ESCUELA SUPERIOR DE INGENIERIA MECANICA Y ELECTRICA

PRESENTA:

“CONTROL DE MOVIMIENTO DE UN CUADRICOPTERO MEDIANTE UN CONTROLADOR ARDUINO”

QUE PARA OBTENER EL TITULO DE INGENIERO EN CONTROL Y AUTOMATIZACION

Presenta:

Álvarez Salgado Brian

Rodríguez Lira Carlos Ricardo

Asesores:

M. en C. Darío Betanzos Ramírez

M. en C. Pedro Francisco Huerta Gonzales

México D.F. Mayo 2015

RESUMEN

Este trabajo da a conocer todos los pasos para controlar un sistema PID aplicado a un cuadricóptero que será controlado remotamente por un dispositivo Android.

El Capítulo 1 consta de una introducción, justificación y objetivos tanto generales como específicos. Dichos elementos se encargan de proporcionar información de carácter válido que sustenta el trabajo de tesis.

Como acercamiento a los conceptos que se tomarán como base para la creación del drone o cuadricóptero, se cuenta con un marco teórico en el Capítulo 2. Los temas a tratar en este apartado son los sistemas de comunicación, sistemas electrónicos y sistemas mecánicos.

El desarrollo del hardware consta de la integración de los elementos que se usarán como el Frame, la placa Arduino y los motores; también se hablará de los variadores de velocidad, las baterías y los módulos de comunicación y adquisición de datos por medio de un acelerómetro digital, comprendiendo así el Capítulo 3.

El Capítulo 4 lo comprende el desarrollo del software. Aquí se dará un acercamiento al desarrollo del programa final aplicado a la placa Arduino y un dispositivo Android, pasando desde la programación del Bluetooth, hasta el desarrollo y comunicación de una aplicación para Android con lenguaje Java.

En el Capítulo 5 se muestra los resultados obtenidos de la relación del desarrollo del hardware y el desarrollo del software, dando así una explicación breve pero detallada acerca de cómo funciono el dron terminado con su aplicación de comunicación.

AGRADECIMIENTOS

Por su apoyo y constante estímulo para continuar en el camino de la formación académica. El presente proyecto no tendría fin gracias a ustedes pues gracias a sus constantes palabras de aliento, palabras que día con día hacen ver que no hay grandes problemas, es uno mismo quien se hace pequeño ante ellos.

Gracias a todos aquellos que demostraron simpatía y optimismo pero especialmente se agradece a todas aquellas personas que de una u otra forma intentaron depreciar este trabajo pues fueron ellos quienes realmente nos enseñaron que íbamos por el camino correcto.

“Ser realista es el camino más rápido hacia la derrota”

Will Smith

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPITULO 1: INTRODUCCIÓN

1.1 INTRODUCCIÓN

Un cuadricóptero es un dispositivo basado en las leyes básicas de un helicóptero convencional, con la variante de que el primero cuenta con 4 motores para mejorar la estabilidad y la rapidez del dispositivo.

Hoy en día los cuadricópteros no solo se limitan al aeromodelismo, ya que gracias a su bajo costo tanto de producción como de mantenimiento se puede ver aplicada esta tecnología en la industria del cine y mapeo o reconocimiento de áreas. También lo podemos encontrar en el campo del entretenimiento deportivo como "repartidores de regalos" e incluso en el campo de la música.

Figura 1.1 Cuadricóptero topográfico y tomas aéreas

Aunque la tecnología no ha sido explotada al cien por ciento, día con día se encuentran diversos tipos de aplicación; alguno de ellos podría ser para la industria agrícola, no solo para reconocer el producto sino también para esparcir la semilla. En ciertos casos se prevé que un cuadricóptero pueda ser usado como mensajero para entregar paquetes de cierto tamaño y peso.

Bluetooth es una especificación industrial para Redes Inalámbricas de Área Personal (WPAN) que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM de los 2,4 GHz. Los principales objetivos que se pretenden conseguir con esta norma son:

- Facilitar las comunicaciones entre equipos móviles.
- Eliminar los cables y conectores entre éstos.
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre equipos personales.

En el proyecto desarrollado se hace uso de esta tecnología para controlar de forma inalámbrica y a distancia un cuadricóptero, el cual tiene la facilidad de moverse dentro de los ejes X, Y, Z; Gracias a la integración de una antena de

Bluetooth acoplada a una tarjeta Arduino que se encargará de controlar por completo las variables que definen la velocidad de cada motor y determinar cierta acción.

Para controlar el cuadricóptero se necesita de un programa e interfaz de usuario diseñados para cualquier dispositivo Android. Esta aplicación se desarrolló desde cero y utiliza botones para determinar la acción que debe tomar el cuadricóptero.

Esta tesis detalla cómo se desarrolló por completo la integración de un controlador Arduino a una estructura de cuadricóptero previamente adquirida, para después agregarle una antena Bluetooth. Posteriormente se verá cómo se programó este controlador para que fuese capaz de transmitir y recibir datos por medio de esta.

Se encuentra como se diseñó y programo la interacción entre el dispositivo móvil y la tarjeta Arduino para finalmente crear la interfaz de usuario.

1.2 OBJETIVO GENERAL

Realizar la comunicación inalámbrica entre un controlador Arduino y un dispositivo móvil con sistema operativo Android aplicado al control de movimiento de un cuadricóptero.

1.3 OBJETIVOS ESPECÍFICOS

- Configurar el protocolo de Comunicación Bluetooth entre una tarjeta Arduino y un dispositivo móvil con Sistema Android.
- Implementar el cuadricóptero, el cual estará conformado por cuatro motores brushless, cuatro variadores de velocidad, cuatro hélices y un frame.
- Desarrollar el programa encargado de dirigir y controlar los motores del cuadricóptero.
- Desarrollar el software Cuadricóptero/Android para interacción entre dispositivos.
- Diseñar y programar la interfaz de usuario para dispositivos móviles Android.

1.3 JUSTIFICACIÓN

La idea de este proyecto surgió con el simple hecho de poder implementar este dispositivo en un ambiente laboral, tal es el caso de reconocimiento de áreas incapaces o difíciles de explorar por el ser humano, así como medios de comunicación en áreas cortas, mensajería entre oficinas de alguna empresas o simplemente como medio de diversión.

Los dispositivos móviles tienen como principal función comunicar al ser humano; hoy en día esa comunicación se ha ampliado y da facilidades para transmitir voz o texto plano, también permiten la transmisión de datos más complejos gracias a la integración de Internet, Bluetooth e Infra Rojo.

Ya sea por cualquier medio de transmisión de datos mencionados anteriormente, se puede accesar a todo tipo de información y conexión siendo que esta se encuentre en una distancia exacta.

En el campo laboral del Control y la Automatización usar ésta tecnología facilita el control de variables, así como la comunicación de información directa y sin precedentes en el dispositivo móvil.

Bajo esta perspectiva y atendiendo a las exigencias de la vanguardia tecnológica, se desarrolla un medio de aplicación Cuadricóptero/Android vía Bluetooth con el fin de demostrar la importancia de la integración de esta tecnología y las facilidades que tiene al contar con todo el control del dispositivo de manera distante.

Esta tecnología se puede aplicar a todo tipo de sistemas que requieran un monitoreo o control constante, facilitando la portabilidad gracias a los dispositivos Android.

1.4 ANTECEDENTES Y GENERALIDADES

En el siglo 20 el científico francés Charles Richet junto con Louis Bréguet construyeron un pequeño helicóptero piloteado que era una aeronave de alas giratorias donde la velocidad aerodinámica responsable de la sustentación proviene principalmente del giro de las palas del rotor, sin embargo, se levantaba del suelo pero carecía de estabilidad y capacidad de mando para efectuar un mando controlado.

El aparato fue denominado “Giroplano” que básicamente es un cuadricóptero con propulsores de 8.1 metros de diámetro y pesaba alrededor de 878 kilos con todo y los pilotos y con un solo motor de combustión interna de 50 Hp que manejaba los rotores a través de una transmisión de correa y polea.

En 1922 en Francia se crea un prototipo llamado Convertanwings Modelo A, diseñado para uso civil y militar. El diseño tenía dos motores que controlaban cuatro rotores, no tenía rotor de cola ya que usaba la diferencia en las velocidades de giro para lograr el desplazamiento.

Con el advenimiento de la nueva tecnología se ha incrementado el interés en el diseño de cuadricópteros, la empresa Bell diseño un Quad – Tiltrotor, capaz de llevar una gran carga útil alcanzando altas velocidades, emplea un pequeño espacio tanto en el despegue como en el aterrizaje y puede modificar la dirección de sus cuatro rotores hacia adelante para tener un vuelo vertical como un avión convencional.

El Moller Skycar es un prototipo de cuadricóptero a manera de auto volador, y consta de cuatro rotores en forma de ductos de ventilación que permite una operación segura y eficiente a bajas velocidades, su inventor fue Paul Moller. Actualmente ha mejorado el modelo aumentando el número de ductos y es capaz de levantarse a 3 metros sobre el suelo.

Los alumnos Verónica Gabriela Ortiz Padilla y Pablo Ramiro Pulla Arévalo de la Escuela Politécnica del Ejército en Sangolqui-Ecuador en la carrera de Ingeniería en Mecatrónica desarrollaron un prototipo bajo el nombre de “Desarrollo y construcción de un cuadricóptero a control remoto” el 27 de septiembre de 2012.

Otro prototipo es el desarrollado por Xavier Legasa Martín-Gil de la Facultad de Informática de Barcelona en la carrera de Ingeniería en Informática bajo el nombre de “Cuadricóptero Arduino por control remoto Android” el 3 de julio de 2012. Consiste en la realización del diseño e implementación de un sistema físico cuadricóptero, que será capaz de comunicarse remotamente para que sea posible controlar su dirección.

Iván Monzón Catalán del Departamento de Informática e Ingeniería de Sistemas de la Universidad de Zaragoza diseño un proyecto terminal de carrera llamado “Desarrollo de un Cuadricóptero operado por ROS” en septiembre de 2013.

En Quito Ecuador, en la Escuela Politécnica Nacional de la facultad de Ingeniería Eléctrica y Electrónica se desarrolló un proyecto denominado “ensamblaje y

control de un Cuadricóptero" por los alumnos William Oswaldo Chamorro Hernández y Jorge Luis Medina Mora, en abril de 2013. [16]

Consta de un dispositivo llamado AR Drone UAV manufacturado por la compañía francesa Parrot, consta de 4 rotores en los 4 extremos del UAV, y en su parte central se encuentra la batería junto con el hardware principal de funcionamiento. Cada par de rotores opuestos giran en el mismo sentido.

En lo que México se refiere, escuelas como la ESIME Ticomán, han desarrollado proyectos parecidos, uno de ellos se ha implementado en zonas como periférico norte, donde un dispositivo llamado Drone, operado bajo control remoto, recorre la ciudad con una cámara de alta definición y visión nocturna para monitorear el alto índice delictivo de la zona, aunque realmente se descartó por el alto precio en mantenimiento pues oscila entre los 30 mil y 40 mil pesos al mes.

En el Centro de Investigación y de Estudios Avanzados (CINVESTAV) existen posgrados que imparten cursos como "Diseño, estudios dinámicos y control de vehículos autónomos aéreos y submarinos" en su modalidad de maestría.

Así también teniendo en cuenta existen varios proyectos que han concluido alumnos de la materia, pues se han hecho Simposio respecto al tema de UAV. Por dar un ejemplo el del alumno Mario Espinosa Santiago que realizó una tesis, bajo el nombre de "Sistema háptico para un vehículo aéreo no tripulado" [16]

[16] Proyecto Final de Carrera Xabier Legasa Martín-Gil Ingeniería Informática Facultad de Informática de Barcelona (FIB) UPC BarcelonaTech 2011/2012.

CAPITULO 2 - MARCO TEORICO

Un cuadricóptero se puede definir como una aeronave que se eleva y se desplaza por el movimiento de cuatro motores colocados en los extremos de una estructura. Normalmente se utiliza el nombre inglés quadrotor aunque también existe la traducción cuadricóptero. La aeronave dispone de cuatro motores con sus hélices respectivas, se utiliza la velocidad de los motores para controlar la estabilidad y movimientos del vehículo aéreo. [12]

Una de las características a destacar es la gran maniobrabilidad que posee este tipo de vehículo. Al disponer de cuatro motores el control es bastante exacto, lo que ayuda a utilizarlo en aplicaciones donde la exactitud de vuelo estacionario sea muy importante. Una aplicación donde se aprecia esta característica es en la navegación de interiores y sitios de espacio reducido. Como en el helicóptero, estos vehículos disponen de una capacidad de vuelo vertical que los hacen únicos, esta función es ventajosa cuando no se quiere tener mucha velocidad horizontal y tener una buena capacidad de vuelo estacionario.

2.1 CLASIFICACIÓN

Existen varias formas para clasificar este tipo de dispositivos denominados Vehículos Aéreos no Tripulados, que por definición es una aeronave que opera sin tripulación humana abordo y que pueden ser controladas desde una base o por funcionamiento autónomo.

Se clasifican según el tipo de misión por el que fueron hechos como: Simulación de blancos, reconocimiento de terreno, combate, logística, investigación y desarrollo, así también como por el alcance de estos (Tabla 1). También se clasifican por el tipo de despegue ya sea vertical u horizontal.

Cada uno posee sus propias características y todo dependerá del uso que se le dé así como del controlador que se emplee.

[12] https://github.com/strangedev/Arduino-Quadcopter/blob/master/quadcopter/quadcopter.ino

Tabla 1 Clasificación de unidades de vuelo no tripuladas

CATEGORÍA	ACRÓNIMO	ALCANCE (Km)	AUTONOMÍA (h)	ALTITUD DE VUELO (m)	CARGA MÁXIMA (Kg)
Micro <250 gr	Micro	< 10	1	250	<5
Mini <25 Kg	Mini	< 10	< 2	150 y 300	<30
Alcance cercano	CR	10 a 30	2 a 4	3000	150
Alcance corto	SR	30 a 70	3 a 6	3000	200
Alcance medio	MR	70 a 200	6 a 10	5000	1250
Altitud baja	LADP	> 250	0.5 a 1	50 a 9000	350
Altitud media	MRE	> 500	10 a 18	8000	1250
Autonomía alta Altitud baja	LALE	> 500	> 24	3000	<30
Autonomía alta Altitud media	MALE	> 500	24 a 48	14000	1500
Autonomía alta Altitud alta	HALE	> 2000	24 a 48	20000	12000
Combate	UCAV	1500	2	10000	10000
Ofensivo	LETH	300	3 a 4	4000	250
Señuelo	DEC	500	4	5000	250
Estratosférico	STRATO	> 2000	> 48	2000 y 30000	ND
EXO-estratosférico	EXO	ND	ND	> 30000	ND

2.2 SISTEMA MECÁNICO

La parte mecánica de un cuadricóptero consta de un bastidor y cuatro hélices dobles (Figura 2.1).

Figura 2.1 Frame de cuadricóptero

2.2.1 HÉLICES

Las hélices constituyen los elementos móviles de la aeronave, las mismas que están estandarizadas; por lo que se seleccionarán de catálogos tomando en cuenta los parámetros: Diámetro y Peso.

2.3 MOVIMIENTOS

2.3.1 GUADAÑA (Yaw)

Se refiere al movimiento cuando el vehículo gira sobre su eje vertical. El cuadricóptero logra este movimiento al aumentar por igual la potencia de giro 8 de los rotores 1 y 3 y disminuir en igual magnitud los motores 2 y 4. Al disminuir esta potencia aumenta el par del motor creando un giro contrario a las hélices que están rotando con mayor potencia.

2.3.2 INCLINACIÓN (Pitch)

Es el movimiento que permite el desplazamiento hacia adelante y atrás. El vehículo mantiene la potencia en el rotor 1 que es opuesto al sentido deseado, reduce al mínimo la del rotor 3 y deja los otros dos a potencia media, así la sustentación del rotor 1 hace que el vehículo se incline a favor del sentido deseado y se desplace.

2.3.3 BAMBOLEO (Roll)

Permite realizar los movimientos a la izquierda o derecha. Usa el mismo principio que el de inclinación, pero lateralmente. La combinación de los tres movimientos mencionados son los que hacen maniobrar al cuadricóptero libremente.

Los movimientos de roll y pitch son giros en torno a los ejes horizontales del cuadricóptero. Una inclinación en cualquiera de estos ejes produce un movimiento lineal en el plano horizontal cuya velocidad depende del ángulo (esto se denomina ángulo de ataque) y la dirección depende de la orientación del cuadricóptero

2.3.4 VERTICAL

Se puede hacer ascender, descender o mantener en vuelo estacionario al cuadricóptero, haciendo que la fuerza de sustentación generada por los 4 pares motores-hélices sea mayor que la fuerza peso generada por la atracción gravitatoria.

Este movimiento se logra al variar la potencia de los cuatro motores en igual medida para no modificar los demás grados de libertad. De esta forma la plataforma puede ascender o descender (figura 2.2).

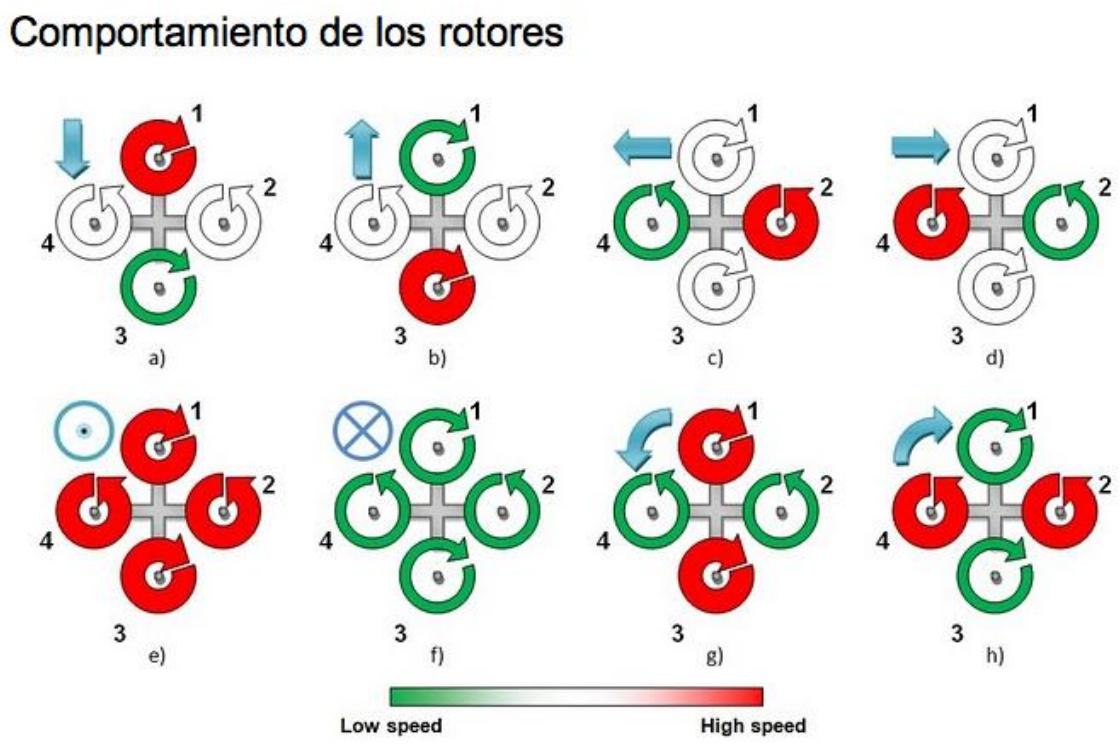


Figura 2.2 Movimiento de un "dron"

Lo dicho anteriormente es para el caso ideal, donde los cuatro motores tienen exactamente las mismas constantes y giran a las mismas velocidades angulares. En el caso real, esto no es posible, porque siempre existen ciertas variaciones en las fuerzas generadas por los motores, ya sea porque las constantes de los motores son diferentes o las velocidades angulares giran a distintas revoluciones.

El diseño de un controlador para el cuadricóptero no es un problema trivial debido a varias razones como son el acoplamiento entre las fuerzas y momentos generados por los motores y hélices, el rozamiento con el aire, la fuerza de empuje del viento, etc. [1]

[1] Ortega, Manuel R. (1989-2006) (en español). Lecciones de Física (4 volúmenes). Monytex

2.4 SISTEMA DE CONTROL.

Arduino es una plataforma de creación de prototipos electrónicos de código abierto basado en hardware y software fácil de usar, flexible, basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares (Figura 2.3).

Figura 2.3 Arduino UNO

Arduino se puede utilizar para desarrollar objetos interactivos autónomos o puede ser conectado a software del ordenador. Las placas se pueden montar a mano o adquirirse. El entorno de desarrollo integrado libre se puede descargar gratuitamente.

Los tableros pueden ser construidos por la mano o comprados pre-montados, el software puede ser descargado de forma gratuita. Los diseños de referencia de hardware (archivos CAD) son disponibles bajo una licencia de código abierto, que son libres de adaptarlos a sus necesidades.

Arduino puede tomar información del entorno a través de sus pines de entrada de toda una gama de sensores y puede afectar aquello que le rodea controlando luces, motores y otros actuadores. El microcontrolador en la placa Arduino se programa mediante el lenguaje de programación Arduino (basado en Wiring) y el entorno de desarrollo Arduino (basado en Processing). Los proyectos hechos con Arduino pueden ejecutarse sin necesidad de conectar a un ordenador, si bien tienen la posibilidad de hacerlo y comunicar con diferentes tipos de software. Las placas pueden ser hechas a mano o compradas montadas de fábrica; el software puede ser descargado de forma gratuita. Los ficheros de diseño de referencia (CAD) están disponibles bajo una licencia abierta, así pues eres libre de adaptarlos a tus necesidades.

La plataforma Arduino se programa mediante el uso de un lenguaje propio basado en el lenguaje de programación de alto nivel Processing. Sin embargo, es posible utilizar otros lenguajes de programación y aplicaciones populares en Arduino, debido a que Arduino usa la transmisión serial de datos soportada por la mayoría de los lenguajes mencionados. Para los que no soportan el formato serie de forma nativa, es posible utilizar software intermediario que traduzca los mensajes enviados por ambas partes para permitir una comunicación fluida. Algunos ejemplos son: Adobe Director, C, C++, Flash, Gambas, Isadora, Java, Matlab, entre otros. [8]

[8] arduino.cc/en/Main/arduinoBoardMega2560

2.4.1 ALGORITMOS DE CONTROL

Un algoritmo es la expresión de una secuencia precisa de operaciones que conduce a la resolución de un problema. Es un sistema de reglas que permiten obtener una salida específica a partir de una entrada específica. Cada paso debe estar definido exactamente, de forma que pueda traducirse a lenguaje de computadora.

Propiedades de los Algoritmos.

- Debe ser finito
- Toda regla debe definir perfectamente la acción a desarrollar
- Todos sus pasos deben ser simples y tener un orden definido.
- Un Algoritmo no debe resolver un solo problema particular sino una clase de problemas.
- Un Algoritmo debe ser eficiente y rápido.

2.4.2 CONTROL PROPORCIONAL

La parte proporcional consiste en el producto entre la señal de error y la constante proporcional para lograr que el error en estado estacionario se aproxime a cero, pero en la mayoría de los casos, estos valores solo serán óptimos en una determinada porción del rango total de control, siendo distintos los valores óptimos para cada porción del rango. Sin embargo, existe también un valor límite en la constante proporcional a partir del cual, en algunos casos, el sistema alcanza valores superiores a los deseados. Este fenómeno se llama sobre oscilación y, por razones de seguridad, no debe sobrepasar el 30%, aunque es conveniente que la parte proporcional ni siquiera produzca sobre oscilación. Hay una relación lineal continua entre el valor de la variable controlada y la posición del elemento final de control. La parte proporcional no considera el tiempo, por lo tanto, la mejor manera de solucionar el error permanente y hacer que el sistema contenga alguna componente que tenga en cuenta la variación respecto al tiempo, es incluyendo y configurando las acciones integral y derivativa.

La fórmula del proporcional está dada por:

$$P_{\text{sal}} = K_p\, e(t) \tag{2.1}$$

El error, la banda proporcional y la posición inicial del elemento final de control se expresan en tanto por uno. Nos indicará la posición que pasará a ocupar el elemento final de control.

2.4.3 CONTROL INTEGRAL

El modo de control Integral tiene como propósito disminuir y eliminar el error en estado estacionario, provocado por el modo proporcional. El control integral actúa cuando hay una desviación entre la variable y el punto de consigna, integrando esta desviación en el tiempo y sumándola a la acción proporcional. El error es integrado, lo cual tiene la función de promediarlo o sumarlo por un período determinado; Luego es multiplicado por una constante I. Posteriormente, la respuesta integral es adicionada al modo Proporcional para formar el control P + I con el propósito de obtener una respuesta estable del sistema sin error estacionario.

El modo integral presenta un desfasamiento en la respuesta de 90º que sumados a los 180º de la retroalimentación acercan al proceso a tener un retraso de 270º, luego entonces solo será necesario que el tiempo muerto contribuya con 90º de retardo para provocar la oscilación del proceso. La ganancia total del lazo de control debe ser menor a 1, y así inducir una atenuación en la salida del controlador para conducir el proceso a estabilidad del mismo. Se caracteriza por el tiempo de acción integral en minutos por repetición. Es el tiempo en que delante una señal en escalón, el elemento final de control repite el mismo movimiento correspondiente a la acción proporcional.

El control integral se utiliza para obviar el inconveniente del offset (desviación permanente de la variable con respecto al punto de consigna) de la banda proporcional.

La fórmula del integral está dada por:

$$I_{\text{sal}} = K_i \int_0^t e(\tau)\, d\tau \qquad (2.2)$$

2.4.4 CONTROL DERIVATIVO

La acción derivativa se manifiesta cuando hay un cambio en el valor absoluto del error.

El error es la desviación existente entre el punto de medida y el valor consigna, o Set Point.

La función de la acción derivativa es mantener el error al mínimo corrigiéndolo proporcionalmente con la misma velocidad que se produce; de esta manera evita que el error se incremente.

Se deriva con respecto al tiempo y se multiplica por una constante D y luego se suma a las señales anteriores (P+I). Es importante adaptar la respuesta de control a los cambios en el sistema ya que una mayor derivativa corresponde a un cambio más rápido y el controlador puede responder acordemente.

La fórmula del derivativo está dada por:

$$D_{\text{sal}} = K_d \frac{de}{dt} \tag{2.3}$$

El control derivativo se caracteriza por el tiempo de acción derivada en minutos de anticipo.

Cuando el tiempo de acción derivada es grande, hay inestabilidad en el proceso. Cuando el tiempo de acción derivada es pequeño la variable oscila demasiado con relación al punto de consigna. Suele ser poco utilizada debido a la sensibilidad al ruido que manifiesta y a las complicaciones que ello conlleva.

El tiempo óptimo de acción derivativa es el que retorna la variable al punto de consigna con las mínimas oscilaciones

La acción derivada puede ayudar a disminuir el rebasamiento de la variable durante el arranque del proceso. Puede emplearse en sistemas con tiempo de retardo considerables, porque permite una repercusión rápida de la variable después de presentarse una perturbación en el proceso.

2.4.5 CONTROL PROPORCIONAL DERIVATIVO

El controlador derivativo se opone a desviaciones de la señal de entrada, con una respuesta que es proporcional a la rapidez con que se producen éstas. Si se considera que:

y(t) = Salida diferencial.

e(t) = Error (diferencia entre medición y punto de consigna [PC]. El PC no es otra cosa que el nivel deseado al que se quiere que vuelva el sistema)

T_d = Tiempo diferencial, se usa para dar mayor o menor trascendencia a la acción derivativa.

La salida de este regulador es:

$$y(t) = t_d \cdot \frac{de(t)}{dt} \tag{2.4}$$

Que en el dominio de Laplace, será:

$$Y(s) = T_d \cdot s \cdot E(s) \tag{2.5}$$

Por lo que su función de transferencia será:

$$G(s) = \frac{Y(s)}{E(s)} = T_d \cdot s \tag{2.6}$$

Si la variable de entrada es constante, no da lugar a respuesta del regulador diferencial, cuando las modificaciones de la entrada son instantáneas, la velocidad de variación será muy elevada, por lo que la respuesta del regulador diferencial será muy brusca, lo que haría desaconsejable su empleo.

El regulador diferencial tampoco actúa exclusivamente, si no que siempre lleva asociada la actuación de un regulador proporcional, la salida del bloque de control responde a la siguiente ecuación:

$$y(t) = K_P \cdot t_d \cdot \frac{de(t)}{dt} + K_P \cdot e(t) \tag{2.7}$$

Kp y Td son parámetros ajustables del sistema. A Td es llamado tiempo derivativo y es una medida de la rapidez con que un controlador PD compensa un cambio en la variable regulada, comparado con un controlador P puro. Que en el dominio de Laplace, será:

$$Y(s) = K_P \cdot T_d \cdot s \cdot E(s) + K_P \cdot E(s) \tag{2.8}$$

Y por tanto la función de transferencia del bloque de control PD será:

$$G(s) = \frac{Y(s)}{E(s)} = K_P \cdot (T_d \cdot s + 1) \tag{2.9}$$

En los controladores diferenciales, al ser la derivada de una constante igual a cero, el control derivativo no ejerce ningún efecto, siendo únicamente práctico en aquellos casos en los que la señal de error varía en el tiempo de forma continua.

Por lo que, el análisis de este controlador ante una señal de error tipo escalón no tiene sentido, por ello, se representa la salida del controlador en respuesta a una señal de entrada en forma de rampa unitaria.

Este tipo de controlador se utiliza en sistemas que deben actuar muy rápidamente, ofreciendo una respuesta tal que provoca que la salida continuamente esté cambiando de valor.

El regulador derivativo no se emplea aisladamente, ya que para señales lentas, el error producido en la salida en régimen permanente sería muy grande y si la señal de mando dejase de actuar durante un tiempo largo la salida tendería hacia cero y con lo que no se realizaría ninguna acción de control.

La ventaja de este tipo de controlador es que aumenta la velocidad de respuesta del sistema de control.

Al actuar conjuntamente con un controlador proporcional las características de un controlador derivativo, provocan una apreciable mejora de la velocidad de respuesta del sistema, aunque pierde precisión en la salida.

2.4.1.4 CONTROL PROPORCIONAL INTEGRAL

No existen controladores que actúen únicamente con acción integral, siempre actúan en combinación con reguladores de una acción proporcional, complementándose los dos tipos de reguladores, primero entra en acción el regulador proporcional, mientras que el integral actúa durante un intervalo de tiempo.

(Ti=tiempo integral)

La Función de transferencia del bloque de control PI responde a la ecuación:

$$G(s) = \frac{Y(s)}{E(s)} = K_P \cdot \left(\frac{1}{T_i \cdot s} + 1\right) \qquad (2.10)$$

Donde Kp y Ti son parámetros que se pueden modificar según las necesidades del sistema. Si Ti es grande la pendiente de la rampa, correspondiente al efecto integral será pequeña y, su efecto será atenuado, y viceversa.

2.4.1.5 CONTROL PROPORCIONAL INTEGRAL DERIVATIVO

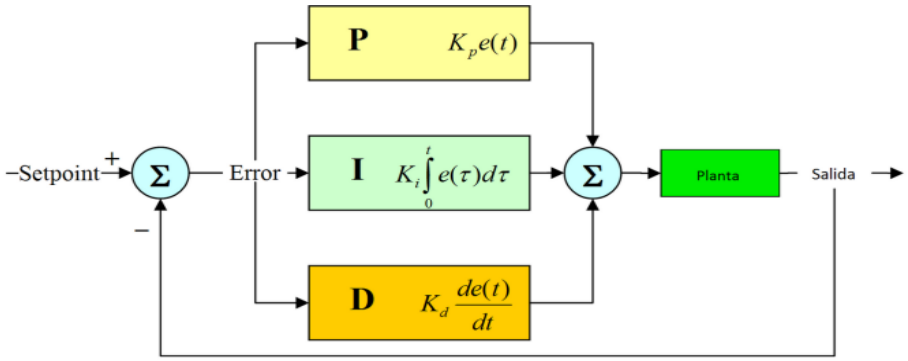
Un PID es un mecanismo de control por realimentación que calcula la desviación o error entre un valor medido y el valor que se quiere obtener, para aplicar una acción correctora que ajuste el proceso. El algoritmo de cálculo del control PID se da en tres parámetros distintos: el proporcional, el integral, y el derivativo. El valor Proporcional determina la reacción del error actual. El Integral genera una corrección proporcional a la integral del error, esto asegura que aplicando un

esfuerzo de control suficiente, el error de seguimiento se reduce a cero. El Derivativo determina la reacción del tiempo en el que el error se produce. La suma de estas tres acciones es usada para ajustar al proceso vía un elemento de control como la posición de una válvula de control o la energía suministrada a un calentador, por ejemplo. Ajustando estas tres variables en el algoritmo de control del PID, el controlador puede proveer un control diseñado para lo que requiera el proceso a realizar. La respuesta del controlador puede ser descrita en términos de respuesta del control ante un error, el grado el cual el controlador llega al set point, y el grado de oscilación del sistema. El uso del PID para control no garantiza control óptimo del sistema o la estabilidad del mismo. Algunas aplicaciones pueden solo requerir de uno o dos modos de los que provee este sistema de control. Un controlador PID puede ser llamado también PI, PD, P o I en la ausencia de las acciones de control respectivas. Los controladores PI son particularmente comunes, ya que la acción derivativa es muy sensible al ruido, y la ausencia del proceso integral puede evitar que se alcance al valor deseado debido a la acción de control.

De la documentación existente sobre sistemas de control, podemos destacar la siguiente ecuación.

$$u(t) = K_p e(t) + \frac{K_p}{T_i}\int_0^t e(t)dt + K_p T_d \frac{de(t)}{dt} \qquad (2.11)$$

Para tener una idea más clara, recurrimos al siguiente diagrama:



(2.12)

De la ecuación, podemos hacer las siguientes afirmaciones:

- e(t) es el error de la señal.
- u(t) salida del controlador y entrada de control al proceso.
- Kp es la ganancia proporcional.
- Ti es la constante de tiempo integral.
- Td es la constante de tiempo derivativa.

Del diagrama de flujo determinamos lo siguiente:

● El primer bloque de control (proporcional) consiste en el producto entre la señal de error y la constante proporcional, quedando un error en estado estacionario casi nulo.

● El segundo bloque de control (integral) tiene como propósito disminuir y eliminar el error en estado estacionario, provocado por el modo proporcional. El control integral actúa cuando hay una desviación entre la variable y el punto de consigna, integrando esta desviación en el tiempo y sumándola a la acción proporcional.

● El tercer bloque de control (Derivativo) considera la tendencia del error y permite una repercusión rápida de la variable después de presentarse una perturbación en el proceso.

2.4.1.6 SINTONIZACION PID

Sintonizar un controlador PID significa establecer el valor que deben tener los parámetros de Ganancia (Banda Proporcional), Tiempo Integral (Reset) y Tiempo derivativo (Rate), para que el sistema responda en una forma adecuada. La primera etapa de todo procedimiento de sintonización consiste en obtener la información estática y dinámica del lazo. Existen diversos métodos para ajustar los parámetros de controladores PID, pero todos caen dentro de dos tipos:

- Métodos en Lazo Cerrado: la información de las características del lazo se obtienen a partir de un test realizado en lazo cerrado, usualmente con un consolador con acción proporcional pura. (figura 2.4)

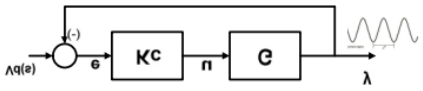


Figura 2.4 lazo de control cerrado

- Métodos en Lazo Abierto: la características estáticas y dinámicas de la planta (Elemento Final de Control + Proceso + Transmisor) se obtienen de un ensayo en lazo abierto, generalmente la respuesta a un escalón (Curva de Respuesta). (figura 2.5) [15]

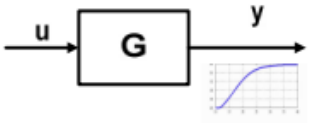


Figura 2.5 lazo de control abierto

[15] OGATA KATSUHIKO, Ingeniería en control moderno 3° Edición Prince Hall Hispanoamericana, 1998.

2.5 SISTEMA DE COMUNICACION

Se denomina Bluetooth al protocolo de comunicaciones diseñado especialmente para dispositivos de bajo consumo, que requieren corto alcance de emisión y basados en transceptores de bajo costo.

Los dispositivos que incorporan este protocolo pueden comunicarse entre ellos cuando se encuentran dentro de su alcance. Las comunicaciones se realizan por radiofrecuencia de forma que los dispositivos no tienen que estar alineados y pueden incluso estar en habitaciones separadas si la potencia de transmisión es suficiente. Estos dispositivos se clasifican como "Clase 1", "Clase 2" o "Clase 3" en referencia a su potencia de transmisión, siendo totalmente compatibles los dispositivos de una caja de ordenador (tabla 2).

Tabla 2 Clasificación de protocolos de comunicación

Clase	Potencia máxima permitida (mW)	Potencia máxima permitida (dBm)	Alcance (aproximado)
Clase 1	100 Mw	20 dBm	~30 metros
Clase 2	2.5 Mw	4 dBm	~10-5 metros
Clase 3	1 Mw	0 dBm	~1 metro

La utilidad Bluetooth fue desarrollada en 1994 por Jaap Haartsen y Mattisson Sven, como reemplazo de cable, que estaban trabajando para Ericsson en Lund, Suecia. La utilidad se basa en la tecnología de saltos de frecuencia de amplio espectro.

Las prestaciones fueron publicadas por el Bluetooth Special Interest Group (SIG). El SIG las anunció formalmente el 20 de mayo de 1998. Hoy cuenta con una membresía de más de 20.000 empresas en todo el mundo. Fue creado por Ericsson, IBM, Intel, Toshiba y Nokia, y posteriormente se sumaron muchas otras compañías. Todas las versiones de los estándares de Bluetooth están diseñadas para la retro compatibilidad, que permite que el último estándar cubra todas las versiones anteriores (Figura 2.6). [13]

Figura 2.6 modulo Bluetooth

[13]http://www.emartee.com/product/41915/HC%2006%20Serial%20Port%20Bluetooth%20Module

2.6 SISTEMA ELECTRONICO

Entre los dispositivos y sensores electrónicos que servirán para controlar la aeronave están:

2.6.1 MOTORES

Un motor es la parte sistemática de una máquina capaz de hacer funcionar el sistema, transformando algún tipo de energía (eléctrica, de combustibles fósiles, etc.) en energía mecánica capaz de realizar un trabajo.

La elección de motores es una decisión importante, existen dos tipos: Los motores con escobillas (brushed motors o dc motors) y los motores sin escobillas (brushless motors). [7]

2.6.1.1 MOTORES BRUSHED

Se compone principalmente de dos partes. El estator da soporte mecánico al aparato y contiene los devanados principales de la máquina, conocidos también con el nombre de polos, que pueden ser de imanes permanentes o devanados con hilo de cobre sobre núcleo de hierro. El rotor es generalmente de forma cilíndrica, también devanado y con núcleo, alimentado con corriente directa mediante escobillas fijas (conocidas también como carbones).

El principal inconveniente de estas máquinas es el mantenimiento, muy caro y laborioso, debido principalmente al desgaste que sufren las escobillas al entrar en contacto con las delgas (Figura 2.7).

Figura 2.7 Motor Brushed

Algunas aplicaciones especiales de estos motores son los motores lineales, cuando ejercen tracción sobre un riel, o bien los motores de imanes permanentes. Los motores de corriente continua (CC) también se utilizan en la construcción de servomotores y motores paso a paso. Además existen motores de CD sin escobillas.

Es posible controlar la velocidad y el par de estos motores utilizando técnicas de control de motores CD.

[7] Ohio Motores Eléctricos. DC de protección del motor. Ohio Motores Eléctricos.2011.

2.6.1.2 MOTORES BRUSHLESS

Un motor eléctrico sin escobillas o motor brushless es un motor eléctrico que no emplea escobillas para realizar el cambio de polaridad en el rotor.

Los motores eléctricos solían tener un colector de delgas o un par de anillos rozantes. Estos sistemas, que producen rozamiento, disminuyen el rendimiento, desprenden calor y ruido, requieren mucho mantenimiento y pueden producir partículas de carbón que manchan el motor de un polvo que, además, puede ser conductor (Figura 2.8).

Figura 2.8 Motor Brushless

Los primeros motores sin escobillas fueron los motores de corriente alterna asíncronos. Hoy en día, gracias a la electrónica, se muestran muy ventajosos, ya que son más baratos de fabricar, pesan menos y requieren menos mantenimiento, pero su control era mucho más complejo. Esta complejidad prácticamente se ha eliminado con los controles electrónicos. [6]

El inversor debe convertir la corriente alterna en corriente continua, y otra vez en alterna de otra frecuencia. Otras veces se puede alimentar directamente con corriente continua, eliminado el primer paso. Por este motivo, estos motores de corriente alterna se pueden usar en aplicaciones de corriente continua, con un rendimiento mucho mayor que un motor de corriente continua con escobillas. Algunas aplicaciones serían los coches y aviones con radiocontrol, que funcionan con pilas.

Existen dos tipos diferentes que son:

° Motores Outrunner que giran más despacio y el par es mucho mayor.

° Motores Irunner que son más eficientes cuando más rápido gira el motor, necesitan de un alimento adicional entre el motor y la hélice.

[6] TG Wilson, PH Trickey, "Máquina de CC. Con Solid State Conmutación", AIEE papel I. CP62-1372, 07 de octubre 1962

Tabla 3 Ventajas y desventajas de los motores brushed y brushless

VENTAJAS	DESVENTAJAS	VENTAJAS	DESVENTAJAS
CONTROL SIMPLE Y SENCILLO	PRECISAN MANTENIMIENTO PERIODICO	CONMUTACION BASADA EN SENSORES	MAS CAROS
OPERA EN SITUACIONES EXTREMAS	DISCIPACION DEL CALOR POBRE	MENOR MANTENIMIENTO	LOS ESC´s SE HACEN NECESARIOS
NO HACE FALTA CONTROLADOR	RANGO DE VELCIDAD MENOR	MAYOR EFICIENCIA	
MAS ECONOMICOS	LAS ESCOBILLAS GENERAN RUIDO	DISIPA MEJOR EL CALOR	
		MAYOR RANGO DE VELOCIDAD	
		GENERA MENOR RUIDO	

2.6.2 GIROSCÓPIO ELECTRÓNICO

El giróscopo o giroscopio es un dispositivo mecánico que sirve para medir, mantener o cambiar la orientación en el espacio de algún aparato o vehículo (Figura 2.9).

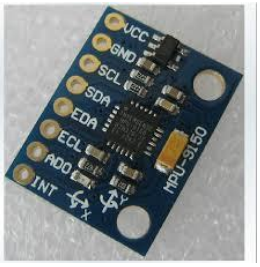


Figura 2.9 Giroscopio electrónico

El giróscopo como tal fue inventado, con ése mismo nombre, en 1852 por Foucault, montando una masa rotatoria en un soporte de Cardano para un experimento de demostración de la rotación de la Tierra. La rotación ya había sido demostrada con el péndulo de Foucault. Sin embargo no comprendía por qué la velocidad de rotación del péndulo era más lenta que la velocidad de rotación de la Tierra por un factor senoide λ, donde λ representa la latitud en que se localiza el péndulo. Se necesitaba otro aparato para demostrar la rotación de la Tierra de forma más simple.

Está formado esencialmente por un cuerpo con simetría de rotación que gira al rededor del eje de dicha simetría. Cuando el giróscopo se somete a un momento de fuerza que tiende a cambiar la orientación de su eje de rotación, tiene un comportamiento aparentemente paradójico, ya que cambia de orientación girando

respecto de un tercer eje, perpendicular tanto a aquel respecto del cual se lo ha empujado a girar, como a su eje de rotación inicial. Si está montado sobre un soporte de cardano que minimiza cualquier momento angular externo, o si simplemente gira libre en el espacio, el giróscopo conserva la orientación de su eje de rotación ante fuerzas externas que tiendan a desviarlo mejor que un objeto no giratorio; se desvía mucho menos, y en una dirección diferente.

2.6.3 ACELERÓMETRO

Se denomina acelerómetro a cualquier instrumento destinado a medir aceleraciones. Esto no es necesariamente la misma que la aceleración de coordenadas, sino que es el tipo de aceleración asociada con el fenómeno de peso experimentado por una masa de prueba que se encuentra en el marco de referencia del dispositivo. Un ejemplo en el que este tipo de aceleraciones son diferentes es cuando un acelerómetro medirá un valor sentado en el suelo, ya que las masas tienen un peso, a pesar de que no hay cambio de velocidad. Sin embargo, un acelerómetro en caída gravitacional libre hacia el centro de la Tierra medirá un valor de cero, ya que, a pesar de que su velocidad es cada vez mayor, está en un marco de referencia en el que no tiene peso.

Actualmente es posible construir acelerómetros de tres ejes (X, Y, Z) en un sólo chip de silicio, incluyendo en el mismo la parte electrónica que se encarga de procesar las señales (Figura 2.10).

Figura 2.10 Acelerómetro electrónico

El principio de operación de los dispositivos, acelerómetros e inclinómetros de tecnología MEMS, están basados en el traspaso térmico, por convección natural.

Estos dispositivos miden cambios internos, de la transferencia de calor causada por la aceleración, ofreciendo ventajas significativas sobre el empleo de una estructura tradicional sólida de masas de prueba.

Ya que la masa de prueba en el diseño de los sensores MEMS son moléculas de gas, las estructuras móviles mecánicas son eliminadas dentro del acelerómetro.

2.6.4 VARIADOR DE VELOCIDAD

El Variador de Velocidad (VSD, por sus siglas en inglés Variable Speed Drive) es en un sentido amplio un dispositivo o conjunto de dispositivos mecánicos, hidráulicos, eléctricos o electrónicos empleados para controlar la velocidad giratoria de maquinaria, especialmente de motores. También es conocido como Accionamiento de Velocidad Variable (ASD, también por sus siglas en inglés Adjustable-Speed Drive). De igual manera, en ocasiones es denominado mediante el anglicismo Drive, costumbre que se considera inadecuada. La maquinaria industrial generalmente es accionada a través de motores eléctricos, a velocidades constantes o variables, pero con valores precisos. No obstante, los motores eléctricos generalmente operan a velocidad constante o casi-constante, y con valores que dependen de la alimentación y de las características propias del motor, los cuales no se pueden modificar fácilmente. Para lograr regular la velocidad de los motores, se emplea un controlador especial que recibe el nombre de variador de velocidad.

2.6.4.1 VARIADORES PARA MOTORES CC

Estos variadores permiten controlar la velocidad de motores de corriente continua serie, derivación, compuesto y de imanes permanentes (Figura 2.11).

Figura 2.11 Variador de velocidad para motores CC

Para el caso de cualquiera de las máquinas anteriores se cumple la siguiente expresión de la Ecuación 1:

$$Vt = K \cdot FM \cdot Nm \tag{2.1}$$

Dónde:

Vt Es el Voltaje terminal (V).

K Es la constante de la máquina.

FM Flujo magnético producido por el campo (Wb)

Nm Velocidad mecánica (rpm).

Despejando la velocidad mecánica, se obtiene la Ecuación 2:

$$Nm = \frac{Vt}{K \cdot FM} \tag{2.2}$$

Entonces, puede observarse que la velocidad mecánica de un motor de CC es directamente proporcional al voltaje terminal (VT) e inversamente proporcional al flujo magnético (FM), el cual a su vez depende de la corriente de campo (IF). Aprovechando esta situación es que este tipo de variadores puede controlar la velocidad de un motor de CC: controlando su voltaje terminal, o bien, manipulando el valor de la corriente de campo.

2.6.5 BATERIAS

Un acumulador eléctrico es un dispositivo capaz de poner en reserva y luego restituir la energía eléctrica a la demanda (Figura 2.12). Un elemento recargable es llamado acumulador. Una batería se compone de varios acumuladores, generalmente puestos en serie para aumentar la tensión. La tensión de la batería es igual a la suma de las tensiones de los acumuladores que la componen. Las baterías electroquímicas tienen propiedades muy interesantes. Entre el polo positivo y el negativo existe una diferencia de potencial que varía muy poco en función de su carga. Las baterías electroquímicas guardan energía modificando la estructura molecular de los elementos químicos que la componen.

Figura 2.12 Baterías LiPo

Las baterías electroquímicas tienen propiedades muy interesantes. Entre el polo positivo y el negativo existe una diferencia de potencial que varía muy poco en función de su carga. Las baterías electroquímicas guardan energía modificando la estructura molecular de los elementos químicos que la componen. El motor extrae la energía y pone a la batería en su estado molecular de origen.

2.7 SISTEMA OPERATIVO ANDROID

Android es un sistema operativo basado en el kernel de Linux diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes o tabletas, inicialmente desarrollado por Android, Inc. Google respaldó económicamente y más tarde compró esta empresa en 2005. Android fue presentado en 2007 junto la fundación del Open Handset Alliance: un consorcio de compañías de hardware, software y telecomunicaciones para avanzar en los estándares abiertos de los dispositivos móviles. El primer móvil con el sistema operativo Android fue el HTC Dream y se vendió en octubre de 2008 (Figura 2.13).

Figura 2.13 Logo del Sistema Operativo Android

En la Tabla 4 se detallan las principales características del sistema operativo Android. Dicha tabla muestra claramente cómo es que Android es un sistema operativo que se encuentra en crecimiento constante y es por ello que el presente proyecto basa su interfaz de comunicación en él.

Tabla 3 Características principales del Sistema Operativo Android

Diseño de dispositivo	La plataforma es adaptable a pantallas de mayor resolución, VGA, biblioteca de gráficos 2D, biblioteca de gráficos 3D basada en las especificaciones de la Open GL ES 2.0 y diseño de teléfonos tradicionales.
Almacenamiento	SQ Lite, una base de datos liviana, que es usada para propósitos de almacenamiento de datos.
Conectividad	Android soporta las siguientes tecnologías de conectividad: GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, HSDPA, HSPA+, NFC y WiMAX.GPRS,UMTS,HSPA+ Y HSDPA+
Mensajería	SMS y MMS son formas de mensajería, incluyendo mensajería de texto y ahora la Android Cloud to Device Messaging Framework (C2DM) es parte del servicio de Push Messaging de Android.
Navegador web	El navegador web incluido en Android está basado en el motor de renderizado de código abierto WebKit, emparejado con el motor JavaScript V8 de Google Chrome. El navegador por defecto de Ice Cream Sándwich obtiene una puntuación de 100/100 en el test Acid3.
Soporte de Java	Aunque la mayoría de las aplicaciones están escritas en Java, no hay una máquina virtual Java en la plataforma. El bytecode Java no es ejecutado, sino que primero se compila en un ejecutable Dalvik y corre en la Máquina Virtual Dalvik. Dalvik es una máquina virtual especializada, diseñada específicamente para Android y optimizada para dispositivos móviles que funcionan con batería y que tienen memoria y procesador limitados. El soporte para J2ME puede ser agregado mediante aplicaciones de terceros como el J2ME MIDP Runner.
Soporte multimedia	Android soporta los siguientes formatos multimedia: WebM, H.263, H.264, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF y BMP.

Soporte para streaming	Streaming RTP/RTSP (3GPP PSS, ISMA), descarga progresiva de HTML (HTML5 <video> tag). Adobe Flash Streaming (RTMP) es soportado mediante el Adobe Flash Player. Se planea el soporte de Microsoft Smooth Streaming con el port de Silverlight a Android. Adobe Flash HTTP Dynamic Streaming estará disponible mediante una actualización de Adobe Flash Player.
Soporte para hardware adicional	Android soporta cámaras de fotos, de vídeo, pantallas táctiles, GPS, acelerómetros, giroscopios, magnetómetros, sensores de proximidad y de presión, sensores de luz, gamepad, termómetro, aceleración por GPU 2D y 3D.
Entorno de desarrollo	Incluye un emulador de dispositivos, herramientas para depuración de memoria y análisis del rendimiento del software. El entorno de desarrollo integrado es Eclipse (actualmente 3.4, 3.5 o 3.6) usando el plugin de Herramientas de Desarrollo de Android.
Google Play	Google Play es un catálogo de aplicaciones gratuitas o de pago en el que pueden ser descargadas e instaladas en dispositivos Android sin la necesidad de un PC.
Multitáctil	Android tiene soporte nativo para pantallas capacitivas con soporte multitáctil que inicialmente hicieron su aparición en dispositivos como el HTC Hero. La funcionalidad fue originalmente desactivada a nivel de kernel (posiblemente para evitar infringir patentes de otras compañías). Más tarde, Google publicó una actualización para el Nexus One y el Motorola Droid que activa el soporte multitáctil de forma nativa.
Bluetooth	El soporte para A2DF y AVRCP fue agregado en la versión 1.5; el envío de archivos (OPP) y la exploración del directorio telefónico fueron agregados en la versión 2.0; y el marcado por voz junto con el envío de contactos entre teléfonos lo fueron en la versión 2.2.</ref> Los cambios incluyeron:
Videollamada	Android soporta videollamada a través de Google Talk desde su versión HoneyComb.
Multitarea	Multitarea real de aplicaciones está disponible, es decir, las aplicaciones que no estén ejecutándose en primer plano reciben ciclos de reloj.
Características	La búsqueda en Google a través de voz está disponible como

basadas en voz	"Entrada de Búsqueda" desde la versión inicial del sistema.
Tethering	Android soporta tethering, que permite al teléfono ser usado como un punto de acceso alámbrico o inalámbrico (todos los teléfonos desde la versión 2.2, no oficial en teléfonos con versión 1.6 o inferiores mediante aplicaciones disponibles en Google Play. Para permitir a un PC usar la conexión de datos del móvil Android se podría requerir la instalación de software adicional.

CAPÍTULO 3. INTRODUCCIÓN DEL HARDWARE

3.1 ELECCION DE MATERIAL

La elección de material del presente trabajo se realizó por medio de una selección de diversos dispositivos, en los cuales se determinan todas las características indispensables como lo son peso y longitud, además de la potencia del motor y así evitar algún tipo de error en el momento de vuelo.

Es importante mencionar que la principal aportación del presente proyecto es el desarrollo de la comunicación así como en la interfaz gráfica para dispositivos móviles Android.

Se realizó un análisis para determinar qué elementos son necesarios para llevar a cabo éste proyecto. En la Figura 3.1 se muestran las partes que componen todo el Hardware de un cuadricóptero; en el presente capítulo se detallará cada uno de estos.

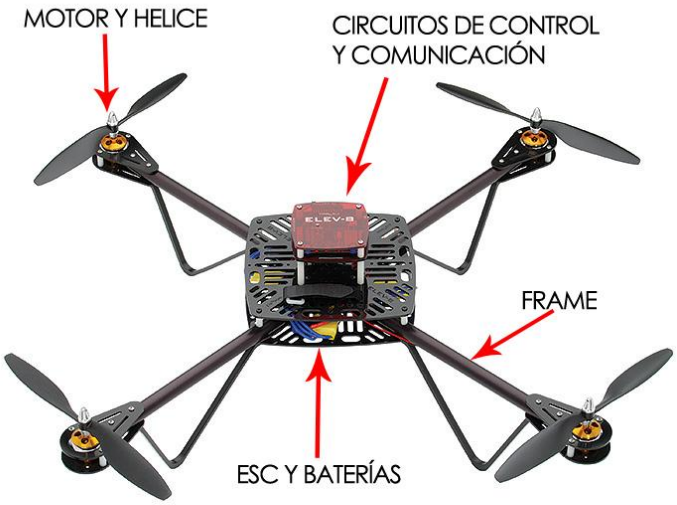


Figura 3.1 Componentes de un cuadricóptero

3.2 FRAME

El frame o armazón, es la estructura física que dará soporte a todos los demás componentes que conforman el cuadricóptero. Dicho elemento tendrá forma de cruz y será de la marca Hobby King modelo X525 V3 de fibra de vidrio y aluminio. (Figura 3.2).

Figura 3.2 Frame Hobby King X525 V3

Como medidas tendrá 60 centímetros de punta a punta, con un peso de 350 gramos. Su rigidez y ligereza son las razones principales por las cuales se usará este modelo, pero no se descarta la cualidad de absorber el "golpe de caída" gracias a sus patas que cuentan con un resorte de amortiguamiento.

Cada extremo del frame contará con un motor y el centro servirá como base para la placa de control y baterías.

3.3 MOTORES

Después del frame, los motores son lo más importante dentro de la construcción del cuadricóptero pues es gracias a la potencia de los mismos, la capacidad que tendrá elevarse y mantenerse en vuelo.

Como se vio en el apartado 2.6.2, hay dos tipos de motores brushless, en este caso se optará por usar motores brushles outrunner y será de la marca Turnigy modelo D23836/8 1100KV (Figura 3.3) y sus características se muestran en la Tabla 5:

Tabla 4 Características del motor Turnigy D23836/8 1100KV

Baterías:	2-4 celdas / 7.4-14.8V
RPM:	1100 RPM
Corriente Máxima:	18 A
Potencia Máxima:	336 W
Resistencia Interna:	0.107 Ohm
Peso:	70 g
Diámetro:	4 cm
Dimensiones:	2.8 x 3.6 cm
Tamaño De Hélices:	7.4V/ 11x7 – 14.8V/7x3
Empuje Máximo:	1130 g

Figura 3.3 Motor Turnigy D23836/8 1100KV

Dentro de las características de este motor se encuentra que tiene un máximo de 1100KV de revoluciones por minuto, es decir la relación que tiene el motor respecto a cada volt aplicado al mismo. Como se verá en el apartado 3.3, se optará por usar un arreglo de baterías que dará un total de 11.1 V; y es con este dato aplicado a la ecuación de las RPM el que se sabrá el número de giros completos de cada motor que son 12,210 RPM.

$$RPM = \ KV\ x\ V = 1100KV\ x\ 11.1V \qquad (3.1)$$

$$RPM = 12210$$

3.4 BATERÍAS

Dentro del aeromodelismo, la batería es un elemento demasiado importante tanto en sus capacidades como en las debilidades. En este caso se tomará el lado económico que representa a lo segundo, pues como se vio en el apartado 2.6.5 la gran variedad de baterías que existen facilitan la decisión de escoger un buen juego de baterías, pero limita el precio que pueden alcanzar las mismas.

Con fines de prueba y demostración, se optará por usar un par baterías LiPo de Turnigy 2200mAh de 3 celdas y 20C (Figura 3.4). Esto dará tiempo de carga suficiente para hacer las pruebas necesarias para determinar la estabilidad y funcionabilidad del sistema.

Figura 3.4 Batería Turnigy 2200mAh 3 celdas - 20C

Tabla 5 Características de la batería Turnigy 2200mAh 3 celdas – 20C

Capacidad Mínima:	2200mAh (Capacidad 100% real)
Configuración:	3S1P / 11.1V / 3 Celdas
Descarga Constante:	20C
Descarga Pico:	30C
Peso:	188g
Dimensiones:	103 x 33 x 24 mm
Enchufe De Carga:	JST-XH
Enchufe de Descarga:	XT60

Dentro de las características (Tabla 6) se encontrará que es una batería de 20C constante, es decir, su ratio de descarga respecto a la corriente. Para calcular la corriente total de cada batería se usa la siguiente ecuación:

$$I = mA\ x\ Descarga\ máxima = 2200mAh\ x\ 20$$

$$I = 44000\ mA$$

Debido a que se requerirá un tiempo de vuelo “aceptable” que será de 8 minutos o más, se agregará otra batería de las mismas características que será colocada en paralelo. Siguiendo las leyes básicas de la corriente, se encontrará que el voltaje se mantendrá en 11.1V pero la corriente se sumará, dando como resultado 88A.

Se tendrá un total de 88A que se repartirán entre los 4 motores, quedando cada uno con 22A. En el Capítulo 3.2 se vio que la corriente máxima de cada motor es de 18A, saliendo entonces nuestras baterías del rango aceptable para cada motor. Para solucionar este inconveniente, se determinará el flujo de corriente máximo

por medio de los ESC y la señal PWM enviada al mismo para así no dañar el equipo.

3.5 VARIADOR DE VELOCIDAD O ESC

Al usar motores brushless es necesario contar con un variador de velocidad también llamado “ESC”, el cual se encarga de dar paso a la corriente necesaria para el giro del motor a diferentes velocidades.

Cada motor contará con un ESC modelo Hobby King 30A ESC 3A UBEC (Figura 3.5) que se conectará a las baterías y a su vez a la placa controladora. Sus características se especifican en la Tabla 7.

Figura 3.5 Variador ESC Hobby King 30A UBEC

Tabla 6 Características del ESC Hobby King 30A UBEC

Corriente Constante:	30 A
Corriente De Arranque:	40 A
Batería:	2-4 Celdas LiPo / 5-12s NiXX
BEC:	5V / 3ª
Tipo de Motor:	Brushless sin sensores
Medidas:	54 x 26 11mm
Peso:	32 g

El ESC cuenta con dos cables de alimentación y pines de entrada para permitir el control gracias a la señal enviada por la placa de control. A su vez tiene tres cables de salida que van conectados al motor, son estos donde se envía la corriente necesaria según demandada por el controlador.

3.6 ARDUINO MEGA

La placa de control es por así decirlo el cerebro del cuadricóptero, y en este caso será una placa Arduino Mega 2560 (Figura 3.6). Dicho elemento cuenta con un procesador ATmega2560 con 54 entradas/salidas digitales de las cuales 15 pueden ser usadas como salidas de PWM. También tiene 16 entradas analógicas, 4 puertos serial, un oscilador a 16MHz, conexión USB, Jack de alimentación, puerto serial de programación ICSP y un botón Reset, sin contar puertos de alimentación y tierra. Para más información revisar el apartado Anexos – Arduino Mega.

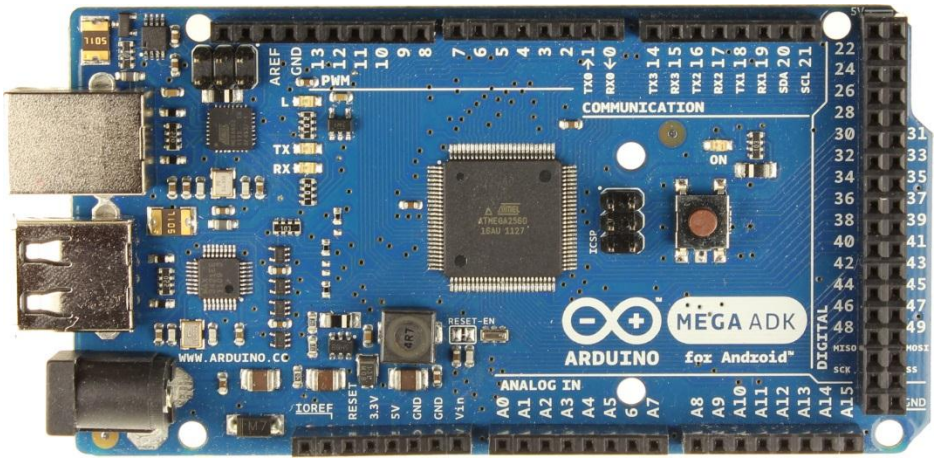


Figura 3.6 Arduino Mega

Para el cuadricóptero se hará uso de 4 pines digitales, un puerto de comunicación serial RX y un TX, los puertos TWI de comunicación SDA y SCL y los puertos de alimentación y tierra. La Tabla 8 muestra cómo es que se deberán de conectar cada uno de los elementos al controlador Arduino.

Tabla 7 Configuración para la conexión en los pines de Arduino

Elemento/Dispositivo	I/O	PIN DE ARDUINO
Bluetooth HC-06:	Transmisor TX	RX0
	Receptor RX	TX0
	GND	GND
	VCC	3.3V
MPU6050	SDA	SDA
	SCL	SCL
	GND	GND
	VCC	3.3V
MOTOR 1	SEÑAL	PIN 4
	GND	GND
MOTOR 1	SEÑAL	PIN 6
	GND	GND
MOTOR 1	SEÑAL	PIN 8
	GND	GND
MOTOR 1	SEÑAL	PIN 10
	GND	GND

Los pines digitales se deberán configurar como salidas de señal PWM, la cual será transmitida a los ESC que la interpretarán para permitir el paso de corriente para el giro de los motores según el programa de control lo demande. Los puertos de comunicación RX y TX servirán para colocar el receptor (Bluetooth) y esperar las

señales de manipulación para el cuadricóptero. Los puertos SDA y SCL serán usados para recibir los datos del acelerómetro, esto se introducirá en el algoritmo de control para determinar la estabilidad del sistema. Los pines de alimentación y tierra servirán de apoyo para el funcionamiento del Bluetooth y acelerómetro.

En el Capítulo 4 se detalla el sketch completo que se le programará al Arduino.

3.7 MODULO BLUETOOTH HC-06

El cuadricóptero será manipulado de forma inalámbrica, y para ello se usará un módulo Bluetooth HC-06 (Figura 3.7) que cuenta con pines de comunicación serial RX/TX que son completamente compatibles con la placa Arduino.

Figura 3.7 Módulo Bluetooth HC-06

En la Tabla 9 se pueden ver las especificaciones del módulo que cumple el protocolo de comunicación estándar v2.0+EDR del 2007.

Tabla 8 Características del módulo Bluetooth HC-06

Protocolo Bluetooth:	Bluetooth v2.0+EDR (2007)
Frecuencia:	2.4GHz ISM
Modulación:	GFSK
Potencia de Emisión:	<= 4dBm, Clase 2
Sensibilidad:	<=84dBm at 0.1% VER
Velocidad Asíncrona:	2.1Mbps(Máximo)/160kbps
Velocidad Síncrona:	1Mbps/1Mbps
Seguridad:	Autentificación y encriptación
Comunicación:	Puerto Serial Bluetooth
Alimentación:	+3.3VDC 50Ma
Temperatura de trabajo:	-20 a 75 Grados Celsius
Dimensiones:	26.9mm x 13mm x 2.2mm
Configuración de Fábrica:	Esclavo
	9600 Baudios
	N
	8
	1
	Pass 1234

3.8 GIROSCOPIO Y ACELEROMETRO MODULO MPU6050

El módulo MPU6050 es un pequeño pero potente elemento dentro del cuadricóptero. Combina un giroscopio de 3 ejes y un acelerómetro de 3 ejes por igual, creando un algoritmo completo para captar 9 ejes de movimiento.

El protocolo de comunicación de este dispositivo es I2C (Inter-Integrated Circuit), donde la velocidad de transmisión es de 100 Kbits/s y su principal característica es que utiliza dos líneas de comunicación para transmitir información: Una para datos (SDA) y otra para la señal de reloj (SCL).

Este protocolo de comunicación es común a la hora de conectar periféricos a un microcontrolador, es por ello que la placa Arduino cuenta con los pines necesarios para hacer uso de éste. Cabe mencionar que el modulo no soporta más de 3.3V por lo que hay que tener cuidado a la hora de conectar a la alimentación. A continuación se detallan las características principales de peste módulo:

- Salida digital por protocolo I2C para fusión de movimiento de 6 o 9 ejes en rotación matricial, quaternion, angulos de Euler o datos en concreto.
- Voltaje de entrada de 2.3 a 3.4V.
- Sensor angular de 3 ejes (giroscópio) con sensibilidad arriba de los 131LSB/dps y rango escalar de +-250, +-500, +-1000 y +-2000dps.
- Acelerometro de tres ejes con rango escalar de +-2g, +-4g, +-8 y +-16g.
- Ingeniería de complemento para Fusión de Movimiento por Proceso Digital de Movimiento (DMP)
- Algoritmos embebidos para lectura en tiempo real para calibración.
- Salida digital de sensor de temperatura

3.9 PESO

Como instancia final antes de comenzar a ensamblar las piezas, se deberá calcular el peso “final” del cuadricóptero con el fin de cerciorarse de que los motores cumplan al 100% su función. En la tabla 10 se ve cuál es la suma de todos los pesos de cada elemento del cuadricóptero.

Tabla 9 Peso final estimado del cuadricóptero

Elemento/Dispositivo	Peso/Unidad en Gramos	Cantidad	Sub Total en Gramos
Frame	350	-	350
Motor Brushless	70	4	280
Baterías	188	4	752
ESC	32	4	128
Placa Arduino Mega	16	-	16
Protoboard Arduino Mega +HC-06 +MPU6050	50	-	25
Batería para Arduino	8	-	8
Hélices	7	4	28
TOTAL			**1587**

Tomando en cuenta que el impulso de cada motor es de 1130 g, se deduce que el uso de los 4 motores es suficientemente fuerte como para elevarlo sin problema alguno.

3.10 INTEGRACIÓN DEL HARDWARE

Durante los siguientes apartados se detallará como es que se realizaron los arreglos necesarios para la integración del Hardware; conexión entre los motores, los ESC y las baterías, para pasar a la conexión de todos los elementos en el controlador Arduino.

El apartado 5.1 se muestra cual es el resultado final dentro de la integración del hardware.

3.10.1 MOTORES, BATERÍAS Y ESC

En la Figura 3.8 se muestra la forma en que se conectarán los motores, las baterías y los ESC.

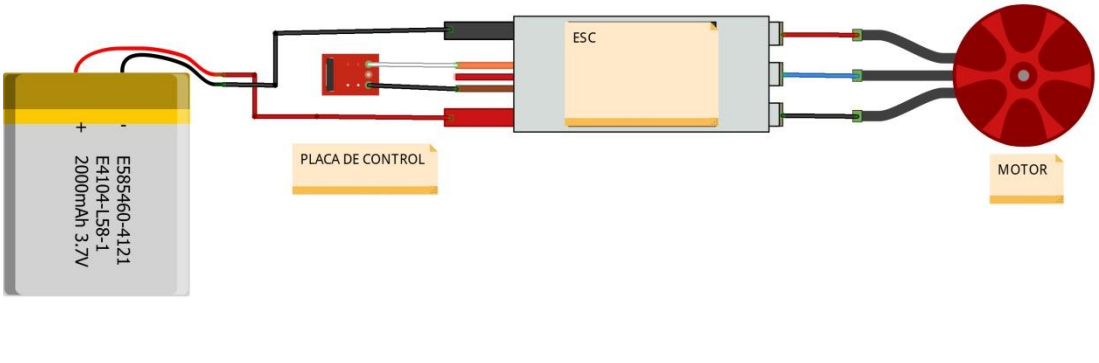


Figura 3.8 Modelo de conexiones entre Motor/ESC/Baterías/Controlador

- Las terminales de alimentación de los motores van a las terminales de salida de los ESC, positivo con positivo, negativo con negativo.
- Ensamblado lo anterior, las terminales de alimentación del ESC van conectados a la batería de la misma forma; positivo con positivo, negativo con negativo.
- Las tres terminales sobrantes van a la placa de control: Alimentación y tierra, y la señal de control. En el apartado 3.5 se detalla a que pines va cada uno de estos.

3.10.2 BLUETOOTH HC-06 Y ARDUINO

Como se vio en la Tabla 8 del apartado 3.5, el módulo Bluetooth HC-06 cuenta con 4 pines: Alimentación y tierra, RX y TX; y van conectados a pines específicos dentro del controlador Arduino.

En la Figura 3.9 se puede ver como se deberá hacer esta conexión. Aunque el módulo soporta hasta 5V de alimentación, se decidió alimentar a 3.3V por comodidad.

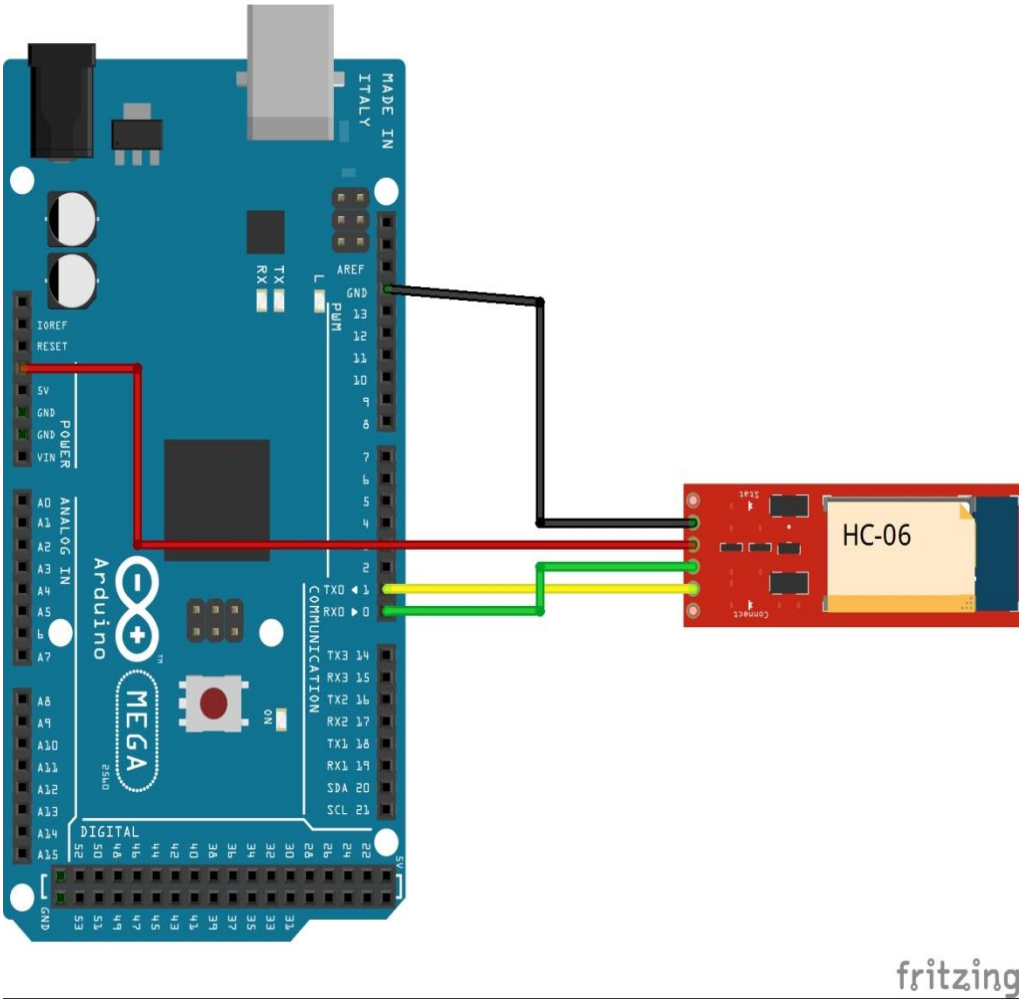


Figura 3.9 Modelo de conexión entre el módulo Bluetooth HC-06 y controlador Arduino

3.10.3 GIROSCOPIO Y ACELERÓMETRO MÓDULO MPU6050 Y ARDUINO

Al igual que el módulo Bluetooth HC-06, el módulo MPU6050 irá conectado a pines específicos y se deberá ser precavido al alimentar al módulo pues según sus especificaciones, no se debe alimentar a más de 3.3V. [11]

En la Figura 3.10 se ejemplifica como deberán hacerse las conexiones.

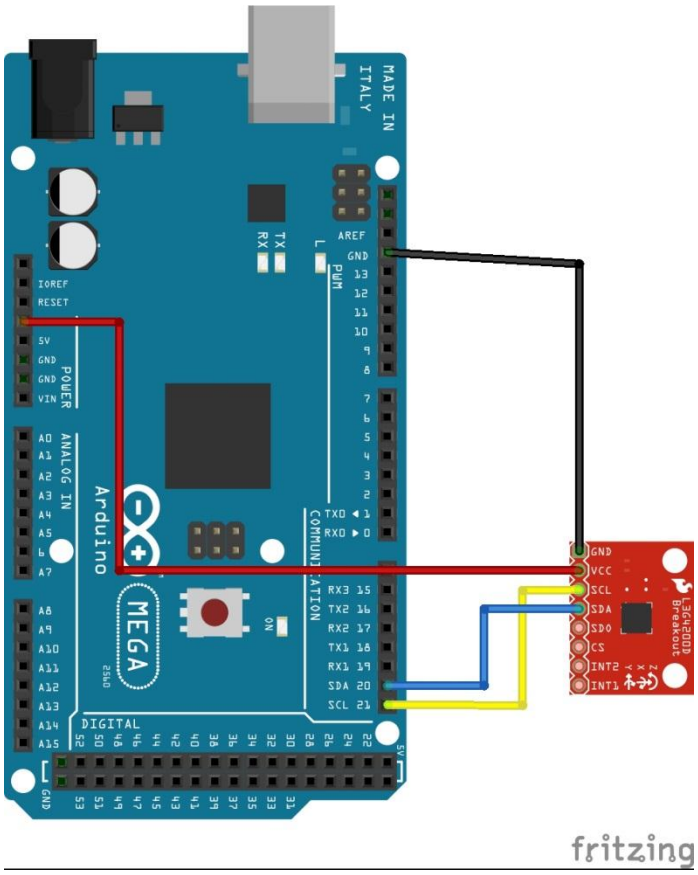


Figura 3.10 Modelo de conexión entre el módulo MPU6050 y el controlador Arduino

3.10.4 INTEGRACIÓN FINAL DEL HARDWARE EN EL CONTROLADOR ARDUINO

Hasta ahora las conexiones hechas en el controlador Arduino han sido de forma individual; en la Figura 3.11 se muestra un diagrama con todas las conexiones hechas según la tabla 8 del apartado 3.5.

[11]https://github.com/jrowberg/i2cdevlib/blob/master/Arduino/MPU6050/Examples/MPU6050_DMP6/MPU6050_DMP6.ino

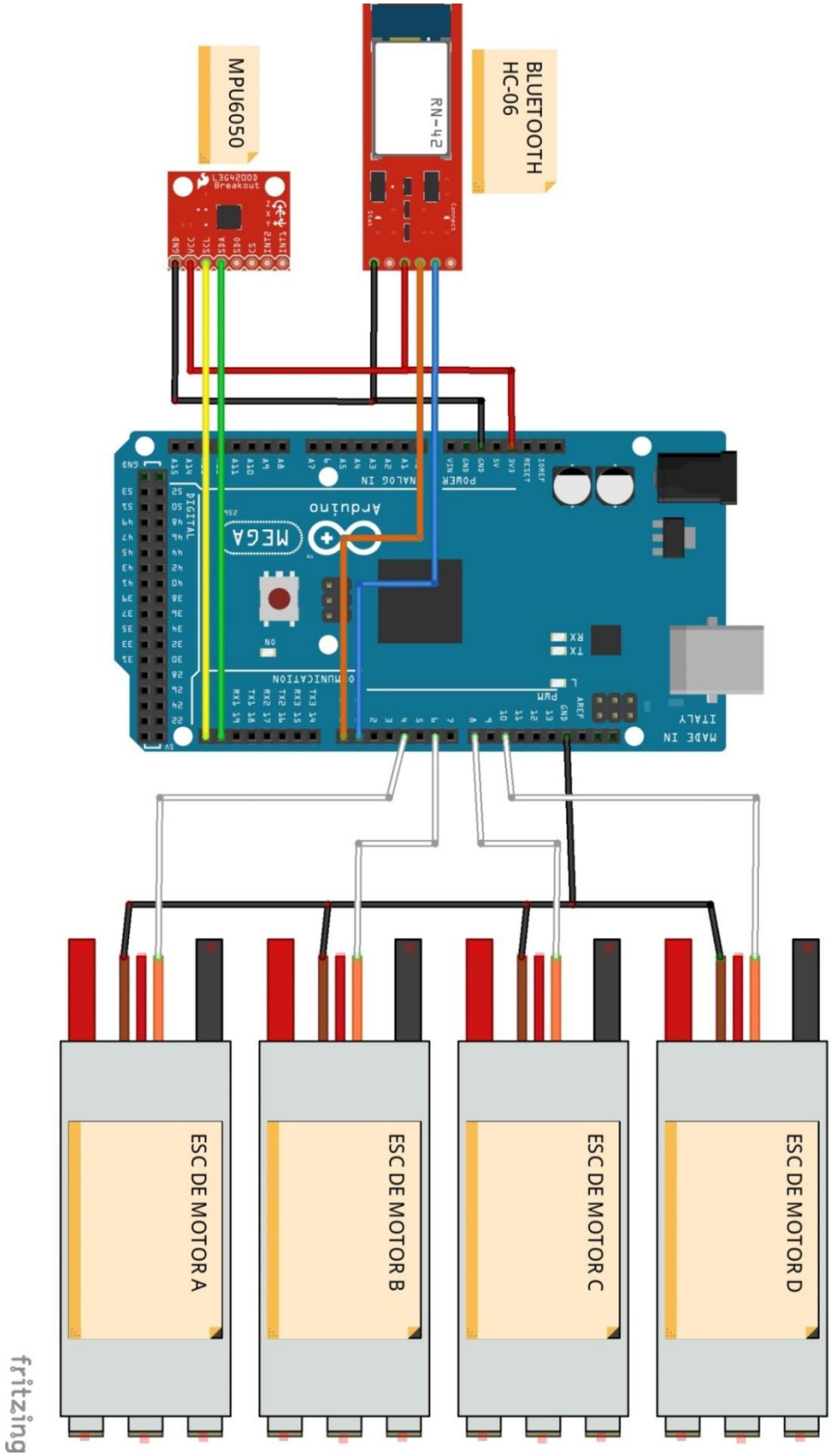


Figura 3.11 Diagrama de conexión final de todos los elementos al controlador Arduino

Para hacer posible todas las conexiones y ahorrar espacio dentro del control del cuadricóptero, se hará uso de una placa de prototipo para Arduino Mega (Arduino Mega Protoboard Shield) (Figura 3.12), la cual cuenta con acceso a todos los pines que Arduino Mega proporciona y a su vez facilita el soldar circuitos como si de una placa perforada se tratase.

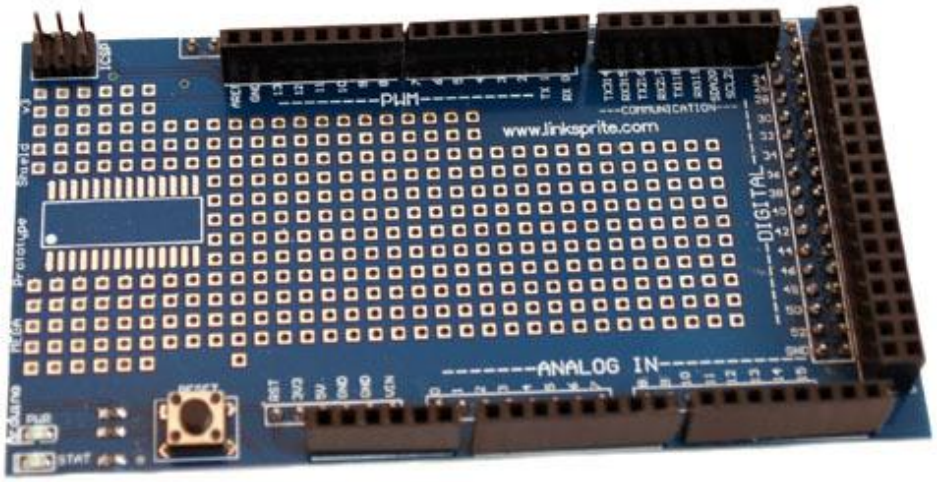


Figura 3.12 Arduino Mega Protoboard Shield

Es en la parte trasera de esta placa, donde se soldarán todos los elementos y sacarán pines de entrada y salida para hacer desmontable todo dentro de “centro de control”. Como dato curioso, el ESC que se usará no requiere que el pin de alimentación sea conectado por lo que solo se conectarán las baterías a los cables necesarios, el pin para la señal de PWM y la tierra.

CAPÍTULO 4. DESARROLLO DEL SOFTWARE

El desarrollo del Software constará de dos partes; por un lado la programación del controlador Arduino y por el otro la programación de la aplicación para dispositivos Android.

El controlador Arduino deberá ser capaz de “escuchar “él módulo Bluetooth para poder realizar todos los comandos previamente programados como lo son el encender y apagar el cuadricóptero de forma remota. En la Figura 4.1 se detalla un diagrama de flujo respecto a cómo deberá realizarse el programa de dicho controlador.

Una vez realizada la programación del controlador Arduino, se procederá a hacer lo mismo para la aplicación en Android. Dicha aplicación deberá permitir conectarse directamente con el cuadricóptero mediante el uso del Bluetooth y enviar de forma remota los comandos de control. Para más detalle, en la Figura 4.2 se muestra un diagrama de flujo de como deberá estar constituida ésta aplicación.

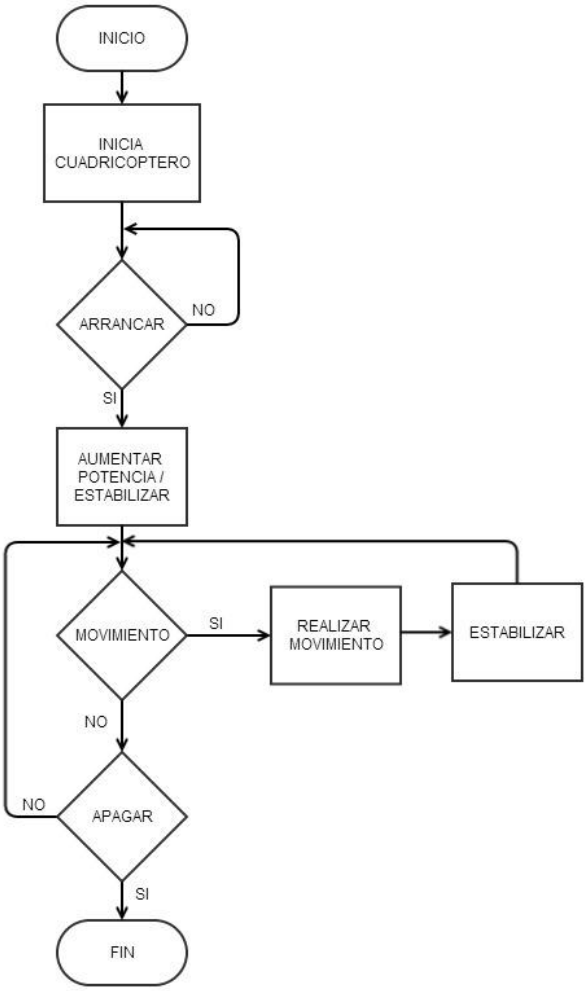


Figura 4.1 Diagrama de flujo general del programa para el controlador Arduino

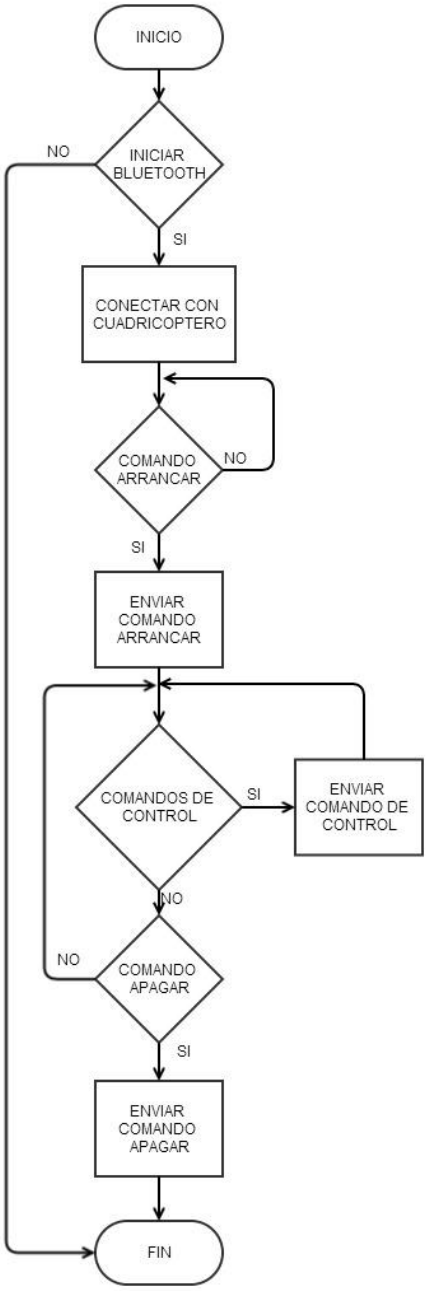


Figura 4.2 Diagrama de flujo general de la aplicación para dispositivos Android

A lo largo de los siguientes subtemas se verá más a fondo en que consiste cada uno de los programas desarrollados para cada plataforma mencionada previamente.

4.1 PROGRAMACIÓN DEL CONTROLADOR ARDUINO

Así como de forma general el desarrollo del Software se compone de dos partes (Arduino y Android), la parte de Arduino se subdivide en varias etapas que conforman todo el programa.

Como primer paso se deberá hacer la programación del módulo Bluetooth para que, por así decirlo, trabaje bajo los parámetros que se requieran. Después de ello se realizará una prueba de adquisición de datos del acelerómetro MPU6050 y del funcionamiento de los motores para finalmente incorporarlo a un programa final que será programado en el controlador Arduino.

Los apartados 4.1.1 al 4.1.3 detallan paso a paso como se realizará la programación de cada componente que conforma esta etapa que son: Bluetooth, acelerómetro y motores; para así finalizar con la integración de cada uno de ellos y conformar el programa final del controlador Arduino.

4.1.1 PROGRAMACIÓN DEL MÓDULO DE BLUETOOTH

Antes de realizar cualquier tipo de programación, se deberá configurar el módulo Bluetooth para que la taza de transferencia de datos sea la adecuada, así como asignarle un nombre propio y en su defecto, alguna contraseña que se crea necesaria para conectar con el mismo.

En la Figura 4.3 se aprecia la forma básica de un programa para Arduino. En el apartado de Creación de variables, se añadirán todos los elementos que se necesiten dentro del programa, aquí mismo se hacen la importación de librerías en caso de ser necesario.

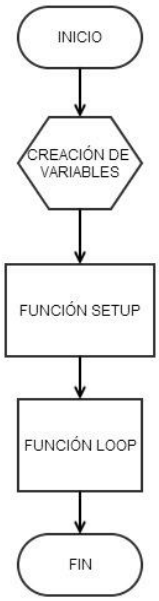


Figura 4.3 Diagrama de flujo de un sketch básico de Arduino

La función SETUP servirá para hacer los arreglos iniciales del controlador como son la declaración y estado de los pines. Esta función solo corre una vez al programar el controlador Arduino, en cambio el LOOP es la parte del programa que se está repitiendo infinitamente y es aquí donde los programas cobran importancia pues en esta función se hacen todas las operaciones que requiera el programador.

A cada programa realizado en Arduino se le conoce como sketch que no es otra cosa sino un archivo con todo el programa de extensión .ino.

Ahora que se conoce como es que está conformado un programa de Arduino, se procederá a la programación del módulo Bluetooth HC-06.

Para programar el módulo HC-06 se envían comandos por medio del puerto serial del controlador Arduino, esto se hace de tal forma como si se fuese a desplegar información en la consola del IDE de Arduino. En la Figura 4.4 se muestra un diagrama de flujo de cómo deberá constituirse el sketch.

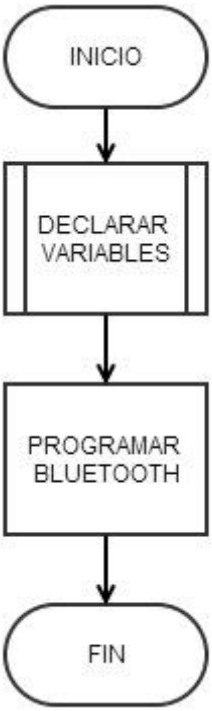


Figura 4.4 Diagrama de flujo para la programación del módulo Bluetooth HC-06

Para entender esta parte de la programación del módulo HC-06, se pasará a explicar cómo se constituyen los dos elementos del diagrama de flujo anterior:

- DECLARAR VARIABLES: Como variables a declarar se usará un nombre, los baudios a los que se comunicará el Bluetooth y una contraseña. Estas variables serán de tipo carácter y deberán ya tener asignado su valor.

- PROGRAMAR BLUETOOTH: Como se mencionó previamente, el módulo HC-06 se comunica por el puerto serial de Arduino, dicho protocolo también es usado para programar el controlador desde la computadora; si el módulo HC-06 está conectado al controlador Arduino mientras éste es programado, entrarán en conflicto y el compilador mandará un error.
- Para evitar esto, debe dar un espacio de tiempo para programar el controlador e inmediatamente colocar el módulo HC-06 para posteriormente ser programado por medio de comandos AT enviados por el puerto Serial. Todo esto se hará dentro de la función SETUP.
- Terminada la programación desde el SETUP, se correrá la función LOOP, a esta función se le agrega un destello al led que se encuentra en el pin 13 del controlador para indicar que el módulo HC-06 ya ha sido programado.

Para información más detallada, revisar el apartado de Anexos 2 donde se encuentra el programa con el que se programará el módulo HC-06

4.1.2 ADQUISICIÓN DE DATOS DEL ACELERÓMETRO

Como se explicó en el apartado 3.7, el acelerómetro y giroscopio módulo MPU6050 se comunica mediante el protocolo I2C y para ello se cuenta con la librería Wire de Arduino que es la encargada en hacer este tipo de comunicaciones dentro de la placa. Desafortunadamente, la librería es muy básica y se deberá implementar una librería externa llamada I2Cdevlib cuya licencia es libre y evita hacer una serie de arreglos para que el MPU6050 funcione correctamente.

Además de la librería I2Cdevlib, se hará uso de otra librería externa dedicada al MPU6050 la cual lleva el mismo nombre. Dicha librería cuenta con todo lo necesario para que en conjunto con las librerías Wire e I2Cdevlib se obtengan los datos del módulo.

En la Figura 4.5 se muestra un diagrama de flujo de un programa básico para la obtención de datos del MPU6050.

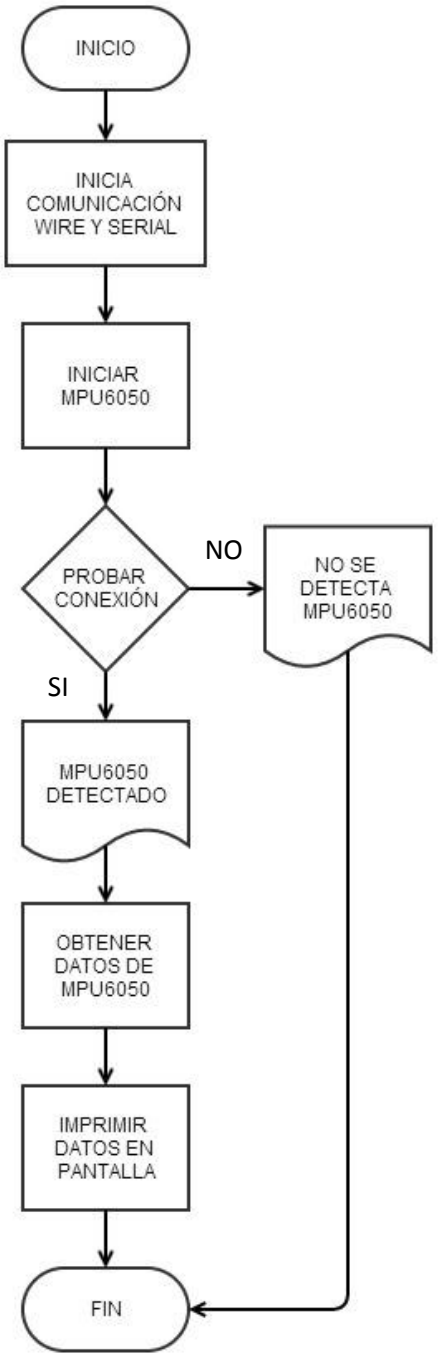


Figura 4.5 Diagrama de flujo para la adquisición de datos del módulo MPU6050

Ahora se procederá a explicar cada etapa del diagrama de flujo anterior:

- INICIA COMUNICACIÓN WIRE Y SERIAL: Previamente importadas las librerías necesarias y declarado las variables a usar, se procederá dentro del SETUP a iniciar la comunicación Wire y Serial.
- INICIAR MPU6050: Con el objeto creado de la librería MPU6050, se mandará a inicializar el acelerómetro dentro del SETUP.
- PROBAR CONEXIÓN: Aún dentro de la función SETUP, se hará una petición para probar la conexión con el módulo MPU6050. Como resultado dará un booleano que si es cierto continua el programa, de lo contrario termina e imprime en pantalla cual fuese el caso
- OBTENER DATOS DE MPU6050: En el LOOP se usará la función getMotion6 para obtener los datos del módulo y serán almacenados en sus respectivas variables.
- IMPRIMIR DATOS EN PANTALLA: Usando la función Serial.print se mostrará en la consola del IDE todos los datos obtenidos del módulo MPU6050

Para ver el código completo para la adquisición de datos visualizar el apartado de Anexos 3.

4.1.3 CONTROL DE LOS MOTORES

Dentro de Arduino controlar un motor brushless se puede comparar con un servomotor, con la diferencia de que el primero hace uso de un variador de velocidad o ESC para interpretar los anchos de pulso PWM que, en vez de determinar una posición, este determina la velocidad a la que gira el motor.

Conociendo lo anterior, se puede deducir que para controlar un motor brushless se hace exactamente igual que un servomotor; por ello se hará uso de la librería Servo proporcionada por Arduino para controlar de forma básica los motores.

La librería Servo dará un control “manual” de los motores para que estos puedan cambiar de velocidad, pero el verdadero control se hace integrando la parte del acelerómetro MPU6050 y una librería PID que proporcionará una señal de salida para cada motor y así determinar la estabilidad de todo el sistema que en este caso es un cuadricóptero.

En la Figura 4.6 se encuentra un diagrama de flujo de cómo deberá estar conformado el programa de la etapa de control de los motores.

Figura 4.6 Diagrama de flujo para el control de los motores

Desglosando el diagrama de flujo anterior, se pasará a detallar cada una de las 4 etapas que conforman la etapa de control de los motores:

- INICIANDO MOTORES: En esta parte, lo que se hará es declarar un arreglo de tipo Servo llamado motores que conformará cada motor por lo que tendrá una dimensión de 3. También se crea un objeto de la librería MPU6050 y otra de la librería PID_v1 con sus respectivos valores.
  Dentro la función SETUP se configurará el pin de cada motor y a su vez se declararán como salida los mismos. Se procederá con enviar señales PWM a cada motor para que lleguen a un estado de “arranque”, dichas señales deberán ir de 0 a 56; por comodidad esto se deberá hacer dentro de un ciclo for.

Estando los motores en estado de arranque, se enviarán nuevamente pulsos PWM pero ahora de 56 a 60.

- ADQUIRIR DATOS DE MPU6050: Este y los siguientes dos puntos se encuentran dentro de la función LOOP. Esta etapa se encuentra explicada en el apartado 4.1.2.
- ENVIAR DATOS A PID: En el paso anterior se obtuvieron datos de aceleración y de giro, en esta ocasión solo se usarán los de aceleración de los ejes X e Y. Dichos datos se envían a 4 PIDs distintos diferenciándose por mayor o menor a X y por mayor o menor a Y.
- ESTABILIZAR: En la etapa de ENVIAR DATOS A PID se obtienen una señal de salida, dicha señal se envía al motor designado dentro de la diferenciación echa anteriormente.

Para información más detallada, revisar el apartado de Anexos 4 donde se encuentra el programa básico para la estabilización de un cuadricóptero.

4.2 PROGRAMACIÓN DE LA APLICACIÓN PARA DISPOSITIVOS ANDROID

Hoy en día hay muchas formas de programar aplicaciones para dispositivos Android e incluso hay opciones que te permiten hacer una aplicación sin siquiera escribir un solo comando de programación como lo es App Inventor o Mobincube.

Para este proyecto se optó por desarrollar una aplicación de forma nativa, es decir, que se usará el kit de desarrollo proporcionado por Google el cual cuenta con todas las herramientas necesarias para crear aplicaciones de alto nivel. Actualmente este kit de desarrollo cuenta con una plataforma más visual y amigable para el desarrollador por lo que en ciertos puntos el desarrollo se facilita. Dejando de lado esto, la programación para dispositivos Android es meramente en lenguaje JAVA por lo que cualquier conocimiento respecto a este lenguaje es útil para la creación de aplicaciones

Este documento no intenta explicar cómo es que se conforma una aplicación en su etapa de desarrollo, por lo que solo se detallarán los elementos más importantes con los cuales se puede entender cómo es que se creó una aplicación para este proyecto.

A lo largo de los siguientes sub temas, se dará a conocer cómo es que se desarrolló una aplicación para controlar de forma remota el cuadricóptero.

4.2.1 DESARROLLO DE LA APLICACIÓN EN ANDROID

Hoy en día, dar un aspecto visual a una aplicación en Android es muy sencillo, la plataforma de desarrollo Eclipse cuenta con un intérprete que permite hacer esta parte de forma visual simplemente arrastrando y soltando los elementos deseados dentro de la plantilla.

En la Figura 4.7, se encuentra una captura de la única pantalla o Layout con la que contará la aplicación en su etapa de desarrollo. Dicho layout cuenta con 5 botones, cada botón será configurando más adelante para que funcione como

comando que será recibido en el controlador Arduino por medio del módulo HC-06.

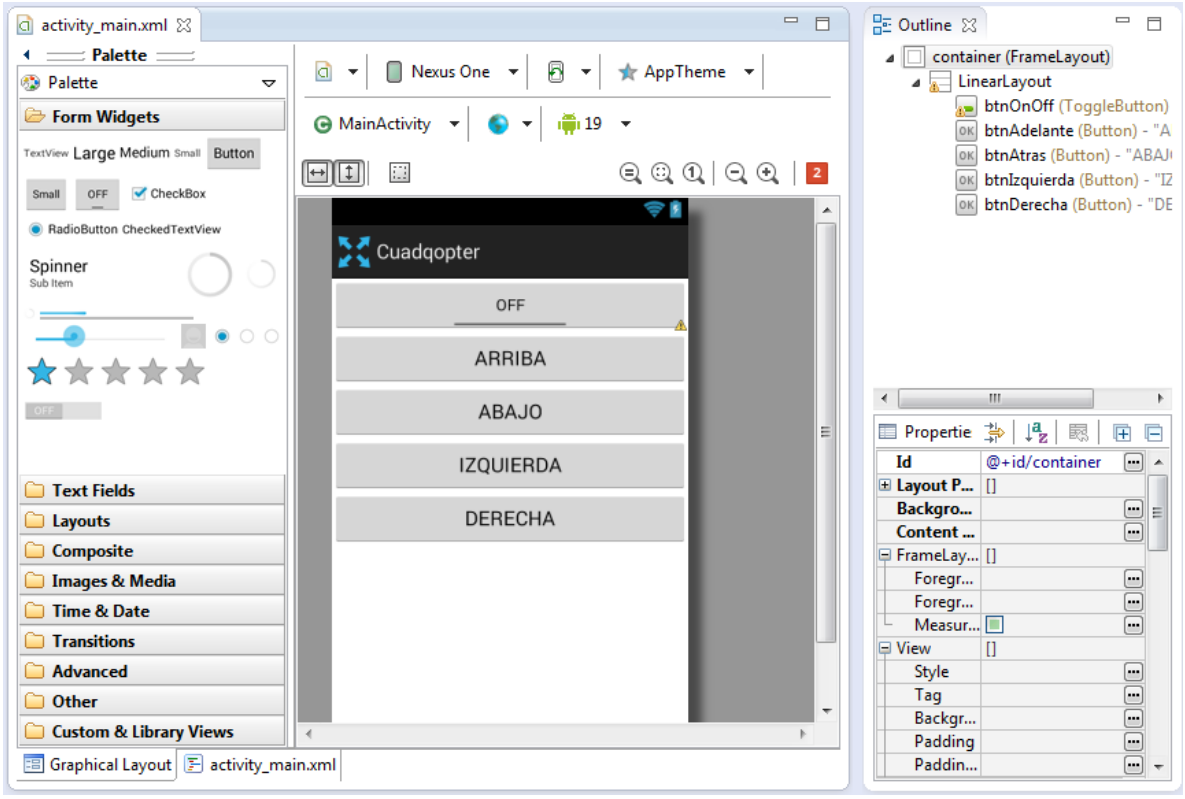


Figura 4.7 Entorno de desarrollo Eclipse para aplicaciones en Android

Editar imagen y describir el ambiente de trabajo

En el ángulo superior izquierdo se encuentra el nombre con el que se guardará el layout cuyo formato es en XML, este formato es un formato de texto que puede ser abierto con cualquier bloc de notas. Dentro se encuentra el código respectivo de la pantalla. Como se mencionó anteriormente, el entorno de desarrollo es relativamente amigable y evita escribir este código gracias a su desarrollo más visual.

En el Anexo 5 se puede ver cómo es que se conforma el código de este archivo XML.

4.2.2 PROGRAMACIÓN DE LA CONEXIÓN.

JAVA es un lenguaje orientado a objetos, por lo que la creación de clases es algo común dentro de los programas creados con este lenguaje. Es aquí donde la aplicación comienza a verse complicada pero se intentará detallar todo de tal forma de que sea entendible y de fácil comprensión para el lector.

Dentro del layout visto previamente existe un menú de configuración al que se le pueden añadir las opciones que uno requiera y darles las acciones por igual. Para

este caso se le añadirá una opción a la que se le llamará Conexión. A esta opción se le asignará un evento que llamará a una actividad llamada ConexionBT.

ConexionBT es un archivo JAVA que puede ser conocido como clase, esta clase se encarga de hacer todo el algoritmo de conexión. La clase cuenta con los elementos necesarios para iniciar o terminar una conexión por Bluetooth con el cuadricóptero, además de poder enviar y recibir datos del mismo.

Al ser una clase, no cuenta con un patrón para poder detallar un diagrama de flujo, por lo tanto se enlistarán las funciones con las que cuenta esta clase:

- ConexionBT: Es el constructor de la clase. Define los parámetros iniciales para hacer uso del Bluetooth
- setState: Asigna el estado en del Bluetooth a una variable en booleano.
- getState: Regresa el estado actual del Bluetooth.
- start: Cancela cualquier otra conexión que tenga establecido el Bluetooth y se prepara para conectarse con algún dispositivo.
- Connect: complemento de la función start.
- Conected: Envía un mensaje al módulo HC-06 para que establezca conexión y se mantengan “informados” uno del otro.
- Stop: detiene la transferencia de datos.
- Write: escribe información para ser enviada.
- ConectionFailed: envía mensaje de error al conectar.
- ConectionLost: Envía mensaje de conexión perdida.

Para más información revisar el programa completo en los Anexos 6

4.2.3 PROGRAMACIÓN DE LA ACTIVIDIDAD PRINCIPAL

La actividad principal o Main Activity, es la parte donde se lleva a cabo todo el control de la aplicación. Esta actividad va ligada con el layout principal y está encargada de hacer que los botones tengan funciones específicas, es por así decirlo, el alma de la aplicación.

El Main Activity es un archivo JAVA que cuenta con todos los comandos del layout principal. Puede ser considerado como una clase, una clase principal. Dentro de ella se encuentran varias funciones que serán detalladas aquí abajo para su comprensión:

- onCreate: función predefinida por Android.
- iniListeners: Función donde se declaran los Listeners para cada uno de los botones.
- SendMessage: Dentro de cada botón se manda a llamar la función sendMessage que se encarga de enviar el comando preestablecido por medio del Bluetooth.
- onStart: función predefinida por Android y es donde se configurará el Bluetooth.

- onDestroy: Función predefinida por Android, en caso de que se cierre la aplicación se manda a llamar esta función para terminar todo tipo de conexiones.
- ConfigBT: Pregunta y configura el uso del Bluetooth.
- IniREFGUI: crea los objetos respectivos a la interfaz gráfica, aquí se declara que es y cómo se llamará cada botón dentro del lenguaje java.

Para información más detallada, revisar el apartado de Anexo 7.

CAPÍTULO 5. RESULTADOS EXPERIMENTALES Y CONCLUSIONES

En los siguientes apartados se dará a conocer cuáles fueron los resultados finales a la hora de elaborar esta tesis.

En Resultados del Hardware se mostrará la problemática encontrada al usar dos baterías con los 4 motores al mismo tiempo y como es que se da solución al mismo. Resultados de Software detallará cual es el sketch final para Arduino y la interfaz para Android.

5.1 RESULTADOS DEL HARDWARE

Como se vio en el Capítulo 3, el primer paso a realizar fue el ensamble del frame el cual se puede ver ya ensamblado. Después se elaboró la conexión entre los motores y los ESC, la Figura 5.1 muestra cómo es que quedaron ensamblados.

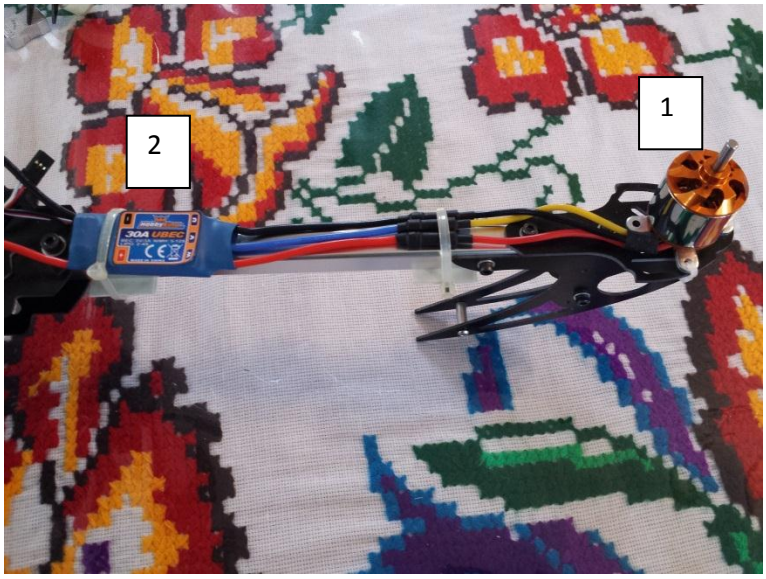


Figura 5.1 Conexión final entre los motores y los ESC

1. Motor Brushless
2. Variador ESC

Basados en la Tabla 8 del apartado 3.5 se procedió a crear el circuito de conexiones tal cual se mencionó en el apartado 3.9 en la Protoboard Shield de Arduino Mega. La Figura 5.2 muestra el resultado final con los módulos Bluetooth HC-06 y MPU6050 ya conectados y ensamblados con el Arduino Mega.

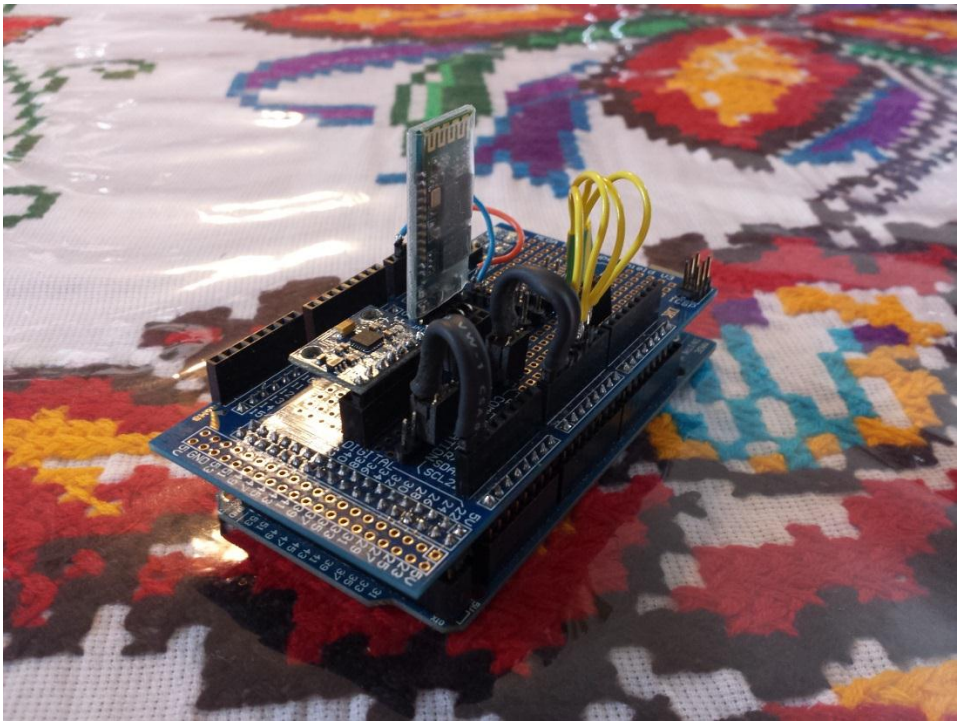


Figura 5.2 Protoboard Shield terminada y montada en el Arduino Mega

Durante las pruebas de control realizadas, se encontró un problema con el uso de las baterías; estas no suministraban la corriente suficiente para el uso de los motores y aunque se realizaron pruebas e investigaciones para dar solución a esto, se determinó que lo más fácil y viable sería usar un par de baterías más (una para cada motor) que además de solucionar el problema, se daría mayor tiempo de vuelo al cuadricóptero.

Solucionado lo anterior, se realizó el montaje completo en el frame del cuadricóptero, en la Figura 5.3 se muestra el resultado final.

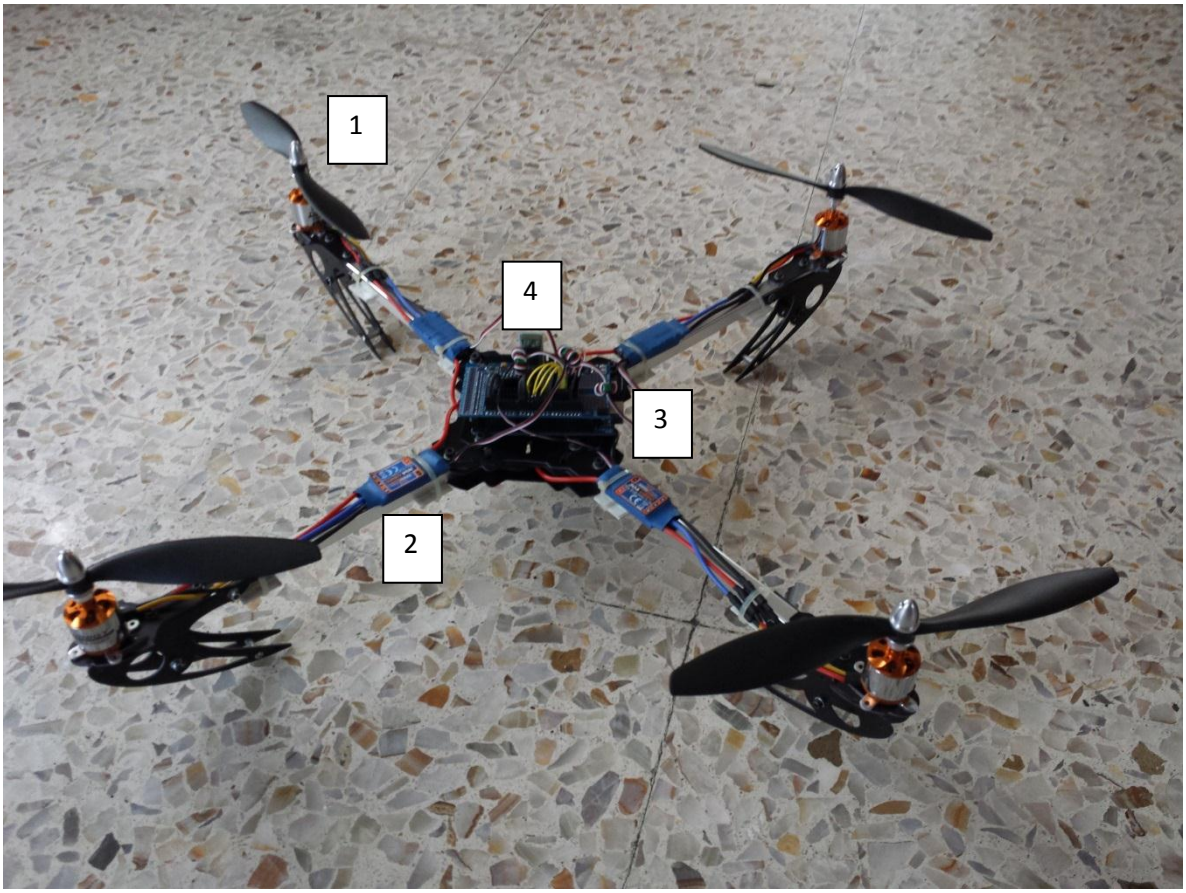


Figura 5.3 Resultado final del cuadricóptero

1. Motor Brushless con hélices
2. Variador ESC
3. Baterías montadas debajo del frame
4. Controlador Arduino con Protoboard Shield montado

5.2 RESULTADOS DEL SOFTWARE

Dentro del controlador Arduino se programó el sketch que se encuentra en el Anexo 8, que no es más que la integración de la adquisición de datos del módulo MPU6050 y el control de los motores.

La información proveniente del acelerómetro es procesada en 3 PID diferentes, estos PID se identifican de la siguiente manera:

- PitchReg: PID para calcular la aceleración de los motores para ir hacia adelante o hacia atrás.
- RollReg: PID para calcular la aceleración de los motores para ir hacia la izquierda o hacia la derecha.
- EstabReg: PID que se encarga de mantener estable al cuadricóptero mientras este esté en el aíre.

A su vez, el sketch permite recibir los comandos por medio del Bluetooth HC-06 que son:

- Encender o apagar el cuadricóptero
- Subir o bajar
- Ir hacia adelante o hacia atrás
- Ir hacia la izquierda o la derecha

En la aplicación para dispositivos Android no se realizaron cambios respecto al apartado 4.2 por lo que los Anexos mencionados en ese Capítulo son los usados como resultado final. En la Figura 5.4 se muestra una captura de como se ve la aplicación en un dispositivo con Android 4.4.

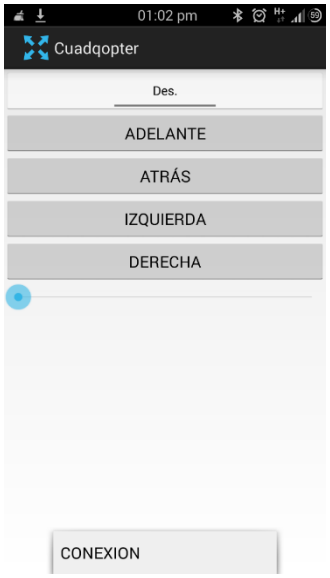


Figura 5.4 Pantalla principal de la aplicación

Al ser de cierta forma la aplicación un HMI para el cuadricóptero, los botones representan la forma en que se manipulará el cuadricóptero por lo tanto cada botón mandará un comando predefinido para realizar el movimiento deseado:

- Para comenzar a enviar datos es necesario conectarse con el módulo Bluetooth HC-06, para ello es necesario presionar el botón Conexión (Figura 5.5) el cual se encargará de hacer un escaneo y en caso de que el cuadricóptero esté a su alcance, realizar el enlace correspondiente.

Figura 5.5 Botón CONEXIÓN

- Realizado lo anterior, se espera el comando de encendido. Dicho comando se envía presionando el botón que se encuentra de cabecera, este botón es un botón de dos estados: Activado y Desactivado y lo que hace es activar los botones siguientes y enviar el comando de encendido para el cuadricóptero (figura 5.6).

Figura 5.6 Botón Act/Des

Una vez activados los botones de movimiento y la barra de aceleración, se podrá hacer uso de ellos para manipular por completo el estado del cuadricóptero.

Los motores se identifican como Motor A, Motor B, Motor C y Motor D, donde Motor A y C conforman el eje X y los motores B y D el eje Y. A y B representan los motores delantero y trasero respectivamente; y B y D los motores izquierdo y derecho.

Mencionado lo anterior se procede a explicar cómo funciona cada uno de los botones de movimientos.

Los botones ADELANTE y ATRÁS responden al movimiento Pitch mencionado en el apartado 2.3.2.

- Al presionar Adelante (figura 5.7), se manda una señal de control a un PID dentro del controlador Arduino el cual fue llamado PitchReg; Este arreglo lo que hace es aumentar la potencia del Motor C y disminuir la del Motor A para que el desplazamiento sea hacia adelante.

Figura 5.7 Botón ADELANTE

- Por el contrario, al presionar el botón ATRÁS (figura 5.8), se manda una señal de control de valor negativo para que el Motor A aumente su potencia y el Motor C la disminuya, mostrando así el movimiento hacia atrás o en reversa.

Figura 5.8 Botón ATRÁS

Los botones IZQUIERDA y DERECHA responden al movimiento Roll que es en sí un movimiento en lateral. Ambos botones envían una señal de control a un PID llamado RollReg.

- Al presionar el botón IZQUIERDA (figura 5.9) se envía el comando para que el PID realice los cálculos necesarios para determinar el aumento de la velocidad para el Motor D y la disminución del motor B para realizar un movimiento lateral que va de izquierda a derecha.

Figura 5.9 Botón IZQUIERDA

- Al presionar el botón DERECHA (figura 5.10), la señal de control calculada determinará la velocidad a la que girará el Motor B y el Motor D para que el movimiento lateral sea de izquierda a derecha.

Figura 5.10 Botón DERECHA

Debajo de los botones de movimientos se encuentra una barra de desplazamiento, dicha barra se encarga de enviar la señal media a la cual girarán los motores. Se dice media pues recordemos que al girar los motores a diferentes velocidades debido a la estabilización, esta velocidad sirve para elevar o descender el cuadricóptero (figura 5.11).

Figura 5.11 Variador de velocidad

5.3 CONCLUSIONES

Como se vio en el capítulo de la introducción, hoy en día, la tecnología avanza firmemente y se puede encontrar que el uso de cuadricópteros comienza a dejar de ser exclusivo para el aeromodelismo.

Siendo este un proyecto ambicioso, se estima que los avances realizados en esta tecnología tienen un gran alcance. Algunos ejemplos en donde se puede ver usada estos dispositivos es en el cine para hacer tomas aéreas de excelente calidad, en el área de la construcción se puede usar para reconocer terrenos, en seguridad social, específicamente en sismos o incendios se puede sobrevolar ciertas áreas para determinar la magnitud de los daños o puntos de peligro.

Aunque los ejemplos anteriormente mencionados requieren más Hardware como lo es una cámara y un GPS , se pueden mencionar algunos ejemplos un poco más bélicos como el anexo de algún arma de fuego para no comprometer a los equipos de asalto de ciertas organizaciones de seguridad.

En el presente trabajo se implementó satisfactoriamente la comunicación inalámbrica entre un controlador Arduino y un dispositivo móvil con sistema operativo Android aplicado al control de movimiento de un cuadricóptero.

Siendo que tanto el objetivo general y los objetivos específicos, se cumplieron de forma satisfactoria se cree que el dispositivo en general puede llegar a tener una gran aceptación para el público en general.

En el capítulo dos, se describió los elementos a utilizar, sus características y sus principios de operación, con esto se puede concluir que para la finalización de este trabajo, es necesario especificar claramente con respecto a lo que se desea lograr los elementos necesarios para llevarlo adecuadamente.

En un principio la idea original era implementar la comunicación vía wi-fi pero por falta de presupuesto se decidió que la mejor opción para terminar el trabajo y que este cumpliera con el objetivo de comunicación era necesario el bluetooth pues daría una idea general de en un futuro poder utilizar otro tipo de sistema para obtener una mayor longitud de alcance.

En el capítulo tres, se determina la elección de materiales conforme a sus características y lo que se pueda aportar a la realización del proyecto. Se decidió por un bastidor ya certificado puesto que la variable de peso pondría en dificultad el vuelo del dispositivo mencionado, así también se optó por motores brushless de mayor potencia para que tuviera una mejor estabilidad y que el peso adicional de los otros elementos no interfiriera con su desempeño.

Es importante mencionar que durante el desarrollo del proyecto se observó que el arreglo de las baterías no fue suficiente para que los 4 motores funcionaran de forma correcta, es por ello que se optó por dedicar una batería a cada motor que,

además de solucionar el problema, añade más tiempo de vuelo para el cuadricóptero.

Respecto a la comunicación por Bluetooth, se observó que el tiempo de respuesta entre el controlador Arduino y Android puede ser algo lenta; aunque no se indagó profundamente en el tema de la comunicación, se determinó que para casos prácticos, usar un RF ayudaría a cubrir ciertas limitaciones que tiene la comunicación por Bluetooth.

En el capítulo 4 se observa que la programación básica es necesaria para la desarrollo del dispositivo, se necesitan de software específicos para que el proyecto cumpla con los requisitos definidos.

El software que se maneja es amigable, requiere conocimientos básicos de C y C++, por lo que su utilización es relativamente sencilla.

Se requirió implementar en la programación cuatro PID's para los diferentes tipos de movimiento que se pueda manejar en el Drone, así también para que cumple cabalmente con las limitaciones de interferencia que se puedan presentar en su desempeño de vuelo.

En el capítulo cinco sé observa que los resultados fueron satisfactorios, aún con pequeños inconvenientes como la limitada alimentación que suministraba las baterías a los motores que se solucionó con la adquisición de más baterías para que cumplieran con su función adecuada y así darle un plus extra al obtener mayor tiempo de vuelo.

Se cree que al finalizar este proyecto satisfactoriamente se le puede dar un mayor impulso a vehículos aéreos no tripulados o sus siglas en ingles UAV para que se desempeñen con las aplicaciones mencionadas.

Como se puede ver, este proyecto no se limita a un uso didáctico sino también comercial e industrial. Las proyecciones a futuro son prometedoras siendo que la tecnología usada puede ser mejorada en todos los ámbitos.

No se descarta que se le puedan hacer mejoras, pues es una infinidad de aplicaciones que se le pueden dar a este tipo de dispositivos, por falta de tiempo y recursos se hizo de una forma sencilla pero consiente de todas las virtudes que posee y que pueda adquirir.

Respecto a la aplicación desarrollada para dispositivos Android, al ser un proyecto escolar se cuenta con un diseño bastante básico pero no se descarta que para futuras mejoras de la aplicación, se modifique parcial o totalmente para una interacción más amigable con el usuario.

ANEXOS

ANEXO 1. ARDUINO MEGA

Microcontrolador	ATmega1280
Voltaje de funcionamiento	5V
Voltaje de entrada (recomendado)	7-12V
Voltaje de entrada (limite)	6-20V
Pines E/S digitales	54 (14 proporcionan salida PWM)
Pines de entrada analógica	16
Intensidad por pin	40 mA
Intensidad en pin 3.3V	50 mA
Memoria Flash	128 KB de las cuales 4 KB las usa el gestor de arranque(bootloader)
SRAM	8 KB
EEPROM	4 KB
Velocidad de reloj	16 MHz

Alimentación

El Arduino Mega puede ser alimentado vía la conexión USB o con una fuente de alimentación externa. El origen de la alimentación se selecciona automáticamente.

Las fuentes de alimentación externas (no-USB) pueden ser tanto un transformador o una batería. El transformador se puede conectar usando un conector macho de 2.1mm con centro positivo en el conector hembra de la placa. Los cables de la batería pueden conectarse a los pines GND y Vin en los conectores de alimentación (POWER)

La placa puede trabajar con una alimentación externa de entre 6 a 20 voltios. Si el voltaje suministrado es inferior a 7V el pin de 5V puede proporcionar menos de 5 Voltios y la placa puede volverse inestable, si se usan más de 12V los reguladores de voltaje se pueden sobrecalentar y dañar la placa. El rango recomendado es de 7 a 12 voltios.

Los pines de alimentación son los siguientes:

VIN. La entrada de voltaje a la placa Arduino cando se está usando una fuente externa de alimentación (en opuesto a los 5 voltios de la conexión USB). Se puede proporcionar voltaje a través de este pin, o, si se está alimentado a través de la conexión de 2.1mm, acceder a ella a través de esta pin.

5V. La fuente de voltaje estabilizado usado para alimentar el microcontrolador y otros componentes de la placa. Esta puede provenir de VIN a través de un regulador integrado en la placa, o proporcionada directamente por el USB u otra fuente estabilizada de 5V.

3V3. Una fuente de voltaje a 3.3 voltios generada en el chip FTDI integrado en la placa. La corriente máxima soportada 50mA.

GND. Pines de toma de tierra.

Memoria

El ATmega1280 tiene 128KB de memoria flash para almacenar código (4KB son usados para el arranque del sistema (bootloader).El ATmega1280 tiene 8 KB de memoria SRAM. El ATmega1280 tiene 4KB de EEPROM, que puede a la cual se puede acceder para leer o escribir con la [Reference/EEPROM |librería EEPROM]].

Entradas y Salidas

Cada uno de los 54 pines digitales en el Arduino pueden utilizarse como entradas o como salidas usando las funciones pinMode (), digitalWrite(), y digitalRead() . Las E/S operan a 5 voltios. Cada pin puede proporcionar o recibir una intensidad máxima de 40mA y tiene una resistencia interna (desconectada por defecto) de 20-50kOhms. Además, algunos pines tienen funciones especializadas:

Serie: 0 (RX) y 1 (TX), Serie 1: 19 (RX) y 18 (TX); Serie 2: 17 (RX) y 16 (TX); Serie 3: 15 (RX) y 14 (TX). Usado para recibir (RX) transmitir (TX) datos a través de puerto serie TTL. Los pines Serie: 0 (RX) y 1 (TX) están conectados a los pines correspondientes del chip FTDI USB-to-TTL.

Interrupciones Externas: 2 (interrupción 0), 3 (interrupción 1), 18 (interrupción 5), 19 (interrupción 4), 20 (interrupción 3), y 21 (interrupción 2). Estos pines se pueden configurar para lanzar una interrupción en un valor LOW (0V), en flancos de subida o bajada (cambio de LOW a HIGH (5V) o viceversa), o en cambios de valor. Ver la función attachInterrupt () para as detalles.

PWM: de 0 a 13. Proporciona una salida PWM (Pulse Wave Modulation, modulación de onda por pulsos) de 8 bits de resolución (valores de 0 a 255) a través de la función analogWrite ().

SPI: 50 (SS), 51 (MOSI), 52 (MISO), 53 (SCK). Estos pines proporcionan comunicación SPI, que a pesar de que el hardware la proporcione actualmente no está incluido en el lenguaje Arduino.

LED: 13. Hay un LED integrado en la placa conectado al pin digital 13, cuando este pin tiene un valor HIGH(5V) el LED se enciende y cuando este tiene un valor LOW(0V) este se apaga.

El Mega tiene 16 entradas analógicas, y cada una de ellas proporciona una resolución de 10bits (1024 valores). Por defecto se mide de tierra a 5 voltios, aunque es posible cambiar la cota superior de este rango usando el pin AREF y la función analogReference(). Además algunos pines tienen funciones especializadas:

I2C: 20 (SDA) y 21 (SCL). Soporte del protocolo de comunicaciones I2C (TWI) usando la librería Wire.

Hay unos otros pines en la placa:

AREF. Voltaje de referencia para la entradas analógicas Usado por analogReference().

Reset. Suministrar un valor LOW (0V) para reiniciar el microcontrolador. Típicamente usado para añadir un botón de reset a los shields que no dejan acceso a este botón en la placa.

Comunicaciones

EL Arduino Mega facilita en varios aspectos la comunicación con el ordenador, otro Arduino u otros microcontroladores. El ATmega1280 proporciona cuatro puertos de comunicación vía serie UART TTL (5V). Un chip FTDI FT232RL integrado en la placa canaliza esta comunicación serie a traes del USB y los drivers FTDI (incluidos en el software de Arduino) proporcionan un puerto serie virtual en el ordenador. El software incluye un monitor de puerto serie que permite enviar y recibir información textual de la placa Arduino. Los LEDS RX y TX de la placa parpadearan cuando se detecte comunicación transmitida través del chip FTDI y la conexión USB (no parpadearan si se usa la comunicación serie a través de los pines 0 y 1).

La librería SoftwareSerial permite comunicación serie por cualquier par de pines digitales de él Mega.

El ATmega1280 también soporta la comunicación I2C (TWI) y SPI. El software de Arduino incluye una librería Wire para simplificar el uso el bus I2C, ver The la documentación para más detalles. Para el uso de la comunicación SPI, mira en la hoja de especificaciones (datasheet) del ATmega1280.

s pinMode(), digitalWrite(), y digitalRead() . Las E/S operan a 5 voltios. Cada pin puede proporcionar o recibir una intensidad máxima de 40mA y tiene una resistencia interna (desconectada por defecto) de 20-50kOhms. Además, algunos pines tienen funciones especializadas:

Serie: 0 (RX) y 1 (TX), Serie 1: 19 (RX) y 18 (TX); Serie 2: 17 (RX) y 16 (TX); Serie 3: 15 (RX) y 14 (TX). Usado para recibir (RX) transmitir (TX) datos a través de

puerto serie TTL. Los pines Serie: 0 (RX) y 1 (TX) estan conectados a los pines correspondientes del chip FTDI USB-to-TTL.

Interrupciones Externas: 2 (interrupción 0), 3 (interrupción 1), 18 (interrupción 5), 19 (interrupción 4), 20 (interrupción 3), y 21 (interrupción 2).. Estos pines se pueden configurar para lanzar una interrupción en un valor LOW(0V), en flancos de subida o bajada (cambio de LOW a HIGH(5V) o vicevers Esquemas y Diseños

ANEXO 2. PROGRAMA PARA CONFIGURAR EL MÓDULO BLUETOOTH HC-06


```
char NOMBRE[10] = "ESIMECuad";
char BPS        = '4'; //1 =1200, 2=2400, 3=4800, 4 =9600.....
char PASS [10]  = "";

void setup ()
{
  Serial.begin (9600);
  pinMode (13,OUTPUT);
  digitalWrite(13,HIGH);
  delay(10000);
  digitalWrite (13,LOW);

  Serial.print("AT");
  delay(1000);

  Serial.print("AT+NAME");
  Serial.print(NOMBRE);
  delay(1000);

  Serial.print("AT+BAUD");
  Serial.print(BPS);
  delay(1000);

  Serial.print("AT+PIN");
  Serial.print(PASS);
  delay(1000);
}

void loop()
{
  digitalWrite (13, !digitalRead(13));
  delay(500);
}
```

ANEXO 3. PROGRAMA PARA LA ADQUISICIÓN DE DATOS DEL MODULO MPU6050


```
#include "Wire.h"
#include "I2Cdev.h"
#include "MPU6050.h"

MPU6050 mpu;

int16_t ax, ay, az;
int16_t gx, gy, gz;

void setup()
{
   Wire.begin();
   Serial.begin(9600);

   Serial.println("Iniciando MPU");
   mpu.initialize();
   Serial.println(mpu.testConnection() ? "Conectado" : "Error al conectar, no se detecta dispositivo");
}

void loop()
{
   mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
    Serial.println("Aceleración:                    Giro:");
   Serial.print("X: ");
Serial.print(ax);
Serial.print("Y: ");
Serial.print(ay);
Serial.print("Z: ");
Serial.print(az);
Serial.print("X: ");
Serial.print(gx);
Serial.print("Y: ");
Serial.print(gy);
Serial.print("Z: ");
Serial.print(gz);     }
```

ANEXO 4. PROGRAMA PARA EL CONTROL DE LOS MOTORES


```
#include <Servo.h>

#define MAXIMOPWM 100

#define MINIMOPWM 0

#define  PASO 1

#define  BAUD 9600

int pulsoMotor;

int ordenTeclado=0;

Servo motora,motorb,motorc,motord;

byte recibiendoByte ;

 boolean iniciado = false;

void setup()

{

  Serial.begin(BAUD);

  motora.attach(4);

  motorb.attach(6);

  motorc.attach(8);

  motord.attach(10);

    Serial.println(" Comienzo del test");

   Serial.println (" Pulsar 'A' para arrancar  \n Cuando escuche el pitido de confirmación");

   while ( iniciado==false ){

          motora.write(0);

          motorb.write(0);

          motorc.write(0);

          motord.write(0);

          recibiendoByte = Serial.read();

          if (recibiendoByte == 65 || recibiendoByte ==97) {
```

```
            iniciado=true;
          }
}
Serial.println("inicio del loop principal  \n Para subir controlar velocidad pulse \n 'A' para subir \n    'Z' para bajar \n    'S' para terminar Stop \n");
}
void loop(){
   ordenTeclado =OrdenSubirBajar ();
  if (ordenTeclado != 0) {
    pulsoMotor = pulsoMotor + ordenTeclado;
    pulsoMotor= constrain( pulsoMotor , MINIMOPWM, MAXIMOPWM); //
         motora.write(pulsoMotor);
         motorb.write(pulsoMotor);
         motorc.write(pulsoMotor);
         motord.write(pulsoMotor);
    Serial.print("Velocidad del pulso-->  ");
    Serial.println (pulsoMotor);
  }
  delay (150);   //delay para no colapsar
}
int OrdenSubirBajar (){
   int  orden=0;
  Serial.flush()
  if (Serial.available() > 0) {
     recibiendoByte = Serial.read();
    if (recibiendoByte == 65 || recibiendoByte ==97) {      // A o A   Mayusculas o minusculas
       Serial.println( " SUBIR");
```

```
        orden = PASO;
      }
    if (recibiendoByte == 90 || recibiendoByte ==122)
      Serial.println( " BAJAR");
      orden = -PASO;
    }
    if (recibiendoByte == 83  || recibiendoByte == 115){
      Serial.println( " Stop!!");
      orden = -(pulsoMotor- MINIMOPWM);
    }
  }
  return (orden);
}
```

ANEXO 5. XML DE LA PANTALLA DE LA APLICACIÓN DE ANDROID

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.esime.cuadqopter.MainActivity"
    tools:ignore="MergeRootFrame" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >

        <ToggleButton
            android:id="@+id/btnOnOff"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="ToggleButton" />

        <Button
            android:id="@+id/btnAdelante"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/ARRIBA" />

        <Button
            android:id="@+id/btnAtras"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/ABAJO" />

        <Button
            android:id="@+id/btnIzquierda"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/IZQUIERDA" />

        <Button
            android:id="@+id/btnDerecha"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/DERECHA" />

    </LinearLayout>
</FrameLayout>
```

ANEXO 6. CLASE CONEXIÓNBT EN JAVA


```
package com.esime.cuadqopter;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.UUID;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothServerSocket;
import android.bluetooth.BluetoothSocket;
import android.content.Context;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;

public class ConexionBT {

    private static final String TAG = "Servicio_Bluetooth";
    private static final boolean D = true;
    private static final String NAME = "BluetoothDEB";
    private static final UUID MY_UUID = UUID.fromString("00001101-0000-1000-
8000-00805F9B34FB");
    private final BluetoothAdapter AdaptadorBT;
    private final Handler mHandler;
    private AcceptThread HebraDeAceptacion;
    private ConnectThread HiloDeConexion;
    private ConnectedThread HiloConetado;
    private int EstadoActual;

    public static final int STATE_NONE = 0;
    public static final int STATE_LISTEN = 1;
    public static final int STATE_CONNECTING = 2;
    public static final int STATE_CONNECTED = 3;


    public ConexionBT(Context context, Handler handler) {
          AdaptadorBT = BluetoothAdapter.getDefaultAdapter();
          EstadoActual = STATE_NONE;
        mHandler = handler;
    }

    private synchronized void setState(int estado) {
        EstadoActual = estado;
```

```
        mHandler.obtainMessage(MainActivity.Mensaje_Estado_Cambiado, estado, -
1).sendToTarget();
    }

    public synchronized int getState() {
        return EstadoActual;
    }

    public synchronized void start() {
        if (D) Log.e(TAG, "start");

        if (HiloDeConexion != null) {HiloDeConexion.cancel(); HiloDeConexion = null;}

        if (HiloConetado != null) {HiloConetado.cancel(); HiloConetado = null;}

        if (HebraDeAceptacion == null) {
          HebraDeAceptacion = new AcceptThread();
          HebraDeAceptacion.start();
        }
        setState(STATE_LISTEN);
    }

    public synchronized void connect(BluetoothDevice device) {
        if (D) Log.e(TAG, "Conectado con: " + device);

        if (EstadoActual == STATE_CONNECTING) {
            if (HiloDeConexion != null) {HiloDeConexion.cancel(); HiloDeConexion =
null;} }

        if (HiloConetado != null) {HiloConetado.cancel(); HiloConetado = null;}

        HiloDeConexion = new ConnectThread(device);
        HiloDeConexion.start();
        setState(STATE_CONNECTING);
    }


    public synchronized void connected(BluetoothSocket socket, BluetoothDevice
device) {
        if (D) Log.e(TAG, "connected");

        if (HiloDeConexion != null) {HiloDeConexion.cancel(); HiloDeConexion = null;}

          if (HiloConetado != null) {HiloConetado.cancel(); HiloConetado = null;}

        if (HebraDeAceptacion != null) {HebraDeAceptacion.cancel();
HebraDeAceptacion = null;}
```

```
        HiloConetado = new ConnectedThread(socket);
        HiloConetado.start();

        Message msg = mHandler.obtainMessage(MainActivity.Mensaje_Nombre_Dispositivo);
        Bundle bundle = new Bundle();
        bundle.putString(MainActivity.DEVICE_NAME, device.getName());
        msg.setData(bundle);
        mHandler.sendMessage(msg);

        setState(STATE_CONNECTED);
    }


    public synchronized void stop() {
        if (D) Log.e(TAG, "stop");
        if (HiloDeConexion != null) {HiloDeConexion.cancel(); HiloDeConexion = null;}
        if (HiloConetado != null) {HiloConetado.cancel(); HiloConetado = null;}
        if (HebraDeAceptacion != null) {HebraDeAceptacion.cancel();
HebraDeAceptacion = null;}
        setState(STATE_NONE);
    }

    public void write(byte[] out) {
        ConnectedThread r; //Creacion de objeto temporal
        synchronized (this)  {
            if (EstadoActual != STATE_CONNECTED) return;
            r = HiloConetado;  }
        r.write(out);
    }
    private void connectionFailed() {
        setState(STATE_LISTEN);
        Message msg = mHandler.obtainMessage(MainActivity.Mensaje_TOAST);
        Bundle bundle = new Bundle();
        bundle.putString(MainActivity.TOAST, "Error de conexión");
        msg.setData(bundle);
        mHandler.sendMessage(msg);
    }

    private void connectionLost() {
        setState(STATE_LISTEN);
        Message msg = mHandler.obtainMessage(MainActivity.Mensaje_TOAST);
        Bundle bundle = new Bundle();
        bundle.putString(MainActivity.TOAST, "Se perdio conexion");
        msg.setData(bundle);
        mHandler.sendMessage(msg);
```

```
        msg = mHandler.obtainMessage(MainActivity.MESSAGE_Desconectado);
        mHandler.sendMessage(msg);

    }

    private class AcceptThread extends Thread {
        private final BluetoothServerSocket mmServerSocket;
        public AcceptThread() {
            BluetoothServerSocket tmp = null;
            try {
                tmp     =     AdaptadorBT.listenUsingRfcommWithServiceRecord(NAME,
MY_UUID);
            } catch (IOException e) {
                Log.e(TAG, "listen() fallo", e);
            }
            mmServerSocket = tmp;
        }
        public void run() {
            if (D) Log.e(TAG, "Comenzar HiloDeAceptacion " + this);
            setName("HiloAceptado");
            BluetoothSocket socket = null;
            while (EstadoActual != STATE_CONNECTED) {
                try {
                    socket = mmServerSocket.accept();
                } catch (IOException e) {
                    Log.e(TAG, "accept() failed", e);
                    break;
                }
                if (socket != null) {
                    synchronized (ConexionBT.this) {
                        switch (EstadoActual) {
                        case STATE_LISTEN:
                        case STATE_CONNECTING:
                            connected(socket, socket.getRemoteDevice());
                            break;
                        case STATE_NONE:
                        case STATE_CONNECTED:
                                try {
                                socket.close();
                            } catch (IOException e) {
                                Log.e(TAG, "No se pudo cerrar el socket no deseado", e);
                            }
                            break;
                        }
                    }
                }
            }
```

```
            if (D) Log.e(TAG, "Fin de HIlodeAceptacion");
        }
        public void cancel() {
            if (D) Log.e(TAG, "Cancela " + this);
            try {
                mmServerSocket.close();
            } catch (IOException e) {
                Log.e(TAG, "close() del servidor FAllo", e);
            }
        }
    }


    private class ConnectThread extends Thread {
        private final BluetoothSocket mmSocket;
        private final BluetoothDevice mmDevice;

        public ConnectThread(BluetoothDevice device) {
            mmDevice = device;
            BluetoothSocket tmp = null;
            try {
                tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
            } catch (IOException e) {
                Log.e(TAG, "create() Fallo", e);
            }
            mmSocket = tmp;
        }
        public void run() {
            Log.e(TAG, "Comenzando HebraConectada");
            setName("HiloConectado");
            AdaptadorBT.cancelDiscovery();
            try {
                mmSocket.connect();
            } catch (IOException e) {
                connectionFailed();
                try {
                    mmSocket.close();
                } catch (IOException e2) {
                    Log.e(TAG, "Imposible cerrar el socket durante la falla de conexion",
e2);
                }
                ConexionBT.this.start();
                return;
            }
            synchronized (ConexionBT.this) {
                    HiloDeConexion = null;
            }
```

```
            connected(mmSocket, mmDevice);
        }
        public void cancel() {
            try {
                mmSocket.close();
            } catch (IOException e) {
                Log.e(TAG, "close() of connect socket failed", e);
            }
        }
    }

    private class ConnectedThread extends Thread {
        private final BluetoothSocket BTSocket;
        private final InputStream INPUT_Stream;
        private final OutputStream OUTPUT_Stream;


        public ConnectedThread(BluetoothSocket socket) {
            Log.d(TAG, "Creacion de HiloConectado");
            BTSocket = socket;
            InputStream tmpIn = null;
            OutputStream tmpOut = null;
            // Obtencion del BluetoothSocket de entrada y saldida
            try {
                tmpIn = socket.getInputStream();
                tmpOut = socket.getOutputStream();
            } catch (IOException e) {
                Log.e(TAG, "Sockets temporales No creados", e);
            }
            INPUT_Stream = tmpIn;
            OUTPUT_Stream = tmpOut;
        }
        public void write(byte[] buffer) {
            try {
                    OUTPUT_Stream.write(buffer);
mHandler.obtainMessage(MainActivity.Mensaje_Escrito,        -1,        -1,
buffer).sendToTarget();
            } catch (IOException e) {Log.e(TAG, "Exception during write", e);}
        }
        public void cancel() {
            try {
                    BTSocket.close();
            } catch (IOException e) {
                Log.e(TAG, "close() del socket conectado Fallo", e);
            }
        }
```

```
        public void run() {
            Log.e(TAG, "Comenzar Hebraconectada");
            byte[] buffer = new byte[1024];
            int bytes;

            while (true) {   try {
  bytes = INPUT_Stream.read(buffer);
String readMessageX = new String(readBufX, 0, bytes);
     mHandler.obtainMessage(MainActivity.Mensaje_Leido,          bytes,          -1,
buffer).sendToTarget();}  catch  (IOException  e)  {Log.e(TAG,  "disconnected",
e);connectionLost();break; }
            }
        }
    }

}
```

ANEXO 7. PROGRAMA DEL MAIN ACTIVITY


```
package com.esime.cuadqopter;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.os.Vibrator;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.CompoundButton;
import android.widget.CompoundButton.OnCheckedChangeListener;
import android.widget.Toast;
import android.widget.ToggleButton;

public class MainActivity extends Activity {
	public static final String TAG = "LEDv0";
	public static final boolean D = true;
	public static final int Mensaje_Estado_Cambiado = 1;
	public static final int Mensaje_Leido = 2;
	public static final int Mensaje_Escrito = 3;
	public static final int Mensaje_Nombre_Dispositivo = 4;
	public static final int Mensaje_TOAST = 5;
	public static final int MESSAGE_Desconectado = 6;
	public static final int REQUEST_ENABLE_BT = 7;

	public static final String DEVICE_NAME = "device_name";
	public static final String TOAST = "toast";
	private String mConnectedDeviceName = null;
	private BluetoothAdapter AdaptadorBT = null;
	private ConexionBT Servicio_BT = null;
	private Vibrator vibrador;
	private boolean seleccionador = false;
	public int Opcion = R.menu.main;
	private ToggleButton btnOnOff;
	private Button btnAdelante;
	private Button btnAtras;
	private OnClickListener btnListeners;
	private OnCheckedChangeListener btnOnCheckListener;
```

```
private Button btnIzquierda;
private Button btnDerecha;

@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        iniRefGIU();
        iniListeners();

}

private void iniListeners() {
        btnListeners = new View.OnClickListener() {

                @Override
                public void onClick(View v) {
                        if(v.equals(btnAdelante)){
                                Log.e("Boton Adelante", "Adelantando...");
                                sendMessage("3");
                        }

                        if(v.equals(btnAtras)){
                                Log.e("Boton Atras", "Atras...");
                                sendMessage("4");
                        }

                        if(v.equals(btnIzquierda)){
                                Log.e("Boton Izquierda", "Izquierda...");
                                sendMessage("5");
                        }

                        if(v.equals(btnDerecha)){
                                Log.e("Boton Derecha", "Derecha...");
                                sendMessage("6");
                        }
                }
        };

        btnOnCheckListener = new
CompoundButton.OnCheckedChangeListener() {

                @Override
                public void onCheckedChanged(CompoundButton buttonView,
boolean isChecked) {
                        if (isChecked){
```

```
                                Log.e("Boton OnOff", "Encendiendo...");
                                sendMessage("1");

                        }else{
                                Log.e("Boton OnOff", "Apagando...");
                                sendMessage("2");

                        }

                }
        };

        btnOnOff.setOnCheckedChangeListener(btnOnCheckListener);

    }

    protected void sendMessage(String message) {
        if (Servicio_BT.getState() == ConexionBT.STATE_CONNECTED) {
            if (message.length() > 0) {
                byte[] send = message.getBytes();
                if (D)
                    Log.e(TAG, "Mensaje enviado:" + message);
                Servicio_BT.write(send);                    }
        } else
            Toast.makeText(this,          "No          conectado",
Toast.LENGTH_SHORT).show();
    }

  public  void onStart() {
  super.onStart();
  ConfigBT();
  }

  @Override
  public void onDestroy(){
      super.onDestroy();
      if (Servicio_BT != null) Servicio_BT.stop();    }

  public void ConfigBT(){
    AdaptadorBT = BluetoothAdapter.getDefaultAdapter();
    if (AdaptadorBT.isEnabled()) {
      if (Servicio_BT == null) {
          Servicio_BT = new ConexionBT(this, mHandler);
      }
    }
      else{ if(D) Log.e("Setup", "Bluetooth apagado...");
```

```
    Intent enableBluetooth = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableBluetooth, REQUEST_ENABLE_BT);
        }
   }

   public void onActivityResult(int requestCode, int resultCode, Intent data) {
            switch (requestCode) {

                case REQUEST_ENABLE_BT:
                   if (resultCode == Activity.RESULT_OK) {
                       ConfigBT();
                   } else {
                      finish();}


                   }
                }

   @Override
   public boolean onPrepareOptionsMenu(Menu menux){
   menux.clear();              if (seleccionador==false)Opcion=R.menu.main;
       if                          (seleccionador==true)Opcion=R.menu.desconecta;
       getMenuInflater().inflate(Opcion, menux);
   return super.onPrepareOptionsMenu(menux);
   }

   @Override
   public boolean onOptionsItemSelected(MenuItem item) {
      switch (item.getItemId()) {
      case R.id.Conexion:
        if(D) Log.e("conexion", "conectandonos");
                    vibrador = (Vibrator) getSystemService(VIBRATOR_SERVICE);
                    vibrador.vibrate(1000);
                    String address = "00:14:02:11:04:22";
BluetoothDevice device = AdaptadorBT.getRemoteDevice(address);
                    Servicio_BT.connect(device);

         return true;

      case R.id.desconexion:
         if (Servicio_BT != null) Servicio_BT.stop();
         return true;
      }
      return false;
   }
      private void iniRefGIU() {
```

```
        btnOnOff = (ToggleButton) findViewById(R.id.btnOnOff);
        btnAdelante = (Button) findViewById(R.id.btnAdelante);
        btnAtras = (Button) findViewById(R.id.btnAtras);
        btnIzquierda = (Button) findViewById(R.id.btnIzquierda);
        btnDerecha = (Button) findViewById(R.id.btnDerecha);

    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {

        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }


    final Handler mHandler = new Handler() {
    @Override
    public void handleMessage(Message msg) {

            switch (msg.what) {
                case Mensaje_Escrito:
                 byte[] writeBuf = (byte[]) msg.obj;
                String writeMessage = new String(writeBuf);
                if(D) Log.e(TAG, "Message_write  =w= "+ writeMessage);
                break;

            case Mensaje_Leido:
                byte[] readBuf = (byte[]) msg.obj;
              String readMessage = new String(readBuf, 0, msg.arg1);
              if(D)   Log.e(TAG,   "Message_read       =w=   "+   readMessage);

                break;
                    case Mensaje_Nombre_Dispositivo:
                mConnectedDeviceName                                    =
msg.getData().getString(DEVICE_NAME);
Toast.makeText(getApplicationContext(),          "Conectado          con          "+
mConnectedDeviceName, Toast.LENGTH_SHORT).show();
        seleccionador=true;
                break;
                    case Mensaje_TOAST:
                Toast.makeText(getApplicationContext(),
msg.getData().getString(TOAST),
                Toast.LENGTH_SHORT).show();
                break;

            case MESSAGE_Desconectado:
```

```
                    if(D) Log.e("Conexion","DESConectados");
                    seleccionador=false;
            break;

                }
              }
            };
```



```
}
```

8. SKETCH FINAL PARA EL CONTROLADOR ARDUINO

```
//Librerías

#include "MPU6050_6Axis_MotionApps20.h"

#include "Servo.h"

#include "Wire.h"

#include "I2Cdev.h"

#include "MPU6050.h"

#include "PID_v1.h"

#include "helper_3dmath.h"

//Declaramos parametros

#define BAUD 9600

#define PWM_MIN 57

#define PWM_MAX 100

#define PWM_OFF 50

#define ESC_A 4

#define ESC_B 6

#define ESC_C 8

#define ESC_D 10

#define PITCH_P 0.5

#define PITCH_I 0

#define PITCH_D 1
```

```
#define ROLL_P 2
#define ROLL_I 5
#define ROLL_D 1
#define YAW_P 2
#define YAW_I 5
#define YAW_D 1
#define PITCH_MIN -30
#define PITCH_MAX 30
#define ROLL_MIN -30
#define ROLL_MAX 30
#define YAW_MIN -180
#define YAW_MAX 180
#define PID_PITCH_INFLUENCE 20
#define PID_ROLL_INFLUENCE 20
#define PID_YAW_INFLUENCE 20
//Motores
Servo Motor_A, Motor_B, Motor_C, Motor_D;
int velocidad =0;
float velocityLast;
int va, vb, vc, vd;
int v_ac, v_bd;
float bal_ac, bal_bd;
float bal_ejes;
//MPU6050
MPU6050 mpu;
uint8_t mpuIntStatus;
uint8_t devStatus;
```

```
uint16_t packetSize;
uint16_t fifoCount;
uint8_t fifoBuffer[64];
volatile bool mpuInterrupt = false;
boolean interruptLock = false;
Quaternion q;
VectorFloat gravity;
//PID y sus parámetros
float ypr[3] = {0.0f,0.0f,0.0f};
float yprLast[3] = {0.0f, 0.0f, 0.0f};
float SetPoint_Pitch = 30;
float SetPoint_Roll = 30;
float SetPoint_Yaw = 30;
float SetPoint_Pitch_N = 30;
float SetPoint_Roll_N = 30;
float SetPoint_Yaw_N = 30;
PID pitchReg(&ypr[1], &bal_bd, &SetPoint_Pitch, PITCH_P, PITCH_I, PITCH_D, REVERSE);
PID rollReg(&ypr[2], &bal_ac, &SetPoint_Roll, ROLL_P, ROLL_I, ROLL_D, REVERSE);
PID yawReg(&ypr[0], &bal_ejes, &SetPoint_Yaw, YAW_P, YAW_I, YAW_D, DIRECT);
PID pitchReg2(&ypr[1], &bal_bd, &SetPoint_Pitch_N, PITCH_P, PITCH_I, PITCH_D, REVERSE);
PID rollReg2(&ypr[2], &bal_ac, &SetPoint_Roll_N, ROLL_P, ROLL_I, ROLL_D, REVERSE);
PID yawReg2(&ypr[0], &bal_ejes, &SetPoint_Yaw_N, YAW_P, YAW_I, YAW_D, DIRECT);
//Variables de comunicación
char cadena[255];
```

```
int i=0;
void setup(){
  Serial.begin(BAUD);
  Wire.begin();
  setup_motores();
  setup_mpu();
  setup_balanceo();
  setup_pid();
}

void loop(){
  getYPR();
  calculo_PID();
  comandos();
  calculo_velocidades();
  actualizar_motores();
}
void setup_motores(){
  Serial.println("Configurando Motores");
  Motor_A.attach(ESC_A);
  Motor_B.attach(ESC_B);
  Motor_C.attach(ESC_C);
  Motor_D.attach(ESC_D);
  for(int i=0;i<=56;i++){
    Motor_A.write(i);
    delay(50);
    Motor_B.write(i);
```

```
      delay(50);
    Motor_C.write(i);
      delay(50);
    Motor_D.write(i);
      delay(50);
    }
  Serial.println("MOTORES EN 56");
}
void setup_mpu(){
  Serial.println("Configurando MPU");
  mpu.initialize();
  devStatus = mpu.dmpInitialize();
  if(devStatus == 0){
    mpu.setDMPEnabled(true);
    attachInterrupt(0, dmpDataReady, RISING);
    mpuIntStatus = mpu.getIntStatus();
    packetSize = mpu.dmpGetFIFOPacketSize();
  }
Serial.println("Saliendo de SETUP MPU");
}
void setup_balanceo(){
  Serial.println("Configurando balanceo");
  bal_ejes=0;
  bal_ac=0;
  bal_bd=0;
  Serial.println("Termina configuración de Balanceo");
}
```

```
void setup_pid(){
  Serial.println("SETUP PID");
  pitchReg.SetMode(AUTOMATIC);
  pitchReg.SetOutputLimits(-PID_PITCH_INFLUENCE, PID_PITCH_INFLUENCE);
  rollReg.SetMode(AUTOMATIC);
  rollReg.SetOutputLimits(-PID_ROLL_INFLUENCE, PID_ROLL_INFLUENCE);
  yawReg.SetMode(AUTOMATIC);
  yawReg.SetOutputLimits(-PID_YAW_INFLUENCE, PID_YAW_INFLUENCE);
  pitchReg2.SetMode(AUTOMATIC);
  pitchReg2.SetOutputLimits(-PID_PITCH_INFLUENCE,
PID_PITCH_INFLUENCE);
  rollReg2.SetMode(AUTOMATIC);
  rollReg2.SetOutputLimits(-PID_ROLL_INFLUENCE, PID_ROLL_INFLUENCE);

  yawReg2.SetMode(AUTOMATIC);
  yawReg2.SetOutputLimits(-PID_YAW_INFLUENCE, PID_YAW_INFLUENCE);
 Serial.println("TERMINA SETUP PID");
}
void dmpDataReady() {
    mpuInterrupt = true;
}
void getYPR(){
  Serial.println("getYPR");
 mpuInterrupt = false;
    mpuIntStatus = mpu.getIntStatus();
    fifoCount = mpu.getFIFOCount();
    if((mpuIntStatus & 0x10) || fifoCount >= 1024){
      mpu.resetFIFO();
```

```
    }else if(mpuIntStatus & 0x02){
      while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();
      mpu.getFIFOBytes(fifoBuffer, packetSize);
      fifoCount -= packetSize;
      mpu.dmpGetQuaternion(&q, fifoBuffer);
      mpu.dmpGetGravity(&gravity, &q);
      mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
    }
}
void  calculo_PID(){
    Serial.println("calculo_PID");
  acquireLock();
   ypr[0] = ypr[0] * 180/M_PI;
  ypr[1] = ypr[1] * 180/M_PI;
  ypr[2] = ypr[2] * 180/M_PI;
  if(abs(ypr[0]-yprLast[0])>30) ypr[0] = yprLast[0];
  if(abs(ypr[1]-yprLast[1])>30) ypr[1] = yprLast[1];
  if(abs(ypr[2]-yprLast[2])>30) ypr[2] = yprLast[2];
  yprLast[0] = ypr[0];
  yprLast[1] = ypr[1];
  yprLast[2] = ypr[2];
 /* pitchReg.Compute();
  rollReg.Compute();
  yawReg.Compute();
    pitchReg2.Compute();
  rollReg2.Compute();
  yawReg2.Compute();
```

```
  */
  releaseLock();
}
void comandos(){
  Serial.println("comandos");
  if(Serial.available()>0){
    char dato= Serial.read();
    cadena[i++]=dato;
    if (strstr(cadena,"ON")!=0){
      Serial.println("Encendido");
   clean(); }
    else if(strstr(cadena,"OFF")!=0){
      Serial.println("Apagando");
    clean();}
    else if(strstr(cadena,"ADELANTE")!=0){
      Serial.println("ADELANTE");
    pitchReg.Compute();
    clean();}
    else if(strstr(cadena,"ATRAS")!=0){
      Serial.println("ATRAS");
      pitchReg2.Compute();
    clean();}
    else if(strstr(cadena,"IZQUIERDA")!=0){
      Serial.println("IZQUIEDA");
     rollReg2.Compute();
    clean();}
        else if(strstr(cadena,"DERECHA")!=0){
```

```
      Serial.println("DERECHA");
      rollReg.Compute();
    clean();}
    else if(dato=='\n'){
      velocidad = atoi(cadena);
      Serial.println(velocidad);
    clean();
          }
  }
}
void acquireLock(){
  interruptLock = true;
}
void releaseLock(){
  interruptLock = false;
}
void clean()
{
  for (int cl=0; cl<=i; cl++)
  {
    cadena[cl]=0;
  }
  i=0;
}
void calculo_velocidades(){
  Serial.println("calculo_velocidades");
  if((velocidad < PWM_MIN) || (velocidad > PWM_MAX)) velocidad = velocityLast;
```

```
  velocityLast = velocidad;
  v_ac = (abs(-100+bal_ejes)/100)*velocidad;
  v_bd = ((100+bal_ejes)/100)*velocidad;
  va = ((100+bal_ac)/100)*v_ac;
  vb = ((100+bal_bd)/100)*v_bd;
  vc = (abs((-100+bal_ac)/100))*v_ac;
  vd = (abs((-100+bal_bd)/100))*v_bd;
}
void actualizar_motores(){
  Serial.println("actualizar_motores");
Motor_A.write(va);
  Motor_B.write(vc);
Motor_C.write(vb);
Motor_D.write(vd);
}
```

BIBLIOGRAFIA

[1] Ortega, Manuel R. (1989-2006) (en español). Lecciones de Física (4 volúmenes). Monytex

[2] Resnick, Robert & Halliday, David (2004) (en español). Física 4ª. CECSA, México

[3] Feynman, Leighton and Sands. Lectures on physics. Addison-Wesley.

[4] Phipps, Clarance A. (1997). Variable Speed Drive Fundamentals. The Fairmont Press, Inc.

[5] arduino.cc/en/Main/arduinoBoardMega2560

[6] TG Wilson, PH Trickey, "Máquina de CC. Con Solid State Conmutación", AIEE papel I. CP62-1372, 07 de octubre 1962

[7] Ohio Motores Eléctricos. DC de protección del motor. Ohio Motores Eléctricos.2011.

[8] http://arduino.cc/

[9] http://developer.android.com/index.html

[10] http://www.botched.co.uk/pic-tutorials/mpu6050-setup-data-aquisition/

[11]https://github.com/jrowberg/i2cdevlib/blob/master/Arduino/MPU6050/Examples/MPU6050_DMP6/MPU6050_DMP6.ino

[12] https://github.com/strangedev/Arduino-Quadcopter/blob/master/quadcopter/quadcopter.ino

[13]http://www.emartee.com/product/41915/HC%2006%20Serial%20Port%20Bluetooth%20Module

[14] https://www.sparkfun.com/products/11028

[15] OGATA KATSUHIKO, Ingeniería en control moderno 3° Edición Prince Hall Hispanoamericana, 1998.

[16] Proyecto Final de Carrera Xabier Legasa Martín-Gil Ingeniería Informática Facultad de Informática de Barcelona (FIB) UPC BarcelonaTech 2011/2012.