hz");
394     break;
395
396   case MPU6050_BAND_5_HZ:
397     Serial.println("5 Hz");
398     break;
399   }
```

Figura 48. Código para configurar la frecuencia de corte del filtro paso bajo del sensor MPU6050.

El último paso para calibrar nuestro MPU6050 y que sea válido consiste en calibrar los valores de aceleración y velocidad angular que nos devuelve este sensor, para ello es necesario tener los motores girando para que así el sensor MPU6050 capte las vibraciones que producen los motores en la calibración y así durante el funcionamiento del dron en vuelo reste estas vibraciones a los valores que nos da el sensor MPU6050 y así nuestro sensor minimice el efecto de las vibraciones sobre las medidas de aceleración y velocidad angular que nos da el sensor MPU6050, además para la calibración el dron debe estar horizontal respecto al suelo y apoyado sobre este, cuando lo tenemos en esta posición tomamos 4000 valores de la velocidad angular y de la aceleración que nos da nuestro sensor, para obtener tanto la velocidad angular como la aceleración se utiliza la función mpu.getEvent(), estos valores los vamos sumando a una variable y hacemos un promedio de todos estos valores que nos ha dado, con esta media nos sirve para que luego cuando pidamos valores de aceleración y velocidad angular solo tenemos que restarle esta media y así siempre vamos a tener una referencia sobre la que estimar nuestros ángulos de pitch, roll y yaw, esta referencia es con el dron horizontal respecto al suelo siempre para qué puede el dron volar y estabilizarse perfectamente, es necesario dar un tiempo entre muestra y muestra para que nuestro MPU6050 nos de datos estables.

```
for (int cal_int = 0; cal_int < 4000 ; cal_int ++){   //calibración.
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);
  gyro_x_cal += (g.gyro.x * 57.29578); //se transforma de rad/s a grados/s.
  gyro_y_cal += (g.gyro.y * 57.29578);
  gyro_z_cal += (g.gyro.z * 57.29578);
  acc_x_cal += a.acceleration.x;
  acc_y_cal += a.acceleration.y;
  acc_z_cal += a.acceleration.z;
  //la calibración se hace con los motores girando.
  digitalWrite(motor1, HIGH);  // MOTOR 1
  digitalWrite(motor2, HIGH);  // MOTOR 2
  digitalWrite(motor3, HIGH);  // MOTOR 3
  digitalWrite(motor4, HIGH);  // MOTOR 4
  timer_start = micros();
 // Cuando se cumpa el tiempo de PWM definido en throttle, se pasa cada señal a 0 (LOW) para terminar el ciclo PWM.
 while (digitalRead(motor1) == HIGH || digitalRead(motor2) == HIGH || digitalRead(motor3) == HIGH || digitalRead(motor4) == HIGH) {
  if (float(timer_start) + throttle <= float(micros())) digitalWrite(motor1, LOW); // MOTOR 1
  if (float(timer_start) + throttle <= float(micros())) digitalWrite(motor2, LOW); // MOTOR 2
  if (float(timer_start) + throttle <= float(micros())) digitalWrite(motor3, LOW); // MOTOR 3
  if (float(timer_start) + throttle <= float(micros())) digitalWrite(motor4, LOW); // MOTOR 4
 }
 delayMicroseconds(5000-throttle);
}
gyro_x_cal /= 4000;// se realiza la media de 4000 muestras para obtener la calibración correcta.
gyro_y_cal /= 4000;
gyro_z_cal /= 4000;
acc_x_cal /= 4000;
acc_y_cal /= 4000;
acc_z_cal /= 4000;
```

Figura 49. Código para calibrar el sensor MPU6050.

7.2 Medir ángulos pitch, roll y yaw

Para obtener los ángulos de pitch, roll y yaw primero hay que obtener los datos de velocidad angular y aceleración que nos da el sensor MPU6050, para ello utilizamos la función mpu.getEvent(), para tener una referencia de 0 grados cuando el dron está horizontal respecto al suelo es necesario restar a cada valor de aceleración y velocidad angular que nos da nuestro sensor MPU6050 el valor de calibración que se obtuvo en el apartado anterior, la función mpu.getEvent() nos da los valores de velocidad angular en rad/s y nosotros lo queremos en dps, por lo que es necesario multiplicar por π/180 para pasar de rad a segundos, los valores de la aceleración están en m/s^2 tal y como deseamos nosotros, además se le pone un filtro complementario a las señales que nos da el MPU6050 para reducir las vibraciones que producen nuestro motor y nuestras hélices al mínimo.

```
sensors_event_t a, g, temp;
mpu.getEvent(&a, &g, &temp);//obtenemos los valores de aceleraccción y velocidad angular que nos da el sensor MPU6050.
tiempo_anterior=tiempo;//guardamos las variables temporales para saber el tiempo que pasa entre cada vez que pasa por este punto.
tiempo=micros();
dt=(tiempo-tiempo_anterior);
gyro_x = (g.gyro.x * 57.29578) - gyro_x_cal;//calibramos la velocidad angular del giroscopio y lo pasamos a grados/s.
gyro_y = (g.gyro.y * 57.29578) - gyro_y_cal ;
gyro_z= (g.gyro.z * 57.29578) - gyro_z_cal;
gyro_x_filt=0.4*gyro_x_filt+gyro_x*0.6;//filtro para eliminar el ruido de las señales.
gyro_y_filt=0.4*gyro_y_filt+gyro_y*0.6;
gyro_z_filt=0.4*gyro_z_filt+gyro_z*0.6;
acc_x=(a.acceleration.x - acc_x_cal);//calibramos la aceleraccción del acelerómetro.
acc_y=(a.acceleration.y - acc_y_cal);
acc_z=(a.acceleration.z - acc_z_cal +9.8);
acc_x_filt=0.4*acc_x_filt+acc_x*0.6;//filtro para eliminar el ruido de las señales.
acc_y_filt=0.4*acc_y_filt+acc_y*0.6;
acc_z_filt=0.4*acc_z_filt+acc_z*0.6;
```

Figura 50. Código para obtener el valor de la aceleración y la velocidad angular en cada uno de los ejes.

Ahora se multiplica los valores de velocidad angular en dps por el tiempo que se tarda en pedir un dato, en nuestro caso pedimos un dato cada 5ms ya que nuestro bucle de funcionamiento para que dé tiempo a que todas las funciones del dron se ejecuten tiene una frecuencia de 200Hz, esto nos sirve para que todas las funciones del bucles se ejecuten síncronamente con la señal de reloj de nuestro ESP32, para calcular el ángulo de pitch se utiliza la velocidad angular en el eje x ya que el giro en el eje x hace que el ángulo de pitch varíe, para calcular en ángulo de roll se utiliza la velocidad angular en el eje y ya que el giro en el eje y hace que el ángulo de roll varíe, para calcular en ángulo de yaw se utiliza la velocidad angular en el eje z ya que el giro en el eje z hace que el ángulo de yaw varíe.

```
509  //calculamos los angulos de pitch roll y yaw con los valores de velocidad angular que nos da el sensor MPU6050
510   angle_pitch += gyro_x_filt * (float(dt)*0.000001);//0.0059  0
511   angle_roll += gyro_y_filt * (float(dt)*0.000001);//0.0061
512   angle_yaw += gyro_z_filt * (float(dt)*0.000001);
513
```

Figura 51. Código para calcular el ángulo de pitch, roll y yaw del dron a partir de la velocidad angular.

```
669  //ejecutamos nuestro bucle cada 5ms(200Hz)
670  while(micros() - loop_timer < 5000) ;
671  loop_timer = micros();
```

Figura 52. Código para obtener datos del MPU6050 cada 5ms.

El siguiente paso es que cuando el dron vaya girando, es decir que el yaw cambie hay que hacer que varíe los ángulos de pitch y roll en función de la rotación en el eje z que tengamos, para ello lo que se hace es que al ángulo pitch que se ha calculado anteriormente se le suma el ángulo de roll y se le multiplica por la función seno (sin(rotacion_eje_z *tiempo)), mientras que al ángulo de roll se le resta el ángulo de pitch multiplicado por la función seno (sin(rotacion_eje_z *tiempo)), esto nos permite por ejemplo que si estamos con el dron con un pitch de 45 grados, el roll de 0 grados y el dron recto sin ningún yaw y luego giramos el dron 90 grados lo que va a pasar es que el ángulo de pitch pase al roll y el roll pase al pitch gracias a la función que se puede ver a continuación y en consecuencia tener un correcto funcionamiento del sensor MPU6050.

```
400  //0.01745329 = (3.142(PI) / 180degr) la función sin en arduino está en radianes.
401  angle_pitch += angle_roll * sin(gyro_z_filt * float(dt) * 0.00000001745);
402  angle_roll -= angle_pitch * sin(gyro_z_filt * float(dt) * 0.00000001745);
```

Figura 53. Código para tener en cuenta en el pitch y roll la rotación del eje z.

Ahora es necesario calcular los ángulos de inclinación con los valores que nos da el acelerómetro, para ello el ángulo de pitch se calcula dividiendo la aceleración en el eje y entre el módulo de la aceleración de los tres ejes, a diferencia que con el giroscopio que se utilizaba la rotación en el eje x ya que es distinto la calcular un ángulo respecto a una

rotación que respecto a la aceleración, tal y como se vio en la explicación del sensor MPU6050, esta aceleración en el eje y se divide entre el módulo de las aceleraciones en todos los ejes, luego es necesario hacer la función arcoseno para obtener el ángulo de inclinación y por último se multiplica por 180/π para pasarlo a grados, para calcular el ángulo de roll se hace lo mismo pero cambiando la aceleración del eje y por la aceleración del eje x.

```
404   acc_total_vector = sqrt((acc_x_filt*acc_x_filt)+(acc_y_filt*acc_y_filt)+(acc_z_filt*acc_z_filt));
405   //57.296 = 1 / (3.142 / 180) la función seno está en radianes.
406   angle_pitch_acc = asin(acc_y_filt/acc_total_vector)* 57.296;
407   angle_roll_acc = asin(acc_x_filt/acc_total_vector)* -57.296;
```

Figura 54. Código para calcular el ángulo de pitch, roll y yaw del dron a partir de la aceleración.

por último para tener bien calculados los ángulos es necesario utilizar tanto los ángulos que hemos obtenido con el giroscopio del MPU6050 como los ángulos obtenidos con el acelerómetro del MPU6050, ya que el problema que tiene el acelerómetro es que es más sensible al ruido producido por el dron y entonces produce que los grados de inclinación sean muy inestables, pero lo bueno que tiene es que tiene muy pocas derivas ya que a pesar de tener ruido siempre suele tener un ángulo muy próximo al real al cabo del tiempo, mientras que el problema del giroscopio es que tiene muchas derivas con el tiempo y si pasa mucho tiempo el ángulo que devuelve el giroscopio es distinto del ángulo que tiene el dron en la realidad, la parte buena que tiene el giroscopio es que es más insensible a las vibraciones producidas por los motores y hélices, por lo que se llegó a la conclusión de utilizar un filtro complementario que utilice en una gran mayoría el ángulo que nos da el giroscopio para evitar ruido, pero que utilice en muy pequeña medida el ángulo del acelerómetro para corregir las derivas al cabo del tiempo, además cabe destacar que al inicio el ángulo que nos da el giroscopio no es muy preciso ya que depende del tiempo que tarda en sacar un dato estable del MPU6050, por lo que se opta por que la primera vez que coge valores de ángulos nuestro sensor utilice los valores del acelerómetro, en la siguiente ilustración se puede ver como se programó esta función.

```
524   if(set_gyro_angles){  //corregir drift ángulos giroscopio
525     angle_pitch = angle_pitch * 0.9996 + angle_pitch_acc * 0.0004;
526
527     angle_roll = angle_roll * 0.9996 + angle_roll_acc * 0.0004;
528   }
529   else{ //al principio
530     angle_pitch = angle_pitch_acc;
531     angle_roll = angle_roll_acc;
532     angle_yaw=0;
533     set_gyro_angles = true;
534   }
```

Figura 55. Filtro complementario para utilizar el pitch, roll y yaw tanto a partir de la velocidad angular como de la aceleración.

7.3 Comunicación protocolo ESP-NOW

para establecer la comunicación entre el ESP32 del dron y el ESP32 del mando es necesario conocer la dirección MAC de cada uno de ellos ya que el emisor para transmitir los datos debe conocer la dirección MAC del receptor, y al ser una comunicación bidireccional cada ESP32 debe conocer la dirección MAC del otro, para obtener la dirección MAC de cada uno se puede obtener gracias a este código que nos imprime la dirección MAC del ESP32, para cada placa esta dirección es única.

```
#include "WiFi.h"

void setup(){
  Serial.begin(115200);
  WiFi.mode(WIFI_MODE_STA);
  Serial.println(WiFi.macAddress());
}
```

Figura 56. Código para obtener la dirección MAC de cada ESP32.

Primero es necesario comprobar que se ha puesto la dirección MAC correctamente y que la comunicación entre los dos ESP32 se ha establecido de forma adecuada, para ello se utiliza la función esp_now_init() que si se estableció la comunicación de forma correcta se devuelve un ESP_OK, por lo que es necesario comprobar si devuelve este valor para corroborar que la comunicación está perfectamente realizada.

```
281  if (esp_now_init() != ESP_OK) {//comprueba si la comunicación entre los esp32 se estableció de forma correcta.
282    Serial.println("Error initializing ESP-NOW");
283    return;
284  }
```

Figura 57. Código para inicializar protocolo ESP-NOW.

Después es necesario crear 4 estructuras de datos, dos en el ESP32 del dron y otros dos en el ESP32 del mando, en cada ESP32 una estructura es donde se almacenan los datos para enviar y otra estructura es donde se almacenan los datos que se van a recibir.

```
//estructura para recibir datos.
typedef struct struct_message2 {
    float incomingroll;
    float incomingpitch;
    int incomingthrottle;
    float incomingyaw;
} struct_message2;
struct_message2 messagereceive;

String success;
//estructura para enviar datos
typedef struct struct_message {
    float angle_pitch;
    float angle_roll;
    float angle_yaw;
    float tension_bateria_filt;

} struct_message;
struct_message MPU6050Readings;
```

Figura 58. Código para crear dos estructuras de datos.

Tras tener perfectamente inicializada la comunicación entre los dos ESP32 es necesario registrar la conexión entre ambos, luego es necesario registrar la conexión entre ambos asociándolo un canal que en nuestro caso será el 0 y encriptando el mensaje por si es necesario mucha seguridad, en nuestro caso no es necesario por lo que no se encripta el mensaje, luego por último será necesario comprobar si la conexión se hizo adecuadamente mediante la función esp_now_add_peer, si es correcta se recibirá un ESP_Ok, es importante comprobar que se recibe una afirmación de que la conexión se estableció correctamente.

```
// Registrar conexión.
memcpy(peerInfo.peer_addr, broadcastAddress, 6);
peerInfo.channel = 0;//canal 0.
peerInfo.encrypt = false;//no se encripta

// Add peer
if (esp_now_add_peer(&peerInfo) != ESP_OK){//comprueba si la conexión se estableció correctamente
  Serial.println("Failed to add peer");
  return;
}
```

Figura 59. Código para establecer conexión entre dos ESP32 por protocolo ESP-NOW.

para enviar y recibir los datos es necesario configurar dos funciones, para la de enviar es necesario especificar la dirección MAC del destino, el mensaje que se envía y la longitud del mensaje, para recibir los datos es necesario especificar la dirección MAC del emisor, los datos que se reciben y la longitud de estos, todo esto nos sirve para comprobar que los mensajes que se reciben corresponden con la longitud de los datos que se envían, por último es necesario guardar los mensajes recibidos en las variables que vamos a utilizar, la función de enviar mensaje nos devuelve un ESP_OK que nos sirve para comprobar si los datos se han enviado correctamente.

```
534  //función de enviar datos
535  esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &MPU6050Readings, sizeof(MPU6050Readings))
536
537  if (result == ESP_OK) {
538   Serial.println("Sent with success");
539  }
540  else {
541   Serial.println("Error sending the data");
542
543  }
```

Figura 60. Código para enviar datos por protocolo ESP-NOW.

```
101  // Función para recibir los datos
102  void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len) {
103    memcpy(&messagereceive, incomingData, sizeof(messagereceive));
104    refroll = messagereceive.incomingroll;
105    refpitch = messagereceive.incomingpitch;
106    incomingthrottle=messagereceive.incomingthrottle;
107    refyaw=messagereceive.incomingyaw;
108
109  }
```

Figura 61. Código para recibir datos por protocolo ESP-NOW.

Cuando se envían datos se entra en una función que comprueba si los datos se han enviado correctamente, aquí se puede ver dicha función.

```
90   // Función para comprobar que se envian bien los datos.
91   void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
92
93     if (status ==0){
94       success = "Delivery Success :)";
95     }
96     else{
97       success = "Delivery Fail :(";
98     }
99   }
```

Figura 62. Código para comprobar que se envían bien los datos por protocolo ESP-NOW.

El ESP32 del dron va a enviar el ángulo de picth, roll y yaw del dron en cada momento y el ESP32 del mando va a enviar el ángulo de roll, pitch y la velocidad ángular en dps en el eje z que queremos que tenga nuestro dron en cada instante.

7.4 Medir voltaje batería

Para medir el voltaje de la batería se utiliza un conversor analógico digital para convertir el voltaje que tiene la batería en cada instante a un valor digital que pueda ser interpretado por el ESP32, además es importante pasarlo por un filtro paso bajo digital este valor para reducir el ruido que lleva la señal de voltaje de la batería, cabe destacar que hay un circuito que convierte los 11,1v cuando la batería está cargada a 2,82V ya que el ESP32 solo es capaz de convertir voltajes entre 0 y 3,3V, pero a nosotros en nuestro código después de haber convertido a digital la tensión que se leyó en el ADC nos interesa pasar de esos 2,82V cuando está cargada a los 11.1V de voltaje que tiene nuestra batería en la realidad ya que así la persona que maneje el dron pueda visualizar

en la pantalla del mando el voltaje que tiene la batería en ese momento, para convertir desde el valor que nos da en ADC del ESP32 al valor real de la batería hay que seguir los siguientes pasos:

- se pasa del valor digital que lee el ESP32 al valor analógico real que corresponde en la entrada del pin al que se conecta la batería, por ello se multiplica el valor digital por 3,23/4095 y se le suma 0,1 que es el offset que no mide los conversores analógicos digitales del ESP32 debido al problema de saturación con voltajes bajos de los ADC.
- Por último, se multiplica por 10,95/2,78 para realizar ese paso que se dijo de convertir los 2,82V en 11,1V, ahora ya se pueden enviar los datos al mando para que se puedan visualizar con pantalla y saber siempre el estado de nuestra batería.

```
602   //leer voltaje de la batería
603   tension_bateria=(float(analogRead(voltaje))*(3.23/4095)+0.1)*10.95/2.78;
604   tension_bateria_filt = 0.95 * tension_bateria_filt + 0.05 * tension_bateria;
```

Figura 63. Código para leer el voltaje de la batería.

7.5 Mando a distancia

Para la comunicación con el otro ESP32 se hacen los mismos pasos que se ha explicado en el apartado anterior.

Lo primero que se tiene que hacer es configurar los conversores analógicos digitales que vamos a utilizar para obtener los valores de salida de los potenciómetros de los joysticks, se han utilizado los pines 34, 35, 36 y 39 los cuales no es necesario indicar que van a funcionar como entradas ya que solo pueden funcionar de esa forma, ahora se procede a leer los valores analógicos que nos dan en cada uno de los pines y convertirlos a un valor digital entre 0 y 4095 que pueda ser interpretado por el ESP32.

```
135   // leemos los valores analógicos que nos da los potenciometros de los joysticks
136   roll_sin=analogRead(34);
137   pitch_sin=analogRead(35);
138   throttle_sin=analogRead(36);
139   yaw_sin=analogRead(39);
```

Figura 64. Código para leer el voltaje de salida de los joysticks.

luego se opta por realizar un filtrado de estas señales para eliminar el ruido y obtener sus valores lo más estable posibles, además luego procedemos a realizar una interpolación lineal mediante la función map() para así obtener el ángulo de referencia de pitch, roll y la velocidad angular de referencia del eje z en función de el voltaje que nos da cada uno de los potenciómetros de los joysticks.

```
129   if((pitch <= 1950) && (pitch >= 1850)){//interpolación lineal.
130     out1=0;
131   }
132   if(pitch > 1950){
133       out1 = map( pitch, 1950, 4095, 0, 1000 );
134     }
135     else if(pitch < 1850){
136       out1 = map( pitch, 0, 1850, -1000, 0 );
137     }
138     ref1=float(out1)/100.0;
139   roll=0.4 * roll +0.6 *roll_sin;//filtrado para eliminar el ruido de las señales.
140   if ((roll <= 1950)&&(roll>=1850)){//interpolación lineal.
141     out=0;
142   }
143   if(roll > 1950){
144       out = map( roll, 1950, 4095, 0, 1000 );
145     }
146     else if(roll < 1850){
147       out = map( roll, 0, 1850, -1000, 0 );
148     }
149     ref= float(out)/100.0;
150   throttle=0.7 * throttle + 0.3 * throttle_sin;//filtrado para eliminar el ruido de las señales.
151   yaw=0.4 * yaw + 0.6 * yaw_sin;//filtrado para eliminar el ruido de las señales.
152   if((yaw <=1940) && (yaw>=1840)){//interpolación lineal.
153     out2=0;
154   }
155   if(yaw > 1940){
156       out2 = map( yaw, 1940, 4095, 0, -100 );
157     }
158   else if(yaw < 1840){
159       out2 = map( yaw, 0, 1840, 100, 0 );
160     }
161     ref2= float(out2)/100;
```

Figura 65. Código para convertir el voltaje de salida de los joysticks a un valor de ese parámetro que queramos de referencia.

Con todo esto ya se puede enviar estos datos al ESP32 de dron los datos del ángulo de pitch y de roll de referencia, el throttle al que van a girar los motores y la velocidad ángulo de referencia que debe de tener nuestro dron en cada instante.

Para visualizar el pitch el roll y el yaw que tiene nuestro dron en cada instante se utilizó una pantalla OLED, para poder utilizar esta pantalla con el ESP32 fue necesario instalar tres librerías que son la librería “Wire.h” que nos permite realizar la comunicación I2C entre el ESP32 y la pantalla OLED, mientras que tanto la librería “Adafruit_GFX.h” como la librería “Adafruit_SSD1306.h” sirven para poder manejar la pantalla OLED de forma adecuada, es necesario indicar al ESP32 las dimensiones del Display que son 128 pixeles de ancho y 64 de largo. cabe destacar que la dirección de la pantalla OLED al actuar como esclavo en el protocolo de comunicaciones I2C es 0x3C.

```
#include <esp_now.h>
#include <WiFi.h>

#include <Wire.h>


#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128  // pixeles ancho display OLED
#define SCREEN_HEIGHT 64  // pixeles largo display OLED
#define led 13

// declarar SSD1306 display conectado por I2C I2C (SDA, SCL pins)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
```

Figura 66. Código para configurar la pantalla OLED.

Ahora hay que comprobar si se encuentra la pantalla oled de forma adecuada, para ello el ESP32 envía una condición de start y tenemos que comprobar si la pantalla nos responde si está funcionando, para ello se utiliza la función display.begin(), si está a uno significa que se ha inicializado bien, por ello hay que comprobar que esta función nos devuelve un uno para saber que la conexión está correctamente inicializada.

```
// Inicializar OLED display
if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
  Serial.println(F("SSD1306 allocation failed"));
  for(;;);
}
```

Figura 67. Código para inicializar la pantalla OLED.

Luego ya nos dedicamos a imprimir todos los datos que queremos en el display, en nuestro caso primero hay que limpiar la pantalla mediante la función display.clearDisplay(), después se pone un formato de que cada letra ocupe un píxel mediante la función display.setTectSixe(), y para terminar de configurar la letras se pone que tengan un color blanco para poder ver bien todo el texto en la pantalla mediante la función display.setTextColor(), después ya lo imprimimos en la pantalla el pitch, el roll , el yaw y el voltaje de la batería en cada instante que nos envía el otro ESP32, es necesario indicar en qué posición queremos que se empiece a escribir mediante la función display.setCursor() y para escribir se utiliza la función display.print()

```
209  void updateDisplay(){
210    // imprimimos las lecturas en el OLED Display
211    display.clearDisplay();
212    display.setTextSize(1);
213    display.setTextColor(WHITE);
214    display.setCursor(0, 0);
215    display.println("Lecturas:");
216    display.setCursor(0, 15);
217    display.print("roll: ");
218    display.print(incomingroll);
219    display.setCursor(0, 25);
220    display.print("pitch: ");
221    display.print(incomingpitch);
222    display.setCursor(0, 35);
223    display.print("yaw: ");
224    display.print(incomingyaw);
225    display.setCursor(0, 45);
226    display.print("bateria:");
227    display.print(voltaje_bateria);
228    display.print("V");
229    display.setCursor(0, 56);
230    display.print(success);
231    display.display();
232  }
```

Figura 68. Código para actualizar la pantalla OLED.

Por último, en el mando es necesario configurar el led rojo que se ha puesto para indicar que el voltaje de la batería está por debajo de 9 voltios, esto nos indica que la batería está casi descargada y necesita ponerse a cargar para que pueda mover los motores con facilidad, para que funcione el led se ha puesto el pin 13 como salida y cuando se baja de este valor de 9 voltios se envía 3,3 voltios por este pin y se enciende el led ya que esta patilla está conectado al ánodo led.

```
96   //poner pin 13 LED como salida
97   pinMode(led,OUTPUT);
98   digitalWrite(led, LOW);
```

Figura 69. Código para Inicializar LED.

```
163  if((voltaje_bateria <= 9.0) &&(voltaje_bateria>=3.0)){//encender LED si el voltaje de la batería baja.
164    digitalWrite(led, HIGH);
165  }
166  else{
167    digitalWrite(led, LOW);
168  }
```

Figura 70. Código para encender el LED cuando la batería se descarga.

7.6 PIDs

Para la programación de los PIDs hay que tener en cuenta toda la teoría de los PIDs que se habló en apartados anteriores, vamos a utilizar tres PIDs, el primero se utilizara para controlar el roll de nuestro dron y que nuestro dron se estabilice en este de forma correcta, en mi caso se va a utilizar el PID para mantener el ángulo de roll en el valor de referencia que se envíe desde nuestro mando, se puede utilizar el ángulo o la velocidad angular que nos da el giroscopio para realizar el PID, es nuestro caso como el modo de vuelo es modo estable por lo que se utiliza el ángulo, mientras que si se quisiese hacer un dron que hiciese acrobacias habría que realizar también un PID utilizando los valores

de la velocidad angular en el eje y, para realizar el PID de roll de forma adecuada se han seguido los siguientes pasos:

- Se calcula el error entre el ángulo de roll que tiene en ese instante el dron y el ángulo de referencia de roll que se envía desde el mando.
- Ahora ya con el error procedemos a calcular la parte proporcional, la parte integral y la parte derivativa, para la parte proporcional se multiplica la constante proporcional por el error. para la parte integral multiplicamos el error por la constante integral y se suma al valor de la parte integral que hubiese en la interacción anterior, nos conviene limitar la parte integral para evitar que en caso de error este valor aumente mucho y nuestro dron se descontrole ya que el motor empezará a girar muy rápido. Para la parte derivativa se multiplica la constante derivativa por el error menos el error anterior, luego se suman la parte proporcional, la integral y la derivativa y ya se tendría el PID del roll, también merece la pena limitar este PID en caso de error para evitar que se descontrole nuestro dron si en caso de error la parte integral empieza a aumentar excesivamente, estas dos medidas de limitar tanto parte integral como el PID son solo medidas preventivas.

```
120  void pidroll(){
121
122    error_roll = angle_roll - refroll;//se calcula error del roll.
123
124    pid_p = kproll*error_roll;//se calcula parte proporcional.
125
126
127    pid_i_roll += (kiroll*error_roll);//se calcula parte integral.
128    if(pid_i_roll>200.0){//se limita la parte integral.
129      pid_i_roll=200.0;
130    }
131    else if(pid_i_roll<-200.0){
132      pid_i_roll=-200.0;
133    }
134    pid_d = kdroll*(error_roll - previous_error_roll);//se calcula parte derivativa.
135
136    PIDROLL = pid_p + pid_i_roll + pid_d;//se calcula PID.
137
138    if(PIDROLL < -400)// se limita el valor del PID
139    {
140      PIDROLL = -400;
141    }
142    if(PIDROLL > 400)
143    {
144      PIDROLL = 400;
145    }
146
147    previous_error_roll = error_roll;
148  }
```

Figura 71. Código para el PID del roll.

Ahora se va a realizar el PID para el pitch, para que nuestro dron se mantenga estable con el ángulo que manda el usuario desde el mando, el PID del pitch utiliza al igual que el roll el ángulo en vez de la velocidad angular ya que estamos funcionando en modo de vuelo estable, para realizar el PID del pitch se siguen los siguientes pasos:

- Se calcula el error entre el ángulo de pitch que tiene en ese instante el dron y el ángulo de referencia de pitch que se envía desde el mando.
- Ahora ya con el error procedemos a calcular la parte proporcional, la parte integral y la parte derivativa, para la parte proporcional se multiplica la constante proporcional por el error. para la parte integral multiplicamos el error por la constante integral y se suma al valor de la parte integral que hubiese en la interacción anterior, nos conviene limitar la parte integral para evitar que en caso de error este valor aumente mucho y nuestro dron se descontrole ya que el motor empezará a girar muy rápido. Para la parte derivativa se multiplica la constante derivativa por el error menos el error anterior, luego se suman la parte proporcional, la integral y la derivativa y ya se tendría el PID del pitch, también merece la pena limitar este PID en caso de error para evitar que se descontrole nuestro dron si en caso de error la parte integral empieza a aumentar excesivamente, estas dos medidas de limitar tanto parte integral como el PID son solo medidas preventivas.

```
169  void pidpitch(){
170
171    error_pitch = angle_pitch - refpitch;//se calcula el error de picth.
172    pid_p = kppitch * error_pitch;//se calcula la parte proporcional.
173    pid_i_pitch += (kipitch * error_pitch);//se calcula la parte integral.
174    if(pid_i_pitch>=200.0){//se limita la parte integral.
175      pid_i_pitch=200.0;
176    }
177    if(pid_i_pitch<=-200.0){
178      pid_i_pitch=-200.0;
179    }
180    pid_d = kdpitch*(error_pitch - previous_error_pitch);//se calcula la parte derivativa.
181    PIDPITCH = pid_p + pid_i_pitch + pid_d;//se calcula el PID.
182    if(PIDPITCH < -400.0)//se limita el PID.
183    {
184      PIDPITCH = -400.0;
185    }
186    if(PIDPITCH > 400.0)
187    {
188      PIDPITCH = 400.0;
189    }
190    previous_error_pitch = error_pitch;
191
192  }
```

Figura 72. Código para el PID del pitch.

Ahora se va a realizar el PID para el yaw, para que nuestro dron se mantenga estable con la velocidad angular que manda el usuario desde el mando, el PID del picth utiliza la velocidad angular del yaw para mantener el dron en la misma posición si el yaw del mando no se mueve, mientras que si se mueve el joystick del yaw el dron se moverá, por lo tanto, el yaw trabaja en modo acrobático ya que se utiliza la velocidad angular, para realizar el pid del yaw se siguen los siguientes pasos:

- Se calcula el error entre la velocidad angular en el eje z que tiene en ese instante el dron y la velocidad angular de referencia en el eje z que se envía desde el mando.

- Ahora ya con el error procedemos a calcular la parte proporcional, la parte integral y la parte derivativa, para la parte proporcional se multiplica la constante proporcional por el error. para la parte integral multiplicamos el error por la constante integral y se suma al valor de la parte integral que hubiese en la interacción anterior, nos conviene limitar la parte integral para evitar que en caso de error este valor aumente mucho y nuestro dron se descontrole ya que el motor empezará a girar muy rápido. Para la parte derivativa se multiplica la constante derivativa por el error menos el error anterior, luego se suman la parte proporcional, la integral y la derivativa y ya se tendría el PID del pitch, también merece la pena limitar este PID en caso de error para evitar que se descontrole nuestro dron si en caso de error la parte integral empieza a aumentar excesivamente, estas dos medidas de limitar tanto parte integral como el PID son solo medidas preventivas.

```
214  void pidyaw(){
215
216    error_yaw = gyro_z_filt - refyaw;//se calcula el error de yaw.
217
218    pid_p = kpyaw * error_yaw;//se calcula la parte proporcional.
219
220    pid_i_yaw += (ki_yaw * error_yaw);//se calcula la parte integral.
221    if(pid_i_yaw>=380.0){//se limita la parte integral
222      pid_i_yaw=380.0;
223    }
224    if(pid_i_yaw<=-380.0){
225      pid_i_yaw=-380.0;
226    }
227
228    PIDYAW = pid_p + pid_i_yaw ;//se calcula el PID del yaw
229
230    if(PIDYAW < -450)//se limita el PID del yaw
231    {
232      PIDYAW=-450;
233    }
234    if (PIDYAW > 450)
235    {
236      PIDYAW=450;
237    }
238
239  }
```

Figura 73. Código para el PID del yaw.

Tras unas cuantas pruebas los valores óptimos de k_p, k_i y k_d son 4, 0,09 y 150 respectivamente, estos valores son iguales para el PID de pitch y de roll ya que los dos ejes sobre los que se encuentran los motores tienen prácticamente la misma longitud y forman entre si un ángulo de aproximadamente 90 grados, sin embargo, para el PID del yaw se pusieron unos valores de k_p y k_i de 1,3 y 0,05 respectivamente.

7.7 Realización señales PWM

Para crear las señales PWM que vayan a cada motor se utilizan los pines 23(motor1), 4(motor2), 13(motor3) y 32(motor4), como están posicionados en nuestro dron se puede

ver en el esquema electrónico, para generar la señal PWM se ponen la salida de estos pines en alto mediante la función digitalWrite(), luego lo que se hace que en función del throttle que nos llega del mando, más los resultados de los PIDs se obtiene un valor en microsegundos que corresponde a la duración de la señal en alta, se puede ver como la variable velocidad es el tiempo en alta de la señal PWM del pin 23, la variable velocidad1 es la señal PWM del pin 4, la variable velocidad2 es la señal PWM del pin 13, la variable velocidad3 es la señal PWM del pin 32, cuando pasa ese tiempo que se necesita para que el dron se mantenga estable se pone la señal en baja, el periodo de la señal es de 5ms, por lo que la señal PWM estará en alta el tiempo que se indique en las variable velocidad y luego estará en baja durante un tiempo hasta que se vuelva a poner en alta tras 5ms, por lo que generamos la señal PWM con frecuencia de 200Hz, hay que esperar a que se llegue a 5ms en nuestro loop en cada iteración para así producir una señal PWM estable de frecuencia de 200Hz.

```
601   if((incomingthrottle > 1743)){//en función del valor del joystick del throttle se selecciona un throttle adecuado.
602       offset = map( incomingthrottle, 1744, 4095, 14000, 17000 );//-5
603       throttle=float(offset)/10;
604
605
606   }
607   else {
608     offset = map( incomingthrottle, 0, 1743, 11000, 14000 );
609     throttle=float(offset)/10;
610
611
612   }
```

Figura 74. Código para ajustar el valor de salida de voltaje del joystick a un valor de throttle adecuado para que se muevan los motores.

```
621   //se calcula el tiempo que las señales tienen que esatr en alta para que se mantenga estable el dron
622   velocidad =throttle + PIDPITCH + PIDROLL +PIDYAW;
623   velocidad1 = throttle + PIDPITCH - PIDROLL - PIDYAW;
624   velocidad2 =  throttle - PIDPITCH - PIDROLL + PIDYAW;
625   velocidad3 = throttle - PIDPITCH + PIDROLL - PIDYAW;
```

```
595   //se pone la señal PWM en alta
596   digitalWrite(motor1, HIGH);  // MOTOR 1
597   digitalWrite(motor2, HIGH);  // MOTOR 2
598   digitalWrite(motor3, HIGH);  // MOTOR 3
599   digitalWrite(motor4, HIGH);  // MOTOR 4
600   timer_start = micros();
601
602   // Cuando se cumpa el tiempo de PWM definido en velocidad, se pasa cada señal a 0 (LOW) para terminar el ciclo PWM.
603   while (digitalRead(motor1) == HIGH || digitalRead(motor2) == HIGH || digitalRead(motor3) == HIGH || digitalRead(motor4) == HIGH) {
604     if (float(timer_start) + velocidad <= float(micros())) digitalWrite(motor1, LOW); // MOTOR 1
605     if (float(timer_start) + velocidad1 <= float(micros())) digitalWrite(motor2, LOW); // MOTOR 2
606     if (float(timer_start) + velocidad2 <= float(micros())) digitalWrite(motor3, LOW); // MOTOR 3
607     if (float(timer_start) + velocidad3 <= float(micros())) digitalWrite(motor4, LOW); // MOTOR 4*/
608   }
609   //ejecutamos nuestro bucle cada 5ms(200Hz)
610   while(micros() - loop_timer < 5000) ;
611   loop_timer = micros();
612
613 }
```

Figura 75. Código para configurar las señales PWM.

Ahora se realizan medidas preventivas para que nuestro dron funcione correctamente, primero se espera a que las variables velocidad que hemos visto antes superen los 1160 microsegundos, para ello lo que hay que hacer es subir el throttle, cuando todas las

variables velocidad pasan de este valor de 1160 microsegundos se pone la variable start a 1 y los PIDs empiezan a funcionar.

```
481   if(start1==0){//Empezamos con ángulos de 0 grados.
482     angle_pitch=0.0;
483     angle_roll=0.0;
484   }
485   if(start!=0){//Hasta que el dron se eleva no funcionan los PIDs.
486     pidpitch();
487     pidroll();
488     pidyaw();
489   }
490   if(programable==0){
491   //se calcula el tiempo que las señales tienen que esatr en alta para que se mantenga estable el dron.
492     velocidad =throttle + PIDPITCH + PIDROLL +PIDYAW;
493     velocidad1 = throttle + PIDPITCH - PIDROLL - PIDYAW;//+50
494     velocidad2 =  throttle - PIDPITCH - PIDROLL + PIDYAW;
495     velocidad3 = throttle - PIDPITCH + PIDROLL - PIDYAW;
496   }
497
498
499   if((velocidad>1160.0) && (velocidad1>1160.0) && (velocidad2>1160.0) && (velocidad3>1160.0) && (error_mot==0)){
500    start=1;
501    start1=1;
502  }
```

Figura 76. Medida preventiva PID.

otra medida que hay que hacer es evitar que si nuestro dron está en el aire los valores de velocidad es decir el tiempo que la señal PWM está en alta no puede ser menor de 1150 microsegundos para evitar que nuestros motores se paran y el dron se caiga, también hay que evitar que el tiempo que la señal PWM en alta sea mayor de 2000 microsegundos para evitar que se calienten en exceso los motores y se estropeen.

```
517   if((start==1) && (error_mot==0)){//limitar el tiempo en alta tanto por arriba como por abajo.
518     if(velocidad < 1150.0)
519     {
520       velocidad= 1150.0;
521     }
522     if(velocidad > 2000.0)
523     {
524       velocidad=2000.0;
525     }
526     if(velocidad1 < 1150.0)
527     {
528       velocidad1= 1150.0;
529     }
530     if(velocidad1 > 2000.0)
531     {
532       velocidad1=2000.0;
533     }
534     if(velocidad2 < 1150.0)
535     {
536       velocidad2= 1150.0;
537     }
538     if(velocidad2 > 2000.0)
539     {
540       velocidad2=2000.0;
541     }
542     if(velocidad3 < 1150.0)
543     {
544       velocidad3= 1150.0;
545     }
546     if(velocidad3 > 2000.0)
547     {
548       velocidad3=2000.0;
549     }
```

Figura 76. Medida preventiva valor máximo y mínimo PWM.

La última medida de prevención es que si el ángulo de pitch o roll pasa de 45 grados o -45 grados los motores se paran, si pasa de estos ángulos de inclinación significa que nuestro dron ha perdido el control y ha habido un fallo, por lo que es necesario parar los motores para evitar que cuando choque con el suelo los motores sigan girando y se calienten y se estropeen o incluso se puedan llegar a quemar.

```
686   if((abs(angle_pitch)>=45) || ((abs(angle_roll))>=45)||(error_mot==1)){
687     //cuando el ángulo de pitch o de roll sea mayor de 45 grados se paran los motores
688     velocidad=1000;
689     velocidad1=1000;
690     velocidad2=1000;
691     velocidad3=1000;
692     error_mot=1;
693     PIDPITCH=0;
694     PIDROLL=0;
695     PIDYAW=0;
696     pid_i_pitch=0;
697     pid_i_roll=0;
698     pid_i_yaw=0;
699     error_roll=0;
700     previous_error_roll=0;
701     error_pitch=0;
702     previous_error_pitch=0;
703     error_yaw=0;
704     previous_error_yaw=0;
705   }
```

Figura 77. Medida preventiva excesivo ángulo de inclinación.

7.8 Realización ruta de vuelo programable

Para crear una ruta de vuelo programable lo primero que hay que hacer es aumentar el throttle de forma progresiva durante un cierto tiempo hasta que el dron tenga un cierto nivel sobre el suelo y así podemos ejecutar los siguientes comandos para realizar la ruta de vuelo programable de nuestro dron, en el caso particular de nuestro dron se puso que el throttle aumentase de forma progresiva durante 2.5ms hasta producir una señal PWM con tiempo en alta de 1475µs, al tener nuestro dron ya en el aire estable se procede a realizar la ruta que hemos programado mediante nuestro código en Arduino, para ello lo que se hace es que se cambia la referencia en el PID de pitch y de roll para llevarlo en un sentido o en otro, al cambiar la referencia en el PID esto va a hacer que los motores cambien su velocidad para llevar el ángulo de inclinación del dron en ese instante al valor de referencia que nosotros hemos indicado por código.

Como ruta programable en nuestro ejemplo lo que hacemos es primero hacer que el dron vaya a la derecha durante dos segundos para ello se pone la referencia de pitch en 8 grados, después lo que hacemos es llevar nuestro dron a la izquierda y se le pone una referencia de pitch de -8 grados durante un tiempo de dos segundos, luego volvemos a poner nuestro dron horizontal respecto al suelo poniendo una referencia de pitch de 0 grados, ahora hacemos que nuestro dron vaya hacia delante para ello se pone una referencia de roll de 8 grados durante dos segundos, después se hace lo mismo para que

el dron vaya hacia atrás poniendo una referencia de roll de -8 grados durante dos segundos, el siguiente paso es volver a poner el dron horizontal respecto al suelo haciendo que la referencia de roll sea 0 grados, y ya por ultimo bajamos el throttle de forma progresiva durante 12,5s para aterrizar nuestro dron de forma adecuada.

```
454  if(programable==1){//ruta  de vuelo programable.
455    if(counter<100){
456      throttle2=1150;//primero se pone un throttle de 1150 us durante un tiempo de 100*5ms.
457    }
458    else if((counter>=100)&&(counter<600)){
459      throttle2=throttle2+0.7;//durante un tiempo de 500*5ms se sube el throttle de forma progresiva hasta 1475us.
460    }
461    else if((counter>=600)&&(counter<1000)){//se pone una referencia de pitch de 8 grados, el dron se inclina a la derecha.
462      refpitch=8.0;
463    }
464    else if((counter>=1000)&&(counter<1400)){//se pone una referencia de pitch de -8 grados, el dron se inclina a la izquierda.
465      refpitch=-8.0;
466    }
467    else if((counter>=1400)&&(counter<1500)){//se pone una referencia de pitch de 0 grados, el dron se vuelve a poner recto.
468      refpitch=0.0;
469    }
470    else if((counter>=1500)&&(counter<1900)){//se pone una referencia de roll de 8 grados, el dron se inclina hacia delante.
471      refroll=8.0;
472    }
473    else if((counter>=1900)&&(counter<2300)){//se pone una referencia de roll de -8 grados,el dron se inclina hacia atras.
474      refroll=-8.0;
475    }
476    else if((counter>=2300)&&(counter<2400)){//se pone una referencia de roll de 0 grados, el dron se vuelve a poner recto.
477      refroll=0.0;
478    }
479    else if((counter>=2400)&&(counter<4900)){// durante 2500*5ms se se baja el throttle de forma progresiva.
480      throttle2=throttle2-0.1;
481    }
482    else if(counter>=5001){
483      counter=5005;
484    }
```

Figura 78. Ruta de vuelo programable.

Después es necesario realizar las señales PWM igual que en el apartado anterior en el que se calculaba el periodo en alta de las señales PWM teniendo en cuenta el throttle, la salida del PID del pitch, del roll y del yaw, en las siguientes figuras se puede ver como se crean estas señales PWM, además se pone un caso en el que la persona que controla el dron pueda volver a coger los mandos de este por si la ruta programable del dron falla, para volver a coger los mandos lo único que tiene que hacer la persona es subir el joystick del throttle.

```
501  else if(programable==1){
502    if(incomingthrottle!=0){//si la persona sube el joystick del throttle, el dron se vuelve a controlar con el mando.
503      programable=0;
504    }
505    //se calcula el tiempo que las señales tienen que esatr en alta para que se mantenga estable el dron.
506    velocidad =throttle2 + PIDPITCH + PIDROLL +PIDYAW;
507    velocidad1 = throttle2 + PIDPITCH - PIDROLL - PIDYAW;//+50
508    velocidad2 =  throttle2 - PIDPITCH - PIDROLL + PIDYAW;
509    velocidad3 = throttle2 - PIDPITCH + PIDROLL - PIDYAW;
510    counter++;
511  }
```

Figura 79. Programación señales PWM y caso de error para que la persona vuelva a coger los mandos del dron.

```
595   //se pone la señal PWM en alta
596   digitalWrite(motor1, HIGH);  // MOTOR 1
597   digitalWrite(motor2, HIGH);  // MOTOR 2
598   digitalWrite(motor3, HIGH);  // MOTOR 3
599   digitalWrite(motor4, HIGH);  // MOTOR 4
600   timer_start = micros();
601
602   // Cuando se cumpa el tiempo de PWM definido en velocidad, se pasa cada señal a 0 (LOW) para terminar el ciclo PWM.
603   while (digitalRead(motor1) == HIGH || digitalRead(motor2) == HIGH || digitalRead(motor3) == HIGH || digitalRead(motor4) == HIGH) {
604     if (float(timer_start) + velocidad <= float(micros())) digitalWrite(motor1, LOW); // MOTOR 1
605     if (float(timer_start) + velocidad1 <= float(micros())) digitalWrite(motor2, LOW); // MOTOR 2
606     if (float(timer_start) + velocidad2 <= float(micros())) digitalWrite(motor3, LOW); // MOTOR 3
607     if (float(timer_start) + velocidad3 <= float(micros())) digitalWrite(motor4, LOW); // MOTOR 4*/
608   }
```

Figura 80. Código para configurar las señales PWM

8 Resultados

8.1 Medición ángulos con el sensor MPU650

Para saber que nuestro MPU6050 medía bien los ángulos la prueba que se hizo fue poner el dron sobre el ángulo de treinta grados de un cartabón y comprobar que el MPU6050 nos devolvía un valor próximo a 30 grados, esta prueba resulto satisfactoria tanto para el pitch como para el roll como se puede ver en las fotos por lo que nuestro MPU6050 estaba funcionando perfectamente, para obtener los resultados se utiliza la función plotter en Arduino que nos saca una gráfica con los valores que queremos.

Figura 81. Cartabón (dibujonavarres, 2023).

https://dibujonavarres.wordpress.com/2oeso-epva/tema-1-trazados-geometricos-basicos/la-escuadra-y-el-cartabon/

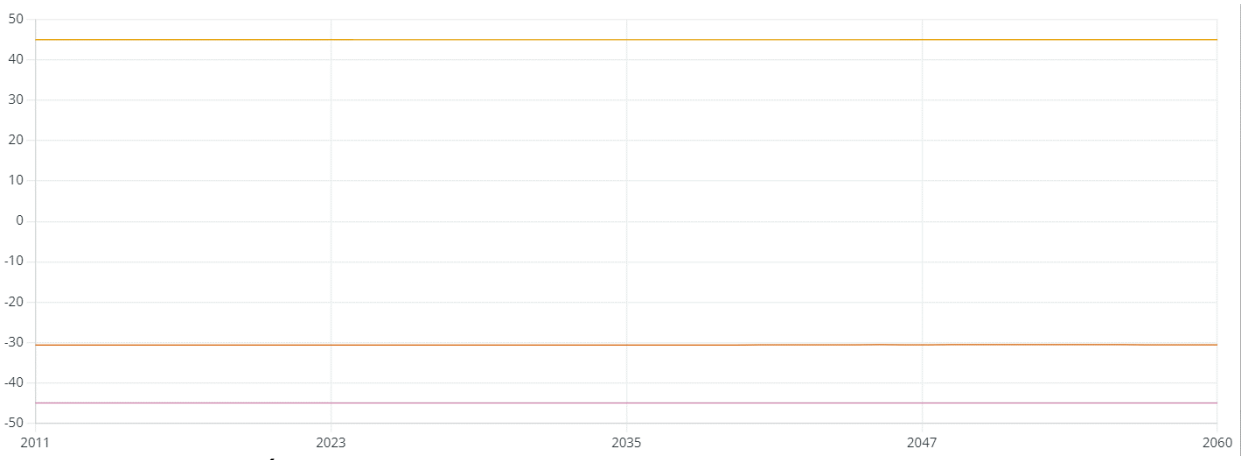


Figura 82. Ángulo de roll de -30 grados (línea naranja oscuro) al ponerlo sobre el ángulo de 30 grados del cartabón.

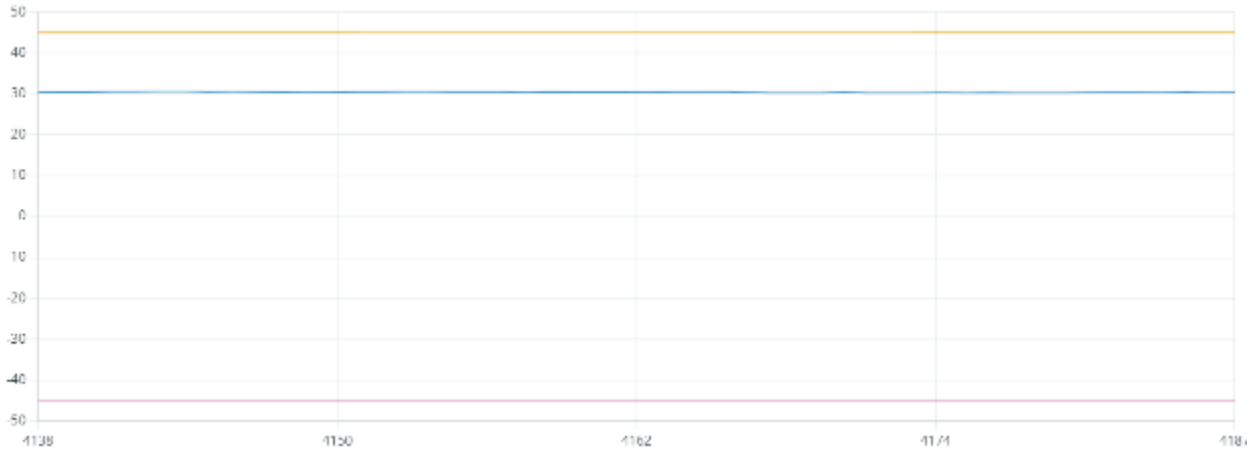


Figura 83. Ángulo de pitch de 30 grados (línea azul oscuro) al ponerlo sobre el ángulo de 30 grados del cartabón.

Para comprobar que el ángulo de yaw se medía correctamente se puso una marca de referencia en el centro de la parte superior del dron, y además aprovechando el ángulo de 90 grados que tiene el cartabón se pusieron dos marcas sobre una mesa, para que si la marca del dron al principio estaba en 0 grados sobre una de las marcas puesta sobre la mesa y luego íbamos a la otra marca que se había puesto en la mesa entonces deberíamos obtener del MPU6050 un ángulo de yaw de 90 grados, y así fue.

Figura 84. Ángulo de yaw de 90 grados (línea verde).

8.2 Generar señales PWM

Para comprobar que las señales PWM se generan correctamente se envió desde el mando un throttle de 1500 microsegundos y se visualizó por el osciloscopio la señal PWM de 200Hz con un tiempo en alta la señal cuadrada de 1.5ms, y se puede observar en la siguiente foto como el resultado es el esperado.

Figura 85. Señal PWM para el funcionamiento de los motores.

8.3 Conversores ADC

Para comprobar que los ADC del mando funcionan correctamente la prueba que se realiza es empezar con el joystick en con un valor de 1.7V, por la mitad del valor total que convierte el ADC del ESP32 y empezar a subir el joystick hasta que el voltaje de salida sea de 3.3V que en digital es 4096 y luego se baja el joystick hasta llegar a los 0V, se va a visualizar con el plotter de Arduino todo lo dicho anteriormente para ver que funciona correctamente todos los joysticks y los ADC del ESP32.

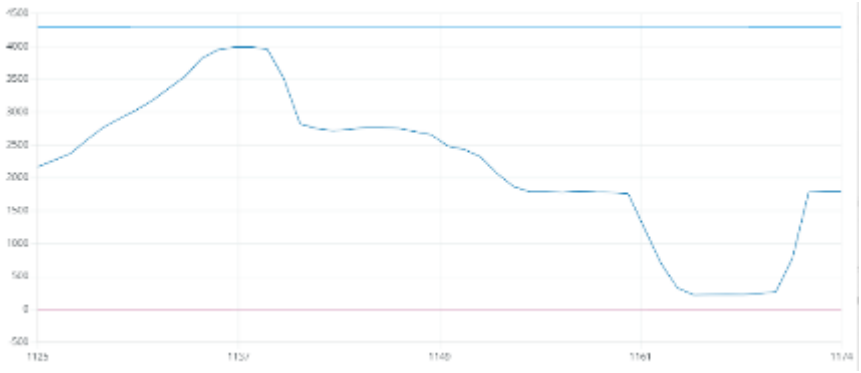


Figura 86. Voltaje de salida del joystick de roll (línea azul oscuro).

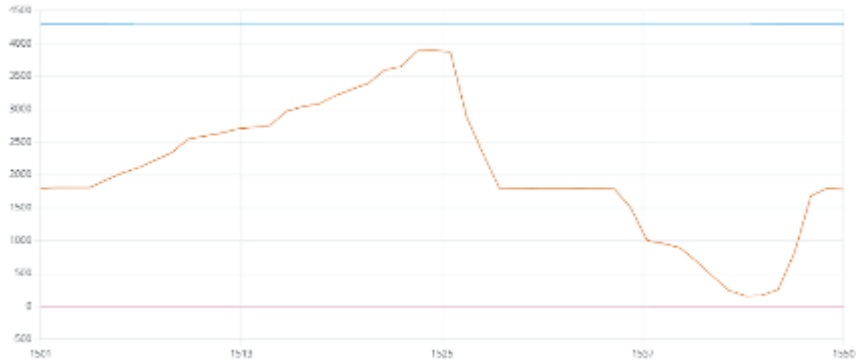


Figura 87. Voltaje de salida del joystick de pitch (línea naranja).

Para comprobar el funcionamiento del joystick del throttle se sube el throttle desde el valor más bajo de 0V al valor más alto de 3.3V tal y como se puede ver en la siguiente imagen.

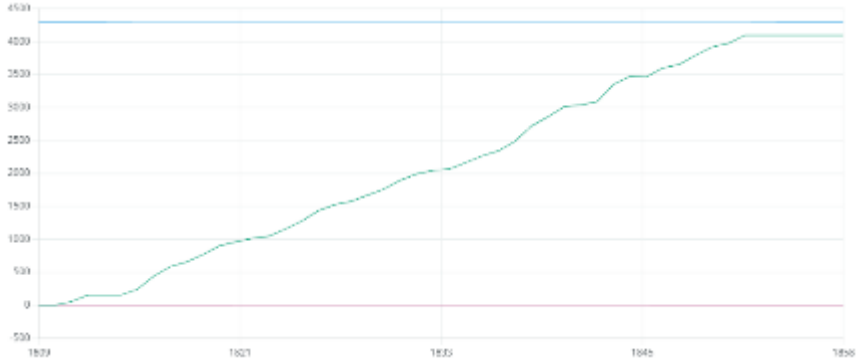


Figura 88. Voltaje de salida del joystick de throttle (línea verde).

Figura 89. Voltaje de salida del joystick de yaw (línea amarilla).

8.4 Comunicaciones protocolo ESP-NOW

Para comprobar que el protocolo ESP-NOW funciona correctamente nos fijamos en que los datos de inclinación de pitch, roll y el ángulo de rotación yaw que tiene el dron en cada instante, para ello nos fijamos en la pantalla OLED que nos muestra estos datos por pantalla, para comprobar que funcionan bien en la pantalla se han realizado las mismas pruebas que en el apartado de medir los ángulos correctamente.

Figura 90. Pantalla OLED con la inclinación de 30 grados en el eje de roll.

Figura 91. Pantalla OLED con la inclinación de 30 grados en el eje de pitch.

Figura 92. Pantalla OLED con la inclinación de 90 grados en el yaw.

8.5 PIDs en estático

Para comprobar que los PIDs están funcionando correctamente lo que se va a proceder es a inclinar el dron hacia delante o hacia atrás o hacia un lado y ver como responden los motores para contrarrestar esta inclinación.

Primero vamos a comprobar el PID de roll, por lo que vamos a inclinar el dron a la derecha y se va a ver como los motores del lado derecho (línea azul oscuro y amarilla) suben su velocidad y los motores del lado izquierdo (línea naranja oscuro y verde) bajan de velocidad para contrarrestar esa inclinación, ya que los motores hacen que el aire vaya hacia abajo y si giran

más rápido ese lado subirá.

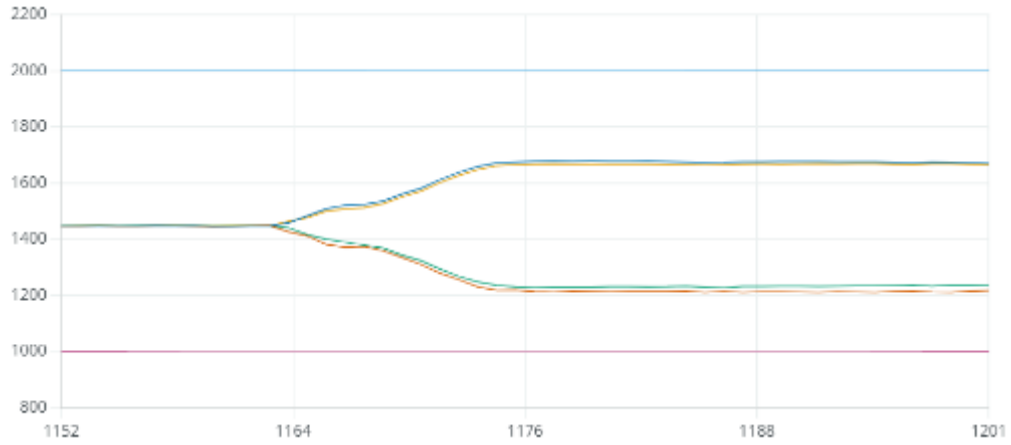


Figura 93. Velocidad al variar el ángulo de inclinación de roll.

Ahora para comprobar que el PID del pitch funciona correctamente lo que se va a hacer es inclinar hacia delante el dron y ver como la velocidad de los motores delanteros (línea naranja oscuro y azul oscuro) sube y la de los motores traseros (línea amarilla y verde) baja para contrarrestar esta inclinación, ya que los motores hacen que el aire vaya hacia abajo y si giran más rápido ese lado subirá.

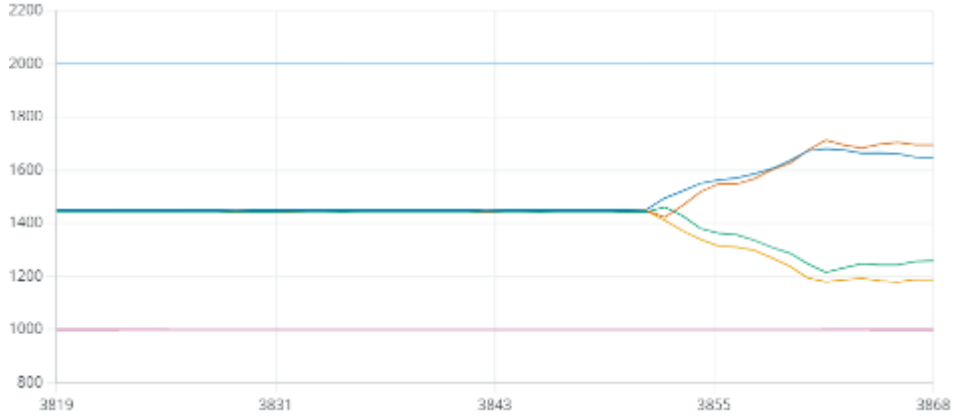


Figura 94. Velocidad al variar el ángulo de inclinación de pitch.

Ahora para comprobar que el PID del yaw funciona correctamente lo que se va a hacer es girar el dron en sentido horario y ver como la velocidad de los motores que gira en sentido horario (línea naranja oscuro y amarilla) sube y la de los motores que giran en sentido antihorario (línea verde y azul oscuro) baja para contrarrestar esta rotación y mantener el dron recto.

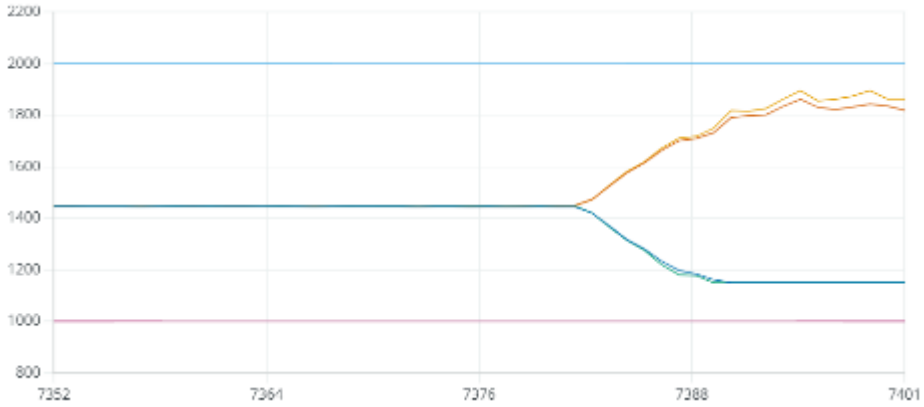


Figura 95. Velocidad al variar la rotación en el eje z.

8.6 PIDs en vuelo

Para comprobar que los PIDs durante el vuelo están funcionando bien se han enviado al mando tanto la inclinación del pitch como del roll como la rotación en el eje z, se han sacado varias capturas a estos valores para que veamos que siempre suelen estar en torno a 0 grados la inclinación de pitch y roll, además la velocidad angular del eje z tiende a estar cerca de 0dps indicando que nuestro dron está horizontal respecto al suelo y estable.

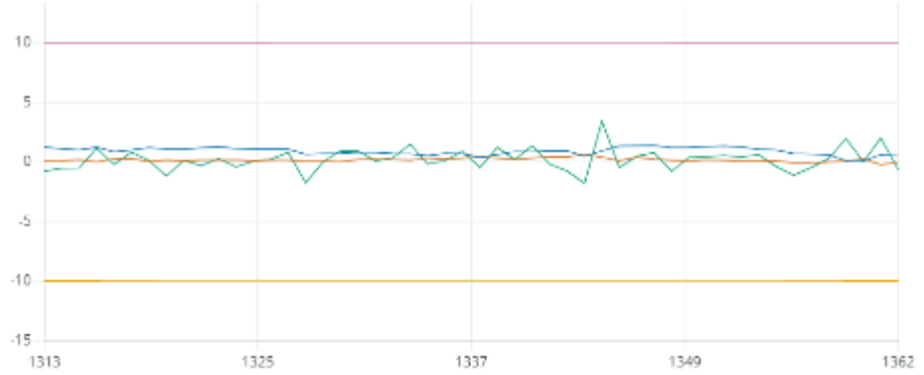


Figura 96. Captura 1 ángulos de inclinación pitch (línea azul oscuro), roll (línea naranja oscuro) y velocidad de rotación eje z (línea verde), tanto el ángulo de pitch como de roll tienden a 0 grados y la velocidad angular del eje z tiende a 0dps.

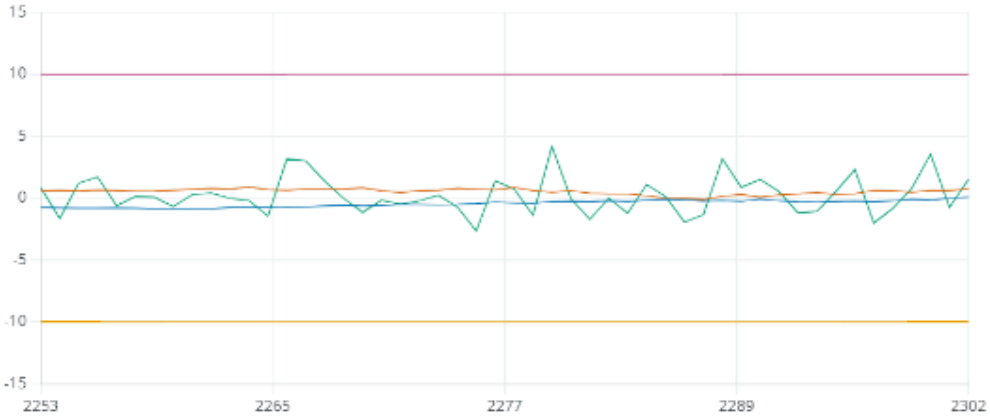


Figura 97. Captura 2 ángulos de inclinación pitch (línea azul oscuro), roll (línea naranja oscuro) y velocidad de rotación eje z (línea verde), tanto el ángulo de pitch como de roll tienden a 0 grados y la velocidad angular del eje z tiende a 0dps.

.

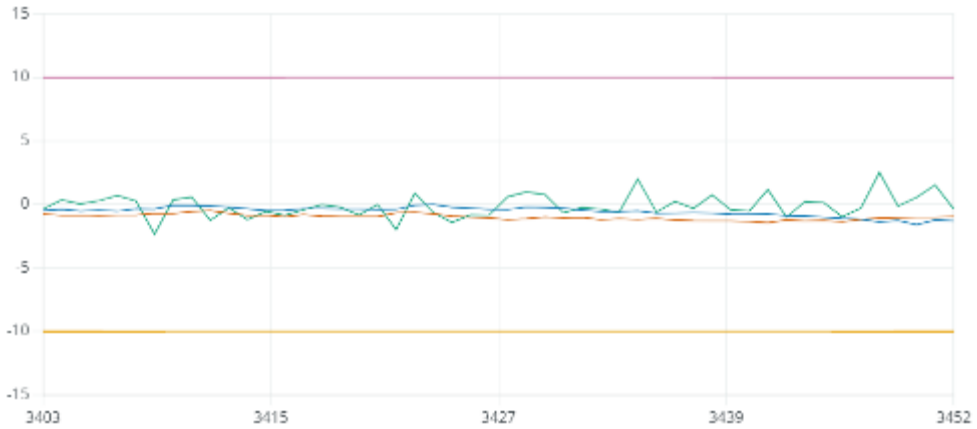


Figura 98. Captura 3 ángulos de inclinación pitch (línea azul oscuro), roll (línea naranja oscuro) y velocidad de rotación eje z (línea verde), tanto el ángulo de pitch como de roll tienden a 0 grados y la velocidad angular del eje z tiende a 0dps.

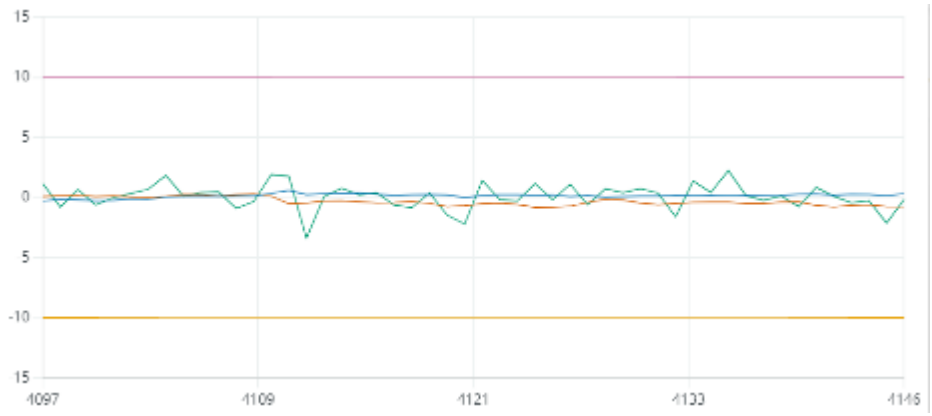


Figura 99. Captura 4 ángulos de inclinación pitch (línea azul oscuro), roll (línea naranja oscuro) y velocidad de rotación eje z (línea verde), tanto el ángulo de pitch como de roll tienden a 0 grados y la velocidad angular del eje z tiende a 0dps.

8.7 Ruta de vuelo programable

Para comprobar que nuestra ruta de vuelo funciona correctamente se monitoreo gracias al plotter de Arduino enviando las señales de pitch, roll y velocidad angular en el eje z al mando, la primera comprobación es cuando ponemos por código que la referencia de pitch sea 8 grados para que el dron se desplace a la derecha, y se puede ver en la primera captura como el pitch del dron (línea azul oscuro) se pone en 8 grados.

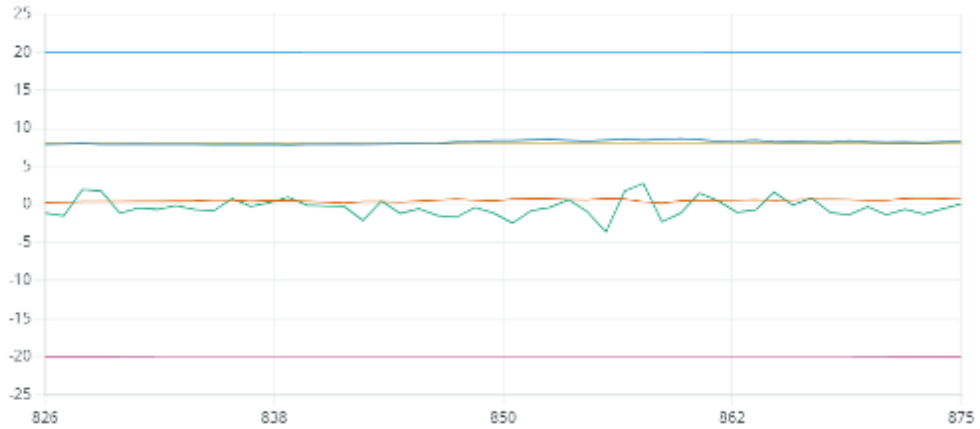


Figura 100. Captura 1 ángulos de inclinación pitch (línea azul oscuro), roll (línea naranja oscuro) y velocidad de rotación eje z (línea verde), se observa como la inclinación pitch esta alrededor de los 8 grados de inclinación para poder desplazar el dron hacia la derecha.

La segunda comprobación es cuando ponemos por código que la referencia de pitch sea -8 grados para que el dron se desplace a la izquierda, y se puede ver en la segunda captura como el pitch del dron (línea azul oscuro) se pone en -8 grados.

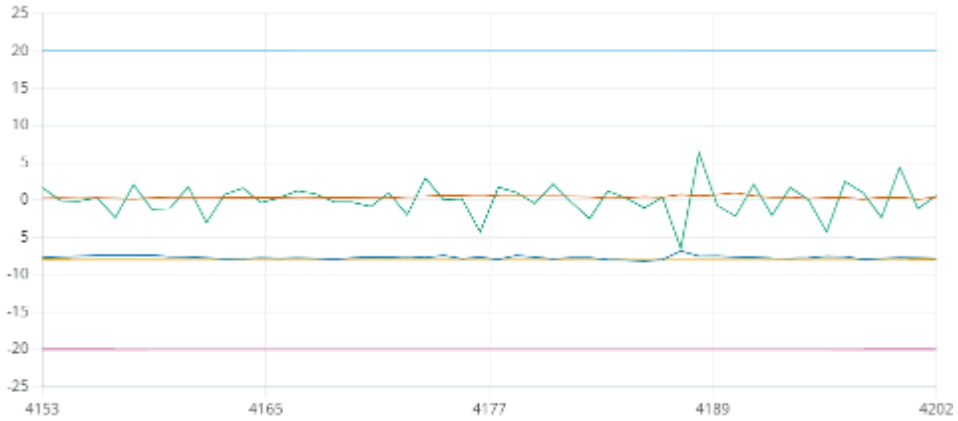


Figura 101. Captura 2 ángulos de inclinación pitch (línea azul oscuro), roll (línea naranja oscuro) y velocidad de rotación eje z (línea verde), se observa como la inclinación pitch esta alrededor de los -8 grados de inclinación para poder desplazar el dron hacia la izquierda.

La tercera comprobación es cuando ponemos por código que la referencia de roll sea 8 grados para que el dron se desplace hacia delante, y se puede ver en la tercera captura como el roll del dron (línea naranja oscuro) se pone en 8 grados, también se puede observar como el ángulo de pitch (línea azul oscuro) volvió a estar en 0 grados.

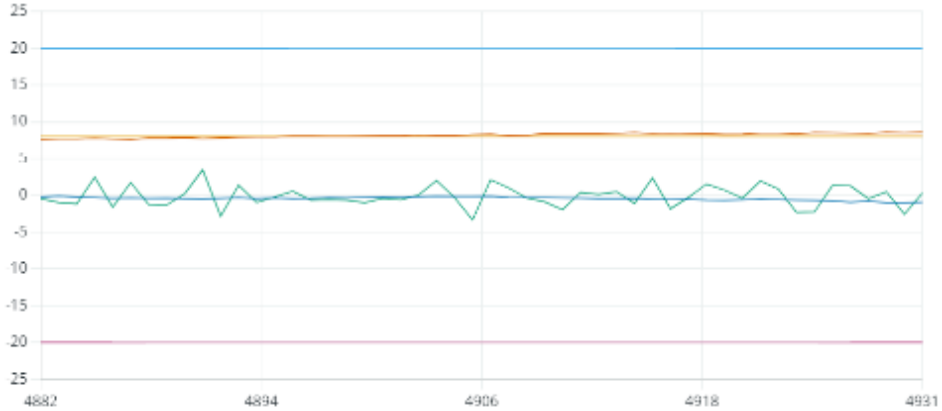


Figura 102. Captura 3 ángulos de inclinación pitch (línea azul oscuro), roll (línea naranja oscuro) y velocidad de rotación eje z (línea verde), se observa como la inclinación roll esta alrededor de los 8 grados de inclinación para poder desplazar el dron hacia la delante.

La cuarta y última comprobación es cuando ponemos por código que la referencia de roll sea -8 grados para que el dron se desplace hacia atrás, y se puede ver en la cuarta captura como el roll del dron (línea naranja oscuro) se pone en -8 grados.

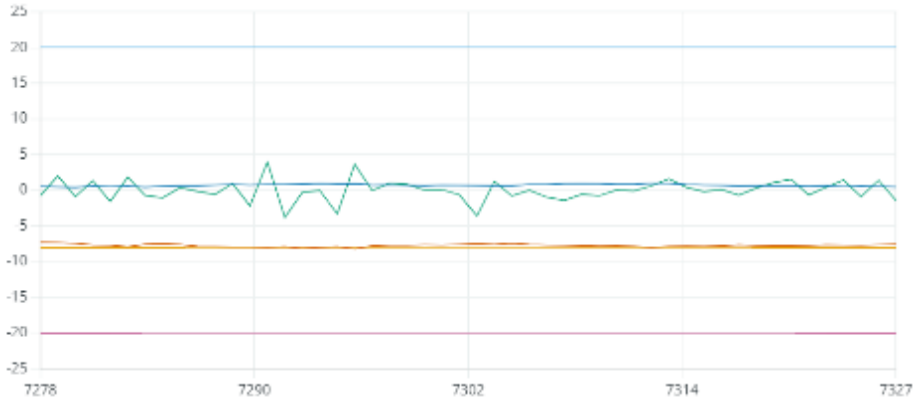


Figura 103. Captura 4 ángulos de inclinación pitch (línea azul oscuro), roll (línea naranja oscuro) y velocidad de rotación eje z (línea verde), se observa como la inclinación roll esta alrededor de los -8 grados de inclinación para poder desplazar el dron hacia atrás.

9 Presupuesto

Componente	Cantidad	Precio/Unidad (Amazon)
ESP-WROOM-32.	2	8,99€
Sensor MPU-6050.	1	6,79€
Motores KEESIN sin escobillas RC 2200Kv de 30ª con pack de hélices.	4	23,99€
Placas de soldadura.	1	0,6€
Pantalla OLED de 0,96 pulgadas I2C SSD1306.	1	6,99€
Módulo Joystick de Arduino.	2	4,99€
Cargador LIPO B3 para 2S y 3S.	1	19,99€
Batería Roaring Top 2200mAh 35C 3S 11.1V.	1	17,5€
Componentes pasivos (2 resistencias, 1 diodo 1N4007, 1 diodo led rojo).		Muy bajo.
Placa de cobre para el montaje mecánico.	1	6,5€
Placa de fibra de carbono para el montaje mecánico	1	13,61€
Palo cuadrado de madera 13,2mm * 30cm.	1	1€
Surtido de tornillos de cabeza plana con tuercas y arandelas hexagonales.	1	7,99

Tabla 1. Costes de fabricación del dron consultado en Amazon.

El gasto total para poder realizar el dron es de 204,89€, aunque en este precio no está incluido el precio de comprar unos pocos cables para realizar ciertas conexiones, en el caso de que no se posea cables sería necesario comprar un kit de 7€ que habría que sumar al precio anterior, por lo que el costo total sería de aproximadamente 212€. Además, hay que tener en cuenta el trabajo realizado por la persona que realiza el proyecto, si se trabajó 300 horas y un ingeniero electrónico junior cobra de media 15 euros/hora eso hace un gasto de personal de 4500.

10 Conclusiones y líneas futuras

10.1 Conclusiones

La conclusión de este proyecto final de carrera es que se ha conseguido realizar el objetivo de que el dron se mantenga estable en el aire y que se pueda controlar de forma adecuada los motores generando la señal PWM como se puede ver en el apartado de resultados, además con el mando se ha podido dirigir de forma adecuada el dron para hacerle que vaya en la dirección que se le indica, también se ha conseguido de forma adecuada visualizar por el display la posición del pitch, roll, yaw y el voltaje que tiene la batería en cada instante para saber cuándo hay que aterrizar el dron, y el led se enciende adecuadamente cuando el voltaje de la batería baja de 9V que indica que la batería está prácticamente descargada, además gracias a la buena estabilidad que adquiere nuestro dron se ha conseguido un aterrizaje muy estable y seguro, también se consiguió el ultimo objetivo del proyecto que consistía en programar un código para que nuestro dron pudiese seguir una ruta de vuelo programable de forma satisfactoria en función de los ángulos de pitch y roll que se le indicase en cada momento.

10.2 líneas futuras

Como línea futura de trabajo se podría poner el sensor barométrico BMP180 que nos permite medir la altura respecto al suelo para indicar a nuestro dron que siempre se mantenga a una distancia determinada del suelo y así proporcionar más estabilidad a nuestro dron, desde ahí el siguiente paso sería realizar un dron que vuele en modo acrobático que fuese capaz de realizar ciertas acrobacias y que tuviese un vuelo más profesional, como mejora al trabajo realizado lo ideal sería hacer la base en una impresora 3D con PLA para así reducir el excesivo peso que tenía nuestro dron, y así si el dron se estrella este material es más resistente que la fibra de carbono con la que hicimos nuestra base, además de ser más aerodinámico, y para mejorar la parte de la ruta de vuelo programable se podría sustituir el ESP32 que estamos utilizando por un ESP32 con módulo GPS, y así poder indicar al dron que vaya hasta cierta posición de forma más exacta y precisa.

Apéndice A

Software del mando (Arduino)

```
#include <esp_now.h>
#include <WiFi.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128  // pixeles ancho display OLED.
#define SCREEN_HEIGHT 64  // pixeles largo display OLED.
#define led 13 //pin del Led del mando.

// declarar SSD1306 display conectado por I2C I2C (SDA, SCL pins).
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

// dirección MAC del receptor.
uint8_t broadcastAddress[] = {0xA0, 0xB7, 0x65, 0x49, 0x11, 0x9C};

// Declaración de variables.
int roll_sin=0;
int roll=0;
int pitch_sin=0;
int pitch=0;
int throttle_sin=0;
int throttle=0;
int yaw_sin=0;
int yaw=0;
float ref=0, ref1=0,ref2=0;
long out=0,out1=0,out2=0;
float voltaje_bateria;
long timer_dentro=0;
// Definir variables para las variables que nos llegan desde el mando.
float incomingroll;
float incomingpitch;
float incomingyaw;
float voltaje_bateria_incoming;
String success;

//estructura para recibir datos.
typedef struct struct_message {
    float incomingpitch;
    float incomingroll;
    float incomingyaw;
    float voltaje_bateria_incoming;
} struct_message;
//estructura para enviar los datos del mando.
typedef struct struct_message2 {
    float roll;
    float pitch;
    int throttle;
    float yaw;
} struct_message2;
// definir dos estructuras de datos.
```

```
struct_message incomingReadings;
struct_message2 messagesend;
//establecer la conexión del protocolo ESP-NOW.
esp_now_peer_info_t peerInfo;

//función que nos dice si los datos se enviaron con exito.
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" :
"Delivery Fail");
  if (status ==0){
    success = "Entrega con exito :)";
  }
  else{
    success = "Entrega con fallo :(";
  }
}

// Función para recibri los datos de pitch, roll, yaw y voltaje de la batería
del dron.
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len) {
  memcpy(&incomingReadings, incomingData, sizeof(incomingReadings));
  Serial.print("Bytes received: ");
  Serial.println(len);
  incomingpitch = incomingReadings.incomingpitch;
  incomingroll = incomingReadings.incomingroll;
  incomingyaw = incomingReadings.incomingyaw;
  voltaje_bateria = incomingReadings.voltaje_bateria_incoming;

}

void setup() {
  //Iniciamos puerto serie.
  Serial.begin(115200);
  //poner pin 13 LED como salida.
  pinMode(led,OUTPUT);
  digitalWrite(led, LOW);
  // Inicializamos la pantalla OLED.
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;);
  }

  // Se pone este ESP32 como estación de Wifi.
  WiFi.mode(WIFI_STA);

  // Inicializamos protocolo de comunicaciones ESP-NOW.
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }

  // se envían datos desde el mando al dron y se comprueba si se envió
correctamente.
  esp_now_register_send_cb(OnDataSent);

  //Se registra la conexión, el canal va a ser el 0 y no se va a encriptar el
mensaje.
```

```
  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  //Añadir conexión.
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }
  //Registra que se reciben los datos.
  esp_now_register_recv_cb(OnDataRecv);
  //Retardo 100ms.
  delay(100);
}

void loop() {

  // leemos los valores analógicos que nos da los potenciometros de los
joysticks.
  roll_sin=analogRead(34);
  pitch_sin=analogRead(35);
  throttle_sin=analogRead(36);
  yaw_sin=analogRead(39);
  pitch=0.7 * pitch + 0.3 * pitch_sin;//filtrado para eliminar el ruido de
las señales.
  if((pitch <= 1950) && (pitch >= 1850)){//interpolación lineal.
    out1=0;
  }
  if(pitch > 1950){
      out1 = map( pitch, 1950, 4095, 0, 1000 );
    }
    else if(pitch < 1850){
      out1 = map( pitch, 0, 1850, -1000, 0 );
    }

    ref1=float(out1)/100.0;
  roll=0.4 * roll +0.6 *roll_sin;//filtrado para eliminar el ruido de las
señales.
  if ((roll <= 1950)&&(roll>=1850)){//interpolación lineal.
    out=0;
  }
  if(roll > 1950){
      out = map( roll, 1950, 4095, 0, 1000 );
    }
    else if(roll < 1850){
      out = map( roll, 0, 1850, -1000, 0 );
    }
    ref= float(out)/100.0;
  throttle=0.7 * throttle + 0.3 * throttle_sin;//filtrado para eliminar el
ruido de las señales.
  yaw=0.4 * yaw + 0.6 * yaw_sin;//filtrado para eliminar el ruido de las
señales.
  if((yaw <=1940) && (yaw>=1840)){//interpolación lineal.
    out2=0;
  }
  if(yaw > 1940){
      out2 = map( yaw, 1940, 4095, 0, -100 );
    }
```

```
  else if(yaw < 1840){
      out2 = map( yaw, 0, 1840, 100, 0 );
    }
    ref2= float(out2)/100;
  if((voltaje_bateria <= 9.0) &&(voltaje_bateria>=3.0)){//encender LED si el
voltaje de la batería baja.
    digitalWrite(led, HIGH);
  }
  else{
    digitalWrite(led, LOW);
  }

  //Se envía mensaje por el protocolo ESP-NOW.
  messagesend.roll=ref;
  messagesend.pitch=ref1;
  messagesend.throttle=throttle;//throttle
  messagesend.yaw=ref2;

  esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &messagesend,
sizeof(messagesend));
  //Indica si los datos se enviaron satisfactoriamente.
  if (result == ESP_OK) {
    Serial.println("Sent with success");
  }
  else {
    Serial.println("Error sending the data");
  }
  updateDisplay();

}


void updateDisplay(){
  // imprimimos las lecturas en el OLED Display
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor(0, 0);
  display.println("Lecturas:");
  display.setCursor(0, 15);
  display.print("roll: ");
  display.print(incomingroll);
  display.setCursor(0, 25);
  display.print("pitch: ");
  display.print(incomingpitch);
  display.setCursor(0, 35);
  display.print("yaw: ");
  display.print(incomingyaw);
  display.setCursor(0, 45);
  display.print("bateria:");
  display.print(voltaje_bateria);
  display.print("V");
  display.setCursor(0, 56);
  display.print(success);
  display.display();
}
```

Apéndice B

Software del Dron (Arduino)

```
#include <esp_now.h>
#include <WiFi.h>
#include <Wire.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <cmath>
#define motor1 23
#define motor2 4
#define motor3 13
#define motor4 32
#define voltaje 34
//Crear objeto del sensor MPU6050.
Adafruit_MPU6050 mpu;
//declarar variables.
int programable=1;
float counter=0;
long timer_dentro=0;
unsigned long timer_start=0;
unsigned int i=0;
int start=0,error_mot=0, error_mot1=0, start1=0;
float gyro_x=0, gyro_y=0, gyro_z=0;
float gyro_x_filt=0, gyro_y_filt=0, gyro_z_filt=0;
unsigned long tiempo_anterior, tiempo, dt;
float acc_x, acc_y, acc_z, acc_total_vector;
float acc_x_filt=0, acc_y_filt=0, acc_z_filt=0;
float acc_x_cal, acc_y_cal, acc_z_cal;
float gyro_x_cal, gyro_y_cal, gyro_z_cal;
unsigned long loop_timer;
float angle_pitch=0,angle_roll=0, angle_yaw=0;
float angle_pitch_acc=0,angle_roll_acc=0;
boolean set_gyro_angles;
float PIDROLL=0, PIDYAW=0, PIDPITCH=0, error_roll, previous_error_roll,
error_pitch, previous_error_pitch, error_yaw, previous_error_yaw;
float pid_p_pitch=0, pid_p_roll=0,pid_p_yaw=0;
float pid_i_roll=0,pid_i_pitch=0,pid_i_yaw=0;
float pid_d_pitch=0, pid_d_roll=0;
//////////////////Constantes PID/////////////////
float kppitch=4;
float kproll=4;
float kpyaw=1.3;
float kipitch=0.09;
float kiroll=0.09;
float ki_yaw=0.05;
float kdpitch=150.00;
float kdroll=150.00;
float throttle=1000.0;
float throttle2=1000.0;
float velocidad = 1000.0;
```

```
float velocidad1 = 1000.0;
float velocidad2 = 1000.0;
float velocidad3 = 1000.0;
float refroll=0, refpitch=0, refyaw=0, refyaw1=0;
int incomingthrottle=0;
// Dirección Mac del ESP32 del mando.
uint8_t broadcastAddress[] = {0xA0, 0xB7, 0x65, 0x4F, 0x38, 0xC8};
float incomingroll=0;
float incomingpitch=0;
float incomingyaw=0;
long offset=0;
float tension_bateria=0, tension_bateria_filt=0;
//estructura para recibir datos.
typedef struct struct_message2 {
    float incomingroll;
    float incomingpitch;
    int incomingthrottle;
    float incomingyaw;
} struct_message2;
struct_message2 messagereceive;

String success;
//estructura para enviar datos.
typedef struct struct_message {
    float angle_pitch;
    float angle_roll;
    float angle_yaw;
    float tension_bateria_filt;

} struct_message;
struct_message MPU6050Readings;
//se crea la conexión.
esp_now_peer_info_t peerInfo;

// Función para comprobar que se envían bien los datos.
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" :
"Delivery Fail");
  if (status ==0){
    success = "Delivery Success :)";
  }
  else{
    success = "Delivery Fail :(";
  }
}

// Función para recibir datos.
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len) {
  memcpy(&messagereceive, incomingData, sizeof(messagereceive));
  /*Serial.print("Bytes received: ");
  Serial.println(len);*/
  refroll = messagereceive.incomingroll;
  refpitch = messagereceive.incomingpitch;
  incomingthrottle=messagereceive.incomingthrottle;
  refyaw=messagereceive.incomingyaw;

}
```

```
void pidroll(){// Función para calcular el PID del roll.

  error_roll = angle_roll - refroll;//se calcula error del roll.
  pid_p_roll = kproll*error_roll;//se calcula parte proporcional.
  pid_i_roll += (kiroll*error_roll);//se calcula parte integral.
  if(pid_i_roll>=200.0){//se limita la parte integral.
    pid_i_roll=200.0;
  }
  else if(pid_i_roll<=-200.0){
    pid_i_roll=-200.0;
  }
  pid_d_roll = kdroll*(error_roll - previous_error_roll);//se calcula parte
derivativa.

  PIDROLL = pid_p_roll + pid_i_roll + pid_d_roll;//se calcula PID.

  if(PIDROLL < -400.0)// se limita el valor del PID.
  {
    PIDROLL = -400.0;
  }
  if(PIDROLL > 400.0)
  {
    PIDROLL = 400.0;
  }

  previous_error_roll = error_roll;
}
void pidpitch(){// Función para calcular el PID del pitch.

  error_pitch = angle_pitch - refpitch;//se calcula el error de picth.
  pid_p_pitch = kppitch * error_pitch;//se calcula la parte proporcional.
  pid_i_pitch += (kipitch * error_pitch);//se calcula la parte integral.
  if(pid_i_pitch>=200.0){//se limita la parte integral.
    pid_i_pitch=200.0;
  }
  if(pid_i_pitch<=-200.0){
    pid_i_pitch=-200.0;
  }
  pid_d_pitch = kdpitch*(error_pitch - previous_error_pitch);//se calcula la
parte derivativa.
  PIDPITCH = pid_p_pitch + pid_i_pitch + pid_d_pitch;//se calcula el PID.
  if(PIDPITCH < -400.0)//se limita el PID.
  {
    PIDPITCH = -400.0;
  }
  if(PIDPITCH > 400.0)
  {
    PIDPITCH = 400.0;
  }

  previous_error_pitch = error_pitch;


}
void pidyaw(){// Función para calcular el PID del yaw.
  refyaw1=0.0;
  error_yaw = gyro_z_filt - refyaw1;//se calcula el error de yaw.
  pid_p_yaw = kpyaw * error_yaw;//se calcula la parte proporcional.
```

```
  pid_i_yaw += (ki_yaw * error_yaw);//se calcula la parte integral.
  if(pid_i_yaw>=380.0){//se limita la parte integral
    pid_i_yaw=380.0;
  }
  if(pid_i_yaw<=-380.0){
    pid_i_yaw=-380.0;
  }

  PIDYAW = pid_p_yaw + pid_i_yaw ;//se calcula el PID del yaw

  if(PIDYAW < -450.0)//se limita el PID del yaw
  {
    PIDYAW=-450.0;
  }
  if (PIDYAW > 450.0)
  {
    PIDYAW=450.0;
  }

}


void setup() {
  //Inicializamos el puerto serie.
  Serial.begin(115200);
  //Se configura este ESP32 como una estación Wifi.
  WiFi.mode(WIFI_STA);
  //comprueba si la comunicación entre los esp32 se estableció de forma
correcta.
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }

  // se envían datos desde el mando al dron y se comprueba si se envió
correctamente.
  esp_now_register_send_cb(OnDataSent);

  // Registrar conexión.
  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.channel = 0;//canal 0.
  peerInfo.encrypt = false;//no se encripta.

  // Add peer
  if (esp_now_add_peer(&peerInfo) != ESP_OK){//comprueba si la conexión se
estableció correctamente.
    Serial.println("Failed to add peer");
    return;
  }
  //Registra que se reciben los datos.
  esp_now_register_recv_cb(OnDataRecv);
  //inicializamos los tiempos a 0.
  tiempo_anterior=0;
  tiempo=0;

  while (!Serial)
    delay(10);//Espera a que se inicie el puerto serie.
  pinMode(voltaje,INPUT);    //pin 34 como entrada.
```

```
  pinMode(motor1, OUTPUT);   // Declarar motor 1.
  pinMode(motor2, OUTPUT);   // Declarar motor 2.
  pinMode(motor3, OUTPUT);   // Declarar motor 3.
  pinMode(motor4, OUTPUT);   // Declarar motor 4.
  digitalWrite(motor1, LOW); // Motor 1 LOW por seguridad.
  digitalWrite(motor2, LOW); // Motor 2 LOW por seguridad.
  digitalWrite(motor3, LOW); // Motor 3 LOW por seguridad.
  digitalWrite(motor4, LOW); // Motor 4 LOW por seguridad.

  Serial.println("Adafruit MPU6050 test!");

  //comprobamos que el MPU6050 funciona correctamente.
  if (!mpu.begin()) {//si no lo encuentra nos quedamos en el bucle.
    Serial.println("Failed to find MPU6050 chip");
    while (1) {
      delay(10);
    }
  }
  Serial.println("MPU6050 Found!");//encontrado

  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);//establecer el rango del
acelerómetro en +-8G.
  Serial.print("Accelerometer range set to: ");
  switch (mpu.getAccelerometerRange()) {
  case MPU6050_RANGE_2_G:
    Serial.println("+-2G");
    break;
  case MPU6050_RANGE_4_G:
    Serial.println("+-4G");
    break;
  case MPU6050_RANGE_8_G:
    Serial.println("+-8G");
    break;
  case MPU6050_RANGE_16_G:
    Serial.println("+-16G");
    break;
  }
  mpu.setGyroRange(MPU6050_RANGE_500_DEG);//establecer rango del giroscopio.
  Serial.print("Gyro range set to: ");
  switch (mpu.getGyroRange()) {
  case MPU6050_RANGE_250_DEG:
    Serial.println("+- 250 deg/s");
    break;
  case MPU6050_RANGE_500_DEG:
    Serial.println("+- 500 deg/s");
    break;
  case MPU6050_RANGE_1000_DEG:
    Serial.println("+- 1000 deg/s");
    break;
  case MPU6050_RANGE_2000_DEG:
    Serial.println("+- 2000 deg/s");
    break;
  }

  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);//configurar filtro paso bajo
digital.
  Serial.print("Filter bandwidth set to: ");
  switch (mpu.getFilterBandwidth()) {
```

```
  case MPU6050_BAND_260_HZ:
    Serial.println("260 Hz");
    break;
  case MPU6050_BAND_184_HZ:
    Serial.println("184 Hz");
    break;
  case MPU6050_BAND_94_HZ:
    Serial.println("94 Hz");
    break;
  case MPU6050_BAND_44_HZ:
    Serial.println("44 Hz");
    break;
  case MPU6050_BAND_21_HZ:
    Serial.println("21 Hz");
    break;
  case MPU6050_BAND_10_HZ:
    Serial.println("10 Hz");
    break;

  case MPU6050_BAND_5_HZ:
    Serial.println("5 Hz");
    break;
  }
  //Los motores empezarán a girar para realizar la calibración.
  while(throttle<1150){
    if((incomingthrottle > 1743)){//1250
        offset = map( long(incomingthrottle), 1744, 4095, 14000, 17000 );//-5
        throttle=float(offset)/10;
        }
    else {//1600
        offset = map( long(incomingthrottle), 0, 1743, 11000, 14000 );
        throttle=float(offset)/10;
    }
    if(incomingthrottle==0){
      throttle=1000;
    }
    digitalWrite(motor1, HIGH);
    digitalWrite(motor2, HIGH);
    digitalWrite(motor3, HIGH);
    digitalWrite(motor4, HIGH);
     timer_start = micros();
  // Cuando se cumpa el tiempo de PWM definido en throttle, se pasa cada
señal a 0 (LOW) para terminar el ciclo PWM.
    while (digitalRead(motor1) == HIGH || digitalRead(motor2) == HIGH ||
digitalRead(motor3) == HIGH || digitalRead(motor4) == HIGH) {
      if (float(timer_start) + throttle <= float(micros()))
digitalWrite(motor1, LOW); // MOTOR 1
      if (float(timer_start) + throttle <= float(micros()))
digitalWrite(motor2, LOW); // MOTOR 2
      if (float(timer_start) + throttle <= float(micros()))
digitalWrite(motor3, LOW); // MOTOR 3
      if (float(timer_start) + throttle <= float(micros()))
digitalWrite(motor4, LOW); // MOTOR 4
    }

    delayMicroseconds(5000-throttle);

  }
```

```
  Serial.println("empieza a calibrar");
  loop_timer=micros();
  for (int cal_int = 0; cal_int < 4000 ; cal_int ++){   //calibración.
      if((refroll<=-8.1)){
        throttle=1000;
      }
      sensors_event_t a, g, temp;
      mpu.getEvent(&a, &g, &temp);
      gyro_x_cal += (g.gyro.x * 57.29578); //se transforma de rad/s a
grados/s.
      gyro_y_cal += (g.gyro.y * 57.29578);
      gyro_z_cal += (g.gyro.z * 57.29578);
      acc_x_cal += a.acceleration.x;
      acc_y_cal += a.acceleration.y;
      acc_z_cal += a.acceleration.z;
      //la calibración se hace con los motores girando.
      digitalWrite(motor1, HIGH);  // MOTOR 1
      digitalWrite(motor2, HIGH);  // MOTOR 2
      digitalWrite(motor3, HIGH);  // MOTOR 3
      digitalWrite(motor4, HIGH);  // MOTOR 4
      timer_start = micros();
    // Cuando se cumpa el tiempo de PWM definido en throttle, se pasa cada
señal a 0 (LOW) para terminar el ciclo PWM.
    while (digitalRead(motor1) == HIGH || digitalRead(motor2) == HIGH ||
digitalRead(motor3) == HIGH || digitalRead(motor4) == HIGH) {
      if (float(timer_start) + throttle <= float(micros())) digitalWrite(motor1, LOW);
// MOTOR 1
      if (float(timer_start) + throttle <= float(micros())) digitalWrite(motor2, LOW);
// MOTOR 2
      if (float(timer_start) + throttle <= float(micros())) digitalWrite(motor3, LOW);
// MOTOR 3
      if (float(timer_start) + throttle <= float(micros())) digitalWrite(motor4, LOW);
// MOTOR 4
    }
    //delayMicroseconds(5000-throttle);
    while(micros() - loop_timer < 5000) ;
//Wait until the loop_timer reaches 5000us (200Hz) before starting the next
loop
    loop_timer = micros();
    if(throttle==1000){
      delay(10000);
    }
  }
  gyro_x_cal /= 4000;//se realiza la media de las 4000 muestras para obtener
la calibración correcta.
  gyro_y_cal /= 4000;
  gyro_z_cal /= 4000;
  acc_x_cal /= 4000;
  acc_y_cal /= 4000;
  acc_z_cal /= 4000;
  error_mot=0;
  Serial.println("calibrado perfecto");
  delay(1500);
  loop_timer=micros();
}

void loop() {
  sensors_event_t a, g, temp;
```

```
  mpu.getEvent(&a, &g, &temp);//obtenemos los valores de aceleraccción y
velocidad angular que nos da el sensor MPU6050.
  tiempo_anterior=tiempo;//guardamos las variables temporales para saber el
tiempo que pasa entre cada vez que pasa por este punto.
  tiempo=micros();
  dt=(tiempo-tiempo_anterior);
  gyro_x = (g.gyro.x * 57.29578) - gyro_x_cal;//calibramos la velocidad
angular del giroscopio y lo pasamos a grados/s.
  gyro_y = (g.gyro.y * 57.29578) - gyro_y_cal ;
  gyro_z= (g.gyro.z * 57.29578) - gyro_z_cal;
  gyro_x_filt=0.4*gyro_x_filt+gyro_x*0.6;//filtro para eliminar el ruido de
las señales.
  gyro_y_filt=0.4*gyro_y_filt+gyro_y*0.4;
  gyro_z_filt=0.4*gyro_z_filt+gyro_z*0.6;
  acc_x=(a.acceleration.x - acc_x_cal);//calibramos la aceleraccción del
acelerómetro.
  acc_y=(a.acceleration.y - acc_y_cal);
  acc_z=(a.acceleration.z - acc_z_cal +9.8);
  acc_x_filt=0.4*acc_x_filt+acc_x*0.6;//filtro para eliminar el ruido de las
señales.
  acc_y_filt=0.4*acc_y_filt+acc_y*0.6;
  acc_z_filt=0.4*acc_z_filt+acc_z*0.6;
  //calculamos los ángulos de pitch, roll y yaw con los valores de velocidad
angular que nos da el sensor MPU6050.
  angle_pitch += gyro_x_filt * (float(dt)*0.000001);
  angle_roll += gyro_y_filt * (float(dt)*0.000001);
  angle_yaw += gyro_z_filt * (float(dt)*0.000001);

  //0.01745329 = (3.142(PI) / 180degr) la función sin en arduino está en
radianes.
  angle_pitch += angle_roll * sin(gyro_z_filt * float(dt) * 0.00000001745);
  angle_roll -= angle_pitch * sin(gyro_z_filt * float(dt) * 0.00000001745);
  //Calcular ángulos de inclinación con el acelerómetro.
  acc_total_vector =
sqrt((acc_x_filt*acc_x_filt)+(acc_y_filt*acc_y_filt)+(acc_z_filt*acc_z_filt))
;
  //57.296 = 1 / (3.142 / 180) la función seno está en radianes.
  angle_pitch_acc = asin(acc_y_filt/acc_total_vector)* 57.296;
  angle_roll_acc = asin(acc_x_filt/acc_total_vector)* -57.296;

  if(set_gyro_angles){//corregir drift ángulos giroscopio.
    angle_pitch = angle_pitch * 0.9996 + angle_pitch_acc * 0.0004;
    angle_roll = angle_roll * 0.9996 + angle_roll_acc * 0.0004;
  }
  else{//al principio
    angle_pitch = angle_pitch_acc;
    angle_roll = angle_roll_acc;
    angle_yaw=0;
    set_gyro_angles = true;
  }
  //leer voltaje de la batería
  tension_bateria=(float(analogRead(voltaje))*(3.23/4095)+0.1)*10.95/2.78;
  tension_bateria_filt = 0.95 * tension_bateria_filt + 0.05 *
tension_bateria;
  //Enviar datos del dron al mando.
  MPU6050Readings.angle_pitch = angle_pitch;
  MPU6050Readings.angle_roll = angle_roll;
  MPU6050Readings.angle_yaw = angle_yaw;
```

```
  MPU6050Readings.tension_bateria_filt = tension_bateria_filt;
  //función de enviar datos
  esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *)
&MPU6050Readings, sizeof(MPU6050Readings));

  if (result == ESP_OK) {
   Serial.println("Sent with success");
  }
  else {
   Serial.println("Error sending the data");

  }
  if((incomingthrottle > 1743)){//en función del valor del joystick del
throttle se selecciona un throttle adecuado.
      offset = map( long(incomingthrottle), 1744, 4095, 13000, 15500 );//-5
      throttle=float(offset)/10;


  }
  else {
    offset = map( long(incomingthrottle), 0, 1743, 11500, 13000 );
    throttle=float(offset)/10;


  }
  if(start1==0){//Empezamos con ángulos de 0 grados.
    angle_pitch=0.0;
    angle_roll=0.0;
  }

  if(programable==1){//ruta  de vuelo programable.
    if(counter<100){
      throttle2=1150;//primero se pone un throttle de 1150 us durante un
tiempo de 100*5ms.
    }
    else if((counter>=100)&&(counter<600)){
      throttle2=throttle2+0.7;//durante un tiempo de 500*5ms se sube el
throttle de forma progresiva hasta 1475us.
    }
    else if((counter>=600)&&(counter<1000)){//se pone una referencia de pitch
de 8 grados, el dron se inclina a la derecha.
      refpitch=8.0;
    }
    else if((counter>=1000)&&(counter<1400)){//se pone una referencia de
pitch de -8 grados, el dron se inclina a la izquierda.
      refpitch=-8.0;
    }
    else if((counter>=1400)&&(counter<1500)){//se pone una referencia de
pitch de 0 grados, el dron se vuelve a poner recto.
      refpitch=0.0;
    }
    else if((counter>=1500)&&(counter<1900)){//se pone una referencia de roll
de 8 grados, el dron se inclina hacia delante.
      refroll=8.0;
    }
    else if((counter>=1900)&&(counter<2300)){//se pone una referencia de roll
de -8 grados,el dron se inclina hacia atras.
      refroll=-8.0;
```

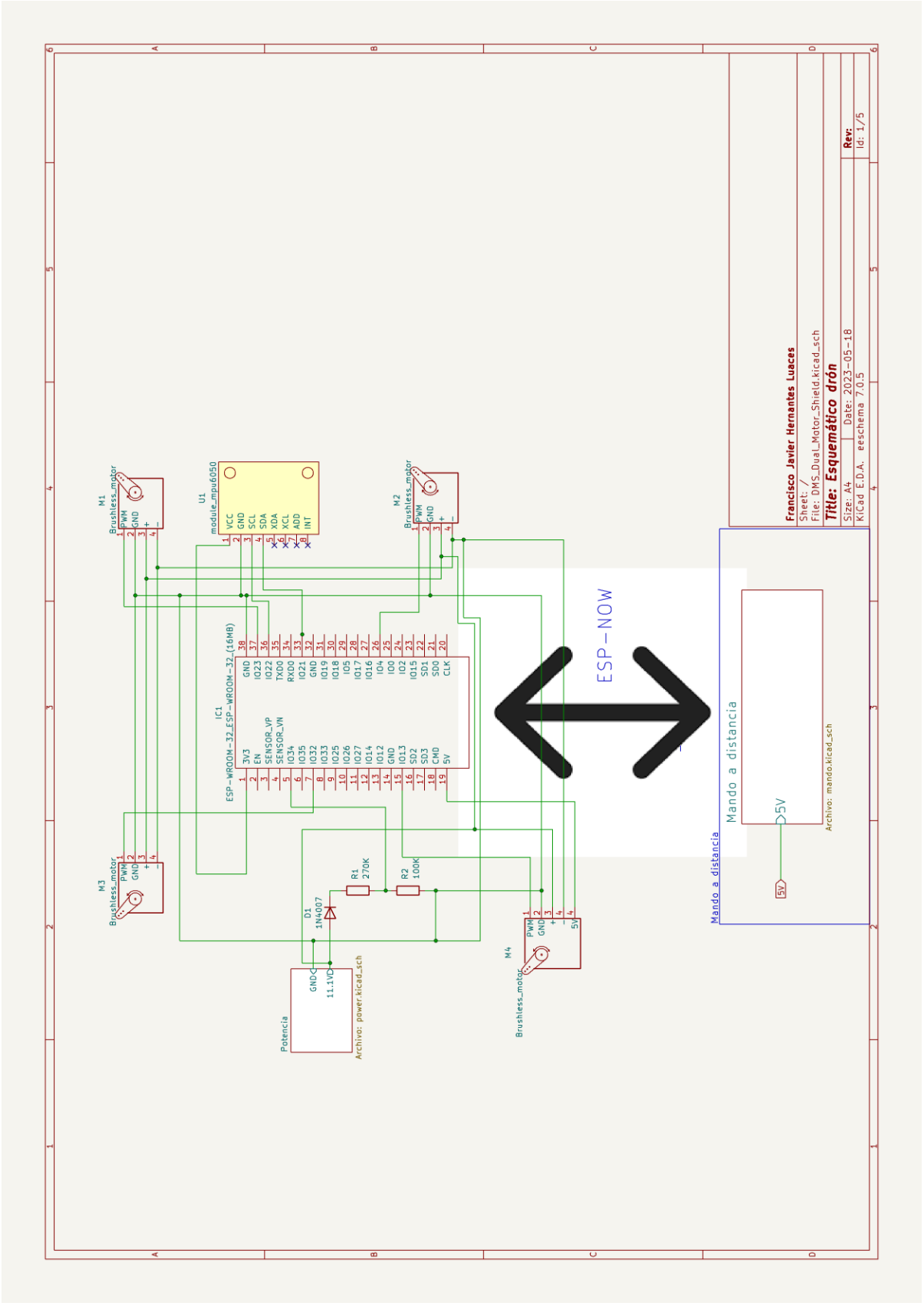
```
    }
    else if((counter>=2300)&&(counter<2400)){//se pone una referencia de roll
de 0 grados, el dron se vuelve a poner recto.
      refroll=0.0;
    }
    else if((counter>=2400)&&(counter<4900)){// durante 2500*5ms se se baja
el throttle de forma progresiva.
      throttle2=throttle2-0.1;
    }
    else if(counter>=5001){
      counter=5005;
    }
    if(incomingthrottle!=0){
      programable=0;
    }
  }
  if(start!=0){//Hasta que el dron se eleva no funcionan los PIDs.
    pidpitch();
    pidroll();
    pidyaw();
  }
  if(programable==0){
  //se calcula el tiempo que las señales tienen que esatr en alta para que se
mantenga estable el dron.
    velocidad =throttle + PIDPITCH + PIDROLL +PIDYAW;
    velocidad1 = throttle + PIDPITCH - PIDROLL - PIDYAW;//+50
    velocidad2 =  throttle - PIDPITCH - PIDROLL + PIDYAW;
    velocidad3 = throttle - PIDPITCH + PIDROLL - PIDYAW;
  }
  else if(programable==1){
    if(incomingthrottle!=0){//si la persona sube el joystick del throttle, el
dron se vuelve a controlar con el mando.
      programable=0;
    }
    //se calcula el tiempo que las señales tienen que esatr en alta para que
se mantenga estable el dron.
    velocidad =throttle2 + PIDPITCH + PIDROLL +PIDYAW;
    velocidad1 = throttle2 + PIDPITCH - PIDROLL - PIDYAW;//+50
    velocidad2 = throttle2 - PIDPITCH - PIDROLL + PIDYAW;
    velocidad3 = throttle2 - PIDPITCH + PIDROLL - PIDYAW;
    counter++;
  }

  if((velocidad>1160.0) && (velocidad1>1160.0) && (velocidad2>1160.0) &&
(velocidad3>1160.0) && (error_mot==0)){
   start=1;
   start1=1;
 }
  if((start==1) && (error_mot==0)){//limitar el tiempo en alta tanto por
arriba como por abajo.
    if(velocidad < 1150.0)
    {
      velocidad= 1150.0;
    }
    if(velocidad > 2000.0)
    {
      velocidad=2000.0;
    }
```

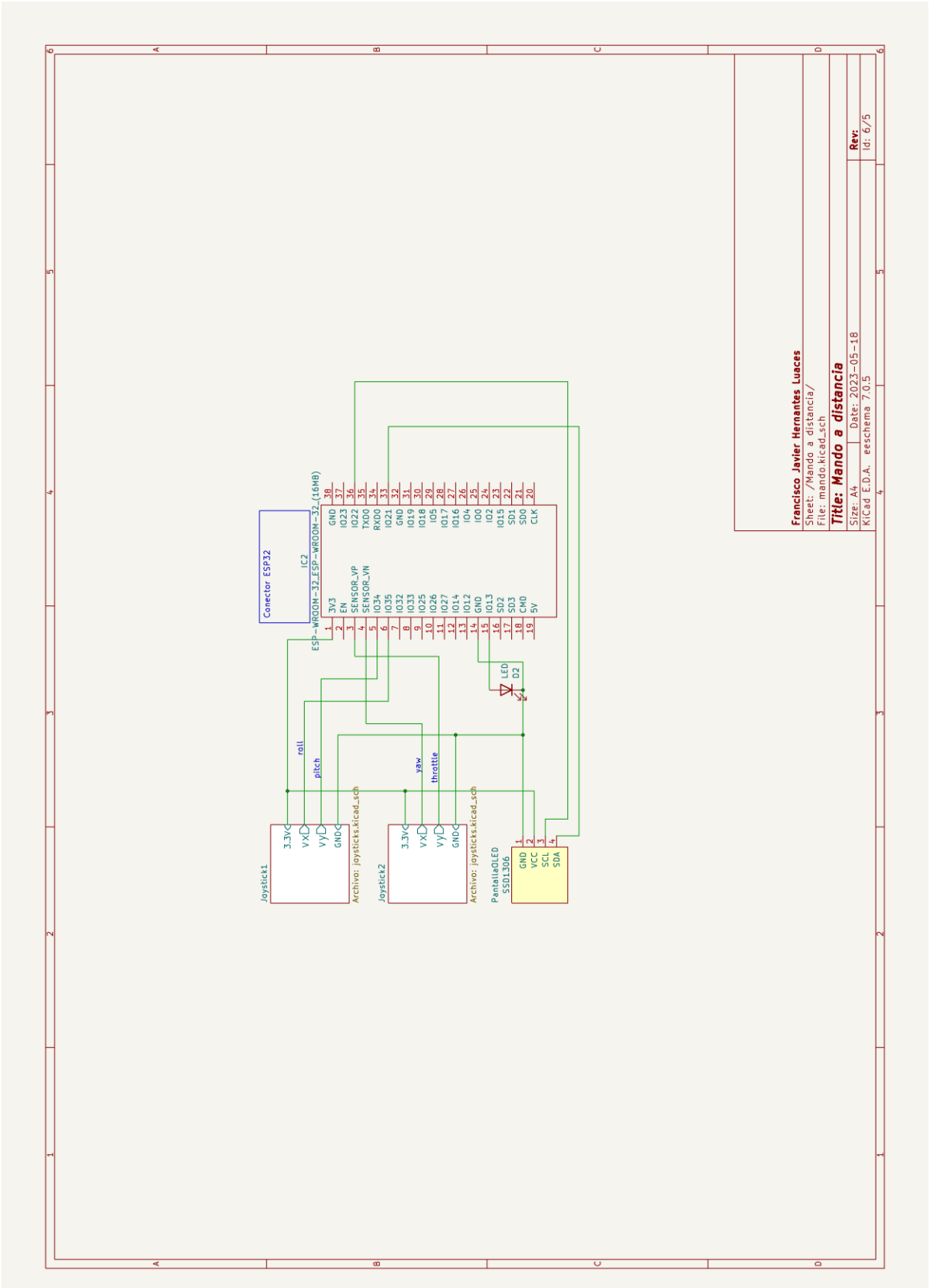
```
    if(velocidad1 < 1150.0)
    {
      velocidad1= 1150.0;
    }
    if(velocidad1 > 2000.0)
    {
      velocidad1=2000.0;
    }
    if(velocidad2 < 1150.0)
    {
      velocidad2= 1150.0;
    }
    if(velocidad2 > 2000.0)
    {
      velocidad2=2000.0;
    }
    if(velocidad3 < 1150.0)
    {
      velocidad3= 1150.0;
    }
    if(velocidad3 > 2000.0)
    {
      velocidad3=2000.0;
    }
  }
  if(error_mot1==1){
    if(incomingthrottle>10){
      error_mot1=0;
    }
  }
  if (((incomingthrottle==0) && (refroll<=-9.1)) ||
(error_mot1==1)){//reseteamos todos los valores y hacemos que los motores no
giren si bajamos el joystick de throttle y roll al mínimo.
    velocidad=1000.0;
    velocidad1=1000.0;
    velocidad2=1000.0;
    velocidad3=1000.0;
    PIDPITCH=0;
    PIDROLL=0;
    PIDYAW=0;
    pid_i_pitch=0;
    pid_i_roll=0;
    pid_i_yaw=0;
    error_roll=0;
    previous_error_roll=0;
    error_pitch=0;
    previous_error_pitch=0;
    error_yaw=0;
    start=0;
    error_mot1=1;
  }
  if((abs(angle_pitch)>=45) || ((abs(angle_roll))>=45)||(error_mot==1)){
    //cuando el ángulo de pitch o de roll sea mayor de 45 grados se paran los
motores.
    velocidad=1000;
    velocidad1=1000;
    velocidad2=1000;
    velocidad3=1000;
```

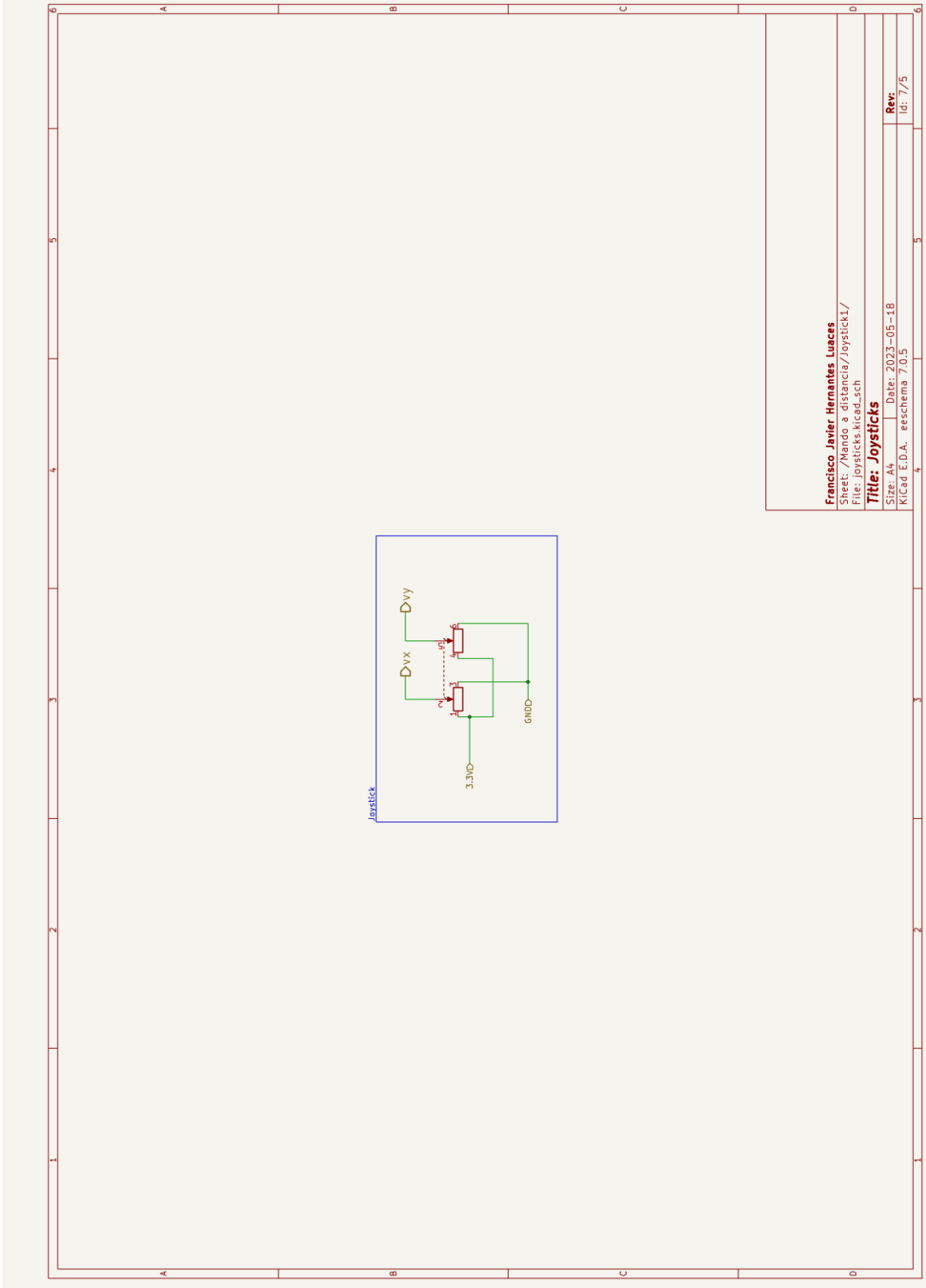
```
    error_mot=1;
    PIDPITCH=0;
    PIDROLL=0;
    PIDYAW=0;
    pid_i_pitch=0;
    pid_i_roll=0;
    pid_i_yaw=0;
    error_roll=0;
    previous_error_roll=0;
    error_pitch=0;
    previous_error_pitch=0;
    error_yaw=0;
    previous_error_yaw=0;
  }
  //se pone la señal PWM en alta
  digitalWrite(motor1, HIGH);  // MOTOR 1
  digitalWrite(motor2, HIGH);  // MOTOR 2
  digitalWrite(motor3, HIGH);  // MOTOR 3
  digitalWrite(motor4, HIGH);  // MOTOR 4
  timer_start = micros();

  // Cuando se cumpa el tiempo de PWM definido en velocidad, se pasa cada
señal a 0 (LOW) para terminar el ciclo PWM.
  while (digitalRead(motor1) == HIGH || digitalRead(motor2) == HIGH ||
digitalRead(motor3) == HIGH || digitalRead(motor4) == HIGH) {
    if (float(timer_start) + velocidad <= float(micros())) digitalWrite(motor1, LOW);
// MOTOR 1
    if (float(timer_start) + velocidad1 <= float(micros())) digitalWrite(motor2, LOW);
// MOTOR 2
    if (float(timer_start) + velocidad2 <= float(micros())) digitalWrite(motor3, LOW);
// MOTOR 3
    if (float(timer_start) + velocidad3 <= float(micros())) digitalWrite(motor4, LOW);
// MOTOR 4*/
  }


  //ejecutamos nuestro bucle cada 5ms(200Hz)
  while(micros() - loop_timer < 5000) ;
  loop_timer = micros();

}
```

Apéndice C

Esquemático


Francisco Javier Hernantes Luaces
Sheet: /
File: DMS_Dual_Motor_Shield.kicad_sch
Title: Esquemático drón
Size: A4
Date: 2023-05-18
Rev:
KiCad E.D.A. eeschema 7.0.5
Id: 1/5
M1
Brushless_motor
PWM
GND
U1
module_mpu6050
VCC
GND
SCL
SDA
XDA
XCL
ADD
INT
M2
Brushless_motor
IC1
ESP-WROOM-32_ESP-WROOM-32_(16MB)
3V3
EN
SENSOR_VP
SENSOR_VN
IO34
IO35
IO32
IO33
IO25
IO26
IO27
IO14
IO12
GND
IO13
SD2
SD3
CMD
5V
GND
IO23
IO22
TXD0
RXD0
IO21
GND
IO19
IO18
IO5
IO17
IO16
IO4
IO0
IO2
IO15
SD1
SD0
CLK
M3
Brushless_motor
R1
270K
R2
100K
D1
1N4007
GNDC
11.1VC
Potencia
Archivo: power.kicad_sch
M4
Brushless_motor
5V
ESP-NOW
Mando a distancia
Mando a distancia
5V
5V
Archivo: mando.kicad_sch


Conector ESP32
IC2
ESP-WROOM-32_ESP-WROOM-32_(16MB)
3V3
EN
SENSOR_VP
SENSOR_VN
IO34
IO35
IO32
IO33
IO25
IO26
IO27
IO14
IO12
GND
IO13
SD2
SD3
CMD
5V
GND
IO23
IO22
TXD0
RXD0
IO21
GND
IO19
IO18
IO5
IO17
IO16
IO4
IO0
IO2
IO15
SD1
SD0
CLK
LED
D2
roll
pitch
yaw
throttle
Joystick1
3.3V
VX
VY
GND
Archivo: joysticks.kicad_sch
Joystick2
3.3V
VX
VY
GND
Archivo: joysticks.kicad_sch
PantallaOLED
SSD1306
GND
VCC
SCL
SDA
Francisco Javier Hernantes Luaces
Sheet: /Mando a distancia/
File: mando.kicad_sch
Title: Mando a distancia
Size: A4
Date: 2023-05-18
Rev:
KiCad E.D.A. eeschema 7.0.5
Id: 6/5


Vy
Vx
GNDD
3.3VD
Joystick
Francisco Javier Hernantes Luaces
Sheet: /Mando a distancia/Joystick1/
File: joysticks.kicad_sch
Title: Joysticks
Size: A4
Date: 2023-05-18
Rev:
KiCad E.D.A. eeschema 7.0.5
Id: 7/5

Bibliografía

1. **Zamora, José Rolando Navia.** *Optimización del proceso de fumigación agrícola mediante la utilización de los drones.* Facultad de Ciencias de la Ingeniería, Universidad Técnica Estatal de Quevedo. 2019. pág. 77, Trabajo Fin de Grado.
2. **Segura, Cristian.** La guerra en Ucrania revoluciona el uso de los drones civiles como arma de matar. *El País.* Febrero de 2023, pág. 2.
3. **Santos, Rui.** Random Nerd Tutorials. [En línea] Septiembre de 2013. https://randomnerdtutorials.com/.
4. **Olivera, Ana Velázquez.** *El Mundo de los Drones.* 2017. pág. 13.
5. **LLamas, Luis.** Luis LLamas Ingeniería,informática y diseño. [En línea] 2 de Enero de 2019. https://www.luisllamas.es/como-conectar-un-esp8266-a-una-red-wifi-modo-sta/.
6. **Lazárraga, Gonzalo Solchaga Pérez de.** *Control motor brushless sensorless.* Ingeniería Eléctrica y Electrónica, Universidad Politécnica de Navarra. 2015. pág. 63, Trabajo Fin de Grado.
7. **Cole, Terry.** *IEEE std 802.11-2007.* Instituto de Ingeniería Eléctrica y Electrónica. Nueva York : s.n., 2007.
8. **Ciampa, Mark.** *Security Guide To Network Security Fundamentals.* Curso de Tecnología. Boston : s.n., 2009.
9. **Calin, Camelia Loredana.** *Aplicación de los drones en los procesos de inspección y gestión de la seguridad en buques.* Departamento de ciencia e ingeniería Náutica, Universidad Politécnica de Catalunya. Barcelona : s.n., 2023. pág. 72, Trabajo Fin de Grado.
10. **Brokking, Joop.** Brooking. [En línea] Marzo de 2018. http://brokking.net/.
11. **aérea, Agencia Estatal de Seguridad.** One Air. [En línea] Octubre de 2022. https://www.oneair.es/normativa-drones-espana-aesa.
12. **Pugliesi, Daniel.** Naylamp mechatronics. [En línea] Abril de 2017. https://naylampmechatronics.com.
13. **Herranz, Francisco.** Industry talks. [En línea] Noviembre de 2022. https://industrytalks.es.
14. **FpvMax.** [En línea] Agosto de 2017. https://www.fpvmax.com.
15. **Colin, Wills.** *Unmanned Combat Air Systems: Technical and Legal Challenges.* Reino Unido : Palgrave Macmillan, 2015. 978-1-349-69822-6.
16. **Blum, Howard.** *the eve of destruction : the untold story of the Yom Kippur War.* s.l. : HarperCollins, 2003. 0-06-001399-0.
17. **Axe, David.** *Strategist: Killer Drones Level Extremists´Advantage.* 2009.
18. **Espressif systems.** [En línea] https://www.espressif.com.
19. **aerocamaras.** [En línea] Febrero de 2021. https://aerocamaras.es.
20. **alldatasheet.** [En línea] Mayo de 2023. https://www.alldatasheet.com.