



**DISEÑO E IMPLEMENTACIÓN DE UN ALGORITMO PROPORCIONAL-
INTEGRAL-DERIVATIVO PARA LA ESTABILIZACIÓN DE UN
CUADRICÓPTERO**

TRABAJO ESPECIAL DE GRADO

presentado ante la

UNIVERSIDAD CATÓLICA ANDRÉS BELLO

como parte de los requisitos para optar al título de

INGENIERO INFORMÁTICO

REALIZADO POR:

Luis Vicens

Yoshua Nava

TUTOR ACADÉMICO:

Ing. Evelenir Barreto

FECHA:

Enero de 2015

DEDICATORIAS

Le dedico este trabajo a mi mamá, a mi abuela, a mi tío, a mi novia, y a todos mis amigos. Les agradezco mucho su cariño, amistad y apoyo.

También deseo agradecer a todos mis profesores. Espero mediante este trabajo honrar su paciencia, dedicación y ejemplo

Yoshua Nava

A mis padres y mi hermano, por ser el pilar fundamental en todo lo que soy, por su incondicional apoyo mantenido a través del tiempo.

A mi novia Gabriela quien me ha dado cariño y me alentó a continuar en los momentos más difíciles.

A Hector, Aileen, Mimia, Andrea, Juan, Marcos, Patricia y todos los que de una manera u otra supieron alentarme para cumplir con este trabajo.

Luis F. Vicens

AGRADECIMIENTOS

Los autores del presente trabajo agradecen a:

- La profesora Evelenir Barreto, tutora del presente trabajo, por su paciencia, dedicación, y por habernos apoyado de principio a fin durante su realización.
- Al profesor Wilmer Pereira, cuyas clases de la electiva Robótica e Inteligencia Artificial dieron pie a la realización de este proyecto y despertaron nuestra pasión por la robótica.
- Al profesor Giovanni Sparacio, por su apoyo, y su solidaridad, al ayudarnos a sustituir uno de los engranajes de transmisión del cuadricóptero y haber siempre tenido la disposición a apoyar el desarrollo del proyecto.
- A los miembros del Grupo de Mecatrónica de la Universidad Simón Bolívar, con especial mención a los profesores José Cappelletto y Gaudí Morantes por su gran paciencia y valiosos consejos.
- A todos los miembros del Centro de Investigación y Desarrollo de Ingeniería de la Universidad Católica Andrés Bello por abrirnos las puertas para trabajar en sus instalaciones y hacernos sentir parte de su gran grupo humano.
- A la profesora Carolina Chang, por promover la robótica en el país, organizar las competencias nacionales de robótica año tras año, y dar ejemplo de humildad, perseverancia y brío a todos los que queremos seguir nuestra carrera en dicha área.

SINOPSIS

El desarrollo de vehículos aéreos no tripulados ha experimentado avances vertiginosos en la última década, motivado por los múltiples adelantos en el área de la electrónica, la teoría de control, las telecomunicaciones, y en el desarrollo de baterías químicas. Sin embargo, no se ha realizado un desarrollo significativo de cuadricópteros de bajo costo, y una estandarización de los algoritmos de control para la estabilización de los mismos. El presente Trabajo Especial de Grado tuvo como propósito el diseño e implementación de un algoritmo Proporcional-Integral-Derivativo para estabilizar la posición angular y altura de un cuadricóptero en vuelo.

Para llevar a cabo el desarrollo del presente trabajo se utilizó un chasis de cuadricópteros de bajo costo Draganflyer V, y componentes electrónicos disponibles en el mercado venezolano. Se desarrolló la electrónica necesaria para el control del cuadricóptero, algoritmos de estimación de estado del mismo, protocolos de comunicación inalámbrica para comando remoto y telemetría, algoritmos de control Proporcional-Integral-Derivativo, y una plataforma de pruebas consistente en un software para comando remoto y telemetría desarrollado sobre Robot Operating System (ROS), y scripts para análisis de las señales de los sensores del cuadricóptero mediante Transformada Rápida de Fourier.

Como esquema metodológico se adoptó la metodología de desarrollo de software en espiral, la cual se fundamenta en el desarrollo, evaluación y re-diseño sucesivo de prototipos incrementales del sistema, hasta alcanzar una solución que satisfaga exitosamente los requerimientos iniciales.

ÍNDICE DE CONTENIDO

I.1	Planteamiento del problema.....	2
I.2	Objetivos	4
I.2.1	- Objetivo General:	4
I.2.2	- Objetivos Específicos:.....	4
I.3	Alcance	5
I.4	Limitaciones	6
I.5	Justificación.....	7
II.1	Cuadricóptero.....	8
II.2	Unidad de Medición Inercial (IMU)	8
II.3	Acelerómetro.....	9
II.4	Giroscopio	10
II.5	Sensor de Ultrasonido.....	11
II.6	XBee	11
II.7	Arduino.....	13
II.8	Teoría de control.	14
II.13	Algoritmo Proporcional-Integral-Derivativo (PID).....	16
II.14	Filtro de Kalman	16
II.15	Robot Operating System (ROS)	18
II.1	Descripción de las etapas de la metodología en espiral	20
II.2	Justificación de la metodología a utilizar	22
IV.1	Circuito de alimentación y control de motores.....	26
IV.2	Circuito de lógica, sensores y comunicación.....	30
IV.3	Estimación de orientación del cuadricóptero.....	32
IV.3.1	Convenciones respecto a los ángulos	32
IV.3.2	Descripción de la Unidad de Medición Inercial (IMU)	32
IV.3.3	Algoritmo de estimación de posición angular y velocidad angular del cuadricóptero	33
IV.4	Estimación de posición y velocidad lineal en el eje Z del cuadricóptero	34
IV.4.1	Descripción del sensor de altura utilizado	34
IV.4.2	Algoritmo de estimación de posición y velocidad en el eje z del cuadricóptero	

IV.5	Comunicación inalámbrica	36
a.	Descripción de las características y configuración de los módulos XBEE utilizados 36	
b.	Descripción del protocolo de comunicación desarrollado.....	37
IV.6	Sistemas de control.....	44
IV.6.1	Arquitectura de control	44
IV.6.2	Simulación	47
IV.6.3	Implementación en el cuadricóptero	48
IV.7	Plataforma de pruebas	52
IV.7.1	Software de telemetría y comandos	52
IV.7.2	Scripts para análisis en el dominio de la frecuencia de datos de los sensores 59	
IV.7.3	Montaje para la ejecución de pruebas en un solo eje del cuadricóptero	60
IV.7.4	Montaje para la ejecución de pruebas en vuelo restringido	60
VI.1	Conclusiones.....	63
VI.2	Recomendaciones.....	64

ÍNDICE DE TABLAS

<i>Tabla 2. 1 Características del Arduino Nano 3.0</i>	14
<i>Tabla 4. 1 Desglose del consumo promedio por componente, y acumulado, del circuito de lógica, sensores y comunicación.</i>	30
<i>Tabla 4. 2 Especificaciones del modulo XBee</i>	36
<i>Tabla 4. 3 Estructura general de los mensajes</i>	38
<i>Tabla 4. 4 Mensajes de la interfaz de telemetría del cuadricóptero</i>	41
<i>Tabla 4. 5 Mensajes de la interfaz de comando remoto del cuadricóptero</i>	41
<i>Tabla 4. 6 retardo de mensajes</i>	42
<i>Tabla 4. 7 Pruebas realizadas con el XBee a diferentes distancias, cada prueba envía un total de 200 mensajes.</i>	42

ÍNDICE DE ILUSTRACIONES

<i>Ilustracion 2. 1 Estructura básica de un cuadricóptero</i>	<i>8</i>
<i>Ilustracion 2. 2 Orientacion porporcionada por una IMU.....</i>	<i>9</i>
<i>Ilustracion 2. 3 Visión interna de un acelerómetro electrónico de 3 ejes</i>	<i>10</i>
<i>Ilustracion 2. 4 Efecto Coriolis</i>	<i>11</i>
<i>Ilustracion 2. 5 Imagen cara superior Arduino Nano 3.0.....</i>	<i>13</i>
<i>Ilustracion 2. 6 Sistemas de control con y sin realimentación.....</i>	<i>15</i>
<i>Ilustracion 2. 7 Una visión del filtro de Kalman</i>	<i>17</i>
<i>Ilustracion 3. 1 Ciclo de vida en espiral.</i>	<i>20</i>
<i>Ilustración 4. 29 Cuadricóptero ensamblado</i>	<i>24</i>
<i>Ilustración 4. 30 Diagrama general del sistema.</i>	<i>25</i>
<i>Ilustración 4. 1 Relación voltaje-corriente de los motores DC del chasis Draganflyer V.</i>	<i>26</i>
<i>Ilustración 4. 2 Diagrama de bloques del sistema de control de velocidad de los motores DC del cuadricóptero.</i>	<i>28</i>
<i>Ilustracion 4. 3 Placa del circuito de alimentación y control de motores.</i>	<i>29</i>
<i>Ilustracion 4. 4 Placa del circuito de lógica, sensores y comunicación.</i>	<i>31</i>
<i>Ilustracion 4. 5 Pololu miniIMU-9 V2.....</i>	<i>32</i>
<i>Ilustracion 4. 6: funcionamiento del algoritmo de estimación de posición y velocidad angular</i>	<i>33</i>
<i>Ilustracion 4. 7 Sensor ultrasónico Parallax Ping.....</i>	<i>34</i>
<i>Ilustracion 4. 8: posición del sensor ultrasónico de distancia en el cuadricóptero.</i>	<i>34</i>
<i>Ilustracion 4. 9: funcionamiento del algoritmo de estimación de posición y velocidad lineal en el eje Z.....</i>	<i>35</i>
<i>Ilustracion 4. 10 (de derecha a izquierda) Modulo XBee Series 1 de 1mW, XBee Explorer Regulated y XBee Explorer USB.</i>	<i>36</i>
<i>Ilustracion 4. 11 Diagrama de movimientos del cuadricóptero y signo que toman los valores de aceleración y posición.</i>	<i>40</i>
<i>Ilustracion 4. 12 Porcentaje de mentajes recibidos en distancia.....</i>	<i>43</i>
<i>Ilustración 4. 13 Arquitectura de control propuesta.....</i>	<i>44</i>
<i>Ilustración 4. 14 Arquitectura de control simplificada.....</i>	<i>46</i>
<i>Ilustración 4. 15 Rendimiento de los sistemas de control PID de la arquitectura propuesta en simulación.....</i>	<i>47</i>
<i>Ilustración 4. 16 Rendimiento de los sistemas de control PID de la arquitectura simplificada en simulación.....</i>	<i>48</i>
<i>Ilustración 4. 17 Ciclo de ejecución de la rutina de estimación de estado, control y comunicación del cuadricóptero</i>	<i>49</i>
<i>Ilustración 4. 18 Rendimiento de los sistemas de control PID de velocidad angular de Pitch (Izquierda) y Roll (Derecha) ante perturbaciones.</i>	<i>50</i>
<i>Ilustración 4. 19 Rendimiento del sistema de control de velocidad angular de Yaw (Izquierda) y de posición angular de Yaw (Derecha) ante perturbaciones.</i>	<i>51</i>
<i>Ilustración 4. 20 Grafo de los nodos desarrollados en ejecución.....</i>	<i>53</i>
<i>Ilustración 4. 21 Nodo comunicación_serial en funcionamiento.</i>	<i>54</i>
<i>Ilustración 4. 22 Nodo logitech_rumblepad_ii en funcionamiento.</i>	<i>55</i>

<i>Ilustración 4. 23 Distribución de botones para ejecución de comandos en el control Logitech Rumblepad II.</i>	56
<i>Ilustración 4. 24 Visualización de datos en rqt_plot.</i>	57
<i>Ilustración 4. 25 Diagrama de funcionamiento del software de telemetría y comandos.</i>	58
<i>Ilustración 4. 26 Captura de pantalla de la ventana de visualización de datos de ángulos en MATLAB.</i>	59
<i>Ilustración 4. 27 Ejemplo de base utilizada para restringir el movimiento del cuadricóptero en un eje.</i>	60
<i>Ilustración 4. 28 Diagrama de movimiento sobre el eje yaw.</i>	60

INTRODUCCIÓN

En tiempos recientes se han realizado importantes progresos en las áreas de electrónica, teoría de control y almacenamiento de energía por medio de baterías químicas, que han fomentado el desarrollo de aeronaves no tripuladas, entre las cuales se encuentran los cuadricópteros: aeronaves sustentadas y propulsadas por cuatro rotores horizontales, que pueden realizar despegues y aterrizajes verticales y se caracterizan por su alta maniobrabilidad. La realización de múltiples trabajos de investigación con cuadricópteros ha permitido el establecimiento de prácticas y técnicas comunes para su construcción, control y operación, y un rápido aumento de su la popularidad por el amplio espectro de aplicaciones prácticas posibles de los cuadricópteros. Esto ha promovido un incremento en la producción de componentes para su construcción, y una disminución significativa en su precio. Sin embargo, la dificultad en la adquisición de componentes para la construcción de cuadricópteros en el mercado local ha evitado en gran medida el desarrollo de proyectos relacionados en Republica Bolivariana de Venezuela.

El presente trabajo especial de grado pretende partir de lo realizado en anteriores proyectos de desarrollo de cuadricópteros de bajo costo que se han centrado en el diseño mecánico y electrónico de los mismos, y realizar un primer acercamiento a la estabilización de la posición angular y altura de un cuadricóptero en vuelo mediante un algoritmo Proporcional-Integral-Derivativo.

CAPÍTULO I – Problema

I.1 Planteamiento del problema

En tiempos recientes, con el desarrollo de las telecomunicaciones y la microelectrónica, existe una tendencia hacia el desarrollo de vehículos aéreos no tripulados, ya sea manejados a distancia o autónomos. En particular, se ha dado especial atención al desarrollo de multi-rotors, ya que estos brindan una gran maniobrabilidad y precisión durante el vuelo, características muy útiles para tareas de reconocimiento, vigilancia y exploración. Entre los multi-rotors, sobre el que se tiene mayor interés en la actualidad, y sobre el cual se trabajará en el siguiente Trabajo Especial de Grado, es el cuadricóptero: un multi-rotor propulsado por cuatro motores posicionados en forma de cruz.

Muchas ideas sobre desarrollo de cuadricópteros vienen de épocas anteriores al descubrimiento del transistor, y solo han podido llevarse a cabo en tiempos recientes, debido al avance de los microprocesadores y las baterías químicas. En principio solo organizaciones militares y de investigación podían participar en el desarrollo de los cuadricópteros, por el alto costo de la microelectrónica en sus primeros años. Pero con el paso del tiempo, el desarrollo de cuadricópteros se ha convertido en un proyecto asequible entre aficionados a la electrónica e investigadores su construcción y estudio, ya que estos pueden acudir a un mercado muy competitivo de piezas y componentes de construcción de estos vehículos, y existe una gran cantidad de información relacionada en la Internet. Incluso, hay importantes proyectos de larga difusión, como el caso de Arduino o Raspberry Pi, que pueden brindar al usuario común la oportunidad de aproximarse al desarrollo de cuadricópteros facilitando el diseño y construcción de circuitos electrónicos, y la programación de los mismos.

Para usuarios novatos y con el prospecto de realizar proyectos de bajo coste, se han desarrollado trabajos como [Nadales 2009] y [Burkamshaw 2010], que han planteado el uso de diversas plataformas de hardware y software para la construcción de cuadricópteros de

bajo coste.

En la actualidad, no se ha logrado la estandarización de los sistemas de control de estabilidad y altura para dichas plataformas, ni de una plataforma de pruebas para el ajuste de parámetros de dichos sistemas de control.

Se propone desarrollar en el siguiente Trabajo Especial de Grado un algoritmo de control proporcional-integral-derivativo para regularizar la posición angular y altura de un cuadricóptero en vuelo.

I.2 Objetivos

I.2.1 - Objetivo General:

Diseñar e implementar un algoritmo Proporcional-Integral-Derivativo que permita estabilizar la posición angular y altura de un cuadricóptero desarrollado sobre la plataforma Arduino.

I.2.2 - Objetivos Específicos:

1. Diseñar e implementar un cuadricóptero con una unidad de control basada en Arduino.
2. Diseñar e implementar una interfaz de comunicación inalámbrica entre el cuadricóptero y una computadora para tareas de encendido, apagado, movimientos simples en tres dimensiones y recopilación de información de los sensores del cuadricóptero.
3. Diseñar e implementar una interfaz de comunicación para la obtención y análisis de datos por telemetría.
4. Diseñar e implementar un algoritmo Proporcional-Integral-Derivativo que permita la estabilización del cuadricóptero.
5. Diseñar e implementar una plataforma de pruebas en tiempo real.

I.3 Alcance

El siguiente Trabajo Especial de Grado tiene como alcance el desarrollo de un algoritmo Proporcional-Integral-Derivativo para la estabilización angular y de altura de un cuadricóptero. El cuadricóptero en cuestión poseerá una unidad de control basada en la plataforma Arduino, y, para objeto de pruebas, podrá establecer comunicación con un agente externo que permitirá realizar las siguientes acciones:

- Encendido y apagado del cuadricóptero de forma remota.
- Control remoto de los movimientos del cuadricóptero desde la computadora. Se implementarán una serie de comandos de control los cuales permitirán mover el cuadricóptero hacia arriba, abajo, adelante, atrás, izquierda, y derecha; sin poder combinar entre sí estos movimientos.
- Obtención y visualización de datos de los sensores de posición angular y altura para evaluar la eficiencia del algoritmo y la estabilidad del robot.

I.4 Limitaciones

- La plataforma a utilizar para el manejo del cuadricóptero fue un microcontrolador Arduino Nano 3.0, que trabaja a una frecuencia máxima de 16MHz.
- El sistema utilizó una batería de 1350mAh que sirvió para alimentar a los motores, y tuvo una duración de entre 10 y 15 minutos de uso continuo.
- El sensor de ultrasonido Parallax PING))) utilizado para medir la distancia respecto al suelo del cuadricóptero, tiene un rango máximo de cuatro metros. Por la magnitud de los retardos que deben programarse para medir largas distancias, los cuales pueden afectar la estabilidad del sistema en vuelo, se limitó la altura de vuelo máxima del cuadricóptero a un metro.
- El cuadricóptero durante el vuelo no puede detectar objetos a su alrededor ni evadirlos, por lo cual el ambiente de pruebas debe estar totalmente despejado.
- Sólo se programó al cuadricóptero para realizar seis movimientos simples guiados por el usuario: adelante, atrás, izquierda, derecha, ascenso y descenso.
- Las pruebas en tiempo real fueron realizadas en un ambiente a puerta cerrada.

I.5 Justificación

Considerando la situación económica actual de la nación, se considera que el presente Trabajo Especial de Grado representa un aporte al desarrollo de cuadricópteros de bajo coste, que pueden ser útiles para propósitos de investigación, educativos, y para el desarrollo de tecnología en el país. En particular, el presente Trabajo Especial de Grado busca complementar y mejorar la plataforma para cuadricópteros de bajo coste basada en Arduino desarrollada en [Nadales 2009], mediante la implementación de los sistemas de control de posición angular y altura, y la obtención de datos por telemetría.

CAPÍTULO II – Marco Referencial

II.1 Cuadricóptero

Un cuadricóptero es un helicóptero con cuatro (4) rotores que están dirigidos hacia arriba y colocados en forma de cuadrado, equidistantes del centro de masa. Está basado en tres (3) ejes ortogonales llamados roll (eje “z”), yaw (eje “x”) y pitch (eje “y”), donde el origen está ubicado en el cruce de los ejes. Los cuadricópteros son controlados por el ajuste de las velocidades de cada rotor. En la Ilustración 2. se puede observar la estructura básica de un cuadricóptero con las velocidades angulares (ω), torques (τ) y fuerzas creadas por los cuatro (4) rotores (f) numeradas del 1 hasta el 4.

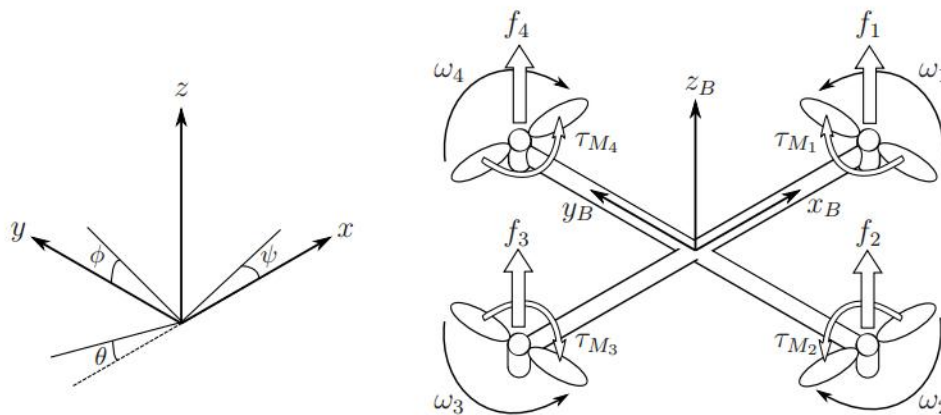
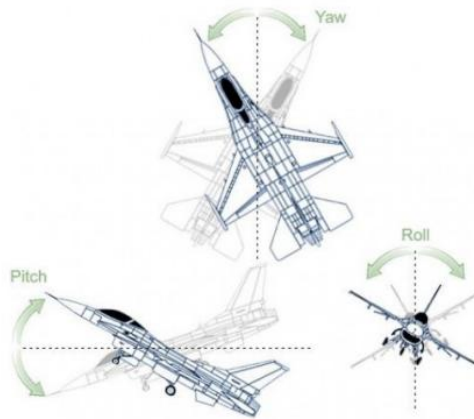


Ilustración 2. 1 Estructura básica de un cuadricóptero

Fuente: http://sal.aalto.fi/publications/pdf-files/eluu11_public.pdf

II.2 Unidad de Medición Inercial (IMU)

Una unidad de medición inercial o IMU es un componente electrónico basado en sensores de aceleración y velocidad angular (acelerómetros y giróscopos respectivamente) la cual reporta el movimiento y orientación que sufre dicha unidad. Es el componente principal de sistemas de guía inercial usados en vehículos aéreos, espaciales, marinos y aplicaciones robóticas.



Ilustracion 2. 2 Orientacion porporcionada por una IMU

Fuente: [Bonastre 2010]

Componentes de una IMU

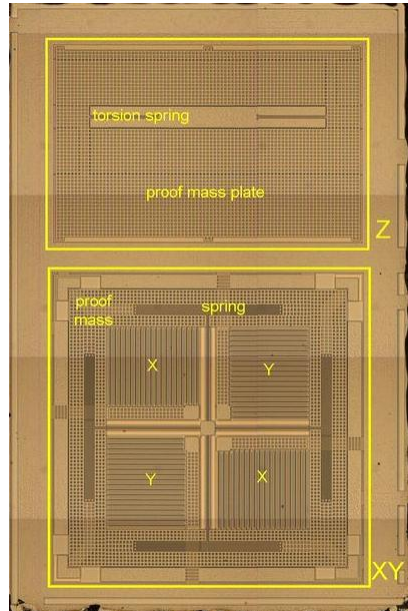
Cualquier unidad de medida inercial está compuesta como mínimo por un acelerómetro y un giróscopo para captar una aceleración y una velocidad angular en concreto. Generalmente, es interesante que las IMUs capten la aceleración y la velocidad angular en los tres ejes de coordenadas para conocer el movimiento exacto del componente.

II.3 Acelerómetro

Son sensores inerciales que miden la segunda derivada de la posición. Por tanto miden la fuerza de inercia generada cuando una masa u objeto es afectado por un cambio de velocidad.[Nadales 2009]

Existen varios tipos de acelerómetros, dependiendo de su fabricación y funcionamiento. Las IMUs incorporan acelerómetros integrados en silicio, utilizando la tecnología llamada MEMS⁶, debido a la necesidad de reducir el tamaño total de la unidad. La mayoría de éstos son capacitivos, y calculan la aceleración mediante el voltaje obtenido entre dos placas una de las cuales varía su posición dependiendo del movimiento del acelerómetro.

Se caracterizan por ser muy precisos en situaciones estables y tener un gran error en situaciones vibratorias o movimientos muy inestables. [Bonastre 2010]



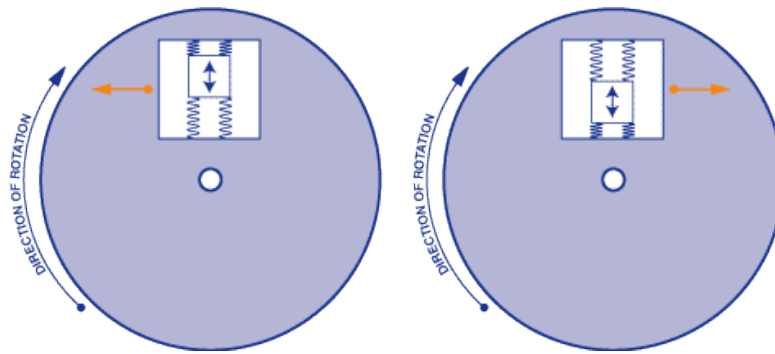
Ilustracion 2. 3 Visión interna de un acelerómetro electrónico de 3 ejes

Fuente: <http://memsjournal.typepad.com/.a/6a00d8345225f869e20147e0f99fd7970b-pi>

II.4 Giroscopio

El giroscopio es un tipo de sensor que se utiliza para medir la velocidad angular de un cuerpo en rotación.

Este sensor aprovecha el efecto coriolis el cual aparece cuando un objeto se mueve en un sistema de referencia en rotación, y consiste en que dicho objeto se vea afectado por una aceleración respecto al sistema en rotación. Esta aceleración es perpendicular al eje de giro del sistema y varía según el objeto se acerca o se aleje



Ilustracion 2. 4 Efecto Coriolis

Fuente: <http://www.analog.com/library/analogdialogue/archives/37-03/Gyro-03.gif>

Las unidades de medida inercial utilizan giróscopos MEMS, es decir, integrados y de tamaño reducido. La salida de dicho sensor es un voltaje, la variación del cual indica en grados por segundo ($V/^{\circ}/s$) la velocidad angular sufrida por el sensor. Se caracterizan por tener un error constante y lineal llamado “bias” el cual se debe tener en cuenta. [Bonastre 2010]

II.5 Sensor de Ultrasonido

Es un sensor utilizado para medir distancias, el cual emite pulsos ultrasónicos que se reflejan sobre un objeto o una superficie. Cuando el eco es recibido por el sensor puede calcular la distancia a la que se encuentra utilizando la diferencia de tiempo que tardó un pulso en ir hasta el objeto y regresar. Los materiales pueden ser sólidos o líquidos. Sin embargo han de ser reflectores de sonidos.

II.6 XBee

Es el nombre de la especificación de un conjunto de protocolos de alto nivel de comunicación inalámbrica para su utilización con radios digitales de bajo consumo, basada en el estándar IEEE 802.15.4 de redes inalámbricas de área personal (wireless personal area network, WPAN). Las principales características de ZIGBEE son:

- Velocidades comprendidas entre 20 kB/s y 250 kB/s.
- Alcance de 10 a 75 mts.
- Redes cambian los canales en forma dinámica en caso de interferencias.
- Alto ahorro de energía.

Ventajas

- Ideal para conexiones punto a punto y punto a multipunto.
- Reduce tiempos de espera en el envío y recepción de paquetes.
- Detección de Energía (ED).
- Baja ciclo de trabajo - Proporciona larga duración de la batería.
- Hasta 65.000 nodos en una red.
- Son más baratos y de construcción más sencilla.
- La tasa de transferencia es muy baja.

Desventajas

- Sólo manipula textos pequeños comparados con otras tecnologías.
- Poca cobertura por ser de tipo WPAN.
- No compatible con bluetooth.

La clasificación de dicha tecnología puede ser:

1. De acuerdo a los dispositivos:

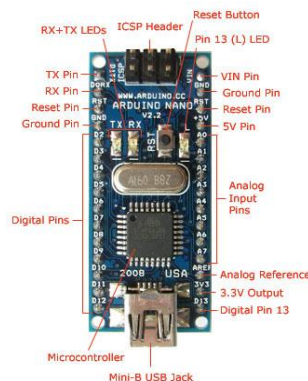
- Coordinador ZIGBEE: se encarga de controlar la red y los caminos que deben seguir los dispositivos para conectarse entre ellos.
- Router ZIGBEE: interconecta dispositivos separados en la topología de la red.
- Dispositivo final: se comunica con su nodo padre pero no puede transmitir información destinada a otros dispositivos.

2. De acuerdo a su funcionalidad:

- Dispositivo de funcionalidad completa: puede funcionar como Coordinador o Router ZIGBEE.
- Dispositivo de funcionalidad reducida: tiene capacidad y funcionalidad limitadas con el objetivo de conseguir un bajo costo y una gran simplicidad.

II.7 Arduino

Arduino es un proyecto de desarrollo de tarjetas controladoras de hardware libre, de bajo costo y fácil programación, con el fin de acelerar el proceso de prototipado y desarrollo de proyectos, y apoyar la educación en electrónica. Consta de una placa con entradas analógicas y digitales, y salidas digitales, y de un entorno integrado de desarrollo que se apoya sobre el lenguaje de programación Processing. Las primeras tarjetas Arduino utilizaban el microcontrolador ATMEL ATmega328, un chip de bajo costo y amplias capacidades de memoria y manejo de entradas y salidas por medio de sus puertos. Conforme ha ido avanzando el tiempo, se han desarrollado tarjetas Arduino con mejores procesadores, como lo son los modelos Arduino Mega2560 (8 bits), Arduino Tre (32 bits, ARM) y Arduino Galileo (32 bits, x86). [Banzi 2011].



Ilustracion 2. 5 Imagen cara superior Arduino Nano 3.0
Fuente: <http://arduino.cc/es/Main/ArduinoBoardNano>

La placa Arduino Nano 3.0, la misma que se utilizará para el desarrollo del Trabajo Especial de Grado, tiene las siguientes características:

Tabla 2. 1 Características del Arduino Nano 3.0

Característica	Descripción
Microcontrolador	Atmel ATmega328
Frecuencia	16 MHz
Memoria SRAM	2 KB
EEPROM	1 KB
Memoria Flash	32 KB
Entradas analógicas	8
Entradas/salidas digitales (De los cuales 6 proveen PWM)	14
Interrupciones	2 externas,
Protocolos de comunicación	USART, I2C, SPI, AREF
Tensión de operación	5V
Tensión de entrada (recomendada)	7-12 V
Tensión de entrada (límites)	6-20 V
Corriente máxima por cada pin	40 mA
Dimensiones	18,5mm x 43,2mm

Fuente: <http://arduino.cc/es/Main/ArduinoBoardNano>

II.8 Teoría de control.

La teoría de control es una teoría matemática que rige la manipulación de los parámetros que afectan el comportamiento de un sistema, para producir un comportamiento deseado u óptimo. [Zabczyk 1993].

La teoría de control se ocupa del diseño de algoritmos de regulación de estado, observadores, e identificación de sistemas. Un sistema de control puede definirse como un arreglo de componentes acoplados de tal manera, que el arreglo pueda comandar, dirigir, o regularse a sí mismo o a otro sistema. [Dulhoste 2011]. Un sistema de control está constituido

por entradas, salidas y estados [Vidyasagar 2010].

Se dice que un sistema o planta está en lazo abierto cuando las entradas no son afectadas o modificadas por los valores en las salidas de la planta [Rodríguez 2013]. La mayoría de los sistemas de lazo abierto son estables con entradas de referencia limitadas. De lo que normalmente carecen los sistemas de lazo abierto es de velocidad y precisión suficientes para seguir la entrada de referencia aplicada al sistema [Aliciatore 2008].

Para un preciso control de un sistema es necesario usar retroalimentación de los sensores (por ejemplo, un codificador o un tacómetro). Al restar una señal de retroalimentación de una señal de entrada deseada (llamada valor de referencia de entrada), se tiene una medición del error en la respuesta. Al cambiar continuamente la señal de comando al sistema con base en la señal de error, se puede mejorar la respuesta del sistema. A esto se le llama control por retroalimentación o de lazo cerrado [Aliciatore 2008].

Sistemas de control

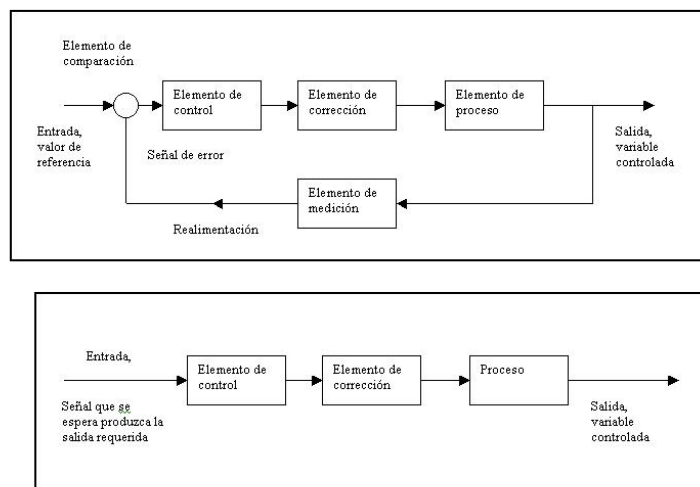


Ilustración 2. 6 Sistemas de control con y sin realimentación

Fuente: <http://ayciaquillo.blogspot.com/2013/02/1-clase.html>

II.13 Algoritmo Proporcional-Integral-Derivativo (PID)

Los controladores generales pueden tomar muchas formas, pero la mayoría de las aplicaciones industriales usan controladores PID o proporcional-integral-derivativo. La forma matemática de un controlador PID, donde la señal de error se expresa como $e(t)$ es la siguiente:

$$\text{señal de comando} = K_p e(t) + K_d \frac{d}{dt} e(t) + K_i \int e(t) dt,$$

donde K_p se refiere como la ganancia proporcional, K_d es la ganancia derivativa y K_i es la ganancia integral.

El control proporcional es el más intuitivo por que la señal de control es proporcional al error. Mientras más grande sea el error, mayor será la acción correctiva. Una enorme ganancia proporcional crea una respuesta rápida, pero puede conducir a exceso y oscilación, en especial si el sistema tiene poco amortiguamiento. La ganancia derivativa responde a la tasa de cambio de la señal de error. Esto permite al controlador anticipar cambios en la respuesta del sistema, que pueden resultar en menos exceso de oscilación amortiguada. La ganancia integral ayuda a eliminar error de estado estacionario al sumar errores a los largo del tiempo. Mientras más tiempo permanezca el error en un lado de la entrada de referencia deseada, más grande se vuelve la acción correctiva como resultado de la ganancia integral [Alciatore 2008].

II.14 Filtro de Kalman

El filtro de Kalman es un conjunto de ecuaciones matemáticas que proveen una solución recursiva eficiente del método de mínimos cuadrados. Esta solución permite calcular un estimador lineal, insesgado y óptimo del estado de un proceso en cada momento del tiempo con base en la información disponible en el momento $t-1$, y actualizar, con la información adicional disponible en el momento t , dichas estimaciones. Este filtro es el principal algoritmo para estimar sistemas dinámicos especificados en la forma de estado-

espacio (State-space). [Ramirez 2003]

El filtro de Kalman es el principal algoritmo para estimar sistemas dinámicos representados en la forma de estado-espacio. En esta representación el sistema es descrito por un conjunto de variables denominadas de estado. El estado contiene toda la información relativa al sistema a un cierto punto en el tiempo. Esta información debe permitir la inferencia del comportamiento pasado del sistema, con el objetivo de predecir su comportamiento futuro. [Ramirez 2003]

Lo que hace al filtro tan interesante es precisamente su habilidad para predecir el estado de un sistema en el pasado, presente y futuro, aun cuando la naturaleza precisa del sistema modelado es desconocida. En la práctica, las variables estado individuales de un sistema dinámico no pueden ser exactamente determinadas por una medición directa. Dado lo anterior, su medición se realiza por medio de procesos estocásticos que involucran algún grado de incertidumbre en la medición. [Ramirez 2003]

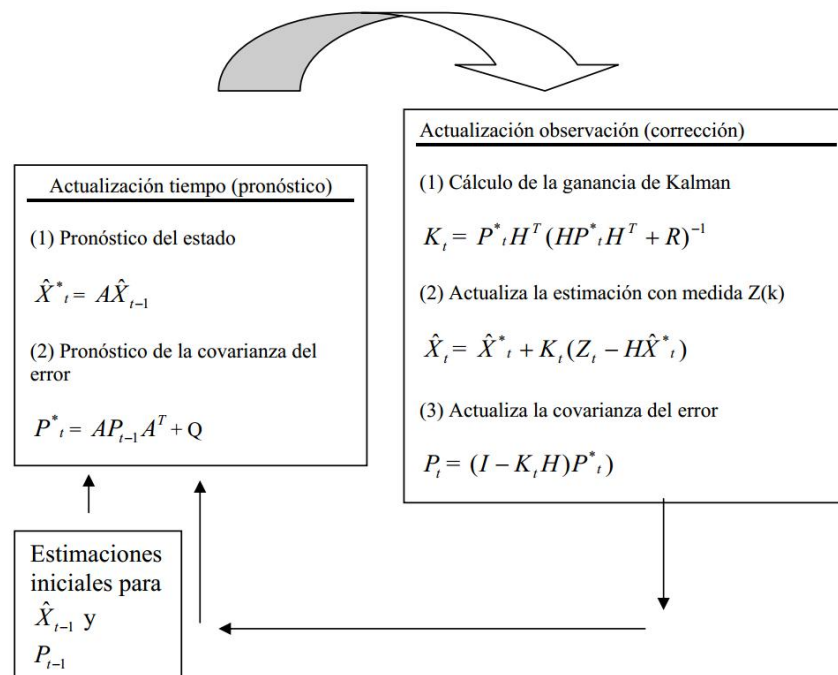


Ilustración 2. 7 Una visión del filtro de Kalman

Fuente: [Ramírez 2003]

II.15 Robot Operating System (ROS)

Es una plataforma de software para robótica desarrollada en 2007 por el Laboratorio de Inteligencia Artificial de Stanford (Stanford Artificial intelligence Laboratory). Aunque no es propiamente un sistema operativo, provee librerías y herramientas para ayudar a los desarrolladores de software en la creación de aplicaciones de robots. Está provisto de abstracción de hardware, controladores de diversos dispositivos, visualizadores, pase de mensajes, manejo de paquetes entre otras características. Unos de sus principales rasgos distintivos es el hecho de que es completamente “open source”, es código abierto bajo el estilo de licencia BSD, es libre de usarse, cambiarse y comercializarse. El objetivo principal de ROS es permitir, o facilitar propiamente, a los desarrolladores el diseño construcción y generación de robots cada vez más capaces, consiguiendo aplicaciones de forma sencilla y rápida. [Álvarez 2012]

Fue creado con la finalidad de integrar a gran escala una gran diversidad de sistemas robóticos, que para la actualidad simplifica la escritura de software para robots debido a su constante cambio y crecimiento. [Álvarez 2012]

Objetivos Filosóficos de ROS:

- Red punto a punto: un número de procesos, con posibilidades de diferentes anfitriones conectados en tiempo real por una topología punto a punto.
- Multilenguaje: está diseñado para soportar diferentes lenguajes de programación (C++, Python, Octave y LISP). Para esta característica se emplea un lenguaje neutral y se hace uso de una interfaz de definición de lenguaje, donde se establecen los mensajes que serán empleados para la comunicación entre los módulos.
- Herramientas: para facilitar el manejo de ROS se cuenta con varias herramientas, utilizadas para generar los diversos componentes de ROS.
- Ligero: los algoritmos generados para la operatividad de los robots existentes podrían ser reutilizados fuera del proyecto.

- Software Libre: el código fuente de ROS es público y se encuentra disponible. Distribuido bajo la licencia BSD que permite el desarrollo de proyectos comerciales como de investigación.

Fundamentos:

Las bases operacionales en las que se encuentra implementado ROS son:

- Nodos: son los módulos de software (programas) que componen el sistema.
- Mensajes: son los datos que se utilizan para la comunicación entre los nodos.
- Tópico: es la configuración de la comunicación en los nodos (esta puede ser escuchar o hablar)
- Cliente-Servidor: es el sistema de comunicación definido por una llamada a un servicio, en la que un cliente realiza una petición y un servidor responde a esta.

CAPÍTULO III - Metodología

Para el desarrollo de este Trabajo Especial de Grado se utilizó una adaptación de la metodología de desarrollo de software en espiral. El modelo en espiral plantea que el software debe ser desarrollado en una serie de entregas incrementales. Durante las primeras iteraciones, la entrega incremental puede ser un modelo en papel o un prototipo. A lo largo de las últimas iteraciones, versiones en aumento más completas de los sistemas desarrollados son producidas. [Pressman 2001].

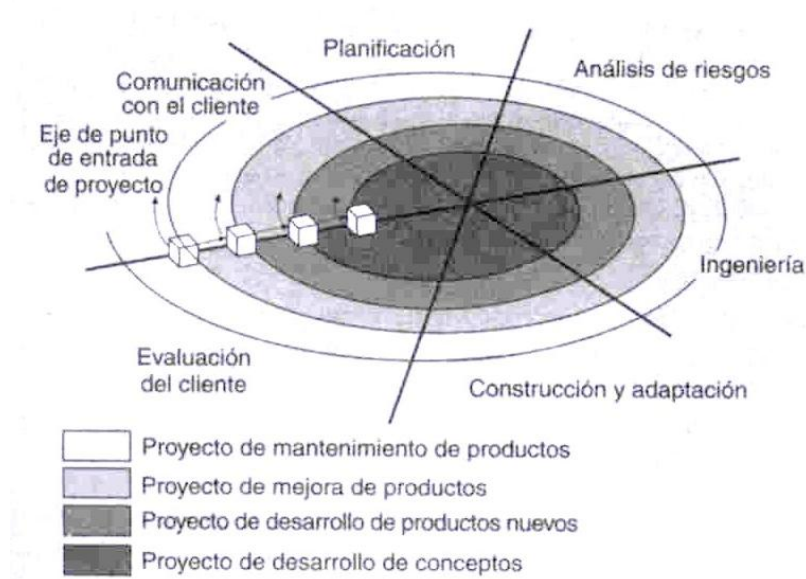


Ilustración 3. 1 Ciclo de vida en espiral.

Fuente:[Pressman 2011]

II.1 Descripción de las etapas de la metodología en espiral

El modelo espiral, originalmente propuesto por Barry Boehm, es un modelo de proceso de software evolutivo que acopla la naturaleza iterativa de prototipos con los aspectos controlados y sistemáticos de un modelo lineal secuencial de desarrollo, ofreciendo el potencial para el desarrollo rápido de versiones incrementales del software. Usando el modelo de espiral, el software se desarrolla en una serie de versiones

incrementales. Durante las primeras iteraciones, la versión incremental podría ser un modelo en papel o prototipo. Durante iteraciones posteriores, se producen versiones cada vez más completas del sistema de ingeniería.[Pressman 2001]

Un modelo en espiral está dividido en un número de etapas definidas, también llamadas regiones de tareas. Típicamente, hay entre tres (3) y seis (6) como se describen a continuación:

- **Comunicación con el cliente:** etapa donde se agrupan las tareas que son necesarias para establecer la comunicación entre el desarrollador y el cliente.
- **Planificación:** etapa donde se agrupan las tareas necesarias para planificar el proyecto resultado de los requerimientos, definiendo los recursos a utilizar, el tiempo y otras informaciones necesarias para el proyecto.
- **Análisis de riesgos:** etapa donde se agrupan las tareas que evalúan riesgos técnicos y de gestión del proyecto.
- **Ingeniería:** etapa donde se agrupan las tareas necesarias para construir una o más representaciones de la aplicación.
- **Construcción y adaptación:** etapa donde se agrupan las tareas necesarias para construir, probar, instalar y proporcionar soporte al usuario.
- **Evaluación el cliente:** etapa donde se agrupan las tareas necesarias para obtener la validación del cliente según la evaluación de las representaciones del software creadas durante la etapa de ingeniería e implementación durante la etapa de instalación.

El modelo en espiral es bastante flexible para adaptarse y aplicarse en todo el ciclo de vida del software, además permite aplicar en enfoque de construcción de prototipo en cualquier etapa. A medida que el software evoluciona, este modelo permite al desarrollador y al cliente comprender mejor los riesgos y reaccionar en cada etapa del proceso evolutivo.[Pressman 2001]

II.2 Justificación de la metodología a utilizar

La elección de la metodología en espiral se debe principalmente, a que el desarrollo de componentes de software en robótica suele necesitar de un constante proceso de diseño, implementación, pruebas, y rediseño, con el fin de perfeccionar el producto final con cada iteración realizada. Esto porque, aunque pudiera realizarse un análisis previo de los requerimientos del software, es posible que hayan ciertas condiciones propias del sistema robótico que no se hayan podido observar antes del desarrollo del primer prototipo, y sí se puedan observar al realizar un rediseño basado en las fortalezas y debilidades de los prototipos anteriormente desarrollados.

Otra razón de peso por la que se eligió el modelo de desarrollo en espiral es porque el proyecto a realizar es de carácter experimental, y para alcanzar un desarrollo óptimo es muy probable que antes tengan que desarrollarse una serie de prototipos que permitan al equipo de desarrollo el análisis de las características cuya implementación pudiera facilitar la consecución de la investigación a realizar.

CAPÍTULO IV - Desarrollo

Para llevar a cabo el desarrollo del presente Trabajo Especial de Grado se partió de un chasis de cuadricóptero Draganflyer V, con su estructura de tubos, cabina central, motores, transmisión y propelas. Se diseñaron circuitos para el control y alimentación de los motores del cuadricóptero, y para servir de interfaz entre la unidad de control Arduino Nano y los sensores y módulos de comunicación utilizados, los cuales son descritos en los apartados IV.1 y IV.2.

Se desarrollaron algoritmos para la estimación de posición angular, velocidad angular, y posición y velocidad lineal en el eje z del cuadricóptero (altura y velocidad de elevación). Para más información sobre los algoritmos implementados pueden consultarse los apartados IV.3 y IV.4 de este capítulo.

Se desarrolló una interfaz de comunicación inalámbrica para llevar a cabo comando remoto del cuadricóptero y telemetría, la cual es descrita con detalle en el apartado IV.5.

Se adaptó y linealizó el modelo dinámico del cuadricóptero Draganflyer V desarrollado en [Kivrak 2006], se analizó la estabilidad del sistema lineal obtenido en condiciones de vuelo en posición horizontal, y se simuló exitosamente dos arquitecturas de controladores PID para estabilizar al cuadricóptero en vuelo horizontal y asegurar el seguimiento de consignas de movimiento simple por parte del usuario. Se implementaron las arquitecturas de control desarrolladas, y se realizaron pruebas. El proceso citado anteriormente se expone con más detalle en el apartado IV.6.

Finalmente, se desarrolló una plataforma de pruebas, que consta de un software de control y telemetría del cuadricóptero implementado sobre la plataforma Robot Operating System, una serie de scripts para el análisis de datos de los sensores del cuadricóptero en el dominio de la frecuencia, y estrategias para realizar pruebas de vuelo bajo condiciones controladas. La plataforma de pruebas en cuestión se expone en el apartado IV.7.

Los circuitos desarrollados, la Unidad de Medición Inercial, el sensor ultrasónico de distancia, los módulos de comunicación XBEE y las baterías fueron montados en el chasis de cuadricóptero Draganflyer V como se ilustra a continuación:

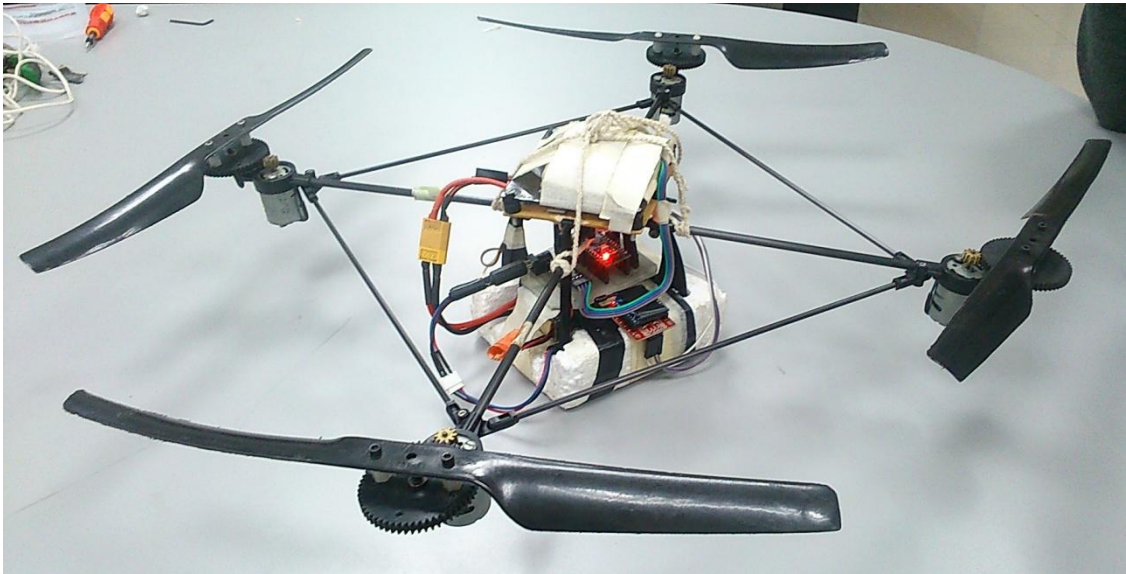


Ilustración 4. 1 Cuadricóptero ensamblado

Fuente: elaboración propia.

En el diagrama que se presenta en la página siguiente se describe la arquitectura general del sistema implementado, incluyendo las interfaces de comunicación desarrolladas, los algoritmos de estimación, la plataforma de software en Robot Operating System y los sistemas de control:

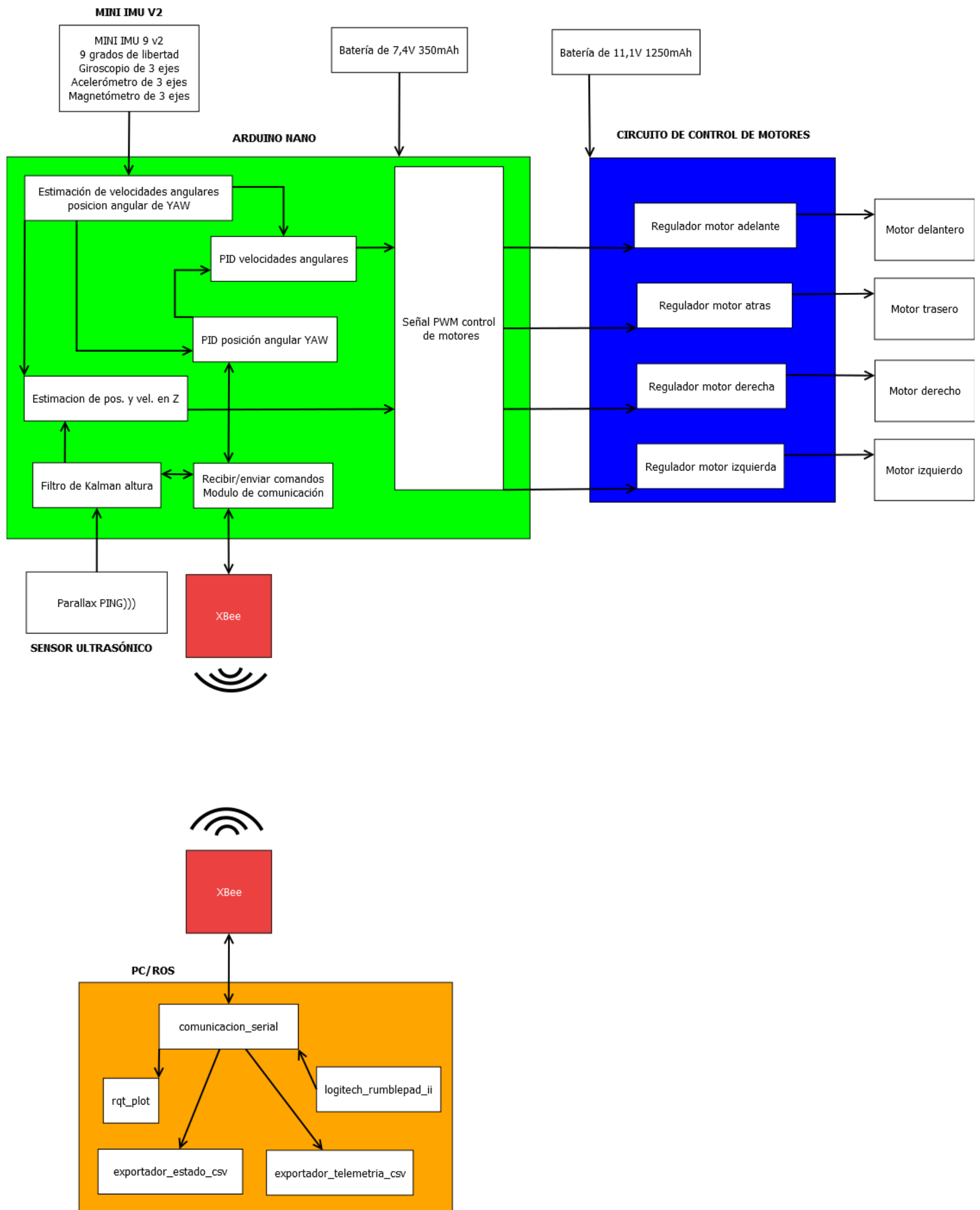


Ilustración 4. 2 Diagrama general del sistema.

Fuente: elaboración propia

IV.1 Circuito de alimentación y control de motores

Se desarrolló un circuito para llevar a cabo la alimentación y regulación de velocidad de los motores de corriente continua del chasis de cuadricóptero Draganflyer V utilizado en el presente trabajo. En primer lugar, se decidió utilizar una batería de Polímero de Litio (Li-Po, del inglés Lithium Polymer) de 3 celdas, ya que es la configuración de alimentación de los motores de corriente continua recomendada por el fabricante en [Draganfly Innovations 2006]. Por otro lado, en [Nadales 2009], se describen las características de funcionamiento de los motores en cuestión, y la relación entre el voltaje de alimentación y la corriente consumida por los mismos al hacer girar las hélices del cuadricóptero. A continuación se detallan las características de los motores citadas previamente:

- Modelo de motores DC: Mabuchi RC 280SA.
- Voltaje de alimentación recomendado: 4,5 a 12 voltios.
- Consumo sin carga: 0,14 amperios.
- Consumo medio: 4 amperios.
- Peso individual: 47 gramos.

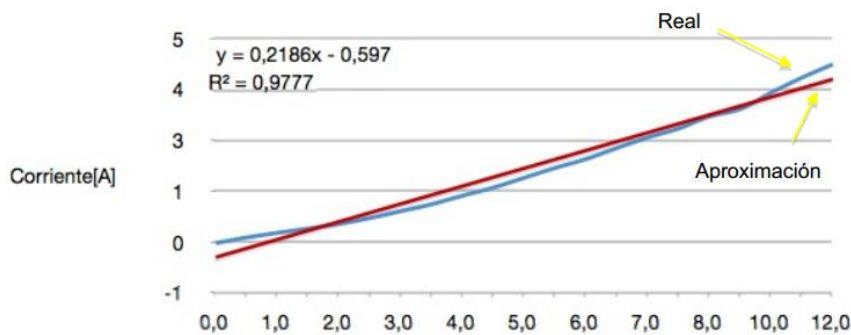


Ilustración 4. 3 Relación voltaje-corriente de los motores DC del chasis Draganflyer V.

Fuente: [Nadales 2009]

Tomando en cuenta las características de los motores anteriormente expuestas, se puede estimar un consumo promedio de 16 amperios al alimentar los cuatro motores de forma

continua, y de hasta 24 amperios considerando un pico de consumo de hasta 2 amperios por motor al realizar el encendido de los mismos, los cuales deben ser provistos por la batería de polímero de litio de forma instantánea. Para una batería de polímero de litio cualquiera, se tiene que:

$$CorrienteDescarga_{Max} = (Capacidad_{Bateria} * Tasa_{Descarga}) \frac{Amperios}{Segundo} \quad (1)$$

$$DuraciónDescarga_{Max} = \left(60 * \frac{Capacidad_{Bateria}}{CorrienteDescarga_{Requerida}} \right) \text{ Minutos} \quad (2)$$

Para satisfacer la necesidad de consumo de los motores del cuadricóptero se utilizó una batería Li-Po Yuntong de 3 celdas, con una capacidad de 1250 miliamperios por hora (mAH), y una tasa de descarga de 25C. Para la batería seleccionada se estimaron los siguientes valores de consumo y duración, asumiendo un ciclo de trabajo máximo por parte de los motores:

$$CorrienteDescarga_{Max} = 1250 \frac{miliAmperios}{Hora} * 25C \quad (3)$$

$$CorrienteDescarga_{Max} = 31,25 \frac{Amperios}{Segundo} \quad (4)$$

$$DuraciónDescarga_{Max} = 60 * \frac{1,25 \frac{Amperios}{Hora}}{16 Amperios} \quad (5)$$

$$DuraciónDescarga_{Max} \sim 5 \text{ Minutos} \quad (6)$$

Para regular la velocidad de los motores de corriente continua del cuadricóptero, se utilizó Modulación por Ancho de Pulso (PWM, del inglés Pulse Width Modulation), enviada desde los puertos de la tarjeta Arduino Nano 3.0, para realizar una conmutación rápida entre los estados de encendido y apagado de los motores, pudiendo mantener un control de lazo abierto sobre su velocidad a partir del ciclo de trabajo del PWM emitido.

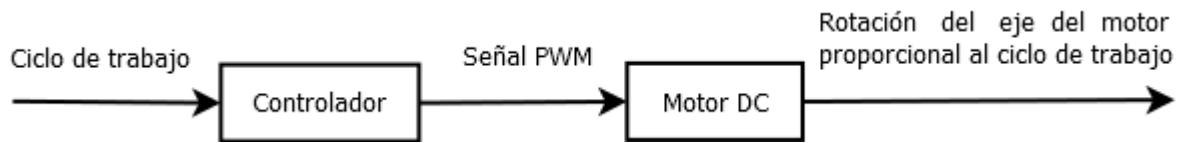


Ilustración 4. 4 Diagrama de bloques del sistema de control de velocidad de los motores DC del cuadricóptero.

Fuente: elaboración propia.

El circuito de control de motores se desarrolló en dos fases:

1. **Primera fase:** se elaboró un prototipo simple con componentes existentes en el mercado venezolano, asumiendo un comportamiento ideal de los mismos en conjunto. Se detectó una alta tasa de disipación, y con ello, una reducción en la velocidad de los motores del cuadricóptero en funcionamiento.
2. **Segunda fase (final):** para disminuir la disipación presentada en el primer prototipo, se caracterizó la respuesta de los mismos ante las cargas de los motores del cuadricóptero, y se modeló el proceso de descarga de la batería para asegurar disipación mínima en los componentes del circuito, y un máximo consumo de corriente por parte de los motores del cuadricóptero. El circuito desarrollado tuvo un rendimiento satisfactorio en pruebas, y fue utilizado para llevar a cabo las pruebas de vuelo del cuadricóptero.

A continuación se expone el circuito impreso construido durante la segunda fase de desarrollo:

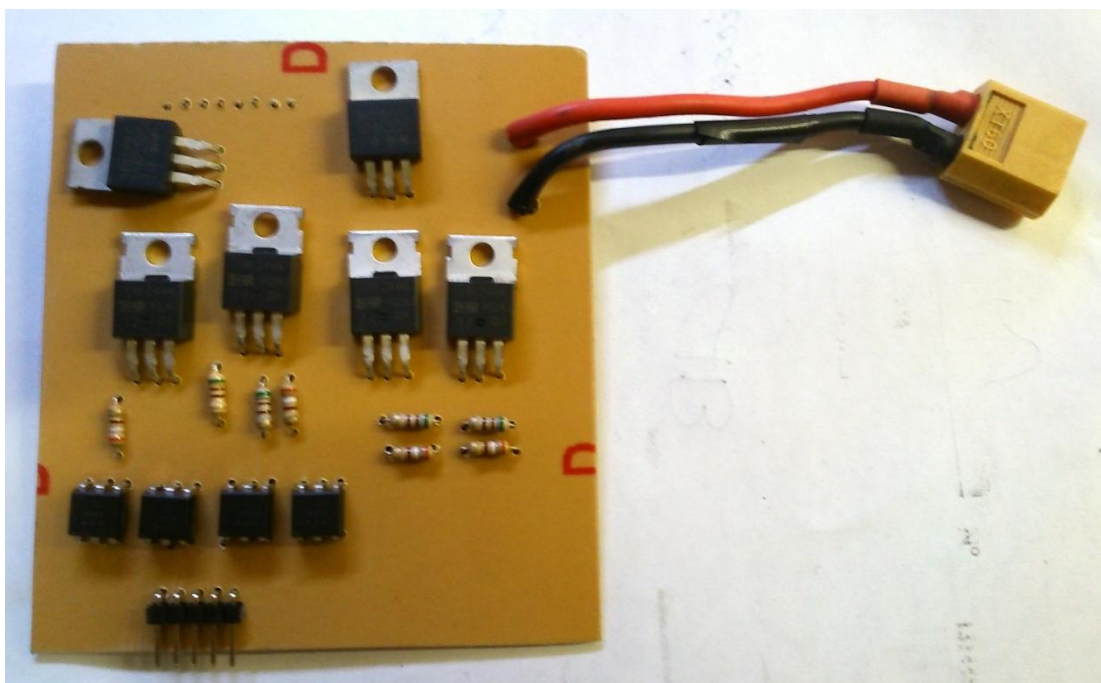


Ilustración 4. 5 Placa del circuito de alimentación y control de motores.

Fuente: elaboración propia.

Para más detalles acerca del proceso de desarrollo del prototipo final del circuito de control de motores, consúltese el *Apéndice A* del presente trabajo.

IV.2 Circuito de lógica, sensores y comunicación

Se diseñó una placa de circuito que consta de una serie de puertos pin header para situar la tarjeta Arduino y la unidad de medición inercial, además de conectar el sensor de ultrasonido, las salidas de PWM y el módulo de XBEE mediante cables “jumper”. La misma sirve de interfaz de comunicación entre la tarjeta Arduino, y los demás módulos del cuadricóptero. Además, permite la alimentación de todo el circuito haciendo uso de la etapa de regulación de voltaje embebida en la tarjeta Arduino, la cual puede ser alimentada con voltajes de entre 6 y 20 voltios. A continuación se detalla el consumo promedio de cada componente del circuito, y el consumo total del mismo:

Tabla 4. 1 Desglose del consumo promedio por componente, y acumulado, del circuito de lógica, sensores y comunicación.

Componente	Consumo promedio
Arduino Nano	50 mA
Pololu miniIMU-9 V2	10mA
XBee Series 1mW Wire Antenna – Series 1	50mA
Parallax Ping	30mA
4 Optocouplers 4N26 – Módulo de control de motores	240 mA
Total	380mA

Fuente: elaboración propia.

Para alimentación del módulo de lógica, sensores y comunicación durante la realización de pruebas de vuelo se utilizó una batería de polímero de litio e-Flite de 300mAH, con un coeficiente de descarga de 3C. A continuación se presenta el cálculo de la máxima capacidad de descarga, y duración de la carga de la misma al alimentar el circuito desarrollado:

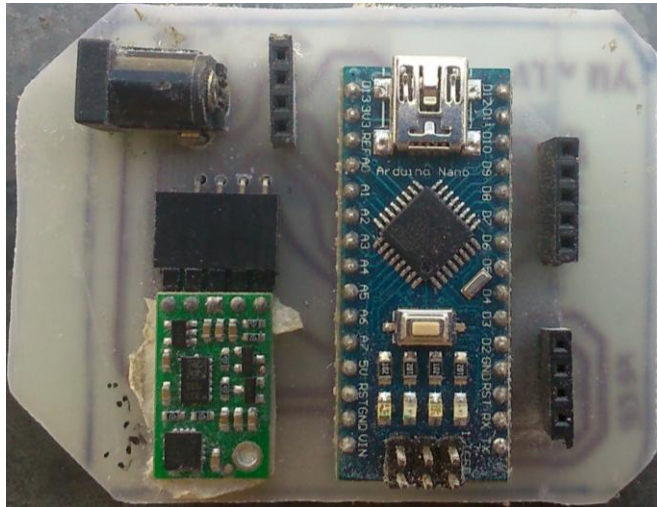
$$CorrienteDescarga_{Max} = 300 \frac{\text{miliAmperios}}{\text{Hora}} * 3C \quad (7)$$

$$CorrienteDescarga_{Max} = 0,9 \frac{\text{Amperios}}{\text{Segundo}} \quad (8)$$

$$DuraciónDescarga_{Max} = 60 * \frac{0,3 \frac{\text{Amperios}}{\text{Hora}}}{0,38 \text{ Amperios}} \quad (9)$$

$$DuraciónDescarga_{Max} \sim 47 \text{ Minutos} \quad (10)$$

En la ilustración que sigue se presenta una fotografía del circuito desarrollado:



Ilustracion 4. 6 Placa del circuito de lógica, sensores y comunicación.

Fuente: elaboración propia.

En la fase de pruebas de los algoritmos de control se encontró un error de diseño en el circuito desarrollado, por una mala asignación de los puertos de PWM del Arduino Nano que genera un problema de contención de recursos del temporizador timer2 del Arduino Nano, con la librería Ping para el manejo del sensor ultrasónico de distancia. El problema en cuestión impide la utilización del sensor ultrasónico de distancia mientras se envían señales de PWM a los motores del cuadricóptero.

IV.3 Estimación de orientación del cuadricóptero

IV.3.1 Convenciones respecto a los ángulos

Para simplificar la implementación de los sistemas de control del cuadricóptero, se decidió representar la posición y velocidad angular del mismo en cada eje mediante ángulos de Euler, bajo la siguiente convención:

- Ángulo de Yaw: ángulo de giro respecto al eje Z del acelerómetro y giroscopio.
- Ángulo de Pitch: ángulo de giro respecto al eje Y del acelerómetro y giroscopio.
- Ángulo de Roll: ángulo de giro respecto al eje X del acelerómetro y giroscopio.
- Velocidad de Yaw: velocidad de giro respecto al eje Z del giroscopio.
- Velocidad de Pitch: velocidad de giro respecto al eje Y del giroscopio.
- Velocidad de Roll: velocidad de giro respecto al eje X del giroscopio.

IV.3.2 Descripción de la Unidad de Medición Inercial (IMU)

Se utilizó una tarjeta Pololu MinIMU-9 v2, la cual está constituida por un giroscopio de tres (3) ejes L3GD20, y por un acelerómetro de tres (3) ejes y un compás de tres (3) ejes LSM303DLHC. La misma provee de una interfaz I²C que permite acceder a las mediciones, por eje, de cada uno de sus sensores, e incluye un regulador de voltaje y un convertidor de nivel de voltaje que permite su operación con una entrada de voltaje de 2,5V a 5,5V.

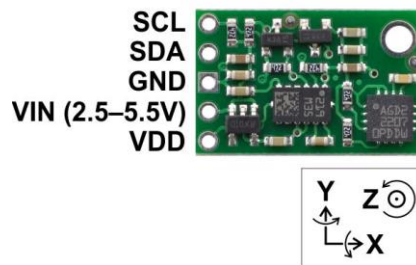


Ilustración 4. 7 Pololu minIMU-9 V2.

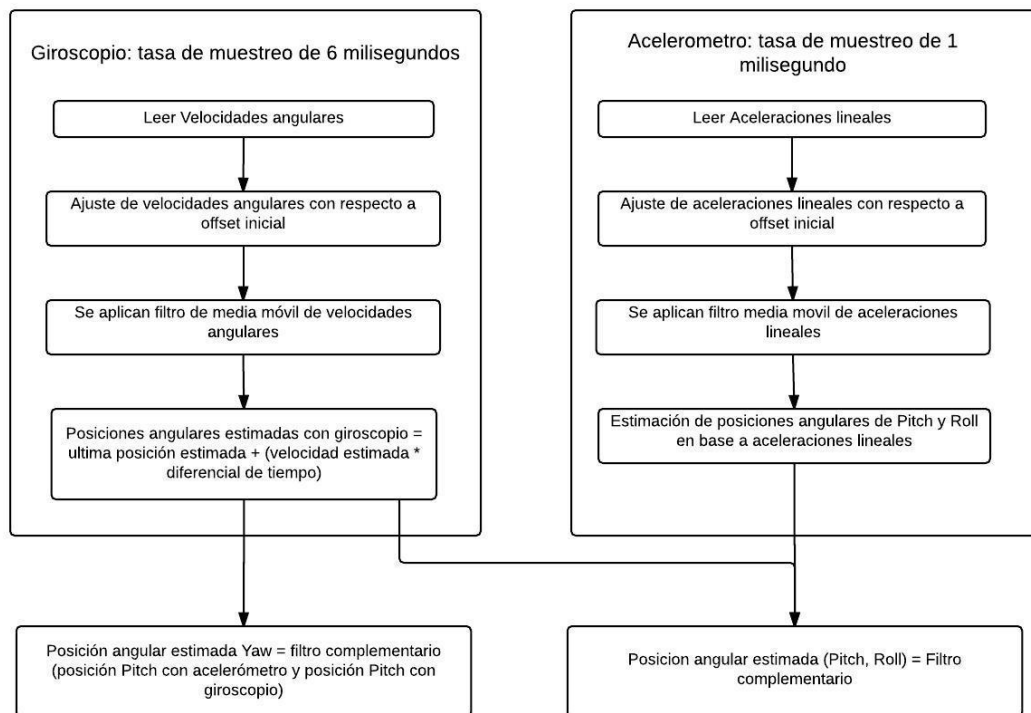
Fuente: <http://www.pololu.com/product/1268>

Para la obtención de datos de la tarjeta Pololu MinIMU-9 v2 se utilizaron las librerías diseñadas por el fabricante para su manejo desde tarjetas Arduino.

IV.3.3 Algoritmo de estimación de posición angular y velocidad angular del cuadricóptero

Para estimar la posición angular y velocidad angular del cuadricóptero se desarrolló un modelo matemático del problema, con base en las características de los sensores acelerómetro y giroscopio. El modelo en cuestión se encuentra en el *Apéndice B* del presente trabajo.

A continuación se describe el algoritmo de estimación de posición angular y velocidad angular desarrollado:



Ilustracion 4. 8: funcionamiento del algoritmo de estimación de posición y velocidad angular

Fuente: elaboración propia.

IV.4 Estimación de posición y velocidad lineal en el eje Z del cuadricóptero

IV.4.1 Descripción del sensor de altura utilizado

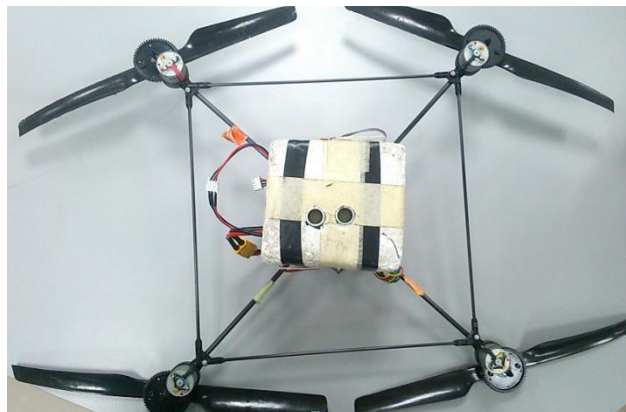
Para la estimación de altura del cuadricóptero se utilizó un sensor Parallax Ping, el cual opera con niveles lógicos TTL, provee un rango de alcance de entre 2cm y 3m, y un peso de 9 gramos. Adicionalmente, el sensor porta un LED en su cara frontal, que indica si el proceso de estimación de altura se está realizando de forma efectiva.



Ilustracion 4. 9 Sensor ultrasónico Parallax Ping.

Fuente: <http://www.parallax.com/product/28015>

El sensor ultrasónico de distancia se ubicó en la parte inferior del cuadricóptero, debajo de las baterías, y dentro de una base de anime esculpida para amortiguar el aterrizaje del cuadricóptero y sostener al sensor ultrasónico en una posición fija.



Ilustracion 4. 10: posición del sensor ultrasónico de distancia en el cuadricóptero.

Fuente: elaboración propia.

IV.4.2 Algoritmo de estimación de posición y velocidad en el eje z del cuadricóptero

Se desarrolló un modelo matemático del proceso de estimación a partir de las características de los sensores utilizados. El mismo puede encontrarse en el *Apéndice C* del presente trabajo.

A continuación se describe el algoritmo desarrollado para la estimación de posición y velocidad en el eje Z del cuadricóptero:

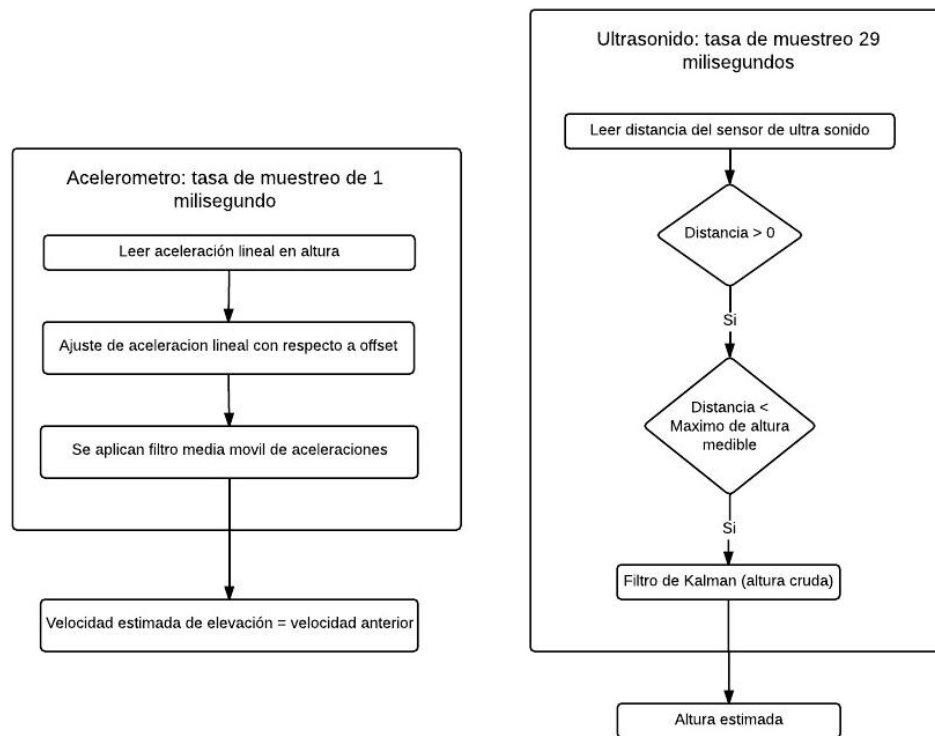


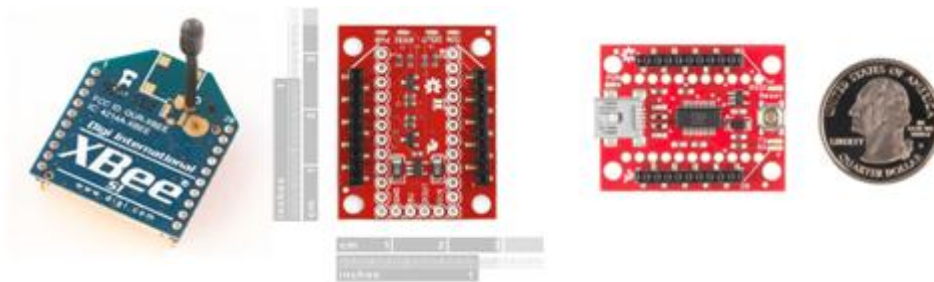
Ilustración 4. 11: funcionamiento del algoritmo de estimación de posición y velocidad lineal en el eje Z.

Fuente: elaboración propia.

IV.5 Comunicación inalámbrica

a. Descripción de las características y configuración de los módulos XBEE utilizados

Se utilizaron dos (2) tarjetas XBee 1mW Series 1 con antena de cable, una tarjeta XBee Explorer USB y una tarjeta XBee Explorer Regulated. Esto porque las tarjetas XBEE 1mW Series 1 proporcionan un alcance de hasta 100 metros planos, y un bajo consumo eléctrico, mientras que las tarjetas Sparkfun XBee Explorer USB y Xbee Explorer Regulated permiten utilizar los módulos XBee como una interfaz serial, por ende, facilitando la programación para la comunicacion entre la PC y el cuadricóptero.



Ilustracion 4. 12 (de derecha a izquierda) Modulo XBee Series 1 de 1mW, XBee Explorer Regulated y XBee Explorer USB.

Fuentes: <http://examples.digi.com/wp-content/uploads/2012/06/1x-XBee-S1-300x300.jpg>
<https://cdn.sparkfun.com/assets/parts/7/1/1/5/11373-02.jpg>
<https://cdn.sparkfun.com/assets/parts/8/1/4/0/11812-03.jpg>

Especificaciones técnicas de los modulos XBee:

Tabla 4. 2 Especificaciones del modulo XBee

Características	ModuloXbee
Rendimiento	
Rango con obstáculos	Hasta 100 pies (30 mts)
Rango sin obstaculos	Hasta 300 pies (90 mts)
Potencia de salida transmitida (seleccionable por software)	1mW (0 dBm)
Tasa de datos en radio frecuencia	250,000 bps

Frecuencia de datos por interfaz serial (seleccionable por software)	1200 bps - 250 kbps (frecuencias sin estandar tambien son soportadas)
Sensibilidad de recepcion	-92 dBm (1% tasa de error de 1%)
Requerimientos de poder	
Voltaje de Entrada	2.8 – 3.4 V
Corriente para enviar data (típica)	45mA (@ 3.3 V)
Corriente para recibir data (típica)	50mA (@ 3.3 V)
Corriente para apagarse	< 10 μ A
General	
Frecuencia de operación	ISM 2.4 GHz
Dimensiones	0,96" x 1,087" (2,438cm x 2,761cm)
Temperatura de operación	-40 to 85° C (industrial)
Opciones de antena	Integrated Whip, Chip or U.FL Connector, RPSMA Connector
Redes y Seguridad	
Topologías de red soportadas	Point-to-point, Point-to-multipoint & Peer-to-peer
Cantidad de canales (seleccionable por software)	16 Direct Sequence Channels
Opciones de direccionamiento	PAN ID, Channel and Addresses

Fuente: <https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf>

b. Descripción del protocolo de comunicación desarrollado

Por razones de velocidad de transmisión y simplicidad en la implementación se utilizaron los módulos XBEE en modo AT (transparent mode), el cual permite diseñar explícitamente los paquetes de comunicación, pudiendo así reducir su longitud al enviar datos en forma serial y diseñar todo el protocolo de comunicación aumentando la velocidad de transmisión de los mensajes.

Se ha identificado un error de comunicación entre el Arduino Nano y los módulos XBee Series 1 cuando se configuran ambos para transmitir a 57600 baudios por segundo, que roza el máximo de tolerancia soportado por la UART (*“Universal Asynchronous Receiver-Transmitter”*, en español, *Transmisor-Receptor Asíncrono Universal*) [Colton 2011 - 1]. Con base en este hecho, se decidió utilizar una frecuencia de transmisión de 38400 baudios por segundo para la realización de pruebas de vuelo, y de 115200 baudios por segundo para la realización de telemetría con fines de análisis de datos de los sensores del cuadricóptero en reposo.

Para reducir al mínimo el tamaño de los paquetes a enviar, se decidió codificar los datos a transmitirse por las interfaces inalámbricas en bytes, como es recomendado en [Colton 2011 – 2], lo cual limitó el rango de valores a enviar mediante la interfaz diseñada,

pudiendo enviarse datos de tipo entero dentro del rango numérico de cero (0) a doscientos cincuenta y cinco (255). Para los casos en que se necesitaba un mayor rango, se decidió dividir los paquetes según su signo, pudiéndose enviar, finalmente, datos dentro del rango numérico de los enteros, de cero (0) a quinientos doce (512).

Para las interfaces de comunicación se desarrolló la siguiente estructura general de los mensajes:

Tabla 4. 3 Estructura general de los mensajes

Cantidad bytes	1	1	1-42	1
Contenido	Cabecera del mensaje (255)	Código del mensaje	Contenido del mensaje	Checksum

Fuente: elaboración propia

Siendo:

- **Cabecera de mensaje:** identificador que indica cuándo inicia un mensaje, siempre será el valor 255.
- **Código de mensaje:** identificador propio de cada tipo de mensaje.
- **Contenido del mensaje:** data útil del mensaje, su longitud varia de 1 a 42 bytes dependiendo del tipo de mensaje.
- **Checksum:** resultado de aplicar la función de compendio XOR a los bytes de la cabecera, código y contenido de cada mensaje. Es utilizado para validar que el mensaje llegue correctamente al destino.

Para este TEG se diseñaron cinco (5) tipos de mensajes diferentes, dos (2) mensajes de telemetría, dos (2) mensajes de comando y un (1) mensaje de acuse de recibo (del inglés “acknowledge”). Los mensajes diseñados fueron los siguientes:

- **Mensaje de estado:** es el paquete que contiene la información de telemetría del

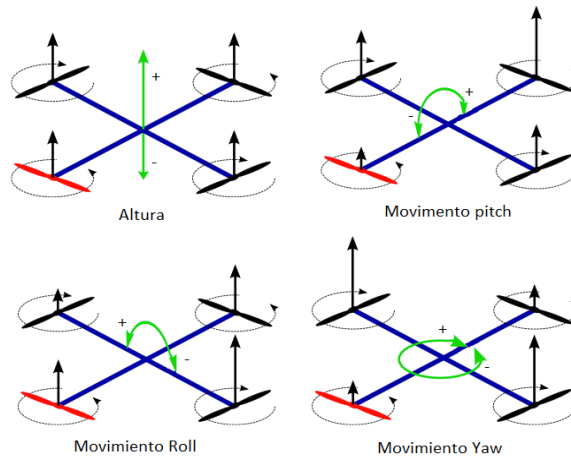
cuadricóptero y su contenido está compuesto por 12 bytes, los cuales indican posiciones y velocidades para los tres (3) ejes del cuadricóptero con respecto a un eje fijo imaginario, además tiene información sobre la altura y si los motores se encuentran encendidos o apagados.

Debido a que tanto la posición como la velocidad pueden ser negativas o positivas y el mayor valor decimal que puede ser contenido en un byte es doscientos cincuenta y cinco (255) se decidió separar los datos cuyo rango de valores superara esta cota máxima, como se expone a continuación:

- Para el eje yaw la velocidad angular puede variar de -255 a 255 y la posición puede variar entre -180 y 180 grados
- Para el eje pitch la velocidad angular puede variar de -255 a 255 y la posición puede variar entre -90 y 90 grados.
- Para el eje roll la velocidad angular puede variar de -255 a 255 y la posición puede variar entre -90 y 90 grados.
- Las aceleraciones lineales y la velocidad en el eje z pueden poseer valores en un rango de -255 a 255.
- Los valores de PWM enviados a los motores se encuentran siempre en el rango de 0 a 255.
- La altura siempre será un valor positivo debido a que representa la distancia entre el cuadricóptero y el piso, su máximo valor es de dos metros y cuarenta centímetros (2,4 metros).
- El estado de los motores indica si estos están encendidos (0) o apagados (1).

- **Mensaje de acknowledge:** es el mensaje que envía el cuadricóptero a la PC para indicar que se ha recibido un mensaje. Contiene es el código del mensaje recibido.
- **Mensaje de encendido de motores:** es el comando que envía la PC al cuadricóptero para apagar o encender los motores.

- **Mensaje de movimiento:** es el comando que se envía para mover al cuadricóptero en el eje roll (derecha o izquierda), Pitch (adelante o atrás) o altura (para subir y bajar el cuadricóptero).



Ilustracion 4. 13 Diagrama de movimientos del cuadricóptero y signo que toman los valores de aceleración y posición.

Fuente: <http://3.bp.blogspot.com/-B8llmJJHUI0/U0cj-KWg95I/AAAAAAAAA/gkaMEopdgCwc/s1600/Screen+Shot+2014-04-10+at+4.06.09+PM.png>

A continuación se expone la estructura de los mensajes de las interfaces de comunicación inalámbrica para ejecución de comandos y telemetría del cuadricóptero desarrolladas:

Tabla 4. 4 Mensajes de la interfaz de telemetría del cuadricóptero

Tipo de mensaje	Código de mensaje	Contenido del mensaje																																					
Estado	6	Posición YAW positiva	Posición YAW negativa	Posición PITCH (de 0 a 180 siendo el valor 90 igual a 0 grados)	Posición ROLL (de 0 a 180 siendo el valor 90 igual a 0 grados)	Velocidad YAW positiva	Velocidad YAW negativa	Velocidad PITCH positiva	Velocidad PITCH Negativa	Velocidad ROLL positiva	Velocidad ROLL negativa	Altura	Estado Motores																										
Telemetría completa	8	Estimado						Bruto						Filtrado						Bruto						Filtrado						Bruto	Filtrado	Estimado		PWM motor delantero	PWM motor trasero	PWM motor derecho	PWM motor izquierdo
		Posición YAW positiva	Posición YAW negativa	Posición PITCH positiva	Posición PITCH negativa	Posición ROLL positiva	Posición ROLL negativa	Velocidad YAW positiva	Velocidad YAW negativa	Velocidad PITCH positiva	Velocidad PITCH negativa	Velocidad ROLL positiva	Velocidad ROLL negativa	Velocidad YAW positiva	Velocidad YAW negativa	Velocidad PITCH positiva	Velocidad PITCH negativa	Velocidad ROLL positiva	Velocidad ROLL negativa	Aceleración YAW positiva	Aceleración YAW negativa	Aceleración PITCH positiva	Aceleración PITCH negativa	Aceleración ROLL positiva	Aceleración ROLL negativa	Aceleración YAW positiva	Aceleración YAW negativa	Aceleración PITCH positiva	Aceleración PITCH negativa	Aceleración ROLL positiva	Aceleración ROLL negativa	Altura	Altura	Velocidad Z positiva	Velocidad Z negativa				

Fuente: elaboración propia.

Tabla 4. 5 Mensajes de la interfaz de comando remoto del cuadricóptero

Tipo de mensaje	Código de mensaje	Contenido del mensaje		
Encendido de motores	0	Para apagar motores: 0 Para apagar motores: 1		
Movimiento	1	Movimiento Eje Pitch	Movimiento Eje Roll	Movimiento de altura
Acknowledge	7	Codigo mensaje recibido		

Fuente: elaboración propia

En la tabla que se presenta a continuación se expone el retardo producido por el envío de cada mensaje de las interfaces de comunicación desarrolladas:

Tabla 4. 6 retardo de mensajes

Mensaje	Retardo (medido en milisegundos)	
	38400 bdps	115200 bdps
Encendido	0,83	0,27
Movimiento	1,25	0,41
Estado	3,125	1,04
TelemetriaTotal	8,75	2,91
Acknowledge	0,83	0,27

Fuente: elaboración propia

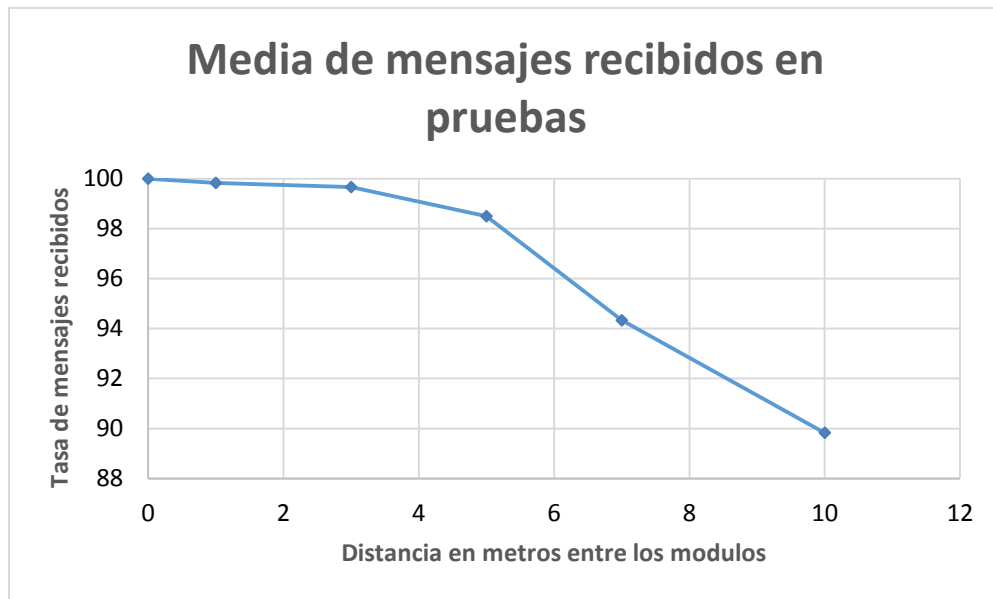
Se realizaron pruebas para medir la tasa de transmisión de mensajes en función de la distancia, a partir de las cuales se obtuvieron los siguientes resultados:

Tabla 4. 7 Pruebas realizadas con el XBee a diferentes distancias, cada prueba envía un total de 200 mensajes.

Distancia	Prueba #1	Prueba #2	Prueba #3	Media de mensajes recibidos	% de mensajes recibidos	% de mensajes no recibidos
	Mensajes recibidos	Mensajes recibidos	Mensajes recibidos			
0	200	200	200	200	100	0
1	200	199	200	200	99.83333333	0.16666667
3	199	200	199	199	99.66666667	0.33333333
5	197	194	200	197	98.5	1.5
7	196	172	198	189	94.33333333	5.66666667
10	148	200	191	180	89.83333333	10.16666667

Fuente: elaboración propia

En la gráfica que se expone a continuación se presenta la media de mensajes recibidos en función de la distancia entre los módulos XBee:



Ilustracion 4. 14 Porcentaje de mentajes recibidos en distancia

Fuente: elaboración propia

IV.6 Sistemas de control

IV.6.1 Arquitectura de control

Se diseñó una arquitectura de control multi-capa para la regulación en vuelo de la posición angular, velocidad angular, velocidad en el eje z, y posición en el eje Z (altura) del mismo, mediante algoritmos de control Proporcional-Integral-Derivativo. La arquitectura de control propuesta se describe a continuación:

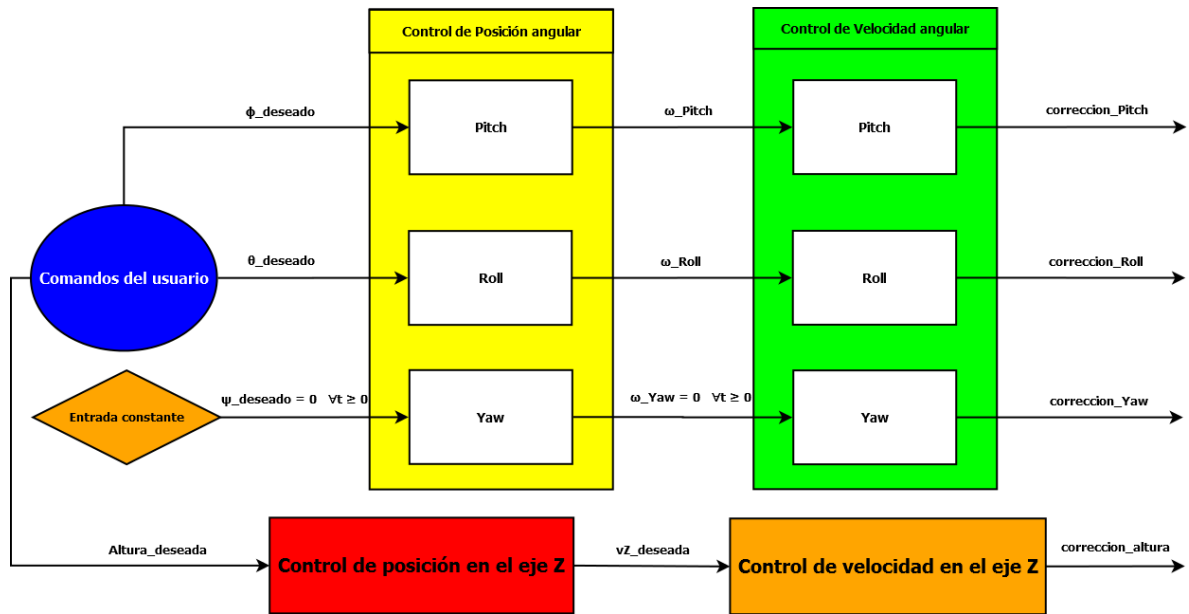


Ilustración 4. 15 Arquitectura de control propuesta

Fuente: elaboración propia.

La arquitectura de control propuesta se desglosa en:

1. Sistema de control de posición en el eje z (Altura).
 - a. Entrada: estimación de altura del cuadricóptero.
 - b. Referencia: altura deseada, especificada por el usuario en tiempo real.
 - c. Salida: velocidad lineal en z para alcanzar la altura deseada.
 - d. Intervalo de ejecución: cada 30 milisegundos.
2. Sistema de control de velocidad lineal en el eje z.
 - a. Entrada: velocidad lineal del cuadricóptero.

- b. Referencia: velocidad lineal en z deseada.
 - c. Salida: corrección de altura.
 - d. Intervalo de ejecución: cada 30 milisegundos.
- 3. Sistema de control de posición angular de Pitch.
 - a. Entrada: ángulo de Pitch del cuadricóptero.
 - b. Referencia: ángulo de Pitch deseado, especificado por el usuario en tiempo real.
 - c. Salida: velocidad angular para alcanzar la posición de Pitch deseada.
 - d. Intervalo de ejecución: cada 20 milisegundos.
- 4. Sistema de control de posición angular de Roll.
 - a. Entrada: ángulo de Roll del cuadricóptero.
 - b. Referencia: ángulo de Roll deseado, especificado por el usuario en tiempo real.
 - c. Salida: velocidad angular para alcanzar la posición de Roll deseada.
 - d. Intervalo de ejecución: cada 20 milisegundos.
- 5. Sistema de control de posición angular de Yaw.
 - a. Entrada: ángulo de Yaw del cuadricóptero.
 - b. Referencia: ángulo de Yaw=0, constante.
 - c. Salida: velocidad angular para alcanzar la posición de Yaw deseada.
 - d. Intervalo de ejecución: cada 20 milisegundos.
- 6. Sistema de control de velocidad angular de Pitch.
 - a. Entrada: velocidad angular del cuadricóptero en el eje de Yaw.
 - b. Referencia: velocidad angular de Pitch deseada, obtenida como salida del sistema de control de posición angular de Pitch.
 - c. Salida: corrección de Pitch.
 - d. Intervalo de ejecución: cada 10 milisegundos.
- 7. Sistema de control de velocidad angular de Roll.
 - a. Entrada: velocidad angular del cuadricóptero en el eje de Yaw.
 - b. Referencia: velocidad angular de Roll deseada, obtenida como salida del sistema de control de posición angular de Pitch.

- c. Salida: corrección de Pitch.
 - d. Intervalo de ejecución: cada 10 milisegundos.
8. Sistema de control de velocidad angular de Yaw.
- a. Entrada: velocidad angular del cuadricóptero en el eje de Yaw.
 - b. Referencia: velocidad angular de Yaw deseada, obtenida como salida del sistema de control de posición angular de Yaw.
 - c. Salida: corrección de Yaw.
 - d. Intervalo de ejecución: cada 10 milisegundos.

Debido a la magnitud de las vibraciones en el chasis del cuadricóptero, sólo se pudo obtener un rendimiento satisfactorio en los sistemas de control PID de velocidad angular de Yaw, Pitch y Roll, y en el sistema de control PID de Yaw. Por ello, se modificó la arquitectura de control propuesta inicialmente, para que el usuario pudiera emitir consignas de velocidad angular y velocidad global de los motores en tiempo real (Tarea que debían realizar los sistemas de control de posición angular y altura en la arquitectura propuesta). La arquitectura final implementada se presenta a continuación:

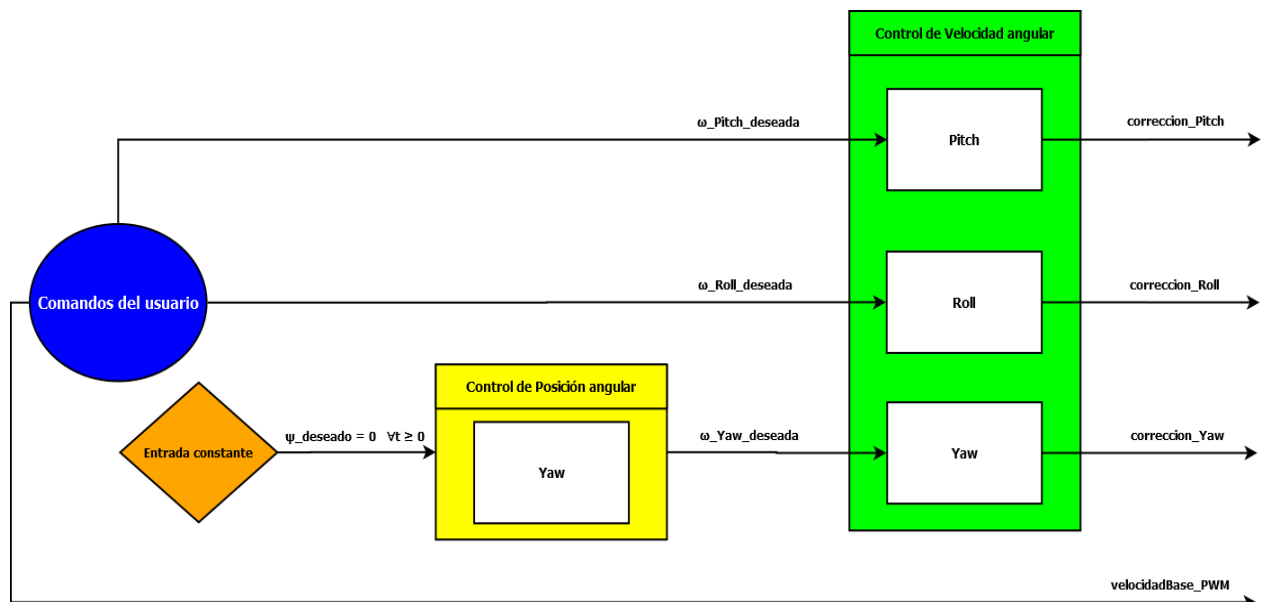


Ilustración 4. 16 Arquitectura de control simplificada

Fuente: elaboración propia.

IV.6.2 Simulación

Se partió del modelo dinámico del cuadricóptero Draganflyer V desarrollado en [Kivrak 2006]. El mismo fue adaptado al cuadricóptero utilizado en este proyecto y linealizado, como es expuesto en el *Apéndice D* del presente trabajo. El modelo linealizado fue implementado en el software MATLAB, y se realizaron simulaciones para verificar la factibilidad de realizar el control de posición angular, velocidad angular y altura del cuadricóptero mediante la arquitectura de control propuesta. Las simulaciones se llevaron a cabo mediante el método de Euler, ya que se partía del modelo linealizado del cuadricóptero y el algoritmo Proporcional-Integral-Derivativo modela la retroalimentación en lazo cerrado como una ecuación diferencial lineal.

El cálculo de los parámetros de ejecución de los algoritmos PID se realizó por prueba y error, y se obtuvo un rendimiento satisfactorio de la arquitectura de control propuesta, con lo cual se verificó la efectividad de la misma. A continuación se exponen los resultados obtenidos en la ejecución de los sistemas de control de posición angular, velocidad angular y altura (posición y velocidad lineal en el eje z):

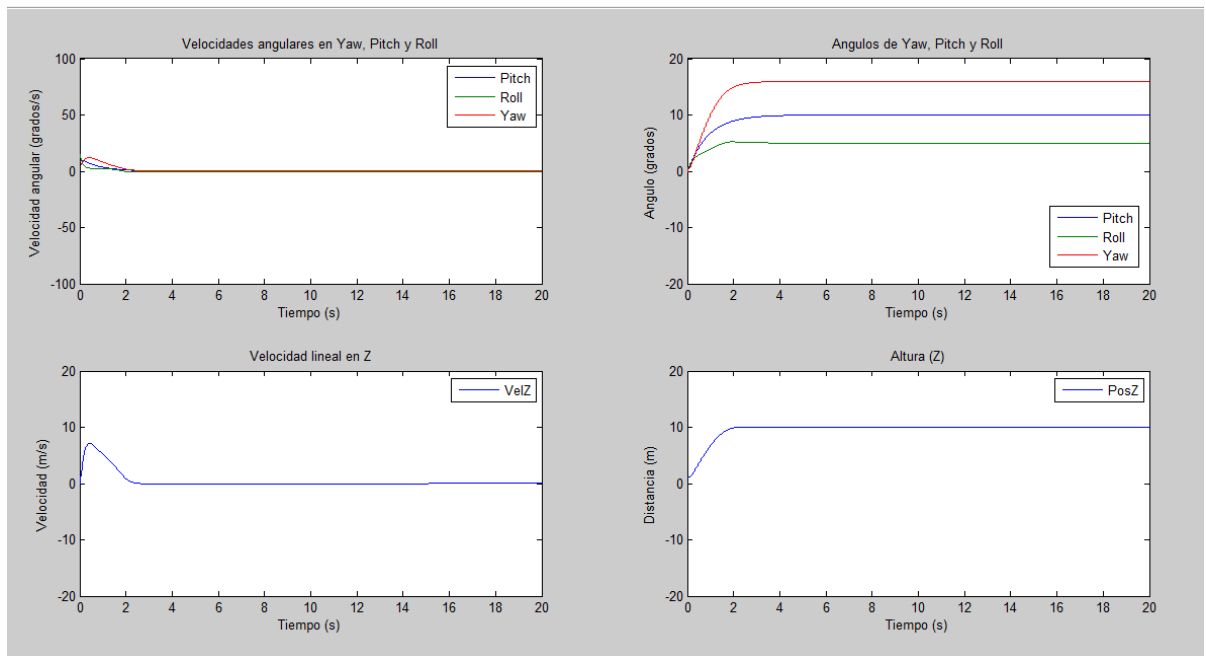


Ilustración 4. 17 Rendimiento de los sistemas de control PID de la arquitectura propuesta en simulación

Fuente: elaboración propia.

De igual manera, se obtuvo un rendimiento satisfactorio al simular la arquitectura de control simplificada que finalmente fue implementada en el cuadricóptero, como se expone a continuación:

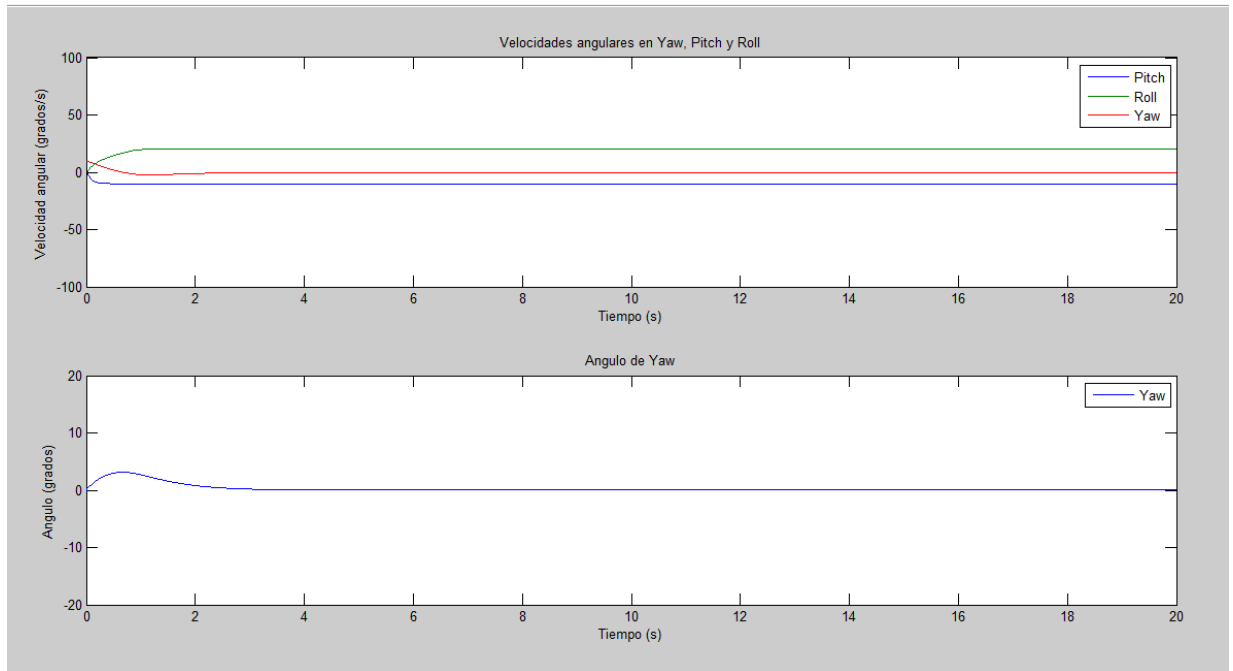


Ilustración 4. 18 Rendimiento de los sistemas de control PID de la arquitectura simplificada en simulación

Fuente: elaboración propia.

IV.6.3 Implementación en el cuadricóptero

Para la implementación de los sistemas de control se utilizó la librería PID de Arduino, la cual permite encapsular en una clase todos los atributos y métodos suficientes para la ejecución de un sistema de control Proporcional-Integral-Derivativo. El cálculo de las constantes de los sistemas de control se realizó mediante prueba y error.

A continuación se expone el flujo de ejecución de la rutina de control implementada en el micro-controlador Arduino Nano, a bordo del cuadricóptero:

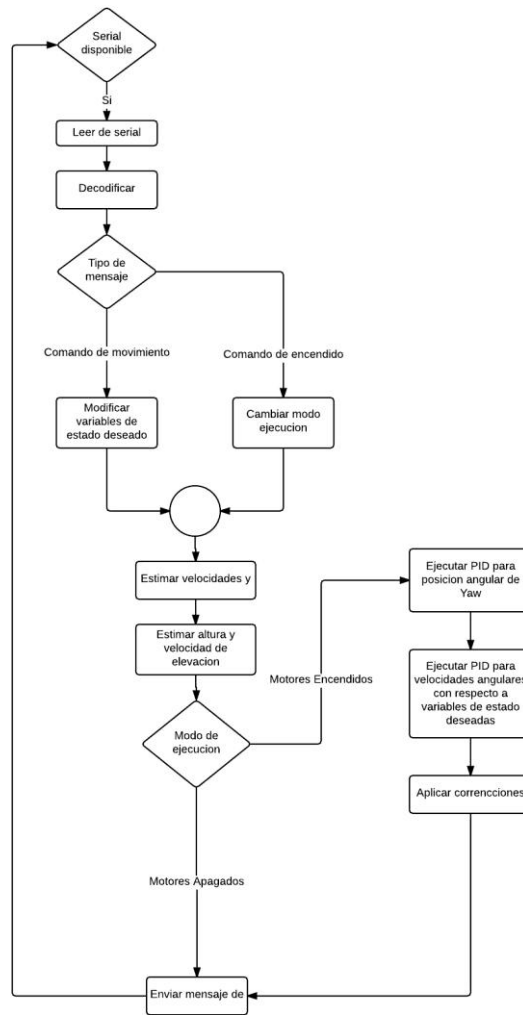


Ilustración 4. 19 Ciclo de ejecución de la rutina de estimación de estado, control y comunicación del cuadricóptero

Fuente: elaboración propia.

Se definió una velocidad base constante, para mantener homogeneidad en el comportamiento dinámico del cuadricóptero durante el proceso de calibración, pruebas, y vuelo. Las señales de Módulación por Ancho de Pulsos (PWM) a enviar a los motores se calcularon de la siguiente manera:

$$PWMMotor_{derecho} = VelocidadBasePWM + Corrección_{Altura} + Corrección_{yaw} + Corrección_{Roll}$$

$$PWMMotor_{Izquierdo} = VelocidadBasePWM + Corrección_{Altura} + Corrección_{Yaw} - Corrección_{Roll}$$

$$PWMMotor_{Delantero} = VelocidadBasePWM + Corrección_{Altura} - Corrección_{Yaw} + Corrección_{Pitch}$$

$$PWMMotor_{Trasero} = VelocidadBasePWM + Corrección_{Altura} - Corrección_{Yaw} - Corrección_{Pitch}$$

Por la inestabilidad de la estimación de posición angular del cuadricóptero producida por las vibraciones del chasis, y la imposibilidad de realizar una estimación de altura adecuada (dependiente de la estimación de posición angular), no se realizaron pruebas con la arquitectura de control propuesta.

Se realizaron pruebas de vuelo con la arquitectura de control simplificada, y se obtuvieron resultados satisfactorios de seguimiento de consignas fuera de la banda de error producida por las vibraciones del chasis. El manejo del cuadricóptero requirió de un lapso de práctica, ya que se necesita una realimentación constante por parte del usuario para ajustar la inclinación y altura del cuadricóptero en vuelo, y sólo puede asegurarse un seguimiento efectivo de consignas si éstas son superiores a 20 grados/segundo. A continuación se expone el rendimiento de los sistemas de control de velocidad angular implementados en los ejes de Pitch y Roll ante perturbaciones:

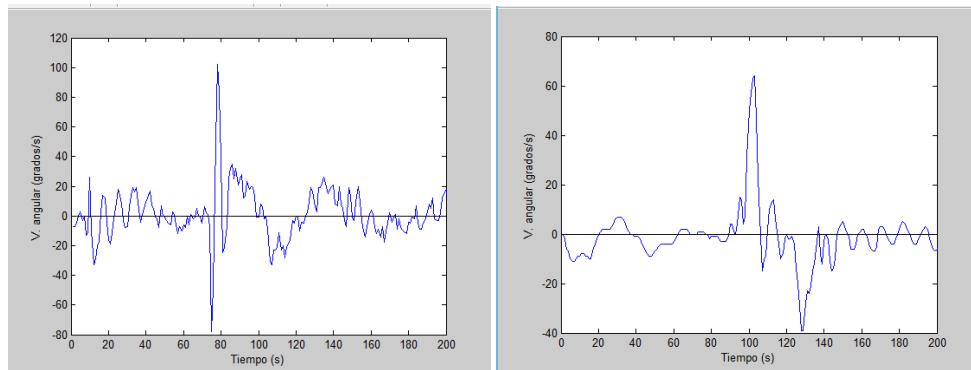


Ilustración 4. 20 Rendimiento de los sistemas de control PID de velocidad angular de Pitch (Izquierda) y Roll (Derecha) ante perturbaciones.

Fuente: elaboración propia.

Finalmente, a continuación se expone el rendimiento de los sistemas de control de posición y velocidad angular en el eje de Yaw ante perturbaciones:

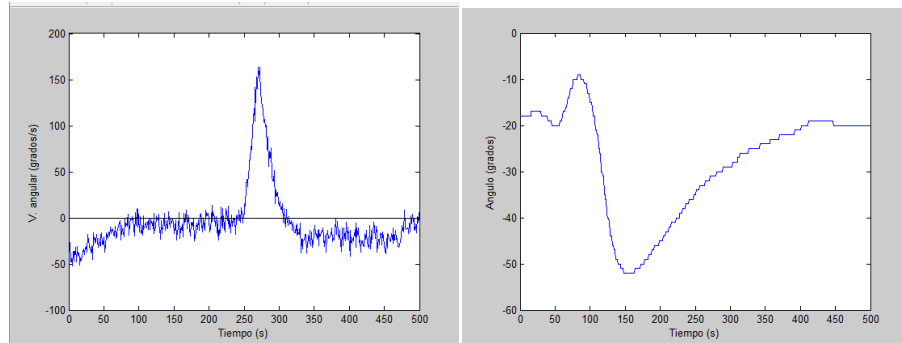


Ilustración 4. 21 Rendimiento del sistema de control de velocidad angular de Yaw (Izquierda) y de posición angular de Yaw (Derecha) ante perturbaciones.

Fuente: elaboración propia.

IV.7 Plataforma de pruebas

IV.7.1 Software de telemetría y comandos

El software de telemetría y comandos se desarrolló en 3 fases, las cuales se describen a continuación:

1. **Primera fase:** se desarrolló un script en MATLAB para la lectura y escritura del puerto serial, graficación de datos recibidos y el envío de comandos de movimiento al cuadricóptero. Se encontraron numerosas dificultades para modularizar el software desarrollado, una alta abstracción que dificultaba la realización exhaustiva de validaciones, y problemas en la ejecución del recolector de basura del intérprete en tiempo de ejecución de MATLAB que llevaban a tener que reiniciar MATLAB continuamente para crear exitosamente un objeto manejador del puerto serial.
2. **Segunda fase:** se desarrollaron módulos en el lenguaje de programación Python para lectura y escritura del puerto serial, graficación de datos recibidos, y detección de comandos del control Logitech Rumblepad II, con las librerías pyserial, pyqtgraph y pygame. Los módulos desarrollados para la lectura del puerto serial y la detección de comandos del control Logitech Rumblepad II ofrecieron un rendimiento satisfactorio en pruebas de ejecución, pero no así el módulo de graficación desarrollado, requiriendo una implementación multi-hilo exhaustiva. Por otro lado, se encontraron problemas a la hora de realizar la comunicación entre los hilos de ejecución de los módulos desarrollados.
3. **Tercera fase (final):** en vista de los problemas de robustez, comunicación entre módulos y rendimiento del software de graficación de datos recibidos por telemetría, se decidió adaptar los módulos de detección de comandos del control Logitech Rumblepad II y de lectura del puerto serial desarrollados para funcionar sobre la plataforma de software para robótica Robot Operating System (ROS).

El software se rediseñó y desarrolló como una serie de paquetes para ROS, que llevan a cabo de forma independiente la transmisión de datos entre el cuadricóptero y la PC, la obtención de comandos del usuario mediante un control Logitech Gamepad II y la exportación de los mensajes de estado del cuadricóptero a formato CSV. La codificación de los mismos se realizó en el lenguaje de programación Python, haciendo uso de la

librería rospy, y siguiendo los estándares de programación de ROS.

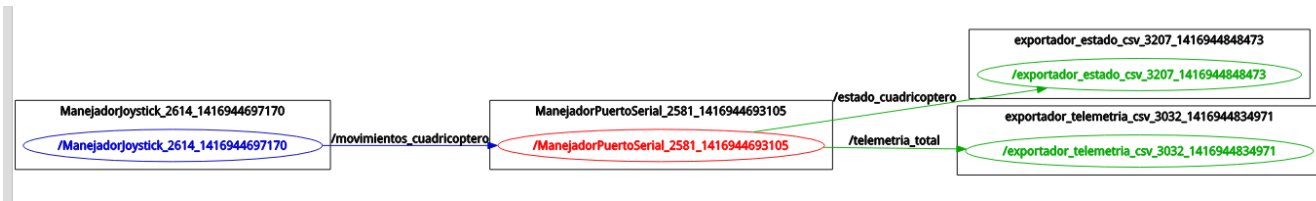


Ilustración 4. 22 Grafo de los nodos desarrollados en ejecución.

Fuente: elaboración propia.

Para realizar la comunicación entre los paquetes de software en tiempo de ejecución se definieron formatos de mensaje personalizados, los cuales se describen a continuación:

Al realizarse la comunicación entre los paquetes de forma asíncrona, como es natural en los paquetes de software desarrollados para ROS, la plataforma de software desarrollada tuvo un muy bajo acoplamiento entre sus componentes, lo cual aseguró una alta robustez del sistema ante fallas.

Tabla 4. 6. Formatos de mensaje utilizados por los nodos desarrollados

Nodo	Encendido	ComandoMovimiento	EstadoCuadricoptero	TelemetriaTotal
comunicacion_serial	X	X	X	X
logitech_rumblepad_ii		X		
exportador_estado_csv			X	
exportador_telemetria_csv				X

Fuente: elaboración propia.

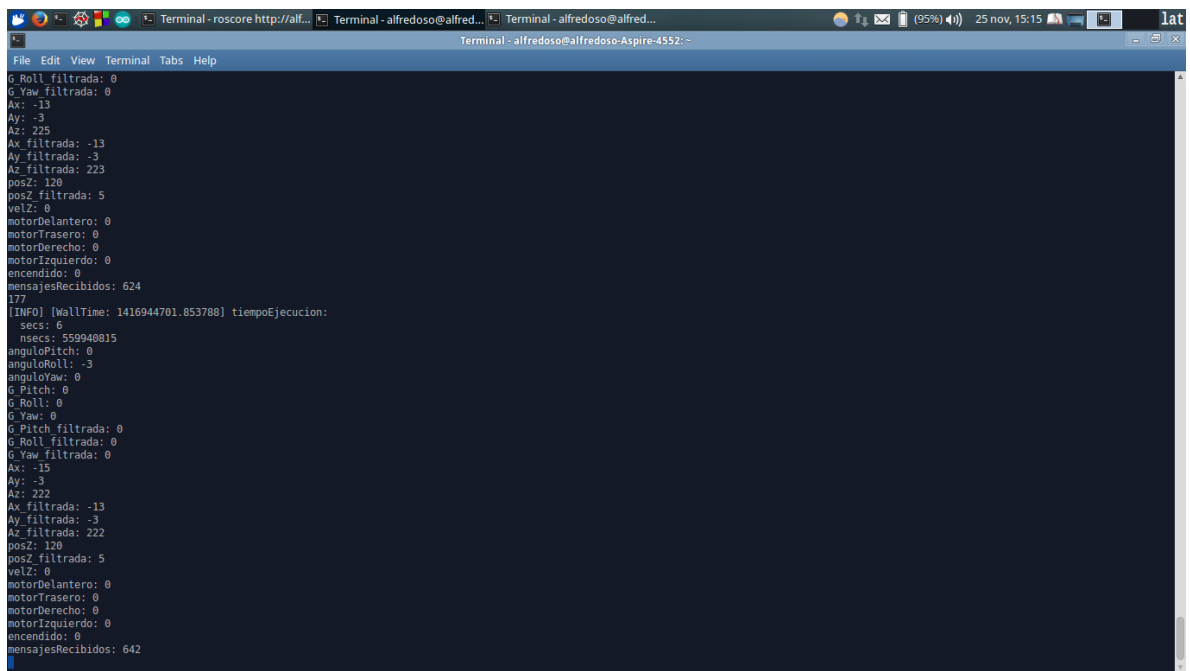
A continuación se describen los paquetes de software desarrollados:

- **comunicación_serial:** al iniciar su ejecución instancia un objeto de la clase handler_serial, el cual encapsula el manejo del puerto serial, y permite al programa principal realizar llamados simples a funciones para llevar a cabo el envío y recepción de mensajes de encendido o apagado, estado, y comandos de movimiento. Posteriormente,

son inicializadas rutinas de subscripción a los tópicos encendido_cuadricoptero y movimientos_cuadricoptero, y rutinas para publicación en el tópico estado_cuadricoptero.

Durante su ejecución, el nodo del paquete publica todos los mensajes de estado recibidos por el objeto handler_serial, y realiza el envío de los comandos de encendido o apagado, y de movimiento. Si se finaliza su ejecución de forma abrupta, o se desconecta el cable de comunicación del puerto serial, el nodo automáticamente detiene la comunicación y libera el puerto serial como recurso del sistema.

El manejo del puerto serial en la clase handler_serial se realiza mediante la librería pyserial, el objeto Serial, y sus métodos correspondientes.



```
File Edit View Terminal Tabs Help
Terminal - alfredoso@alfredoso-Aspire-4552:~$ rosrun comunicacion_serial comunicacion_serial.py
G_Roll_filtrada: 0
G_Yaw_filtrada: 0
Ax: -13
Ay: -3
Az: 225
Ax_filtrada: -13
Ay_filtrada: -3
Az_filtrada: 223
posZ: 120
posZ_filtrada: 5
velZ: 0
motorDelantero: 0
motorTrasero: 0
motorDerecho: 0
motorIzquierdo: 0
encendido: 0
mensajesRecibidos: 624
177
[INFO] [WallTime: 1416944701.853788] tiempoEjecucion:
  secs: 6
  nsecs: 559948015
anguloPitch: 0
anguloRoll: -3
anguloYaw: 0
G_Pitch: 0
G_Roll: 0
G_Yaw: 0
G_Pitch_filtrada: 0
G_Roll_filtrada: 0
G_Yaw_filtrada: 0
Ax: -15
Ay: -3
Az: 222
Ax_filtrada: -13
Ay_filtrada: -3
Az_filtrada: 222
posZ: 120
posZ_filtrada: 5
velZ: 0
motorDelantero: 0
motorTrasero: 0
motorDerecho: 0
motorIzquierdo: 0
encendido: 0
mensajesRecibidos: 642
```

Ilustración 4. 23 Nodo comunicación_serial en funcionamiento.

Fuente: elaboración propia.

- **logitech_rumblepad_ii:** al iniciar su ejecución instancia un objeto de la clase `handler_joystick`, el cual encapsula la detección y manejo de los eventos del control. Igualmente, el nodo inicializa rutinas para la publicación de los comandos del usuario en los tópicos `encendido_cuadricoptero` y `movimientos_cuadricoptero`.

Durante su ejecución el nodo del paquete verifica constantemente la emisión de comandos por parte del usuario, y en caso de haber sido emitido un comando desde el control, se calcula la consigna de movimiento mediante un polinomio de tercer grado, el nodo publica dicho comando al tópico que corresponda. Si se finaliza su ejecución de forma abrupta, o se desconecta el cable del control, el nodo automáticamente detiene la publicación de mensajes y libera el control como recurso del sistema.

```

comandoAltura: 116.290283203
[INFO] [WallTime: 1416944766.159764] Comando de movimiento!
[INFO] [WallTime: 1416944766.160347] comandoPitch: -0.0
comandoRoll: 0.0
comandoAltura: 110.104980469
[INFO] [WallTime: 1416944766.169569] Comando de movimiento!
[INFO] [WallTime: 1416944766.170082] comandoPitch: -0.0
comandoRoll: 0.0
comandoAltura: 107.629394531
[INFO] [WallTime: 1416944766.179561] Comando de movimiento!
[INFO] [WallTime: 1416944766.180250] comandoPitch: -0.0
comandoRoll: 0.0
comandoAltura: 105.157470703
[INFO] [WallTime: 1416944766.189484] Comando de movimiento!
[INFO] [WallTime: 1416944766.190155] comandoPitch: -0.0
comandoRoll: 0.0
comandoAltura: 105.157470703
[INFO] [WallTime: 1416944766.199710] Comando de movimiento!
[INFO] [WallTime: 1416944766.200465] comandoPitch: -0.0
comandoRoll: 0.0
comandoAltura: 102.681884766
[INFO] [WallTime: 1416944766.279458] Comando de movimiento!
[INFO] [WallTime: 1416944766.279842] comandoPitch: -0.0
comandoRoll: 0.0
comandoAltura: 110.104980469
[INFO] [WallTime: 1416944766.299674] Comando de movimiento!
[INFO] [WallTime: 1416944766.300558] comandoPitch: -0.0
comandoRoll: 0.0
comandoAltura: 112.589506486
[INFO] [WallTime: 1416944766.349546] Comando de movimiento!
[INFO] [WallTime: 1416944766.350261] comandoPitch: -0.0
comandoRoll: 0.0
comandoAltura: 116.290283203
[INFO] [WallTime: 1416944766.359675] Comando de movimiento!
[INFO] [WallTime: 1416944766.360719] comandoPitch: -0.0
comandoRoll: 0.0
comandoAltura: 116.290283203
[INFO] [WallTime: 1416944766.379739] Comando de movimiento!
[INFO] [WallTime: 1416944766.381070] comandoPitch: -0.0
comandoRoll: 0.0
comandoAltura: 118.765869141
[INFO] [WallTime: 1416944766.419779] Comando de encendido!
[INFO] [WallTime: 1416944766.420802] encendido: 1
[INFO] [WallTime: 1416944767.199703] Comando de encendido!
[INFO] [WallTime: 1416944767.200725] encendido: 0

```

Ilustración 4. 24 Nodo logitech_rumblepad_ii en funcionamiento.

Fuente: elaboración propia.

El manejo del puerto serial en la clase `handler_joystick` se realiza mediante la librería `pygame`, el objeto `Joystick`, y sus métodos correspondientes.



Ilustración 4. 25 Distribución de botones para ejecución de comandos en el control Logitech Rumblepad II.

Fuente: elaboración propia.

- **exportador_estado_csv:** el nodo consta de una rutina de subscripción al tópico estado_cuadricoptero, que escribe todos los mensajes publicados en el mismo en un archivo en formato CSV (siguiendo las pautas de codificación de archivos CSV para su lectura con MATLAB).
- **exportador_telemetria_csv:** el nodo consta de una rutina de subscripción al tópico telemetría_total, que escribe todos los mensajes publicados en el mismo en un archivo en formato CSV (siguiendo las pautas de codificación de archivos CSV para su lectura con MATLAB).

No se desarrolló un paquete para la visualización de datos obtenidos mediante telemetría por considerarse que los paquetes para visualización de gráficas en tiempo real para ROS existentes (rxplot y rqt_plot) brindan un rendimiento óptimo y una variedad de alternativas de configuración adecuadas para la visualización del estado del cuadricóptero en tiempo real.

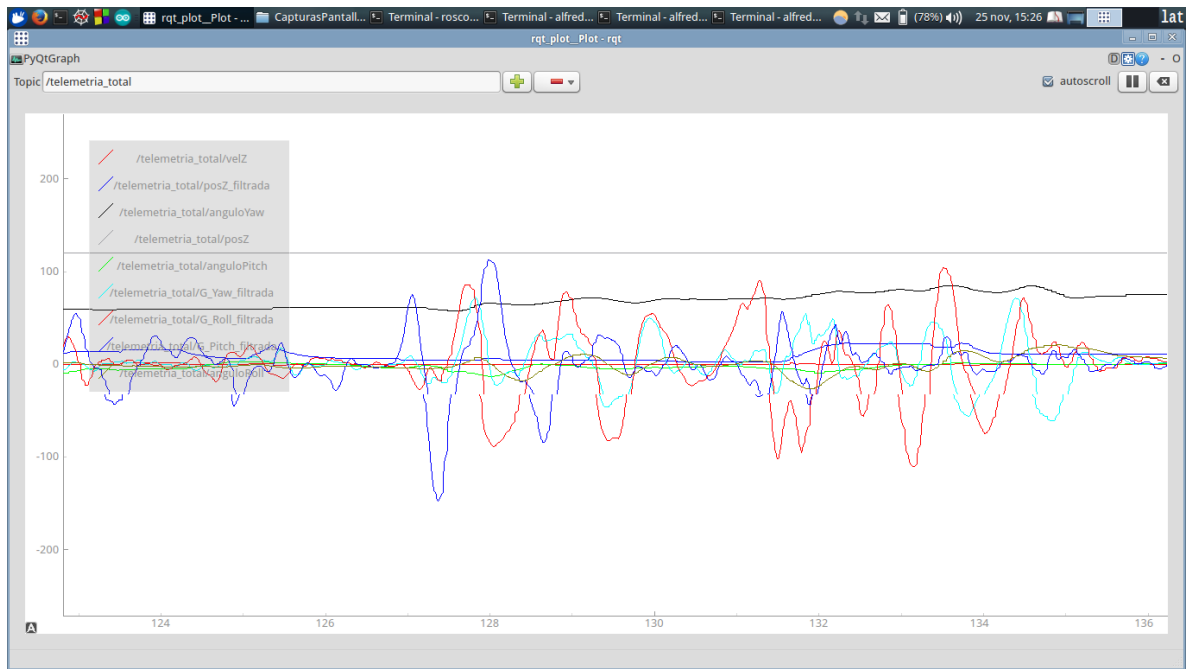


Ilustración 4. 26 Visualización de datos en rqt_plot.

Fuente: elaboración propia.

Finalmente, en el diagrama que se presenta a continuación se describe a detalle el flujo de ejecución del software desarrollado:

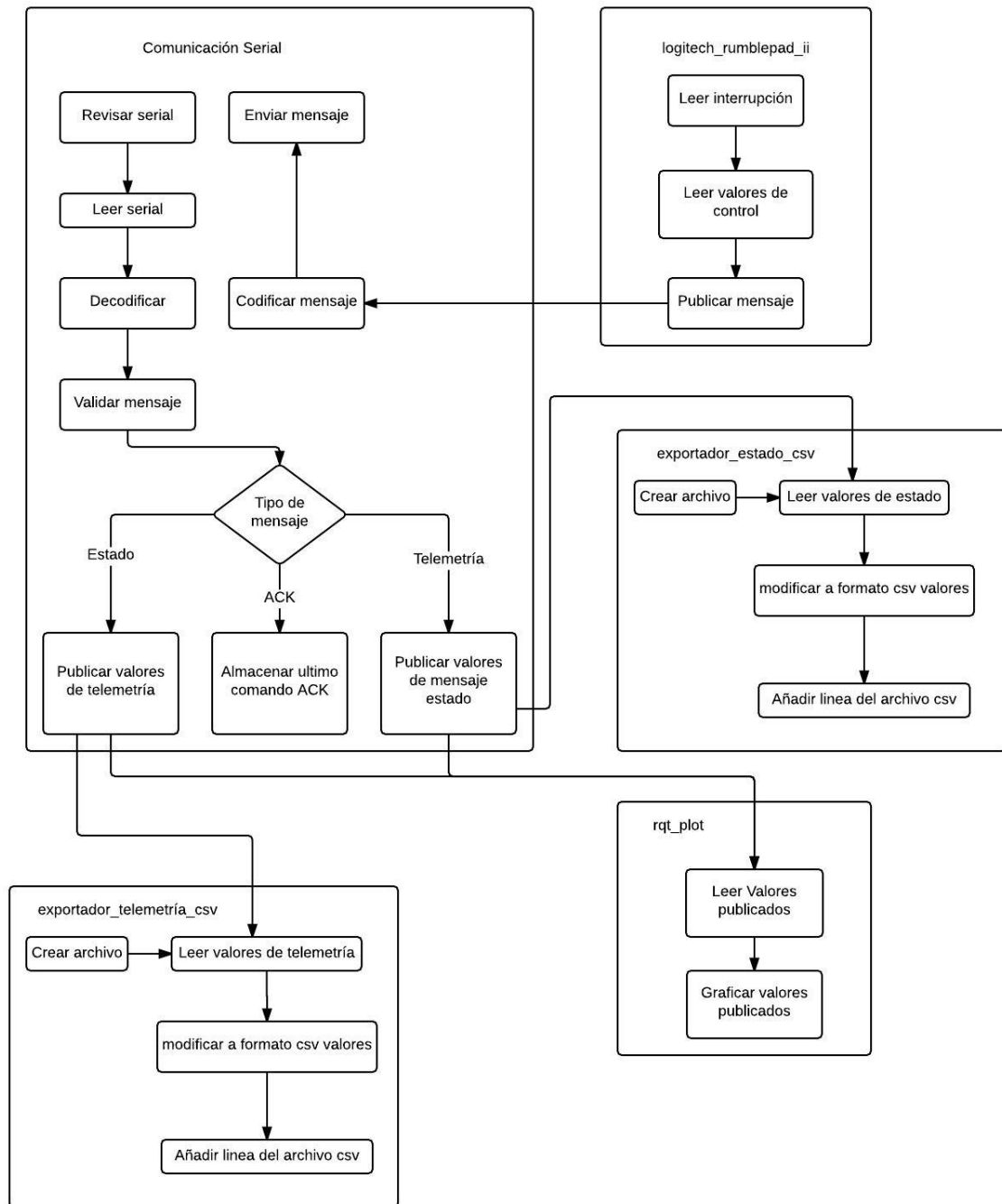


Ilustración 4. 27 Diagrama de funcionamiento del software de telemetría y comandos.

Fuente: elaboración propia

IV.7.2 Scripts para análisis en el dominio de la frecuencia de datos de los sensores

Se desarrollaron una serie de scripts de MATLAB para realizar análisis en el dominio de la frecuencia de los datos crudos y filtrados de los sensores, a partir de archivos CSV generados por los nodos exportador_estado_csv y exportador_telemetria_csv, desarrollados como parte de la plataforma de software de telemetría y comandos.

Los scripts desarrollados aplican un algoritmo de transformada rápida de Fourier sobre los datos de los sensores, realizan la traslación de los mismos en el dominio de la frecuencia, y muestran al usuario gráficas de los datos obtenidos en el dominio del tiempo y de la frecuencia, para su análisis.

A continuación se presenta una captura de pantalla de la ventana de visualización de ángulos, que forma parte del conjunto de ventanas que se muestran al ejecutar los scripts desarrollados, con un formato similar:

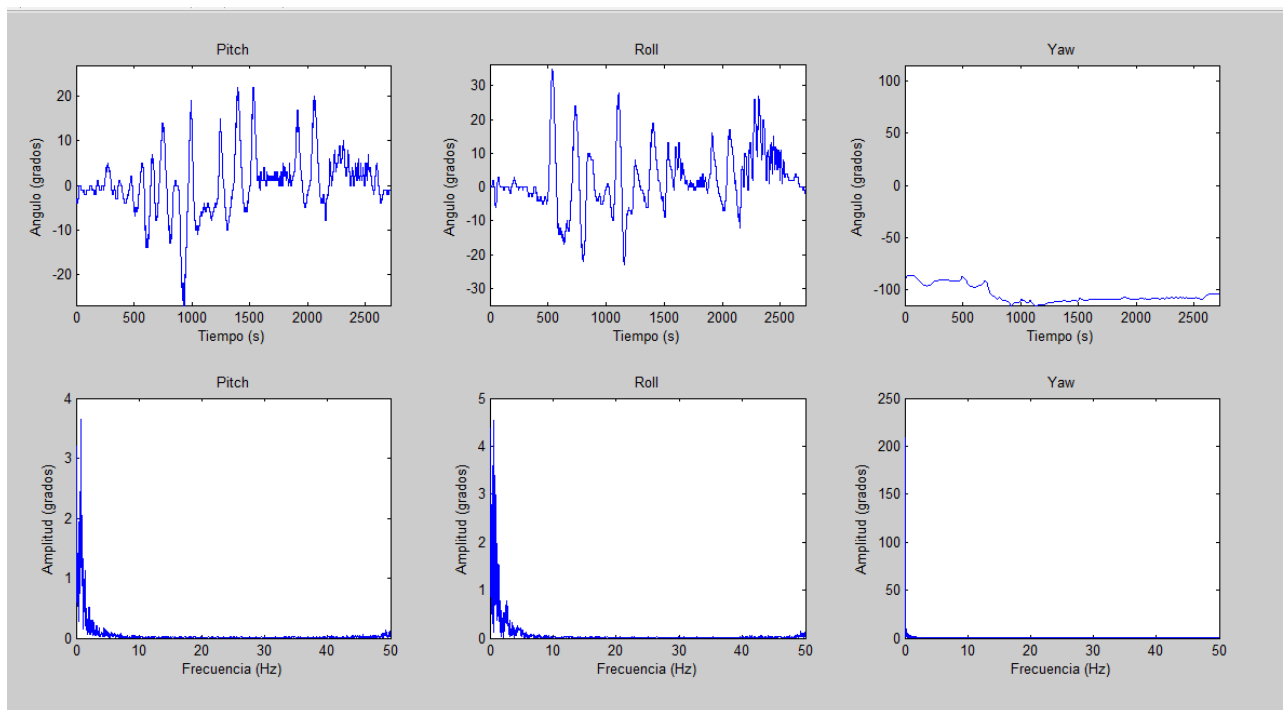


Ilustración 4. 28 Captura de pantalla de la ventana de visualización de datos de ángulos en MATLAB

Fuente: elaboración propia

IV.7.3 Montaje para la ejecución de pruebas en un solo eje del cuadricóptero

Una vez comprobado el montaje de todos los componentes sobre el cuadricóptero se procedió a trabajar con el algoritmo de estabilización de manera independiente para los ejes roll y pitch, esto se logro utilizando una base de madera la cual restringía el movimiento del cuadricóptero a solo uno de los ejes.



Ilustración 4. 29 Ejemplo de base utilizada para restringir el movimiento del cuadricóptero en un eje

Fuente: elaboración propia

IV.7.4 Montaje para la ejecución de pruebas en vuelo restringido

Una vez estabilizados los ejes de roll y pitch de forma independiente se empezó a calibrar el eje yaw, para esto se sujetó al cuadricóptero de una cuerda, la cual fue amarrada al centro de la estructura, y se colgó en un lugar suficientemente alto y amplio como para que pudiera moverse sin chocar contra otros objetos. Además se pudo probar el ajuste de roll y pitch trabajando de forma simultánea y la robustez del sistema de control frente a perturbaciones del ambiente.

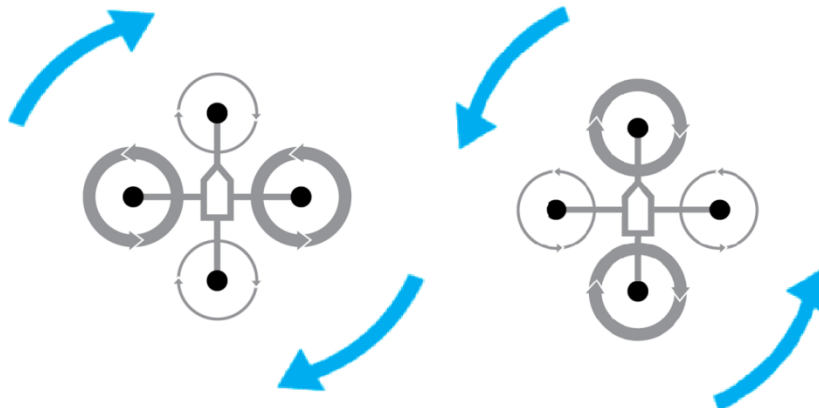


Ilustración 4. 30 Diagrama de movimiento sobre el eje yaw.

Fuente: <http://blog.oscarliang.net/wp-content/uploads/2013/06/quadcopter-rotating.png>

CAPÍTULO V - Resultados

- 1. Objetivo I - Diseñar e implementar un cuadricóptero con una unidad de control basada en Arduino.**
 - a. Electrónica:
 - i. Circuito de alimentación y control de motores de corriente continua.
 - ii. Circuito de lógica, sensores y comunicación.
 - b. Instrumentación:
 - i. Implementación para Arduino de algoritmo para estimación de posición angular del cuadricóptero a partir de mediciones de sensores acelerómetro y giroscopio de tres ejes.
 - ii. Implementación para Arduino de algoritmo para estimación de velocidad y posición en el eje Z (altura) del cuadricóptero a partir de mediciones de un sensor ultrasónico de distancia y un acelerómetro de tres (3) ejes.
- 2. Objetivo II - Diseñar e implementar una interfaz de comunicación inalámbrica entre el cuadricóptero y una computadora para tareas de encendido, apagado, movimientos simples en tres dimensiones y recopilación de información de los sensores del cuadricóptero:**
 - a. Interfaz de comunicación inalámbrica para el envío de comandos de encendido y movimientos simples en tres dimensiones..
- 3. Objetivo III - Diseñar e implementar una interfaz de comunicación para la obtención y análisis de datos por telemetría:**
 - a. Interfaz de comunicación inalámbrica para la obtención y análisis de datos por telemetría.
- 4. Objetivo IV - Diseñar e implementar un algoritmo Proporcional-Integral-Derivativo que permita la estabilización del cuadricóptero.**
 - a. Implementación en MATLAB del modelo físico del cuadricóptero Draganflyer V desarrollado en [Kivrak 2006].
 - b. Implementación y simulación en MATLAB de los algoritmos de control PID de posición angular, velocidad angular, y altura del cuadricóptero.

- c. Implementación en el lenguaje de programación Arduino de algoritmos de control PID de posición angular, velocidad angular y altura del cuadricóptero.

5. Objetivo V - Diseñar e implementar una plataforma de pruebas en tiempo real:

- a. Aplicación de escritorio, desarrollada para ROS, que permite el manejo del cuadricóptero mediante un control Logitech Rumblepad 2, comunicación mediante puerto serial con el módulo XBEE Explorer USB para comando remoto del cuadricóptero y recepción de mensajes de telemetría; y exportación de mensajes de telemetría a formato CSV.
- b. Scripts para la realización de análisis de respuesta en frecuencia de los sensores del cuadricóptero.

CAPÍTULO VI – Conclusiones y recomendaciones

VI.1 Conclusiones

1. La metodología de desarrollo de software de espiral es adecuada para la realización de proyectos de robótica como el desarrollado en el presente trabajo de investigación, ya que permite el perfeccionamiento de los componentes individuales del mismo a partir del desarrollo de una serie de prototipos incrementales desarrollados. Esto promueve un proceso de desarrollo, análisis de resultados e introducción de mejoras constante, asegura una mayor calidad de los componentes finales y simplifica el proceso de integración de la plataforma final.
2. La tarjeta Arduino Nano 3.0 utilizada en el presente trabajo, y basada en el microcontrolador ATMEGA328, ha presentado un rendimiento satisfactorio, una alta confiabilidad, y ha permitido reducir el tiempo de implementación y pruebas del proyecto en gran medida, pero, las características técnicas de la misma, extremadamente simples, han obstaculizado el desarrollo de los sistemas de control de altura del cuadricóptero, y la implementación de algoritmos de estimación y comunicación de mayor complejidad.
3. En el desarrollo de circuitos de potencia es recomendable realizar un análisis incremental, partiendo de la respuesta de la mínima combinación de componentes bajo las condiciones de trabajo esperadas, y aumentando la complejidad conforme se realizan pruebas y se recogen datos de funcionamiento. Esto permite identificar las características de funcionamiento particulares de cada componente, y, de los circuitos diseñados, y es vital para alcanzar las condiciones óptimas de funcionamiento de los mismos.
4. Las vibraciones mecánicas producidas por la rotación de las aspas y el accionamiento de los motores a alta velocidad pueden llegar a perturbar de forma significativa, e incluso, inutilizar, las estimaciones de estado realizadas mediante giroscopios y acelerómetros ubicados en el cuadricóptero. La construcción de una estructura física

que limite la transmisión de vibraciones, y en menor medida, la colocación física de los sensores en el chasis e implementación de algoritmos de filtrado de las señales de los mismos, son aspectos críticos para el éxito de cualquier proyecto que involucre estimación de estado de un cuadricóptero mediante Unidades de Medición Inercial.

5. Las hélices flexibles del chasis Draganflyer V, a pesar de poseer una alta durabilidad y representar un menor peligro para el usuario final, reducen en gran medida la eficiencia energética del cuadricóptero y la carga máxima que este puede levantar.
6. Se comprueba, en simulación, la hipótesis presentada en [Nadales 2009] acerca de la posibilidad de estabilizar un cuadricóptero en vuelo mediante la aplicación de un algoritmo Proporcional-Integral-Derivativo. Más aún, se verifica que a partir de una estimación de estado basada en mediciones de giroscopios, acelerómetros y sensores de distancia, y mediante la aplicación de una arquitectura de control basada en algoritmos PID, es posible realizar un control satisfactorio de la orientación angular y altura de un cuadricóptero en vuelo.
7. La hipótesis presentada en [Nadales 2009] se verifica parcialmente con base en los resultados de pruebas reales de vuelo con las arquitecturas de control del cuadricóptero desarrolladas. Por la magnitud de las vibraciones físicas del cuadricóptero, las cuales introducen incertidumbre a la estimación de estado del mismo, no se puede asegurar seguimiento preciso de consignas con las arquitecturas de control PID desarrolladas.
8. La velocidad de transmisión, la verificación de errores, y la coordinación de la comunicación entre la unidad de control del cuadricóptero y cualquier otra unidad o estación de control, son variables críticas para una correcta ejecución de los sistemas de control, el correcto envío y ejecución de comandos, y una retroalimentación efectiva del estado del cuadricóptero mediante telemetría.

VI.2 Recomendaciones

1. Para implementar exitosamente los sistemas de control de posición angular y altura en el cuadricóptero partiendo de la estimación de estado obtenida a partir de sensores inerciales, se recomienda utilizar un chasis para cuadricópteros que limite la

generación y transmisión de vibraciones mecánicas.

2. Se recomienda partir de la interfaz de comunicación serial desarrollada para realizar la retroalimentación entre el Módulo de Lógica, Sensores y Comunicación desarrollado, y cualquier otra unidad que ejecute un algoritmo de control de mayor nivel, como pueden ser, algoritmos de control de posición o trayectoria.
3. Para solucionar el problema de contención de recursos del segundo temporizador del Arduino Nano producido por la librería NewPing y la asignación de los puertos de PWM del circuito de lógica, sensores y comunicación, se recomienda modificar el circuito desarrollado para utilizar únicamente los puertos 5, 6, 9 y 10 del Arduino Nano para emitir señales de PWM, de modo que sólo sean utilizados los temporizadores timer0 y timer1 del mismo, y el temporizador timer2 quede libre para su utilización por parte de la librería NewPing.
4. En caso de perseguirse el desarrollo de sistemas de control para realizar vuelo acrobático o maniobras agresivas, se recomienda mejorar el algoritmo de estimación de ángulos mediante un filtro complementario validando la dirección del vector de aceleración gravitacional al realizar la estimación de ángulo a partir de los datos del acelerómetro, o utilizando cuaterniones en lugar de ángulos de Euler.
5. Para una estimación más robusta del ángulo de Yaw se recomienda la utilización de un magnetómetro en conjunción con un giroscopio. Se sugiere consultar la investigación presentada en [Madgwick 2010].
6. Para disminuir el tiempo de respuesta e incrementar la robustez y precisión de los sistemas de control se recomienda implementar control de velocidad individual de los motores del cuadricóptero.
7. Para la realización de pruebas de algoritmos de estimación de estado, control y comunicación se recomienda limitar el movimiento del cuadricóptero en uno, dos, o tres ejes, ya que ello permite realizar un estudio exhaustivo del rendimiento de los mismos bajo condiciones controladas, e identificar posibles problemas de funcionamiento e integración con mayor rapidez y precisión.
8. La estrategia desarrollada para el cálculo de velocidad lineal en el eje Z del cuadricóptero está constantemente sujeta a errores de estimación por obtenerse a

partir de integración numérica, y por la alta sensibilidad del acelerómetro a las vibraciones mecánicas. Se recomienda mejorar la precisión de la misma complementando la estimación mediante el acelerómetro con la estimación a partir de otro sensor infrarrojo de distancia o de odometría visual, y realizando la fusión de ambas estimaciones con un filtro de Kalman extendido.

9. Para la realización de investigación avanzada en control de cuadricópteros se recomienda fervientemente utilizar tecnología de punta, para luego extrapolar los resultados a plataformas de bajo costo, con el fin de simplificar el proceso de desarrollo. Tecnologías como los motores sin escobillas, Electronic Speed Controllers, y unidades de control como el Ardupilot Mega, Pixhawk y Asctec Mastermind, al estar basadas en estándares del área permiten mantener condiciones homogéneas en la arquitectura de los cuadricópteros sobre los cuales se realiza la investigación.

REFERENCIAS BIBLIOGRÁFICAS

[Alciatore 2008] Alciatore, D.; Histan, M. (2008). *Introducción a la Mecatrónica y los sistemas de medición*. Tercera edición. Interamericana editores.

[Álvarez 2012] Alvarez, E (2012) Desarrollo de una plataforma de control para el robot móvil roomba utilizando ROS e integración de un sensor de kinect.

[Banzi 2011] Banzi, M. y Cuartielles, D., *Descripción de la plataforma Arduino*. Obtenido de <http://www.arduino.cc/>

[Bonastre 2010] Bonastre, A. (2010) *Desarrollo de un Sistema Integrado de Navegación Inercial: Interficie IMU + FPGA*. Universitat Autònoma de Barcelona. Barcelona, España.

[Burgard 2005] Burgard, W. (2005). *Recursive Bayes Filtering: Advanced AI*. Obtenido de <http://gki.informatik.uni-freiburg.de/teaching/ws0405/advanced/BayesFiltering.pdf>

[Burkamshaw 2010] Burkamshaw, L. (2010). *Towards a Low Cost Quadrotor Research Platform*. Naval Postgraduate School. California, Estados Unidos de América.

[Chin Kar 2007] Chin Kar Wei. (2007). *Flight Dynamics and Control for an Indoor UAV*. Trabajo Especial de Grado de Ingeniería Mecánica. Universidad Nacional de Singapur.

[Colton 2007] Colton, S. (2007). *The Balance Filter: A Simple Solution for Integrating Accelerometer and Gyroscope Measurements for a Balancing Platform*. Massachusetts Institute of Technology. Massachusetts. Estados Unidos de América.

[Colton 2011 - 1] Colton, S. (2011). *The great XBee 57.6kpbs mystery finally solved*. Obtenido de <http://scolton.blogspot.com/2011/09/great-xbee-576kpbs-mystery-finally.html>

[Colton 2011 – 2] Colton, S. (2011). *PCB Quadrotor (Brushless)*. Obtenido de <http://www.instructables.com/id/PCB-Quadrotor-Brushless/?lang=es>

[Dignyu 2007] Dignyu, X., YangQuan, C y Atherton, D. (2007). *Linear feedback control*. Siam.

[Draganfly Innovations 2006] (2006). *Draganflyer V Ti User Manual*. Draganfly Innovations. Estados Unidos de América.

[Dulhoste 2011] Dulhoste, J. *Teoría de control*. Universidad de los Andes. Mérida, Venezuela.

[Gaydou 2007] Gaydou, D. (2007). *Filtro complementario para estimación de actitud aplicado al controlador embebido de un cuatrirrotor*. Universidad Tecnológica Nacional. Córdoba. Argentina.

[Hibbeler 2012] Russell, Hibbeler (2012). *Mecanica vectorial para ingenieros*. McGrawHill. Mexico.

[Kivrak 2006] Kivrak, A. (2006). *Design of control systems for a quadrotor flight vehicle equipped with inertial sensors*. Atihm University. Turquía.

[Madgwick 2010] Madgwick, S. (2010). *An efficient orientation filter for inertial and inertial/magnetic sensor arrays*. University of Bristol. Reino Unido.

[Nadales 2009] Nadales, C. (2009). *Control de un quadrotor mediante la plataforma Arduino*. Trabajo Especial de Grado de Ingeniería Técnica de Telecomunicaciones, Universidad Politécnica de Catalunya, Barcelona, España.

[Ogata 2011] Ogata, K. (2011). *Ingeniería de control moderna*. Quinta edición. Pearson.

[Pressman 2001] Pressman, R. (2001). *Software Engineering: A practitioner's approach*. Quinta edición. Mc-Graw Hill.

[Ramírez 2003] Ramírez A. (2003). *El filtro del kalman*. Banco central de costa rica, división económica, departamento de investigaciones económicas, nota técnica.

[Rodríguez 2013] Rodríguez, M. (2013). *Control a lazo abierto*. Obtenido de: http://prof.usb.ve/mirodriguez/control/Sistemas_y_transformada_de_laplace/control_a_lazo_abierto.html

[Rouse 2007] Rouse, M. (2007). *Spiral model (spiral lifecycle model)*. Obtenido de <http://searchsoftwarequality.techtarget.com/definition/spiral-model>

[Shakev 2011] Shakev, N.; Topalov, A.; Kaynak, O.; y Borisov, K. (2011). *Comparative Results on Stabilization of the Quadrotor Rotorcraft Using Bounded Feedback Controllers*. Trabajo presentado en el Journal of Intelligent and Robotics Systems 2011.

[Sinha 2012] Himanshu Sinha (2012). *An Autonomous Quadrotor Flying Robot..* Pune University. India

[STMicroElectronics 2010] STMicroElectronics. (2010). *Tilt Measurement using a low-g 3-axis accelerometer*.

[Sturm 2013] Sturm, J. (2013). *Probabilistic Models and State Estimation*. Obtenido de: http://vision.in.tum.de/media/teaching/ss2013/visnav2013/lecture3_state.pdf

[UniLeon 2013] *El controlador PID básico*. Material de la cátedra de Laboratorio Remoto de Automática. Universidad de León. León, España.

[Vidyasagar 2010] Vidyasagar, M. (2010). *A tutorial overview of Control Theory for Non-Engineers*. The University of Texas at Dallas. Texas, Estados Unidos de América.

[Zabczyk 1993] Zabczyk, J. (1993). *Mathematical control theory: An introduction*. Birkhäuser. Boston, Massachusetts, Estados Unidos de América.