Universidad Michoacana de San Nicolás de Hidalgo

Facultad de Ciencias Físico Matemáticas
Mat. Luis Manuel Rivera Gutiérrez

Diseño, ensamblaje y programación de un dron autónomo

TESIS DE MAESTRÍA

PARA OBTENER EL TÍTULO DE:

Maestro en Ciencias en Ingeniería Física

PRESENTA:

Ricardo Tovar Duarte

DIRECTOR DEL TRABAJO:

Dr. Francisco Shidartha Guzmán Murillo

Morelia Michoacán, México

Febrero 2022

Resumen

En los últimos años el área de la automatización de vehículos ha despegado gracias al desarrollo en la electrónica, con la miniaturización de los componentes electrónicos y aumento de la eficiencia de baterías, la computación en el incremento de la velocidad de procesamiento de información y algoritmos, la ingeniería en el aumento de la eficiencia de los componentes mecánicos, así como el empleo de la física y la matemática en modelación y teoría de control.

Este proyecto consiste en el diseño, ensamblaje y programación de un dron desde los primeros pasos, es decir, se eligen los componentes mecánicos y electrónicos necesarios y su funcionamiento. Para cada sensor se desarrollan códigos basados en el lenguaje de programación Arduino; un lenguaje enfocado a proyectos educativos de robótica y de uso libre y gratuito. Se realizan pruebas de funcionamiento y se verifican estos datos. Una vez pasadas las pruebas se desarrolla el programa principal de la controladora de vuelo que será la encargada de asegurar un vuelo seguro del dron, esta contará con todos los datos obtenidos por los sensores: giroscópio, acelerómetro, brújula digital, barómetro y GPS, después de procesar los datos calculará parámetros de vuelo como ángulos de inclinación y orientación, altitud y posiciónamiento por GPS. Una vez conocidos los parámetros la controladora será capaz de cumplir órdenes específicas mediante controladores PID para cada parámetro. Se realizan modificaciones al controlador PID como aproximación del error mediante polinomios de interpolación de Lagrange de primer y segundo orden.

En la tesis se presentan los componentes mecánicos y electrónicos, sus características y especificaciones, los códigos de prueba usados, los datos obtenidos de las pruebas, los problemas, inconvenientes, pruebas fallidas y finalmente una estimación de los costos del proyecto y mejoras propuestas.

Palabras clave: Control de vuelo, control PID, Arduino, MPU6050, MS5611, HMC5883L, Motores brushless.

Abstract

In the last few years vehicle automation has advanced thanks to the development in electronics due to the miniaturization of electronic components and the increase battery efficiency, the increasing processing speed and algorithms, the increase in the efficiency of mechanical components in engineering as well as the application of physics and mathematics in control theory and modeling.

This project consists in the design, assembly and programming of a UAV step by step choosing the electronic and mechanical components and how they work. For each sensor we develop codes based on Arduino which is a programming language focused on educational robotics projects and free use.

We test about sensors and verify data. Once the tests are passed we develop the main code of the flying controller, it will be responsible for ensuring a safe flight of the UAV, the flying controller will receive the data measured by the sensors including gyroscope, accelerometer, digital compass, barometer and GPS, after data processing it will calculate flight parameters such as pitch, roll and yaw angles, altitude, and GPS positioning. Once the parameters are known, the flight controller will be able to carry out specific orders through PID controllers for each parameter. We made some modifications to PID controller as an approximation of the error using first and second order Lagrange interpolation polynomials.

In this thesis we present the characteristics and specifications of the electronic and mechanical components, the codes used for tests, the results of the tests, the problems, inconveniences, failed tests and finally an estimate of project costs and proposed improvements.

Keywords: flying controller, PID controller, Arduino, MPU6050, MS5611, HMC5883L, Brushless motors.

Índice

1. Introducción

En la actualidad se tiene una idea de que los drones, UAV o VANT son un invento nuevo, sim embargo esta percepción es errónea, su uso y desarrollo data del siglo XIX, principalmente con intenciones militares.

Actualmente gracias al desarrollo exponencial en las áreas de computación, instrumentación, electrónica e ingeniería el desarrollo de drones es un campo de moda que se ha dotado con tecnología apuntando a lo que en la actualidad resulta más eficiente y productivo: los vehículos autónomos.

Los vehículos autónomos conocidos actualmente se desarrollan en el sector automotriz, actualmente la principal empresa encargada de ellos es Tesla, no hay mejor ejemplo del trabajo y desarrollo que hay detrás de esta tecnología, así como los peligros del fallo de estas tecnologías. En comparación con los autos los drones autónomos no son un campo de desarrollo púbico tan notorio aún.

Hace algunos años los drones eran principalmente un entretenimiento o pasatiempo para las personas, usados en su mayoría para tomar fotos, videos o hacer carreras. Estas actividades han aumentado la exigencia de los drones, exigidos a contar con funciones nuevas como seguimiento de objetos, seguimiento de trayectorias, etc. Es por estas razones que se ha impulsado el alcance de los drones, actualmente son usados para tareas más específicas, de impacto social y sin duda impacto visual, por ejemplo, drones para irrigar cultivo, atrapar objetos en caída libre, mover y equilibrar cargas, vigilar zonas de interés, obtener datos climáticos y actividades militares. Todas estas tareas mencionadas anteriormente son el resultado de la combinación de ingeniería, electrónica, computación, física y matemáticas empleadas en la teoría de los controladores inmersos en el dron.

El funcionamiento de un cuadricóptero es en realidad complejo; debido a la composición de sus elementos mecánicos, eléctricos y algorítmos computacionales, todos estos dependientes de mútiples sensores a bordo del dron que lo dotan de información acerca de su estado físico. Cada uno de los sensores se debe someter a pruebas de funcionamiento y veracidad de los datos proporcionados.

El área de los drones tiene un gran futuro cada vez con mejor proyeccción por las nuevas tecnologías, actualmente el mundo apuesta por la electrificación del sector industrial, esto por si solo ha traido una evolución apresurada de las baterías.

El futuro en los drones apunta a ser impresionantemente eficaz, preciso y complejo. Es esta la razón por la que se realiza el presente proyecto, personalmente motivado por un video producido por las conferencias TED, titulado Raffaello D'Andrea: La asombrosa potencia atlética de los cuadricópteros [35]. Siendo este video un ejemplo de lo que se busca conseguir de este proyecto a largo plazo.

El proyecto está enfocado al diseño, ensamblaje y principalmente a la programación del control de vuelo de un dron para llevar sus capacidades lo más cercano a la autonomía, para garantizar esto se deben desarrollar

códigos para medir datos y procesar información de sensores para que después actúen los sistemas de control.

Hacerlo funcionar correctamente está lleno de problemas e inconvenientes característicos de un proyecto de ingeniería: fallas de funcionamiento, pruebas fallidas y accidentes.

“Una persona que nunca cometió un error, nunca intentó nada nuevo”.

— **Albert Einstein.**

Objetivos

Los objetivos generales son:

1. Construir un dron con navegación autónoma.
2. Diseñar los programas y funciones que permitan la navegación autónoma del dron.

2. Drones

Se define como pequeño vehículo aéreo no tripulado, utilizado en el ámbito militar (para reconocimiento táctico desde gran altura, vigilancia del campo de batalla o guerra electrónica) y civil (vigilancia de manifestaciones, control de la contaminación y de incendios forestales, etc.).

El término dron se usa para reemplazar las iniciales UAV (Unmaned Aerial Vehicle), se definen como un vehículo reutilizable no tripulado capaz de volar de forma controlada y propulsado por motores. Los UAV son conocidos en idioma español como VANT debido a la traducción [32].

2.1. Historia

La aviación tripulada y no tripulada fueron desarrolladas a la par desde que los pioneros como Cayley, Ninomiya, Du Temple, Langley o Cody construyeron las primeras naves no tripuladas en Europa. Fueron los primeros en desarrollar los principios físicos de la aeronáutica y los aplicaron para diseñar modelos sin piloto a bordo, estos pueden considerarse los primeros UAVs de la historia. Los pioneros de los demás países progresaron de la misma manera, evolucionando de los planeadores hasta los vuelos tripulados. Sin embargo, se toparon con un obstáculo, carecían de la tecnología necesaria, pues no disponían de un motor lo suficientemente potente para sus aplicaciones en sus diseños [11].

Nikola Tesla quien se considera creador del misil crucero y la aviación no tripulada, inventó en 1898 el ´Teleautomaton´: un vehículo naval capaz de moverse, detenerse, girar a los lados y enviar diferentes señales de radio.

Posteriormente, en 1908, el oficial de la artillería francesa René Lorin propuso una bomba voladora propulsada a reacción, que podía controlarse de forma remota mediante señales de radio.

Como menciona Cuerno-Rejado [11], durante la Primera Guerra Mundial la aviación tripulada evolucionó, al contrario de la no tripulada por falta de desarrollo tecnológico. Los principales problemas eran de estabilización automática, control remoto y navegación autónoma. La primera persona en resolver estos problemas fue Elmer Ambrose quien realizó experimentos con giroscopios para aplicaciones marítimas, esto lo llevó a desarrollar en 1909 un giroestabilizador para un avión, sin embargo, no obtuvo buen comportamiento. En 1911 su invento fue mejorado y probado con el apoyo del pionero de la aviación Glenn Hammond Curtiss, este sistema permitía controlar el avión respecto a los tres ejes, este sistema le valió ganar un premio en una exposición en Francia.

En 1915 el inventor Peter Cooper Hewitt contactó a Sperry para retomar las ideas de Tesla utilizando de base el dispositivo inventado por Sperry. En 1916 se llevó a cabo la primera demostración del dispositivo para guiar un avión convencional, era el avión automático Hewitt-Sperry, cuyo piloto debía despegar antes de conectar el piloto automático. El avión volaba una ruta programada y luego caía en picada. El piloto

recuperaba la aeronave en ese momento mientras regresaba a la pista de aterrizaje. En 1917 la Armada de EE.UU. financió la idea y entregó cinco hidroaviones "Curtis N-9" para desarrollar el experimento.

Paralelamente la Curtiss Aeroplane & Motor Company se embarcaron en la fabricación de fuselajes para torpedos aéreos no tripulados, entregando los primeros seis llamados Speed Scout a finales de 1917. El primer vuelo controlado con éxito de un avión no tripulado tuvo lugar finalmente el 6 de marzo de 1918, 14 años después del de los hermanos Wright. En octubre de 1918 fue equipado con catapultas. El posteriormente llamado CurtissSperry Aerial Torpedo era un biplano de madera no tripulado, con un peso de sólo 270 kg, incluyendo una carga útil de 136 kg y era impulsado por un motor Ford de 40 CV. El método de guiado hacia su objetivo era primitivo pero ingenioso. Una vez conocidos el viento y la distancia al objetivo, se calculaba la velocidad del motor requerida para alcanzarlo. El avión se controlaba con un simple giróscopo teniendo, además, un barómetro aneroide disponible a bordo. Una vez alcanzada la velocidad calculada, las alas se separaban del fuselaje, dejándolo caer sobre el objetivo. Los primeros sistemas desarrollados como armas de largo alcance (precursores de los actuales misiles de crucero) fueron dispositivos como el anteriormente citado torpedo aéreo americano de 1917, el torpedo aéreo Liberty Eagle, más conocido como Kettering "Bug" de 1918 y el blanco aéreo británico "AT", iniciado en 1914. "Kettering bug" era un biplano más ligero diseñado para transportar una carga útil de 82 kg, y tuvo un comportamiento similar al del torpedo Sperry. Por otro lado, el A.T. británico era un avión monoplano no tripulado radio-controlado y propulsado por un motor de 35 caballos de potencia. El concepto de la serie A.T. sirvió para demostrar la viabilidad del uso de señales de radio como sistema de guiado para volar el avión a su destino [11].

En los años 1970-1980 se desarrollaron UAS (Unmanned Aircraft System) principalmente utilizados para misiones de reconocimiento y vigilancia. el "Boeing YQM-94 Gull", o "Cope-B" fue el ganador en 1971 de la competición Compass Cope USAF para el desarrollo de un sistema de reconocimiento HALE (High Altitude Long Endurance). El objetivo del programa era llegar a los 16.770 metros de altitud, y 20 horas de autonomía de vuelo llevando una carga útil de 680 kg. Esta carga útil incluía equipos para reconocimiento visual, relé de comunicaciones e inteligencia de señales (SIGINT) en un rango de 300 km, que funcionase de día y de noche y bajo cualquier condición climática [11].

Los registros del uso de UAV datan de 1849, el ejército Austriaco lanzó a la ciudad de Venecia globos cargados con explosivos, estos volaban hacia Venecia y una vez sobre la ciudad explotaban debido a un sistema electrónico dejando caer los explosivos. No fueron exitosos debido a que dependían totalmente de la dirección del viento [32].

2.2. Clasificación

Según menciona Ruipérez en su trabajo [32], hay varias formas en que se clasifican los drones, las cuales se mencionan a continuación.

2.2.1. Uso

Consecuente a su fin o misión se clasifican en:

- **De blanco:** simulación de aviones
- **Reconocimiento:** Recaudación de información militar, control de áreas.
- **Combate:** Dedicados a misiones ofensivas o ataques bélicos.
- **Logística:** Transporte de mercancía y recursos.
- **Investigación o desarrollo:** Recaudación de información de interés científico como variables climatológicas.
- **Civil:** Son los drones comerciales, donde su uso es la fotografía, transmisión de imágenes, prevención y control de incendios, carreras de drones y ocio en general.

2.2.2. Método de control

- **Autónomo:** El dron se rige, decide y realiza tareas controlado por algoritmos, mediante el uso de datos obtenidos por sus propios sensores.
- **Monitorizado:** El piloto proporciona información al dron para que este realice la tarea.
- **Supervisado:** Un piloto controla el dron, sin embargo, el dron puede realizar algunas tareas automáticamente.
- **Preprogramado:** El dron sigue una instrucción o plan de vuelo, no puede cambiar su misión ni adaptarse a cambios.
- **Controlado remotamente:** Son aquellos que se pilotan mediante un radio control.

2.2.3. Forma de Sustentación

Drones de ala fija: La estructura consta de fuselaje y alas fijas, son propulsados por uno o dos motores de combustión o turbinas, son parecidos a un avión a menor escala.

Drones multirrotor: Son aquellos drones que son propulsados por más de dos motores. Normalmente están formados por brazos donde se anclan los motores. Para lograr el vuelo las hélices tienen un sentido de giro especial dependiendo el número de motores. Estos pueden mantener una posición de manera estable debido a los componentes electrónicos con los que se controla [32].

2.3. Cuadrucópteros

Los cuadrucópteros o Cuads por abreviación constan de 4 motores, existen dos configuraciones dependiendo de la ubicación de sus motores respecto al frente del dron, estas son la configuración X y la configuración + [32].

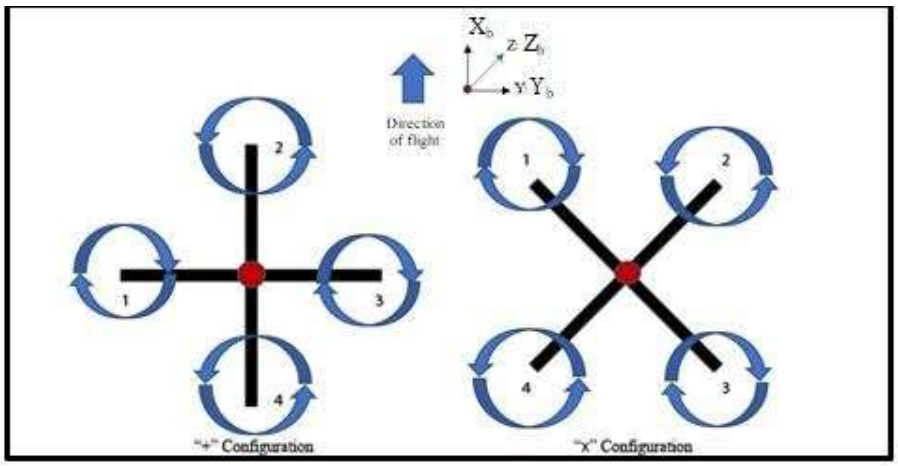


Figura 2.1: Configuraciones Quad+ y QuadX [27].

La mayoría de drones de uso civil son multirrotor, esto se debe a que cumplen bien con el propósito para el cual son fabricados: la toma de imágenes y videos.

Las principales ventajas de los cuads son las siguientes

- Su despegue y aterrizaje es vertical, permite prescindir de espacio como pistas de aterrizaje.
- Es posible mantenerse en vuelo en un punto fijo o a baja velocidad, esto es deseable en aplicaciones como vigilancia o reconocimiento y toma de fotos y videos.
- Tienen mayor maniobrabilidad y precisión en el vuelo, debido a los cuatro motores el dron puede moverse libremente en un espacio tridimensional y no necesita grandes radios de giro como los sistemas de ala fija.

Algunas de sus desventajas son

- Al usar más motores su autonomía se reduce significativamente.
- Tienden a ser muy ruidosos.
- Son más sensibles a ráfagas de viento.

El campo de desarollo de los cuads es de reciente y constante crecimiento, a medida que la tecnología avance los cuads serán capaces de realizar tareas más complejas, de mayor precisión y de manera más eficiente [30].

Figura 2.2: Drones comerciales [5].

3. Componentes Teóricos

3.1. Motores Brushless de Corriente Continua

El uso de los motores de corriente continua sin escobillas (BLDC) o motor brushless, ha ido incrementando en los últimos años en aplicaciones de consumo y sectores insdustriales como el automovilístico, aeroespacial, automatización, instrumentación, etc. Debido a su eficiencia y tamaño compacto.

Los motores BLDC tienen la característica de no emplear escobillas en la conmutación para la transferencia de energía, en este caso, la conmutación se realiza electrónicamente. Esta propiedad elimina uno de los grandes problemas que poseen los motores eléctricos convencionales con escobillas: la disminución del rendimiento debido al rozamiento, con su consecuente producción de calor, que deriva en motores más ruidosos y una necesidad más alta de mantenimiento [26].

Los motores brushless se componen de diferentes piezas:

- **Estator.**

  El estator del motor BLDC consiste en un conjunto de láminas de acero apiladas con bobinados colocados en las ranuras de forma axial a lo largo de la periferia interna. El estator se asemeja a un motor de inducción, sin embargo, las bobinas se distribuyen de una manera diferente. La mayoría de los motores BLDC tienen tres fases en el estator conectado en estrella. Cada una de estas fases está construida por numerosas espiras interconectadas, una o más bobinas colocadas en las ranuras se interconectan para que formar una fase, cada una de estas bobinas se distribuye en la periferia del estator para formar un número par de polos[26].

- **Rotor.**

  El rotor es de imán permanente y puede variar desde dos hasta ocho pares de polos alternativos de Norte (N) y Sur (S), el material magnético adecuado para hacerlo es elegido en función de la densidad del campo magnético requerido en el rotor. Los imanes de ferrita se utilizan tradicionalmente para hacer imanes permanentes.

  A diferencia de un motor de escobillas de corriente continua, la conmutación de un motor BLDC se controla electrónicamente. Para provocar el giro en el motor BLDC, los bobinados del estator deberán estar activados secuencialmente. Es fundamental conocer la posición del rotor para poder entender cómo deben ser activadas las bobinas según la secuencia de activación [26].

Figura 3.1: Partes de un motor BLDC [1].

PRINCIPIO DE FUNCIONAMIENTO

El funcionamiento se basa en aplicar la corriente eléctrica directamente por los bobinados del estator. La corriente eléctrica genera un campo electromagnético que interacciona con el campo magnético creado por los imanes permanentes del rotor, generando así una fuerza que hace girar al rotor junto con su eje. Al no tener escobillas el control del motor dependerá totalmente de un controlador electrónico de velocidad (ESC) que controla el voltaje suministrado al motor [2].

3.2. PWM

Las salidas PWM (Pulse Width Modulation) con Arduino son muy útiles, sirven para controlar la velocidad de un motor eléctrico, el brillo de un LED o un servomotor con Arduino.

3.2.1. Señales Eléctricas

Primeramente definamos una señal con simples ejemplos, un movimiento con tu mano es una señal, el movimiento de la cola de tu perro también lo es, así pues, una señal sirve para avisar algo, es decir, su objetivo es transmitir información. Una señal eléctrica transmite información mediante la electricidad, por ello recibe el nombre de señal eléctrica. Las señales eléctricas varían con el tiempo en voltaje o corriente. Existen dos tipos de señales:

- Las **señales analógicas** tienen valores infinitos y puede ser representada mediante una función continua. Un ejemplo se puede encontrar en el contacto de una casa, en el cual se encuentra un voltaje de 220 0 120 volts, dependiendo del lugar, esto significa que el voltaje varía entre valores de $\pm$220V.

- Las **señales digitales** son aquellas que tienen un conjunto finito de valores, por ejemplo un apagador solo puede tomar 2 valores o estados: encendido y apagado. No existen estados intermedios entre estos dos. Otro ejemplo claro son las posiciones de la palanca de cambios de un auto [18].

Como menciona Guerra [18], los pines digitales de la placa arduino generan señales digitales y binarias, es decir, solo pueden tener dos estados:

- **Alto o High:** 5V o 3.3V dependiendo la placa.
- **Bajo o low:** 0V.

Sin embargo, las señales digitales pueden tener más de dos estados.

Una **señal periódica** es aquella que se repite con una frecuencia y a intervalos determinados. El tiempo que transcurre entre una repetición y la siguiente se denomina periodo de la señal y se expresa en segundos. Normalmente se expresa con la letra **T**.

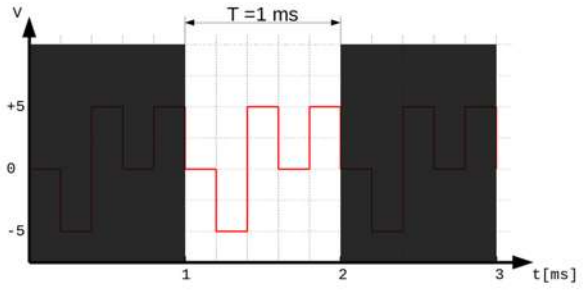


Figura 3.2: Señal con periodo de 1 milisegundo [18].

La repetitividad de una señal también puede expresarse usando la **frecuencia** que se define como la cantidad de veces por segundo que se repite la señal, la unidad usada son hertz (Hz) y se calcula como el inverso del periodo:

$$f = \frac{1}{T}.$$

Una señal con un periodo de 1 ms tiene una frecuencia de 1 kHz o 1,000 Hz lo que significa que la señal se repite 1,000 veces por segundo [18].

La siguiente imagen muestra una señal digital conocida como tren de pulsos cuadrados. Se denomina así porque el tiempo de la señal que se encuentra en estado alto (HIGH) es igual al que está en estado bajo (LOW) y se asemeja a una serie de cuadrados o rectángulos.

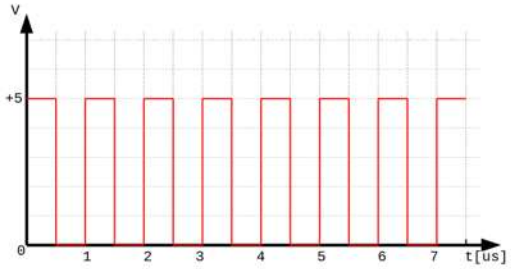


Figura 3.3: Tren de pulsos cuadrados [18].

El periodo de la señal es de $1\mu s$, lo que equivale a una frecuencia de $1MHz$, o lo que significa que los pulsos se repiten 1 millón (1,000,000 Hz) veces por segundo [18].

3.2.2. PWM

Como menciona Guerra en su trabajo [18], el término **PWM** proviene de **Pulse Width Modulation** que significa modulación por ancho de pulsos. Una **señal PWM** es una señal similar al tren de pulsos cuadrados. La principal diferencia con el tren de pulsos es que, en la señal PWM, es posible variar el tiempo que la señal se mantiene en estado alto, pero siempre manteniendo el periodo constante:

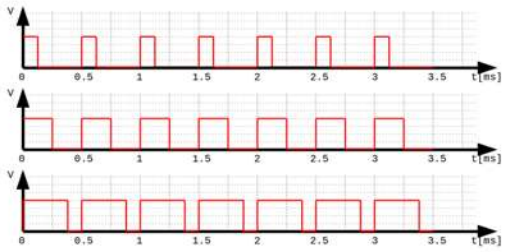


Figura 3.4: Señal PWM [18].

Esta capacidad de variar el tiempo en estado alto, es lo que realmente hace que la señal PWM sea útil y práctica:

- Pueden utilizarse para controlar un servomotor mediante señales pulsantes con diferentes tiempos en alto.
- Sirven para emular una salida analógica.
- Es empleada, con frecuencia, en algunas fuentes de alimentación en la etapa de regulación.

Si tomamos un periodo de la señal tenemos lo siguiente:

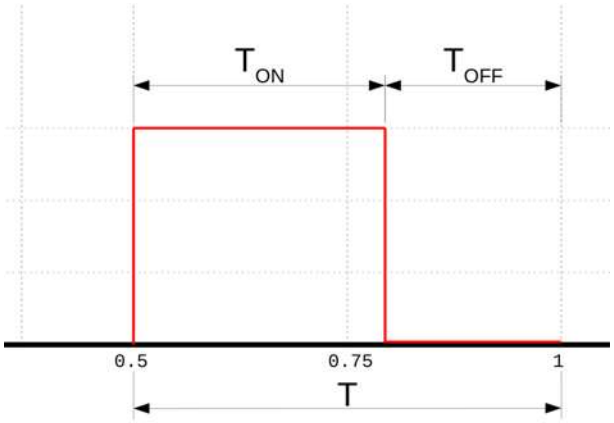


Figura 3.5: Periodo de señal PWM [18].

En la imagen anterior observamos tres valores de tiempo definidos como sigue:

T_{ON} : tiempo que la señal se mantiene en estado alto.
T_{OFF} : tiempo que la señal se mantiene en estado bajo.
T : periodo de la señal.

El **ciclo de trabajo** o **duty cicle** se representa mediante la letra D y se define como la razón entre el tiempo en estado alto y el periodo de la señal:

$$D = \frac{T_{ON}}{T} * 100\,\%.$$

Se multiplica por 100 porque se suele expresar en tanto por ciento y representa el tiempo en el que la señal está en estado alto dentro de un periodo.

Dicho ciclo de trabajo permite controlar el voltaje promedio de la señal, es decir, variar el voltaje de salida. Es posible cambiar el voltaje de salida de 5V a 3.75V, por ejemplo. Este parámetro es fundamental cuando la señal PWM se emplea como conversor digital-analógico [18].

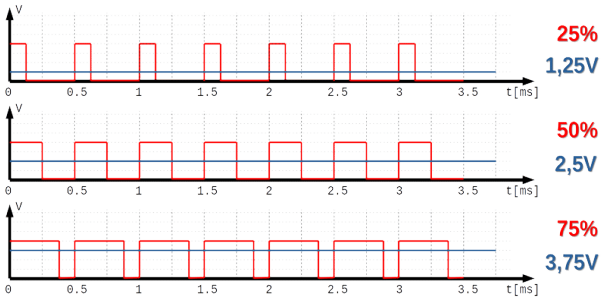


Figura 3.6: Ciclo de trabajo de señal PWM [18].

El voltaje promedio se calcula mediante la siguiente expresión:

$$V_{prom} = V_H - V_L * \frac{D}{100}$$

donde:

V_H es el voltaje en estado alto.

V_L es el voltaje en estado bajo.

3.3. Giroscopio y Acelerómetro

3.3.1. Giroscópio

El giroscópio es un aparato que mide la orientación de un objeto en el espacio. Los giroscopios con tecnología MEMS (Microelectromechanical Systems) se fundamentan en el efecto coriolis para medir, el cual consiste en la existencia de una aceleración relativa del cuerpo en movimiento en un sistema de rotación, esta aceleración es siempre perpendicular al eje de rotación del sistema y a las componentes tangencial y radical del mismo. Todo cuerpo en rotación sufre este efecto, incluso la Tierra [4].

Los giroscópios clásicos consisten de un disco con gran inercia que rota respecto a un eje. Debido a la ley de conservación de momento angular, este móvil rotativo mantiene su orientación respecto del sistema de referencia en el que fue impulsado aunque el móvil dentro del cual se aloja cambie de orientación [16].

Figura 3.7: Giroscópio [21].

3.3.2. Acelerómetro

Como se menciona en [16], los acelerómetros miden la fuerza de gravedad sobre ellos mismos. La tecnología MEMS permite combinar elementos mecánicos y electrónicos en piezas de tamaño diminuto.

Utiliza un material piezoeléctrico casi en estado sólido, cuarzo por ejemplo, el cual está cargado eléctricamente, sobre él hay una masa conocida que lo hace comprimirse dependiendo de la gravedad a la que se someta. La presión produce un cambio en la alineación de los electrones y protones del material piezoeléctrico que provoca una acumulación de cargas opuestas en ambas superficies del material, así se mide una variación en la carga eléctrica que es proporcional al esfuerzo experimentado por el material, elcual es proporcional a la aceleración de la gravedad [16].

3.4. Ángulos de Inclinación

Giroscopio: Conociendo la velocidad angular instantánea $\dot{\theta}$ medida por el giroscopio e integrándola en un diferencial del tiempo obtenemos el ángulo:

$$\theta_{giroscopio} = \theta_{previo} + \dot{\theta} \cdot \Delta t.$$

Acelerómetro: Imaginemos que el acelerómetro cuenta con una inclinación θ respecto a un solo eje, g_x, g_z los valores de la aceleración en dirección a los ejes x y z respectivamente y g la fuerza de gravedad en dirección al centro de la tierra, todo lo anterior representado en la siguiente imagen.

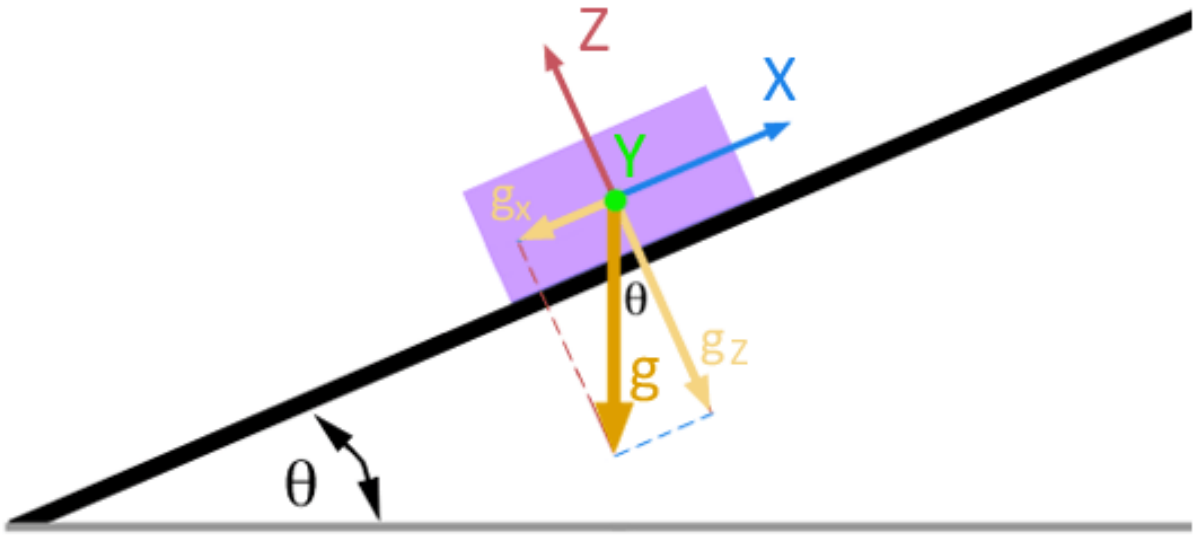


Figura 3.8: Cálculo de ángulo de inclinación mediante acelerómetro [23].

Aplicando trigonometría obtenemos:

$$\theta = \arcsin\left(\frac{g_x}{g}\right) = \arcsin\left(\frac{g_x}{\sqrt{g_x^2 + g_z^2}}\right). \tag{3.1}$$

Haciendo un análisis análogo al modelo 2D, para un modelo 3D tenemos las ecuaciones:

$$\theta_x = \arcsin\left(\frac{g_x}{\sqrt{g_x^2 + g_y^2 + g_z^2}}\right), \tag{3.2}$$

$$\theta_y = \arcsin\left(\frac{g_y}{\sqrt{g_x^2 + g_y^2 + g_z^2}}\right). \tag{3.3}$$

Para transferir el efecto de el giro respecto a yaw del dron a los ángulos respecto a pitch y roll, podemos usar la función *Seno* y realizar la actualización del ángulo mediante las siguientes expresiones:

$$\theta_x = \theta_x \pm \theta_y \sin(\Delta\theta_z), \tag{3.4}$$

$$\theta_y = \theta_y \pm \theta_x \sin(\Delta\theta_z), \tag{3.5}$$

donde $\Delta\theta_z$ es el cambio instantáneo en el ángulo yaw. El signo $+$ ó $-$ se define dependiendo el sistema de referencia [6].

Esto último es necesario por el siguiente ejemplo, supongamos que colocamos el MPU6050 sobre una superficie con una inclinación de $+45°$ en pitch, giramos $90°$ en dirección yaw sin depegar el sensor de la superficie, ahora los ángulos respecto a pitch y yaw han cambiado, si no realizamos esta última actualización, este cambio no tendrá efecto en la medición de los ángulos y nos dará datos erróneos [6].

3.5. Filtro complementario

Como menciona Basarte en [4], la medición del ángulo de inclinación puede verse afectada por ruidos del acelerómetro causados principalmente por vibraciones y derivas que se producen al integrar los valores del giroscopio. Las medidas de los sensores deben ser lo más parecidas a las reales, para ello necesitaremos el filtro complementario.

El filtro complementario aplicado a la medición de un ángulo de inclinación θ tiene la forma

$$\theta = \alpha \cdot \theta_{giroscopio} + (1 - \alpha) \cdot \theta_{acelerometro}, \tag{3.6}$$

donde α es llamado parámetro del filtro complementario y debe cumplir la condición $0 \leq \alpha \leq 1$. Es inmediato observar que si $\alpha = 0$ no se involucra la medición del giroscopio, si $\alpha = 1$ es el acelerómetro el que no se considera.

$\theta_{giroscopio}$ es el ángulo calculado con los datos del giroscópio y $\theta_{acelerometro}$ el ángulo calculado con los datos del acelerómetro.

El filtro complementario combina dos formas diferentes de medir el ángulo de inclinación usando dos sensores diferentes, el filtro de la señal resultante funciona de la siguiente manera:

Filtro paso bajo: Con este filtro nos encargamos de los cambios rápidos característicos de las mediciones con el acelerómetro, este filtro elimina el ruido del acelerómetro.

Filtro paso alto: Con él se elimina la deriva en los giroscópios, es contrario al filtro de paso bajo pues permite el paso de cambios rápidos y no permite los lentos.

Complementario: Se refiere a que las dos constantes α y $(1 - \alpha)$ que componen el filtro complementario suman 1.

El filtro complementario se considera una simplificación del filtro de Kalman, se prefiere el filtro complementario por la simpleza de los cálculos ahorrando tiempo de cómputo al chip, sin embargo, para chips más potentes el filtro de Kalman es buena opción [4].

A continuación se presenta una comparación con gráficas sobre el ángulo de inclinación respecto a un eje medido con el filtro complementario, con el giroscópio y con el acelerómetro, además una comparación del filtro complementario para diferentes valores del parámetro α.

En las imágenes siguientes, la imagen superior presenta la medición del ángulo con el filtro complementario usando $\alpha = 0.9$ en color azul, medido con el giroscópio en color rojo y medido con el acelerómetro en color verde.

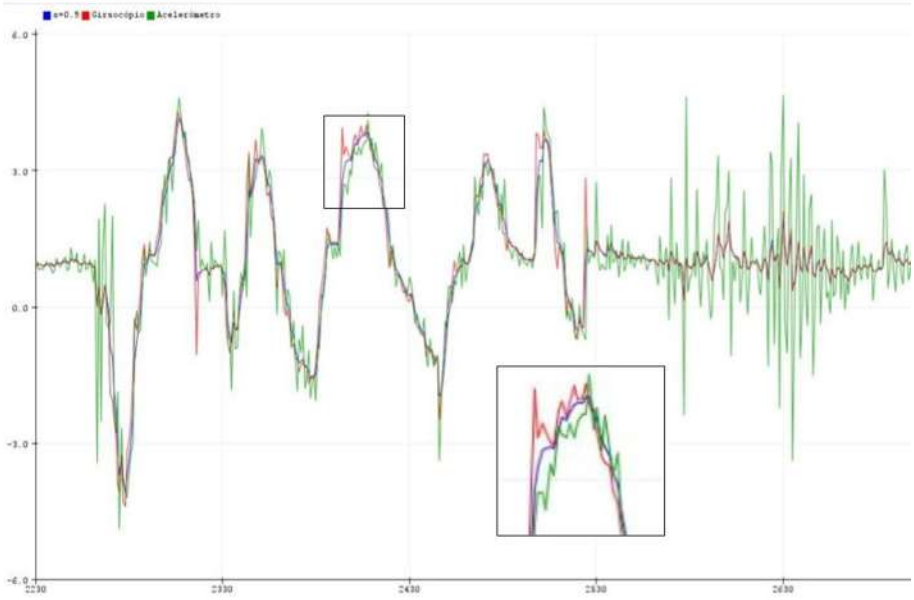


Figura 3.9: Mediciones experimentales del ángulo de inclinación usando filtro complementario $\alpha = 0.9$, giroscópio y acelerómetro.

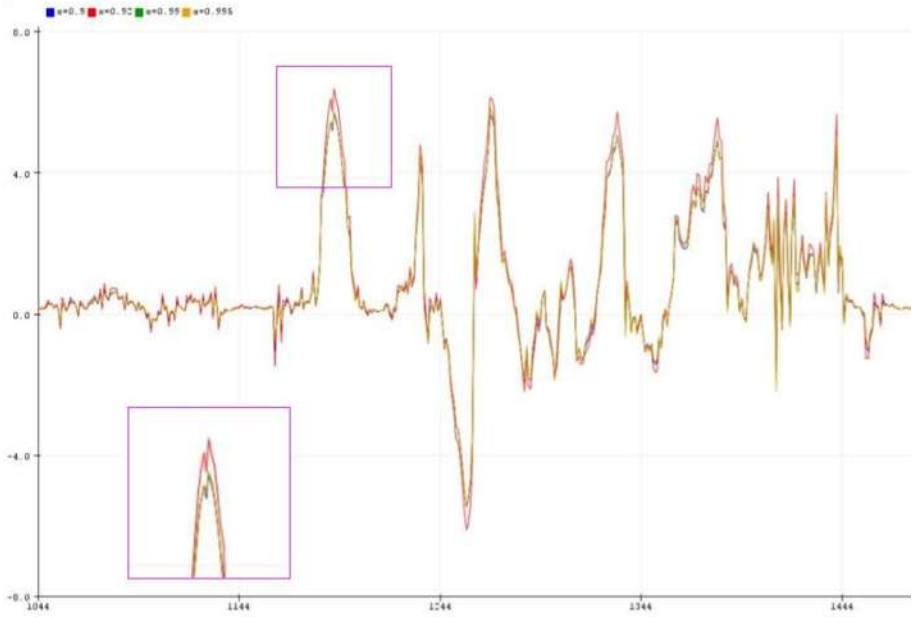


Figura 3.10: Mediciones experimentales del ángulo de inclinación usando filtro complementario con $\alpha = 0.9, 0.92, 0.99, 0.995$ tomada de experimentos.

Para fines del dron donde se busca de una combinación entre respuesta rápida, precisión y eliminación de ruido se usará el valor $\alpha = 0.99$ para el filtro complementario.

3.6. Filtro Media Movil Exponencial (EMA)

En busca de implementar filtros para la reducción del ruido en mediciones por muestreo múltiple de sensores, que en nuestro caso es provocado por vibraciones, se explicará el Filtro Media Movil Exponencial (EMA). Mencionado en [22], este filtro es uno de los más usados en la actualidad en electrónica digital por sus buenos resultados y bajo costo de cálculos y memoria. Su expresión se ve representada de la siguiente manera:

$$A_n = \alpha M + (1 - \alpha)A_{n-1}, \tag{3.7}$$

donde:
A_n es el valor filtrado
A_{n-1} el valor filtrado anterior
M es el valor medido de la señal a filtrar
α es un valor entre 0 y 1.

El filtro EMA da como resultado un dato con información nueva proporcionada por M y un efecto suavizado basado en el valor filtrado anterior A_{n-1}. El resultado A_n es una señal suavizada cuya suavidad depende del valor de α.

Este filtro tiene una ventaja, ante otros filtros que requieren guardar N valores y realizar cálculos con ellos, ya que el filtro EMA solo requiere guardar el valor filtrado anterior, esto aporta eficiencia computacional y ahorro de memoria [22].

Los casos límite del valor α afectan a la señal filtrada de la siguiente manera:

- Un valor de $\alpha = 1$ proporciona la señal sin filtrar, ya que prescinde del efecto filtrado que proporciona la medición anterior.
- Un valor de $\alpha = 0$ provoca que el valor filtrado siempre sea 0, ya que prescinde de la información nueva que aporta la medición al sistema.

Si se disminuye el valor de α aumenta el suavizado de la señal filtrada, además de que aumenta el tiempo de respuesta del sistema, esto se traduce en un retraso entre la señal originada y la filtrada además de que la señal tarda en estabilizarse. El valor óptimo de α dependerá de la señal y de las necesidades de filtrado, los valores más habituales de α se encuentran entre 0.2 y 0.6.

A continuación se muestran distintos comportamientos del filtro, obtenidos de [22], en función del valor de α, se observa cómo cambia la suavidad y el tiempo de respuesta de la señal filtrada (gráfica anaranjada) respecto a la señal real (gráfica azul).

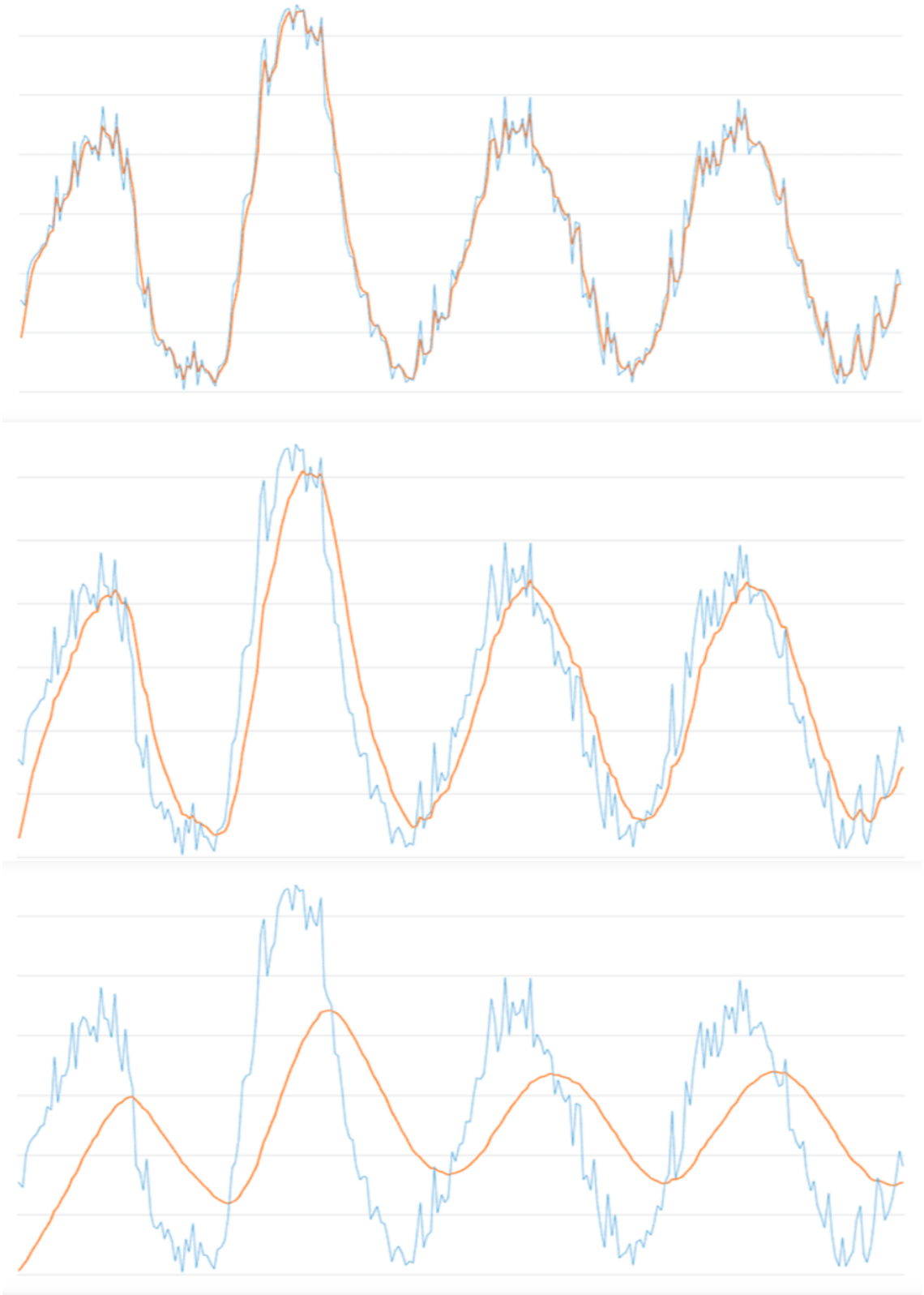


Figura 3.11: Filtro EMA con $\alpha = 0.6, 0.2, 0.05$ [22].

3.7. Polinomios Interpolantes de Lagrange

En experimentos o medición de datos es frecuente encontrarse con tablas de valores, en ocasiones se necesita conocer los valores en puntos distintos a los medidos, La interpolación se usa como un método para resolver este problema, es decir, dados un conjunto de datos en el plano, podemos ajustar una función que exprese el comportamiento de ello, si lo hacemos con un polinomio, es probable que podamos aproximar los valores de la función en puntos ajenos a los dados [39].

Teorema: Sean $x_0, x_1, ..., x_n$ todos ellos distintos y sean $y_o, y_1, ..., y_n$ números reales. Entonces existe un único polinomio $p(x)$ de grado menor o igual que n que cumple que $p(x_k) = y_k$ para $k = 0, 1, ..., n$. Es más, podemos escribir

$$p(x) = y_0 L_0(x) + y_1 L_1(x) + ... + y_n L_n(x) = \sum_{k=0}^{n} y_k L_k(x), \tag{3.8}$$

donde

$$L_k(x) = \prod_{i=0, i\neq k}^{n} \frac{x - x_i}{x_k - x_i}. \tag{3.9}$$

Podemos escribir el polinomio interpolador de una función $f(x)$ en los puntos $x_0, x_1, ..., x_n$ como

$$P_n(x) = \sum_{i=0}^{n} f(x_i) \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}. \tag{3.10}$$

Las dos interpolaciones más usadas son la lineal y parabólica [14]:

Lineal

$$P_1(x) = \frac{x - x_1}{x_0 - x_1} f_0 + \frac{x - x_0}{x1 - x_0} f_1, \tag{3.11}$$

Parabólica

$$P_2(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} f_0 + \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} f_1 + \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} f_2. \tag{3.12}$$

En nuestro caso, lo que se busca al combinar Lagrange, filtro complementario y filtro EMA es predecir el error de los ángulos de inclinación teniendo siempre una medición real, con velocidad de reacción y el menor ruido posible.

3.8. Control PID

El controlador PID es el algoritmo de control más importante del dron, debido a que los algoritmos de estabilidad, orientación, altitud y posición estarán basados en este controlador.

El control PID es un algoritmo que de forma retroalimentada determina el error, definido como la diferencia entre un valor medido y uno deseado. La salida del controlador trata de ajustar un sistema o proceso hasta minimizar el error. Una vez obtenido el error, el algoritmo tiene tres partes, la parte proporcional, integral y derivativa, cada parte tiene una relación con el error y de ahí su nombre. La salida del controlador es la suma de las tres partes. El peso de cada parte del controlador está gobernada por el valor de tres constantes, una para cada parte, estas constantes son K_p, K_i y K_d, refiriéndose a las partes proporcional, integral y derivativa respectivamente [9].

El comportamiento del controlador dependerá totalmente del valor de estas constantes. El valor de las constantes depende del sistema a controlar y el comportamiento que se quiera lograr. En la siguiente imagen se presenta un esquema del control PID, obtenido de [9].

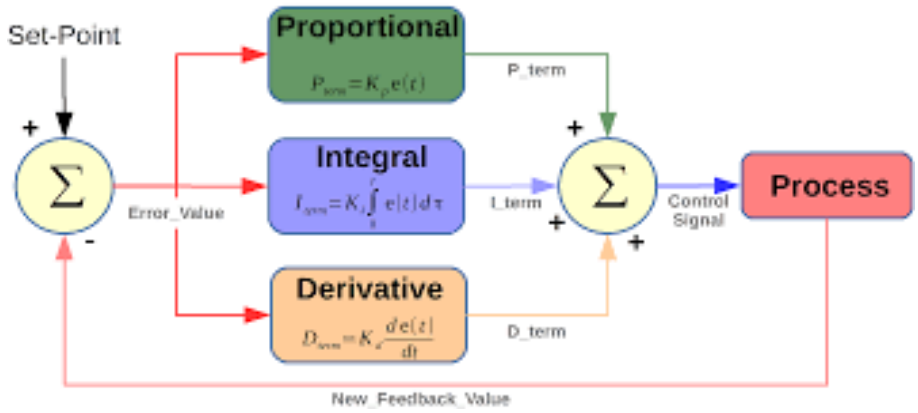


Figura 3.12: Esquema control PID [9].

Una vez definido el error, se pueden considerar cada una de las partes del controlador:

3.8.1. Parte Proporcional

Definida por Cortés [9], la parte proporcional consiste en multiplicar el error por la constante proporcional K_p. Esta parte genera una respuesta que tiende al valor deseado de forma oscilatoria. La parte proporcional da una respuesta en base al presente.

$$P = k_p e(t). \tag{3.13}$$

Por ejemplo, supongamos que el error tiene un valor de 10 y que $K_p = 3$, entonces la parte proporcional tendrá un valor de $3(10) = 30$, ahora supongamos que con la respuesta anterior logramos un error de -3, así el valor de la parte proporcional será $3(-3) = -9$ y el error oscila hacia 0, que es lo que busca el controlador.

Incrementar K_p hace más rápida la respuesta del sistema a los cambios, sin embargo, si K_p es muy alto el sistema puede oscilar sin control y aumentar el error. Por otro lado si K_p es bajo no tendrá efecto en el sistema [9].

3.8.2. Parte Integral

La parte integral suma el error en los tiempos pasados y se multiplica por la constante K_i, sirve para disminuir la oscilación producida por la parte proporcional en el estado estacionario. La parte integral se incrementa en el tiempo si el error es diferente de 0, la parte integral proporciona una respuesta en base al pasado [9].

$$I = K_i \int e(\tau) d\tau. \tag{3.14}$$

Si K_i es baja, el sistema reaccionará más lentamente pero no se sobrepasará el valor deseado, K_i hará más rápida la reacción del sistema pero puede causar oscilaciones.

3.8.3. Parte Derivativa

Mencionada por Cortés [9], la parte derivativa se define como la multiplicación entre la constante K_d y el error:

$$D = K_d \frac{de}{dt}. \tag{3.15}$$

Realizando la derivada del error respecto del tiempo podemos predecir el valor futuro de éste. La derivada actúa solamente cuando el error cambia, si el error no cambia la parte derivativa no tendrá aporte. La parte derivativa suaviza el comportamiento del sistema, funciona de manera semejante a la fuerza de fricción en el movimiento. La parte derivativa proporciona una respuesta en base al futuro.

Si K_d es demasiado alta el sistema sobre reaccionará ante cambios del error, si es baja el sistema no tendrá una respuesta ante dichos cambios [9]. Finalmente la salida del PID es la suma de cada una de las partes:

$$PID = P + I + D. \tag{3.16}$$

3.9. Global Positioning System (GPS)

El Sistema de Posicionamiento Global ha funcionado desde el año 1973 y fue desarrollado por el Departamento de Defensa de Estados Unidos con fines militares, el lanzamiento del primer satélite fue el 22 de Febrero de 1978 [28].

3.9.1. Funcionamiento

El sistema GPS se basa en la medición simultánea de la distancia comprendida entre el receptor y al menos 4 satélites. La información que proporciona el sistema es la posición del receptor y la referencia temporal de alta precisión. Los satélites completan dos vueltas a la tierra al día a una altitud de 20200 Km, durante su recorrido emiten su posición y hora. Cada satélite cuenta con un reloj atómico con una precisión de un segundo en 30 años [7].

El GPS funciona con señales de radio que viajan a la velocidad de la luz, por eso la precisión del reloj debe ser de billonésimas de segundo. La distancia entre el receptor y un satélite se calcula por el retardo temporal desde que el satélite envía la señal hasta que el receptor la recibe. El GPS utiliza satélites en el espacio como puntos de referencia, teniendo acceso a al menos 3 satélites permite triangular la posición del receptor en cualquier parte de la Tierra, aire, mar y el espacio cercano.

Mencionado por Capdevila [7], si se supone que se mide la distancia entre el satélite 1 al receptor y es de d_1 km. Esto significa que la posición del receptor está en una esfera E_{d1} de radio d_1 km con centro en el satélite 1. Si ahora medimos la distancia d_2 del satélite 2 al receptor, este se encuentra en una esfera E_{d2} de radio d_1 km con centro en el satélite 2. Juntando estas dos condiciones, el receptor debe encontrarse en la intersección de esferas E_{d1} y E_{d2}, la posición aún no es precisa, por lo que se toma otro satélite y se retpite el proceso.

Finalmente la posición del receptor se encuentra en la intersección de tres esferas, esta intersección se limita únicamente a dos puntos. Normalmente uno de los puntos resulta improbable por la lejanía de la superficie de la Tierra, sin embargo, con tres satélites la medición de la distancia debe ser extremadamente precisa por nuestro receptor lo cual implicaría que el receptor integre un reloj atómico, sin embargo, el costo de un reloj atómico ronda de 50000 a 100000 dólares, esto haría el GPS inalcanzable para la mayoría de las aplicaciones de uso cotidiano.

Resulta que si tres mediciones perfectas posicionan un punto en un espacio tridimensional, cuatro mediciones imperfectas también lo harán [7].

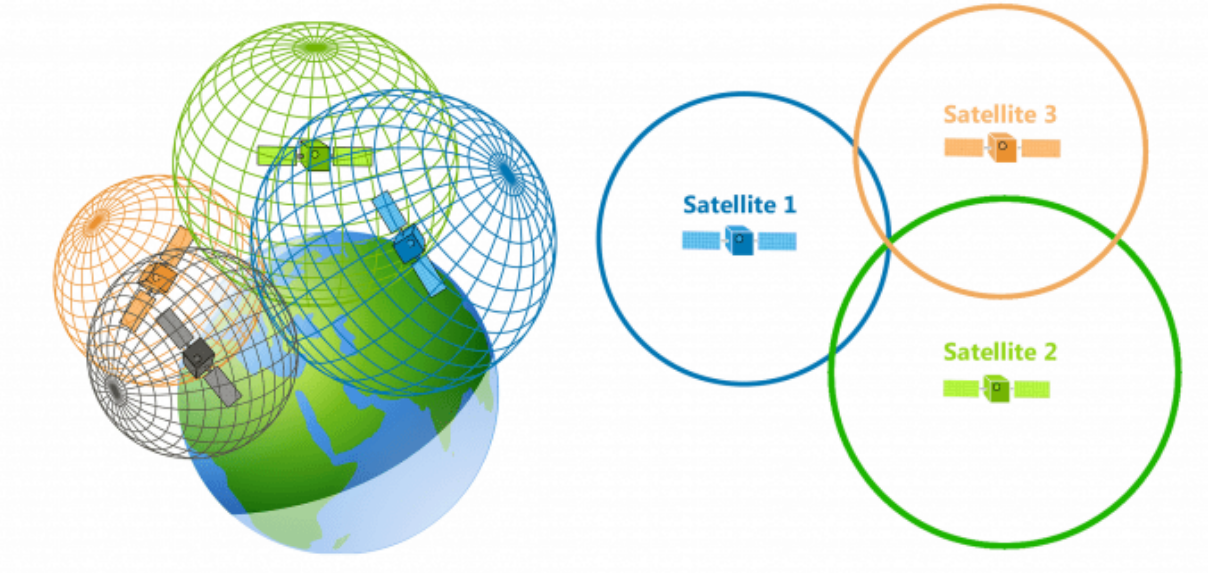


Figura 3.13: Posicionamiento mediante GPS [3].

Tomando la medición de la distancia de un cuarto satélite y al no intersectar con las otras tres mediciones, la computadora del receptor GPS detectará la diferencia y se atribuirá a una sincronización imperfecta con la hora universal, como consecuencia buscará una corrección y ajustará la hora del receptor, esto lo convierte en un reloj atómico. Una vez realizada esta correción tenemos la posición del receptor en un solo punto. La consecuencia principal de esto es que todo GPS debe ser capaz de sintonizar al menos cuatro satélites. Actualmente casi todos los GPS comerciales tienen acceso a más de 6 satélites simultáneamente [7].

4. Componentes Estructurales y Hardware

4.1. Marco F330

Las características del Marco F330 ([38], [13]) son las siguientes:

- Está hecho de fibra de PA66+30GF (Poliamida 66 con 30 % de Fibra de Vidrio).
- Aporta rigidez estructural al dron.
- El marco tiene forma X.
- Se compone de 4 brazos del mismo tamaño y una plataforma central de 2 niveles.
- Cada brazo cuenta con agujeros para fijar los motores.
- Cada brazo cuenta con una pata que sirve como tren de aterrizaje.
- El nivel superior de la plataforma central es óptimo para colocar sobre ella el cerebro del dron.
- El nivel inferior de la plataforma central cuenta con un espacio para colocar la batería.
- El nivel inferior de la plataforma central cuenta con conexiones internas y polos eléctricos +/- para facilitar la conexión de ESCs y batería.
- La plataforma central cuenta con ranuras que facilitan sujeción de componentes.
- La plataforma central cuenta con orificios para la fijación de un tren de aterrizaje.
- Se fija mediante tornillos.
- Es un marco comercial y de fácil acceso.
- Medida diagonal: 330mm.
- Peso: 145g.
- Económico.

Figura 4.1: Marco comercial F330, piezas, ensamble y especificaciones del fabricante.

4.2. Motores A2212/13T 1000kV

Las características del Motor A2212/13T 1000kV, obtenidas de [31], son las siguientes:

- Motor brushless (sin escobillas).
- 1000KV (revoluciones por minuto por Volt).
- Óptimos para drones de más de 800 gramos.
- Compatibles con baterías Lipo 3S.
- Compatibles con distintos tamaños de hélices.
- Se pueden controlar mediante los ESCs.
- Se deben calibrar.
- Cuenta con tres cables de salida: VCC, GND y de comunicación que se conectan al ESC para su control.
- Su sentido de giro es configurable mediante la conexión.
- Cuentan con una base y agujeros para fijarse al marco.
- Las medidas y posición de los agujeros de fijación son compatibles con el marco F450.
- Se fijan mediante tornillos.
- Existen en el mercado almoadillas antivibración para este modelo de motor.
- Eficiencia máxima: 80 %.
- Peso: 52.7g.
- Polos: 14.
- Óptimos para usarse con hélices de 10 o más pulgadas.
- Empuje de 885g con hélice 1045.

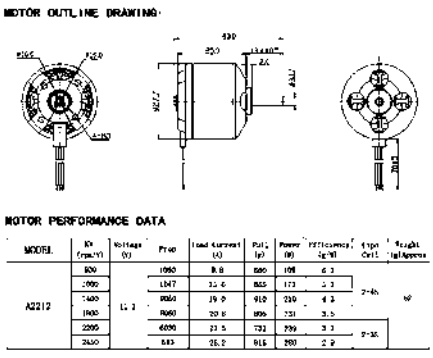


Figura 4.2: Motor A2212/13T 1000Kv, piezas desarmables, ensamble y datos del fabricante.

4.3. Elementos antivibración

Las vibraciones se generan cuando las hélices rompen el viento y los motores giran, se propagan en el dron y afectan los sensores haciendo que sus mediciones presenten ruido y cambios bruscos especialmente en componentes electrónicos sensibles como el acelerómetro y girsocopio, si bien es imposible eliminar completamente las vibraciones, se puede disminuir su propagación al dron y a sus componentes electrónicos de diferentes formas.

4.3.1. Almoadillas antivibración

Las almoadillas antivibración ayudan a que las vibraciones generadas por los motores no se transmitan por el dron. Estas almoadillas se colocan entre el motor y el brazo del dron. Son hechas de caucho y encajan perfectamente con el motor y el brazo del dron [10].

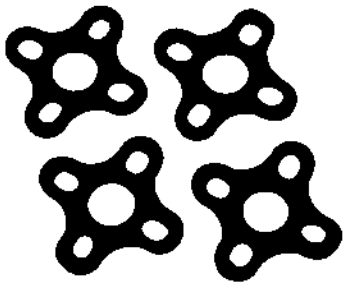


Figura 4.3: Almoadillas antivibración.

4.3.2. Plataforma antivibración

Esta plataforma consta de dos subplataformas: superior e inferior. Sobre la plataforma antivibración superior se coloca el cerebro del dron, esto aisla parcialmente el marco del dron y el cerebro, la plataforma antivibración tiene forma rectangular y en cada esquina una goma une las dos subplataformas. Esta plataforma antivibración se fija a la plataforma superior del marco del dron [29].

Figura 4.4: Plataforma antivibraciones.

4.4. Hélices

Las hélices o propelas tienen como función aprovechar el giro de los motores generando fuerza de sustentación, esta fuerza resulta de el choque del viento con las hélices gracias a la forma de estas. Existen hélices bipala y tripala, dos sentidos de giro: turning clockwise (CW) que giran en sentido de las manecillas del reloj y counter clockwise (CCW) que giran al contrario de las manecillas del reloj. Existen muchos tamaños de hélices, para este proyecto usaremos las hélices 8045 cuyas características tecnicas son:

- Material: ABS mas fibra de carbono nylon.
- Rotación: CW y CCW.
- Cuenta con cojines para ensamblar perfectamente con el motor.
- Recomendadas para usarse con motores brushless A2212/13T
- Cojines propulsor: 2mm / 3mm / 4mm / 5mm / 6mm / 7.8 mm.
- Espesor de centro: 9.7 mm.
- Diámetro hélice: 8 pulgadas.
- Diámetro del eje: 6 mm.
- Distancia entre ejes recomendado: 330 mm - 550 mm.
- Color: variedad de colores.
- Paso: 4.5 pulgadas.
- Peso: 12 g.

Figura 4.5: Hélices 8045.

Cabe mencionar que las hélices son peligrosas cuando están girando, se recomienda mantener medidas de seguridad y equipo de protección para las pruebas con drones en general, evite colocar las hélices para pruebas ajenas al vuelo como medición con sensores.

4.5. Electronic Speed Controler ESC 30A

Un Electronic Speed Controler (ESC) es un circuito electrónico que se encarga de modular la velocidad de un motor y su sentido de giro de un motor conectado a él mediante la regulación de voltaje subministrado para la activación de cada uno de los grupos de bobinados del motor. Este controlador tiene una gran cantidad de aplicaciones, es excelente para desarrollar prototipos de dron, aeromodelismo, ventiladores, turbinas, entre otras [12].

Es necesario calibrar los ESC para que se encuentren coordinados entre sí y sean precisos en la velocidad que se les indica llevar al motor.
Los ESC cuentan con 3 grupos de cables: 2 tercias y una dupla. La dupla es la alimentación del ESC. La tercia de cables azules se conectan al motor, la otra tercia son alimentación de salida a 5V por si deseamos alimentar el chip con el ESC, el último cable por mencionar va conectado a un chip controlador como Arduino. Los ESC funcionan mediante señales PWM de 1 a 2 ms, donde 1 ms es el motor estático y 2 ms es la máxima potencia del motor [17].
Las características obtenidas de [34], son las siguientes:

- Corriente continua: 30A.
- Corriente máxima: 40A
- Salida Regulador BEC: 2A/5V.
- Tipo de batería: 2 – 3 Celdas.
- Programable: Sí.
- Peso: 37 g.

Figura 4.6: ESC 30A.

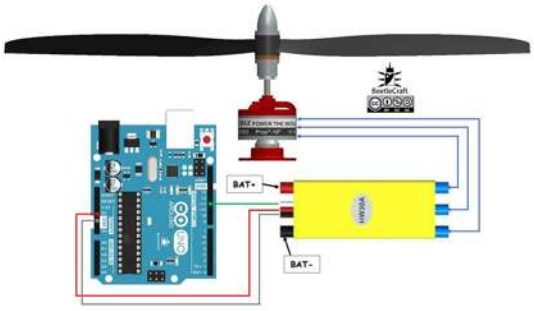


Figura 4.7: Conexión entre motor, ESC y chip controlador Arduino.

4.6. Bateria ZIPPY 2200 mAh 11.1V

La bateria es la encargada de alimentar todos los componentes electrónicos del dron, existen varios tipos de batería de acuerdo al material del que están hechas, en el mundo de los drones y aeromodelismo se usan las baterías de polímeros de litio debido a su alta tasa de descarga, sin embargo, la batería es el elemento donde más pueden evolucionar los drones. Actualmente las baterías son caras y no muy eficientes, sin embargo, las grandes empresas tecnológicas actuales se encuentran desarrollando esta área de la electrificación. Es posible que en esta década las baterías evolucionen aceleradamente y los drones alcancen un tiempo de vuelo superiores de orden de horas y reduciendo el precio de estos.

Las características de la bateria ZIPPY 2200 mAh 11.1V, obtenidas de [36], son:

- Batería a base de polímeros de Litio.
- Actualmente el mejor tipo de batería calidad/precio para drones.
- Capacidad ideal para alimentar motores.
- Contiene tres celdas de 3.7V conectadas en serie.
- 11.1 V.
- 2200 mAh.
- Tasa de descarga de 30-40C.
- Es recargable.
- Peso: 181g.
- Dimensiones: 114x34x24mm
- Conector de descarga: XT60.

Figura 4.8: Batería Lipo 3S 11.1V 2200mAh.

Al momento de elegir la batería del dron se debe considerar la relación de mAh/g, es decir, existen baterías de más capacidad, pero su peso aumenta. El peso es uno de los aspectos más importantes de un dron, con un peso mayor, los motores necesitarán funcionar a mayor velocidad y habrá más consumo de energía, por lo tanto menos tiempo de vuelo.

La batería también es un elemento peligroso del dron, se debe manejar con cuidado y con equipo de protección especialmente si se desean manipular los cables de polaridad.

4.7. Arduino

Arduino se trata de una plataforma de código abierto basado en lenguaje de programación C [16]. Los chips de arduino nos sirven para programar el software que gobierne a los sensores, módulos y actuadores de un aparato electrónico, en este caso un dron. Algo que diferencia a Arduino de otros microcontroladores es la gran comunidad de internautas que colaboran y proponen soluciones a diferentes problemas en los foros oficiales de Arduino. Otra ventaja es que arduino cuenta con librerías que facilitan el uso, calibración y configuración de sensores o módulos.

Arduino cuenta con diferentes placas, cada una para diferentes tipos de proyectos, para un dron hay tres opciones directas: Arduino Uno, Arduino Nano y Arduino Mini cuyas especificaciones proporcionadas por la página oficial [33] son las siguientes:

	Arduino Uno	Arduino Nano	Arduino Mini
Microcontrolador	ATmega328P	ATmega328P	ATmega328P
Voltaje de operación	5V	5V	5V
Voltaje de entrada	7-12V	7-12V	7-9V
Pines digitales	14 (6 PWM)	22 (6 PWM)	14 (6 PWM)
Pines analógicos	6	8	8
Memoria flash	32 KB	32 KB	32 KB
SRAM	2 KB	2 KB	2 KB
Velocidad del reloj	16 MHz	16 MHz	16 MHz
EEPROM	1 KB	1 KB	1 KB
Peso	25g	7g	2g
Largo	68.6 mm	18mm	30mm
Ancho	53.4 mm	45mm	18mm

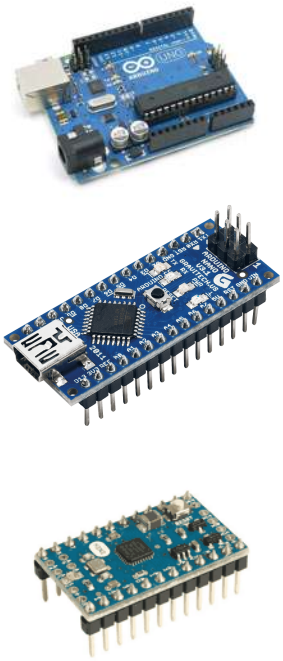


Figura 4.9: Arduino Uno, Nano, Mini.

En la página oficial de arduino https://www.arduino.cc/ podemos encontrar documentación, especificaciones de los componentes electrónicos, foros de dudas, tutoriales, etc.

Se eligió para el proyecto el Arduino Nano debido al número de pines digitales y analógicos, peso, dimensiones y disponibilidad.

4.8. MPU6050

El sensor MPU6050 tiene las siguientes especificaciones técnicas descritas por el fabricante [20].

- Voltaje de alimentación de 3.6V.
- Regulador de tensión propio.
- Giroscópio de tres ejes.
- La sensibilidad del giroscópio puede ajustarse a valores de $\pm 250, \pm 500, \pm 1000$ y $\pm 2000°/s$.
- Dispone de un filtro de paso bajo (DLPF) programable a $5, 10, 20, 42, 98, 188$ y 256 Hz para disipar vibraciones.
- Consumo: 3.6mA.
- Acelerómetro de tres ejes.
- La sensibilidad del giroscópio puede ajustarse a valores de $\pm 2, \pm 4, \pm 8$ y $\pm 16g$.
- Consumo: $500\mu A$.
- Es económico.
- Comucación mediante IIC de hasta 400Hz.

Figura 4.10: MPU6050.

Con el sensor MPU6050 podemos obtener las aceleraciónes en dirección a los ejes x, y, z y las velocidades angulares respecto a los ejes x, y, z. Estos datos serán útiles para calcular otros datos mediante ellos, por ejemplo, el ángulo de inclinación del dron respecto de cada eje y poder desarrollar simulaciones de trayectoria mediante a sus ecuaciones de movimiento.

Es importante que se haga una calibración de este sensor que consiste en cambiar los offsets del sensor, estos offsets se calculan en una superficie totalmente plana y sin aceleraciones sobre el dron diferentes a la gravedad.

Si se desea conocer los datos en unidades específicas, (m/s) y $(°/s)$ por ejemplo, se debe hacer un reescalamiento. Tanto el código de reescalamiento para las unidades mencionadas como el de la calibración del sensor estarán incluidos en la sección de códigos.

4.9. Barómetro MS5611

El Barómetro MS5611 tiene las siguientes especificaciones del fabricante [8]:

- Módulo: GY-63.
- Chip: MS5611.
- Voltaje de operación: 3.3V - 5V DC.
- Interfaz de comunicación digital: I2C o SPI.
- ADC interno de 24 bits.
- Rango de medición Presión: 10 a 1200 mBar.
- Resolución de Presión: 0.012 mbar (configurable por software).
- Rango de medición Temperatura: -40 °C a 85 °C.
- Resolución de Temperatura: 0.01 °C.
- Resolución de Temperatura: 0.01 °C.
- Ultra-bajo consumo de energía: 1 µA (en espera <0,15 µA).

Figura 4.11: Barómetro MS5611.

4.10. Magnetómetro HMC5883L

El módulo magnetómetro HMC5883L cuenta con las siguientes descripciones del fabricante [19]:

- Modelo: GY-271.
- Chip: HMC5883L.
- Magnetómetro, brujula, compás digital de 3 ejes.
- Interfaz con el microcontrolador a través de I2C.
- Alimentación de 3 a 5 volts.
- Rango completo de +/- 8 Gauss.
- Resolución de 5 milli- Gauss.
- Adaptación de niveles para sistemas de 5 volts.
- Precisión de medida: $\pm 2^{o}$.
- Medidas: 14mm x 15 mm.
- Bajo consumo.

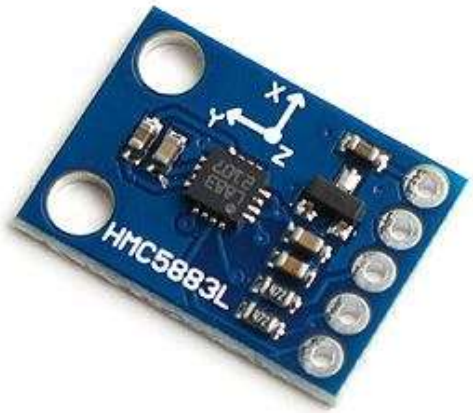


Figura 4.12: Magnetómetro HMC5883L.

4.11. GPS Neo M8N

Las especificaciones del GPS Neo M8N descritas por el fabricante [37] y vendedor [15] son las siguientes:

- Fabricante: U-Blox
- Módulo GPS.
- Serie: NEO-M8N
- Precisión de la posición horizontal: 2.5 m
- Voltaje de alimentación operativo: 1.65 a 3.6 V
- Tipo de interfaz: I2C, SPI, UART, USB
- Temperatura de trabajo máxima: + 85°C
- Temperatura de trabajo mínima: - 40°C
- Velocidad de actualización de navegación de hasta 10 Hz
- -167 dBm de sensibilidad de navegación

Figura 4.13: GPS Neo M8N.

4.12. Sensor ultrasónico HC-SR04

Las especificaciones del Sensor ultrasónico HC-SR04 según [25] son las siguientes:

- Voltaje de Operación: 5V DC.
- Corriente de reposo: <2mA.
- Corriente de trabajo: 15mA.
- Rango de medición: 2cm a 450cm.
- Precisión: ± 3mm.
- Ángulo de apertura: 15°.
- Frecuencia de ultrasonido: 40KHz.
- Duración mínima del pulso de disparo TRIG (nivel TTL): 10 S.
- Duración del pulso ECO de salida (nivel TTL): 100-25000 S.
- Dimensiones: 45mm x 20mm x 15mm.
- Tiempo mínimo de espera entre una medida y el inicio de otra 20ms (recomendable 50ms).

Figura 4.14: Sensor ultrasónico HC-SR04.

4.13. Módulo Bluetooth HC-06

Las especificaciones del módulo Bluetooth HC-06 descritas por [24] son las siguientes:

- Voltaje de operación: 3.3V - 5V DC.
- Corriente de operación: <40mA.
- Corriente modo sleep: <1mA.
- Chip: BC417143
- Bluetooth: V2.0+EDR.
- Frecuencia: Banda ISM de 2,4 GHz.
- Modulación: GFSK (Gaussian Frequency Shift Keying)
- Potencia de emisión: 4 dBm, clase 2.
- Sensibilidad: -84dBm a 0.1
- Alcance 10 metros.
- Interfaz de comunicación: Serial UART TTL.
- Velocidad de transmisión: 1200bps hasta 1.3Mbps.
- Baudrate por defecto: 9600,8,1,n.
- Velocidad asíncrona: 2.1Mbps (máx.) / 160 kbps.
- Velocidad síncrona: 1Mbps/1Mbps.
- Seguridad: Autenticación y encriptación.
- Compatible con Android.
- Dimensiones: 37*16 mm.
- Peso: 3.2 gramos.

Figura 4.15: GPS Neo M8N.

Existen chips que integran varios chips en uno solo, por ejemplo, existe en el mercado el chip GY-86 que integra los chips MPU6050, HMC5883L y MS5611. Además existen versiones de Arduino Nano que ya integran bluetooth.

5. Software

En este capítulo se encuentran los diagramas de conexiones electrónicas, configuraciones y códigos desarrollados para el funcionamiento de cada componente del dron.

5.1. Parámetros

El vuelo, configuración y comportamiento del dron depende de varios parámetros de vuelo, estos pueden ser modificados en tiempo real mientras se está en vuelo, una vez el dron sea autónomo totalmente, estos parámetros serán optimizados y fijados por los algoritmos de control.

El código de la definición de estos parámetros es el siguiente:

```
1  void ParametrosVuelo(){
2  vel=1000;
3  deseoVel=1000;
4  VelDespegue=1450;
5  VelAterrizaje=1150;
6  deseoP_v = 0.0;
7  deseoR_v = 0.0;
8  deseoY_v = 0.0;
9  deseoLon =0;
10 deseoLat =0;
11 deseoAlt=0;
12 Accion = 0;
13 Despegado == false;
14 MPUcalibrado = !false;
15 ESCcalibrado = !false; //WARNING
16 }
```

Las variables con el prefijo deseo en su nombre son las variables a mantener por los sistemas de control, por ejemplo, deseoP_v, deseoR_v, deseoY_v son los ángulos que deseamos que tenga el dron respecto a los ejes x, y, z. En la fase de pruebas estos parámetros pueden modificarse en vuelo mediante comunicación bluetooth.

5.2. HC-06

5.2.1. Conexión y Configuración

Es necesario configurar nuestro modulo de comunicación bluethooth HC-06 a nuestra conveniencia, este viene configurado a una velocidad de transmisión de 9600 baudios, sin embargo esto es muy lento para nosotros. En la configuración del módulo podemos modificar el nombre del dispositivo, la contraseña y la velocidad de transmisión.

Para configurar el módulo debemos conectarlo a nuestro arduino. El módulo Bluetooth HC-06 tiene 4 pins para establecer la conexión que deben ir conectados como sigue:

- Alimentación VCC pin. Normalmente conectado al pin 5V del Arduino.
- Masa GND. Normalmente conectado al pin GND del Arduino.
- Pin de recepción RX. Normalmente conectado a un pin de transmisión Arduino (TX)
- Pin de transmisión TX. Normalmente conectado a un pin de recepción Arduino (RX)

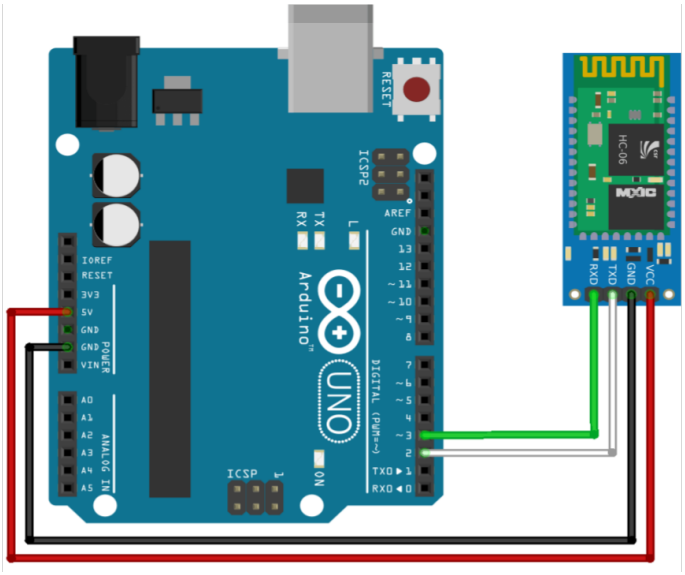


Figura 5.1: Conexión de módulo HC-06 a arduino.

La configuración del módulo Bluetooth puede ser interesante para verificar el funcionamiento del módulo y para modificar sus parámetros, especialmente cuando se utilizan varios módulos. El módulo debe estar alimentado pero no sincronizado (el LED del módulo debe parpadear).

El siguiente código se usa para cambiar el nombre, el código PIN y la velocidad de comunicación (baudrate) del módulo HC-06.

```
#include <SoftwareSerial.h>
SoftwareSerial hc06(2,3);

void setup(){
  //Iinicializamos el monitor serial
  Serial.begin(9600);
  Serial.println("introduce comando AT:");
  //Inicializamos el hc-06
  hc06.begin(9600);
}

void loop(){
  //escribimos del hc-06 a monitor serial
  if (hc06.available()){
    Serial.write(hc06.read());
  }

  //escribe de serial a hc-06
  if (Serial.available()){
    hc06.write(Serial.read());
  }
}
```

Para probar la comunicación, escriba AT en el monitor de serie de Arduino. Seleciona las optiones «No fin de linea» y baudrate 9600 por la comunicación. Si todo está bien, el módulo debe responder OK. Si no funciona, compruebe la conexión y la versión del módulo.

Para cambiar el nombre del módulo, escribe AT + NAMEmodulename. El módulo debe responder OK setname. (Por ejemplo: si desea cambiar el nombre del módulo a BTDron, escriba AT + NAMEBTDron).

Para cambiar el código PIN del módulo, escribe AT + PINxxxx. El módulo debe responder OKsetPIN. (Por ejemplo: si desea cambiar el PIN a 0000, escriba AT + PIN0000).

Para cambiar la velocidad de comunicación del módulo (solo si es necesario), escriba AT + BAUDx. Donde x es un entero de 1 a 8 que se corresponde con una velocidad de trasmisión de la siguiente manera:

- 1 para 1200
- 2 para 2400
- 3 para 4800

- 4 para 9600
- 5 para 19200
- 6 para 38400
- 7 para 57600
- 8 para 115200

Ej: si desea cambiar la tasa de baudios en 9600 tipo AT + BAUD4. El módulo debe responder OK9600.

Para nuestra aplicación del dron usaremos la velocidad de 115200 baudios.

5.2.2. Sincronización con Dispositivo y App

Cuando haya configurado el módulo como desee, puede emparejar el HC-06 con el otro sistema como cualquier otro dispositivo Bluetooth. Seleccione el nombre de la lista de dispositivos detectados e ingrese el código PIN que eligió. Cuando se hace esto, el LED en el módulo debe dejar de parpadear.

Para facilitar la comunicación bluetooth con serial haremos uso de una aplicación gratuita para smartphones android que podemos encontrar de manera gratuita en Playstore de Google llamada Serial Bluetooth Terminal:

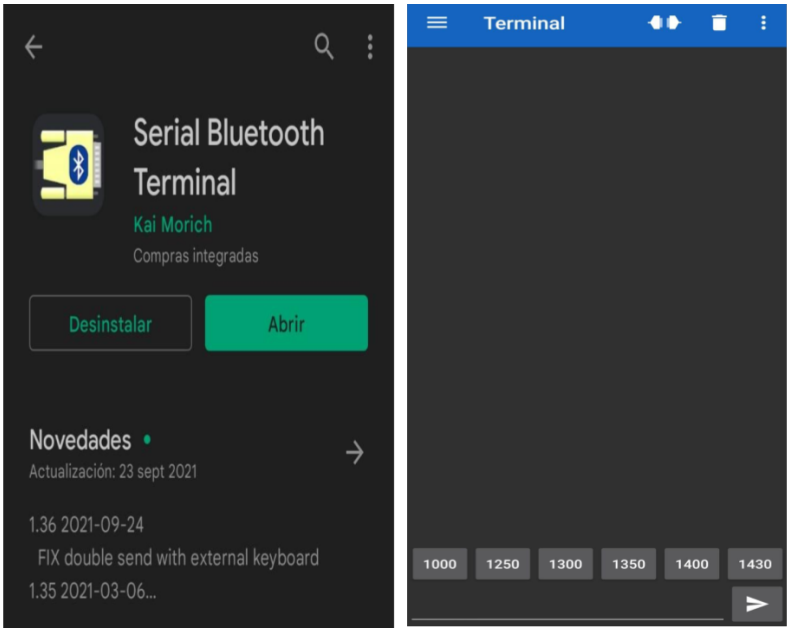


Figura 5.2: Serial Bluetooth Terminal App.

Esta app es muy facil e intuitiva de usar, basta con escribir y enviar los comandos al dispositivo bluetooth emparejado, también podemos recibir datos del dispositivo al smartphone, estos los veremos en la misma app en forma de terminal de computadora.

5.3. Recepción y Lectura de Comandos

La recepción de datos del dron se realiza mediente comunicación serial, es decir, pueden enviarse datos al dron conectándolo a una pc mediante el puerto USB y también de manera inhalámbrica por bluetooth. Para las pruebas se desarrolló un código que permite cambiar parámetros en vuelo, así podemos ajustar los parámetros de los controladores PID sin tener que volver a cargar todo el código a la placa. el código es el siguiente:

```
int dato;
char frase2[10];
boolean datos=false;
String frase,num,letra;
String v="V";

void LecturaSerial(){
//dato=Serial.parseInt();
//Serial.println(dato);
frase=Serial.readString();                          //leemos cadena
frase.toCharArray(frase2, frase.length());          //arreglo de char
for (int i=0;i<frase.length();i++){
if(isDigit(frase2[i])){num= num + frase2[i];}
else{letra=letra + frase2[i];}
}
//Serial.println(letra);Serial.println(num);

if(letra.charAt(0)=='P'){deseoP_v= num.toInt() - 360;}        //deseoInc en P
else if(letra.charAt(0)=='R'){deseoR_v= num.toInt() - 360;}   //deseoInc en R
else if(letra.charAt(0)=='Y'){deseoY_v= num.toInt() - 360;}   //deseoInc en Y
else if(letra.charAt(0)=='A'){deseoAlt= num.toInt();}         //deseo altura
else if(letra.charAt(0)=='L'){
  if(letra.charAt(1)=='O'){deseoLon= num.toFloat();}          //deseo longitud
  else if(letra.charAt(1)=='A'){deseoLat= num.toFloat();}     //deseo latitud
}
else if(letra.charAt(0)=='C'){                                //parametrosPID
  if(letra.charAt(1)=='P'){                                   //PID PITCH
    if(letra.charAt(2)=='P'){Kp_pitch_v=num.toFloat();}
    if(letra.charAt(2)=='I'){Ki_pitch_v=num.toFloat();}
    if(letra.charAt(2)=='D'){Kd_pitch_v=num.toFloat();}
  }
  else if(letra.charAt(1)=='R'){                              //PID ROLL
```

```
33      if(letra.charAt(2)=='P'){Kp_roll_v=num.toFloat();}
34      if(letra.charAt(2)=='I'){Ki_roll_v=num.toFloat();}
35      if(letra.charAt(2)=='D'){Kd_roll_v=num.toFloat();}
36    }
37    else if(letra.charAt(1)=='Y'){                                  //PID YAW
38      if(letra.charAt(2)=='P'){Kp_yaw_v=num.toFloat();}
39      if(letra.charAt(2)=='I'){Ki_yaw_v=num.toFloat();}
40      if(letra.charAt(2)=='D'){Kd_yaw_v=num.toFloat();}
41    }
42    else if(letra.charAt(1)=='A'){                                  //PID ALTITUD
43      if(letra.charAt(2)=='P'){Kp_alt=num.toFloat();}
44      if(letra.charAt(2)=='I'){Ki_alt=num.toFloat();}
45      if(letra.charAt(2)=='D'){Kd_alt=num.toFloat();}
46    }
47    else if(letra.charAt(1)=='L'){
48      if(letra.charAt(2)=='O'){                                     //PID LONGITUD
49        if(letra.charAt(3)=='P'){Kp_lon=num.toFloat();}
50        if(letra.charAt(3)=='I'){Ki_lon=num.toFloat();}
51        if(letra.charAt(3)=='D'){Kd_lon=num.toFloat();}
52      }
53      else if(letra.charAt(2)=='A'){                                //PID LATITUD
54        if(letra.charAt(3)=='P'){Kp_lat=num.toFloat();}
55        if(letra.charAt(3)=='I'){Ki_lat=num.toFloat();}
56        if(letra.charAt(3)=='D'){Kd_lat=num.toFloat();}
57      }
58    }
59  }
60  if(letra.charAt(0)=='V'){deseoVel= num.toInt();}                  //DESEO VELOCIDAD
61  if(deseoVel<=1000){deseoVel=1000;}                                //LIM VELOCIDAD
62
63  //IMPRESI N DE DATOS
64  if(datos==true){Serial.print(F("[V, A, Lon, Lat] = "));
65  Serial.print(" [");Serial.print(deseoVel);Serial.print(",");
66  Serial.print(deseoAlt);Serial.print(",");
67  Serial.print(deseoLon);Serial.print(",");
68  Serial.print(deseoLat);Serial.println("]");
69  Serial.print(F("[ P ,  R ,  Y ] = "));Serial.print(" [");
70  Serial.print(deseoP_v);Serial.print(",");
71  Serial.print(deseoR_v);Serial.print(",");
```

```
72  Serial.print(deseoY_v);Serial.println("]");
73  Serial.println(F("================================"));}
74  num="";letra="";
75  }
```

Mediante este código podemos modificar las inclinaciones del dron respecto a cada eje y la velocidad de vuelo. La forma de cambiar dichos parámetros se explica en el siguiente cuadro:

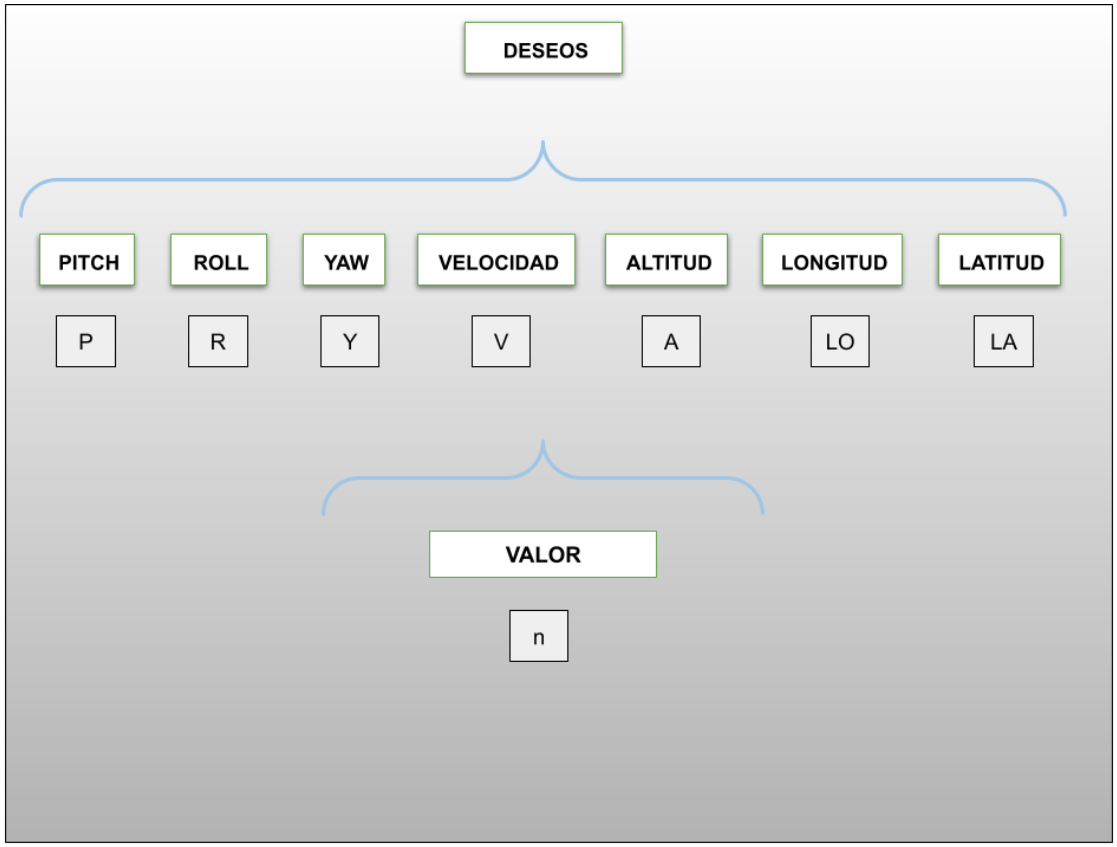


Figura 5.3: Cambio de parámetros deseados.

Los ángulos de inclinación y la velocidad del dron siempre deberán cumplir con:

$$1000 \leq Velocidad \leq 2000.$$

$$anguloreal\_P = deseo\_P - 360.$$

$$anguloreal\_R = deseo\_R - 360.$$

$$anguloreal\_Y = deseo\_Y - 360.$$

Si se desea cambiar el ángulo de inclinación del dron a $+15°$ en pitch, debemos mandar el comando **P 375** al dron.

Si se desea una inclinación de $-10°$ en pitch, debemos mandar el comando **P 350** al dron.

La modificación de las constantes K_P, K_I, K_D de cada controlador PID se realiza de forma análoga, las combinaciones para ello son las siguientes.

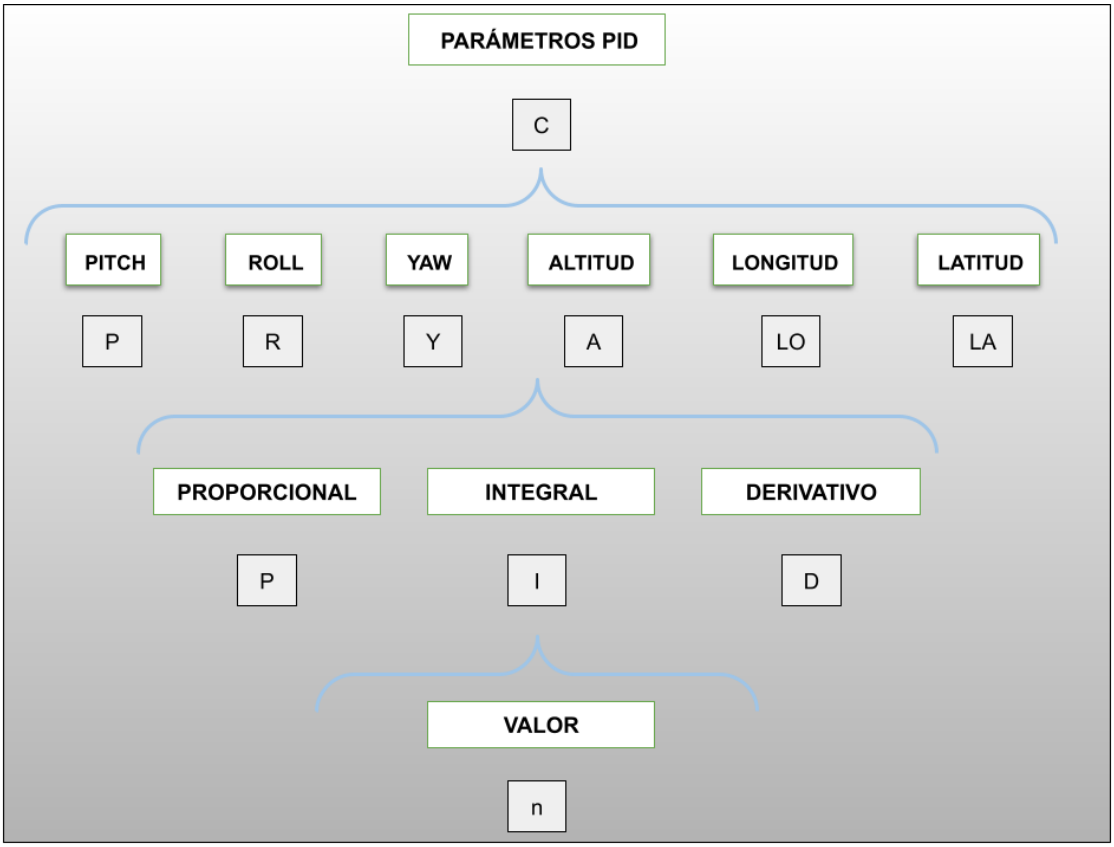


Figura 5.4: Cambio de parámetros PID.

Si se quiere cambiar el parámetro proporcional del control PID pitch a un valor de 2.5 deberá mandar el comando **CPP 2.5** al dron.

Si se quiere cambiar el parámetro derivativo del control PID de altitud a un valor de 15 deberá mandar el comando **CAD 15** al dron.

5.4. MPU6050

Este sensor nos proveerá de las aceleraciones en los 3 ejes y la velocidad angular respecto a cada eje, con estos datos podemos calcular ángulos de inclinación, básicamente este sensor es el más importante para el vuelo del dron.

5.4.1. Conexión

La conexión del módulo al Arduino se realiza como sigue:

- Alimentación VCC pin. Normalmente conectado al pin 5V del Arduino.
- Masa GND. Normalmente conectado al pin GND del Arduino.
- Pin SDA. Conectado a pin A4 del Arduino.
- Pin SCL. Conectado a pin A5 del Arduino.

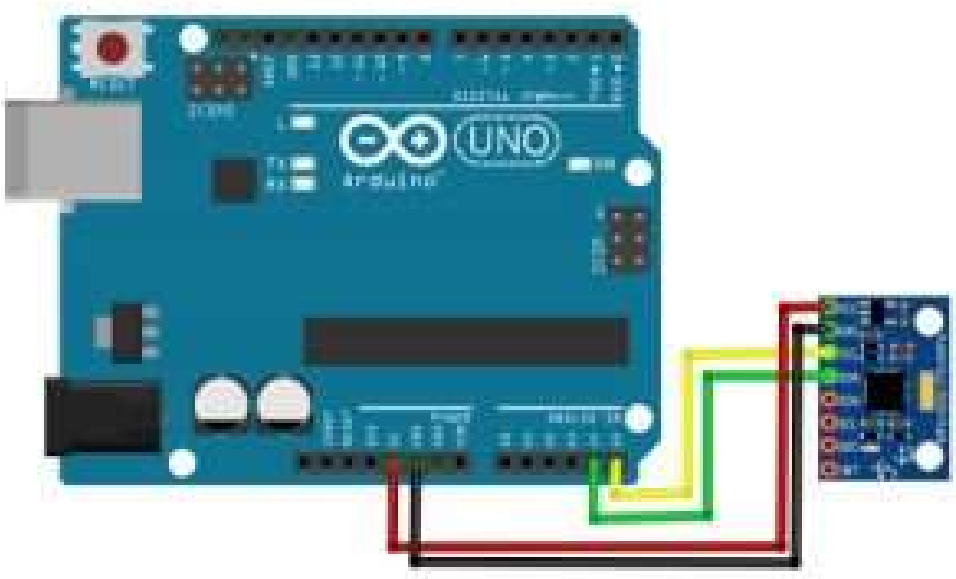


Figura 5.5: Conexión de MPU6050 a arduino.

Usaremos la librería **MPU6050** desarrollada por **Jeff Rowberg**, la librería se descargar de forma gratuita en **https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU6050**.

Esta librería trabaja con una librería adicional **I2C** desarrollada por **Jeff Rowberg** para la comunicación I2C descargable en **https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/I2Cdev**. Ambas librerías consultadas en Julio de 2021.

5.4.2. Iniciación y Configuración

El sensor obtiene mediciones en rangos de $2g/4g/8g/16g$ para el acelerómetro y de $250/500/1000/2000(/s)$ para el giroscopio, además puede activarse el filtro pasa bajos ya integrado en frecuencias de $5, 10, 20, 42, 98, 188, 256$ Hz. Dichos rangos y frecuencias de filtro pueden configurarse con el siguiente código

```
void init_MPU6050() {
  Wire.beginTransmission(gyro_address);
  Wire.write(0x6B);                     //Registro 6B hex)
  Wire.write(0x00);                     //00000000 para activar giroscopio
  Wire.endTransmission();
  Wire.beginTransmission(gyro_address);
  Wire.write(0x1B);                     //Register 1B hex
  Wire.write(0x08);                     //Girscopio a 500dps
  Wire.endTransmission();
  Wire.beginTransmission(gyro_address);
  Wire.write(0x1C);                     //Register (1A hex)
  Wire.write(0x10);                     //Acelerometro a  +/- 8g
  Wire.endTransmission();
  Wire.beginTransmission(gyro_address); // Probar MPU6050
  Wire.write(0x1B);
  Wire.endTransmission();
  Wire.requestFrom(gyro_address, 1);
  while(Wire.available() < 1);
    if(Wire.read() != 0x08){
      digitalWrite(12,HIGH);
      while(1)delay(10);
    }
  // Activar y configurar filtro pasa bajos DLPF que incorpora el sensor
  Wire.beginTransmission(gyro_address);
  Wire.write(0x1A);
  Wire.write(0x04);
  //Filtro pasa bajos->256Hz(0ms):0x00<->188Hz(2ms):0x01<->98Hz(3ms):0x02<->42Hz
      (4.9ms):0x03<->20Hz(8.5ms):0x04<->10Hz(13.8ms):0x05<->5Hz(19ms):0x06
  Wire.endTransmission();
}
```

5.4.3. Calibración

Si el módulo ya está calibrado, definimos en el código de parámetros

```
MPUcalibrado = !false;
```

Si el módulo nunca ha sido calibrado y se desea hacerlo, definimos en el código de parámetros

```
MPUcalibrado = false;
```

El siguiente código toma la definición de MPUcalibrado, si ya está calibrado se toman e imprimen los valores OFFSET anteriormente fijados en la memoria del MPU6050, si no, el código procede a la calibración del módulo.

```
void PrecalibradoMPU(){
boolean confirmacion;
// Leer los offset los offsets anteriores
ax_o=MPU.getXAccelOffset();
ay_o=MPU.getYAccelOffset();
az_o=MPU.getZAccelOffset();
gx_o=MPU.getXGyroOffset();
gy_o=MPU.getYGyroOffset();
gz_o=MPU.getZGyroOffset();

Serial.println(ax_o);
Serial.println(ay_o);
Serial.println(az_o);
Serial.println(gx_o);
Serial.println(gy_o);
Serial.println(gz_o);

Serial.println("Calibrando, no mover IMU");
for (int i=0;i<=20000;i++){calibrarMPU();}
MPUcalibrado==true;
Serial.println("Calibrado");
}
```

Para llevar a cabo la calibración del acelerómetro y giroscopio del MPU6050 este debe encontrarse en una superficie totalmente plana ya que este será el nivel de referencia para la calibración, es decir, se toman

los valores del sensor como que este se encuentra unicamente afectado por la aceleración de la gravedad en dirección z y sin someterse a velocidades angulares.

El calibrado consiste en ajustar cada 100 mediciones el valor de los OFFSETS tomando 20000 mediciones intentando eliminar el error con la medida real que deseamos, es decir $ax = 0, ay = 0, az = -g, gx = 0, gy = 0, gz = 0$, al terminar tenemos los valores de estos OFFSETS que podemos incluir en el código para no calibrar el sensor cada vez que encienda.

El código es el siguiente:

```
1  void calibrarMPU(){
2  // Leer las aceleraciones y velocidades angulares
3  MPU.getAcceleration(&aceX, &aceY, &aceZ);
4  MPU.getRotation(&velX, &velY, &velZ);
5  // Filtrar las lecturas
6  f_ax = f_ax-(f_ax>>5)+aceX;
7  p_ax = f_ax>>5;
8  f_ay = f_ay-(f_ay>>5)+aceY;
9  p_ay = f_ay>>5;
10 f_az = f_az-(f_az>>5)+aceZ;
11 p_az = f_az>>5;
12 f_gx = f_gx-(f_gx>>3)+velX;
13 p_gx = f_gx>>3;
14 f_gy = f_gy-(f_gy>>3)+velY;
15 p_gy = f_gy>>3;
16 f_gz = f_gz-(f_gz>>3)+velZ;
17 p_gz = f_gz>>3;
18 if (counter==100){ //Cada 100 lecturas corregir el offset
19     /*Mostrar las lecturas separadas por un [tab]
20     Serial.print("promedio:"); Serial.print("t");Serial.print(p_ax); Serial.print
           ("\t");Serial.print(p_ay); Serial.print("\t");Serial.print(p_az); Serial.
           print("\t");
21     Serial.print(p_gx); Serial.print("\t");Serial.print(p_gy); Serial.print("\t");
           Serial.println(p_gz);*/
22     //Calibrar el acelerometro a 1g en el eje z (ajustar el offset)
23     if (p_ax>0) ax_o--;
24     else {ax_o++;}
25     if (p_ay>0) ay_o--;
26     else {ay_o++;}
27     if (p_az-16384>0) az_o--;
```

```
28      else {az_o++;}
29      MPU.setXAccelOffset(ax_o);
30      MPU.setYAccelOffset(ay_o);
31      MPU.setZAccelOffset(az_o);
32      Serial.println(ax_o);
33      Serial.println(ay_o);
34      Serial.println(az_o);
35      //Calibrar el giroscopio a 0  /s en todos los ejes (ajustar el offset)
36      if (p_gx>0) gx_o--;
37      else {gx_o++;}
38      if (p_gy>0) gy_o--;
39      else {gy_o++;}
40      if (p_gz>0) gz_o--;
41      else {gz_o++;}
42      MPU.setXGyroOffset(gx_o);
43      MPU.setYGyroOffset(gy_o);
44      MPU.setZGyroOffset(gz_o);
45      Serial.println(gx_o);
46      Serial.println(gy_o);
47      Serial.println(gz_o);
48      counter=0;
49    }
50    counter++;
51  }
```

Una vez obtenidos los OFFSETS definimos variables con estos valores, es necesario 6 OFFSETS, 3 para las aceleraciones en cada eje y 3 para las velocidades angulares respecto a cada eje.

```
1   void YaCalibrado(){
2   ax_o= -3086;
3   ay_o= -672;
4   az_o= 778;
5   gx_o= 46;
6   gy_o= -56;
7   gz_o= 8;
8
9   MPU.setXAccelOffset(ax_o);
10  MPU.setYAccelOffset(ay_o);
11  MPU.setZAccelOffset(az_o);
```

```
12 MPU.setXGyroOffset(gx_o);
13 MPU.setYGyroOffset(gy_o);
14 MPU.setZGyroOffset(gz_o);
15 }
```

Al incluir los OFFSETS ya no es necesario volver a calibrar el sensor, estos quedarán establecidos y al leerlos en el código por el sensor, este se caloibrará automáticamente. Es importante realizar el calibrado antes de modificar la configuración del sensor. Es probable que el valor de OFFSETS para un sensor no sean los mismos que otro sensor del mismo tipo.

5.4.4. Medición, Filtrado y Cálculo de ángulos

Para la medición de los datos del sensor usamos el siguiente codigo:

```
1 void GirosccopioAcelerometro(){
2 dt = (micros()-loop_timer1)/1000;
3 loop_timer1 = micros();
4 MPU.getMotion6(&aceX, &aceY, &aceZ,&velX, &velY, &velZ);
5 //Serial.print("MPU_t:");
6 //Serial.println(dt);
7 }
```

En él registramos el tiempo transcurrido en cada medición, este tiempo será necesario para realizar la integración de la velocidad angular y calcular el ángulo desplazado. Las mediciones del sensor nos dan datos crudos que dependerán de la sensibilidad configurada anteriormente, es decir los valores proporcionados por el giroscópio deben dividirse por un factor como sigue

FS_SEL	Full Scale Range	LSB Sensitivity
0	± 250 °/s	131 LSB/°/s
1	± 500 °/s	65.5 LSB/°/s
2	± 1000 °/s	32.8 LSB/°/s
3	± 2000 °/s	16.4 LSB/°/s

Figura 5.6: Factor de escalado de datos de giroscopio.

Una vez obtenidos los datos del sensor aplicamos los filtros, integraciones y escalados de estos para obtener los ángulos de inclinación del sensor.

```
1  void Angulos(){
2  //velocidades angulares [grad/seg]
3  vel_ang_x= 0.7*vel_ang_x + 0.3*(float)velX/65.5;
4  vel_ang_y= 0.7*vel_ang_y + 0.3*(float)velY/65.5;
5  vel_ang_z= 0.7*vel_ang_z + 0.3*(float)velZ/65.5;
6
7  //calcular angulos con giroscopio [grad]
8  ang_Pitch +=(velX/65.5)*dt/1000; //65.5*1000: 65.5 de conversion,1000 de s a ms.
       grad
9  ang_Roll  +=(velY/65.5)*dt/1000;
10 ang_Pitch += ang_Roll*sin((float)velZ*dt*0.000000266); //0.000000266= (1/65.5)
       *(3.1416/180)/1000
11 ang_Roll -= ang_Pitch*sin((float)velZ*dt*0.000000266);
12
13 //Calcular los  ngulos  con acelerometro [grad]
14 acc_total_vector= sqrt(pow(aceX,2)+pow(aceY,2)+pow(aceZ,2));
15 accel_ang_x=asin((float)aceY / acc_total_vector)*(57.32);
16 accel_ang_y=asin((float)aceX / acc_total_vector)*(-57.32);
17
18 if (set_gyro_angles) {
19 //Calcular angulo de rotaci n con giroscopio y filtro complemento
20 ang_Pitch = 0.99*ang_Pitch + 0.01*accel_ang_x;
21 ang_Roll  = 0.99*ang_Roll + 0.01*accel_ang_y;
22 }
23 else {
24     ang_Pitch = accel_ang_x;
25     ang_Roll = accel_ang_y;
26     ang_Pitch_ant=0;
27     ang_Roll_ant=0;
28     set_gyro_angles = true;
29 }
30 angP = ang_Pitch;
31 angR = ang_Roll;
32
33 //filtro media movil exponencial
34 /*
35 angP = 0.2*ang_Pitch + 0.8*ang_Pitch_ant;
36 angR = 0.2*ang_Roll + 0.8*ang_Roll_ant;
```

```
37  ang_Pitch_ant = angP;
38  ang_Roll_ant = angR;
39  //*/
40  /*
41  vel_ang_x= 0.5*vel_ang_x + 0.5*vel_ang_x_ant;
42  vel_ang_y= 0.5*vel_ang_y + 0.5*vel_ang_y_ant;
43  vel_ang_x_ant= vel_ang_x;
44  vel_ang_y_ant= vel_ang_y;
45  //*/
46  }
```

Con los cálculos de este código obtenemos las inclinaciones ya filtradas del dron respecto al eje x e y.

5.5. MS5611

Este sensor lo usaremos para obtener datos de la presión barométrica y calcular la altitud relativa del dron respecto a una altura inicial de referencia.

5.5.1. Conexión

La conexión del módulo al Arduino se realiza como sigue:

- Alimentación VCC pin. Normalmente conectado al pin 5V del Arduino.
- Masa GND. Normalmente conectado al pin GND del Arduino.
- Pin SDA. Conectado a pin A4 del Arduino.
- Pin SCL. Conectado a pin A5 del Arduino.

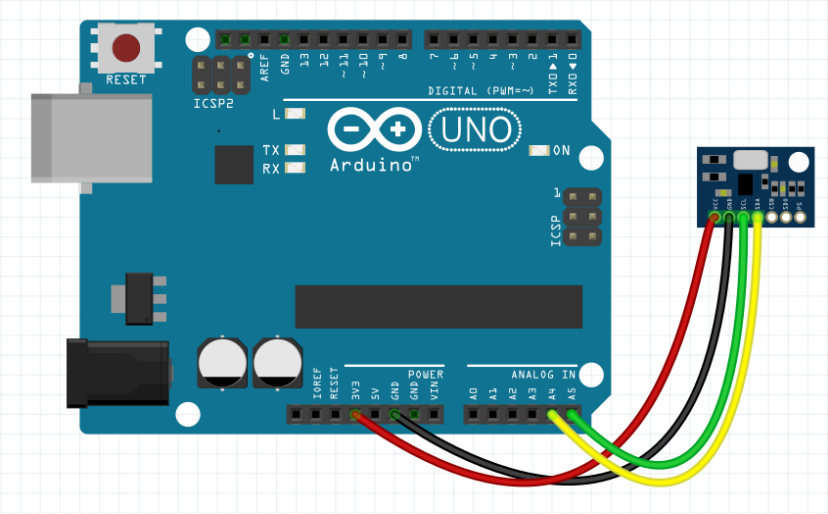


Figura 5.7: Conexiones del MS5611 a Arduino.

Este módulo servirá para medir la presión barométrica, en función de ésta y un valor de referencia, se calcula la altitud relativa al punto de despegue del dron. Esta altitud relativa será usada para mantener una altitud por el controlador PID de altitud.

5.5.2. Código

Usaremos una librería **MS5611** desarrollada por **Korneliusz Jarzebski** descargada en **https://github.com/jarzebski/Arduino-MS5611/blob/master/MS5611.h** consultada en Agosto de 2021.

Abrimos el código de ejemplo Simple siguiente:

```
#include <MS5611.h>
int contadorBARO;
float relativeAltitudetemp;
void checkSettings(){
  //Serial.print("Oversampling: ");
  //Serial.println(ms5611.getOversampling());
}
void Altimetro(){
  contadorBARO ++;
  // Read raw values
  uint32_t rawTemp = ms5611.readRawTemperature();
  uint32_t rawPressure = ms5611.readRawPressure();
  // Read true temperature & Pressure
  double realTemperature = ms5611.readTemperature();
  long realPressure = ms5611.readPressure();
  // Calculate altitude
  float absoluteAltitude = ms5611.getAltitude(realPressure);
  float relativeAltitude = ms5611.getAltitude(realPressure, referencePressure);
/*
  Serial.print(" rawTemp = ");  Serial.print(rawTemp);
  Serial.print(", realTemp = ");  Serial.print(realTemperature);
  Serial.println(" *C");
  Serial.print(" rawPressure = ");  Serial.print(rawPressure);
  Serial.print(", realPressure = ");  Serial.print(realPressure);
  Serial.println(" Pa");
*/
relativeAltitudetemp = relativeAltitudetemp + relativeAltitude;
  //Serial.print(" absoluteAltitude = ");  Serial.print(absoluteAltitude);
  if(contadorBARO == 5){
  Serial.print("Altitud Relativa(m):");  Serial.print(relativeAltitudetemp/6);
  relativeAltitudetemp=0;
  contadorBARO=0;
  Altura = relativeAltitudetemp/6;
  }
}
```

Con este código obtenemos el valor de la altura relativa del dron realizando el promedio de 6 mediciones.

5.6. HMC5883L

El módulo HMC5883L es una brújula digital, nos servirá para calcular el ángulo de inclinación en Yaw del dron respecto al norte geográfico de la Tierra, es necesario considerar en los cálculos la declinación magnética de nuestra localización, dicho dato podemos obtenerlo de **http://www.magnetic-declination.com/**, en el caso de nuestra localización (Morelia, Michoacán) dicha declinación es de **5° 1'**.

5.6.1. Conexión

La conexión del módulo al Arduino se realiza como sigue:

- Alimentación VCC pin. Normalmente conectado al pin 5V del Arduino.
- Masa GND. Normalmente conectado al pin GND del Arduino.
- Pin SDA. Conectado a pin A4 del Arduino.
- Pin SCL. Conectado a pin A5 del Arduino.

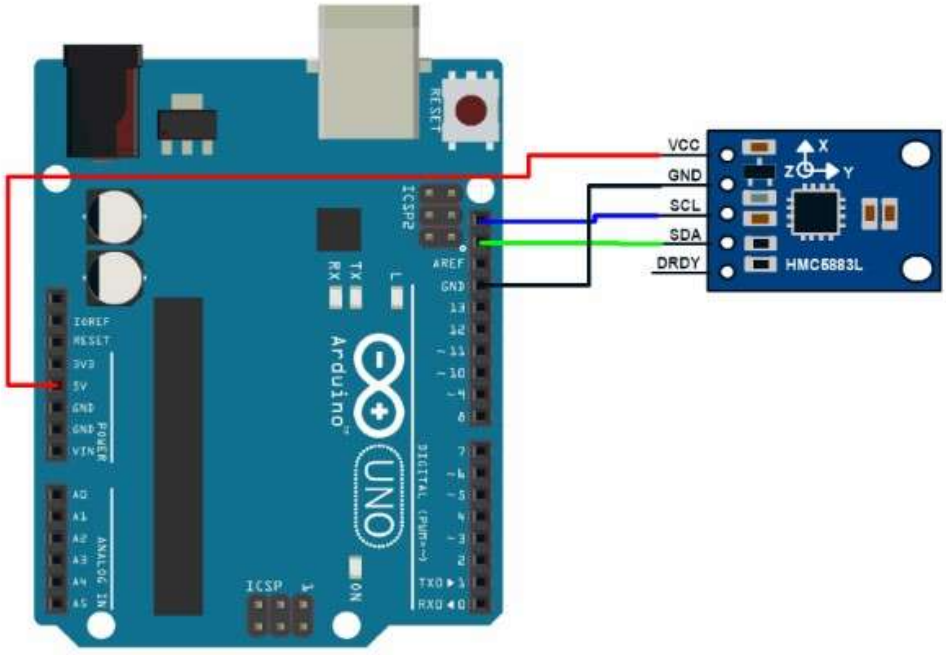


Figura 5.8: Conexiones del HMC5883L a Arduino.

5.6.2. Código

Usaremos la librería **HMC5883L** desarrollada por **Jeff Rowberg**, consultada en Julio de 2021., podemos encontrarla en: **https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/HMC5883L**

Usando el siguiente código

```
int16_t mx, my, mz;
float declinacion=5.02; //declinaci n de +5   1' (Morelia mich)

void setup() {
    Serial.begin(9600);
    Serial.println("Inicializando Magnetometro...");
    //Inicializamos la comunicaci n I2C y el magnet metro
    Wire.begin();
    magnetometro.initialize();
}

void loop() {
    //Obtenemos del magnet metro las componentes del campo magn tico
    magnetometro.getHeading(&mx, &my, &mz);

    //Calculamos el  ngulo  del eje X con respecto al norte
    float angulo = atan2(my, mx);
    angulo=angulo*(180/M_PI);//convertimos de Radianes a grados
    angulo=angulo-declinacion; //corregimos la declinaci n magn tica
    //Mostramos el angulo entre el eje X y el Norte
    Serial.print("AnguloX-N: ");
    Serial.print(angulo,0);

    //calculamos el  ngulo  equivalente de [-180 180] a [0 360]
    if(angulo<0) angulo=angulo+360;
    Serial.print("\tN");
    Serial.println(angulo,0);
}
```

El resultado es el ángulo de inclinación del dron en Yaw respecto al polo norte geográfico de la Tierra.

5.7. GPS

Usaremos el módulo GPS para ubicar en el planeta al dron por medio de sus coordenadas de longitud y latitud, estos datos serán usados por un controlador PID de posición para mantener inmovil al dron si lo deseamos.

5.7.1. Conexión

Las conexiones del GPS al arduino se realizan de la siguiente manera

- Alimentación VCC pin. Normalmente conectado al pin 5V del Arduino.
- Masa GND. Normalmente conectado al pin GND del Arduino.
- Pin de recepción RX. Normalmente conectado a un pin de transmisión Arduino (TX).
- Pin de transmisión TX. Normalmente conectado a un pin de recepción Arduino (RX).

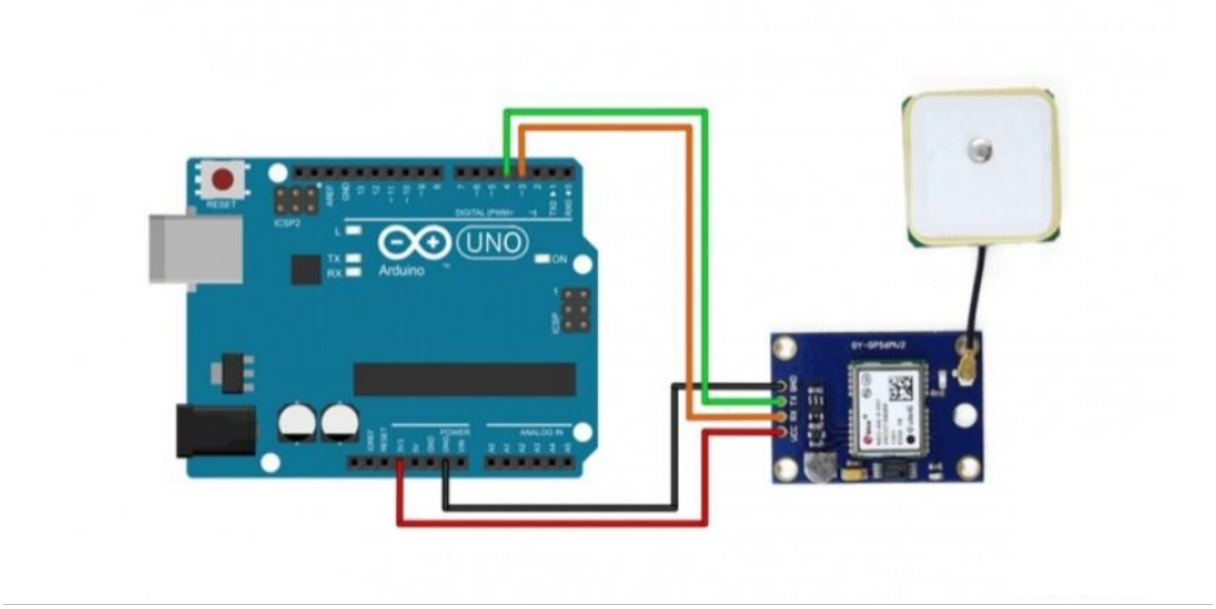


Figura 5.9: Conexiones del GPS M8N a Arduino.

5.7.2. Código

Usaremos el código de ejemplo **FullExample** de la librería **TinyGPS++** consultada en Octubre de 2021, desarrollada por **Mikal Hart**, su descarga es pública y gratuita en https://github.com/mikalhart/TinyGPSPlus/blob/master/src/TinyGPS

El código es el siguiente:

```
#include <TinyGPS++.h>
#include <SoftwareSerial.h>
/*
   This sample sketch demonstrates the normal use of a TinyGPS++ (TinyGPSPlus)
       object.
   It requires the use of SoftwareSerial, and assumes that you have a
   4800-baud serial GPS device hooked up on pins 4(rx) and 3(tx).
*/
static const int RXPin = 4, TXPin = 3;
static const uint32_t GPSBaud = 4800;

// The TinyGPS++ object
TinyGPSPlus gps;

// The serial connection to the GPS device
SoftwareSerial ss(RXPin, TXPin);

void setup()
{
  Serial.begin(115200);
  ss.begin(GPSBaud);

  Serial.println(F("DeviceExample.ino"));
  Serial.println(F("A simple demonstration of TinyGPS++ with an attached GPS
      module"));
  Serial.print(F("Testing TinyGPS++ library v. ")); Serial.println(TinyGPSPlus::
      libraryVersion());
  Serial.println(F("by Mikal Hart"));
  Serial.println();
}

void loop()
```

```
30  {
31    // This sketch displays information every time a new sentence is correctly
          encoded.
32    while (ss.available() > 0)
33      if (gps.encode(ss.read()))
34        displayInfo();
35
36    if (millis() > 5000 && gps.charsProcessed() < 10)
37    {
38      Serial.println(F("No GPS detected: check wiring."));
39      while(true);
40    }
41  }
42
43  void displayInfo()
44  {
45    Serial.print(F("Location: "));
46    if (gps.location.isValid())
47    {
48      Serial.print(gps.location.lat(), 6);
49      Serial.print(F(","));
50      Serial.print(gps.location.lng(), 6);
51    }
52    else
53    {
54      Serial.print(F("INVALID"));
55    }
56
57    Serial.print(F("  Date/Time: "));
58    if (gps.date.isValid())
59    {
60      Serial.print(gps.date.month());
61      Serial.print(F("/"));
62      Serial.print(gps.date.day());
63      Serial.print(F("/"));
64      Serial.print(gps.date.year());
65    }
66    else
67    {
```

```
68      Serial.print(F("INVALID"));
69    }
70
71    Serial.print(F(" "));
72    if (gps.time.isValid())
73    {
74      if (gps.time.hour() < 10) Serial.print(F("0"));
75      Serial.print(gps.time.hour());
76      Serial.print(F(":"));
77      if (gps.time.minute() < 10) Serial.print(F("0"));
78      Serial.print(gps.time.minute());
79      Serial.print(F(":"));
80      if (gps.time.second() < 10) Serial.print(F("0"));
81      Serial.print(gps.time.second());
82      Serial.print(F("."));
83      if (gps.time.centisecond() < 10) Serial.print(F("0"));
84      Serial.print(gps.time.centisecond());
85    }
86    else
87    {
88      Serial.print(F("INVALID"));
89    }
90
91    Serial.println();
92  }
```

Con este código obtenemos del GPS longitud, latitud, número de satélites, fecha, hora, velocidad, etc. Para esta aplicación solo será de interés la longitud y latitud de nuestro dron.

5.8. Constantes PID

En el siguiente código se definen y dan valor a los parámetros de cada controlador PID: las constantes de las partes proporcional, integral y derivativa, así como los valores límite de la parte integral y el valor de la salida PID.

```
int a,b,c,d,tol;
float P_pitch_v, I_pitch_v, D_pitch_v, PID_pitch_v;
float P_roll_v, I_roll_v, D_roll_v, PID_roll_v;
float P_yaw_v, I_yaw_v, D_yaw_v, PID_yaw_v;
float P_alt, I_alt, D_alt, PID_alt;
float P_lon, I_lon, D_lon, PID_lon;
float P_lat, I_lat, D_lat, PID_lat;
float PID_lim,I_lim,PID_pos_lim,I_pos_lim,PID_alt_lim,I_alt_lim;
void constantesPID(){
//============================================================ANGULOS
PID_lim=200;
I_lim=25;
//================================PITCH
Kp_pitch_v=1;//2.2;//2.8;
Ki_pitch_v=0;//0.001;//0.012;
Kd_pitch_v=0;//48;//-65;
//================================ROLL
Kp_roll_v= 0;//2.8;1.9
Ki_roll_v= 0;//0.012;
Kd_roll_v= 0;//-65;
//================================YAW
Kp_yaw_v=0;
Ki_yaw_v=0;
Kd_yaw_v=0;
//============================================================ALT
Kp_alt=0;
Ki_alt=0;
Kd_alt=0;
I_alt_lim=100;
PID_alt_lim=200;
//============================================================POSICION
I_pos_lim =100;
PID_pos_lim=200;
```

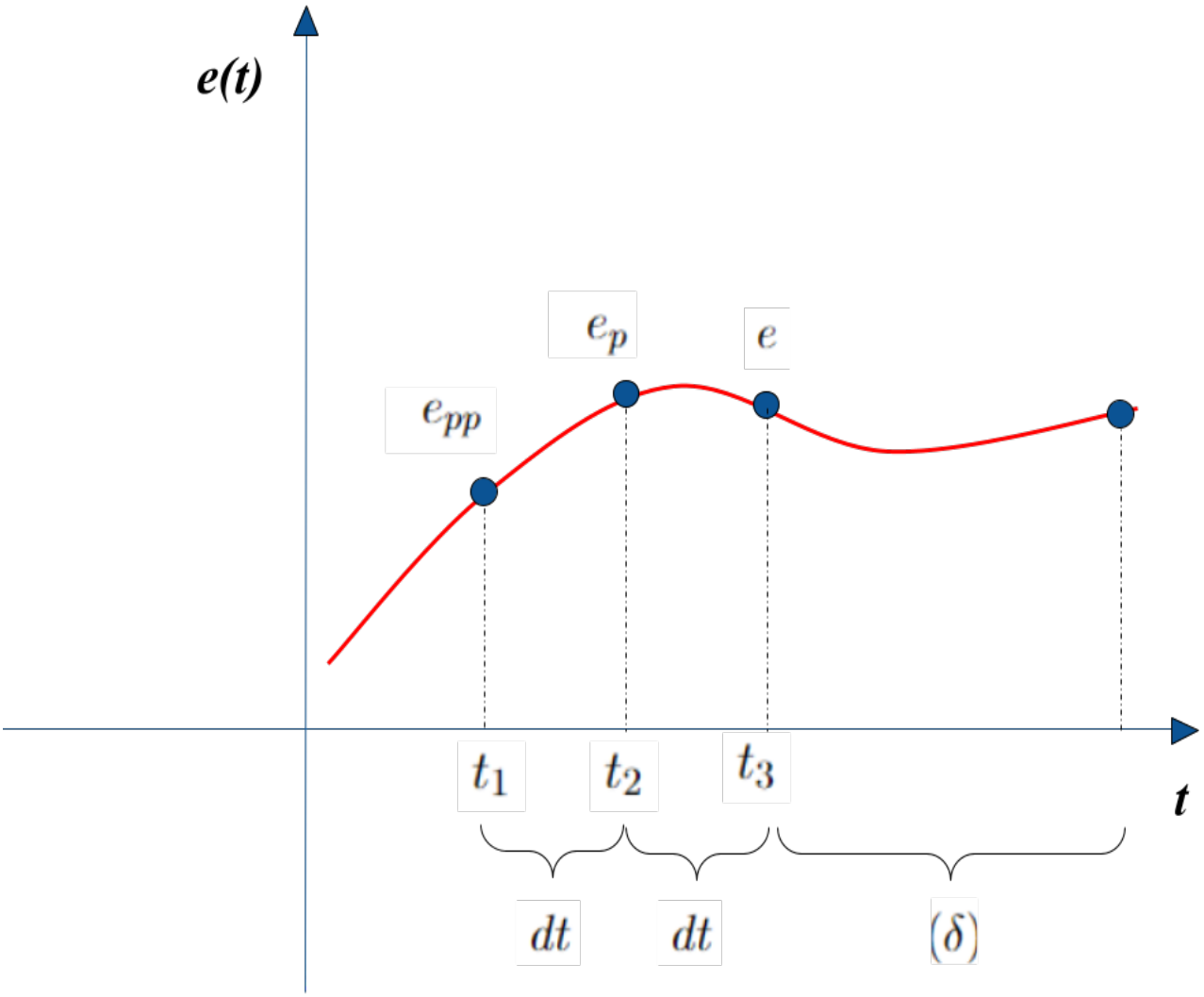
```
34  //================================LON
35  Kp_lon=0;
36  Ki_lon=0;
37  Kd_lon=0;
38  //================================LAT
39  Kp_lat=0;
40  Ki_lat=0;
41  Kd_lat=0;
42  }
```

5.9. Control PID de Ángulos

Una vez medidos los ángulos de inclinación del dron, el controlador PID de ángulos mantendrá el dron a una inclinación deseada, en el modo estable esta inclinación deseada es de 0 grados para los ángulos respecto a pitch y roll.

Con intención de mejorar este controlador se implementó interpolación de Lagrange de segundo orden para predecir el error en el siguiente ciclo y así actuar de forma anticipada.

La aproximación de Lagrange de segundo orden aplicada al error del control PID en nuestro proyecto se visualiza con la siguiente imagen.



Se eligió dicha predicción por su baja demanda de cálculos y memoria del chip.

En base a la imagen anterior, la expresión (3.10) se lee como:

$$e(\delta) = \frac{(dt+\delta)\delta}{2dt^2}e_{pp} - \frac{(2dt+\delta)\delta}{dt^2}e_p + \frac{(2dt+\delta)(dt+\delta)}{2dt^2}e. \tag{5.1}$$

Donde:

e_{pp} := error pasado pasado.

e_p := error pasado.

e := error actual.

dt := tiempo entre mediciónes.

El código implementado del controlador PID de ángulos con la interpolación de Lagrange es el siguiente:

```
1  void PID_av(){
2  //INTERPOLACION DE LAGRANGE PARA PREDECIR ERROR DELTA MILISEGUNDOS DESPUES
3  //_ea: ahora. _ep: pasado. _epp:pasadopasado
4  delta = dt + 1.65;                  //1.65 ES TARDANZA ENTRE MEDIR Y ACCION
5  //=================================================================PITCH PID
6  pitch_ea = angP - deseoP_v;                                          //Error
7  //ORDEN 1
8  //pitch_error_v = -(delta/dt)*pitch_ep + ((dt + delta)/dt)*pitch_ea;
9  //ORDEN 2
10 pitch_error_v=((dt+delta)*delta/(2*pow(dt,2)))*pitch_epp - ((2*dt + delta)*delta/
       pow(dt,2))*pitch_ep + (2*dt+delta)*(dt+delta)/(2*pow(dt,2))*pitch_ea;
11
12 pitch_ep = pitch_ea;
13 pitch_epp = pitch_ep;
14
15 P_pitch_v = Kp_pitch_v*pitch_error_v;                               //Parte P
16 I_pitch_v += (Ki_pitch_v*pitch_error_v);                            //Parte I
17 I_pitch_v = constrain(I_pitch_v,-I_lim,I_lim);                    //Limitar I
18 D_pitch_v = Kd_pitch_v*(pitch_error_v-pitch_error_anterior_v);      //Parte D
19 PID_pitch_v = P_pitch_v + I_pitch_v + D_pitch_v;              //-->Salida PID
20 PID_pitch_v = constrain(PID_pitch_v,-PID_lim,PID_lim)           //Limitar PID
21 pitch_error_anterior_v = pitch_error_v;                       //Error Anterior
22
23 //=================================================================ROLL PID
24 roll_ea = angR-deseoP_v;                                            //Error
25
26 //ORDEN 1
```

```
27 //roll_error_v = -(delta/dt)*roll_ep + ((dt + delta)/dt)*roll_ea;
28 //ORDEN 2
29 roll_error_v=((dt+delta)*delta/(2*pow(dt,2)))*roll_epp - ((2*dt + delta)*delta/pow
       (dt,2))*roll_ep + (2*dt+delta)*(dt+delta)/(2*pow(dt,2))*roll_ea;
30
31 roll_ep = roll_ea;
32 roll_epp = roll_ep;
33
34 P_roll_v = Kp_roll_v*pow(roll_error_v,1);                          //Parte P
35 I_roll_v += Ki_roll_v*(roll_error_v);                              //Parte I
36 I_roll_v = constrain(I_roll_v,-I_lim,I_lim);                      //Limitar I
37 D_roll_v = Kd_roll_v*pow(roll_error_v-roll_error_anterior_v,1);    //Parte D
38 PID_roll_v = P_roll_v + I_roll_v + D_roll_v;                  //-->Salida PID
39 PID_roll_v = constrain(PID_roll_v,-PID_lim,PID_lim);          // Limitar PID
40 roll_error_anterior_v = roll_error_v;                       // Error Anterior
41
42 //=================================================================YAW PID
43 yaw_error_v = deseoY_v - ang_Yaw;                                  //Error
44 P_yaw_v = Kp_yaw_v*yaw_error_v;                                    //Parte P
45 I_yaw_v += Ki_yaw_v*yaw_error_v;                                   //Parte I
46 I_yaw_v = constrain(I_yaw_v,-I_lim,I_lim);                       //Limitar I
47 D_yaw_v  = Kd_yaw_v*(yaw_error_v-yaw_error_anterior_v);           //Parte D
48 PID_yaw_v  = P_yaw_v + I_yaw_v + D_yaw_v;                    //-->Salida PID
49 PID_yaw_v  = constrain(PID_yaw_v,-PID_lim,PID_lim);          //Limitar  PID
50 yaw_error_anterior_v = yaw_error_v;                         //Error Anterior
51
52 //=================================================================================
53 //dt_PID = (micros()-loop_timer2)/1000;
54 //loop_timer2 = micros();
55 //Serial.print("PID_t:");Serial.println(dt_PID);
56 PWM();                                        //Calculo de potencias a motores
57 }
```

Una vez superadas las pruebas de este controlador el dron será capaz de mantener una inclinación deseada y mantenerse estable en vuelo.

5.10. Control PID de Altitud

El código del control PID de altitud funciona con los datos de la altitud relativa proporcionados por el sensor barométrico MS5611 y es el siguiente

```
void PID_altura(){
alt_error = deseoAlt - Altura;                                  //error
P_alt = Kp_alt*alt_error;                                       //Parte proporcional
I_alt += Ki_alt*alt_error;                                      //Parte integral
I_alt = constrain(I_alt,-I_alt_lim,I_alt_lim);                  //Lim parte integral
D_alt  = Kd_alt*(alt_error-alt_error_anterior);                 //Parte Derivativa
PID_alt  = P_alt + I_alt + D_alt;                               //->salida PID
PID_alt  = constrain(PID_alt,-PID_alt_lim,PID_alt_lim);         //Limitar salida
alt_error_anterior = alt_error;                                 //Error enterior
}
```

La salida del PID de altitud reemplazará al parámetro velocidad una vez que el dron sea autónomo, así los motores aumentarán, disminuirán o mantendrán su potencia según el dron deba ascender, descender o mantener una altitud.

Una vez completas y superadas las pruebas de este controlador podemos implementar algoritmos de despegue y aterrizaje autónomo.

5.11. Control PID de Posición

El código del control PID para la posición funciona con las coordenadas de latitud y longitud proporcionadas por el GPS y es el siguiente

```
1  void PID_posicion(){
2
3  //=============================================================LONGITUD
4  lon_error = deseoLon - LongitudGPS;                              //error
5  P_lon = Kp_lon*lon_error;                                        //Parte proporcional
6  I_lon += Ki_lon*lon_error;                                       //Parte integral
7  I_lon = constrain(I_lon,-I_pos_lim,I_pos_lim);                   //Lim parte integral
8  D_lon  = Kd_lon*(lon_error-lon_error_anterior);                  //Parte Derivativa
9  PID_lon  = P_lon + I_lon + D_lon;                                //-->salida PID
10 PID_lon  = constrain(PID_lon,-PID_pos_lim,PID_pos_lim);          //Limitar salida
11 lon_error_anterior = lon_error;                                  //Error enterior
12
13 //=============================================================LATITUD
14 lat_error = deseoLat - LatitudGPS;                              //error
15 P_lat = Kp_lat*lat_error;                                       //Parte proporcional
16 I_lat += Ki_lat*lat_error;                                      //Parte integral
17 I_lat = constrain(I_lat,-I_pos_lim,I_pos_lim);                  //Lim parte integral
18 D_lat  = Kd_lat*(lat_error-lat_error_anterior);                 //Parte Derivativa
19 PID_lat  = P_lat + I_lat + D_lat;                               //->salida PID
20 PID_lat  = constrain(PID_lat,-PID_pos_lim,PID_pos_lim);         //Limitar salida
21 lat_error_anterior = lat_error;                                 //Error enterior
22 }
```

La salida de los controladores PID de longitud y latitud afectarán los parámetros deseoP_v y deseoR_v haciendo que el dron modifique sus ángulos de inclinación para realizar movimientos en las direcciones x e y.

Una vez completas y superadas las pruebas de este controlador, podemos implementar el viaje autónomo del dron entre dos puntos mediante coordenadas GPS y mantenerse inmóvil en una posición.

5.12. Motores

5.12.1. Inicialización y Calibración de Motores

Para la inicialización de los motores simplemente definimos en el código el pin al que está conectado el ESC del motor.

Para la calibración es necesario revisar los datasheet de cada fabricante, en nuestro caso, debemos de mandar a los motores pulsos máximos (2000 microsegundos) por 3 segundos, escucharemos un par de **beep** cortos, esto indica que los motores se están calibrando. Después mandamos pulsos mínimos (2000 microsegundos) por otros 3 segundos, finalizado este proceso escucharemos tres **beep** cortos y seguido un **beep** largo, esto indica que los ESC y los motores están calibrados. Una vez calibrados los motores, estos tendrán el mismo rango de velocidad.

```
1  void InicializarMotores(){
2  pinMode(9, OUTPUT);
3  pinMode(3, OUTPUT);
4  pinMode(10, OUTPUT);
5  pinMode(11, OUTPUT);
6  }
7  void CalibracionESC(){
8  //// ESC inicializacion y calibracion
9  previousMillis = millis();
10 Serial.println("throttle up");
11 while (currentMillis < 3000) {
12   currentMillis = millis() - previousMillis;
13   Serial.println(currentMillis);
14   digitalWrite(9, HIGH);
15   digitalWrite(3, HIGH);
16   digitalWrite(10, HIGH);
17   digitalWrite(11, HIGH);
18   delayMicroseconds(max_throttle);
19   digitalWrite(9, LOW);
20   digitalWrite(3, LOW);
21   digitalWrite(10, LOW);
22   digitalWrite(11, LOW);
23   delay(20);
24 } //beep- beep-
25 currentMillis = 0;
```

```
previousMillis = millis();
Serial.println("throttle down");
while (currentMillis < 4500) {
  currentMillis = millis() - previousMillis;
  Serial.println(currentMillis);
  digitalWrite(9, HIGH);
  digitalWrite(3, HIGH);
  digitalWrite(10, HIGH);
  digitalWrite(11, HIGH);
  delayMicroseconds(min_throttle);
  digitalWrite(9, LOW);
  digitalWrite(3, LOW);
  digitalWrite(10, LOW);
  digitalWrite(11, LOW);
  delay(20);
} // beep--  // 1 2 3
Serial.println("ESC calibrados");
}
```

Este proceso se suele llevar a cabo con un radiocontrol, sin embargo por los elevados precios y por no ser el sentido del proyecto prescindimos de él. Básicamente el código anterior emula el proceso con el radiocontrol.

5.12.2. Cálculo de potencia a motores

Una vez que se han calculado las salidas PID de cada controlador, la potencia con la que debe actuar cada motor depende de la orientación del sensor MPU6050 y cómo están posicionados los motores, en el caso de nuestra consfiguración del dron, las potencias se calculan como sigue:

```
void PWM(){

if(vel>=VelDespegue){
  PotenciaM1  =  vel  + PID_pitch_v + PID_roll_v + PID_yaw_v;
  PotenciaM2  =  vel  + PID_pitch_v - PID_roll_v - PID_yaw_v;
  PotenciaM3  =  vel  - PID_pitch_v - PID_roll_v + PID_yaw_v;
  PotenciaM4  =  vel  - PID_pitch_v + PID_roll_v - PID_yaw_v;
}
else{
  PotenciaM1  =  vel;
  PotenciaM2  =  vel;
```

```
12    PotenciaM3  =  vel;
13    PotenciaM4  =  vel;
14    I_pitch_v=0;
15    I_roll_v=0;
16    I_yaw_v=0;
17  }
18  //================================--LIMITACIONES--====================
19  if(PotenciaM1<=1000){PotenciaM1=1000;}if(PotenciaM2<=1000){PotenciaM2=1000;}
20  if(PotenciaM3<=1000){PotenciaM3=1000;}if(PotenciaM4<=1000){PotenciaM4=1000;}
21  if(PotenciaM1>=1950){PotenciaM1=1950;}if(PotenciaM2>=1950){PotenciaM2=1950;}
22  if(PotenciaM3>=1950){PotenciaM3=1950;}if(PotenciaM4>=1950){PotenciaM4=1950;}
23  //=====================================================================
24  //dt_PWM = (micros()-loop_timer3)/1000;
25  //loop_timer3 = micros(); Serial.print("Pwm_t:");Serial.println(dt_PWM);
26  }
```

En el código la influencia de los controladores PID sobre los motores solo se permite si la velocidad del dron es mayor a la de despegue, esto evitará que al despegar reaccione de manera violenta. También se limita a que la velocidad de los motores siempre se encuentre entre 1 y 2 microsegundos.

5.12.3. Accionamiento de motores

Para accionar los motores una vez calculada la potencia que se les debe administrar tenemos el siguiente código

```
1
2   bool aM1, aM2, aM3, aM4 = false;
3   float accion_m1, accion_m2, accion_m3, accion_m4;
4   void Vuelo() {
5     digitalWrite(9, HIGH); //Motor 1 HIGH
6     digitalWrite(3, HIGH); //Motor 2 HIGH
7     digitalWrite(10, HIGH); //Motor 3 HIGH
8     digitalWrite(11, HIGH); //Motor 4 HIGH
9     aM1 = true; aM2 = true; aM3 = true; aM4 = true;
10    accion_m1 = PotenciaM1 + loop_timer;
11    accion_m2 = PotenciaM2 + loop_timer;
12    accion_m3 = PotenciaM3 + loop_timer;
13    accion_m4 = PotenciaM4 + loop_timer;
14    while (aM1 || aM2 || aM3 || aM4 == true) {
```

```
15      esc_loop_timer = micros();
16      if (accion_m1 <= esc_loop_timer) { // Motor LOW
17        aM1 = false;
18        digitalWrite(9, LOW);
19      }
20      esc_loop_timer = micros();
21      if (accion_m2 <= esc_loop_timer) { // Motor 2 LOW
22        aM2 = false;
23        digitalWrite(3, LOW);
24      }
25      esc_loop_timer = micros();
26      if (accion_m3 <= esc_loop_timer) { // Motor 3 LOW
27        aM3 = false;
28        digitalWrite(10, LOW);
29      }
30      esc_loop_timer = micros();
31      if (accion_m4 <= esc_loop_timer) { // Motor 4 LOW
32        aM4 = false;
33        digitalWrite(11, LOW);
34      }
35    }
36    //dt_PWM = (micros()-loop_timer3)/1000;
37    loop_timer3 = micros(); Serial.print("Pwm_t:");Serial.println(dt_PWM);
38  }
```

En términos simples, la potencia a los motores se modera a través del ciclo de trabajo del motor, es decir, del tiempo en que el pin de su ESC está en modo high, una vez activados los 4 motores esperamos un pequeño lapso de tiempo en microsegundos hasta que el tiempo desde que se activaron al actual sea exactamente la potencia calculada que deseamos para el motor, una vez que superamos este lapso de tiempo apagamos el motor. Realizamos este proceso para cada motor.

Este proceso se hace a una frecuencia definida en el código de nombre UsCiclo, en este caso usamos 10000 microsegundos, por esto el ciclo de encender los motores a una potencia deseada se realiza con frecuencia de 100 Hz.

5.13. Inicialización de sensores

Es necesario inicializar los sensores en el código antes de ser usados, de lo contrario nunca obtendremos datos de ellos, para esto usamos el siguiente código

```
1  void InicializarSensores(){
2  boolean confirmaciones;
3  confirmaciones = false;
4  Wire.begin();
5  TWBR = 12;
6
7  // ==================== MPU6050
8  init_MPU6050();
9  if(confirmaciones==true){Serial.print("Testing Accel/Gyro... ");
10 Serial.println(MPU.testConnection() ? "MPU6050 connection successful" : "MPU6050
       connection failed");}
11 // ==================== HMC5883L
12 /*MAG.initialize();
13 if(confirmaciones==true){Serial.print("Testing Mag...  ");
14 Serial.println(MAG.testConnection() ? "HMC5883L connection successful" : "HMC5883L
       connection failed");}*/
15 // ==================== Ms5611
16 while(!ms5611.begin()){Serial.println("Could not find a valid MS5611 sensor, check
       wiring!");delay(500);}
17 referencePressure = ms5611.readPressure();          // Referencia altitud relativa
18 checkSettings();                                          // Check settings
19 if(confirmaciones==true){Serial.print("Testing Pressure...  ");}
20 //Serial.println(BAR.testConnection() ? "BMP085 connection successful" : "BMP085
       connection failed");}
21 if(confirmaciones==true){Serial.println("Inicializaci n completa");}
22 }
```

5.14. Código Principal

El código principal es fácil de comprender, el esquema de funcionamiento es el siguiente.

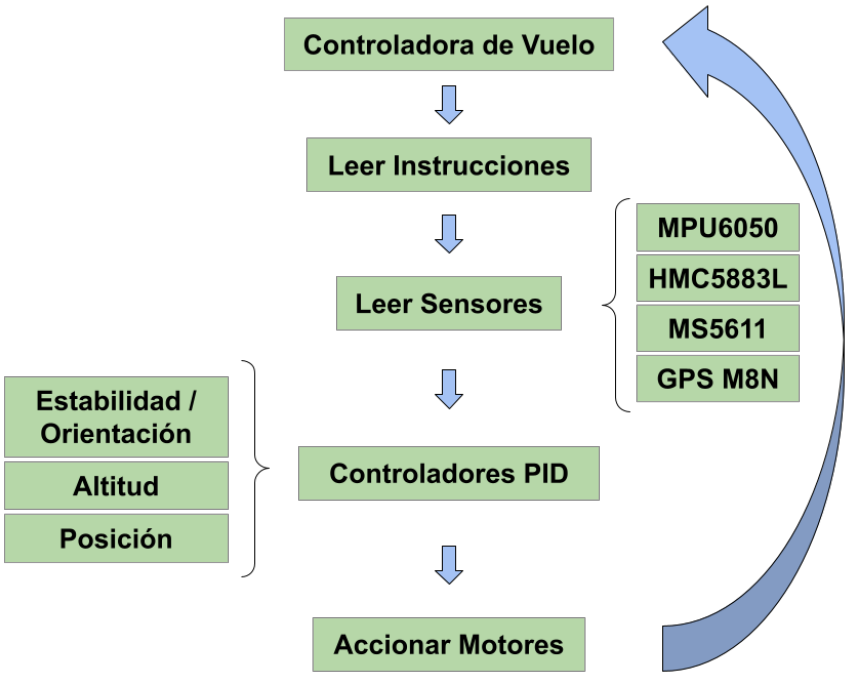


Figura 5.10: Esquema de funcionamiento de loop de código principal (Elaboración propia).

```
#include <I2Cdev.h>
#include <MPU6050.h>
#include <Wire.h>
#include "HMC5883L.h"
#include <Adafruit_BMP085.h>
#include "BMP085.h"
#include <SFE_BMP180.h>
#include <MS5611.h>
#include <TinyGPS++.h>
#include <SoftwareSerial.h>
#define gyro_address 0x68
#define usCiclo 10000  // Ciclo de ejecucion de software en microsegundos

MS5611 ms5611;
HMC5883L MAG;
MPU6050 MPU;
```

```
17 //=====================================================
18 // TIEMPOS
19 long loop_timer, loop_timer1, loop_timer2, loop_timer3, esc_loop_timer;
20 long dt, dt_PID, dt_PWM, ciclo_timer;
21 //SOFTWARE=============================================
22 boolean Despegado, MPUcalibrado, ESCcalibrado;
23 int counter, counter2, opciondevuelo, vel, VelDespegue, VelAterrizaje, Accion;
24 //GPS==================================================
25 float LatitudGPS,LongitudGPS,AltitudGPS;
26 //MOTORES==============================================
27 int16_t PotenciaM1, PotenciaM2, PotenciaM3, PotenciaM4;
28 unsigned long currentMillis, previousMillis;
29 int min_throttle = 1000;
30 int max_throttle = 2000;
31 //MPU6050,//HMC5883l, //Ms6511=========================
32 int16_t aceX,aceY,aceZ,velX,velY,velZ,magX,magY,magZ;          //VALORES RAW
33 double referencePressure;
34 float angP,angR;
35 float Altura;
36 float accel_ang_x, accel_ang_y, accel_ang_z, acc_total_vector;  //VALORES ECALADOS
37 float ang_Pitch, ang_Roll, ang_Yaw, ang_Pitch_ant, ang_Roll_ant, ang_Yaw_ant;
38 float vel_ang_x, vel_ang_y, vel_ang_z, vel_ang_x_ant, vel_ang_y_ant;
39 long f_ax,f_ay, f_az,f_gx,f_gy, f_gz;                          //CALIBRADORES MPU
40 int p_ax, p_ay, p_az, p_gx, p_gy, p_gz, ax_o, ay_o, az_o, gx_o, gy_o, gz_o;
41 //PID==================================================
42 int deseoP_v,deseoR_v,deseoY_v,deseoVel,deseoP_a,deseoR_a,deseoY_a;     //DESEOS
43 float deseoAlt,deseoLon,deseoLat;
44 float pitch_error_a, pitch_error_anterior_a;                          //PITCH_a
45 float Kp_pitch_v,Ki_pitch_v,Kd_pitch_v;
46 float pitch_error_v, pitch_error_anterior_v;                          //PITCH_v
47 float Kp_roll_v,Ki_roll_v,Kd_roll_v,roll_error_v,roll_error_anterior_v; //ROLL
48 float Kp_yaw_v,Ki_yaw_v,Kd_yaw_v,yaw_error_v, yaw_error_anterior_v;     //YAW
49 float Kp_alt,Ki_alt,Kd_alt,alt_error, alt_error_anterior;              //ALTITUD
50 float Kp_lon,Ki_lon,Kd_lon,lon_error, lon_error_anterior;              //LONGITUD
51 float Kp_lat,Ki_lat,Kd_lat,lat_error, lat_error_anterior;;             //LATITUD
52 //=============================================================================
53 void setup(){
54 //==========PAR METROS DE VUELO
55 ParametrosVuelo();
```

```
56  constantesPID();
57  //============INICIALIZACIONES
58  Serial.begin(115200);
59  Serial.setTimeout(4);
60  InicializarSensores();
61  InicializarMotores();
62  delay(500);
63  //=============CALIBRACIONES
64  if(MPUcalibrado==false){PrecalibradoMPU();}
65  else{YaCalibrado();}
66  if(ESCcalibrado==false){CalibracionESC();} //WARNING
67  delay(50);
68  //Serial.println("===RDRONE ONLINE===");
69  ciclo_timer=micros();
70  loop_timer=micros();
71  GirosccopioAcelerometro();
72  }
73  //================================================
74  void loop() {
75  if(vel<deseoVel){vel=vel+5;}        //INCREMENTO SUAVE DE VEL (MULTIPLOS DE 5)
76  else if(vel>deseoVel){vel=vel-5;}
77  //Serial.print("Tciclo[ms]:");
78  //Serial.println(micros()-ciclo_timer);ciclo_timer=micros();    //TIEMPO DE CICLO
79  if(Serial.available() && Serial.available()!=2)
80  {LecturaSerial();}                          //LECTURA DE ORDENES BT/SERIAL
81  Altimetro();                                //CALCULO ALTITUD
82  CampoMagn tico();                            //CALCULO INCLINACION YAW
83  GPS();                                      //CALCULO POSICION
84  Angulos();                                  //CALCULO DE ANG
85  PID_altura();                                //PID ALTURA
86  PID_posicion();                              //PID POSICION
87  PID_av();                                   //CALCULOS PID
88    while (micros() - loop_timer < usCiclo);  //COMIENZO NUEVO CICLO
89    //Serial.println(micros() - loop_timer);
90    loop_timer = micros();                    //INSTANTE COMIENZO DE CICLO
91    Vuelo();                                  //ACCIONA MOTORES
92  while (micros() - loop_timer < 2200);       //LECTURA EXACTA DE MPU6050
93  GirosccopioAcelerometro();                  //LECTURA MPU6050
94  }
```

6. Pruebas de Funcionamiento

Se debe asegurar que los módulos estén conectados correctamente, funcionen y proporcionen datos correctos individualmente para después concentrar todos los datos en un código, para esto se deben hacer pruebas y comparar los datos con métodos tradicionales o con base en otras tecnologías.

En este capitulo se encuentran resultados de las pruebas realizadas a los componentes del dron y los códigos desarrollados.

6.1. Ensamblaje Mecánico

El ensamblaje y armado de las piezas estructurales y mecánicas del dron se realizó en el siguiente orden.

1. Ubicar y reconocer cada parte del marco F330.
2. Soldar a la plataforma inferior del marco los cables de corriente que irán conectados a la batería.
3. Soldar cada ESC a los polos (+) y (-) de la plataforma inferior del marco y fijarlos a su brazo correspondiente.
4. Atornillar los 4 brazos del dron a la plataforma inferior del marco.
5. Colocar la plataforma superior del marco del dron y atornillar los brazos.
6. Colocar las almoadillas antivibración en cada brazo del dron.
7. Armar cada motor.
8. Colocar cada motor en cada brazo del marco.
9. Fijar sobre la plataforma superior del marco la plataforma antivibraciones.
10. Colocar y fijar el cerebro del dron a la plataforma superior del dron.
11. Conectar cada motor a su ESC teniendo en cuenta el sentido de giro correcto.
12. Conectar cada ESC al cerebro del dron.
13. Colocar y fijar la batería en la superficie inferior del marco.
14. Balancear las hélices.
15. Colocar y fijar las hélices a cada motor teniendo en cuenta el sentido de giro correcto.

Una vez armado el dron, se procede a la programación de este. El proceso de ensamblaje del dron se muestra en el siguiente grupo de imágenes.

Figura 6.1: Proceso de ensamblaje mecánico del dron.

La colocación de los motores y las direcciones pitch y roll del dron son la siguientes:

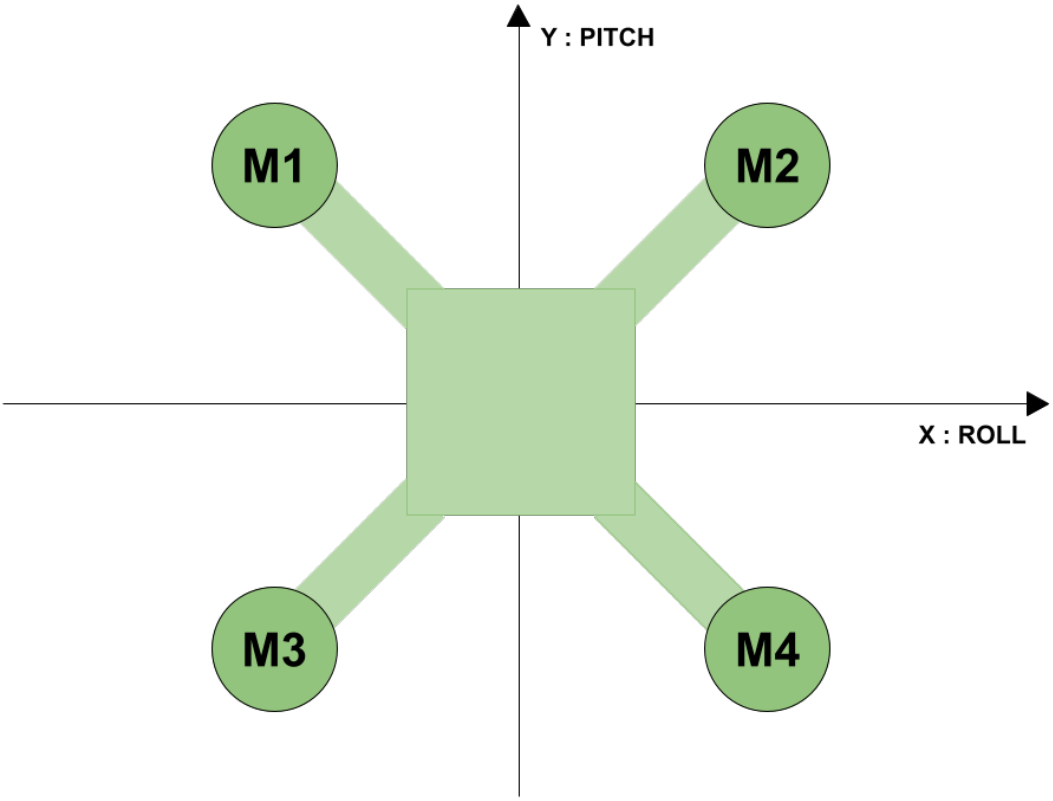


Figura 6.2: Orientación pitch, roll y acomodo de motores en el dron.

Es importante considerar el orden y la posición en que se acomodan los motores y el sentido positivo del giro respecto a pitch y roll ya que en los cálculos realizados por el controlador PID de estabilidad intervienen sumas o restas dependiendo de estos.

6.2. Diagrama de Conexiones

Usando el cautín y soldadura, se realizaron las conexiones electrónicas entre el arduino nano, los módulos y los ESC´s. El diagrama de conexión es el siguiente.

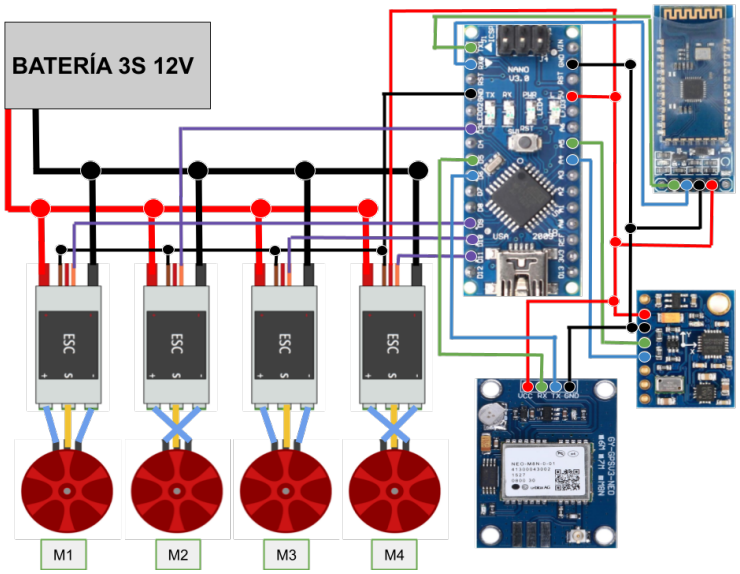


Figura 6.3: Conexiones electrónicas del dron.

Los componentes mostadros en la imagen anterior son suficientes y necesarios para garantizar el vuelo autónomo en áreas abiertas del dron, si se desea tener vuelo autónomo en presencia de obstáculos como pueden ser edificios es necesario incluir sensores ultrasónicos para identificar los obstáculos y tomar acciones.

6.3. Cerebro Intercambiable

En la construcción y conexión del cerebro del dron se consideran posibles fallas o rupturas de cada módulo, es por ello que se construye de modo que cada módulo incluyendo el Arduino Nano sean reemplazables.

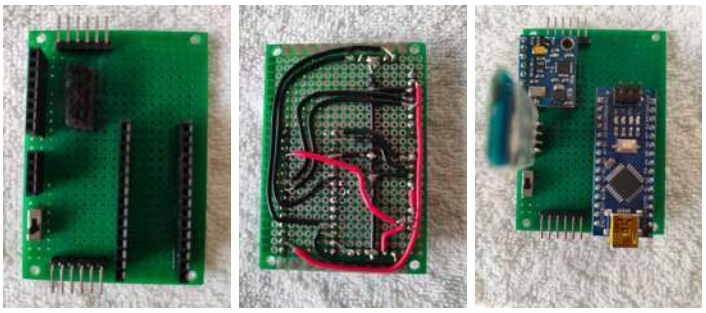


Figura 6.4: Cerebro con partes intercambiables.

6.4. Balance de Hélices

El balanceo de hélices es importante, si las hélices no se encuentran balanceadas estas pueden producir vibraciones que se disipan en el dron provocando mediciones erroneas del sensor MPU6050. Para realizar el balanceo de hélices usaremos un balanceador de hélices como el que se muestra a continuación.

Figura 6.5: Balanceador de hélices.

Debemos colocar la hélice en el eje giratorio, una hélice equilibrada deberá mantenerse totalmente horizontal, si alguno de los datos cae, significa que debemos compensar esta pequeña diferencia de peso, existen dos formas de hacerlo

- Lijar la pala más pesada de la hélice hasta compensar la diferencia de peso.
- Compensar la diferencia de peso agregando pedazos pequeños de cinta aislante a la pala más ligera.

Usaremos el segundo método. Este proceso debe de realizarse con cada hélice.

Figura 6.6: Izquierda: Hélice no balanceada. **Derecha:** Hélice balanceada usando cinta aislante.

6.5. MPU6050

Este sensor es el más importante del dron y del sistema de estabilidad en vuelo, antes de hacer mediciones de ángulos de inclinación debemos tener nuestro MPU6050 calibrado, para esto colocamos el dron estático en una superficie totalmente nivelada, usando el código de calibración obtenemos los OFFSETS de nuestro MPU6050:

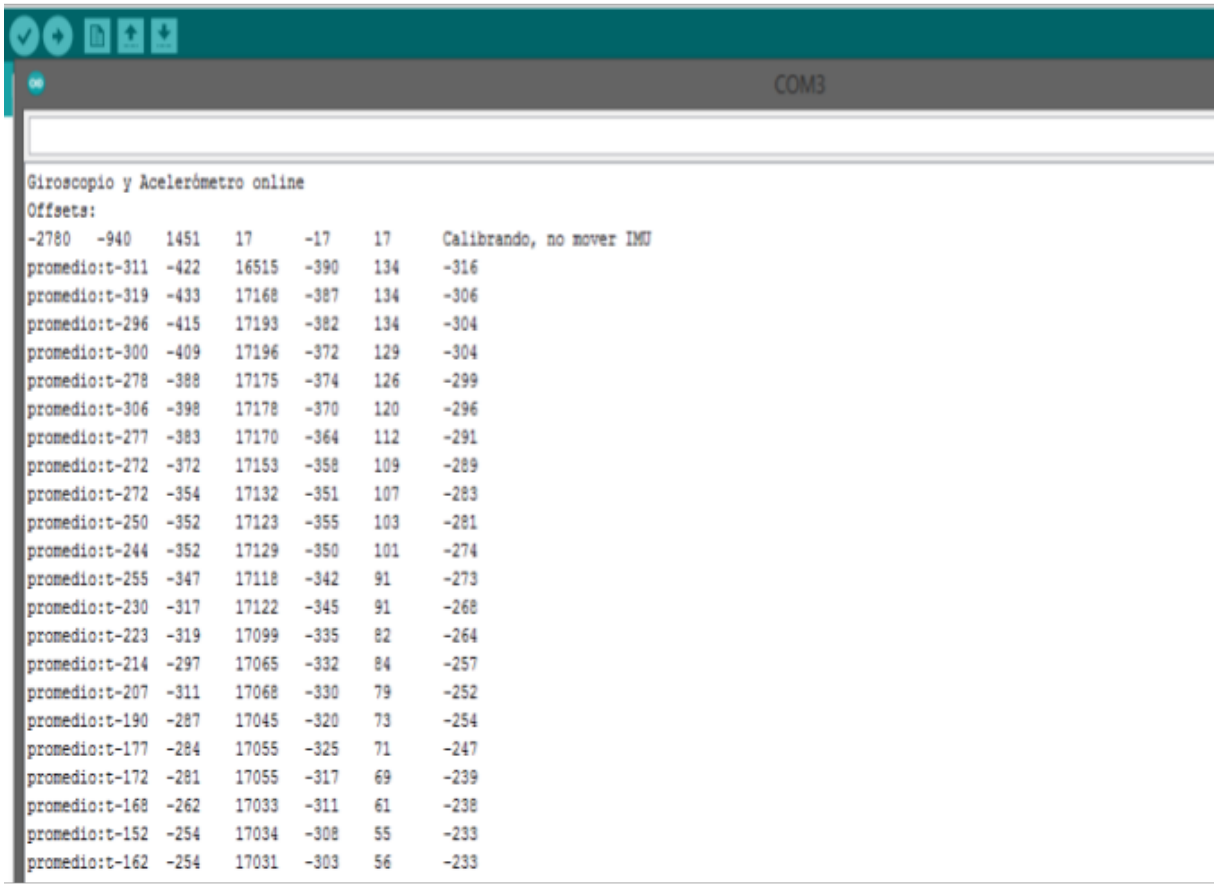


Figura 6.7: Valores de los 6 OFFSETS para el MPU6050 obtenidos en algoritmo de calibración.

Una vez obtenidos los OFFSETS los implementamos en el código. No será necesario volver a calibrar el sensor. Con el sensor calibrado estamos listos para realizar mediciones.

La medición de los ángulos de inclinación en pitch y roll arrojan los siguientes datos:

```
Pitch Roll
 0.01 0.03
Pitch Roll
 0.01 0.03
Pitch Roll
 0.01 0.04
Pitch Roll
 0.01 0.04
Pitch Roll
 0.01 0.04
Pitch Roll
 0.00 0.04
Pitch Roll
 0.00 0.03
Pitch Roll
 0.00 0.03
Pitch Roll
 0.00 0.03
Pitch Roll
 0.01 0.03
Pitch Roll
 0.01 0.03
Pitch Roll
 0.01 0.03
Pitch Roll
 0.01 0.03
Pitch Roll
 0.01 0.03
Pitch Roll
 0.01 0.03
```

```
Pitch Roll
 -0.43 45.12
Pitch Roll
 -0.44 45.12
Pitch Roll
 -0.44 45.12
Pitch Roll
 -0.44 45.12
Pitch Roll
 -0.44 45.12
Pitch Roll
 -0.44 45.12
Pitch Roll
 -0.43 45.12
Pitch Roll
 -0.43 45.12
Pitch Roll
 -0.43 45.12
Pitch Roll
 -0.43 45.12
Pitch Roll
 -0.44 45.12
Pitch Roll
 -0.44 45.12
Pitch Roll
 -0.44 45.12
Pitch Roll
 -0.44 45.12
Pitch Roll
 -0.43 45.12
Pitch Roll
 -0.43 45.12
```

Figura 6.8: Ángulos de inclinación calculados por software comparados con una medición simple mediante transportador.

6.6. MS5611

Para obtener la altitud relativa al punto de despegue realizamos mediciones de la presión y la comparamos con la presión tomada en a la altura del despegue, cada ciclo del código principal realizamos una medición, después de 50 mediciones calculamos el promedio de estas y así eliminamos ruido de las mediciones.

Como prueba realizamos mediciones sin despegar el dron para observar la variación de los datos, estos son los datos obtenidos:

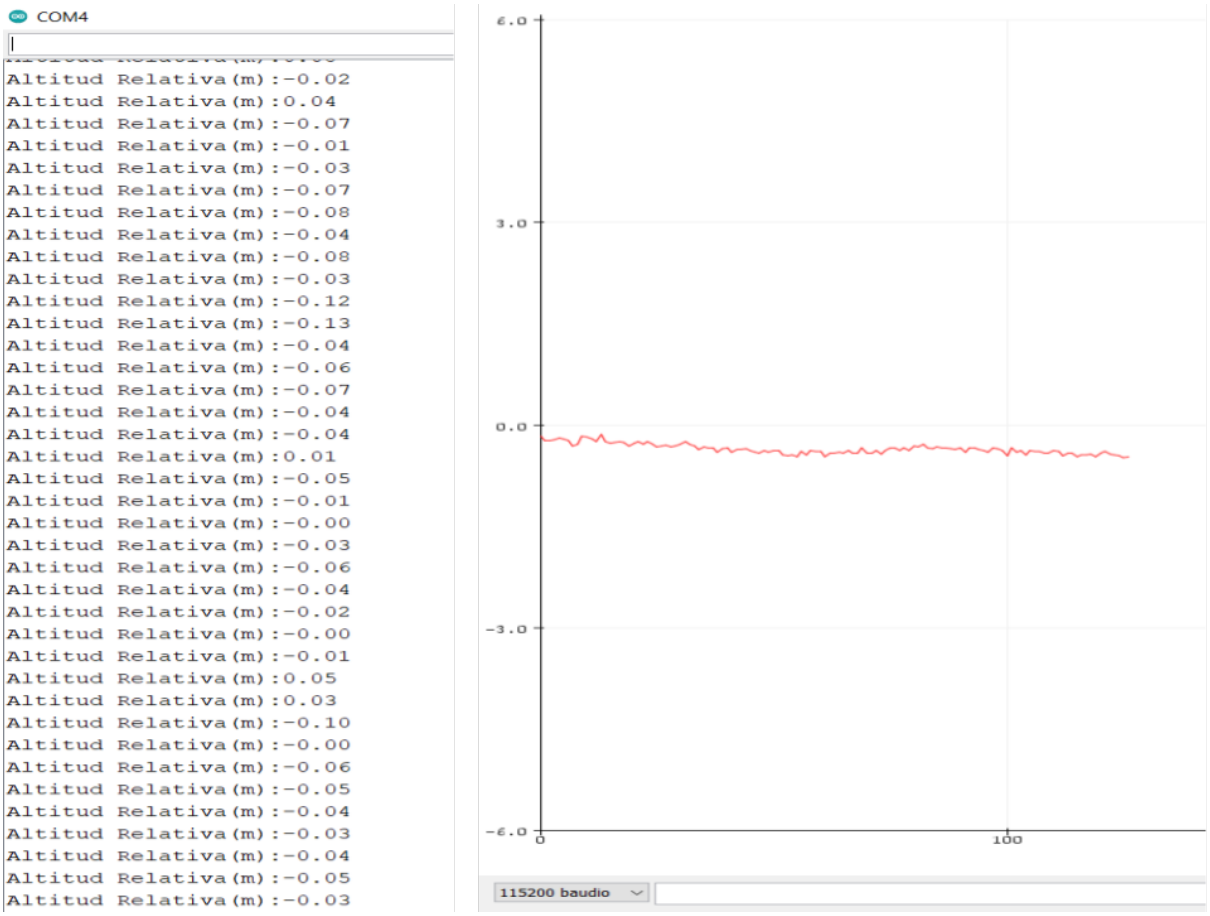


Figura 6.9: Altitud relativa del dron obtenida con el sensor MS5611 sin despegue del dron.

La precisión es de aproximadamente 20 cm como menciona el fabricante del sensor. Recordemos que un sensor de presión es muy sensible y puede ser afectado por ráfagas de aire e incluso por los rayos del sol por lo que debe estar protegido.

6.7. HMC5883L

En las pruebas de medición de la brújula digital HMC5883L y los cálculos para obtener el ángulo de inclinación en yaw respecto al norte geográfico de la tierra obtenemos los siguientes datos

```
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -61
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -61
AnguloX-N: -62
AnguloX-N: -61
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -61
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
AnguloX-N: -62
```

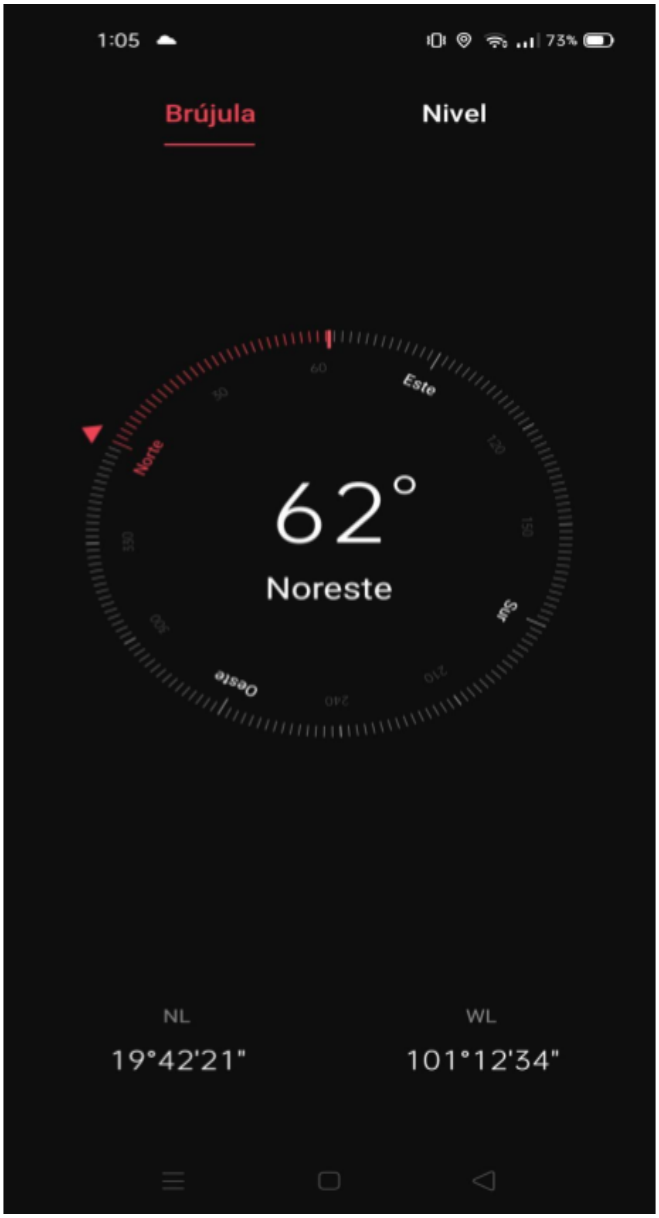


Figura 6.10: Ángulo de inclinación Yaw respecto al norte geográfico comparado con una brújula de smartphone.

Los datos obtenidos por la brújula digital HMC5883L son precisos comparándolos con la brújula digital de un smatphone, estas mediciones se verán afectadas si el sensor está en presencia de algun campo magnético externo como un iman, es por ello que es importante mantener este sensor lejos de los imanes de los motores.

6.8. Bluetooth HC-06

Probaremos el envío y recepción de datos del dron mediante bluetooth, para esto usaremos la aplicación mencionada en la sección de software. Probaremos recibiendo en el smartphone los valores actuales de la inclinación del dron respecto a pitch y Yaw, recibimos los siguientes datos en el smartphone

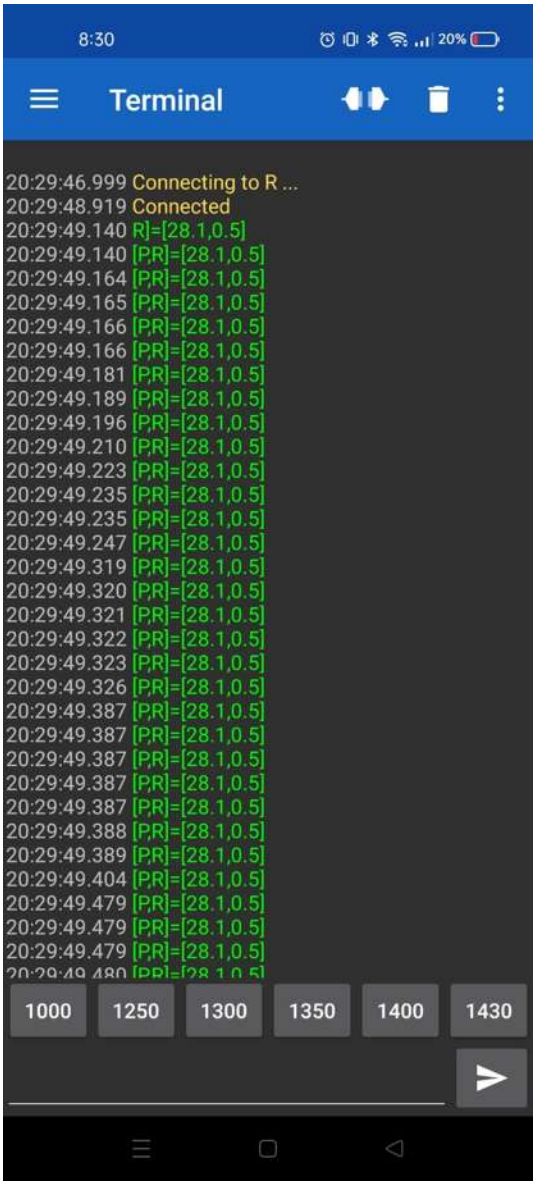


Figura 6.11: Datos enviados por el dron a smatphone mediante comunicación bluetooth de ángulos de inclinación calculados y filtrados por software.

Una vez comprobada la comunicación bluetooth podemos mandar órdenes al dron en vuelo y recibir algunos datos de él que sean de nuestro interés. El módulo HC-06 tiene un alcance de 10m y se usa para pruebas, una vez conseguida la autonomía del dron se optará por otro tipo de comunicación de mayor alcance.

6.9. GPS

El módulo GPS nos proporciona varios datos, por ejemplo longitud, latitud, fecha, altitud, fecha, etc. Sin embargo para nuestra aplicación solo usaremos longitud, latitud y altitud para ubicar al dron geográficamente. En la prueba del sensor y filtrado de datos obtenemos los siguientes:

LAT=	LON=	ALT=
19.705972	-101.209541	1887.60
LAT=	LON=	ALT=
19.705963	-101.209541	1887.60
LAT=	LON=	ALT=
19.705961	-101.209548	1887.60
LAT=	LON=	ALT=
19.705959	-101.209541	1887.60
LAT=	LON=	ALT=
19.705959	-101.209548	1887.60
LAT=	LON=	ALT=
19.705961	-101.209548	1887.60
LAT=	LON=	ALT=
19.705963	-101.209548	1887.60
LAT=	LON=	ALT=
19.705955	-101.209548	1887.60
LAT=	LON=	ALT=
19.705951	-101.209548	1887.60
LAT=	LON=	ALT=
19.705947	-101.209548	1887.60
LAT=	LON=	ALT=
19.705947	-101.209556	1887.60
LAT=	LON=	ALT=
19.705944	-101.209556	1887.60
LAT=	LON=	ALT=
19.705942	-101.209556	1887.60
LAT=	LON=	ALT=
19.705940	-101.209556	1887.60
LAT=	LON=	ALT=
19.705936	-101.209564	1887.60
LAT=	LON=	ALT=
19.705934	-101.209564	1887.60
LAT=	LON=	ALT=
19.705932	-101.209564	1887.60
LAT=	LON=	ALT=
19.705930	-101.209564	1887.60
LAT=	LON=	ALT=
19.705923	-101.209564	1887.60

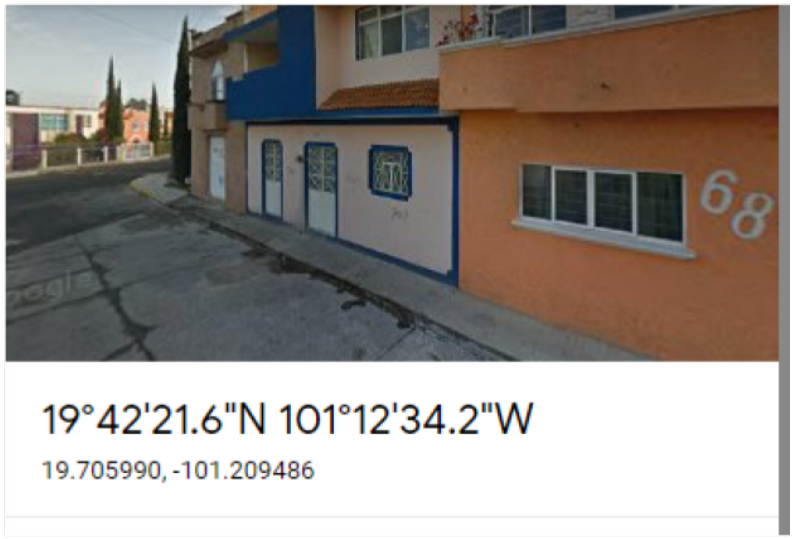


Figura 6.12: Coordenadas de longitud y latitud proporcionadas por el GPS Neo M8N comparado con los datos proporcionados por la aplicación de Google Maps.

Los datos obtenidos los comparamos con longitud y latitud proporcionada por google maps de nuestra ubicación. Los valores obtenidos por el GPS son precisos. Es importante saber que en exterior la precisión del GPS mejora notablemente.

Los datos del GPS serán usados por el controlador PID de posición para hacer al dron mantener una posición, también para realizar vuelos de un punto a otro con solo proporcionar las coordenadas de longitud y latitud del lugar donde se desea llegar.

6.10. Control de Vuelo

Es importante realizar pruebas de controlde vuelo y estabilidad en un ambiente controlado antes de volar al aire libre, para ello el primer paso y posiblemente el más importante es el control de estabilidad, el correcto funcionamiento de este nos asegurará que el dron no caiga, pues una caída sería devastadora para todos los componentes electrónicos y mecánicos del dron.

6.10.1. Plataforma de Pruebas de Estabilidad

Se adecuó una plataforma de pruebas basada en la modificación de un mueble de cómputo casero. Su funcionamiento es similar a un péndulo invertido, se permite solo el movimiento angular del dron respecto a un eje para probar el control de estabilidad en dicha eje, una vez superadas las pruebas en ese eje se cambia al ortogonal. La plataforma cuenta con barras de aluminio a los lados para evitar fisicamente la sobreinclinación del dron, en el centro cuenta con una varilla redonda de aluminio que es donde se fija el dron, esta puede rotar, dicha varilla está fija a la estructura. En las barras de aluminio se colocan pedazos de hule para evitar golpes fuertes.

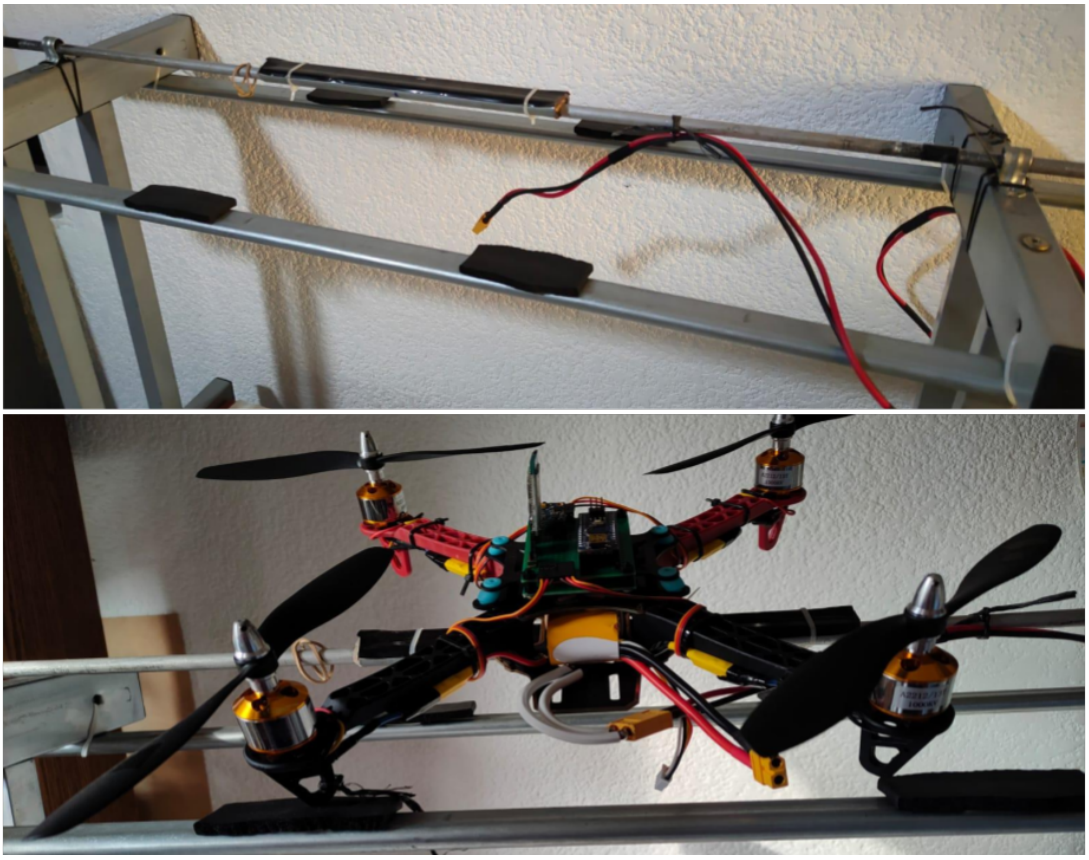


Figura 6.13: Plataforma de pruebas de estabilidad.

6.10.2. Fuente de Voltaje

En las pruebas realizadas en la plataforma de pruebas es indispensable contar con una fuente de poder para los motores, esto se hace para sustitur el uso de una batería, la batería nos puede permitir 10 minutos de vuelo a cambio de horas de carga, esto reduciría enormemente la cantidad de pruebas que podemos realizar al día, reduciría la vida útil de la batería y además deberíamos compensar por software la pérdida de voltaje de la batería a medida que se descarga.

La fuente de voltaje que usaremos cuenta con las siguientes especificaciones detalladas por el fabricante:

- Voltaje de entrada 110V 220V.
- Voltaje de salida: 12 V regulables.
- Amperaje: 30 A.
- Cuenta con un ventilador interno para evitar sobrecalentamiento.
- Carcasa de aluminio.
- Medidas: 21.5CM X 11.5CM X 5CM.
- Potencia: 360W.

Figura 6.14: Fuente de poder 12V 30A.

La conexión es simplemente sustituir los cables (+) y (-) de la batería por los de la fuente de poder. Esta proporcionará un voltaje constante de 12V al dron y permitirá hacer pruebas continuamente.

6.10.3. Control PID de Estabilidad

Se realizaron pruebas sobre la plataforma de pruebas con un ámplio espectro de valores para las constantes proporcional, integral y derivativo, variando el parámetro α de los filtros complementario y EMA, se hicieron modificaciones al control PID: basado en el ángulo de inclinación y en la velocidad angular.

Finalmente se implementó al controlador la predicción del error en un tiempo a futuro de δ milisegundos en el futuro mediante interpolación de Lagrange de segundo orden.

Todas las pruebas abarcan más de 4000 experimentos. No se obtuvo un comportamiento de estabilización del dron que fuera seguro y satisfactorio, El dron no tiene una precisión y estabilidad deseable para potencias mayores al 30 % de los motores haciéndolo incapaz de despegar seguro.

Se sospechan que alguna o la combinación de algunas de las siguientes causas sea la razón

- El Arduino Nano es incapaz de controlar con presición los ESC debido a ser de 8 bits.
- Los motores tienen un retraso mecánico no despreciable causando resonancia.
- El controlador PID simple no es suficiente para garantizar la estabilidad.
- Los filtros complementario y EMA no son confiables para sistemas expuestos a vibraciones.

6.10.4. Control PID de Altitud

Es necesario superar antes las pruebas de estabilidad.

6.10.5. Control PID de Posición

Es necesario superar antes las pruebas de estabilidad.

7. Mejoras

Sin limitaciones de presupuesto y más tiempo el proyecto puede mejorar, las propuestas son las siguientes.

7.1. Mecánicas

- Añadir una PCB estructural.
- Añadir una batería estructural.
- Marco de una sola pieza impreso en 3D en fibra de carbono.
- Usar hélices de fibra de carbono.
- Comparar diversos motores en el mercado.

7.2. Electrónicas

- Integrar todo el circuito electrónico con modulos, sensores y chip en una PCB.
- Sustituir el chip Arduino Nano por uno más potente: STM-32 por ejemplo.
- Divir el dron en dos chips, uno encargado de las mediciones de sensores y otro solamente solo de algoritmos de control de vuelo.

7.3. Programación

- Cambiar filtro comlementario por filtro de Kalman.
- Probar PID concatenados.
- Probar diferentes algoritmos de control.
- Probar nuevas tecnologías como redes neuronales.
- Programar una app que proporcione todos los datos en tiempo real del dron.

7.4. Generales

- Construir plataformas de pruebas.
- Construir un campo de pruebas.

8. Costos

Se proporcionan los costes aproximados bajo modelo de compra minorista, las compras fueron realizadas en el año 2021. La mayoría de los componentes se compraron por **MercadoLibre** debido a su baja disponibilidad en la ciudad (Morelia, Michoacán), los costes no incluyen gastos de envío y están sujetos a cambios por el fabricante.

Algunos componentes no disponibles en México se consiguieron mediante la tienda online **AliExpress**, los precios son bajos incluso a una diferencia del 50 % respecto a **MercadoLibre**, sin embargo la espera de los productos comprados en **AliExpress** puede variar entre semanas y meses.

Los costos se calculan en pesos mexicanos y son solo de caracter informativo para interesados en este tipo de proyectos.

Los precios son de los productos especificados en la sección de componentes del dron.

Item	Precio unitario	Unidades	Costo Total (MXN)
Marco F330	500	1	500
Motor A2212/13T	200	4	800
Esc 30A	200	4	800
Hélice 8045	40	4	160
Batería	1000	1	1000
Plataforma Antivibración	1	160	160
Almoadillas Antivibración	25	4	100
Arduino Nano	160	1	160
GY-86	350	1	350
HC-06	100	1	100
GPS	160	1	160
		Total:	4290

Tabla 8.1: Costos de componentes del dron

Los costos anteriores son solamente de los componentes del dron para su construcción y en un escenario ideal (Sin estropear o romper los componentes).

Para estimar los costos totales se deben considerar el riesgo que las pruebas conllevan y las herramientas o instrumentos necesarias para llevarlas a cabo, por ejemplo, fuentes de poder, multímetro, soldadura, cautín, pinzas, cargador de batería, cable, tornillos, desarmadores,llaves allen, cinchos, cinta aislante, por mencionar algunas.

El componente más fácil de estropear son las hélices, en segundo lugar los brazos del marco.

9. Trabajo a Futuro

Como trabajo a futuro, se proponen lo siguiente:

- Usar el filtro de Kalman para la estimación de los datos de vuelo como ángulos de inclinación, fuerzas g y velocidades lineales del dron.
- Usar una tarjeta de desarrollo STM32 F4, esto nos permitirá ampliar las capacidades del dron y con esto mejorar la presición del vuelo.
- Se probarán, mejorarán o desarrollarán algoritmos basados en inteligencia artificial para potenciar las capacidades del dron.
- Una vez conseguido que el dron sea estable, mantenga una altitud y posición, nos enfocaremos en el vuelo autónomo del dron, es decir, será capaz de detectar y evadir obstáculos que le impidan viajar de un punto A a un B, el despegue y aterrizaje será autónomo mediante reconocimiento de zonas de aterrizaje definidas. Una vez superadas estas pruebas, desarrollaremos la detección y evasión de obstáculos en movimiento.
- Se probará el vuelo y funcionamiento del dron en condiciones artificiales de atmósfera rarificada simulando las condiciones en otros planetas o zonas poco accesibles de la Tierra.
- A lo largo de la ejecución de los procesos mencionados anteriormente, se llevará a cabo a la par la optimización del dron, optimización de costos, reducción de tamaño, peso, eficiencia y duración de la batería, etc.

Lo mencionado anteriormente se espera se desarrolle en un lapso de cuatro años, esto sujeto a cambios dependientes en la disponibilidad de chips y la superación exitosa de las pruebas a realizar.

10. Conclusiones

La elaboración de un proyecto de ingeniería como este es de esperarse que con el progreso aparezcan problemas y obstáculos que no se esperan y se deben afrontar. Los componentes electrónicos tienen detrás todo un arte de uso, desde su conexión, su configuración, calibración y el código con el que se usan, en un proyecto como este existen muchas variables relacionadas entre sí que dictarán el resultado final.

La autonomía de un dron es algo alcanzable, sin embargo, como en este proyecto se presentó el control PID para la estabilidad de un dron basado en la inclinación de este no es suficiente. Se realizaron pruebas de estabilidad no exitosas sobre una superficie de pruebas de construcción propia y usando una fuente de voltaje como reemplazo de la batería, las pruebas se realizaron con un amplio espectro de los parámetros proporcional, integral y derivativo del controlador PID de ángulos, se modificó el controlador PID de ángulos por uno basado en minimizar la velocidad angular experimentada por el dron, igualmente sin resultados satisfactorios. Se realizaron pruebas variando el tiempo de ciclo del código principal desde los 100 milisegundos hasta los 10, que es el tiempo mínimo requerido por el chip Arduino Nano. Se probaron distintas implementaciones en los códigos siempre optimizando el tiempo y los cálculos requeridos. Se comparó el funcionamiento de los motores usando la librería **Servo** y código crudo, Se reemplazaron módulos, sensores y componentes mecánicos para minimizar el peso del dron. Se probaron distintas configuraciones de los sensores, Arduino y el código principal. Se calcularon los tiempos requeridos de cada sensor para realizar mediciones.

Los resultados ocultan en sí gran cantidad de interrogantes, desde cuestionarse si las capacidades de los motores, ESC, sensores y chip Arduino son suficientes para un proyecto de este tipo. Arduino está orientado a trabajos escolares, pasatiempos, incluso diversión, sin embargo, para proyectos donde se requieren mayores requerimientos como velocidad del chip, memoria, realizar tareas de presición y reducción de tiempos de ejecución se queda corto, esto es mejorable cambiando de placa de desarrollo por una más potente, STM32 por ejemplo.

“Una persona que nunca cometió un error, nunca intentó nada nuevo”.

— **Albert Einstein.**

Referencias

[1] Omar Estalin Alarcon Balseca. *Controlador Electrónico de Velocidad para Cuatro Motores sin Escobillas de un Drom.* Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, 2018.

[2] O. E. Alarcón Balseca. Controlador Electrónico de Velocidad para Cuatro Motores sin Escobillas de un Dron.

[3] ArcGeek. ¿Cómo funcionan los dispositivos GPS? Trilateración vs Triangulación. https://acolita.com/como-funcionan-los-dispositivos-gps-trilateracion-vs-triangulacion/, Mayo 2018.

[4] L. F. Basarte Bozal. Diseño y Fabricación de una Tarjeta PCB ("Shield") para el Control de un Robot Autobalanceado Basado en Arduino, 6 2014.

[5] Blogging. 10 mejores cuadricópteros de este año. https://blogging-techies.com/10-mejores-cuadricopteros-de-este-ano/.

[6] Joop Brokking. MPU-6050 6dof IMU tutorialfor auto-leveling quadcopters with Arduino source code. https://www.youtube.com/watch?v=4BoIE8YQwM8, 2016.

[7] J. A. Capdevila. El Sistema de Posicionamiento Global G.P.S. https://ansenuza.unc.edu.ar/comunidades/bitstream/handle/11086.1/1258/El%20Sistema%20de%20Posicionamiento%2

[8] TE Connectivity. *MS5611-01BA03 Barometric Pressure Sensor, with stainless steel cap.* TE Connectivity, Junio 2017.

[9] A. Cortés Sáez. Control de Posicionamiento de un Cuadricóptero, 2015.

[10] CREBOTS. Almohadilla Anti Vibración, Descripción de producto. https://articulo.mercadolibre.com.mx/MLM-776037292-almohadilla-anti-vibracion-para-motor-22xx-drone-f450-qav220-$_JM$.

[11] C. Cuerno-Rejado, L. García-Hernández, A. Sánchez-Carmona, Sánchez-López J., and P. Campoy-Cervera. Evolución Histórica de los Vehículos Aéreos no Tripulados Hasta la Actualidad. *DYNA*, 91:282–288, Mayo 2016.

[12] Optimus Digital. 30A BLDC ESC, Manual. https://www.optimusdigital.ro/index.php?controller=attachmentid_attachment=4 2012.

[13] DJI. *FlameWheel 450, User Manual.* DJI, Mayo 2015.

[14] J. Díaz. Interpolación, Apuntes c++. https://www.uv.es/diazj/cn$_tema5.pdf$.

[15] Mouser Electronics. Neo-m8n, Especificaciones. https://www.mouser.mx/ProductDetail/u-blox/NEO-M8N-0?qs=zW32dvEIR3unZhZI0KRbew%3D%3D.

[16] J. Etxeberria Mendez. Implementación de un Dron Cuadricóptero con Arduino, 6 2015.

[17] Aula Facil. Conexión ESC a los motores. https://www.aulafacil.com/cursos/maquetas/como-hacer-un-dron-con-pasos-muy-sencillos/conexion-esc-a-los-motores-l40166. Consultado en: 2020.

[18] J. Guerra Carmenate. Señal PWM con Arduinoy AnalogWrite, . Consultado en Diciembre 2021, en https://programarfacil.com/blog/arduino-blog/pwm-con-arduino-analogico/.

[19] Honeywell. *3-Axis Digital Compass IC HMC5883L*. Honeywell, Febrero 2013.

[20] InvenSense Inc. *MPU-6000 and MPU-6050 Product Specification*. InvenSense Inc., Agosto 2013.

[21] Leboriz. Giroscopio. https://www.leboriz.com/producto/giroscopio/.

[22] L. Llamas. *Filtro Paso Bajo y Paso Alto Exponencial (EMA) en Ardino*. Ingeniería, Informática y Diseño, Marzo 2017.

[23] Luis Llamas. Cómo Usar un Acelerómetro en Nuestros Proyectos de Arduino. Septiembre 2016.

[24] Naylamp Mechatronics. HC-06, Descripción de producto. https://naylampmechatronics.com/inalambrico/24-modulo-bluetooth-hc06.html.

[25] Naylamp Mechatronics. Hc-sr04, Descripción de producto. https://naylampmechatronics.com/sensores-proximidad/10-sensor-ultrasonido-hc-sr04.html.

[26] J. Morales Rodríguez. Control de Velocidad de Motores Brushless Mediante Modulación PWM, 6 2018.

[27] Khairuddin Osman N. M. Salma. Modelling and PID control system integration for quadcopter DJIF450 attitude stabilization. *Indonesian Journal of Electrical Engineering and Computer Science*, Septiembre 2020.

[28] Zayas J. Peñafiel, J. *Fundamentos del Sistema GPS y Aplicaciones en la Topología*. Colegio Oficial de Ingenieros Técnicos en Topografia Delegación Territorial de Madrid-Castilla-La Mancha, Septiembre 2001.

[29] RCDRONE. Plataforma Antivibración, Descripción de producto. https://www.rcdrone.cl/producto/plataforma-antivibracion-p/.

[30] Fabian Reuter. *Los drones y sus aplicaciones a la ingeniería*. Universidad Nacional de Santiago del Estero, 2019.

[31] Rhydolabz. A2212/13T TECHNICAL DATA. https://www.rhydolabz.com/documents/26/BLDC$_A$2212$_1$3T.*pdf*.

[32] P. Ruipérez Martín. Diseño y Fabricación de un Dron, 10 2014.

[33] Arduino Store. https://store-usa.arduino.cc/collections/most-popular.

[34] TECNEU. Controlador de Velocidad ESC 30a Electrónico Multi Axis Dron, Descripción del Producto. https://articulo.mercadolibre.com.mx/MLM-759175054-controlador-de-velocidad-esc-30a-electronico-multi-axis-dron-_JM?matt_tool=18816949matt_word=gclid=Cj0KCQiArt6PBhCoARIsAMF5wahh-XhJmJfSQEEQ9SOQs0xlos-eNRkN3D6lNWkVTA4xlyfQ1bxwXQsaAgcUEALw_wcB. Consultado en: 2020.

[35] TED. Raffaello d'andrea: Raffaello d'andrea: La asombrosa potencia atlética de los cuadricópteros. https://www.youtube.com/watch?v=w2itwFJCgFQ, 2013.

[36] TURNIGY. Batería Lipo Zippy 2200mah 3s 11.1v 35c Dron Robótica Xt60, Descripción del producto. https://articulo.mercadolibre.com.mx/MLM-829134401-bateria-lipo-zippy-2200mah-3s-111v-35c-dron-robotica-xt60-$_JM?matt_tool = 99486937matt_word = \&matt_source = googlematt_campaign_id = 15693562203matt_ad_group_id = 132817878898matt_match_type = matt_network = gmatt_device = cmatt_creative = 571796003518matt_keyword = matt_ad_position = matt_ad_type = plamatt_merchant_id = 110614582matt_product_id = MLM829134401matt_product_partition_id = 1418803903092matt_target_id = pla − 1418803903092gclid = Cj0KCQiA0eOPBhCGARIsAFIwTs641Jdu − ic3DQcbGikQj6Nhj5i4FM46_ − imwbQBcXLZIpV6442CDewaAuEXEALw_wcB.Consultadoen : 2020.

[37] u blox. *Neo - 8M.* u-blox, Octubre 2021.

[38] J. Zapata Blandón. Diseño de un Drone para Carga Util de 0.5kg, 2016.

[39] Beltrán Álvarez Carlos. *Métodos Numéricos.* Universidad de Cantabria, 2019.