Capítulo VII: Apéndices y anexos

1. **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:



Ilustracion 4. 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:

Ilustracion 4. 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:

Ilustracion 4. 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 el rango de valores entre 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:

****

Ilustracion 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%:



Ilustracion 4. 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:



Ilustracion 4. 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:



Ilustracion 4. Diagrama de circuito de regulación de velocidad diseñado.

Fuente: elaboración propia.

Finalmente, se diseñó y construyó una placa de circuito impreso de una sóla 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:



Ilustracion 4. Diseño de circuito impreso del circuito de alimentación y control de motores.

Fuente: elaboración propia.

1. **Apéndice B: Descripción del modelo de estimación de posición angular y velocidad angular del cuadricóptero**
   1. **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:

siendo:

* 1. **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:



Ilustracion 4. Modelo de estimación de angulos

Fuente: [STMicroElectronics]

siendo:

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, en base a 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.

* 1. **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:

donde:

*: 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 deriva constante y a alejarse de los valores reales a medir, por efecto de vibraciones mecánicas, cambios de temperatura y errores numéricos.

* 1. **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]:



Ilustracion 4. Modelo del Filtro complementario

Fuente: [Gaydou 2007]

En donde:

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

Donde:

constante de tiempo del filtro.

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 , 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, más no a brindar posibilidades de realizar vuelo acrobático.

**Gráfica que demuestre precisión y estabilidad del ángulo calculado con el filtro complementario.**

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

Se recolectaron datos del giroscopio y del acelerómetro, 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 mecánica, 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..

Al someter los datos recopilados a un análisis de respuesta en 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 0 a 20 Hz, y no poseían modas definidas, por lo cual no se podía rechazar ninguna banda en particular.

**Gráfica que demuestre el ruido y espectro de frecuencias sin la base, en el acelerómetro y giroscopio. Motores apagados.**

**Gráfica que demuestre el ruido y espectro de frecuencias sin la base, en el acelerómetro y giroscopio. Motores encendidos**

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 coloco 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 la influencia de las vibraciones sobre el sensor.

**Diagrama que ilustre cómo fue el montaje final de la IMU.**

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

**Gráfica que demuestre el ruido y espectro de frecuencias con la base, en el acelerómetro y giroscopio.**

Nuevamente se recogieron datos del cuadricóptero con los motores encendidos, y se realizó un análisis de respuesta en 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. Para mantener una frecuencia efectiva de muestreo suficientemente alta como para aplicar control al cuadricóptero, se configuró la Unidad de Medición Inercial como se expone en la siguiente tabla:

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

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.

**Gráfica que demuestre el ruido y espectro de frecuencias con la base, y el filtro de medias móviles, en el acelerómetro y giroscopio.**

En la tabla que se presenta a continuación se presentan los valores de media y desviación estándar de la señal, antes y después de implementar los filtros:

**Tabla de valores estadísticos de la señal en presencia de vibraciones, sin filtro, y con filtro. Valores estadísticos: desviación estándar, frecuencia efectiva de muestreo, media**

1. **Apéndice C: Descripción del modelo de estimación de posición y velocidad lineal en el eje z del cuadricóptero**
   1. **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 distancia a partir de las mediciones del sensor ultrasónico se realiza a partir del tiempo de retorno de los pulsos emitidos, como se expone a continuación:

donde:

velocidad del sonido, constante, de valor 0,0343 cm/µs.

: tiempo de retorno del pulso, medido en microsegundos (µs).

Se modeló el cálculo de altura del cuadricóptero en vuelo, estando este sujeto a inclinaciones en los ejes de Pitch, Roll y Yaw. Se partió de que el sensor ultrasónico provee una medida de la extensión de un segmento de recta perpendicular al cuadricóptero en todo caso, que corta al plano descrito por el suelo en un ángulo entre cero (0) y ciento ochenta (180) grados, no siempre representando ésta la distancia efectiva del cuadricóptero respecto al suelo. Mediante álgebra vectorial se derivaron ecuaciones para obtener la longitud del segmento de recta perpendicular al suelo que intersecta el plano descrito por el cuadricóptero. Las ecuaciones obtenidas se exponen a continuación:

=

La velocidad lineal en el eje Z del cuadricóptero se cálculo mediante integración de la señal de aceleración del cuadricóptero en el eje Z. Ello requirió del cálculo de la inclinación del cuadricóptero en tiempo real, y de la supresión de la señal de aceleración de la gravedad en cada eje del mismo. A continuación se describen las ecuaciones que se derivaron para llevar a cabo el cálculo de las aceleraciones lineales independientes de la gravedad, y las velocidades lineales en el eje Z:

donde:

* : componente del vector gravedad en el eje z del acelerómetro.
* : señal del acelerómetro en el eje z.
* : aceleración lineal en el eje z no producida por la gravedad.
* : velocidad lineal en el eje z.
* : diferencial de tiempo.

1. **Caracterización y respuesta en frecuencia del sensor de altura utilizado**

Se sometió el cuadricóptero a distintos ángulos de inclinación para medir la robustez del algoritmo de estimación de altura desarrollado. En la tabla que se presenta a continuación se presentan los parámetros del mismo:

**Tabla de valores estadísticos de la estimación de altura función de los ángulos de inclinación en Pitch y Roll.**

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 éste 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.

**Gráfico de respuesta en frecuencia del sensor**

**Gráfico de las mediciones del sensor ultrasónico sin filtro y con filtro**

1. **Apéndice D: Modelo dinámico del cuadricóptero**
   1. **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:

Las ecuaciones que describen la dinámica del cuadricóptero son:

* .
* .
* .
* .

Así 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:

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

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



Ilustración 4.23: Diagrama simplificado del modelo dinámico del cuadricóptero Draganflyer V.

Fuente: [Kivrak 2006]

* 1. **Cálculo de los parámetros de inercia del modelo**

Para el estudio de inercia del modelo del cuadricóptero y 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 y densidad uniforme.

**Para realizar el cálculo de inercia se hicieron las siguientes medidas:**

|  |  |
| --- | --- |
| **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 |
| Altura de cabina | 0.1 |
| Masa de motor | 0.051 Kg |
| Masa de cabina | 071Kg |

Aprovechando la simetría de la estructura el resultado de la inercia para el eje roll es igual para el del eje pitch.

Las formulas tomadas de [Sinha 2012] para el calulo de la inercia de los motores y Las formulas del [Hibbeler 2012] para el cálculo de inercia de la cabina:

Pitch y Roll:

**Motores sobre el eje:**

**Motores fuera del eje:**

**Cabina:**

**Inercia total:**

Ademas calculó la inercia sobre el eje yaw donde se aprovechó que la inercia que ejerce cada motor sobre el eje Z es igual:

**Inercia de motores:**

**Inercia de cabina:**

**Inercia total:**

Como resultado de estos calculos se obtiene la siguiente matriz que representa la inercia ejercida por el modelo del cuadricóptero**:**

Donde:

* 1. **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 . Luego:

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 , mediante el cálculo de su matriz Jacobiana respecto al vector de estados y al vector de entradas de control Las matrices obtenidas se evaluaron en un punto de equilibrio , para obtener una representación del sistema de la forma , donde es una matriz que describe el comportamiento del sistema alrededor de un punto de equilibrio , y es una matriz que describe el comportamiento del sistema ante perturbaciones.

Se seleccionó el vector , 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:

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

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

Se modeló la estimación de estado del cuadricóptero como un sistema de ecuaciones diferenciales lineales de la forma , 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:

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

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

Se obtuvo que el rango de la matriz 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.

1. **Apéndice E: Código de estimación, control y comunicación a bordo del cuadricóptero**
2. **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

1. **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)

1. **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()

1. **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()

1. **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()

1. **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

1. **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

1. **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')

1. **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.

La dirección del repositorio del proyecto es:

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