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

(APÉNDICES Y ANEXOS)

TRABAJO ESPECIAL DE GRADO

presentado ante la

UNIVERSIDAD CATÓLICA ANDRÉS BELLO

como parte de los requisitos para optar al título de

INGENIERO INFORMÁTICO

REALIZADO POR: **Luis Vicens**

Yoshua Nava

TUTOR ACADÉMICO: **Ing. Evelenir Barreto**

FECHA: **Enero de 2015**

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

Capítulo VII: Apéndices y anexos

Apéndice A: Diseño del circuito de alimentación y control de motores

Para realizar la conmutación a alta velocidad de cada motor en un solo sentido, se utilizó un MOSFET IRFZ44N y un diodo U16C20C, en configuración de rodada libre, para forzar la descarga de la inductancia del motor al cerrar el canal del MOSFET. Estos fueron seleccionados de entre todos los componentes disponibles en el mercado venezolano por su bajo precio, alta velocidad de conmutación, alta tolerancia y estabilidad ante valores altos de corriente y voltaje, y baja resistencia drenador-surtidor en el caso del IRFZ44N. Se realizaron pruebas para caracterizar la conductividad y resistencia interna del MOSFET ante distintos valores de voltaje aplicados en la compuerta del mismo, al encender uno de los motores del cuadricóptero de forma continua (Ciclo de trabajo de 100% con un voltaje Vgs fijo), con una alimentación de 12V -cercana al voltaje máximo de carga de la batería de polímero de litio seleccionada para alimentar el conjunto-.

Para realizar la caracterización de los MOSFET IRFZ44N se utilizó el siguiente circuito de prueba:

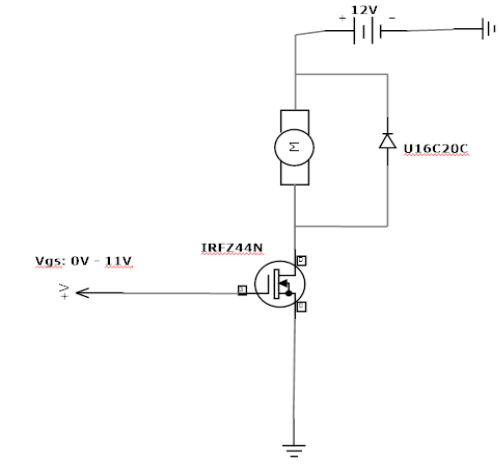

Ilustración 7. 1 Diagrama de circuito para estimación de Vgs óptimo para conmutar MOSFET con carga de motor DC en el surtidor.

Fuente: elaboración propia

A continuación se presenta la curva característica de corriente drenador-surtidor (Ids) en función del voltaje de compuerta (Vgs) del MOSFET IRFZ44N, identificada a partir de las pruebas:

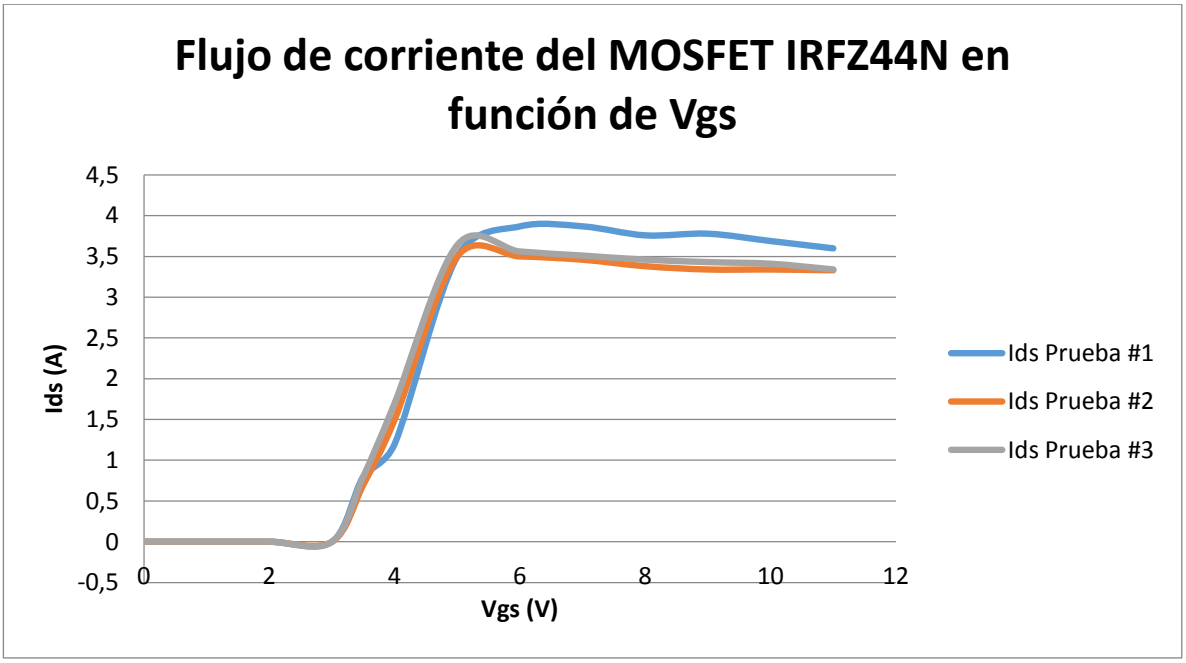

*Ilustración 7. 2 Relación **Voltaje de compuerta** – **Corriente drenador-surtidor** del MOSFET IRFZ44N, en presencia de carga del motor.*

Fuente: elaboración propia.

De igual manera, a continuación se presenta la curva característica de resistencia drenador-surtidor (Ids) en función del voltaje de compuerta (Vgs) del MOSFET IRFZ44N:

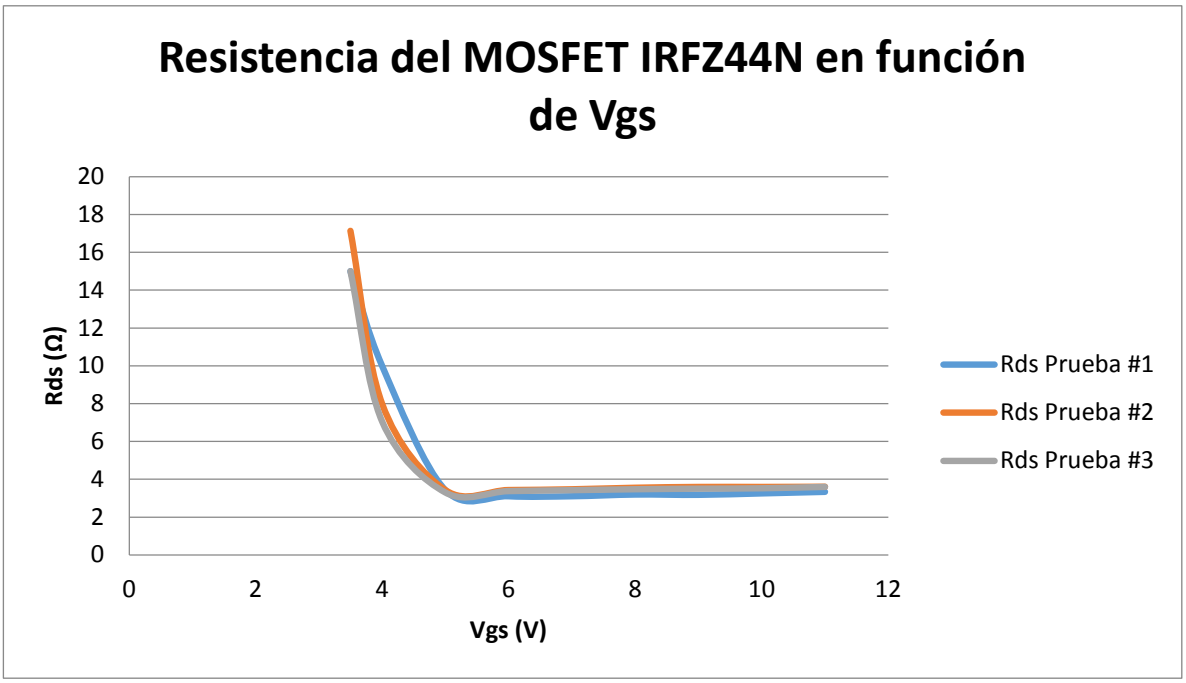

*Ilustración 7. 3 Relación **Voltaje de compuerta** - **Corriente drenador-surtidor** del MOSFET IRFZ44N, en presencia de carga del motor.*

Fuente: elaboración propia.

A partir del análisis realizado se identificó 4,5 y 6 voltios como el rango de valores de tensión de compuerta óptimos para el funcionamiento del MOSFET con los motores del cuadricóptero como carga en el drenador. Considerando el efecto de descarga de la batería a utilizar, se decidió utilizar un voltaje de al menos 6V, siendo éste cota superior del rango de valores óptimos identificado, como tensión de activación de la compuerta del MOSFET IRFZ44N.

Para disminuir los efectos de ruido electromagnético que pudieran ser introducidos por la rápida conmutación de los MOSFETS y los motores al circuito de lógica, sensores y comunicación, el cual envía las señales de PWM para la regulación de velocidad de los motores, se decidió utilizar opto-acopladores para separar totalmente la etapa. Se seleccionó el modelo 4N26, presente en el mercado de componentes electrónicos venezolano, por su alta velocidad de conmutación, tolerancia a altos valores de voltaje y corriente, bajo precio, y simplicidad de configuración. Para obtener la tensión de salida de 6V para conmutar la compuerta del MOSFET IRFZ44N se diseñó un circuito que hace uso de un divisor de voltaje

en el emisor del fototransistor, en configuración colector común, como se presenta a continuación:

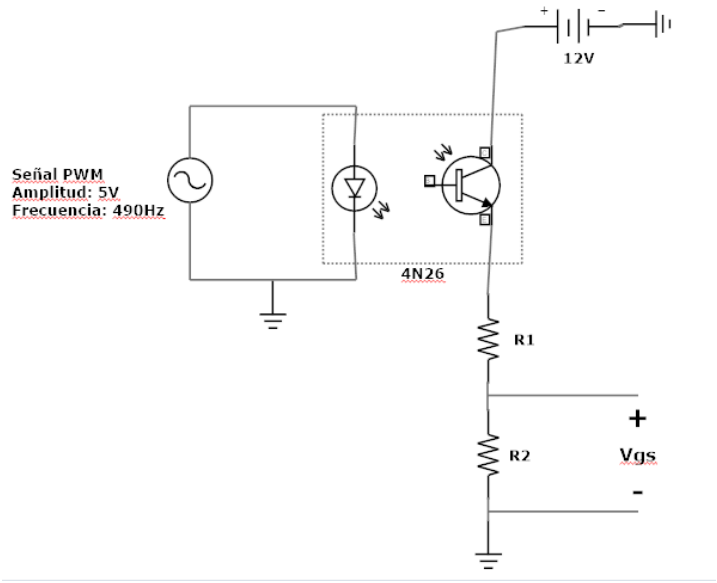

Ilustración 7. 4 Diagrama de circuito para estimación de carga total de foto-transistor del opto-acoplador 4N26 en configuración colector común.

Fuente: elaboración propia.

Para calcular la resistencia total de la carga del colector común del fototransistor se evaluó el tiempo de respuesta del mismo ante impulsos de PWM de 490 Hz. Se realizaron pruebas con ayuda de un osciloscopio digital para medir el rendimiento del opto-acoplador 4N26, y se seleccionó un valor de 900Ω para la resistencia de carga total del opto-acoplador en configuración colector común, por obtenerse una alta velocidad de respuesta en el opto-acoplador. A partir del valor de resistencia total seleccionado se realizó el cálculo de las resistencias R1 y R2 para el circuito de conmutación con divisor de voltaje. El valor calculado para R1 fue de 390Ω, y para R2 fue de 510Ω.

En la figura que sigue se presenta la respuesta del circuito de conmutación diseñado al recibir una señal de PWM con ciclo de trabajo de 98%:

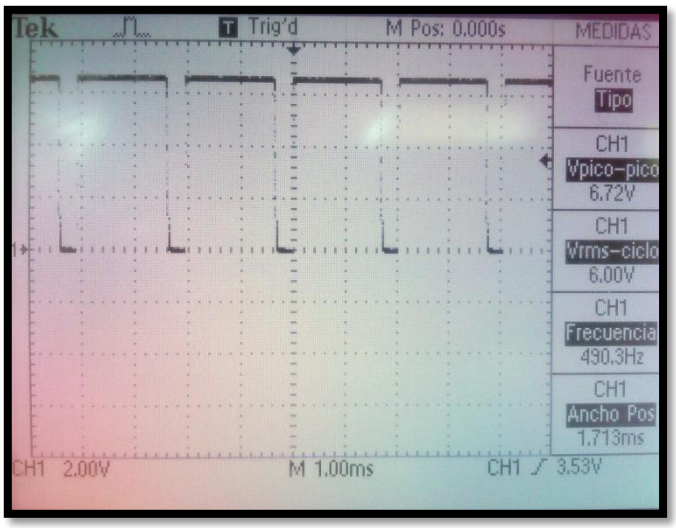

Ilustración 7. 5 Señal de salida del opto-acoplador ante una señal de PWM con un ciclo de trabajo de 98%.

Fuente: elaboración propia.

Posteriormente, se sometió al circuito a una señal de PWM con ciclo de trabajo de 23%, como se ilustra a continuación:

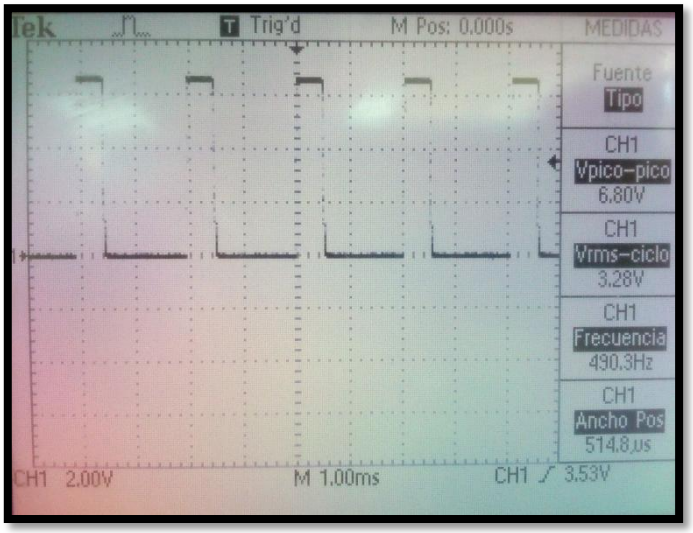

Ilustración 7. 6 Señal de salida del opto-acoplador ante una señal de PWM con un ciclo de trabajo de 23%.

Fuente: elaboración propia.

El circuito de regulación de velocidad diseñado, para el control de velocidad individual de cada motor se presenta en el diagrama que sigue:

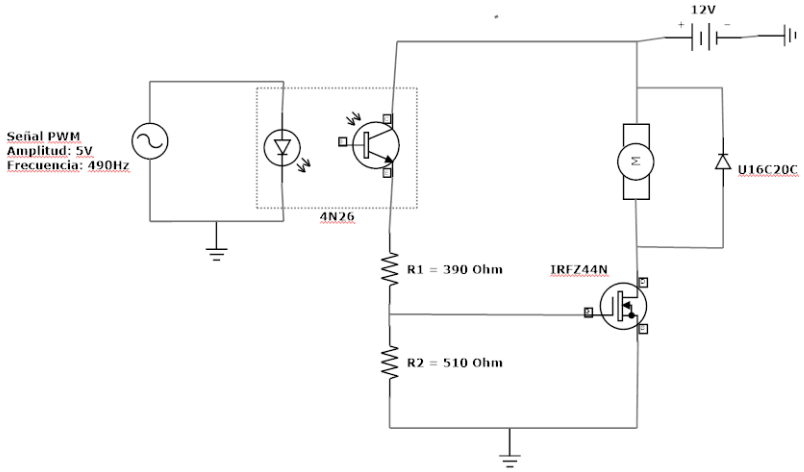

Ilustración 7. 7 Diagrama de circuito de regulación de velocidad diseñado.

Fuente: elaboración propia.

Finalmente, se diseñó y construyó una placa de circuito impreso de una sola capa consistente en un arreglo de cuatro módulos para regulación de velocidad desarrollados en el presente trabajo. A continuación se muestra el diseño del mismo:

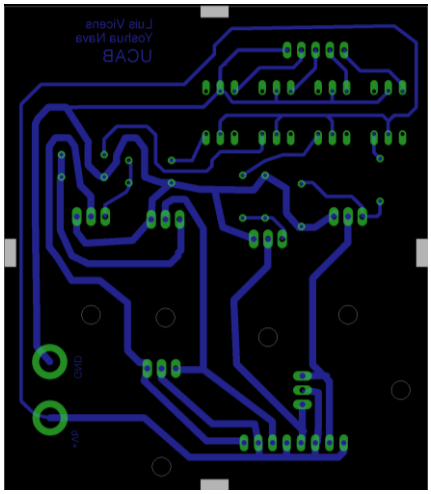

Ilustración 7. 8 Diseño de circuito impreso del circuito de alimentación y control de motores.

Fuente: elaboración propia.

Apéndice B: Descripción del modelo de estimación de posición angular y velocidad angular del cuadricóptero

a. Cálculo de velocidad angular

El cálculo de velocidad angular se realizó a partir de las mediciones realizadas con el giroscopio. Partiendo de la información provista por la hoja de datos del sensor L3GD20, el mismo puede configurarse para obtener una sensibilidad de 8,75, 17,5 y 70 milésimas de grado por segundo por cada dígito de medición obtenido (mdps/digit – en inglés *millidegrees per second per digit*), y un rango de medición de 250, 500 y 2000 grados por segundo (dps – en inglés *degree per second*). Se decidió configurar el rango de medición a 250 grados por segundo y la senbilidad del sensor a 8,75 mdps/digit, por considerarse rango y senbilidad suficientes para medir las velocidades del cuadricóptero realizando movimientos simples en vuelo. En base a esto último se calculó la ganancia del giroscopio, para convertir todas las mediciones obtenidas mediante el mismo, en milésimas de grado por segundo, a grados por segundo, como se ilustra a continuación:

$$G_{GYRO} = \frac{Sensibilidad\ (\boldsymbol{\frac{mdps}{digit}})}{1000} = \frac{8{,}75\ \boldsymbol{\frac{mdps}{digit}}}{1000} = 0{,}00875\ \boldsymbol{\frac{dps}{digit}} \quad \textbf{(1)}$$

$$\omega_g = G_{GYRO} * \omega_m \quad \textbf{(2)}$$

siendo:

G_{GYRO}: $ganancia\ del\ giroscopio$

$Senbilidad\ \left(\boldsymbol{\frac{mdps}{digit}}\right)$: $sensibilidad\ del\ giroscopio.$

ω_g: $velocidad\ angular\ medida\ en\ grados\ por\ segundo.$

ω_m: $velocidad\ angular\ medida\ en\ milésimas\ de\ grado\ por\ segundo.$

b. Estimación de ángulos de Pitch y Roll a partir del acelerómetro

Se realizó un estimado del ángulo de inclinación del cuadricóptero a partir de las mediciones del acelerómero, el cual provee una descomposición de la fuerza de aceleración del cuadricóptero en tres (3) ejes perpendiculares (x, y, z). El acelerómetro puede detectar constantemente la fuerza de gravedad, en magnitud, dirección, y fuerza, en cada uno de sus tres (3) ejes, y en base a ello puede establecerse un marco de referencia absoluto a partir del cual calcular los ángulos de Pitch y Roll del sensor, y en consecuencia, del cuadricóptero. La estimación de ángulos se realizó siguiendo el procedimiento expuesto en [STMicroElectronics 2010], el cual es presentado a continuación:

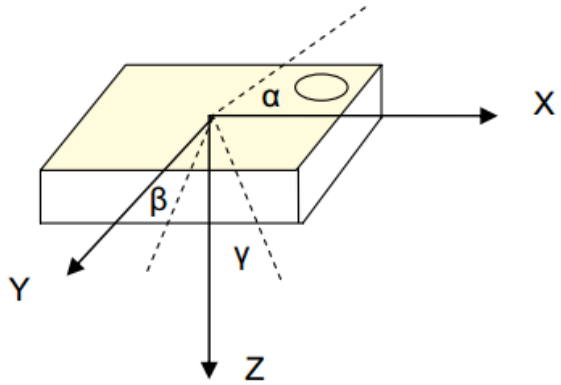

Ilustración 7. 9 Modelo de estimación de angulos

Fuente: [STMicroElectronics]

$$\alpha = angulo_{Roll} = \tan^{-1}\left(\frac{A_x}{\sqrt{\left(A_y{}^2 + A_z{}^2\right)}}\right) \quad \textbf{(3)}$$

$$\beta = angulo_{Pitch} = \tan^{-1}\left(\frac{A_y}{\sqrt{\left(A_x{}^2 + A_z{}^2\right)}}\right) \quad \textbf{(4)}$$

siendo:

Ax: $valor\ de\ las\ fuerzas\ sobre\ el\ eje\ x\ del\ acelerómetro.$

Ay: $valor\ de\ las\ fuerzas\ sobre\ el\ eje\ y\ del\ acelerómetro.$

Az: $valor\ de\ las\ fuerzas\ sobre\ el\ eje\ z\ del\ acelerómetro.$

Las estimaciones de los ángulos de Pitch y Roll calculadas a partir de los datos

del acelerómetro, a pesar de ser precisas y permitir mantener un marco de referencia absoluto, con base en la fuerza de gravedad de la tierra, presentan un alto porcentaje de ruido, ya que el acelerómetro es altamente sensible a perturbaciones provocadas por fuerzas externas que incidan sobre el mismo.

c. Estimación de posición angular a partir del giroscopio

Se realizó un estimado del ángulo de inclinación del cuadricóptero, mediante integración numérica de las velocidades de rotación de Yaw, Pitch y Roll, como se describe a continuación:

$$Ángulo_{Yaw} = \int_0^T \omega_{Yaw} * dt \approx \sum_0^T \omega_{Yaw} * dt \quad \textbf{(5)}$$

$$Ángulo_{Pitch} = \int_0^T \omega_{Pitch} * dt \approx \sum_0^T \omega_{Pitch} * dt \quad \textbf{(6)}$$

$$Ángulo_{Roll} = \int_0^T \omega_{Roll} * dt \approx \sum_0^T \omega_{Roll} * dt \quad \textbf{(7)}$$

donde:

ω_{Yaw}: $velocidad\ angular\ de\ Yaw\ medida\ en\ grados\ por\ segundo.$

ω_{Pitch}: $velocidad\ angular\ de\ Pitch\ medida\ en\ grados\ por\ segundo.$

ω_{Roll}: $velocidad\ angular\ de\ Roll\ medida\ en\ grados\ por\ segundo.$

dt: *diferencial de tiempo.*

Durante un lapso corto de tiempo, este estimado del ángulo de inclinación en cada eje puede ser preciso, pero, tiende a presentar derivada constante y a alejarse de los valores reales a medir, por efecto de vibraciones mecánicas, cambios de temperatura y errores numéricos.

d. Combinación de las estimaciones de posición angular del acelerómetro y giroscopio

Al combinar la precisión del acelerómetro para medir inclinación respecto al marco de referencia absoluto del planeta Tierra, con la sensibilidad y estabilidad de la estimación de ángulo realizada a partir de los datos del giroscopio para medir los movimientos de rotación alrededor de cada eje del sensor, puede obtenerse una estimación de ángulo precisa, estable, y de alta sensibilidad.

Se decidió utilizar un filtro digital de uso altamente extendido conocido como *Filtro Complementario* [Gaydou 2007], [Colton 2007], el cual se fundamenta en la combinación de un filtro pasa bajos y un filtro pasa altos, ambos de primer orden, para la composición de los espectros de frecuencias de dos señales lineales invariantes en el tiempo en una tercera señal de salida. El filtro complementario puede ser descrito mediante las siguientes ecuaciones en el dominio de la frecuencia [Gaydou 2007]:

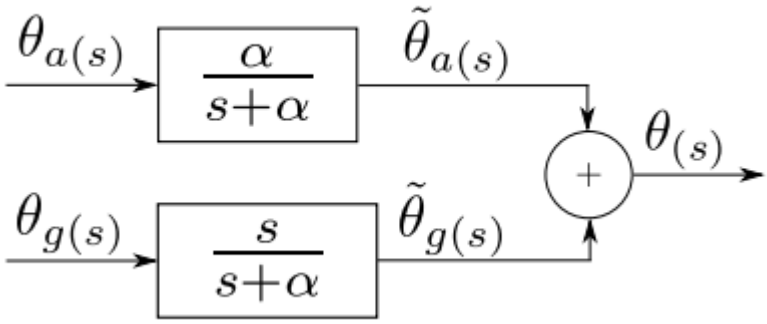

Ilustración 7. 10 Modelo del Filtro complementario
Fuente: [Gaydou 2007]

$$H_{A(s)} * F_{PA}(s) + H_{A(s)} * \left(1 - F_{PA}(s)\right) = 1 \quad \textbf{(8)}$$

$$F_{PA}(s) = \frac{\alpha}{s + \alpha} \quad \textbf{(9)}$$

En donde:

$H_{A(s)}$: función de transferencia del acelerómetro

$H_{G(s)}$: función de transferencia del giroscopio.

F_{PB}: filtro pasa altos de primer orden.

α: frecuencia de corte del filtro pasa altos de primer orden.

La ganancia del filtro pasa altos puede ser descrita a partir de la siguiente ecuación:

$$k = \frac{dt}{\tau + dt} \quad \textbf{(10)}$$

Donde:

τ: constante de tiempo del filtro.

dt: diferencial de tiempo.

El cálculo de la ganancia del filtro pasa altos se realizó mediante prueba y error, teniendo como criterio de selección la estabilidad, precisión y exactitud de la estimación de ángulo realizada, y el tiempo de respuesta de la misma. Se estableció un valor de k=0,03 para la ganancia del filtro.

Es importante recalcar que por las características de funcionamiento del acelerómetro, y el rango de la función $\tan^{-1}(x)$, el procedimiento de estimación de ángulos de Pitch y Roll presentado sólo permite la aproximación de los mismos en un rango entre menos noventa y noventa grados, sin poder detectar si el cuadricóptero se encuentra en un ángulo fuera de ese rango o volteado. No obstante, el algoritmo desarrollado satisface las necesidades del proyecto realizado en el presente trabajo de investigación, ya que el algoritmo a desarrollar sólo apunta a estabilizar el cuadricóptero, mas no a brindar posibilidades de realizar vuelo acrobático.

e. Respuesta en frecuencia de la Unidad de Medición Inercial y filtrado de datos

Se implementaron algoritmos en el micro-controlador Arduino Nano a partir del modelo de estimación de posición y velocidad angular desarrollado, y se recolectaron datos de los sensores, con los motores apagados y encendidos, a una frecuencia de 100Hz. Se encontró una gran cantidad de ruido e inestabilidad en la estimación en la muestra que se tomó con los motores del cuadricóptero encendidos. Se asumió que dicho ruido podría ser debido a interferencia eléctrica o vibraciones mecánicas, por lo cual se repitieron las pruebas con la IMU fuera del chasis del cuadricóptero, pero próxima al lugar en que iba a ser situada dentro del mismo. En las pruebas realizadas no se evidenció ruido o inestabilidad en las estimaciones, por lo cual se descartó que fueran producidas por interferencia eléctrica.

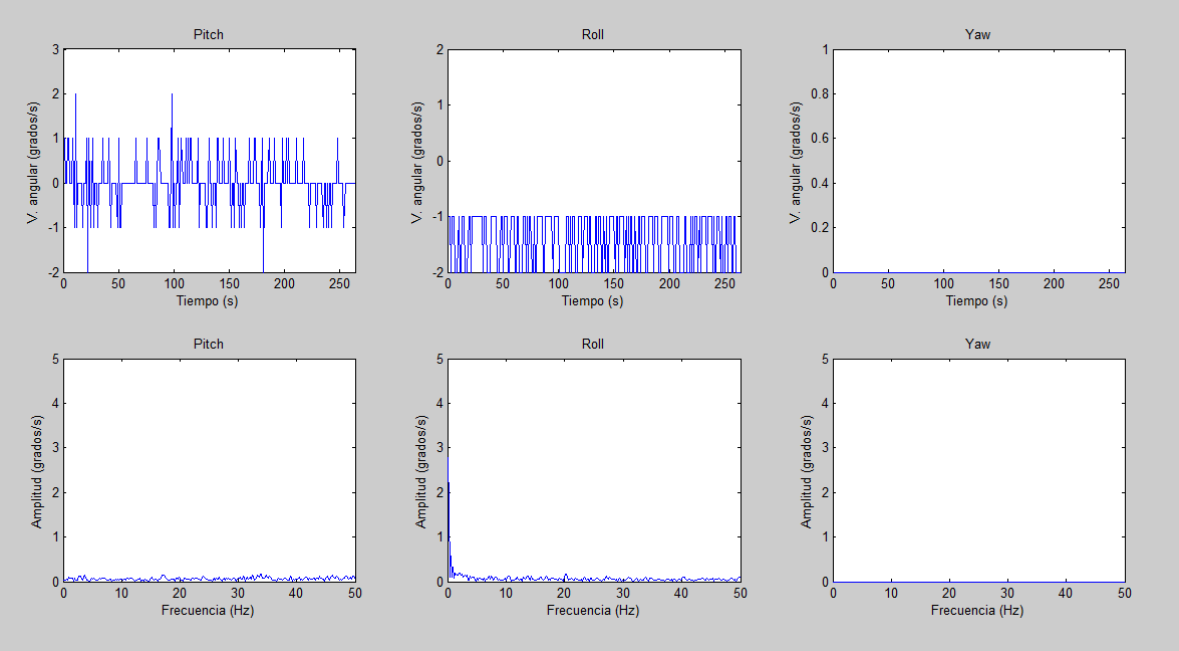

Ilustración 7. 11 Datos del giroscopio, con los motores encendidos y la IMU fuera de la cabina del cuadricóptero.

Fuente: elaboración propia.

Al someter los datos recopilados a un análisis en el dominio de la frecuencia mediante transformada rápida de Fourier, se encontró que las vibraciones se extendían a lo largo de todo el espectro de frecuencias, tenían una mayor potencia en el rango de cero (0) a veinte (20) Hz, y no poseían modas definidas, por lo cual no se podía rechazar ninguna banda en particular para reducir las vibraciones de forma significativa. A continuación se pueden visualizar datos extraídos del giroscopio, con los motores encendidos, y dentro de la cabina:

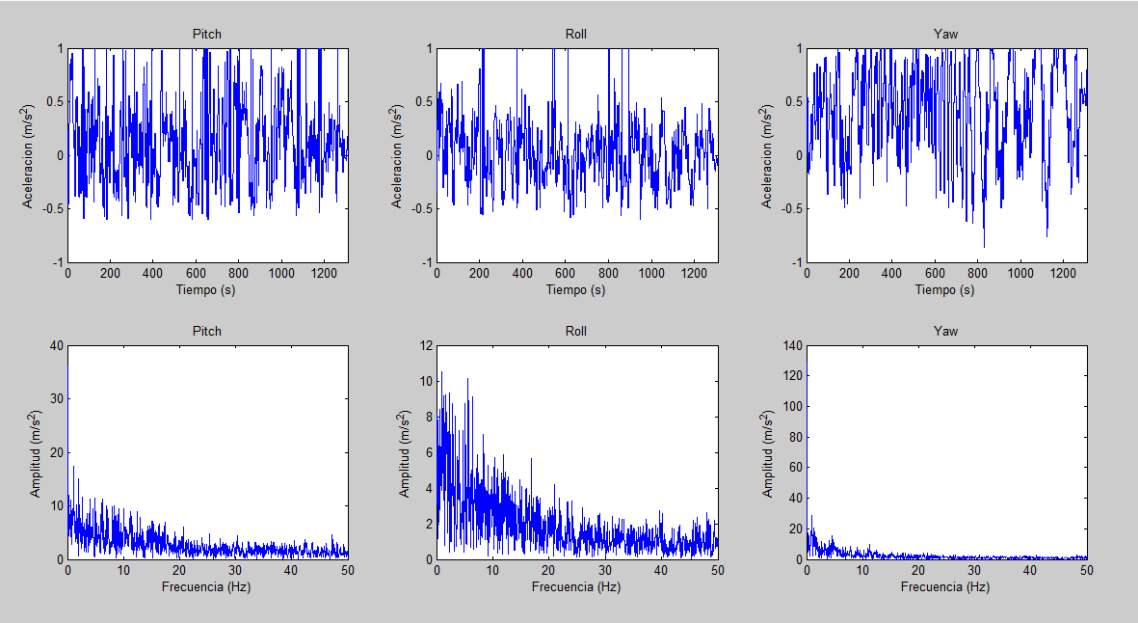

Ilustración 7. 12 Datos del acelerómetro, con los motores encendidos y la IMU dentro de la cabina del cuadricóptero.

Fuente: elaboración propia.

Bajo las mismas condiciones se recogieron datos del acelerómetro, los cuales se exponen a continuación:

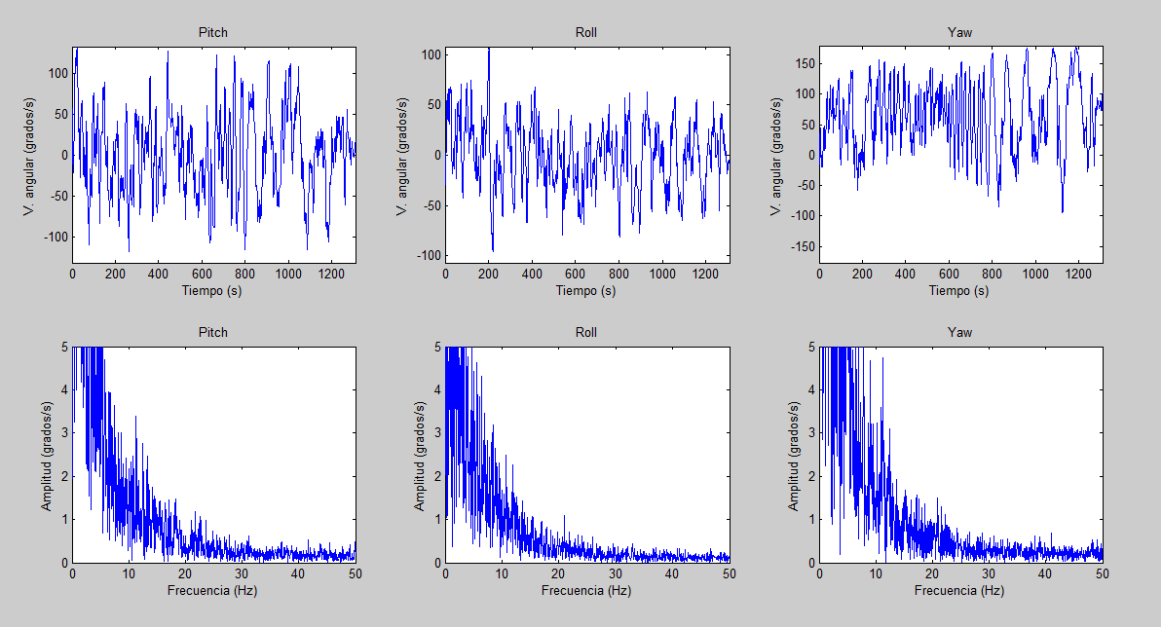

Ilustración 7. 13 Datos del giroscopio, con los motores encendidos y la IMU dentro de la cabina del cuadricóptero.

Fuente: elaboración propia.

Para disminuir las vibraciones que transfería el cuadricóptero a la IMU se construyó una base compuesta por Moongel y goma-espuma, para reducir las vibraciones de alta y baja frecuencia, respectivamente Dicha base se colocó alrededor de la IMU para que funcionara de amortiguación entre el cuadricóptero y esta. Adicionalmente, se agregó peso a la base construída mediante arandelas de hierro pequeñas recubiertas con cinta aislante, para aumentar la inercia del conjunto, y disminuir la influencia de las vibraciones sobre el sensor.

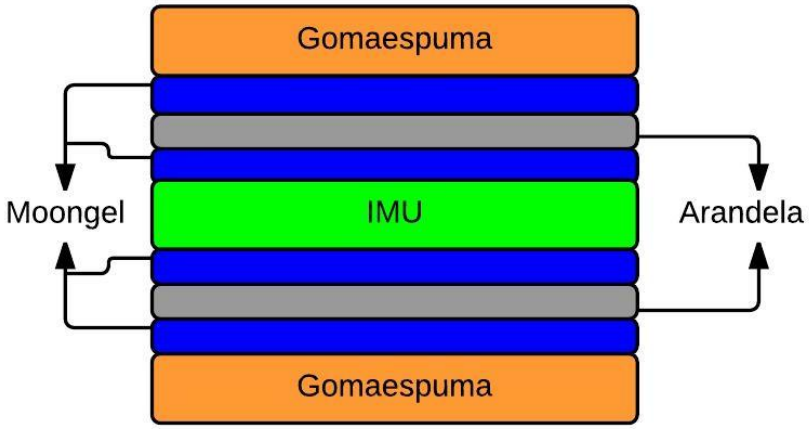

Ilustración 7. 14 Posicionamiento de la IMU sobre la base para amortiguación.

Fuente: elaboración propia.

Al recoger los datos con esta base se evidenció que la amplitud de las vibraciones disminuyó.

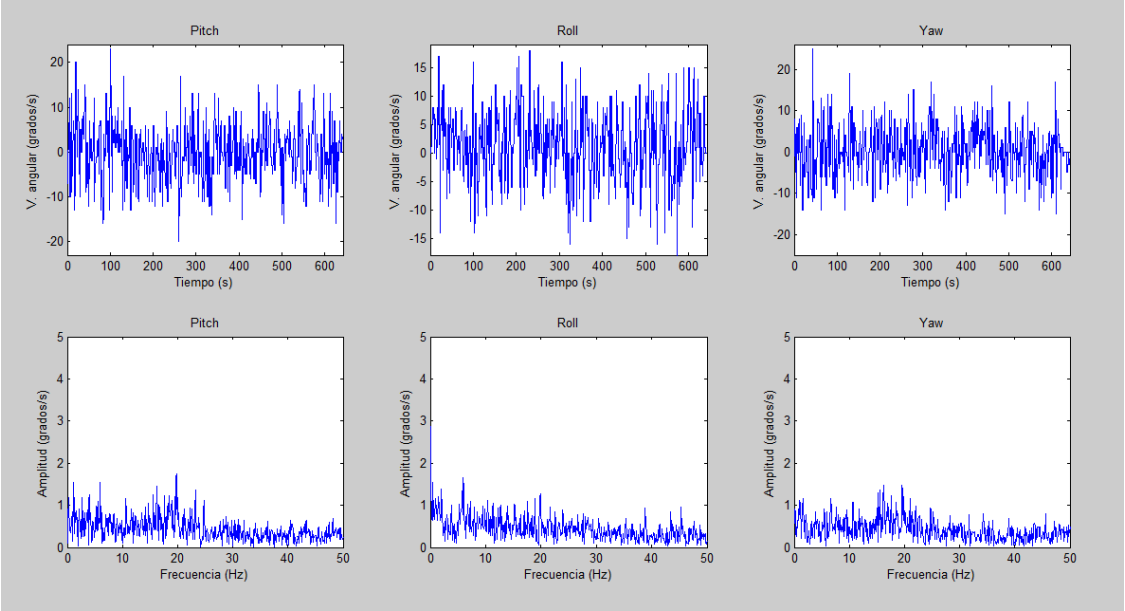

Ilustración 7. 15 Datos del giroscopio con motores encendidos y base de amortiguación.

Fuente: elaboración propia.

Nuevamente se recogieron datos del cuadricóptero con los motores encendidos, y se realizó un análisis en el dominio de la frecuencia a los datos del acelerómetro y el giroscopio. Se determinó que las vibraciones de mayor amplitud eran de baja frecuencia, y se identificó una alta dispersión en los datos obtenidos. Se asumió que dicha dispersión era provocada por un fenómeno de aliasing, y se diseñaron e implementaron filtros de medias móviles de 40 ventanas y 2 ventanas para el acelerómetro y el giroscopio, respectivamente. Con la implementación de los filtros de medias móviles se encontró que la dispersión de los datos disminuyó tanto en el acelerómetro como en el giroscopio, y con ello, la desviación estándar de las estimaciones en presencia de vibraciones.

En la ilustración que sigue se puede evidenciar la reducción en la amplitud de la señal del giroscopio, y el efecto de filtrado sobre las señales de vibración de alta frecuencia del filtro de medias móviles:

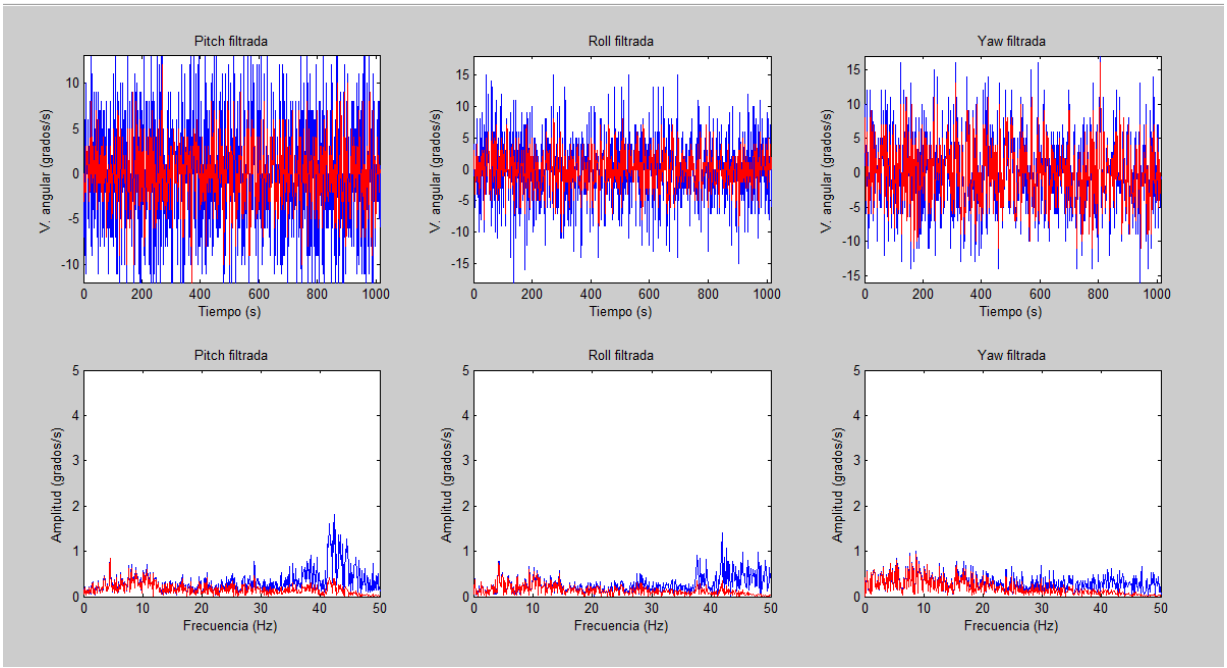

Ilustración 7. 16 Datos crudos y filtrados del giroscopio con motores encendidos y base de amortiguación.

Fuente: elaboración propia.

Igualmente, en la ilustración que se presenta a continuación se puede evidenciar la reducción en la amplitud de la señal del acelerómetro, y la precisión obtenida en la señal filtrada. Es importante resaltar que ante el fuerte filtrado de la señal se pierde la posibilidad de detectar de forma efectiva movimientos con el acelerómetro a una frecuencia mayor a 2Hz:

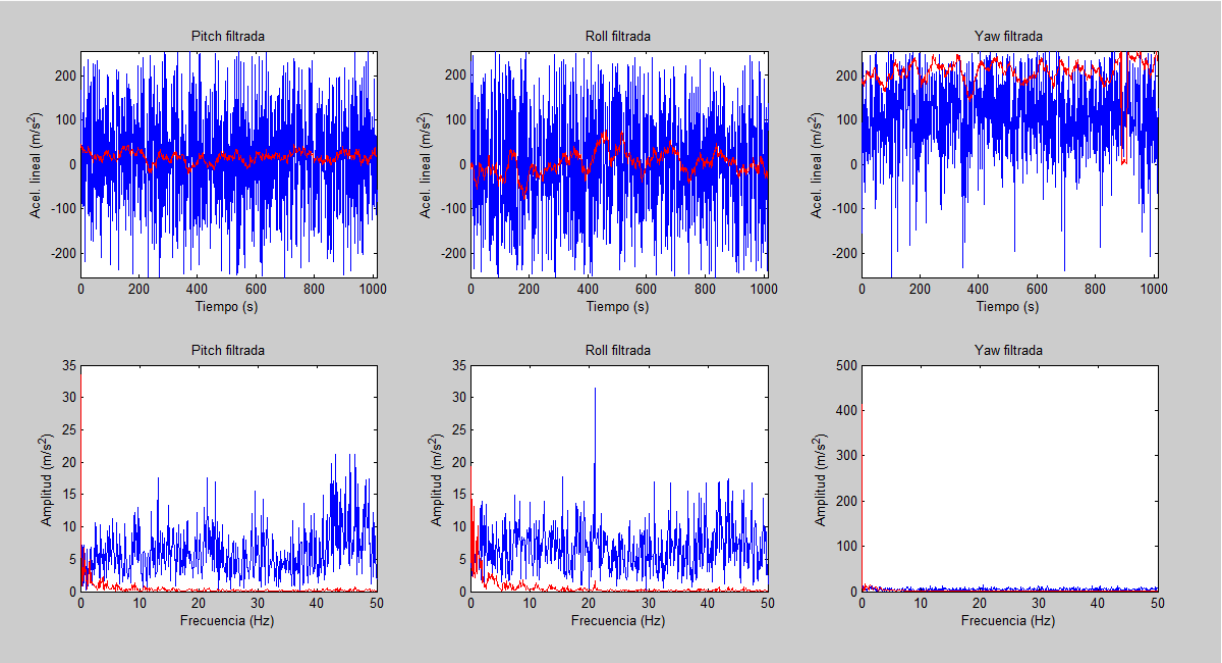

Ilustración 7. 17 Datos crudos y filtrados del acelerómetro con motores encendidos y base de amortiguación.

Fuente: elaboración propia.

Para mantener una frecuencia efectiva de muestreo suficientemente alta como para aplicar filtrado a las señales de los sensores, y obtener una estimación adecuada para realizar control de inclinación del cuadricóptero en tiempo real, se configuró la Unidad de Medición Inercial como se expone en la siguiente tabla:

Tabla 7. 1 Frecuencias de muestreo efectivas para sensores de la IMU

Sensor	Frecuencia de salida	Frecuencia corte filtro pasa-bajos integrado
Giroscopio L3GD20	190Hz	12,5Hz
Acelerómetro LSM303DLHC	1000Hz	72,5Hz

Fuente: elaboración propia

Apéndice C: Descripción del modelo de estimación de posición y velocidad lineal en el eje z del cuadricóptero

a. Obtención de datos y modelado del problema

Para la obtención de datos del sensor ultrasónico de distancia se utilizó la librería NewPing V1.3 para Arduino, la cual permite el manejo del sensor mediante la encapsulación del mismo como un objeto de la clase NewPing, y basa su funcionamiento en la emisión de un pulso del sensor, y la ejecución de una rutina para detección y medición del tiempo de retorno del pulso ultrasónico emitido. La librería NewPing ofrece una mayor precisión que el método de detección de pulsos utilizado por defecto en Arduino mediante la función pulseIn(), y no interfiere en la ejecución del resto del código, ya que el proceso de detección de pulsos se ejecuta como una rutina de interrupción sobre el temporizador timer2 de las tarjetas Arduino, a una frecuencia máxima de 34Hz.

La estimación de posición en el eje z a partir de las mediciones del sensor ultrasónico se realiza a partir del tiempo de retorno de los pulsos emitidos, sin considerar la inclinación del cuadricóptero respecto al suelo debido a la alta magnitud de los problemas de vibraciones mecánicas del mismo, que afectan el funcionamiento de los algoritmos de estimación angular. El cálculo de la distancia respecto al suelo del cuadricóptero se realizó siguiendo la siguiente ecuación:

$$Altura = V_{Sonido} * T_{Retorno}$$

donde:

V_{Sonido}: velocidad del sonido, constante, de valor 0,0343 cm/µs.

$T_{Retorno}$: tiempo de retorno del pulso, medido en microsegundos (µs).

La estimación de velocidad lineal en el eje Z del cuadricóptero se realiza mediante derivación numérica por el método de diferencias hacia atrás de los valores de estimación de posición en el eje Z, como se expone a continuación:

$$V_Z[k+1] = \frac{Altura[k] - Altura[k-1]}{dt}$$

donde:

$\mathrm{Altura[k]}$: k-ésima medición de altura.

dt: diferencial de tiempo, calculado en tiempo real.

b. Caracterización y respuesta en frecuencia del sensor de altura utilizado

A partir de la realización de pruebas emulando las condiciones teóricas de posicionamiento e inclinación con el cuadricóptero en vuelo, se pudo constatar la inestabilidad de las mediciones del sensor utilizado, las cuales poseían una alta cantidad de discontinuidades al ser este inclinado. Para solucionar dicho problema, y con ello realizar una estimación de altura más robusta y precisa se implementó un Filtro de Kalman de una variable, con una covarianza del error del proceso físico constante, no relacionándolo directamente al modelo físico del cuadricóptero. Los parámetros del Filtro de Kalman implementado se calcularon por prueba y error, aplicando como criterio la obtención de la mayor velocidad de respuesta y robustez posible en las estimaciones de altura.

Se recolectaron datos de estimación de posición y velocidad lineal en el eje z con los motores apagados, y sometiendo al cuadricóptero a una serie de movimientos simples de traslación y rotación en el aire. En la gráfica siguiente se puede observar la señal original de estimación de posición en el eje z, y la correspondiente señal filtrada, supurpuestas:

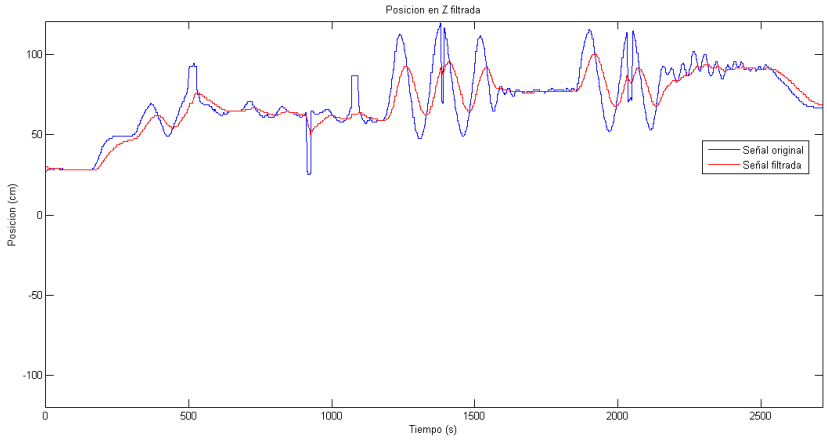

Ilustración 7. 18 Señal cruda y señal filtrada del sensor ultrasónico de distancia con los motores apagados y el cuadricóptero en movimiento.

Fuente: elaboración propia.

En la gráfica que se presenta a continuación se expone el espectro de

frecuencias de la señal anteriormente expuesta, pudiendo observarse la reducción del espectro de frecuencias de la señal filtrada:

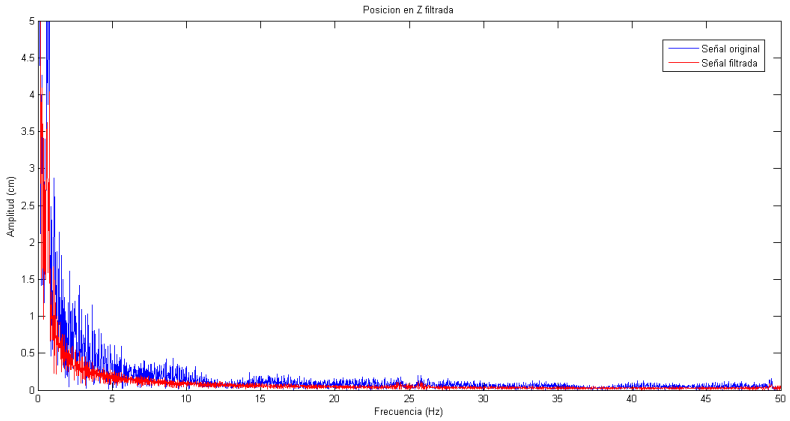

Ilustración 7. 19 Espectrograma de la señal cruda y de la señal filtrada del sensor ultrasónico de distancia con los motores apagados y el cuadricóptero en movimiento.

Fuente: elaboración propia.

Se realizó análisis en el dominio de la frecuencia a la señal de estimación de velocidad lineal en el eje z con los motores apagados, obteniéndose una señal con muy poco ruido, pero muy alto tiempo de respuesta, como se puede observar a continuación:

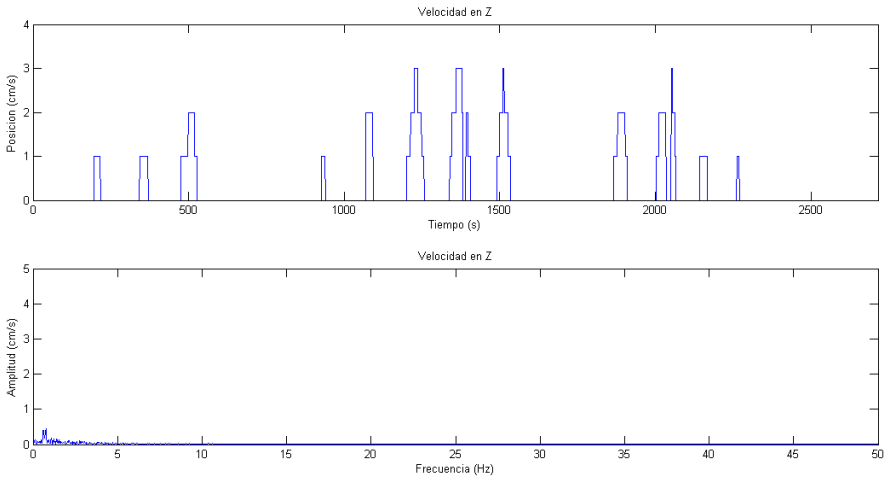

Ilustración 7. 20 Espectrograma de la señal cruda y de la señal filtrada del sensor ultrasónico de distancia con los motores apagados y el cuadricóptero en movimiento.

Fuente: elaboración propia.

Por problemas de diseño del circuito de lógica, sensores y comunicación no se pudieron realizar pruebas de funcionamiento de los algoritmos de estimación de altura y velocidad lineal en el eje z con los motores encendidos.

Apéndice D: Modelo dinámico del cuadricóptero

a. Modelo dinámico del cuadricóptero

Se utilizó una versión reducida del modelo físico del Draganflyer V desarrollado en [Kivrak 2006], por el alto nivel de detalle con el que describe la dinámica de rotación y traslación en el eje 'Z' del mismo. Las asunciones realizadas por el modelo son:

1) La estructura de fibra de carbono del cuadricóptero es rígida.
2) El cuadricóptero posee una estructura completamente simétrica, y no se presenta acoplamiento cruzado entre los ejes del mismo, por lo cual la matriz de inercia del cuadricóptero es una matriz diagonal.
3) El cuadricóptero se encuentra en condición de vuelo.
4) Los movimientos del cuadricóptero pueden ser modelados de forma efectiva respecto a dos ejes de coordenadas: uno centrado en el suelo, debajo del cuadricóptero, y que representa el marco de referencia inercial de la tierra; y otro ubicado en el centro de gravedad del cuadricóptero.
5) La relación (voltaje aplicado) – (fuerza de empuje ejercida por las hélices) de cada uno de los motores del cuadricóptero es lineal.
6) El aleteo de las hélices, el efecto suelo, la fricción del aire, y el retardo característico de los motores del cuadricóptero tienen un efecto despreciable sobre la dinámica de vuelo del mismo en espacios cerrados.

La matriz de estados del sistema viene dada por:

$$X = \begin{bmatrix} p \\ q \\ r \\ \phi \\ \theta \\ \psi \\ u \\ v \\ w \\ z \end{bmatrix}$$

Las ecuaciones que describen la dinámica del cuadricóptero son:

$$\dot{z} = -sen(\theta) * u + sen(\phi) * \cos(\theta) * v + \cos(\theta) * \cos(\phi) * w \quad \textbf{(11)}$$

$$\dot{u} = \frac{1}{m} * F_x + g * (\cos(\psi) * sen(\theta) * \cos(\phi) + sen(\psi) * sen(\phi)) - q * w + r * v \quad \textbf{(12)}$$

$$\dot{v} = \frac{1}{m} * F_y + g * (\text{sen}(\psi) * sen(\theta) * \cos(\phi) - cos(\psi) * sen(\phi)) - r * u + p * w \quad \textbf{(13)}$$

$$\dot{w} = \frac{1}{m} * F_z + g * cos(\theta) * \cos(\phi) - p * v + q * u \quad \textbf{(14)}$$

$$\dot{p} = \frac{M_x}{I_{xx}} + \frac{(I_{yy} * q * r)}{I_{zz}} - \frac{(I_{zz} * q * r)}{I_{yy}} \quad \textbf{(15)}$$

$$\dot{q} = \frac{M_y}{I_{yy}} - \frac{(I_{xx} * p * r)}{I_{zz}} + \frac{(I_{zz} * p * r)}{I_{xx}} \quad \textbf{(16)}$$

$$\dot{r} = \frac{M_z}{I_{zz}} + \frac{(I_{xx} * p * q)}{I_{yy}} - \frac{(I_{yy} * p * q)}{I_{xx}} \quad \textbf{(17)}$$

$$\dot{\phi} = p + q * \tan(\theta) * \dot{sen(\phi)} + r * \tan(\theta) * \cos(\phi) \quad \textbf{(18)}$$

$$\dot{\theta} = q * \dot{cos(\phi)} - r * \text{sen}(\phi) \quad \textbf{(19)}$$

$$\dot{\psi} = q * \sec(\theta) * \dot{sen(\phi)} + r * \sec(\theta) * \cos(\phi) \quad \textbf{(20)}$$

$donde$:

- ϕ: á$ngulo\ de\ Pitch.$
- θ: á$ngulo\ de\ Roll.$
- ψ: á$ngulo\ de\ Yaw.$
- $\dot{\phi}$: $velocidad\ angular\ de\ Pitch.$
- $\dot{\theta}$: $velocidad\ angular\ de\ Roll.$
- $\dot{\psi}$: $velocidad\ angular\ de\ Yaw.$
- p: $velocidad\ angular\ de\ giro\ alrededor\ del\ eje\ x\ del\ cuadricóptero.$
- q: $velocidad\ angular\ de\ giro\ alrededor\ del\ eje\ y\ del\ cuadricóptero.$
- r: $velocidad\ angular\ de\ giro\ alrededor\ del\ eje\ z\ del\ cuadricóptero.$
- $\dot{p}$: $velocidad\ angular\ de\ giro\ alrededor\ del\ eje\ x\ del\ cuadricóptero.$
- $\dot{q}$: $velocidad\ angular\ de\ giro\ alrededor\ del\ eje\ y\ del\ cuadricóptero.$

- $\dot{r}$: $velocidad\ angular\ de\ giro\ alrededor\ del\ eje\ z\ del\ cuadricóptero.$
- u: $velocidad\ lineal\ en\ el\ eje\ x\ del\ cuadricóptero.$
- v: $velocidad\ lineal\ en\ el\ eje\ y\ del\ cuadricóptero.$
- w: $velocidad\ lineal\ en\ el\ eje\ z\ del\ cuadricóptero.$
- $\dot{u}$: $aceleración\ lineal\ en\ el\ eje\ x\ del\ cuadricóptero.$
- $\dot{v}$: $aceleración\ lineal\ en\ el\ eje\ y\ del\ cuadricóptero.$
- $\dot{w}$: $aceleración\ lineal\ en\ el\ eje\ z\ del\ cuadricóptero.$
- z: $posición\ en\ el\ eje\ z\ del\ marco\ de\ referencia\ inercial\ de\ la\ tierra.$
- $\dot{z}$: $velocidad\ lineal\ en\ el\ eje\ z\ del\ marco\ de\ referencia\ inercial\ de\ la\ tierra.$
- g: $constante\ de\ aceleración\ por\ efecto\ de\ la\ gravedad\ del\ planeta\ Tierra.$
- F_x: $fuerza\ actuando\ sobre\ el\ cuadricóptero\ en\ el\ eje\ x\ del\ marco\ de\ referencia\ inercial\ de\ la\ tierra.$
- F_y: $fuerza\ actuando\ sobre\ el\ cuadricóptero\ en\ el\ eje\ y\ del\ marco\ de\ referencia\ inercial\ de\ la\ tierra.$
- F_z: $fuerza\ actuando\ sobre\ el\ cuadricóptero\ en\ el\ eje\ z\ del\ marco\ de\ referencia\ inercial\ de\ la\ tierra.$
- M_x: $momento\ de\ fuerza\ actuando\ sobre\ el\ cuadricóptero\ en\ el\ eje\ x\ del\ marco\ de\ referencia\ inercial\ de\ la\ tierra.$
- M_y: $momento\ de\ fuerza\ actuando\ sobre\ el\ cuadricóptero\ en\ el\ eje\ y\ del\ marco\ de\ referencia\ inercial\ de\ la\ tierra.$
- M_z: $momento\ de\ fuerza\ actuando\ sobre\ el\ cuadricóptero\ en\ el\ eje\ z\ del\ marco\ de\ referencia\ inercial\ de\ la\ tierra.$
- m: $masa\ total\ del\ cuadricóptero.$
- I_{xx}: $inercia\ del\ cuadricóptero\ en\ el\ eje\ x.$
- I_{yy}: $inercia\ del\ cuadricóptero\ en\ el\ eje\ y.$
- I_{zz}: $inercia\ del\ cuadricóptero\ en\ el\ eje\ z.$

Asi mismo, se utilizó el modelo lineal de la relación (voltaje aplicado) - (fuerza de empuje ejercida por las hélices) de cada uno de los motores del cuadricóptero Draganflyer

V, desarrollado en [Kivrak 2006]. El modelo en cuestión es descrito mediante las siguientes ecuaciones:

$$F_1 = 9{,}81 * \frac{(22{,}4935 * V_{m1} - 9{,}732)}{1000} \quad \textbf{(21)}$$

$$F_2 = 9{,}81 * \frac{(22{,}4926 * V_{m1} - 9{,}5271)}{1000} \quad \textbf{(22)}$$

$$F_3 = 9{,}81 * \frac{(22{,}6127 * V_{m1} - 9{,}8941)}{1000} \quad \textbf{(23)}$$

$$F_4 = 9{,}81 * \frac{(22{,}6184 * V_{m1} - 9{,}5805)}{1000} \quad \textbf{(24)}$$

$donde$:

- F_1: $fuerza\ ejercida\ por\ el\ motor\ 1\ del\ cuadricóptero.$
- F_2: $fuerza\ ejercida\ por\ el\ motor\ 1\ del\ cuadricóptero.$
- F_3: $fuerza\ ejercida\ por\ el\ motor\ 1\ del\ cuadricóptero.$
- F_4: $fuerza\ ejercida\ por\ el\ motor\ 1\ del\ cuadricóptero.$

Finalmente, los momentos de fuerza en los ejes x, y, z del cuadricóptero vienen dados por:

$$M_x = L * (F_2 - F_4) \quad \textbf{(25)}$$

$$M_y = L * (F_1 - F_3) \quad \textbf{(26)}$$

$$M_z = c * -(F_1 + F_2 + F_3 + F_4) \quad \textbf{(27)}$$

El convenio de numeración de variables y la fuerza ejercida por cada motor, junto a las variables de estado del cuadricóptero se expone en el gráfico que se presenta a continuación:

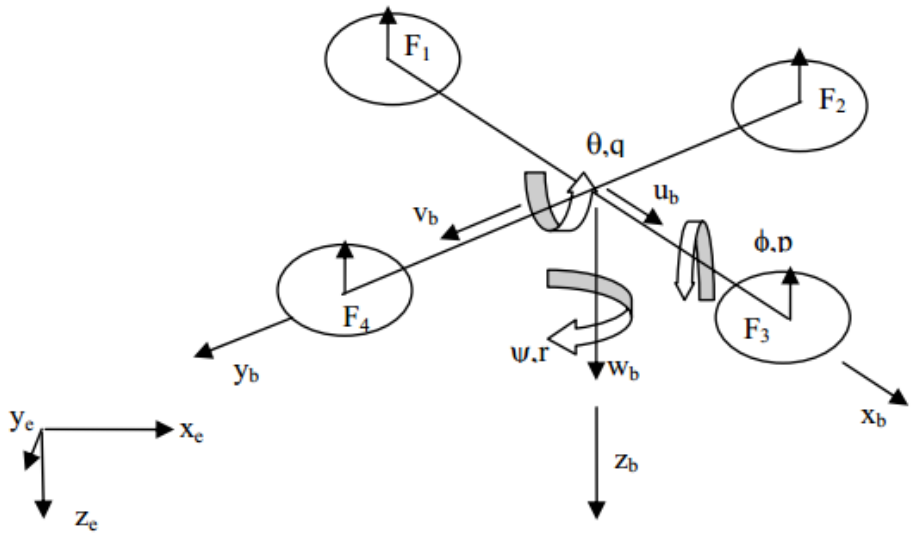

Ilustración 7. 21 Diagrama simplificado del modelo dinámico del cuadricóptero Draganflyer V. Fuente: [Kivrak 2006]

b. Cálculo de los parámetros de inercia del modelo

Con el objetivo de simplificar los cálculos solo se toman como referencia la inercia generada por la masa de los motores y la cabina. Los objetos se modelaron como cuerpos rigidos de densidad uniforme.

Para realizar el cálculo de los parámetros de inercia se realizaron las siguientes medidas:

Tabla 7. 2 Parametros para cálculo de matriz de inercia

Medida:	Valor
Masa de motor	0.051Kg
Radio de motor	0.0125 m
Altura Motor	0.095 m
Longitud de eje	0.47 m
Lado cabina	0.09 m
Altura de cabina	0.1 m
Masa de motor	0.051 Kg
Masa de cabina	071Kg

Fuente: elaboración propia

Se siguió el método desarrollado en [Sinha 2012] para el cálulo de la inercia de los motores, y las ecuaciones de [Hibbeler 2012] para el cálculo de inercia de la

cabina.

A continuación se presenta el cálculo de los parámetros de inercia para los ejes de Pitch y Roll:

Motores sobre el eje:

$$I_{x1} = I_{x3} = I_{y2} = I_{y4} = \frac{1}{12} m_m \left(3 * (R_m)^2 + {A_m}^2\right) \quad \textbf{(28)}$$

$$I_{x1} = I_{x3} = I_{y2} = I_{y4} = \ 4.4333e-05 \mathrm{Kg} * \mathrm{m}^2 \quad \textbf{(29)}$$

Motores fuera del eje:

$$I_{x2} = I_{x4} = I_{y1} = I_{y3} = \frac{1}{12} m_m \left(3 * (R_m)^2 + {A_m}^2\right) + D_x * M_m \quad \textbf{(30)}$$

$$I_{x2} = I_{x4} = I_{y1} = I_{y3} = 0.0120 \mathrm{Kg} * \mathrm{m}^2 \quad \textbf{(31)}$$

Cabina:

$$I_{cx} = I_{cy} = \frac{1}{12}\ M_c * ({L_c}^2 + {A_c}^2) \quad \textbf{(32)}$$

$$I_{cx} = I_{cy} = \ 0.0011 \mathrm{Kg} * \mathrm{m}^2 \quad \textbf{(33)}$$

Inercia total:

$$I_{xx} = I_{yy} = \ I_{x1} + I_{x3} + I_{x2} + I_{x4} + I_{cx} \quad \textbf{(34)}$$

$$= I_{y1} + I_{y3} + I_{y2} + I_{y4} + I_{cy}$$

$$I_{xx} = I_{yy} = \ 0.0252 \mathrm{Kg} * \mathrm{m}^2 \quad \textbf{(35)}$$

Asi mismo, se presenta el cálculo de los parámetros de inercia para el eje de Yaw, basados en la suposición de que la inercia que ejerce cada motor sobre el eje Z es igual a cero:

Inercia de motores:

$$I_{z1} = I_{z2} = I_{z3} = I_{z4} = \ \frac{1}{12} m_m \left(3 * (R_m)^2 + {A_m}^2\right) + \ D_x * M_m \quad \textbf{(36)}$$

$$I_{z1} = I_{z2} = I_{z3} = I_{z4} = 0.0240 \mathrm{Kg} * \mathrm{m}^2 \quad \textbf{(37)}$$

Inercia de cabina:

$$I_{cz} = \frac{1}{12}\, M_c * ({L_c}^2 + {L_c}^2) \quad \textbf{(38)}$$

$$I_{cz} = 9.7132\text{e} - 04\text{Kg} * \text{m}^2 \quad \textbf{(39)}$$

Inercia total:

$$I_{zz} = I_{cy} + I_{z1} + I_{z2} + I_{z3} + I_{z4} \quad \textbf{(40)}$$

$$I_{zz} = 0.0970\text{Kg} * \text{m}^2 \quad \textbf{(41)}$$

donde:

I_{xx}: $inercia\ en\ el\ eje\ de\ Pitch.$

I_{yy}: $inercia\ en\ el\ eje\ de\ Roll.$

I_{zz}: $inercia\ en\ el\ eje\ de\ Yaw.$

I_{x1}: $inercia\ del\ motor\ delantero\ en\ el\ eje\ de\ Pitch.$

I_{x2}: $inercia\ del\ motor\ derecho\ en\ el\ eje\ de\ Pitch.$

I_{x3}: $inercia\ del\ motor\ izquierdo\ en\ el\ eje\ de\ Pitch.$

I_{x4}: $inercia\ del\ motor\ izquierdo\ en\ el\ eje\ de\ Pitch.$

I_{y1}: $inercia\ del\ motor\ delantero\ en\ el\ eje\ de\ Roll.$

I_{y2}: $inercia\ del\ motor\ derecho\ en\ el\ eje\ de\ Roll.$

I_{y3}: $inercia\ del\ motor\ trasero\ en\ el\ eje\ de\ Roll.$

I_{y4}: $inercia\ del\ motor\ izquierdo\ en\ el\ eje\ de\ Roll.$

I_p: $inercia\ de\ la\ cabina\ en\ el\ eje\ de\ Pitch.$

I_p: $inercia\ de\ la\ cabina\ en\ el\ eje\ de\ Pitch.$

I_p: $inercia\ de\ la\ cabina\ en\ en\ el\ eje\ de\ Pitch.$

D_x: $distancia\ entre\ el\ centro\ de\ masas\ del\ cuadricóptero\ y\ cada\ motor.$

L_c: $longitud\ de\ cada\ lado\ de\ la\ cabina.$

A_c: $altura\ de\ la\ cabina.$

R_m: $radio\ de\ los\ motores.$

A_m: $altura\ de\ los\ motores.$

Como resultado de estos calculos se obtiene la siguiente matriz de inercia:

$$M_{inercia} = \begin{bmatrix} 0.0252 & 0 & 0 \\ 0 & 0.0252 & 0 \\ 0 & 0 & 0.0970 \end{bmatrix}$$

c. Linealización del modelo dinámico

Partiendo de la asunción de que las fuerzas externas en los ejes X e Y del cuadricóptero son despreciables, se puede representar la dinámica del cuadricóptero como la función vectorial f(X,U), la cual integra las ecuaciones del modelo físico del mismo; y las entradas de control del sistema por el vector U. Luego:

$$f(X,U) = \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \dot{u} \\ \dot{v} \\ \dot{w} \\ \dot{z} \end{bmatrix} \quad \textbf{(42)}$$

$$f(X,U)=\begin{bmatrix} \frac{M_x}{I_{xx}}+\frac{(I_{yy}*q*r)}{I_{zz}}-\frac{(I_{zz}*q*r)}{I_{yy}} \\ \frac{M_y}{I_{yy}}-\frac{(I_{xx}*p*r)}{I_{zz}}+\frac{(I_{zz}*p*r)}{I_{xx}} \\ \frac{M_z}{I_{zz}}+\frac{(I_{xx}*p*q)}{I_{yy}}-\frac{(I_{yy}*p*q)}{I_{xx}} \\ p+q*\tan(\theta)*sen(\phi)+\ r*\tan(\theta)*\cos(\phi) \\ q*cos(\phi)-\ r*\text{sen}\ (\phi) \\ q*\sec(\theta)*sen(\phi)+\ r*\sec\ (\theta)*\cos\ (\phi) \\ g*(\cos(\psi)*sen(\theta)*\cos(\phi)+\ sen(\psi)*sen(\phi))-q*w+r*v \\ g*(\text{sen}(\psi)*sen(\theta)*\cos(\phi)-\ cos(\psi)*sen(\phi))-r*u+p*w \\ \frac{1}{m}*F_z+g*cos(\theta)*\cos(\phi)-p*v+q*u \\ -sen(\theta)*u+sen(\phi)*\cos(\theta)*v+\cos(\theta)*\cos(\phi)*w \end{bmatrix} \quad \textbf{(43)}$$

$$U=\begin{bmatrix} V_{m1} \\ V_{m2} \\ V_{m3} \\ V_{m4} \end{bmatrix} \quad \textbf{(44)}$$

Se llevó al cabo el cálculo de los parámetros de inercia del cuadricóptero, como es expuesto en el Apéndice A, y posteriormente, se linealizó el modelo dinámico descrito por el vector f, mediante el cálculo de su matriz Jacobiana respecto al vector de estados X y al vector de entradas de control U. Las matrices obtenidas se evaluaron en un punto de equilibrio X_0, para obtener una representación del sistema de la forma $\dot{X}=A*X+\ B*U$, donde A es una matriz que describe el comportamiento del sistema alrededor de un punto de equilibrio X_0, y B es una matriz que describe el comportamiento del sistema ante señales de control.

$$A=J_f(X)|_{X_0}=\begin{bmatrix} \frac{\partial f_1}{\partial X_1} & \cdots & \frac{\partial f_1}{\partial X_{10}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{10}}{\partial X_1} & \cdots & \frac{\partial f_{10}}{\partial X_{10}} \end{bmatrix}_{X_0} \quad \textbf{(45)}$$

$$B = J_f(U)|_{X_0} = \begin{bmatrix} \frac{\partial f_1}{\partial U_1} & \cdots & \frac{\partial f_1}{\partial U_4} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{10}}{\partial U_1} & \cdots & \frac{\partial f_{10}}{\partial X_4} \end{bmatrix}_{X_0} \quad \textbf{(46)}$$

Se seleccionó el vector $\mathrm{X}_0 = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1]^{\mathrm{T}}$, para asegurar que el modelo lineal represente el comportamiento del cuadricóptero en vuelo, con angulos, velocidades angulares, y velocidades lineales iguales a cero (0), y una altura inicial de un (1) metro, cercana a la altura máxima a alcanzar por el cuadricóptero desarrollado en el presente Trabajo Especial de Grado. Las matrices A y B calculadas se exponen a continuación:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 9{,}81 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -9{,}81 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad \textbf{(47)}$$

$$B = \begin{bmatrix} 0 & 3{,}6516 & 0 & -3{,}6721 \\ 3{,}6518 & 0 & -3{,}6711 & 0 \\ -3{,}1079 & 3{,}1078 & -3{,}1244 & 3{,}1252 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -0{,}3678 & -0{,}3678 & -0{,}3697 & -0{,}3698 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \textbf{(48)}$$

Se verificó la controlabilidad del sistema con base en el rango de la matriz de controlabilidad C, la cual puede hallarse a partir de las matrices A y B, como se expone a continuación:

$$\mathrm{C} = [B : AB : A^2B : \ldots : A^{n-1}B] \quad \textbf{(49)}$$

Se obtuvo que el rango de la matriz C es igual a diez (10), el número de estados del sistema, por lo cual todos los estados del sistema son controlables.

Se modeló la estimación de estado del cuadricóptero como un sistema de ecuaciones diferenciales lineales de la forma $\mathrm{Y} = \mathrm{C} * \mathrm{X}$, donde Y es la observación sobre el estado del cuadricóptero realizada a partir de los sensores disponibles en el mismo, X es el estado del cuadricóptero, y C es la matriz de salida del cuadricóptero, que representa el arreglo de sensores disponibles. Los sensores existentes en el cuadricóptero permiten la estimación de las velocidades y posiciones angulares, y la velocidad lineal y posición en el eje Z, a partir de lo cual se modeló la matriz C como se presenta a continuación:

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(50)}$$

Para todo X, la estimación de estado del cuadricóptero realizada a partir de los sensores del mismo viene dada por:

$$Y = \begin{bmatrix} p \\ q \\ r \\ \phi \\ \theta \\ \psi \\ 0 \\ 0 \\ w \\ z \end{bmatrix} \quad \textbf{(51)}$$

Se verificó la observabilidad del sistema cpn base en el rango de la matriz de observabilidad O, la cual puede hallarse a partir de las matrices A y B, como se expone a continuación:

$$O = [C : CA : CA^2 : \ldots : CA^{n-1}] \quad \textbf{(52)}$$

Se obtuvo que el rango de la matriz O es igual a ocho (8), siendo diez (10) el número de estados del sistema, por lo cual dos estados del cuadricóptero no podrán ser estimados a partir del arreglo de sensores disponible. En particular, dichos estados son las velocidades lineales en los ejes x e y.

Finalmente, se evaluaron los autovalores de la matriz de transición de estados del sistema A, y se obtuvo que todos tienen valor nulo, por lo cual el sistema es críticamente estable alrededor del punto de equilibrio en el que fue linealizada la ecuación, y es necesaria la implementación de un mecanismo de regulación para asegurar la estabilidad del sistema.

Apéndice E: Código de estimación, control y comunicación a bordo del cuadricóptero

Archivo CuadricopteroUCAB_VelocidadAngular.ino (Principal):

```
#include <FiltroLP_Giroscopio.h>
#include <FiltroLP_Acelerometro.h>
#include <FiltroMediaMovil_Acelerometro.h>
#include <FiltroMediaMovil_Giroscopio.h>
#include <Wire.h>
#include <L3G.h>
#include <LSM303.h>
#include <PID_v1.h>
#include <NewPing.h>


/*********************************************        CONSTANTES y VARIABLES
*********************************************/
//MOTORES:
#define PUERTOMOTORDERECHO 5 //puerto de PWM del motor derecho
#define PUERTOMOTORIZQUIERDO 9 //puerto de PWM del motor izquierdo
#define PUERTOMOTORINFERIOR 10 //puerto de PWM del motor inferior
#define PUERTOMOTORSUPERIOR 11 //puerto de PWM del motor superior
#define PWM_MAXIMO 240 //maximo PWM que puede enviar el arduino a los motores
int velocidadBasePWM = 120;
char modoEjecucion = '_';
int motorDerecho = 0;
int motorIzquierdo = 0;
int motorDelantero = 0;
int motorTrasero = 0;
//FIN MOTORES


//CODIGOS DE COMUNICACION:
#define modoTelemetriaTotal 0
#define DT envioDatosTelemetriaTotal 5
#define DT envioDatosEstado 30
#define LED_ENCENDIDO 13
#define CODIGO_INICIO_MENSAJE 255
#define CODIGO_ENCENDIDO 0
#define CODIGO_MOVIMIENTO 1
#define CODIGO_ACK 6
#define CODIGO_ESTADO 7
#define CODIGO_TELEMETRIA_TOTAL 8
#define MAXIMO_ANGULO_COMANDO 70
unsigned char headerMensaje;
unsigned char codigoRecibido;
unsigned char comandoEncendidoRecibido;
unsigned char comandoPitch;
unsigned char comandoRoll;
unsigned char comandoAltura;
unsigned char checksum;
unsigned char mensajeEstado[15];
unsigned char mensajeTelemetriaTotal[42];
unsigned char ack[4];
long tiempoUltimoEnvio = 0;
//FIN CODIGOS DE COMUNICACION


//ULTRASONIDO:
#define USPIN 15 //puerto de datos del ultradonido.
#define ALTURA_MAXIMA 150
#define INCREMENTO_ALTURA_COMANDO 5 ////////////////////////////// CAMBIAR A 5????
NewPing sonar(USPIN, USPIN, ALTURA_MAXIMA);
long tiempoUltimoMuestreoAltura = 0;
unsigned int uS;
double alturaDeseada = 0;
double distancia = 0;
double USAltura = 0; // Distancia medida por el sensor de ultrasonido
double correccionAltura = 0;
double covarianzaProcesoFisicoAltura = 0.1;
double covarianzaRuidoSensorAltura = 10.0;
double estimacionAltura = 0.0;
double alturaSuelo = 0.0;
```

```
double covarianzaRuidoEstimacionAltura = 3.0;
double gananciaKalman = 0.0;
double U_velocidad_Z = 0.0;
double U_Z_previo = 0.0;
double Az_sinG = 0.0;
double A_velocidad_Z = 0.0;
double velocidad_Z = 0.0;
double velocidadDeseadaZ = 0.0;
//FIN ULTRASONIDO


//IMU
#define ToRad(x) ((x)*0.01745329252)  // *pi/180
#define ToDeg(x) ((x)*57.2957795131)  // *180/pi
#define G_GYRO 0.00875
#define G_ACC 0.015874
#define K_COMP 0.97
#define DT_sensor_altura 29
#define DT_acelerometro 1
#define DT giroscopio 6
L3G gyro;
LSM303 compass;
int numMuestras;
double G_offsetYPR [3] = {
  0, 0, 0
};
double A_offsetYPR [3] = {
  14946, -355, -1957
};
double anguloDeseadoYPR[3] = {
  0, 0, 0
};
double G_velocidadYPR_filtrada[3] = {
  0, 0, 0
};
double G_velocidadYPR[3] = {
  0, 0, 0
};
double G_anguloYPR[3] = {
  0, 0, 0
};
double A_aceleracionYPR[3] = {
  0, 0, 0
};
double A_aceleracionYPR_filtrada[3] = {
  0, 0, 0
};
double A_anguloYPR[3] = {
  0, 0, 0
};
double A_anguloYPR_filtrado[3] = {
  0, 0, 0
};
double anguloYPR_filtrado[3] = {
  0, 0, 0
};
double velocidadDeseadaYPR[3] = {
  0, 0, 0
};
double correccionPWM_YPR[3] = {
  0, 0, 0
};
FiltroMediaMovil_Giroscopio filtroVelocidadYPR [3];
FiltroMediaMovil_Acelerometro filtroAceleracionYPR [3];
double DT = 0;
long tiempoUltimoMuestreoGiroscopio = 0;
long tiempoUltimoMuestreoAcelerometro = 0;
long tiempoUltimoMuestreoAngulos = 0;
//FIN IMU


//SISTEMAS DE CONTROL PID
#define DT_PID_velZ 20
#define DT_PID_posZ 50
#define DT_PID_posicionAngular 20
#define DT_PID_velocidadAngular 6
PID PID_vAngular_Yaw(&G_velocidadYPR[0], &correccionPWM_YPR[0], &velocidadDeseadaYPR[0], 0, 0,
0, DIRECT);
```

```
PID PID_vAngular_Pitch(&G_velocidadYPR_filtrada[1], &correccionPWM_YPR[1],
&velocidadDeseadaYPR[1], 0, 0, 0, DIRECT);
PID PID_vAngular_Roll(&G_velocidadYPR_filtrada[2], &correccionPWM_YPR[2],
&velocidadDeseadaYPR[2], 0, 0, 0, DIRECT);
PID PID_pAngular_Yaw(&anguloYPR_filtrado[0], &velocidadDeseadaYPR[0], &anguloDeseadoYPR[0], 0,
0, 0, DIRECT);
PID PID_pAngular_Pitch(&anguloYPR_filtrado[1], &velocidadDeseadaYPR[1], &anguloDeseadoYPR[1],
0, 0, 0, DIRECT);
PID PID_pAngular_Roll(&anguloYPR_filtrado[2], &velocidadDeseadaYPR[2], &anguloDeseadoYPR[2],
0, 0, 0, REVERSE);
PID PID_posZ(&estimacionAltura, &velocidadDeseadaZ, &alturaDeseada, 0, 0, 0, DIRECT);
PID PID_velZ(&velocidad_Z, &correccionAltura, &velocidadDeseadaZ, 0, 0, 0, DIRECT);
byte i;
//FIN PID
/*******************************************        FIN DE CONSTANTES y VARIABLES
*******************************************/


void setup() {
  // Modo FAST PWM en los pines 9, 10 y 11 //
  TCCR2A = _BV(COM2A1) | _BV(COM2B1) | _BV(WGM21) | _BV(WGM20);
  TCCR2B = _BV(CS22);

  TCCR1A = _BV(COM1A1) | _BV(COM1B1) | _BV(WGM10);
  TCCR1B = _BV(CS11) | _BV(CS10) | _BV(WGM12);
  ////////////////////////////////////////////

  // Configuracion de los puertos para sensores, motores y bandera de encendido//
  pinMode(LED_ENCENDIDO, OUTPUT);
  pinMode(PUERTOMOTORDERECHO, OUTPUT);
  pinMode(PUERTOMOTORIZQUIERDO, OUTPUT);
  pinMode(PUERTOMOTORSUPERIOR, OUTPUT);
  pinMode(PUERTOMOTORINFERIOR, OUTPUT);
  analogWrite(PUERTOMOTORDERECHO, 0);
  analogWrite(PUERTOMOTORIZQUIERDO, 0);
  analogWrite(PUERTOMOTORSUPERIOR, 0);
  analogWrite(PUERTOMOTORINFERIOR, 0);
  ////////////////////////////////////////////////////////////

  // Inicializacion de la comunicacion Serial, I2C y acelerometro/giroscopio //
  if (modoTelemetriaTotal == 1)
  {
    Serial.begin(115200);
  }
  else
  {
    Serial.begin(38400);
  }
  Wire.begin();
  if (!gyro.init())
  {
    Serial.println("Failed to autodetect gyro type!");
    while (1);
  }
  gyro.enableDefault();
  compass.init();
  compass.enableDefault();
  CalcularOffsetGiroscopio();
//  CalcularOffsetAcelerometro();
  ////////////////////////////////////////////////////////////////////////////////

  // Parametros de los Algoritmos PID //
  PID_pAngular_Yaw.SetSampleTime(DT_PID_posicionAngular);
  PID_pAngular_Pitch.SetSampleTime(DT_PID_posicionAngular);
  PID_pAngular_Roll.SetSampleTime(DT_PID_posicionAngular);
  PID_vAngular_Yaw.SetSampleTime(DT_PID_velocidadAngular);
  PID_vAngular_Pitch.SetSampleTime(DT_PID_velocidadAngular);
  PID_vAngular_Roll.SetSampleTime(DT_PID_velocidadAngular);
  PID_posZ.SetSampleTime(DT_PID_posZ);
  PID_posZ.SetSampleTime(DT_PID_velZ);
  PID_pAngular_Yaw.SetOutputLimits(-180.0, 180.0);
  PID_pAngular_Pitch.SetOutputLimits(-180.0, 180.0);
  PID_pAngular_Roll.SetOutputLimits(-180.0, 180.0);
  PID_vAngular_Yaw.SetOutputLimits(-PWM_MAXIMO, PWM_MAXIMO);
  PID_vAngular_Pitch.SetOutputLimits(-PWM_MAXIMO, PWM_MAXIMO);
  PID_vAngular_Roll.SetOutputLimits(-PWM_MAXIMO, PWM_MAXIMO);
  PID_posZ.SetOutputLimits(-50, 50);
  PID_velZ.SetOutputLimits(-PWM_MAXIMO, PWM_MAXIMO);
```

```
  PID_pAngular_Yaw.SetMode(AUTOMATIC);
  PID_pAngular_Pitch.SetMode(AUTOMATIC);
  PID_pAngular_Roll.SetMode(AUTOMATIC);
  PID_vAngular_Yaw.SetMode(AUTOMATIC);
  PID_vAngular_Pitch.SetMode(AUTOMATIC);
  PID_vAngular_Roll.SetMode(AUTOMATIC);
  PID_posZ.SetMode(AUTOMATIC);
  PID_velZ.SetMode(AUTOMATIC);
  PID_pAngular_Yaw.SetTunings(1, 0.01, 0);
  PID_pAngular_Pitch.SetTunings(0, 0, 0);
  PID_pAngular_Roll.SetTunings(0, 0, 0);
  PID_vAngular_Yaw.SetTunings(0.4, 0, 0);
  PID_vAngular_Pitch.SetTunings(0.95, 0.01, 0.005);
  PID_vAngular_Roll.SetTunings(0.95, 0.1, 0.005);
  PID_posZ.SetTunings(0, 0, 0);
  PID_velZ.SetTunings(0, 0, 0);
  ////////////////////////////////////

  // Inicio de conteo para manejo de frecuencia de envio de datos y DT de muestreo //
  tiempoUltimoMuestreoAngulos = micros();
  tiempoUltimoMuestreoAltura = millis();
  tiempoUltimoEnvio = millis();
  /////////////////////////////////////////////////////////////////////////////////
}



////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////
//////////////////////////////////////////      PROCEDIMIENTO PRINCIPAL
//////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////
/* El procedimiento loop se ejecuta de forma constante e ininterrumpida, y realiza llamados a
las rutinas de
   recepcion de comandos, secuencia de inicio, y secuencia de vuelo.
*/
void loop()
{
  modoEjecucion = '_';
  RecibirComando();
  SecuenciaDeInicio();
  SecuenciaDeVuelo();
}



////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////
//////////////////////////////////////////      PROCEDIMIENTO PARA LLEVAR A CABO ESTIMACION DE
ESTADO, TELEMETRIA  E INICIO PAULATINO DE LOS MOTORES DEL CUADRICOPTERO
//////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////
/*
  Se realiza muestreo de los sensores en espera de recibir un comando de encendido de motores
desde la PC. En cuanto se encienden los motores se ejecuta una secuencia de encendido
  gradual de los motores.
*/
void SecuenciaDeInicio()
{
  //Ciclo para realizar muestreo y telemetria del cuadricoptero de forma constante, en espera
de comandos de encendido de motores desde la PC.
  i = 0;
  while (i < 50)
  {
    FiltroComplementario();
    CalcularAltura();
    EnviarMensajesTelemetriaPC();
    RecibirComando();
    anguloDeseadoYPR[1] = 0;
    anguloDeseadoYPR[2] = 0;
    alturaDeseada = 0;
    i++;
  }
```

```
  //Ciclo para iniciar lentamente los motores, hasta la mitad de la velocidad base de PWM por
defecto, con un retardo de 1ms por cada escalon de velocidad.
  //Se ejecuta si y solo si ha sido enviado un mensaje de encendido de motores desde la PC.
(modoEjecucion == 'T')
  i = 0;
  if (modoEjecucion != '_')
  {
    while (i < velocidadBasePWM / 2)
    {
      if (modoEjecucion == '_')
      {
        motorDerecho = 0;
        motorIzquierdo = 0;
        motorDelantero = 0;
        motorTrasero = 0;
      }
      if (modoEjecucion == 'T')
      {
        motorDerecho = i;
        motorIzquierdo = i;
        motorDelantero = i;
        motorTrasero = i;
        PIDAltura();
        PID_VelocidadAngular();
        AplicarPWMmotores(i);
      }
      FiltroComplementario();
      CalcularAltura();
      PID_PosicionAngular();
      PID_VelocidadAngular();
      AplicarPWMmotores(velocidadBasePWM);
      EnviarMensajesTelemetriaPC();
      RecibirComando();
      i++;
      delay(1);
    }
  }
  else
  {
    analogWrite(PUERTOMOTORDERECHO, 0);
    analogWrite(PUERTOMOTORIZQUIERDO, 0);
    analogWrite(PUERTOMOTORSUPERIOR, 0);
    analogWrite(PUERTOMOTORINFERIOR, 0);
  }
}

////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
////////////////
//////////////////////////////////////////      PROCEDIMIENTO PARA LLEVAR A CABO ESTIMACION DE
ESTADO, CONTROL Y TELEMETRIA DEL CUADRICOPTERO EN VUELO
////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
////////////////
/* Ciclo que se ejecuta mientras no se envie comando de apagado de los motores (mientras
modoEjecucion sea distinto de '_').
   Secuencialmente, se reciben comandos, se estima el estado del cuadricoptero (posicion
angular, velocidad angular y altura), se ejecutan los algoritmos PID, se aplican sus salidas a
los
   motores, y se envian mensajes de telemetria a la PC.
*/
void SecuenciaDeVuelo()
{
  while (modoEjecucion != '_')
  {
    RecibirComando();
    FiltroComplementario();
    CalcularAltura();
    //PIDAltura();
    PID_PosicionAngular();
    PID_VelocidadAngular();
    AplicarPWMmotores(velocidadBasePWM);
    EnviarMensajesTelemetriaPC();
  }
}
```

```
/////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////////
////////////////
//////////////////////////////////////////      PROCEDIMIENTO PARA ESTIMAR LA MEDIA DE RUIDO
POR CONDICIONES INICIALES DE VOLTAJE Y TEMPERATURA DEL GIROSCOPIO
//////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////////
////////////////
/*
  Se realizan 500 muestreos del giroscopio y se calcula la media de los mismos, la cual
representa el offset del sensor debido a las condiciones iniciales de voltaje de alimentacion
y temperatura.
*/
void CalcularOffsetGiroscopio() {
  numMuestras = 500;
  for (int n = 0; n < numMuestras ; n++) {
    gyro.read();
    G_offsetYPR[0] += gyro.g.z;
    G_offsetYPR[1] += gyro.g.x;
    G_offsetYPR[2] += gyro.g.y;
  }
  G_offsetYPR [0] = G_offsetYPR[0] / numMuestras;
  G_offsetYPR [1] = G_offsetYPR[1] / numMuestras;
  G_offsetYPR [2] = G_offsetYPR[2] / numMuestras;
}


/////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////      PROCEDIMIENTO PARA ESTIMAR LA MEDIA DE RUIDO
POR INCLINACION DEL ACELEROMETRO SOBRE EL CHASIS
//////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////////
/*
  Se realizan 500 muestreos del acelerometro y se calcula la media de los mismos, la cual
representa el offset del sensor debido a la inclinacion del mismo sobre el chasis del
cuadricoptero.
*/
void CalcularOffsetAcelerometro() {
  numMuestras = 500;
  for (int n = 0; n < numMuestras ; n++) {
    compass.read();
    A_offsetYPR[0] += compass.a.z;
    A_offsetYPR[1] += compass.a.y;
    A_offsetYPR[2] += compass.a.x;
  }
  A_offsetYPR [0] = A_offsetYPR[0] / numMuestras;
  A_offsetYPR [1] = A_offsetYPR[1] / numMuestras;
  A_offsetYPR [2] = A_offsetYPR[2] / numMuestras;
}


/////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////////
///////
//////////////////////////////////////////      PROCEDIMIENTO PARA ESTIMAR LA VELOCIDAD
ANGULAR, Y LA POSICION ANGULAR MEDIANTE FILTRO COMPLEMENTARIO
//////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////////
///////
/* El procedimiento FiltroComplementario() adquiere datos desde el acelerometro y el
giroscopio, los somete a filtrado por medias moviles, y calcula la posicion angular del
   cuadricoptero en Yaw, Pitch y Roll mediante filtro complementario. Se maneja estrictamente
la frecuencia de muestreo de los sensores a la hora de muestrear los datos y
   filtrar los datos.
*/
void FiltroComplementario() {
  gyro.read();
  compass.read();

  DT = (double)(micros() - tiempoUltimoMuestreoAngulos) / 1000000;
```

```
  if (millis() - tiempoUltimoMuestreoGiroscopio >= DT_giroscopio)
  {
    G_velocidadYPR[0] = (double) ((gyro.g.z - G_offsetYPR[0]) * G_GYRO );
    G_velocidadYPR[1] = (double) ((gyro.g.x - G_offsetYPR[1]) * G_GYRO );
    G_velocidadYPR[2] = (double) ((gyro.g.y - G_offsetYPR[2]) * G_GYRO );

    G_velocidadYPR_filtrada[0] = filtroVelocidadYPR [0].step ((double) G_velocidadYPR[0]);
    G_velocidadYPR_filtrada[1] = filtroVelocidadYPR [1].step ((double) G_velocidadYPR[1]);
    G_velocidadYPR_filtrada[2] = filtroVelocidadYPR [2].step ((double) G_velocidadYPR[2]);

    tiempoUltimoMuestreoGiroscopio = millis();

    anguloYPR_filtrado[0] = (double) (anguloYPR_filtrado[0] + G_velocidadYPR_filtrada[0] *
DT);
    anguloYPR_filtrado[1] = (double) (K_COMP * (anguloYPR_filtrado[1] +
G_velocidadYPR_filtrada[1] * DT) + (1 - K_COMP) * A_anguloYPR_filtrado[1]);
    anguloYPR_filtrado[2] = (double) (K_COMP * (anguloYPR_filtrado[2] +
G_velocidadYPR_filtrada[2] * DT) + (1 - K_COMP) * A_anguloYPR_filtrado[2]);
  }

  if (millis() - tiempoUltimoMuestreoAcelerometro >= DT_acelerometro)
  {
    A_aceleracionYPR[0] = (double) (compass.a.z) * G_ACC;
    A_aceleracionYPR[1] = (double) (compass.a.y - A_offsetYPR[1]) * G_ACC;
    A_aceleracionYPR[2] = (double) (compass.a.x - A_offsetYPR[2]) * G_ACC;


    A_aceleracionYPR_filtrada[0] = filtroAceleracionYPR[0].step((double) A_aceleracionYPR[0]);
    A_aceleracionYPR_filtrada[1] = filtroAceleracionYPR[1].step((double) A_aceleracionYPR[1]);
    A_aceleracionYPR_filtrada[2] = filtroAceleracionYPR[2].step((double) A_aceleracionYPR[2]);


    A_anguloYPR[0] = 0;
    A_anguloYPR[1] = (double) atan2(A_aceleracionYPR[1], sqrt(A_aceleracionYPR[0] *
A_aceleracionYPR[0] + A_aceleracionYPR[2] * A_aceleracionYPR[2]));
    A_anguloYPR[1] = ToDeg(A_anguloYPR[1]);
    A_anguloYPR[2] = (double) atan2(A_aceleracionYPR[2], sqrt(A_aceleracionYPR[0] *
A_aceleracionYPR[0] + A_aceleracionYPR[1] * A_aceleracionYPR[1]));
    A_anguloYPR[2] = ToDeg(A_anguloYPR[2]);

    A_anguloYPR_filtrado[0] = 0;
    A_anguloYPR_filtrado[1] = (double) atan2(A_aceleracionYPR_filtrada[1],
sqrt(A_aceleracionYPR_filtrada[0] * A_aceleracionYPR_filtrada[0] +
A_aceleracionYPR_filtrada[2] * A_aceleracionYPR_filtrada[2]));
    A_anguloYPR_filtrado[1] = ToDeg(A_anguloYPR_filtrado[1]);
    A_anguloYPR_filtrado[2] = (double) atan2(A_aceleracionYPR_filtrada[2],
sqrt(A_aceleracionYPR_filtrada[0] * A_aceleracionYPR_filtrada[0] +
A_aceleracionYPR_filtrada[1] * A_aceleracionYPR_filtrada[1]));
    A_anguloYPR_filtrado[2] = ToDeg(A_anguloYPR_filtrado[2]);

    //Az_sinG = (double) (9.81*(A_aceleracionYPR[0]-sqrt(1-
sin(anguloYPR_filtrado[1])*sin(anguloYPR_filtrado[1])-
sin(anguloYPR_filtrado[2])*sin(anguloYPR_filtrado[2]))));
    //A velocidad Z = (double) (A velocidad Z + Az sinG);
    tiempoUltimoMuestreoAcelerometro = millis();
  }

  anguloYPR_filtrado[0] = ToRad(anguloYPR_filtrado[0]);
  anguloYPR_filtrado[0] = (double) atan2(sin(anguloYPR_filtrado[0]),
cos(anguloYPR_filtrado[0]));
  anguloYPR_filtrado[0] = ToDeg(anguloYPR_filtrado[0]);

  anguloYPR_filtrado[1] = ToRad(anguloYPR_filtrado[1]);
  anguloYPR_filtrado[1] = (double) atan2(sin(anguloYPR_filtrado[1]),
cos(anguloYPR_filtrado[1]));
  anguloYPR_filtrado[1] = ToDeg(anguloYPR_filtrado[1]);

  anguloYPR_filtrado[2] = ToRad(anguloYPR_filtrado[2]);
  anguloYPR_filtrado[2] = (double) atan2(sin(anguloYPR_filtrado[2]),
cos(anguloYPR_filtrado[2]));
  anguloYPR_filtrado[2] = ToDeg(anguloYPR_filtrado[2]);

  tiempoUltimoMuestreoAngulos = micros();
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////
//////////////////////////////////////////      PROCEDIMIENTOS PARA CALCULAR LA ALTURA DEL
CUADRICOPTERO      ////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////
/*
  * En CalcularAltura() se calcula la distancia del sensor ultrasonico al suelo emitiendo un
pulso, y calculando el tiempo de respuesta del mismo.
    Conociendo previamente la velocidad del sonido, puede calcularse la distancia que recorrió
la onda, ida y vuelta. El procedimiento se apoya en
    la libreria NewPing para Arduino, que utiliza rutinas con temporizadores.
  * En FiltroKalmanAltura() se estima la distancia real del sensor ultrasonico al suelo
mediante un filtro de Kalman de primer orden.
*/
void CalcularAltura()
{
  if (millis() - tiempoUltimoMuestreoAltura > DT_sensor_altura)
  {
    uS = sonar.ping();
    distancia = (double) (uS / US_ROUNDTRIP_CM);

    if ((distancia >= 0) && (distancia < ALTURA_MAXIMA))
    {
      USAltura = distancia;
      FiltroKalmanAltura();
      alturaSuelo = estimacionAltura*1;
      mensajeEstado[8] = estimacionAltura;
      U_velocidad_Z = (estimacionAltura - U_Z_previo)/((double)DT_sensor_altura/1000.0);
      U_Z_previo = estimacionAltura;
      //velocidad_Z = (U_velocidad_Z + A_velocidad_Z)/2;
      velocidad_Z = U_velocidad_Z;
    }
    tiempoUltimoMuestreoAltura = millis();
  }
}
void FiltroKalmanAltura()
{
  covarianzaRuidoEstimacionAltura = covarianzaRuidoEstimacionAltura +
covarianzaProcesoFisicoAltura;
  gananciaKalman = covarianzaRuidoEstimacionAltura / (covarianzaRuidoEstimacionAltura +
covarianzaRuidoSensorAltura);
  estimacionAltura = estimacionAltura + gananciaKalman * (USAltura - estimacionAltura);
  covarianzaRuidoEstimacionAltura = (1 - gananciaKalman) * covarianzaRuidoEstimacionAltura;
}



////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////
//////////////////////////////////////      RUTINAS PARA EJECUTAR LOS PID DE POSICION
ANGULAR, VELOCIDAD ANGULAR Y ALTURA      ////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////
void PID_PosicionAngular()
{
  PID_pAngular_Yaw.Compute();
//  PID_pAngular_Pitch.Compute();
//  PID_pAngular_Roll.Compute();
}
void PID_VelocidadAngular()
{
  PID_vAngular_Yaw.Compute();
  PID_vAngular_Pitch.Compute();
  PID_vAngular_Roll.Compute();
}
void PIDAltura()
{
  PID_velZ.Compute();
  PID_posZ.Compute();
}



//////////////////////////////////////      RUTINA DE APLICACION DE PWM A LOS MOTORES
//////////////////////////////////////
// Valida el estado de la variable modoEjecucion:
//      * Si es igual a '_', envia PWM=0 a todos los puertos.
```

```
//      * Si es igual a 'T', se calcula el valor de PWM a enviar a los motores a partir de la
velocidad base del PWM establecida y las
//        salidas de los sistemas de control PID de velocidad angular y altura.
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////
void AplicarPWMmotores(int velocidadMotoresPWM)
{
  if (modoEjecucion == '_')
  {
    motorDerecho = 0;
    motorIzquierdo = 0;
    motorDelantero = 0;
    motorTrasero = 0;
  }
  if (modoEjecucion == 'T')
  {
    motorDerecho = velocidadMotoresPWM - correccionPWM_YPR[2] + correccionPWM_YPR[0] +
correccionAltura;
    motorIzquierdo = velocidadMotoresPWM + correccionPWM_YPR[2] + correccionPWM_YPR[0] +
correccionAltura;
    motorDelantero = velocidadMotoresPWM + correccionPWM_YPR[1] - correccionPWM_YPR[0] +
correccionAltura;
    motorTrasero = velocidadMotoresPWM - correccionPWM_YPR[1] - correccionPWM_YPR[0] +
correccionAltura;
  }

  if (motorDelantero > PWM_MAXIMO)
  {
    motorDelantero = PWM_MAXIMO;
  }
  else if (motorDelantero < 0)
  {
    motorDelantero = 0;
  }
  if (motorTrasero > PWM_MAXIMO)
  {
    motorTrasero = PWM_MAXIMO;
  }
  else if (motorTrasero < 0)
  {
    motorTrasero = 0;
  }
  if (motorDerecho > PWM_MAXIMO)
  {
    motorDerecho = PWM_MAXIMO;
  }
  else if (motorDerecho < 0)
  {
    motorDerecho = 0;
  }
  if (motorIzquierdo > PWM_MAXIMO)
  {
    motorIzquierdo = PWM_MAXIMO;
  }
  else if (motorIzquierdo < 0)
  {
    motorIzquierdo = 0;
  }

  analogWrite(PUERTOMOTORDERECHO, motorDerecho);
  analogWrite(PUERTOMOTORIZQUIERDO, motorIzquierdo);
  analogWrite(PUERTOMOTORSUPERIOR, motorDelantero);
  analogWrite(PUERTOMOTORINFERIOR, motorTrasero);
}


////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////
////////////////////////////////////////      RUTINA DE PREPARACION DEL PAQUETE DE MENSAJE DE
ESTADO      ////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////
/* Procedimiento para preparar paquetes de mensaje de estado. (14 bytes)
   posicion 0 = HEADER            (255)
   posicion 1 = CODIGO DE MENSAJE (7)
   posicion 2 = POSICION ANGULAR YAW (Valor positivo o == 0)
   posicion 3 = POSICION ANGULAR YAW (Valor negativo)
```

```
   posicion 4 = POSICION ANGULAR PITCH
   posicion 5 = POSICION ANGULAR ROLL
   posicion 6 = VELOCIDAD ANGULAR YAW (Valor positivo o == 0)
   posicion 7 = VELOCIDAD ANGULAR YAW (Valor negativo)
   posicion 8 = VELOCIDAD ANGULAR PITCH (Valor positivo o == 0)
   posicion 9 = VELOCIDAD ANGULAR PITCH (Valor negativo)
   posicion 10 = VELOCIDAD ANGULAR ROLL (Valor positivo o == 0)
   posicion 11 = VELOCIDAD ANGULAR ROLL (Valor negativo)
   posicion 12 = ALTURA
   posicion 13 = CHECKSUM (HECHO CON XOR DE LOS BYTES 0 AL 12)
 */
void PrepararPaqueteMensajeEstado()
{
  mensajeEstado[0] = CODIGO_INICIO_MENSAJE; //HEADER
  mensajeEstado[1] = CODIGO_ESTADO; //Codigo del mensaje

  if (anguloYPR_filtrado[0] >= 0)
  {
    mensajeEstado[2] = anguloYPR_filtrado[0];
    mensajeEstado[3] = 0;
  }
  else
  {
    mensajeEstado[3] = abs(anguloYPR_filtrado[0]);
    mensajeEstado[2] = 0;
  }
  mensajeEstado[4] = anguloYPR_filtrado[1] + 90;
  mensajeEstado[5] = anguloYPR_filtrado[2] + 90;

  if (G_velocidadYPR[0] >= 0)
  {
    mensajeEstado[6] = G_velocidadYPR[0];
    mensajeEstado[7] = 0;
  }
  else
  {
    mensajeEstado[7] = abs(G_velocidadYPR[0]);
    mensajeEstado[6] = 0;
  }
  if (G_velocidadYPR_filtrada[1] >= 0)
  {
    mensajeEstado[8] = G_velocidadYPR_filtrada[1];
    mensajeEstado[9] = 0;
  }
  else
  {
    mensajeEstado[9] = abs(G_velocidadYPR_filtrada[1]);
    mensajeEstado[8] = 0;
  }
  if (G_velocidadYPR_filtrada[2] >= 0)
  {
    mensajeEstado[10] = G_velocidadYPR_filtrada[2];
    mensajeEstado[11] = 0;
  }
  else
  {
    mensajeEstado[11] = abs(G_velocidadYPR_filtrada[2]);
    mensajeEstado[10] = 0;
  }

  mensajeEstado[12] = velocidadBasePWM;
//  mensajeEstado[12] = estimacionAltura;
  //  mensajeEstado[12] = alturaDeseada;
  if (modoEjecucion == 'T')
  {
    mensajeEstado[13] = 1;
  }
  else
  {
    mensajeEstado[13] = 0;
  }
  mensajeEstado[14] = (mensajeEstado[0] ^  mensajeEstado[1] ^   mensajeEstado[2] ^
mensajeEstado[3] ^ mensajeEstado[4] ^ mensajeEstado[5] ^ mensajeEstado[6] ^ mensajeEstado[7] ^
mensajeEstado[8] ^ mensajeEstado[9] ^ mensajeEstado[10] ^ mensajeEstado[11] ^
mensajeEstado[12] ^ mensajeEstado[13]); //CHECKSUM
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////
//////////////////////////////////////      RUTINA DE PREPARACION DEL PAQUETE DE MENSAJE DE
TELEMETRIA TOTAL     //////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////
/* Procedimiento para preparar paquetes de mensaje de telemetria total.  (42 bytes)
   posicion 0 = HEADER            (255)
   posicion 1 = CODIGO DE MENSAJE (7)
   posicion 2 = POSICION ANGULAR YAW (Valor positivo o == 0)
   posicion 3 = POSICION ANGULAR YAW (Valor negativo)
   posicion 4 = POSICION ANGULAR PITCH (Valor positivo o == 0)
   posicion 5 = POSICION ANGULAR PITCH (Valor negativo)
   posicion 6 = POSICION ANGULAR ROLL (Valor positivo o == 0)
   posicion 7 = POSICION ANGULAR ROLL (Valor negativo)
   posicion 8 = VELOCIDAD ANGULAR YAW (Valor positivo o == 0)
   posicion 9 = VELOCIDAD ANGULAR YAW (Valor negativo)
   posicion 10 = VELOCIDAD ANGULAR PITCH (Valor positivo o == 0)
   posicion 11 = VELOCIDAD ANGULAR PITCH (Valor negativo)
   posicion 12 = VELOCIDAD ANGULAR ROLL (Valor positivo o == 0)
   posicion 13 = VELOCIDAD ANGULAR ROLL (Valor negativo)
   posicion 14 = VELOCIDAD ANGULAR YAW FILTRADA (Valor positivo o == 0)
   posicion 15 = VELOCIDAD ANGULAR YAW FILTRADA (Valor negativo)
   posicion 16 = VELOCIDAD ANGULAR PITCH FILTRADA (Valor positivo o == 0)
   posicion 17 = VELOCIDAD ANGULAR PITCH FILTRADA (Valor negativo)
   posicion 18 = VELOCIDAD ANGULAR ROLL FILTRADA (Valor positivo o == 0)
   posicion 19 = VELOCIDAD ANGULAR ROLL FILTRADA (Valor negativo)
   posicion 20 = ACELERACION LINEAL Z (Valor positivo o == 0)
   posicion 21 = ACELERACION LINEAL Z (Valor negativo)
   posicion 22 = ACELERACION LINEAL Y (Valor positivo o == 0)
   posicion 23 = ACELERACION LINEAL Y (Valor negativo)
   posicion 24 = ACELERACION LINEAL X (Valor positivo o == 0)
   posicion 25 = ACELERACION LINEAL X (Valor negativo)
   posicion 26 = ACELERACION LINEAL Z FILTRADA (Valor positivo o == 0)
   posicion 27 = ACELERACION LINEAL Z FILTRADA (Valor negativo)
   posicion 28 = ACELERACION LINEAL Y FILTRADA (Valor positivo o == 0)
   posicion 29 = ACELERACION LINEAL Y FILTRADA (Valor negativo)
   posicion 30 = ACELERACION LINEAL X FILTRADA (Valor positivo o == 0)
   posicion 31 = ACELERACION LINEAL X FILTRADA (Valor negativo)
   posicion 32 = ALTURA - POSICION EN Z
   posicion 33 = ALTURA FILTRADA - POSICION EN Z FILTRADA
   posicion 34 = VELOCIDAD EN Z (Valor positivo o == 0)
   posicion 35 = VELOCIDAD EN Z (Valor negativo)
   posicion 36 = PWM MOTOR DELANTERO
   posicion 37 = PWM MOTOR TRASERO
   posicion 38 = PWM MOTOR DERECHO
   posicion 39 = PWM MOTOR IZQUIERDO
   posicion 40 = ESTADO DE ENCENDIDO DE MOTORES
   posicion 41 = CHECKSUM (HECHO CON XOR DE LOS BYTES 0 AL 40)
 */
void PrepararPaqueteMensajeTelemetriaTotal()
{
  mensajeTelemetriaTotal[0] = CODIGO_INICIO_MENSAJE; //HEADER
  mensajeTelemetriaTotal[1] = CODIGO_TELEMETRIA_TOTAL; //Codigo del mensaje
  if (anguloYPR_filtrado[0] >= 0)
  {
    mensajeTelemetriaTotal[2] = anguloYPR_filtrado[0];
    mensajeTelemetriaTotal[3] = 0;
  }
  else
  {
    mensajeTelemetriaTotal[3] = abs(anguloYPR_filtrado[0]);
    mensajeTelemetriaTotal[2] = 0;
  }
  if (anguloYPR_filtrado[1] >= 0)
  {
    mensajeTelemetriaTotal[4] = anguloYPR_filtrado[1];
    mensajeTelemetriaTotal[5] = 0;
  }
  else
  {
    mensajeTelemetriaTotal[5] = abs(anguloYPR_filtrado[1]);
    mensajeTelemetriaTotal[4] = 0;
  }
  if (anguloYPR_filtrado[2] >= 0)
  {
    mensajeTelemetriaTotal[6] = anguloYPR_filtrado[2];
    mensajeTelemetriaTotal[7] = 0;
```

```
}
else
{
  mensajeTelemetriaTotal[7] = abs(anguloYPR_filtrado[2]);
  mensajeTelemetriaTotal[6] = 0;
}

if (G_velocidadYPR[0] >= 0)
{
  mensajeTelemetriaTotal[8] = G_velocidadYPR[0];
  mensajeTelemetriaTotal[9] = 0;
}
else
{
  mensajeTelemetriaTotal[9] = abs(G_velocidadYPR[0]);
  mensajeTelemetriaTotal[8] = 0;
}
if (G_velocidadYPR[1] >= 0)
{
  mensajeTelemetriaTotal[10] = G_velocidadYPR[1];
  mensajeTelemetriaTotal[11] = 0;
}
else
{
  mensajeTelemetriaTotal[11] = abs(G_velocidadYPR[1]);
  mensajeTelemetriaTotal[10] = 0;
}
if (G_velocidadYPR[2] >= 0)
{
  mensajeTelemetriaTotal[12] = G_velocidadYPR[2];
  mensajeTelemetriaTotal[13] = 0;
}
else
{
  mensajeTelemetriaTotal[13] = abs(G_velocidadYPR[2]);
  mensajeTelemetriaTotal[12] = 0;
}

if (G_velocidadYPR_filtrada[0] >= 0)
{
  mensajeTelemetriaTotal[14] = G_velocidadYPR_filtrada[0];
  mensajeTelemetriaTotal[15] = 0;
}
else
{
  mensajeTelemetriaTotal[15] = abs(G_velocidadYPR_filtrada[0]);
  mensajeTelemetriaTotal[14] = 0;
}
if (G_velocidadYPR_filtrada[1] >= 0)
{
  mensajeTelemetriaTotal[16] = G_velocidadYPR_filtrada[1];
  mensajeTelemetriaTotal[17] = 0;
}
else
{
  mensajeTelemetriaTotal[17] = abs(G_velocidadYPR_filtrada[1]);
  mensajeTelemetriaTotal[16] = 0;
}
if (G_velocidadYPR_filtrada[2] >= 0)
{
  mensajeTelemetriaTotal[18] = G_velocidadYPR_filtrada[2];
  mensajeTelemetriaTotal[19] = 0;
}
else
{
  mensajeTelemetriaTotal[19] = abs(G_velocidadYPR_filtrada[2]);
  mensajeTelemetriaTotal[18] = 0;
}

if (A_aceleracionYPR[2] >= 0)
{
  mensajeTelemetriaTotal[20] = A_aceleracionYPR[2];
  mensajeTelemetriaTotal[21] = 0;
}
else
{
  mensajeTelemetriaTotal[21] = abs(A_aceleracionYPR[2]);
  mensajeTelemetriaTotal[20] = 0;
```

```
  }
  if (A_aceleracionYPR[1] >= 0)
  {
    mensajeTelemetriaTotal[22] = A_aceleracionYPR[1];
    mensajeTelemetriaTotal[23] = 0;
  }
  else
  {
    mensajeTelemetriaTotal[23] = abs(A_aceleracionYPR[1]);
    mensajeTelemetriaTotal[22] = 0;
  }
  if (A_aceleracionYPR[0] >= 0)
  {
    mensajeTelemetriaTotal[24] = A_aceleracionYPR[0];
    mensajeTelemetriaTotal[25] = 0;
  }
  else
  {
    mensajeTelemetriaTotal[25] = abs(A_aceleracionYPR[0]);
    mensajeTelemetriaTotal[24] = 0;
  }

  if (A_aceleracionYPR_filtrada[2] >= 0)
  {
    mensajeTelemetriaTotal[26] = A_aceleracionYPR_filtrada[2];
    mensajeTelemetriaTotal[27] = 0;
  }
  else
  {
    mensajeTelemetriaTotal[27] = abs(A_aceleracionYPR_filtrada[2]);
    mensajeTelemetriaTotal[26] = 0;
  }
  if (A_aceleracionYPR_filtrada[1] >= 0)
  {
    mensajeTelemetriaTotal[28] = A_aceleracionYPR_filtrada[1];
    mensajeTelemetriaTotal[29] = 0;
  }
  else
  {
    mensajeTelemetriaTotal[29] = abs(A_aceleracionYPR_filtrada[1]);
    mensajeTelemetriaTotal[28] = 0;
  }
  if (A_aceleracionYPR_filtrada[0] >= 0)
  {
    mensajeTelemetriaTotal[30] = A_aceleracionYPR_filtrada[0];
    mensajeTelemetriaTotal[31] = 0;
  }
  else
  {
    mensajeTelemetriaTotal[31] = abs(A_aceleracionYPR_filtrada[0]);
    mensajeTelemetriaTotal[30] = 0;
  }

  mensajeTelemetriaTotal[32] = USAltura;
//  mensajeTelemetriaTotal[32] = velocidadBasePWM;
  mensajeTelemetriaTotal[33] = estimacionAltura;
  if (velocidad_Z >= 0)
  {
    mensajeTelemetriaTotal[34] = velocidad_Z;
    mensajeTelemetriaTotal[35] = 0;
  }
  else
  {
    mensajeTelemetriaTotal[35] = abs(velocidad_Z);
    mensajeTelemetriaTotal[34] = 0;
  }
  mensajeTelemetriaTotal[36] = motorDelantero;
  mensajeTelemetriaTotal[37] = motorTrasero;
  mensajeTelemetriaTotal[38] = motorDerecho;
  mensajeTelemetriaTotal[39] = motorIzquierdo;

  if (modoEjecucion == 'T')
  {
    mensajeTelemetriaTotal[40] = 1;
  }
  else
  {
    mensajeTelemetriaTotal[40] = 0;
```

```
  }
  mensajeTelemetriaTotal[41] = (mensajeTelemetriaTotal[0] ^  mensajeTelemetriaTotal[1] ^
mensajeTelemetriaTotal[2] ^ mensajeTelemetriaTotal[3] ^ mensajeTelemetriaTotal[4] ^
mensajeTelemetriaTotal[5] ^ mensajeTelemetriaTotal[6] ^ mensajeTelemetriaTotal[7] ^
mensajeTelemetriaTotal[8] ^ mensajeTelemetriaTotal[9] ^ mensajeTelemetriaTotal[10] ^
mensajeTelemetriaTotal[11] ^ mensajeTelemetriaTotal[12] ^ mensajeTelemetriaTotal[13] ^
mensajeTelemetriaTotal[14] ^ mensajeTelemetriaTotal[15] ^ mensajeTelemetriaTotal[16] ^
mensajeTelemetriaTotal[17] ^ mensajeTelemetriaTotal[18] ^ mensajeTelemetriaTotal[19] ^
mensajeTelemetriaTotal[20] ^ mensajeTelemetriaTotal[21] ^ mensajeTelemetriaTotal[22] ^
mensajeTelemetriaTotal[23] ^ mensajeTelemetriaTotal[24] ^ mensajeTelemetriaTotal[25] ^
mensajeTelemetriaTotal[26] ^ mensajeTelemetriaTotal[27] ^ mensajeTelemetriaTotal[28] ^
mensajeTelemetriaTotal[29] ^ mensajeTelemetriaTotal[30] ^ mensajeTelemetriaTotal[31] ^
mensajeTelemetriaTotal[32] ^ mensajeTelemetriaTotal[33] ^ mensajeTelemetriaTotal[34] ^
mensajeTelemetriaTotal[35] ^ mensajeTelemetriaTotal[36] ^ mensajeTelemetriaTotal[37] ^
mensajeTelemetriaTotal[38] ^ mensajeTelemetriaTotal[39] ^ mensajeTelemetriaTotal[40]);
//CHECKSUM
}


////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////
//////////////////////////////////////////      PROCEDIMIENTO PARA ENVIAR MENSAJES DE
TELEMETRIA A LA PC      ////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////
/* Se valida el estado de la bandera modoTelemetriaTotal.
    * Si modoTelemetriaTotal==0, se envian mensajes de estado.
    * Si modoTelemetriaTotal==1, se envian mensajes de telemetria total.
*/
void EnviarMensajesTelemetriaPC()
{
  if (modoTelemetriaTotal == 0)
  {
    if (millis() - tiempoUltimoEnvio >= DT_envioDatosEstado)
    {
      PrepararPaqueteMensajeEstado();
      Serial.write(mensajeEstado, 15);//ENVIAR EL PAQUETE DE 15 BYTES
      tiempoUltimoEnvio = millis();
    }
  }
  else
  {
    if (millis() - tiempoUltimoEnvio >= DT_envioDatosTelemetriaTotal)
    {
      PrepararPaqueteMensajeTelemetriaTotal();
      Serial.write(mensajeTelemetriaTotal, 42);//ENVIAR EL PAQUETE DE 42 BYTES
      tiempoUltimoEnvio = millis();
    }
  }
}


////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////
//////////////////////////////////////      PROCEDIMIENTO PARA RECIBIR MENSAJES DESDE LA PC
//////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////
/* Se lee el puerto serial, y se verifica, de forma sucesiva que:
    1) El primer dato recibido sea un char de codigo 255 (Header de los mensajes del
protocolo).
    2) El valor del segundo dato recibido (Codigo del mensaje), a partir de lo cual se recibe:
        a) Comando de encendido/apagado (Codigo==0).
        b) Comando de movimiento (Codigo==1).
    3) Se realizar un checksum con todos los datos recibidos, y se compara con el ultimo dato
recibido en cada caso (Checksum recibido).
    4) Si el checksum recibido es igual al checksum calculado, el mensaje ha llegado
correctamente, y se ejecuta el comando modificando
       las variables modoEjecucion, velocidadDeseadaYPR y velocidadBasePWM segun corresponda.
Si se recibio un mensaje de encendido, se
       envia un mensaje de acknowledge a la PC.
  Si en algun momento no se recibe un mensaje, o si el checksum recibido no coincide con el
checksum calculado, se detiene la recepcion
  del mensaje, y la ejecucion del procedimiento llega a su fin.
*/
void RecibirComando()
```

```
{
  if (Serial.available() > 0)
  {
    if (Serial.available() > 0)
    {
      headerMensaje = Serial.read();
      delay(1);
      if (headerMensaje == CODIGO_INICIO_MENSAJE)
      {
        if (Serial.available() > 0)
        {
          codigoRecibido = Serial.read();
          delay(1);
          if (codigoRecibido == CODIGO_ENCENDIDO)
          {
            if (Serial.available() > 0)
            {
              comandoEncendidoRecibido = Serial.read();
              delay(1);
              if (Serial.available() > 0)
              {
                checksum = Serial.read();
                delay(1);
                if (CODIGO_INICIO_MENSAJE ^ CODIGO_ENCENDIDO ^ comandoEncendidoRecibido ==
checksum)
                {
                  if (comandoEncendidoRecibido == 1)
                  {
                    modoEjecucion = 'T';
                    digitalWrite(LED_ENCENDIDO, HIGH);
                  }
                  if (comandoEncendidoRecibido == 0)
                  {
                    modoEjecucion = '_';
                    velocidadDeseadaYPR[1] = 0.0;
                    velocidadDeseadaYPR[2] = 0.0;
                    alturaDeseada = 0;
                    digitalWrite(LED_ENCENDIDO, LOW);
                  }
                  EnviarAcknowledge(CODIGO_ENCENDIDO);
                }
              }
            }
          }
          if (codigoRecibido == CODIGO_MOVIMIENTO)
          {
            if (Serial.available() > 0)
            {
              comandoPitch = Serial.read();
              delay(1);
              if (Serial.available() > 0)
              {
                comandoRoll = Serial.read();
                delay(1);
                if (Serial.available() > 0)
                {
                  comandoAltura = Serial.read();
                  delay(1);
                  if (Serial.available() > 0)
                  {
                    checksum = Serial.read();
                    delay(1);
                    if (CODIGO_INICIO_MENSAJE ^ CODIGO_MOVIMIENTO ^ comandoPitch ^ comandoRoll
^ comandoAltura == checksum)
                    {
                      if ((abs(comandoPitch - MAXIMO_ANGULO_COMANDO) < MAXIMO_ANGULO_COMANDO)
&& (abs(comandoRoll - MAXIMO_ANGULO_COMANDO) < MAXIMO_ANGULO_COMANDO))
                      {
                        velocidadDeseadaYPR[1] = -(comandoPitch - MAXIMO_ANGULO_COMANDO);
                        velocidadDeseadaYPR[2] = (comandoRoll - MAXIMO_ANGULO_COMANDO);
                        if (comandoAltura <= PWM_MAXIMO)
                        {
                          velocidadBasePWM = comandoAltura;
                        }
                      }
                    }
                  }
                }
```

```
            }
          }
        }
      }
    }
  }
}


////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////
//////////////////////////////////////////      PROCEDIMIENTO PARA ENVIAR MENSAJE DE
ACKNOWLEDGE A LA PC      ////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////
/*  Se prepara y envia un mensaje de acknowledge, cuyo contenido viene dado por codigoMensaje.
*/
void EnviarAcknowledge(unsigned char codigoMensaje)
{
  ack[0] = CODIGO_INICIO_MENSAJE;
  ack[1] = CODIGO_ACK;
  ack[2] = codigoMensaje;
  ack[3] = (ack[0] ^ ack[1] ^ ack[2]);
  Serial.write(ack, 4);
}
```

Archivo FiltroMediaMovilGiroscopio.h:

```
class FiltroMediaMovil_Giroscopio
{
  public:
    FiltroMediaMovil_Giroscopio()
    {
      numeroVentanas = 2;
      factorAmortiguacion =  1/((double) numeroVentanas);
      for(i=0; i < numeroVentanas; i++)
      {
        v[i]=0.0;
      }
    }
  private:
    int i;
    double v[2];
    int numeroVentanas;
    double factorAmortiguacion;
    double valorFiltrado;
  public:
    double step(double x) //class II
    {
      for(i=0; i < numeroVentanas-1; i++)
      {
        v[i] = v[i+1];
      }
      v[numeroVentanas-1] = x;
      valorFiltrado = 0;
      for(i=0; i < numeroVentanas; i++)
      {
        valorFiltrado += (double) factorAmortiguacion*v[i];
      }
      return valorFiltrado;
    }
};
```

Archivo FiltroMediaMovilAcelerometro.h:

```
class FiltroMediaMovil Acelerometro
{
    public:
        FiltroMediaMovil_Acelerometro()
        {
            numeroVentanas = 40;
            factorAmortiguacion =  1/((double) numeroVentanas);
            for(i=0; i < numeroVentanas; i++)
            {
                v[i]=0.0;
            }
        }
    private:
        int i;
        double v[40];
        int numeroVentanas;
        double factorAmortiguacion;
        double valorFiltrado;
    public:
        double step(double x) //class II
        {
            for(i=0; i < numeroVentanas-1; i++)
            {
                v[i] = v[i+1];
            }
            v[numeroVentanas-1] = x;
            valorFiltrado = 0;
            for(i=0; i < numeroVentanas; i++)
            {
                valorFiltrado += (double) factorAmortiguacion*v[i];
            }
            return valorFiltrado;
        }
};
```

Apéndice F: Especificación de mensajes del software de control implementado en ROS

Archivo ComandoEncendido.msg:

uint8 encendido

Archivo ComandoMovimiento.msg:

int16 comandoPitch
int16 comandoRoll
int16 comandoAltura

Archivo EstadoCuadricoptero.msg:

time tiempoEjecucion
int32 anguloPitch
int32 anguloRoll
int32 anguloYaw
int32 velocidadPitch
int32 velocidadRoll
int32 velocidadYaw
int32 altura
int32 encendido
int64 mensajesRecibidos

Archivo TelemetriaTotal.msg:

time tiempoEjecucion

int32 anguloPitch
int32 anguloRoll
int32 anguloYaw
int32 G_Pitch
int32 G_Roll
int32 G_Yaw
int32 G_Pitch_filtrada
int32 G_Roll_filtrada
int32 G_Yaw_filtrada
int32 Ax
int32 Ay
int32 Az
int32 Ax_filtrada
int32 Ay_filtrada
int32 Az_filtrada
int32 posZ
int32 posZ_filtrada
int32 velZ
int32 motorDelantero
int32 motorTrasero
int32 motorDerecho

int32 motorIzquierdo
int32 encendido
int64 mensajesRecibidos

Apéndice G: Código del paquete de ROS comunicación_serial

Archivo run.py:

```
#!/usr/bin/env python

from handler_serial import HandlerSerial
from comunicacion_serial.msg import *
import rospy

#################################################VARIABLES Y CONSTANTES#########################
__MAXIMO_ANGULO_COMANDO = 70
__CODIGO_MENSAJE_ESTADO = 7
__CODIGO_MENSAJE_TELEMETRIA_TOTAL = 8
__MODO_TELEMETRIA_TOTAL = 1
tiempoInicioEjecucion = 0
publisherEstado = None
publisherTelemetria = None
handlerSerial = None


#################################################################################################################
#########################################
#################################PROCEDIMIENTO DE ENVIO DE SENAL DE ENCENDIDO O APAGADO DE
MOTORES#########################################################
#################################################################################################################
#########################################
#Envia el comando para encender o apagar motores por el serial del XBee

def callbackEnvioComandoEncendido(mensaje):
    #rospy.loginfo(rospy.get_caller_id()+"I heard %s",data)
    global handlerSerial
    handlerSerial.enviarComandoEncendido(mensaje.encendido)

#################################################################################################################
#########################################
#################################PROCEDIMIENTO DE ENVIO DE SENAL DE MOVIMIENTO DE
MOTORES#########################################################
#################################################################################################################
#########################################
#Envia comandos de movimientos (pitch,roll y altura) por el serial de Xbee

def callbackEnvioComandoMovimiento(mensaje):
    #rospy.loginfo(rospy.get_caller_id()+"I heard %s",data)
    global handlerSerial
    handlerSerial.enviarComandoMovimiento(mensaje.comandoPitch + __MAXIMO_ANGULO_COMANDO, mensaje.comandoRoll +
__MAXIMO_ANGULO_COMANDO, mensaje.comandoAltura)


#################################################################################################################
#########################################
#################################PROCEDIMIENTO DE RECEPCION DE MENSAJES
#########################################################
#################################################################################################################
#########################################


def secuenciaRecepcion():
    global tiempoInicioEjecucion, publisherEstado, publisherTelemetria, handlerSerial, __CODIGO_MENSAJE_ESTADO,
__CODIGO_MENSAJE_TELEMETRIA_TOTAL
```

```
    resultadoAccion = handlerSerial.recibirComandos()
    if (resultadoAccion == True):
    if (handlerSerial.mensajesRecibidos == 1):
        tiempoInicioEjecucion = rospy.get_rostime()
        if (handlerSerial.ultimoComandoRecibido == __CODIGO_MENSAJE_ESTADO):
            estado = EstadoCuadricoptero()
            estado.tiempoEjecucion = rospy.get_rostime() - tiempoInicioEjecucion
            estado.anguloPitch = handlerSerial.anguloPitch
            estado.anguloRoll = handlerSerial.anguloRoll
            estado.anguloYaw = handlerSerial.anguloYaw
            estado.velocidadPitch = handlerSerial.G_Pitch
            estado.velocidadRoll = handlerSerial.G_Roll
            estado.velocidadYaw = handlerSerial.G_Yaw
            estado.altura = handlerSerial.posZ
            estado.encendido = handlerSerial.encendidoMotores
            estado.mensajesRecibidos = handlerSerial.mensajesRecibidos
            rospy.loginfo(estado)
            publisherEstado.publish(estado)
        if (handlerSerial.ultimoComandoRecibido == __CODIGO_MENSAJE_TELEMETRIA_TOTAL):
            telemetria = TelemetriaTotal()
            telemetria.tiempoEjecucion = rospy.get_rostime() - tiempoInicioEjecucion
            telemetria.anguloPitch = handlerSerial.anguloPitch
            telemetria.anguloRoll = handlerSerial.anguloRoll
            telemetria.anguloYaw = handlerSerial.anguloYaw
            telemetria.G_Pitch = handlerSerial.G_Pitch
            telemetria.G_Roll = handlerSerial.G_Roll
            telemetria.G_Yaw = handlerSerial.G_Yaw
            telemetria.G_Pitch_filtrada = handlerSerial.G_Pitch_filtrada
            telemetria.G_Roll_filtrada = handlerSerial.G_Roll_filtrada
            telemetria.G_Yaw_filtrada = handlerSerial.G_Yaw_filtrada
            telemetria.Ax = handlerSerial.Ax
            telemetria.Ay = handlerSerial.Ay
            telemetria.Az = handlerSerial.Az
            telemetria.Ax_filtrada = handlerSerial.Ax_filtrada
            telemetria.Ay_filtrada = handlerSerial.Ay_filtrada
            telemetria.Az_filtrada = handlerSerial.Az_filtrada
            telemetria.posZ = handlerSerial.posZ
            telemetria.posZ_filtrada = handlerSerial.posZ_filtrada
            telemetria.velZ = handlerSerial.velZ
            telemetria.motorDelantero = handlerSerial.motorDelantero
            telemetria.motorTrasero = handlerSerial.motorTrasero
            telemetria.motorDerecho = handlerSerial.motorDerecho
            telemetria.motorIzquierdo = handlerSerial.motorIzquierdo
            telemetria.encendido = handlerSerial.encendidoMotores
            telemetria.mensajesRecibidos = handlerSerial.mensajesRecibidos
            rospy.loginfo(telemetria)
            publisherTelemetria.publish(telemetria)



def run():
    global publisherEstado, publisherTelemetria, handlerSerial, tiempoInicioEjecucion
    publisherEstado = rospy.Publisher('estado_cuadricoptero', EstadoCuadricoptero, queue_size=10)
    publisherTelemetria = rospy.Publisher('telemetria_total', TelemetriaTotal, queue_size=10)
    rospy.init_node('ManejadorPuertoSerial', anonymous=True)
    frecuencia = rospy.Rate(1000)
    if (__MODO_TELEMETRIA_TOTAL == 1):
        handlerSerial = HandlerSerial(nombrePuerto = "/dev/ttyUSB0", tasaBaudios = 115200)
    else:
        handlerSerial = HandlerSerial(nombrePuerto = "/dev/ttyUSB0", tasaBaudios = 38400)
    handlerSerial.abrirPuerto()
    rospy.Subscriber("encendido_cuadricoptero", ComandoEncendido, callbackEnvioComandoEncendido)
    rospy.Subscriber("movimientos_cuadricoptero", ComandoMovimiento, callbackEnvioComandoMovimiento)
    while not rospy.is_shutdown():
        secuenciaRecepcion()
        frecuencia.sleep()
```

```
    handlerSerial.cerrarPuerto()

if __name__ == '__main__':
    try:
        run()
    except rospy.ROSInterruptException: pass
```

Archivo handler_serial.py:

```
#!/usr/bin/env python

import serial
import numpy as np
from threading import Thread


class HandlerSerial:
    def __init__(self, nombrePuerto, tasaBaudios):
        ################################CONSTANTES#########################
        self.__CODIGO_INICIO_MENSAJE = 255
        self.__CODIGO_MENSAJE_ENCENDIDO = 0
        self.__CODIGO_MENSAJE_COMANDO = 1
        self.__CODIGO_MENSAJE_ACK_QR = 6
        self.__CODIGO_MENSAJE_ESTADO = 7
        self.__CODIGO_MENSAJE_TELEMETRIA_TOTAL = 8

        ################################VARIABLES##########################
        self.anguloYaw = 0
        self.anguloPitch = 0
        self.anguloRoll = 0
        self.G_Yaw = 0;
        self.G_Pitch = 0;
        self.G_Roll = 0;

        #Variables de mensaje de telemetria total.
        self.G_Yaw_filtrada = 0;
        self.G_Pitch_filtrada = 0;
        self.G_Roll_filtrada = 0;
        self.Ax = 0;                    #Aceleracion Cruda en el eje X
        self.Ay = 0;                    #Aceleracion Cruda en el eje Y
        self.Az = 0;                    #Aceleracion_Cruda en el eje Z
        self.Ax_filtrada = 0;
        self.Ay_filtrada = 0;
        self.Az_filtrada = 0;
        self.posZ = 0;                   #Altura
        self.posZ_filtrada = 0;
        self.velZ = 0;                  #Velocidad_Altura
        self.motorDelantero = 0;
        self.motorTrasero = 0;
        self.motorDerecho = 0;
        self.motorIzquierdo = 0;
        #Fin de variables mensaje de telemetria total.

        self.encendidoMotores = 0;
        self.mensajesRecibidos = 0
        self.nombrePuerto = nombrePuerto
        self.tasaBaudios = tasaBaudios

        self.puertoSerial = serial.Serial(port=self.nombrePuerto, baudrate=self.tasaBaudios, timeout=0.05)

    ################################PROCEDIMIENTOS Y FUNCIONES#########################

    def abrirPuerto(self):
        self.puertoSerial = serial.Serial(self.nombrePuerto, self.tasaBaudios)
```

```
    def cerrarPuerto(self):
        self.puertoSerial.close()

        ##########################################################################
        ##ENVIA EL MENSAJE DE MOVIMIENTOS EN EL CUADRICOPTERO                   ##
        ##########################################################################
        ##MENSAJE:                                                             ##
        ##  posicion 1     header: __codigo_inicio_mensaje (255)               ##
        ##  posicion 2     codigo del mensaje: __codigo_mensaje_comando (1)    ##
        ##  posicion 3     comando pitch: velocidad angular deseada en pitch   ##
        ##  posicion 4     comando roll: velocidad angular deseada en roll     ##
        ##  posicion 5     comando altura: velocidad angular deseada en altura ##
        ##  posicion 6     checksum: xor de contenido de mensaje               ##
        ##########################################################################
    def enviarComandoMovimiento(self, comandoPitch, comandoRoll, comandoAltura):
        checksum = self.__CODIGO_INICIO_MENSAJE ^ self.__CODIGO_MENSAJE_COMANDO ^ int(comandoPitch) ^ int(comandoRoll) ^
int(comandoAltura)
        paquete = chr(self.__CODIGO_INICIO_MENSAJE) + chr(self.__CODIGO_MENSAJE_COMANDO) + chr(int(comandoPitch)) +
chr(int(comandoRoll)) + chr(int(comandoAltura)) + chr(checksum)
        self.puertoSerial.write(paquete)
        """for i in range(6):
            print (ord(paquete[i]))"""

        ##########################################################################
        ##PROCEDIMIENTO PARA ENVIAR MENSAJES DE ENCENDIDO O APAGADO DE MOTORES  ##
        ##########################################################################
        ##MENSAJE:                                                             ##
        ##  posicion 1  header: __codigo_inicio_mensaje (255)                  ##
        ##  posicion 2  codigo del mensaje: __CODIGO_MENSAJE_ENCENDIDO (0)     ##
        ##  posicion 3  comando: Comando de motores APAGAR (0) ENCENDER(1)     ##
        ##  posicion 4  checksum: xor de contenido de mensaje                  ##
        ##########################################################################
    def enviarComandoEncendido(self, comando):
        checksum = self.__CODIGO_INICIO_MENSAJE ^ self.__CODIGO_MENSAJE_ENCENDIDO ^ comando
        paquete = chr(self.__CODIGO_INICIO_MENSAJE) + chr(self.__CODIGO_MENSAJE_ENCENDIDO) + chr(comando) + chr(checksum)
        self.puertoSerial.write(paquete)
        #self.mensajesEstadoRecibidos = 0
        """        for i in range(4):
            print (ord(paquete[i]))"""


##########################################################################################################################
############################
        ##PROCEDIMIENTO PARA RECIBIR MENSAJES DE TELEMETRIA Y ACK                                                       ##

##########################################################################################################################
############################
        ##Se lee el puerto serial, y se verifica, de forma sucesiva que:                                               ##
        ##   1) El primer dato recibido sea un char de codigo 255 (Header de los mensajes del protocolo).                 ##
        ##   2) El valor del segundo dato recibido (Codigo del mensaje), a partir de lo cual se recibe:                ##
        ##       a) Mensaje ACK (Codigo==6).                                                                          ##
        ##       b) Mensaje de estado (Codigo==7).                                                                    ##
        ##       b) Mensaje de telemetria total (Codigo==8).                                                          ##
        ##   3) Se realiza un checksum con todos los datos recibidos, y se compara con el ultimo dato recibido en cada caso (Checksum
recibido).    ##
        ##   4) Si el checksum recibido es igual al checksum calculado, el mensaje ha llegado correctamente, y se ejecuta el comando
modificando    ##
        ##      las variables modoEjecucion, velocidadDeseadaYPR y velocidadBasePWM segun corresponda. Si se recibio un mensaje de
encendido, se    ##
        ##      envia un mensaje de acknowledge a la PC.                                                             ##
        ## Si en algun momento no se recibe un mensaje, o si el checksum recibido no coincide con el checksum calculado, se detiene la
recepcion    ##
        ## del mensaje, y la ejecucion del procedimiento llega a su fin.                                              ##

##########################################################################################################################
############################
```

```
    def recibirComandos(self):
        header = self.cardinalCaracter_Validar(self.puertoSerial.read())
        if (header == self.__CODIGO_INICIO_MENSAJE):
            comando = self.cardinalCaracter_Validar(self.puertoSerial.read())
            if (comando == self.__CODIGO_MENSAJE_ACK_QR):
                codigoACK = self.cardinalCaracter_Validar(self.puertoSerial.read())
                checksum = self.cardinalCaracter_Validar(self.puertoSerial.read())


                if ((header ^ comando ^ codigoACK) == checksum):
                    print "ACK DE ENCENDIDO/APAGADO!!!!!"
                    self.ultimoComandoRecibido = comando
                    return True
                else:
                    return False


            if (comando == self.__CODIGO_MENSAJE_ESTADO):
                P_anguloYaw = self.cardinalCaracter_Validar(self.puertoSerial.read())
                N_anguloYaw = self.cardinalCaracter_Validar(self.puertoSerial.read())
                anguloPitch = self.cardinalCaracter_Validar(self.puertoSerial.read())
                anguloRoll = self.cardinalCaracter_Validar(self.puertoSerial.read())
                P_velocidadYaw = self.cardinalCaracter_Validar(self.puertoSerial.read())
                N_velocidadYaw = self.cardinalCaracter_Validar(self.puertoSerial.read())
                P_velocidadPitch = self.cardinalCaracter_Validar(self.puertoSerial.read())
                N_velocidadPitch = self.cardinalCaracter_Validar(self.puertoSerial.read())
                P_velocidadRoll = self.cardinalCaracter_Validar(self.puertoSerial.read())
                N_velocidadRoll = self.cardinalCaracter_Validar(self.puertoSerial.read())
                posZ = self.cardinalCaracter_Validar(self.puertoSerial.read())
                encendidoMotores = self.cardinalCaracter_Validar(self.puertoSerial.read())
                checksum = self.cardinalCaracter_Validar(self.puertoSerial.read())

                if((header != None) and (comando != None) and (P_anguloYaw != None) and (anguloPitch != None) and (anguloRoll != None) and
(P_velocidadYaw != None) and (P_velocidadPitch != None) and (P_velocidadRoll != None) and (N_anguloYaw != None) and (N_velocidadYaw
!= None) and (N_velocidadPitch != None) and (N_velocidadRoll != None) and (posZ != None) and (encendidoMotores != None) and
(checksum != None)):
                    if ((header ^ comando ^ P_anguloYaw ^ N_anguloYaw ^ anguloPitch ^ anguloRoll ^ P_velocidadYaw ^ N_velocidadYaw ^
P_velocidadPitch ^ N_velocidadPitch ^ P_velocidadRoll ^ N_velocidadRoll ^ posZ ^ encendidoMotores) == checksum):

                        self.mensajesRecibidos = self.mensajesRecibidos + 1

                            if (P_anguloYaw > 0):
                                self.anguloYaw = P_anguloYaw
                            else:
                                self.anguloYaw = - N_anguloYaw

                            self.anguloPitch = anguloPitch - 90
                            self.anguloRoll = anguloRoll - 90

                            if (P_velocidadYaw > 0):
                                self.G_Yaw = P_velocidadYaw
                            else:
                                self.G_Yaw = - N_velocidadYaw
                            if (P_velocidadPitch > 0):
                                self.G_Pitch = P_velocidadPitch
                            else:
                                self.G_Pitch = - N_velocidadPitch
                            if (P_velocidadRoll > 0):
                                self.G_Roll = P_velocidadRoll
                            else:
                                self.G_Roll = - N_velocidadRoll

                            self.posZ = posZ

                            self.encendidoMotores = encendidoMotores
                            self.ultimoComandoRecibido = comando
```

```
                return True
            else:
                return False


        if (comando == self.__CODIGO_MENSAJE_TELEMETRIA_TOTAL):
            P_anguloYaw = self.cardinalCaracter_Validar(self.puertoSerial.read())
            N_anguloYaw = self.cardinalCaracter_Validar(self.puertoSerial.read())
            P_anguloPitch = self.cardinalCaracter_Validar(self.puertoSerial.read())
            N_anguloPitch = self.cardinalCaracter_Validar(self.puertoSerial.read())
            P_anguloRoll = self.cardinalCaracter_Validar(self.puertoSerial.read())
            N_anguloRoll = self.cardinalCaracter_Validar(self.puertoSerial.read())
            P_G_Yaw = self.cardinalCaracter_Validar(self.puertoSerial.read())
            N_G_Yaw = self.cardinalCaracter_Validar(self.puertoSerial.read())
            P_G_Pitch = self.cardinalCaracter_Validar(self.puertoSerial.read())
            N_G_Pitch = self.cardinalCaracter_Validar(self.puertoSerial.read())
            P_G_Roll = self.cardinalCaracter_Validar(self.puertoSerial.read())
            N_G_Roll = self.cardinalCaracter_Validar(self.puertoSerial.read())
            P_G_Yaw_filtrada = self.cardinalCaracter_Validar(self.puertoSerial.read())
            N_G_Yaw_filtrada = self.cardinalCaracter_Validar(self.puertoSerial.read())
            P_G_Pitch_filtrada = self.cardinalCaracter_Validar(self.puertoSerial.read())
            N_G_Pitch_filtrada = self.cardinalCaracter_Validar(self.puertoSerial.read())
            P_G_Roll_filtrada = self.cardinalCaracter_Validar(self.puertoSerial.read())
            N_G_Roll_filtrada = self.cardinalCaracter_Validar(self.puertoSerial.read())
            P_Ax = self.cardinalCaracter_Validar(self.puertoSerial.read())
            N_Ax = self.cardinalCaracter_Validar(self.puertoSerial.read())
            P_Ay = self.cardinalCaracter_Validar(self.puertoSerial.read())
            N_Ay = self.cardinalCaracter_Validar(self.puertoSerial.read())
            P_Az = self.cardinalCaracter_Validar(self.puertoSerial.read())
            N_Az = self.cardinalCaracter_Validar(self.puertoSerial.read())
            P_Ax_filtrada = self.cardinalCaracter_Validar(self.puertoSerial.read())
            N_Ax_filtrada = self.cardinalCaracter_Validar(self.puertoSerial.read())
            P_Ay_filtrada = self.cardinalCaracter_Validar(self.puertoSerial.read())
            N_Ay_filtrada = self.cardinalCaracter_Validar(self.puertoSerial.read())
            P_Az_filtrada = self.cardinalCaracter_Validar(self.puertoSerial.read())
            N_Az_filtrada = self.cardinalCaracter_Validar(self.puertoSerial.read())
            posZ = self.cardinalCaracter_Validar(self.puertoSerial.read())
            posZ_filtrada = self.cardinalCaracter_Validar(self.puertoSerial.read())
            P_velZ = self.cardinalCaracter_Validar(self.puertoSerial.read())
            N_velZ = self.cardinalCaracter_Validar(self.puertoSerial.read())
            motorDelantero = self.cardinalCaracter_Validar(self.puertoSerial.read())
            motorTrasero = self.cardinalCaracter_Validar(self.puertoSerial.read())
            motorDerecho = self.cardinalCaracter_Validar(self.puertoSerial.read())
            motorIzquierdo = self.cardinalCaracter_Validar(self.puertoSerial.read())
            encendidoMotores = self.cardinalCaracter_Validar(self.puertoSerial.read())
            checksum = self.cardinalCaracter_Validar(self.puertoSerial.read())
            #print encendidoMotores
            print checksum

            validacionCamposMensaje = (header != None) and (comando != None) and (P_anguloYaw != None) and (P_anguloPitch != None)
and (P_anguloRoll != None) and (P_G_Yaw != None) and (P_G_Pitch != None) and (P_G_Roll != None) and (P_G_Yaw_filtrada != None) and
(P_G_Pitch_filtrada != None) and (P_G_Roll_filtrada != None)
            validacionCamposMensaje = validacionCamposMensaje and (N_anguloYaw != None) and (N_anguloPitch != None) and
(N_anguloRoll != None) and (N_G_Yaw != None) and (N_G_Pitch != None) and (N_G_Roll != None) and (N_G_Yaw_filtrada != None) and
(N_G_Pitch_filtrada != None) and (N_G_Roll_filtrada != None)
            validacionCamposMensaje = validacionCamposMensaje and (P_Ax != None) and (N_Ax != None) and (P_Ay != None) and (N_Ay !=
None) and (P_Az != None) and (N_Az != None) and (P_Ax_filtrada != None) and (N_Ax_filtrada != None) and (P_Ay_filtrada != None) and
(N_Ay_filtrada != None) and (P_Az_filtrada != None) and (N_Az_filtrada != None)
            validacionCamposMensaje = validacionCamposMensaje and (posZ != None) and (posZ_filtrada != None) and (P_velZ != None) and
(N_velZ != None) and (motorDelantero != None) and (motorTrasero != None) and (motorDerecho != None) and (motorIzquierdo != None) and
(encendidoMotores != None) and (checksum != None)
            if(validacionCamposMensaje):
                checksumMensajeRecibido = (header ^ comando ^ P_anguloYaw ^ N_anguloYaw ^ P_anguloPitch ^ N_anguloPitch ^
P_anguloRoll ^ N_anguloRoll ^ P_G_Yaw ^ N_G_Yaw ^ P_G_Pitch ^ N_G_Pitch ^ P_G_Roll ^ N_G_Roll ^ P_G_Yaw_filtrada ^
N_G_Yaw_filtrada ^ P_G_Pitch_filtrada ^ N_G_Pitch_filtrada ^ P_G_Roll_filtrada ^ N_G_Roll_filtrada)
```

```
            checksumMensajeRecibido = (checksumMensajeRecibido ^ P_Ax ^ N_Ax ^ P_Ay ^ N_Ay ^ P_Az ^ N_Az ^ P_Ax_filtrada ^
N_Ax_filtrada ^ P_Ay_filtrada ^ N_Ay_filtrada ^ P_Az_filtrada ^ N_Az_filtrada ^ posZ ^ posZ_filtrada ^ P_velZ ^ N_velZ ^ motorDelantero ^
motorTrasero ^ motorDerecho ^ motorIzquierdo ^ encendidoMotores)
            if (checksumMensajeRecibido == checksum):

              self.mensajesRecibidos = self.mensajesRecibidos + 1

              if (P_anguloYaw > 0):
                 self.anguloYaw = P_anguloYaw
              else:
                 self.anguloYaw = - N_anguloYaw
              if (P_anguloPitch > 0):
                 self.anguloPitch = P_anguloPitch
              else:
                 self.anguloPitch = - N_anguloPitch
              if (P_anguloRoll > 0):
                 self.anguloRoll = P_anguloRoll
              else:
                 self.anguloRoll = - N_anguloRoll

              if (P_G_Yaw > 0):
                 self.G_Yaw = P_G_Yaw
              else:
                 self.G_Yaw = - N_G_Yaw
              if (P_G_Pitch > 0):
                 self.G_Pitch = P_G_Pitch
              else:
                 self.G_Pitch = - N_G_Pitch
              if (P_G_Roll > 0):
                 self.G_Roll = P_G_Roll
              else:
                 self.G_Roll = - N_G_Roll

              if (P_G_Yaw_filtrada > 0):
                 self.G_Yaw_filtrada = P_G_Yaw_filtrada
              else:
                 self.G_Yaw_filtrada = - N_G_Yaw_filtrada
              if (P_G_Pitch_filtrada > 0):
                 self.G_Pitch_filtrada = P_G_Pitch_filtrada
              else:
                 self.G_Pitch_filtrada = - N_G_Pitch_filtrada
              if (P_G_Roll_filtrada > 0):
                 self.G_Roll_filtrada = P_G_Roll_filtrada
              else:
                 self.G_Roll_filtrada = - N_G_Roll_filtrada

              if (P_Ax > 0):
                 self.Ax = P_Ax
              else:
                 self.Ax = - N_Ax
              if (P_Ay > 0):
                 self.Ay = P_Ay
              else:
                 self.Ay = - N_Ay
              if (P_Az > 0):
                 self.Az = P_Az
              else:
                 self.Az = - N_Az

              if (P_Ax_filtrada > 0):
                 self.Ax_filtrada = P_Ax_filtrada
              else:
                 self.Ax_filtrada = - N_Ax_filtrada
              if (P_Ay_filtrada > 0):
                 self.Ay_filtrada = P_Ay_filtrada
              else:
                 self.Ay_filtrada = - N_Ay_filtrada
```

```
                if (P_Az_filtrada > 0):
                    self.Az_filtrada = P_Az_filtrada
                else:
                    self.Az_filtrada = - N_Az_filtrada

                self.posZ = posZ
                self.posZ_filtrada = posZ_filtrada

                if (P_velZ > 0):
                    self.velZ = P_velZ
                else:
                    self.velZ = - N_velZ

                self.motorDelantero = motorDelantero
                self.motorTrasero = motorTrasero
                self.motorDerecho = motorDerecho
                self.motorIzquierdo = motorIzquierdo

                self.encendidoMotores = encendidoMotores
                self.ultimoComandoRecibido = comando

                return True
            else:
                return False

    ##########################################################################
    ##PROCEDIMIENTO CONVERTIR DE CARACTER A UN NUMERO                       ##
    ##########################################################################
    ##Recibe un caracter y retorna su valor en ASCII (entre 0 y 255)        ##
    ##en caso de error retorna 1000                                         ##
    ##########################################################################
  def cardinalCaracter_Validar(self, caracter):
    if(len(caracter) > 0):
        if((ord(caracter) >= 0) and (ord(caracter) <= 255)):
            return ord(caracter)
    else:
        return 1000


#hilo = HiloSerial(None, 100000, "/dev/ttyUSB0", 38400)
```

Apéndice H: Código del paquete de ROS logitech_gamepad_ii

Archivo run.py:

```
#!/usr/bin/env python

from handler_joystick import HandlerJoystick
from logitech_rumblepad_ii.msg import *
import rospy




def run():
    publisherEncendido = rospy.Publisher('encendido_cuadricoptero', ComandoEncendido, queue_size=10)
    publisherMovimientos = rospy.Publisher('movimientos_cuadricoptero', ComandoMovimiento, queue_size=10)
    rospy.init_node('ManejadorJoystick', anonymous=True)
    frecuencia = rospy.Rate(100)
    handlerJoystick = HandlerJoystick()
    handlerJoystick.detectarJoystick()

    while not rospy.is_shutdown():
        handlerJoystick.run()
        if (handlerJoystick.comandoEncendidoEjecutado == True):
            mensajeEncendido = ComandoEncendido()
            mensajeEncendido.encendido = handlerJoystick.motoresEncendidos
            rospy.loginfo("Comando de encendido!")
            rospy.loginfo(mensajeEncendido)
            publisherEncendido.publish(mensajeEncendido)
        if (handlerJoystick.comandoMovimientoEjecutado == True):
            mensajeMovimiento = ComandoMovimiento()
            mensajeMovimiento.comandoPitch = handlerJoystick.comandoPitch
            mensajeMovimiento.comandoRoll = handlerJoystick.comandoRoll
            mensajeMovimiento.comandoAltura = handlerJoystick.comandoAltura
            rospy.loginfo("Comando de movimiento!")
            rospy.loginfo(mensajeMovimiento)
            publisherMovimientos.publish(mensajeMovimiento)
        frecuencia.sleep()

if __name__ == '__main__':
    try:
        run()
    except rospy.ROSInterruptException: pass
```

Archivo handler_joystick.py:

```
# -*- coding: utf-8 -*-
"""
Created on Mon Aug 11 08:10:57 2014

@author: alfredoso
"""

# -*- coding: utf-8 -*-

import pygame
import pygame.display
import pygame.joystick
import time
from threading import Thread



class HandlerJoystick:

    def __init__(self):
    ##########################CONSTANTES#########################
        self.__BOTON_2 = 1          #ENCENDER MOTORES
        self.__BOTON_4 = 3          #APAGAR MOTORES
        self.__BOTON_5 = 4
        self.__BOTON_6 = 5          #CONSIGNAS EN 0
        self.__BOTON_7 = 6
        self.__RUEDA_DERECHA_x = 2    #CONTROL DE PITCH
        self.__RUEDA_DERECHA_y = 3    #CONTROL DE YAW
        self.__RUEDA_IZQUIERDA_x = 0  #CONTROL VELOCIDAD BASE PWM
        self.__RUEDA_IZQUIERDA_y = 1

        self.__CODIGO_AUMENTAR_ALTURA = '+'
        self.__CODIGO_DISMINUIR_ALTURA = '-'
        self.__CODIGO_MANTENER_ALTURA = '='
        self.__CODIGO_ENCENDER = 1
        self.__CODIGO_APAGAR = 0
        self.__MAXIMO_ANGULO_COMANDO = 70
        self.__VELOCIDAD_MAXIMA_PWM = 240
        self.__VELOCIDAD_BASE_PWM = 120
        self.__MAXIMA_ALTURA = 150
        self.__FACTOR_EXPONENCIAL = 0.7
    ##########################VARIABLES#########################
        self.comandoPitch = 0.0
        self.comandoRoll = 0.0
        self.comandoAltura = self.__VELOCIDAD_BASE_PWM
        self.motoresEncendidos = 0
        self.comandoEncendidoEjecutado = False
        self.comandoMovimientoEjecutado = False

        self.hilo = Thread(target=self.run)



        ##########################################################################
        ##DETECTA EL JOYSTICK IMPRIME LA INFORMACION Y RETORNA EL OBJETO JOYSTICK##
        ##########################################################################
    def detectarJoystick(self):
        pygame.joystick.init() #initialize joystick module
        pygame.joystick.get_init() #verify initialization (boolean)
        joystick_count = pygame.joystick.get_count()#get number of joysticks
```

```
        print('%d joystick(s) connected' %joystick_count)#print number

        if (joystick_count == 1):
            joystick_object = pygame.joystick.Joystick(0)
            #create an instance of a joystick
            #first joystick is [0] in the list
            #haven't had much luck with multiple joysticks

            joystick_object.get_name()
            print joystick_object.get_name()
            #grab joystick name - flightstick, gravis...
            #can (and is in this case) be done before initialization

            joystick_object.init()

            joystick_object.get_init()
            #verify initialization (maybe cool to do some
            #error trapping with this so game doesn't crash

            num_axes = joystick_object.get_numaxes()
            num_buttons = joystick_object.get_numbuttons()

            print 'Joystick has %d axes and %d buttons' %(num_axes, num_buttons)


            #necessary for os to pass joystick events
            self.joystick_object = joystick_object



##########################################################################################################################
############################
        ##HILO DE EJECUCION DEL JOYSTICK                                                                  ##

##########################################################################################################################
############################
        ## Busca el control de joystick y a patir de alli maneja los siguientes eventos de manera separada:                ##
        ##          1)Rueda derecha: maneja movimientos en pitch(rueda derecha x) y en roll (rueda derecha y)                 ##
        ##          2)Rueda izquierda: maneja las velocidades en altura                                                     ##
        ##          3)Botorn 2: envia senal de encendido al cuadricoptero                                                   ##
        ##          4)boton 4: envia senal de apagado al cuadricoptero                                                      ##
        ##          5)boton 6 (R1): Envia 0 a las consignas de pitch y roll                                                 ##

##########################################################################################################################
############################
    def run(self):
        pygame.display.init()
        pygame.event.pump()

        if (self.joystick_object != None):

            estado_boton_DELTA = self.joystick_object.get_button(self.__BOTON_4)

            #while (estado_boton_DELTA == 0):
            self.comandoEncendidoEjecutado = False
            self.comandoMovimientoEjecutado = False
            eventos = pygame.event.get()
            for evento in eventos:
                if evento.type == pygame.JOYAXISMOTION:
                    movimientoX_ruedaDerecha = self.joystick_object.get_axis(self.__RUEDA_DERECHA_x)
                    movimientoY_ruedaDerecha = -self.joystick_object.get_axis(self.__RUEDA_DERECHA_y)

                    movimientoX_ruedaIzquierda = self.joystick_object.get_axis(self.__RUEDA_IZQUIERDA_x)
                    movimientoY_ruedaIzquierda = self.joystick_object.get_axis(self.__RUEDA_IZQUIERDA_y)

                    if ((movimientoY_ruedaDerecha == 0) and (self.comandoPitch != 0)):
```

```
                self.comandoPitch = 0

            if ((movimientoX_ruedaDerecha == 0) and (self.comandoRoll != 0)):
                self.comandoRoll = 0

            if ((movimientoY_ruedaIzquierda == 0) and (self.comandoAltura != self.__VELOCIDAD_BASE_PWM)):
                self.comandoAltura = 0

            if (movimientoX_ruedaDerecha != 0) or (movimientoY_ruedaDerecha != 0) or (movimientoY_ruedaIzquierda != 0):
                #print 'Rueda Derecha, Posicion en x= %f' %movimientoX_ruedaDerecha
                #print 'Rueda Derecha, Posicion en y= %f' %movimientoY_ruedaDerecha
                exponencialPitch = ((1-self.__FACTOR_EXPONENCIAL)*movimientoY_ruedaDerecha) +
(self.__FACTOR_EXPONENCIAL*movimientoY_ruedaDerecha**3)
                self.comandoPitch = exponencialPitch*self.__MAXIMO_ANGULO_COMANDO
                exponencialRoll = ((1-self.__FACTOR_EXPONENCIAL)*movimientoX_ruedaDerecha) +
(self.__FACTOR_EXPONENCIAL*movimientoX_ruedaDerecha**3)
                self.comandoRoll = exponencialRoll*self.__MAXIMO_ANGULO_COMANDO
                self.comandoAltura = self.__VELOCIDAD_BASE_PWM + (self.__VELOCIDAD_MAXIMA_PWM -
self.__VELOCIDAD_BASE_PWM)*(-1)*movimientoY_ruedaIzquierda

                #print 'Comando de pitch= %f' %(self.comandoPitch)
                #print 'Comando de roll= %f' %(self.comandoRoll)

                self.comandoMovimientoEjecutado = True
#         time.sleep(0.05)


          if evento.type == pygame.JOYBUTTONDOWN:
            estado_boton_X = self.joystick_object.get_button(self.__BOTON_2)
            estado_boton_DELTA = self.joystick_object.get_button(self.__BOTON_4)
            estado_boton_R1 = self.joystick_object.get_button(self.__BOTON_6)

            if(estado_boton_X == 1) or (estado_boton_DELTA == 1):
                if (estado_boton_X == 1):
                    self.motoresEncendidos = 1
                    #print 'Boton X: ' + str(estado_boton_X)

                if (estado_boton_DELTA == 1):
                    self.motoresEncendidos = 0
                    #print 'Boton DELTA: ' + str(estado_boton_DELTA)

                self.comandoEncendidoEjecutado = True

            else:
                if (estado_boton_R1 == 1):
                    self.comandoPitchEnviar = 0.0
                    self.comandoRollEnviar = 0.0
                movimientoY_ruedaIzquierda = self.joystick_object.get_axis(self.__RUEDA_IZQUIERDA_y)
                    self.comandoAltura = self.__VELOCIDAD_BASE_PWM + (self.__VELOCIDAD_MAXIMA_PWM -
self.__VELOCIDAD_BASE_PWM)*(-1)*movimientoY_ruedaIzquierda
#                     self.comandoAltura = self.__VELOCIDAD_BASE_PWM
                    self.comandoMovimientoEjecutado = True


 #               print 'Boton R1, Valor %d' %(estado_boton_R1)
        #time.sleep(0.075) #Permitir que otros hilos pasen a ejecutarse. Es como un yield()
```

Apéndice I: Código del paquete de ROS exportador_estado_csv

Archivo run.py:

```
#!/usr/bin/env python

import rospy
from exportador_estado_csv.msg import *
from os.path import expanduser

handlerArchivo = None
numMensajes = 0
separadorColumnas = ","
separadorFilas = "\n"
carpetaHome = expanduser("~")

def procesarMensajeEstado(mensajeEstado):
    global numMensajes, handlerArchivo
    if (numMensajes == 0):
        handlerArchivo = open(carpetaHome + "/estado.csv", "w+")
        filaArchivo = "TiempoEjecucion" + separadorColumnas + "AnguloPitch" + separadorColumnas + "AnguloRoll" + separadorColumnas +
"AnguloYaw" + separadorColumnas + "VelocidadPitch" + separadorColumnas + "VelocidadRoll" + separadorColumnas + "VelocidadYaw" +
separadorColumnas + "Altura" + separadorColumnas + "EstadoEncendido" + separadorColumnas + "MensajesRecibidos" + separadorFilas
        handlerArchivo.write(filaArchivo)
    else:
        filaArchivo = str(mensajeEstado.tiempoEjecucion.secs + float(mensajeEstado.tiempoEjecucion.nsecs)/1000000000) +
separadorColumnas
        filaArchivo = filaArchivo + str(mensajeEstado.anguloPitch) + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeEstado.anguloRoll) + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeEstado.anguloYaw)  + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeEstado.velocidadPitch) + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeEstado.velocidadRoll) + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeEstado.velocidadYaw)  + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeEstado.altura) + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeEstado.encendido) + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeEstado.mensajesRecibidos)  + separadorFilas
        handlerArchivo.write(filaArchivo)
    print filaArchivo
    numMensajes = numMensajes + 1

    #rospy.loginfo(mensajePose)



def listener():
    rospy.init_node('exportador_estado_csv', anonymous=True)
    rospy.Subscriber('estado_cuadricoptero', EstadoCuadricoptero, procesarMensajeEstado)
    rospy.spin()

if __name__ == '__main__':
    try:
        listener()
    except rospy.ROSInterruptException:
        if (handlerArchivo != None):
            handlerArchivo.close()
```

Apéndice J: Código del paquete de ROS exportador_telemetria_csv.

Archivo run.py:

```
#!/usr/bin/env python

import rospy
from exportador_telemetria_csv.msg import *
from os.path import expanduser

handlerArchivo = None
numMensajes = 0
separadorColumnas = ","
separadorFilas = "\n"
carpetaHome = expanduser("~")

def procesarMensajeTelemetria(mensajeTelemetria):
    global numMensajes, handlerArchivo
    if (numMensajes == 0):
        handlerArchivo = open(carpetaHome + "/telemetria.csv", "w+")
        filaArchivo = "TiempoEjecucion" + separadorColumnas + "AnguloPitch" + separadorColumnas + "AnguloRoll" + separadorColumnas +
"AnguloYaw" + separadorColumnas
        filaArchivo = filaArchivo + "G_Pitch" + separadorColumnas + "G_Roll" + separadorColumnas + "G_Yaw" + separadorColumnas
        filaArchivo = filaArchivo + "G_Pitch_filtrada" + separadorColumnas + "G_Roll_filtrada" + separadorColumnas + "G_Yaw_filtrada" +
separadorColumnas
        filaArchivo = filaArchivo + "Ax" + separadorColumnas + "Ay" + separadorColumnas + "Az" + separadorColumnas
        filaArchivo = filaArchivo + "Ax_filtrada" + separadorColumnas + "Ay_filtrada" + separadorColumnas + "Az_filtrada" + separadorColumnas
        filaArchivo = filaArchivo + "posZ" + separadorColumnas + "posZ_filtrada" + separadorColumnas + "velZ" + separadorColumnas
        filaArchivo = filaArchivo + "motorDelantero" + separadorColumnas + "motorTrasero" + separadorColumnas + "motorDerecho" +
separadorColumnas + "motorIzquierdo" + separadorColumnas
        filaArchivo = filaArchivo + "EstadoEncendido" + separadorColumnas + "MensajesRecibidos" + separadorFilas
        handlerArchivo.write(filaArchivo)
    else:
        filaArchivo = str(mensajeTelemetria.tiempoEjecucion.secs + float(mensajeTelemetria.tiempoEjecucion.nsecs)/1000000000) +
separadorColumnas
        filaArchivo = filaArchivo + str(mensajeTelemetria.anguloPitch) + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeTelemetria.anguloRoll) + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeTelemetria.anguloYaw)  + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeTelemetria.G_Pitch) + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeTelemetria.G_Roll) + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeTelemetria.G_Yaw)  + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeTelemetria.G_Pitch_filtrada) + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeTelemetria.G_Roll_filtrada) + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeTelemetria.G_Yaw_filtrada)  + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeTelemetria.Ax) + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeTelemetria.Ay) + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeTelemetria.Az)  + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeTelemetria.Ax_filtrada) + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeTelemetria.Ay_filtrada) + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeTelemetria.Az_filtrada)  + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeTelemetria.posZ) + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeTelemetria.posZ_filtrada) + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeTelemetria.velZ) + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeTelemetria.motorDelantero)  + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeTelemetria.motorTrasero)  + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeTelemetria.motorDerecho)  + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeTelemetria.motorIzquierdo)  + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeTelemetria.encendido) + separadorColumnas
        filaArchivo = filaArchivo + str(mensajeTelemetria.mensajesRecibidos)  + separadorFilas
        handlerArchivo.write(filaArchivo)
```

```
    print filaArchivo
    numMensajes = numMensajes + 1

    #rospy.loginfo(mensajePose)



def listener():
    rospy.init_node('exportador_telemetria_csv', anonymous=True)
    rospy.Subscriber('telemetria_total', TelemetriaTotal, procesarMensajeTelemetria)
    rospy.spin()

if __name__ == '__main__':
    try:
        listener()
    except rospy.ROSInterruptException:
        if (handlerArchivo != None):
            handlerArchivo.close()
```

Apéndice K: Código del script de MATLAB para análisis de respuesta en frecuencia a partir de la interfaz de telemetría, adaptado al tópico de ROS telemetría_total

```
function VisualizacionAnalisisFrecuencia(nombreArchivoCSV, frecuenciaMuestreo)
    datosArchivo = csvread(nombreArchivoCSV, 1, 0);
    numeroDatos = size(datosArchivo,1);
    anguloPitch = datosArchivo(:,2);
    anguloRoll = datosArchivo(:,3);
    anguloYaw = datosArchivo(:,4);
    velocidadPitch = datosArchivo(:,5);
    velocidadRoll = datosArchivo(:,6);
    velocidadYaw = datosArchivo(:,7);
    velocidadPitch_filtrada = datosArchivo(:,8);
    velocidadRoll_filtrada = datosArchivo(:,9);
    velocidadYaw_filtrada = datosArchivo(:,10);
    aceleracionPitch = datosArchivo(:,11);
    aceleracionRoll = datosArchivo(:,12);
    aceleracionYaw = datosArchivo(:,13);
    aceleracionPitch_filtrada = datosArchivo(:,14);
    aceleracionRoll_filtrada = datosArchivo(:,15);
    aceleracionYaw_filtrada = datosArchivo(:,16);
    posZ = datosArchivo(:,17);
    posZ_filtrada = datosArchivo(:,18);
    velZ = datosArchivo(:,19);
    motorDelantero = datosArchivo(:,20);
    motorTrasero = datosArchivo(:,21);
    motorDerecho = datosArchivo(:,22);
    motorIzquierdo = datosArchivo(:,23);




    NFFT_senal = 2^nextpow2(numeroDatos);
    f = frecuenciaMuestreo/2*linspace(0,1,NFFT_senal/2+1);



    figure_angulos = figure('position', [0, 0, 9999, 9999],'name','Angulos')
    subplot(2,3,1)
    plot(anguloPitch)
    title('Pitch')
    xlabel('Tiempo (s)')
    ylabel('Angulo (grados)')
    if (abs(min(anguloPitch)) > max(anguloPitch))
        ylim([min(anguloPitch) abs(min(anguloPitch))])
    else
        ylim([-max(anguloPitch) 1+max(anguloPitch)])
    end
    xlim([0 length(anguloPitch)])
```

```
subplot(2,3,2)
plot(anguloRoll)
title('Roll')
xlabel('Tiempo (s)')
ylabel('Angulo (grados)')
if (abs(min(anguloRoll)) > max(anguloRoll))
   ylim([min(anguloRoll) abs(min(anguloRoll))])
else
   ylim([-max(anguloRoll) 1+max(anguloRoll)])
end
xlim([0 length(anguloRoll)])
subplot(2,3,3)
plot(anguloYaw)
title('Yaw')
xlabel('Tiempo (s)')
ylabel('Angulo (grados)')
xlim([0 length(anguloYaw)])
if (abs(min(anguloYaw)) > max(anguloYaw))
   ylim([min(anguloYaw) abs(min(anguloYaw))])
else
   ylim([-max(anguloYaw) 1+max(anguloYaw)])
end

fft_anguloPitch = fft(anguloPitch, NFFT_senal)/numeroDatos;
fft_anguloRoll = fft(anguloRoll, NFFT_senal)/numeroDatos;
fft_anguloYaw = fft(anguloYaw, NFFT_senal)/numeroDatos;

subplot(2,3,4)
plot(f,2*abs(fft_anguloPitch(1:NFFT_senal/2+1)))
title('Pitch')
xlabel('Frecuencia (Hz)')
ylabel('Amplitud (grados)')
subplot(2,3,5)
plot(f,2*abs(fft_anguloRoll(1:NFFT_senal/2+1)))
title('Roll')
xlabel('Frecuencia (Hz)')
ylabel('Amplitud (grados)')
subplot(2,3,6)
plot(f,2*abs(fft_anguloYaw(1:NFFT_senal/2+1)))
title('Yaw')
xlabel('Frecuencia (Hz)')
ylabel('Amplitud (grados)')



figure_velocidades_angulares = figure('position', [0, 0, 9999, 9999],'name','Velocidades angulares')
subplot(2,3,1)
plot(velocidadPitch)
title('Pitch')
xlabel('Tiempo (s)')
ylabel('V. angular (grados/s)')
```

```
if (abs(min(velocidadPitch)) > max(velocidadPitch))
   ylim([min(velocidadPitch) abs(min(velocidadPitch))])
else
   ylim([-max(velocidadPitch) 1+max(velocidadPitch)])
end
xlim([0 length(velocidadPitch)])
subplot(2,3,2)
plot(velocidadRoll)
title('Roll')
xlabel('Tiempo (s)')
ylabel('V. angular (grados/s)')
if (abs(min(velocidadRoll)) > max(velocidadRoll))
   ylim([min(velocidadRoll) abs(min(velocidadRoll))])
else
   ylim([-max(velocidadRoll) 1+max(velocidadRoll)])
end
xlim([0 length(velocidadRoll)])
subplot(2,3,3)
plot(velocidadYaw)
title('Yaw')
xlabel('Tiempo (s)')
ylabel('V. angular (grados/s)')
xlim([0 length(velocidadYaw)])
if (abs(min(velocidadYaw)) > max(velocidadYaw))
   ylim([min(velocidadYaw) abs(min(velocidadYaw))])
else
   ylim([-max(velocidadYaw) 1+max(velocidadYaw)])
end


fft_velocidadPitch = fft(velocidadPitch, NFFT_senal)/numeroDatos;
fft_velocidadRoll = fft(velocidadRoll, NFFT_senal)/numeroDatos;
fft_velocidadYaw = fft(velocidadYaw, NFFT_senal)/numeroDatos;

subplot(2,3,4)
plot(f,2*abs(fft_velocidadPitch(1:NFFT_senal/2+1)))
ylim([0 5])
title('Pitch')
xlabel('Frecuencia (Hz)')
ylabel('Amplitud (grados/s)')
subplot(2,3,5)
plot(f,2*abs(fft_velocidadRoll(1:NFFT_senal/2+1)))
ylim([0 5])
title('Roll')
xlabel('Frecuencia (Hz)')
ylabel('Amplitud (grados/s)')
subplot(2,3,6)
plot(f,2*abs(fft_velocidadYaw(1:NFFT_senal/2+1)))
ylim([0 5])
title('Yaw')
xlabel('Frecuencia (Hz)')
ylabel('Amplitud (grados/s)')
```

```
    figure_velocidades_angulares_filtradas = figure('position', [0, 0, 9999, 9999],'name','Velocidades angulares
filtradas')
    subplot(2,3,1)
    plot(velocidadPitch_filtrada)
    title('Pitch filtrada')
    xlabel('Tiempo (s)')
    ylabel('V. angular (grados/s)')
    if (abs(min(velocidadPitch_filtrada)) > max(velocidadPitch_filtrada))
        ylim([min(velocidadPitch_filtrada) abs(min(velocidadPitch_filtrada))])
    else
        ylim([-max(velocidadPitch_filtrada) 1+max(velocidadPitch_filtrada)])
    end
    xlim([0 length(velocidadPitch_filtrada)])
    subplot(2,3,2)
    plot(velocidadRoll_filtrada)
    title('Roll filtrada')
    xlabel('Tiempo (s)')
    ylabel('V. angular (grados/s)')
    if (abs(min(velocidadRoll_filtrada)) > max(velocidadRoll_filtrada))
        ylim([min(velocidadRoll_filtrada) abs(min(velocidadRoll_filtrada))])
    else
        ylim([-max(velocidadRoll_filtrada) 1+max(velocidadRoll_filtrada)])
    end
    xlim([0 length(velocidadRoll_filtrada)])
    subplot(2,3,3)
    plot(velocidadYaw_filtrada)
    title('Yaw filtrada')
    xlabel('Tiempo (s)')
    ylabel('V. angular (grados/s)')
    xlim([0 length(velocidadYaw_filtrada)])
    if (abs(min(velocidadYaw_filtrada)) > max(velocidadYaw_filtrada))
        ylim([min(velocidadYaw_filtrada) abs(min(velocidadYaw_filtrada))])
    else
        ylim([-max(velocidadYaw_filtrada) 1+max(velocidadYaw_filtrada)])
    end

    fft_velocidadPitch_filtrada = fft(velocidadPitch_filtrada, NFFT_senal)/numeroDatos;
    fft_velocidadRoll_filtrada = fft(velocidadRoll_filtrada, NFFT_senal)/numeroDatos;
    fft_velocidadYaw_filtrada = fft(velocidadYaw_filtrada, NFFT_senal)/numeroDatos;

    subplot(2,3,4)
    plot(f,2*abs(fft_velocidadPitch_filtrada(1:NFFT_senal/2+1)))
    ylim([0 5])
    title('Pitch filtrada')
    xlabel('Frecuencia (Hz)')
    ylabel('Amplitud (grados/s)')
    subplot(2,3,5)
    plot(f,2*abs(fft_velocidadRoll_filtrada(1:NFFT_senal/2+1)))
    ylim([0 5])
    title('Roll filtrada')
    xlabel('Frecuencia (Hz)')
    ylabel('Amplitud (grados/s)')
```

```
subplot(2,3,6)
plot(f,2*abs(fft_velocidadYaw_filtrada(1:NFFT_senal/2+1)))
ylim([0 5])
title('Yaw filtrada')
xlabel('Frecuencia (Hz)')
ylabel('Amplitud (grados/s)')



figure_aceleraciones_angulares = figure('position', [0, 0, 9999, 9999],'name','Aceleraciones lineales')
subplot(2,3,1)
plot(aceleracionPitch)
title('Pitch')
xlabel('Tiempo (s)')
ylabel('Acel. lineal (m/s^2)')
if (abs(min(aceleracionPitch)) > max(aceleracionPitch))
   ylim([min(aceleracionPitch) abs(min(aceleracionPitch))])
else
   ylim([-max(aceleracionPitch) 1+max(aceleracionPitch)])
end
xlim([0 length(aceleracionPitch)])
subplot(2,3,2)
plot(aceleracionRoll)
title('Roll')
xlabel('Tiempo (s)')
ylabel('Acel. lineal (m/s^2)')
if (abs(min(aceleracionRoll)) > max(aceleracionRoll))
   ylim([min(aceleracionRoll) abs(min(aceleracionRoll))])
else
   ylim([-max(aceleracionRoll) 1+max(aceleracionRoll)])
end
xlim([0 length(aceleracionRoll)])
subplot(2,3,3)
plot(aceleracionYaw)
title('Yaw')
xlabel('Tiempo (s)')
ylabel('Acel. lineal (m/s^2)')
xlim([0 length(aceleracionYaw)])
if (abs(min(aceleracionYaw)) > max(aceleracionYaw))
   ylim([min(aceleracionYaw) abs(min(aceleracionYaw))])
else
   ylim([-max(aceleracionYaw) 1+max(aceleracionYaw)])
end

fft_aceleracionPitch = fft(aceleracionPitch, NFFT_senal)/numeroDatos;
fft_aceleracionRoll = fft(aceleracionRoll, NFFT_senal)/numeroDatos;
fft_aceleracionYaw = fft(aceleracionYaw, NFFT_senal)/numeroDatos;

subplot(2,3,4)
plot(f,2*abs(fft_aceleracionPitch(1:NFFT_senal/2+1)))
title('Pitch')
```

```
xlabel('Frecuencia (Hz)')
ylabel('Amplitud (m/s^2)')
subplot(2,3,5)
plot(f,2*abs(fft_aceleracionRoll(1:NFFT_senal/2+1)))
title('Roll')
xlabel('Frecuencia (Hz)')
ylabel('Amplitud (m/s^2)')
subplot(2,3,6)
plot(f,2*abs(fft_aceleracionYaw(1:NFFT_senal/2+1)))
title('Yaw')
xlabel('Frecuencia (Hz)')
ylabel('Amplitud (m/s^2)')



figure_aceleraciones_angulares_filtradas = figure('position', [0, 0, 9999, 9999],'name','Aceleraciones lineales
filtradas')
subplot(2,3,1)
plot(aceleracionPitch_filtrada)
title('Pitch filtrada')
xlabel('Tiempo (s)')
ylabel('Acel. lineal (m/s^2)')
if (abs(min(aceleracionPitch_filtrada)) > max(aceleracionPitch_filtrada))
   ylim([min(aceleracionPitch_filtrada) abs(min(aceleracionPitch_filtrada))])
else
   ylim([-max(aceleracionPitch_filtrada) 1+max(aceleracionPitch_filtrada)])
end
xlim([0 length(aceleracionPitch_filtrada)])
subplot(2,3,2)
plot(aceleracionRoll_filtrada)
title('Roll filtrada')
xlabel('Tiempo (s)')
ylabel('Acel. lineal (m/s^2)')
if (abs(min(aceleracionRoll_filtrada)) > max(aceleracionRoll_filtrada))
   ylim([min(aceleracionRoll_filtrada) abs(min(aceleracionRoll_filtrada))])
else
   ylim([-max(aceleracionRoll_filtrada) 1+max(aceleracionRoll_filtrada)])
end
xlim([0 length(aceleracionRoll_filtrada)])
subplot(2,3,3)
plot(aceleracionYaw_filtrada)
title('Yaw filtrada')
xlabel('Tiempo (s)')
ylabel('Acel. lineal (m/s^2)')
xlim([0 length(aceleracionYaw_filtrada)])
if (abs(min(aceleracionYaw_filtrada)) > max(aceleracionYaw_filtrada))
   ylim([min(aceleracionYaw_filtrada) abs(min(aceleracionYaw_filtrada))])
else
   ylim([-max(aceleracionYaw_filtrada) 1+max(aceleracionYaw_filtrada)])
end

fft_aceleracionPitch_filtrada = fft(aceleracionPitch_filtrada, NFFT_senal)/numeroDatos;
fft_aceleracionRoll_filtrada = fft(aceleracionRoll_filtrada, NFFT_senal)/numeroDatos;
```

```
fft_aceleracionYaw_filtrada = fft(aceleracionYaw_filtrada, NFFT_senal)/numeroDatos;

subplot(2,3,4)
plot(f,2*abs(fft_aceleracionPitch_filtrada(1:NFFT_senal/2+1)))
title('Pitch filtrada')
xlabel('Frecuencia (Hz)')
ylabel('Amplitud (m/s^2)')
subplot(2,3,5)
plot(f,2*abs(fft_aceleracionRoll_filtrada(1:NFFT_senal/2+1)))
title('Roll filtrada')
xlabel('Frecuencia (Hz)')
ylabel('Amplitud (m/s^2)')
subplot(2,3,6)
plot(f,2*abs(fft_aceleracionYaw_filtrada(1:NFFT_senal/2+1)))
title('Yaw filtrada')
xlabel('Frecuencia (Hz)')
ylabel('Amplitud (m/s^2)')



figure_posicion_velocidad_z = figure('position', [0, 0, 9999, 9999],'name','Posicion y velocidad en Z (Altura)')
subplot(2,3,1)
plot(posZ)
title('Posicion en Z')
xlabel('Tiempo (s)')
ylabel('Posicion (cm)')
if (abs(min(posZ)) > max(posZ))
   ylim([min(posZ) abs(min(posZ))])
else
   ylim([-max(posZ) 1+max(posZ)])
end
xlim([0 length(posZ)])
subplot(2,3,2)
plot(posZ_filtrada)
title('Posicion en Z filtrada')
xlabel('Tiempo (s)')
ylabel('Posicion (cm)')
if (abs(min(posZ_filtrada)) > max(posZ_filtrada))
   ylim([min(posZ_filtrada) abs(min(posZ_filtrada))])
else
   ylim([-max(posZ_filtrada) 1+max(posZ_filtrada)])
end
xlim([0 length(posZ_filtrada)])
subplot(2,3,3)
plot(velZ)
title('Velocidad en Z')
xlabel('Tiempo (s)')
ylabel('Posicion (cm/s)')
xlim([0 length(velZ)])
ylim([0 1+abs(max(velZ))])

fft_posZ = fft(posZ, NFFT_senal)/numeroDatos;
fft_posZ_filtrada = fft(posZ_filtrada, NFFT_senal)/numeroDatos;
```

```
    fft_velZ = fft(velZ, NFFT_senal)/numeroDatos;

    subplot(2,3,4)
    plot(f,2*abs(fft_posZ(1:NFFT_senal/2+1)))
    ylim([0 5])
    title('Posicion en Z')
    xlabel('Frecuencia (Hz)')
    ylabel('Amplitud (cm)')
    subplot(2,3,5)
    plot(f,2*abs(fft_posZ_filtrada(1:NFFT_senal/2+1)))
    ylim([0 5])
    title('Posicion en Z filtrada')
    xlabel('Frecuencia (Hz)')
    ylabel('Amplitud (cm)')
    subplot(2,3,6)
    plot(f,2*abs(fft_velZ(1:NFFT_senal/2+1)))
    ylim([0 5])
    title('Velocidad en Z')
    xlabel('Frecuencia (Hz)')
    ylabel('Amplitud (cm/s)')
end
```

Apéndice K: Código del script de MATLAB para análisis de respuesta en frecuencia a partir de la interfaz de telemetría, adaptado al tópico de ROS estado_cuadricoptero

```
function VisualizacionAnalisisFrecuencia(nombreArchivoCSV, frecuenciaMuestreo)
    datosArchivo = csvread(nombreArchivoCSV, 1, 0);
    numeroDatos = size(datosArchivo,1);
    anguloPitch = datosArchivo(:,2);
    anguloRoll = datosArchivo(:,3);
    anguloYaw = datosArchivo(:,4);
    velocidadPitch = datosArchivo(:,5);
    velocidadRoll = datosArchivo(:,6);
    velocidadYaw = datosArchivo(:,7);
    posZ = datosArchivo(:,8);

        NFFT_senal = 2^nextpow2(numeroDatos);
    f = frecuenciaMuestreo/2*linspace(0,1,NFFT_senal/2+1);

    figure_angulos = figure('position', [0, 0, 9999, 9999],'name','Angulos')
    subplot(2,3,1)
    plot(anguloPitch)
    title('Pitch')
    xlabel('Tiempo (s)')
    ylabel('Angulo (grados)')
    if (abs(min(anguloPitch)) > max(anguloPitch))
        ylim([min(anguloPitch) abs(min(anguloPitch))])
    else
        ylim([-max(anguloPitch) 1+max(anguloPitch)])
    end
    xlim([0 length(anguloPitch)])
    subplot(2,3,2)
    plot(anguloRoll)
    title('Roll')
    xlabel('Tiempo (s)')
    ylabel('Angulo (grados)')
    if (abs(min(anguloRoll)) > max(anguloRoll))
        ylim([min(anguloRoll) abs(min(anguloRoll))])
    else
        ylim([-max(anguloRoll) 1+max(anguloRoll)])
    end
    xlim([0 length(anguloRoll)])
    subplot(2,3,3)
    plot(anguloYaw)
    title('Yaw')
    xlabel('Tiempo (s)')
    ylabel('Angulo (grados)')
    xlim([0 length(anguloYaw)])
    if (abs(min(anguloYaw)) > max(anguloYaw))
```

```
    ylim([min(anguloYaw) abs(min(anguloYaw))])
else
    ylim([-max(anguloYaw) 1+max(anguloYaw)])
end

fft_anguloPitch = fft(anguloPitch, NFFT_senal)/numeroDatos;
fft_anguloRoll = fft(anguloRoll, NFFT_senal)/numeroDatos;
fft_anguloYaw = fft(anguloYaw, NFFT_senal)/numeroDatos;

subplot(2,3,4)
plot(f,2*abs(fft_anguloPitch(1:NFFT_senal/2+1)))
title('Pitch')
xlabel('Frecuencia (Hz)')
ylabel('Amplitud (grados)')
subplot(2,3,5)
plot(f,2*abs(fft_anguloRoll(1:NFFT_senal/2+1)))
title('Roll')
xlabel('Frecuencia (Hz)')
ylabel('Amplitud (grados)')
subplot(2,3,6)
plot(f,2*abs(fft_anguloYaw(1:NFFT_senal/2+1)))
title('Yaw')
xlabel('Frecuencia (Hz)')
ylabel('Amplitud (grados)')

figure_velocidades_angulares = figure('position', [0, 0, 9999, 9999],'name','Velocidades angulares')
subplot(2,3,1)
plot(velocidadPitch)
title('Pitch')
xlabel('Tiempo (s)')
ylabel('V. angular (grados/s)')
if (abs(min(velocidadPitch)) > max(velocidadPitch))
    ylim([min(velocidadPitch) abs(min(velocidadPitch))])
else
    ylim([-max(velocidadPitch) 1+max(velocidadPitch)])
end
xlim([0 length(velocidadPitch)])
subplot(2,3,2)
plot(velocidadRoll)
title('Roll')
xlabel('Tiempo (s)')
ylabel('V. angular (grados/s)')
if (abs(min(velocidadRoll)) > max(velocidadRoll))
    ylim([min(velocidadRoll) abs(min(velocidadRoll))])
else
    ylim([-max(velocidadRoll) 1+max(velocidadRoll)])
end
xlim([0 length(velocidadRoll)])
subplot(2,3,3)
plot(velocidadYaw)
title('Yaw')
xlabel('Tiempo (s)')
ylabel('V. angular (grados/s)')
```

```
    xlim([0 length(velocidadYaw)])
    if (abs(min(velocidadYaw)) > max(velocidadYaw))
        ylim([min(velocidadYaw) abs(min(velocidadYaw))])
    else
        ylim([-max(velocidadYaw) 1+max(velocidadYaw)])
    end

    fft_velocidadPitch = fft(velocidadPitch, NFFT_senal)/numeroDatos;
    fft_velocidadRoll = fft(velocidadRoll, NFFT_senal)/numeroDatos;
    fft_velocidadYaw = fft(velocidadYaw, NFFT_senal)/numeroDatos;

    subplot(2,3,4)
    plot(f,2*abs(fft_velocidadPitch(1:NFFT_senal/2+1)))
    ylim([0 5])
    title('Pitch')
    xlabel('Frecuencia (Hz)')
    ylabel('Amplitud (grados/s)')
    subplot(2,3,5)
    plot(f,2*abs(fft_velocidadRoll(1:NFFT_senal/2+1)))
    ylim([0 5])
    title('Roll')
    xlabel('Frecuencia (Hz)')
    ylabel('Amplitud (grados/s)')
    subplot(2,3,6)
    plot(f,2*abs(fft_velocidadYaw(1:NFFT_senal/2+1)))
    ylim([0 5])
    title('Yaw')
    xlabel('Frecuencia (Hz)')
    ylabel('Amplitud (grados/s)')

    figure_posicion_velocidad_z = figure('position', [0, 0, 9999, 9999],'name','Posicion y velocidad en Z (Altura)')
    subplot(2,2,1)
    plot(posZ)
    title('Posicion en Z')
    xlabel('Tiempo (s)')
    ylabel('Posicion (cm)')
    if (abs(min(posZ)) > max(posZ))
        ylim([min(posZ) abs(min(posZ))])
    else
        ylim([-max(posZ) 1+max(posZ)])
    end

    fft_posZ = fft(posZ, NFFT_senal)/numeroDatos;

    subplot(2,2,3)
    plot(f,2*abs(fft_posZ(1:NFFT_senal/2+1)))
    ylim([0 5])
    title('Posicion en Z')
    xlabel('Frecuencia (Hz)')
    ylabel('Amplitud (cm)')
end
```

Apéndice L: Código del script de MATLAB para realizar la simulación de la arquitectura de control propuesta

```
%%%%%   Parametros del cuadricoptero   %%%%%
masaBateriaPequena = 0.0265;
masaBateriaGrande = 0.090;
masaBaseAnime = 0.0075;
masaMotor = 0.051;
masaUltrasonido = 0.009;
masaChasisCircuitos = 0.482;
masaTotal = 0.6;
masaCentral = 0.23;
longitudAspa = 0.31;
longitudEje =  0.47;
alturaMotor = 0.095;
diametroMotor = 0.025;
diametroEje = 0.005;
alturaCajaCentral = 0.1;
anchuraCajaCentral = 0.09;
radioMotor = diametroMotor/2;

%%MATRIZ DE INERCIA%%
   %%PITCH%%
      %%MOTORES SOBRE EL EJE(para un motor)%%
      inerciaPitch(1) = 1/12*(masaMotor)*((3*(radioMotor/2))^2+alturaMotor^2);
      %%MOTORES EN EJE OPUESTO (para un motor)%%
      inerciaPitch(2) = 1/12*(masaMotor)*((3*(radioMotor/2))^2+alturaMotor^2)+(masaMotor*longitudEje);
      %%CAJA CENTRAL%%
      inerciaPitch(3) = 1/12 * (masaCentral+masaBaseAnime+masaChasisCircuitos) *
(alturaCajaCentral^2+anchuraCajaCentral^2);
      %%TOTAL INERCIA PITCH%%
      inercia(1,1)= 2*inerciaPitch(1)+2*inerciaPitch(2) + inerciaPitch(3);
   %%ROLL%%
      %%MOTORES SOBRE EL EJE(para un motor)%%
      inerciaRoll(1) = 1/12*(masaMotor)*((3*(radioMotor/2))^2+alturaMotor^2);
      %%MOTORES EN EJE OPUESTO (para un motor)%%
      inerciaRoll(2) = 1/12*(masaMotor)*((3*(radioMotor/2))^2+alturaMotor^2)+(masaMotor*longitudEje);
      %%CAJA CENTRAL%%
      inerciaRoll(3) = 1/12 * (masaCentral+masaBaseAnime+masaChasisCircuitos) *
(alturaCajaCentral^2+anchuraCajaCentral^2);
      %%TOTAL INERCIA ROLL%%
      inercia(2,2)= 2*inerciaRoll(1)+2*inerciaRoll(2) + inerciaRoll(3);
    %%YAW%%
      %%PARA CADA MOTOR%%
      inerciaYaw(1) = 1/12*(masaMotor)*((3*(radioMotor/2))^2+alturaMotor^2)+(masaMotor*longitudEje);
      %%CAJA CENTRAL%%
      inerciaYaw(2) = 1/12 * (masaCentral+masaBaseAnime+masaChasisCircuitos) *
(anchuraCajaCentral^2+anchuraCajaCentral^2);


      inercia(3,3) = 4 * inerciaYaw(1) + inerciaYaw(2);
```

```
Ixx = inercia(1,1)
Iyy = inercia(2,2)
Izz = inercia(3,3)
% Ixx = 0.0142;
% Iyy = 0.0142;
% Izz = 0.0071;
longitudEje =  0.47;
L = longitudEje/2;
masa = 0.6;
gravedad = 9.81;
c = 0.1; %Consultar referencia. Que es eso?


syms Vm1 Vm2 Vm3 Vm4;
Fm1(Vm1) = 9.81*(22.4935*Vm1 - 9.732)/1000;
Fm2(Vm2) = 9.81*(22.4926*Vm2 - 9.5271)/1000;
Fm3(Vm3) = 9.81*(22.6127*Vm3 - 9.8941)/1000;
Fm4(Vm4) = 9.81*(22.6184*Vm4 - 9.5805)/1000;


syms pitch roll yaw;
Rx(pitch) = [1, 0, 0; 0, cos(pitch), -sin(pitch); 0, sin(pitch), cos(pitch)];
Ry(roll) = [cos(roll), 0, sin(roll); 0, 1, 0; -sin(roll), 0, cos(roll)];
Rz(yaw) = [cos(yaw), -sin(yaw), 0; sin(yaw), cos(yaw), 0; 0, 0, 1];

R(yaw,roll,pitch) = Rz(yaw) * Ry(roll) * Rx(pitch);

syms u v w;
Vw(u,v,w) = [u; v; w];
dZdt(roll,pitch,u,v,w) = -sin(roll)*u + sin(pitch)*cos(roll)*v + cos(roll)*cos(pitch)*w;

T(yaw,roll,pitch) = [1, tan(roll)*sin(pitch), tan(roll)*cos(pitch); 0, cos(pitch), -sin(pitch); 0, sec(roll)*sin(pitch),
sec(roll)*cos(pitch)];
syms p q r;
Omega(p,q,r) = [p; q; r];
dYPRdt(yaw,roll,pitch,p,q,r) = T(yaw,roll,pitch)*Omega(p,q,r); % Derivada de yaw, roll y pitch, en funcion de las
velocidades angulares p, q y r

G = [0; 0; gravedad];
F(Vm1,Vm2,Vm3,Vm4) = [0; 0; -Fm1(Vm1) - Fm2(Vm2) - Fm3(Vm3) - Fm4(Vm4)];
Aw(Vm1,Vm2,Vm3,Vm4,p,q,r,u,v,w) = (1/masa)*F(Vm1,Vm2,Vm3,Vm4) + R(yaw,roll,pitch)*G -
cross(Omega(p,q,r),Vw(u,v,w)); %Aceleracion del cuerpo en funcion de:
                                                                %Las velocidades angulares p, q y r
                                                                %El voltaje de los motores Vm1, Vm2,
Vm3 y Vm4
                                                                %Las velocidades lineales del cuerpo u,
v y w

I = [Ixx, 0, 0; 0, Iyy, 0; 0, 0, Izz];
```

```
Torques(Vm1,Vm2,Vm3,Vm4) = [L*(Fm2(Vm2) - Fm4(Vm4)); L*(Fm1(Vm1) - Fm3(Vm3)); c*(-Fm1(Vm1) +
Fm2(Vm2) - Fm3(Vm3) + Fm4(Vm4))];
dOmegadt(Vm1,Vm2,Vm3,Vm4,p,q,r) = inv(I)*Torques(Vm1,Vm2,Vm3,Vm4) - cross(inv(I)*Omega(p,q,r),
I*Omega(p,q,r)); %Aceleraciones angulares en funcion de:
                                                        %Las velocidades angulares p, q y r
                                                        %El voltaje de los motores Vm1, Vm2,
Vm3 y Vm4
syms z;
estado(p,q,r,pitch,roll,yaw,u,v,w,z) = [p; q; r; pitch; roll; yaw; u; v; w; z];
senalControl(Vm1,Vm2,Vm3,Vm4) = [Vm1; Vm2; Vm3; Vm4];

funcionIncremento(p,q,r,pitch,roll,yaw,Vm1,Vm2,Vm3,Vm4,u,v,w,z) = [dOmegadt(Vm1,Vm2,Vm3,Vm4,p,q,r);
dYPRdt(yaw,roll,pitch,p,q,r); Aw(Vm1,Vm2,Vm3,Vm4,p,q,r,u,v,w); dZdt(roll,pitch,u,v,w)]

A_jacobiano_f_estado(p,q,r,pitch,roll,yaw,u,v,w,z) =
jacobian(funcionIncremento(p,q,r,pitch,roll,yaw,Vm1,Vm2,Vm3,Vm4,u,v,w,z), estado(p,q,r,pitch,roll,yaw,u,v,w,z))
B_jacobiano_f_senalControl(Vm1,Vm2,Vm3,Vm4) =
jacobian(funcionIncremento(p,q,r,pitch,roll,yaw,Vm1,Vm2,Vm3,Vm4,u,v,w,z), senalControl(Vm1,Vm2,Vm3,Vm4))



p_0 = 0;
q_0 = 0;
r_0 = 0;
pitch_0 = 0;
roll_0 = 0;
yaw_0 = 0;
u_0 = 0;
v_0 = 0;
w_0 = 0;
x_0 = 0;
y_0 = 0;
z_0 = 1;
numEstados = 10;
numEntradasControl = 4;



A = double(A_jacobiano_f_estado(p_0, q_0, r_0, pitch_0, roll_0, yaw_0, u_0, v_0, w_0, z_0))
B = double(B_jacobiano_f_senalControl(0,0,0,0))
C = eye(numEstados);
C(7,7) = 0;
C(8,8) = 0;
C
D = zeros(numEstados,numEntradasControl)

dt = 0.0001;



%Respuesta en lazo abierto a condiciones iniciales distintas de cero.
t = 0:0.01:2;
u = [zeros(size(t)); zeros(size(t)); zeros(size(t)); zeros(size(t))];
%u = [ones(size(t)); ones(size(t)); ones(size(t)); ones(size(t))];
```

```
x0 = [10; 15; 10; 10; 10; 10; 10; 10; 10; 10]; %Condiciones iniciales en formato [vPitch, vRoll, vYaw, aPitch, aRoll, aYaw, vX, vY, vZ].

sys = ss(A,B,C,D);

[y,t,x] = lsim(sys,u,t,x0);
plot(t,y)
axis([0 2 -20 20])
title('Respuesta en lazo abierto a condiciones iniciales distintas de cero.')
xlabel('Tiempo (s)')
ylabel('Posicion del cuadricoptero (m)')

matrizObservabilidad = obsv(sys);
estadosNoObservables = length(A) - rank(matrizObservabilidad)

matrizControlabilidad = ctrb(sys);
estadosNoControlables = length(A) - rank(matrizControlabilidad)

polos_A = eig(A)


tF = 20;
t = 0:dt:tF;
numIteraciones = round(tF/dt);
voltajeMaximoMotores = 11.1;
fE_vA = 0.01;
fE_pA = 0.02;
fE_vZ = 0.04;
fE_pZ = 0.05;


kP_pA = [1.0 1.0 1.0];
kI_pA = [0 0 0];
kD_pA = [0 0 0];
pA_deseada = [10 5 16];
error_pA = [0 0 0];
errorPrevio_pA = [0 0 0];
integral_pA = [0 0 0];
derivada_pA = [0 0 0];
pid_pA = [0 0 0];


kP_vA = [5.0 5.0 5.0];
kI_vA = [0 0 0];
kD_vA = [0 0 0];
vA_deseada = [0 0 0];
error_vA = [0 0 0];
errorPrevio_vA = [0 0 0];
integral_vA = [0 0 0];
derivada_vA = [0 0 0];
pid_vA = [0 0 0];
```

```
kP_pZ = 1;
kI_pZ = 0;
kD_pZ = 0;
pZ_deseada = 10;
error_pZ(1) = 0;
errorPrevio_pZ = 0;
integral_pZ = 0;
derivada_pZ = 0;
pid_pZ = 0;

kP_vZ = 3;
kI_vZ = 0;
kD_vZ = 0;
vZ_deseada = 0;
error_vZ(1) = 0;
errorPrevio_vZ = 0;
integral_vZ = 0;
derivada_vZ = 0;
pid_vZ = 0;

u = zeros(numEntradasControl, numIteraciones+1);
x_dot = zeros(numEstados, numIteraciones+1);
x = zeros(numEstados, numIteraciones+1);
x(1:numEstados,1) = [10 15 5 0 0 0 0 0 0 1];  %p, q, r, pitch, roll, yaw, u, v, w
y = zeros(numEstados, numIteraciones+1);

for i = 1:numIteraciones
    y(1:numEstados, i) = C*x(1:numEstados, i);
    vPitch = y(1, i);
    vRoll = y(2, i);
    vYaw = y(3, i);
    aPitch = y(4, i);
    aRoll = y(5, i);
    aYaw = y(6, i);
    vZ = y(9, i);
    pZ = y(10, i);

    if (mod(i*dt,fE_pA) == 0)
        error_pA(1) = pA_deseada(1) - aPitch;
        error_pA(2) = pA_deseada(2) - aRoll;
        error_pA(3) = pA_deseada(3) - aYaw;
        integral_pA = integral_pA + error_pA;
        derivada_pA = error_pA - errorPrevio_pA;
        errorPrevio_pA = error_pA;
        pid_pA(1) = kP_pA(1)*error_pA(1) + kI_pA(1)*integral_pA(1) + kD_pA(1)*derivada_pA(1);
        pid_pA(2) = kP_pA(2)*error_pA(2) + kI_pA(2)*integral_pA(2) + kD_pA(2)*derivada_pA(2);
        pid_pA(3) = kP_pA(3)*error_pA(3) + kI_pA(3)*integral_pA(3) + kD_pA(3)*derivada_pA(3);
    end
    if (mod(i*dt,fE_vA) == 0)
        error_vA(1) = pid_pA(1) - vPitch;
        error_vA(2) = pid_pA(2) - vRoll;
        error_vA(3) = pid_pA(3) - vYaw;
```

```
        integral_vA = integral_vA + error_vA;
        derivada_vA = error_vA - errorPrevio_vA;
        errorPrevio_vA = error_vA;
        pid_vA(1) = kP_vA(1)*error_vA(1) + kI_vA(1)*integral_vA(1) + kD_vA(1)*derivada_vA(1);
        pid_vA(2) = kP_vA(2)*error_vA(2) + kI_vA(2)*integral_vA(2) + kD_vA(2)*derivada_vA(2);
        pid_vA(3) = kP_vA(3)*error_vA(3) + kI_vA(3)*integral_vA(3) + kD_vA(3)*derivada_vA(3);
    end

    if (mod(i*dt,fE_pZ) == 0)
        error_pZ(i+1) = (pZ_deseada - pZ);
        integral_pZ = integral_pZ + error_pZ(i+1);
        derivada_pZ = error_pZ(i+1) - errorPrevio_pZ;
        errorPrevio_pZ = error_pZ(i+1);
        pid_pZ = kP_pZ*error_pZ(i+1) + kI_pZ*integral_pZ + kD_pZ*derivada_pZ;
    end

    if (mod(i*dt,fE_vZ) == 0)
        error_vZ(i+1) = -(pid_pZ - vZ);
        integral_vZ = integral_vZ + error_vZ(i+1);
        derivada_vZ = error_vZ(i+1) - errorPrevio_vZ;
        errorPrevio_vZ = error_vZ(i+1);
        pid_vZ = kP_vZ*error_vZ(i+1) + kI_vZ*integral_vZ + kD_vZ*derivada_vZ;
    end

    %pitch: u2 positivo, u4 negativo
    %roll: u1 positivo, u3 negativo
    %yaw: u2, u4 positivos; u1, u3 negativos
    u(2, i) = pid_vA(1) + pid_vA(3) + pid_vZ;
    u(4, i) = -pid_vA(1) + pid_vA(3) + pid_vZ;
    u(3, i) = -pid_vA(2) - pid_vA(3) + pid_vZ;
    u(1, i) = pid_vA(2) - pid_vA(3) + pid_vZ;
    if (u(1,i) > voltajeMaximoMotores)
        u(1,i) = voltajeMaximoMotores;
    end
    if (u(2,i) > voltajeMaximoMotores)
        u(2,i) = voltajeMaximoMotores;
    end
    if (u(3,i) > voltajeMaximoMotores)
        u(3,i) = voltajeMaximoMotores;
    end
    if (u(4,i) > voltajeMaximoMotores)
        u(4,i) = voltajeMaximoMotores;
    end

    x_dot(1:numEstados, i) = A*x(1:numEstados, i) + B*u(1:numEntradasControl, i);
    x(1:numEstados,i+1) = x(1:numEstados, i) + dt*x_dot(1:numEstados, i);
    if (x(10,i+1) < 0)
        x(10,i+1) = 0;
    end
end

figure()
subplot(2,2,1)
```

```
plot(t,x(1:3,:))
axis([0 tF -100 100])
xlabel('Tiempo (s)')
ylabel('Velocidad angular (grados/s)')
title('Velocidades angulares en Yaw, Pitch y Roll')
legend('Pitch','Roll','Yaw');
%figure()
subplot(2,2,2)
plot(t,x(4:6,:))
axis([0 tF -20 20])
xlabel('Tiempo (s)')
ylabel('Angulo (grados)')
title('Angulos de Yaw, Pitch y Roll')
legend('Pitch','Roll','Yaw');
%figure()
subplot(2,2,3)
plot(t,x(9,:))
axis([0 tF -20 20])
xlabel('Tiempo (s)')
ylabel('Velocidad (m/s)')
title('Velocidad lineal en Z')
legend('VelZ');
%figure()
subplot(2,2,4)
plot(t,x(10,:))
axis([0 tF -20 20])
xlabel('Tiempo (s)')
ylabel('Distancia (m)')
title('Altura (Z)')
legend('PosZ');
figure()
plot(t,u)
axis([0 tF -20 20])
xlabel('Tiempo (s)')
ylabel('Voltaje (V)')
title('Senales de control')
```

Apéndice M: Repositorio del proyecto

Como parte del desarrollo del presente Trabajo de Grado, se cargaron todos los archivos de código, documentación, pruebas y referencias en un repositorio web, bajo la licencia MIT. Con ello, se busca alentar el desarrollo de una plataforma para cuadricópteros de bajo costo basada en plataformas de hardware y software libre que pueda ser reproducida con componentes disponibles en el mercado venezolano.

La dirección del repositorio del proyecto es:

http://www.github.com/yoshuanava/tesiscuadricoptero

Glosario de términos

- **Ángulos de Euler**: método para la representación de la orientación 3D de un objeto mediante tres ángulos de rotación -Roll, Pitch y Yaw- alrededor de tres ejes distintos. [1]
- **Controlabilidad**: se dice que un sistema es controlable en el tiempo si se puede transferir desde cualquier estado inicial a cualquier otro estado, mediante un vector de control sin restricciones, en un intervalo de tiempo finito. [2]
- **Entrada de referencia**: valor deseado de las variables a controlar de un sistema. [2]
- **Filtro pasa-altos**: filtro que deja pasar las pulsaciones o frecuencias por encima de una frecuencia dada. [3]
- **Filtro pasa-bajos**: filtro que deja pasar las pulsaciones o frecuencias por debajo de una frecuencia dada. [3]
- **Matriz de transición de estados**: matriz que contiene toda la información sobre la evolución de las trayectorias del vector de estado del sistema. [4]
- **Observabilidad**: se dice que un sistema es observable en el tiempo si, al encontrarse el sistema en un estado determinado, es posible determinar dicho estado a partir de la observación de la salida durante un intervalo de tiempo finito. [2]
- **Perturbación**: señal que tiende a afectar negativamente el valor de la salida de un sistema. Si la perturbación se genera dentro del sistema se denomina *interna*, mientras que una perturbación *externa* se genera fuera del sistema y es una entrada. [2]
- **Pulse Width Modulation (PWM)**: técnica que se basa en la modificación del ciclo de trabajo de una señal periódica, ya sea para transmitir información a través de un canal de comunicaciones o para controlar la cantidad de energía que se envía a una carga. [5]
- **Transformada Rápida de Fourier**: método muy eficiente para determinar el espectro en frecuencia de una señal. Permite transformar una secuencia de valores en el dominio del tiempo a una secuencia de valores equivalente en el dominio de la frecuencia. [6]
- **Variables de estado**: menor conjunto de variables que determinan el estado del sistema dinámico. [2]

---

[1] CHRobotics. *Understanding Euler angles.* Fuente: http://www.chrobotics.com/library/understanding-euler-angles

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

[3] *Análisis y síntesis de circuitos*. Departamento de Teoría de la Señal y Comunicaciones de la Universidad de Alcalá de Henares. Fuente: http://agamenon.tsc.uah.es/Asignaturas/ittse/asc/apuntes/TEMA1.pdf

[4] Barragán, A. (2011). *Matriz de transición de estado y respuesta temporal*. Fuente: http://uhu.es/antonio.barragan/content/24-matriz-transicion-estado-y-respuesta-temporal

[5] García Gago, S. *Sistemas de Modulación.* UNESCO. Fuente: http://www.analfatecnicos.net/archivos/15.SistemasModulacionWikipedia.pdf

[6] Posadas Yagüe, J. (1998). *Transformada rápida de Fourier (FFT) e interpolación en tiempo real.* Universidad Politécnica de Valencia. Fuente: http://www14.brinkster.com/aleatoriedad/FFT.pdf