MODELADO Y CONTROL DE UN CUADRICÓPTERO

Ernesto Paiva-Peredo

Piura, mayo de 2016

FACULTAD DE INGENIERÍA

Máster en Ingeniería Mecánico-Eléctrica con Mención en Automática y Optimización

Paiva, E. (2016). *Modelado y control de un cuadricóptero* (Tesis de Máster en Ingeniería Mecánico-Eléctrica con Mención en Automática y Optimización). Universidad de Piura. Facultad de Ingeniería. Piura, Perú.

U N I V E R S I D A D D E P I U R A

FACULTAD DE INGENIERÍA

Modelado y Control de un Cuadricóptero

Tesis para optar el Grado de Máster en
Ingeniería Mecánica Eléctrica con mención en Automática y Optimización

Ing. Ernesto Alonso Paiva Peredo

Piura, mayo de 2016

A mi madre, Rosana del Carmen Peredo Vílchez y a mi tío, Eduardo Alonso Vilela Peredo, por sus consejos, valores, motivación y amor.

A mi familia por su cariño y preocupación por mi futuro.

A mis amigos, en especial a Catherin Girón por su apoyo constante e incondicional.

Prólogo

En la Universidad de Piura se vienen realizando investigaciones utilizando algoritmos de procesamiento de imágenes aplicadas a los cultivo. El propósito de estas investigaciones es desarrollar un sistema de supervisión y monitoreo de cultivos, ayudados también con los UAV's (*Unmanned Aerial Vehicles*) para hacer una mejor gestión de los cultivos o de la agroindustria. Por ejemplo la evaluación de los índices de vegetación que nos dan información importante acerca de los cultivos, permitiendo por ejemplo estimar el estrés hídrico en los cultivos. Esta información servirá a los agricultores para mejorar la gestión de sus cultivos, un manejo eficiente del agua, aumentando la productividad y competitividad en el sector agroindustrial. Este enfoque se le conoce también como agricultura de precisión. Dentro los UAV's hay un tipo que está constituido por cuatro rotores, por ello se le denomina cuadricóptero o Quad-Rotor, en el cual se suelen alojar los sensores o sistemas para recopilar información. En el laboratorio de sistemas automáticos de control se está intentado unir la parte de control y supervisión de cuadricópteros con el procesamiento de información de sensores que sobre él se pueden montar. Esto da lugar a desarrollo de ingenierías de software en sistemas embebidos orientadas a mejorar la eficiencia de los cultivos. En una zona cuya agroindustria es relevante, este tipo de proyectos de desarrollo tecnológico es adecuado para posibles aplicaciones futuros.

La motivación principal de esta tesis es realizar un estudio de la teoría del modelo matemático de los cuadricópteros, las estrategias de control utilizadas en los sistemas comerciales e implementar algoritmos de control PID para el control de ángulos. Dicho estudio formará las bases de futuros proyectos enfocados en la recopilación de datos en cultivos como humedad, temperatura, zonas infectadas por el estrés hídrico, permitiendo mejorar la gestión del agro y aumento de la producción agrícola en la región Piura. El trabajo realizado ha permitido:

Un modelo no lineal de buenas prestaciones

El diseño de un controlador de vuelo robusto, con experimentos en tiempo real.

Implementar un control de orientación, el cual es el primero que se desarrolla en nuestro país. En el sentido que se suele adquirir este software.

Se ha publicado un *paper* en un evento internacional (CHILECON 2015) con base de datos en IEEEXplorer con ISBN 978-1-4673-8755-2 el cual es titulado *Modeling and PID cascade control of a Quadcopter for trajectory tracking*.

Se dejan abiertos desarrollos futuros como: implementar algoritmos de visión artificial para localización y control de UAV, control de posición de un UAV, diseño de nuevos controladores para un UAV en sistemas embebidos más potentes e implementación de un UAV con un escáner LIDAR, el control de posición de cuadricópteros *indoor* por medio de cámaras situadas en un ambiente cerrado.

Durante el desarrollo de la tesis también se tuvo una pasantía en la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla, España. Se le agradece al CONCYTEC por la beca de estudio en ésta Maestría en Ingeniería Mecánica Eléctrica en la prestigiosa Universidad de Piura.

Resumen

El presente proyecto de tesis está dirigido a los vehículos aéreos no tripulados, los cuales han tenido mucha popularidad en los últimos años gracias a la miniaturización de los componentes electrónicos y mecánicos. El presente trabajo consiste la realización del modelo matemático, la estimación de parámetros físicos, el diseño de controles PID para nuestro cuadricóptero, la validación del modelo matemático y la implementación experimental del algoritmo de control en un sistema embebido para estabilizar la orientación de la aeronave. La tesis contempla realizar el modelo matemático de la aerodinámica del cuadricóptero agregando la dinámica de los propulsores. Se linealiza el modelo en base a documentos publicados en revistas importantes y se estiman las constantes y coeficientes utilizados en el modelo matemático por medio de pruebas en laboratorio y con ayuda de "*software*" especializados. En base a publicaciones y a sistemas embebido comerciales especiales para "*drones*" se realiza el diseño del control del cuadricóptero en el entorno Simulink de Matlab. Para poder comprobar el correcto funcionamiento se desarrolla un entorno virtual en tres dimensiones también en Simulink. También se plantea un algoritmo de control de orientación que permita mejorar la maniobrabilidad y robustez frente a perturbaciones. Finalmente se implementa el algoritmo de control de orientación en un sistema embebido comercial para desarrolladores y se llevan a cabo pruebas de vuelo.

Además como parte del programa de maestría, se ha realizado un viaje de investigación científica en la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla de una duración aproximada de siete semanas. El propósito académico era investigar nuevas tecnologías y líneas de investigación futuras enfocadas a la problemática de transporte de carga por medio de cuadricópteros.

Índice General

Introducción

La más reciente innovación en el rubro de los vehículos aéreos convencionales y los satélites en la recopilación de datos de forma remota son los llamados vehículos aéreos no tripulados (VANT). Los primeros VANT se emplearon como aviones teledirigidos durante la Segunda Guerra Mundial y para entrenamiento de los operarios de los cañones antiaéreos; según el Instituto Nacional de Estadística y Geografía de México [1] fue a finales del siglo XX cuando se utilizó el término VANT en las aeronaves que se operaban mediante radio control, con todas las características de autonomía.

El Dr. Samir Bouabdallah [2] explica que los importantes progresos en los últimos años en tecnologías de detección, almacenamiento de energía de alta densidad, sensores miniatura y procesamiento de datos, han hecho posibles el desarrollo de vehículos aéreos no tripulados. Esta nueva situación ha abierto el camino a varias aplicaciones militares y civiles complejas y muy importantes.

A pesar que los VANT tuvieron como objetivo inicial de su creación el entrenamiento bélico, luego de conocer las increíbles posibilidades que podían dar, fueron utilizados en los conflictos bélicos del Golfo (1990-91) y Bosnia (1992-95) [3].

Su éxito radica en que estos vehículos se controlan a partir de mandos a distancia, evitando así que se arriesgue la vida de los tripulantes, además al ser tan pequeños dan la posibilidad de ingresar a zonas de alto riesgo y difícil acceso.

Los campos en los cuales se puede aplicar esta tecnología son indeterminados. Uno de estos campos es la agricultura, donde son utilizados para determinar el estado de crecimiento, estrés hídrico, exceso o defecto de componentes químicos y/o temperatura de los cultivos, así mismo son empleados para controlar incendios forestales. En el ámbito medioambiental son utilizados para en hidrología, monitoreo de densidad forestal y vigilancia de especies en peligro de extinción.

Actualmente se conocen varias aplicaciones de los VANT en el mundo, una de ellas es la que está utilizando una gran compañía de comercio electrónico estadounidense, la cual se encuentra planeando un sistema de entregas cuasi inmediatas por medio de los VANT, entregando a sus clientes los productos en sus hogares. Por otro lado una compañía francesa de automóviles ha incorporado una VANT a uno de sus modelos de auto para avisar de posibles riesgos en el recorrido, determinar la dimensión del atasco en el tráfico o ayudar a determinar algunas salidas para evitarlo. Incluso se ha lanzado una aplicación para llevar a pasear a un canino, donde sólo es necesario configurar el recorrido por donde el VANT debe llevar a la mascota. Y en el Perú se están realizando investigaciones para controlar el tráfico limeño [4] y monitoreo de calidad del aire.

En el año 2004, investigaciones realizadas en la Escuela Politécnica Federal de Zúrich dieron origen a publicaciones como *Design and Control of an Indoor Micro Quadrotor* [5] en la cual se presenta el proyecto llamado *Omnidirectional Stationary Flying Outstretched Robot* (OS4), realizando una comparación entre diferentes tipos de VANT, muestra el modelo dinámico de su cuadricóptero y de los rotores. En este mismo trabajo se implementa un control de Lyapunov mostrando simulaciones y datos experimentales.

En el mismo año Bouabdallah y su equipo presentaron un artículo [2] donde se realizan una serie de pruebas en simulación y experimentales de su OS4 implementando el algorítmo de control clásico PID y control avanzado LQ. En el año 2005 se publica [6] donde se presentan mejoras en el *hardware* del OS4 y resultados obtenidos utilizando control de Lyapunov, además de esto se presenta la simulación en tres dimensiones de un cuadricóptero en el software *Webots*.

En el mismo año presentan los resultados de dos técnicas de control no lineales (*Backstepping* y *Sliding-mode*) aplicadas a un cuadricóptero [7], además se presenta un esquema de control en cascada para el control de posición. Años después, se publica un artículo [8] introduciendo un modelo de simulación que tiene en cuenta la variación de los coeficientes aerodinámicos debido al movimiento del vehículo.

El equipo liderado por el Dr. Samir Bouabdallah es uno de los pioneros en temas relacionados al control de cuadricópteros. La gran mayorías de sus artículos científicos se basan de su tesis doctoral [9]. Los trabajos del Dr. Bouabdallah son una de las principales referencias para el presente trabajo.

Por otra parte, en el 2008 Guenard, Hamel y Mahony [10] implementan un control visual basado en imágenes para un VANT con capacidad de vuelo estacionario o casi estacionario con una cámara montada a bordo del vehículo. El objetivo consistía en obtener posiciones fijas de vuelo utilizando un conjunto finito de puntos disjuntos en un plano para estimar la posición del VANT y cerrar el lazo de control. El control de la dinámica de posición y orientación están desacoplados.

En el mismo año, se publica una investigación [11] en la cual dos tareas diferentes se consideran, el primero se refiere a la estabilidad de vuelo y el segundo a preocupaciones de aterrizaje automático utilizando el flujo óptico como información de retroalimentación. Posteriormente se investiga una serie de algoritmos de control servo visual basado en imágenes para la regulación de la posición de un VANT [12]. Otro trabajo similar pero considerando una plataforma en movimiento es [13].

Los trabajos enfocados en control del VANT utilizando cámaras buscan estimar principalmente la posición del vehículo. El control de posición en ambientes cerrados y/o reducidos no es posible utilizar sensores GPS (*Global Positioning System*). Los documentos [10], [11], [12] y [13] representan algunas de las investigaciones más significativas en control de posición *indoor*.

Otro grupo de investigación enfocado al control visual de cuadricóptero es el encabezado por Altuğ y Ostrowski, entre sus publicaciones se tiene [14] donde muestra resultados de controladores implementados en un procesador ubicado en tierra y utilizando una cámara para estimar la posición y la orientación del cuadricóptero. Realiza un trabajo similar en [15] utilizando dos cámaras para estimar las variables a controlar. En ambas publicaciones [14] y [15] se logran observar las bondades del control de posición por medio de cámaras, siendo inspiración de futuros trabajos de investigación.

En [16] se desarrolla un algoritmo para generar trayectorias optimizadas considerando puntos de paso asignados por el usuario. Otro algoritmo generador de trayectorias es desarrollado en [17], el cual genera trayectorias optimizadas para movimientos agresivos considerando puntos de paso con ángulos de inclinación, dichos ángulos son designados por obstáculos en forma de ventanas dentro de las cuales debe ingresar el vehículo utilizando previamente técnicas de visión artificial.

Estos trabajos engloban tareas de control y procesamiento de imágenes obteniendo resultados sorprendentes, sin embargo hasta la actualidad, las unidades de procesamiento se encuentra en fuera del VANT debido a la alta carga computacional y por lo tanto no se puede aún extender estas teorías a aplicaciones *outdoor*.

Otro trabajo importante es el publicado por Rubio [18], donde se sintoniza un controlador PID utilizando las normas de controladores *H-infinitive*. Este trabajo se desarrolla en un entorno de simulación y muestra robustez del VANT frente a disturbios externos.

En la presente tesis se implementa un algoritmo de control PID modificado resistente a perturbación del viento en un cuadricóptero, comparando la robustez en simulación con los resultados de [18]. En muchos documentos como [10], [11], [12], [13], [14], [15], [16], [17] y [18] no consideran la dinámica de los motores, sin embargo en los trabajos del Dr. Samir Bouabdallah sí considera la constante de tiempo del proceso del motor. En el presente trabajo de tesis se estudia la metodología descrita en las publicaciones de Boaudballah debido a que la naturaleza de los motores del cuadricóptero en estudio tiene constante de tiempo no despreciables.

Capítulo 1
Modelado Matemático del Cuadricóptero

1.1 Configuración del Cuadricóptero

Un Cuadricóptero se puede describir como un VANT con cuatro hélices en configuración transversal. La Figura 1 muestra la distribución de los rotores de un cuadricóptero configurado en cruz el cual se encuentra en ascenso, donde todas las hélices tienen la misma velocidad. [5].

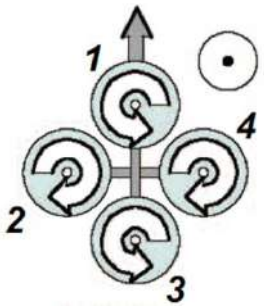

Figura 1 Configuración en cruz de un Cuadricóptero [5]

Los dos pares de hélices (1,3) y (2,4) giran en direcciones opuestas. Las hélices de la parte delantera (1) y trasera (3) giran en sentido anti-horario, mientras que las hélices de la izquierda (2) y derecha (4) giran en sentido horario. Esta configuración de direcciones de giro opuestos permite equilibrar los pares de arrastre, eliminando la necesidad de un rotor de cola (este rotor extra es necesario en la estructura del helicóptero estándar). [19]

Mediante la variación de la velocidad del rotor, se puede cambiar la fuerza de sustentación y crear movimiento como se describe en la Figura 2. [5]

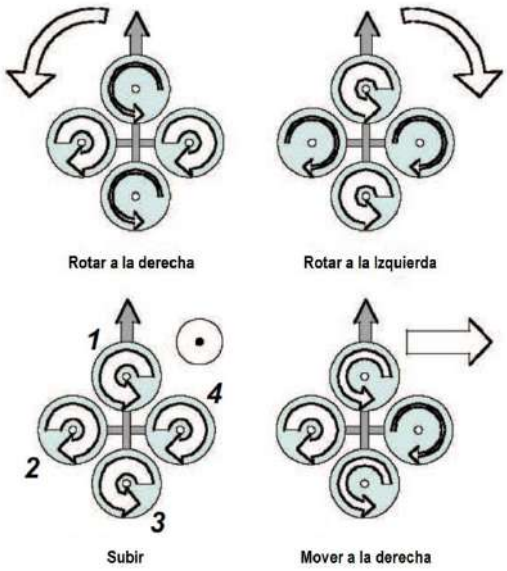

Figura 2 Cambio de velocidad en hélices para generar movimiento [5]

Esta configuración permite cuatro movimientos básicos, los cuales habilitan al cuadricóptero para alcanzar una posición exacta. A continuación se describen estos movimientos básicos según se explica en [19]

Empuje o *throttle*

Este movimiento es proporcionado por el aumento (o disminución) de velocidad de todas las hélices en la misma cantidad generando una fuerza vertical que sube, baja o mantiene suspendido el cuadricóptero respecto a su propio sistema inercial. Sin embargo, si el cuadricóptero no se encuentra en posición horizontal respecto al sistema de referencia inercial (fijado en la tierra), el empuje proporcionado por lo rotores generará aceleraciones verticales y horizontales. Desde ahora en adelante en este texto, la fuerza de empuje o *throttle* será representado por la variable U_1.

Alabeo o *roll*

Este movimiento es proporcionado por el aumento (o disminución) de velocidad de la hélice izquierda y por disminución (o aumento) de la velocidad en la hélice derecha. Esto produce un par de torsión con respecto al eje x_B que hace girar el cuadricóptero como se muestra en la Figura 3. Este desequilibro de fuerzas sólo conduce a una aceleración en el ángulo de alabeo ϕ (en primera aproximación). En adelante en este texto, el desequilibrio de fuerzas en el eje x_B será representado por la variable U_2.

Cabeceo o *pitch*

Este movimiento es muy similar al *roll* y se genera mediante el aumento (o disminución) la velocidad de la hélice posterior y por la disminución (o aumento) de la delantera. El par de torsión con respecto al eje y_B hace girar el cuadricóptero como se muestra en la Figura 3. Este desequilibrio de fuerzas sólo conduce a una aceleración en el ángulo de cabeceo θ (en primera aproximación). En adelante en este texto, el desequilibrio de fuerzas en el eje y_B será representado por la variable U_3.

Guiñada o *yaw*

Este movimiento es proporcionado por el aumento (o disminución) velocidad de la hélice delantera y trasera y por la disminución (o aumento) de la hélice de la izquierda y derecha. Esto conduce a un par de torsión con respecto al eje z_B que hace girar el cuadricóptero como se muestra en la Figura 3. El movimiento de guiñada se genera gracias a que las hélices de izquierda y derecha giran en sentido horario mientras que las hélices delantero-trasero giran en sentido anti-horario. Este movimiento sólo conduce a una aceleración de ángulo de guiñada (en primera aproximación). En adelante en este texto, el desequilibrio de pares en el eje z_B será representado por la variable U_4.

1.2 Modelado dinámico del Cuadricóptero

El paso previo al desarrollo del control es un adecuado modelado del sistema dinámico. El modelo dinámico del cuadricóptero se presenta bajo la formulación matemática de Newton-Euler. Esta formulación se basa en la física de Newton (equilibrio de sumatoria de fuerzas y momentos) y la relación de los sistemas inerciales por medio de los ángulos de Euler.

El cuadricóptero puede ser considerado como un cuerpo rígido en el espacio, sujeto a una fuerza principal que produce el movimiento *throttle* y tres momentos producidos por desequilibrios en las fuerzas de empuje de los rotores. En la Figura 3 se muestran las fuerzas de cada uno de los rotores que producen el movimiento "*roll*", "*pitch*" y "*yaw*".

El par generador del movimiento de balanceo o de "*roll*" (ángulo ϕ) se realiza mediante un desequilibrio entre las fuerzas f_2 y f_4. Para el movimiento de cabeceo o de "*pitch*" (ángulo θ), el desequilibrio se realiza entre las fuerzas f_1 y f_3. El movimiento de guiñada o "*yaw*" (ángulo ψ), se produce por desequilibrio entre pares horarios contra pares anti-horarios los cuales son proporcionales a las fuerzas de empujes. Finalmente, el empuje total, para el desplazamiento perpendicularmente al plano de los rotores, se obtiene como la suma de las cuatro fuerzas f_1, f_2, f_3 y f_4 que ejercen los rotores.

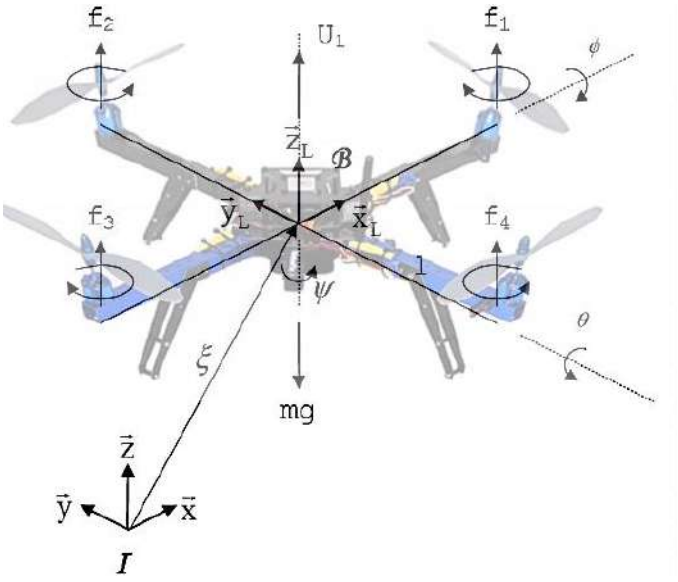

Figura 3 Posición y orientación de un Cuadricóptero. Fuente: Elaboración propia.

Especialmente para sistemas de vuelo de peso ligero, sus modelos dinámicos deben incluir idealmente los efectos giroscópicos resultantes tanto de la rotación del cuerpo rígido en el espacio y la rotación de las hélices. Estos aspectos han sido a menudo descuidados en trabajos anteriores. Sin embargo, los principales efectos que actúan sobre un helicóptero se describen brevemente en la Tabla 1. [5] [20] donde C representa términos constantes, Ω es la velocidad del rotor, J_R es el momento de inercia rotacional del rotor alrededor de su eje, l es la distancia del centro de masa hacia los rotores, J es el momento de inercia del cuerpo rígido y ϕ, θ y ψ son los ángulos de Tait-Bryan [20].

Tabla 1 Principales efectos físicos que afectan a un helicóptero [20]

Efecto	Fuente	Formulación
Efectos Aerodinámicos	Rotación de los rotores. Giro de las hélices.	$C\Omega^2$
Pares inerciales opuestos	Cambio en la velocidad de rotación de la hélice	$J_R\dot{\Omega}$
Efectos de la gravedad	Posición del centro de masa	l
Efectos giroscópicos	Cambio en la orientación del cuerpo rígido. Cambio en la orientación del plano de los rotores.	$I\theta\psi$ $J\Omega\theta,\phi$
Fricción	Todo movimiento del helicóptero	$C\dot{\phi},\dot{\theta},\dot{\psi}$

El cuadricóptero es un sistema mecánico con 6 grados de libertad (3 posiciones lineales y 3 posiciones angulares) y solamente 4 entradas de control (f_1, f_2, f_3 y f_4). Se asume que el centro de masa del cuadricóptero coincide con el origen del sistema de coordenadas fijo al mismo, además se supone que la estructura del cuadricóptero es simétrica, lo cual produce que su matriz de inercia sea diagonal.

1.2.1 Orientación del Cuadricóptero

Antes de obtener el modelo matemático del cuadricóptero, se obtendrá el modelo de posición y orientación de un cuerpo rígido respecto a un sistema de coordenadas de referencia inercial. El cuadricóptero, como sólido rígido, está caracterizado por un sistema de coordenadas ligado a él y con origen en su centro de masa [20].

En la Figura 3 se muestra el esquema de un cuadricóptero con un sistema de coordenadas fijo al cuerpo en su centro de masa $\vec{\boldsymbol{B}} = [\overrightarrow{X_L}, \overrightarrow{Y_L}, \overrightarrow{Z_L}]$ y un sistema de coordenadas de referencia inercial $\vec{\boldsymbol{I}} = [\vec{X}, \vec{Y}, \vec{Z}]$ el cual es considerado fijo respecto a la tierra.

El sistema de coordenadas $\vec{\boldsymbol{B}} = [\overrightarrow{X_L}, \overrightarrow{Y_L}, \overrightarrow{Z_L}]$ corresponde al sistema de coordenadas fijo al cuadricóptero en su centro de masa, donde el eje $\overrightarrow{X_L}$ es la dirección normal de ataque del cuadricóptero, $\overrightarrow{Y_L}$ es ortogonal a $\overrightarrow{X_L}$ y es positivo hacia estribor en el plano horizontal, mientras que el eje $\overrightarrow{Z_L}$ está orientado en sentido ascendiente y ortogonal al plano $\overrightarrow{X_L}\overrightarrow{Y_L}$.

El sistema de coordenadas $\vec{\boldsymbol{I}} = [\vec{X}, \vec{Y}, \vec{Z}]$ corresponde al sistema de coordenadas fijo respecto a la tierra, donde $\vec{X}$ es el eje tangente al meridiano magnético y es positivo cuando señala al

polo norte magnético, $\vec{Z}$ es el eje vertical local, que apunta hacia arriba (sentido de decrecimiento de la gravedad g), $\vec{Y}$ es el eje definido tal que forma un triedro directo con $\vec{X}$ y $\vec{Z}$. [21].

La orientación del vehículo se supondrá que es dada por una matriz de rotación ortonormal $\boldsymbol{R}$, donde $\boldsymbol{R}$ pertenece al grupo ortogonal especial $SO(3)$, que transforma los cambios de orientación en el sistema $\vec{\boldsymbol{B}}$ hacia el sistema $\vec{\boldsymbol{I}}$ [2]. La rotación de un VANT o, en líneas más generales, la de un cuerpo rígido puede ser obtenida utilizando los ángulos de Euler.

Mediante los ángulos de Euler se puede representar la orientación relativa de dos sistemas de coordenadas. En este trabajo se utiliza la convención XYZ (giro alrededor de X, Y', Z''), la cual es muy utilizada para aplicaciones de ingeniería aeroespacial y se nombra ángulos de Tait-Bryan, también conocidos por ángulos Cardano [20].

Los ángulos de Tait-Bryan [20] son tres ángulos que describirán la rotación de un cuerpo rígido interpretándolo como consecuencia de la rotación sucesiva en torno a los tres eje $\overrightarrow{X_L}, \overrightarrow{Y_L}$ y $\overrightarrow{Z_L}$. En la Figura 4 se muestra la rotación de un cuerpo rígido a través de rotaciones sucesivas y se definen los ángulos de rotación que serán de mucha importancia para definir la matriz de rotación ortonormal $\boldsymbol{R}$.

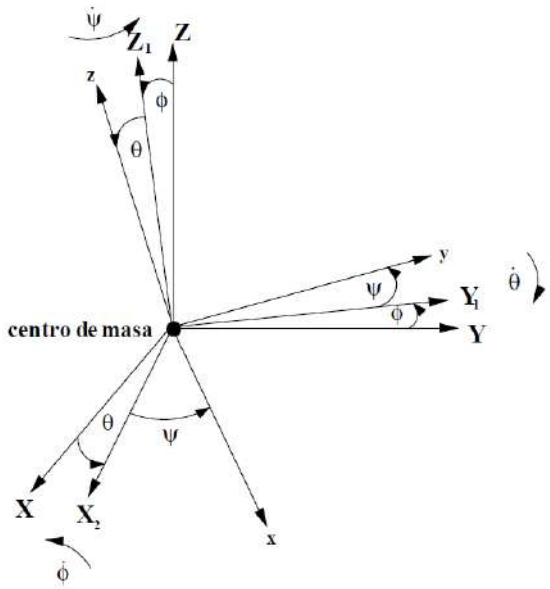

Figura 4 Rotación de un cuerpo rígido [20].

A continuación se describe la rotación de un cuerpo rígido:

Rotación según $\vec{X}$ de ϕ:

El primer giro se realiza respecto al eje $\overrightarrow{X_L}$, correspondiente al ángulo *roll* o de balanceo. En la Figura 4 se muestra el sistema de coordenadas con un primer giro en un ángulo ϕ. Como consecuencia de dicho giro, el eje $\overrightarrow{Y_L}$ y $\overrightarrow{Z_L}$ se han girado un ángulo ϕ y para efectos de cálculos serán renombrados como $\overrightarrow{Y_1}$ y $\overrightarrow{Z_1}$. Matemáticamente se tiene la siguiente expresión:

$$\begin{bmatrix} \overrightarrow{X_1} \\ \overrightarrow{Y_1} \\ \overrightarrow{Z_1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & -sen\phi \\ 0 & sen\phi & cos\phi \end{bmatrix} \begin{bmatrix} \overrightarrow{X_L} \\ \overrightarrow{Y_L} \\ \overrightarrow{Z_L} \end{bmatrix} \tag{1}$$

Rotación según $\vec{Y}$ de θ:

El segundo giro se realiza respecto al eje $\overrightarrow{Y_1}$, correspondiente al ángulo *pitch* o de cabeceo. En la Figura 4 se muestra el sistema de coordenadas con un giro de θ. Como consecuencia de dicho giro, el eje $\overrightarrow{X_1}$ y $\overrightarrow{Z_1}$ se han girado un ángulo θ y para efectos de cálculos serán renombrados como $\overrightarrow{X_2}$ y $\overrightarrow{Z_2}$. Matemáticamente se tiene la siguiente expresión:

$$\begin{bmatrix} \overrightarrow{X_2} \\ \overrightarrow{Y_2} \\ \overrightarrow{Z_2} \end{bmatrix} = \begin{bmatrix} cos\theta & 0 & sen\theta \\ 0 & 1 & 0 \\ -sen\theta & 0 & cos\theta \end{bmatrix} \begin{bmatrix} \overrightarrow{X_1} \\ \overrightarrow{Y_1} \\ \overrightarrow{Z_1} \end{bmatrix} \tag{2}$$

Rotación según $\vec{Z}$ de ψ:

El giro se realiza respecto al eje $\overrightarrow{Z_2}$, correspondiente al ángulo *yaw* o de guiñada. En la Figura 4 se muestra el sistema de coordenadas con un giro de ψ. Como consecuencia de dicho giro, el eje $\overrightarrow{X_2}$ y $\overrightarrow{Y_2}$ se han girado un ángulo ψ y para efectos de cálculos serán renombrados como $\overrightarrow{X_3}$ y $\overrightarrow{Y_3}$. Matemáticamente se tiene la siguiente expresión:

$$\begin{bmatrix} \overrightarrow{X_3} \\ \overrightarrow{Y_3} \\ \overrightarrow{Z_3} \end{bmatrix} = \begin{bmatrix} cos\psi & -sen\psi & 0 \\ sen\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \overrightarrow{X_2} \\ \overrightarrow{Y_2} \\ \overrightarrow{Z_2} \end{bmatrix} \tag{3}$$

A partir de las rotaciones presentadas anteriormente, se define la matriz de rotación completa de $\vec{\boldsymbol{B}}$ respecto a $\vec{\boldsymbol{I}}$, que relaciona las tres rotaciones sucesivas de los ángulos de Tait-Bryan del sistema fijado sobre el Cuadricóptero con la rotación respecto al sistema de inercia fijo en la tierra. Dicho de otra manera, se determina a partir de los ángulos de Tait-Bryan la orientación del sistema $\vec{\boldsymbol{B}}$ respecto al sistema $\vec{\boldsymbol{I}}$. La matriz de transformación $\boldsymbol{R}$ es llamada Matriz Coseno Directa y se deduce como se explica en [2].

$$\boldsymbol{R} = \begin{bmatrix} cos\psi & -sen\psi & 0 \\ sen\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} cos\theta & 0 & sen\theta \\ 0 & 1 & 0 \\ -sen\theta & 0 & cos\theta \end{bmatrix} . \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & -sen\phi \\ 0 & sen\phi & cos\phi \end{bmatrix}$$

$$\boldsymbol{R} = \begin{bmatrix} cos\psi cos\theta & cos\psi sen\theta sen\phi - sen\psi cos\phi & cos\psi sen\theta cos\phi + sen\psi cos\phi \\ sen\psi cos\theta & sen\psi sen\theta sen\phi + cos\psi cos\phi & sen\psi sen\theta cos\phi - cos\psi sen\phi \\ -sen\theta & cos\theta sen\phi & cos\theta cos\phi \end{bmatrix} \tag{4}$$

Siendo $\omega = (p, q, r)^T$ el vector de velocidades angulares respecto al sistema fijado al cuerpo rígido y $\Theta = (\phi, \theta, \psi)^T$ el vector de ángulos de Euler. La relación entre $\dot{\Theta}$ y ω es según [22].

$$\dot{\Theta} = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \boldsymbol{T}\omega \tag{5}$$

Donde la variable $\boldsymbol{T}$ es llamada matriz de Euler y es dada por [22]:

$$\boldsymbol{T} = \begin{bmatrix} 1 & sen\phi tan\theta & cos\phi tan\theta \\ 0 & cos\phi & -sen\phi \\ 0 & sen\phi sec\theta & cos\phi sec\theta \end{bmatrix} \tag{6}$$

Reemplazando la ecuación (6) en la ecuación (5) se obtiene la relación entre Θ y ω:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & sen\phi tan\theta & cos\phi tan\theta \\ 0 & cos\phi & -sen\phi \\ 0 & sen\phi sec\theta & cos\phi sec\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{7}$$

La transformación de cantidades angulares relaciona las velocidades angulares p, q y r en los ejes del sistema de referencia fijo al cuadricóptero y las componentes de la velocidad angular $\dot{\phi}$,$\dot{\theta}$ y $\dot{\psi}$ respecto a los ejes del sistema inercial. Cuando los ángulos ϕ, θ y ψ son pequeños, la ecuación (7) puede aproximarse como se muestra en la ecuación (8):

$$\begin{aligned} p &= \dot{\phi} \\ q &= \dot{\theta} \\ r &= \dot{\psi} \end{aligned} \tag{8}$$

La ecuación (8) es ampliamente utilizada en trabajos de control de estabilización del vuelo, ya que en estas condiciones los ángulos son relativamente pequeños.

1.2.2 Modelo Aerodinámico para configuración en cruz

La cinemática de un cuerpo rígido con 6 grados de libertar viene dado como [23, 19]:

$$\dot{\xi} = J_{\Theta}\nu \tag{9}$$

Donde ξ está compuesto de un vector de posición lineal y angular respecto al sistema de coordenadas inerciales:

$$\xi = \begin{bmatrix} \Gamma \\ \Theta \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ \phi \\ \theta \\ \psi \end{bmatrix} \tag{10}$$

De manera similar, ν está compuesto de un vector de velocidad lineal y velocidad angular respecto al sistema de coordenadas fijado en el cuerpo rígido:

$$\nu = \begin{bmatrix} V \\ \omega \end{bmatrix} = \begin{bmatrix} u \\ v \\ w \\ p \\ q \\ r \end{bmatrix} \tag{11}$$

Y J_Θ es la matriz de conversión:

$$J_\Theta = \begin{bmatrix} \boldsymbol{R} & 0_{3x3} \\ 0_{3x3} & \boldsymbol{T} \end{bmatrix} \tag{12}$$

La matriz de rotación $\boldsymbol{R}$ se ha definido en la ecuación (4). La matriz de transferencia $\boldsymbol{T}$ se define según [19, 20]:

$$\boldsymbol{T} = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & sen\phi tan\theta & cos\phi tan\theta \\ 0 & cos\phi & -sen\phi \\ 0 & sen\phi sec\theta & cos\phi sec\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{13}$$

Las derivadas de los ángulos de Tait-Bryan $(\dot{\phi}, \dot{\theta}, \dot{\psi})$ son distintas de las velocidades angulares en el sistema de coordenadas del cuerpo rígido (p, q, r). La matriz de transformación entre las velocidades angulares del sistema fijado al cuerpo y la variación de los ángulos de Tait-Bryan viene dado por la inversa de $\boldsymbol{T}$:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -sen\theta \\ 0 & cos\phi & sen\phi cos\theta \\ 0 & -sen\phi & cos\phi cos\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{14}$$

La dinámica de un cuerpo rígido bajo fuerzas y torques externos aplicados al centro de masa y expresados en el sistema de coordenadas ligado al cuerpo se puede obtener a través de la formulación de Newton-Euler [24]:

$$\begin{bmatrix} mI_{3x3} & 0_{3x3} \\ 0_{3x3} & I \end{bmatrix} \begin{bmatrix} \dot{V} \\ \dot{\omega} \end{bmatrix} + \begin{bmatrix} \omega x m V \\ \omega x I \omega \end{bmatrix} = \begin{bmatrix} F_B \\ \tau_B \end{bmatrix} \tag{15}$$

Donde $I_{3x3} \in \Re^{3x3}$ es la matriz identidad, V es el vector velocidad lineal del cuerpo respecto a $\vec{\boldsymbol{B}}$, ω es la velocidad angular del cuerpo respecto a $\vec{\boldsymbol{B}}$ y m es la masa total del cuerpo. $I \in \Re^3$ es la matriz de inercia respecto a $\vec{\boldsymbol{B}}$ [20] y se supone diagonal.

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{16}$$

Para un sistema de coordenadas cuyo origen coincide con el centro de masa del cuerpo, el término $\omega x m V$ es igual a cero, por lo tanto la ecuación (15) se simplifica.

$$\begin{bmatrix} mI_{3x3} & 0_{3x3} \\ 0_{3x3} & I \end{bmatrix} \begin{bmatrix} \dot{V} \\ \dot{\omega} \end{bmatrix} + \begin{bmatrix} 0 \\ \omega x I \omega \end{bmatrix} = \begin{bmatrix} F_B \\ \tau_B \end{bmatrix} \tag{17}$$

Se pueden escribir las ecuaciones de movimiento de un cuerpo rígido como se muestra en las ecuaciones (18), (19) y (20) según [22]:

$$\dot{\Gamma} = [\dot{X} \quad \dot{Y} \quad \dot{Z}]^T = v \tag{18}$$

$$m\dot{v} = \boldsymbol{R}F_B \tag{19}$$

$$\dot{v} = [\ddot{X} \quad \ddot{Y} \quad \ddot{Z}]^T \tag{20}$$

Además:

$$I\dot{\omega} = -\omega x I\omega + \tau_B \tag{21}$$

$$\dot{\omega} = [\dot{p} \quad \dot{q} \quad \dot{r}]^T \tag{22}$$

En la ecuación (19), $F_B \in \vec{\boldsymbol{B}}$ denota a las fuerzas externas aplicadas al cuerpo rígido que consisten en su propio peso, en el vector de fuerzas aerodinámicas y el empuje. En la ecuación (21), $\tau_B \in \vec{\boldsymbol{B}}$ denota a los pares externos aplicados al cuerpo rígido que consisten en los pares desarrollados por los cuatro motores. Estas fuerzas y pares pueden ser expresados según [20] y [6] como se muestra en las ecuaciones (23) y (24).

$$\boldsymbol{R}F_B = -mge_3 + \boldsymbol{R}_{e_3}\left(b\sum_{i=1}^{4}{\Omega_i}^2\right) \tag{23}$$

$$\tau_B = \underbrace{-\sum_{i=1}^{4} J_R(\omega x e_3).\Omega_i}_{efecto\ giroscópico} + \tau_a \tag{24}$$

Donde g es la aceleración de la gravedad, e_3 es una componente básica de $\Re^3$, J_R es el momento de inercia rotacional del rotor alrededor de su eje, b es el coeficiente de empuje aplicado por los rotores y Ω_i es la velocidad angular del i-ésimo rotor. El cuadricóptero experimenta un torque giroscópico de acuerdo a la ecuación (24) cuando existe un desbalance general en la suma algebraica de las velocidades de los rotores.

La fuerza principal aplicada al helicóptero U_1 y responsable del movimiento vertical o t*hrottle*, viene modelada como [25]:

$$U_1 = \sum_{i=1}^{4} f_i = \sum_{i=1}^{4} b{\Omega_i}^2 \tag{25}$$

Donde f_i es la fuerza de empuje generada por el i-ésimo rotor.

Para modelar las demás fuerzas que generan los movimientos de cabeceo, balanceo y guiñada se debe tener en cuenta que el momento aplicado en el cuerpo a lo largo de un eje es causado por la diferencia de empujes entre los rotores ubicados en el eje perpendicular. Por lo tanto, el movimiento de cabeceo (*pitch*) se obtiene debido a la diferencia de empuje del rotor frontal y el rotor trasero. El movimiento de balanceo (*roll*) se obtiene debido a la diferencia de empuje entre el rotor de la izquierda y el rotor de la derecha. El movimiento

de guiñada (*yaw*) se obtiene debido a la diferencia de los pares entre los dos rotores que giran en sentido horario y los dos rotores que giran en sentido anti-horario. Los pares aplicados en los tres ejes de orientación del cuadricóptero, vienen expresados como indica la ecuación (26) según [25].

$$\tau_a = \begin{bmatrix} l(f_4 - f_2) \\ l(f_3 - f_1) \\ \sum_{i=1}^{4} \tau_{M_i} \end{bmatrix} = \begin{bmatrix} lb({\Omega_4}^2 - {\Omega_2}^2) \\ lb({\Omega_3}^2 - {\Omega_1}^2) \\ d({\Omega_2}^2 + {\Omega_4}^2 - {\Omega_1}^2 - {\Omega_3}^2) \end{bmatrix} = \begin{bmatrix} lU_2 \\ lU_3 \\ U_4 \end{bmatrix} \tag{26}$$

Donde l es la distancia entre los motores y el centro de gravedad. Las constantes b y d corresponden al coeficiente de empuje y coeficiente de arrastre de los rotores respectivamente.

Reemplazando en la ecuación (17) las ecuaciones (18) hasta la (26) se obtiene el modelo matemático no lineal del cuadricóptero indicado entre la ecuación (27) hasta la ecuación (35) según [20].

$$\ddot{X} = (sen\psi sen\phi + \cos\psi\, sen\theta cos\phi)\frac{U_1}{m} \tag{27}$$

$$\ddot{Y} = (-\cos\psi\, sen\phi + sen\psi sen\theta cos\phi)\frac{U_1}{m} \tag{28}$$

$$\ddot{Z} = -g + (\cos\theta cos\phi)\frac{U_1}{m} \tag{29}$$

$$\dot{\phi} = p + qsen\phi tan\theta + rcos\phi tan\theta \tag{30}$$

$$\dot{\theta} = qcos\phi - rsen\phi \tag{31}$$

$$\dot{\psi} = qsen\phi sec\theta + rcos\phi sec\theta \tag{32}$$

$$\dot{p} = \frac{I_{YY} - I_{ZZ}}{I_{XX}} qr - \frac{J_{TP}}{I_{XX}} q\Omega + \frac{lU_2}{I_{XX}} \tag{33}$$

$$\dot{q} = \frac{I_{ZZ} - I_{XX}}{I_{YY}} pr + \frac{J_{TP}}{I_{YY}} p\Omega + \frac{lU_3}{I_{YY}} \tag{34}$$

$$\dot{r} = \frac{I_{XX} - I_{YY}}{I_{ZZ}} pq + \frac{U_4}{I_{ZZ}} \tag{35}$$

Donde Ω se define como la velocidad general mediante la ecuación (36). Cabe tener en cuenta que los sentidos de giro de los cuatro rotores no son los mismos y por lo tanto algunas velocidades de giro son tomadas con signo negativo.

$$\Omega = \Omega_1 + \Omega_2 + \Omega_3 + \Omega_4 \tag{36}$$

La relación entre la velocidad del rotor y las fuerzas sobre el cuadricóptero vienen expresadas a partir de la ecuación (37) Donde b y d son los coeficientes de empuje y de arrastre respectivamente.

$$\begin{bmatrix} U_2 \\ U_3 \\ U_4 \end{bmatrix} = \begin{bmatrix} b({\Omega_4}^2 - {\Omega_2}^2) \\ b({\Omega_3}^2 - {\Omega_1}^2) \\ d({\Omega_2}^2 + {\Omega_4}^2 - {\Omega_1}^2 - {\Omega_3}^2) \end{bmatrix} \tag{37}$$

Por lo tanto el modelo completo de las fuerzas o también llamados movimientos básicos es:

$$\begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} = \begin{bmatrix} b({\Omega_1}^2 + {\Omega_2}^2 + {\Omega_3}^2 + {\Omega_4}^2) \\ b({\Omega_4}^2 - {\Omega_2}^2) \\ b({\Omega_3}^2 - {\Omega_1}^2) \\ d({\Omega_2}^2 + {\Omega_4}^2 - {\Omega_1}^2 - {\Omega_3}^2) \end{bmatrix} \tag{38}$$

En la ecuación (38) se describe la relación entre las velocidades angulares y las fuerzas básicas sobre el cuadricóptero.

El modelo aerodinámico completo del cuadricóptero considerando la igualdad de la ecuación (8) se expresa finalmente en el sistema de ecuaciones (39) hasta (44) según [23], [6], y [19].

$$\ddot{X} = (sen\psi sen\phi + \cos\psi\, sen\theta cos\phi)\frac{U_1}{m} \tag{39}$$

$$\ddot{Y} = (-\cos\psi\, sen\phi + sen\psi sen\theta cos\phi)\frac{U_1}{m} \tag{40}$$

$$\ddot{Z} = -g + (\cos\theta cos\phi)\frac{U_1}{m} \tag{41}$$

$$\ddot{\phi} = \dot{p} = \frac{I_{YY} - I_{ZZ}}{I_{XX}} qr - \frac{J_{TP}}{I_{XX}} q\Omega + \frac{lU_2}{I_{XX}} \tag{42}$$

$$\ddot{\theta} = \dot{q} = \frac{I_{ZZ} - I_{XX}}{I_{YY}} pr + \frac{J_{TP}}{I_{YY}} p\Omega + \frac{lU_3}{I_{YY}} \tag{43}$$

$$\ddot{\psi} = \dot{r} = \frac{I_{XX} - I_{YY}}{I_{ZZ}} pq + \frac{U_4}{I_{ZZ}} \tag{44}$$

Con este planteamiento es posible (en teoría) determinar la posición y orientación del cuadricóptero por doble integración de sus aceleraciones lineales y angulares. Para realizar esta operación, sólo basta conocer el punto de partida de las variables internas y las fuerzas sobre los ejes del cuadricóptero U_1, U_2, U_3 y U_4. Este proceso también se conoce como cinemática directa y la dinámica directa.

Cara a la implementación es importante tener en cuenta que las variables U_1, U_2, U_3 y U_4 son desbalances de fuerzas, dichas fuerzas son producidas por la velocidad de giro de las hélices, las cuales son dependientes de los voltajes que alimentan a los motores.

Sin embargo, el objetivo en la realidad para estabilizar el cuadricóptero se basa en encontrar aquellos valores de voltaje que alimentan a los motores para mantener el cuadricóptero en una cierta posición requerida en la tarea. Este proceso también se conoce como cinemática inversa y dinámica inversa. A diferencia de los directos, las operaciones inversas no siempre son posibles y no siempre únicas. Por estas razones su consideración es mucho más complicada.

Se explica en [19] que la dinámica de un cuadricóptero debe simplificarse mucho para ofrecer un modelo inverso fácil que puede ser implementado en los algoritmos de control. Las ecuaciones (39) hasta (44) pueden ser reorganizadas según tres consideraciones de [19].

• "Las contribuciones angulares provienen de productos cruzados de velocidades angulares (efectos giroscópicos y coriolis) son bastante complejas en el modelo matemático. Puesto que el movimiento del cuadricóptero se puede suponer cerca de la condición de vuelo estacionario, pequeños cambios angulares se producen (especialmente para balanceo y cabeceo). De ello se desprende que estos términos pueden simplificarse porque son más pequeños que los efectos principales", ver sistema de ecuaciones (45).

$$\frac{I_{YY}-I_{ZZ}}{I_{XX}}qr-\frac{J_{TP}}{I_{XX}}q\Omega=0$$

$$\frac{I_{ZZ}-I_{XX}}{I_{YY}}pr+\frac{J_{TP}}{I_{YY}}p\Omega=0 \tag{45}$$

$$\frac{I_{XX}-I_{YY}}{I_{ZZ}}pq=0$$

• "Las aceleraciones angulares hacen referencia a los ángulos del cuadricóptero medidos en el sistema de referencia fijo al VANT. Estas aceleraciones no son iguales a las aceleraciones de los ángulos de Euler que determinan la orientación en el sistema de referencia inercial. La matriz de transferencia $\boldsymbol{T}$ define la relación entre las velocidades angulares en el sistema de referencia inercial en tierra y aquellas velocidades en el sistema de referencia fijo al cuerpo. Dado que en condiciones cercanas a la estabilidad, la matriz T es aproximada a la matriz identidad, las ecuaciones de aceleración se referencian directamente a las aceleraciones angulares de Euler", ver sistema de ecuaciones (46).

$$\ddot{\phi}=\dot{p}$$

$$\ddot{\theta}=\dot{q} \tag{46}$$

$$\ddot{\psi}=\dot{r}$$

• "El algoritmo de control calcula las señales adecuadas a los motores. Puesto que son cuatro señales manipulables, no más de cuatro variables se pueden controlar en el bucle. Desde el inicio del proyecto, se ha decidido a estabilizar el cuadricóptero (ángulos de Euler) y la altura. De acuerdo con esta elección, se han eliminado las ecuaciones que describen la posición X e Y".

Por lo tanto las ecuaciones utilizadas para el control de ángulos del cuadricóptero son las indicadas en (47).

$$\ddot{Z}=-g+(\cos\theta cos\phi)\frac{U_1}{m} \tag{47}$$

$$\ddot{\phi} = \dot{p} = \frac{lU_2}{I_{XX}}$$

$$\ddot{\theta} = \dot{q} = \frac{lU_3}{I_{YY}}$$

$$\ddot{\psi} = \dot{p} = \frac{U_4}{I_{ZZ}}$$

1.2.3 Modelo Aerodinámico para configuración en equis

El modelo del cuadricóptero presentado en la sección 1.2.2 considera los motores frontal y trasero alineados con el eje $\overrightarrow{X_L}$ y los motores izquierdo y derecho alineados con el eje $\overrightarrow{Y_L}$. En esta sección se presentan las ecuaciones para un cuadricóptero en configuración tipo equis, donde se considera dos motores frontales y dos motores traseros. El modelo dinámico también está basado en la formulación de Newton-Euler. En la Figura 5 [26] se muestra una representación de una cuadricóptero con configuración en equis además de los sistemas de referencias y fuerzas [26].

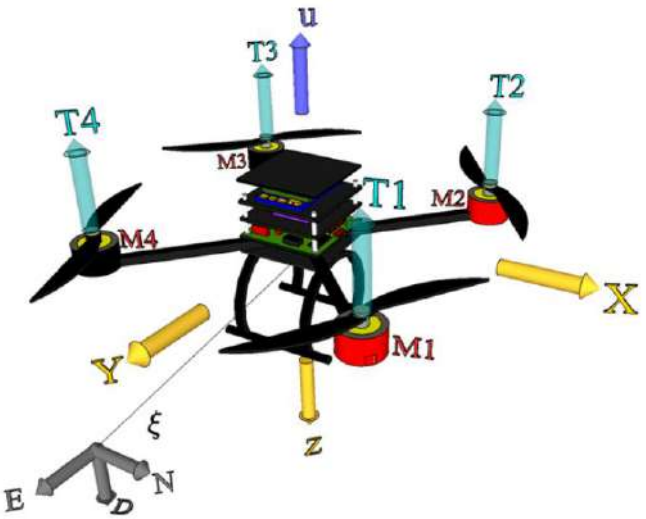

Figura 5 Posición y orientación de un cuadricóptero con configuración en equis [26].

El vector posición del centro de masa del cuadricóptero es denominado por $\xi = (x, y, z)^T$, representa las coordenadas de posición del vehículo respecto al sistema de referencia inercial.

La dinámica completa no lineal del cuadricóptero puede ser expresada como:

$$\begin{bmatrix} mI_{3x3} & 0_{3x3} \\ 0_{3x3} & I \end{bmatrix} \begin{bmatrix} \dot{V} \\ \dot{\omega} \end{bmatrix} + \begin{bmatrix} 0 \\ \omega x I \omega \end{bmatrix} = \begin{bmatrix} F_B \\ \tau_B \end{bmatrix} \tag{48}$$

Se denota U_1 como la fuerza responsable del movimiento vertical, U_2 y U_3 como el desequilibrio de fuerzas que producen movimiento de *"roll"* y *"Pitch"* respectivamente y U_4 es el torque que produce movimiento de *"yaw"*.

$$\begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} = \begin{bmatrix} b({\Omega_1}^2 + {\Omega_2}^2 + {\Omega_3}^2 + {\Omega_4}^2) \\ b({\Omega_4}^2 + {\Omega_3}^2 - {\Omega_1}^2 - {\Omega_2}^2) \\ b({\Omega_2}^2 + {\Omega_3}^2 - {\Omega_1}^2 - {\Omega_4}^2) \\ d({\Omega_1}^2 + {\Omega_3}^2 - {\Omega_2}^2 - {\Omega_4}^2) \end{bmatrix} \tag{49}$$

$$\begin{cases} {\Omega_1}^2 = \frac{1}{4b}U_1 + \frac{1}{4b}U_2 - \frac{1}{4b}U_3 - \frac{1}{4d}U_4 \\ {\Omega_2}^2 = \frac{1}{4b}U_1 - \frac{1}{4b}U_2 - \frac{1}{4b}U_3 + \frac{1}{4d}U_4 \\ {\Omega_3}^2 = \frac{1}{4b}U_1 - \frac{1}{4b}U_2 + \frac{1}{4b}U_3 - \frac{1}{4d}U_4 \\ {\Omega_4}^2 = \frac{1}{4b}U_1 + \frac{1}{4b}U_2 + \frac{1}{4b}U_3 + \frac{1}{4d}U_4 \end{cases} \tag{50}$$

La ecuación (49) es la única diferencia entre las configuraciones en cruz y equis. A partir de la ecuación (48), siguiendo el mismo procedimiento utilizado para determinar el modelo con configuración en equis y considerando la ecuación (49), se obtiene el modelo no lineal presentado en las ecuaciones (51) hasta (59).

$$\ddot{X} = (sen\psi sen\phi + \cos\psi\, sen\theta cos\phi)\frac{U_1}{m} \tag{51}$$

$$\ddot{Y} = (-\cos\psi\, sen\phi + sen\psi sen\theta cos\phi)\frac{U_1}{m} \tag{52}$$

$$\ddot{Z} = -g + (\cos\theta cos\phi)\frac{U_1}{m} \tag{53}$$

$$\dot{\phi} = p + qsen\phi tan\theta + rcos\phi tan\theta \tag{54}$$

$$\dot{\theta} = qcos\phi - rsen\phi \tag{55}$$

$$\dot{\psi} = qsen\phi sec\theta + rcos\phi sec\theta \tag{56}$$

$$\dot{p} = \frac{I_{YY} - I_{ZZ}}{I_{XX}}qr - \frac{J_{TP}}{I_{XX}}q\Omega + \frac{lU_2}{I_{XX}} \tag{57}$$

$$\dot{q} = \frac{I_{ZZ} - I_{XX}}{I_{YY}}pr + \frac{J_{TP}}{I_{YY}}p\Omega + \frac{lU_3}{I_{YY}} \tag{58}$$

$$\dot{r} = \frac{I_{XX} - I_{YY}}{I_{ZZ}}pq + \frac{U_4}{I_{ZZ}} \tag{59}$$

Las ecuaciones (51) hasta (59) correspondientes al modelo aerodinámico para la configuración en equis del cuadricóptero es igual a la presentada 1.2.2 para la configuración en cruz. Sin embargo, para un mismo cuadricóptero, las inercias I_{XX}, I_{YY} y I_{ZZ} son distintas para cada configuración debido al cambio de dirección de los ejes $\vec{\boldsymbol{B}}$.

El modelo simplificado para la configuración en equis tiene las consideraciones presentadas en la sección 1.2.2 y por lo tanto las mimas ecuaciones que la configuración en cruz.

$$\ddot{Z} = -g + (\cos\theta cos\phi)\frac{U_1}{m}$$

$$\ddot{\phi} = \dot{p} = \frac{lU_2}{I_{XX}}$$

$$\ddot{\theta} = \dot{q} = \frac{lU_3}{I_{YY}} \tag{60}$$

$$\ddot{\psi} = \dot{p} = \frac{U_4}{I_{ZZ}}$$

Es importante recalcar que en la configuración en equis las variables U_2, U_3 son calculadas según la ecuación (49) mientras en la configuración en cruz se utiliza la ecuación (38).

1.3 Modelo del rotor

El rotor es un actuador el cual convierte la energía eléctrica en energía mecánica y viceversa. El circuito del motor DC es controlado por un voltaje de alimentación v en voltios. En teoría, el modelo debe ser compuesto por una resistencia R en serie representando las pérdidas. Un circuito eléctrico básico de un motor está compuesto además de una inductancia L. Por lo tanto el modelo viene dado por:

$$v = Ri + L\frac{\partial i}{\partial t} + K_E\omega_M \tag{61}$$

Donde $i[A]$ es la corriente a través del motor, $K_E[V/rad]$ es llamada constante del motor y $\omega_M[rad/s]$ es la velocidad angular del motor.

El aporte de la parte inductiva es importa para determinar la característica del motor DC. Sin embargo muchas veces es despreciado en los cálculos mecánicos por tres principales aspectos según [19].

Muchos de los motores usados en robots tienen inductancias pequeñas gracias a su óptima construcción.

El polo de la parte eléctrica es siempre mucho más rápido que el polo mecánico, por lo tanto la velocidad de respuesta del sistema será determinado sólo por el polo más lento.

Es mucho más fácil del resolver una ecuación diferencial de primer orden que una de segundo orden.

Entonces la ecuación (61) puede ser simplificada.

$$v = Ri + K_E\omega_M \tag{62}$$

La dinámica del motor es según [19] se describe como se indica en la ecuación (63).

$$J_{TM}\dot{\omega_M} = T_M - T_L \tag{63}$$

Donde $J_{TM}[Nms^2]$ es el momento de inercial rotacional total del motor, $\dot{\omega_M}[rad/s^2]$ es la aceleración angular del motor, $T_M[Nm]$ es el torque del motor y $T_L[Nm]$ es el torque de la carga. El torque del motor T_M es proporcional a la corriente eléctrica i. El coeficiente $K_M[Nm/A]$ es una constante del motor que relaciona la corriente $i[A]$ y el torque $T_M[Nm]$.

$$J_{TM}\dot{\omega_M} = K_M i - T_L \tag{64}$$

Combinando la ecuación (62) y (64) se obtienen (65).

$$J_{TM}\dot{\omega_M} = -\frac{K_E K_M}{R}\omega_M - T_L + \frac{K_M}{R}v \tag{65}$$

Ambas constantes K_E y K_M tienen el mismo valor, sin embargo son dimensionalmente diferentes.

Se considera una caja de cambios con relación de transformación $r = \frac{\omega_M}{\omega_P}$ y eficiencia de la caja de engranajes η, donde $\omega_P[rad/s]$ es la velocidad del *propeller* o hélice. Se despeja de la ecuación (65) la variable de interés $\dot{\omega_M}$ como se muestra en (66) según se demuestra en [19].

$$\dot{\omega_M} = -\frac{{K_M}^2}{RJ_{TM}}\omega_M - \frac{d}{\eta r^3 J_{TM}}{\omega_M}^2 + \frac{K_M}{RJ_{TM}}v \tag{66}$$

Donde $d[Nms^2]$ es el factor de arrastre. Además $J_{TM} = J_M + \frac{J_P}{\eta r^2}$ es la inercia total en el motor. La ecuación (66) puede ser linealizada alrededor de un punto de operación $\dot{\omega_0}$ según lo demuestran [5] [6].

$$\dot{\omega_M} = -A_M\omega_M + B_M v + C_M \tag{67}$$

Donde las constantes se calculan según se indica en [6], ver ecuaciones (68) hasta (70).

$$A_M = \left(\frac{{K_M}^2}{RJ_T} + \frac{2d\omega_0}{\eta r^3 J_T}\right) \tag{68}$$

$$B_M = \left(\frac{K_M}{RJ_T}\right) \tag{69}$$

$$C_M = \left(\frac{d{\omega_0}^2}{\eta r^3 J_T}\right) \tag{70}$$

La ecuación (66) puede ser reescrita referenciada a la hélice. Este procedimiento será útil para enlazar con las ecuaciones aerodinámicas del cuadricóptero.

$$\dot{\omega_P} = -\frac{{K_M}^2}{RJ_{TP}}\eta r^2\omega_P - \frac{d}{J_{TP}}{\omega_P}^2 + \frac{K_M}{RJ_{TP}}\eta r v \tag{71}$$

Donde $J_{TP} = J_P + \eta r^2 J_M$ es la inercia total en la hélice.

Debido a que la ecuación diferencial (71) es no lineal, se linealiza la ecuación alrededor de su punto de trabajo. Utilizando el método de aproximación por series de Taylor de primer orden se deriva la ecuación (72) según [19].

$$\dot{\omega_P} = A_P \omega_P + B_P v + C_P \tag{72}$$

Donde las constantes se calculan como se indican en las ecuaciones (73) hasta (75).

$$A_P = -\left(\frac{{K_M}^2 \eta r^2}{R J_{TP}} + \frac{2 d \omega_0}{J_{TP}}\right) \tag{73}$$

$$B_P = \left(\frac{K_M \eta r}{R J_{TP}}\right) \tag{74}$$

$$C_P = \left(\frac{d {\omega_0}^2}{J_{TP}}\right) \tag{75}$$

Entonces, con sólo tres parámetros es posible determinar la dinámica de los 4 motores del sistema. La ecuación (76) muestra la ecuación diferencia en forma matricial:

$$\dot{\vec{\Omega}} = -A_P \vec{\Omega} + B_P \vec{\boldsymbol{v}} + C_P \tag{76}$$

Donde $\vec{\Omega}$ es el vector de velocidades en los rotores, $\dot{\vec{\Omega}}$ es el vector de aceleraciones de los rotores y $\vec{\boldsymbol{v}}$ es el vector de voltajes de entrada.

En la actualidad, en los UAV de rotores no se utilizan motores DC con caja reductora de cambios regulados por voltaje, en su lugar se emplean motores DC *brushless* manejados con ESC (*Electronic Speed Control*) y regulados por medio de señales PWM.

Capítulo 2
Estimación de parámetros

En esta sección se utiliza el enfoque experimental para determinar las constantes y completar el modelamiento del cuadricóptero. El cuadricóptero utilizado en esta tesis es el 3DR ArduCopter Quad-C, con el cual se cuenta en el laboratorio de Sistemas Automáticos de Control de la Universidad de Piura, ver Figura 6.

Figura 6 3DR ArduCopter Quad-C [27]

El motor utilizado en este cuadricóptero es del modelo 850Kv AC2830-358, el cual es un motor *brushless* DC de tres fases. Obsérvese que el motor *brushless* es diferente al estudiado en la sección 1.3 el cual fue un motor de corriente continua con escobillas. El motor *brushless* se encuentra acoplado a su manejador o *driver*. De esta manera, el sistema *speed control*-motor admite señales de entrada PWM de 50 Hz con un *duty cycle* entre 5% a 10% o de 500 Hz con *duty cycle* entre 45% a 100%. En la Figura 7 se muestran los elementos del sistema *speed control*-motor con los cuales se cuenta en este trabajo de tesis.

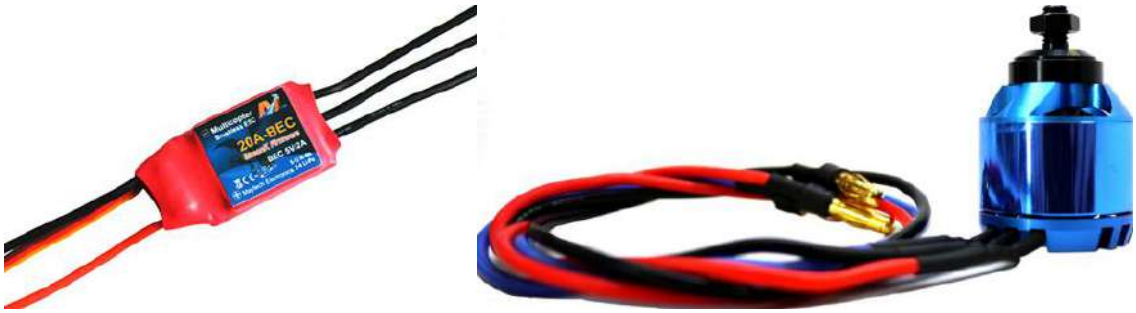

Figura 7 Sistema "*speed control*"-motor [28], [29].

El controlador de velocidad es gobernado por una tarjeta de control o sistema embebido del cuadricóptero. La señal de salida de la tarjeta de control es del tipo PWM, la cual se toma como señal de entrada del *driver* del motor. A su vez este último es alimentado por una tensión de 5 voltios y arroja en su salida una señal trifásica con tensión DC variable dependiente de la señal de entrada PWM.

Las hélices empleadas son 10X4.5 de la marca APC, ver Figura 8. Para poder determinar los parámetros de empuje b y de arrastre d, es necesario relacionar la velocidad angular con las fuerzas. Para evitar errores de medición de fuerza debido al peso, sujeción, vibración o movimiento del encoder, se realizan pruebas por separado.

Figura 8 Hélice LP 10045MR 10X4.5 MRP, marca APC [30].

A continuación se detalla cada una de las pruebas realizadas, se presentan los datos, las ecuaciones de aproximación y los resultados finales.

2.1 Prueba de empuje

El coeficiente de empuje b relaciona la velocidad angular al cuadrado de una hélice con la fuerza de empuje que produce. Para determinar el coeficiente de empuje de la ecuación (77) se ha construido un banco de pruebas en base a una estructura metálica e inspirada en los siguientes trabajos [31], [32], [33] y [34].

$$U_1 = \sum_{i=1}^{4} f_i = \sum_{i=1}^{4} b{\Omega_i}^2 \tag{77}$$

El banco de pruebas está conformado por un eje, rodamientos y un tubo cuadrado de aluminio donde se emperna el motor como se muestra en la Figura 9. El motor con su hélice se ubica de forma vertical en un extremo del tubo cuadrado de tal manera que ejerza una fuerza de empuje en dirección perpendicular al eje. En el otro extremo del tubo cuadrado se coloca una balanza para medir la fuerza.

El empuje producido por el rotor para determinado *duty cycle* ingresado al driver del motor, se aproxima a una recta y está dado por la ecuación (78).

$$E = a_e * PWM - b_e \tag{78}$$

E es el empuje en kilogramos, a_e y b_e son las constantes que definen la recta de aproximación.

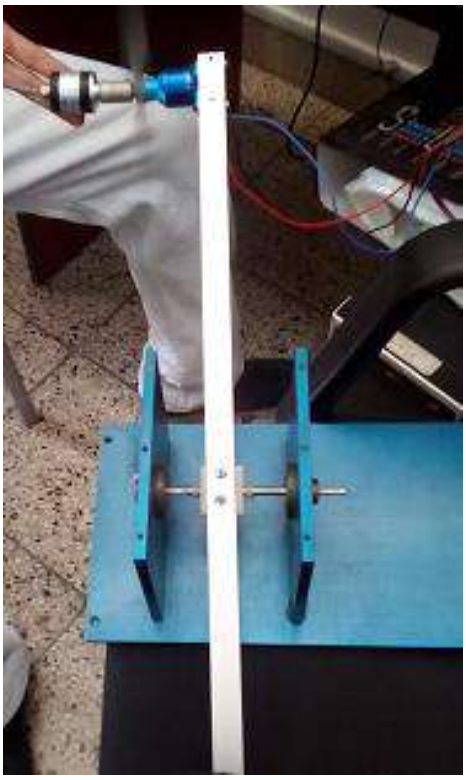

Figura 9 Banco de pruebas. Fuente: Elaboración propia.

En la Figura 10 se muestran los resultados de la prueba de empuje, donde la variable independiente es el *duty cycle* de la señal PWM y la variable dependiente es el peso marcado

por la balanza en kilogramos. En esta primera prueba se ha considerado una señal PWM de alimentación hacia el ESC con una frecuencia de 50 Hz.

El motor tiene una zona muerta entre 5% hasta 5.20% donde no se presenta movimiento en el rotor y por lo tanto tampoco se tiene empuje. En la zona entre 9.5% hasta 10% el motor se satura.

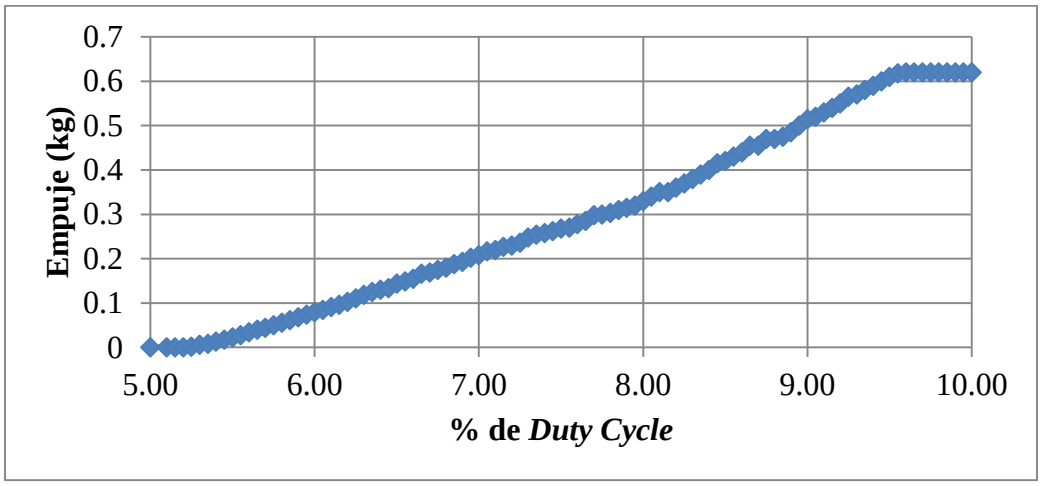

Figura 10 PWM vs Empuje Fuente: Elaboración propia.

Se realiza una linealización entre 5.20 hasta 9.5 y se obtiene la siguiente ecuación (79).

$$E = 0.1398 * PWM - 0.7623 \tag{79}$$

Donde E es el empuje expresado en kg y PWM es el valor de "*duty cycle*" entre 5.2 a 9.5.

2.2 Prueba de arrastre

Esta prueba ayuda a determinar el coeficiente de arrastre d de la ecuación (80).

$$\sum_{i=1}^{4} \tau_{M_i} = d({\Omega_2}^2 + {\Omega_4}^2 - {\Omega_1}^2 - {\Omega_3}^2) = U_4 \tag{80}$$

Se utiliza el mismo banco de pruebas que el mostrado en la sección 2.1 como se muestra en la Figura 9. El motor con su hélice se ubica de forma horizontal en un extremo del tubo cuadrado de tal manera que ejerza una fuerza de empuje en dirección paralela al eje y una fuerza de arrastre en dirección perpendicular al eje, mientras que en el otro extremo del tubo cuadrado se coloca una balanza.

El arrastre producido por el rotor para determinado *duty cycle* ingresado en el driver del motor ha sido calculado experimentalmente y se aproxima a una recta, ver ecuación (81).

$$A = a_a * PWM - b_a \tag{81}$$

A es el arrastre en kilogramos, a_a y b_a son las constantes que definen la recta de aproximación.

En la Figura 11 se muestran los resultados de la prueba de arrastre, donde la variable independiente es el *duty cycle* de la señal PWM y la variable dependiente es el peso marcado por la balanza en kilogramos.

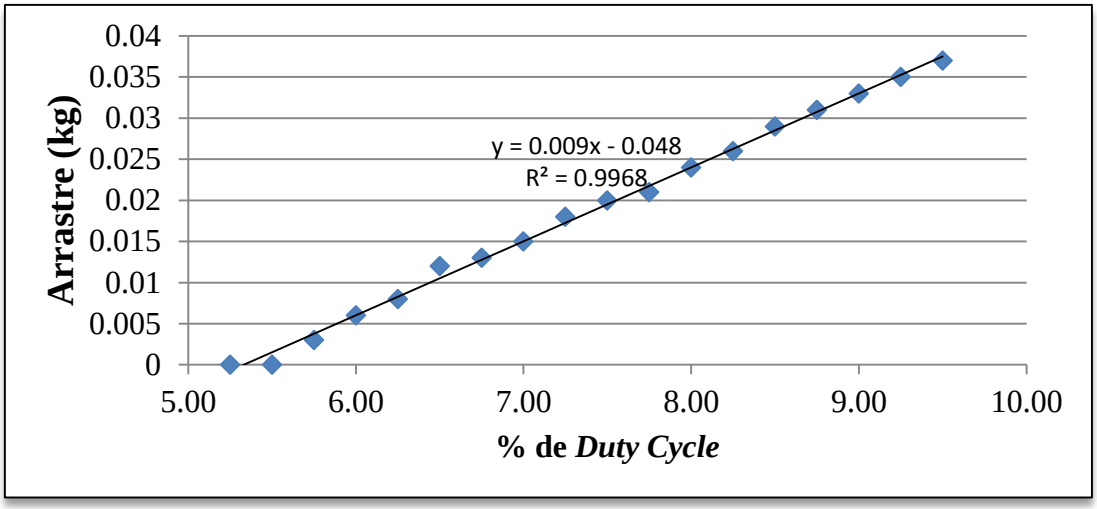

Figura 11 PWM vs Arrastre. Fuente: Elaboración propia.

El motor tiene una zona muerta entre 5% hasta 5.20% donde no se presenta movimiento y por lo tanto tampoco no se tiene arrastre. Se realiza una linealización entre 5.2% hasta 9.5% y se obtiene la ecuación (82).

$$A = 0.009 * PWM - 0.048 \tag{82}$$

Donde A es el peso de arrastre expresado en kg y PWM es el porcentaje de *duty cycle* el cual va entre 5.2% y 9.5%.

2.3 Prueba de velocidad

La prueba de velocidad se realiza con un *encoder* para determinar la relación lineal entre el porcentaje *duty cycle* de la PWM con la velocidad angular al cuadrado.

La prueba de velocidad determina la relación lineal entre la PWM con la velocidad angular al cuadrado. Los datos se aproximan a una recta.

$$\Omega^2 = a_\Omega * PWM - b_\Omega \tag{83}$$

Donde Ω^2 es la velocidad angular al cuadrado, a_Ω y b_Ω son las constantes que definen la recta de aproximación.

Los datos de la prueba de velocidad del rotor se grafican en la Figura 12, donde se observa que la velocidad angular cuadrática tiene una aproximación lineal respecto al porcentaje de *duty cycle*.

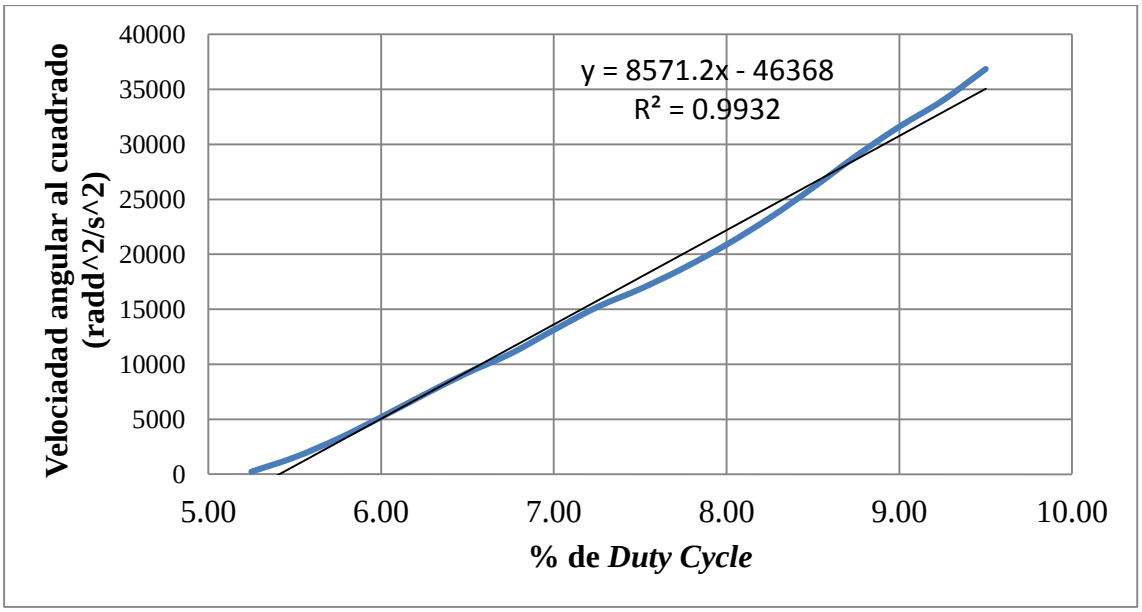

Figura 12 Prueba de velocidad. Fuente: Elaboración propia.

Se realiza una linealización entre 5.25 hasta 9.5 y se obtiene la siguiente ecuación:

$$\Omega^2 = 8571.2 * PWM - 46368 \tag{84}$$

Donde Ω^2 es la velocidad angular cuadrática $(rad/s)^2$ y PWM representa el valor de "*duty cycle*" comprendida entre 5.25 y 9.5.

2.4 Cálculo de coeficientes

Se despeja PWM a partir de la ecuación (83) y se reemplaza en (78), se multiplica por la gravedad para obtener unidades de Newton. En la ecuación (85) se muestra como se calcula b:

$$E = a_e * PWM - b_e$$

$$E * g = a_e * g * PWM - b_e * g$$

$$E * g = a_e * g * \left(\frac{\Omega^2 + b_\Omega}{a_\Omega}\right) - b_e * g$$

$$E * g = \frac{a_e * g * \Omega^2}{a_\Omega} + \frac{a_e * g * b_\Omega}{a_\Omega} - b_e * g$$

$$b = \frac{E * g}{\Omega^2} = \frac{a_e * g}{a_\Omega} + \frac{a_e * g * b_\Omega}{a_\Omega * \Omega^2} - \frac{b_e * g}{\Omega^2}$$

$$b = \frac{a_e * g}{a_\Omega} \tag{85}$$

Por lo tanto se obtiene un coeficiente de empuje $b = \frac{0.1398*9.81}{8571.2} = 16 * 10^{-5}\ [N.s^2]$.

De igual manera, se despeja PWM a partir de la ecuación (83) y se reemplaza en (81) y se multiplica por la gravedad para obtener unidades de Newton. En la ecuación (86) se muestra como se calculó d:

$$A = a_a * PWM - b_a$$

$$A * g = a_a * g * PWM - b_a * g$$

$$A * g = a_a * g * \left(\frac{\Omega^2 + b_\Omega}{a_\Omega}\right) - b_a * g$$

$$A * g = \frac{a_a * g * \Omega^2}{a_\Omega} + \frac{a_a * g * b_\Omega}{a_\Omega} - b_a * g$$

$$\frac{A * g}{\Omega^2} = \frac{a_a * g}{a_\Omega} + \frac{a_a * g * b_\Omega}{a_\Omega * \Omega^2} - \frac{b_a * g}{\Omega^2}$$

$$d = \frac{a_a * g * l}{a_\Omega} \tag{86}$$

Por lo tanto se obtiene un coeficiente de arrastre $d = \frac{0.9*9.81*0.2686}{8571.2} = 2.7668 * 10^{-6}\ [N.m.s^2]$ donde l es la longitud entre el centro de los motores al centro de gravedad en el plano xy.

2.5 Identificación de motor *brushless*

En muchos de trabajos como [14], [15], [20], [35] y [25] no consideran el polo mecánico del rotor. Sin embargo trabajos como [2], [8] y [9] sí toman en cuenta la dinámica del rotor. En este trabajo se demostrará la importancia de considerar dicho polo. Para evaluar el comportamiento de los algoritmos de control, serán simulados con un modelo matemático de los motores lo más real posible.

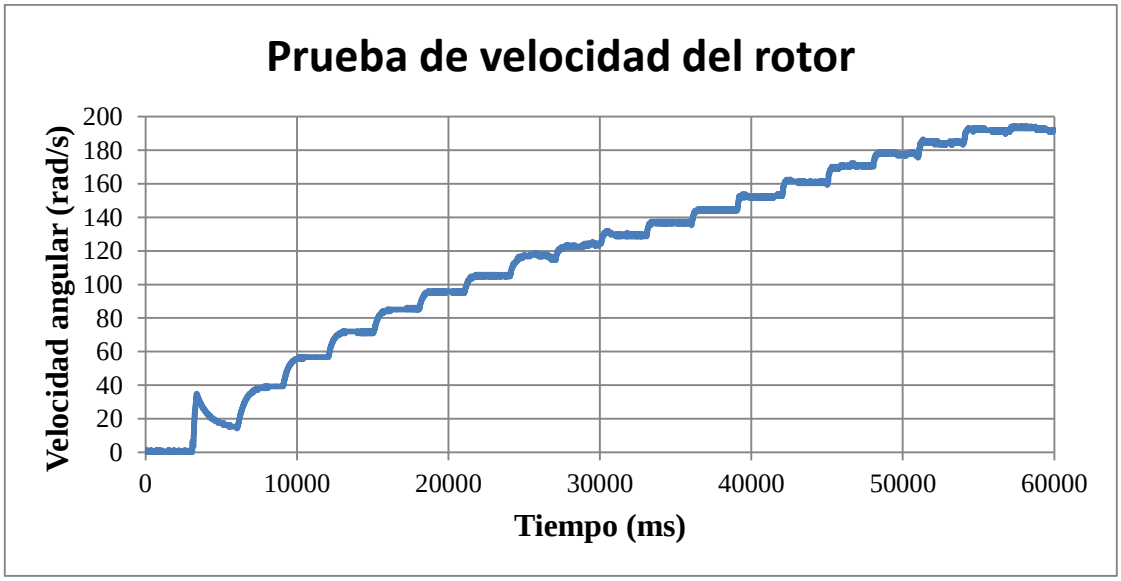

Figura 13 Velocidad angular del rotor frente a entradas escalón de 0.25% a 50 Hz. Fuente: Elaboración propia.

Se realiza un estudio de la respuesta dinámica del sistema "*speed control* – motor" para diferentes valores de PWM con saltos de 0.25% desde 5% hasta 10%. En la Figura 13 se muestra la gráfica de velocidad angular del motor en el tiempo. A partir de la Figura 13 se puede construir hasta 19 funciones transferencias.

La dinámica presentada en Figura 13 es claramente no lineal, debido a que no cumple la condición de proporcionalidad. Además se observa que el transitorio de cambio de PWM de 5% a 5.5% presenta un dinámica muy diferente a las siguientes. Este transitorio se debe a la corriente de arranque del motor, debido a su complejidad y por estar lejos de la zona de trabajo normal del cuadricóptero, no se tendrá en cuenta.

Para evitar la complejidad de introducir 19 funciones transferencias en la plataforma de Simulink, se decide reducir la cantidad de funciones hasta 10, promediando valores de constantes y ganancias. En la Tabla 2 se presentan 10 funciones transferencias que permiten linealizar por tramos la dinámica no lineal del motor.

Tabla 2 Aproximación lineal con 10 funciones transferencias para PWM a 50 Hz

Rango de *Duty Cycle*	**Ganancia estática K**	**Constante de tiempo Tao (ms)**
9.5 a 10	00.00	00.00
9 a 9.5	28.38	124.0
8.5 a 9	32.74	143.5
8 a 8.5	33.81	134.0
7.5 a 8	29.45	171.5
7 a 7.5	30.54	180.5
6.5 a 7	37.08	240.0
6 a 6.5	48.00	292.5
5.5 a 6	65.46	375.5
5 a 5.2	130.90	551.0

En la sección 3.3 se presentará un controlador mejorado el cual se desarrolla tomando en cuenta un sistema embebido comercial y un código descargable. Dicho sistema tiene como señal de salida una PWM la cual trabaja a aproximadamente 500 Hz. Trabajar a frecuencias más elevadas permite actualizaciones más rápidas en la señal manipulable de los rotores. Por lo expuesto, se debe calcular las funciones transferencias para rangos de trabajo de los motores cuando la señal PWM trabaja a 500 Hz.

En la Figura 14 se muestra la prueba realizada al mismo rotor de la Figura 13. Se observa que el rango de velocidades es similar, en ambos casos supera los 180 rad/s como límite máximo. Sin embargo en esta prueba se han utilizado otros rangos de operación de *duty cycle* de la señal PWM.

El rango del *duty cycle* se ha divido en 12 segmentos, las ganancia estática y constantes de tiempo de las funciones de primer orden aproximadas para cada rango se presentan en la Tabla 3.

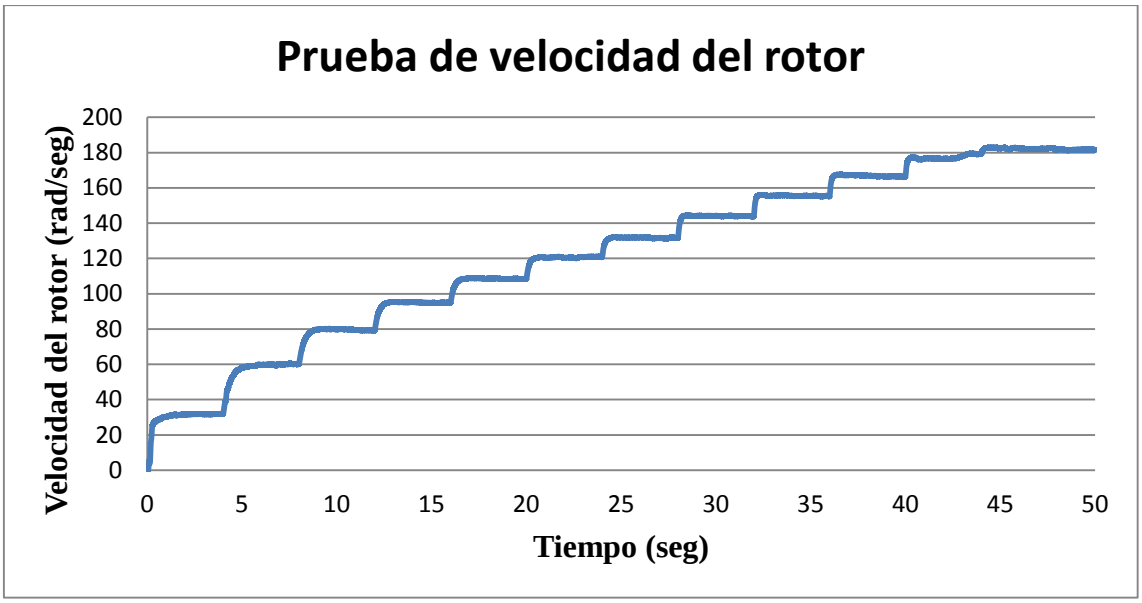

Figura 14 Prueba de velocidad del rotor con señal PWM a 500 Hz. Fuente: Elaboración propia.

Tabla 3 Aproximación lineal con 12 funciones transferencias para PWM a 500 Hz

Rango de *Duty Cycle*	Ganancia estática K	Constante de tiempo Tao (ms)
35.5 - 40.875	5.92	235
40.875 - 46.25	5.26	360
46.25 - 51.625	3.55	251
51.625 – 57	2.92	196
57 - 62.375	2.52	178
62.375 - 67.75	2.30	151
67.75 - 73.125	2.02	115
73.125 - 78.5	2.25	86
78.5 - 83.875	2.13	68
83.875 - 89.25	2.05	67
89.25 - 94.625	2.44	99
94.625 – 100	0.40	53

Al trabajar en un diferente rango de operación de *duty cycle*, también se hace necesario reescribir la ecuación (84). Recordemos que en la prueba de velocidad se determina la relación lineal entre el porcentaje *duty cycle* de la PWM con la velocidad angular al cuadrado del rotor. Los datos obtenidos en esta prueba se muestran en la Tabla 4.

Tabla 4 Prueba de velocidad del motor con PWM a 500 Hz

PWM	rad/s	rad^2/s^2
35.50	0.00	0.00
40.88	31.83	1013
46.25	60.12	3614
51.63	79.21	6274
57.00	94.93	9011
62.38	108.47	11766
67.75	120.83	14599
73.13	131.69	17343
78.50	143.80	20678
83.88	155.26	24105
89.25	166.27	27646
94.63	179.39	32179
100.00	181.52	32950

Los datos de presentados en la Tabla 4 se grafican en la Figura 15, donde se observa que la velocidad angular cuadrática nuevamente tiene una aproximación lineal.

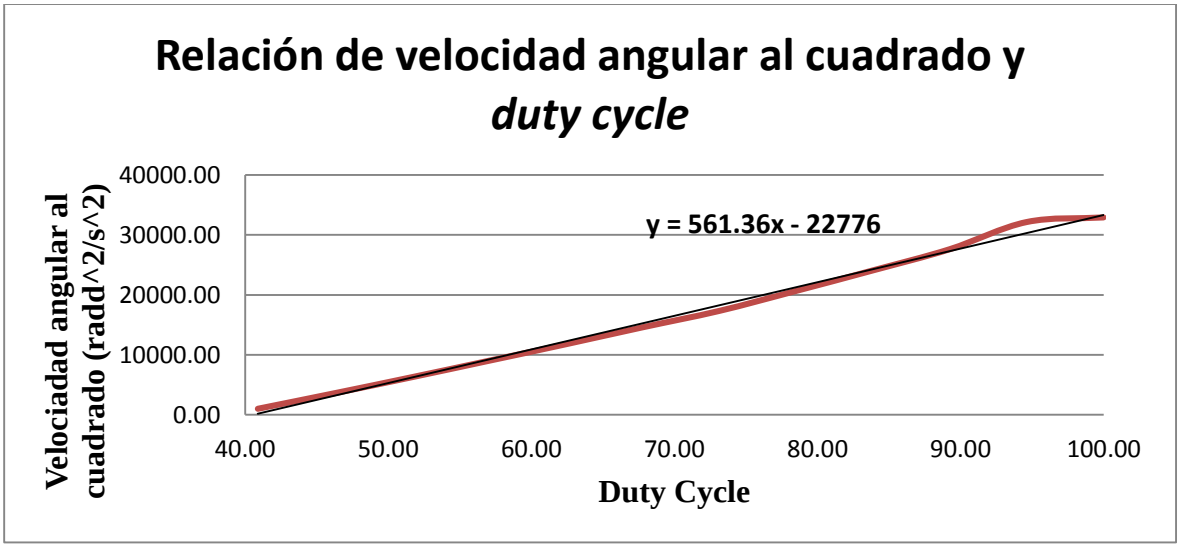

Figura 15 Relación velocidad angular al cuadrado y *duty cycle* con PWM a 500 Hz. Fuente: Elaboración propia.

Se linealiza para un *duty cycle* comprendido de 40 hasta 94.63 por ciento, ya que para valores de *duty cycle* cercanos a 100 el motor se satura, ver ecuación (87).

$$\Omega^2 = 565.53 * PWM - 23020 \tag{87}$$

Donde Ω^2 es la velocidad angular cuadrática $(rad/s)^2$ y PWM es el valor de "*duty cycle*" entre 40 a 100. La ecuación (87) será utilizada en sección 4.2 donde se implementa la estrategia de control.

Capítulo 3
Algoritmos de control

3.1 Verificación del simulador

En la actualidad existen diversos trabajos sobre cuadricópteros. Para verificar la correcta implementación de las ecuaciones vistas en el Capítulo 1 que describen el modelo no lineal del cuadricóptero, se decide construir en Simulink el modelo en base a los parámetros de [19] y observar las coherencias.

3.1.1 Modelo de control

La dinámica del cuadricóptero es muy bien descrita en el Capítulo 1. Los conceptos más importantes se pueden resumir en las ecuaciones (88), (89) y (90). El primer sistema de ecuaciones (88) muestra cómo el cuadricóptero acelera según los comandos básicos del movimiento.

$$\begin{cases} \ddot{Z} = -g + (\cos\theta cos\phi)\dfrac{U_1}{m} \\ \ddot{\phi} = \dfrac{lU_2}{I_{XX}} \\ \ddot{\theta} = \dfrac{lU_3}{I_{YY}} \\ \ddot{\psi} = \dfrac{U_4}{I_{ZZ}} \end{cases} \tag{88}$$

El segundo sistema de ecuaciones explica cómo los movimientos básicos están relacionados con la velocidad al cuadrado las hélices.

$$\begin{cases} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} = \begin{bmatrix} b({\Omega_1}^2 + {\Omega_2}^2 + {\Omega_3}^2 + {\Omega_4}^2) \\ b({\Omega_4}^2 - {\Omega_2}^2) \\ b({\Omega_3}^2 - {\Omega_1}^2) \\ d({\Omega_2}^2 + {\Omega_4}^2 - {\Omega_1}^2 - {\Omega_3}^2) \end{bmatrix} \\ \Omega = \Omega_1 + \Omega_2 + \Omega_3 + \Omega_4 \end{cases} \tag{89}$$

La tercera ecuación tiene en cuenta la dinámica de los motores y muestra la relación entre la velocidad del rotor y el voltaje en los motores.

$$\dot{\vec{\Omega}} = -\left(\frac{{K_M}^2 \eta r^2}{R J_{TP}} + \frac{2 d \omega_0}{J_{TP}}\right)\vec{\Omega} + \left(\frac{K_M \eta r}{R J_{TP}}\right)\vec{v} + \left(\frac{d {\omega_0}^2}{J_{TP}}\right) \tag{90}$$

El algoritmo de control recibe, como entradas, los datos de los sensores y de la consigna. Durante el cálculo se utiliza una gran cantidad de constantes y variables que describen la dinámica y los estados del cuadricóptero. La salida del algoritmo determina la señal PWM de los cuatro motores, ver Figura 16.

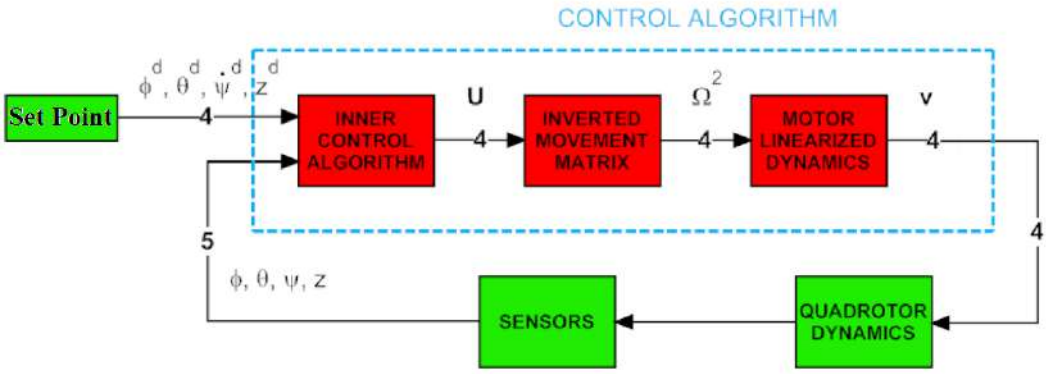

Figura 16 Diagrama de bloques del control [19]

El "Algoritmos de control interno" representa el núcleo de los algoritmos de control. Procesa la consigna y los datos de los sensores y proporciona una señal para cada movimientos básicos (U_i) que equilibra el error de posición. La ecuación (88) se utiliza en este bloque para transferir una consigna de aceleración a un movimiento básico. Las reglas de control utilizados para estimar las órdenes de aceleración son técnicas PID.

La Matriz de Movimientos Invertidos es el segundo bloque en la cadena de control. Se utiliza para calcular la velocidad al cuadrado de las hélices de los cuatro rotores. El cálculo se muestra en la ecuación (91).

A partir de la ecuación (89) se tiene:

$$\begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} = \begin{bmatrix} b & b & b & b \\ 0 & -b & 0 & b \\ -b & 0 & b & 0 \\ -d & d & -d & d \end{bmatrix} \begin{bmatrix} {\Omega_1}^2 \\ {\Omega_2}^2 \\ {\Omega_3}^2 \\ {\Omega_4}^2 \end{bmatrix}$$

Despejando las variables de interés:

$$\begin{bmatrix} {\Omega_1}^2 \\ {\Omega_2}^2 \\ {\Omega_3}^2 \\ {\Omega_4}^2 \end{bmatrix} = \begin{bmatrix} \frac{1}{4b} & 0 & \frac{-1}{2b} & \frac{-1}{4d} \\ \frac{1}{4b} & \frac{-1}{2b} & 0 & \frac{1}{4d} \\ \frac{1}{4b} & 0 & \frac{-1}{2b} & \frac{-1}{4d} \\ \frac{1}{4b} & \frac{-1}{2b} & 0 & \frac{1}{4d} \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix}$$

$$\begin{cases} {\Omega_1}^2 = \frac{1}{4b} U_1 - \frac{1}{2b} U_3 - \frac{1}{4d} U_4 \\ {\Omega_2}^2 = \frac{1}{4b} U_1 - \frac{1}{2b} U_2 + \frac{1}{4d} U_4 \\ {\Omega_3}^2 = \frac{1}{4b} U_1 + \frac{1}{2b} U_3 - \frac{1}{4d} U_4 \\ {\Omega_4}^2 = \frac{1}{4b} U_1 + \frac{1}{2b} U_2 + \frac{1}{4d} U_4 \end{cases} \tag{91}$$

El bloque Modelo Linealizado del Moto” es el tercero en la cadena de control. La dinámica del motor es una ecuación diferencial no lineal representada en la ecuación (66) o en su forma linealizada representada en la ecuación (90). Varios enfoques pueden ser seguidos en este bloque. El primero de ellos es simplemente resolver numéricamente la ecuación para obtener los voltajes adecuados. El segundo es para linealizar la ecuación para reducir el cálculo (en comparación con la opción anterior).

El tercer método es caracterizar experimentalmente el comportamiento motor y utilizar la relación deducida. Este tercer método ha sido adoptado por su sencillez y la certeza de la descripción del modelo. A pesar de que la relación Ω^2 a v muestra una función de transferencia de primer orden, se ha decidido caracterizar sólo con una ganancia sobre DC para reducir aún más la complejidad. Esta simplificación radical no es influente debido a que la fuerza de la actuación se encuentra en un bucle cerrado.

3.1.2 Control PD

En el área industrial de los reguladores más utilizados son sin duda el PID. En robótica, técnica PID representa los conceptos básicos de control. A pesar de que una gran cantidad de diferentes algoritmos proporcionan un mejor rendimiento que los PID, esta última estructura se elige a menudo por su estructura simple, presentar buenos rendimientos para diversos procesos y ajustable incluso sin tener un modelo del proceso.

Sin embargo la estructura tradicional PID presenta dos inconvenientes principales:

• La acción derivativa se calcula a partir del error. Si la consigna presenta un cambio escalón, la salida del derivador presentaría un impulso. Este movimiento brusco puede saturar los actuadores y alejar el sistema de la zona lineal. Por estas razones la arquitectura PID presenta la acción derivada sólo de la salida del proceso.

• La acción integral combinado con una saturación de actuador puede proporcionar un efecto no lineal, que puede disminuir el rendimiento del sistema de control. Cuando el valor de la integral es grande y el error cambia de signo, es necesario esperar mucho tiempo antes de que el sistema retome su comportamiento lineal (después de la "descarga" de la acción integral). Este fenómeno se llama *wind-up* integral. Para evitarlo, se debe añadir un saturador después de la integral para limitar su valor máximo y mínimo.

Por lo anteriormente explicado, el esquema de control podría representarse como se muestra en la Figura 17.

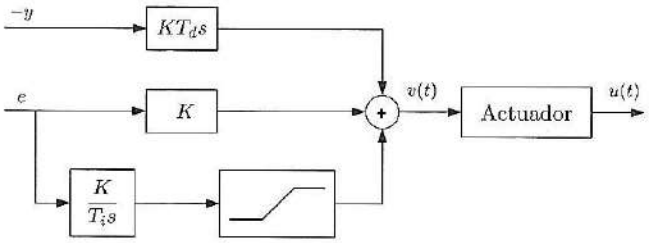

Figura 17 Estructura de control PID con limitación del término integral y parte derivativa sólo en la salida del proceso [36]

Pero dada la dinámica del cuadricóptero representada por la ecuación (88) se observa que las funciones contienen dos integradores. Esto implica que se puede diseñar un control proporcional derivativo (PD) para controlar ϕ, θ, ψ y Z mediante las fuerzas básicas U_i. El esquema final de control se muestra en la Figura 18, donde se observa que la parte derivativa se obtiene a partir de la salida del proceso y la parte proporcional a partir del error.

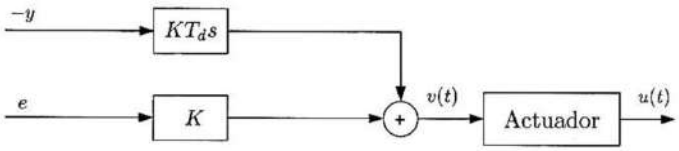

Figura 18 Estructura de control PD con parte derivativa sólo en la salida del proceso. Fuente: Elaboración Propia

Control de "*Roll*"

En la Figura 19 se muestra el esquema de control del ángulo ϕ. Se tiene como estradas la consigna del ángulo ϕ y el valor actual de ϕ. Se tiene como variable de manipulable U_2.

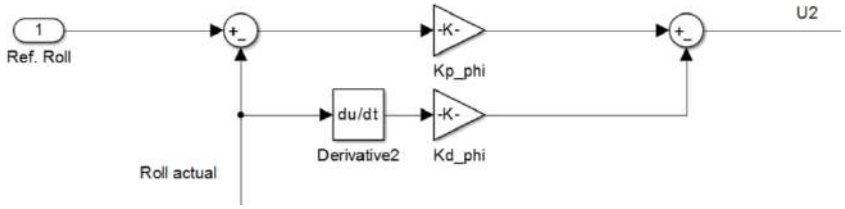

Figura 19 Control de Roll. Fuente: Elaboración propia.

Control de "*Pitch*"

La estructura de control del ángulo θ es muy similar a la anterior. En la Figura 20 se muestra el esquema de control del ángulo θ. Se tiene como estradas la consigna del ángulo θ y el valor actual de θ. Se tiene como variable de manipulable U_3.

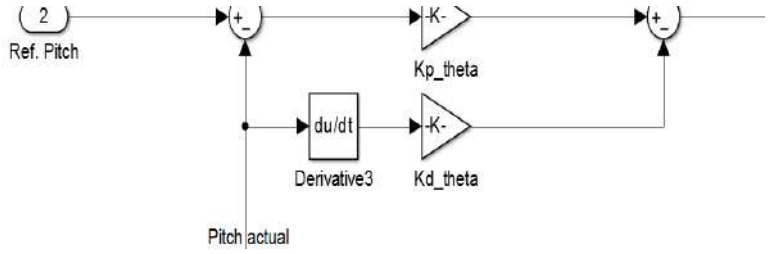

Figura 20 Control de Pitch. Fuente: Elaboración propia.

Control de "*Yaw*"

La estructura de control del ángulo ψ es muy similar a la anterior. En la Figura 21 se muestra el esquema de control del ángulo ψ. Se tiene como estradas la consigna del ángulo ψ y el valor actual de ψ. Se tiene como variable de manipulable U_4.

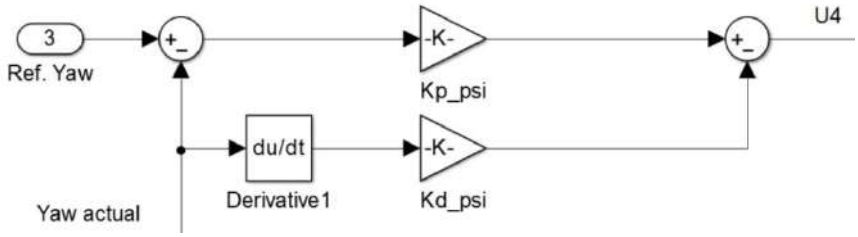

Figura 21 Control de Yaw. Fuente: Elaboración propia.

Control de "*Altura*"

La estructura de control de altura es más compleja que las anteriores y está basada en [15] donde se propone la siguiente ley de control:

$$U_1 = \frac{g + K_{p_z}\left(z_{ref} - z_{actual}\right) + K_{d_z}\left(\dot{z_{ref}} - \dot{z_{actual}}\right)}{cos\theta cos\phi} \tag{92}$$

Considerando la aplicación de la parte derivativa sólo a partir de la salida del proceso, la ecuación (92) se puede reescribir como se muestra en la ecuación (93).

$$U_1 = \frac{g + K_{p_z}\left(z_{ref} - z_{actual}\right) - K_{d_z}\dot{z}_{actual}}{cos\theta cos\phi} \tag{93}$$

En la Figura 22 se muestra el esquema de control de la altura z. Se tiene como estradas la consigna de la altura y el valor actual de z. Se tiene como variable de manipulable U_1.

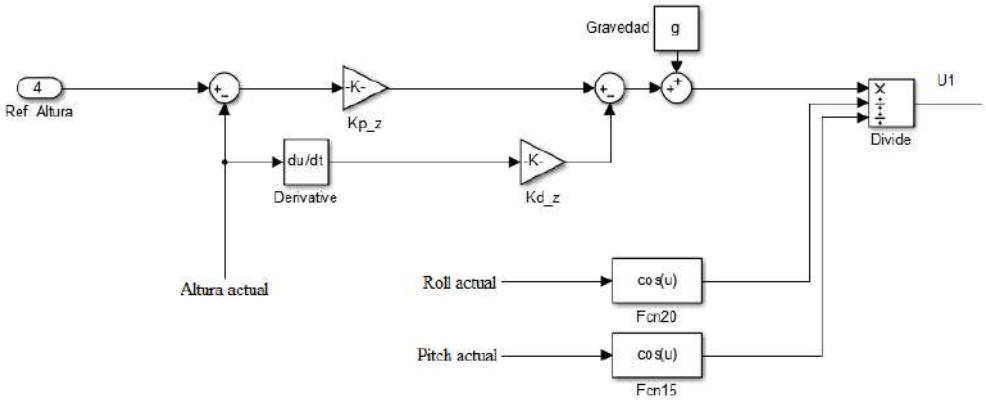

Figura 22 Control de Pitch. Fuente: Elaboración propia.

3.1.3 Simulación

En la Figura 23 se muestra el esquema completo para la simulación del control PD del cuadricóptero implementado en Simulink. El diagrama de bloques está constituido por 6 subsistemas llamados "*Joystick*", Algoritmo de control interno, Matriz de movimientos invertidos, Caracterización experimental del motor, Modelo no lineal del cuadricóptero y mundo virtual.

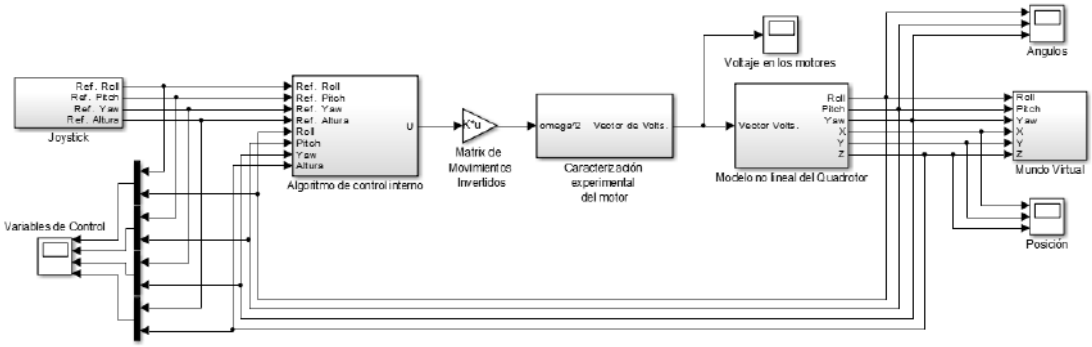

Figura 23 Diagrama de bloques del simulador implementado en Simulink. Fuente: Elaboración propia.

El primer subsistema es llamado *Joystick*, el cual trata de representar un control remoto complejo comunicado por radio frecuencia que se implementará en el proyecto. En la Figura 24 se muestra el diagrama de bloques del subsistema "*Joystick*". El bloque "*Joystick input*" realiza la comunicación de un mando conectado por puerto USB con Simulink. Debido a que las señales a la salida del mando presentan valores muy pequeños, éstos deben ser escalados para obtener referencias razonables.

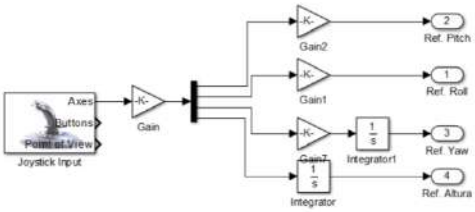

Figura 24 Diagrama de bloques del subsistema *Joystick*. Fuente: Elaboración propia.

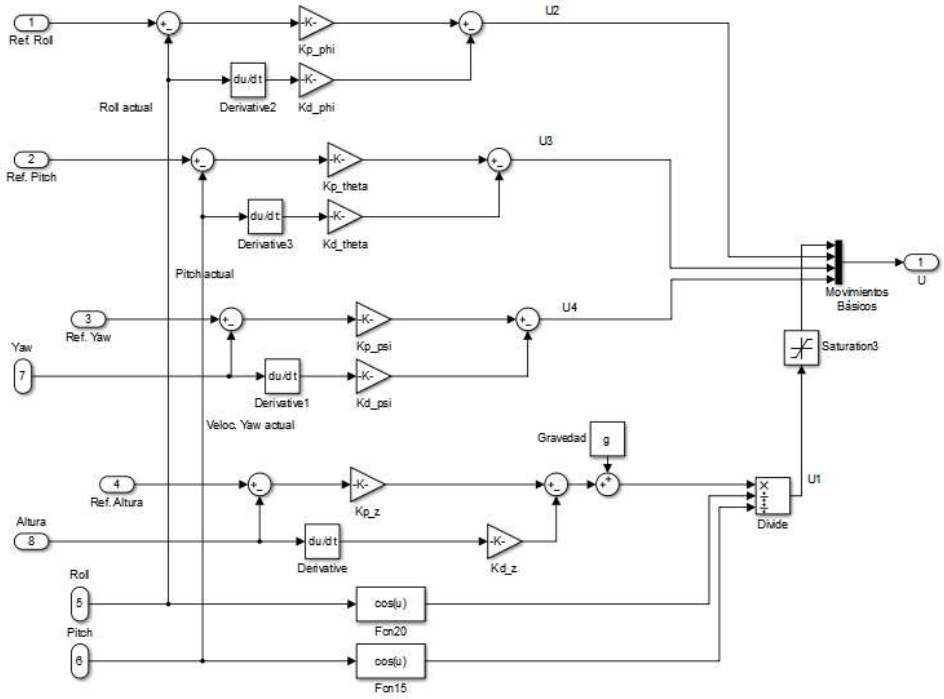

Figura 25 Diagrama de bloques del subsistema Algoritmo de control interno. Fuente: Elaboración propia.

El segundo subsistema es el llamado Algoritmo de control interno, el cual contiene los cuatro controladores PD explicados en la sección 3.2. En la Figura 25 se muestra el diagrama de bloques del subsistema Algoritmo de control interno.

El tercer bloque es llamado Matriz de movimientos invertidos. Dicho bloque es una ganancia matricial que permite realizar la equivalencia entre las fuerzas básicas U_i y las velocidades angulares cuadráticas de los rotores para tal propósito. Dicha matríz proviene de la ecuación (91), ver Figura 26.

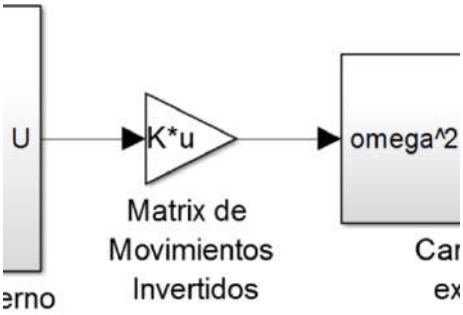

Figura 26 Matriz de Movimientos Invertidos. Fuente: Elaboración propia.

El siguiente bloque busca encontrar una relación entre las velocidades angulares establecidas por el bloque anterior y los voltajes necesarios en los motores para obtener dichas velocidades angulares. Experimentalmente se puede obtener una expresión lineal como se muestra en la ecuación (94).

$$\Omega^2 = C + D * v \tag{94}$$

Considerando la ecuación presentada en [19] se tiene que $\mathrm{C} = -14.7 * 10^3$ y $D = 6.4 * 10^3$. En la Figura 27 se muestra el diagrama de bloques que resuelve la ecuación (94) despejando v.

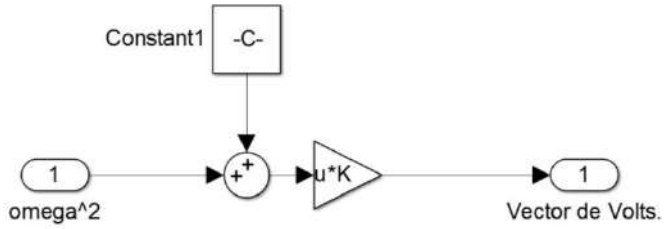

Figura 27 Caracterización experimental del motor. Fuente: Elaboración propia.

A la salida de este bloque ya se cuenta con los valores de voltaje para cada uno de los motores del cuadricóptero que deben ser impuestos. Hasta ahora el procedimiento ha consistido en determinar a partir de un sistema SISO con U_i como variables manipulables, las variables manipulables de un sistema MIMO con v como variables manipulables.

El penúltimo subsistema llamado Modelo no lineal del Cuadricóptero se constituye de otros 5 subsistemas que implementan el sistema de ecuaciones (27), (38) y (63). En la Figura 28 se muestra el diagrama de bloques del subsistema Modelo no lineal del Cuadricóptero con sus subsistemas: Modelo no lineal del Motor donde se introducen las ecuaciones no lineales del motor, Modelo aerodinámico de los rotores donde se implementan las ecuaciones que relacionan las fuerzas U_i con las velocidades angulares cuadráticas Ω_i, Modelo dinámico del Cuadricóptero donde se introducen las ecuaciones no lineales del cuadricóptero sin simplificaciones, Transformación que relaciona los ángulo de Euler con los ángulos del sistema fijo al cuadricóptero y finalmente el llamado Modelo de Posición que determina la posición que sigue el cuadricóptero.

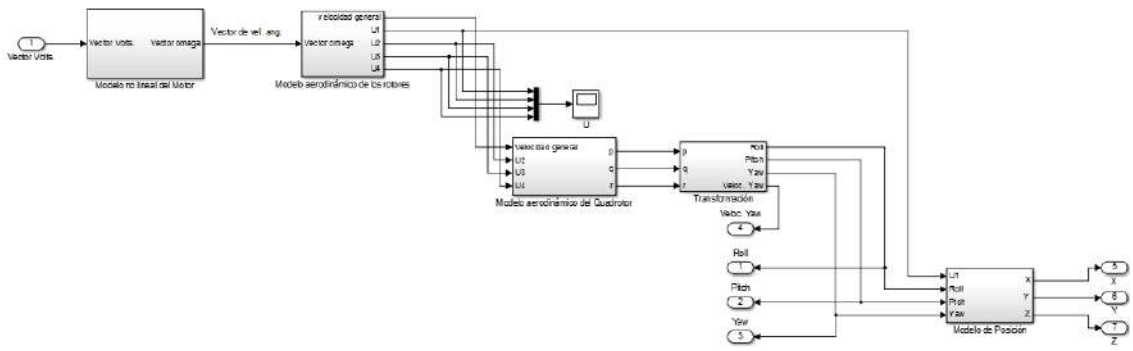

Figura 28 Diagrama de bloques del subsistema Modelo no lineal del Cuadricóptero. Fuente: Elaboración propia.

En la Tabla 5 se realiza un resumen de las constantes utilizadas para propósitos de simulación extraídas del trabajo [19]

Tabla 5 Valores de constantes para el modelo del cuadricóptero

Nombre	Símbolo	Valor
Constante del motor	K_M	$6.3 * 10^{-3}[\mathrm{NmA}^{-1}]$
Resistencia del motor	R	$0.6[ohms]$
Momento total de inercia respecto al eje del rotor	J_{TP}	$104 * 10^{-6}[\mathrm{Nms}^2]$
Rendimiento de la caja	n	0.9
Relación de reducción de la caja	r	5.6
Factor de arrastre	d	$1.1 * 10^{-6}$
Masa del Cuadricóptero	m	$1[kg]$
Aceleración de la gravedad	g	$9.81[\mathrm{ms}^2]$
Momento de inercia del cuerpo respecto al eje x	I_{XX}	$8.1 * 10^{-3}[\mathrm{Nms}^2]$
Momento de inercia del cuerpo respecto al eje y	I_{YY}	$8.1 * 10^{-3}[\mathrm{Nms}^2]$
Momento de inercia del cuerpo respecto al eje z	I_{ZZ}	$14.2 * 10^{-3}[\mathrm{Nms}^2]$
Distancia entre el centro del Cuadricóptero y el centro de los rotores	l	$0.24[\mathrm{m}]$
Factor de empuje	b	$54.2 * 10^{-6}$

El último subsistema es el llamado Mundo Virtual. En este subsistema se ha utilizado un mundo virtual incorporado de las librerías de Matlab con el nombre vrtkoff_hud.wrl. Además de realizarle adiciones, se ha cambiado el objeto virtual de vuelo por un cuadricóptero. En la Figura 29 se muestra el diagrama de bloques construido para este propósito.

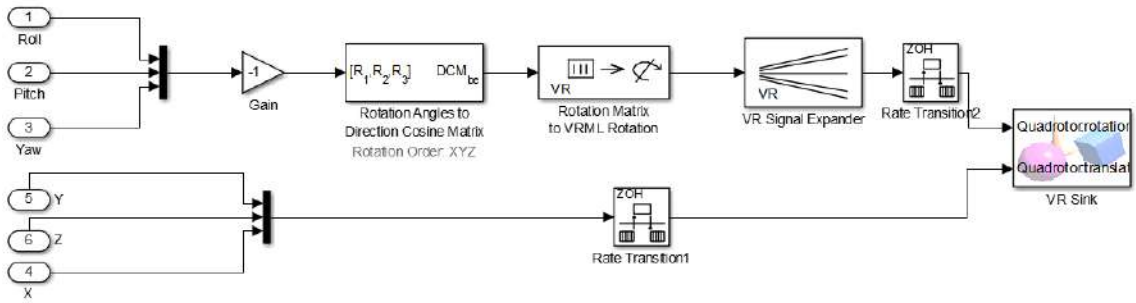

Figura 29 Diagrama de bloques del subsistema Mundo virtual. Fuente: Elaboración propia.

Los parámetros de los controladores PD se resumen en la Tabla 6.

Tabla 6 Parámetros de sintonización del controlador PD

Parámetro	Valor
K_{p_ϕ}	40
K_{d_ϕ}	6
K_{p_θ}	40
K_{d_θ}	6
K_{p_ψ}	5
K_{d_ψ}	1.5
K_{p_z}	64
K_{d_z}	16

Se realiza una prueba de vuelo y a continuación se muestran los resultados. En la Figura 30 se muestra el punto de partida del cuadricóptero en el mundo virtual.

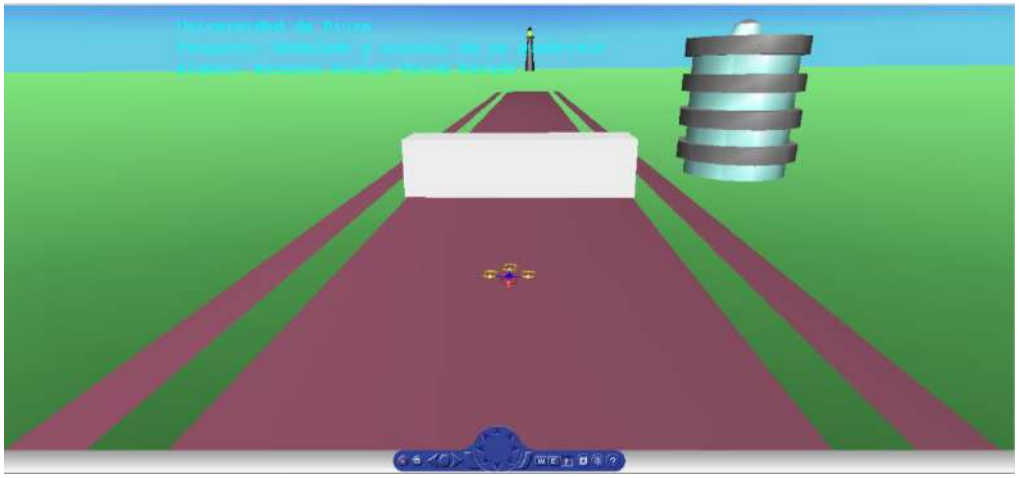

Figura 30 Inicio de simulación. Fuente: Elaboración propia.

En la Figura 31 se muestra la comparación de las variables de control con sus respectivas referencias las cuales van cambiando según el vuelo realizado. Como se observa en la Figura 31 todas las variables de control tienen un tiempo de establecimiento menor al medio segundo. Además nótese que la altura del cuadricóptero no se ve afectada frente a cambios en las otras variables de control.

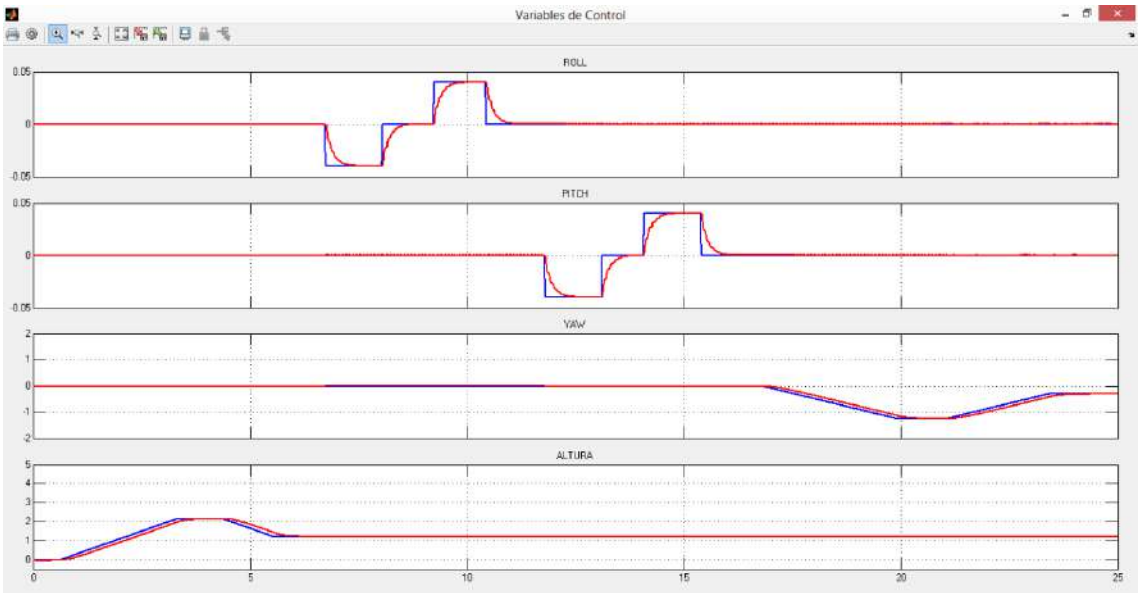

Figura 31 Variables de Control. Fuente: Elaboración propia.

En la Figura 32 se muestran las variables de manipulables las cuales van cambiando según el vuelo realizado. Se debe tener en cuenta que el valor máximo del voltaje de la fuente de alimentación es de 12 voltios. Como se observa en la Figura 32 los voltajes llegan a sus restricciones cuando se ha solicitado un cambio en la velocidad del "*yaw*", esto a causa que el cambio en el ángulo ψ se produce como consecuencia de diferencia de velocidades en los rotores.

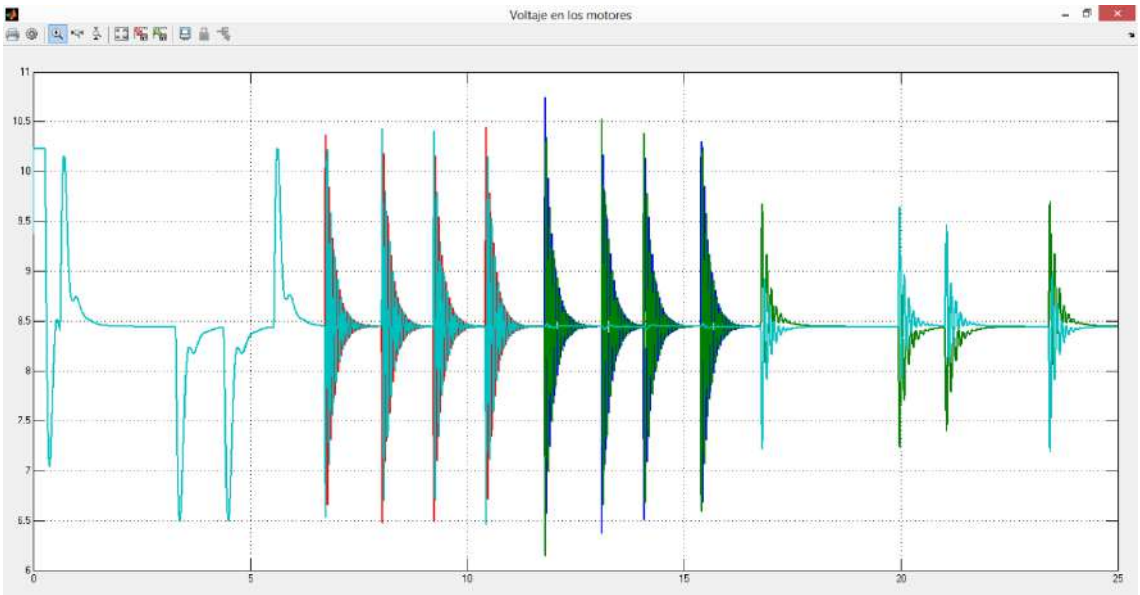

Figura 32 Variables manipulables. Fuente: Elaboración propia.

En la Figura 33 se muestran las fuerzas U_i las cuales van cambiando según el vuelo realizado. Se debe tener en cuenta que para valores pequeños de ϕ, θ y ψ las fuerzas U_2, U_3 y U_4 deben ser cercanas a cero.

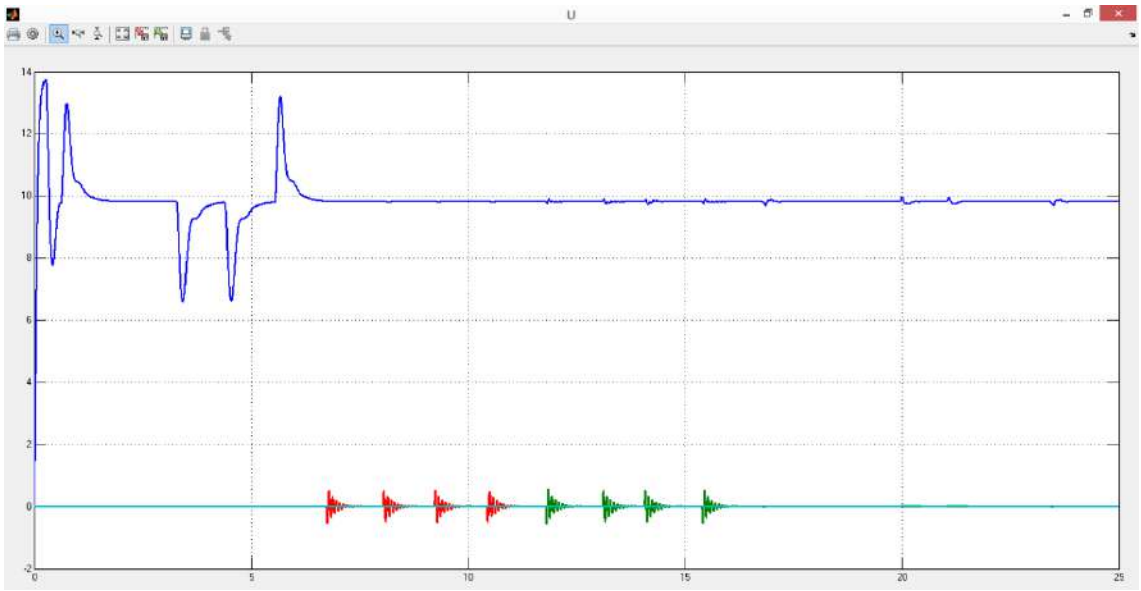

Figura 33 Fuerzas U_i sobre el cuadricóptero. Fuente: Elaboración propia.

Finalmente en la Figura 34 se muestra los cambios en las coordenadas de posición según la prueba de vuelo. Nótese que el movimiento en el eje X y Y puede ser indefinido hasta no forzar movimiento en el sentido contrario. Esto debido a que no se considera la resistencia del aire y se mantiene el movimiento debido a la inercia del cuerpo.

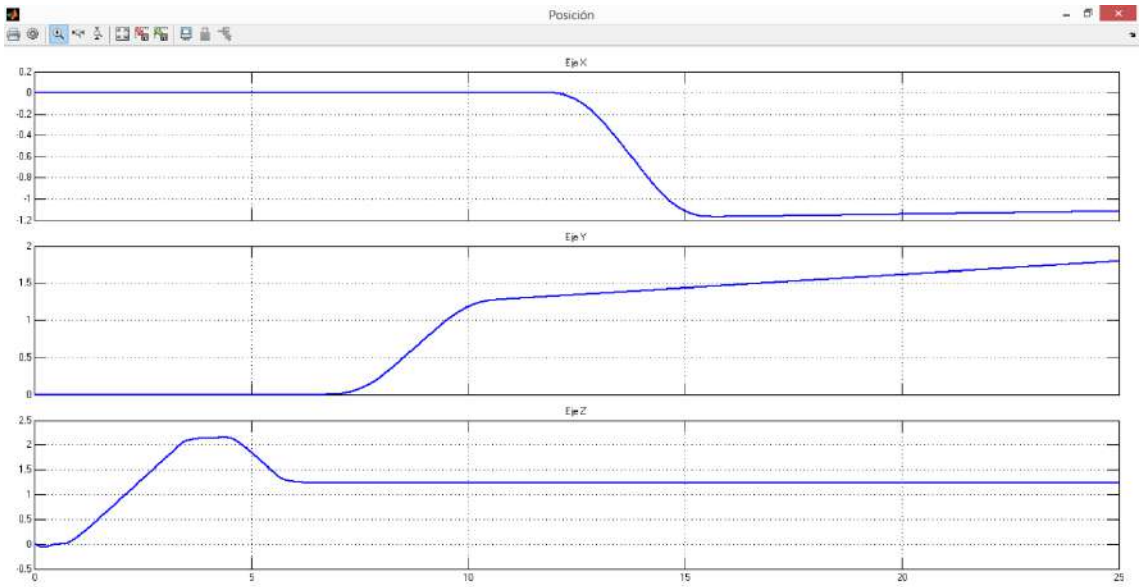

Figura 34 Coordenadas de Posición. Fuente: Elaboración propia.

3.2 Desarrollo de controladores para el ArduCopter Quad-C

En la sección 3.1 se demostró el correcto funcionamiento del modelo matemático implementado en Simulink. En esta sección se trabajará con el modelo adquirido por el laboratorio. Para tal propósito se realizan modificaciones al modelo implementado en la sección 3.1. Las principales modificaciones corresponden a los parámetros de empuje, arrastre, inercias, masa y en el sistemas controlador de velocidad – motor.

En la Tabla 7 se realiza un resumen de las constantes utilizadas para este proyecto, correspondientes al ArduCopter Quad-C. Los parámetros inerciales han sido estimados con la metodología de [37], los otros parámetros corresponden a las características técnicas del módulo experimental que se tiene en el laboratorio de Sistemas Automáticos de Control.

Tabla 7 Valores de constantes para el modelo ArduCopter Quad-C

Nombre	Símbolo	Valor
Factor de arrastre	d	$2.767 * 10^{-6} N.m.s^2$
Masa del Cuadricóptero	m	$0.763kg$
Aceleración de la gravedad	g	9.81ms^2
Momento de inercia al eje x para configuración en cruz	I_{XX}	$8.6574 * 10^{-3}\text{Nms}^2$
Momento de inercia al eje y para configuración en cruz	I_{YY}	$8.6574 * 10^{-3}\text{Nms}^2$
Momento de inercia al eje z para configuración en cruz	I_{ZZ}	$16.0584 * 10^{-3}\text{Nms}^2$
Momento de inercia en x para configuración en equis	I_{XX}	$8.4651 * 10^{-3}\text{Nms}^2$
Momento de inercia en y para configuración en equis	I_{YY}	$8.8497 * 10^{-3}\text{Nms}^2$
Momento de inercia en z para configuración en equis	I_{ZZ}	$16.0584 * 10^{-3}\text{Nms}^2$
Distancia entre centros del Cuadricóptero y los motores	l	0.267m
Factor de empuje	b	$160 * 10^{-6} N.s^2$

Como se mostró en la Tabla 2, el sistema controlador de velocidad – motor es expresado como una estructura de 10 funciones transferencias. En la Figura 35 se muestra el diagrama de bloques de su implementación en Simulink.

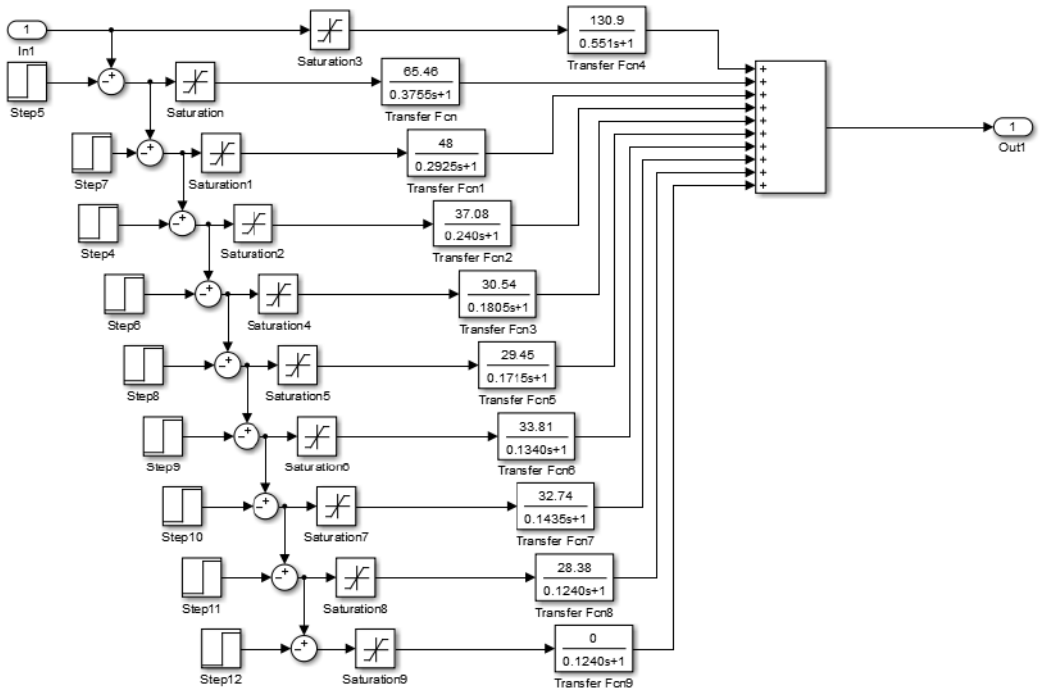

Figura 35 Modelo del motor linealizado por tramos. Fuente: Elaboración propia.

3.2.1 Control de Orientación

Como se observó en la ecuación (88), el sistema de orientación es doblemente integral e inestable con dos polos en cero. Por tal motivo, no será necesario implementar un control con término integral, sino sólo el término proporcional y el término derivativo para garantizar estabilidad.

En la ecuación (95) se muestra el sistema de orientación:

$$\begin{aligned} \ddot{\phi} &= \frac{lU_2}{I_{XX}} \\ \ddot{\theta} &= \frac{lU_3}{I_{YY}} \\ \ddot{\psi} &= \frac{U_4}{I_{ZZ}} \end{aligned} \tag{95}$$

Dado que U_2 y U_3 son las fuerzas que producen el movimiento de "*Roll*" y "*Pitch*" respectivamente, además que U_4 es el momento torsor que produce el movimiento de "*Yaw*". Teniendo en cuenta las estructuras de control de la Figura 19, Figura 20 y Figura 21, las variables de salida del control U_{Control2}, U_{Control3} y U_{Control4} son dinámicamente diferentes a las fuerzas y momentos sobre el cuadricóptero representados por U_2, U_3 y U_4. El motivo de la diferencia es provocado por la dinámica de los motores. Por tal motivo, se incluye la dinámica de los motores como se muestra en la ecuación (96).

$$\begin{aligned} \ddot{\phi} &= \frac{lU_{\text{Control2}}}{I_{XX}s^2(\tau_m s + 1)} \\ \ddot{\theta} &= \frac{lU_{\text{Control3}}}{I_{YY}s^2(\tau_m s + 1)} \\ \ddot{\psi} &= \frac{U_{\text{Control4}}}{I_{ZZ}s^2(\tau_m s + 1)} \end{aligned} \tag{96}$$

Donde τ_m es el tiempo característico del motor. Dado que el motor es un proceso no lineal, se debe considerar características del motor cerca del punto de trabajo. Esta teoría se extiende para el proceso de altura. En la ecuación (97) se muestra la función transferencia que gobierna la altura del cuadricóptero.

$$Z = \frac{U_{\text{Control1}}}{ms^2(\tau_m s + 1)} \tag{97}$$

Se analiza la estabilidad del sistema a lazo cerrado considerando un control PD con parte derivativa de la variable de salida. En la ecuación (98) se presenta la función transferencia del sistema a lazo cerrado.

$$\frac{Y}{R} = \frac{KP}{1 + (K + Ds)P} \tag{98}$$

Donde P representa el proceso a lazo abierto, la cual puede ser reemplazada por las ecuaciones (97) o (98). Los términos K y D son las ganancias proporcionales y derivativas respectivamente del control PD.

Los polos del sistema a lazo cerrado son:

$$1 + (K + Ds)P = 0$$

Considerando el caso del "*Roll*":

$$s^3 - p_m s^2 - \frac{Dlp_m}{I_{XX}} s - \frac{Klp_m}{I_{XX}} = 0 \tag{99}$$

Donde p_m es el polo del motor, $p_m = -1/\tau_m$.

Se sintoniza el control PD por medio de la técnica de asignación de polo. Considerando p_1, p_2 y p_3 los polos del sistema a lazo cerrado. El polinomio característico es:

$$s^3 - \underbrace{(p_1 + p_2 + p_3)}_{E} s^2 + \underbrace{(p_2p_3 + p_1p_3 + p_1p_2)}_{F} s - \underbrace{(p_1p_2p_3)}_{G} = 0 \tag{100}$$

Igualando (99) y (100) se tiene que:

$$p_m = p_1 + p_2 + p_3 \tag{101}$$

$$K = \frac{GI_{XX}\tau_m}{l} \tag{102}$$

$$D = \frac{FI_{XX}\tau_m}{l} \tag{103}$$

El procedimiento se extendiendo para el caso del *pitch*, *yaw* y altura. En la Tabla 8 se presenta un resumen de la regla para sintonizar por asignación de polos un controlador PD para un cuadricóptero.

Tabla 8 Resumen de regla para calcular el controlador PD por asignación de polos

Variables	Ganancia proporcional *K*	Ganancia derivativa *D*
Roll	$K_{Roll} = \frac{GI_{XX}\tau_m}{l}$	$D_{Roll} = \frac{FI_{XX}\tau_m}{l}$
Pitch	$K_{Pitch} = \frac{GI_{YY}\tau_m}{l}$	$D_{Pitch} = \frac{FI_{YY}\tau_m}{l}$
Yaw	$K_{Yaw} = GI_{ZZ}\tau_m$	$D_{Yaw} = FI_{ZZ}\tau_m$
Altura	$K_Z = Gm\tau_m$	$D_Z = Fm\tau_m$

Donde $F = p_2p_3 + p_1p_3 + p_1p_2$ y $G = p_1p_2p_3$.

Se buscan los polos que presenten el menor tiempo de establecimiento. Los polos seleccionados son: $p_1 = -1.2 + j3.2$, $p_2 = -1.2 - j3.2$, $p_3 = -1.7667$. Se obtienen los siguientes parámetros para la configuración en cruz y en equis como se muestra en la Tabla 9.

Tabla 9 Parámetros PD para configuración en cruz y en equis

Variable	Proporcional *K*	Derivativa *D*	Configuración
Roll	$K_{Roll} = 0.1596$	$D_{Roll} = 0.1231$	Cruz
Pitch	$K_{Pitch} = 0.1596$	$D_{Pitch} = 0.1231$	Cruz
Yaw	$K_{Yaw} = 0.0795$	$D_{Yaw} = 0.0614$	Cruz
Altura	$K_Z = 3.7812$	$D_Z = 2.9173$	Cruz
Roll	$K_{Roll} = 0.1561$	$D_{Roll} = 0.1204$	Equis
Pitch	$K_{Pitch} = 0.1632$	$D_{Pitch} = 0.1259$	Equis
Yaw	$K_{Yaw} = 0.0795$	$D_{Yaw} = 0.0614$	Equis
Altura	$K_Z = 3.7812$	$D_Z = 2.9173$	Equis

En la Figura 36 se muestran las variables de control para el caso en que el cuadricóptero tiene configuración en cruz. Como se observa las variables de orientación son perfectamente controladas y tienen un tiempo de establecimiento de 1.23 segundos aproximadamente. La altura presenta un "*offset*" debido a la linealización para estimar las señales PWM necesarias para el empuje de control.

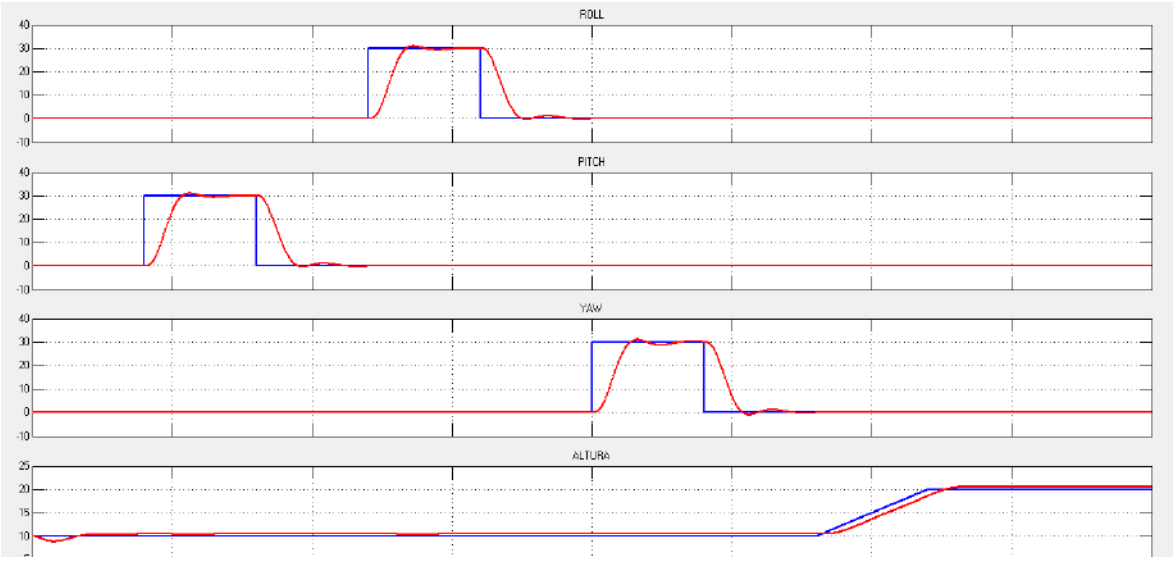

Figura 36 Variables de control para configuración en cruz. Fuente: Elaboración propia.

En la Figura 37 se muestran las señales PWM aplicadas a los motores. Como se observa, las señales se encuentran dentro de los límites de trabajo.

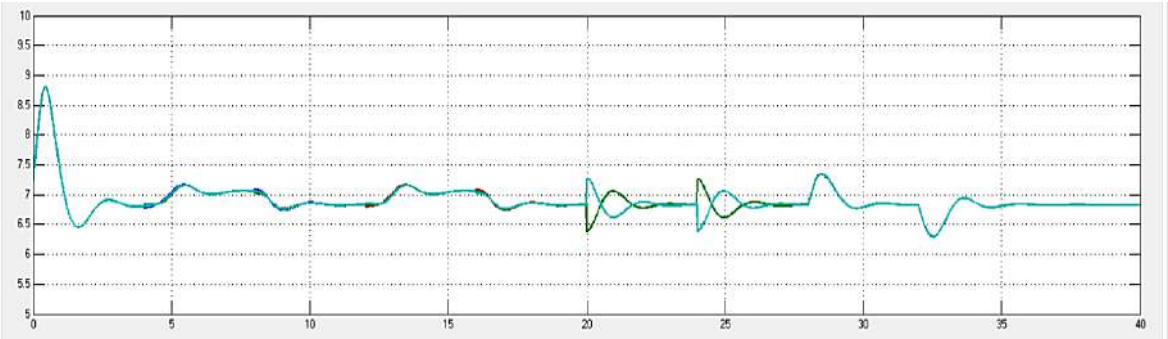

Figura 37 PWM para configuración en cruz. Fuente: Elaboración propia.

En la Figura 38 se muestran las variables de control para el caso en que el cuadricóptero tiene configuración en equis. Como se observa las variables de orientación son perfectamente controladas y tienen un tiempo de establecimiento de 1.23 segundos aproximadamente igual al caso en cruz. De igual manera, la altura presenta un "offset".

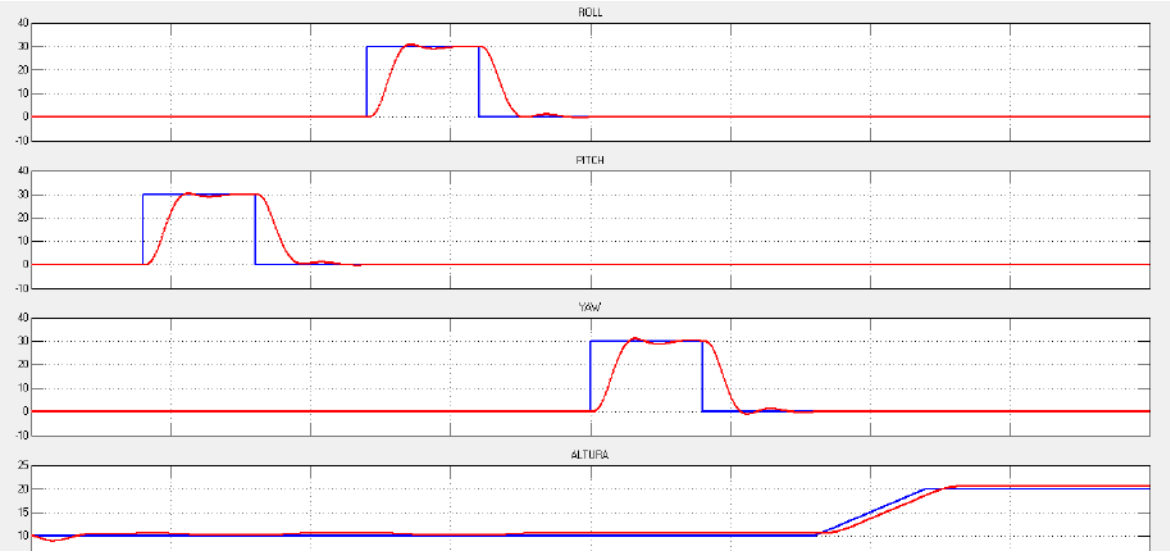

Figura 38 Variable de control para configuración en equis. Fuente: Elaboración propia.

En la Figura 39 se muestran las señales PWM aplicadas a los motores. Como se observa, las señales se encuentran dentro de los límites de trabajo.

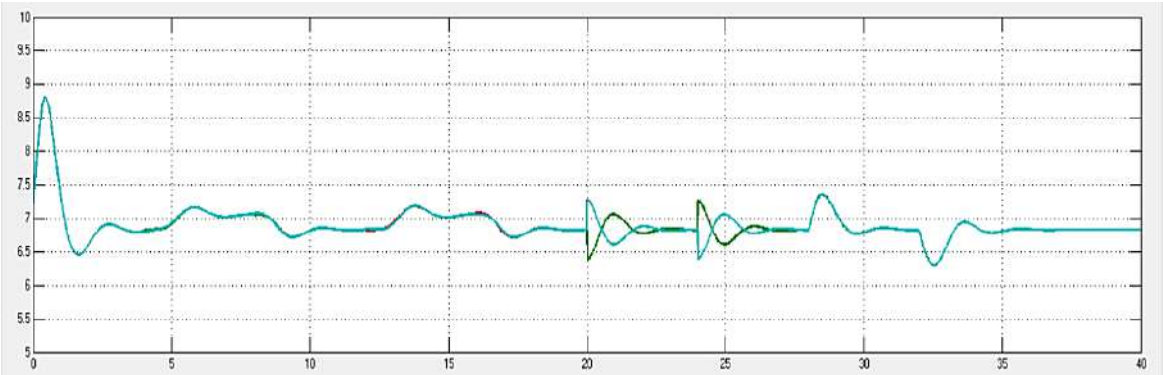

Figura 39 PWM para configuración en equis. Fuente: Elaboración propia.

Aparentemente los parámetros del control PD presentados en la Tabla 9 darán solución a nuestro problema de control. Pero para este caso no se ha tenido en cuenta la posible diferencia métrica entre el centro geométrico (CG) y el centro másico (CM) del cuadricóptero. La diferencia entre los CG y CM y otros efectos despreciados son englobados en los pares A_p, A_q y A_r. En la ecuación (104) se presenta el modelo considerando estas últimas variables [20].

$$\dot{p} = \frac{I_{YY} - I_{ZZ}}{I_{XX}} qr - \frac{J_{TP}}{I_{XX}} q\Omega + \frac{lU_2}{I_{XX}} + \frac{A_\phi}{I_{XX}}$$

$$\dot{q} = \frac{I_{ZZ} - I_{XX}}{I_{YY}} pr + \frac{J_{TP}}{I_{YY}} p\Omega + \frac{lU_3}{I_{YY}} + \frac{A_\theta}{I_{XX}} \tag{104}$$

$$\dot{r} = \frac{I_{XX} - I_{YY}}{I_{ZZ}} pq + \frac{U_4}{I_{ZZ}}$$

Considerando una diferencia entre CG y CM de apenas 1 milímetro ($A_p = 0.015$, $A_q = 0.015$, $A_r = 0$), se obtienen grandes perturbaciones en los ángulos de control. En la Figura 40 se observa un "*offset*" de alrededor de 20 grados. Además de oscilaciones pronunciadas de 10 grados de amplitud aproximadamente.

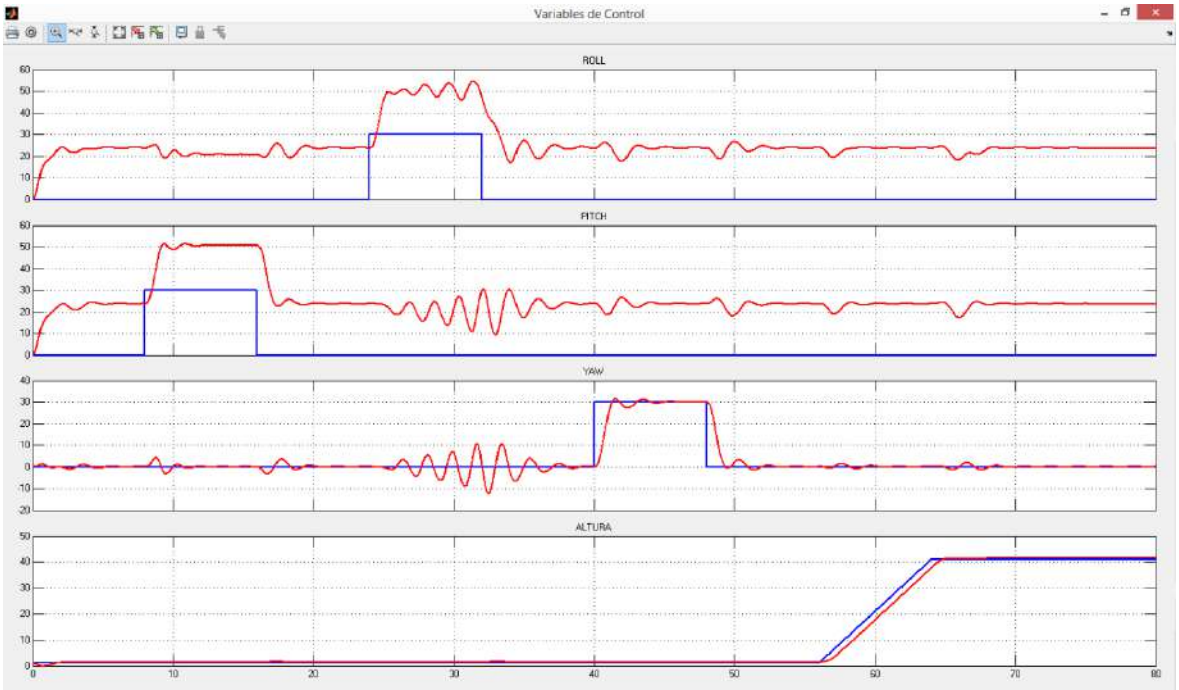

Figura 40 Variable de control frente a $A_p = 0.015\ N.m$, $A_q = 0.015\ N.m$, $A_r = 0\ N.m$. Fuente: Elaboración propia.

La Figura 40 deja en claro la necesidad de utilizar una ganancia integral de tal manera de sobreponerse a disturbios o perturbaciones en el sistema de orientación. El control PD funciona muy bien bajo la consideración que no existe disturbios en el sistema de orientación. Pero con un enfoque hacia un sistema real, el control PD es totalmente ineficaz.

Para realizar una sintonización de los parámetros PID se utiliza el sistema de orientación simplificados en las ecuaciones (96) y (97). El modelo del controlador se presenta en la ecuación (105).

$$\frac{Y}{R} = \frac{(K + I/s)P}{1 + (K + Ds + I/s)P} \tag{105}$$

El análisis se basa en estudiar el comportamiento de la dinámica de cada uno de los procesos a lazo cerrado. Los sistemas a lazo cerrado son los siguientes:

$$P_{\phi_{CL}} = \frac{\phi}{\phi_d} = \frac{K_{Roll} * l * s + I_{Roll} * l}{(I_{XX} * \tau_m)s^4 + I_{XX}s + (D_{Roll} * l)s^2 + (K_{Roll} * l)s + I_{Roll} * l} \quad (106)$$

$$P_{\theta_{CL}} = \frac{\theta}{\theta_d} = \frac{K_{Pitch} * l * s + I_{Pitch} * l}{(I_{YY} * \tau_m)s^4 + I_{YY}s + (D_{Pitch} * l)s^2 + (K_{Pitch} * l)s + I_{Pitch} * l} \quad (107)$$

$$P_{\psi_{CL}} = \frac{\psi}{\psi_d} = \frac{K_{Yaw} * s + I_{Yaw}}{(I_{ZZ} * \tau_m)s^4 + I_{ZZ}s + D_{Yaw}s^2 + K_{Yaw}s + I_{Yaw}} \quad (108)$$

$$\frac{Z}{Z_d} = \frac{K_Z * s + I_Z}{(m * \tau_m)s^4 + ms + D_Z s^2 + K_Z s + I_Z} \quad (109)$$

Con ayuda de un lazo iterativo que determine los parámetros cuya respuesta a lazo cerrado genere el menor tiempo de establecimiento posible se construye la Tabla 10.

Tabla 10 Parámetros PID para configuración en cruz

Variable	Proporcional *K*	Derivativa *D*	Integral *I*
Roll	$K_{Roll} = 1.7537$	$D_{Roll} = 0.8094$	$I_{Roll} = 2.8$
Pitch	$K_{Pitch} = 1.7537$	$D_{Pitch} = 0.8094$	$I_{Pitch} = 2.8$
Yaw	$K_{Yaw} = 0.2239$	$D_{Yaw} = 0.1369$	$I_{Yaw} = 0.2$
Altura	$K_Z = 4.2$	$D_Z = 3.2$	$I_Z = 0.1$

En la Figura 41 se muestran las variables de control frente disturbios 10 veces mayor que en el caso del control PD. Además se ha agregado disturbios en el eje z en forma de ruido gaussiano de amplitud 0.026 N.m.

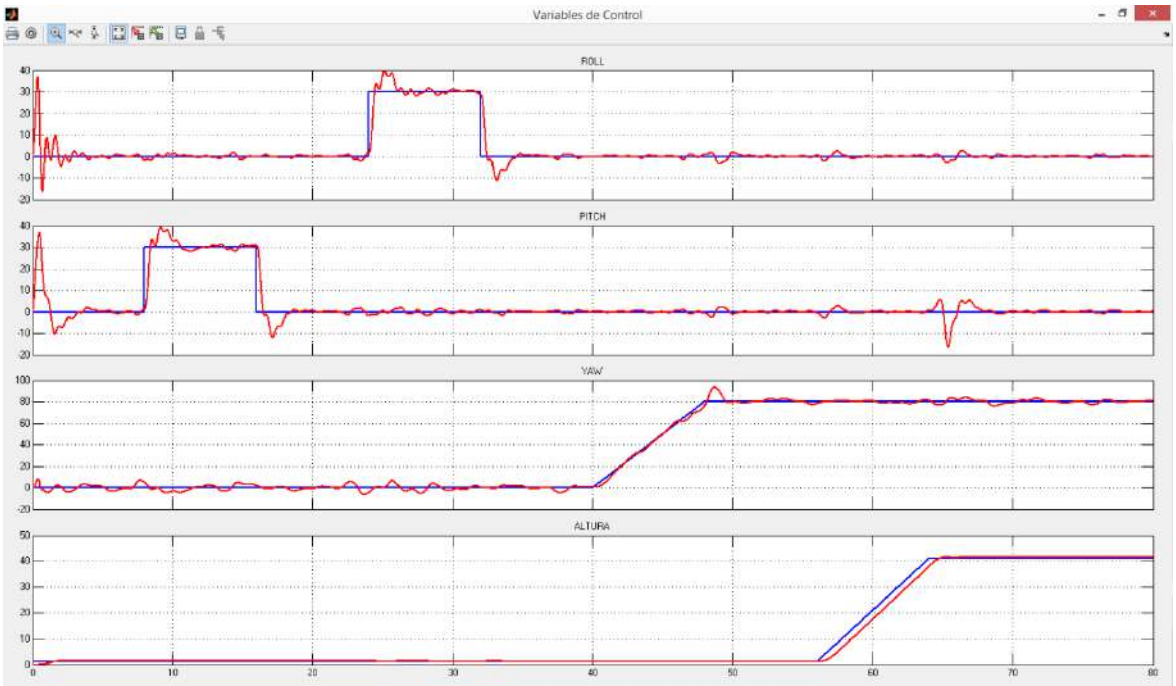

Figura 41 Variable de control frente a $A_p = 0.15N.m$, $A_q = 0.15N.m$, $A_r = 0.026N.m$. Configuración en Cruz. Fuente: Elaboración propia.

De igual manera se determinan los parámetros para la configuración en equis. Se construye la Tabla 11.

Tabla 11 Parámetros PID para configuración en equis

Variable	Proporcional K	Derivativa D	Integral I
Roll	$K_{Roll} = 2.1584$	$D_{Roll} = 0.9443$	$I_{Roll} = 2.8$
Pitch	$K_{Pitch} = 1.7537$	$D_{Pitch} = 0.8094$	$I_{Pitch} = 2.8$
Yaw	$K_{Yaw} = 0.0933$	$D_{Yaw} = 0.0653$	$I_{Yaw} = 0.1$
Altura	$K_Z = 4.2$	$D_Z = 3.2$	$I_Z = 0.1$

En la Figura 42 se muestran las variables de control para la configuración en equis bajo las mismas condiciones al caso anterior.

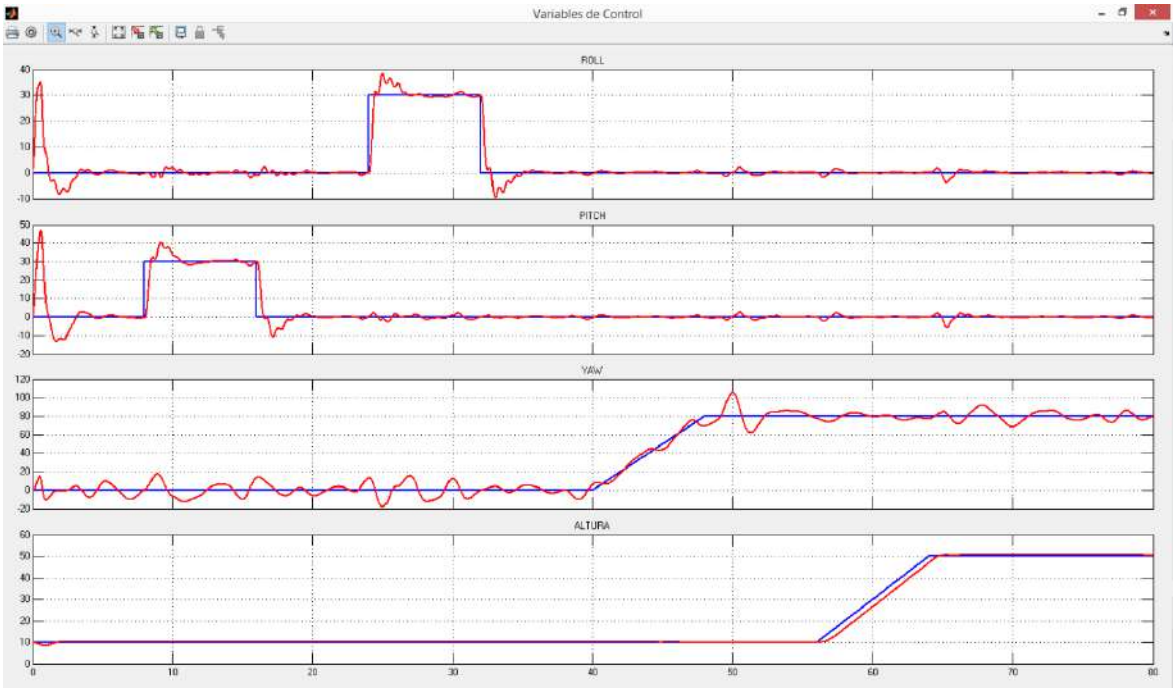

Figura 42 Variable de control con $A_p = 0.15N.m$, $A_q = 0.15N.m$, $A_r = 0.026N.m$. Configuración en Equis. Fuente: Elaboración propia.

Se comprueba la importancia de agregar la parte integral al control PD para el correcto funcionamiento del cuadricóptero bajo disturbios. Además se debe tener en cuenta que con bajos parámetros del control PID no se tienen oscilaciones en la señal PWM.

Se observa la aparición de *overshoot* al adicionar la parte integral en el sistema. Realizadas varias pruebas se ha comprobado que existe una correlación entre el aumento en el *overshoot*, la reducción en el tiempo de establecimiento, robustez frente a disturbio pero acercamiento hacia la inestabilidad con el aumento de la parte integral.

Como se ha observado en muchas de las figuras anteriormente vistas, el comportamiento del cuadricóptero con configuración cruz y equis son muy similares. Por tal motivo, en adelante sólo se trabajará con la configuración en equis.

3.2.2 Control de Posición

El objetivo de este trabajo es diseñar un controlador que permita el seguimiento de trayectorias. Un cuadricóptero es un sistema sub-actuado es decir, presenta más variables de salida que variables manipulables. Por tal motivo y para lograr nuestro objetivo es necesario implementar dos controladores en cascada.

A partir de las ecuaciones que gobiernan la dinámica sobre el eje X, Y y Z como se muestran en la ecuación (110).

$$\ddot{X} = (sen\psi sen\phi + \cos\psi\, sen\theta cos\phi)\frac{U_1}{m} + \frac{A_X}{m}$$

$$\ddot{Y} = (-\cos\psi\, sen\phi + sen\psi sen\theta cos\phi)\frac{U_1}{m} + \frac{A_Y}{m} \tag{110}$$

$$\ddot{Z} = -g + (\cos\theta cos\phi)\frac{U_1}{m} + \frac{A_Z}{m}$$

Donde A_X, A_Y y A_Z son fuerzas sobre los ejes X, Y y Z respectivamente como producto de perturbaciones externas como por ejemplo la carga del viento. De (110) se pueden obtener las ecuaciones (111) donde se muestra como calcular los ángulos "*Roll*" y "*pitch*" para alcanzar determinadas posiciones.

$$\phi_d = \left(\ddot{X} sen\psi - \ddot{Y}_d \cos\psi\right)\frac{m}{U_1}$$

$$\theta_d = \left(\ddot{X} cos\psi + \ddot{Y}_d\, sen\,\psi\right)\frac{m}{U_1} \tag{111}$$

Las estructuras de control serán las siguientes:

$$\ddot{X}_{Ctrl} = (X_d - X) * K_X + (X_d - X) * \frac{I_X}{s} - X * s * D_X$$

$$\ddot{Y}_{Ctrl} = (Y_d - Y) * K_Y + (Y_d - Y) * \frac{I_Y}{s} - Y * s * D_Y \tag{112}$$

En la Figura 43 se esquematiza la estructura de control en cascada. Se observa la necesidad de dos transformaciones.

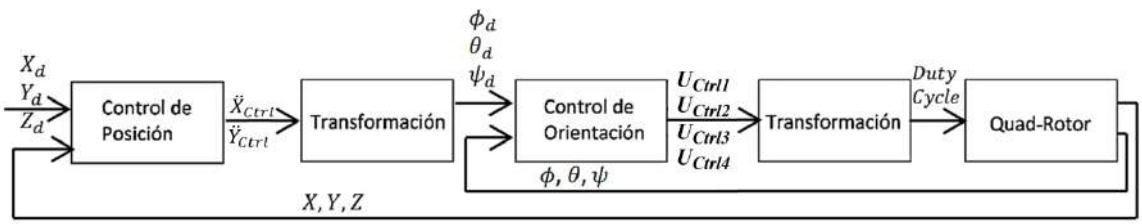

Figura 43 Control de posición en cascada. Fuente: Elaboración propia.

El mismo esquema de control se ha construido en Simulink y se muestra en la Figura 44. Se han separado en 3 bloques, en el primero se construye la referencia, en el segundo se tiene el control de posición y en el tercero se tiene el control de orientación.

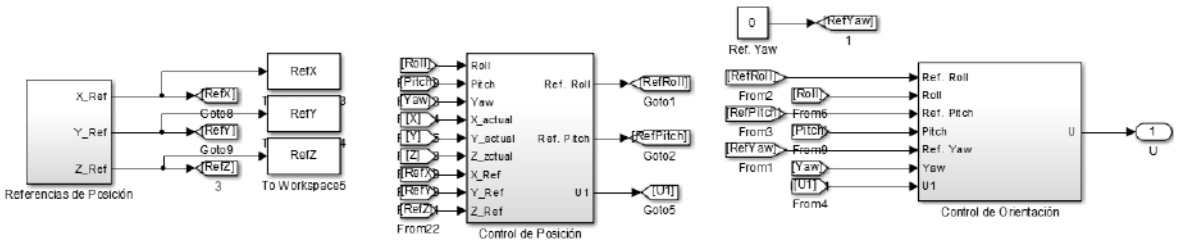

Figura 44 Esquema de control en Simulink. Fuente: Elaboración propia.

Para la determinación de los parámetros del controlador se realiza el mismo procedimiento al caso de control de orientación. Primero se buscará la función transferencia de lazo cerrado del sistema de control mostrado en la Figura 43. La función transferencia del control de orientación es integrada por $P_{\phi_{CL}}$, $P_{\theta_{CL}}$y $P_{\psi_{CL}}$ para el *roll*, *pitch* y *yaw* respectivamente.

El ángulo consigna es asignado a través de las ecuaciones (111). Para esta ecuación se realizada dos hipótesis. En primero lugar se considera un ángulo *yaw* igual a cero. En segundo lugar se considera un empuje vertical constante e igual al peso del cuerpo. La relación entre los ángulos consigna ϕ_d, θ_d y las salidas de los controladores de posición en el plano $\ddot{X}_{control}$ y $\ddot{Y}_{control}$ son las que se muestran en la ecuación (113).

$$P_{\theta_X} = \frac{\theta_d}{\ddot{X}_{Ctrl}} = 1/g$$
$$P_{\phi_Y} = \frac{\phi_d}{\ddot{Y}_{Ctrl}} = 1/g \tag{113}$$

Los ángulos del sistema quedan expresados como se muestran en la ecuación (114).

$$\theta = \theta_d * P_{\theta_{CL}}$$
$$\phi = \phi_d * P_{\phi_{CL}} \tag{114}$$
$$\psi = \psi_d * P_{\psi_{CL}}$$

La posición del cuerpo viene expresada por la ecuación (110). Si se considera $\psi = 0$. Se puede reescribir la ecuación (110) como se muestra en (115).

$$\ddot{X} = (sen\theta cos\phi)\frac{U_1}{m} \tag{115}$$

$$\ddot{Y} = (-sen\phi)\frac{U_1}{m} \tag{116}$$

$$\ddot{Z} = -g + (\cos\theta cos\phi)\frac{U_1}{m} \tag{117}$$

Para determinado instante, dado un θ y ϕ, el empuje vertical para una altura fija se expresa como se ve en la ecuación (118).

$$U_1 = \frac{mg}{\cos\theta cos\phi} \tag{118}$$

Reemplazando la ecuación (118) en (115) y (116) se obtiene la ecuación (119).

$$\begin{gathered} \ddot{X} = g(tg\theta) \\ \ddot{Y} = -g(tg\phi) \end{gathered} \tag{119}$$

Se puede aproximar la tangente al valor del ángulo. Por lo tanto la relación entre θ, ϕ y la posición en el plano X, Y respectivamente se puede expresar mediante la ecuación (25).

$$\begin{gathered} P_{X_\theta} = \frac{X}{\theta} = \frac{g}{s^2} \\ P_{Y_\phi} = \frac{Y}{\phi} = -\frac{g}{s^2} \end{gathered} \tag{120}$$

Por lo tanto se puede finalmente expresar el sistema de posición en lazo abierto teniendo el sistema de orientación en lazo cerrado mediante la ecuación (121).

$$\begin{gathered} P_X = \frac{X}{\ddot{X}_{Ctrl}} = P_{\theta_X} * P_{\theta_{CL}} * P_{X_\theta} = P_{\theta_{CL}} * \frac{1}{s^2} \\ P_Y = \frac{Y}{\ddot{Y}_{Ctrl}} = P_{\phi_Y} * P_{\phi_{CL}} * P_{Y_\phi} = P_{\phi_{CL}} * \frac{1}{s^2} \\ P_Z = \frac{Z}{U_{\mathrm{Ctrl1}}} = \frac{1}{ms^2(\tau_m s + 1)} \end{gathered} \tag{121}$$

Por lo tanto el sistema a lazo cerrado se escribe en la ecuación (122).

$$\begin{gathered} P_{X_{CL}} = \frac{X}{X_d} = \frac{(K_X + I_X/s)P_X}{1 + \left(K_X + D_X s + \frac{I_X}{s}\right)P_X} \\ P_{Y_{CL}} = \frac{Y}{Y_d} = \frac{(K_Y + I_Y/s)P_Y}{1 + \left(K_Y + D_Y s + \frac{I_Y}{s}\right)P_Y} \\ P_{Z_{CL}} = \frac{Z}{Z_d} = \frac{(K_Z + I_Z/s)P_Z}{1 + \left(K_Z + D_Z s + \frac{I_Z}{s}\right)P_Z} \end{gathered} \tag{122}$$

Para este caso es muy importante determinar las posiciones angulares que se necesitan para un *setpoint* de posición del cuadricóptero.

$$T_{\theta_{CL}} = \frac{\theta}{X_d} = \frac{(K_X + I_X/s)P_X * s^2/g}{1 + \left(K_X + D_X s + \frac{I_X}{s}\right) P_X}$$

$$T_{\phi_{CL}} = \frac{\phi}{Y_d} = \frac{(K_Y + I_Y/s)P_Y * s^2/g}{1 + \left(K_Y + D_Y s + \frac{I_Y}{s}\right) P_Y} \tag{123}$$

$$P_{\psi_{CL}} = \frac{\psi}{\psi_d} = \frac{\left(K_\psi + I_\psi/s\right)P_\psi}{1 + \left(K_\psi + D_\psi s + \frac{I_\psi}{s}\right) P_\psi}$$

A la vez se puede determinar los movimientos principales U_1, U_2, U_3 y U_4

$$T_{Z_{CL}} = \frac{U_1}{Z_d} = \frac{P_Z}{1 + \left(K_Z + D_Z s + \frac{I_Z}{s}\right) P_Z}$$

$$T_{U_{2CL}} = \frac{U_2}{Y_d} = \frac{(K_Y + I_Y/s)P_Y * s^2/g}{1 + \left(K_Y + D_Y s + \frac{I_Y}{s}\right) P_Y} * \frac{s^4 I_{XX}}{gl}$$

$$T_{U_{3CL}} = \frac{U_3}{X_d} = \frac{(K_X + I_X/s)P_X}{1 + \left(K_X + D_X s + \frac{I_X}{s}\right) P_X} * \frac{s^4 I_{YY}}{gl} \tag{124}$$

$$T_{\psi_{CL}} = \frac{U_4}{\psi_d} = \frac{P_\psi}{1 + \left(K_\psi + D_\psi s + \frac{I_\psi}{s}\right) P_\psi}$$

Por lo tanto se implementan funciones de costo para el diseño de los controladores. La función de costo son las mostradas en el sistema de ecuaciones (125).

$$J_X = \left(time_X * (X - X_d)\right)^T * (X - X_d) + R_X (time_\theta * \theta)^T * \theta \tag{125}$$

$$J_Y = \left(time_Y * (Y - Y_d)\right)^T * (Y - Y_d) + R_Y \left(time_\phi * \phi\right)^T * \phi \tag{126}$$

$$J_Z = \left(time_Z * (Z - Z_d)\right)^T * (Z - Z_d) + R_Z \left(time_{U_1} * U_1\right)^T * U_1 \tag{127}$$

$$J_\theta = (time_\theta * \theta)^T * \theta + R_\theta \left(\frac{\partial U_3}{\partial t}\right)^2 \tag{128}$$

$$J_\phi = \left(time_\phi * \phi\right)^T * \phi + R_\phi \left(\frac{\partial U_2}{\partial t}\right)^2 \tag{129}$$

$$J_\psi = \left(time_\psi * \psi\right)^T * \psi + R_\psi \left(\frac{\partial U_4}{\partial t}\right)^2 \tag{130}$$

Los parámetros establecidos se muestran a continuación en la Tabla 12.

Tabla 12 Parámetros PID para control en cascada

Variable	Proporcional *K*	Derivativa *D*	Integral *I*
X	$K_X = 1.64$	$D_X = 2.77$	$I_X = 0.0252$
Y	$K_Y = 1.64$	$D_Y = 2.77$	$I_Y = 0.0252$
Z	$K_Z = 7.48$	$D_Z = 5.98$	$I_Z = 0.1$
Roll	$K_{Roll} = 11.22$	$D_{Roll} = 5.10$	$I_{Roll} = 1.02$
Pitch	$K_{Pitch} = 11.22$	$D_{Pitch} = 5.10$	$I_{Pitch} = 1.02$
Yaw	$K_{Yaw} = 0.50$	$D_{Yaw} = 0.27$	$I_{Yaw} = 0.25$

Para verificar el correcto control se han realizado dos pruebas. La primera bajo condiciones perfectas es decir, considerando ausencia de perturbaciones, mientras que en el segundo caso se han considerado perturbaciones en todas las variables.

Los resultados se muestran a continuación en la Figura 45 y Figura 46.

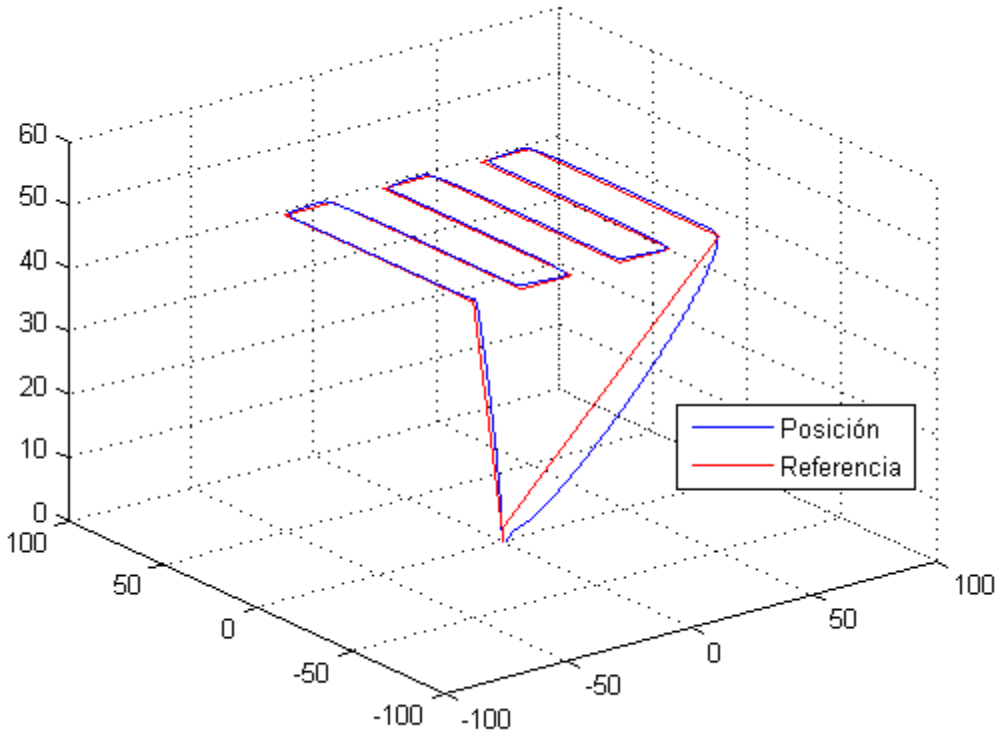

Figura 45 Variables de posición en 3D sin perturbaciones. Fuente: Elaboración propia.

Como se observa en la Figura 45 el control de posición se da satisfactoriamente. La simulación se ha dado sin perturbaciones y ha durado poco más de 5 minutos en realizar todo el recorrido, desde el despegue hasta el aterrizaje.

Para el segundo caso se han realizado las siguientes consideraciones: $A_p = 0.5N.m$, $A_q = 0.5N.m$ y $A_r = 0.02\ N.m$, $A_X = A_Y = 5\ N$ y $A_Z = -5N$ los cuales son impulsos de $0.02\ segundos$ de duración. Estos valores han sido también utilizados en [18].

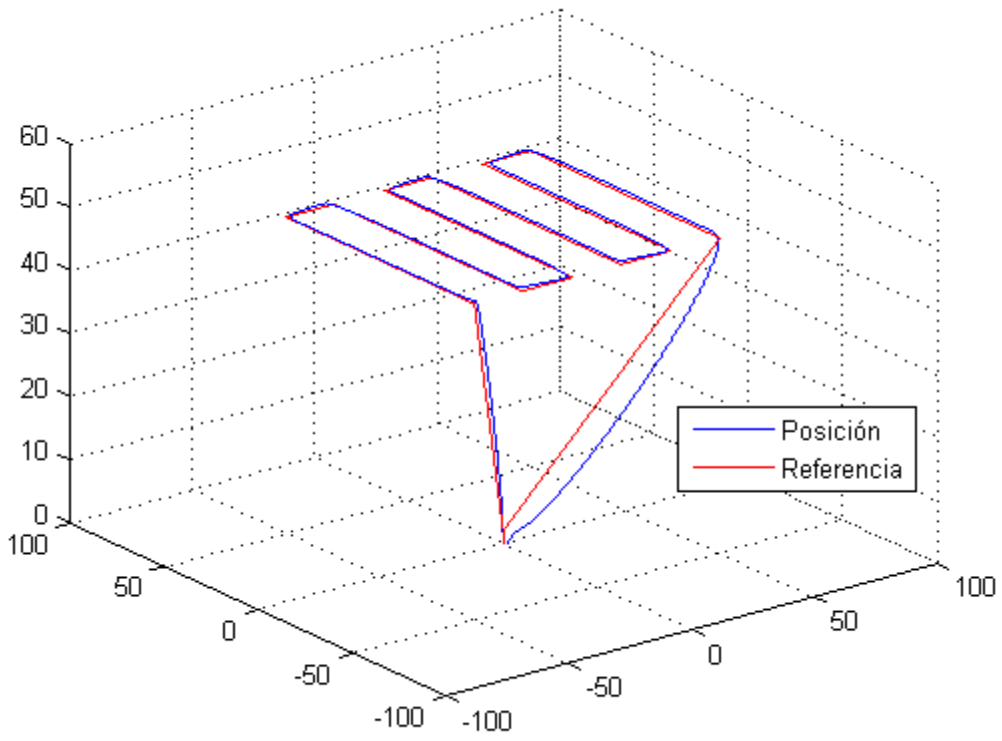

Figura 46 Variables de posición en 3D con perturbaciones. Fuente: Elaboración propia.

Se observa las buenas prestaciones del control. Donde ha podido sobre ponerse a todas las perturbaciones tanto de pares como de fuerzas sobre el cuadricóptero. Otra vista de la trayectoria recorrida y las posición referencias se muestra en la Figura 47. Se observa que las variables de salida de posición siguen satisfactoriamente a su respectiva referencia, sin embargo existe un pequeño retardo de aproximadamente 5 $segundos$

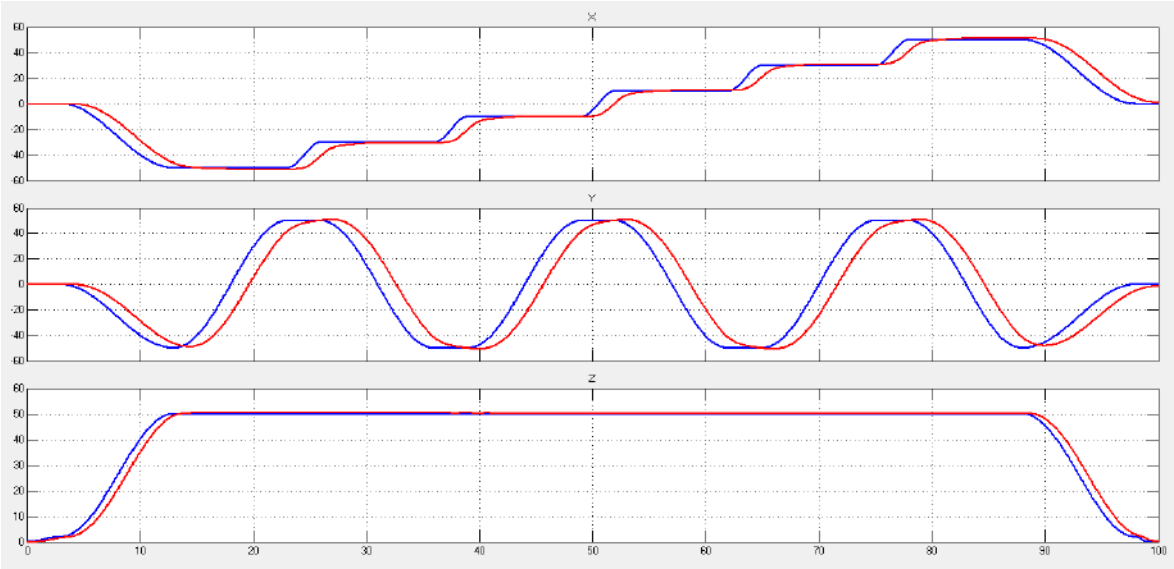

Figura 47 Comparación de trayectoria recorrida y la referencia de posición. Fuente: Elaboración propia.

Otra gráfica importante es la dibujada por la dinámica de los ángulos de orientación para conseguir las posiciones esperadas. En la Figura 48 se observa que todos los ángulos están bajo control y con una dinámica suave.

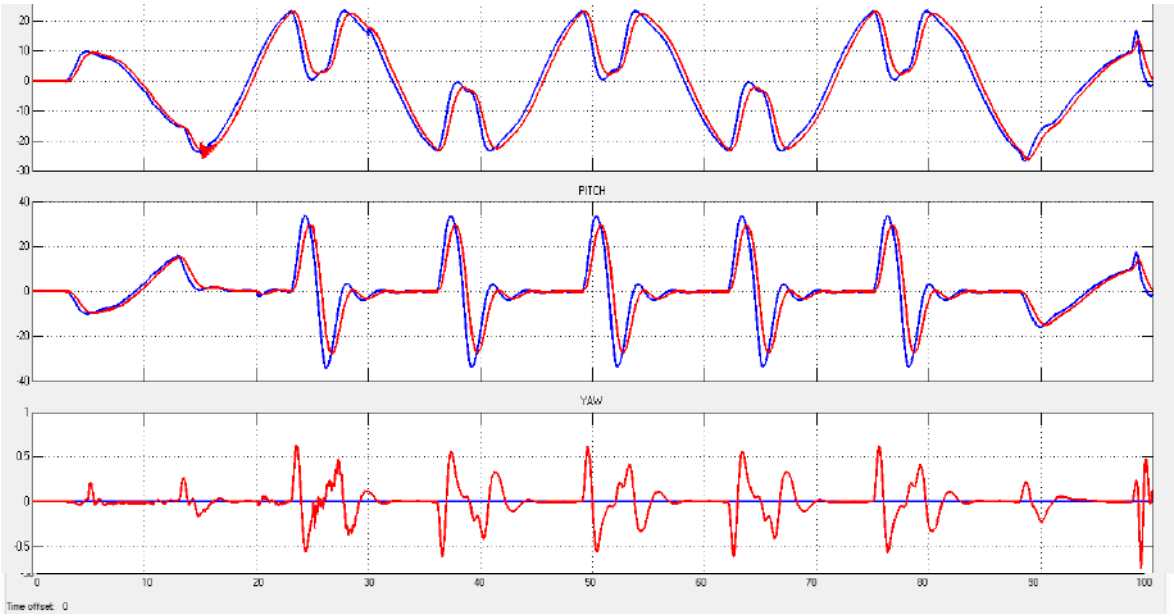

Figura 48 Ángulos "roll", "pitch" y "yaw" del cuadricóptero y sus referencias. Fuente: Elaboración propia.

Muy importante es saber el cómo se comporta la variable manipulable. En este caso, la variable manipulable es el "*duty cycle*" que alimenta a los driver del motor. En la Figura 49 se observa que la variable manipulable no se encuentra saturada, sin embargo a los 15 y 25 segundos se presentan cambios fuertes en el "*duty cycle*".

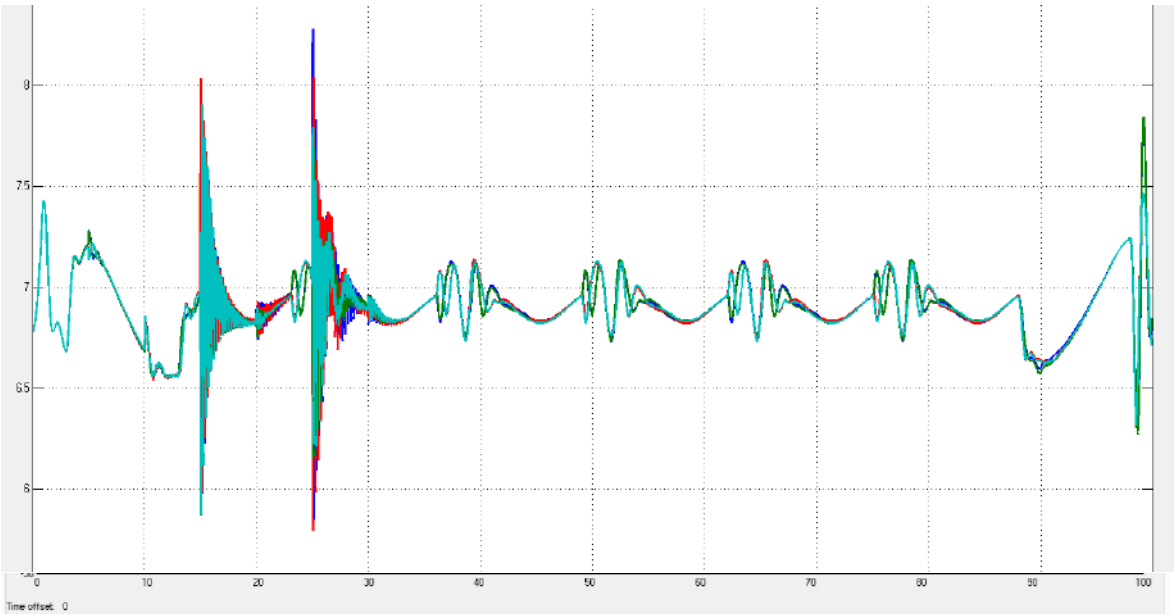

Figura 49 Duty Cycle de los motores. Fuente: Elaboración propia.

Los cambios fuertes del "*duty cycle*" corresponden a las perturbaciones simuladas en el "*roll*" y en el "*pitch*". Esta dinámica rápida podría representar serios problema al intentar implementar el algoritmo de control, ya que se hay problemas entre el tiempo de muestre y los cambios bruscos en tiempos cortos como es en este caso.

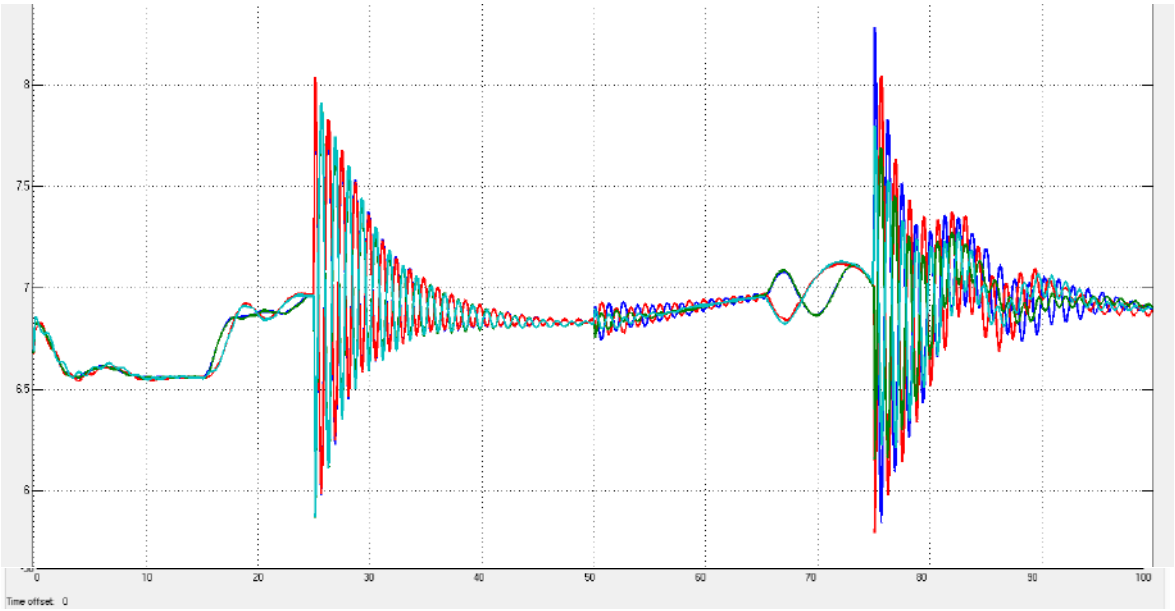

Figura 50 *Duty cycle* frente a perturbaciones en el *roll* y *pitch*. Fuente: Elaboración propia.

Otra gráfica muy importante es aquella que permite observar las fuerzas que realmente se están ejerciendo sobre el dron para alcanzar determinados movimientos. En la Figura 51 se puede observar variaciones alternas de las fuerzas para producir "*roll*" y "*pitch*". Resulta poco lógica la necesidad de aplicar fuerzas alternadas para estabilizar un cuerpo al cual es afectado por un disturbio.

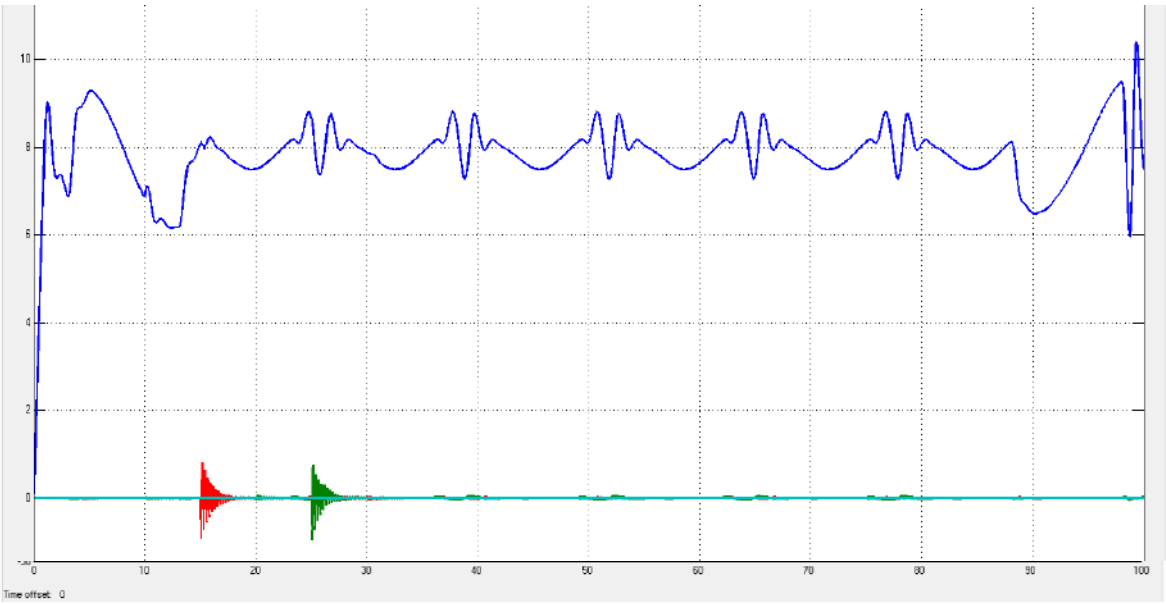

Figura 51 Movimientos principales. Fuente: Elaboración propia.

En la Figura 52 se muestran dos casos. El primer caso (señalado como perturbación) corresponde a las fuerzas principales producidas como respuesta a la perturbación externa para mantener la estabilidad de la nave. En el segundo caso (señalado como cambio de ángulo consigna) muestra el comportamiento de los movimientos principales para seguir los ángulos consigna mostrados en la Figura 48.

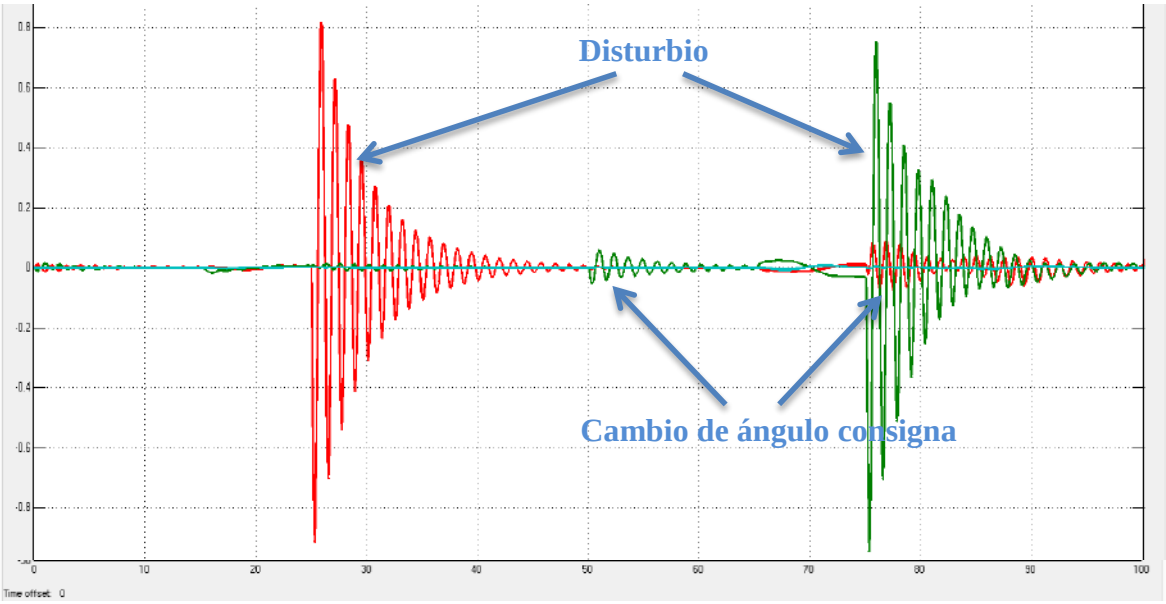

Figura 52 Fuerzas para producir movimiento y anteponerse a disturbios en el *roll* y *pitch*.

Fuente: Elaboración propia.

En ambos casos parece absurda la forma como se aplica la fuerza sobre el cuadricóptero. Se realizaron muchas pruebas con diferentes parámetros del PID, sin lograr mejorar este comportamiento. Recordemos la Figura 39 donde aparece el "*duty cycle*" para los controladores PD en donde no se observar cambios aleatorios. Por lo tanto se concluye que

el añadir un término integral puede mejorar la robustez del sistema frente a perturbaciones, sin embargo el control no es eficiente y produce fuerzas no deseadas sobre el cuadricóptero.

3.3 Controladores mejorados para el ArduCopter Quad-C

En el proceso de estudio del código Multiwii 2.4 [38], se descubrió que el algoritmo de control en la versión 2.4 considera la realimentación de la velocidad angular. En el enfoque hacia la implementación, el cálculo de la derivada del ángulo de Euler puede causar serios problemas, en su lugar se utiliza la velocidad angular proporcionada por el giroscopio.

En el caso de la Multiwii como en otros sistemas comerciales preparados para controlar diversos UAV, no consideran la matriz de movimientos inversos, lo cual produce muchas complicaciones al sintonizar el controlador. Una explicación clara es, considere sustituir la ecuación (91) por cualquier otra matriz, esto echaría abajo su utilidad, la cual fue diseñar un controlador MIMO, en 4 controladores SISO (*roll*, *pitch*, *yaw* y altura).

Otro atributo importante encontrado dentro del código fue, que en el sistema Multiwii se considera la aceleración angular, ésta el calculada por medio de la variación de la velocidad angular en un intervalo de tiempo. Además se considera utilizar para este cálculo un intervalo de tiempo mayor o igual a 3 veces el tiempo de muestreo del sistema. La aceleración angular es multiplicada por una ganancia (para este trabajo se llamará $D_{2_{Roll}}$ o $D_{2_{Pitch}}$ según corresponda) y se restará a la salida de nuestro PID de orientación diseñado en la sección 3.2.

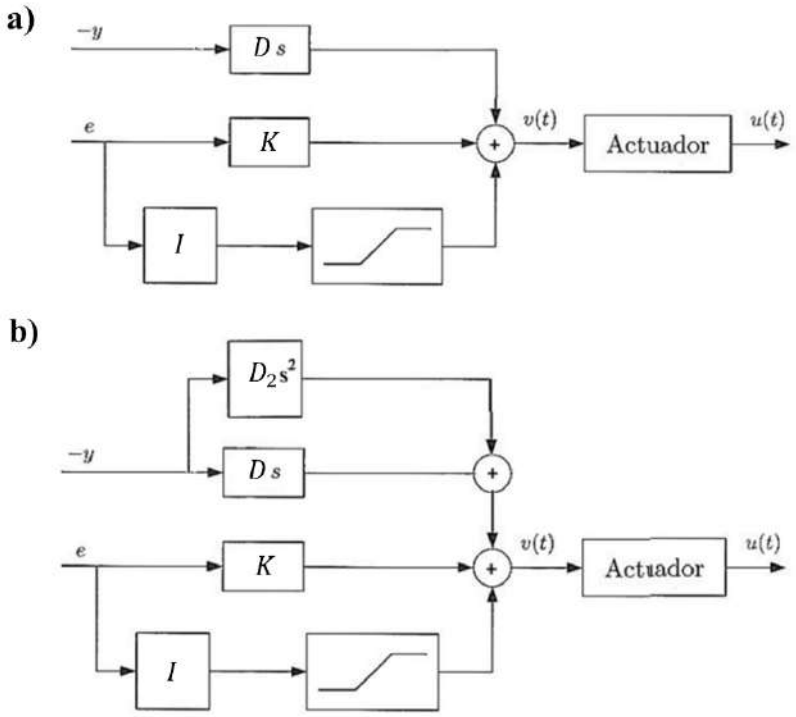

Figura 53 Comparación de estructuras de control. a) Estructura planteada en la sección 3.2. b) Nueva estructura planteada. Fuente: Elaboración Propia.

Tomando en cuenta la estructura de control de la Figura 53 b), se obtienen las nuevas funciones transferencias de lazo cerrado para el control de ángulos, ver ecuaciones (131) y

(132). Se observa en las ecuaciones (131) y (132) que el nuevo término D_2 influye directamente en el término inercial del sistema.

$$P_{\phi_{CL}} = \frac{\phi}{\phi_d} = \frac{K_{Roll} * l * s + I_{Roll} * l}{I_{XX}\tau_m s^4 + (I_{XX} + D_{2_{Roll}} l)s^3 + (D_{Roll} l)s^2 + (K_{Roll} * l)s + I_{Roll} l} \quad (131)$$

$$P_{\theta_{CL}} = \frac{\theta}{\theta_d} = \frac{K_{Pitch} * l * s + I_{Pitch} * l}{I_{YY}\tau_m s^4 + \left(I_{YY} + D_{2_{Pitch}} l\right)s^3 + (D_{Pitch} l)s^2 + (K_{Pitch} l)s + I_{Pitch} l} \quad (132)$$

En pruebas de simulación se observó que no es necesario el término D_2 para el movimiento de *yaw*. Por tal motivo, la ecuación a lazo cerrado para dicho eje, se determina mediante la ecuación (133):

$$P_{\psi_{CL}} = \frac{\psi}{\psi_d} = \frac{K_{Yaw} * s + I_{Yaw}}{I_{ZZ}\tau_m s^4 + I_{ZZ} s^3 + (D_{Yaw})s^2 + (K_{Yaw})s + I_{Yaw}} \quad (133)$$

Basados en el sistema embebido Multiwii, la salida es una señal PWM de 488 Hz. La identificación de la función transferencia del motor se realizó a 50 Hz. Dado que el motor es un sistema no lineal, se procede a realizar nuevamente la identificación del motor. Como resultado de la aproximación a un sistema de primer orden, se obtiene la ecuación (134).

$$M(s) = \frac{K_m}{\tau_m s + 1} \quad (134)$$

Donde τ_m es la contante de tiempo del motor, la cual es 0.24 segundos para una señal de 50 Hz. Sin embargo para este caso se utiliza τ_m igual a 0.155 segundos por tratarse de una señal de salida que trabaja a 500 Hz.

Para sintonizar el control PID, se tendrá en cuenta la respuesta del sistema bajo un disturbio en forma de escalón de magnitud A_p, A_q y A_r para el ϕ, θ y ψ respectivamente. La respuesta del sistema frente al disturbio se describe en las ecuaciones (135), (136) y (137).

$$N_\phi(s) = \frac{A_p(\tau_m s^2 + s)}{\tau_m I_{XX} s^4 + \left(I_{XX} + D_{2_{Roll}} l\right)s^3 + D_{Roll} l s^2 + P_{Roll} l s + I_{Roll} l} \quad (135)$$

$$N_\theta(s) = \frac{A_q(\tau_m s^2 + s)}{\tau_m I_{YY} s^4 + \left(I_{YY} + D_{2_{Pitch}} l\right)s^3 + D_{Pitch} l s^2 + P_{Pitch} l s + I_{Pitch} l} \quad (136)$$

$$N_\psi(s) = \frac{A_r(\tau_m s^2 + s)}{\tau_m I_{YY} s^4 + I_{ZZ} s^3 + D_{Yaw} s^2 + P_{Yaw} s + I_{Yaw}} \quad (137)$$

Otra señal importante es la variable manipulable. Teniendo en cuenta las importantes diferencias que existen entre U_{ctrl} y U se debe hacer un análisis de ambas variables. A continuación se describen las funciones transferencias de U_{ctrl}.

$$U_{ctrl_\phi} = \frac{(I_{XX}P_{Roll}\tau_m)s^4 + (I_{XX}P_{Roll} + I_{XX}I_{Roll}\tau_m)s^3 + (I_{XX}I_{Roll})s^2}{\tau_m I_{XX}s^4 + \left(I_{XX} + D_{2_{Roll}}l\right)s^3 + D_{Roll} * ls^2 + P_{Roll}ls + I_{Roll} * l} \quad (138)$$

$$U_{ctrl_\theta} = \frac{(I_{YY}P_{Pitch}\tau_m)s^4 + (I_{YY}P_{Pitch} + I_{YY}I_{Pitch}\tau_m)s^3 + (I_{YY}I_{Pitch})s^2}{\tau_m I_{YY}s^4 + \left(I_{YY} + D_{2_{Pitch}}l\right)s^3 + D_{Pitch} * ls^2 + P_{Pitch}ls + I_{Pitch} * l} \quad (139)$$

$$U_{ctrl_\psi} = \frac{(I_{ZZ}P_{Yaw}\tau_m)s^4 + (I_{ZZ}P_{Yaw} + I_{ZZ}I_{Yaw}\tau_m)s^3 + (I_{ZZ}I_{Yaw})s^2}{\tau_m I_{ZZ}s^4 + (I_{ZZ})s^3 + D_{Yaw} * ls^2 + P_{Yaw}ls + I_{Yaw} * l} \quad (140)$$

De las ecuaciones (138), (139) y (140) se puede demostrar que frente a un cambio de referencia escalón de 1 radián ejercen una señal de control igual a P_{Roll}, P_{Pitch} y P_{Yaw} respectivamente en el instante cero. Teniendo en cuenta que U_{ctrl} representa la fuerza para "*roll*" y "*pitch*" y momento para "*yaw*" deseado por el sistema de control para corregir el error presente. Analíticamente se sabe que las máximas fuerzas y momentos están determinadas por los rotores según la configuración del cuadricóptero. Considerando que la configuración en equis por ser la más estable y además considerando saturación de los actuadores, ver ecuación (141), (142):

$$U_{\phi_{max}} = U_{\theta_{max}} = 2b * (\Omega_{max}^2 - \Omega_{min}^2) = 8.28\ N \quad (141)$$

$$U_{\psi_{max}} = d * (\Omega_{max}^2 - \Omega_{min}^2) = 0.1432\ N.m \quad (142)$$

Dado que las condiciones de las ecuaciones (141) y (142) sólo son posibles cuando el cuadricóptero se encuentra en condición de vuelo estacionario en el centro del rango de operación de la PWM. Teniendo en cuenta que es muy probable que las condiciones de vuelo estacionario puede darse cerca de las zonas de saturación de la señal PWM. Por lo tanto, dependiendo del factor de seguridad que se desee aplicar y al máximo cambio de consigna escalón que sea permitido, se tendrá que diseñar la ganancia proporcional.

Sin embargo, la variable manipulable de interés es U y no U_{ctrl}. A continuación se describen las funciones transferencias de U. Ver ecuación (143), (144) y (145).

$$U_\phi = \frac{(I_{XX}P_{Roll})s^3 + (I_{XX}I_{Roll})s^2}{\tau_m I_{XX}s^4 + \left(I_{XX} + D_{2_{Roll}}l\right)s^3 + D_{Roll} * ls^2 + P_{Roll}ls + I_{Roll} * l} \quad (143)$$

$$U_\theta = \frac{(I_{YY}P_{Pitch})s^3 + (I_{YY}I_{Pitch})s^2}{\tau_m I_{YY}s^4 + \left(I_{YY} + D_{2_{Pitch}}l\right)s^3 + D_{Pitch} * ls^2 + P_{Pitch}ls + I_{Pitch} * l} \quad (144)$$

$$U_\psi = \frac{(I_{ZZ}P_{Yaw})s^3 + (I_{ZZ}I_{Yaw})s^2}{\tau_m I_{ZZ}s^4 + \left(I_{ZZ} + D_{2_{Yaw}}\right)s^3 + D_{Yaw} * ls^2 + P_{Yaw}ls + I_{Yaw} * l} \quad (145)$$

Teniendo como objetivo realizar pruebas experimentales, se ha procedido a editar el código Multiwii versión 2.4. En el Anexo F se presenta el código principal editado para nuestro propósito, en donde se encuentra programado nuestro controlador. Para esta sección es importante mencionar, que cada unidad en "roll" y "pitch" en el embebido equivale a 0.1 grados sexagesimales. Además hay que tener en cuenta que cada unidad del giroscopio es

equivalente a 1/2.4414 °/segundo, por este motivo se decide multiplicar a las ganancias derivativas por 2.4414 para compensar dicho escalamiento.

En el Anexo G se presenta el código para la lectura de datos del sensor IMU embebido en la tarjeta Multiwii pro. En este segmento sólo se ha editado la escala del ángulo "*yaw*". Cada unidad actual es equivalente a 0.1 grados sexagesimales. Antes cada unidad era equivalente a 1 grado sexagesimal.

En el Anexo H se presenta el código para la señal de salida, en este bloque se edita la matriz de movimientos inversos y la caracterización del motor. Además se elimina una restricción para el ángulo "yaw".

3.4 Sintonización de Controladores

En este trabajo se utilizó un método iterativo que busca minimizar una función de costo específica. La lógica consiste básicamente en sancionar el error de la variable de controlable con respecto a su referencia considerando un disturbio externo. Además se añade la sanción de la energía de la variable manipulable para evitar oscilaciones. Este método arroja valores a partir de los cuales se pueden variar según determine el diseñador. El objetivo de diseñar un método iterativo que minimice una función de coste, es encontrar de manera sencilla, parámetros del controlador aceptables.

Las funciones de costo se muestran a continuación, para el *roll*:

$$y = step\left(\frac{Par_{Roll}(\tau s^2 + s) + Kp_{Roll}lS + Ki_{Roll}l}{\tau I_{XX}s^4 + (I_{XX} + Kd2_{Roll}l)s^3 + Kd_{Roll}ls^2 + Kp_{Roll}ls + Ki_{Roll}l}\right) \quad (146)$$

$$u = step(\frac{I_{XX}Kp_{Roll}s^3 + I_{XX}Ki_{Roll}s^2}{\tau I_{XX}s^4 + (I_{XX} + Kd2_{Roll}l)s^3 + Kd_{Roll}ls^2 + Kp_{Roll}ls + Ki_{Roll}l} \quad (147)$$

$$J_{Roll} = (y-1)' * (y-1) * R_{Roll} + (timeu.* u)' * (u); \quad (148)$$

Para el *pitch*:

$$y = step\left(\frac{Par_{Pitch}(\tau s^2 + s) + Kp_{Pitch}lS + Ki_{Pitch}l}{\tau I_{YY}s^4 + (I_{YY} + Kd2_{Pitch}l)s^3 + Kd_{Pitch}ls^2 + Kp_{Pitch}ls + Ki_{Pitch}l}\right) \quad (149)$$

$$u = step(\frac{I_{YY}Kp_{Pitch}s^3 + I_{YY}Ki_{Pitch}s^2}{\tau I_{YY}s^4 + (I_{YY} + Kd2_{Pitch}l)s^3 + Kd_{Pitch}ls^2 + Kp_{Pitch}ls + Ki_{Pitch}l} \quad (150)$$

$$J_{Pitch} = (y-1)' * (y-1) * R_{Pitch} + (timeu.* u)' * (u); \quad (151)$$

Para el *yaw*:

$$y = step\left(\frac{Par_{Yaw} * (s^2 + s) + Kp_{Yaw} * s\, Ki_{Yaw}}{\tau I_{ZZ}s^4 + (I_{ZZ} + Kd2_{Yaw})s^3 + Kd_{Yaw} * s^2 + Kp_{Yaw} * s + Ki_{Yaw}}\right) \quad (152)$$

$$u = step\left(\frac{I_{ZZ}Kp_{Yaw}s^3 + I_{ZZ}Ki_{Yaw}s^2}{\tau I_{ZZ}s^4 + (I_{ZZ} + Kd2_{Yaw})s^3 + Kd_{Yaw} * s^2 + Kp_{Yaw} * s + Ki_{Yaw}}\right) \quad (153)$$

$$J_{Yaw} = (y-1)' * (y-1) \; + \; (timeu.* u)' * (u) * R_{Yaw} \qquad (154)$$

Donde:

Par_{Roll}, Par_{Pitch} y Par_{Yaw} corresponden a disturbios externos en forma de momento par.

R_{Roll}, R_{Pitch} y R_{Yaw} son los pesos para cada función de costo.

Los parámetros establecidos se muestran a continuación en la Tabla 14.

Tabla 13 Parámetros PID2 para control en cascada mejorado

Variable	Ganancia *K*	Ganancia *I*	Ganancia *D*	Ganancia D_2
X	$K_X = 1.64$	$I_X = 0.0252$	$D_X = 2.77$	No aplica
Y	$K_Y = 1.64$	$I_Y = 0.0252$	$D_Y = 2.77$	No aplica
Z	$K_Z = 7.48$	$I_Z = 0.1$	$D_Z = 5.98$	No aplica
Roll	$K_{Roll} = 6.20$	$I_{Roll} = 5.3$	$D_{Roll} = 2.00$	$D_{2_{Roll}} = 0.0850$
Pitch	$K_{Pitch} = 6.6$	$I_{Pitch} = 4.70$	$D_{Pitch} = 2.70$	$D_{2_{Pitch}} = 0.103$
Yaw	$K_{Yaw} = 0.26$	$I_{Yaw} = 0.21$	$D_{Yaw} = 0.15$	$D_{2_{Yaw}} = 0.00$

Para verificar el correcto control se han realizado dos pruebas. La primera bajo condiciones perfectas es decir, considerando ausencia de perturbaciones, mientras que en el segundo caso se han considerado perturbaciones en todas las variables. Los resultados se muestran a continuación en la Figura 54.

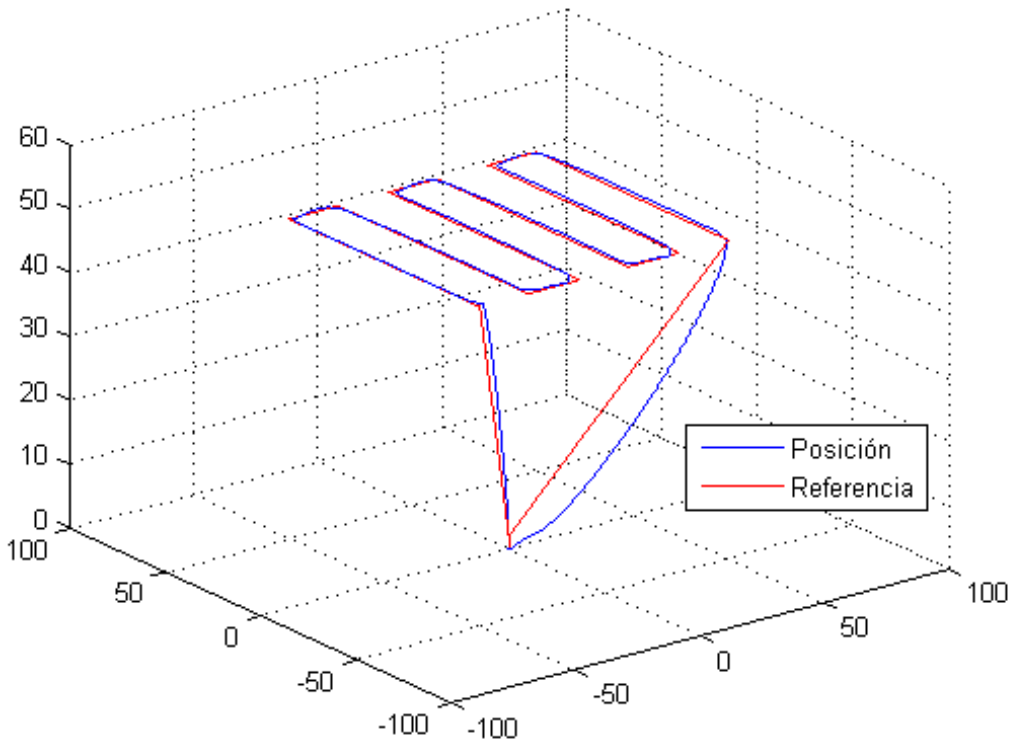

Figura 54 Variables de posición en 3D sin perturbaciones para el caso PID2. Fuente: Elaboración propia.

Como se observa en la Figura 45 el control de posición se da satisfactoriamente. La simulación se ha dado sin perturbaciones y ha durado poco más de 5 minutos en realizar todo el recorrido, desde el despegue hasta el aterrizaje.

Para el segundo caso se han realizado las siguientes consideraciones: $A_p = 0.1N.m$, $A_q = 0.1N.m$ y $A_r = 0.05\,N.m$, $A_X = A_Y = 1\,N$ y $A_Z = -1N$ los cuales son escalones mantenidos. Valores similares han sido también utilizados en [18].

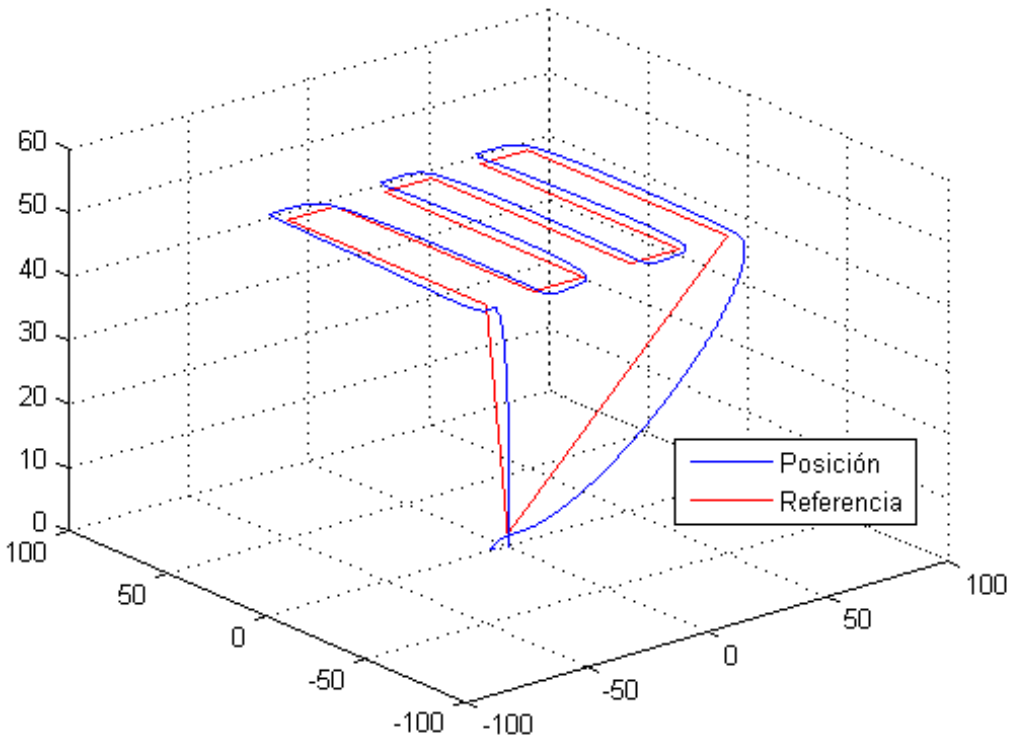

Figura 55 Variables de posición en 3D con perturbaciones para el caso PID2. Fuente: Elaboración propia.

Se observa buenas prestaciones muy similares al de la sección 3.2.2. Otra vista de la trayectoria recorrida y las posición referencias se muestra en la Figura 56. Se observa que las variables de salida de posición siguen satisfactoriamente a su respectiva referencia, sin embargo existe un pequeño retardo de aproximadamente 5 $segundos$

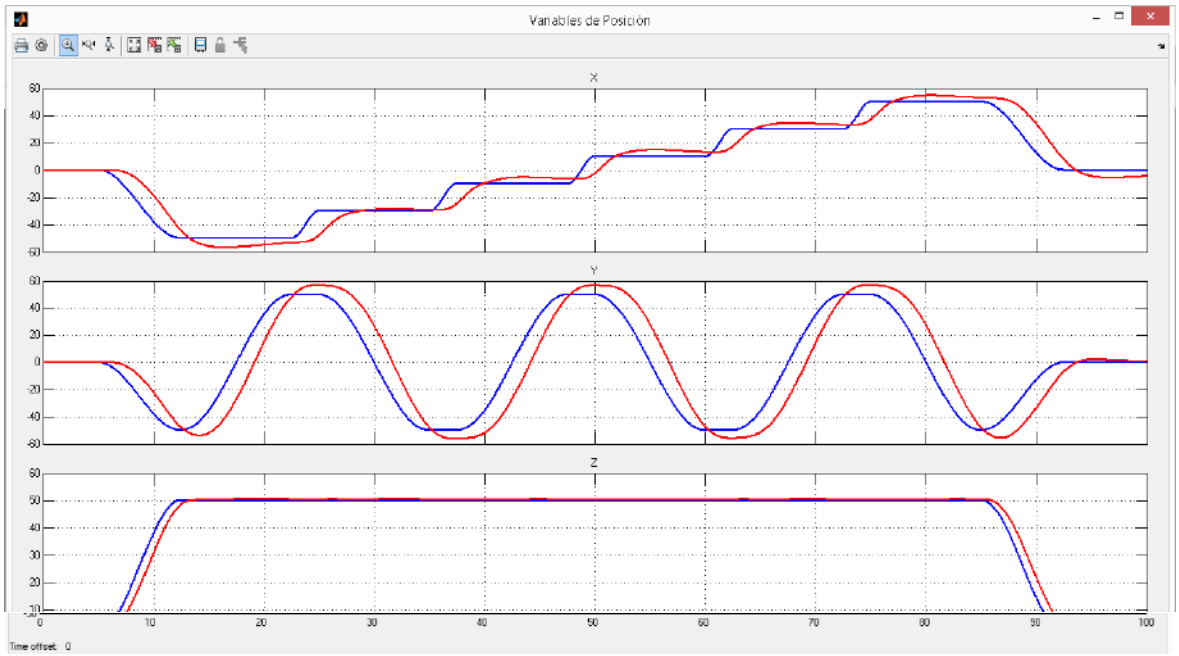

Figura 56 Comparación de trayectoria recorrida y la referencia de posición. Fuente: Elaboración propia.

Otra gráfica importante es la dibujada por la dinámica de los ángulos de orientación para conseguir las posiciones esperadas. En la Figura 57 se observa que todos los ángulos están bajo control y con una dinámica suave.

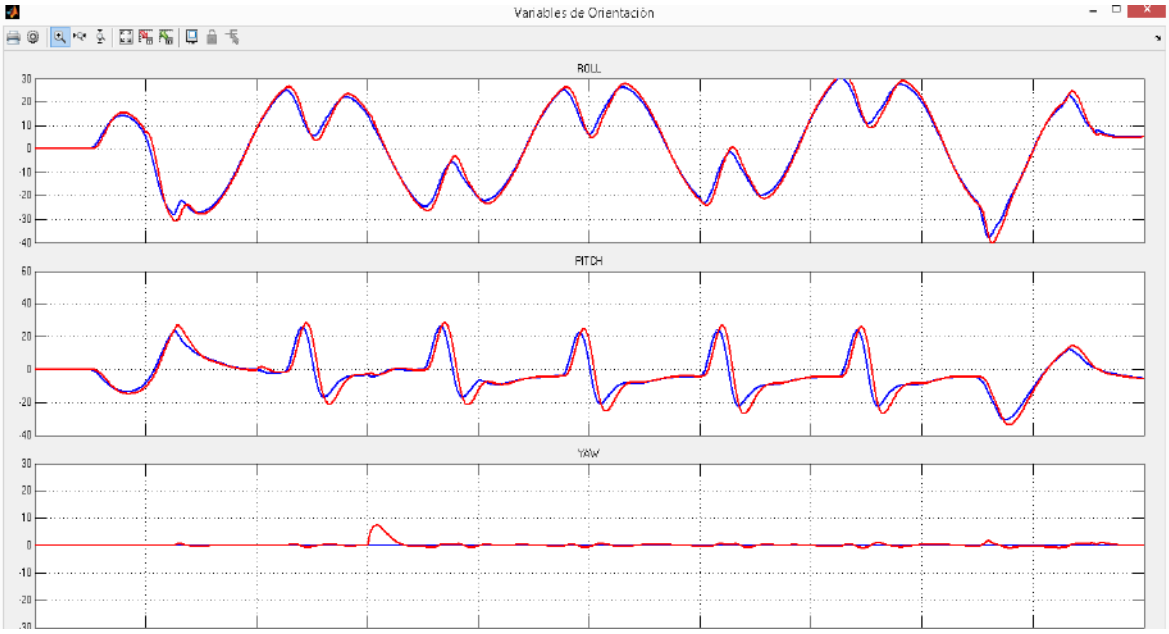

Figura 57 Ángulos "roll", "pitch" y "yaw" del cuadricóptero y sus referencias. Fuente: Elaboración propia.

Muy importante es saber el comportamiento de la variable manipulable y verificar la mejora con respecto al resultado de la sección 3.2.2. En este caso, la variable manipulable es el "*duty*

cycle" que alimenta a los driver del motor. En la Figura 49 se observa que la variable manipulable no se encuentra saturada y sin fuertes oscilaciones.

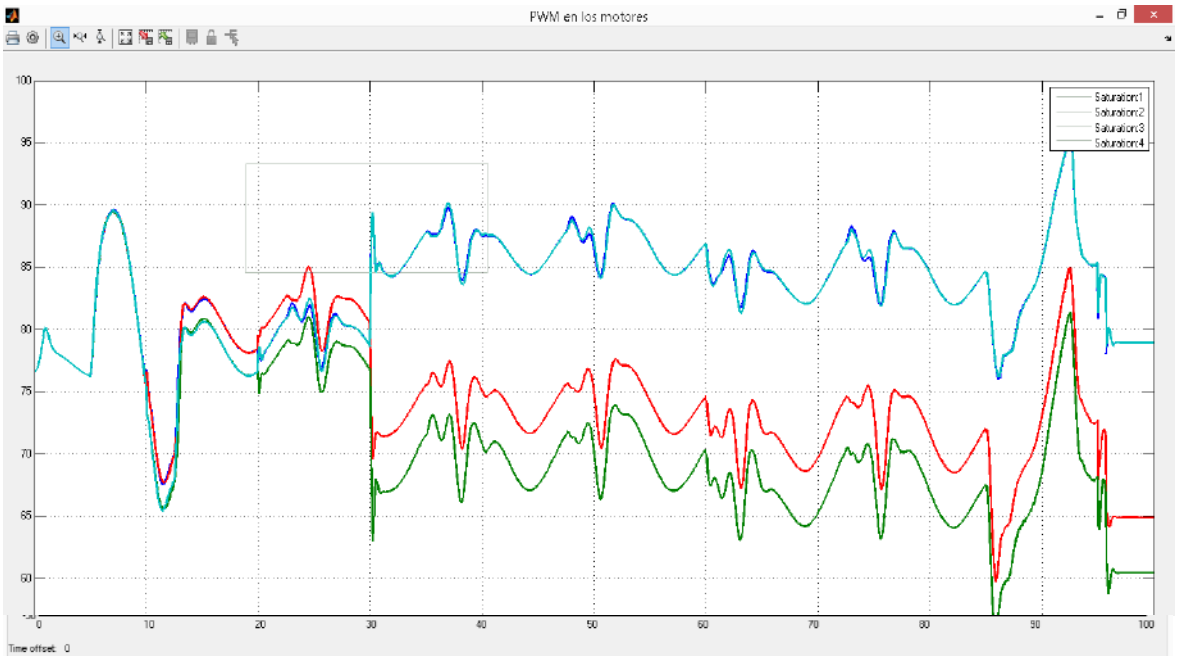

Figura 58 Duty Cycle de los motores con control PID2. Fuente: Elaboración propia.

Por lo tanto se puede concluir que el controlador PID2 ha mejorado las prestaciones del controlador PID convencional en nuestras simulaciones.

Capítulo 4
Implementación de algoritmo de control de Orientación

En este capítulo se presentan las bases para implementar el algoritmo de control de orientación en un sistema embebido. Se revisarán los problemas para ser implementados los algoritmos de control propuestos en el Capítulo 3. Se proponen nuevos algoritmos de control los cuales son implementados en el sistema embebido llamado Multiwii.

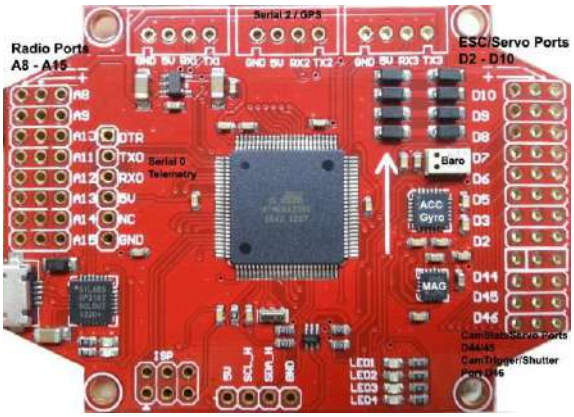

Figura 59 Circuito Embebido Multiwii. Imagen sólo referencial

En la Figura 59 se muestra una imagen representativa del Multiwii utilizado para nuestra implementación. Se ha utilizado como base el código libre proporcionado por [38]. Para este proyecto se ha utilizado la versión 2.4.

4.1 Validación del Modelo Matemático

Un cuadricóptero es un sistema no lineal e inestable, lo cual imposibilita realizar la validación del sistema a lazo abierto, por lo tanto el proceso de validación del modelo matemático se realiza a lazo cerrado.

El cuadricóptero del laboratorio de Sistemas Automáticos de Control sufrió diversas transformaciones a lo largo de la investigación, motivo por el cual se ve la necesidad de evaluar nuevamente las inercias y pesos del sistema con ayuda del sistema *SolidWorks*. En la Figura 60 se muestra el modelo construido el cual considera el sistema embebido, accesorios, motores, estructura, motores, hélices, pernos, turcas y separadores.

Figura 60 Modelo construido en *SolidWorks*. Fuente: Elaboración propia.

En la Tabla 14 se muestra el valor de las inercias calculadas y el peso del cuadricóptero. Los coeficientes de empuje b y de arrastre d se mantienen, dado que las hélices no se han cambiado. Las hélices con las que se dispone son de la marca APC 4.5x10”.

Tabla 14 Parámetros calculados en *SolidWorks*

Nombre	Símbolo	Valor
Inercia del VANT en su eje X	I_{XX}	$15.19 * 10^{-3}[\mathrm{Nm}s^2]$
Inercia del VANT en su eje Y	I_{YY}	$28.37 * 10^{-3}[\mathrm{Nm}s^2]$
Inercia del VANT en su eje Z	I_{ZZ}	$15.06 * 10^{-3}[\mathrm{Nm}s^2]$
Momento total de inercia del motor respecto a su eje Z	J_{TP}	$50 * 10^{-6}[\mathrm{Nms}^2]$
Masa del cuadricóptero	m	1.295 [kg]

Los experimentos se deben realizar en un ambiente controlado y seguro. Buscando dicha finalidad, se construye un banco de pruebas inspirando en el trabajo de [5] y [39] El sistema consiste en una base articulada con 3 grados de libertad como se muestra en Figura 61.

La captura de datos para la validación se realiza utilizando una placa Arduino UNO, un sensor IMU MPU6050, el programa Arduino versión 1.6.3, Visual Studio 2012. La lógica es la siguiente, el sensor IMU MPU6050 está compuesto de 3 acelerómetros, 3 giroscopios y 3 magnetómetros. La placa Arduino tiene dos funciones, la primera es recoger los datos del MPU6050 y estimar los ángulos de Euler; la segunda función es enviar los datos hacia la computadora. El software Visual Studio se utiliza para capturar los datos enviados a través de USB por el Arduino, además ordena y guarda los datos en un archivo .csv.

Un proceso de validación debe indicar numéricamente el grado aproximación entre el modelo y la realidad. En este trabajo se utiliza el coeficiente de determinación R^2 el cual toma valores entre 0 y 1 para explicar la correlación entre dos variables. Además se utiliza el RMSE (*Error Root Mean Squared*) el cual es una medida de uso frecuente para indicar la diferencia entre un modelo estimado y los valores realmente observados.

Figura 61 Banco de pruebas del cuadricóptero con 3 grados de libertad. Fuente: Elaboración propia.

En las Figura 62, Figura 63 y Figura 64 se muestran gráficas comparativas de los tres ángulos controlados entre datos experimentales (en color azul) y datos de simulación (en color rojo). Como se observa en Figura 62, Figura 63 las curvas son bastante semejantes. En la Figura 64 se observa una mayor diferencia entre los datos, los cuales se podría explicar debido a que el banco de pruebas posee un rozamiento el cual es considerable recordando el bajo

coeficiente de arrastre d. Además el centro de gravedad del cuadricóptero no coincide con el centro de giro del mismo cuando se encuentra empotrado en el banco de pruebas.

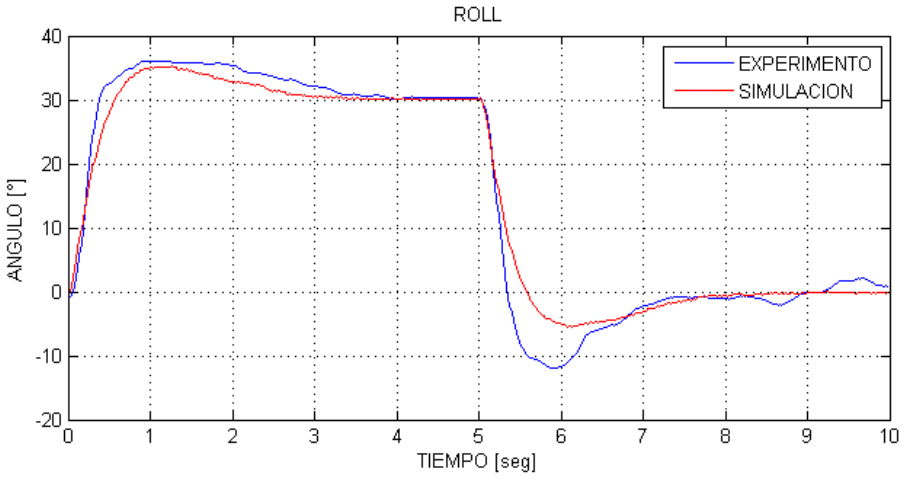

Figura 62 Comparación de datos experimentales y simulación del ángulo *roll*. Fuente: Elaboración propia.

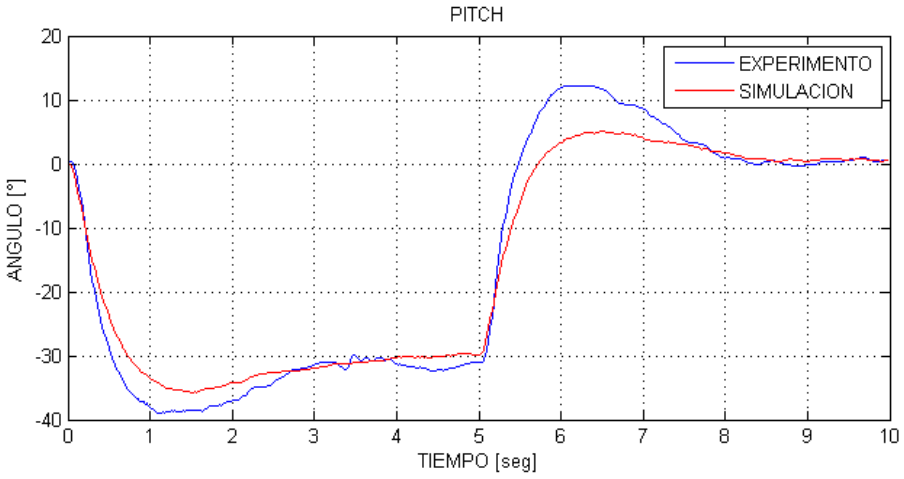

Figura 63 Comparación de datos experimentales y simulación del ángulo *pitch*. Fuente: Elaboración propia.

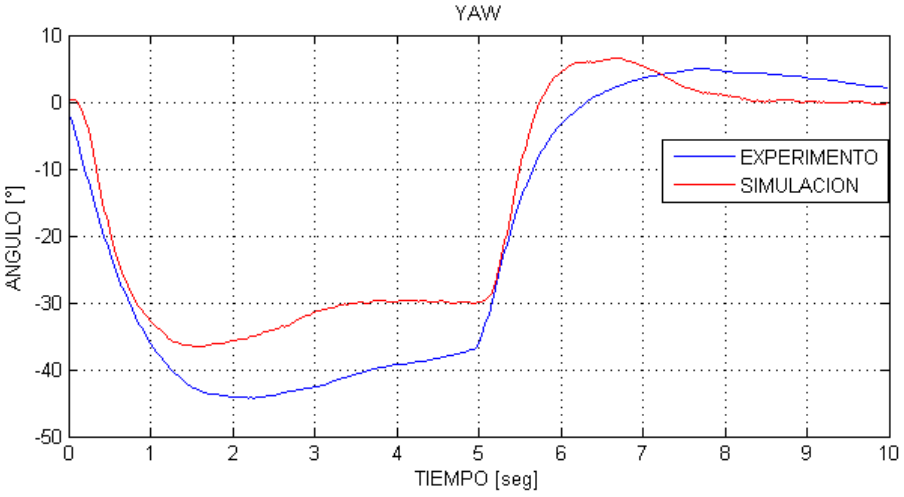

Figura 64 Comparación de datos experimentales y simulación del ángulo *yaw*. Fuente: Elaboración propia.

En la Tabla 15 se muestra un resumen de los parámetros de validación utilizados. En general obtienen valores satisfactorios, aunque menos favorables para el "*yaw*". Sin embargo se concluye que el modelo esta validado

Tabla 15 Cuadro resumen de la validación del modelo

Roll		Pitch		Yaw	
R^2	$RMSE$	R^2	$RMSE$	R^2	$RMSE$
0.9652	3.37	0.9662	3.44	0.8847	6.67

4.2 Programas y Código

Este proyecto se ha basado en el código Multiwii 2.4. Dicho código tiene diversas ventajas, entre estas se tiene que se actualiza aproximadamente 2 veces al año, además de ser compatible con diversos sistemas embebidos como Cruis y Multiwii Pro. Además en el mercado existen nuevas tarjetas de control como Naze32 las cuales tienen su propio código basados en Multiwii.

Los códigos de Multiwii se encuentran desarrollados en Arduino, por lo tanto para editar y grabar nuestro controlador es necesario instalar Arduino 1.6.3 o la última versión actualizada, ver Figura 65.

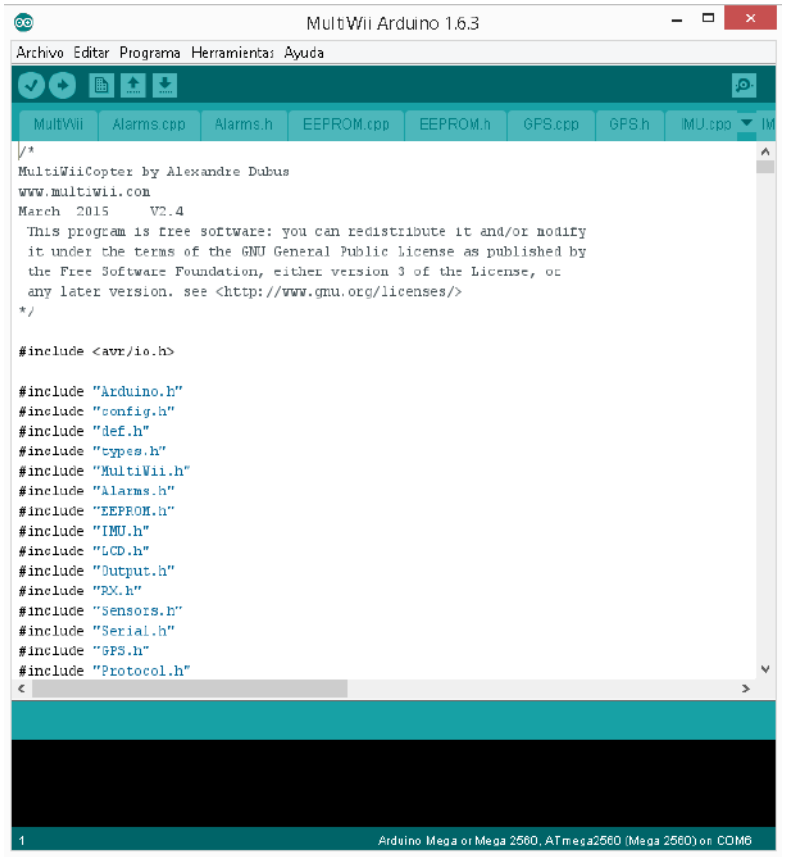

Figura 65 Código base de Multiwii en Arduino. . Fuente: Elaboración propia.

Para facilitar la activación y desactivación de funciones del UAV, Multiwii ha desarrollado una GUI en *Processing*. Esta GUI además de activar rutinas, permite visualizar algunas

variables como son los ángulos, velocidades angulares, magnetómetros, altura, calibrar sensores, entre otros.

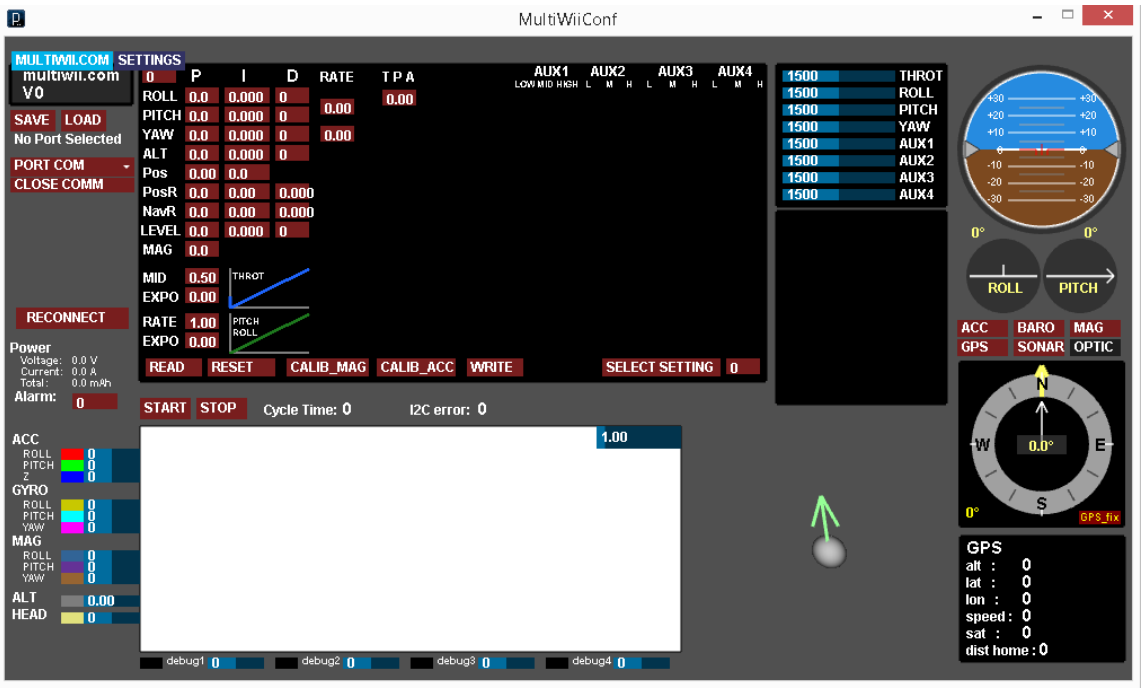

Figura 66 GUI desarrollada en *Processing* por Multiwii. . Fuente: Elaboración propia.

Los cálculos matemáticos de punto flotante son mucha más lentos que los del tipo de números enteros, por lo que debe su eso si es posible [40]. Para evitar tiempos de muestro elevados producidos por operaciones numéricas con puntos flotante, se evita ingresar valores decimales en las constantes de los controladores PID. Por dicha razón, se generan nuevas variables P_8, I_8, D_8, D_{28} generadas a partir de K, I, D, D_2 y multiplicadas por 10, 100 o 1000 según sea el caso, como se muestra entre las ecuaciones (155) y (166)

$$P_{8_{Roll}} = K_{Roll} * 10 \tag{155}$$

$$I_{8_{Roll}} = I_{Roll} * 10 \tag{156}$$

$$D_{8_{Roll}} = D_{Roll} * 10 \tag{157}$$

$$D_{28_{Roll}} = D_{2_{Roll}} * 1000 \tag{158}$$

$$P_{8_{Pitch}} = K_{Pitch} * 10 \tag{159}$$

$$I_{8_{Pitch}} = I_{Pitch} * 10 \tag{160}$$

$$D_{8_{Pitch}} = D_{Pitch} * 10 \tag{161}$$

$$D_{28_{Pitch}} = D_{2_{Pitch}} * 1000 \tag{162}$$

$$P_{8_{Yaw}} = K_{Yaw} * 100 \tag{163}$$

$$I_{8_{Yaw}} = I_{Yaw} * 100 \tag{164}$$

$$D_{8_{Yaw}} = D_{Yaw} * 1000 \tag{165}$$

$$D_{28_{Yaw}} = D_{2_{Yaw}} * 1000 \tag{166}$$

Las ecuaciones (155) hasta (166) se pueden ver implementadas en la

```
conf.pid[ROLL].P8 = constrain(62,0,80);    // Multiplicado por 10
conf.pid[ROLL].I8 =  53;                   // Multiplicado por 10
conf.pid[ROLL].D8 = 20;                    // Multiplicado por 10
D28[ROLL] = 85;                            // Multiplicado por 1000

conf.pid[PITCH].P8 = constrain(66,0,80);   // Multiplicado por 10
conf.pid[PITCH].I8 = 47;                   // Multiplicado por 10
conf.pid[PITCH].D8 = 27;                   // Multiplicado por 10
D28[PITCH] = 103;                          // Multiplicado por 1000

conf.pid[YAW].P8 = 26;                     // P8[2] = P_Yaw*100
conf.pid[YAW].I8 = 21;                     // I8[2] = I_Yaw*100
conf.pid[YAW].D8 = 150;                    // D8[2] = D_Yaw*1000
D28[YAW] = 0;                              // D28[2] = D2_Yaw*100*1000
```

Figura 67 Asignación de valores del controlador PID2. Fuente: Elaboración propia.

La parte principal de este proyecto es el desarrollo de un controlador PID para un cuadricóptero, en la Figura 68 se muestra la sección de código donde se implementa la lógica de control de la Figura 53 b). En la Figura 67 se muestra que las ganancias del controlador han cambiado en diferentes magnitudes; teniendo en cuenta que las salidas de la parte proporcional, integral, derivativa y segunda derivativa se deben sumar para conformar finalmente la salida del controlador como se muestra en Figura 53 b), sus unidades deben ser las mismas. En el código mostrado en la Figura 68 se re-escalan los valores numéricos $PTerm$, $ITerm$, $DTerm$ y D_2Term antes de ser sumados en la salida del controlador $AxisPID$.

Se decide trabajar en términos separados para resolver la ley del controlador llamados $PTerm$, $ITerm$, $DTerm$ y D_2Term debido a que la implementación de una sola ecuación recursiva con valores elevados para eliminar el punto flotante genera problemas. La suma y multiplicación de números grandes arroja como resultados valores numéricos mayores a 2^{15} (máximo valor permitido para una variable de 16 bits).

Otro dato importante que se puede observar de la Figura 68, es el número 2.4414. Esto se debe a que las unidades del giroscopio no se muestran en rad/seg como se explicó en la sección 3.3.

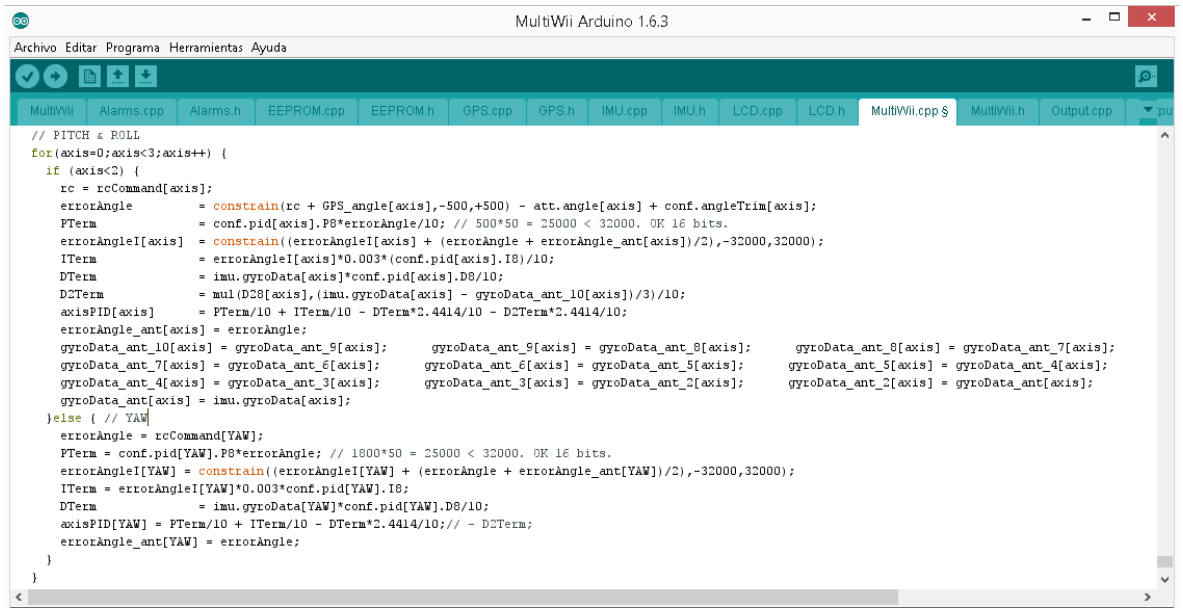

Figura 68 Código desarrollado en Arduino del controlador PID2. . Fuente: Elaboración propia.

Los controladores PID son diseñados para trabajar con referencias de ángulos en radianes, sin embargo, en los sistemas Arduino se trabaja con grados sexagesimales multiplicados por 10 para evitar el punto flotante. El cambio de unidades debe tomarse en cuenta, ya que este cambio trabaja como una ganancia proporcional multiplicativa a todo el PID, provocando indirectamente cambio en los parámetros del controlador. En este punto, las variables $axisPID_{Roll}$ y $axisPID_{Yaw}$ están aumentadas en $57.296 = \frac{180}{pi}$, mientras que $axisPID_{Yaw}$ está aumentada en $5729.6 = \frac{180}{pi} * 100$.

Para evitar confusión en el cambio de unidades, se aprovecha la matriz de movimientos inversos para ordenar las unidades. Recordemos que un cuadricóptero posee dos configuraciones (cruz y equis), como se muestra en (49) y (91). Los coeficientes de empuje y arrastre son respectivamente $b = 160/10^6[N.s^2]$ y $d = 2.767 * 10^- 6[N.m.s^2]$ respectivamente.

Los elementos de las matrices de las ecuaciones (49) y (91) tienen la forma $1/4b$, $1/2b$ y $1/4d$. Si se reemplaza el valor de las variables según corresponde, se obtienen los resultados mostrados en la segunda columna de la Tabla 16. La principal complicación de estos resultados radica en que son valores elevados los cuales serán multiplicados por la salida de los controladores $axisPID$, pudiendo obtener mayores a 2^{15}.

El escalamiento consiste en dividir los elementos de las matrices por 57.296 y adicionalmente por 10 para asegurar valores menores a 2^{15}. Las unidades de $axisPID_{Yaw}$ están amentadas en 100 veces con respecto a $axisPID_{Roll}$ y $axisPID_{Pitch}$. Los únicos elementos de la Matriz de Movimientos Inversos que interactúa con $axisPID_{Pitch}$ es el $1/4d$ por lo tanto, adicionalmente se debe dividir $1/4d$ por 100. Los resultados del escalamiento se muestran en la Tabla 16. Es importante recalcar que las unidades a la salida de las Matrices de Movimientos Inversos se expresan en rad^2/s^2 divididas por 10 como se muestra en la ecuación (167). Más adelante debe ser compensado la reducción entre 10. En la Figura 69 se muestra el código para implementar en Arduino la Matriz de Movimientos Inversos con sus elementos escalados.

$$\Omega'^2 = \Omega^2/10 \tag{167}$$

Tabla 16 Escalamiento de los elementos de las matrices de movimientos inversos

Elemento	Valor	Valor escalado
$1/4b$	1562.5	2.7271
$1/2b$	3125	5.4542
$1/4d$	90350.6	1.5769

```
#elif defined( QUADP ) //Nuestro
  motor[0] = PIDMIX( 0,+5.4542,-1.5770); //REAR
  motor[1] = PIDMIX(-5.4542, 0,+1.5770); //RIGHT
  motor[2] = PIDMIX(+5.4542, 0,+1.5770); //LEFT
  motor[3] = PIDMIX( 0,-5.4542,-1.5770); //FRONT
#elif defined( QUADX ) //Nuestro
  motor[0] = PIDMIX(-2.7271,+2.7271,-1.5770); //REAR_R
  motor[1] = PIDMIX(-2.7271,-2.7271,+1.5770); //FRONT_R
  motor[2] = PIDMIX(+2.7271,+2.7271,+1.5770); //REAR_L
  motor[3] = PIDMIX(+2.7271,-2.7271,-1.5770); //FRONT_L
```

Figura 69 Implementación la Matriz de Movimientos Inversos en Arduino. Fuente: Elaboración propia.

Otro cambio necesario para la correcta implementación se debe realizar en la ecuación (87), la cual puede escribirse como se muestran en (168).

$$PWM = 0.0018 * \Omega^2 + 40.71 \tag{168}$$

Multiwii acepta valores de *duty cycle* comprendidos entre 0 a 2048 los cuales corresponden al rango de 0% a 100%. La ecuación (87) fue diseñada para *duty cycle* en el rango de 0 a 100 unidades. Se realiza la conversión multiplicando por $20.48 = 2048/100$ y se obtiene la ecuación (169).

$$PWM = 0.0362 * \Omega^2 + 833.64 \tag{169}$$

A la salida de la Matriz de Movimientos Inversos se dispone de Ω'^2. Se reemplaza la ecuación (168) en (169) y se obtiene (170).

$$PWM = 0.362 * \Omega'^2 + 834 \tag{170}$$

La ecuación (170) se escribe en nuestro código. Todos los códigos desarrollados en este proyecto de tesis se encuentran en el Anexo F, Anexo G y Anexo H. Por ultimo para verificar nuestros resultados en la realidad, se realizan pruebas en el banco de pruebas, ver Figura 70.

Figura 70 Pruebas en laboratorio. Fuente: Elaboración propia.

En las pruebas realizadas se observó que el cuadricóptero a pesar de iniciar en condiciones desfavorables, es capaz de sobre ponerse t controlar su orientación, demás se representaron disturbios en forma de fuerza humana y se observó que el cuadricóptero retorna a sus valores consignados. Finalmente se realiza una prueba de campo para verificar los buenos resultados como se observa en la Figura 71.

Figura 71 Prueba final de vuelo en el Campus de la Universidad de Piura. Fuente: Elaboración propia.

.

Capítulo 5
Sistema Cuadricóptero-Carga suspendida

Volar con una carga suspendida es, en general, todo un desafío y en ocasiones puede ser peligroso ya que la carga puede variar significativamente las condiciones de vuelo de la aeronave. Por tanto se debe estudiar cuidadosamente la estabilidad del sistema aeronave-carga suspendida y dotar a la aeronave de la capacidad de adaptarse ante los cambios que se puedan producir, de manera que se pueda controlar la dinámica de la carga suspendida y reducir en la medida de lo posible el balanceo de la misma.

El problema es separado en tres etapas. En la primera de ellas será necesaria la localización del objeto, su identificación y seguimiento para conocer en todo momento su posición y en función de ella, conocer los esfuerzos que la carga suspendida ejercerá sobre el cuadricóptero.

En una segunda etapa, se debe considerar que en ciertas ocasiones no será posible la detección del objeto, ya que éste puede encontrarse fuera del campo de visión de la cámara instalada a bordo y, por tanto, será necesario recurrir a métodos de estimación capaces de realizar una predicción acertada de la posición de la carga.

La tercera etapa consiste en realizar un controlador que permita reducir las oscilaciones de la carga suspendida a partir de los esfuerzos que se generen por el sistema. Todas las etapas son fundamentales al instante de resolver el problema, ya que cualquier fallo en alguna de las etapas podría poner en riesgo tanto a la aeronave como a la carga que transporte, provocando que la misión sea un fracaso.

Se debe considerar que la detección y seguimiento de la carga suspendida (resolviendo para ello el problema de visión artificial) se ha desarrollado en el proyecto de fin de carrera “Detección, seguimiento y estimación del centro de masas de un objeto suspendido de un

cuadricóptero” en la Universidad de Sevilla por Jesús Cárdenas Egea y supervisado por el Dr. Manuel Vargas Villanueva.

Por tanto la investigación realizada en la Universidad de Sevilla se centra en la segunda etapa, donde se abarca la realización de un modelo válido del cuadricóptero que permita conocer la dinámica del conjunto aeronave-carga suspendida, el estudio y la aplicación de técnicas óptimas de estimación (para obtener, en primer lugar, la posición de la carga suspendida aun cuando sea imposible medirla a través de la cámara instalada y, en segundo lugar, eliminar el ruido que produce el sistema y aparato de medida) y finalmente, el desarrollo de un sencillo experimento que permita validar de un modo real los modelos matemáticos planteados.

5.1 Cálculo del ángulo de carga a partir de la posición centro de masas

El algoritmo de visión artificial permitirá determinar el centro de gravedad de la carga en una imagen, éstas coordenadas de serán llamadas X_{img} y Y_{img} y se miden en pixeles, los cuales tienen como origen en una imagen la parte superior izquierda.

Una de las cuestiones que queda por resolver es la obtención del ángulo de la carga suspendida. Si bien se tiene su posición localizada en la imagen mediante algoritmos de visión artificial, es necesario establecer una relación entre la posición en píxeles en la imagen (X_{img}, Y_{img}) y la posición real de la carga (X, Y, Z). Esta relación pasa por las propiedades de la cámara y algunas relaciones geométricas, como se va a demostrar. Se considera que la carga puede moverse en todas las direcciones del espacio. En la Figura 72 se puede observar las relaciones geométricas espaciales para poder determinar la posición (X, Y, Z) de la carga suspendida.

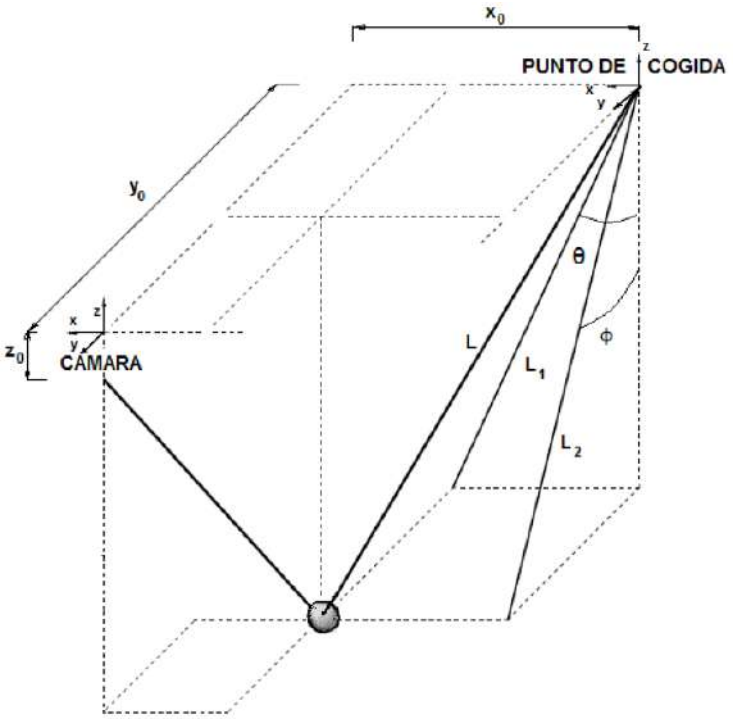

Figura 72 Representación geométrica de la carga suspendida y la cámara [41].

Para determinar la posición en el espacio de la carga se intersectan dos lugares geométricos. El primer lugar geométrico corresponde a la recta conformada por los posibles puntos en el espacio donde se encuentra la carga. Esto debido a que al trabajar con una sola cámara no se dispone con información de profundidad. El segundo lugar geométrico corresponde a la esfera cuyo radio es igual a la longitud del cable de la carga suspendida y con centro en el punto de cogida. Al intersectar ambos lugares geométricos se encuentra el punto estimado de carga suspendida.

Para encontrar la ecuación de la recta es necesario determinar algunas características internas de la cámara como la distancia focal F, el número de píxeles por milímetro en la dirección x e y, denominadas como S_x y S_y respectivamente. Además de ubicar el centro de la imagen en pixeles en la dirección x e y, denominas como C_X y C_y respectivamente. Dado un plano de la imagen como se muestra en la se puede mostrar la relación entre los pixeles de imagen y las coordenadas del espacio en la ecuación (171) y (172).

$$x_{imagen} = F_x * \frac{X}{Z} + C_x \tag{171}$$

$$y_{imagen} = F_y * \frac{Y}{Z} + C_y \tag{172}$$

Donde las variables F_x y F_y se muestra en (173).

$$F_x = F * S_X$$
$$F_y = F * S_y \tag{173}$$

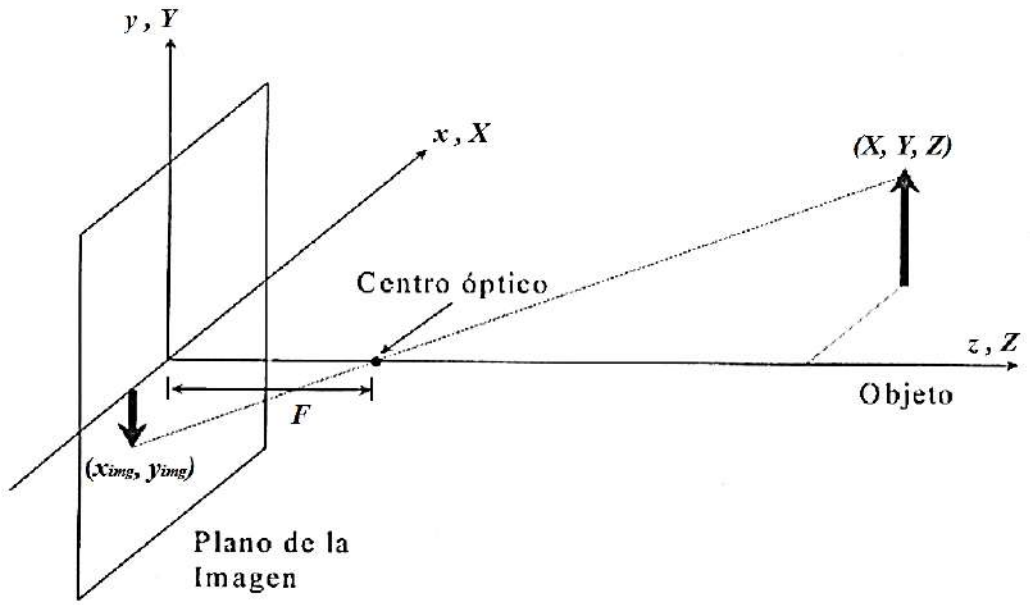

Figura 73 Modelo de formación de imágenes. Fuente: Elaboración propia.

Por lo tanto se puede despejar la ecuación de la recta:

$$X = Z * \frac{X_{imagen} - Cx}{F_x}$$

$$Y = Z * \frac{Y_{imagen} - Cy}{F_y} \tag{174}$$

$$Z = Z$$

La recta representada en la ecuación (174) se debe intersectar con la esfera de radio L y con centro en X_0, Y_0 y Z_0 respecto al sistema de referencia ubicado en el centro óptico. Finalmente las coordenadas de la carga pueden encontrarse mediante la ecuación (175):

$$Z = \frac{\left(-b + sqrt(b.^2 - 4 * a * c)\right)}{2 * a}$$

$$X = Z.* \frac{Ximg - Cx}{Fx} \tag{175}$$

$$Y = Z.* \frac{Yimg - Cy}{Fy}$$

Donde las variables a, b, y c:

$$a = \frac{\frac{(Cx - Ximg).^2}{Fx^2} + (Cy - Yimg).^2}{Fy^2} + 1$$

$$b = \frac{2 * X_0 * (Cx - Ximg)}{Fx} - 2 * Z_0 + \frac{2 * Y_0 * (Cy - Yimg)}{Fy} \tag{176}$$

$$c = -(L)^2 + {X_0}^2 + {Y_0}^2 + {Z_0}^2$$

Aplicando trigonometría, se encuentran las variables objetivos de este proyecto.

$$\phi_L = atan\left(.\frac{(Y_0 - Y)}{-Z_0 + Z}\right)$$

$$\theta_L = atan\left(\frac{(X_0 - X)}{-Z_0 + Z}\right) \tag{177}$$

5.2 Dinámica del movimiento tridimensional

Para analizar y modelar la dinámica del sistema cuadricóptero-carga suspendida se procede a proponer un modelo. Al igual que en la sección anterior, se comenzará por mostrar un dibujo que ilustre la situación, los sistemas de referencia y las magnitudes que se utilizarán. De esta manera, el modelo tridimensional toma la forma mostrada en la Figura 74.

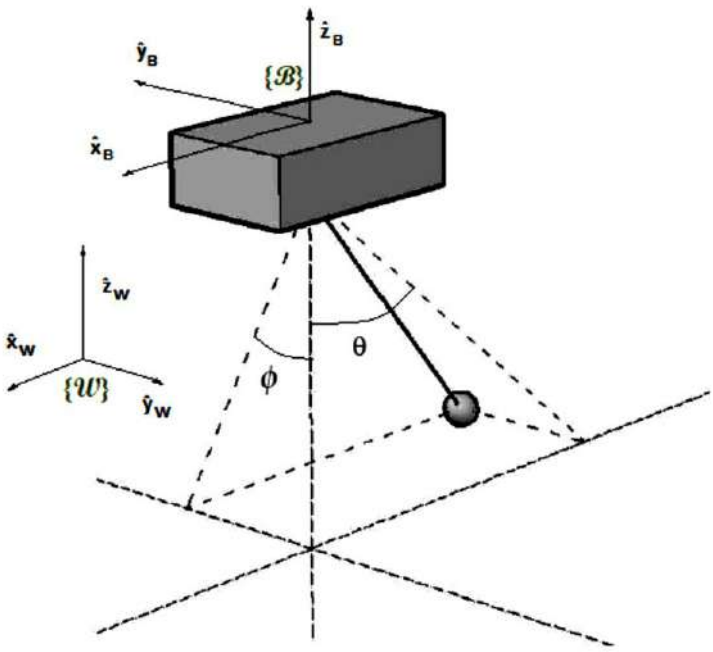

Figura 74 Problema tridimensional [41]

Como se puede observar, se ha tomado un sistema de referencia solidario al cuadricóptero, formado por los ejes x_B, y_B, z_B; coincidiendo el origen de coordenadas con el centro de gravedad de la aeronave. Se considera un cable rígido e inextensible que se extiende desde dicho centro de gravedad hasta la carga suspendida, donde la posición de ésta puede fijarse mediante los ángulos θ_L y ϕ_L.

Por tanto, la carga suspendida no está restringida a un plano y puede moverse en las tres direcciones del espacio. Algo parecido ocurre con el cuadricóptero. En este caso, aparte de tener tres grados de libertad en las tres direcciones del espacio, posee dos de los tres grados de libertad existentes para el giro, ya que el giro en guiñada no influirá en la posición de la carga suspendida ni provocará ningún esfuerzo significativo en el análisis que sigue a continuación.

Las hipótesis que se consideran, son las siguientes [41]:

• Se considera una fricción en el punto de cogida no despreciable.

• Cable que sujeta la carga es rígido e inextensible.

• El centro de gravedad del cuadricóptero y el punto de cogida coinciden.

• Atmósfera en calma (no existe viento).

• Se considera que el giro en guiñada de la aeronave no afecta a la dinámica de la carga suspendida.

En primer lugar, es conveniente hacer las siguientes definiciones:

• Vector ${}^{W}_{B}\xi$: también definido en ocasiones simplemente como ξ_B, es el vector que recoge la posición horizontal y vertical del centro de gravedad del cuadricóptero respecto a un sistema de referencia inercial.

• Vector ${}^{W}_{B}V$: vector que recoge la velocidad horizontal y vertical del centro de gravedad del cuadricóptero respecto a un sistema de referencia inercial.

• Vector ${}^{W}_{B}\dot{V}$: vector que recoge la aceleración horizontal y vertical del centro de gravedad del cuadricóptero respecto a un sistema de referencia inercial.

• Vector ${}^{W}_{L}\xi$: vector que recoge la posición horizontal y vertical del centro de gravedad de la carga suspendida respecto a un sistema de referencia inercial.

• Vector ${}^{W}_{L}V$: vector que recoge la velocidad horizontal y vertical del centro de gravedad de la carga suspendida respecto a un sistema de referencia inercial.

• Vector ${}^{W}_{L}\dot{V}$: vector que recoge la aceleración horizontal y vertical del centro de gravedad de la carga suspendida respecto a un sistema de referencia inercial.

• Vector ${}^{W}_{B}n$: es el vector que recoge los ángulos de balanceo, cabeceo y guiñada del cuadricóptero.

• Vector ${}^{L}_{B}\xi$: vector que recoge la posición del centro de gravedad del cuadricóptero respecto a un sistema de referencia no inercial solidario con la carga suspendida.

Por tanto según las definiciones anteriores, e incluyendo las variables q se tiene:

$$\boldsymbol{q} = \begin{bmatrix} {}^{W}_{B}\xi \\ {}^{W}_{B}n \\ \phi_L \\ \theta_L \end{bmatrix} \tag{178}$$

Y según se ha definido [41]:

$${}^{W}_{B}\xi = \begin{bmatrix} x_B \\ y_B \\ z_B \end{bmatrix} \tag{179}$$

$${}^{W}_{B}n = \begin{bmatrix} \phi_B \\ \theta_B \\ \psi_B \end{bmatrix} \tag{180}$$

Donde ϕ_B, θ_B y ψ_B son los ángulos de balanceo, cabeceo y guiñada respectivamente. [41]

$${}^{L}_{B}\xi = \begin{bmatrix} 0 \\ 0 \\ L \end{bmatrix} \tag{181}$$

$$\boldsymbol{g} = \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \tag{182}$$

Las matrices de giro se definen según [41]:

$$ {}^{W}_{L}R_{\theta} = \begin{bmatrix} cos\theta_L & 0 & sin\theta_L \\ 0 & 1 & 0 \\ -sin\theta_L & 0 & cos\theta_L \end{bmatrix} \tag{183}$$

$$ {}^{W}_{L}R_{\phi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi_L & -sin\phi_L \\ 0 & sin\phi_L & cos\phi_L \end{bmatrix} \tag{184}$$

Por otro lado, las matrices de ayuda para el cálculo de las derivadas se definen como:

$$A_{\phi} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix};\ A_{\theta} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix} \tag{185}$$

Según se ha definido la posición, velocidad y aceleración de la carga suspendida, se obtiene que [41]:

$$ {}^{W}_{L}\xi = {}^{W}_{B}\xi - {}^{W}_{L}R_{\theta}\,{}^{W}_{L}R_{\phi}\,{}^{L}_{B}\xi \tag{186}$$

$$ {}^{W}_{L}v = {}^{W}_{B}v - {}^{W}_{L}R_{\theta}A_{\theta}\,{}^{W}_{L}R_{\phi}\,{}^{L}_{B}\xi\dot{\theta}_L - {}^{W}_{L}R_{\theta}\,{}^{W}_{L}R_{\phi}A_{\phi}\,{}^{L}_{B}\xi\dot{\phi}_L \tag{187}$$

$$\begin{aligned} {}^{W}_{L}\dot{v} = {}^{W}_{B}\dot{v} &- {}^{W}_{L}R_{\theta}A_{\theta}{}^{2}\,{}^{W}_{L}R_{\phi}\,{}^{L}_{B}\xi\dot{\theta}_L{}^{2} - 2\,{}^{W}_{L}R_{\theta}A_{\theta}\,{}^{W}_{L}R_{\phi}A_{\phi}\,{}^{L}_{B}\xi\dot{\theta}_L\dot{\phi}_L \\ &- {}^{W}_{L}R_{\theta}\,{}^{W}_{L}R_{\phi}A_{\phi}{}^{2}\,{}^{L}_{B}\xi\dot{\phi}_L{}^{2} - {}^{W}_{L}R_{\theta}A_{\theta}\,{}^{W}_{L}R_{\phi}\,{}^{L}_{B}\xi\ddot{\theta}_L - {}^{W}_{L}R_{\theta}\,{}^{W}_{L}R_{\phi}A_{\phi}\,{}^{L}_{B}\xi\ddot{\phi}_L \end{aligned} \tag{188}$$

Por tanto, ya es posible hacer el cálculo de la energía cinética y potencial a partir de las ecuaciones (186), (187) y (188).

La energía cinética de la carga se puede expresar de la siguiente manera:

$$H = \frac{1}{2}\,{}^{W}_{L}v^{T}m_L\,{}^{W}_{L}v \tag{189}$$

Y su energía potencial:

$$U = m_L\,{}^{W}_{L}\xi^{T}g \tag{190}$$

Por tanto, para obtener las ecuaciones de la dinámica del sistema hay que resolver las ecuaciones de Lagrange (192) para las coordenadas generalizadas $\boldsymbol{q}$:

$$\mathcal{L} = H - U \tag{191}$$

$$\frac{d}{dt}\left(\frac{\partial\mathcal{L}}{\partial\dot{\boldsymbol{q}}}\right) - \frac{\partial\mathcal{L}}{\partial\boldsymbol{q}} = \boldsymbol{\tau} - \boldsymbol{\tau}^{f} - \boldsymbol{\tau}^{a} \tag{192}$$

Escrito en formato matricial, el sistema de ecuaciones queda como se muestra en las ecuaciones (193) hasta (196) [41]:

$$\boldsymbol{M}(\boldsymbol{q}) = \begin{bmatrix} -\left({}^{W}_{L}\boldsymbol{R}_{\theta}\,{}^{W}_{L}\boldsymbol{R}_{\phi}\boldsymbol{A}_{\phi}\,{}^{L}_{B}\xi\right)^{T} m_L & \boldsymbol{0}_{1x3} & \left({}^{W}_{L}\boldsymbol{R}_{\theta}\,{}^{W}_{L}\boldsymbol{R}_{\phi}\boldsymbol{A}_{\phi}\,{}^{L}_{B}\xi\right)^{T} m_L\left({}^{W}_{L}\boldsymbol{R}_{\theta}\,{}^{W}_{L}\boldsymbol{R}_{\phi}\boldsymbol{A}_{\phi}\,{}^{L}_{B}\xi\right) & \left({}^{W}_{L}\boldsymbol{R}_{\theta}\,{}^{W}_{L}\boldsymbol{R}_{\phi}\boldsymbol{A}_{\phi}\,{}^{L}_{B}\xi\right)^{T} m_L\left({}^{W}_{L}\boldsymbol{R}_{\theta}\boldsymbol{A}_{\theta}\,{}^{W}_{L}\boldsymbol{R}_{\phi}\,{}^{L}_{B}\xi\right) \\ -\left({}^{W}_{L}\boldsymbol{R}_{\theta}\boldsymbol{A}_{\theta}\,{}^{W}_{L}\boldsymbol{R}_{\phi}\,{}^{L}_{B}\xi\right)^{T} m_L & \boldsymbol{0}_{1x3} & \left({}^{W}_{L}\boldsymbol{R}_{\theta}\boldsymbol{A}_{\theta}\,{}^{W}_{L}\boldsymbol{R}_{\phi}\,{}^{L}_{B}\xi\right)^{T} m_L\left({}^{W}_{L}\boldsymbol{R}_{\theta}\,{}^{W}_{L}\boldsymbol{R}_{\phi}\boldsymbol{A}_{\phi}\,{}^{L}_{B}\xi\right) & \left({}^{W}_{L}\boldsymbol{R}_{\theta}\boldsymbol{A}_{\theta}\,{}^{W}_{L}\boldsymbol{R}_{\phi}\,{}^{L}_{B}\xi\right)^{T} m_L\,{}^{W}_{L}\boldsymbol{R}_{\theta}\boldsymbol{A}_{\theta}\,{}^{W}_{L}\boldsymbol{R}_{\phi}\,{}^{L}_{B}\xi \end{bmatrix} \tag{193}$$

$$\boldsymbol{V}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \begin{bmatrix} \boldsymbol{0}_{1x3} & \boldsymbol{0}_{1x3} & \left({}^{W}_{L}\boldsymbol{R}_{\theta}\,{}^{W}_{L}\boldsymbol{R}_{\phi}\boldsymbol{A}_{\phi}\,{}^{L}_{B}\xi\right)^{T} m_L\left(2\,{}^{W}_{L}\boldsymbol{R}_{\theta}\boldsymbol{A}_{\theta}\,{}^{W}_{L}\boldsymbol{R}_{\phi}\boldsymbol{A}_{\phi}\,{}^{L}_{B}\xi\dot{\theta}_L + {}^{W}_{L}\boldsymbol{R}_{\theta}\,{}^{W}_{L}\boldsymbol{R}_{\phi}\boldsymbol{A}_{\phi}{}^{2}\,{}^{L}_{B}\xi\dot{\phi}_L\right) & \left({}^{W}_{L}\boldsymbol{R}_{\theta}\,{}^{W}_{L}\boldsymbol{R}_{\phi}\boldsymbol{A}_{\phi}\,{}^{L}_{B}\xi\right)^{T} m_L\left({}^{W}_{L}\boldsymbol{R}_{\theta}\boldsymbol{A}_{\theta}{}^{2}\,{}^{W}_{L}\boldsymbol{R}_{\phi}\,{}^{L}_{B}\xi\dot{\theta}_L\right) \\ \boldsymbol{0}_{1x3} & \boldsymbol{0}_{1x3} & \left({}^{W}_{L}\boldsymbol{R}_{\theta}\boldsymbol{A}_{\theta}\,{}^{W}_{L}\boldsymbol{R}_{\phi}\,{}^{L}_{B}\xi\right)^{T} m_L\left({}^{W}_{L}\boldsymbol{R}_{\theta}\,{}^{W}_{L}\boldsymbol{R}_{\phi}\boldsymbol{A}_{\phi}{}^{2}\,{}^{L}_{B}\xi\dot{\phi}_L\right) & \left({}^{W}_{L}\boldsymbol{R}_{\theta}\boldsymbol{A}_{\theta}\,{}^{W}_{L}\boldsymbol{R}_{\phi}\,{}^{L}_{B}\xi\right)^{T} m_L\left({}^{W}_{L}\boldsymbol{R}_{\theta}\boldsymbol{A}_{\theta}{}^{2}\,{}^{W}_{L}\boldsymbol{R}_{\phi}\,{}^{L}_{B}\xi\dot{\theta}_L + 2\,{}^{W}_{L}\boldsymbol{R}_{\theta}\boldsymbol{A}_{\theta}\,{}^{W}_{L}\boldsymbol{R}_{\phi}\boldsymbol{A}_{\phi}\,{}^{L}_{B}\xi\dot{\phi}_L\right) \end{bmatrix} \tag{194}$$

$$\boldsymbol{G}(\boldsymbol{q}) = \begin{bmatrix} -m_L\left({}^{W}_{L}\boldsymbol{R}_{\theta}\,{}^{W}_{L}\boldsymbol{R}_{\phi}\boldsymbol{A}_{\phi}\,{}^{L}_{B}\xi\right)^{T}\boldsymbol{g} \\ -m_L\left({}^{W}_{L}\boldsymbol{R}_{\theta}\boldsymbol{A}_{\theta}\,{}^{W}_{L}\boldsymbol{R}_{\phi}\,{}^{L}_{B}\xi\right)^{T}\boldsymbol{g} \end{bmatrix} \tag{195}$$

Resultando [41]:

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{V}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{\tau} - \boldsymbol{\tau}^{f} - \boldsymbol{\tau}^{a} \tag{196}$$

5.3 Linealización del modelo tridimensional

A continuación se expone el modelo tridimensional que se utilizará para implementar el filtro de Kalman para la ampliación del experimento al caso tridimensional. Desarrollando estas ecuaciones y despreciando la resistencia aerodinámica se obtiene [41]:

$$\begin{aligned} B\left(\dot{\theta}_B - \dot{\theta}_L\right) = {} & m_L L^2\ddot{\theta}_L cos^2\phi_L - 2L^2 m_L cos\phi_L sin\phi_L\dot{\theta}_L\dot{\phi}_L + Lm_L g sin\theta_L cos\phi_L \\ & + Lm_L sin\theta_L cos\phi_L\ddot{z}_B - Lm_L cos\theta_L cos\phi_L\ddot{x}_B \end{aligned} \tag{197}$$

La ecuación (197) para la coordenada θ_L, y:

$$\begin{aligned} B\left(\dot{\phi}_B - \dot{\phi}_L\right) = {} & m_L L^2\ddot{\phi}_L + m_L L^2 cos\phi_L sin\phi_L\dot{\theta}_L^2 + Lm_L g cos\theta_L sin\phi_L \\ & + Lm_L cos\theta_L sin\phi_L\ddot{z}_B + Lm_L sin\theta_L sin\phi_L\ddot{x}_B + Lm_L cos\phi_L\ddot{y}_B \end{aligned} \tag{198}$$

La ecuación (198) para la coordenada ϕ_L.

Linealizando en torno al punto de equilibrio correspondiente al instante inicial en que la pelota se encuentra en reposo, es decir, $\theta_{L0} = 0$, $\dot{\theta}_{L0} = 0$, $\phi_{L0} = 0$, $\dot{\phi}_{L0} = 0$ y suponiendo condiciones iniciales nulas, se obtienen las siguientes matrices simplificadas:

$$\boldsymbol{x} = \begin{bmatrix} \dot{\phi}_L \\ \phi_L \\ \dot{\theta}_L \\ \dot{\theta}_L \end{bmatrix} \tag{199}$$

$$\boldsymbol{u} = \begin{bmatrix} \dot{\phi}_B \\ \dot{\theta}_B \\ \ddot{x}_B \\ \ddot{y}_B \\ \ddot{z}_B \end{bmatrix} \tag{200}$$

$$\boldsymbol{A} = \begin{bmatrix} -\frac{B}{m_L L^2} & -\frac{gLm_L}{m_L L^2} & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & -\frac{B}{m_L L^2} & -\frac{gLm_L}{m_L L^2} \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{201}$$

$$\boldsymbol{B} = \begin{bmatrix} \frac{B}{m_L L^2} & 0 & 0 & -\frac{Lm_L}{m_L L^2} & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{B}{m_L L^2} & \frac{Lm_L}{m_L L^2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{202}$$

$$\boldsymbol{C} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{203}$$

$$\boldsymbol{D} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{204}$$

Nótese que, al suponer dicho punto de equilibrio y condiciones iniciales nulas (se parte del reposo) las ecuaciones quedan bastante simplificadas. Adicionalmente, y como consecuencia de las hipótesis aplicadas, se tiene un sistema muy simple que está desacoplado en cada dirección.

Este modelo obtenido en forma de espacio de estados será, por tanto, el que se use en adelante y al que se le aplique el filtro de Kalman con el objetivo de ampliar el experimento a las tres dimensiones.

5.4 Algoritmo basado en el filtro de Kalman

El filtro de Kalman fue desarrollado por Rudolph Kalman sobre 1960, aunque Peter Swerling desarrollo un algoritmo muy similar en 1958. El filtro tomo el nombre que actualmente lleva porque Kalman publicó sus resultados en una revista más prestigiosa, siendo su trabajo más general y completo. Como en muchas de las nuevas tecnologías, el filtro de Kalman fue desarrollado para resolver un problema específico, en este caso, la navegación de la aeronave (*spacecraft*) del programa espacial Apollo.

Desde entonces, el filtro de Kalman ha tenido aplicación en cientos de áreas diversas, incluyendo todas las formas de navegación (aeroespacial, terrestre y marina), instrumentación de las plantas industriales de producción de energía, modelos demográficos, manufacturación, detección de radioactividad del terreno, etc.

Un buen algoritmo de filtrado puede eliminar el ruido provocado por las señales electromagnéticas mientras retiene la información útil. El filtro de Kalman es una herramienta con las que se puede estimar las variables en un amplio rango de procesos. El filtro de Kalman no trabaja sólo bien en la práctica, sino que es teóricamente atractivo ya que se puede demostrar que de todos los filtros posibles es el único que minimiza la varianza del error de estimación [41].

Nuestro sistema se modelará mediante las siguientes ecuaciones:

$$\boldsymbol{x}_k = \boldsymbol{A}\boldsymbol{x}_{k-1} + \boldsymbol{B}\boldsymbol{u}_{k-1} + \boldsymbol{w}_{k-1}; \quad \boldsymbol{w}_k \sim N(\boldsymbol{0}, Q_k) \tag{205}$$

$$\boldsymbol{y}_k = \boldsymbol{C}\boldsymbol{x}_k + \boldsymbol{v}_k; \quad \boldsymbol{v}_k \sim N(\boldsymbol{0}, R_k) \tag{206}$$

Donde, $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$ son matrices; k es el índice temporal; $\boldsymbol{x}$ es el vector de estados del sistema; $\boldsymbol{u}$ es la entrada del sistema; $\boldsymbol{y}$ es la salida que se mide; y $\mathbf{w}$ y $\boldsymbol{v}$ representan el ruido. La variable $\boldsymbol{w}$ es el ruido del proceso y $\boldsymbol{v}$ el ruido que se produce al medir la salida. El vector $\boldsymbol{x}$ contiene la información sobre el estado actual del sistema, pero es una información interna del sistema, por lo que no es posible medirlo directamente. En su lugar, medimos $\boldsymbol{y}$, la cual está relacionada con $\boldsymbol{x}$ aunque se ve afectada por el ruido $\boldsymbol{v}$. Por ello, podemos usar $\boldsymbol{y}$ como ayuda para obtener una estimación de $\boldsymbol{x}$, pero hay que ser precavidos, ya que su valor esta modificado por el ruido. Por ello, aunque se pueda usar esta información con cierta exactitud, no hay que depositar nuestra total confianza en ella.

Se asume que el valor medio del ruido $\boldsymbol{w}$ y el valor medio del ruido $\boldsymbol{v}$ son ambos iguales a cero y son ruido blanco. Además hay que asumir posteriormente el conocimiento de las varianzas y covarianzas del ruido. Estas matrices de covarianza del ruido se definen como, la covarianza del ruido del proceso, Q_k y la covarianza del ruido de la medida, R_k.

Existen muchas alternativas para formular las ecuaciones del filtro de Kalman, aunque una de las más comunes es la que sigue a continuación:

Siendo el modelo del sistema las ecuaciones mostradas en (205) y (206), se puede realizar una estimación del vector de estados [41]:

$$\hat{\boldsymbol{x}}_k^{-} = \boldsymbol{A}\hat{\boldsymbol{x}}_{k-1} + \boldsymbol{B}\boldsymbol{u}_{k-1} \tag{207}$$

$$\boldsymbol{P}_k^{-} = \boldsymbol{A}\boldsymbol{P}_{k-1}\boldsymbol{A}^T + \boldsymbol{Q}_{k-1} \tag{208}$$

Una vez realizada la estimación se puede corregir el vector de estados estimado:

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^{-} + \boldsymbol{K}_k[\boldsymbol{y}_k - \boldsymbol{C}\hat{\boldsymbol{x}}_k^{-}] \tag{209}$$

Y actualizar el error de la covarianza y la matriz de ganancia de Kalman:

$$\boldsymbol{P}_k = [\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{C}]\boldsymbol{P}_k^{-} \tag{210}$$

$$\boldsymbol{K}_k = \boldsymbol{P}_k^{-}\boldsymbol{C}^T[\boldsymbol{C}\boldsymbol{P}_k^{-}\boldsymbol{C}^T + \boldsymbol{R}_k]^{-1} \tag{211}$$

5.5 Experimentos

En esta sección se presenta brevemente los materiales, instrumentos y equipos utilizados en el experimento. Se presenta el modelo en espacio de estado con valores numéricos. Se demuestra la validación del modelo matemático a través de la comparación con los datos capturados en el experimento. Se presenta las buenas prestaciones del filtro de Kalman.

5.5.1 Descripción del material

El material empleado para este experimento viene detallado en la siguiente lista:

- Pelota de plástico de color naranja de tamaño similar a una pelota de ping-pong la cual actuará como objeto a detectar y seguir.

- Hilo de pesca. Hace la función de cable rígido que sujeta a la carga suspendida.
- Placa de plástico rectangular. Hace la función de cuadricóptero. Está acoplada al brazo del robot mediante unas piezas metálicas y a su vez lleva acoplada la cámara y la IMU. Por un orificio realizado en ella sale el hilo que sujeta a la carga suspendida.
- Cámara web Logitech C525.
- Brazo robótico RX90 de la compañía STÄUBLI UNIMATION.
- Unidad de medidas inerciales (IMU) MPU-6050.
- Microsoft Visual C++ 2010 utilizado para compilar y ejecutar el algoritmo de visión artificial.
- Matlab R2013, R2015 y Simulink. Se han utilizado para implementar los algoritmos de estimación y ejecutar simulaciones entre otras tareas.

Brazo robótico RX90

El brazo robótico RX90 se encuentra dentro de la categoría de Robots especializados definida por STÄUBLI UNIMATION para sus brazos robóticos. Este brazo permitirá realizar experimentos controlados además de permitir mantener la cámara web en dirección perpendicular al piso ya que no se cuenta con un sistema *gimbal* que garantice dicha dirección.

MPU-6050

El componente MPU-6050 es uno de los dispositivos más comunes que se pueden encontrar actualmente en el mercado. Capaz de proporcionar una salida digital en seis ejes, está diseñado para trabajar con baja potencia. Su bajo coste y alto rendimiento. Estos dispositivos combinan un giróscopo de 3 ejes y un acelerómetro de 3 ejes. Cuenta con una resolución de 16-bits, lo cual significa que divide el rango dinámico en 65536 fracciones, las cuales se aplican para cada eje. Además el MPU-6050 contiene un sensor de temperatura embebido.

Cámara web Logitech C525

El uso de la *webcam* Logitech C525 vino motivado por la necesidad de una cámara portátil para realizar los experimentos que a su vez tuviera una fácil instalación y cuyo precio fuera el menor posible. Entre las características de esta cámara se pueden encontrar las siguientes: Tipo de conexión USB 2.0, resolución óptica 1280x720 y enfoque automático

5.5.2 Calibración de la cámara

Para el algoritmo de visión es necesario determinar parámetros intrínsecos de la cámara como se observa en las ecuaciones (175) y (176). Para dicho propósito se utilizó la aplicación *cameraCalibrator*, la cual se encuentra disponible en Matlab R2015.

El procedimiento para utilizar la aplicación cameraCalibrator se encuentra muy bien explicada en http://www.mathworks.com/help/vision/ug/single-camera-calibrator-app.html. Sin embargo se debe tener en cuenta algunas anotaciones.

- El tablero debe ocupar aproximadamente la cuarta parte del área de la imagen.
- El tablero de tener diversos ángulos de inclinación.

- No se ha considerado el cálculo de la distorsión tangencial.
- No se ha considerado el cálculo de la desviación de ejes.
- Se ha considerado en el cálculo la distorsión radial pero no se ha corregido esta distorsión en las imágenes.
- Desactivar la función autofocus de la cámara.

En la Figura 75 se muestran las fotos utilizadas para la calibración de la cámara donde se observan las consideraciones tomadas sobre el ángulo de inclinación del patrón y el área abarcada.

Figura 75 Fotos utilizadas para la calibración de la webcam Logitech C525. Fuente: Elaboración propia.

Como resultado de la calibración se estimará la posición del centroide de la imagen C_X y C_Y y la distancia focal multiplicada por cantidad de pixeles por unidad de milímetro en la dirección X y Y, llamados F_X y F_Y respectivamente. Los resultados de la calibración se muestran en la Tabla 17

Tabla 17 Resultados de la calibración de la cámara Logitech C525

Parámetro	Símbolo	Valor	Unidades
Centro de la imagen en dirección X.	C_X	625.3	pixeles
Centro de la imagen en dirección Y.	C_Y	346.3	pixeles
Distancia focal en dirección X.	F_X	1091.3	pixeles
Distancia focal en dirección Y.	F_Y	1089.5	pixeles

Por defecto la cámara captura imágenes de 1280 por 720 pixeles haciendo un total de 921600 pixeles en cada captura. Pruebas realizadas arrojaron un tiempo de muestro promedio de 150 a 120 milisegundos. Se decidió reducir las dimensiones de la matriz de la imagen por medio de la función *cvSize* en Visual Studio. La matriz re-escalada es de 320 por 180 pixeles haciendo un total de 57600 pixeles. Gracias al re-escalamiento de la imagen captura, el algoritmo de visión artificial tiene un tiempo de muestreo promedio de 33 milisegundos. Estos cambios en el número de pixeles de la imagen capturada hace cambios en C_X, C_Y, F_X y F_Y. Si observamos con detenimiento, la matriz de datos de la imagen se ha reducido en la cuarta parte en dirección X e igualmente en la dirección Y, por tal motivo el muy fácil concertar en los valores de los parámetros de la Tabla 18, los cuales son la cuarta parte de los originales.

Tabla 18 Parámetros intrínsecos utilizados después del re-escalamiento.

Parámetro	Símbolo	Valor	Unidades
Centro de la imagen en dirección X.	C_X	156.3	pixeles
Centro de la imagen en dirección Y.	C_Y	86.6	pixeles
Distancia focal en dirección X.	F_X	272.8	pixeles
Distancia focal en dirección Y.	F_Y	272.3	pixeles

5.5.3 Validación del sistema

A continuación se presenta el procedimiento realizado para validar el modelo matemático presentado en las ecuaciones (197) y (198). En la Tabla 19 se presentan los valores de los parámetros utilizados en nuestro experimento.

Tabla 19 Parámetros del modelo matemático

Parámetro	Símbolo	Valores	Unidades
Parámetro de Fricción	B	0.00022	m^2 $* kg/s$
Longitud del cable	L	0.59	m
Masa de la carga suspendida	m_L	0.033	kg
Gravedad	g	9.81	m/s^2

La longitud del cable L corresponde a la longitud desde el punto de cogida del cable con el cuadricóptero hasta el centro de gravedad de la carga suspendida. La masa de la carga se ha calculado con una balanza de 1 gramo de precisión. La gravedad se ha asumido igual a 9.81 m/s^2 y el parámetro de fricción fue también asumido.

La validación del modelo se ha realizado en base a dos experimentos, el primer experimento consiste en un movimiento de la plataforma plástica que simula el cuadricóptero en sólo la dirección X y el segundo experimento consiste en un movimiento más complejo en el plano X y Y.

En ambos experimentos el procedimiento consiste en realizar un movimiento programado con el brazo robótico mientras que en paralelo va corriendo el algoritmo de visión artificial (Anexo B). El sensor IMU debe tener entre 15 a 30 segundos previos al inicio del experimento para correr su rutina de calibración que se ejecuta en el Arduino. El algoritmo de visión artificial desarrollando en Visual Studio escribe las aceleraciones proporcionadas por la IMU, las coordenadas en pixeles del centro de la carga suspendida y el tiempo en un archivo llamado `datos_camshift.csv` el cual se puede encontrar dentro de la carpeta desde donde se ejecuta el algoritmo.

Para nuestro análisis, los datos de aceleración no fueron filtrados utilizado el comando *idealfilter* ni cualquier otro filtro de Matlab. Utilizando las ecuaciones (175), (176) y (177) se logra determinar los ángulos de la carga ϕ_L y θ_L. Las condiciones geométricas del experimento se muestran en la Tabla 20. El algoritmo que implementa este procesamiento de datos se presenta en el Anexo para ejecutarse en Matlab.

Tabla 20 Parámetros Geométricos.

Parámetro	Símbolo	Valor	Unidades
Punto de cogida de la carga en dirección X	X_0	0	mm
Punto de cogida de la carga en dirección Y	Y_0	45	mm
Punto de cogida de la carga en dirección Z	Z_0	-20	mm

Los ángulos ϕ_L y θ_L provenientes del experimento gracias al algoritmo de visión artificial son comparados con el modelo expresado en las ecuaciones (197) y (198) las cuales son implementadas en Simulink en el archivo Carga_Suspendida_3D.slx. En la Figura 76 se puede observar en color negro el ángulo "*pitch*" de la carga suspendida según el algoritmo de visión artificial, en color azul el mismo ángulo según el modelo matemático construido en Simulink.

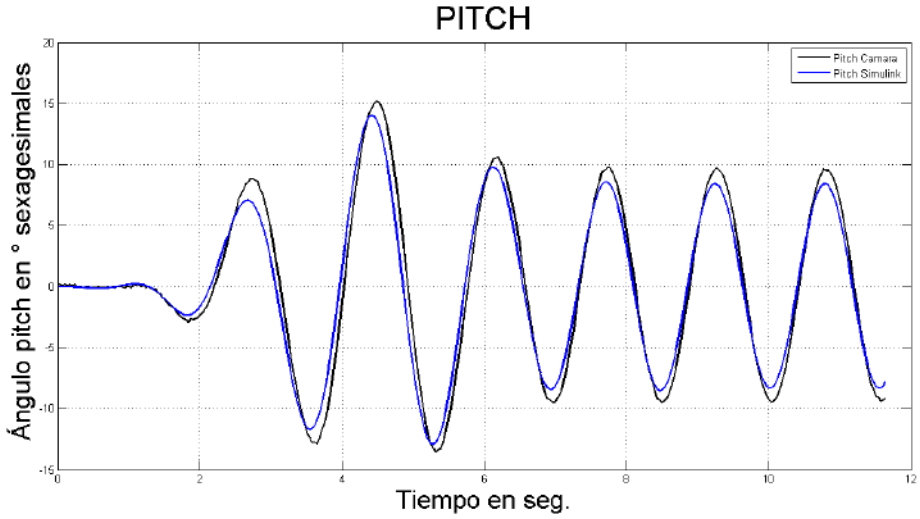

Figura 76 Dinámica del ángulo pitch en el experimento controlado con movimiento sólo en el eje *X*. Fuente: Elaboración propia.

Como se muestra en la Figura 76 el modelo matemático y la dinámica capturada por la cámara son bastante similares. El coeficiente R^2 *R-squared* es calculado para determinar en qué porcentaje la variación de la variable es explicado por el modelo matemático. Para el experimento controlado con movimiento sólo en el eje X se obtuvo $R^2 = 0.96$, comprobando numéricamente la similitud de los datos de la Figura 76.

En la Figura 77 se puede observar en color negro el ángulo "*pitch*" de la carga suspendida según el algoritmo de visión artificial y en color azul el mismo ángulo según el modelo matemático construido en Simulink.

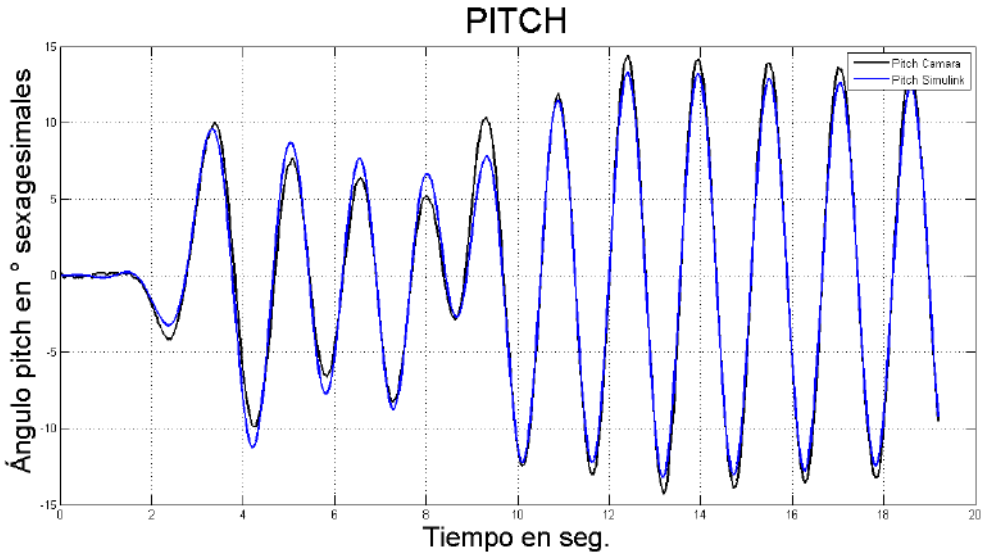

Figura 77 Dinámica del ángulo "*pitch*" en el experimento controlado con movimientos en el eje *X* y *Y*. Fuente: Elaboración propia.

La Figura 77 muestra que el modelo matemático y la dinámica de la carga capturada por la cámara son bastante similares. El coeficiente R^2 "*R-squared*" es nuevamente calculado para determinar en qué porcentaje la variación de la variable es explicado por el modelo matemático. Para el experimento controlado con movimientos en los ejes X y Y se obtuvo $R^2 = 0.88$ en el ángulo "*pitch*", comprobando numéricamente la similitud de los datos de la Figura 77.

En la Figura 78 se puede observar en color negro el ángulo "*roll*" de la carga suspendida según el algoritmo de visión artificial y en color azul el mismo ángulo según el modelo matemático construido en Simulink.

Igual que en los casos anteriores, la Figura 78 muestra que el modelo matemático y la dinámica de la carga capturada por la cámara son bastante similares. El coeficiente R^2 *R-squared* es nuevamente calculado. Para el mismo experimento controlado con movimientos en los ejes X y Y se obtuvo $R^2 = 0.97$, comprobando numéricamente la similitud de los datos de la Figura 78.

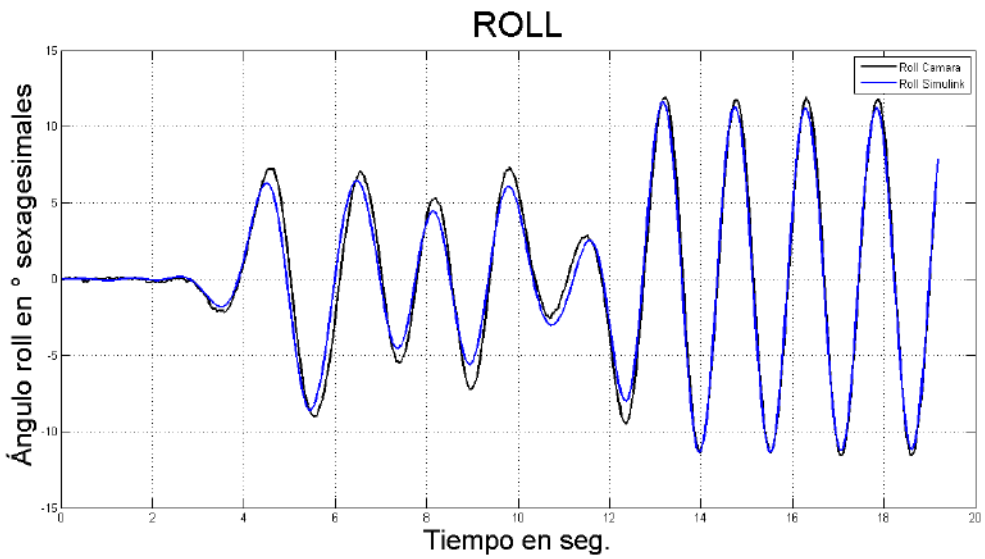

Figura 78 Dinámica del ángulo "*roll*" en el experimento controlado con movimientos en el eje *X* y *Y*. Fuente: Elaboración propia.

5.5.4 Predicción por medio de Kalman

Se reemplazan los parámetros de la Tabla 19 en las ecuaciones (201) a (204) y se obtienen las ecuaciones (212) a (215) las cuales describen a nuestro sistema.

$$\boldsymbol{A_c} = \begin{bmatrix} -0.0192 & -16.6271 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & -0.0192 & -16.6271 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{212}$$

$$\boldsymbol{B_c} = \begin{bmatrix} 0.0192 & 0 & 0 & -1.6949 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0.0192 & 1.6949 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{213}$$

$$\boldsymbol{C_c} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{214}$$

$$\boldsymbol{D_c} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{215}$$

El subíndice D en las ecuaciones (212) a (215) hace referencia a que el sistema es expresado en tiempo continuo. Para discretizar el sistema es necesario definir un tiempo de muestreo, este valor será fijado por el sistema de adquisición de datos. Dado que el sistema en ejecución no tiene el mismo tiempo de muestreo para cada iteración, se debe asumir un tiempo de muestreo igual al promedio del tiempo de muestreo de varias iteraciones. Se realizaron algunas pruebas de ensayo del algoritmo, para este caso se obtuvo un tiempo de muestreo promedio de 33.5 ms.

El sistema es discretizado por medio de las ecuaciones (216) hasta (219). El subíndice c en estas ecuaciones hace referencia a que el sistema es expresado en tiempo discreto.

$$\boldsymbol{A_d} = \exp(\boldsymbol{A_c} T_s) \tag{216}$$

$$\boldsymbol{B_d} = \boldsymbol{A_c}^{-1}(\boldsymbol{A_d} - \boldsymbol{I})\boldsymbol{B_c} \tag{217}$$

$$\boldsymbol{C_d} = \boldsymbol{C_c} \tag{218}$$

$$\boldsymbol{D_d} = \boldsymbol{D_c} \tag{219}$$

Se reemplazan las ecuaciones (212) a (215) en las ecuaciones (216) hasta (219) y para nuestro caso se obtiene el sistema en tiempo discreto presentado en las ecuaciones (220) hasta (223).

$$\boldsymbol{A_D} = \begin{bmatrix} 0.99 & -0.5554 & 0 & 0 \\ 0.0334 & 0.9907 & 0 & 0 \\ 0 & 0 & 0.99 & -0.5554 \\ 0 & 0 & 0.0334 & 0.9907 \end{bmatrix} \tag{220}$$

$$\boldsymbol{B_D} = \begin{bmatrix} 0.0006 & 0 & 0 & -0.0566 & 0 \\ 0 & 0 & 0 & -0.0009 & 0 \\ 0 & 0.0006 & 0.0566 & 0 & 0 \\ 0 & 0 & 0.0009 & 0 & 0 \end{bmatrix} \tag{221}$$

$$\boldsymbol{C_D} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{222}$$

$$\boldsymbol{D_D} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{223}$$

El algoritmo de seguimiento del objetivo no ha funcionado correctamente, lo cual ha producido la pérdida del objetivo en experimentos en los cuales la carga suspendida alcanza velocidades elevadas o sale fuera del ángulo de visión de la cámara.

Para simular un experimento en el cual la carga suspendida oscila hasta salir fuera del ángulo de visón de la cámara, se ha recortado la información de los experimentos anteriores mostrado en la Figura 76 para el experimento 1 y en la Figura 77 y Figura 78 para el experimento 2. El ángulo "*roll*" ha sido limitado en $\pm 5°$ y el ángulo pitch ha sido limitado en $\pm 7°$.

El cálculo de los coeficientes del filtro de Kalman se basa en determinar las covarianzas del error en los sensores de las señales de entrada y las covarianzas del error de las señales de salida. Ambas covarianzas del error se pueden determinar realizando comparaciones de las señales originales y las señales filtradas. Las señales filtradas son calculadas por medio del comando *idealfilter* de Matlab.

Las señales de entrada de los sensores son proporcionados por los acelerómetros del sensor IMU. Calculada la diferencia o error entre las señales originales y las señales filtradas se puede determinar las varianzas de las aceleraciones en las direcciones de los ejes $\sigma_{a_X}^2, \sigma_{a_Y}^2, \sigma_{a_Z}^2$ y las velocidades angulares del quad-rotor $\sigma_{\dot{\phi}_B}^2, \sigma_{\dot{\theta}_B}^2$.

En la ecuación (205) se observa que los estados $\boldsymbol{x}_k$ son afectado por la variable de entrada u_{k-1} multiplicada por $\boldsymbol{B}$. Dado que el error proviene de la variable de entrada $\boldsymbol{u}$, el error $\boldsymbol{w}$ tiene como varianza la misma que el de la entrada $\boldsymbol{u}$ escalada por $\boldsymbol{B}$.

$$\boldsymbol{x}_k = \boldsymbol{A}_D \boldsymbol{x}_{k-1} + \boldsymbol{B}_D \boldsymbol{u}_{k-1} + \boldsymbol{B}_D \begin{bmatrix} N\left(0, \sigma_{\dot{\phi}_B}^2\right) \\ N\left(0, \sigma_{\dot{\theta}_B}^2\right) \\ N\left(0, \sigma_{a_X}^2\right) \\ N\left(0, \sigma_{a_Y}^2\right) \\ N\left(0, \sigma_{a_Z}^2\right) \end{bmatrix} \tag{224}$$

El experimento se ha desarrollado sin desplazamiento en el eje Z y sin giros del cuerpo, por lo tanto se debe considerar $\sigma_{a_Z}^2 = 0, \sigma_{\dot{\phi}_B}^2 = 0, \sigma_{\dot{\theta}_B}^2 = 0$. Por lo tanto la matriz de covarianzas de las variables de estados se define en Q_k como se muestra en la ecuación (225).

$$Q_k = \begin{bmatrix} \sigma_{a_Y}^2 B_{D(1,4)}{}^2 & \left|\left(\sigma_{a_Y} B_{D(1,4)}\right)\left(\sigma_{a_Y} B_{D(2,4)}\right)\right| & 0 & 0 \\ \left|\left(\sigma_{a_Y} B_{D(1,4)}\right)\left(\sigma_{a_Y} B_{D(2,4)}\right)\right| & \sigma_{a_Y}^2 B_{D(2,4)}{}^2 & 0 & 0 \\ 0 & 0 & \sigma_{a_X}^2 B_{D(3,3)}{}^2 & \left|\left(\sigma_{a_X} B_{D(3,3)}\right)\left(\sigma_{a_X} B_{D(4,3)}\right)\right| \\ 0 & 0 & \left|\left(\sigma_{a_X} B_{D(3,3)}\right)\left(\sigma_{a_X} B_{D(4,3)}\right)\right| & \sigma_{a_X}^2 B_{D(4,3)}{}^2 \end{bmatrix} \tag{225}$$

Por medio de la cámara se obtiene la posición de la pelota, con ayuda de las ecuaciones (175), (176) y (177) se diseña un sensor virtual el cual tiene como señales de salida los ángulos estimados de la pelota. Para estas señales se ha considerado que poseen un error con desviación estándar de 0.5°, por lo tanto se tiene $\sigma_{\phi_L}^2 = \sigma_{\theta_L}^2 = \left(\frac{0.5}{180\pi}\right)^2$.

En la ecuación (206) se observa que el valor de la salida $\boldsymbol{y}_k$ es afectado por los estados medidos $\boldsymbol{x}_k$ multiplicados por $\boldsymbol{C}$. Dado que el error proviene de las variables medidas $\boldsymbol{x}$, el error $\boldsymbol{v}$ tiene como varianza la misma que el de los estados medibles $\boldsymbol{x}$ escalados por $\boldsymbol{C}$.

$$\boldsymbol{y}_k = \boldsymbol{C}_D \boldsymbol{x}_k + \boldsymbol{C}_D \begin{bmatrix} N\left(0, \sigma_{\dot{\phi}_L}^2\right) \\ N\left(0, \sigma_{\phi_L}^2\right) \\ N\left(0, \sigma_{\dot{\theta}_L}^2\right) \\ N\left(0, \sigma_{\theta_L}^2\right) \end{bmatrix} \tag{226}$$

En nuestro sistema, las salidas $\boldsymbol{y}_k$ sólo dependen del segundo y cuarto estado, por lo tanto $N\left(0, \sigma_{\dot{\phi}_L}^2\right)$ y $N\left(0, \sigma_{\dot{\theta}_L}^2\right)$ no son necesarios o se puede considerar $\sigma_{\dot{\phi}_L}^2 = 0, \sigma_{\dot{\theta}_L}^2 = 0$. Por lo tanto la matriz de covarianzas de las variables de salida se define en R_k como se muestra en la ecuación (227).

$$R_k = \begin{bmatrix} \sigma_{\phi_L}^2 & 0 \\ 0 & \sigma_{\theta_L}^2 \end{bmatrix} \tag{227}$$

Se considera $P = Q$ para inicializar el algoritmo de Kalman mostrado en las ecuaciones (207) hasta (211). Se hace uso del modelo del sistema discreto mostrado en las ecuaciones (220) hasta (223) y de las matrices de covarianzas de ruido (225) y (227). El filtro de Kalman es implementado en Matlab en el archivo Mi_Kalman.m y el código se encuentra en el Anexo D.

El resultado de aplicar el filtro de Kalman en el experimento 1 se muestra en la Figura 79. En color negro se presenta el ángulo calculado según el algoritmo de visión y en el color rojo se presenta el ángulo corregido o predicho según Kalman, Se observa un buen comportamiento del filtro.

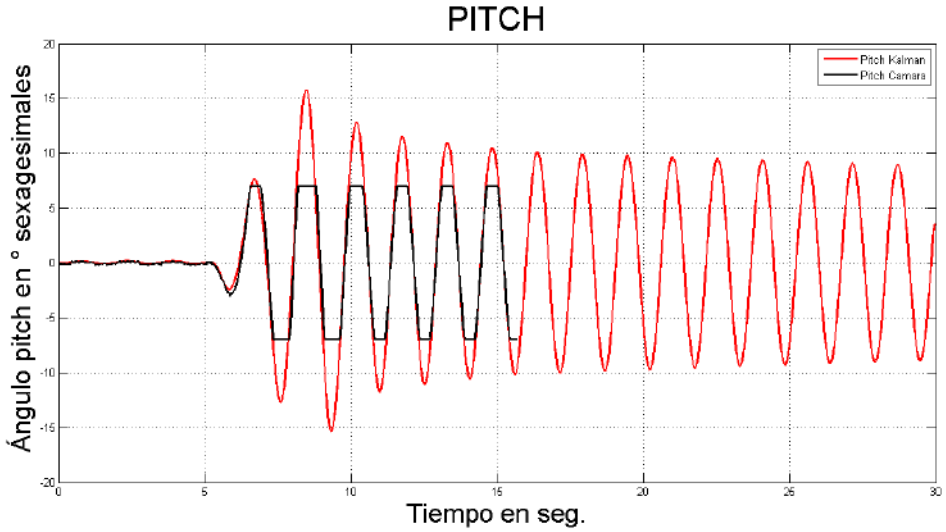

Figura 79 Aplicación del Filtro de Kalman del ángulo "*pitch*" de la carga suspendida. Experimento 1. Fuente: Elaboración propia.

Los resultados de aplicar el filtro de Kalman en el experimento 2 se muestran a continuación. En la Figura 80 se muestra el ángulo "*pitch*" de la carga suspendida a la salida del filtro de Kalman con color rojo y el mismo ángulo según el algoritmo de visión en color negro. Se puede observar corrección dentro del rango de visión y predicción del ángulo de la carga suspendida fuera del rango de visión de la cámara.

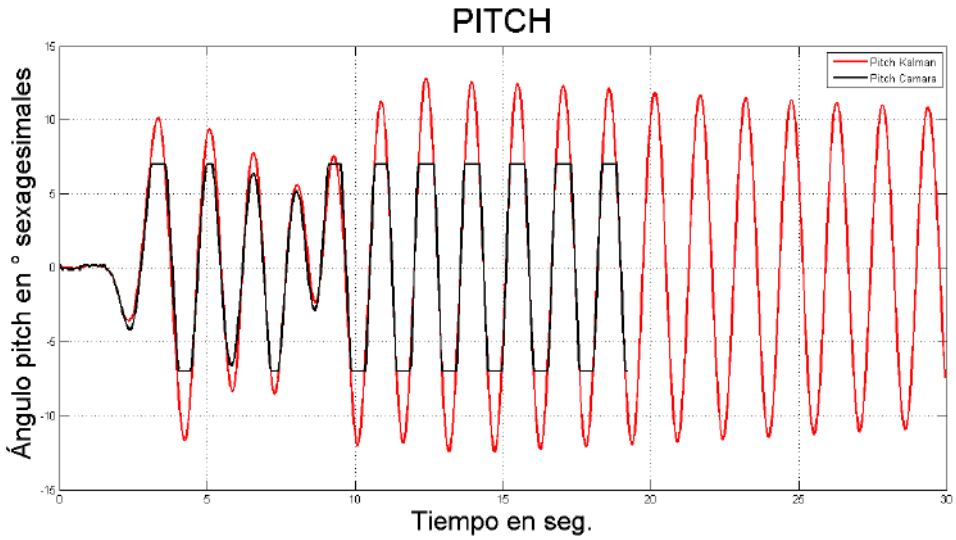

Figura 80 Aplicación del Filtro de Kalman del ángulo "*pitch*" de la carga suspendida. Experimento 2. Fuente: Elaboración propia.

En la Figura 81 se muestra el ángulo "*roll*" de la carga suspendida a la salida del filtro de Kalman con color rojo y el mismo ángulo según el algoritmo de visión en color negro. Se puede observar corrección dentro del rango de visión y predicción del ángulo de la carga suspendida fuera del rango de visión de la cámara.

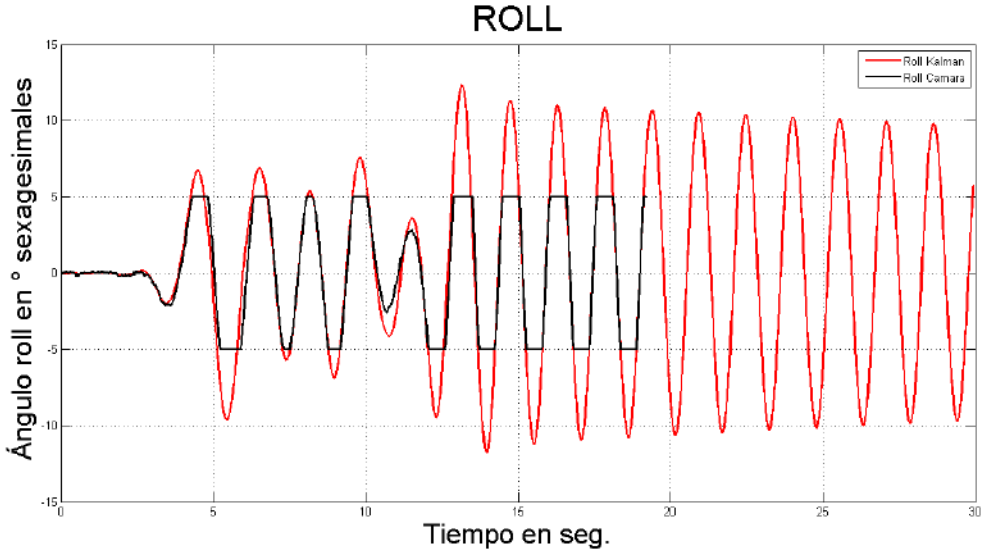

Figura 81 Aplicación del Filtro de Kalman del ángulo "*roll*" de la carga suspendida. Experimento 2. Fuente: Elaboración propia.

Finalmente se puede concluir que el modelo desarrollado sigue bastante bien a la realidad tanto para movimientos unidimensionales como para movimientos bidimensionales, esto se ha corroborado con el cálculo de coeficientes cuyos valores son mayores a 0.88. Además se ha podido comprobar el buen funcionamiento del filtro de Kalman en nuestros experimentos. Como trabajos futuros se plantea realizar experimentos con movimientos tridimensionales, implementar todos los algoritmos en el sistema embebido para ser ejecutados a tiempo real y mejorar el algoritmo de seguimiento.

Conclusiones

Se ha comprobado que la orientación de un VANT puede ser modelada utilizando los ángulos de Euler, mediante los cuales se puede representar la orientación relativa entre dos sistemas de coordenadas como consecuencia de tres rotaciones sucesivas.

Un caso particular de los ángulos de Euler llamado ángulos de Tait-Bryan se ha utilizado satisfactoriamente para modelar rotaciones de cuerpos rígidos en procesos aeroespaciales.

Mediante simulación se verifica que bajo condiciones de ángulos de Euler pequeños, sus derivadas son aproximadamente iguales a las velocidades angulares del cuadricóptero. Lo cual hace válido considerar que las aceleraciones angulares del cuadricóptero son iguales a las aceleraciones de Euler.

Mediante las simplificaciones consideradas en este trabajo, se ha deducido el sistema de ecuaciones (47) las cuales simplifican considerablemente la complejidad el modelo matemático del cuadricóptero. Además permite tener cuatro sistemas SISO en lugar de un sistema MIMO.

Las ecuaciones (50) y (91) son fundamentales para relacionar el sistema de cuatro procesos SISO y el sistema MIMO. El control implementado considera como variables manipulables $U_1, U_2, U_3\ y\ U_4$ mientras que en hardware se manipula el valor del *duty cycle* de la señal que comanda los driver de los cuatro motores.

La implementación de un mundo virtual ha sido muy importante para observar el funcionamiento del cuadricóptero con condiciones controladas.

Se ha publicado un *paper* en la base de datos IEEExplorer titulado *Modeling and PID cascade control of a Quadcopter for trajectory tracking* presentado en 2015 *CHILEAN*

Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON) con ISBN 978-1-4673-8755-2.

En simulación se observó que los movimientos de traslación del cuadricóptero no se detienen debido a que no se han considerado los coeficientes de rozamiento con el aire. En futuros trabajos se deben calcular dichos coeficientes con ayuda de softwares de aeromodelismo para obtener un simulador más real.

El control PD presentado en publicaciones importante como [39], muestra una aparente solución a nuestro problema de control. Sin embargo, si se considera un pequeño torque externo sobre el cuadricóptero, se obtienen grandes errores en el control de ángulos.

El controlador PD funciona muy bien considerando que no existen disturbios en el sistema de orientación. Enfocando el problema a un sistema real, el control PD es totalmente ineficaz. Por lo tanto se deduce la necesidad de agregar un término integral al controlador.

En simulación la robustez del sistema frente a perturbaciones mejora añadiendo un término integral, sin embargo el control no es eficiente ya que se producen pequeñas oscilaciones no deseadas sobre el cuadricóptero. La maniobrabilidad y la robustez del cuadricóptero aumentan al cambiar a una configuración de estructura en equis, debido a que en esta configuración siempre trabajan los cuatro rotores para realizar movimientos de *roll* y de *pitch*.

En simulación se ha comprobado que el control PID en cascada para el seguimiento de trayectorias ha podido sobreponerse a todas las perturbaciones tanto de pares como de fuerzas sobre el cuadricóptero.

En la Figura 62, Figura 63 y Figura 64 se muestran gráficas comparativas del *roll*, *pitch* y *yaw* respectivamente entre datos experimentales (en color azul) y los datos de simulación (en color rojo). Como se observa en Figura 62, Figura 63 las curvas son bastante semejantes. En la Figura 64 se observa una mayor diferencia entre los datos, los cuales se explica debido a que el banco de pruebas posee un rozamiento el cual no se ha calculado, además el centro de gravedad del cuadricóptero no coincide con el centro de giro cuando el cuadricóptero se encuentra empotrado en el banco de pruebas.

En la Tabla 15 se muestra un resumen de los parámetros de validación utilizados. En general se obtienen valores satisfactorios, aunque menos favorables para el *yaw*. Sin embargo se concluye que el modelo esta validado.

Se realizaron algunas pruebas experimentales del controlador de la sección 3.2 Desarrollo de controladores para el ArduCopter Quad-C donde se observó que el cuadricóptero sí lograba estabilizar. Sin embargo bajo influencia de pequeñas corrientes de viento o cambios de consignas el cuadricóptero osciló hasta perder la estabilidad.

Las pruebas realizadas con la lógica de control propuesta en la sección 3.3 Controladores mejorados para el ArduCopter Quad-C y con las ganancias mostradas en 3.4 Sintonización de Controladores se observó un vuelo suave, siempre bajo control y estable frente a perturbaciones externas.

Como prueba final, se realizaron vuelos del cuadricóptero en la cancha de gras de la Universidad de Piura entre las 6 p.m. y 7 p.m. a una altura aproximada de 10 a 20 metros, mostrando gran robustez frente a las fuertes corrientes de viento.

Se validó el modelo matemático de la carga suspendida desde un cuadricóptero desarrollado en la estancia científica llevada a cabo en la Universidad de Sevilla. Así mismo, se comprobó el buen funcionamiento del filtro de Kalman para corregir señales y para predecir el ángulo de la carga suspendida.

Se propone como nuevas líneas de investigación usando como base esta tesis, implementar algoritmos de visión artificial para localización y control de VANT, control de posición de VANT, diseño de nuevos controladores para VANT en sistemas embebido más potentes e implementación de un VANT con un escáner LIDAR.

Se dispone en el Anexo B el código implementado en Visual Studio 2010 el cual servirá para futuras investigaciones en el campo de visión artificial, específicamente en el seguimiento de objetos. Esto se podría aplicar para el control de posición de cuadricópteros *indoor* por medio de cámaras situadas en un ambiente cerrado.

Otros trabajo futuro planteado consiste en realizar experimentos con movimientos tridimensionales, implementar todos los algoritmos en el sistema embebido para ser ejecutados a tiempo real y mejorar el algoritmo de seguimiento.

Los estudios realizados durante la estancia científica en la Universidad de Sevilla, serán las bases para futuras investigaciones en el campo de transporte de carga suspendida con un cuadricóptero, tema que no ha sido desarrollado en el Perú.

Bibliografía

[1] Instituto Nacional de Estadística y Geografía, «El uso de vehículos aéreos no tripulados en las tareas geoespaciales,» *Ciencia y Tecnología,* p. 5, 8 Abril 2014.

[2] S. Bouabdallah, A. Noth y R. Siegwart, «PID vs LQ Control Techniques Applied to an Indoor Micro Quadrotor,» *Proceedings. 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems,* vol. 3, pp. 2451-2456, 2004.

[3] J. Calvo, E. Anna, C. Blanco y S. Gabriela, «El Centre d'Estudis per la Pau,» Marzo 2014. [En línea]. Available: http://www.centredelas.org/images/stories/informes/informe23_cas.pdf. [Último acceso: 16 Febrero 2015].

[4] E. Camacho, «La Republica,» 23 Junio 2014. [En línea]. Available: http://larepublica.pe/23-06-2013/drones-sobre-los-campos-de-lima. [Último acceso: 23 Febrero 2015].

[5] S. Bouabdallah, P. Murrieri y R. Siegwart, «Design and Control of an Indoor Micro Quadrotor,» *Proceedings of the International Conference on Robotics and Automation,* vol. 5, pp. 4393 - 4398, 2004.

[6] S. Bouabdallah, P. Murrieri y R. Siegwart, «Towards Autonomous Indoor Micro VTOL,» *Autonomous Robots,* vol. 18(2), pp. 171-183, 2005.

[7] S. Bouabdallah y R. Siegwart, «Backstepping and Sliding-mode Techniques Applied to an Indoor Micro Quadrotor,» *Proceedings of the 2005 IEEE International Conference on Robotics and Automation,* pp. 2247 - 2252, 2005.

[8] S. Bouabdallah y R. Siegwart, «Full Control of a Quadrotor,» *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2007. IROS 2007.,* pp. 153 - 158, 2007.

[9] S. Bouabdallah, «Design and control of quadrotors with application to autonomous flying,» Tesis doctoral, Laboratoire de systèmes autonomes, ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE, LAUSANNE, 2007.

[10] N. Guenard, T. Hamel y R. Mahony, «A Practical Visual Servo Control for an Unmanned Aerial Vehicle,» *IEEE Transactions on Robotics,* vol. 4(2), pp. 331 - 340, 2008.

[11] B. Herisse, F.-X. Russotto, T. Hamel y R. Mahony, «Hovering flight and vertical landing control of a VTOL Unmanned Aerial Vehicle using Optical Flow,» *IEEE/RSJ International Conference on Intelligent Robots and Systems,* pp. 801 - 806, 2008.

[12] O. Bourquardez, R. Mahony, N. Guenard, F. Chaumette, T. Hamel y L. Eck, «Image-based Visual Servo Control of the Translation Kinematics of a Quadrotor Aerial Vehicle,» *IEEE Transactions on Robotics,* vol. 25(3), pp. 743 - 749, 2009.

[13] B. Hérissé, T. Hamel, R. Mahony y F.-X. Russotto, «Landing a VTOL Unmanned Aerial Vehicle on a moving platform using optical flow,» *IEEE Transactions on Robotics,* vol. 28(1), pp. 77 - 89, 2012.

[14] E. Altug, J. P. Ostrowski y M. Robert, «Control of a Quadrotor Helicopter Using Visual Feedback,» *IEEE International Conference on Robotics and Automation,* vol. 1, pp. 72 - 77, 2002.

[15] E. Altug, J. P. Ostrowski y C. J. Taylor, «Control of a Quadrotor Helicopter Using Dual Camera Visual Feedback,» *IEEE International Conference on Robotics and Automation,* vol. 24(5), pp. 329 - 341, 2003.

[16] D. Mellinger y V. Kumar, «Minimum Snap Trajectory Generation and Control for Quadrotors,» *IEEE International Conference on Robotics and Automation,* pp. 2520 - 2525, 2011.

[17] D. Mellinger, N. Michael y V. Kumar, «Trajectory Generation and Control for Precise Aggressive Maneuvers with Quadrotors,» *Experimental Robotics. The 12th International Symposium on Experimental Robotics,* vol. 79, pp. 361-373, 2014.

[18] R. A. García, F. R. Rubio y M. G. Ortega, «Robust PID Control of the Quadrotor Helicopter,» *IFAC Conference on Advances in PID Control,* vol. 2(5), pp. 229-334, 2012.

[19] T. Bresciani, «Modelling, Identification and Control of a Quadrotor Helicopter,» Tesis M.S., Dept. of Automatic Control, Lund University, Lund, 2008.

[20] R. Guilherme Vianna, «Modelado y control de un helicóptero Quadrotor,» Tesis M.S., Dept. Ingeniería de Sistemas y Automática, Universidad de Sevilla, Sevilla, 2007.

[21] Universidad de Alcalá, «Servidor de Teoria de la Señal,» [En línea]. Available: http://agamenon.tsc.uah.es/Asignaturas/it/rd/apuntes/handout4.pdf.

[22] R. Olfati-Saber, «Nonlinear control of underactuated mechanical Systems with Application to Robotics and Aerospace Vehicles,» Tesis M.S., Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Massachusetts, 2001.

[23] H. Fernando, A. De Silva, M. De Soysa, S. Munasinghe y K. Dilshan, «Modelling, Simulation and Implementation of a Quadrotor UAV,» *IEEE International Conference on Industrial and Information Systems,* pp. 207 - 212, 2013.

[24] R. M. Murray, Z. Li y S. S. Sastry, A Mathematical Introduction to Robotic Manipulation, Boca Ratón: CRC Press, 1994.

[25] P. Castillo, R. Lozano y A. E. Dzul, Modelling and Control of Mini-Flying Machines, London: Springer-Verlag London, 2005.

[26] L. R. García Carrillo, A. E. Dzul López, R. Lozano y C. Pégard, Quad Rotorcraft Control - Vision-Based Hovering and Navigation, Verlag London: Springer, 2012.

[27] Dronefever.com, «Dronefever.com,» [En línea]. Available: http://dronefever.com/3DR-ArduCopter-Quad-D-Almost-Ready-to-Fly.html. [Último acceso: 19 Febrero 2015].

[28] RobotShop, «RobotShop,» [En línea]. Available: http://www.robotshop.com/en/20a-bec-multirotor-esc-without-connectors.html. [Último acceso: 19 Febrero 2015].

[29] Planetarobot, «Planetarobot,» [En línea]. Available: http://planetarobot.com/product_info.php?products_id=35. [Último acceso: 19 Febrero 2015].

[30] HeliDirect, «HeliDirect,» [En línea]. Available: http://www.helidirect.com/apc-10x4-5-multirotor-propeller.html. [Último acceso: 19 Febrero 2015].

[31] P. Ortiz González, «Modelado y control robusto de un vehículo aéreo no tripulado quadrotor, en ambientes cerrados,» Tesis de Maestría en Control y Automatización Industrial, Universidad Politécnica Salesiana, Cuenca, 2014.

[32] E. Saadé Latorre, «Propulsion system optimization for an unmanned lightweight quadrotor,» Tesis Master in Aerospace Science and Technology, Universitat Politècnica de Catalunya, Catalunya, 2011.

[33] M. J. Reinoso Avecillas, «Diseño de un sistema de control por régimen deslizante para el seguimiento de trayectoria lineal de un quadrotor,» Tesis Maestría en Control y Automatización Industrial, Universidad Politécnica Salesiana, Cuenca, 2014.

[34] J. M. Cotte Corredor y A. F. Moreno Pineda, «Diseño de control robusto de velocidad de motores brushless para Robótica aerea,» Tesis Trabajo de Grado para optar por el titulo de Ingeniero Electrónico, Facultad de Ingeniería Departamento de Ingeniería Eléctrica y Electrónica, Universidad Nacional de Colombia, Bogotá, 2010.

[35] P. Castillo, P. García, R. Lozano y P. Albertos, «Modelado y estabilización de un helicóptero con cuatros rotres,» *Revista Iberoamericana de Automática e Informática,* vol. 4, nº 1, pp. 41-57, 2007.

[36] W. Ipanaqué Alama, Control automático de procesos. Innovando los procesos productivos., Primera ed., Lima: CONCYTEC, 2012.

[37] R. G. Rodriguez Torres , Comparación de modelos matemáticos y controladores PID vs LQR para un cuadricóptero, Piura: Universidad de Piura, 2016.

[38] A. Dubus, «Google Developers,» Marzo 2015. [En línea]. Available: https://code.google.com/p/multiwii/. [Último acceso: 12 Enero 2016].

[39] M. K. Bayrakceken y A. Arisoy, «An Educational Setup for Nonlinear Control Systems,» *IEEE Control Systems,* vol. 33, nº 2, pp. 64-81, 2013.

[40] Arduino, «Arduino,» [En línea]. Available: http://playground.arduino.cc/ArduinoNotebookTraduccion/Datatypes. [Último acceso: 13 Enero 2016].

[41] J. Cárdenas Egea, Detección, seguimiento y estimación del centro de masas de un objeto suspendido de un quadrotor, Sevilla, España: Universidad de Sevilla, 2015.

[42] J. J. Craig, Introduction to Robotics Mechanics ans Control, Segunda ed., USA: Addison-Wesley Publishing, 1989.

Anexo A

```
%clear all, close all, clc,

%%%%%%%%%%% Tiempo de Inicio %%%%%%%%%%
Inicio=15/mean(diff(VarName6(1:end-1)));    %Se estable el tiempo de inicio.
% Debe ser por no menos 15 segundos para que la IMU se calibre.
% Recomiendo establecer en 30 segundos.

%%%%%%%%%%%%%% Obtener Aceleraciones en m/s^2 %%%%%%%%%%
Ax=VarName1(Inicio:end)/16384*9.81*2;
Ay=VarName2(Inicio:end)/16384*9.81*2;
Az=VarName3(Inicio:end)/16384*9.81*2;

%%%%%%% Se define la variable Tiempo %%%%%%%%%%
t=VarName6(Inicio:end)-VarName6(round(Inicio));

%%%%%%%%%% Se definen los pixeles de la carga %%%%%%%%%%
Ximg=VarName4(Inicio:end);
Yimg=VarName5(Inicio:end);

%%%%%% Corrección de datos NAN %%%%%%%%
if isnan(t(end))
   t(end)=t(end-1)+(t(end-1)-t(end-2));
   Ax(end)=0;
   Ay(end)=0;
   Az(end)=0;
   Ximg(end)=Ximg(end-1);
   Yimg(end)=Yimg(end-1);
end

%%%%%%%%%%%%
%%% OBTENCIÓN DE LA POSICIÓN DEL CM Y ÁNGULO BETA DE LA MASA SUSPENDIDA %%%
%%%%%%%%%%%%

%%%%%%%  DATOS INICIALES  %%%%%%%%%%%%%
Cx=625.293341482621/4;  % Centro del plano sensor en eje X
Cy=346.289622915326/4;  % Centro del plano sensor en eje Y
Fx=1091.24858443191/4;  % Fx=F*Sx. Donde F es distancia focal y Sx es pixeles por milímetro en dirección x
Fy=1089.45369875399/4;  % Fy=F*Sy. Donde F es distancia focal y Sy es pixeles por milímetro en dirección y

%%%%%%%%% Coordenadas del punto de cogida %%%%%%
X0=0;
Y0=45;
Z0=-20;

%%%%%%%%% Corrección de distorsión radial %%%%%%
%Kc1=-0.0240113916806306;
%Kc2=-0.0923769324510727;
%r2=((Ximg-Cx)/Fx).^2+((Yimg-Cy)/Fy).^2;
%r4=r2.^2;%((Ximg-Cx)/Fx).^4+((Yimg-Cy)/Fy).^4;
%Ximg=(1+Kc1*r2+Kc2*r4).*Ximg;
%Yimg=(1+Kc1*r2+Kc2*r4).*Yimg;

%%%%%%%%%%%%%%  CALCULO DE POSICION  %%%%%%%%%%%
a=(Cx - Ximg).^2/Fx^2 + (Cy - Yimg).^2/Fy^2 + 1;
b=(2*X0*(Cx - Ximg))/Fx - 2*Z0 + (2*Y0*(Cy - Yimg))/Fy;
c=- (L*1000)^2 + X0^2 + Y0^2 + Z0^2;
Z=(-b+sqrt(b.^2-4*a*c))./(2*a);
X=Z.*(Ximg-Cx)/Fx;
Y=Z.*(Yimg-Cy)/Fy;

%%%%%%%%%%%%  Corrección de Posición  %%%%%%%%%%
%%%% Esto es debido a que la webcam no mira exactamente en dirección Z %%%%
X=X-(mean(X(1:30))-X0);
Y=Y-(mean(Y(1:30))-Y0);
```

```
%%%%%%  CALCULO DE ANGULO  %%%%%%%%%%%%
Pitch=atan((X0-X)./(-Z0+Z));
Roll=atan((Y0-Y)./(-Z0+Z));
```

Anexo B

```
// OpenCV246.cpp: archivo de proyecto principal.
#include "stdafx.h"
#include <opencv\cv.h>
#include <opencv\cv.hpp>
#include <opencv\highgui.h>
#include <math.h>
#include <stdio.h>
#include <time.h>
#include <Windows.h>
#include <iostream>
#include <opencv2/highgui/highgui.hpp>
#include <Strmif.h>
#include <Unknwn.h>

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <stdint.h>
#include <windows.h>
#include <winbase.h>

#include <tchar.h>
#include <strsafe.h>
#include <string>


using namespace std;

#ifndef __cplusplus
#define TRUE 1
#define FALSE 0
#endif

#define MAX_THREADS 3
#define BUF_SIZE 255

#define DEVICE L"\\\\.\\COM3"

// Baud-rate
#define BAUDRATE 115200

// Windows file handle
static HANDLE hCom = INVALID_HANDLE_VALUE;

DWORD WINAPI MyThreadFunction( LPVOID lpParam );
void ErrorHandler(LPTSTR lpszFunction);

typedef struct MyData
{
	int idHilo;
	float accx;
	float accy;
	float accz;
	int semaforo;

} MYDATA, *PMYDATA;

static bool serial_open();
static bool serial_close();
static int serial_read(byte buffer[], int size);
static int read_and_print_message(float*,float*,float*);

// Variables para IMU - Hilo 0
```

```
float _accx,_accy,_accz;
float _center_elipse_x,_center_elipse_y;
int datosNuevosIMU;
int datosNuevosCAM;

// Variables para CAM - Hilo 1
float M[3][3];

int main()
{
	PMYDATA pDataArray[MAX_THREADS];
	DWORD   dwThreadIdArray[MAX_THREADS];
	HANDLE  hThreadArray[MAX_THREADS];

	_accx = 0;_accy = 0;_accz = 0;
	_center_elipse_x = 0,_center_elipse_y = 0;
	datosNuevosIMU = 0;
	datosNuevosCAM = 0;

	for( int i=0; i<MAX_THREADS; i++ )
	{
		// Allocate memory for thread data.

		pDataArray[i] = (PMYDATA) HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY,
			sizeof(MYDATA));

		if( pDataArray[i] == NULL )
			ExitProcess(2);

		// Generate unique data for each thread to work with.
		pDataArray[i]->idHilo = i;
		//pDataArray[i]->val2 = i+100;

		// Create the thread to begin execution on its own.

		hThreadArray[i] = CreateThread(
			NULL,                   // default security attributes
			0,                      // use default stack size
			MyThreadFunction,       // thread function name
			pDataArray[i],          // argument to thread function
			0,                      // use default creation flags
			&dwThreadIdArray[i]);   // returns the thread identifier


		// Check the return value for success.
		// If CreateThread fails, terminate execution.
		// This will automatically clean up threads and memory.

		if (hThreadArray[i] == NULL)
		{
			ErrorHandler(TEXT("CreateThread"));
			ExitProcess(3);
		}
	} // End of main thread creation loop.

	// Wait until all threads have terminated.

	WaitForMultipleObjects(MAX_THREADS, hThreadArray, TRUE, INFINITE);

	// Close all thread handles and free memory allocations.

	for(int i=0; i<MAX_THREADS; i++)
	{
		CloseHandle(hThreadArray[i]);
		if(pDataArray[i] != NULL)
		{
			HeapFree(GetProcessHeap(), 0, pDataArray[i]);
```

```
                        pDataArray[i] = NULL;    // Ensure address is not reused.
                }
        }

        return 0;

}

DWORD WINAPI MyThreadFunction( LPVOID lpParam )
{
        HANDLE hStdout;
        PMYDATA pDataArray;

        // Make sure there is a console to receive output results.

        hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
        if( hStdout == INVALID_HANDLE_VALUE )
                return 1;

        // Cast the parameter to the correct data type.
        // The pointer is known to be valid because
        // it was checked for NULL before the thread was created.

        pDataArray = (PMYDATA)lpParam;

        FILE *pf=fopen("datos_camshift.csv","wt");


        // Print the parameter values using thread-safe functions.
        if(pDataArray->idHilo == 0)
        {
                // Hilo de la IMU
                if (serial_open())
                {
                        //printf("Starting on %s @ B%d\n", DEVICE, BAUDRATE);
                        //printf("Hit any key to exit\n\n");
                        float accx = 0,accy = 0,accz = 0;

                        while (!kbhit())
                        {
                                if(read_and_print_message(&accx,&accy,&accz))
                                {
                                        //printf("HILO 0: %.0f %.0f %.0f\n", accx,accy,accz);
                                        datosNuevosIMU = 1;
                                        pDataArray->semaforo = 1;
                                        _accx= accx;
                                        _accy= accy;
                                        _accz= accz;
                                        pDataArray->semaforo = 0;
                                }



                        }

                        serial_close();
                }
        }
        else if(pDataArray->idHilo == 1)
        {
                // Hilo de OpenCV
                int contador = 0;
//              while(1)
//              {
//              int main(int argc, char *argv[])
//              {
                        //////////// CAPTURA Y DEFINICIÓN DE ESTRUCTURAS ////////////////
```

```
CvCapture* capture =capture = cvCaptureFromCAM(1); // Constructores de captura de vídeo
cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_WIDTH, 1280);
cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_HEIGHT, 720);
cout<<"Ancho
"<<cvGetCaptureProperty(capture,CV_CAP_PROP_FRAME_WIDTH)<<endl;
cout<<"Alto
"<<cvGetCaptureProperty(capture,CV_CAP_PROP_FRAME_HEIGHT)<<endl;
//      cvNamedWindow("BackProjection", 0); //Crear una ventana llamada BackProjection
cvNamedWindow("Detector", 0); //Crear una ventana llamada Detector

IplImage *frame, *hsv, *backproj, *frame2; // estructura de una imagen
IplImage *h_plane, *s_plane, *v_plane;          // estructura de una imagen
IplImage* planes[3];          // estructura de una imagen
CvSize size; //Tamaño de un rectángulo o un imagen
CvSize resize; //Tamaño de un rectángulo o un imagen
CvSeq* results; //Secuencia Dinámicamente creciente
CvMemStorage* storage = cvCreateMemStorage(0);      //CvMemStorage.-          Un
almacenamiento para diferentes estructuras de datos dinámicas OpenCV, como CvSeq, CvSet etc. //cvCreateMemStorage.-
Crea el almacenamiento de memoria
CvPoint pt, pi1, pi2,p_mean,pmax;     //constructs CvPoint structure
CvRect ventana,roi1;          //struct CvRect
float* p;
bool readhist=false;
bool read_area_inicial=false;
CvRect track_window;      //struct CvRect
CvBox2D track_box;        //struct CvBox2D coordenadas de un rectángulo girado.
CvConnectedComp track_comp;      //Connected component structure
float center_elipse_x,center_elipse_y,p1,p2,error; //Centro de la elípse en X y Y
float val;
int             width=cvGetCaptureProperty(capture,CV_CAP_PROP_FRAME_WIDTH);
//cvGetCaptureProperty.- obtiene prop. del video. Anchura de los frames de vídeo
int             height=cvGetCaptureProperty(capture,CV_CAP_PROP_FRAME_HEIGHT);
//cvGetCaptureProperty.- obtiene prop. del video. Anchura de los frames de vídeo
size=cvSize(width,height); // define el Size of an image.
int a=4;
resize=cvSize(width/a,height/a);
//      cout<< width << height <<endl;
//              cout<<"Resize "<<resize<<endl;
hsv = cvCreateImage(resize, IPL_DEPTH_8U, 3 ); // configura la estructura de imagen "hsv"
con tamaño "size", con datos de 8 bits  y 3 canales
backproj= cvCreateImage( resize, IPL_DEPTH_8U, 1 );//configura la estructura de imagen
"backproj" con tamaño "size", con datos de 8 bits  y 1 canales
h_plane = cvCreateImage( resize, IPL_DEPTH_8U, 1 );//configura la estructura de imagen
"h_plane" con tamaño "size", con datos de 8 bits  y 1 canales
s_plane = cvCreateImage( resize, IPL_DEPTH_8U, 1 );//configura la estructura de imagen
"s_plane" con tamaño "size", con datos de 8 bits  y 1 canales
v_plane = cvCreateImage( resize, IPL_DEPTH_8U, 1 );//configura la estructura de imagen
"v_plane" con tamaño "size", con datos de 8 bits  y 1 canales
frame = cvCreateImage(resize, IPL_DEPTH_8U, 3 ); // configura la estructura de imagen
"hsv" con tamaño "size", con datos de 8 bits  y 3 canales
planes[0]=h_plane;
planes[1]=s_plane;
planes[2]=v_plane;// Se construye la imagen planes a partir de 3 matríces
//////////////////////////////////////////////////////////////
//////////// DEFININICIÓN DE PUNTOS Y VENTANA PREVIA /////////////
pi1=cvPoint((width/2-30)/a,(height/2-30)/a);      // coordenas del punto 1
pi2=cvPoint((width/2+30)/a,(height/2+30)/a);      // coordenas del punto 2
ventana=cvRect((width/2-30)/a,(height/2-30)/a,60/a,60/a); //define con rectángulo con
coordenadas de las esquinas y ancho y altura
//////////////////////////////////////////////////////////////
//////////////////// CREACIÓN DEL HISTOGRAMA ////////////////////
int h_bins = 16, s_bins=16, v_bins=16;//
CvHistogram* hist;
{
        int hist_size[] = { h_bins, s_bins, v_bins};
        float h_ranges[] = { 0 , 256 };
```

```
float s_ranges[] = { 0 , 256 };
float v_ranges[] = { 0 , 256 };
float* ranges[]= { h_ranges, s_ranges, v_ranges };
hist = cvCreateHist( 3, hist_size, CV_HIST_ARRAY, ranges, 1 ); //Crea la
estructura del histograma. de 3 dimensiones, tamaño de cada dimensión,
}
//////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////
clock_t timeclock;
clock_t timeclock_bucle;
float tiempo;
float tiempo_bucle;
//////////////////////  BUCLE DE INICIALIZACIÓN  ///////////////////////
while(readhist==false)
{
	frame2 = cvQueryFrame(capture); // Captura de imágenes
	cvResize(frame2,frame,CV_INTER_LINEAR);
	timeclock=clock(); // Para contabilizar los segundos antes de realizar la captura
	tiempo=float(timeclock)/CLOCKS_PER_SEC;
	cvRectangle(frame,pi1,pi2,cvScalar(255,255,255),2); // dibuja un rectángulo en el
centro de imagen de la cámara de color blanco.
	cvCvtColor(frame, hsv, CV_RGB2HSV ); //convierte una imágen de un espacio de
colores a otro
	cvSplit(hsv, h_plane, s_plane, v_plane, NULL ); //convierte una imágen en 3
canales

	if (tiempo>=4)
	{
		cvSetImageROI( h_plane, ventana );	// Define región de interés de
"h_plane"
		cvSetImageROI( s_plane, ventana );	// Define región de interés de
"s_plane"
		cvSetImageROI( v_plane, ventana );	// Define región de interés de
"v_plane"
		cvCalcHist(planes, hist); // Calcula el histograma de "planes"
		float max_val;
		cvGetMinMaxHistValue(hist,0,&max_val,0,0);	//Encuentra	los	bin
máximo y mínimo y sus posiciones // histograma, min, max, coordenadas de min, coord. de máx.
		cvNormalizeHist(hist,255*3600/max_val); //normailiza el histograma, //
histograma,factor de normalización.
		cvResetImageROI( h_plane );		//Libera "h_plane"
		cvResetImageROI( s_plane );		//Libera "s_plane"
		cvResetImageROI( v_plane );		//Libera "v_plane"
		track_window = ventana;
		readhist=true;
	}
	cvShowImage( "Detector", frame ); //Muestra la imagen en la ventana especificada
llamada "Detector"
	// Escape
	char c = cvWaitKey(1);		// Si teclea ! se paraliza el proceso
	if( c == 27 ) break; // Si teclea ESC se detiene el proceso
}
//////////////////////////////////////////////////////////////////
////////////////////////  BUCLE PRINCIPAL  ////////////////////////
clock_t timeclock_stop1=clock();
clock_t timeclock_stop2=clock();
float tiempo_stop;



while (readhist==true)
//	while (tiempo_stop<10)
{
		timeclock_bucle=clock(); // Para medir los tiempos del bucle

		frame2 = cvQueryFrame(capture); // Captura de imágenes
		cvResize(frame2,frame,CV_INTER_LINEAR);
```

```
cvCvtColor(frame, hsv, CV_RGB2HSV );        //Converts input image
from one color space to another //1ms
cvSplit(hsv, h_plane, s_plane, v_plane, NULL ); //Divide una matriz
multi-canal en matrices de un solo canal por separad0 //1 ms
cvCalcBackProject(planes, backproj, hist );        //BackProject de
"planes"
cvSmooth(backproj, backproj, CV_GAUSSIAN, 9 , 9 );    //Suaviza
imagen utilizando uno de varios métodos
cvMinMaxLoc(backproj,NULL,NULL,NULL,&pmax,NULL);          //
Encuentra valores de los elementos mínimos y máximos y sus posicione //2 a 4 ms
track_comp.value.val[3]=0;   //Ni idea
cvCamShift(backproj,                track_window,cvTermCriteria(
CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 10, 1 ),&track_comp, &track_box );   // se encuentra un centro de objeto
usando cvMeanShift y, después de eso, calcula el tamaño y la orientación del objeto
cvEllipseBox( frame, track_box, cvScalar(0,0,255), 3, 8, 0 );//Dibuja un
arco elíptico
track_window = track_comp.rect;// Se actualiza el valor de track_window
val=(float)track_comp.value.val[3];
if ( val==0 ) //  Hay que mejorar las condiciones, para el caso de regiones
pequeñas y mayores
{
        results                                                =
cvHoughCircles(backproj,storage,CV_HOUGH_GRADIENT,3,backproj->width,200,50,5,60);    //Encuentra círculos en
la imagen en escala de grises utilizando alguna modificación de Hough
        for( int i = 0; i < results->total; i++ )
        {
                p = (float*)cvGetSeqElem( results, i ); //OBTIENE
RESULTADOS
                pt     =     cvPoint(cvRound(p[0]),cvRound(p[1]));
//OBTIENE EL PUNTO
                cvCircle(frame,pt,2, CV_RGB(255,255,0), 2, 8, 0 );
                cvCircle(frame,pt,cvRound(p[2]),
CV_RGB(255,255,0), 2, 8, 0 );
                track_window=cvRect(cvRound(p[0])-
30,cvRound(p[1])-30,60,60);
        }
}
center_elipse_x=(float)track_box.center.x;
center_elipse_y=(float)track_box.center.y;
_center_elipse_x=center_elipse_x;
_center_elipse_y=center_elipse_y;

p_mean=cvPoint(center_elipse_x,center_elipse_y);
cvCircle(frame,p_mean,2, CV_RGB(255,0,255), 2, 8, 0 );
cvCircle(frame,pmax,2, CV_RGB(0,255,255), 2, 8, 0 );

cvShowImage( "BackProjection", backproj );
cvShowImage( "Detector", frame );
// Escape
char c = cvWaitKey(1);
if( c == 27 ) break;
timeclock_bucle=clock()-timeclock_bucle;
tiempo_bucle=float(timeclock_bucle)/CLOCKS_PER_SEC*1000;

datosNuevosCAM = 1;

timeclock_stop2=clock();
tiempo_stop=float(timeclock_stop2-timeclock_stop1)/CLOCKS_PER_SEC;
}
cvReleaseCapture( &capture );
cvDestroyWindow( "BackProjection" );
cvDestroyWindow( "Detector" );
//          }
//          }
    }
    else if(pDataArray->idHilo == 2)
    {
```

```
		// Hilo de fusion
		while(1)
		{
			if(pDataArray->semaforo == 0 && datosNuevosIMU == 1 && datosNuevosCAM == 1)
			{

				fprintf(pf,"%.0f,		%.0f,		%.0f,		%3.4f,		%3.4f,
%3.4f\n",_accx,_accy,_accz,_center_elipse_x,_center_elipse_y,float(clock())/CLOCKS_PER_SEC-4);

				datosNuevosIMU = 0;
				datosNuevosCAM = 0;
			}


		}

	}

	fclose(pf);
	return 0;
}



void ErrorHandler(LPTSTR lpszFunction)
{
	// Retrieve the system error message for the last-error code.

	LPVOID lpMsgBuf;
	LPVOID lpDisplayBuf;
	DWORD dw = GetLastError();

	FormatMessage(
		FORMAT_MESSAGE_ALLOCATE_BUFFER |
		FORMAT_MESSAGE_FROM_SYSTEM |
		FORMAT_MESSAGE_IGNORE_INSERTS,
		NULL,
		dw,
		MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
		(LPTSTR) &lpMsgBuf,
		0, NULL );

	// Display the error message.

	lpDisplayBuf = (LPVOID)LocalAlloc(LMEM_ZEROINIT,
		(lstrlen((LPCTSTR) lpMsgBuf) + lstrlen((LPCTSTR) lpszFunction) + 40) * sizeof(TCHAR));
	StringCchPrintf((LPTSTR)lpDisplayBuf,
		LocalSize(lpDisplayBuf) / sizeof(TCHAR),
		TEXT("%s failed with error %d: %s"),
		lpszFunction, dw, lpMsgBuf);
	MessageBox(NULL, (LPCTSTR) lpDisplayBuf, TEXT("Error"), MB_OK);

	// Free error-handling buffer allocations.

	LocalFree(lpMsgBuf);
	LocalFree(lpDisplayBuf);
}

static bool serial_open()
{
	bool fSuccess;
	DCB dcb;

	hCom = CreateFile(DEVICE, GENERIC_READ | GENERIC_WRITE,
		0, NULL, OPEN_EXISTING, 0, 0/*NULL*/);

	if (hCom == INVALID_HANDLE_VALUE) {
```

```
		printf("Cannot open %s\n", DEVICE);
		return FALSE;
	}

	COMMTIMEOUTS timeouts;
	fSuccess = GetCommTimeouts(hCom, &timeouts);
	if (!fSuccess) {
		printf("Cannot get timeouts on %s\n", DEVICE);
		return FALSE;
	}

	// No (minimal) blocking!
	timeouts.ReadIntervalTimeout = 1;
	timeouts.ReadTotalTimeoutMultiplier = 0;
	timeouts.ReadTotalTimeoutConstant = 1;
	timeouts.WriteTotalTimeoutMultiplier = 0;
	timeouts.WriteTotalTimeoutConstant = 0;
	fSuccess = SetCommTimeouts(hCom, &timeouts);

	if (!fSuccess) {
		printf("Cannot set timeouts on %s\n", DEVICE);
		return FALSE;
	}

	fSuccess = GetCommState(hCom, &dcb);
	if (!fSuccess) {
		printf("Cannot comm-state on %s\n", DEVICE);
		return FALSE;
	}

	dcb.BaudRate = BAUDRATE;
	dcb.ByteSize = 8;
	dcb.Parity = NOPARITY;
	dcb.StopBits = ONESTOPBIT;
	dcb.fRtsControl = RTS_CONTROL_DISABLE;
	fSuccess = SetCommState(hCom, &dcb);

	if (!fSuccess) {
		printf("Cannot set comm-state on %s\n", DEVICE);
		return FALSE;
	}

	return TRUE;
};

/* Closes the serial interface. */
static bool serial_close()
{
	if (hCom != INVALID_HANDLE_VALUE) {
		CloseHandle(hCom);
		hCom = INVALID_HANDLE_VALUE;
		return TRUE;
	} else {
		return FALSE;
	}
};

/* Read a couple of bytes from the serial line. */
static int serial_read(byte buffer[], int size)
{
	DWORD cnt = 0;

	BOOL fSucess = ReadFile(hCom, buffer, size, &cnt, NULL);

	if (!fSucess) {
		printf("Cannot read from serial %s\n", DEVICE);
		return 0;
```

```
	}

	return cnt;
}


static int read_and_print_message(float *accx, float *accy, float *accz)
{
	int pos = 0;
	byte buf = 0;
	char msg[16];
	int fin = 0;
	memset(msg, 0, sizeof(msg));
	int retorno = 0;

	while ((pos < 15) && !kbhit())
	{
		if (serial_read(&buf, 1) == 1)
		{
			if (buf=='$')
			{
				pos = 0;
				msg[pos++] = buf;
			}
			else if (buf == '#' && pos > 5)
			{
				msg[pos++] = buf;
				msg[pos] = '\0';

				//printf("CADENA: %s - ",msg);
				char * pch;
				pch = strtok(msg, " $,");
				int contador = 0;

				while (pch != NULL)
				{
					if(contador == 0)
						*accx = stof(pch);
					else if(contador == 1)
						*accy = stof(pch);
					else
						*accz = stof(pch);

					contador++;
	  pch = strtok (NULL, " ,#");
				}

				retorno = 1;
				return retorno;
			}
			else
			{
				msg[pos++] = buf;
			}
		}
	}

	return retorno;
}
```

Anexo C

```
%%%%%%%%%%% Tiempo de Inicio %%%%%%%%%%
Inicio=15/mean(diff(VarName6(1:end-1)));    %Se estable el tiempo de inicio.
% Debe ser por no menos 15 segundos para que la IMU se calibre.
% Recomiendo establecer en 30 segundos.

%%%%%%%%%%%%%% Obtener Aceleraciones en m/s^2 %%%%%%%%%%
Ax=VarName1(Inicio:end)/16384*9.81*2;
Ay=VarName2(Inicio:end)/16384*9.81*2;
Az=VarName3(Inicio:end)/16384*9.81*2;

%%%%%%% Se define la variable Tiempo %%%%%%%%%%
t=VarName6(Inicio:end)-VarName6(round(Inicio));

%%%%%%%%%% Se definen los pixeles de la carga %%%%%%%%%%
Ximg=VarName4(Inicio:end);
Yimg=VarName5(Inicio:end);

%%%%%% Corrección de datos NAN %%%%%%%%
if isnan(t(end))
   t(end)=t(end-1)+(t(end-1)-t(end-2));
   Ax(end)=0;
   Ay(end)=0;
   Az(end)=0;
   Ximg(end)=Ximg(end-1);
   Yimg(end)=Yimg(end-1);
end

%countx = timeseries(Ax,t);
%countmean_x = mean(countx);
%count1ZeroMean_x = countx;
%interval = [0 1];
%Ax = idealfilter(count1ZeroMean_x,interval,'pass')+countmean_x*0;
%Ax=Ax.data;
%county = timeseries(Ay,t);
%countmean_y = mean(county);
%count1ZeroMean_y = county;
%interval = [0 1];
%Ay = idealfilter(count1ZeroMean_y,interval,'pass')+countmean_y*0;
%Ay=Ay.data;
%countz = timeseries(Az,t);
%countmean_z = mean(countz);
%count1ZeroMean_z = countz;
%interval = [0 100];
%Az = idealfilter(count1ZeroMean_z,interval,'pass')+countmean_z*0;
%Az=Az.data*0;


%%%%%%%%%%%%
%%% OBTENCIÓN DE LA POSICIÓN DEL CM Y ÁNGULO BETA DE LA MASA SUSPENDIDA %%%
%%%%%%%%%%%%

%%%%%%%  DATOS INICIALES  %%%%%%%%%%%%%
Cx=625.293341482621/4;  % Centro del plano sensor en eje X
Cy=346.289622915326/4;  % Centro del plano sensor en eje Y
Fx=1091.24858443191/4;  % Fx=F*Sx. Donde F es distancia focal y Sx es pixeles por milímetro en dirección x
Fy=1089.45369875399/4;  % Fy=F*Sy. Donde F es distancia focal y Sy es pixeles por milímetro en dirección y

%%%%%%%%% Coordenadas del punto de cogida %%%%%%
X0=0;
Y0=45;
Z0=-20;

%%%%%%%%% Corrección de distorsión radial %%%%%%
%Kc1=-0.0240113916806306;
```

```
%Kc2=-0.0923769324510727;
%r2=((Ximg-Cx)/Fx).^2+((Yimg-Cy)/Fy).^2;
%r4=r2.^2;%((Ximg-Cx)/Fx).^4+((Yimg-Cy)/Fy).^4;
%Ximg=(1+Kc1*r2+Kc2*r4).*Ximg;
%Yimg=(1+Kc1*r2+Kc2*r4).*Yimg;

%%%%%%%%%%%%%  CALCULO DE POSICION  %%%%%%%%%%
a=(Cx - Ximg).^2/Fx^2 + (Cy - Yimg).^2/Fy^2 + 1;
b=(2*X0*(Cx - Ximg))/Fx - 2*Z0 + (2*Y0*(Cy - Yimg))/Fy;
c=- (L*1000)^2 + X0^2 + Y0^2 + Z0^2;
Z=(-b+sqrt(b.^2-4*a*c))./(2*a);
X=Z.*(Ximg-Cx)/Fx;
Y=Z.*(Yimg-Cy)/Fy;

%%%%%%%%%%%  Corrección de Posición  %%%%%%%%%
%%%% Esto es debido a que la webcam no mira exactamente en dirección Z %%%%
X=X-(mean(X(1:30))-X0);
Y=Y-(mean(Y(1:30))-Y0);

%%%%%%  CALCULO DE ANGULO  %%%%%%%%%%%
Pitch=atan((X0-X)./(-Z0+Z));
Roll=atan((Y0-Y)./(-Z0+Z));
```

Anexo D

```
%%%%%%%%%%%%%% Sistema Linealizado %%%%%%%%%%%%
Ac =[ -B/(L^2*mL), -g/L,          0,    0;
        1,    0,          0,    0;
        0,    0, -B/(L^2*mL), -g/L;
        0,    0,          1,    0]


Bc =[ B/(L^2*mL),          0,   0, -1/L, 0;
        0,          0,   0,    0, 0;
        0, B/(L^2*mL), 1/L,    0, 0;
        0,          0,   0,    0, 0]


Cc =[     0     1     0     0;
    0     0     0     1]


Dc =[     0     0     0     0     0;
    0     0     0     0     0]

%%%%%%%  Sistema Linealizado y discretizado %%%%%%%%%%%%%
Ts=mean(diff(t));   % Tiempo de muestreo aproximado
Ad=expm(Ac*Ts);
Bd=inv(Ac)*(Ad-eye(4))*Bc;
Cd=Cc;
Dd=Dc;

%%%%%%%%%%%%%  Comparación angulos según Camara VS Simulación %%%%%%%%%%%%%
figure;
plot(0:Ts:Ts*(length(Roll)-1),Roll*180/pi,'k','LineWidth',2);
hold on
plot(0:Ts:Ts*(length(Roll)-1),Roll_Model,'b','LineWidth',2);
legend('Roll Camara','Roll Simulink');
title('ROLL','FontName','Arial','FontSize', 30);
xlabel('Tiempo en seg.','FontName','Arial','FontSize', 24);
ylabel('Ángulo roll en ° sexagesimales','FontName','Arial','FontSize', 24);

hold off

grid

figure;
plot(0:Ts:Ts*(length(Pitch)-1),Pitch*180/pi,'k','LineWidth',2);
hold on
plot(0:Ts:Ts*(length(Pitch)-1),Pitch_Model,'b','LineWidth',2);
legend('Pitch Camara','Pitch Simulink');
title('PITCH','FontName','Arial','FontSize', 30);
xlabel('Tiempo en seg.','FontName','Arial','FontSize', 24);
ylabel('Ángulo pitch en ° sexagesimales','FontName','Arial','FontSize', 24);
grid
hold off;

%%%%%%%%%%%%
%%%%%%%  CALCULO DE COEFICIENTES DEL FILTRO %%%%%%%%%%%%
%%%%%%%%%%%%

%%%%%%%%%%%  Filtrado de la aceleración %%%%%%%%
countx = timeseries(Ax,t);  % Creación de serie de tiempo
countmean_x = mean(countx); % Valor medio
count1ZeroMean_x = countx;
interval = [0 1];           % Frecuencias que pasan
Ax_filter = idealfilter(count1ZeroMean_x,interval,'pass')+countmean_x*0;    % Filtrado de la señal
Ax_filter=Ax_filter.data;   % Extracción de datos
```

```
county = timeseries(Ay,t);  % Creación de serie de tiempo
countmean_y = mean(county); % Valor medio
count1ZeroMean_y = county;
interval = [0 1];           % Frecuencias que pasan
Ay_filter = idealfilter(count1ZeroMean_y,interval,'pass')+countmean_y*0;    % Filtrado de la señal
Ay_filter=Ay_filter.data;   % Extracción de datos

varx=var(Ax-Ax_filter);     % Varianza del ruido del sensor en dirección X
vary=var(Ax-Ax_filter);     % Varianza del ruido del sensor en dirección Y

%%%%%%%%%%%%%  Matriz de covarianzas de ruido del sensor %%%%%%%%%
Q=[vary*Bd(1,4)^2 sqrt(vary*Bd(1,4)^2)*sqrt(vary*Bd(2,4)^2) 0 0;
sqrt(vary*Bd(1,4)^2)*sqrt(vary*Bd(2,4)^2) vary*Bd(2,4)^2 0 0;
0 0 varx*Bd(3,3)^2 sqrt(varx*Bd(3,3)^2)*sqrt(varx*Bd(4,3)^2);
0 0 sqrt(varx*Bd(3,3)^2)*sqrt(varx*Bd(4,3)^2) varx*Bd(4,3)^2];

%%%%%%%%%%%%  Matriz de covarianzas de ruido de la cámara %%%%%%%%
R=[(0.5/180*pi)^2 0; 0 (0.5/180*pi)^2];


%%%%%%%%%%%%  ELIMINACIÓN DE DATOS DE ANGULOS DE LA CARGA %%%%%%%%
Roll_max=5;     % Se delimita el angulo roll máximo premitido
Roll=Roll-Roll.*(Roll>(Roll_max*pi/180))+(Roll>(Roll_max*pi/180)).*(Roll_max*pi/180);
Roll=Roll-Roll.*(Roll<(-Roll_max*pi/180))-(Roll<(-Roll_max*pi/180)).*(Roll_max*pi/180);

Pitch_max=6;   % Se delimita el angulo pitch máximo premitido
Pitch=Pitch-Pitch.*(Pitch>(Pitch_max*pi/180))+(Pitch>(Pitch_max*pi/180)).*(Pitch_max*pi/180);
Pitch=Pitch-Pitch.*(Pitch<(-Pitch_max*pi/180))-(Pitch<(-Pitch_max*pi/180)).*(Pitch_max*pi/180);

%%%%%%%%%%%  Construcción de variables de entrada y salida %%%%%%%
U=[zeros(length(Ax),2) Ax Ay Az];
Y=[Roll Pitch];

%%%%%%%  IMPLEMENTACIÓN DEL FILTRO DE KALMAN %%%%%%%%%%%%
m=1;
P=Q;
X=[0; 0; 0; 0];
%duration=t(end);
%for k=0:Ts:duration
for k=0:Ts:30-Ts

   if m>length(Y)
      U(m,:)=[0 0 0 0 0];
   end

   X(:,m)=Ad*X(:,m)+Bd*U(m,:)';
   P=Ad*P*Ad'+Q;
   K=P*Cd'*inv(Cd*P*Cd'+R);
   P=(eye(4)-K*Cd)*P;

   if (abs(X(2,m))>=(Roll_max*pi/180))|(abs(X(4,m))>=(Pitch_max*pi/180))|m>length(Y)
      Ymod=Cd*X(:,m);
      X(:,m)=X(:,m)+K*(Ymod-Cd*X(:,m));
   else
      X(:,m)=X(:,m)+K*(Y(m,:)'-Cd*X(:,m));
   end

   X(:,m+1)=X(:,m);
   m=m+1;

end

%%%%%%  Comparación angulos según Camara VS Kalman %%%%%%
figure;
plot(0:Ts:Ts*(length(X)-1),X(2,:)'*180/pi,'r','LineWidth',2);
hold on
plot(0:Ts:Ts*(length(Y)-1),Y(:,1)*180/pi,'k','LineWidth',2);
```

```
legend('Roll Kalman','Roll Camara');
title('ROLL','FontName','Arial','FontSize', 30);
xlabel('Tiempo en seg.','FontName','Arial','FontSize', 24);
ylabel('Ángulo roll en ° sexagesimales','FontName','Arial','FontSize', 24);
grid
hold off
figure;
plot(0:Ts:Ts*(length(X)-1),X(4,:)'*180/pi,'r','LineWidth',2);
hold on
plot(0:Ts:Ts*(length(Y)-1),Y(:,2)*180/pi,'k','LineWidth',2);
legend('Pitch Kalman','Pitch Camara');
title('PITCH','FontName','Arial','FontSize', 30);
xlabel('Tiempo en seg.','FontName','Arial','FontSize', 24);
ylabel('Ángulo pitch en ° sexagesimales','FontName','Arial','FontSize', 24);
grid
hold off;

[r2_Roll rmse_Roll] = rsquare(Roll,Roll_Model/180*pi)
[r2_Pitch rmse_Pitch] = rsquare(Pitch,Pitch_Model/180*pi)
```

Anexo E

IEEE CHILECON2015 - ISSN xxxx-xxxx, Pág. xxxx.xxx. Santiago de Chile, 28 al 30 de Octubre 2015

Modeling and PID cascade control of a Quadcopter for trajectory tracking
(Modelado y Control PID en cascada de un Cuadricóptero para seguimiento de trayectorias)

Ernesto A. Paiva, Juan C. Soto, Julio A. Salinas and William Ipanaqué

Abstract— **This paper presents the mathematical model of a quadricopter under the Newton-Euler formulation. A Transfer function has been chosen, which represents a brushless motor and its driver as one system. PID cascade control had been designed to solve the path tracking problem for a quadcopter. The controller is evaluated in a 3D environment in Simulink.**

Index Terms—— **Unmanned aerial vehicles, modeling, simulation, position control, PID cascade.**

I. Introducción

En el 2004, investigaciones realizadas en la Escuela Politécnica Federal de Zúrich dieron origen a publicaciones como [1] donde se presenta un cuadricóptero llamado "*Omnidirectional Stationary Flying Outstretched Robot*" (OS4) desarrollado en su laboratorio. Así mismo se desarrolla el modelo aerodinámico y se considera una función transferencia de los rotores mostrando resultados en simulaciones y experimentales.

En el mismo año Bouabdallah [2] presenta simulaciones y resultados experimentales del OS4 implementando el algorítmo de control clásico PID y control avanzado LQ. En el 2005, [3] presenta mejoras en el hardware del OS4 y resultados experimentales. Además de esto se muestra una simulación en tres dimensiones en el software Webots. En el mismo año, en [4] se muestran los resultados de dos técnicas de control no lineales (*Backstepping* y *Sliding-mode*) aplicadas al cuadricóptero. También en [4] se presenta un esquema de control en cascada para el control de posición. Años después, en [5] se introduce un modelo de simulación que tiene en cuenta la variación de los coeficientes aerodinámicos debido al movimiento del vehículo.

Por otra parte, en [6] donde se implementa un "*Visual servo algorithms*" basado en imágenes para un vehículo aéreo no tripulado (VANT) con una cámara bordo del vehículo. En el mismo año, en [7] se controla la estabilidad del vuelo y del aterrizaje utilizando una cámara y un sensor de orientación.

En [8] se investiga una serie de "*Visual servo algorithms*" basados en imágenes para la regulación de la posición de un VANT. Otro trabajo similar pero considerando una plataforma en movimiento es [9].

Se han publicado otros trabajos citados muchas veces, como el grupo de investigación encabezado por Altuğ y Ostrowski enfocado al control visual de cuadricóptero, [10] y [11]. Estudios realizados sobre control de trayectorias en cuadricóptero se tiene trabajos como [12], [13] y [14].

Este trabajo presenta el modelamiento aerodinámico de un cuadricóptero y de sus motores. Se resuelve el problema de seguimiento de trayectoria por medio de controladores PID en cascada para los subsistemas de posición y orientación. La sintonización de los controladores se determina mediante las funciones de costo propuestas.

En la sección II se explica el modelo aerodinámico de un cuadricóptero para las configuraciones en cruz y en equis bajo la formulación de Newton-Euler. En la sección III se utiliza el enfoque experimental para determinar las constantes del modelo del cuadricóptero. En la sección IV se explica la lógica de control de orientación. Se inserta la dinámica de los motores a las ecuaciones que gobiernan la orientación muy pocas veces considerada en trabajos anteriores. En la sección V se explica la lógica de control de posición. Se presentan las ecuaciones utilizadas para sintonizar los controladores PID a partir de funciones de costo. Finalmente en la sección VI se muestran los resultados del controlador PID en cascada por medio de simulaciones. Para comprobar sus buenas prestaciones, se adhieren disturbios al modelo matemático del cuadricóptero.

II. Modelado Matemático del Quadricóptero

A. Configuración del Quadricóptero

El cuadricóptero se puede describir como un VANT con cuatro hélices configuradas en cruz o en equis. En la configuración en cruz los dos pares de hélices (1,3) y (2,4) giran en direcciones opuestas, lo cual elimina la necesidad de un rotor de cola [15], ver Fig. 1. Mediante la variación de la velocidad del rotor, se puede cambiar la fuerza de empuje y crear movimiento [1], ver Fig. 1.

Manuscript received September 25, 2015. This work was supported in part by Laboratory of Automatic Control Systems, Universidad de Piura.

E. A. Paiva is with Laboratory of Automatic Control Systems, Universidad de Piura (e-mail: ernesto.paiva@posgrado.udep.edu.pe).

J. C. Soto is with Laboratory of Automatic Control Systems, Universidad de Piura (e-mail: juan.soto@udep.pe)

J. A. Salinas is with Laboratory of Automatic Control Systems, Universidad de Piura (e-mail: julio.salinas@pregrado.udep.edu.pe)

W. Ipanaqué is with Laboratory of Automatic Control Systems, Universidad de Piura (e-mail: william.ipanaque@udep.pe)

Anexo F

```
/*
MultiWiiCopter by Alexandre Dubus
www.multiwii.com
March  2012     V2.0
 This program is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 any later version. see <http://www.gnu.org/licenses/>
Edit by Ernesto Paiva
Octubre  2015     V2.0
*/

#include "config.h"
#include "def.h"
#include <avr/pgmspace.h>
#define  VERSION  20

/*********** RC alias *****************/
#define ROLL       0
#define PITCH      1
#define YAW        2
#define THROTTLE   3
#define AUX1       4
#define AUX2       5
#define AUX3       6
#define AUX4       7

#define PIDALT     3
#define PIDVEL     4
#define PIDGPS     5
#define PIDLEVEL   6
#define PIDMAG     7

#define BOXACC       0
#define BOXBARO      1
#define BOXMAG       2
#define BOXCAMSTAB   3
#define BOXCAMTRIG   4
#define BOXARM       5
#define BOXGPSHOME   6
#define BOXGPSHOLD   7
#define BOXPASSTHRU  8
#define BOXHEADFREE  9
#define BOXBEEPERON  10

#define CHECKBOXITEMS 11
#define PIDITEMS 8

static uint32_t currentTime = 0;
static uint16_t previousTime = 0;
static uint16_t cycleTime = 0;     // this is the number in micro second to achieve a full loop, it can differ a little and is taken
into account in the PID loop
static uint16_t calibratingA = 0;  // the calibration is done is the main loop. Calibrating decreases at each cycle down to 0,
then we enter in a normal mode.
static uint8_t  calibratingM = 0;
static uint16_t calibratingG;
static uint8_t  armed = 0;
static uint16_t acc_1G;            // this is the 1G measured acceleration
static int16_t  acc_25deg;
static uint8_t  nunchuk = 0;
static uint8_t  accMode = 0;       // if level mode is a activated
static uint8_t  magMode = 0;       // if compass heading hold is a activated
static uint8_t  baroMode = 0;      // if altitude hold is activated
static uint8_t  GPSModeHome = 0;   // if GPS RTH is activated
```

```
static uint8_t  GPSModeHold = 0;    // if GPS PH is activated
static uint8_t  headFreeMode = 0;   // if head free mode is a activated
static uint8_t  passThruMode = 0;   // if passthrough mode is activated
static int16_t  headFreeModeHold;
static int16_t  gyroADC[3],accADC[3],accSmooth[3],magADC[3];
static int16_t  accTrim[2] = {0, 0};
static int16_t  heading,magHold;
static uint8_t  calibratedACC = 0;
static uint8_t  vbat;               // battery voltage in 0.1V steps
static uint8_t  okToArm = 0;
static uint8_t  rcOptions[CHECKBOXITEMS];
static int32_t  BaroAlt;
static int32_t  EstAlt;             // in cm
static int16_t  BaroPID = 0;
static int32_t  AltHold;
static int16_t  errorAltitudeI = 0;
static uint8_t  buzzerState = 0;
static int16_t  debug1,debug2,debug3,debug4;

//for log
static uint16_t cycleTimeMax = 0;       // highest ever cycle timen
static uint16_t cycleTimeMin = 65535;   // lowest ever cycle timen
static uint16_t powerMax = 0;           // highest ever current

static int16_t  i2c_errors_count = 0;
static int16_t  annex650_overrun_count = 0;

// ***********************
//Automatic ACC Offset Calibration
// ***********************
static uint16_t InflightcalibratingA = 0;
static int16_t AccInflightCalibrationArmed;
static uint16_t AccInflightCalibrationMeasurementDone = 0;
static uint16_t AccInflightCalibrationSavetoEEProm = 0;
static uint16_t AccInflightCalibrationActive = 0;

// ***********************
// power meter
// ***********************
#define PMOTOR_SUM 8                    // index into pMeter[] for sum
static uint32_t pMeter[PMOTOR_SUM + 1];  // we use [0:7] for eight motors,one extra for sum
static uint8_t pMeterV;                 // dummy to satisfy the paramStruct logic in ConfigurationLoop()
static uint32_t pAlarm;                 // we scale the eeprom value from [0:255] to this value we can directly compare to the
sum in pMeter[6]
static uint8_t powerTrigger1 = 0;       // trigger for alarm based on power consumption
static uint16_t powerValue = 0;         // last known current
static uint16_t intPowerMeterSum, intPowerTrigger1;

// ***********************
// telemetry
// ***********************
static uint8_t telemetry = 0;
static uint8_t telemetry_auto = 0;

// *******************
// rc functions
// *******************
#define MINCHECK 1100
#define MAXCHECK 1900

volatile int16_t failsafeCnt = 0;
static int16_t failsafeEvents = 0;
static int16_t rcData[8];    // interval [1000;2000]
static int16_t rcCommand[4]; // interval [1000;2000] for THROTTLE and [-500;+500] for ROLL/PITCH/YAW
static uint8_t rcRate8;
static uint8_t rcExpo8;
static int16_t lookupRX[7]; //  lookup table for expo & RC rate
```

```
volatile uint8_t rcFrameComplete; //for serial rc receiver Spektrum


// ***************
// gyro+acc IMU
// ***************
static int16_t gyroData[3] = {0,0,0};
static int16_t gyroZero[3] = {0,0,0};
static int16_t accZero[3]  = {0,0,0};
static int16_t magZero[3]  = {0,0,0};
static int16_t angle[2]    = {0,0};  // absolute angle inclination in multiple of 0.1 degree    180 deg = 1800
static int8_t  smallAngle25 = 1;

// NUESTRO
static int16_t u[3] = {0,0,0};
static int16_t errorAngle_ant[3] = {0,0,0};
static uint8_t D28[3] = {0,0,0}; // 8 bits is much faster and the code is much shorter
static int16_t gyroData_ant[3] = {0,0,0};
static int16_t gyroData_ant_2[3] = {0,0,0};
static int16_t gyroData_ant_3[3] = {0,0,0};
static int16_t gyroData_ant_4[3] = {0,0,0};
static int16_t gyroData_ant_5[3] = {0,0,0};
static int16_t gyroData_ant_6[3] = {0,0,0};
static int16_t gyroData_ant_7[3] = {0,0,0};
static int16_t gyroData_ant_8[3] = {0,0,0};
static int16_t gyroData_ant_9[3] = {0,0,0};
static int16_t gyroData_ant_10[3] = {0,0,0};
static int16_t u_ant[3] = {0,0,0};
// Nuestro para altura
static int16_t Alt_ant[1] = {0};
static int16_t Alt_ant_2[1] = {0};
static int16_t Alt_ant_3[1] = {0};
static int16_t Alt_ant_4[1] = {0};
static int16_t errorAlt_ant[1] = {0};
int16_t errorAlt; // Nuestro
int16_t PTermAlt,ITermAlt,DTermAlt;
int16_t contador=1;
int16_t a_z=20;

// **************************
// motor and servo functions
// **************************
static int16_t axisPID[3];
static int16_t motor[8];
static int16_t servo[8] = {1500,1500,1500,1500,1500,1500,1500,1500};
static uint16_t wing_left_mid  = WING_LEFT_MID;
static uint16_t wing_right_mid = WING_RIGHT_MID;
static uint16_t tri_yaw_middle = TRI_YAW_MIDDLE;

// **********************
// EEPROM & LCD functions
// **********************
static uint8_t P8[8], I8[8], D8[8]; // 8 bits is much faster and the code is much shorter
static uint8_t dynP8[3], dynI8[3], dynD8[3];
static uint8_t rollPitchRate;
static uint8_t yawRate;
static uint8_t dynThrPID;
static uint8_t activate1[CHECKBOXITEMS];
static uint8_t activate2[CHECKBOXITEMS];

// **********************
// GPS
// **********************
static int32_t  GPS_latitude,GPS_longitude;
static int32_t  GPS_latitude_home,GPS_longitude_home;
static int32_t  GPS_latitude_hold,GPS_longitude_hold;
static uint8_t  GPS_fix , GPS_fix_home = 0;
```

```
static uint8_t  GPS_numSat;
static uint16_t GPS_distanceToHome,GPS_distanceToHold;       // distance to home or hold point in meters
static int16_t  GPS_directionToHome,GPS_directionToHold;     // direction to home or hol point in degrees
static uint16_t GPS_altitude,GPS_speed;                      // altitude in 0.1m and speed in 0.1m/s - Added by Mis
static uint8_t  GPS_update = 0;                              // it's a binary toogle to distinct a GPS position update
static int16_t  GPS_angle[2];                                // it's the angles that must be applied for GPS correction

void blinkLED(uint8_t num, uint8_t wait,uint8_t repeat) {
  uint8_t i,r;
  for (r=0;r<repeat;r++) {
    for(i=0;i<num;i++) {
      LEDPIN_TOGGLE; //switch LEDPIN state
      BUZZERPIN_ON;
      delay(wait);
      BUZZERPIN_OFF;
    }
    delay(60);
  }
}

void annexCode() { // this code is excetuted at each loop and won't interfere with control loop if it lasts less than 650
microseconds
  static uint32_t buzzerTime,calibratedAccTime;
  #if defined(LCD_TELEMETRY)
   static uint16_t telemetryTimer = 0, telemetryAutoTimer = 0, psensorTimer = 0;
  #endif
  #if defined(LCD_TELEMETRY_AUTO)
   static uint8_t telemetryAutoIndex = 0;
   static char telemetryAutoSequence []  = LCD_TELEMETRY_AUTO;
  #endif
  #ifdef VBAT
    static uint8_t vbatTimer = 0;
  #endif
  static uint8_t  buzzerFreq;        // delay between buzzer ring
  uint8_t axis,prop1,prop2;
  #if defined(POWERMETER_HARD)
    uint16_t pMeterRaw;              // used for current reading
  #endif

  // PITCH & ROLL only dynamic PID adjustemnt,  depending on throttle value
  if      (rcData[THROTTLE]<1500) prop2 = 100;
  else if (rcData[THROTTLE]<2000) prop2 = 100 - (uint16_t)dynThrPID*(rcData[THROTTLE]-1500)/500;
  else                            prop2 = 100 - dynThrPID;

  for(axis=0;axis<3;axis++) {
    uint16_t tmp = min(abs(rcData[axis]-MIDRC),500);
    #if defined(DEADBAND)
      if (tmp>DEADBAND) { tmp -= DEADBAND; }
      else { tmp=0; }
    #endif
    if(axis!=2) { //ROLL & PITCH
      uint16_t tmp2 = tmp/100;
      rcCommand[axis] = lookupRX[tmp2] + (tmp-tmp2*100) * (lookupRX[tmp2+1]-lookupRX[tmp2]) / 100;
      prop1 = 100-(uint16_t)rollPitchRate*tmp/500;
      prop1 = (uint16_t)prop1*prop2/100;
    } else { //YAW
      rcCommand[axis] = tmp;
      prop1 = 100-(uint16_t)yawRate*tmp/500;
    }
    dynP8[axis] = (uint16_t)P8[axis]*prop1/100;
    dynD8[axis] = (uint16_t)D8[axis]*prop1/100;
    if (rcData[axis]<MIDRC) rcCommand[axis] = -rcCommand[axis];
  }
  rcCommand[THROTTLE]        =        MINTHROTTLE        +        (int32_t)(MAXTHROTTLE-MINTHROTTLE)*
(rcData[THROTTLE]-MINCHECK)/(2000-MINCHECK); // [1000 a 2000]

  if(headFreeMode) {
```

```
  float radDiff = (heading - headFreeModeHold) * 0.0174533f; // where PI/180 ~= 0.0174533
  float cosDiff = cos(radDiff);
  float sinDiff = sin(radDiff);
  int16_t rcCommand_PITCH = rcCommand[PITCH]*cosDiff + rcCommand[ROLL]*sinDiff;
  rcCommand[ROLL] =  rcCommand[ROLL]*cosDiff - rcCommand[PITCH]*sinDiff;
  rcCommand[PITCH] = rcCommand_PITCH;
 }

 #if defined(POWERMETER_HARD)
  if (! (++psensorTimer % PSENSORFREQ)) {
   pMeterRaw =  analogRead(PSENSORPIN);
   powerValue = ( PSENSORNULL > pMeterRaw ? PSENSORNULL - pMeterRaw : pMeterRaw - PSENSORNULL); //
do not use abs(), it would induce implicit cast to uint and overrun
   if ( powerValue < 333) {  // only accept reasonable values. 333 is empirical
   #ifdef LOG_VALUES
     if (powerValue > powerMax) powerMax = powerValue;
   #endif
   } else {
     powerValue = 333;
   }
   pMeter[PMOTOR_SUM] += (uint32_t) powerValue;
 }
 #endif

 #if defined(VBAT)
  static uint8_t ind = 0;
  uint16_t vbatRaw = 0;
  static uint16_t vbatRawArray[8];
  if (! (++vbatTimer % VBATFREQ)) {
          vbatRawArray[(ind++)%8] = analogRead(V_BATPIN);
          for (uint8_t i=0;i<8;i++) vbatRaw += vbatRawArray[i];
          vbat = vbatRaw / (VBATSCALE/2);              // result is Vbatt in 0.1V steps
  }
  if ( rcOptions[BOXBEEPERON] ){ // unconditional beeper on via AUXn switch
    buzzerFreq = 7;
  } else  if ( ( (vbat>VBATLEVEL1_3S)
  #if defined(POWERMETER)
                 && ( (pMeter[PMOTOR_SUM] < pAlarm) || (pAlarm == 0) )
  #endif
                )  || (NO_VBAT>vbat)                        ) // ToLuSe
  {                                  // VBAT ok AND powermeter ok, buzzer off
    buzzerFreq = 0; buzzerState = 0; BUZZERPIN_OFF;
  #if defined(POWERMETER)
  } else if (pMeter[PMOTOR_SUM] > pAlarm) {                        // sound alarm for powermeter
    buzzerFreq = 4;
  #endif
  } else if (vbat>VBATLEVEL2_3S) buzzerFreq = 1;
  else if (vbat>VBATLEVEL3_3S)   buzzerFreq = 2;
  else                       buzzerFreq = 4;
  if (buzzerFreq) {
    if (buzzerState && (currentTime > buzzerTime + 250000) ) {
      buzzerState = 0;
      BUZZERPIN_OFF;
      buzzerTime = currentTime;
    } else if ( !buzzerState && (currentTime > (buzzerTime + (2000000>>buzzerFreq))) ) {
      buzzerState = 1;
      BUZZERPIN_ON;
      buzzerTime = currentTime;
    }
  }
 #endif

 if ( (calibratingA>0 && (ACC || nunchuk) ) || (calibratingG>0) ) {  // Calibration phasis
  LEDPIN_TOGGLE;
 } else {
  if (calibratedACC == 1) {LEDPIN_OFF;}
  if (armed) {LEDPIN_ON;}
```

```
  }

  #if defined(LED_RING)
    static uint32_t LEDTime;
    if ( currentTime > LEDTime ) {
      LEDTime = currentTime + 50000;
      i2CLedRingState();
    }
  #endif

  if ( currentTime > calibratedAccTime ) {
    if (smallAngle25 == 0) {
      calibratedACC = 0; // the multi uses ACC and is not calibrated or is too much inclinated
      LEDPIN_TOGGLE;
      calibratedAccTime = currentTime + 500000;
    } else
      calibratedACC = 1;
  }

  serialCom();

  #if defined(POWERMETER)
    intPowerMeterSum = (pMeter[PMOTOR_SUM]/PLEVELDIV);
    intPowerTrigger1 = powerTrigger1 * PLEVELSCALE;
  #endif

  #ifdef LCD_TELEMETRY_AUTO
    if ( (telemetry_auto) && (! (++telemetryAutoTimer % LCD_TELEMETRY_AUTO_FREQ) )  ){
      telemetry = telemetryAutoSequence[++telemetryAutoIndex % strlen(telemetryAutoSequence)];
      LCDclear(); // make sure to clear away remnants
    }
  #endif
  #ifdef LCD_TELEMETRY
    if (! (++telemetryTimer % LCD_TELEMETRY_FREQ)) {
      #if (LCD_TELEMETRY_DEBUG+0 > 0)
        telemetry = LCD_TELEMETRY_DEBUG;
      #endif
      if (telemetry) lcd_telemetry();
    }
  #endif

  #if GPS
    static uint32_t GPSLEDTime;
    if ( currentTime > GPSLEDTime && (GPS_fix_home == 1)) {
      GPSLEDTime = currentTime + 150000;
      LEDPIN_TOGGLE;
    }
  #endif
}


void setup() {
  SerialOpen(0,SERIAL_COM_SPEED);
  LEDPIN_PINMODE;
  POWERPIN_PINMODE;
  BUZZERPIN_PINMODE;
  STABLEPIN_PINMODE;
  POWERPIN_OFF;
  initOutput();
  readEEPROM();
  checkFirstTime();
  configureReceiver();
  initSensors();
  previousTime = micros();
  #if defined(GIMBAL)
   calibratingA = 400;
  #endif
```

```
  calibratingG = 400;
  #if defined(POWERMETER)
    for(uint8_t i=0;i<=PMOTOR_SUM;i++)
      pMeter[i]=0;
  #endif
  #if defined(GPS_SERIAL)
    SerialOpen(GPS_SERIAL,GPS_BAUD);
  #endif
  #if defined(LCD_ETPP)
    initLCD();
  #elif defined(LCD_LCD03)
    initLCD();
  #endif
  #ifdef LCD_TELEMETRY_DEBUG
    telemetry_auto = 1;
  #endif
  #ifdef LCD_CONF_DEBUG
    configurationLoop();
  #endif
  ADCSRA |= _BV(ADPS2) ; ADCSRA &= ~_BV(ADPS1); ADCSRA &= ~_BV(ADPS0); // this speeds up analogRead
without loosing too much resolution: http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1208715493/11
// Nuestro
P8[0] = 4.4162;//9.9;8.2130;//4.15;//9;//9.1256;//2.7103;//8.2130;//9.1256;//27;//9.03;
P8[1] = 4.4162;//9.9;8.2130;//4.15;//9;//9.1256;//2.7103;//8.2130;//9.1256;//27;//9.0;
P8[2] = 8.27;// P8[2] = P_Yaw*100
I8[0] = 6.2626;//7.8;5.3264;//2.08;//23/5;//18.6770;//1.0821;//0.4381;//4.3810;//23;//8.90;
I8[1] = 6.2626;//7.8;5.3264;//2.08;//23/5;//18.6770;//1.0821;//0.4381;//4.3810;//23;//8.90;
I8[2] = 5.26;// I8[2] = I_Yaw*100
D8[0]                                                                                                    =
1.3966*2.4414;//2.5*2.4414;3.2852*2.4414;//1.46*2.4414;//5;//2.6352;//1.1716;//2.0555;//2.2838;//5;//1.58;//5.8560;
D8[1]                                                                                                    =
1.3966*2.4414;//2.5*2.4414;3.2852*2.4414;//1.46*2.4414;//5;//2.6352;//1.1716;//2.0555;//2.2838;//5;//1.58;//5.8560;
D8[2] = 6.25*2.4414;// D8[2] = D_Yaw*100
D28[0]   =   50*2.4414;//74*2.4414;592*2.4414;//56*2.4414;//26*1.5;//88;//0;//131.8;//119.8;//266;//32.3;//   D28[0]   =
D2_Roll*1000
D28[1]   =   50*2.4414;//74*2.4414;592*2.4414;//56*2.4414;//26*1.5;//88;//0;//131.8;//119.8;//266;//32.3;//   D28[1]   =
D2_Pitch*1000
D28[2] = 0;// D28[2] = D2_Yaw*100*1000
//Los parámetros que afectan medidas del giroscopio deben ser multiplicados por (2000/8192)/0.1 = 2.4414
// Nuestro Altura
P8[3] = 3.59;
I8[3] = 0.0320;
D8[3] = 2.3933;
}



// ******** Main Loop *********
void loop () {
  static uint8_t rcDelayCommand; // this indicates the number of time (multiple of RC measurement at 50Hz) the sticks
must be maintained to run or switch off motors
  uint8_t axis,i;
  int16_t error,errorAngle;
  int16_t delta,deltaSum;
  int16_t PTerm,ITerm,DTerm;
  int16_t D2Term;  // Nuestro
  static int16_t lastGyro[3] = {0,0,0};
  static int16_t delta1[3],delta2[3];
  static int16_t errorGyroI[3] = {0,0,0};
  static int16_t errorAngleI[3] = {0,0,0}; // Nuestro
  static uint32_t rcTime  = 0;
  static int16_t initialThrottleHold;

  #if defined(SPEKTRUM)
    if (rcFrameComplete) computeRC();
  #endif
```

```
 if (currentTime > rcTime ) { // 50Hz
   rcTime = currentTime + 20000;
   #if !(defined(SPEKTRUM) ||defined(BTSERIAL))
     computeRC();
   #endif
   // Failsafe routine - added by MIS
   #if defined(FAILSAFE)
     if ( failsafeCnt > (5*FAILSAVE_DELAY) && armed==1) {                // Stabilize, and set Throttle to specified level
       for(i=0; i<3; i++) rcData[i] = MIDRC;                        // after specified guard time after RC signal is lost (in 0.1sec)
       rcData[THROTTLE] = FAILSAVE_THR0TTLE;
       if (failsafeCnt > 5*(FAILSAVE_DELAY+FAILSAVE_OFF_DELAY)) {           // Turn OFF motors after specified
Time (in 0.1sec)
         armed = 0;   // This will prevent the copter to automatically rearm if failsafe shuts it down and prevents
         okToArm = 0; // to restart accidentely by just reconnect to the tx - you will have to switch off first to rearm
       }
       failsafeEvents++;
     }
     failsafeCnt++;
   #endif
   // end of failsave routine - next change is made with RcOptions setting
   if (rcData[THROTTLE] < MINCHECK) {
     errorGyroI[ROLL] = 0; errorGyroI[PITCH] = 0; errorGyroI[YAW] = 0;
     errorAngleI[ROLL] = 0; errorAngleI[PITCH] = 0;
     errorAngleI[YAW] = 0; //Nuestro
     rcDelayCommand++;
     if (rcData[YAW] < MINCHECK && rcData[PITCH] < MINCHECK && armed == 0) {
       if (rcDelayCommand == 20) calibratingG=400;
     } else if (rcData[YAW] > MAXCHECK && rcData[PITCH] > MAXCHECK && armed == 0) {
       if (rcDelayCommand == 20) {
         #ifdef TRI
           servo[5] = 1500; // we center the yaw servo in conf mode
           writeServos();
         #endif
         #ifdef FLYING_WING
           servo[0]  = wing_left_mid;
           servo[1]  = wing_right_mid;
           writeServos();
         #endif
         #if defined(LCD_CONF)
           configurationLoop(); // beginning LCD configuration
         #endif
         previousTime = micros();
       }
     }
     #if defined(InflightAccCalibration)
       else if (armed == 0 && rcData[YAW] < MINCHECK && rcData[PITCH] > MAXCHECK && rcData[ROLL] >
MAXCHECK){
         if (rcDelayCommand == 20){
           if (AccInflightCalibrationMeasurementDone){                // trigger saving into eeprom after landing
             AccInflightCalibrationMeasurementDone = 0;
             AccInflightCalibrationSavetoEEProm = 1;
           }else{
             AccInflightCalibrationArmed = !AccInflightCalibrationArmed;
             if (AccInflightCalibrationArmed){blinkLED(10,1,2);}else{blinkLED(10,10,3);}
           }
         }
       }
    #endif
     else if ((activate1[BOXARM] > 0) || (activate2[BOXARM] > 0)) {
       if ( rcOptions[BOXARM] && okToArm ) {
         armed = 1;
         headFreeModeHold = heading;
       } else if (armed) armed = 0;
       rcDelayCommand = 0;
     } else if ( (rcData[YAW] < MINCHECK || rcData[ROLL] < MINCHECK)  && armed == 1) {
       if (rcDelayCommand == 20) armed = 0; // rcDelayCommand = 20 => 20x20ms = 0.4s = time to wait for a specific
RC command to be acknowledged
```

```
    } else if ( (rcData[YAW] > MAXCHECK || rcData[ROLL] > MAXCHECK) && rcData[PITCH] < MAXCHECK &&
armed == 0 && calibratingG == 0 && calibratedACC == 1) {
      if (rcDelayCommand == 20) {
        armed = 1;
        headFreeModeHold = heading;
      }
    #ifdef LCD_TELEMETRY_AUTO
     } else if (rcData[ROLL] < MINCHECK && rcData[PITCH] > MAXCHECK && armed == 0) {
      if (rcDelayCommand == 20) {
        if (telemetry_auto) {
          telemetry_auto = 0;
          telemetry = 0;
        } else
          telemetry_auto = 1;
      }
    #endif
     } else
      rcDelayCommand = 0;
    } else if (rcData[THROTTLE] > MAXCHECK && armed == 0) {
     if (rcData[YAW] < MINCHECK && rcData[PITCH] < MINCHECK) {        // throttle=max, yaw=left, pitch=min
      if (rcDelayCommand == 20) calibratingA=400;
      rcDelayCommand++;
     } else if (rcData[YAW] > MAXCHECK && rcData[PITCH] < MINCHECK) { // throttle=max, yaw=right, pitch=min
      if (rcDelayCommand == 20) calibratingM=1; // MAG calibration request
      rcDelayCommand++;
     } else if (rcData[PITCH] > MAXCHECK) {
       accTrim[PITCH]+=2;writeParams();
       #if defined(LED_RING)
         blinkLedRing();
       #endif
     } else if (rcData[PITCH] < MINCHECK) {
       accTrim[PITCH]-=2;writeParams();
       #if defined(LED_RING)
         blinkLedRing();
       #endif
     } else if (rcData[ROLL] > MAXCHECK) {
       accTrim[ROLL]+=2;writeParams();
       #if defined(LED_RING)
         blinkLedRing();
       #endif
     } else if (rcData[ROLL] < MINCHECK) {
       accTrim[ROLL]-=2;writeParams();
       #if defined(LED_RING)
         blinkLedRing();
       #endif
     } else {
      rcDelayCommand = 0;
     }
    }
   #ifdef LOG_VALUES
    if (cycleTime > cycleTimeMax) cycleTimeMax = cycleTime; // remember highscore
    if (cycleTime < cycleTimeMin) cycleTimeMin = cycleTime; // remember lowscore
   #endif

    #if defined(InflightAccCalibration)
     if (AccInflightCalibrationArmed && armed == 1 && rcData[THROTTLE] > MINCHECK &&
!rcOptions[BOXARM] ){ // Copter is airborne and you are turning it off via boxarm : start measurement
      InflightcalibratingA = 50;
      AccInflightCalibrationArmed = 0;
     }
     if (rcOptions[BOXPASSTHRU]) {      // Use the Passthru Option to activate : Passthru = TRUE Meausrement started,
Land and passtrhu = 0 measurement stored
      if (!AccInflightCalibrationActive && !AccInflightCalibrationMeasurementDone){
        InflightcalibratingA = 50;
      }
     }else if(AccInflightCalibrationMeasurementDone && armed == 0){
      AccInflightCalibrationMeasurementDone = 0;
```

```
      AccInflightCalibrationSavetoEEProm = 1;
    }
  #endif

  for(i=0;i<CHECKBOXITEMS;i++) {
    rcOptions[i] = (
    ( (rcData[AUX1]<1300)    | (1300<rcData[AUX1] && rcData[AUX1]<1700)<<1 | (rcData[AUX1]>1700)<<2
     |(rcData[AUX2]<1300)<<3 | (1300<rcData[AUX2] && rcData[AUX2]<1700)<<4 | (rcData[AUX2]>1700)<<5) &
activate1[i]
    )||(
    ( (rcData[AUX3]<1300)    | (1300<rcData[AUX3] && rcData[AUX3]<1700)<<1 | (rcData[AUX3]>1700)<<2
     |(rcData[AUX4]<1300)<<3 | (1300<rcData[AUX4] && rcData[AUX4]<1700)<<4 | (rcData[AUX4]>1700)<<5) &
activate2[i]);
  }

  // note: if FAILSAFE is disable, failsafeCnt > 5*FAILSAVE_DELAY is always false
  if (( rcOptions[BOXACC] || (failsafeCnt > 5*FAILSAVE_DELAY) ) && (ACC || nunchuk)) {
    // bumpless transfer to Level mode
    if (!accMode) {
      errorAngleI[ROLL] = 0; errorAngleI[PITCH] = 0;
      accMode = 1;
    }
  } else accMode = 0;  // modified by MIS for failsave support

  if (rcOptions[BOXARM] == 0) okToArm = 1;
  if (accMode == 1) {STABLEPIN_ON;} else {STABLEPIN_OFF;}

  #if BARO
    if (rcOptions[BOXBARO]) {
      if (baroMode == 0) {
        baroMode = 1;
        AltHold = EstAlt;
        initialThrottleHold = rcCommand[THROTTLE]/57.2958/10; // [0 a 20]
        //initialThrottleHold = (rcCommand[THROTTLE] - 1000)/48; //Nuestro
        errorAltitudeI = 0;
        BaroPID=0;
      }
    } else baroMode = 0;
  #endif
  #if MAG
    if (rcOptions[BOXMAG]) {
      if (magMode == 0) {
        magMode = 1;
        magHold = heading;
      }
    } else magMode = 0;
    if (rcOptions[BOXHEADFREE]) {
      if (headFreeMode == 0) {
        headFreeMode = 1;
      }
    } else headFreeMode = 0;
  #endif
  #if GPS
    if (rcOptions[BOXGPSHOME]) {GPSModeHome = 1;}
    else GPSModeHome = 0;
    if (rcOptions[BOXGPSHOLD]) {
      if (GPSModeHold == 0) {
        GPSModeHold = 1;
        GPS_latitude_hold = GPS_latitude;
        GPS_longitude_hold = GPS_longitude;
      }
    } else {
      GPSModeHold = 0;
    }
  #endif
  if (rcOptions[BOXPASSTHRU]) {passThruMode = 1;}
  else passThruMode = 0;
```

```
} else { // not in rc loop
  static int8_t taskOrder=0; // never call all functions in the same loop, to avoid high delay spikes
  switch (taskOrder) {
    case 0:
      taskOrder++;
      #if MAG
        Mag_getADC();
        break;
      #endif
    case 1:
      taskOrder++;
      #if BARO
        Baro_update();
        break;
      #endif
    case 2:
      taskOrder++;
      #if BARO
        getEstimatedAltitude();
        break;
      #endif
    case 3:
      taskOrder++;
      #if GPS
        GPS_NewData();
        break;
      #endif
    default:
      taskOrder=0;
      break;
  }
}

computeIMU(); // Aquí se ejecuta annexCode Y se definen rcCommand[THROTTLE] entre [1000 2000]
// Measure loop rate just afer reading the sensors
currentTime = micros();
cycleTime = BaroPID;//currentTime - previousTime;
previousTime = currentTime;

#if MAG
  if (abs(rcCommand[YAW]) <70 && magMode) {
    int16_t dif = heading - magHold;
    //if (dif <= - 180) dif += 360;
    if (dif <= - 1800) dif += 3600; //Nuestro
    //if (dif >= + 180) dif -= 360;
    if (dif >= + 1800) dif -= 3600; //Nuesttro
    //if ( smallAngle25 ) rcCommand[YAW] -= dif*P8[PIDMAG]/30;  // 18 deg
    if ( smallAngle25 ) rcCommand[YAW] -= dif;// Nuestro
  } else magHold = heading;
#endif
//Nuestro
rcCommand[THROTTLE] = (rcCommand[THROTTLE] - 1000)/48; // [0 20]
#if BARO
  if (baroMode) {
    // Nuestro rcCommand[THROTTLE] entre [0 20], initialThrottleHold entre [0 a 20]
    if (abs(rcCommand[THROTTLE]-initialThrottleHold)>20) {
       baroMode = 0; // so that a new althold reference is defined
    }
    rcCommand[THROTTLE] = initialThrottleHold + BaroPID;
  }
#endif
#if GPS
  uint16_t GPS_dist;
  int16_t  GPS_dir;

  if ( (GPSModeHome == 0 && GPSModeHold == 0) || (GPS_fix_home == 0) ) {
    GPS_angle[ROLL]  = 0;
```

```
      GPS_angle[PITCH] = 0;
    } else {
      if (GPSModeHome == 1) {
        GPS_dist = GPS_distanceToHome;
        GPS_dir = GPS_directionToHome;
      }
      if (GPSModeHold == 1) {
        GPS_dist = GPS_distanceToHold;
        GPS_dir = GPS_directionToHold;
      }
      float radDiff = (GPS_dir-heading) * 0.0174533f;
      GPS_angle[ROLL]  = constrain(P8[PIDGPS] * sin(radDiff) * GPS_dist / 10,-D8[PIDGPS]*10,+D8[PIDGPS]*10); //
with P=5.0, a distance of 1 meter = 0.5deg inclination
      GPS_angle[PITCH] = constrain(P8[PIDGPS] * cos(radDiff) * GPS_dist / 10,-D8[PIDGPS]*10,+D8[PIDGPS]*10); //
max inclination = D deg
    }
  #endif

  //Nuestro
  rcCommand[THROTTLE] = rcCommand[THROTTLE]*57.2958*10; // [0 a 12000]
  //**** PITCH & ROLL & YAW PID ****
  for(axis=0;axis<3;axis++) {
    if (accMode == 1 && axis<2 ) { //LEVEL MODE
    errorAngle = constrain(rcCommand[axis] - GPS_angle[axis],-500,+500) - angle[axis] + accTrim[axis];
    PTerm = P8[axis]*errorAngle; // 500*50 = 25000 < 32000. OK 16 bits.
    errorAngleI[axis] = constrain((errorAngleI[axis] + (errorAngle + errorAngle_ant[axis])/2),-30000,30000);
    ITerm = errorAngleI[axis]*0.0035;
    ITerm = ITerm*I8[axis];
    DTerm = D8[axis]*gyroData[axis];  // 10*1500 < 32000
    D2Term = (D28[axis]*(gyroData[axis] - gyroData_ant_3[axis])/3)/3.5; // D28 = D2*1000 --> 1/(1000*Ts) =
1/(1000*0.003) = 1/(3)
    axisPID[axis] = PTerm + ITerm - DTerm - D2Term;
    errorAngle_ant[axis] = errorAngle;
    gyroData_ant_10[axis] = gyroData_ant_9[axis];
    gyroData_ant_9[axis] = gyroData_ant_8[axis];
    gyroData_ant_8[axis] = gyroData_ant_7[axis];
    gyroData_ant_7[axis] = gyroData_ant_6[axis];
    gyroData_ant_6[axis] = gyroData_ant_5[axis];
    gyroData_ant_5[axis] = gyroData_ant_4[axis];
    gyroData_ant_4[axis] = gyroData_ant_3[axis];
    gyroData_ant_3[axis] = gyroData_ant_2[axis];
    gyroData_ant_2[axis] = gyroData_ant[axis];
    gyroData_ant[axis] = gyroData[axis];
    } else { //ACRO MODE or YAW axis
    errorAngle = rcCommand[YAW];
    PTerm = P8[YAW]*errorAngle; // 1800*50 = 25000 < 32000. OK 16 bits.
    errorAngleI[YAW] = constrain((errorAngleI[YAW] + (errorAngle + errorAngle_ant[YAW])/2),-30000,30000);
    ITerm = errorAngleI[YAW]*0.0035*I8[YAW];
    DTerm = D8[YAW]*gyroData[YAW];  // 10*1500 < 32000
    axisPID[YAW] = PTerm + ITerm - DTerm;// - D2Term;
    errorAngle_ant[YAW] = errorAngle;
    }
  }

  mixTable();
  writeServos();
  writeMotors();
}
```

Anexo G

```
void computeIMU () {
  uint8_t axis;
  static int16_t gyroADCprevious[3] = {0,0,0};
  int16_t gyroADCp[3];
  int16_t gyroADCinter[3];
  static uint32_t timeInterleave = 0;

  //we separate the 2 situations because reading gyro values with a gyro only setup can be acchieved at a higher rate
  //gyro+nunchuk: we must wait for a quite high delay betwwen 2 reads to get both WM+ and Nunchuk data. It works with
3ms
  //gyro only: the delay to read 2 consecutive values can be reduced to only 0.65ms
  if (!ACC && nunchuk) {
    annexCode();
    while((micros()-timeInterleave)<INTERLEAVING_DELAY) ; //interleaving delay between 2 consecutive reads
    timeInterleave=micros();
    WMP_getRawADC();
    getEstimatedAttitude(); // computation time must last less than one interleaving delay
    while((micros()-timeInterleave)<INTERLEAVING_DELAY) ; //interleaving delay between 2 consecutive reads
    timeInterleave=micros();
    while(WMP_getRawADC() != 1) ; // For this interleaving reading, we must have a gyro update at this point (less delay)

    for (axis = 0; axis < 3; axis++) {
      // empirical, we take a weighted value of the current and the previous values
      // /4 is to average 4 values, note: overflow is not possible for WMP gyro here
      gyroData[axis] = (gyroADC[axis]*3+gyroADCprevious[axis]+2)/4;
      gyroADCprevious[axis] = gyroADC[axis];
    }
  } else {
    #if ACC
      ACC_getADC();
      getEstimatedAttitude();
    #endif
    #if GYRO
      Gyro_getADC();
    #else
      WMP_getRawADC();
    #endif
    for (axis = 0; axis < 3; axis++)
      gyroADCp[axis] =  gyroADC[axis];
    timeInterleave=micros();
    annexCode();
    if ((micros()-timeInterleave)>650) {
       annex650_overrun_count++;
    } else {
       while((micros()-timeInterleave)<650) ; //empirical, interleaving delay between 2 consecutive reads
    }
    #if GYRO
      Gyro_getADC();
    #else
      WMP_getRawADC();
    #endif
    for (axis = 0; axis < 3; axis++) {
      gyroADCinter[axis] =  gyroADC[axis]+gyroADCp[axis];
      // empirical, we take a weighted value of the current and the previous values
      gyroData[axis] = (gyroADCinter[axis]+gyroADCprevious[axis]+1)/3;
      gyroADCprevious[axis] = gyroADCinter[axis]/2;
      if (!ACC) accADC[axis]=0;
    }
  }
  #if defined(GYRO_SMOOTHING)
    static uint8_t Smoothing[3]  = GYRO_SMOOTHING; // How much to smoothen with per axis
    static int16_t gyroSmooth[3] = {0,0,0};
    for (axis = 0; axis < 3; axis++) {
```

```
      gyroData[axis] = (gyroSmooth[axis]*(Smoothing[axis]-1)+gyroData[axis]+1)/Smoothing[axis];
      gyroSmooth[axis] = gyroData[axis];
    }
  #elif defined(TRI)
    static int16_t gyroYawSmooth = 0;
    gyroData[YAW] = (gyroYawSmooth*2+gyroData[YAW]+1)/3;
    gyroYawSmooth = gyroData[YAW];
  #endif
}

// ****************************************************
// Simplified IMU based on "Complementary Filter"
// Inspired by http://starlino.com/imu_guide.html
//
// adapted by ziss_dm : http://www.multiwii.com/forum/viewtopic.php?f=8&t=198
//
// The following ideas was used in this project:
// 1) Rotation matrix: http://en.wikipedia.org/wiki/Rotation_matrix
// 2) Small-angle approximation: http://en.wikipedia.org/wiki/Small-angle_approximation
// 3) C. Hastings approximation for atan2()
// 4) Optimization tricks: http://www.hackersdelight.org/
//
// Currently Magnetometer uses separate CF which is used only
// for heading approximation.
//
// Modified: 19/04/2011  by ziss_dm
// Version: V1.1
//
// code size reduction and tmp vector intermediate step for vector rotation computation: October 2011 by Alex
// ****************************************************

//******  advanced users settings *******************
/* Set the Low Pass Filter factor for ACC */
/* Increasing this value would reduce ACC noise (visible in GUI), but would increase ACC lag time*/
/* Comment this if  you do not want filter at all.*/
/* Default WMC value: 8*/
#define ACC_LPF_FACTOR 4

/* Set the Low Pass Filter factor for Magnetometer */
/* Increasing this value would reduce Magnetometer noise (not visible in GUI), but would increase Magnetometer lag
time*/
/* Comment this if  you do not want filter at all.*/
/* Default WMC value: n/a*/
//#define MG_LPF_FACTOR 4

/* Set the Gyro Weight for Gyro/Acc complementary filter */
/* Increasing this value would reduce and delay Acc influence on the output of the filter*/
/* Default WMC value: 300*/
#define GYR_CMPF_FACTOR 310.0f

/* Set the Gyro Weight for Gyro/Magnetometer complementary filter */
/* Increasing this value would reduce and delay Magnetometer influence on the output of the filter*/
/* Default WMC value: n/a*/
#define GYR_CMPFM_FACTOR 200.0f

//****** end of advanced users settings **************

#define INV_GYR_CMPF_FACTOR   (1.0f / (GYR_CMPF_FACTOR  + 1.0f))
#define INV_GYR_CMPFM_FACTOR  (1.0f / (GYR_CMPFM_FACTOR + 1.0f))
#if GYRO
  #define GYRO_SCALE ((2380 * PI)/((32767.0f / 4.0f ) * 180.0f * 1000000.0f)) //should be 2279.44 but 2380 gives better
result
  // +-2000/sec deg scale
  //#define GYRO_SCALE ((200.0f * PI)/((32768.0f / 5.0f / 4.0f ) * 180.0f * 1000000.0f) * 1.5f)
  // +- 200/sec deg scale
  // 1.5 is emperical, not sure what it means
  // should be in rad/sec
```

```
#else
  #define GYRO_SCALE (1.0f/200e6f)
  // empirical, depends on WMP on IDG datasheet, tied of deg/ms sensibility
  // !!!!should be adjusted to the rad/sec
#endif
// Small angle approximation
#define ssin(val) (val)
#define scos(val) 1.0f

typedef struct fp_vector {
  float X;
  float Y;
  float Z;
} t_fp_vector_def;

typedef union {
  float   A[3];
  t_fp_vector_def V;
} t_fp_vector;

int16_t _atan2(float y, float x){
  #define fp_is_neg(val) ((((uint8_t*)&val)[3] & 0x80) != 0)
  float z = y / x;
  int16_t zi = abs(int16_t(z * 100));
  int8_t y_neg = fp_is_neg(y);
  if ( zi < 100 ){
    if (zi > 10)
     z = z / (1.0f + 0.28f * z * z);
   if (fp_is_neg(x)) {
     if (y_neg) z -= PI;
     else z += PI;
   }
  } else {
   z = (PI / 2.0f) - z / (z * z + 0.28f);
   if (y_neg) z -= PI;
  }
  z *= (180.0f / PI * 10);
  return z;
}

// Rotate Estimated vector(s) with small angle approximation, according to the gyro data
void rotateV(struct fp_vector *v,float* delta) {
  fp_vector v_tmp = *v;
  v->Z -= delta[ROLL]  * v_tmp.X + delta[PITCH] * v_tmp.Y;
  v->X += delta[ROLL]  * v_tmp.Z - delta[YAW]   * v_tmp.Y;
  v->Y += delta[PITCH] * v_tmp.Z + delta[YAW]   * v_tmp.X;
}

void getEstimatedAttitude(){
  uint8_t axis;
  int32_t accMag = 0;
  static t_fp_vector EstG;
#if MAG
  static t_fp_vector EstM;
#endif
#if defined(MG_LPF_FACTOR)
  static int16_t mgSmooth[3];
#endif
#if defined(ACC_LPF_FACTOR)
  static int16_t accTemp[3];   //projection of smoothed and normalized magnetic vector on x/y/z axis, as measured by
magnetometer
#endif
  static uint16_t previousT;
  uint16_t currentT = micros();
  float scale, deltaGyroAngle[3];

  scale = (currentT - previousT) * GYRO_SCALE;
```

```
  previousT = currentT;

  // Initialization
  for (axis = 0; axis < 3; axis++) {
    deltaGyroAngle[axis] = gyroADC[axis]  * scale;
    #if defined(ACC_LPF_FACTOR)
      accTemp[axis] = (accTemp[axis] - (accTemp[axis] >>ACC_LPF_FACTOR)) + accADC[axis];
      accSmooth[axis] = accTemp[axis]>>ACC_LPF_FACTOR;
      #define ACC_VALUE accSmooth[axis]
    #else
      accSmooth[axis] = accADC[axis];
      #define ACC_VALUE accADC[axis]
    #endif
//    accMag += (ACC_VALUE * 10 / (int16_t)acc_1G) * (ACC_VALUE * 10 / (int16_t)acc_1G);
    accMag += (int32_t)ACC_VALUE*ACC_VALUE ;
    #if MAG
      #if defined(MG_LPF_FACTOR)
        mgSmooth[axis] = (mgSmooth[axis] * (MG_LPF_FACTOR - 1) + magADC[axis]) / MG_LPF_FACTOR; // LPF for
Magnetometer values
        #define MAG_VALUE mgSmooth[axis]
      #else
        #define MAG_VALUE magADC[axis]
      #endif
    #endif
  }
  accMag = accMag*100/((int32_t)acc_1G*acc_1G);

  rotateV(&EstG.V,deltaGyroAngle);
  #if MAG
    rotateV(&EstM.V,deltaGyroAngle);
  #endif

  if ( abs(accSmooth[ROLL])<acc_25deg && abs(accSmooth[PITCH])<acc_25deg && accSmooth[YAW]>0)
    smallAngle25 = 1;
  else
    smallAngle25 = 0;

  // Apply complimentary filter (Gyro drift correction)
  // If accel magnitude >1.4G or <0.6G and ACC vector outside of the limit range => we neutralize the effect of
accelerometers in the angle estimation.
  // To do that, we just skip filter, as EstV already rotated by Gyro
  if ( ( 36 < accMag && accMag < 196 ) || smallAngle25 )
    for (axis = 0; axis < 3; axis++) {
      int16_t acc = ACC_VALUE;
      #if !defined(TRUSTED_ACCZ)
        if (smallAngle25 && axis == YAW)
          //We consider ACCZ = acc_1G when the acc on other axis is small.
          //It's a tweak to deal with some configs where ACC_Z tends to a value < acc_1G when high throttle is applied.
          //This tweak applies only when the multi is not in inverted position
          acc = acc_1G;
      #endif
      EstG.A[axis] = (EstG.A[axis] * GYR_CMPF_FACTOR + acc) * INV_GYR_CMPF_FACTOR;
    }
  #if MAG
    for (axis = 0; axis < 3; axis++)
      EstM.A[axis] = (EstM.A[axis] * GYR_CMPFM_FACTOR  + MAG_VALUE) * INV_GYR_CMPFM_FACTOR;
  #endif

  // Attitude of the estimated vector
  angle[ROLL]  =  _atan2(EstG.V.X , EstG.V.Z) ;
  angle[PITCH] =  _atan2(EstG.V.Y , EstG.V.Z) ;
  #if MAG
    // Attitude of the cross product vector GxM
    //heading = _atan2( EstG.V.X * EstM.V.Z - EstG.V.Z * EstM.V.X , EstG.V.Z * EstM.V.Y - EstG.V.Y * EstM.V.Z  ) /
10;
    heading = _atan2( EstG.V.X * EstM.V.Z - EstG.V.Z * EstM.V.X , EstG.V.Z * EstM.V.Y - EstG.V.Y * EstM.V.Z  );
//Nuestro
```

```
  #endif
}

#define UPDATE_INTERVAL 25000    // 40hz update rate (20hz LPF on acc)
#define INIT_DELAY      4000000  // 4 sec initialization delay
#define BARO_TAB_SIZE   40

void getEstimatedAltitude(){
  uint8_t index;
  static uint32_t deadLine = INIT_DELAY;

  static int16_t BaroHistTab[BARO_TAB_SIZE];
  static int8_t BaroHistIdx;
  static int32_t BaroHigh,BaroLow;
  int32_t temp32;
  int16_t last;

  if (currentTime < deadLine) return;
  deadLine = currentTime + UPDATE_INTERVAL;

  //**** Alt. Set Point stabilization PID ****
  //calculate speed for D calculation
  last = BaroHistTab[BaroHistIdx];
  BaroHistTab[BaroHistIdx] = BaroAlt/10;
  BaroHigh += BaroHistTab[BaroHistIdx];
  index = (BaroHistIdx + (BARO_TAB_SIZE/2))%BARO_TAB_SIZE;
  BaroHigh -= BaroHistTab[index];
  BaroLow  += BaroHistTab[index];
  BaroLow  -= last;

  BaroHistIdx++;
  if (BaroHistIdx == BARO_TAB_SIZE) BaroHistIdx = 0;

  /*
  //D
  temp32 = D8[PIDALT]*(BaroHigh - BaroLow) / 40;
  BaroPID-=temp32;

  EstAlt = BaroHigh*10/(BARO_TAB_SIZE/2);

  temp32 = AltHold - EstAlt;
  if (abs(temp32) < 10 && abs(BaroPID) < 10) BaroPID = 0;  //remove small D parametr to reduce noise near zero position

  //P
  BaroPID += P8[PIDALT]*constrain(temp32,(-2)*P8[PIDALT],2*P8[PIDALT])/100;
  BaroPID = constrain(BaroPID,-150,+150); //sum of P and D should be in range 150

  //I
  errorAltitudeI += temp32*I8[PIDALT]/50;
  errorAltitudeI = constrain(errorAltitudeI,-30000,30000);
  temp32 = errorAltitudeI / 500; //I in range +/-60
  BaroPID+=temp32;*/
  BaroPID = 0;
  EstAlt = BaroHigh*10/(BARO_TAB_SIZE/2); // EstAlt ---> 1 metro = 100 unidades; 1 cm = 1 unidad;

  errorAlt = constrain(AltHold - EstAlt,-300,+300); // 30 ---> 3 metros de error
  PTermAlt = P8[PIDALT]*errorAlt; // 500*50 = 25000 < 32000. OK 16 bits.
  errorAltitudeI = constrain((errorAltitudeI + (errorAlt + errorAlt_ant[1])/2),-30000,30000);
  ITermAlt = errorAltitudeI*0.003;
  ITermAlt = ITermAlt*I8[PIDALT];
  errorAlt_ant[1] = errorAlt;

  // Nuestro
  if (contador == a_z){
  //BaroPID = 0;
  // DTerm
  DTermAlt = (EstAlt - Alt_ant_3[1])*D8[PIDALT];
```

```
  DTermAlt = DTermAlt/(0.003*a_z);
  DTermAlt = DTermAlt/3;
  Alt_ant_4[1] = Alt_ant_3[1];
  Alt_ant_3[1] = Alt_ant_2[1];
  Alt_ant_2[1] = Alt_ant[1];
  Alt_ant[1] = EstAlt;
  contador=1;
  }  else {contador=contador+1;
  }
  BaroPID = (PTermAlt + ITermAlt - DTermAlt)/100; // entre 10 por unidades
}
```

Anexo H

```
#if defined(PROMINI)
  uint8_t PWM_PIN[8] = {9,10,11,3,6,5,A2,12};   //for a quad+: rear,right,left,front
#endif
#if defined(PROMICRO)
  #if !defined(HWPWM6)
    uint8_t PWM_PIN[8] = {9,10,5,6,4,A2,A0,A1};   //for a quad+: rear,right,left,front
  #else
    uint8_t PWM_PIN[8] = {9,10,5,6,11,13,A0,A1};   //for a quad+: rear,right,left,front
  #endif
#endif
#if defined(MEGA)
  uint8_t PWM_PIN[8] = {3,5,6,2,7,8,9,10};      //for a quad+: rear,right,left,front   //+ for y6: 7:under right  8:under left
#endif
#if !defined(PROMICRO) || defined(HWPWM6)
  volatile uint8_t atomicServo[8] = {125,125,125,125,125,125,125,125};
  //for HEX Y6 and HEX6/HEX6X flat for promini
  volatile uint8_t atomicPWM_PIN5_lowState;
  volatile uint8_t atomicPWM_PIN5_highState;
  volatile uint8_t atomicPWM_PIN6_lowState;
  volatile uint8_t atomicPWM_PIN6_highState;
  //for OCTO on promini
  volatile uint8_t atomicPWM_PINA2_lowState;
  volatile uint8_t atomicPWM_PINA2_highState;
  volatile uint8_t atomicPWM_PIN12_lowState;
  volatile uint8_t atomicPWM_PIN12_highState;
#else
  volatile uint16_t atomicServo[8] = {8000,8000,8000,8000,8000,8000,8000,8000};
  //for HEX Y6 and HEX6/HEX6X and for Promicro
  volatile uint16_t atomicPWM_PIN5_lowState;
  volatile uint16_t atomicPWM_PIN5_highState;
  volatile uint16_t atomicPWM_PIN6_lowState;
  volatile uint16_t atomicPWM_PIN6_highState;
  //for OCTO on Promicro
  volatile uint16_t atomicPWM_PINA2_lowState;
  volatile uint16_t atomicPWM_PINA2_highState;
  volatile uint16_t atomicPWM_PIN12_lowState;
  volatile uint16_t atomicPWM_PIN12_highState;
#endif

void writeServos() {
  #if defined(SERVO)
    #if defined(PRI_SERVO_FROM)    // write primary servos
      for(uint8_t i = (PRI_SERVO_FROM-1); i < PRI_SERVO_TO; i++){
        #if !defined(PROMICRO) || defined(HWPWM6)
          atomicServo[i] = (servo[i]-1000)>>2;
        #else
          atomicServo[i] = (servo[i]-1000)<<4;
        #endif
      }
    #endif
    #if defined(SEC_SERVO_FROM)   // write secundary servos
      #if defined(SERVO_TILT) && defined(MMSERVOGIMBAL)
        // Moving Average Servo Gimbal by Magnetron1
        static int16_t mediaMobileServoGimbalADC[3][MMSERVOGIMBALVECTORLENGHT];
        static int32_t mediaMobileServoGimbalADCSum[3];
        static uint8_t mediaMobileServoGimbalIDX;
        uint8_t axis;

        mediaMobileServoGimbalIDX = ++mediaMobileServoGimbalIDX % MMSERVOGIMBALVECTORLENGHT;
        for (axis=(SEC_SERVO_FROM-1); axis < SEC_SERVO_TO; axis++) {
          mediaMobileServoGimbalADCSum[axis]                                                    -=
mediaMobileServoGimbalADC[axis][mediaMobileServoGimbalIDX];
          mediaMobileServoGimbalADC[axis][mediaMobileServoGimbalIDX] = servo[axis];
```

```
        mediaMobileServoGimbalADCSum[axis]                                                    +=
mediaMobileServoGimbalADC[axis][mediaMobileServoGimbalIDX];
        #if !defined(PROMICRO) || defined(HWPWM6)
          atomicServo[axis]  =  (mediaMobileServoGimbalADCSum[axis]  /  MMSERVOGIMBALVECTORLENGHT  -
1000)>>2;
        #else
          atomicServo[axis]  =  (mediaMobileServoGimbalADCSum[axis]  /  MMSERVOGIMBALVECTORLENGHT  -
1000)<<4;
        #endif
       }
     #else
       for(uint8_t i = (SEC_SERVO_FROM-1); i < SEC_SERVO_TO; i++){
        #if !defined(PROMICRO) || defined(HWPWM6)
          atomicServo[i] = (servo[i]-1000)>>2;
        #else
          atomicServo[i] = (servo[i]-1000)<<4;
        #endif
       }
     #endif
   #endif
  #endif
}

void writeMotors() { // [1000;2000] => [125;250]
  #if defined(MEGA)// [1000:2000] => [8000:16000] for timer 3 & 4 for mega
   #if (NUMBER_MOTOR > 0)
     #ifndef EXT_MOTOR_RANGE
       OCR3C = motor[0]<<3; //  pin 3
     #else
       OCR3C = ((motor[0]<<4) - 16000) + 128;
     #endif
   #endif
   #if (NUMBER_MOTOR > 1)
     #ifndef EXT_MOTOR_RANGE
       OCR3A = motor[1]<<3; //  pin 5
     #else
       OCR3A = ((motor[1]<<4) - 16000) + 128;
     #endif
   #endif
   #if (NUMBER_MOTOR > 2)
     #ifndef EXT_MOTOR_RANGE
       OCR4A = motor[2]<<3; //  pin 6
     #else
       OCR4A = ((motor[2]<<4) - 16000) + 128;
     #endif
   #endif
   #if (NUMBER_MOTOR > 3)
     #ifndef EXT_MOTOR_RANGE
       OCR3B = motor[3]<<3; //  pin 2
     #else
       OCR3B = ((motor[3]<<4) - 16000) + 128;
     #endif
   #endif
   #if (NUMBER_MOTOR > 4)
     #ifndef EXT_MOTOR_RANGE
       OCR4B = motor[4]<<3; //  pin 7
       OCR4C = motor[5]<<3; //  pin 8
     #else
       OCR4B = ((motor[4]<<4) - 16000) + 128;
       OCR4C = ((motor[5]<<4) - 16000) + 128;
     #endif
   #endif
   #if (NUMBER_MOTOR > 6)
     #ifndef EXT_MOTOR_RANGE
       OCR2B = motor[6]>>3; //  pin 9
       OCR2A = motor[7]>>3; //  pin 10
     #else
```

```
      OCR2B = ((motor[6]>>2) - 250) + 2);
      OCR2A = ((motor[7]>>2) - 250) + 2);
    #endif
  #endif
#endif
#if defined(PROMICRO)
  #if (NUMBER_MOTOR > 0) // Timer 1 A & B [1000:2000] => [8000:16000]
    OCR1A = motor[0]<<3; //  pin 9
  #endif
  #if (NUMBER_MOTOR > 1)
    OCR1B = motor[1]<<3; //  pin 10
  #endif
  #if (NUMBER_MOTOR > 2) // Timer 4 A & D [1000:2000] => [1000:2000]
    #if !defined(HWPWM6)
      // to write values > 255 to timer 4 A/B we need to split the bytes
      static uint8_t pwm4_HBA;
      static uint16_t pwm4_LBA; // high and low byte for timer 4 A
      pwm4_LBA = 2047-motor[2]; pwm4_HBA = 0; // channel A is inverted
      while(pwm4_LBA > 255){
        pwm4_HBA++;
        pwm4_LBA-=256;
      }
      TC4H = pwm4_HBA; OCR4A = pwm4_LBA; //  pin 5
    #else
      OCR3A = motor[2]<<3; //  pin 5
    #endif
  #endif
  #if (NUMBER_MOTOR > 3)
    static uint8_t pwm4_HBD;
    static uint16_t pwm4_LBD; // high and low byte for timer 4 D
    pwm4_LBD = motor[3]; pwm4_HBD = 0;
    while(pwm4_LBD > 255){
      pwm4_HBD++;
      pwm4_LBD-=256;
    }
    TC4H = pwm4_HBD; OCR4D = pwm4_LBD; //  pin 6
  #endif
  #if (NUMBER_MOTOR > 4)
    #if !defined(HWPWM6)
      atomicPWM_PIN5_highState = ((motor[4]-1000)<<4)+320;
      atomicPWM_PIN5_lowState = 15743-atomicPWM_PIN5_highState;
      atomicPWM_PIN6_highState = ((motor[5]-1000)<<4)+320;
      atomicPWM_PIN6_lowState = 15743-atomicPWM_PIN6_highState;
    #else
      OCR1C = motor[4]<<3; //  pin 11
      static uint8_t pwm4_HBA;
      static uint16_t pwm4_LBA; // high and low byte for timer 4 A
      pwm4_LBA = motor[5]; pwm4_HBA = 0;
      while(pwm4_LBA > 255){
        pwm4_HBA++;
        pwm4_LBA-=256;
      }
      TC4H = pwm4_HBA; OCR4A = pwm4_LBA; //  pin 13
    #endif
  #endif
  #if (NUMBER_MOTOR > 6)
    #if !defined(HWPWM6)
      atomicPWM_PINA2_highState = ((motor[6]-1000)<<4)+320;
      atomicPWM_PINA2_lowState = 15743-atomicPWM_PINA2_highState;
      atomicPWM_PIN12_highState = ((motor[7]-1000)<<4)+320;
      atomicPWM_PIN12_lowState = 15743-atomicPWM_PIN12_highState;
    #else
      atomicPWM_PINA2_highState = ((motor[6]-1000)>>2)+5;
      atomicPWM_PINA2_lowState = 245-atomicPWM_PINA2_highState;
      atomicPWM_PIN12_highState = ((motor[7]-1000)>>2)+5;
      atomicPWM_PIN12_lowState = 245-atomicPWM_PIN12_highState;
    #endif
```

```
    #endif
  #endif
  #if defined(PROMINI)
    #if (NUMBER_MOTOR > 0)
      #ifndef EXT_MOTOR_RANGE
        OCR1A = motor[0]>>3; //  pin 9
      #else
        OCR1A = ((motor[0]>>2) - 250) + 2;
      #endif
    #endif
    #if (NUMBER_MOTOR > 1)
      #ifndef EXT_MOTOR_RANGE
        OCR1B = motor[1]>>3; //  pin 10
      #else
        OCR1B = ((motor[1]>>2) - 250) + 2;
      #endif
    #endif
    #if (NUMBER_MOTOR > 2)
      #ifndef EXT_MOTOR_RANGE
        OCR2A = motor[2]>>3; //  pin 11
      #else
        OCR2A = ((motor[2]>>2) - 250) + 2;
      #endif
    #endif
    #if (NUMBER_MOTOR > 3)
      #ifndef EXT_MOTOR_RANGE
        OCR2B = motor[3]>>3; //  pin 3
      #else
        OCR2B = ((motor[3]>>2) - 250) + 2;
      #endif
    #endif
    #if (NUMBER_MOTOR > 4) //note: EXT_MOTOR_RANGE not possible here
      atomicPWM_PIN6_highState = ((motor[4]-1000)>>2)+5;
      atomicPWM_PIN6_lowState  = 245-atomicPWM_PIN6_highState;
      atomicPWM_PIN5_highState = ((motor[5]-1000)>>2)+5;
      atomicPWM_PIN5_lowState  = 245-atomicPWM_PIN5_highState;
    #endif
    #if (NUMBER_MOTOR > 6) //note: EXT_MOTOR_RANGE not possible here
      atomicPWM_PINA2_highState = ((motor[6]-1000)>>2)+5;
      atomicPWM_PINA2_lowState  = 245-atomicPWM_PINA2_highState;
      atomicPWM_PIN12_highState = ((motor[7]-1000)>>2)+5;
      atomicPWM_PIN12_lowState  = 245-atomicPWM_PIN12_highState;
    #endif
  #endif
}

void writeAllMotors(int16_t mc) {   // Sends commands to all motors
  for (uint8_t i =0;i<NUMBER_MOTOR;i++)
    motor[i]=mc;
  writeMotors();
}

void initOutput() {
  for(uint8_t i=0;i<NUMBER_MOTOR;i++)
    pinMode(PWM_PIN[i],OUTPUT);
  #if defined(MEGA)
    #if (NUMBER_MOTOR > 0)
      // init 16bit timer 3
      TCCR3A |= (1<<WGM31); // phase correct mode
      TCCR3A &= ~(1<<WGM30);
      TCCR3B |= (1<<WGM33);
      TCCR3B &= ~(1<<CS31); // no prescaler
      ICR3   |= 0x3FFF; // TOP to = 16383;

      TCCR3A |= _BV(COM3C1); // connect pin 3 to timer 3 channel C
    #endif
    #if (NUMBER_MOTOR > 1)
```

```
    TCCR3A |= _BV(COM3A1); // connect pin 5 to timer 3 channel A
  #endif
  #if (NUMBER_MOTOR > 2)
    // init 16bit timer 4
    TCCR4A |= (1<<WGM41); // phase correct mode
    TCCR4A &= ~(1<<WGM40);
    TCCR4B |= (1<<WGM43);
    TCCR4B &= ~(1<<CS41); // no prescaler
    ICR4   |= 0x3FFF; // TOP to = 16383;

    TCCR4A |= _BV(COM4A1); // connect pin 6 to timer 4 channel A
  #endif
  #if (NUMBER_MOTOR > 3)
    TCCR3A |= _BV(COM3B1); // connect pin 2 to timer 3 channel B
  #endif
  #if (NUMBER_MOTOR > 4)
    TCCR4A |= _BV(COM4B1); // connect pin 7 to timer 4 channel B
    TCCR4A |= _BV(COM4C1); // connect pin 8 to timer 4 channel C
  #endif
  #if (NUMBER_MOTOR > 6)
    // timer 2 is a 8bit timer so we cant change its range
    TCCR2A |= _BV(COM2B1); // connect pin 9 to timer 2 channel B
    TCCR2A |= _BV(COM2A1); // connect pin 10 to timer 2 channel A
  #endif
 #endif
 #if defined(PROMICRO)
  #if (NUMBER_MOTOR > 0)
    TCCR1A |= (1<<WGM11); TCCR1A &= ~(1<<WGM10); TCCR1B |= (1<<WGM13);  // phase correct mode
    TCCR1B &= ~(1<<CS11); ICR1 |= 0x3FFF; // no prescaler & TOP to 16383;
    TCCR1A |= _BV(COM1A1); // connect pin 9 to timer 1 channel A
  #endif
  #if (NUMBER_MOTOR > 1)
    TCCR1A |= _BV(COM1B1); // connect pin 10 to timer 1 channel B
  #endif
  #if (NUMBER_MOTOR > 2)
    #if !defined(HWPWM6) // timer 4A
      TCCR4E |= (1<<ENHC4); // enhanced pwm mode
      TCCR4B &= ~(1<<CS41); TCCR4B |= (1<<CS42)|(1<<CS40); // prescaler to 16
      TCCR4D |= (1<<WGM40); TC4H = 0x3; OCR4C = 0xFF; // phase and frequency correct mode & top to 1023 but
with enhanced pwm mode we have 2047
      TCCR4A |= (1<<COM4A0)|(1<<PWM4A); // connect pin 5 to timer 4 channel A
    #else // timer 3A
      TCCR3A |= (1<<WGM31); TCCR3A &= ~(1<<WGM30); TCCR3B |= (1<<WGM33);  // phase correct mode
      TCCR3B &= ~(1<<CS31); ICR3 |= 0x3FFF; // no prescaler & TOP to 16383;
      TCCR3A |= _BV(COM3A1); // connect pin 5 to timer 3 channel A
    #endif
  #endif
  #if (NUMBER_MOTOR > 3)
    #if defined(HWPWM6)
      TCCR4E |= (1<<ENHC4); // enhanced pwm mode
      TCCR4B &= ~(1<<CS41); TCCR4B |= (1<<CS42)|(1<<CS40); // prescaler to 16
      TCCR4D |= (1<<WGM40); TC4H = 0x3; OCR4C = 0xFF; // phase and frequency correct mode & top to 1023 but
with enhanced pwm mode we have 2047
    #endif
    TCCR4C |= (1<<COM4D1)|(1<<PWM4D); // connect pin 6 to timer 4 channel D
  #endif
  #if (NUMBER_MOTOR > 4)
    #if defined(HWPWM6)
      TCCR1A |= _BV(COM1C1); // connect pin 11 to timer 1 channel C
      TCCR4A |= (1<<COM4A1)|(1<<PWM4A); // connect pin 13 to timer 4 channel A
    #else
      initializeSoftPWM();
    #endif
  #endif
  #if (NUMBER_MOTOR > 6)
    #if defined(HWPWM6)
      initializeSoftPWM();
```

```
      #endif
    #endif
  #endif
  #if defined(PROMINI)
    #if (NUMBER_MOTOR > 0)
      TCCR1A |= _BV(COM1A1); // connect pin 9 to timer 1 channel A
    #endif
    #if (NUMBER_MOTOR > 1)
      TCCR1A |= _BV(COM1B1); // connect pin 10 to timer 1 channel B
    #endif
    #if (NUMBER_MOTOR > 2)
      TCCR2A |= _BV(COM2A1); // connect pin 11 to timer 2 channel A
    #endif
    #if (NUMBER_MOTOR > 3)
      TCCR2A |= _BV(COM2B1); // connect pin 3 to timer 2 channel B
    #endif
    #if (NUMBER_MOTOR > 5)  // PIN 5 & 6 or A0 & A1
      initializeSoftPWM();
      #if defined(A0_A1_PIN_HEX) || (NUMBER_MOTOR > 6)
        pinMode(5,INPUT);pinMode(6,INPUT);     // we reactivate the INPUT affectation for these two PINs
        pinMode(A0,OUTPUT);pinMode(A1,OUTPUT);
      #endif
    #endif
  #endif

  writeAllMotors(MINCOMMAND);
  delay(300);
  #if defined(SERVO)
    initializeServo();
  #endif
}

#if defined(SERVO)
void initializeServo() {
  #if (PRI_SERVO_FROM == 1) || (SEC_SERVO_FROM == 1)
    SERVO_1_PINMODE;
  #endif
  #if (PRI_SERVO_FROM <= 2 && PRI_SERVO_TO >= 2) || (SEC_SERVO_FROM <= 2 && SEC_SERVO_TO >= 2)
    SERVO_2_PINMODE;
  #endif
  #if (PRI_SERVO_FROM <= 3 && PRI_SERVO_TO >= 3) || (SEC_SERVO_FROM <= 3 && SEC_SERVO_TO >= 3)
    SERVO_3_PINMODE;
  #endif
  #if (PRI_SERVO_FROM <= 4 && PRI_SERVO_TO >= 4) || (SEC_SERVO_FROM <= 4 && SEC_SERVO_TO >= 4)
    SERVO_4_PINMODE;
  #endif
  #if (PRI_SERVO_FROM <= 5 && PRI_SERVO_TO >= 5) || (SEC_SERVO_FROM <= 5 && SEC_SERVO_TO >= 5)
    SERVO_5_PINMODE;
  #endif
  #if (PRI_SERVO_FROM <= 6 && PRI_SERVO_TO >= 6) || (SEC_SERVO_FROM <= 6 && SEC_SERVO_TO >= 6)
    SERVO_6_PINMODE;
  #endif
  #if (PRI_SERVO_FROM <= 7 && PRI_SERVO_TO >= 7) || (SEC_SERVO_FROM <= 7 && SEC_SERVO_TO >= 7)
    SERVO_7_PINMODE;
  #endif
  #if (PRI_SERVO_FROM <= 8 && PRI_SERVO_TO >= 8) || (SEC_SERVO_FROM <= 8 && SEC_SERVO_TO >= 8)
    SERVO_8_PINMODE;
  #endif

  #if !defined(PROMICRO) || defined(HWPWM6)
    TCCR0A = 0; // normal counting mode
    TIMSK0 |= (1<<OCIE0A); // Enable CTC interrupt
  #else
    TCCR3A &= ~(1<<WGM30); // normal counting mode
    TCCR3B &= ~(1<<CS31); // no prescaler
    TIMSK3 |= (1<<OCIE3A); // Enable CTC interrupt
  #endif
```

```
}

#if !defined(PROMICRO) || defined(HWPWM6)
  #define SERVO_ISR TIMER0_COMPA_vect
  #define SERVO_CHANNEL OCR0A
  #define SERVO_1K_US 250
#else
  #define SERVO_ISR TIMER3_COMPA_vect
  #define SERVO_CHANNEL OCR3A
  #define SERVO_1K_US 16000
#endif

// prescaler is set by default to 64 on Timer0
// Duemilanove : 16MHz / 64 => 4 us
// 256 steps = 1 counter cycle = 1024 us
ISR(SERVO_ISR) {
  static uint8_t state = 0;
  static uint8_t count;
  if (state == 0) {
    #if (PRI_SERVO_FROM == 1) || (SEC_SERVO_FROM == 1)
      SERVO_1_PIN_HIGH;
    #endif
    SERVO_CHANNEL+= SERVO_1K_US; // 1000 us
    state++ ;
  } else if (state == 1) {
    SERVO_CHANNEL+= atomicServo[0]; // 1000 + [0-1020] us
    state++;
  } else if (state == 2) {
    #if (PRI_SERVO_FROM == 1) || (SEC_SERVO_FROM == 1)
      SERVO_1_PIN_LOW;
    #endif
    #if (PRI_SERVO_FROM <= 2 && PRI_SERVO_TO >= 2) || (SEC_SERVO_FROM <= 2 && SEC_SERVO_TO >=
2)
      SERVO_2_PIN_HIGH;
    #endif
    SERVO_CHANNEL+= SERVO_1K_US; // 1000 us
    state++;
  } else if (state == 3) {
    SERVO_CHANNEL+= atomicServo[1]; // 1000 + [0-1020] us
    state++;
  } else if (state == 4) {
    #if (PRI_SERVO_FROM <= 2 && PRI_SERVO_TO >= 2) || (SEC_SERVO_FROM <= 2 && SEC_SERVO_TO >=
2)
      SERVO_2_PIN_LOW;
    #endif
    #if (PRI_SERVO_FROM <= 3 && PRI_SERVO_TO >= 3) || (SEC_SERVO_FROM <= 3 && SEC_SERVO_TO >=
3)
      SERVO_3_PIN_HIGH;
    #endif
    state++;
    SERVO_CHANNEL+= SERVO_1K_US; // 1000 us
  } else if (state == 5) {
    SERVO_CHANNEL+= atomicServo[2]; // 1000 + [0-1020] us
    state++;
  } else if (state == 6) {
    #if (PRI_SERVO_FROM <= 3 && PRI_SERVO_TO >= 3) || (SEC_SERVO_FROM <= 3 && SEC_SERVO_TO >=
3)
      SERVO_3_PIN_LOW;
    #endif
    #if (PRI_SERVO_FROM <= 4 && PRI_SERVO_TO >= 4) || (SEC_SERVO_FROM <= 4 && SEC_SERVO_TO >=
4)
      SERVO_4_PIN_HIGH;
    #endif
    state++;
    SERVO_CHANNEL+= SERVO_1K_US; // 1000 us
  } else if (state == 7) {
    SERVO_CHANNEL+= atomicServo[3]; // 1000 + [0-1020] us
```

```
    state++;
  } else if (state == 8) {
    #if (PRI_SERVO_FROM <= 4 && PRI_SERVO_TO >= 4) || (SEC_SERVO_FROM <= 4 && SEC_SERVO_TO >=
4)
      SERVO_4_PIN_LOW;
    #endif
    #if (PRI_SERVO_FROM <= 5 && PRI_SERVO_TO >= 5) || (SEC_SERVO_FROM <= 5 && SEC_SERVO_TO >=
5)
      SERVO_5_PIN_HIGH;;
    #endif
    state++;
    SERVO_CHANNEL+= SERVO_1K_US; // 1000 us
  } else if (state == 9) {
    SERVO_CHANNEL+= atomicServo[4]; // 1000 + [0-1020] us
    state++;
  } else if (state == 10) {
    #if (PRI_SERVO_FROM <= 5 && PRI_SERVO_TO >= 5) || (SEC_SERVO_FROM <= 5 && SEC_SERVO_TO >=
5)
      SERVO_5_PIN_LOW;
    #endif
    #if (PRI_SERVO_FROM <= 6 && PRI_SERVO_TO >= 6) || (SEC_SERVO_FROM <= 6 && SEC_SERVO_TO >=
6)
      SERVO_6_PIN_HIGH;;
    #endif
    state++;
    SERVO_CHANNEL+= SERVO_1K_US; // 1000 us
  } else if (state == 11) {
    SERVO_CHANNEL+= atomicServo[5]; // 1000 + [0-1020] us
    state++;
  } else if (state == 12) {
    #if (PRI_SERVO_FROM <= 6 && PRI_SERVO_TO >= 6) || (SEC_SERVO_FROM <= 6 && SEC_SERVO_TO >=
6)
      SERVO_6_PIN_LOW;
    #endif
    #if (PRI_SERVO_FROM <= 7 && PRI_SERVO_TO >= 7) || (SEC_SERVO_FROM <= 7 && SEC_SERVO_TO >=
7)
      SERVO_7_PIN_HIGH;
    #endif
    state++;
    SERVO_CHANNEL+= SERVO_1K_US; // 1000 us
  } else if (state == 13) {
    SERVO_CHANNEL+= atomicServo[6]; // 1000 + [0-1020] us
    state++;
  } else if (state == 14) {
    #if (PRI_SERVO_FROM <= 7 && PRI_SERVO_TO >= 7) || (SEC_SERVO_FROM <= 7 && SEC_SERVO_TO >=
7)
      SERVO_7_PIN_LOW;
    #endif
    #if (PRI_SERVO_FROM <= 8 && PRI_SERVO_TO >= 8) || (SEC_SERVO_FROM <= 8 && SEC_SERVO_TO >=
8)
      SERVO_8_PIN_HIGH;
    #endif
    state++;
    SERVO_CHANNEL+= SERVO_1K_US; // 1000 us
  } else if (state == 15) {
    SERVO_CHANNEL+= atomicServo[7]; // 1000 + [0-1020] us
    state++;
  } else if (state == 16) {
    #if (PRI_SERVO_FROM <= 8 && PRI_SERVO_TO >= 8) || (SEC_SERVO_FROM <= 8 && SEC_SERVO_TO >=
8)
      SERVO_8_PIN_LOW;
    #endif
    count = 2;
    state++;
    SERVO_CHANNEL+= SERVO_1K_US; // 1000 us
  } else if (state == 17) {
    if (count > 0) count--;
```

```
    else state = 0;
    SERVO_CHANNEL+= SERVO_1K_US;
  }
}
#endif

#if (NUMBER_MOTOR > 4) && (defined(PROMINI) || defined(PROMICRO))

  #if !defined(PROMICRO) || defined(HWPWM6)
    #define SOFT_PWM_ISR1 TIMER0_COMPB_vect
    #define SOFT_PWM_ISR2 TIMER0_COMPA_vect
    #define SOFT_PWM_CHANNEL1 OCR0B
    #define SOFT_PWM_CHANNEL2 OCR0A
  #else
    #define SOFT_PWM_ISR1 TIMER3_COMPB_vect
    #define SOFT_PWM_ISR2 TIMER3_COMPC_vect
    #define SOFT_PWM_CHANNEL1 OCR3B
    #define SOFT_PWM_CHANNEL2 OCR3C
  #endif

  void initializeSoftPWM() {
    #if !defined(PROMICRO) || defined(HWPWM6)
      TCCR0A = 0; // normal counting mode
      #if (NUMBER_MOTOR > 4) && !defined(HWPWM6)
        TIMSK0 |= (1<<OCIE0B); // Enable CTC interrupt
      #endif
      #if (NUMBER_MOTOR > 6)
        TIMSK0 |= (1<<OCIE0A);
      #endif
    #else
        TCCR3A &= ~(1<<WGM30); // normal counting mode
        TCCR3B &= ~(1<<CS31); // no prescaler
        TIMSK3 |= (1<<OCIE3B); // Enable CTC interrupt
        #if (NUMBER_MOTOR > 6)
          TIMSK3 |= (1<<OCIE3C);
        #endif
    #endif
  }

  // HEXA with just OCR0B
  ISR(SOFT_PWM_ISR1) {
    static uint8_t state = 0;
    if(state == 0){
      #if !defined(A0_A1_PIN_HEX) && (NUMBER_MOTOR < 8)
        SOFT_PWM_1_PIN_HIGH;
      #else
        PORTC |= 1<<0;//PIN A0
      #endif
      SOFT_PWM_CHANNEL1 += atomicPWM_PIN5_highState;
      state = 1;
    }else if(state == 1){
      #if !defined(A0_A1_PIN_HEX) && (NUMBER_MOTOR < 8)
        SOFT_PWM_2_PIN_LOW;
      #else
        PORTC &= ~(1<<1);
      #endif
      SOFT_PWM_CHANNEL1 += atomicPWM_PIN6_lowState;
      state = 2;
    }else if(state == 2){
      #if !defined(A0_A1_PIN_HEX) && (NUMBER_MOTOR < 8)
       SOFT_PWM_2_PIN_HIGH;
      #else
        PORTC |= 1<<1;//PIN A1
      #endif
      SOFT_PWM_CHANNEL1 += atomicPWM_PIN6_highState;
      state = 3;
    }else if(state == 3){
```

```
      #if !defined(A0_A1_PIN_HEX) && (NUMBER_MOTOR < 8)
        SOFT_PWM_1_PIN_LOW;
      #else
        PORTC &= ~(1<<0);
      #endif
      SOFT_PWM_CHANNEL1 += atomicPWM_PIN5_lowState;
      state = 0;
    }
  }

  //the same with digital PIN A2 & 12 OCR0A counter for OCTO
  #if (NUMBER_MOTOR > 6) && !defined(HWPWM6)
    ISR(SOFT_PWM_ISR2) {
      static uint8_t state = 0;
      if(state == 0){
        SOFT_PWM_3_PIN_HIGH;
        SOFT_PWM_CHANNEL2 += atomicPWM_PINA2_highState;
        state = 1;
      }else if(state == 1){
        SOFT_PWM_4_PIN_LOW;
        SOFT_PWM_CHANNEL2 += atomicPWM_PIN12_lowState;
        state = 2;
      }else if(state == 2){
        SOFT_PWM_4_PIN_HIGH;
        SOFT_PWM_CHANNEL2 += atomicPWM_PIN12_highState;
        state = 3;
      }else if(state == 3){
        SOFT_PWM_3_PIN_LOW;
        SOFT_PWM_CHANNEL2 += atomicPWM_PINA2_lowState;
        state = 0;
      }
    }
  #endif
#endif

void mixTable() {
  int16_t maxMotor;
  uint8_t i;
  //Nuestro
  #define PIDMIX(X,Y,Z) constrain((rcCommand[THROTTLE]*2.7271 + X*axisPID[ROLL] + Y*axisPID[PITCH] +
Z*axisPID[YAW])*0.0363 + 834,1155,1845)
  //#define PIDMIX(X,Y,Z) 1600 + axisPID[ROLL]/100
  //#if NUMBER_MOTOR > 3
  //  //prevent "yaw jump" during yaw correction
  //  axisPID[YAW] = constrain(axisPID[YAW],-100-abs(rcCommand[YAW]),+100+abs(rcCommand[YAW]));
  //#endif
  #ifdef BI
    motor[0] = PIDMIX(+1, 0, 0); //LEFT
    motor[1] = PIDMIX(-1, 0, 0); //RIGHT
    servo[4]  = constrain(1500 + YAW_DIRECTION * (axisPID[YAW] + axisPID[PITCH]), 1020, 2000); //LEFT
    servo[5]  = constrain(1500 + YAW_DIRECTION * (axisPID[YAW] - axisPID[PITCH]), 1020, 2000); //RIGHT
  #endif
  #ifdef TRI
    motor[0] = PIDMIX( 0,+4/3, 0); //REAR
    motor[1] = PIDMIX(-1,-2/3, 0); //RIGHT
    motor[2] = PIDMIX(+1,-2/3, 0); //LEFT
    servo[5] = constrain(tri_yaw_middle + YAW_DIRECTION * axisPID[YAW], TRI_YAW_CONSTRAINT_MIN,
TRI_YAW_CONSTRAINT_MAX); //REAR
  #endif
  #ifdef QUADP //Nuestro
    motor[0] = PIDMIX( 0,+5.4542,-1.5770); //REAR
    motor[1] = PIDMIX(-5.4542, 0,+1.5770); //RIGHT
    motor[2] = PIDMIX(+5.4542, 0,+1.5770); //LEFT
    motor[3] = PIDMIX( 0,-5.4542,-1.5770); //FRONT
  #endif
  #ifdef QUADX //Nuestro
    motor[0] = PIDMIX(-2.7271,+2.7271,-1.5770); //REAR_R
```

```
  motor[1] = PIDMIX(-2.7271,-2.7271,+1.5770); //FRONT_R
  motor[2] = PIDMIX(+2.7271,+2.7271,+1.5770); //REAR_L
  motor[3] = PIDMIX(+2.7271,-2.7271,-1.5770); //FRONT_L
#endif
#ifdef Y4
  motor[0] = PIDMIX(+0,+1,-1);   //REAR_1 CW
  motor[1] = PIDMIX(-1,-1, 0); //FRONT_R CCW
  motor[2] = PIDMIX(+0,+1,+1);   //REAR_2 CCW
  motor[3] = PIDMIX(+1,-1, 0); //FRONT_L CW
#endif
#ifdef Y6
  motor[0] = PIDMIX(+0,+4/3,+1); //REAR
  motor[1] = PIDMIX(-1,-2/3,-1); //RIGHT
  motor[2] = PIDMIX(+1,-2/3,-1); //LEFT
  motor[3] = PIDMIX(+0,+4/3,-1); //UNDER_REAR
  motor[4] = PIDMIX(-1,-2/3,+1); //UNDER_RIGHT
  motor[5] = PIDMIX(+1,-2/3,+1); //UNDER_LEFT
#endif
#ifdef HEX6
  motor[0] = PIDMIX(-1/2,+1/2,+1); //REAR_R
  motor[1] = PIDMIX(-1/2,-1/2,-1); //FRONT_R
  motor[2] = PIDMIX(+1/2,+1/2,+1); //REAR_L
  motor[3] = PIDMIX(+1/2,-1/2,-1); //FRONT_L
  motor[4] = PIDMIX(+0  ,-1  ,+1); //FRONT
  motor[5] = PIDMIX(+0  ,+1  ,-1); //REAR
#endif
#ifdef HEX6X
  motor[0] = PIDMIX(-1/2,+1/2,+1); //REAR_R
  motor[1] = PIDMIX(-1/2,-1/2,+1); //FRONT_R
  motor[2] = PIDMIX(+1/2,+1/2,-1); //REAR_L
  motor[3] = PIDMIX(+1/2,-1/2,-1); //FRONT_L
  motor[4] = PIDMIX(-1  ,+0  ,-1); //RIGHT
  motor[5] = PIDMIX(+1  ,+0  ,+1); //LEFT
#endif
#ifdef OCTOX8
  motor[0] = PIDMIX(-1,+1,-1); //REAR_R
  motor[1] = PIDMIX(-1,-1,+1); //FRONT_R
  motor[2] = PIDMIX(+1,+1,+1); //REAR_L
  motor[3] = PIDMIX(+1,-1,-1); //FRONT_L
  motor[4] = PIDMIX(-1,+1,+1); //UNDER_REAR_R
  motor[5] = PIDMIX(-1,-1,-1); //UNDER_FRONT_R
  motor[6] = PIDMIX(+1,+1,-1); //UNDER_REAR_L
  motor[7] = PIDMIX(+1,-1,+1); //UNDER_FRONT_L
#endif
#ifdef OCTOFLATP
  motor[0] = PIDMIX(+7/10,-7/10,+1); //FRONT_L
  motor[1] = PIDMIX(-7/10,-7/10,+1); //FRONT_R
  motor[2] = PIDMIX(-7/10,+7/10,+1); //REAR_R
  motor[3] = PIDMIX(+7/10,+7/10,+1); //REAR_L
  motor[4] = PIDMIX(+0   ,-1   ,-1); //FRONT
  motor[5] = PIDMIX(-1   ,+0   ,-1); //RIGHT
  motor[6] = PIDMIX(+0   ,+1   ,-1); //REAR
  motor[7] = PIDMIX(+1   ,+0   ,-1); //LEFT
#endif
#ifdef OCTOFLATX
  motor[0] = PIDMIX(+1  ,-1/2,+1); //MIDFRONT_L
  motor[1] = PIDMIX(-1/2,-1  ,+1); //FRONT_R
  motor[2] = PIDMIX(-1  ,+1/2,+1); //MIDREAR_R
  motor[3] = PIDMIX(+1/2,+1  ,+1); //REAR_L
  motor[4] = PIDMIX(+1/2,-1  ,-1); //FRONT_L
  motor[5] = PIDMIX(-1  ,-1/2,-1); //MIDFRONT_R
  motor[6] = PIDMIX(-1/2,+1  ,-1); //REAR_R
  motor[7] = PIDMIX(+1  ,+1/2,-1); //MIDREAR_L
#endif
#ifdef VTAIL4
  motor[0] = PIDMIX(+0,+1, -1/2);      //REAR_R
  motor[1] = PIDMIX(-1, -1, +2/10); //FRONT_R
```

```
  motor[2] = PIDMIX(+0,+1, +1/2);      //REAR_L
  motor[3] = PIDMIX(+1, -1, -2/10); //FRONT_L
 #endif

 #ifdef SERVO_TILT
  #if defined(A0_A1_PIN_HEX) && (NUMBER_MOTOR == 6) && defined(PROMINI)
    #define S_PITCH servo[2]
    #define S_ROLL  servo[3]
  #else
    #define S_PITCH servo[0]
    #define S_ROLL  servo[1]
  #endif
  S_PITCH = TILT_PITCH_MIDDLE + rcData[AUX3]-1500;
  S_ROLL  = TILT_ROLL_MIDDLE  + rcData[AUX4]-1500;
  if (rcOptions[BOXCAMSTAB]) {
    S_PITCH += TILT_PITCH_PROP * angle[PITCH] /16 ;
    S_ROLL  += TILT_ROLL_PROP  * angle[ROLL]  /16 ;
  }
  S_PITCH = constrain(S_PITCH, TILT_PITCH_MIN, TILT_PITCH_MAX);
  S_ROLL  = constrain(S_ROLL , TILT_ROLL_MIN, TILT_ROLL_MAX  );
 #endif
 #ifdef GIMBAL
  servo[0]  =  constrain(TILT_PITCH_MIDDLE  +  TILT_PITCH_PROP  *  angle[PITCH]  /16  +  rcCommand[PITCH],
TILT_PITCH_MIN, TILT_PITCH_MAX);
  servo[1]  =  constrain(TILT_ROLL_MIDDLE  +  TILT_ROLL_PROP     *  angle[ROLL]   /16  +  rcCommand[ROLL],
TILT_ROLL_MIN, TILT_ROLL_MAX);
 #endif
 #ifdef FLYING_WING
  motor[0] = rcCommand[THROTTLE];
  if (passThruMode) {// do not use sensors for correction, simple 2 channel mixing
    servo[0]    =  PITCH_DIRECTION_L  *  (rcData[PITCH]-MIDRC)  +  ROLL_DIRECTION_L  *  (rcData[ROLL]-
MIDRC);
    servo[1]    =  PITCH_DIRECTION_R  *  (rcData[PITCH]-MIDRC)  +  ROLL_DIRECTION_R  *  (rcData[ROLL]-
MIDRC);
  } else { // use sensors to correct (gyro only or gyro+acc according to aux1/aux2 configuration
    servo[0]  = PITCH_DIRECTION_L * axisPID[PITCH]       + ROLL_DIRECTION_L * axisPID[ROLL];
    servo[1]  = PITCH_DIRECTION_R * axisPID[PITCH]       + ROLL_DIRECTION_R * axisPID[ROLL];
  }
  servo[0]  = constrain(servo[0] + wing_left_mid , WING_LEFT_MIN,  WING_LEFT_MAX );
  servo[1]  = constrain(servo[1] + wing_right_mid, WING_RIGHT_MIN, WING_RIGHT_MAX);
 #endif
 #if defined(CAMTRIG)
  static uint8_t camCycle = 0;
  static uint8_t camState = 0;
  static uint32_t camTime = 0;
  if (camCycle==1) {
   if (camState == 0) {
    servo[2] = CAM_SERVO_HIGH;
    camState = 1;
    camTime = millis();
   } else if (camState == 1) {
    if ( (millis() - camTime) > CAM_TIME_HIGH ) {
     servo[2] = CAM_SERVO_LOW;
     camState = 2;
     camTime = millis();
    }
   } else { //camState ==2
    if ( (millis() - camTime) > CAM_TIME_LOW ) {
     camState = 0;
     camCycle = 0;
    }
   }
  }
  if (rcOptions[BOXCAMTRIG]) camCycle=1;
 #endif

 maxMotor=motor[0];
```

```
  for(i=1;i< NUMBER_MOTOR;i++)
    if (motor[i]>maxMotor) maxMotor=motor[i];
  for (i = 0; i < NUMBER_MOTOR; i++) {
    if (maxMotor > MAXTHROTTLE) // this is a way to still have good gyro corrections if at least one motor reaches its
max.
      motor[i] -= maxMotor - MAXTHROTTLE;
    motor[i] = constrain(motor[i], MINTHROTTLE, MAXTHROTTLE);
    if ((rcData[THROTTLE]) < MINCHECK)
      #ifndef MOTOR_STOP
        motor[i] = MINTHROTTLE;
      #else
        motor[i] = MINCOMMAND;
      #endif
    if (armed == 0)
      motor[i] = MINCOMMAND;
  }

  #if (LOG_VALUES == 2) || defined(POWERMETER_SOFT)
    uint32_t amp;
    /* true cubic function; when divided by vbat_max=126 (12.6V) for 3 cell battery this gives maximum value of ~ 500 */
    /* Lookup table moved to PROGMEM 11/21/2001 by Danal */
    static uint16_t amperes[64] =   {   0,  2,  6, 15, 30, 52, 82,123,
                                    175,240,320,415,528,659,811,984,
                                    1181,1402,1648,1923,2226,2559,2924,3322,
                                    3755,4224,4730,5276,5861,6489,7160,7875,
                                    8637 ,9446 ,10304,11213,12173,13187,14256,15381,
                                    16564,17805,19108,20472,21900,23392,24951,26578,
                                    28274,30041,31879,33792,35779,37843,39984,42205,
                                    44507,46890,49358,51910,54549,57276,60093,63000};

    if (vbat) { // by all means - must avoid division by zero
      for (i =0;i<NUMBER_MOTOR;i++) {
        amp = amperes[ ((motor[i] - 1000)>>4) ] / vbat; // range mapped from [1000:2000] => [0:1000]; then break that up
into 64 ranges; lookup amp
                #if (LOG_VALUES == 2)
          pMeter[i]+= amp; // sum up over time the mapped ESC input
        #endif
        #if defined(POWERMETER_SOFT)
          pMeter[PMOTOR_SUM]+= amp; // total sum over all motors
        #endif
      }
    }
  #endif
}
```