Universidad Zaragoza
1542

Trabajo Fin de Grado

Diseño e implementación de un sistema de control para un quadcopter

Design and Implementation of a control sytem to a quadcopter

Autor/es

Rubén Abad Torrén

Director/es

José Luis Villroel Salcedo

EINA

2015/2016

RESUMEN

En el siguiente documento se aborda el desarrollo e implementación del sistema de control de una aeronave no tripulada del tipo quadcoptero (o dron), también se estudiaran e implementaran los algoritmos de sensado de la orientación, así como los procesos de fusión sensorial para la navegación.

El control deberá ser capaz de lograr que la aeronave obtenga la orientación de forma estable en un tiempo de respuesta menor a un segundo, así como de variar el empuje perpendicular aportado, para ello se sensa la posición y se actúa en cada propulsor.

Para lograr dicho propósito se modelara el sistema mediante ecuaciones diferenciales, para posteriormente identificar las constantes, una vez obtenido el modelo, se calcularan los parámetros del control elegido.

También se implementan las tareas y procesos necesarios para la operación de la aeronave, la comunicación con la emisora y la emisión de la telemetría.

La ejecución simultánea de las tareas correrá sobre un sistema en tiempo real, asi como la gestión de los periféricos.

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Ruben Abad Torren,

con nº de DNI 73205858-D en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado en Ingeniería Electrónica y Automática, (Título del Trabajo) Diseño e implementación de un sistema de control para un quadcopter

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 24 de Junio de 2016

R. Abad

Fdo: Rubén Abad Torrén

ÍNDICE DE CONTENIDOS

1. INTRODUCCIÓN

1.1. ESTADO DEL ARTE

El campo de las aeronaves no tripuladas ha experimentado un crecimiento exponencial, se ha pasado de proyectos militares de millones de dólares a juguetes de apenas 10€.

Hace unos años únicamente fuerzas militares disponían de “drones”, cuya finalidad consistía en el espionaje y bombardeo táctico, evitando la posible captura de un piloto por el derribo de la aeronave.

Hoy en día, su uso está mucho más extendido, si bien siguen usándose los drones militares, el desarrollo de drones para usos civiles está en pleno auge, dándose el caso de que muchos países han debido legislar este tipo de aparato, así como su utilización.

Si bien existe otra filosofía de “drone”, basada en helicóptero, con la capacidad de mantenerse estático en un punto del espacio, es este tipo de aeronave la más usada tano por profesionales, como por el aficionado, tareas como la fotografía o captura de video pueden hacerse ahora desde planos aéreos, así como la inspección de edificios y pequeñas áreas de terreno.

Ante este auge, existen en el mercado distintos fabricantes, que proporcionan desde el sistema al completo, hasta las distintas partes mecánicas, así como el control, del mismo modo existen también proyectos de código abierto con tal fin.

1.2. REQUISITOS

- El control deberá ser capaz de lograr que la aeronave obtenga una orientación de forma estable en un tiempo de respuesta menor a un segundo, así como de variar el empuje aportado.
- Sensado de la ubicación y orientación de la aeronave, para la realimentación del control.
- Implementan las tareas y procesos necesarios para el pilotaje de la aeronave, la comunicación con la emisora y la emisión de la telemetría.
- Coste: requisito común a todos los proyectos, se debe mantener el coste lo más bajo posible, sin que ello implique la pérdida de funcionalidades.
- Compacto y ligero: La naturaleza del sistema impone que el control ni puede ser pesado, lo que dispara el consumo de los motores para mantener el vuelo, ni voluminoso. Lo que disminuye la aerodinámica del dron.
- Modular: un diseño que permita añadir y quitar funcionalidades, lo que permite exportar el diseño a proyectos similares

2. ESTRUCTURA Y FUNCIONAMIENTO

2.1. TEORIA DE FUNCIONAMIENTO

El principio de funcionamiento de la aeronave se basa en variar el empuje de cada propulsor. Ubicando los ejes de referencia, tal y como muestra la Figura 1 se obtiene las siguientes aceleraciones angulares:

- La diferencia de empuje entre Ω0 y Ω2 produce una aceleración angular sobre el eje YB.
- La diferencia de empuje ente Ω1 y Ω3 produce una aceleración angular sobre el eje Xb.
- La diferencia de empuje de Ω0 mas Ω2 entre Ω1 más Ω3 produce una aceleración angular sobre el eje Zb.

Estas fuerzas son producidas por dos fenómenos. Para el empuje vertical (flechas de la Figura 1), se usa el empuje producido por la rotación de la hélice en el fluido que es el aire. Dado que al rotar, el motor genera un par contrario a la rotación de la hélice, (Tercera ley de Newton o Principio de Acción y Reacción), por este motivo se genera un momento en el eje Zb, que proporciona la rotación en dicho eje, siendo necesario que haya el mismo número de motores rotando en un sentido que en el otro, para mantener el equilibrio.

Es decir, la diferencia de velocidad de rotación de entre los motores situados en los extremos de cada brazo del mismo eje produce rotación en dicho eje (X e Y), la diferencia de velocidad (Teniendo en cuenta el signo) produce la rotación en torno al eje Z.

El empuje total se corresponde con la suma del empuje individual de cada motor.

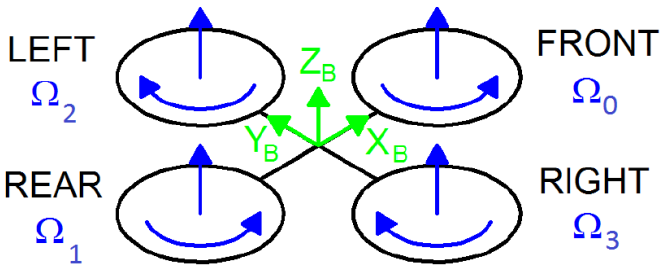


Figura 1. Ejes de referencia, sentido de giro de los motores y empujes

2.2. ESTRUCTURA & ACTUADORES

La elección de la estructura no es un aspecto crítico del sistema, una vez escogido el tamaño aproximado de la aeronave, así como su número de impulsores, debiendo tenerse en cuenta como parámetros para su elección el peso y la rigidez. Se opta por una estructura “Q450”, que posee un tamaño estándar de 45cm entre ejes, 4 motores, la rigidez y un peso apropiado

La tecnología de los motores es BLDC, dado la potencia que son capaces de desarrollar aun para su reducido peso y tamaño, un motor BLDC tiene un diseño similar a un motor trifásico, si bien la conmutación de corriente por los devanados se realiza de forma electrónica, amén de ser siempre positiva.

Puesto que los motores escogidos son del tipo BLDC, por ser los adecuados para este propósito se requiere de un ESC (Electronic Speed Controller), que básicamente es un inversor trifásico definido positivo. Todos tienen un comportamiento similar, por lo que únicamente se requiere que sean capaces de aportar la corriente necesaria, y que el *firmware* se “Simonk”. Dado que tiene una tasa de muestreo interna de hasta 400Hz. Para comandar el ESC, se le aplica una señal de mínimo 1ms (Potencia cero), a 2ms (Potencia máxima). Siendo estas las entradas del sistema.

Como se desea un comportamiento pausado en lugar de uno agresivo (usados en acrobacias), se escogen motores de bajo Kv, y hélices grandes, aptas para “bajas” rpm. Así se escogen motores Sunnysky x2212 KV980, hélices de 1045 (10” de radio, 4.5” de paso”) y una batería de 3 celdas con una capacidad de 2200mAh.

3. DISEÑO ARQUITECTUAL

El sistema (como se describió anteriormente) consta de una estructura en forma de cruz con un actuador en el extremo de cada eje, por medio de un ESC usando un PWM se controla el actuador.

Fijados al chasis se encuentran los sensores necesarios para medir la orientación de la aeronave, así como su velocidad angular, ambas usadas para la realimentación del control, la comunicación entre el control y los sensores se realiza vía I2C.

El controlador dispondrá de un radio enlace vía UART de tecnología Bluetooth, para la emisión de la telemetría, de ser necesario aumentar la distancia podría cambiarse la tecnología de comunicación siempre que use protocolo UART.

El control dispone de un microprocesador auxiliar, este procesador lee los PWM del receptor de Radio de la Emisora trasmitiéndolos al procesador principal vía I2C, sin embargo su función principal reside en desactivar el procesador principal, como función de seguridad. Para dicho propósito activa el pin de reset.

El enlace para pilotar el quadcopero corre a cargo de la emisora Turgnigy 9XR Pro, con 8 canales de datos, emitiendo sobre la banda de 2.4Ghz, garantizando la conexión a más de un kilómetro.

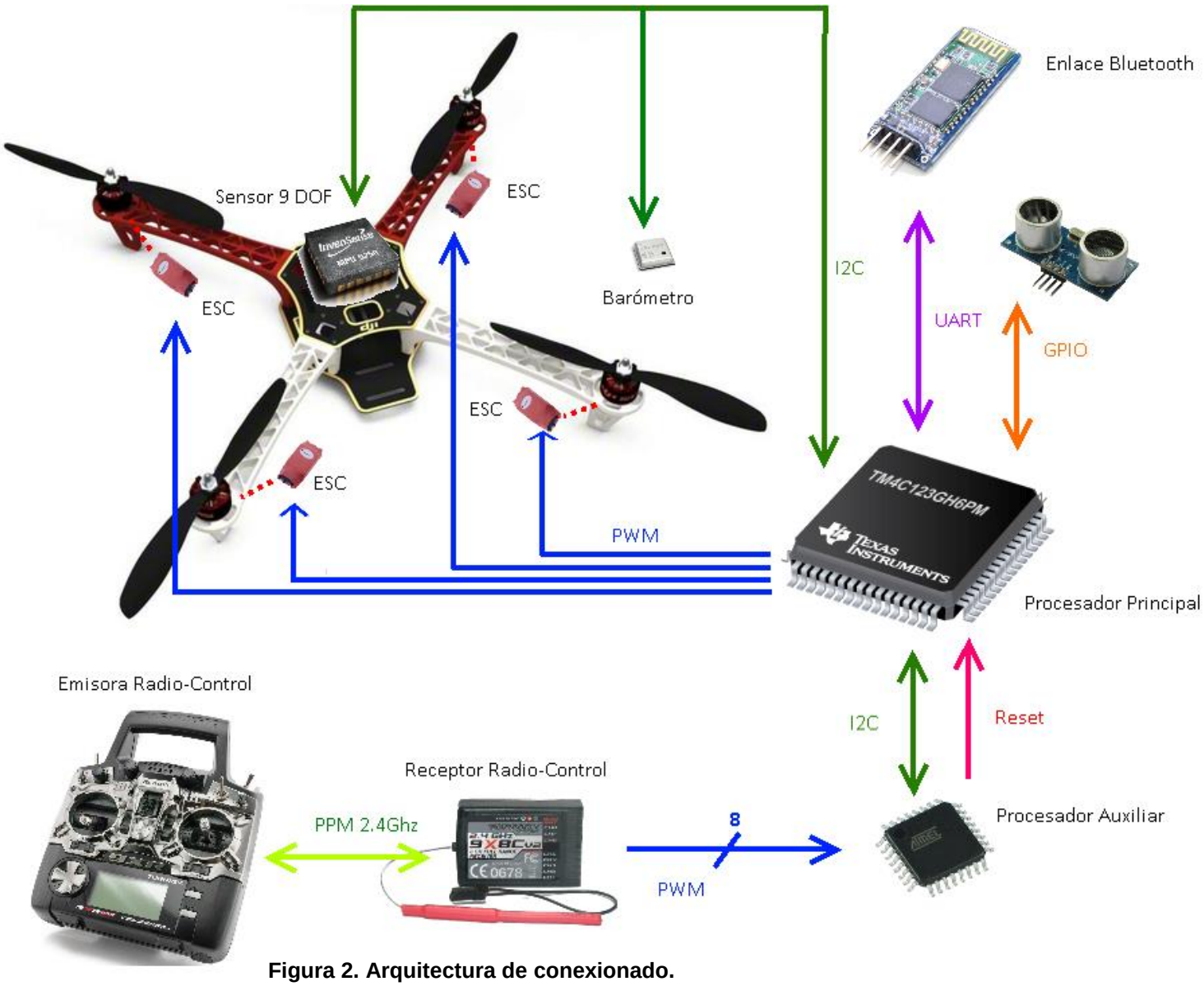


Figura 2. Arquitectura de conexionado.

4. CONTROL

En primer lugar se procederá al modelado matemático del conjunto, tras lo cual se procederá a la identificación de las constantes de las ecuaciones de modelado.

4.1. MODELADO MATEMATICO DEL CONJUNTO

Para el modelado matemático, el modelo se dividirá en dos subconjuntos, el motor pese a ser un BLDC, puede modelarse como un motor de corriente continua clásico[1], que se puede modelar como un sistema de primer orden, (si se desprecia la inductancia del bobinado).

El comportamiento dinámico del chasis puede modelarse mediante las ecuaciones de Newton-Euler.

4.1.1. **Modelado del motor**

Ecuación del motor en el campo transformado de Laplace, en la forma estándar:

$\frac{V_{rpm}}{Accion} = \frac{K_m{}'}{1+\tau_{m*s}}$, Siendo τ_m el retraso característico del motor.

Como el empuje del motor es cuadrático con la velocidad de rotación, será necesaria una linealización:

$$F_{motor} = rpm \times K_\omega = \frac{K_\omega \times K_m{}'}{1+\tau_m} = \frac{K_m}{1+\tau_m}$$

Siendo K_ω la proporción de velocidad y el empuje en Newton.

4.1.2. **Identificación del motor.**

La identificación del motor consta de dos apartados, el primero donde analizaremos la respuesta dinámica, midiendo la velocidad del motor, analizando el tiempo que le cuesta alcanzar el permanente con el cambio de entrada. Esta identificación puede obviarse porque al analizar el sistema completo el polo más rápido será el correspondiente al motor.

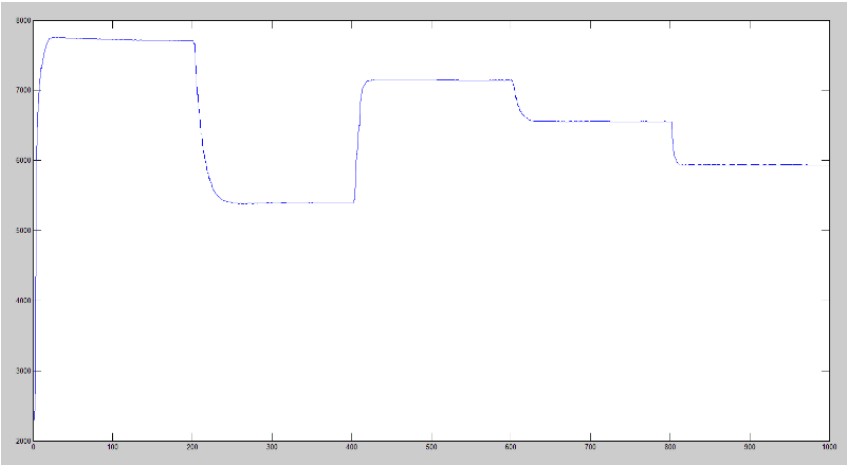
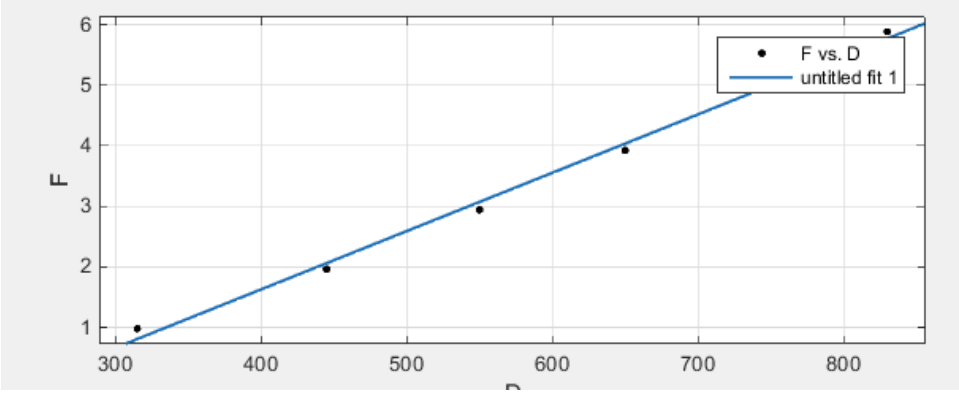


Figura 3. Evolución de las rpm del motor a distintas acciones.

[1] AN857 “Brushless DC Motor Control Made Easy” de Microchip

La siguiente identificación es la ganancia del motor K_m, la relación de $\frac{V_{rpm}}{Accion}$, se procede a medir la fuerza ejercida por el motor a diferentes valores de entrada, obtenido los siguientes resultados:



Goodness of fit:
SSE: 0.0848
R-square: 0.995
Adjusted R-square: 0.9937
RMSE: 0.1456

Figura 4. Aproximación lineal de la relación de F / Accion

La TF sistema tiene un comportamiento prácticamente lineal, con un valor de correlación R de 0.99.

4.1.3.**Modelado de la estructura**

A partir de las ecuaciones de Newton-Euler:

$$\left[\ \boldsymbol{I}\ \right]\left[\dot{\boldsymbol{\omega}}^{B}\right]+\left[\boldsymbol{\omega}^{B}\times\left(\boldsymbol{I}\ \boldsymbol{\omega}^{B}\right)\right]=\left[\boldsymbol{\tau}^{B}\right]$$

Siendo Ƭᵇ el momento angular, al ser proporcional a la fuerza, se modela el sistema en función de la fuerza, modificándose la ganancia del sistema, añadir un rozamiento viscoso y linealizar el sistema, la ecuación de rotación de un eje es:

$$F\ \times L-f\times\dot{\theta}=I\ \times\ \ddot{\theta}$$

En modelado de Laplace:

$$\Theta\ =\frac{Kq}{s(s+\boldsymbol{\tau}_q)}$$

4.1.4.**Modelado del sistema completo**

Multiplicando las dos ecuaciones anteriores, normalizando la función obtenida queda:

$$\Theta\ =\frac{Km*Kq}{s(s+\boldsymbol{\tau}_m)(s+\boldsymbol{\tau}_q)}$$

Donde $\boldsymbol{\tau}_m$ es el polo correspondiente al motor, y $\boldsymbol{\tau}_q$ es el polo correspondiente a la rotación de la estructura.

Y el modelado del sistema mediante variables de estado:

$$\begin{bmatrix} \ddot{\theta}_x \\ \dot{\theta}_x \\ \ddot{\theta}_y \\ \dot{\theta}_y \\ \ddot{\theta}_z \\ \dot{\theta}_z \\ \Delta\dot{F}1 \\ \Delta\dot{F}2 \\ \Delta\dot{F}3 \\ \dot{F}_{total} \end{bmatrix} = \begin{bmatrix} \frac{-1}{Jq1}, 0, 0, 0, 0, 0, \frac{Kq1}{Jq1}, 0, 0, 0, \\ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 \\ 0, 0, \frac{-1}{Jq2}, 0, 0, 0, 0, \frac{Kq1}{Jq2}, 0, 0 \\ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0 \\ 0, 0, 0, 0, \frac{-1}{Jq3}, 0, 0, 0, \frac{Kq3}{Jq3}, 0 \\ 0, 0, 0, 0, 0, 0, \frac{-1}{Jm}, 0, 0, 0 \\ 0, 0, 0, 0, 0, 0, 0, \frac{-1}{Jm}, 0, 0 \\ 0, 0, 0, 0, 0, 0, 0, 0, \frac{-1}{Jm}, 0 \\ 0, 0, 0, 0, 0, 0, 0, 0, 0, \frac{-1}{Jm} \end{bmatrix} \times \begin{bmatrix} \dot{\theta}_x \\ \theta_x \\ \dot{\theta}_y \\ \theta_y \\ \dot{\theta}_z \\ \theta_z \\ \Delta F1 \\ \Delta F2 \\ \Delta F3 \\ F_{total} \end{bmatrix} + \begin{bmatrix} 0, 0, 0, 0 \\ 0, 0, 0, 0 \\ 0, 0, 0, 0 \\ 0, 0, 0, 0 \\ 0, 0, 0, 0 \\ 0, 0, 0, 0 \\ \frac{Km}{Jm}, 0, \frac{-Km}{Jm}, 0 \\ 0, \frac{Km}{Jm}, 0, \frac{-Km}{Jm} \\ \frac{Km}{Jm}, \frac{-Km}{Jm}, \frac{Km}{Jm}, \frac{-Km}{Jm} \\ \frac{Km}{Jm}, \frac{Km}{Jm}, \frac{Km}{Jm}, \frac{Km}{Jm} \end{bmatrix} \times \begin{bmatrix} U1 \\ U2 \\ U3 \\ U4 \end{bmatrix}$$

4.1.5. Identificación del sistema completo

Para la identificación del sistema se ancla el chasis a una estructura metálica que permite la libre rotación en torno a un único eje.

Para ello se generan señales aleatorias cuadradas con un determinado margen espectral, se han generado varios espectros para buscar la mejor respuesta y varias señales dentro de cada espectro para comprobar que dan resultados similares.

El propio algoritmo arranca la tarea de identificación de la aeronave, recepciona y almacena la respuesta emitida por la aeronave. Con esto obtenemos el vector de entrada al sistema, y el vector de salida. Introduciendo los datos en la herramienta obtenemos la siguiente respuesta

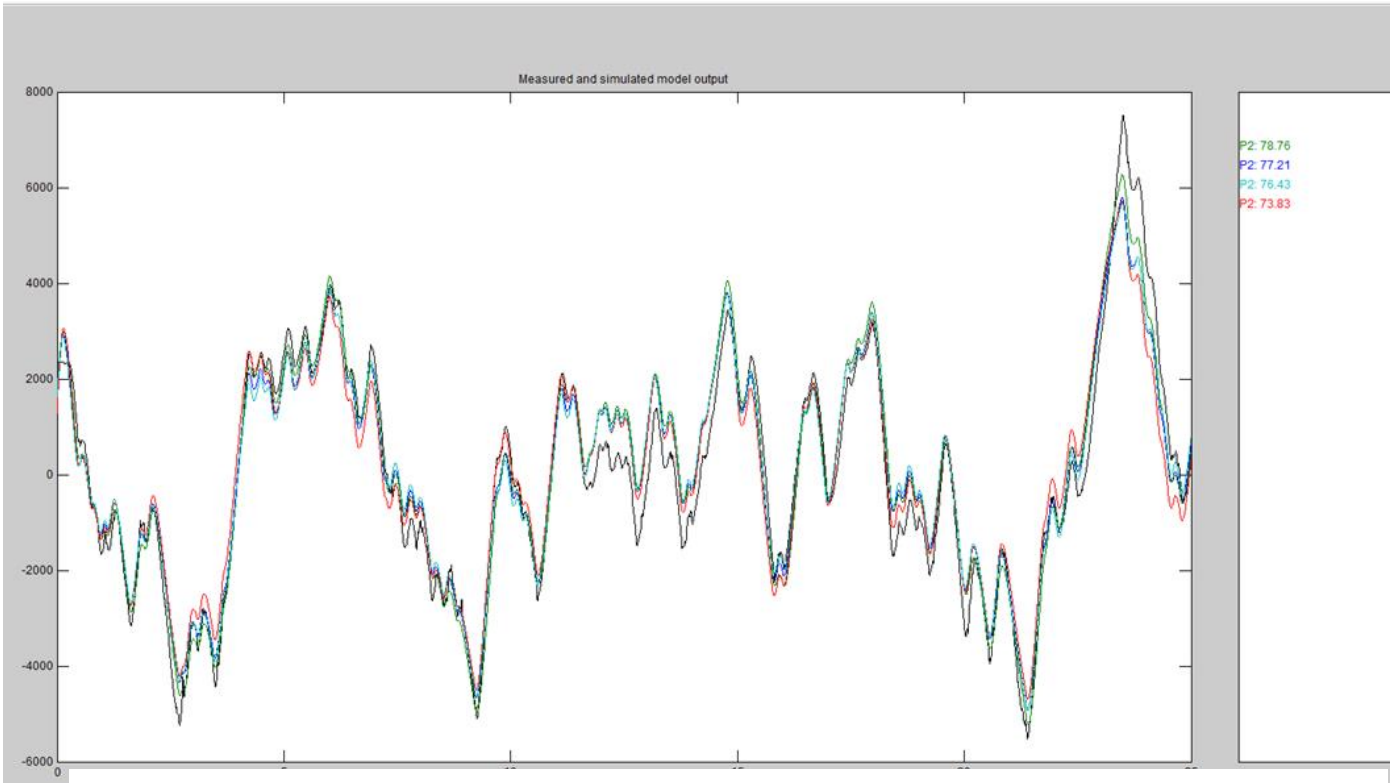


Figura 5. Respuesta sistema real e identificados, con valores identificacos cercanos al 80%

Resultando la ecuación global entorno a un eje:

$$\Theta = \frac{0.009637 * 1.89310976}{s(0.09639 * s + 1)(0.8613 * s + 1)}$$

4.2. CONTROL

4.2.1. Técnicas de Control

Pueden abarcarse dos puntos de vista para la implementación del control, al ser un sistema MIMO, (Multiple Inputs Multiple Outputs), puede optarse por implementar un control por eje, con bucles de control independientes, o modelar un control que contemple todos los bucles.

4.2.1.1. Modelado por controles independientes

Con la trasformada de la función de transferencia al plano Z, la ecuación pasa a ser un conjunto de sumas y restas de las entradas y salidas actuales y pasadas multiplicadas por unas constantes.

Disponen de una única entrada así como de una única salida, por lo que no es el idóneo para sistemas MIMO (Multiple Inputs Multiple Outpus), si bien siempre puede aplicarse el teorema de la superposición.

4.2.1.2. Control en variables de estado

Se basa en la conversión de una ecuación diferencial de orden superior o conjunto de ellas, (admite sistemas MIMO) en un sistema de ecuaciones diferenciales de primer orden.

Estas ecuaciones pueden escribirse en forma matricial, en función a un vector de las variables del sistema, llamado variables de estado, por estar compuesto por las variables del sistema que determinan su estado (tales como posición o velocidad).

Los polos del sistema (correspondientes a las soluciones de las ecuaciones diferenciales) pueden calcularse como los valores propios de la ecuación, lo que proporciona una potente herramienta matemática para el estudio de control. Al estar todo el sistema descrito en forma de matrices facilita en gran manera la simulación del sistema, permitiendo un incorporar elementos como estimador de estado y perturbaciones.

Como el sistema es MIMO, y se desea incorporar funciones como el estimador de perturbaciones, esta será la técnica escogida, además, esta implementación permite conocer el estado del sistema de todo momento, ya que las variables que lo definen están almacenadas en el vector.

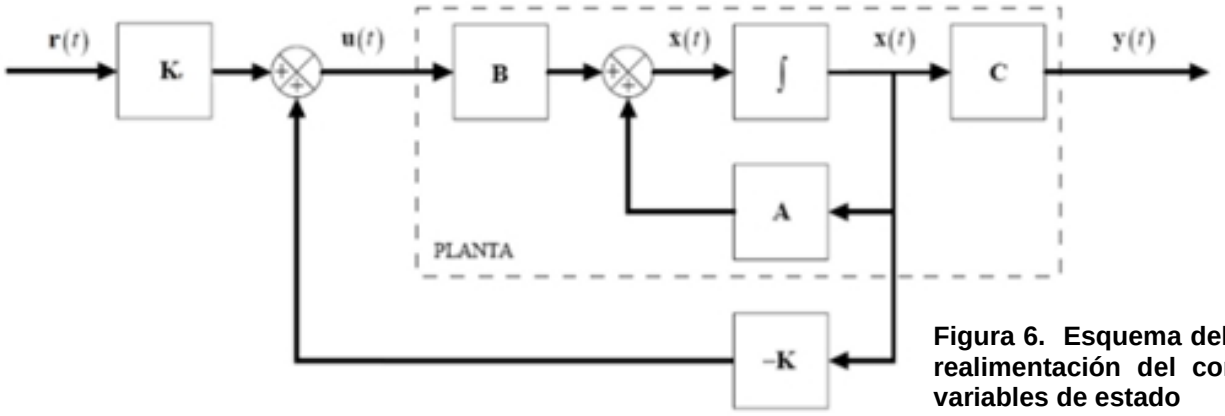


Figura 6. Esquema del lazo de realimentación del control en variables de estado

4.2.2.**Topología de control utilizada**

4.2.2.1. **Estimador de estado y perturbaciones**

En lugar la técnica de la integración del error para la corrección de las perturbaciones, así como garantizar que el permanente alcanza la referencia, se opta por implementar un estimador de perturbaciones. El estimador de perturbaciones se basa en comparar la medida real del sistema, con la medida que ha simulado en función a las ecuaciones que definen el sistema, con esta comparación, es capaz de calcular la magnitud de la perturbación para posteriormente corregirla.

Esta técnica evita los principales problemas típicos de la acción integral, como la saturación de los actuadores sin previo aviso, o el wind-up, además de que la estimación es más rápida que la integración del error.

Concretamente en un sistema de este tipo resulta interesante saber el estado de cada motor (como la perdida de empuje), así como el punto de trabajo en el que está, dado que la saturación conlleva a la pérdida del control, y en este caso muy posiblemente a colisionar la aeronave con el suelo. Conociendo la perturbación de cada motor el sistema podría avisar de una inminente perdida de un motor.

Otra función del estimador de perturbaciones es corregir las no linealidades del modelo, que estimará como perturbaciones.

Esta técnica precisa de un modelado del sistema muy preciso, de lo contrario el comportamiento del sistema puede interpretarse como perturbaciones, descontrolando el sistema.

De forma idéntica al estimador de perturbaciones, mientras el sistema sea observable se pueden estimar variables de estado, esto permite aplicar una filtración a la señal medida, (no usado en este proyecto), o conocer el valor de aquellas variables de estado que no son medidas, (en este caso el empuje de los motores)

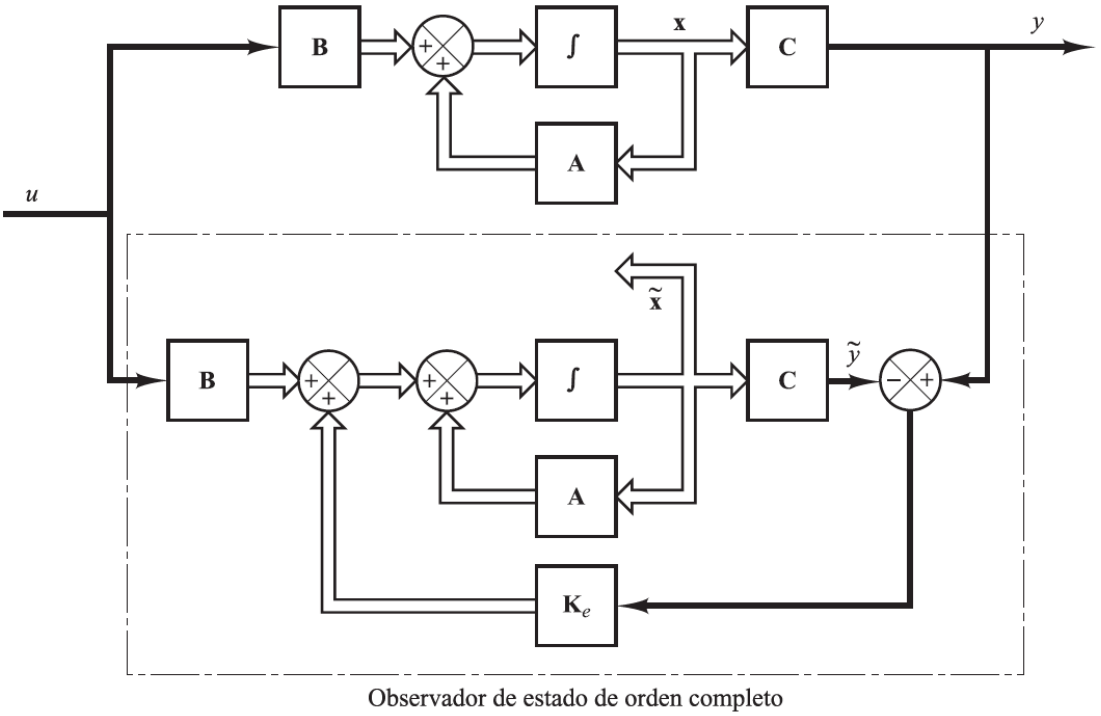


Figura 7. Esquema de control con estimador de estados

4.2.2.2. **Integrador de perturbaciones**

Como alternativa al estimador de perturbaciones, se ha implementado la acción integral para corregir perturbaciones, similar a la clásica implementación, en lugar de integrar la acción, se integra una variable intermedia, de esta manera, si bien el comportamiento idéntico, de esta manera es posible conocer el valor de la perturbación que se está corrigiendo.

Tiene las desventajas de un menor tiempo de respuesta, y los problemas asociados al integral, y la ventaja de que su sencillo funcionamiento es independiente de otro parámetro que no sea el error medido. Se contempla su uso en el caso de que el modelo nos sea lo bastante preciso como para que estimador de perturbaciones funcione correctamente.

4.2.2.3. **Pre alimentador de perturbaciones**

Las perturbaciones que pueden ser calculadas y predichas, tales como el peso de la batería el salir de la posición de equilibrio, los efectos giroscópicos (no contemplados en este proyecto)…, se prealimentarán, haciendo su impacto nulo, y no siendo necesario estimarlas o integrarlas.

4.2.3.**Lazo de realimentación**

Se podría usar Matlab para calcular la matriz de realimentación para ubicar los polos en las posiciones deseadas (en función de unos parámetros de respuesta deseados), una opción perfectamente válida para sistemas más simples, con parámetros mejor definidos y salidas más estables.

Dada la naturaleza de la planta del sistema a controlar, se opta por ubicar los polos de una forma manual, ya que la matriz de realimentación obtenida de Matlab realimenta todos los estados, cosa no idónea en este sistema, al no saber la exactitud del modelo y el difícil ajuste a mano que supone.

Usando una estructura más simple, el ServoControl podemos ubicar los polos con únicamente dos lazos de realimentación, dependientes cada uno de una única variable de estado, (Realmente podrían usarse tres, para acelerar la respuesta del motor).

Esta topología si permite un fácil ajuste a mano por tener únicamente dos parámetros a variar, con comportamientos radicalmente distintos, la ganancia proporcional aumenta la acción aplicada en función del error, la ganancia de la velocidad reduce a la acción en función de la velocidad del sistema. Básicamente, la ganancia de la velocidad se opone a todo movimiento, y la proporcional encara al sistema hacia la referencia. Esto permite respuestas rápidas sin oscilaciones, ya que al desaparecer la acción integral con la reducción del error, el sistema se frena por la acción de la ganancia de la velocidad.

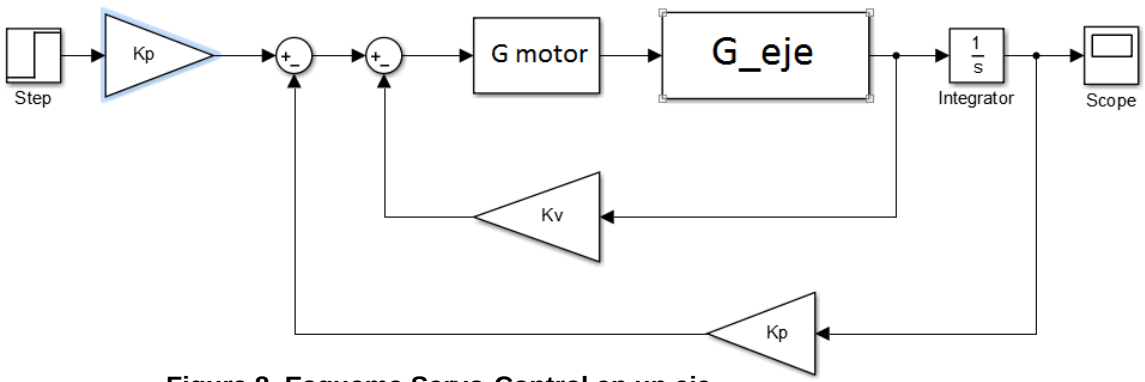


Figura 8. Esquema Servo-Control en un eje

Existen 4 actuadores en total, para los ejes X, e Y la acción se reparte entre los 2 motores enfrentados, 0 y 2 para el eje X, 1 y 3 para el eje Y, para el eje Z, (tanto rotacional como para el empuje) se usan los 4 motores, de ahí los factores de división en las constantes, así como su signo, en función de su aporte a la acción.

Para su implementación mediante variables de estado se incorporaran las constantes del lazo a la matriz de realimentación y a la matriz de pre alimentación. Quedando el sistema (excluyendo el estimador o integrador de perturbaciones):

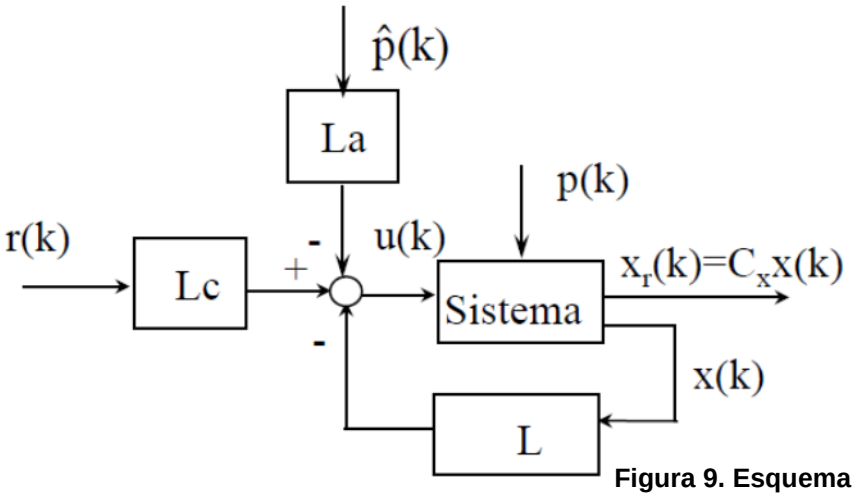


Figura 9. Esquema Pre alimentación

Quedando la matriz de realimentación y prealimentación:

```
L = [[  Kv1/2,  Kp1/2,       0,       0,  Kv3/4,  Kp3/4,    Kpr,       0,       0, 0]
     [      0,      0,  Kv2/2,  Kp2/2, -Kv3/4, -Kp3/4,      0,     Kpr,       0, 0]
     [ -Kv1/2, -Kp1/2,       0,       0,  Kv3/4,  Kp3/4,      0,       0,     Kpr, 0]
     [      0,      0, -Kv2/2, -Kp2/2, -Kv3/4, -Kp3/4,      0,       0,       0, 0]]

La = [[  Kp1/2,       0,  Kp3/4, 1 ]
      [      0,  Kp2/2, -Kp3/4, 1 ]
      [ -Kp1/2,       0,  Kp3/4, 1 ]
      [      0, -Kp2/2, -Kp3/4, 1 ]]
```

Dado que la acción se aplica en varios motores, la constante del lazo se divide por ese número de motores, así como por el signo de la aplicación para esa acción.

4.2.4.**Cálculos del control**

Los cálculos de las constantes se realizan en función de las ecuaciones del Servo-Control, mostradas en la Figura 10. La realización de los cálculos, la simulación del sistema y del eje están en los anexos

El polo más rápido se desprecia por tener una dinámica más de cinco veces más rápida al otro polo.

$$F(s)=\frac{1}{\frac{\tau}{K_r K}s^2+\frac{1+K_v K}{K_r K}s+1},$$

$$\text{con}\begin{cases}\omega_n=\sqrt{\frac{K_r K}{\tau}}\\ \xi=\frac{1+K_v K}{2\sqrt{K_r K\tau}}\end{cases}$$

Figura 10. Ecuaciones ajuste Servo Control en sistema de segundo orden con integrador

5. SENSORES

En todo sistema de control, algo tan importante como el propio sistema de compensación de la planta, es el sensado de las variables a controlar, puesto que una incorrecta medición conllevará a una incorrecta respuesta del controlador, así como una señal de salida pobre comprometerá la respuesta de todo el sistema.

Como se vio durante el modelado matemático del sistema, existen 4 magnitudes físicas a medir en las variables de estado, posición angular, velocidad angular, Fuerza y RPM.

Dada la necesidad de reducir el peso, todos los sensores estarán basados en tecnología MEMS, La tecnología MEMS se basa en el micro mecanizado del silicio, lo que confiere tamaños micrométricos y por ende poco peso, suele basarse en la variación de la capacitancia de un condensador mecanizado por una fuerza externa a él.

Los sensores a utilizar son un giróscopo, para la medición de la velocidad angular, un acelerómetro, el cual es el capaz de medir las fuerzas que actúan en el sistema y una brújula que realmente mide campos magnéticos. Todos ellos dan la medición respecto en los tres ejes, es decir en forma de vector, las RPM no se medirán, dado que no son fundamentales para el lazo de control, y se procederá a su estimación.

5.1. SENSORES ORIENTACIÓN

La orientación se mide en se compone de la orientación respecto del eje z del plano suelo (YAW) y la inclinación del chasis respecto del plano suelo (PITCH y ROLL).

5.1.1. **Acelerómetro**

Como la gravedad es única e inamovible, con una orientación perpendicular al plano terrestre, puede ser usada para calcular la orientación del chasis respecto al plano terrestre.

La gravedad es una fuerza, y por ende proporcional a la aceleración, pude medirse con un acelerómetro. Sin embargo el acelerómetro es extremadamente sensible a las vibraciones, por lo que queda imposibilitado su uso exclusivamente para calcular la posición, dado que el sistema sobre el que se monta genera mucho ruido mecánico (vibración) de alta frecuencia. Otro problema asociado al acelerómetro es que mide cualquier tipo de aceleración, no únicamente la gravedad, por lo que la aparición de otras fuerzas (siempre que el sumatorio de fuerzas que actúan sobre la aeronave no sea igual a 0), conlleva al falseamiento de la medida.

5.1.2. **Giróscopo**

El uso de un giróscopo, si bien no aporta la medición del ángulo, si lo hace de la velocidad, por lo que integrando esta velocidad se obtiene el ángulo girado desde el instante en que comienza la integral, por lo que se obtiene el angulo girado desde el punto inicial, no el absoluto, únicamente coincidirán si el angulo inicial es 0.

Al igual que todo sensor, el giróscopo acontece de errores en la medida, tales como no linealidad, o un valor de *offset,* que es una componente de continua (producida por efectos no ideales de la propia electrónica del sensor), es este insignificante valor de offset el que nos causa problemática con la técnica de la integración, al ser integrado, produce una salida lineal dependiente con el tiempo, causando un efecto deriva en el sensor, midiéndose una pequeña rotación con el tiempo, aun estando el sensor completamente estático.

5.1.3. **Brújula**

Conocido como magnómetro, usando el efecto hall es capaz de sensar la magnitud de un campo magnético en los tres ejes cardinales, por lo que pude ser usado para medir la orientación

usando el campo magnético terrestre, que si bien no es tan único e inamovible como la gravedad, es lo suficientemente estático para ser usado.

La problemática de referenciar una orientación usando un campo magnético es la variación de este, dado que si bien la emisión es estable, el flujo magnético en un punto del espacio es dependiente de la reluctancia del espacio que lo rodea. Por lo que la rotar la aeronave, y por tanto el chasis varia la reluctancia de los tres ejes espaciales del sensor, esto puede paliarse aplicando factores de corrección para la rotación del sensor en la estructura, pero no cuando lo que varía es el entorno.

Otro problema es el hecho de que las corrientes generan campos magnéticos, dada la naturaleza de impulsión eléctrica de la aeronave, es bastante factible la aparición de ruido en la medida provocada por acción de los motores eléctricos.

5.1.4. **Algoritmo de fusión sensorial.**

La solución para calcular la orientación deja de ser trivial, siendo necesario un algoritmo que fusione los sensores para paliar los desventajas que poseen por separado, obteniendo una medición de las variables precisa, absoluta, y libre de ruido.

Existen múltiples algoritmos de fusión sensorial, de varias eficiencias y costos computaciones, sin embargo todos comparten con más o menos acierto la misma idea, estimar la posición mediante la integración, corrigiendo posteriormente con el acelerómetro y la brújula.

5.1.4.1. **Filtro Complementario:**

El diseño más simple de todos, calcula la posición mediante la integración y mediante la acción de la gravedad (campo magnético para eje Z), a cada una de estas estimaciones se le asigna un tanto por uno, (estas constantes suman uno, de ahí complementario), es decir, la salida es un media ponderada entre el ángulo integrado por el giroscopio y el angulo medido por el acelerómetro.

$$Angle = \alpha \cdot (Angulo + Velocidad\ Angular \cdot dt) + (1 - \alpha) \cdot angulo\ acelerometro$$

Un filtro muy simple y con un redimiento bastante pobre.

5.1.4.2. **Filtro de Kalman:**

Específicamente desarrollado para la fusión sensorial en los años 60, este algoritmo ha sido ampliamente usado para la navegación de vehículos y guía de misiles espaciales, también permite estimar las variables no medibles, y es excelente frente al ruido blanco, es recursivo, por lo que se ajusta a las variaciones del sistema, es polivalente y puede adaptarse para cualquier sistema.

Su funcionamiento consiste en un estimador en el que la K de realimentación se calcula en función de las varianzas de los ruidos, es decir calcula cuanto de precisa es la medición respecto de la simulación. Dando como salida una medida entre ambos valores aportando peso a cada una en función de lo precisa y probable que es.

Sus inconvenientes son que se precisa de un modelo matemático preciso, así como de potencia de cálculo, dado que es computacionalmente complejo y pesado.

5.1.4.3. **Algoritmo de Mahony:**

Específicamente desarrollado para la navegación, es un algoritmo para la estimación de la actitud y rumbo (AHRS), una solución intermedia a las anteriores, se basa en el cálculo de la posición mediante el giróscopo y la posterior corrección con los otros sensores.

Utiliza la matriz de cosenos directores (DCM), aunque también existe la posibilidad de realizarlo con quaternios, lo que permite obtener la orientación del sistema en cualquier sistema de coordenadas con unas pocas operaciones.

El algoritmo se basa en rotar dicha matriz en función del ángulo que se estima que se ha rotado integrando la velocidad, para dicha rotación el algoritmo establece que se use una aproximación, sin embargo una opción más precisa, aunque algo más costosa computacionalmente, es usar la matriz de rotación respecto de un vector. El vector de rotación corresponde a las medidas del giróscopo en x,y,z, por lo que solo debe convertirse el vector en unitario y obtener su magnitud (ahí radica el aumento computacional) , una vez obtenida la matriz de rotación, se multiplica por la DCM del momento anterior, (obteniendo por las propiedades de las matrices de rotación) la nueva DCM.

El algoritmo corrige los errores numéricos detectando la perdida de ortogonalidad de la DCM, para corregirla posteriormente. Tras lo cual corrige la posición mediante un PI, usando como referencia las medidas obtenidas del acelerómetro y de la brújula.

Este es el algoritmo escogido, ya que ofrece un filtrado muy robusto, es independiente del sensor, a nivel computacional es bastante más liviano que el filtro de Kalman, y únicamente dispone de dos parámetros, su ajuste se prevé un fácil ajuste, parámetro muy deseable, debido a que el ajuste se realizará de forma experimental.

5.2. TRATAMIENTO DE LA SEÑAL OBTENIDA DE LOS SENSORES

Si bien el giróscopo es mucho menos sensible a las vibraciones mecánicas que el giróscopo, esto no indica que sea inmune. En la mayoría de sistemas de este tipo, un amortiguador mecánico sirve para este propósito (montar el sistema sobre silent-blocks), también al usarse un sistema que calcula la posición basándose en la integración de la velocidad, el ruido blanco tiene un impacto mínimo. Sin embargo el esquema de control que se usara (explicado en la sección 6.1.4) se usa la medición de la velocidad directamente, por lo que es necesario una medición libre de ruido para que el control no se vea afectado.

Tras analizar la señal de la velocidad, puede apreciarse un ruido senoidal, con una frecuencia aproximada de 100Hz, una frecuencia relativamente baja dadas las revoluciones a los que los motores funcionan, así como las conmutaciones eléctricas de los motores, por lo que posiblemente sea la frecuencia de resonancia de la estructura.

Para lograr limpiar la medida, se debe utilizar algún filtro. Como la frecuencia de respuesta del sistema es inferior, un filtro paso bajo es una opción válida dado que las frecuencias mayores nunca tendrán una magnitud considerable, por lo que un filtro pasa banda es innecesario. La problemática de este filtro es que la frecuencia de corte no es lejana a la de respuesta del sistema, debe buscarse un filtro que no atenúe las frecuencias correspondientes al funcionamiento del sistema, pero lo más alejada posible de la frecuencia del ruido. Se debe escoger una ancho de banda que no limite el tiempo de respuesta del sistema.

Otra gran ventaja de la incorporar el filtro es la eliminación de datos espureos, así como de su impacto negativo para el control. Experimentalmente se ha determinado que un filtro de 20Hz de orden 4, topología chebisev, proporciona una medida limpia de ruido, con un retraso aceptable, entendiendo como aceptable un retraso equivalente un periodo de muestreo del control, por lo que el sistema controla con la posición anterior, que puede parecer mucho, pero es inferior a 5ms y la dinámica del sistema, este retraso no afecta al control, podría haberse usado el filtrado solo para

la velocidad, eliminando de esta manera el retraso en el sensado de la posición (al ser integrada no le afecta el ruido blanco), pero experimentalmente no se apreció retraso significativo, por lo que se descartó.

La señal original puede verse en la Figura 11, tras pasar por el filtro anteriormente descrito, la señal queda apta para ser usada por el control, el efecto del filtro puede apreciarse en la Figura 12.

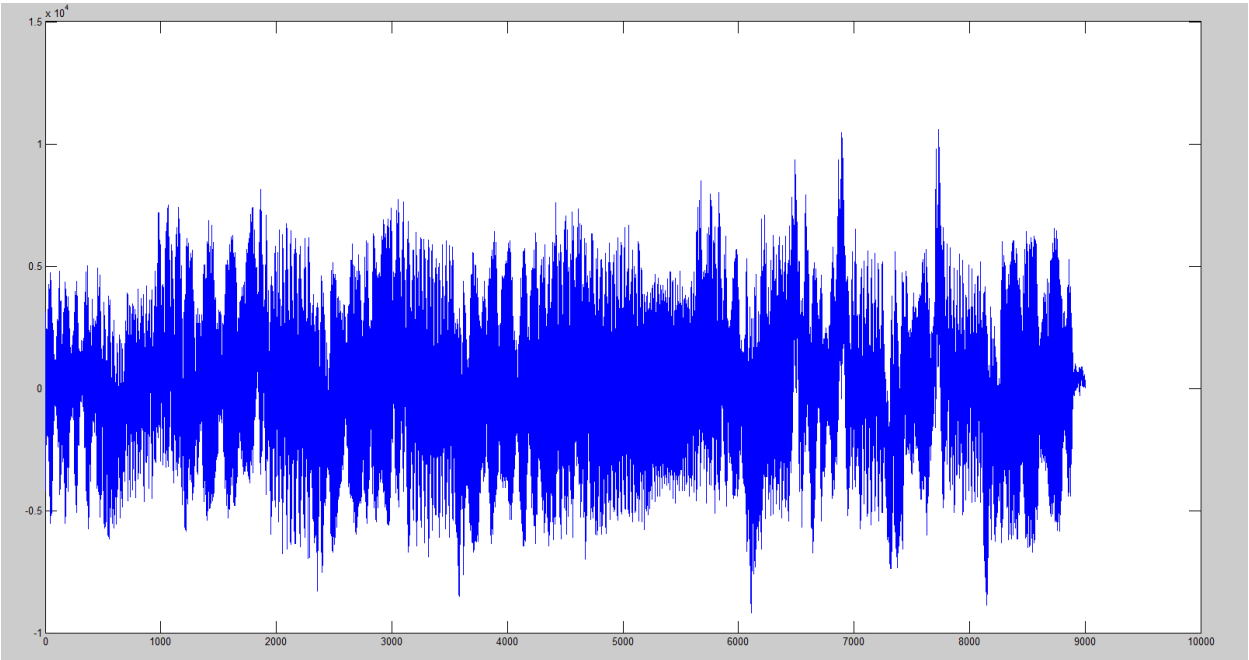


Figura 11. Medidas del giróscopo IMU9250 sin procesar

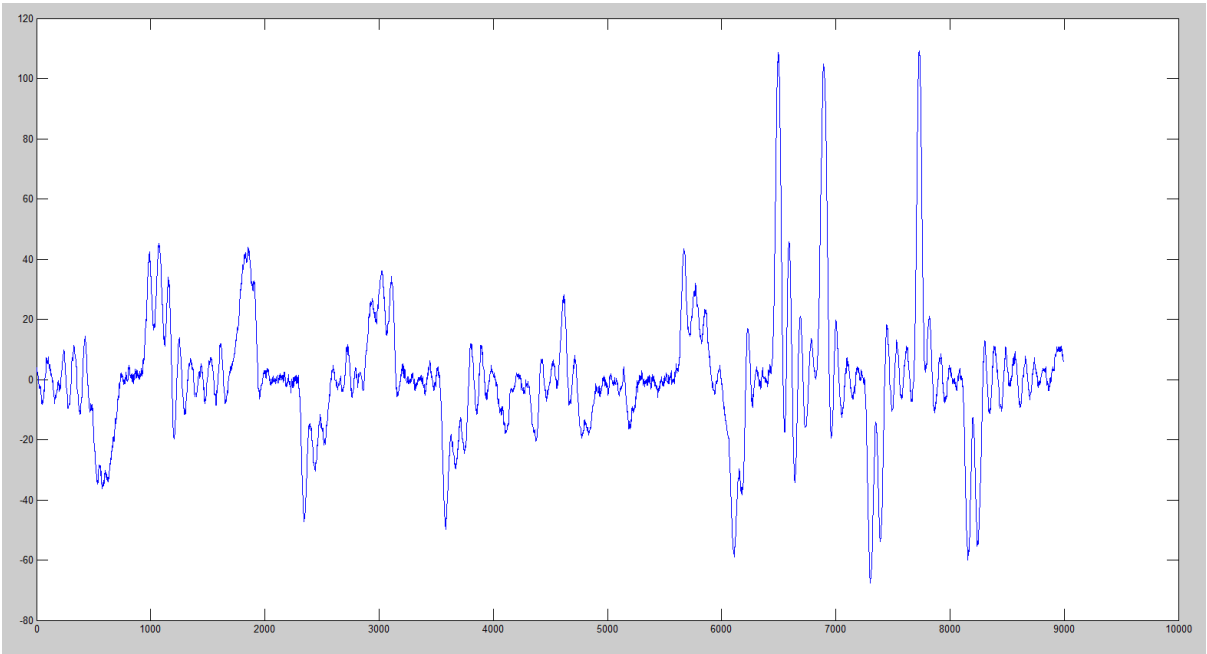


Figura 12. Medidas del giróscopo IMU9250 filtradas y convertidas a grados/segundo

6. ESTRUCTURA HARDWARE

6.1. DISPOSITIVOS HARDWARE

6.1.1. Procesador principal

Dada el esquema de control utilizado, así como el procesado de señal de los filtros, se pude deducir que la implementación del controlador estará caracterizada por un alto costo computacional.

Como también se aprecia, (y detalla más adelante), para implementar todos los procesos periódicos se opta por el uso de un Sistema Operativo en Tiempo Real (RTOS).

A nivel de periféricos, es necesario al menos 4 PWM, un I2C y una UART.

El microprocesador escogido ha sido el TM4C123GH6PM de Texas Instruments, que reúne todos los anteriores requisitos:

- Arquitectura ARM Cortex M4-F: Microprocesador ARM de la gama de alto desempeño (100 MIPS @80 Hz), equipado con unidad de coma flotante de 32 bits.
- Funciones de DSP: lo que permite realizar cálculos numéricos en pocos ciclos de reloj.
- Periféricos: 16 PWM, 4 SPI, 4 I2C, 8 UART, 2 CAN y USB.
- TI-RTOS: Sistema Operativo en Tiempo Real desarrollado por ti para sus integrados.
- Tiva Launchpad: Placa de desarrollo de bajo costo con JTAG integrado.

Dada la cantidad de periféricos se implemetaran dos I2C, una para sensores, y otra para lo demás. Así como 3 UART para telemetría y usos futuros.

6.1.2. Procesador Auxiliar

Dado que las funciones que realiza ni son críticas en tiempo, ni revisten costo computacional, se opta por un procesador simple y de bajo coste, que disponga de al menos 8 GPIO, y un I2C.

El µprocesador escogido es el Atmega8 que cumple los requisitos, se ha optado por este modelo que es pin a pin compatible con los modelos más potentes de la misma familia, por si en un futuro fuera necesaria más potencia de cálculo.

6.1.3. Sensores

Todos los sensores utilizados (excepto el US), están basados en la tecnología MEMS por el poco peso y reducido tamaño que poseen, también son digitales con un bus de comunicaciones I2C.

El sensor digital incorpora el conversor ADC, lo que evita que el ruido electromagnético se superponga a la señal de salida, al ser digitales también son configurables en cuanto al muestreo y sensibilidad. El interfaz I2C permite conectar hasta 255 dispositivos en un mismo bus, consistente en dos líneas (SCL y SDA), por lo que se ahorran pines en el procesador principal.

Los modelos usados son el MPU-6050 del fabricante Invesense, consistente en acelerómetro y Giróscopo, y como brújula la HMC5883L de HoneyWell.

También se ha trabajado con el MPU-9250 de Invesense, que incorpora los tres sensores en un único integrado.

6.2. DESARROLLO DE LA ELECTRÓNICA ASOCIADA

Ha sido necesario el desarrollo de circuitería adicional, los esquemáticos se encuentran en los anexos.

6.2.1. Diseño sobre placa de desarrollo

Se ha desarrollado una PCB donde se inserta la placa de desarrollo "*Tiva Launchpad*", la función de esta PCB, es dar tanto apoyo mecánico a la placa de desarrollo para fijarla al chasis de la aeronave, como para ubicar la circuitería adicional requerida:

- Microprocesador Auxiliar
- Circuitería de alimentación
- Conectores para los sensores
- Circuitería de los buses de comunicación

6.2.2. Diseño de PCB de la versión final

Tras comprobar el correcto funcionamiento del montaje en la placa de desarrollo se implementa la misma circuitería en una PCB, prescindiendo esta vez de la placa de evaluación y desarrollando una placa de control completa, que no necesita de ningún elemento externo para el control, se han añadido los sensores de orientación, consistentes en un único integrado.

Eliminando todos los elementos no necesarios (conectores, pulsador, JTAG), se obtiene una PCB cuadrada de 5cm de lado, reduciendo de esta manera el peso, tamaño, y coste de la placa controladora.

También se han añadido conexiones a periféricos no usados con vistas a futuros desarrollos.

7. ESTRUCTURA SOFTWARE

7.1. ESTRUCTURA DEL RTOS EN µPROCESADOR PRINCIPAL

Se adopta una planificación de tareas basada en prioridades estáticas, dado que el plazo de respuesta es igual al periodo, la técnica empleada será Rate Monotonic, ordenando la prioridad de las tareas en función de su periodicidad, usando la técnica de techo de prioridad para minimizar el efecto de la inversión de prioridad.

Se ha evitado el uso de variables globales en la medida de lo posible, sustituyéndolas por variables protegidas por "Mutex" (mecanismo de exclusión mutua), evitando colisiones entere tareas que comparten datos. Únicamente los "handlers" de las tareas y periféricos, y variables relacionadas con interrupciones (dado que estas no pueden hacer uso de las "Mutex"), sin embargo estas todas estas son variables del tamaño de palabra del procesador, por lo que sus instrucciones son atómicas, anulando la posibilidad de error.

Para transferencia de datos sin interés global (séase comunicación punto a punto) entre tareas se hace uso de los "Mailbox"

Para la sincronización de tareas tanto periódicas como en el arranque se utiliza la función de "Semaphore", activadas por temporizadores software (en el caso de la activación periódica).

La comprobación de que el sistema cumple plazos, así como las capturas del tiempo de ejecución de las tareas se detallan en los anexos, para dichos cálculos se asume un tiempo despreciable de bloqueo por acceso a variables compartidas, dado el escaso tamaño de estas, y por la velocidad interna del bus del procesador, el bus compartido si se ha contemplado como bloqueo.

7.1.1.**Tareas**

Las tareas y su descripción se listan por orden de prioridad

7.1.1.1. **Lectura IMU**

T = 1ms, C = 0.080ms

La función de esta tarea consiste en leer los datos de los sensores de orientación (giróscopo, acelerómetro, brújula), sean o no parte de un mismo integrado.

Tras leer los datos estos son filtrados mediante el filtro anteriormente descrito en la sección **4.1 Sensores Orientación,** también se les aplica una rotación para obtener la medida en base a los ejes referencia del chasis, para esto se usan las propiedades de la matriz de rotación, para finalmente almacenar el dato en la variable protegida.

El periodo de esta tarea es de 2ms, para permitir filtrados de hasta 200Hz, las funciones de DSP anteriormente descritas consiguen reducir considerablemente el tiempo de la ejecución de los 6 filtros, y por ende el de la tarea que se ejecuta 500 veces por segundo.

7.1.1.2. **AHRS**

T = 2ms, C = 0.413ms

La tarea ejecuta el algoritmo de fusión sensorial anteriormente descrito a razón de 200Hz, un tiempo lo suficientemente pequeño para mantener una medida discreta muy similar a la continua, sin ser demasiado pequeño para producir errores por diferencia de tamaño en la coma flotante.

Para la ejecución del algoritmo, la tarea obtiene los datos AHRS anteriores, y las medidas más recientes del IMU, tras esto las procesa para obtener la nueva medida y la almacena.

7.1.1.3. **Control**

T = 5ms, C =0.400ms

Tarea principal del sistema en la que se ha implementado el algoritmo de control y estimación, en función de los parámetros del sistema ejecuta el método de control escogido, así como el estimador de variables y perturbaciones. Por la propia estructura del control, es la tarea computacionalmente más pesada.

Adicionalmente también trasmite vía UART la telemetría consistente en las variables de estado del sistema, perturbaciones estimadas, medidas de los sensores, referencias de entrada, y acciones aplicadas a cada motor.

7.1.1.4. **Medida Altura**

T = 50ms, C = 0.050ms

Tarea consistente en lanzar un pulso al sensor de US e iniciar el contador de tiempo asociado, así como de iniciar la medida de presión del barómetro, tras lo que entra en

estado inactivo el tiempo que el sensor necesita para sensar la presión atmosférica, tras lo que se activa para leer y procesar la media, almacenándola.

La tarea no se encarga de la recepción del eco del sensor (si lo hubiere), eso es gestionado mediante interrupción.

7.1.1.5. **Coordinador**

T = 25ms, C = 0.030ms

La tarea encargada de comunicarse con el microprocesador auxiliar par obtener vía I2C las medidas del receptor.

Tras esto determina que actuación debe tener la aeronave, configurando los parámetros para el control, gestionar los modos de funcionamiento, y activar o desactivar las tareas en consecuencia.

7.1.1.6. **Identificación**

Ejecución exclusiva, no en el ciclo inicial. Únicamente usada para la identificación del sistema.

Tarea que lee a través de la UART una secuencia de datos correspondientes a las acciones a aplicar en los motores, almacenándolas en la memoria dinámica.

Tras la recepción aplica a los motores la acción requerida, enviando por la UART la velocidad sensada.

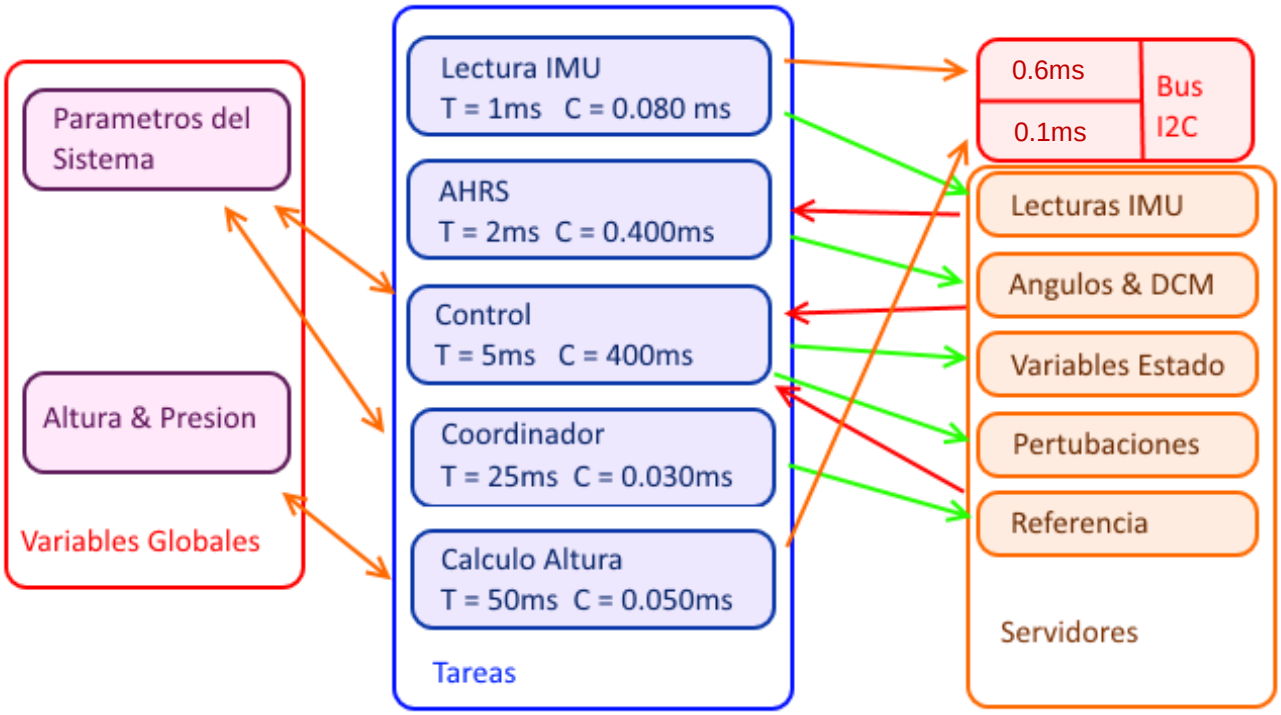


Figura 13. Esquema de Tareas y recursos compartidos del Sistema Operativo.

7.2. ESTRUCTURA DEL µPROCESADOR AUXILIAR

La inclusión de un microprocesador auxiliar no surge por la necesidad de paralelizar tareas para aliviar al principal, su inclusión es debida a un aspecto de seguridad.

7.2.1. **Sistema de desactivación de emergencia.**

Si bien el procesador principal cuenta con dispositivos de protección, tales como un WatchDog, no se puede garantizar la no existencia de errores.

Pese a que estos aspectos han sido tenidos en cuenta durante el desarrollo, y ante la imposibilidad de demostrar que no existe error, por improbable que sea, o un error de hardware debido a factores externos como el ruido (más improbable aun), se ha optado por usar un microprocesador externo, con una programación “simple” (y por ende, menos propensa a errores), consistente en un bucle eterno que detenga el sistema en caso de error, como es habitual en los sistemas de robótica.

Así pues, este micro activa el reset en el microprocesador principal cuando pierde en enlace de radio. Lo que se consigue analizando la trama PWM de la siguiente manera:

Con la recepción del primer flanco de la señal, se resetea un contador descendente, el valor de este contador es mayor que la distancia periódica entre tramas de 20ms, por lo que nunca llegara a 0 mientras reciba la señal del receptor, si esta se pierde, al llegar a 0 ejecuta la rutina que resetea el micro principal activando el correspondiente pin

7.2.2. **Recepción de radio & sensado de batería**

Dado que la señal de radio es mecanismo de activación, se ha implementado que sea este procesador quien capture la señal PWM periódica del receptor, así como los niveles de tensión de las diferentes celdas de la batería.

Con esto evitamos el uso de pines en el microprocesador principal, así como las interrupciones que generarían, lo que penalizaría la ejecución debido a la segmentación del procesador.

En contra, el micro principal lee los valores recibidos del receptor del micro auxiliar a través de una comunicación I2C.

8. VUELO:

8.1. PRUEBAS DE VUELO

Tras los ajustes pertinentes, se determina que el estimador de perturbaciones no es lo suficientemente preciso, por lo que puede estimar las perturbaciones, pero no corregirlas, desestabilizando el sistema.

Por ese motivo se emplea el integrador de perturbaciones en el bucle de control, logrando así un error de permanente nulo, como se aprecia en las imágenes obtenidas de la telemetría:

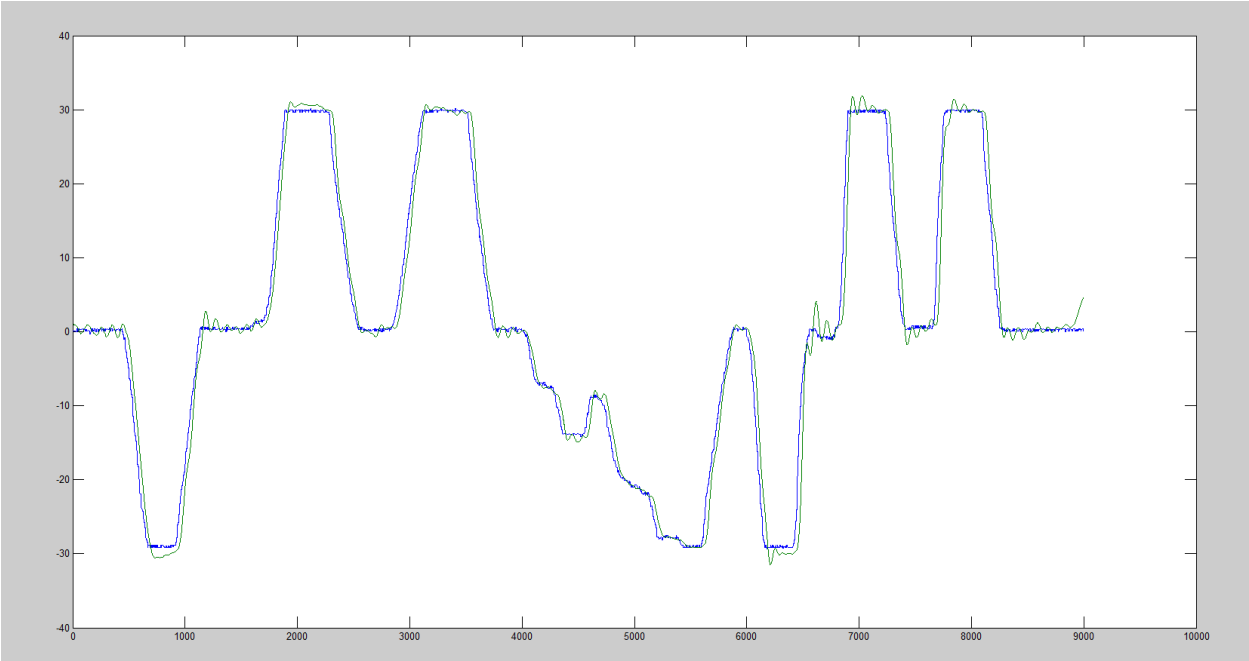


Figura 14. Captura de telemetría, Referencia en azul, Posición angular en verde

Con una perturbación estimada (incluyendo la pre alimentación) de:

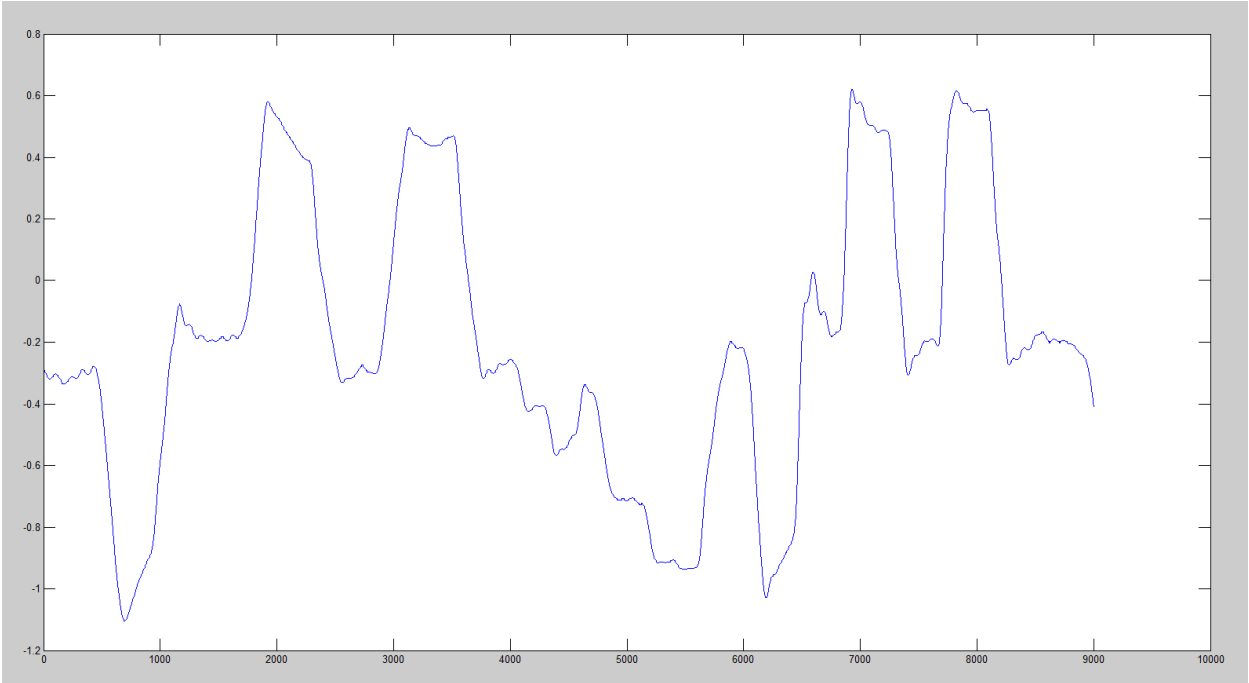


Figura 15. Captura de telemetría, Perturbación modelada como fuerza en el eje.

Problemas que sobre el modelo teórico nunca se predijeron, tales como:

- Ruido en los sensores, producido por las vibraciones mecánicas que llevo a reenfocar todo el algoritmo de sensado, fijación del sensor…, con el consecuente retraso de todo el proyecto.
- Cambio en el control, en un principio se pretendía usar un joystick con enlace de Bluetooth, lo que termino por descartarse dado el alcance del mismo, como por su incapacidad de mandar datos en tiempos fijos.
- Recableado completo del sistema, los grandes picos de corriente de los motores llegan a inducir tensiones en los cableados de control así como en la propia referencia del motor, lo que producía que los motores tuvieran comportamientos erráticos.
- Efecto suelo y otros efectos no lineales, sobre todo a bajas rpm, muy lejos de la zona linealizada, lo que produce que la aeronave tenga un comportamiento mas oscilante.
- Necesidad de una completa telemetría, para identificar las causas del mal funcionamiento del sistema, o de la dinámica del mismo, gracias a la cual se pudo descubrir el ruido y como afectaba al control.
- Fallos de montaje, como orientación incorrecta del sensor, cableados incorrectos, o cables demasiado largos que acabaron enganchándose en las hélices, todos ellos concluyendo en un violento impacto contra el suelo.
- Inexperiencia en el pilotaje, lo que complico las pruebas de vuelo, asi como la extracción de conclusiones.
- Micro Auxiliar de seguridad, tras un bloqueo del TI-RTOS

9. CONCLUSIONES

Tras el desarrollo del control, y visto el comportamiento del mismo, se puede deducir que para este método de control es imprescindible, sobre todo en lo referente al estimador, que el modelo sea lo más exacto posible, es necesaria una identificación más precisa de lo que se obtiene en la estructura, quizás una identificación en vuelo sería lo más preciso, compensando las perturbaciones para hacer más precisa la identificación.

Salvando ese obstáculo, o sustituyéndolo por el integrador de perturbaciones, el control Servo-Control ha demostrado ser válido para este tipo de aplicación, precisando de únicamente la velocidad y posición del eje, que pueden medirse directamente, no siendo necesaria la implementación en variables de estado sino desea el estimador.

Para esta aplicación, una medida limpia de ruido en la velocidad es determinante.

El algoritmo de AHRS, junto con el filtrado de los sensores, logra una medida limpia de la ubicación y velocidad del sistema, consiguiendo eliminar los efectos de deriva.

El proyecto no está cerrado, pueden seguirse añadiendo funcionalidades, como el control de posición en el espacio cartesiano, el control de altura…, podría mejorarse el sistema de control usando ESC que realimenten las revoluciones de los motores, o re-identificar el sistema con los datos obtenidos durante el vuelo.

10. REFERENCIAS & BIBLIOGRAFIA

- "INGENIERIA DE CONTROL MODERNA 5ª ED", Katsuhiko Ogata.
- "SISTEMAS DE CONTROL EN TIEMPO DISCRETO", Katsuhiko Ogata.
- "COMPLEMENTARY FILTER DESIGN ON THE SPECIAL ORTHOGONAL GROUP", Robert Mahony, Tarek Hamel, Jean-Michel Pflimlin.
- "MODELLING, IDENTIFICATION AND CONTROL OF A QUADROTOR HELICOPTER", Tommaso Bresciani.

11. ANEXOS

A. CÁLCULOS SISTEMA OPERATIVO

Para el cálculo de los tiempos de ejecución del sistema, se contemplan como despreciables el tiempo de acceso a las variables compartidas, dada la frecuencia de trabajo del procesador, así como el tamaño de las mismas.

El bus I2C compartido se contempla como recurso compartido, el tiempo de acceso a los datos se ha contabilizado fuera del tiempo de ejecución de la tarea, dado que el procesador puede atender otras tareas. Se ha modelado como retraso fijo, al que se le añade el retraso producido por que una tarea de menos prioridad este usando el recurso compartido, lo cual se calcula en función del periodo de ambas tareas.

Tarea Lectura IMU

- T=D=1ms, C = 0.080ms
- Retraso propio de acceso al I2C de 0.6ms y por otras tareas de menor prioridad de 0.1ms.

$$d_{lectura\ IMU} = C_{lectura\ IMU} + b_{lectura\ IMU\ I2C} + b_{I2C} = 0.080\text{ms} + 0.6\text{ms} + 0.1\text{ms} = 0.78\text{ms} < 1\text{ms}$$

Tarea AHRS

- T=D=2ms, C = 0.400ms

$$d_{AHRS} = \left(\left\lceil \frac{P_{Lectura\ IMU}}{P_{AHRS}} \right\rceil (C_{lectura\ IMU}) + \left\lceil \frac{P_{Cálculo\ Altura}}{P_{AHRS}} \right\rceil b_{I2C}\right) + C_{AHRS} =$$

$$2 * (0.08ms) + 1 * (0.1ms) + 0.4ms = 0.66ms < 2ms$$

Tarea Control

- T=D=5ms, C = 0.600ms

$$d_{Control} = \left(\left\lceil \frac{P_{Lectura\ IMU}}{P_{Control}} \right\rceil (C_{lectura\ IMU}) + \left\lceil \frac{P_{Cálculo\ Altura}}{P_{Control}} \right\rceil b_{I2C}\right) + \left\lceil \frac{P_{AHRS}}{P_{Tarea\ Control}} \right\rceil C_{AHRS} + C_{Control} =$$

$$5 * (0.08ms) + 1 * (0.1ms) + 2 * 0.4ms + 0.4ms = 1.7ms < 5ms$$

Tarea Coordinador

- T=D=25ms, C = 0.030ms

$$d_{Coordinador} = \left(\left\lceil \frac{P_{Lectura\ IMU}}{P_{Coordinador}} \right\rceil (C_{lectura\ IMU}) + \left\lceil \frac{P_{Cálculo\ Altura}}{P_{Coordinador}} \right\rceil b_{I2C}) + \left\lceil \frac{P_{AHRS}}{P_{Coordinador}} \right\rceil C_{AHRS} + \left\lceil \frac{P_{Control}}{P_{Coordinador}} \right\rceil C_{Control} + C_{Coordinador} = 25 * (0.08ms) + 1 * (0.1ms) + 12 * 0.4ms + 5 * 0.4ms + 0.03ms = 8.93ms < 25ms$$

Tarea Cálculo Altura

- T=D=50ms, C = 0.050ms
- Retraso propio de acceso al I2C de 0.6ms

$$d_{Cálculo\ Altura} = \left\lceil \frac{P_{Lectura\ IMU}}{P_{Cálculo\ Altura}} \right\rceil C_{lectura\ IMU} + \left\lceil \frac{P_{AHRS}}{P_{Cálculo\ Altura}} \right\rceil C_{AHRS} + \left\lceil \frac{P_{Control}}{P_{Cálculo\ Altura}} \right\rceil C_{Control} + \left\lceil \frac{C_{Coordinador}}{P_{Cálculo\ Altura}} \right\rceil C_{Coordinador} + C_{Cálculo\ Altura} + b_{Cálculo\ Altura\ I2C} = 50 * (0.08ms) + 25 * 0.4ms + 10 * 0.4ms + 2 * 0.03ms + 0.050ms + 0.1ms = 18.21ms < 50ms$$

Con lo que queda demostrado que todas las tareas cumplen plazos.

B. CÁLCULOS CONTROL

Extracto del fichero .m de Matlab

```
% Probar distintas ganancias para el observador sin la ralimentacion de los motores

Km  = 0.009637; % Fuerza(N) / Accion (microseg)

K   = 31.047  / 16.4;   % Angulo(grados) / Accion (microseg)          % Angulo(grados) / Accion (microseg)

Kq  = K / Km;  % Angulo(grados) / Fuerza(N)
Kq2 = K * 0.15 / Km;  % Angulo(grados) / Fuerza(N)

Jq1 = 0.8613;
Jq2 = 0.8613;
Jq3 = 0.8613;

Jm  = 0.096369;

%Aproximamos sistema de segundo orden con:
T_muestreo = 0.005

Wn = 10;
chi = 1;

Kp1 = Wn^2*Jq1/K;
Kp2 = Wn^2*Jq2/K;
Kp3 = Wn^2*Jq3/(K * 0.15);

Kv1 = (2*chi*Wn - 1) / K;
Kv2 = (2*chi*Wn - 1) / K;
Kv3 = (2*chi*Wn - 1) / (K * 0.15);
```

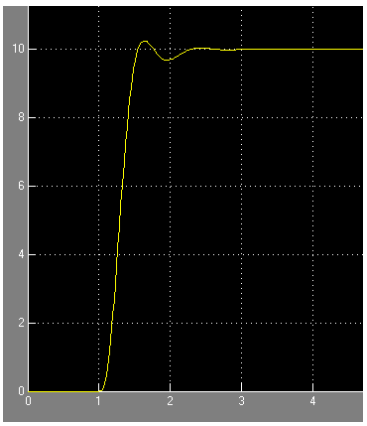
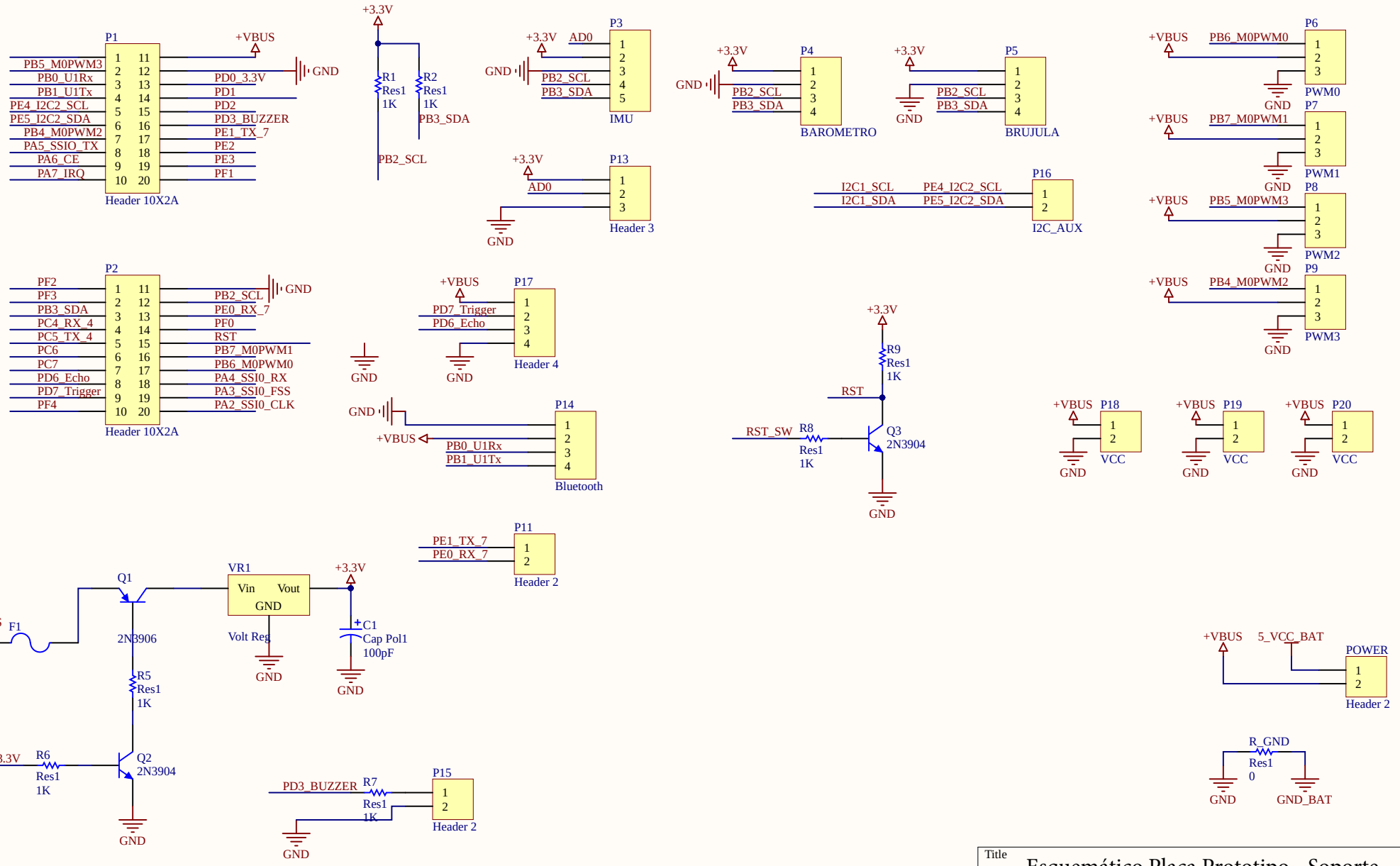
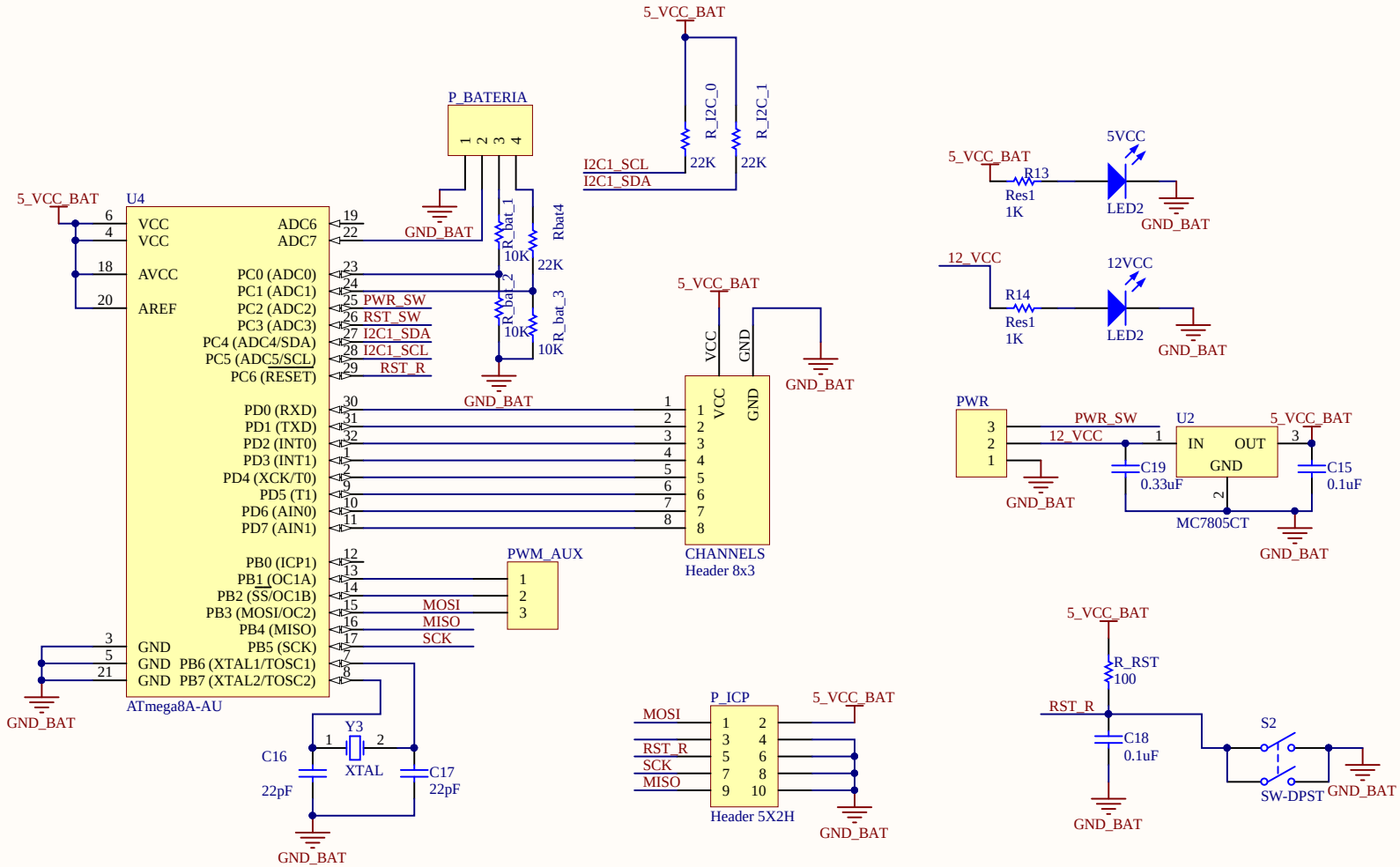


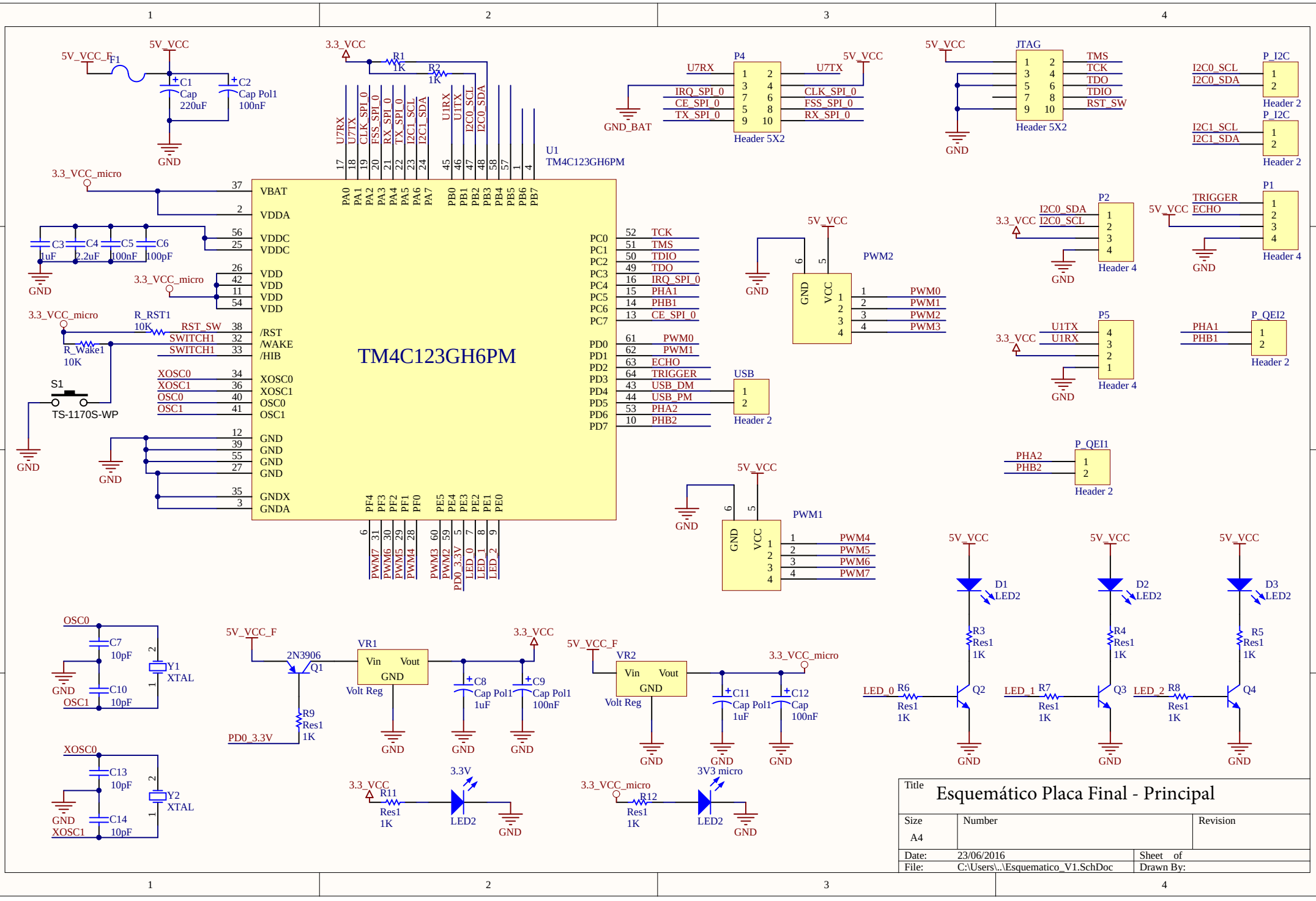
Figura 16. Respuesta al escalón del sistema en Simulink

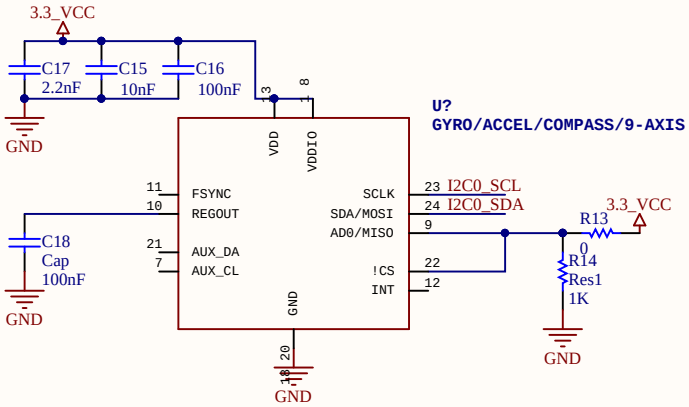


Title	Esquemático Placa Prototipo - Soporte	
Size: A4	Number:	Revision:
Date: 23/06/2016		Sheet of
File: C:\Users\..\PlacaSoporte_v2.SchDoc		Drawn By:

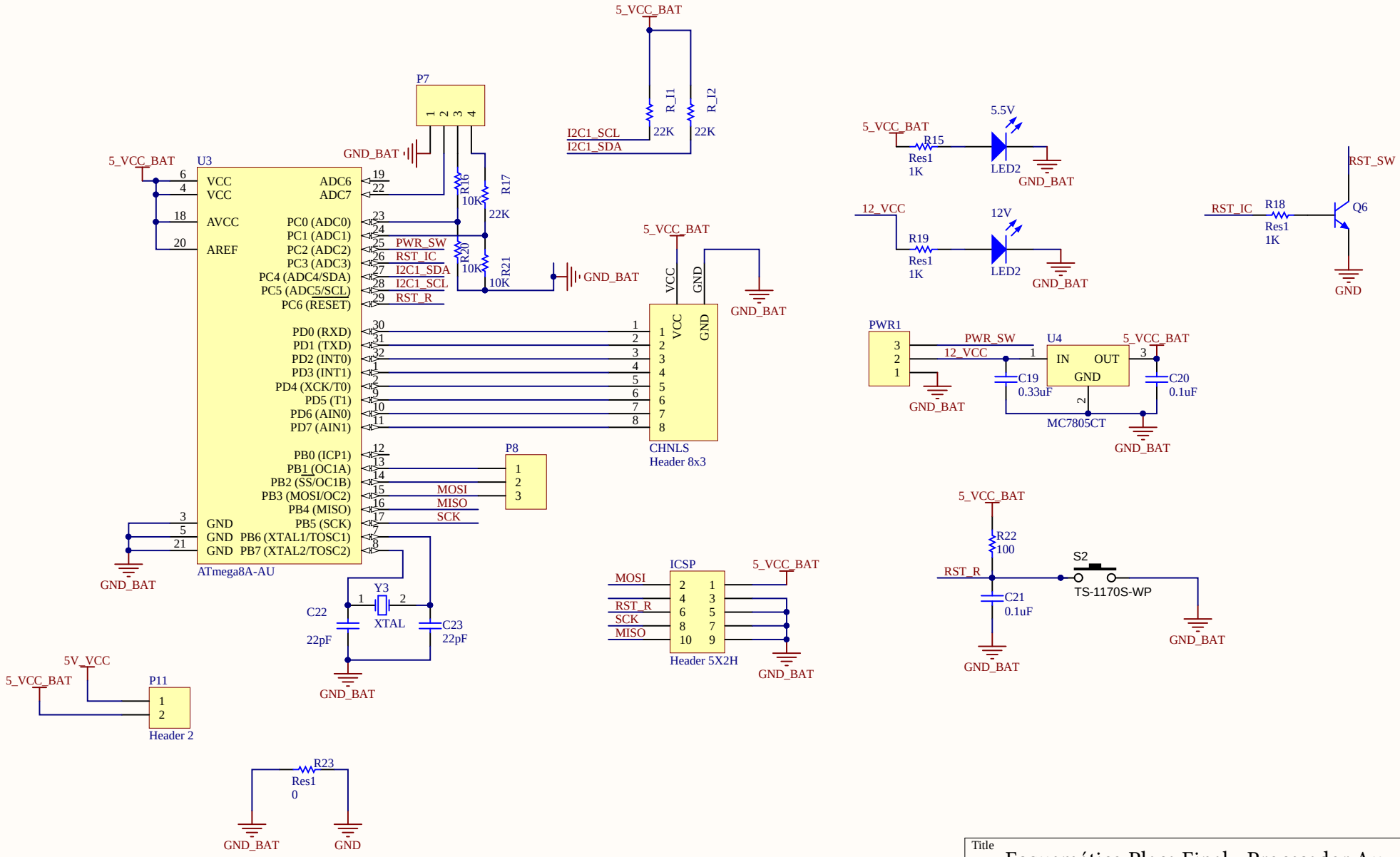


Title	Esquemático Placa Prototipo - Procesador Aux	
Size: A4	Number:	Revision:
Date: 23/06/2016		Sheet of
File: C:\Users\..\uC_Aux.SchDoc		Drawn By:


TM4C123GH6PM
U1
TM4C123GH6PM
5V_VCC_F
5V_VCC
3.3_VCC
3.3_VCC_micro
GND
GND_BAT
C1
Cap
220uF
C2
Cap Pol1
100nF
R1
1K
R2
1K
F1
P4
Header 5X2
U7RX
U7TX
IRQ_SPI_0
CLK_SPI_0
CE_SPI_0
FSS_SPI_0
TX_SPI_0
RX_SPI_0
JTAG
TMS
TCK
TDO
TDIO
RST_SW
P_I2C
I2C0_SCL
I2C0_SDA
I2C1_SCL
I2C1_SDA
Header 2
P2
Header 4
P1
TRIGGER
ECHO
P5
U1TX
U1RX
P_QEI2
PHA1
PHB1
P_QEI1
PHA2
PHB2
PWM2
PWM1
PWM0
PWM1
PWM2
PWM3
PWM4
PWM5
PWM6
PWM7
USB
USB_DM
USB_PM
PHA2
PHB2
C3
1uF
C4
2.2uF
C5
100nF
C6
100pF
R_RST1
10K
R_Wake1
10K
RST_SW
SWITCH1
S1
TS-1170S-WP
XOSC0
XOSC1
OSC0
OSC1
C7
10pF
C10
10pF
Y1
XTAL
C13
10pF
C14
10pF
Y2
XTAL
2N3906
Q1
R9
Res1
1K
PD0_3.3V
VR1
Vin
Vout
Volt Reg
C8
Cap Pol1
1uF
C9
Cap Pol1
100nF
VR2
C11
Cap Pol1
1uF
C12
Cap
100nF
3.3V
3V3 micro
R11
R12
LED2
D1
D2
D3
LED2
R3
R4
R5
Res1
1K
LED_0
LED_1
LED_2
R6
R7
R8
Q2
Q3
Q4
Title
Esquemático Placa Final - Principal
Size
A4
Number
Revision
Date:
23/06/2016
Sheet of
File:
C:\Users\..\Esquematico_V1.SchDoc
Drawn By:



Title	Esquemático Placa Final - Sensor		
Size A4	Number		Revision
Date:	23/06/2016	Sheet of	
File:	C:\Users\..\Sensores.SchDoc	Drawn By:	



Title	Esquemático Placa Final - Procesador Aux	
Size A4	Number	Revision
Date: 23/06/2016		Sheet of
File: C:\Users\..\uC_Aux.SchDoc		Drawn By:

```c
/*
 * AHRS.c
 *
 *  Created on: 22/12/2015
 *      Author: Ruben
 */
#include "AHRS.h"
#include "Servidores.h"

void Compensacion_Sensor_magnetico(tpAHRS *AHRS){
    //Rota el eje magnetico para alinearlo con el suelo, usando la ultima referencia ROLL
    PITCH
    float32_t mag_x;
    float32_t mag_y;
    float32_t cos_roll;
    float32_t sin_roll;
    float32_t cos_pitch;
    float32_t sin_pitch;

    cos_roll = arm_cos_f32(AHRS->Roll);
    sin_roll = arm_sin_f32(AHRS->Roll);
    cos_pitch = arm_cos_f32(AHRS->Pitch);
    sin_pitch = arm_sin_f32(AHRS->Pitch);

    // Rotamos
    mag_x = AHRS->Vector_Magnetico[0] * cos_pitch + AHRS->Vector_Magnetico[1] * sin_roll *
    sin_pitch + AHRS->Vector_Magnetico[2] * cos_roll * sin_pitch;
    mag_y = AHRS->Vector_Magnetico[1] * cos_roll - AHRS->Vector_Magnetico[2] * sin_roll;
    AHRS->Orientacion_YAW = atan2(-mag_y, mag_x);
}

void Actualizar_Matriz_DCM(tpAHRS *AHRS){
    float32_t Velocidad_Total[3] = {0, 0, 0};

    float32_t Rot_matriz[3][3] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
    arm_matrix_instance_f32 Rotacion = {3, 3, (float32_t *)Rot_matriz};

    float32_t Aux_matriz[9] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
    arm_matrix_instance_f32 Aux = {3, 3, Aux_matriz};

    Velocidad_Total[0] = AHRS->Vector_Velocidad_Angular[0] +
    AHRS->Correccion_Proporcional[0] + AHRS->Correccion_Integral[0];
    Velocidad_Total[1] = AHRS->Vector_Velocidad_Angular[1] +
    AHRS->Correccion_Proporcional[1] + AHRS->Correccion_Integral[1];
    Velocidad_Total[2] = AHRS->Vector_Velocidad_Angular[2] +
    AHRS->Correccion_Proporcional[2] + AHRS->Correccion_Integral[2];

    Rot_matriz[0][0] =  1;
    Rot_matriz[0][1] = -AHRS->Periodo_Muestreo*Velocidad_Total[2];//-z
    Rot_matriz[0][2] =  AHRS->Periodo_Muestreo*Velocidad_Total[1];//y
    Rot_matriz[1][0] =  AHRS->Periodo_Muestreo*Velocidad_Total[2];//z
    Rot_matriz[1][1] =  1;
    Rot_matriz[1][2] = -AHRS->Periodo_Muestreo*Velocidad_Total[0];//-x
    Rot_matriz[2][0] = -AHRS->Periodo_Muestreo*Velocidad_Total[1];//-y
    Rot_matriz[2][1] =  AHRS->Periodo_Muestreo*Velocidad_Total[0];//x
    Rot_matriz[2][2] =  1;

    arm_mat_mult_f32(&AHRS->DCM, &Rotacion, &Aux);
    arm_copy_f32(Aux.pData, AHRS->DCM.pData, 9);
}

void Actualizar_Matriz_DCM_V2(tpAHRS *AHRS){
    float32_t Velocidad_Total[3] = {0, 0, 0};
    float32_t Vector_Rotacion[3] = {0, 0, 0};

    float32_t Angulo_Rotacion = 0;
    float32_t Seno = 0;
    float32_t Coseno = 0;
```

```
    float32_t Rot_matriz[3][3] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
    arm_matrix_instance_f32 Rotacion = {3, 3, (float32_t *)Rot_matriz};

    float32_t Aux_matriz[9] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
    arm_matrix_instance_f32 Aux = {3, 3, Aux_matriz};

    Velocidad_Total[0] = AHRS->Vector_Velocidad_Angular[0] +
    AHRS->Correccion_Proporcional[0] + AHRS->Correccion_Integral[0];
    Velocidad_Total[1] = AHRS->Vector_Velocidad_Angular[1] +
    AHRS->Correccion_Proporcional[1] + AHRS->Correccion_Integral[1];
    Velocidad_Total[2] = AHRS->Vector_Velocidad_Angular[2] +
    AHRS->Correccion_Proporcional[2] + AHRS->Correccion_Integral[2];

    //Velocidad absoluta
    arm_sqrt_f32((Velocidad_Total[0]*Velocidad_Total[0] +
    Velocidad_Total[1]*Velocidad_Total[1] +
                  Velocidad_Total[2]*Velocidad_Total[2]), &Angulo_Rotacion);

    //Normalizamos vector rotacion
    if (Angulo_Rotacion != 0.0){
        arm_scale_f32(Velocidad_Total, 1/Angulo_Rotacion, Vector_Rotacion, 3);
    }

    //Pasamos de velocidad a angulo
    Angulo_Rotacion = Angulo_Rotacion*AHRS->Periodo_Muestreo;

    Coseno = arm_cos_f32(Angulo_Rotacion);
    Seno = arm_sin_f32(Angulo_Rotacion);

    Rot_matriz[0][0] = Coseno + Vector_Rotacion[0]*Vector_Rotacion[0]*(1 - Coseno);
    Rot_matriz[0][1] = Vector_Rotacion[0]*Vector_Rotacion[1]*(1 - Coseno) -
    Vector_Rotacion[2]*Seno;
    Rot_matriz[0][2] = Vector_Rotacion[0]*Vector_Rotacion[2]*(1 - Coseno) +
    Vector_Rotacion[1]*Seno;
    Rot_matriz[1][0] = Vector_Rotacion[1]*Vector_Rotacion[0]*(1 - Coseno) +
    Vector_Rotacion[2]*Seno;
    Rot_matriz[1][1] = Coseno + Vector_Rotacion[1]*Vector_Rotacion[1]*(1 - Coseno);
    Rot_matriz[1][2] = Vector_Rotacion[1]*Vector_Rotacion[2]*(1 - Coseno) -
    Vector_Rotacion[0]*Seno;
    Rot_matriz[2][0] = Vector_Rotacion[2]*Vector_Rotacion[0]*(1 - Coseno) -
    Vector_Rotacion[1]*Seno;
    Rot_matriz[2][1] = Vector_Rotacion[2]*Vector_Rotacion[1]*(1 - Coseno) +
    Vector_Rotacion[0]*Seno;
    Rot_matriz[2][2] = Coseno + Vector_Rotacion[2]*Vector_Rotacion[2]*(1 - Coseno);

    arm_mat_mult_f32(&AHRS->DCM, &Rotacion, &Aux);
    arm_copy_f32(Aux.pData, AHRS->DCM.pData, 9);
}

void Normalizar_DCM(tpAHRS *AHRS){
    float error = 0;

    float32_t Vector_Aux[3] = {0, 0, 0};
    float32_t Matriz_Ortogonal[3][3] = {0, 0, 0, 0, 0, 0, 0, 0, 0};

    arm_dot_prod_f32(&AHRS->DCM_matriz[0][0], &AHRS->DCM_matriz[1][0], 3, &error);
    error *= -0.5;

    arm_scale_f32(&AHRS->DCM_matriz[1][0], error, Vector_Aux, 3);
    arm_add_f32(&AHRS->DCM_matriz[0][0], Vector_Aux,  &Matriz_Ortogonal[0][0], 3); //Vector
    X ortogonal

    arm_scale_f32(&AHRS->DCM_matriz[0][0], error, Vector_Aux, 3);
    arm_add_f32(&AHRS->DCM_matriz[1][0], Vector_Aux,  &Matriz_Ortogonal[1][0], 3); //Vector
    Y ortogonal

    //Producto Cruz
```

```
    Matriz_Ortogonal[2][0] = Matriz_Ortogonal[0][1] * Matriz_Ortogonal[1][2] -
    Matriz_Ortogonal[0][2] * Matriz_Ortogonal[1][1];
    Matriz_Ortogonal[2][1] = Matriz_Ortogonal[0][2] * Matriz_Ortogonal[1][0] -
    Matriz_Ortogonal[0][0] * Matriz_Ortogonal[1][2];
    Matriz_Ortogonal[2][2] = Matriz_Ortogonal[0][0] * Matriz_Ortogonal[1][1] -
    Matriz_Ortogonal[0][1] * Matriz_Ortogonal[1][0  ];

    arm_dot_prod_f32(&Matriz_Ortogonal[0][0], &Matriz_Ortogonal[0][0], 3, &error);
    error = 0.5*(3.0 - error);
    arm_scale_f32(&Matriz_Ortogonal[0][0], error, &AHRS->DCM_matriz[0][0], 3);

    arm_dot_prod_f32(&Matriz_Ortogonal[1][0], &Matriz_Ortogonal[1][0], 3, &error);
    error = 0.5*(3.0 - error);
    arm_scale_f32(&Matriz_Ortogonal[1][0], error, &AHRS->DCM_matriz[1][0], 3);

    arm_dot_prod_f32(&Matriz_Ortogonal[2][0], &Matriz_Ortogonal[2][0], 3, &error);
    error = 0.5*(3.0 - error);
    arm_scale_f32(&Matriz_Ortogonal[2][0], error, &AHRS->DCM_matriz[2][0], 3);
}

void Correccion_deriva(tpAHRS *AHRS){
    float32_t error[3] = {0, 0, 0};
    float32_t Aux[3] = {0, 0, 0};

    //ROLL PITCH
    //faltaria filtrar???
    //Producto cruz
    error[0] = AHRS->Vector_Aceleracion_lineal[1] * AHRS->DCM_matriz[2][2] -
    AHRS->Vector_Aceleracion_lineal[2] * AHRS->DCM_matriz[2][1];
    error[1] = AHRS->Vector_Aceleracion_lineal[2] * AHRS->DCM_matriz[2][0] -
    AHRS->Vector_Aceleracion_lineal[0] * AHRS->DCM_matriz[2][2];
    error[2] = AHRS->Vector_Aceleracion_lineal[0] * AHRS->DCM_matriz[2][1] -
    AHRS->Vector_Aceleracion_lineal[1] * AHRS->DCM_matriz[2][0];

    arm_scale_f32((float32_t *)&error, AHRS->Kp_Roll_Pitch, AHRS->Correccion_Proporcional, 3);
    arm_scale_f32((float32_t *)&error, AHRS->Ki_Roll_Pitch, Aux, 3);
    arm_add_f32((float32_t *)&Aux, AHRS->Correccion_Integral, AHRS->Correccion_Integral, 3);

    //YAW
    arm_scale_f32(&AHRS->DCM_matriz[2][0],
    AHRS->DCM_matriz[0][0]*arm_sin_f32(AHRS->Orientacion_YAW) -
    AHRS->DCM_matriz[1][0]*arm_cos_f32(AHRS->Orientacion_YAW), error, 3);

    arm_scale_f32(error, AHRS->Kp_Yaw, Aux, 3);
    arm_add_f32((float32_t *)Aux, AHRS->Correccion_Proporcional,
    AHRS->Correccion_Proporcional, 3);

    arm_scale_f32(error, AHRS->Ki_Yaw, Aux, 3);
    arm_add_f32((float32_t *)&Aux, AHRS->Correccion_Integral, AHRS->Correccion_Integral, 3);

}

void Correccion_deriva_NO_YAW(tpAHRS *AHRS){
    float32_t error[3] = {0, 0, 0};
    float32_t Aux[3] = {0, 0, 0};

    //ROLL PITCH
    //faltaria filtrar???
    //Producto cruz
    error[0] = AHRS->Vector_Aceleracion_lineal[1] * AHRS->DCM_matriz[2][2] -
    AHRS->Vector_Aceleracion_lineal[2] * AHRS->DCM_matriz[2][1];
    error[1] = AHRS->Vector_Aceleracion_lineal[2] * AHRS->DCM_matriz[2][0] -
    AHRS->Vector_Aceleracion_lineal[0] * AHRS->DCM_matriz[2][2];
    error[2] = AHRS->Vector_Aceleracion_lineal[0] * AHRS->DCM_matriz[2][1] -
    AHRS->Vector_Aceleracion_lineal[1] * AHRS->DCM_matriz[2][0];

    arm_scale_f32((float32_t *)&error, AHRS->Kp_Roll_Pitch, AHRS->Correccion_Proporcional, 3);
    arm_scale_f32((float32_t *)&error, AHRS->Ki_Roll_Pitch, Aux, 3);
```

```
    arm_add_f32((float32_t *)&Aux, AHRS->Correccion_Integral, AHRS->Correccion_Integral, 3);
}

void Angulos_Euler(tpAHRS *AHRS){
    AHRS->Pitch = -asin(AHRS->DCM_matriz[2][0]);
    AHRS->Roll = atan2(AHRS->DCM_matriz[2][1],AHRS->DCM_matriz[2][2]);
    AHRS->Yaw = atan2(AHRS->DCM_matriz[1][0],AHRS->DCM_matriz[0][0]);
}

void ResetDCM(){
    tpLecturas_IMU Lecturas_IMU = {0, 0, 0, 0, 0, 0, 0 };
    tpLecturas_Brujula Lecturas_Brujula = {0, 0, 0};

    float32_t DCM_matriz[3][3] = {1, 0, 0, 0, 1, 0, 0, 0, 1};
    float32_t Roll = 0;
    float32_t Pitch = 0;
    float32_t Yaw = 0;

    float32_t sin_Roll = 0;
    float32_t cos_Roll = 0;
    float32_t sin_Pitch = 0;
    float32_t cos_Pitch = 0;
    float32_t sin_Yaw = 0;
    float32_t cos_Yaw = 0;

    Leer_servidor_Lecturas_IMU(&Lecturas_IMU);
    Leer_servidor_Lecturas_Brujula(&Lecturas_Brujula);

    Pitch = -atan2(Lecturas_IMU.Valor.x_acel, sqrt(Lecturas_IMU.Valor.y_acel *
    Lecturas_IMU.Valor.y_acel + Lecturas_IMU.Valor.z_acel * Lecturas_IMU.Valor.z_acel));
    Roll =   atan2(Lecturas_IMU.Valor.y_acel, sqrt(Lecturas_IMU.Valor.x_acel *
    Lecturas_IMU.Valor.x_acel + Lecturas_IMU.Valor.z_acel * Lecturas_IMU.Valor.z_acel));

    sin_Roll = arm_sin_f32(Roll);
    cos_Roll = arm_cos_f32(Roll);
    sin_Pitch = arm_sin_f32(Pitch);
    cos_Pitch = arm_cos_f32(Pitch);

    Yaw = -atan2( Lecturas_Brujula.Valor.Magnetismo_y * cos_Roll -
    Lecturas_Brujula.Valor.Magnetismo_z * sin_Roll, Lecturas_Brujula.Valor.Magnetismo_x *
    cos_Pitch + Lecturas_Brujula.Valor.Magnetismo_y * sin_Roll * sin_Pitch +
    Lecturas_Brujula.Valor.Magnetismo_z * cos_Roll * sin_Pitch);
    sin_Yaw = arm_sin_f32(Yaw);
    cos_Yaw = arm_cos_f32(Yaw);

    DCM_matriz[0][0] = cos_Pitch*cos_Yaw;
    DCM_matriz[0][1] = cos_Yaw*sin_Roll*sin_Pitch - cos_Roll*sin_Yaw;
    DCM_matriz[0][2] = sin_Roll*sin_Yaw + cos_Roll*cos_Yaw*sin_Pitch;

    DCM_matriz[1][0] = cos_Pitch*sin_Yaw;
    DCM_matriz[1][1] = cos_Roll*cos_Yaw + sin_Roll*sin_Pitch*sin_Yaw;
    DCM_matriz[1][2] = cos_Roll*sin_Pitch*sin_Yaw - cos_Yaw*sin_Roll;

    DCM_matriz[2][0] = -sin_Pitch;
    DCM_matriz[2][1] = cos_Pitch*sin_Roll;
    DCM_matriz[2][2] = cos_Roll*cos_Pitch;

    Escribir_servidor_DCM((float32_t*)DCM_matriz);
    Escribir_servidor_RPY(&Roll, &Pitch, &Yaw);
}

void Algortimo_DCM_MAG(tpAHRS *AHRS){

    Compensacion_Sensor_magnetico(AHRS);
    Actualizar_Matriz_DCM_V2(AHRS);
    Normalizar_DCM(AHRS);
    Correccion_deriva(AHRS);
    Angulos_Euler(AHRS);
```

```
}

void Algortimo_DCM_NO_YAW(tpAHRS *AHRS){

    Actualizar_Matriz_DCM_V2(AHRS);
    Normalizar_DCM(AHRS);
    Correccion_deriva_NO_YAW(AHRS);
    Angulos_Euler(AHRS);
}
```

```c
/*
 * AHRS.c
 *
 *  Created on: 22/12/2015
 *      Author: Ruben
 */
#include "AHRS.h"
#include "Servidores.h"

void Compensacion_Sensor_magnetico(tpAHRS *AHRS){
    //Rota el eje magnetico para alinearlo con el suelo, usando la ultima referencia ROLL
    PITCH
    float32_t mag_x;
    float32_t mag_y;
    float32_t cos_roll;
    float32_t sin_roll;
    float32_t cos_pitch;
    float32_t sin_pitch;

    cos_roll = arm_cos_f32(AHRS->Roll);
    sin_roll = arm_sin_f32(AHRS->Roll);
    cos_pitch = arm_cos_f32(AHRS->Pitch);
    sin_pitch = arm_sin_f32(AHRS->Pitch);

    // Rotamos
    mag_x = AHRS->Vector_Magnetico[0] * cos_pitch + AHRS->Vector_Magnetico[1] * sin_roll *
    sin_pitch + AHRS->Vector_Magnetico[2] * cos_roll * sin_pitch;
    mag_y = AHRS->Vector_Magnetico[1] * cos_roll - AHRS->Vector_Magnetico[2] * sin_roll;
    AHRS->Orientacion_YAW = atan2(-mag_y, mag_x);
}

void Actualizar_Matriz_DCM(tpAHRS *AHRS){
    float32_t Velocidad_Total[3] = {0, 0, 0};

    float32_t Rot_matriz[3][3] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
    arm_matrix_instance_f32 Rotacion = {3, 3, (float32_t *)Rot_matriz};

    float32_t Aux_matriz[9] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
    arm_matrix_instance_f32 Aux = {3, 3, Aux_matriz};

    Velocidad_Total[0] = AHRS->Vector_Velocidad_Angular[0] +
    AHRS->Correccion_Proporcional[0] + AHRS->Correccion_Integral[0];
    Velocidad_Total[1] = AHRS->Vector_Velocidad_Angular[1] +
    AHRS->Correccion_Proporcional[1] + AHRS->Correccion_Integral[1];
    Velocidad_Total[2] = AHRS->Vector_Velocidad_Angular[2] +
    AHRS->Correccion_Proporcional[2] + AHRS->Correccion_Integral[2];

    Rot_matriz[0][0] =  1;
    Rot_matriz[0][1] = -AHRS->Periodo_Muestreo*Velocidad_Total[2];//-z
    Rot_matriz[0][2] =  AHRS->Periodo_Muestreo*Velocidad_Total[1];//y
    Rot_matriz[1][0] =  AHRS->Periodo_Muestreo*Velocidad_Total[2];//z
    Rot_matriz[1][1] =  1;
    Rot_matriz[1][2] = -AHRS->Periodo_Muestreo*Velocidad_Total[0];//-x
    Rot_matriz[2][0] = -AHRS->Periodo_Muestreo*Velocidad_Total[1];//-y
    Rot_matriz[2][1] =  AHRS->Periodo_Muestreo*Velocidad_Total[0];//x
    Rot_matriz[2][2] =  1;

    arm_mat_mult_f32(&AHRS->DCM, &Rotacion, &Aux);
    arm_copy_f32(Aux.pData, AHRS->DCM.pData, 9);
}

void Actualizar_Matriz_DCM_V2(tpAHRS *AHRS){
    float32_t Velocidad_Total[3] = {0, 0, 0};
    float32_t Vector_Rotacion[3] = {0, 0, 0};

    float32_t Angulo_Rotacion = 0;
    float32_t Seno = 0;
    float32_t Coseno = 0;
```

```
    float32_t Rot_matriz[3][3] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
    arm_matrix_instance_f32 Rotacion = {3, 3, (float32_t *)Rot_matriz};

    float32_t Aux_matriz[9] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
    arm_matrix_instance_f32 Aux = {3, 3, Aux_matriz};

    Velocidad_Total[0] = AHRS->Vector_Velocidad_Angular[0] +
    AHRS->Correccion_Proporcional[0] + AHRS->Correccion_Integral[0];
    Velocidad_Total[1] = AHRS->Vector_Velocidad_Angular[1] +
    AHRS->Correccion_Proporcional[1] + AHRS->Correccion_Integral[1];
    Velocidad_Total[2] = AHRS->Vector_Velocidad_Angular[2] +
    AHRS->Correccion_Proporcional[2] + AHRS->Correccion_Integral[2];

    //Velocidad absoluta
    arm_sqrt_f32((Velocidad_Total[0]*Velocidad_Total[0] +
    Velocidad_Total[1]*Velocidad_Total[1] +
                  Velocidad_Total[2]*Velocidad_Total[2]), &Angulo_Rotacion);

    //Normalizamos vector rotacion
    if (Angulo_Rotacion != 0.0){
        arm_scale_f32(Velocidad_Total, 1/Angulo_Rotacion, Vector_Rotacion, 3);
    }

    //Pasamos de velocidad a angulo
    Angulo_Rotacion = Angulo_Rotacion*AHRS->Periodo_Muestreo;

    Coseno = arm_cos_f32(Angulo_Rotacion);
    Seno = arm_sin_f32(Angulo_Rotacion);

    Rot_matriz[0][0] = Coseno + Vector_Rotacion[0]*Vector_Rotacion[0]*(1 - Coseno);
    Rot_matriz[0][1] = Vector_Rotacion[0]*Vector_Rotacion[1]*(1 - Coseno) -
    Vector_Rotacion[2]*Seno;
    Rot_matriz[0][2] = Vector_Rotacion[0]*Vector_Rotacion[2]*(1 - Coseno) +
    Vector_Rotacion[1]*Seno;
    Rot_matriz[1][0] = Vector_Rotacion[1]*Vector_Rotacion[0]*(1 - Coseno) +
    Vector_Rotacion[2]*Seno;
    Rot_matriz[1][1] = Coseno + Vector_Rotacion[1]*Vector_Rotacion[1]*(1 - Coseno);
    Rot_matriz[1][2] = Vector_Rotacion[1]*Vector_Rotacion[2]*(1 - Coseno) -
    Vector_Rotacion[0]*Seno;
    Rot_matriz[2][0] = Vector_Rotacion[2]*Vector_Rotacion[0]*(1 - Coseno) -
    Vector_Rotacion[1]*Seno;
    Rot_matriz[2][1] = Vector_Rotacion[2]*Vector_Rotacion[1]*(1 - Coseno) +
    Vector_Rotacion[0]*Seno;
    Rot_matriz[2][2] = Coseno + Vector_Rotacion[2]*Vector_Rotacion[2]*(1 - Coseno);

    arm_mat_mult_f32(&AHRS->DCM, &Rotacion, &Aux);
    arm_copy_f32(Aux.pData, AHRS->DCM.pData, 9);
}

void Normalizar_DCM(tpAHRS *AHRS){
    float error = 0;

    float32_t Vector_Aux[3] = {0, 0, 0};
    float32_t Matriz_Ortogonal[3][3] = {0, 0, 0, 0, 0, 0, 0, 0, 0};

    arm_dot_prod_f32(&AHRS->DCM_matriz[0][0], &AHRS->DCM_matriz[1][0], 3, &error);
    error *= -0.5;

    arm_scale_f32(&AHRS->DCM_matriz[1][0], error, Vector_Aux, 3);
    arm_add_f32(&AHRS->DCM_matriz[0][0], Vector_Aux,  &Matriz_Ortogonal[0][0], 3); //Vector
    X ortogonal

    arm_scale_f32(&AHRS->DCM_matriz[0][0], error, Vector_Aux, 3);
    arm_add_f32(&AHRS->DCM_matriz[1][0], Vector_Aux,  &Matriz_Ortogonal[1][0], 3); //Vector
    Y ortogonal

    //Producto Cruz
```

```
    Matriz_Ortogonal[2][0] = Matriz_Ortogonal[0][1] * Matriz_Ortogonal[1][2] -
    Matriz_Ortogonal[0][2] * Matriz_Ortogonal[1][1];
    Matriz_Ortogonal[2][1] = Matriz_Ortogonal[0][2] * Matriz_Ortogonal[1][0] -
    Matriz_Ortogonal[0][0] * Matriz_Ortogonal[1][2];
    Matriz_Ortogonal[2][2] = Matriz_Ortogonal[0][0] * Matriz_Ortogonal[1][1] -
    Matriz_Ortogonal[0][1] * Matriz_Ortogonal[1][0  ];

    arm_dot_prod_f32(&Matriz_Ortogonal[0][0], &Matriz_Ortogonal[0][0], 3, &error);
    error = 0.5*(3.0 - error);
    arm_scale_f32(&Matriz_Ortogonal[0][0], error, &AHRS->DCM_matriz[0][0], 3);

    arm_dot_prod_f32(&Matriz_Ortogonal[1][0], &Matriz_Ortogonal[1][0], 3, &error);
    error = 0.5*(3.0 - error);
    arm_scale_f32(&Matriz_Ortogonal[1][0], error, &AHRS->DCM_matriz[1][0], 3);

    arm_dot_prod_f32(&Matriz_Ortogonal[2][0], &Matriz_Ortogonal[2][0], 3, &error);
    error = 0.5*(3.0 - error);
    arm_scale_f32(&Matriz_Ortogonal[2][0], error, &AHRS->DCM_matriz[2][0], 3);
}

void Correccion_deriva(tpAHRS *AHRS){
    float32_t error[3] = {0, 0, 0};
    float32_t Aux[3] = {0, 0, 0};

    //ROLL PITCH
    //faltaria filtrar???
    //Producto cruz
    error[0] = AHRS->Vector_Aceleracion_lineal[1] * AHRS->DCM_matriz[2][2] -
    AHRS->Vector_Aceleracion_lineal[2] * AHRS->DCM_matriz[2][1];
    error[1] = AHRS->Vector_Aceleracion_lineal[2] * AHRS->DCM_matriz[2][0] -
    AHRS->Vector_Aceleracion_lineal[0] * AHRS->DCM_matriz[2][2];
    error[2] = AHRS->Vector_Aceleracion_lineal[0] * AHRS->DCM_matriz[2][1] -
    AHRS->Vector_Aceleracion_lineal[1] * AHRS->DCM_matriz[2][0];

    arm_scale_f32((float32_t *)&error, AHRS->Kp_Roll_Pitch, AHRS->Correccion_Proporcional, 3);
    arm_scale_f32((float32_t *)&error, AHRS->Ki_Roll_Pitch, Aux, 3);
    arm_add_f32((float32_t *)&Aux, AHRS->Correccion_Integral, AHRS->Correccion_Integral, 3);

    //YAW
    arm_scale_f32(&AHRS->DCM_matriz[2][0],
    AHRS->DCM_matriz[0][0]*arm_sin_f32(AHRS->Orientacion_YAW) -
    AHRS->DCM_matriz[1][0]*arm_cos_f32(AHRS->Orientacion_YAW), error, 3);

    arm_scale_f32(error, AHRS->Kp_Yaw, Aux, 3);
    arm_add_f32((float32_t *)Aux, AHRS->Correccion_Proporcional,
    AHRS->Correccion_Proporcional, 3);

    arm_scale_f32(error, AHRS->Ki_Yaw, Aux, 3);
    arm_add_f32((float32_t *)&Aux, AHRS->Correccion_Integral, AHRS->Correccion_Integral, 3);

}

void Correccion_deriva_NO_YAW(tpAHRS *AHRS){
    float32_t error[3] = {0, 0, 0};
    float32_t Aux[3] = {0, 0, 0};

    //ROLL PITCH
    //faltaria filtrar???
    //Producto cruz
    error[0] = AHRS->Vector_Aceleracion_lineal[1] * AHRS->DCM_matriz[2][2] -
    AHRS->Vector_Aceleracion_lineal[2] * AHRS->DCM_matriz[2][1];
    error[1] = AHRS->Vector_Aceleracion_lineal[2] * AHRS->DCM_matriz[2][0] -
    AHRS->Vector_Aceleracion_lineal[0] * AHRS->DCM_matriz[2][2];
    error[2] = AHRS->Vector_Aceleracion_lineal[0] * AHRS->DCM_matriz[2][1] -
    AHRS->Vector_Aceleracion_lineal[1] * AHRS->DCM_matriz[2][0];

    arm_scale_f32((float32_t *)&error, AHRS->Kp_Roll_Pitch, AHRS->Correccion_Proporcional, 3);
    arm_scale_f32((float32_t *)&error, AHRS->Ki_Roll_Pitch, Aux, 3);
```

```
    arm_add_f32((float32_t *)&Aux, AHRS->Correccion_Integral, AHRS->Correccion_Integral, 3);
}

void Angulos_Euler(tpAHRS *AHRS){
    AHRS->Pitch = -asin(AHRS->DCM_matriz[2][0]);
    AHRS->Roll = atan2(AHRS->DCM_matriz[2][1],AHRS->DCM_matriz[2][2]);
    AHRS->Yaw = atan2(AHRS->DCM_matriz[1][0],AHRS->DCM_matriz[0][0]);
}

void ResetDCM(){
    tpLecturas_IMU Lecturas_IMU = {0, 0, 0, 0, 0, 0, 0 };
    tpLecturas_Brujula Lecturas_Brujula = {0, 0, 0};

    float32_t DCM_matriz[3][3] = {1, 0, 0, 0, 1, 0, 0, 0, 1};
    float32_t Roll = 0;
    float32_t Pitch = 0;
    float32_t Yaw = 0;

    float32_t sin_Roll = 0;
    float32_t cos_Roll = 0;
    float32_t sin_Pitch = 0;
    float32_t cos_Pitch = 0;
    float32_t sin_Yaw = 0;
    float32_t cos_Yaw = 0;

    Leer_servidor_Lecturas_IMU(&Lecturas_IMU);
    Leer_servidor_Lecturas_Brujula(&Lecturas_Brujula);

    Pitch = -atan2(Lecturas_IMU.Valor.x_acel, sqrt(Lecturas_IMU.Valor.y_acel *
    Lecturas_IMU.Valor.y_acel + Lecturas_IMU.Valor.z_acel * Lecturas_IMU.Valor.z_acel));
    Roll =   atan2(Lecturas_IMU.Valor.y_acel, sqrt(Lecturas_IMU.Valor.x_acel *
    Lecturas_IMU.Valor.x_acel + Lecturas_IMU.Valor.z_acel * Lecturas_IMU.Valor.z_acel));

    sin_Roll = arm_sin_f32(Roll);
    cos_Roll = arm_cos_f32(Roll);
    sin_Pitch = arm_sin_f32(Pitch);
    cos_Pitch = arm_cos_f32(Pitch);

    Yaw = -atan2( Lecturas_Brujula.Valor.Magnetismo_y * cos_Roll -
    Lecturas_Brujula.Valor.Magnetismo_z * sin_Roll, Lecturas_Brujula.Valor.Magnetismo_x *
    cos_Pitch + Lecturas_Brujula.Valor.Magnetismo_y * sin_Roll * sin_Pitch +
    Lecturas_Brujula.Valor.Magnetismo_z * cos_Roll * sin_Pitch);
    sin_Yaw = arm_sin_f32(Yaw);
    cos_Yaw = arm_cos_f32(Yaw);

    DCM_matriz[0][0] = cos_Pitch*cos_Yaw;
    DCM_matriz[0][1] = cos_Yaw*sin_Roll*sin_Pitch - cos_Roll*sin_Yaw;
    DCM_matriz[0][2] = sin_Roll*sin_Yaw + cos_Roll*cos_Yaw*sin_Pitch;

    DCM_matriz[1][0] = cos_Pitch*sin_Yaw;
    DCM_matriz[1][1] = cos_Roll*cos_Yaw + sin_Roll*sin_Pitch*sin_Yaw;
    DCM_matriz[1][2] = cos_Roll*sin_Pitch*sin_Yaw - cos_Yaw*sin_Roll;

    DCM_matriz[2][0] = -sin_Pitch;
    DCM_matriz[2][1] = cos_Pitch*sin_Roll;
    DCM_matriz[2][2] = cos_Roll*cos_Pitch;

    Escribir_servidor_DCM((float32_t*)DCM_matriz);
    Escribir_servidor_RPY(&Roll, &Pitch, &Yaw);
}

void Algortimo_DCM_MAG(tpAHRS *AHRS){

    Compensacion_Sensor_magnetico(AHRS);
    Actualizar_Matriz_DCM_V2(AHRS);
    Normalizar_DCM(AHRS);
    Correccion_deriva(AHRS);
    Angulos_Euler(AHRS);
```

```
}

void Algortimo_DCM_NO_YAW(tpAHRS *AHRS){

    Actualizar_Matriz_DCM_V2(AHRS);
    Normalizar_DCM(AHRS);
    Correccion_deriva_NO_YAW(AHRS);
    Angulos_Euler(AHRS);
}
```

```
#ifndef QUADROTOR_V1_3_1_FUNCIONES_TRANSFERENCIA_H_
#define QUADROTOR_V1_3_1_FUNCIONES_TRANSFERENCIA_H_

#define Jq1 0.8613
#define Jq2 0.8613
#define Jq3 0.8613

#define Jm  0.096369

#define Km 0.009637    //Fuerza(N) / Accion (microseg)
#define K_sistema 1.8931
#define K_q K_sistema/Km;

#define Wn 10;
#define Wn_2 10;
#define chi 1;

#define Kpr 0.0
/*
Kp1 = Wn^2*Jq1/K;
Kp2 = Wn^2*Jq2/K;
Kp3 = Wn^2*Jq3/K2;

Kv1 = (2*chi*Wn - 1) / K;
Kv2 = (2*chi*Wn - 1) / K;
Kv3 = (2*chi*Wn - 1) / K2;
 */

/*
#define Kp1 0
#define Kv1 0

#define Kp2 0
#define Kv2 0
*/

#define Kp1 15.0
#define Kv1 2.5

#define Kp2 15.0
#define Kv2 2.5

#define Kp3 15.0
#define Kv3 2.5

#define Ki 0.02
#define Ki_EST 2


const float32_t F_matriz[] = {
           0.994211639269529,                    0,                    0,
           0,                    0,                    0,
           1.108053901852224,                    0,                    0,                    0,
           0.004985515097154,    1.000000000000000,                    0,
           0,                    0,                    0,
           0.002796838162984,                    0,                    0,                    0,
                             0,                    0,    0.994211639269529,
                             0,                    0,                    0,
                             0,    1.108053901852224,                    0,                    0,
                             0,                    0,    0.004985515097154,
                             1.000000000000000,                    0,
                             0,                    0,    0.002796838162984,
                             0,                    0,
                             0,                    0,                    0,
                             0,    0.994211639269529,                    0,
                             0,                    0,    0.166208085277834,                    0,
                             0,                    0,                    0,
```

```
                        0,   0.004985515097154,   1.000000000000000,
                        0,                   0,   0.000419525724448,                   0,
                        0,                   0,                   0,
                        0,                   0,                   0,
                        0.949439085978879,                   0,
                        0,                   0,
                        0,                   0,                   0,
                        0,                   0,                   0,
                        0,   0.949439085978879,                   0,                   0,
                        0,                   0,                   0,
                        0,                   0,                   0,
                        0,                   0,   0.949439085978879,                   0,
                        0,                   0,                   0,
                        0,                   0,                   0,
                        0,                   0,                   0,   0.949439085978879
};
const arm_matrix_instance_f32 F = {10, 10, (float32_t *)F_matriz};


const float32_t G_matriz[] = {
          0.000279686718516044,                   0,
          -0.000279686718516044,                   0,
          0.000000468381516574,                   0,
          -0.000000468381516574,                   0,
                        0,   0.000279686718516044,                   0,
                        -0.000279686718516044,
                        0,   0.000000468381516574,                   0,
                        -0.000000468381516574,
          0.000041953007777407,  -0.000041953007777407,   0.000041953007777407,
          -0.000041953007777407,
          0.000000070257227486,  -0.000000070257227486,   0.000000070257227486,
          -0.000000070257227486,
          0.000487255528421544,                   0,
          -0.000487255528421544,                   0,
                        0,   0.000487255528421544,                   0,
                        -0.000487255528421544,
          0.000487255528421544,  -0.000487255528421544,   0.000487255528421544,
          -0.000487255528421544,
          0.000487255528421544,   0.000487255528421544,   0.000487255528421544,
          0.000487255528421544
};
const arm_matrix_instance_f32 G = {10, 4, (float32_t *)G_matriz};


const float32_t Gp_matriz[] = {

         -0.029022176872060,                   0,                   0,                   0,
          -0.000048602419485,                   0,                   0,                   0,
                        0,  -0.029022176872060,                   0,                   0,
                        0,  -0.000048602419485,                   0,                   0,
                        0,                   0,  -0.004353326530809,                   0,
                        0,                   0,  -0.000007290362923,                   0,
          -0.050560914021121,                   0,                   0,                   0,
                        0,  -0.050560914021121,                   0,                   0,
                        0,                   0,  -0.050560914021121,                   0,
                        0,                   0,                   0,  -0.050560914021121

};
const arm_matrix_instance_f32 Gp = {10, 4, (float32_t *)Gp_matriz};


const float32_t K_4_matriz[] = {  Kv1/2.0,   Kp1/2.0,         0,          0,    Kv3/4.0,
Kp3/4.0,   Kpr/2.0,          0,    Kpr/4.0,    Kpr/4.0,
                                        0,          0,  Kv2/2.0,   Kp2/2.0,   -Kv3/4.0,
                                       -Kp3/4.0,          0,   Kpr/2.0,   -Kpr/4.0,
                                       Kpr/4.0,
                                 -Kv1/2.0,  -Kp1/2.0,         0,          0,    Kv3/4.0,
                                 Kp3/4.0,  -Kpr/2.0,          0,    Kpr/4.0,    Kpr/4.0,
```

```
                                        0,          0, -Kv2/2.0,  -Kp2/2.0,   -Kv3/4.0,
                                        -Kp3/4.0,          0,  -Kpr/2.0,   -Kpr/4.0,
                                        Kpr/4.0   };
const arm_matrix_instance_f32 K_4 = {4, 10, (float32_t *)K_4_matriz};

const float32_t K_3_matriz[] = {  Kv1/2,   Kp1/2,        0,        0,     Kv3/4,
Kp3/4,          0,          0,   0,    0,
                                      0,        0,   Kv2/2,   Kp2/2,   - Kv3/4,
                                     -Kp3/4,          0,          0,   0,    0,
                                -Kv1/2,  -Kp1/2,        0,        0,     Kv3/4,
                                Kp3/4,          0,          0,   0,    0,
                                      0,        0,  -Kv2/2,  -Kp2/2,   - Kv3/4,
                                     -Kp3/4,          0,          0,   0,    0    };
const arm_matrix_instance_f32 K_3 = {4, 10, (float32_t *)K_3_matriz};

/*
const float32_t K_pre_4_matriz [] = {       Kp1/2.0,          0,   Kp3/4.0,
Kpr/4.0*(1.0+1.0/(Km*Kpr)),
                                            0,   Kp2/2.0,  -Kp3/4.0,
                                            Kpr/4.0*(1.0+1.0/(Km*Kpr)),
                                       -Kp1/2.0,          0,   Kp3/4.0,
                                       Kpr/4.0*(1.0+1.0/(Km*Kpr)),
                                            0,  -Kp2/2.0,  -Kp3/4.0,
                                            Kpr/4.0*(1.0+1.0/(Km*Kpr))   };
*/
const float32_t K_pre_4_matriz [] = {       Kp1/2.0,          0,   Kp3/4.0,    0,
                                            0,   Kp2/2.0,  -Kp3/4.0,    0,
                                       -Kp1/2.0,          0,   Kp3/4.0,    0,
                                            0,  -Kp2/2.0,  -Kp3/4.0,    0   };

const arm_matrix_instance_f32 K_pre_4 = {4, 4, (float32_t *)K_pre_4_matriz};

const float32_t K_pre_3_matriz [] = {       Kp1/2,        0,   Kp3/4,    1,
                                            0,   Kp2/2,  -Kp3/4,    1,
                                       -Kp1/2,        0,   Kp3/4,    1,
                                            0,  -Kp2/2,  -Kp3/4,    1   };
const arm_matrix_instance_f32 K_pre_3 = {4, 4, (float32_t *)K_pre_3_matriz};

const float32_t Lo_per_matriz[] = {

          0.198434923933560,  -0.006363850507977,  -0.007659457807024,
          -0.027653725397731,  -0.040303351039343,   0.032195672338781,
          0,                    0,                    0,  -0.119404737466306,
          0.001086363950669,   0.188835322750985,  -0.016657957077870,
          -0.024043417453002,  -0.020908295732614,  -0.011056332513453,
          0,                    0,                    0,   0.001170982384024,
         -0.005045608483105,  -0.048871496093713,   0.157744315915222,
         -0.041854699495902,   0.044350331784821,  -0.024550313312358,
         0,                    0,                    0,   0.016240261993865,
         -0.005309682991509,  -0.023052431159453,  -0.007214113163155,
         0.201894030185689,  -0.019095064558748,  -0.004329015112383,
         0,                    0,                    0,   0.004022347367876,
         -0.002378769745064,  -0.002178446370954,   0.002055236651581,
         -0.002107845235214,   0.111605875609966,   0.002983178665231,
         0,                    0,                    0,   0.004293308919750,
          0.003865396849945,  -0.011754342406059,  -0.008725820535339,
          -0.005326913222497,   0.035796312502292,   0.207633665528118,
          0,                    0,                    0,  -0.002449943423081,
          0.037373088816668,   0.008171926131680,  -0.000749832805752,
          0.012867191640016,   0.003870818022841,  -0.007501931764926,
          0,                    0,                    0,  -0.065266474827869,
         -0.000255055739259,   0.049442260602111,   0.037401895624638,
         0.033213986376183,  -0.001506645855643,   0.025470558191721,
         0,                    0,                    0,   0.001253319942105,
          0.000379721483837,   0.022713862804404,  -0.000356562869897,
          0.021199489086857,   0.258298575110510,  -0.036706281736491,
          0,                    0,                    0,   0.033732925270074,
          0.021869479811627,   0.001408963971122,  -0.000641233000648,
```

```
          0.010283330987783,  -0.030403824233784,  -0.009898280860548,
          0,                   0,                   0,   0.217541943060802

};
const arm_matrix_instance_f32 Lo_per = {10, 10, (float32_t *)Lo_per_matriz};

const float32_t Lp_matriz[] = {

          -0.105325610557535,  -0.002714467035512,   0.0,
          0.0,                 0.0,                                  0.0,
          0,                   0,                   0,   0.0,
           0.0,                -0.0,                -0.105325610557535,
           -0.002714467035512,  -0.0,                                -0.0,
           0,                   0,                   0,  -0.0,
           0.0,                -0.0,                -0.0,
           -0.0,                -0.105325610557535,   -0.002714467035512,
           0,                   0,                   0,  -0.0,
          -0.0,                -0.0,                 0.0,
          -0.0,                 0.0,                                  0.0,
          0,                   0,                   0,  -0.278795314926213

/*
          -0.105325610557535,   0.002714467035512,   0.008732682462314,
          0.002559603908597,   0.014552758777108,   0.000528954644569,
          0,                   0,                   0,   0.102919655628366,
           0.008312078498193,  -0.003819091813717,  -0.075060605398528,
           -0.002074987127944,  -0.019057795543476,  -0.005361511672757,
           0,                   0,                   0,  -0.001723807565582,
           0.004362113688891,  -0.013310664761345,  -0.004605587466494,
           -0.014805749036137,  -0.378215610185594,   0.031471607720973,
           0,                   0,                   0,  -0.048012960151969,
          -0.055502910072872,  -0.000414764856061,   0.005678012486231,
          -0.012687401959021,   0.057454318249112,   0.016414143824185,
          0,                   0,                   0,  -0.278795314926213
*/
};
const arm_matrix_instance_f32 Lp = {4, 10, (float32_t *)Lp_matriz};

const float32_t La_matriz[] = {
/*
         -51.883366192798583,                   0, -25.941683096399295, -25.941683096399299,
                           0, -51.883366192798583,  25.941683096399295, -25.941683096399299,
          51.883366192798583,                   0, -25.941683096399302, -25.941683096399302,
                           0,  51.883366192798583,  25.941683096399302, -25.941683096399302
*/

         -51.883366192798583,                   0, 0, 0,
                           0, -51.883366192798583, 0, 0,
          51.883366192798583,                   0, 0, 0,
                           0,  51.883366192798583, 0, 0

};
const arm_matrix_instance_f32 La = {4, 4, (float32_t *)La_matriz};




//Parametros Filtros
#define Kc 1
#define numero_muestras_filtro_media 5


#define Kp_ROLLPITCH 0.00005
#define Ki_ROLLPITCH 0.00000001
//#define Kp_YAW 0.00005
//#define Ki_YAW 0.00000001

#define Kp_YAW 0.00005
```

```
#define Ki_YAW 0.00000001

//50Hz
/*
#define num_etapas_Filtro_Vel 2
const float32_t Coeficientes_Filtro_Vel_Valores[5*num_etapas_Filtro_Vel] =  {
0.361615673042922, 2*0.361615673042922, 0.361615673042922,    0,  -0.446462692171690,
0.259891532474145,  2*0.259891532474145, 0.259891532474145, 0,  -0.0395661298965801};
*/

#define num_etapas_Filtro_Per 2
const float32_t Coeficientes_Filtro_Pre_Valores[5*num_etapas_Filtro_Per] =  {
0.361615673042922, 2*0.361615673042922, 0.361615673042922,    0,  -0.446462692171690,
0.259891532474145,  2*0.259891532474145, 0.259891532474145, 0,  -0.0395661298965801};

//Filtro 20Hz
#define num_etapas_Filtro_Vel 2
const float32_t Coeficientes_Filtro_Vel_Valores[5*num_etapas_Filtro_Vel]  =  {
0.00376220298169900, 2*0.00376220298169900,  0.00376220298169900, 1.89341560102250,
-0.908464412949295, 0.00353349592337797, 2*0.00353349592337797,  0.00353349592337797,
1.77831348813944, -0.792447471832947};

#define num_etapas_Filtro_Acel 2
const float32_t Coeficientes_Filtro_Acel_Valores[5*num_etapas_Filtro_Acel] = {
0.00376220298169900, 2*0.00376220298169900,  0.00376220298169900, 1.89341560102250,
-0.908464412949295, 0.00353349592337797, 2*0.00353349592337797,  0.00353349592337797,
1.77831348813944, -0.792447471832947};

#define Coeficientes_Filtro_Mag_Valores {0, 0.0425, 0.0367, 1.565, -0.6442}
#define num_etapas_Filtro_Mag 1

/*
#define num_etapas_Filtro_Per 2
const float32_t Coeficientes_Filtro_Pre_Valores[5*num_etapas_Filtro_Per] =  {
0.0779563405164626, 2*0.0779563405164626, 0.0779563405164626, 1.32091343081943,
-0.632738792885277, 0.0618851952997645, 2*0.0618851952997645, 0.0618851952997645,
1.04859957636261, -0.296140357561670 };
*/

//10Hz
/*
#define num_etapas_Filtro_Per 2
const float32_t Coeficientes_Filtro_Per_Valores[5*num_etapas_Filtro_Per] =  {
0.0218838519679430,   2*0.0218838519679430,   0.0218838519679430, 1.70096433194353,
-0.788499739815298, 0.0190368315878239, 2*0.0190368315878239, 0.0190368315878239,
1.47967421693119, -0.555821543282489};
*/

#endif /* QUADROTOR_V1_3_1_FUNCIONES_TRANSFERENCIA_H_ */
```

```
#ifndef PARAMETROS_DEF
#define PARAMETROS_DEF

#include "arm_math.h"

#define Frecuencia_CPU (uint32_t)80000000UL

#define BAUD_RATE_9600 (uint32_t)9600UL
#define BAUD_RATE_115200 (uint32_t)115200UL
#define BAUD_RATE_128000 (uint32_t)128000UL
#define BAUD_RATE_460800 (uint32_t)460800UL
#define BITRATE_SPI (uint32_t)1000000UL

//Integrado AUX
#define Dir_AUX 0x01

//Parametros Tareas
#define PRIORIDAD_Leer_IMU 10
#define PERIODO_Leer_IMU 1
#define Timeout_Clk_Leer_IMU 1

#define PRIORIDAD_Calculo_AHRS 9
#define PERIODO_Calculo_AHRS 2
#define Timeout_Clk_Calculo_AHRS 250

#define PRIORIDAD_Control 8
#define PERIODO_Control 5
#define Timeout_Clk_Control 5

#define PRIORIDAD_Identificacion 8
#define PERIODO_Identificacion 5
#define Timeout_Clk_Identificacion 5

#define PRIORIDAD_Coordinador 7
#define PERIODO_Coordinador 25
#define Timeout_Clk_Coordinador 25

#define PRIORIDAD_Calculo_Altura 6
#define PERIODO_Calculo_Altura 60
#define Timeout_Clk_Calculo_Altura 60

//Parametros Funcionamiento
#define numCanales 8
#define ticks_arranque_vuelo 3*1000/PERIODO_Coordinador
#define Numero_Muetras_calibracion_IMU 1000
#define Numero_Muetras_calibracion_Brujula 100
#define Longitud_buffer 9000
#define Num_intentos_conexion_Identificacion 5

typedef enum {VUELO, ESPERA, ERROR, IDENTIFICACION, CALIBRACION, DEBUG, ERROR_CONEXION}
tpEstado_Sistema;
//typedef enum {IDENTIFICACION_EJE, IDENTIFICACION_MOTOR,
IDENTIFIACION_NULA}tpEstado_Identificacion;
typedef enum {NO_TELEMETRIA, TELEMETRIA_CONTROL, TELEMETRIA_YPR, TELEMETRIA_IMU,
TELEMETRIA_BRUJULA}tpModoTelemetria;
typedef enum {CORREGIR_PERTURBACIONES, NO_CORREGIR_PERTURBACIONES, INTEGRAR_PERTURBACIONES,
INTEGRAR_PERTURBACIONES_ESTIMADAS}tpModoPerturbaciones;
typedef enum {ANGULOS_4, ANGULOS_3, ANGULOS_1, EMPUJE}tpModo_Control;
typedef enum {PARADA_EMER, ESTABILIZACION_EMER, ATERIZAJE_EMER, NO_WATCHDOG}tpModoWatchdog;
typedef enum {CALIBRACION_COMPLETA_IMU, CALIBRACION_GIROSCOPO, CALIBRACION_ACELEROMETRO,
NO_CALIBRAR_IMU}tpModoCalibracionIMU;
typedef enum {START = '#', FINAL = '*', IDENTIFICAR ='I', CALIBRAR = 'C', TELEMETRIA = 'T',
DATO_ANTERIOR = 0xFF } tpOrden;
typedef enum {TELE_0, TELE_1, TELE_2, TELE_3, TELE_4} tpInfoTelemetria;
// TELE_0 Telemetria Control, Sin perturcaciones, control 3 Angulos
// TELE_1 Telemetria Control, perturcaciones, control 3 Angulos
// TELE_2 Telemetria Control, perturcaciones integradas estimadas, control 3 Angulos
// TELE_3 Telemetria Control, perturcaciones integradas, control 3 Angulos
```

```
// TELE_4 Telemetria Control, EMPUJE


typedef struct{
    uint8_t Inicio;
    uint8_t InfoTelemetria;
    int16_t Referencia[4];
    uint16_t Accion[4];
    int16_t Variables_Estado[10];
    int16_t Perturbaciones[4];
    uint16_t Altura_Barometrica;
    uint16_t Altura_US;
    int16_t Acel[3];
    int16_t Gyro[3];
    int16_t Magnetics[3];
    uint8_t Final;
}tpTelemetria_Control;

typedef struct{
    uint8_t Inicio;
    int16_t Yaw;
    int16_t Pitch;
    int16_t Roll;
    uint8_t Final;
}tpTelemetria_YPR;

typedef struct{
    tpOrden Inicio;
    uint16_t Acel[3];
    uint16_t Gyro[3];
    uint16_t Mag[3];
    tpOrden Final;
}tpTelemetria_IMU;

typedef struct{
    float32_t Yaw_offset;
    float32_t Pitch_offset;
    float32_t Roll_offset;
}tpPuntoInicial;

//VALORES MAXIMOS
#define Angulo_Maximo 30.0//º
#define Velocidad_Angular_Maxima 10.0 //º/s
#define Valor_Empuje_Maximo 800
#define Valor_Fuerza_Maximo 2.0
#define Valor_perturbacion_Motor_MAX 5.0
#define Valor_perturbacion_Motor_MIN 0.0

//VALORES CONVERSION Q_16
#define Angulo_Max_Q16 180.0
#define Velocidad_Max_Q16 1000.0
#define F_Max_Q16 100.0

//TIMEOUTS
#define Timeout_UART_BLUETOOTH_Lectura 10

//TAREAS
#define nTokensIniciales_0 0
#define nTokensIniciales_1 1

//Parametros Ultrasonido US
#define Pulso_arranque_us 10 //us
#define Distancia_MAX_us 4 //m
#define Velocidad_Sonido 340 // m/s
#define Max_pulso_us 25000 // (4.25 metros)
#define Min_pulso_us 120 // 2 cm

typedef enum{INT_TRIGGER, INT_LLEGADA}tpIntUS;
```

```
//Parametros Motor
#define Offset_rpm 1174
#define Accion_Maxima 1000
#define Accion_Minima 100

#define Pulso_minimo_PWM_motor 1000U
#define Pulso_maximo_PWM_motor 2000U

//Parametros Chasis
#define MasaQuadrotor 0.9 //Kg
#define Pert_Fuerza_Bateria 0.3 //N

#endif
```

```
/*
AHRS como tarea
Todos Sensores IMU en una tarea.
Telemetria sensores sin filtrar. paso de dato con buzon
Añadida correcccion de posicion mediante rotacion de la medidas, con calculo de matriz al
inicio del ciclo



Sincronizacion de arranaque cambiada, las tareas inician su propio clock excepto el IMU
IMU->Se calibra, Guarda la primera lectura, Inicio AHRS , Tarea ciclico.
                                            *Reset de los parametros, Guarda la primera
                                            lectura, Inicio Coordinador , Tarea ciclico.

     Arranca WD, SYNCRO, Tarea ciclico.


*/

/* PARAMETROS CONFIGURACION */
#define Estimador_Parcial
//#define Filtro_Perturbaciones
#define Filtrado_Vel_IMU

#define IMU_MPU9250
//#define IMU_MPU6050
//#define GYRO_L3G4200
//#define COMPASS_HMC5883L
#define BAR_BMP280
//#define Sensor_RPM

//#define MAG


/* XDCtools Header files */
#include <xdc/std.h>
#include <xdc/cfg/global.h>
#include <xdc/runtime/System.h>
#include <xdc/runtime/Error.h>
#include <xdc/runtime/Memory.h>
//#include <xdc/runtime/IHeap.h>

/* BIOS Header files */
#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/Knl/Task.h>
#include <ti/sysbios/Knl/Clock.h>
#include <ti/sysbios/Knl/Semaphore.h>
#include <ti/sysbios/Knl/Mailbox.h>
#include <ti/sysbios/Knl/Swi.h>
#include <ti/sysbios/gates/GateMutexPri.h>
#include <ti/sysbios/hal/Timer.h>
#include <ti/sysbios/hal/Hwi.h>

/* TI-RTOS Header files */
#include <ti/drivers/GPIO.h>
#include <ti/drivers/I2C.h>
#include <ti/drivers/PWM.h>
#include <ti/drivers/UART.h>
#include <ti/drivers/SPI.h>
#include <ti/drivers/Watchdog.h>

/* Periphals libraries */
#include <driverlib/eeprom.h>

/* Board Header file */
#include "Quad_Board.h"
#include "arm_math.h"
```

```
#include "Parametros.h"
#include "Funciones_Transferencia.h"

#include "AHRS.h"
#include "Sensores.h"
#include "Servidores.h"
//#include "Transmisores.h"

//.................Variables.....................................................//
//...Sistema....//
Ptr Datos;

tpCalibracion_Receptor Calibracion_Receptor = { //POENR PROTEGIDO???
        {.Rango_Salida = {-1, 1},
         .Rango_Entrada = {966, 1973}
        },
        {.Rango_Salida = {-1, 1},
         .Rango_Entrada = {966, 1973}
        },
        {.Rango_Salida = {0, 1},
         .Rango_Entrada = {966, 1973}
        },
        {.Rango_Salida = {-1, 1},
         .Rango_Entrada = {966, 1973}
        },
        {.Rango_Salida = {-1, 1},
         .Rango_Entrada = {966, 1973}
        },
        {.Rango_Salida = {-1, 1},
         .Rango_Entrada = {966, 1973}
        },
        {.Rango_Salida = {-1, 1},
         .Rango_Entrada = {966, 1973}
        },
        {.Rango_Salida = {-100, 100},
         .Rango_Entrada = {966, 1973}
        }
};
tpCalibracion_IMU Calibracion_IMU = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        .Correccion_Alineamiento_matriz = {1, 0, 0, 0, 1, 0, 0, 0, 1},
        .Correccion_Alineamiento.numCols = 3,
        .Correccion_Alineamiento.numRows = 3,
        .Correccion_Alineamiento.pData = Calibracion_IMU.Correccion_Alineamiento_matriz,
        .Giro = {47.5, 0, 0}
};
tpCalibracion_Brujula Calibracion_Brujula = {0, 0, 0, 0, 0, 0};
tpModoCalibracionIMU ModoCalibracionIMU = CALIBRACION_COMPLETA_IMU;
//CALIBRACION_COMPLETA_IMU;

tpEstado_Sistema Estado_Sistema  = ESPERA;
tpEstado_Sistema Estado_Sistema_Anterior = ESPERA;

tpModoTelemetria ModoTelemetria = TELEMETRIA_CONTROL;
tpInfoTelemetria InfoTelemetria = TELE_0;
const tpOrden START_FRAME = START;
const tpOrden FINAL_FRAME = FINAL;

tpModo_Control Modo_Control = ANGULOS_3;
tpModo_Control Modo_Control_Anterior = ANGULOS_3;
tpModoPerturbaciones ModoPerturbaciones = NO_CORREGIR_PERTURBACIONES;
//NO_CORREGIR_PERTURBACIONES;

tpModoWatchdog ModoWatchdog = PARADA_EMER;


//....SENSORES....//
```

```
#ifdef IMU_MPU9250
tpIMU9250 IMU9250 = {
    .Direccion_IMU = DIR_0_IMU_MPU9250,
    .Direccion_MAG = Dir_MAG_MPU9250,
    .SMPLRT_DIV = 0,
    .Ganancia_Acel = Gain_Acel_16G,
    .Sensibilidad_Acel = 32768.0/16.0,
    .Ganancia_Gyro = Gain_Gyro_500,
    .Sensibilidad_Giroscopo = 32768.0 / 500.0,
    .Sensibilidad_Brujula = 32768.0/ 4800.0,

    .DLPF_CFG_ACEL = DLPF_CFG_ACEL_NO,
    .DLPF_CFG_GYRO = DLPF_CFG_GYRO_NO
};
#endif

#ifdef IMU_MPU6050
tpIMU6050 IMU6050 = {
    .Direccion = Dir_0_IMU_MPU6050,
    .SMPLRT_DIV = 0,
    .DLPF_CFG = DLPF_CFG_1,
    .Ganancia_Gyro = Gain_Gyro_500,
    .Ganancia_Acel = Gain_Acel_16G,
    .Sensibilidad_Giroscopo = 32768.0 / 500.0,
    .Sensibilidad_Acel = 32768.0 /16.0
};
#endif

#ifdef GYRO_L3G4200
tpGiroscopo_L3G4200 Giroscopo_L3G4200 = {
        .Direccion = Dir_1_L3G4200,
        .Ganancia = dps_2000,
        .Sensibilidad_Giroscopo = 32768.0 / 2000.0,
        .ODR = ODR_800_Hz,
        .BDU = BDU_Continuo,
        .Modo = Bypass,
        .BLE = BLE_Big_Endian,
        .BW_LPF = LPF1_1,
        .BW_HPF = HPF_0,
        .HPF_activar = HPF_No_Filtro,
        .HPF_modo = HPF_Normal,
        .Modo_Filtro = Filtrado_LPF2
};
#endif

#ifdef COMPASS_HMC5883L
tpHMC5883L Brujula_HMC5883L = {
    .Angulo_Rotacion = 0,
    .Ganancia = Gauss_1_3,
    .Modo_Operacion = Continuo,
    .Modo_Medida = Normal,
    .Muestras_Media = MEDIA_4,
    .ODR = ODR_75_Hz,
    .Velocidad_I2C = I2C_400_Khz,
    .Sensibilidad = S_1
};
#endif

#ifdef BAR_BMP280
tpBarometro_BMP280 Barometro_BMP280 = {
    .Direccion = BMP_280_DIR_0,
    .Modo = Mode_Normal,
    .Oversampling_Presion = x16,
    .Oversampling_Temperatura = x16,
    .t_sampling = ms05
};
#endif
```

```
//..Variables del sistema...//
float32_t Posicion_inicial = 0;
uint16_t Altura_US_mm = 0;
uint16_t Altura_Presion_mm = 0;
float32_t Gravedad;

//......Identificacion.........//
uint32_t nDatos_Identifiacion = 0;
uint32_t PuntoTrabajo_motor = 0;
uint16_t nDatos_leidos = 0;
//uint16_t Ticks_por_RPS = 0;

//................Instancias.....................................................//
//.......PWM.............//
PWM_Handle PWM0;
PWM_Handle PWM1;
PWM_Handle PWM2;
PWM_Handle PWM3;

PWM_Params PARAMS_PWM0;
PWM_Params PARAMS_PWM1;
PWM_Params PARAMS_PWM2;
PWM_Params PARAMS_PWM3;

//.......I2C.............//
I2C_Handle I2C_PRINCIPAL;
I2C_Handle I2C_AUX;

I2C_Params PARAMS_I2C;

//I2C_Params PARAMS_I2C_AUX;

//.......UART.............//
UART_Handle UART_USB;
//UART_Handle UART_BT_MANDO;
UART_Handle UART_BT_TELEMETRIA;
UART_Handle UART_AUX;

UART_Params PARAMS_UART_USB;
UART_Params PARAMS_UART_BT_TELEMETRIA;
UART_Params PARAMS_UART_AUX;

//......SPI...nRF24L01......//
//tp_nRF24L01 nRF24L01;
//SPI_Params PARAMS_SPI_0;

//.....Timer.............//
Timer_Handle US_Timer;
Timer_Params PARAMS_US_Timer;

//....WATCHDOG...........//
Watchdog_Handle WatchDog_0;
Watchdog_Params PARAMS_WatchDog_0;

//...BUZON............//
Mailbox_Handle Buzon_Lecturas_IMU;
Mailbox_Params PARAMS_Buzon;

/*
Mailbox_Handle Buzon_Calibracion_IMU;
Mailbox_Handle Buzon_Calibracion_Brujula;
Mailbox_Params PARAMS_Buzon_Calibracion;
*/
```

```
//.....................TASK.......SEMAPHORES.............CLOCK.......................//

Task_Params Parametros_Tarea;
Semaphore_Params Parametros_Semaforo;
Clock_Params Parametos_Clock;

//....CONTROL....//
Task_Handle TASK_Control;
Semaphore_Handle SEMAPHORE_Control;
Clock_Handle CLOCK_Control;

//....ALTURA_US..//
Task_Handle TASK_Calculo_Altura;
Semaphore_Handle SEMAPHORE_Calculo_Altura;
Clock_Handle CLOCK_Calculo_Altura;

//....Leer_IMU...//
Task_Handle TASK_Leer_IMU;
Semaphore_Handle SEMAPHORE_Leer_IMU;
Clock_Handle CLOCK_Leer_IMU;

//...Calculo_AHRS....//
Task_Handle TASK_Calculo_AHRS;
Semaphore_Handle SEMAPHORE_Calculo_AHRS;
Clock_Handle CLOCK_Calculo_AHRS;

//....Identificacion......//
Task_Handle TASK_Identificacion;
Semaphore_Handle SEMAPHORE_Identificacion;
Clock_Handle CLOCK_Identificacion;

//....Coordinador......//
Task_Handle TASK_Coordinador;
Semaphore_Handle SEMAPHORE_Coordinador;
Clock_Handle CLOCK_Coordinador;

//............ERROR.........................//
Error_Block eb;

//...................FUNCIONES.....................................................//
void Rotacion_X(arm_matrix_instance_f32 *Matriz, float32_t Giro, bool Radianes);
void Rotacion_Y(arm_matrix_instance_f32 *Matriz, float32_t Giro, bool Radianes);
void Rotacion_Z(arm_matrix_instance_f32 *Matriz, float32_t Giro, bool Radianes);
void Rotacion_ZYZp(arm_matrix_instance_f32 *Matriz, float32_t Giro[3], bool Radianes);
void Rotacion_ZXY(arm_matrix_instance_f32 *Matriz, float32_t Giro[3], bool Radianes);
//void Rotacion_XYZ(arm_matrix_instance_f32 *Matriz, float32_t Giro[3], bool Radianes);

//.........WATCHDOG.................................//
void FuncionWatchDog();
void Reestablecer_Conexion();
//...Control.....//
void Control(UArg arg0, UArg arg1);
void CLK_Control();
//...US.........//
void ISR_Timer_US();
void ISR_GPIO_US();
void Calculo_Altura(UArg arg0, UArg arg1);
void CLK_Calculo_Altura();
//....Leer_IMU...//
void Lectura_Datos_IMU(UArg arg0, UArg arg1);
void CLK_Lectura_Datos_IMU();
//....Calculo_AHRS...//
void Calculo_AHRS(UArg arg0, UArg arg1);
void CLK_Calculo_AHRS();
//....Identificacion......//
void Identificacion(UArg arg0, UArg arg1);
void CLK_Identificacion();
#ifdef Sensor_RPM
```

```
    void ISR_GPIO_RPM();
#endif

//....Coordinador......//
void Coordinador(UArg arg0, UArg arg1);
void CLK_Coordinador();


/*
 *  ======== main ========
 */
int main(void){
    Error_init(&eb);
//.........................INICIALIZACION...DRIVERS.....................................
.........//
    /* Call board init functions */
    QUAD_BOARD_initGeneral();
    QUAD_BOARD_initGPIO();
    QUAD_BOARD_initPWM();
    QUAD_BOARD_initI2C();
    QUAD_BOARD_initSPI();
    QUAD_BOARD_initUART();
    QUAD_BOARD_initWatchdog();

//......SERVIDORES.........//
    Iniciar_Servidores();

    GPIO_write(QUAD_BOARD_LED_RED, 1);
    GPIO_write(QUAD_BOARD_LED_GREEN, 0);
    GPIO_write(QUAD_BOARD_LED_BLUE, 0);

//......PWM........//
    PWM_Params_init(&PARAMS_PWM0);
    PARAMS_PWM0.period = 5000;
    PARAMS_PWM0.dutyMode = PWM_DUTY_TIME;
    PWM_Params_init(&PARAMS_PWM1);
    PARAMS_PWM1.period = 5000;
    PARAMS_PWM1.dutyMode = PWM_DUTY_TIME;
    PWM_Params_init(&PARAMS_PWM2);
    PARAMS_PWM2.period = 5000;
    PARAMS_PWM2.dutyMode = PWM_DUTY_TIME;
    PWM_Params_init(&PARAMS_PWM3);
    PARAMS_PWM3.period = 5000;
    PARAMS_PWM3.dutyMode = PWM_DUTY_TIME;

    PWM0 = PWM_open(QUAD_BOARD_PWM0, &PARAMS_PWM0);
    PWM1 = PWM_open(QUAD_BOARD_PWM1, &PARAMS_PWM1);
    PWM2 = PWM_open(QUAD_BOARD_PWM2, &PARAMS_PWM2);
    PWM3 = PWM_open(QUAD_BOARD_PWM3, &PARAMS_PWM3);
/*
    PWM_setDuty(PWM0, Pulso_maximo_PWM_motor);
    PWM_setDuty(PWM1, Pulso_maximo_PWM_motor);
    PWM_setDuty(PWM2, Pulso_maximo_PWM_motor);
    PWM_setDuty(PWM3, Pulso_maximo_PWM_motor);
*/
    PWM_setDuty(PWM0, 0);
    PWM_setDuty(PWM1, 0);
    PWM_setDuty(PWM2, 0);
    PWM_setDuty(PWM3, 0);
//.......I2C.............//
    I2C_Params_init(&PARAMS_I2C);
    PARAMS_I2C.bitRate = I2C_400kHz;
    PARAMS_I2C.transferMode = I2C_MODE_BLOCKING;
    PARAMS_I2C.transferCallbackFxn = NULL;

    I2C_PRINCIPAL = I2C_open(QUAD_BOARD_I2C0, &PARAMS_I2C);

//    I2C_Params_init(&PARAMS_I2C_AUX);
```

```
//    PARAMS_I2C.bitRate = I2C_400kHz;
//    PARAMS_I2C.transferMode = I2C_MODE_BLOCKING;
//    PARAMS_I2C.transferCallbackFxn = NULL;

    I2C_AUX = I2C_open(QUAD_BOARD_I2C2, &PARAMS_I2C);

//........UART.........//
    UART_Params_init(&PARAMS_UART_BT_TELEMETRIA);
    PARAMS_UART_BT_TELEMETRIA.baudRate = BAUD_RATE_460800;
    PARAMS_UART_BT_TELEMETRIA.dataLength = UART_LEN_8;
    PARAMS_UART_BT_TELEMETRIA.parityType = UART_PAR_NONE;
    PARAMS_UART_BT_TELEMETRIA.stopBits = UART_STOP_ONE;
    PARAMS_UART_BT_TELEMETRIA.readEcho = UART_ECHO_OFF;

    PARAMS_UART_BT_TELEMETRIA.readReturnMode = UART_RETURN_FULL;
    PARAMS_UART_BT_TELEMETRIA.writeDataMode = UART_DATA_BINARY;
    PARAMS_UART_BT_TELEMETRIA.readDataMode = UART_DATA_BINARY;
    PARAMS_UART_BT_TELEMETRIA.readMode = UART_MODE_BLOCKING;
    PARAMS_UART_BT_TELEMETRIA.writeMode = UART_MODE_BLOCKING;
    PARAMS_UART_BT_TELEMETRIA.writeTimeout = BIOS_WAIT_FOREVER;
    //PARAMS_UART_BT_TELEMETRIA.readTimeout = BIOS_NO_WAIT;
    PARAMS_UART_BT_TELEMETRIA.readTimeout = 180;

    UART_BT_TELEMETRIA = UART_open(QUAD_BOARD_UART5_BT_TELEMETRIA,
    &PARAMS_UART_BT_TELEMETRIA);

//.......SPI..........//

/*
    SPI_Params_init(&PARAMS_SPI_0);

    PARAMS_SPI_0.transferMode = SPI_MODE_BLOCKING;
    PARAMS_SPI_0.transferTimeout = BIOS_WAIT_FOREVER;
    PARAMS_SPI_0.mode = SPI_MASTER;
    PARAMS_SPI_0.bitRate = BITRATE_SPI;
    PARAMS_SPI_0.dataSize = 8;
    PARAMS_SPI_0.frameFormat = SPI_POL0_PHA0;

    nRF24L01.SPI = SPI_open(QUAD_BOARD_SPI0, &PARAMS_SPI_0);
    nRF24L01.PIN_CE = QUAD_BOARD_SPI_CE;
    nRF24L01.PIN_CSN = QUAD_BOARD_SPI_CSN;
    nRF24L01.PIN_IRQ = NULL;
*/

//.....WATCHDOG.....................//
    Watchdog_Params_init(&PARAMS_WatchDog_0);
    PARAMS_WatchDog_0.callbackFxn = FuncionWatchDog;
    PARAMS_WatchDog_0.debugStallMode = Watchdog_DEBUG_STALL_ON;
    PARAMS_WatchDog_0.resetMode = Watchdog_RESET_OFF;

//.....BUZON.............//
/*
    Mailbox_Params_init(&PARAMS_Buzon_Calibracion);
    Buzon_Calibracion_IMU = Mailbox_create(sizeof(tpLecturas_IMU), 1,
    &PARAMS_Buzon_Calibracion, &eb);
    Buzon_Calibracion_Brujula = Mailbox_create(sizeof(tpLecturas_Brujula), 1,
    &PARAMS_Buzon_Calibracion, &eb);
*/
    Mailbox_Params_init(&PARAMS_Buzon);
#ifdef IMU_MPU9250
    Buzon_Lecturas_IMU = Mailbox_create(sizeof(tpLecturas_IMU), 1, &PARAMS_Buzon, &eb);
#endif
#ifdef IMU_MPU9250
    Buzon_Lecturas_IMU = Mailbox_create(sizeof(tpLecturas_9DOF_IMU), 1, &PARAMS_Buzon, &eb);
#endif
//.........TIMER.............//
    Timer_Params_init(&PARAMS_US_Timer);
    PARAMS_US_Timer.period = Pulso_arranque_us;  //Para una distancia MAX de 4 metros (aprox)
```

```
    PARAMS_US_Timer.periodType = Timer_PeriodType_MICROSECS;
    PARAMS_US_Timer.runMode = Timer_RunMode_ONESHOT;
    PARAMS_US_Timer.startMode = Timer_StartMode_USER;

    US_Timer = Timer_create(5, ISR_Timer_US, &PARAMS_US_Timer, &eb );

//.................INTERRUPCIONES.....................................//
    GPIO_setCallback(QUAD_BOARD_ECHO, ISR_GPIO_US);
    GPIO_enableInt(QUAD_BOARD_ECHO);
#ifdef Sensor_RPM
    GPIO_setCallback(QUAD_BOARD_RPM, ISR_GPIO_RPM);
    GPIO_enableInt(QUAD_BOARD_RPM);
#endif

//.........................INICIALIZACION...TAREAS.....................................
.........//
//....Control....//
    Task_Params_init(&Parametros_Tarea);
    Parametros_Tarea.priority = PRIORIDAD_Control;
    Parametros_Tarea.stackSize = 2304;
    TASK_Control = Task_create(Control, &Parametros_Tarea, &eb);

    Semaphore_Params_init(&Parametros_Semaforo);
    Parametros_Semaforo.mode = Semaphore_Mode_BINARY;
    SEMAPHORE_Control = Semaphore_create(nTokensIniciales_0, &Parametros_Semaforo, &eb);

    Clock_Params_init(&Parametos_Clock);
    Parametos_Clock.period = PERIODO_Control;
    Parametos_Clock.startFlag = false;
    CLOCK_Control = Clock_create(CLK_Control, Timeout_Clk_Control, &Parametos_Clock, &eb);


//....Altura..//
    Task_Params_init(&Parametros_Tarea);
    Parametros_Tarea.priority = PRIORIDAD_Calculo_Altura;
    Parametros_Tarea.stackSize = 768;
    TASK_Calculo_Altura = Task_create(Calculo_Altura, &Parametros_Tarea, &eb);

    Semaphore_Params_init(&Parametros_Semaforo);
    Parametros_Semaforo.mode = Semaphore_Mode_BINARY;
    SEMAPHORE_Calculo_Altura = Semaphore_create(nTokensIniciales_0, &Parametros_Semaforo,
    &eb);

    Clock_Params_init(&Parametos_Clock);
    Parametos_Clock.period = PERIODO_Calculo_Altura;
    Parametos_Clock.startFlag = false;
    CLOCK_Calculo_Altura = Clock_create(CLK_Calculo_Altura, Timeout_Clk_Calculo_Altura,
    &Parametos_Clock, &eb);

//....Leer_IMU...//
    Task_Params_init(&Parametros_Tarea);
    Parametros_Tarea.priority = PRIORIDAD_Leer_IMU;
    Parametros_Tarea.stackSize = 1256;
    TASK_Leer_IMU = Task_create(Lectura_Datos_IMU, &Parametros_Tarea, &eb);

    Semaphore_Params_init(&Parametros_Semaforo);
    Parametros_Semaforo.mode = Semaphore_Mode_BINARY;
    SEMAPHORE_Leer_IMU = Semaphore_create(nTokensIniciales_0, &Parametros_Semaforo, &eb);

    Clock_Params_init(&Parametos_Clock);
    Parametos_Clock.period = PERIODO_Leer_IMU;
    Parametos_Clock.startFlag = true;
    CLOCK_Leer_IMU = Clock_create(CLK_Lectura_Datos_IMU, Timeout_Clk_Leer_IMU,
    &Parametos_Clock, &eb);

//...Calculo AHRS....//
    Task_Params_init(&Parametros_Tarea);
    Parametros_Tarea.priority = PRIORIDAD_Calculo_AHRS;
```

```
    TASK_Calculo_AHRS = Task_create(Calculo_AHRS, &Parametros_Tarea, &eb);

    Semaphore_Params_init(&Parametros_Semaforo);
    Parametros_Semaforo.mode = Semaphore_Mode_BINARY;
    SEMAPHORE_Calculo_AHRS = Semaphore_create(nTokensIniciales_0, &Parametros_Semaforo, &eb);

    Clock_Params_init(&Parametos_Clock);
    Parametos_Clock.period = PERIODO_Calculo_AHRS;
    Parametos_Clock.startFlag = false;
    CLOCK_Calculo_AHRS = Clock_create(CLK_Calculo_AHRS, Timeout_Clk_Calculo_AHRS,
    &Parametos_Clock, &eb);

//....Identificacion......//
    Task_Params_init(&Parametros_Tarea);
    Parametros_Tarea.priority = PRIORIDAD_Identificacion;
    Parametros_Tarea.stackSize = 512;
    TASK_Identificacion = Task_create(Identificacion, &Parametros_Tarea, &eb);

    Semaphore_Params_init(&Parametros_Semaforo);
    Parametros_Semaforo.mode = Semaphore_Mode_BINARY;
    SEMAPHORE_Identificacion = Semaphore_create(nTokensIniciales_0, &Parametros_Semaforo,
    &eb);

    Clock_Params_init(&Parametos_Clock);
    Parametos_Clock.period = PERIODO_Identificacion;
    Parametos_Clock.startFlag = false;
    CLOCK_Identificacion = Clock_create(CLK_Identificacion, Timeout_Clk_Identificacion,
    &Parametos_Clock, &eb);

//......Coordinador.......//

    Task_Params_init(&Parametros_Tarea);
    Parametros_Tarea.priority = PRIORIDAD_Coordinador;
    TASK_Coordinador = Task_create(Coordinador, &Parametros_Tarea, &eb);

    Semaphore_Params_init(&Parametros_Semaforo);
    Parametros_Semaforo.mode = Semaphore_Mode_BINARY;
    SEMAPHORE_Coordinador = Semaphore_create(nTokensIniciales_0, &Parametros_Semaforo, &eb);

    Clock_Params_init(&Parametos_Clock);
    Parametos_Clock.period = PERIODO_Coordinador;
    Parametos_Clock.startFlag = false;
    CLOCK_Coordinador = Clock_create(CLK_Coordinador, Timeout_Clk_Coordinador,
    &Parametos_Clock, &eb);

/*
    PWM_setDuty(PWM0, Pulso_minimo_PWM_motor + Accion_Maxima);
    PWM_setDuty(PWM1, Pulso_minimo_PWM_motor + Accion_Maxima);
    PWM_setDuty(PWM2, Pulso_minimo_PWM_motor + Accion_Maxima);
    PWM_setDuty(PWM3, Pulso_minimo_PWM_motor + Accion_Maxima);
    while(GPIO_read(QUAD_BOARD_SW2));

    PWM_setDuty(PWM0, Pulso_minimo_PWM_motor + Accion_Minima);
    PWM_setDuty(PWM1, Pulso_minimo_PWM_motor + Accion_Minima);
    PWM_setDuty(PWM2, Pulso_minimo_PWM_motor + Accion_Minima);
    PWM_setDuty(PWM3, Pulso_minimo_PWM_motor + Accion_Minima);
    while(!GPIO_read(QUAD_BOARD_SW2));
*/

/*
        PWM_setDuty(PWM0, Accion_Maxima + Pulso_minimo_PWM_motor);
        PWM_setDuty(PWM1, Accion_Maxima + Pulso_minimo_PWM_motor);
        PWM_setDuty(PWM2, Accion_Maxima + Pulso_minimo_PWM_motor);
        PWM_setDuty(PWM3, Accion_Maxima + Pulso_minimo_PWM_motor);
    while(GPIO_read(QUAD_BOARD_SW2));
        PWM_setDuty(PWM0, Pulso_minimo_PWM_motor);
        PWM_setDuty(PWM1, Pulso_minimo_PWM_motor);
        PWM_setDuty(PWM2, Pulso_minimo_PWM_motor);
```

```
        PWM_setDuty(PWM3, Pulso_minimo_PWM_motor);
    while(!GPIO_read(QUAD_BOARD_SW2));
*/

    //Iniciamos la matriz de correccion
    Rotacion_ZXY(&Calibracion_IMU.Correccion_Alineamiento, Calibracion_IMU.Giro, false);

    while(GPIO_read(QUAD_BOARD_SW2));

   /* Start BIOS */
    BIOS_start();

    return (0);
}
////////////////////////////////////////////////////////////////////////////////////////////
/////////////
void FuncionWatchDog(){
    float32_t Referencia[4] = {0, 0, 0, 0};
    UInt Key, Key2, Key3;

    Key = Task_disable();
    Key2 = Hwi_disable();
    Key3 = Swi_disable();

    Watchdog_clear(WatchDog_0);

    switch(Estado_Sistema){
        case VUELO:
        case ESPERA:
        default:
            Estado_Sistema_Anterior = Estado_Sistema;
            Modo_Control_Anterior = Modo_Control;

            Estado_Sistema = ERROR_CONEXION;

            GPIO_write(QUAD_BOARD_LED_RED, 0);
            GPIO_write(QUAD_BOARD_LED_GREEN, 0);
            GPIO_write(QUAD_BOARD_LED_BLUE, 0);
        break;
    }

    switch(ModoWatchdog){
        case PARADA_EMER:
            GPIO_write(QUAD_BOARD_LED_RED, 1);
            GPIO_write(QUAD_BOARD_LED_GREEN, 0);
            GPIO_write(QUAD_BOARD_LED_BLUE, 0);

            //Stop a todos los clocks y tareas
            Clock_stop(CLOCK_Control);
            Semaphore_reset(SEMAPHORE_Control, 0);
            Clock_stop(CLOCK_Identificacion);
            Semaphore_reset(SEMAPHORE_Identificacion, 0);

            PWM_setDuty(PWM0, Pulso_minimo_PWM_motor);
            PWM_setDuty(PWM1, Pulso_minimo_PWM_motor);
            PWM_setDuty(PWM2, Pulso_minimo_PWM_motor);
            PWM_setDuty(PWM3, Pulso_minimo_PWM_motor);

            while(1);
            //System_exit(0);
        //break;
        case ESTABILIZACION_EMER:
            GPIO_toggle(QUAD_BOARD_LED_RED);
            GPIO_toggle(QUAD_BOARD_LED_GREEN);
            GPIO_toggle(QUAD_BOARD_LED_BLUE);

            Modo_Control = ANGULOS_4;
            memcpy(Direccion_servidor_Referencia(), Referencia, sizeof(Referencia));
```

```
        break;
    }

    Swi_restore(Key3);
    Hwi_restore(Key2);
    Task_restore(Key);
}

void Reestablecer_Conexion(){

    Modo_Control = Modo_Control_Anterior;
    Estado_Sistema = Estado_Sistema_Anterior;

    switch(Estado_Sistema){
        case VUELO:
            GPIO_write(QUAD_BOARD_LED_RED, 0);
            GPIO_write(QUAD_BOARD_LED_GREEN, 1);
            GPIO_write(QUAD_BOARD_LED_BLUE, 0);
            break;
        case ESPERA:
            GPIO_write(QUAD_BOARD_LED_RED, 1);
            GPIO_write(QUAD_BOARD_LED_GREEN, 1);
            GPIO_write(QUAD_BOARD_LED_BLUE, 0);
            break;
    }
}
void Rotacion_X(arm_matrix_instance_f32 *Matriz, float32_t Giro, bool Radianes){
    float32_t Seno = 0;
    float32_t Cos = 0;

    float32_t Aux_Matriz[9] = {0,0,0,0,0,0,0,0,0};
    arm_matrix_instance_f32 Aux = {Matriz->numRows, Matriz->numCols, Aux_Matriz};

    float32_t Matriz_Rotacion_Matriz[9] = {0,0,0,0,0,0,0,0,0};
    arm_matrix_instance_f32 Matriz_Rotacion = {3, 3, Matriz_Rotacion_Matriz};

    if(Radianes){ Giro = Giro * 180.0/PI; }
    arm_sin_cos_f32(Giro, &Seno, &Cos);
    Matriz_Rotacion.pData[0] = 1;
    Matriz_Rotacion.pData[4] = Cos;
    Matriz_Rotacion.pData[5] = -Seno;
    Matriz_Rotacion.pData[7] = Seno;
    Matriz_Rotacion.pData[8] = Cos;

    arm_copy_f32(Matriz->pData, Aux.pData, 3*Matriz->numCols);
    Aux.numCols = Matriz->numCols;
    Aux.numRows = Matriz->numRows;

    arm_mat_mult_f32(&Aux, &Matriz_Rotacion, Matriz);

}
void Rotacion_Y(arm_matrix_instance_f32 *Matriz, float32_t Giro, bool Radianes){
    float32_t Seno = 0;
    float32_t Cos = 0;

    float32_t Aux_Matriz[9] = {0,0,0,0,0,0,0,0,0};
    arm_matrix_instance_f32 Aux = {Matriz->numRows, Matriz->numCols, Aux_Matriz};

    float32_t Matriz_Rotacion_Matriz[9] = {0,0,0,0,0,0,0,0,0};
    arm_matrix_instance_f32 Matriz_Rotacion = {3, 3, Matriz_Rotacion_Matriz};

    if(Radianes){ Giro = Giro * 180.0/PI; }
    arm_sin_cos_f32(Giro, &Seno, &Cos);
    Matriz_Rotacion.pData[0] = Cos;
    Matriz_Rotacion.pData[2] = Seno;
    Matriz_Rotacion.pData[4] = 1;
    Matriz_Rotacion.pData[6] = -Seno;
    Matriz_Rotacion.pData[8] = Cos;
```

```
    arm_copy_f32(Matriz->pData, Aux.pData, 3*Matriz->numCols);
    Aux.numCols = Matriz->numCols;
    Aux.numRows = Matriz->numRows;

    arm_mat_mult_f32(&Aux, &Matriz_Rotacion, Matriz);
}
void Rotacion_Z(arm_matrix_instance_f32 *Matriz, float32_t Giro, bool Radianes){
    float32_t Seno = 0;
    float32_t Cos = 0;

    float32_t Aux_Matriz[9] = {0,0,0,0,0,0,0,0,0};
    arm_matrix_instance_f32 Aux = {Matriz->numRows, Matriz->numCols, Aux_Matriz};

    float32_t Matriz_Rotacion_Matriz[9] = {0,0,0,0,0,0,0,0,0};
    arm_matrix_instance_f32 Matriz_Rotacion = {3, 3, Matriz_Rotacion_Matriz};

    if(Radianes){ Giro = Giro * 180.0/PI; }
    arm_sin_cos_f32(Giro, &Seno, &Cos);
    Matriz_Rotacion.pData[0] = Cos;
    Matriz_Rotacion.pData[1] = -Seno;
    Matriz_Rotacion.pData[3] = Seno;
    Matriz_Rotacion.pData[4] = Cos;
    Matriz_Rotacion.pData[8] = 1;

    arm_copy_f32(Matriz->pData, Aux.pData, 3*Matriz->numCols);
    Aux.numCols = Matriz->numCols;
    Aux.numRows = Matriz->numRows;

    arm_mat_mult_f32(&Aux, &Matriz_Rotacion, Matriz);
}
/*
void Rotacion_ZYZp(arm_matrix_instance_f32 *Matriz, float32_t Giro[3], bool Radianes){
    float32_t Aux_Matriz[9] = {1,0,0,0,1,0,0,0,1};
    arm_matrix_instance_f32 Aux = {Matriz->numRows, Matriz->numCols, Aux_Matriz};
    float32_t Aux_Matriz2[9] = {1,0,0,0,1,0,0,0,1};
    arm_matrix_instance_f32 Aux2 = {Matriz->numRows, Matriz->numCols, Aux_Matriz2};
    float32_t Aux_Matriz3[9] = {1,0,0,0,1,0,0,0,1};
    arm_matrix_instance_f32 Aux3 = {Matriz->numRows, Matriz->numCols, Aux_Matriz3};

    Rotacion_Z(&Aux, Giro[0], Radianes);
    Rotacion_Y(&Aux2, Giro[1], Radianes);
    arm_mat_mult_f32(&Aux, &Aux2, &Aux3);
    Rotacion_Z(&Aux, Giro[2], Radianes);
    arm_mat_mult_f32(&Aux3, &Aux, Matriz);
}
*/
void Rotacion_ZXY(arm_matrix_instance_f32 *Matriz, float32_t Giro[3], bool Radianes){

    Matriz->pData[0] = 1;
    Matriz->pData[1] = 0;
    Matriz->pData[2] = 0;
    Matriz->pData[3] = 0;
    Matriz->pData[4] = 1;
    Matriz->pData[5] = 0;
    Matriz->pData[6] = 0;
    Matriz->pData[7] = 0;
    Matriz->pData[8] = 1;

    Rotacion_Z(Matriz, Giro[0], Radianes);
    Rotacion_X(Matriz, Giro[1], Radianes);
    Rotacion_Y(Matriz, Giro[2], Radianes);
}

void Rotacion_ZYZp(arm_matrix_instance_f32 *Matriz, float32_t Giro[3], bool Radianes){

    Matriz->pData[0] = 1;
    Matriz->pData[1] = 0;
```

```
    Matriz->pData[2] = 0;
    Matriz->pData[3] = 0;
    Matriz->pData[4] = 1;
    Matriz->pData[5] = 0;
    Matriz->pData[6] = 0;
    Matriz->pData[7] = 0;
    Matriz->pData[8] = 1;

    Rotacion_Z(Matriz, Giro[0], Radianes);
    Rotacion_Y(Matriz, Giro[1], Radianes);
    Rotacion_Z(Matriz, Giro[2], Radianes);
}
```

```
//....Leer_IMU...//
void Lectura_Datos_IMU(UArg arg0, UArg arg1){

#ifdef IMU_MPU6050
    tpLecturas_IMU Lecturas_IMU = {0, 0, 0, 0, 0, 0, 0};
#endif
#ifdef IMU_MPU9250
    tpLecturas_9DOF_IMU Lecturas_9DOF_IMU = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
#endif
#ifdef GYRO_L3G4200
    tpLecturas_Giroscopo Lecturas_Giroscopo = {0, 0, 0, 0};
#endif
#ifdef COMPASS_HMC5883L
    tpLecturas_Brujula Lecturas_Brujula;
#endif

    float32_t Lecturas_matriz[3];
    float32_t Lecturas_Corregidas_matriz[3];

    arm_matrix_instance_f32 Lecturas = {3, 1, Lecturas_matriz};
    arm_matrix_instance_f32 Lecturas_Corregidas = {3, 1, Lecturas_Corregidas_matriz};

    float32_t aux;
    uint16_t nMuestras = 0;

    tpTelemetria_IMU Telemetria_IMU = {
            .Inicio = START_FRAME,
            .Final = FINAL_FRAME
    };

#ifdef Filtrado_Vel_IMU
    float32_t Estado_filtro_Vel_X[4*num_etapas_Filtro_Vel];
    float32_t Estado_filtro_Vel_Y[4*num_etapas_Filtro_Vel];
    float32_t Estado_filtro_Vel_Z[4*num_etapas_Filtro_Vel];
#endif

    float32_t Estado_filtro_Acel_X[4*num_etapas_Filtro_Acel];
    float32_t Estado_filtro_Acel_Y[4*num_etapas_Filtro_Acel];
    float32_t Estado_filtro_Acel_Z[4*num_etapas_Filtro_Acel];

#ifdef Filtrado_Vel_IMU
    arm_biquad_casd_df1_inst_f32 Filtro_Vel_X = {num_etapas_Filtro_Vel, Estado_filtro_Vel_X,
    (float32_t *)Coeficientes_Filtro_Vel_Valores};
```

```
    arm_biquad_casd_df1_inst_f32 Filtro_Vel_Y = {num_etapas_Filtro_Vel, Estado_filtro_Vel_Y,
    (float32_t *)Coeficientes_Filtro_Vel_Valores};
    arm_biquad_casd_df1_inst_f32 Filtro_Vel_Z = {num_etapas_Filtro_Vel, Estado_filtro_Vel_Z,
    (float32_t *)Coeficientes_Filtro_Vel_Valores};
#endif

    arm_biquad_casd_df1_inst_f32 Filtro_Acel_X = {num_etapas_Filtro_Acel,
    Estado_filtro_Acel_X, (float32_t *)Coeficientes_Filtro_Acel_Valores};
    arm_biquad_casd_df1_inst_f32 Filtro_Acel_Y = {num_etapas_Filtro_Acel,
    Estado_filtro_Acel_Y, (float32_t *)Coeficientes_Filtro_Acel_Valores};
    arm_biquad_casd_df1_inst_f32 Filtro_Acel_Z = {num_etapas_Filtro_Acel,
    Estado_filtro_Acel_Z, (float32_t *)Coeficientes_Filtro_Acel_Valores};


    Semaphore_pend(SEMAPHORE_Leer_IMU, BIOS_WAIT_FOREVER);

//Inicializamos el filtro
#ifdef Filtrado_Vel_IMU
    arm_fill_f32(0.0, Estado_filtro_Vel_X, 4*num_etapas_Filtro_Vel);
    arm_fill_f32(0.0, Estado_filtro_Vel_Y, 4*num_etapas_Filtro_Vel);
    arm_fill_f32(0.0, Estado_filtro_Vel_Z, 4*num_etapas_Filtro_Vel);
#endif

    arm_fill_f32(0.0, Estado_filtro_Acel_X, 4*num_etapas_Filtro_Acel);
    arm_fill_f32(0.0, Estado_filtro_Acel_Y, 4*num_etapas_Filtro_Acel);
    arm_fill_f32(0.0, Estado_filtro_Acel_Z, 4*num_etapas_Filtro_Acel);

#ifdef IMU_MPU6050
    Iniciar_IMU_MPU6050(I2C_PRINCIPAL, IMU6050);
#endif

#ifdef IMU_MPU9250
    Iniciar_IMU_MPU9250(I2C_PRINCIPAL, IMU9250);
#endif

#ifdef GYRO_L3G4200
    Iniciar_Giroscopo_L3G4200(I2C_PRINCIPAL, Giroscopo_L3G4200);
#endif

#ifdef COMPASS_HMC5883L
    Iniciar_Brujula_HMC5883L(I2C_PRINCIPAL, Brujula_HMC5883L);
#endif

//........Calibracion....media........//
    GPIO_write(QUAD_BOARD_LED_RED, 0);
    GPIO_write(QUAD_BOARD_LED_GREEN, 1);
    GPIO_write(QUAD_BOARD_LED_BLUE, 1);

    Datos = Memory_alloc(NULL, Numero_Muetras_calibracion_IMU*4, 0, &eb);

#ifdef IMU_MPU6050

    if(ModoCalibracionIMU == CALIBRACION_ACELEROMETRO || ModoCalibracionIMU ==
    CALIBRACION_COMPLETA_IMU){
        for(nMuestras=0; nMuestras<Numero_Muetras_calibracion_IMU; nMuestras++){
            Semaphore_pend(SEMAPHORE_Leer_IMU, BIOS_WAIT_FOREVER);
            Leer_IMU_MPU6050(I2C_PRINCIPAL, IMU6050, &Lecturas_IMU);
            *((float32_t*)Datos + nMuestras) = Lecturas_IMU.Valor.x_acel;
            Task_sleep(PERIODO_Leer_IMU);
        }
        arm_mean_f32((float32_t*)Datos, Numero_Muetras_calibracion_IMU,
        &Calibracion_IMU.Des_est_Acel_x);
        Calibracion_IMU.Media_Acel_x = (int16_t)Calibracion_IMU.Des_est_Acel_x;
        arm_std_f32((float32_t*)Datos, Numero_Muetras_calibracion_IMU,
        &Calibracion_IMU.Des_est_Acel_x);

        for(nMuestras=0; nMuestras<Numero_Muetras_calibracion_IMU; nMuestras++){
            Semaphore_pend(SEMAPHORE_Leer_IMU, BIOS_WAIT_FOREVER);
```

```
            Leer_IMU_MPU6050(I2C_PRINCIPAL, IMU6050, &Lecturas_IMU);
            *((float32_t*)Datos + nMuestras) = Lecturas_IMU.Valor.y_acel;
            Task_sleep(PERIODO_Leer_IMU);
        }
        arm_mean_f32((float32_t*)Datos, Numero_Muetras_calibracion_IMU,
        &Calibracion_IMU.Des_est_Acel_y);
        Calibracion_IMU.Media_Acel_y = (int16_t)Calibracion_IMU.Des_est_Acel_y;
        arm_std_f32((float32_t*)Datos, Numero_Muetras_calibracion_IMU,
        &Calibracion_IMU.Des_est_Acel_y);

        for(nMuestras=0; nMuestras<Numero_Muetras_calibracion_IMU; nMuestras++){
            Semaphore_pend(SEMAPHORE_Leer_IMU, BIOS_WAIT_FOREVER);
            Leer_IMU_MPU6050(I2C_PRINCIPAL, IMU6050, &Lecturas_IMU);
            *((float32_t*)Datos + nMuestras) = Lecturas_IMU.Valor.z_acel;
            Task_sleep(PERIODO_Leer_IMU);
        }
        arm_mean_f32((float32_t*)Datos, Numero_Muetras_calibracion_IMU,
        &Calibracion_IMU.Des_est_Acel_z);
        Calibracion_IMU.Media_Acel_z = (int16_t)Calibracion_IMU.Des_est_Acel_z;
        Gravedad  = sqrt(pow(Calibracion_IMU.Media_Acel_x,2) +
        pow(Calibracion_IMU.Media_Acel_y,2) + pow(Calibracion_IMU.Media_Acel_z,2));
        Calibracion_IMU.Media_Acel_z -= (int16_t)sqrt(pow(Calibracion_IMU.Media_Acel_x,2) +
        pow(Calibracion_IMU.Media_Acel_y,2) + pow(Calibracion_IMU.Media_Acel_z,2));
        arm_std_f32((float32_t*)Datos, Numero_Muetras_calibracion_IMU,
        &Calibracion_IMU.Des_est_Acel_z);
    }

    if(ModoCalibracionIMU == CALIBRACION_GIROSCOPO || ModoCalibracionIMU ==
    CALIBRACION_COMPLETA_IMU){
        for(nMuestras=0; nMuestras<Numero_Muetras_calibracion_IMU; nMuestras++){
            Semaphore_pend(SEMAPHORE_Leer_IMU, BIOS_WAIT_FOREVER);
            Leer_IMU_MPU6050(I2C_PRINCIPAL, IMU6050, &Lecturas_IMU);
            *((float32_t*)Datos + nMuestras) = Lecturas_IMU.Valor.x_vel;
            Task_sleep(PERIODO_Leer_IMU);
        }
        arm_mean_f32((float32_t*)Datos, Numero_Muetras_calibracion_IMU,
        &Calibracion_IMU.Des_est_Vel_x);
        Calibracion_IMU.Media_Vel_x = (int16_t)Calibracion_IMU.Des_est_Vel_x;
        arm_std_f32((float32_t*)Datos, Numero_Muetras_calibracion_IMU,
        &Calibracion_IMU.Des_est_Vel_x);

        for(nMuestras=0; nMuestras<Numero_Muetras_calibracion_IMU; nMuestras++){
            Semaphore_pend(SEMAPHORE_Leer_IMU, BIOS_WAIT_FOREVER);
            Leer_IMU_MPU6050(I2C_PRINCIPAL, IMU6050, &Lecturas_IMU);
            *((float32_t*)Datos + nMuestras) = Lecturas_IMU.Valor.y_vel;
            Task_sleep(PERIODO_Leer_IMU);
        }
        arm_mean_f32((float32_t*)Datos, Numero_Muetras_calibracion_IMU,
        &Calibracion_IMU.Des_est_Vel_y);
        Calibracion_IMU.Media_Vel_y = (int16_t)Calibracion_IMU.Des_est_Vel_y;
        arm_std_f32((float32_t*)Datos, Numero_Muetras_calibracion_IMU,
        &Calibracion_IMU.Des_est_Vel_y);

        for(nMuestras=0; nMuestras<Numero_Muetras_calibracion_IMU; nMuestras++){
            Semaphore_pend(SEMAPHORE_Leer_IMU, BIOS_WAIT_FOREVER);
            Leer_IMU_MPU6050(I2C_PRINCIPAL, IMU6050, &Lecturas_IMU);
            *((float32_t*)Datos + nMuestras) = Lecturas_IMU.Valor.z_vel;
            Task_sleep(PERIODO_Leer_IMU);
        }
        arm_mean_f32((float32_t*)Datos, Numero_Muetras_calibracion_IMU,
        &Calibracion_IMU.Des_est_Vel_z);
        Calibracion_IMU.Media_Vel_z = (int16_t)Calibracion_IMU.Des_est_Vel_z;
        arm_std_f32((float32_t*)Datos, Numero_Muetras_calibracion_IMU,
        &Calibracion_IMU.Des_est_Vel_z);
    }

    Escribir_servidor_Lecturas_IMU(&Lecturas_IMU);
#endif
```

```
#ifdef IMU_MPU9250

    if(ModoCalibracionIMU == CALIBRACION_ACELEROMETRO || ModoCalibracionIMU ==
    CALIBRACION_COMPLETA_IMU){
        for(nMuestras=0; nMuestras<Numero_Muetras_calibracion_IMU; nMuestras++){
            Semaphore_pend(SEMAPHORE_Leer_IMU, BIOS_WAIT_FOREVER);
            Leer_IMU_MPU9250(I2C_PRINCIPAL, IMU9250, &Lecturas_9DOF_IMU);
            *((float32_t*)Datos + nMuestras) = Lecturas_9DOF_IMU.Valor.x_acel;
            Task_sleep(PERIODO_Leer_IMU);
        }
        arm_mean_f32((float32_t*)Datos, Numero_Muetras_calibracion_IMU,
        &Calibracion_IMU.Des_est_Acel_x);
        Calibracion_IMU.Media_Acel_x = (int16_t)Calibracion_IMU.Des_est_Acel_x;
        arm_std_f32((float32_t*)Datos, Numero_Muetras_calibracion_IMU,
        &Calibracion_IMU.Des_est_Acel_x);

        for(nMuestras=0; nMuestras<Numero_Muetras_calibracion_IMU; nMuestras++){
            Semaphore_pend(SEMAPHORE_Leer_IMU, BIOS_WAIT_FOREVER);
            Leer_IMU_MPU9250(I2C_PRINCIPAL, IMU9250, &Lecturas_9DOF_IMU);
            *((float32_t*)Datos + nMuestras) = Lecturas_9DOF_IMU.Valor.y_acel;
            Task_sleep(PERIODO_Leer_IMU);
        }
        arm_mean_f32((float32_t*)Datos, Numero_Muetras_calibracion_IMU,
        &Calibracion_IMU.Des_est_Acel_y);
        Calibracion_IMU.Media_Acel_y = (int16_t)Calibracion_IMU.Des_est_Acel_y;
        arm_std_f32((float32_t*)Datos, Numero_Muetras_calibracion_IMU,
        &Calibracion_IMU.Des_est_Acel_y);

        for(nMuestras=0; nMuestras<Numero_Muetras_calibracion_IMU; nMuestras++){
            Semaphore_pend(SEMAPHORE_Leer_IMU, BIOS_WAIT_FOREVER);
            Leer_IMU_MPU9250(I2C_PRINCIPAL, IMU9250, &Lecturas_9DOF_IMU);
            *((float32_t*)Datos + nMuestras) = Lecturas_9DOF_IMU.Valor.z_acel;
            Task_sleep(PERIODO_Leer_IMU);
        }
        arm_mean_f32((float32_t*)Datos, Numero_Muetras_calibracion_IMU,
        &Calibracion_IMU.Des_est_Acel_z);
        Calibracion_IMU.Media_Acel_z = (int16_t)Calibracion_IMU.Des_est_Acel_z;
        Gravedad  = sqrt(pow(Calibracion_IMU.Media_Acel_x,2) +
        pow(Calibracion_IMU.Media_Acel_y,2) + pow(Calibracion_IMU.Media_Acel_z,2));
        Calibracion_IMU.Media_Acel_z -= (int16_t)sqrt(pow(Calibracion_IMU.Media_Acel_x,2) +
        pow(Calibracion_IMU.Media_Acel_y,2) + pow(Calibracion_IMU.Media_Acel_z,2));
        arm_std_f32((float32_t*)Datos, Numero_Muetras_calibracion_IMU,
        &Calibracion_IMU.Des_est_Acel_z);
    }

    if(ModoCalibracionIMU == CALIBRACION_GIROSCOPO || ModoCalibracionIMU ==
    CALIBRACION_COMPLETA_IMU){
        for(nMuestras=0; nMuestras<Numero_Muetras_calibracion_IMU; nMuestras++){
            Semaphore_pend(SEMAPHORE_Leer_IMU, BIOS_WAIT_FOREVER);
            Leer_IMU_MPU9250(I2C_PRINCIPAL, IMU9250, &Lecturas_9DOF_IMU);
            *((float32_t*)Datos + nMuestras) = Lecturas_9DOF_IMU.Valor.x_vel;
            Task_sleep(PERIODO_Leer_IMU);
        }
        arm_mean_f32((float32_t*)Datos, Numero_Muetras_calibracion_IMU,
        &Calibracion_IMU.Des_est_Vel_x);
        Calibracion_IMU.Media_Vel_x = (int16_t)Calibracion_IMU.Des_est_Vel_x;
        arm_std_f32((float32_t*)Datos, Numero_Muetras_calibracion_IMU,
        &Calibracion_IMU.Des_est_Vel_x);

        for(nMuestras=0; nMuestras<Numero_Muetras_calibracion_IMU; nMuestras++){
            Semaphore_pend(SEMAPHORE_Leer_IMU, BIOS_WAIT_FOREVER);
            Leer_IMU_MPU9250(I2C_PRINCIPAL, IMU9250, &Lecturas_9DOF_IMU);
            *((float32_t*)Datos + nMuestras) = Lecturas_9DOF_IMU.Valor.y_vel;
            Task_sleep(PERIODO_Leer_IMU);
        }
        arm_mean_f32((float32_t*)Datos, Numero_Muetras_calibracion_IMU,
        &Calibracion_IMU.Des_est_Vel_y);
```

```
        Calibracion_IMU.Media_Vel_y = (int16_t)Calibracion_IMU.Des_est_Vel_y;
        arm_std_f32((float32_t*)Datos, Numero_Muetras_calibracion_IMU,
        &Calibracion_IMU.Des_est_Vel_y);

        for(nMuestras=0; nMuestras<Numero_Muetras_calibracion_IMU; nMuestras++){
            Semaphore_pend(SEMAPHORE_Leer_IMU, BIOS_WAIT_FOREVER);
            Leer_IMU_MPU9250(I2C_PRINCIPAL, IMU9250, &Lecturas_9DOF_IMU);
            *((float32_t*)Datos + nMuestras) = Lecturas_9DOF_IMU.Valor.z_vel;
            Task_sleep(PERIODO_Leer_IMU);
        }
        arm_mean_f32((float32_t*)Datos, Numero_Muetras_calibracion_IMU,
        &Calibracion_IMU.Des_est_Vel_z);
        Calibracion_IMU.Media_Vel_z = (int16_t)Calibracion_IMU.Des_est_Vel_z;
        arm_std_f32((float32_t*)Datos, Numero_Muetras_calibracion_IMU,
        &Calibracion_IMU.Des_est_Vel_z);
    }

    Escribir_servidor_Lecturas_IMU_9DOF(&Lecturas_9DOF_IMU);
#endif

    Memory_free(NULL, Datos, Numero_Muetras_calibracion_IMU*4);

    PWM_setDuty(PWM0, Pulso_minimo_PWM_motor);
    PWM_setDuty(PWM1, Pulso_minimo_PWM_motor);
    PWM_setDuty(PWM2, Pulso_minimo_PWM_motor);
    PWM_setDuty(PWM3, Pulso_minimo_PWM_motor);

//ARRANCA LA TAREA DE AHRS
    Semaphore_post(SEMAPHORE_Calculo_AHRS);

    GPIO_write(QUAD_BOARD_LED_RED, 1);
    GPIO_write(QUAD_BOARD_LED_GREEN, 1);
    GPIO_write(QUAD_BOARD_LED_BLUE, 0);

    while(1){
        Semaphore_pend(SEMAPHORE_Leer_IMU, BIOS_WAIT_FOREVER);

        //..IMU.............//
#ifdef IMU_MPU6050
        Leer_IMU_MPU6050(I2C_PRINCIPAL, IMU6050, &Lecturas_IMU);

        //..Offset..//
        Lecturas_IMU.Valor.x_vel -= Calibracion_IMU.Media_Vel_x;
        Lecturas_IMU.Valor.y_vel -= Calibracion_IMU.Media_Vel_y;
        Lecturas_IMU.Valor.z_vel -= Calibracion_IMU.Media_Vel_z;

        Mailbox_pend(Buzon_Lecturas_IMU, NULL, BIOS_NO_WAIT);
        Mailbox_post(Buzon_Lecturas_IMU, &Lecturas_IMU, BIOS_NO_WAIT);

        //..Filtrado...Lectura...IMU.........//
        aux = (float32_t)Lecturas_IMU.Valor.x_acel;
        arm_biquad_cascade_df1_f32(&Filtro_Acel_X, &aux, &Lecturas_matriz[0], 1);
        aux = (float32_t)Lecturas_IMU.Valor.y_acel;
        arm_biquad_cascade_df1_f32(&Filtro_Acel_Y, &aux, &Lecturas_matriz[1], 1);
        aux = (float32_t)Lecturas_IMU.Valor.z_acel;
        arm_biquad_cascade_df1_f32(&Filtro_Acel_Z, &aux, &Lecturas_matriz[2], 1);

        arm_mat_mult_f32(&Calibracion_IMU.Correccion_Alineamiento, &Lecturas,
        &Lecturas_Corregidas);

        Lecturas_IMU.Valor.x_acel = (int16_t)Lecturas_Corregidas_matriz[0];
        Lecturas_IMU.Valor.y_acel = (int16_t)Lecturas_Corregidas_matriz[1];
        Lecturas_IMU.Valor.z_acel = (int16_t)Lecturas_Corregidas_matriz[2];

    #ifdef Filtrado_Vel_IMU
        aux = (float32_t)Lecturas_IMU.Valor.x_vel;
        arm_biquad_cascade_df1_f32(&Filtro_Vel_X, &aux, &Lecturas_matriz[0], 1);
        aux = (float32_t)Lecturas_IMU.Valor.y_vel;
```

```
        arm_biquad_cascade_df1_f32(&Filtro_Vel_Y, &aux, &Lecturas_matriz[1], 1);
        aux = (float32_t)Lecturas_IMU.Valor.z_vel;
        arm_biquad_cascade_df1_f32(&Filtro_Vel_Z, &aux, &Lecturas_matriz[2], 1);

        arm_mat_mult_f32(&Calibracion_IMU.Correccion_Alineamiento, &Lecturas,
        &Lecturas_Corregidas);

        Lecturas_IMU.Valor.x_vel = (int16_t)Lecturas_Corregidas_matriz[0];
        Lecturas_IMU.Valor.y_vel = (int16_t)Lecturas_Corregidas_matriz[1];
        Lecturas_IMU.Valor.z_vel = (int16_t)Lecturas_Corregidas_matriz[2];

    #endif
        Escribir_servidor_Lecturas_IMU(&Lecturas_IMU);
#endif

#ifdef IMU_MPU9250
        Leer_IMU_MPU9250(I2C_PRINCIPAL, IMU9250, &Lecturas_9DOF_IMU);

        //..Offset..//
        Lecturas_9DOF_IMU.Valor.x_vel -= Calibracion_IMU.Media_Vel_x;
        Lecturas_9DOF_IMU.Valor.y_vel -= Calibracion_IMU.Media_Vel_y;
        Lecturas_9DOF_IMU.Valor.z_vel -= Calibracion_IMU.Media_Vel_z;

        Mailbox_pend(Buzon_Lecturas_IMU, NULL, BIOS_NO_WAIT);
        Mailbox_post(Buzon_Lecturas_IMU, &Lecturas_9DOF_IMU, BIOS_NO_WAIT);

        //..Filtrado...Lectura...IMU.........//
        aux = (float32_t)Lecturas_9DOF_IMU.Valor.x_acel;
        arm_biquad_cascade_df1_f32(&Filtro_Acel_X, &aux, &Lecturas_matriz[0], 1);
        aux = (float32_t)Lecturas_9DOF_IMU.Valor.y_acel;
        arm_biquad_cascade_df1_f32(&Filtro_Acel_Y, &aux, &Lecturas_matriz[1], 1);
        aux = (float32_t)Lecturas_9DOF_IMU.Valor.z_acel;
        arm_biquad_cascade_df1_f32(&Filtro_Acel_Z, &aux, &Lecturas_matriz[2], 1);

        arm_mat_mult_f32(&Calibracion_IMU.Correccion_Alineamiento, &Lecturas,
        &Lecturas_Corregidas);

        Lecturas_9DOF_IMU.Valor.x_acel = (int16_t)Lecturas_Corregidas_matriz[0];
        Lecturas_9DOF_IMU.Valor.y_acel = (int16_t)Lecturas_Corregidas_matriz[1];
        Lecturas_9DOF_IMU.Valor.z_acel = (int16_t)Lecturas_Corregidas_matriz[2];

    #ifdef Filtrado_Vel_IMU
        aux = (float32_t)Lecturas_9DOF_IMU.Valor.x_vel;
        arm_biquad_cascade_df1_f32(&Filtro_Vel_X, &aux, &Lecturas_matriz[0], 1);
        aux = (float32_t)Lecturas_9DOF_IMU.Valor.y_vel;
        arm_biquad_cascade_df1_f32(&Filtro_Vel_Y, &aux, &Lecturas_matriz[1], 1);
        aux = (float32_t)Lecturas_9DOF_IMU.Valor.z_vel;
        arm_biquad_cascade_df1_f32(&Filtro_Vel_Z, &aux, &Lecturas_matriz[2], 1);

        arm_mat_mult_f32(&Calibracion_IMU.Correccion_Alineamiento, &Lecturas,
        &Lecturas_Corregidas);

        Lecturas_9DOF_IMU.Valor.x_vel = (int16_t)Lecturas_Corregidas_matriz[0];
        Lecturas_9DOF_IMU.Valor.y_vel = (int16_t)Lecturas_Corregidas_matriz[1];
        Lecturas_9DOF_IMU.Valor.z_vel = (int16_t)Lecturas_Corregidas_matriz[2];

    #endif
        Escribir_servidor_Lecturas_IMU_9DOF(&Lecturas_9DOF_IMU);
#endif

#ifdef GYRO_L3G4200
        Leer_Giroscopo_L3G4200(I2C_PRINCIPAL, Giroscopo_L3G4200, &Lecturas_Giroscopo);
#endif

#ifdef ROT_GYRO
        Lecturas_Giroscopo_Rotadas.Valor.x_vel = -Lecturas_Giroscopo.Valor.y_vel *
        Rot_sin_giro + Lecturas_Giroscopo.Valor.x_vel * Rot_cos_giro;
        Lecturas_Giroscopo_Rotadas.Valor.y_vel =  Lecturas_Giroscopo.Valor.x_vel *
```

```
        Rot_sin_giro + Lecturas_Giroscopo.Valor.y_vel * Rot_cos_giro;
        Lecturas_Giroscopo_Rotadas.Valor.z_vel =  Lecturas_Giroscopo.Valor.z_vel;
#endif

#ifdef GYRO_L3G4200
        Escribir_servidor_Lecturas_Giroscopo(&Lecturas_Giroscopo);
#endif

#ifdef COMPASS_HMC5883L
        Leer_Brujula_HMC5883L(I2C_PRINCIPAL, Brujula_HMC5883L, &Lecturas_Brujula);
        //Mailbox_post(Buzon_Calibracion_Brujula, &Lecturas_Brujula, BIOS_NO_WAIT);
        Escribir_servidor_Lecturas_Brujula(&Lecturas_Brujula);
#endif

#ifdef IMU_MPU6050
    switch(ModoTelemetria){
        case(TELEMETRIA_IMU):
                Telemetria_IMU.Acel[0] = Lecturas_IMU.Valor.x_acel;
                Telemetria_IMU.Acel[1] = Lecturas_IMU.Valor.y_acel;
                Telemetria_IMU.Acel[2] = Lecturas_IMU.Valor.z_acel;

#ifdef GYRO_L3G4200
                        Telemetria_IMU.Gyro[0] = Lecturas_IMU.Valor.x_vel;
                        Telemetria_IMU.Gyro[1] = Lecturas_IMU.Valor.y_vel;
                        Telemetria_IMU.Gyro[2] = Lecturas_IMU.Valor.z_vel;
#endif
                Telemetria_IMU.Mag[0] = Lecturas_Brujula.Valor.Magnetismo_x;
                Telemetria_IMU.Mag[1] = Lecturas_Brujula.Valor.Magnetismo_y;
                Telemetria_IMU.Mag[2] = Lecturas_Brujula.Valor.Magnetismo_z;
                UART_write(UART_BT_TELEMETRIA, &Telemetria_IMU, sizeof(Telemetria_IMU));
        break;
        }
#endif

#ifdef IMU_MPU9250
    switch(ModoTelemetria){
        case(TELEMETRIA_IMU):
                Telemetria_IMU.Acel[0] = Lecturas_9DOF_IMU.Valor.x_acel;
                Telemetria_IMU.Acel[1] = Lecturas_9DOF_IMU.Valor.y_acel;
                Telemetria_IMU.Acel[2] = Lecturas_9DOF_IMU.Valor.z_acel;

                Telemetria_IMU.Gyro[0] = Lecturas_9DOF_IMU.Valor.x_vel;
                Telemetria_IMU.Gyro[1] = Lecturas_9DOF_IMU.Valor.y_vel;
                Telemetria_IMU.Gyro[2] = Lecturas_9DOF_IMU.Valor.z_vel;

                Telemetria_IMU.Mag[0] = Lecturas_9DOF_IMU.Valor.x_mag;
                Telemetria_IMU.Mag[1] = Lecturas_9DOF_IMU.Valor.y_mag;
                Telemetria_IMU.Mag[2] = Lecturas_9DOF_IMU.Valor.z_mag;
                UART_write(UART_BT_TELEMETRIA, &Telemetria_IMU, sizeof(Telemetria_IMU));
        break;
        }
#endif
    }
}

void CLK_Lectura_Datos_IMU(){
    Semaphore_post(SEMAPHORE_Leer_IMU);
}

//....Calculo_AHRS...........//
void Calculo_AHRS(UArg arg0, UArg arg1){

#ifdef IMU_MPU6050
    tpLecturas_IMU Lecturas_IMU;
#endif
#ifdef IMU_MPU9250
    tpLecturas_9DOF_IMU Lecturas_9DOF_IMU;
#endif
```

```
#ifdef GYRO_L3G4200
    tpLecturas_Giroscopo Lecturas_Giroscopo;
#endif
#ifdef COMPASS_HMC5883L
    tpLecturas_Brujula Lecturas_Brujula;
#endif

    static tpAHRS AHRS = {
        .DCM_matriz = {1, 0, 0, 0, 1, 0, 0, 0, 1},
        .DCM = {3, 3, (float32_t *)AHRS.DCM_matriz},
        .Kp_Roll_Pitch = Kp_ROLLPITCH,
        .Ki_Roll_Pitch = Ki_ROLLPITCH,
        .Kp_Yaw = Kp_YAW,
        .Ki_Yaw = Ki_YAW,
        .Periodo_Muestreo = PERIODO_Calculo_AHRS / 1000.0
    };


    Semaphore_pend(SEMAPHORE_Calculo_AHRS, BIOS_WAIT_FOREVER);

    Clock_start(CLOCK_Calculo_AHRS);
    Semaphore_pend(SEMAPHORE_Calculo_AHRS, BIOS_WAIT_FOREVER);

    ResetDCM();
    Leer_servidor_DCM((float32_t*)AHRS.DCM_matriz);
    Leer_servidor_RPY(&AHRS.Roll, &AHRS.Pitch, &AHRS.Yaw);

    Semaphore_post(SEMAPHORE_Coordinador);
    while(1){
        Semaphore_pend(SEMAPHORE_Calculo_AHRS, BIOS_WAIT_FOREVER);
#ifdef IMU_MPU6050
        Leer_servidor_Lecturas_IMU(&Lecturas_IMU_Control);
#endif
#ifdef IMU_MPU9250
        Leer_servidor_Lecturas_IMU_9DOF(&Lecturas_9DOF_IMU);
#endif
#ifdef GYRO_L3G4200
        Leer_servidor_Lecturas_Giroscopo(&Lecturas_Giroscopo_Control);
#endif
#ifdef COMPASS_HMC5883L
        Leer_servidor_Lecturas_Brujula(&Lecturas_Brujula_control);
#endif

        //.................AHRS.................//
                //..ACEL..//
#ifdef IMU_MPU6050

        AHRS.Vector_Aceleracion_lineal[0] = Lecturas_IMU.Valor.x_acel;
        AHRS.Vector_Aceleracion_lineal[1] = Lecturas_IMU.Valor.y_acel;
        AHRS.Vector_Aceleracion_lineal[2] = Lecturas_IMU.Valor.z_acel;
#endif
#ifdef IMU_MPU9250
        AHRS.Vector_Aceleracion_lineal[0] = Lecturas_9DOF_IMU.Valor.x_acel;
        AHRS.Vector_Aceleracion_lineal[1] = Lecturas_9DOF_IMU.Valor.y_acel;
        AHRS.Vector_Aceleracion_lineal[2] = Lecturas_9DOF_IMU.Valor.z_acel;
#endif
                //..GYRO..//
#ifdef IMU_MPU6050
        AHRS.Vector_Velocidad_Angular[0] =
        CONVERTIR_A_RADIANES((float32_t)Lecturas_IMU.Valor.x_vel /
        IMU6050.Sensibilidad_Giroscopo);
        AHRS.Vector_Velocidad_Angular[1] =
        CONVERTIR_A_RADIANES((float32_t)Lecturas_IMU.Valor.y_vel /
        IMU6050.Sensibilidad_Giroscopo);
        AHRS.Vector_Velocidad_Angular[2] =
        CONVERTIR_A_RADIANES((float32_t)Lecturas_IMU.Valor.z_vel /
        IMU6050.Sensibilidad_Giroscopo);
#endif
```

```
#ifdef IMU_MPU9250
        AHRS.Vector_Velocidad_Angular[0] =
        CONVERTIR_A_RADIANES((float32_t)Lecturas_9DOF_IMU.Valor.x_vel /
        IMU9250.Sensibilidad_Giroscopo);
        AHRS.Vector_Velocidad_Angular[1] =
        CONVERTIR_A_RADIANES((float32_t)Lecturas_9DOF_IMU.Valor.y_vel /
        IMU9250.Sensibilidad_Giroscopo);
        AHRS.Vector_Velocidad_Angular[2] =
        CONVERTIR_A_RADIANES((float32_t)Lecturas_9DOF_IMU.Valor.z_vel /
        IMU9250.Sensibilidad_Giroscopo);
#endif

#ifdef GYRO_L3G4200
        AHRS.Vector_Velocidad_Angular[0] =
        CONVERTIR_A_RADIANES((float32_t)Lecturas_Giroscopo_Control.Valor.x_vel /
        Giroscopo_L3G4200.Sensibilidad_Giroscopo);
        AHRS.Vector_Velocidad_Angular[1] =
        CONVERTIR_A_RADIANES((float32_t)Lecturas_Giroscopo_Control.Valor.y_vel /
        Giroscopo_L3G4200.Sensibilidad_Giroscopo);
        AHRS.Vector_Velocidad_Angular[2] =
        CONVERTIR_A_RADIANES((float32_t)Lecturas_Giroscopo_Control.Valor.z_vel /
        Giroscopo_L3G4200.Sensibilidad_Giroscopo);
#endif

                //...BRUJULA...//
#ifdef COMPASS_HMC5883L
        AHRS.Vector_Magnetico[0] = Lecturas_Brujula_control.Valor.Magnetismo_x;
        AHRS.Vector_Magnetico[1] = Lecturas_Brujula_control.Valor.Magnetismo_y;
        AHRS.Vector_Magnetico[2] = Lecturas_Brujula_control.Valor.Magnetismo_z;
#endif

#ifdef MAG
        Algortimo_DCM_MAG(&AHRS);
#else
        Algortimo_DCM_NO_YAW(&AHRS);
#endif
        Escribir_servidor_DCM((float32_t*)AHRS.DCM_matriz);
        Escribir_servidor_RPY(&AHRS.Roll, &AHRS.Pitch, &AHRS.Yaw);
    }
}

void CLK_Calculo_AHRS(){
    Semaphore_post(SEMAPHORE_Calculo_AHRS);
}
//....Coordinador......//

void Coordinador(UArg arg0, UArg arg1){
//..Ref.............................

    tpOrden Orden = DATO_ANTERIOR;
    uint16_t i = 0;

    tpLectura_Radio Lectura_Radio;
    I2C_Transaction I2C_Transmision;
    bool estado_Transmision = false;
    I2C_Transmision.slaveAddress = Dir_AUX;
    I2C_Transmision.writeBuf = NULL;
    I2C_Transmision.writeCount = 0;
    I2C_Transmision.readBuf = &Lectura_Radio;
    I2C_Transmision.readCount = sizeof(Lectura_Radio)-1;

    static uint8_t Temporizador_Ticks = ticks_arranque_vuelo; //3seg

    float32_t Canal[8];
    float32_t Referencia[4] = {0, 0, 0, 0};
    float32_t Angulos[3] = {0, 0, 0};
```

```
    Semaphore_pend(SEMAPHORE_Coordinador, BIOS_WAIT_FOREVER);

    WatchDog_0 = Watchdog_open(QUAD_BOARD_WATCHDOG0 , &PARAMS_WatchDog_0);
    Watchdog_clear(WatchDog_0);

    Clock_start(CLOCK_Coordinador);

    estado_Transmision = I2C_transfer(I2C_AUX, &I2C_Transmision);


    while(!(estado_Transmision && Lectura_Radio.Canal_PWM[2] < 1100)){
        Semaphore_pend(SEMAPHORE_Coordinador, BIOS_WAIT_FOREVER);
        estado_Transmision = I2C_transfer(I2C_AUX, &I2C_Transmision);
        if(Lectura_Radio.Error_conexion != 0){
            Watchdog_clear(WatchDog_0);
        }
    }

    while(1){
        Semaphore_pend(SEMAPHORE_Coordinador, BIOS_WAIT_FOREVER);

        estado_Transmision = I2C_transfer(I2C_AUX, &I2C_Transmision);
//      Lectura_Radio.Canal_PWM[7] =
Lectura_Radio.Canal_PWM[6];////////////////////////////////////////////////////////////////
/


        if(estado_Transmision && Lectura_Radio.Error_conexion != 0){
            Watchdog_clear(WatchDog_0);

            Canal[0] = ( ( Lectura_Radio.Canal_PWM[0] -
            Calibracion_Receptor[0].Rango_Entrada[0] ) * (
            Calibracion_Receptor[0].Rango_Salida[1] -
            Calibracion_Receptor[0].Rango_Salida[0] ) / (
            Calibracion_Receptor[0].Rango_Entrada[1] -
            Calibracion_Receptor[0].Rango_Entrada[0] ) +
            Calibracion_Receptor[0].Rango_Salida[0] );
            Canal[1] = ( ( Lectura_Radio.Canal_PWM[1] -
            Calibracion_Receptor[1].Rango_Entrada[0] ) * (
            Calibracion_Receptor[1].Rango_Salida[1] -
            Calibracion_Receptor[1].Rango_Salida[0] ) / (
            Calibracion_Receptor[1].Rango_Entrada[1] -
            Calibracion_Receptor[1].Rango_Entrada[0] ) +
            Calibracion_Receptor[1].Rango_Salida[0] );
            Canal[2] = ( ( Lectura_Radio.Canal_PWM[2] -
            Calibracion_Receptor[2].Rango_Entrada[0] ) * (
            Calibracion_Receptor[2].Rango_Salida[1] -
            Calibracion_Receptor[2].Rango_Salida[0] ) / (
            Calibracion_Receptor[2].Rango_Entrada[1] -
            Calibracion_Receptor[2].Rango_Entrada[0] ) +
            Calibracion_Receptor[2].Rango_Salida[0] );
            Canal[3] = ( ( Lectura_Radio.Canal_PWM[3] -
            Calibracion_Receptor[3].Rango_Entrada[0] ) * (
            Calibracion_Receptor[3].Rango_Salida[1] -
            Calibracion_Receptor[3].Rango_Salida[0] ) / (
            Calibracion_Receptor[3].Rango_Entrada[1] -
            Calibracion_Receptor[3].Rango_Entrada[0] ) +
            Calibracion_Receptor[3].Rango_Salida[0] );
            Canal[4] = ( ( Lectura_Radio.Canal_PWM[4] -
            Calibracion_Receptor[4].Rango_Entrada[0] ) * (
            Calibracion_Receptor[4].Rango_Salida[1] -
            Calibracion_Receptor[4].Rango_Salida[0] ) / (
            Calibracion_Receptor[4].Rango_Entrada[1] -
            Calibracion_Receptor[4].Rango_Entrada[0] ) +
            Calibracion_Receptor[4].Rango_Salida[0] );
            Canal[5] = ( ( Lectura_Radio.Canal_PWM[5] -
            Calibracion_Receptor[5].Rango_Entrada[0] ) * (
```

```
            Calibracion_Receptor[5].Rango_Salida[1] -
            Calibracion_Receptor[5].Rango_Salida[0] ) / (
            Calibracion_Receptor[5].Rango_Entrada[1] -
            Calibracion_Receptor[5].Rango_Entrada[0] ) +
            Calibracion_Receptor[5].Rango_Salida[0] );
            Canal[6] = ( ( Lectura_Radio.Canal_PWM[6] -
            Calibracion_Receptor[6].Rango_Entrada[0] ) * (
            Calibracion_Receptor[6].Rango_Salida[1] -
            Calibracion_Receptor[6].Rango_Salida[0] ) / (
            Calibracion_Receptor[6].Rango_Entrada[1] -
            Calibracion_Receptor[6].Rango_Entrada[0] ) +
            Calibracion_Receptor[6].Rango_Salida[0] );
            Canal[7] = ( ( Lectura_Radio.Canal_PWM[7] -
            Calibracion_Receptor[7].Rango_Entrada[0] ) * (
            Calibracion_Receptor[7].Rango_Salida[1] -
            Calibracion_Receptor[7].Rango_Salida[0] ) / (
            Calibracion_Receptor[7].Rango_Entrada[1] -
            Calibracion_Receptor[7].Rango_Entrada[0] ) +
            Calibracion_Receptor[7].Rango_Salida[0] );

            //......PULSADORES.................................//
            if(!GPIO_read(QUAD_BOARD_SW2)){
                while(!GPIO_read(QUAD_BOARD_SW2));

                switch(Estado_Sistema){
                    case ESPERA:

                    break;
                    case DEBUG:
                        Estado_Sistema = CALIBRACION;

                        GPIO_write(QUAD_BOARD_LED_RED, 0);
                        GPIO_write(QUAD_BOARD_LED_GREEN, 1);
                        GPIO_write(QUAD_BOARD_LED_BLUE, 1);

                        Calibracion_Receptor[0].Rango_Entrada[0] = 1500;
                        Calibracion_Receptor[0].Rango_Entrada[1] = 1500;
                        Calibracion_Receptor[1].Rango_Entrada[0] = 1500;
                        Calibracion_Receptor[1].Rango_Entrada[1] = 1500;
                        Calibracion_Receptor[2].Rango_Entrada[0] = 1500;
                        Calibracion_Receptor[2].Rango_Entrada[1] = 1500;
                        Calibracion_Receptor[3].Rango_Entrada[0] = 1500;
                        Calibracion_Receptor[3].Rango_Entrada[1] = 1500;
                        Calibracion_Receptor[4].Rango_Entrada[0] = 1500;
                        Calibracion_Receptor[4].Rango_Entrada[1] = 1500;
                        Calibracion_Receptor[5].Rango_Entrada[0] = 1500;
                        Calibracion_Receptor[5].Rango_Entrada[1] = 1500;
                        Calibracion_Receptor[6].Rango_Entrada[0] = 1500;
                        Calibracion_Receptor[6].Rango_Entrada[1] = 1500;
//                      Calibracion_Receptor[7].Rango_Entrada[0] = 1500;
Calibracion_Receptor[7].Rango_Entrada[1] = 1500;

                    break;
                    case CALIBRACION:
                        Estado_Sistema = ESPERA;

                        GPIO_write(QUAD_BOARD_LED_RED, 1);
                        GPIO_write(QUAD_BOARD_LED_GREEN, 1);
                        GPIO_write(QUAD_BOARD_LED_BLUE, 0);

                    break;
                }

            //...........STICK_IZQUIERDA
            NEGATIVO.....................................................//
            }else if((Canal[0] <= -0.9) && (Canal[2] < 0.05)){
                if (Temporizador_Ticks-- == 0){
```

```
            switch(Estado_Sistema){
                case DEBUG:
                    //resetear variables
                break;
                case ESPERA:
                    Estado_Sistema = DEBUG;

                    GPIO_write(QUAD_BOARD_LED_RED, 1);
                    GPIO_write(QUAD_BOARD_LED_GREEN, 1);
                    GPIO_write(QUAD_BOARD_LED_BLUE, 1);
                break;
                case VUELO:
                    Estado_Sistema = ESPERA;

                    Clock_stop(CLOCK_Control);
                    Semaphore_reset(SEMAPHORE_Control, 0);

                    PWM_setDuty(PWM0, Pulso_minimo_PWM_motor);
                    PWM_setDuty(PWM1, Pulso_minimo_PWM_motor);
                    PWM_setDuty(PWM2, Pulso_minimo_PWM_motor);
                    PWM_setDuty(PWM3, Pulso_minimo_PWM_motor);

                    GPIO_write(QUAD_BOARD_LED_RED, 1);
                    GPIO_write(QUAD_BOARD_LED_GREEN, 1);
                    GPIO_write(QUAD_BOARD_LED_BLUE, 0);
                break;
            }
        }
    //...........STICK_IZQUIERDA
    POSITIVO.....................................................//
    }else if((Canal[0] >= 0.9) && (Canal[2]) < 0.05) {
        if (--Temporizador_Ticks == 0){

            switch(Estado_Sistema){
                case DEBUG:
                    Estado_Sistema = ESPERA;

                    GPIO_write(QUAD_BOARD_LED_RED, 1);
                    GPIO_write(QUAD_BOARD_LED_GREEN, 1);
                    GPIO_write(QUAD_BOARD_LED_BLUE, 0);

                    Leer_servidor_RPY(NULL, NULL, &Posicion_inicial);
                    Posicion_inicial = CONVERTIR_A_GRADOS(Posicion_inicial);

                break;
                case ESPERA:
                    Estado_Sistema = VUELO;

                    GPIO_write(QUAD_BOARD_LED_RED, 0);
                    GPIO_write(QUAD_BOARD_LED_GREEN, 1);
                    GPIO_write(QUAD_BOARD_LED_BLUE, 0);

                    Clock_start(CLOCK_Control);
                    Resetear_servidor_Perturbaciones_Estimadas();

                break;
            }
        }
    //...........STICK DERECHA
    NEGATIVO.....................................................//
    }else if((Canal[3] <= -0.9) && (Canal[2]) < 0.05) {
        if (--Temporizador_Ticks == 0){

            switch(Estado_Sistema){
                case DEBUG:
                break;
                case ESPERA:
                    //RESET
```

```
                    Leer_servidor_RPY(&Angulos[1], &Angulos[0], &Angulos[2]);

                    //Iniciamos la matriz de correccion
                    Calibracion_IMU.Giro[1] -= CONVERTIR_A_GRADOS(Angulos[0]);
                    Calibracion_IMU.Giro[2] -= CONVERTIR_A_GRADOS(Angulos[1]);

                    Rotacion_ZXY(&Calibracion_IMU.Correccion_Alineamiento,
                    Calibracion_IMU.Giro, false);

                    GPIO_write(QUAD_BOARD_LED_RED, 0);
                    GPIO_write(QUAD_BOARD_LED_GREEN, 0);
                    GPIO_write(QUAD_BOARD_LED_BLUE, 1);

                    Task_sleep(250);

                    ResetDCM();

                    GPIO_write(QUAD_BOARD_LED_RED, 1);
                    GPIO_write(QUAD_BOARD_LED_GREEN, 1);
                    GPIO_write(QUAD_BOARD_LED_BLUE, 0);
                break;
            }
            Temporizador_Ticks = ticks_arranque_vuelo;
        }
    //...........STICK DERECHA
    POSITIVO.....................................................................//
    }else if((Canal[3] >= 0.9) && (Canal[2]) < 0.05) {
        if (--Temporizador_Ticks == 0){

            switch(Estado_Sistema){
                case DEBUG:
                    Estado_Sistema = IDENTIFICACION;

                    GPIO_write(QUAD_BOARD_LED_RED, 1);
                    GPIO_write(QUAD_BOARD_LED_GREEN, 0);
                    GPIO_write(QUAD_BOARD_LED_BLUE, 1);

                    i = 0;

                    do{
                        Orden = START;
                        UART_write(UART_BT_TELEMETRIA, &Orden, 1);
                        Orden = IDENTIFICAR;
                        UART_write(UART_BT_TELEMETRIA, &Orden, 1);
                        Orden = FINAL;
                        UART_write(UART_BT_TELEMETRIA, &Orden, 1);

                        UART_read(UART_BT_TELEMETRIA, &Orden, 1);

                        Watchdog_clear(WatchDog_0);
                    }while(Orden != IDENTIFICAR && ++i <
                    Num_intentos_conexion_Identificacion);

                    if(Orden == IDENTIFICAR){
                        UART_read(UART_BT_TELEMETRIA, &nDatos_Identifiacion, 4);
                        UART_read(UART_BT_TELEMETRIA, &PuntoTrabajo_motor ,4);

                        Datos = Memory_alloc(NULL, nDatos_Identifiacion*2, 0, &eb);

                        for(i=0; i<nDatos_Identifiacion; i++){
                            UART_read(UART_BT_TELEMETRIA, (int16_t *)Datos + i, 2);
                            Watchdog_clear(WatchDog_0);
                        }

                        UART_read(UART_BT_TELEMETRIA, &Orden, 1);

                        if(Orden == FINAL){
```

```
                            nDatos_leidos = 0;
                            Clock_start(CLOCK_Identificacion);
                        }else{
                            Estado_Sistema = ESPERA;

                            GPIO_write(QUAD_BOARD_LED_RED, 1);
                            GPIO_write(QUAD_BOARD_LED_GREEN, 1);
                            GPIO_write(QUAD_BOARD_LED_BLUE, 0);
                        }
                    }else{

                        Estado_Sistema = ESPERA;

                        GPIO_write(QUAD_BOARD_LED_RED, 1);
                        GPIO_write(QUAD_BOARD_LED_GREEN, 1);
                        GPIO_write(QUAD_BOARD_LED_BLUE, 0);
                    }
                }
            }
        }else{
            Temporizador_Ticks = ticks_arranque_vuelo;
        }

        //.............Accion.............................//
        switch(Estado_Sistema){
            case VUELO:
                if(Canal[7] < -83.2){   //...ERROR....//
//                  Estado_Sistema_Anterior = Estado_Sistema;
                    Estado_Sistema = ERROR;

                    GPIO_write(QUAD_BOARD_LED_RED, 1);
                    GPIO_write(QUAD_BOARD_LED_GREEN, 0);
                    GPIO_write(QUAD_BOARD_LED_BLUE, 0);

                    //Stop a todos los clocks y tareas
                    Clock_stop(CLOCK_Control);
                    Semaphore_reset(SEMAPHORE_Control, 0);
                    Clock_stop(CLOCK_Identificacion);
                    Semaphore_reset(SEMAPHORE_Identificacion, 0);

                    //Parada_motores
                    PWM_setDuty(PWM0, Pulso_minimo_PWM_motor);
                    PWM_setDuty(PWM1, Pulso_minimo_PWM_motor);
                    PWM_setDuty(PWM2, Pulso_minimo_PWM_motor);
                    PWM_setDuty(PWM3, Pulso_minimo_PWM_motor);
                }
                else if((Canal[7] > -83.2) && (Canal[7] <= -50.0)){ Modo_Control =
                ANGULOS_3; ModoPerturbaciones = CORREGIR_PERTURBACIONES; InfoTelemetria
                = TELE_1; }
                else if((Canal[7] > -50.0) && (Canal[7] <= -16.6)){ Modo_Control =
                ANGULOS_3; ModoPerturbaciones = INTEGRAR_PERTURBACIONES_ESTIMADAS;
                InfoTelemetria = TELE_2;}
                else if((Canal[7] > -16.6) && (Canal[7] <=  16.6)){ Modo_Control =
                ANGULOS_3; ModoPerturbaciones = INTEGRAR_PERTURBACIONES; InfoTelemetria
                = TELE_3;}
                else if((Canal[7] >  16.6) && (Canal[7] <=  50.0)){ Modo_Control =
                ANGULOS_3; ModoPerturbaciones = NO_CORREGIR_PERTURBACIONES;
                InfoTelemetria = TELE_0;}
                else if((Canal[7] >  50.0) && (Canal[7] <=  83.2)){ Modo_Control =
                ANGULOS_3; ModoPerturbaciones = NO_CORREGIR_PERTURBACIONES;
                InfoTelemetria = TELE_0;}
                else if((Canal[7] >  83.2))                       { Modo_Control =
                ANGULOS_3; ModoPerturbaciones = NO_CORREGIR_PERTURBACIONES;
                InfoTelemetria = TELE_0;}

                switch(Modo_Control){
                    case ANGULOS_3:
```

```
                        Referencia[0] = Canal[1] * Angulo_Maximo;
                        Referencia[1] = Canal[3] * Angulo_Maximo;
                        Referencia[2] = Canal[0] * Angulo_Maximo;
                        Referencia[3] = Canal[2] * Valor_Empuje_Maximo;
                    break;
                    case ANGULOS_4:
                        Referencia[0] = Canal[1] * Angulo_Maximo;
                        Referencia[1] = Canal[3] * Angulo_Maximo;
                        Referencia[2] = Canal[0] * Angulo_Maximo;
                        Referencia[3] = Canal[2] * Valor_Fuerza_Maximo;
                    break;
                    case EMPUJE:
                        Referencia[0] = Canal[1] * Angulo_Maximo;
                        Referencia[1] = Canal[3] * Angulo_Maximo;
                        Referencia[2] = Canal[0] * Angulo_Maximo;
                        Referencia[3] = Canal[2] * Valor_Empuje_Maximo;
                    break;
                }
                Escribir_servidor_Referencia(Referencia, NULL);
            break;
            case CALIBRACION:

                Calibracion_Receptor[0].Rango_Entrada[0] = Lectura_Radio.Canal_PWM[0] <
                Calibracion_Receptor[0].Rango_Entrada[0] ? Lectura_Radio.Canal_PWM[0] :
                Calibracion_Receptor[0].Rango_Entrada[0];
                Calibracion_Receptor[0].Rango_Entrada[1] = Lectura_Radio.Canal_PWM[0] >
                Calibracion_Receptor[0].Rango_Entrada[1] ? Lectura_Radio.Canal_PWM[0] :
                Calibracion_Receptor[0].Rango_Entrada[1];
                Calibracion_Receptor[1].Rango_Entrada[0] = Lectura_Radio.Canal_PWM[1] <
                Calibracion_Receptor[1].Rango_Entrada[0] ? Lectura_Radio.Canal_PWM[1] :
                Calibracion_Receptor[1].Rango_Entrada[0];
                Calibracion_Receptor[1].Rango_Entrada[1] = Lectura_Radio.Canal_PWM[1] >
                Calibracion_Receptor[1].Rango_Entrada[1] ? Lectura_Radio.Canal_PWM[1] :
                Calibracion_Receptor[1].Rango_Entrada[1];
                Calibracion_Receptor[2].Rango_Entrada[0] = Lectura_Radio.Canal_PWM[2] <
                Calibracion_Receptor[2].Rango_Entrada[0] ? Lectura_Radio.Canal_PWM[2] :
                Calibracion_Receptor[2].Rango_Entrada[0];
                Calibracion_Receptor[2].Rango_Entrada[1] = Lectura_Radio.Canal_PWM[2] >
                Calibracion_Receptor[2].Rango_Entrada[1] ? Lectura_Radio.Canal_PWM[2] :
                Calibracion_Receptor[2].Rango_Entrada[1];
                Calibracion_Receptor[3].Rango_Entrada[0] = Lectura_Radio.Canal_PWM[3] <
                Calibracion_Receptor[3].Rango_Entrada[0] ? Lectura_Radio.Canal_PWM[3] :
                Calibracion_Receptor[3].Rango_Entrada[0];
                Calibracion_Receptor[3].Rango_Entrada[1] = Lectura_Radio.Canal_PWM[3] >
                Calibracion_Receptor[3].Rango_Entrada[1] ? Lectura_Radio.Canal_PWM[3] :
                Calibracion_Receptor[3].Rango_Entrada[1];
                Calibracion_Receptor[4].Rango_Entrada[0] = Lectura_Radio.Canal_PWM[4] <
                Calibracion_Receptor[4].Rango_Entrada[0] ? Lectura_Radio.Canal_PWM[4] :
                Calibracion_Receptor[4].Rango_Entrada[0];
                Calibracion_Receptor[4].Rango_Entrada[1] = Lectura_Radio.Canal_PWM[4] >
                Calibracion_Receptor[4].Rango_Entrada[1] ? Lectura_Radio.Canal_PWM[4] :
                Calibracion_Receptor[4].Rango_Entrada[1];
                Calibracion_Receptor[5].Rango_Entrada[0] = Lectura_Radio.Canal_PWM[5] <
                Calibracion_Receptor[5].Rango_Entrada[0] ? Lectura_Radio.Canal_PWM[5] :
                Calibracion_Receptor[5].Rango_Entrada[0];
                Calibracion_Receptor[5].Rango_Entrada[1] = Lectura_Radio.Canal_PWM[5] >
                Calibracion_Receptor[5].Rango_Entrada[1] ? Lectura_Radio.Canal_PWM[5] :
                Calibracion_Receptor[5].Rango_Entrada[1];
                Calibracion_Receptor[6].Rango_Entrada[0] = Lectura_Radio.Canal_PWM[6] <
                Calibracion_Receptor[6].Rango_Entrada[0] ? Lectura_Radio.Canal_PWM[6] :
                Calibracion_Receptor[6].Rango_Entrada[0];
                Calibracion_Receptor[6].Rango_Entrada[1] = Lectura_Radio.Canal_PWM[6] >
                Calibracion_Receptor[6].Rango_Entrada[1] ? Lectura_Radio.Canal_PWM[6] :
                Calibracion_Receptor[6].Rango_Entrada[1];
//                  Calibracion_Receptor[7].Rango_Entrada[0] = Lectura_Radio.Canal_PWM[7] <
Calibracion_Receptor[7].Rango_Entrada[0] ? Lectura_Radio.Canal_PWM[7] :
Calibracion_Receptor[7].Rango_Entrada[0];
//                  Calibracion_Receptor[7].Rango_Entrada[1] = Lectura_Radio.Canal_PWM[7] >
```

```
Calibracion_Receptor[7].Rango_Entrada[1] ? Lectura_Radio.Canal_PWM[7] :
Calibracion_Receptor[7].Rango_Entrada[1];
                break;
                case ERROR:
                    if (Canal[7] > -83.2){ //..FIN_ERROR..//
                        Estado_Sistema = ESPERA;

                        GPIO_write(QUAD_BOARD_LED_RED, 1);
                        GPIO_write(QUAD_BOARD_LED_GREEN, 1);
                        GPIO_write(QUAD_BOARD_LED_BLUE, 0);

                    }
/*                  if((Canal[7] > -83.2) && (Canal[7] < -50.0)){ Modo_Control = ANGULOS_3;
ModoPerturbaciones = Perturbaciones_Seleccionada; InfoTelemetria = TELE_0; }
                    else if((Canal[7] > -50.0) && (Canal[7] < -16.6)){ Modo_Control =
                    ANGULOS_4; ModoPerturbaciones = Perturbaciones_Seleccionada;
                    InfoTelemetria = TELE_1;}
                    else if((Canal[7] > -16.6) && (Canal[7] <  16.6)){ Modo_Control =
                    EMPUJE; ModoPerturbaciones = Perturbaciones_Seleccionada; InfoTelemetria
                    = TELE_0;}
                    else if((Canal[7] >  16.6) && (Canal[7] <  50.0)){ Modo_Control =
                    ANGULOS_3; ModoPerturbaciones = NO_CORREGIR_PERTURBACIONES;
                    InfoTelemetria = TELE_2;}
                    else if((Canal[7] >  50.0) && (Canal[7] <  83.2)){ Modo_Control =
                    ANGULOS_4; ModoPerturbaciones = NO_CORREGIR_PERTURBACIONES;
                    InfoTelemetria = TELE_3;}
                    else if((Canal[7] > -83.2)){ Modo_Control = EMPUJE; ModoPerturbaciones =
                    NO_CORREGIR_PERTURBACIONES; InfoTelemetria = TELE_4;}
*/                  break;
                case ESPERA:
                case DEBUG:
                    if(Canal[7] < -83.2){   //...ERROR..../
                        Estado_Sistema = ERROR;

                        GPIO_write(QUAD_BOARD_LED_RED, 1);
                        GPIO_write(QUAD_BOARD_LED_GREEN, 0);
                        GPIO_write(QUAD_BOARD_LED_BLUE, 0);

                        //Stop a todos los clocks y tareas
                        Clock_stop(CLOCK_Control);
                        Semaphore_reset(SEMAPHORE_Control, 0);
                        Clock_stop(CLOCK_Identificacion);
                        Semaphore_reset(SEMAPHORE_Identificacion, 0);

                        //Parada_motores
                        PWM_setDuty(PWM0, Pulso_minimo_PWM_motor);
                        PWM_setDuty(PWM1, Pulso_minimo_PWM_motor);
                        PWM_setDuty(PWM2, Pulso_minimo_PWM_motor);
                        PWM_setDuty(PWM3, Pulso_minimo_PWM_motor);
                    }
/*                  else if((Canal[7] > -83.2) && (Canal[7] < -50.0)){ Modo_Control =
ANGULOS_3; ModoPerturbaciones = Perturbaciones_Seleccionada; InfoTelemetria = TELE_0; }
                    else if((Canal[7] > -50.0) && (Canal[7] < -16.6)){ Modo_Control =
                    ANGULOS_4; ModoPerturbaciones = Perturbaciones_Seleccionada;
                    InfoTelemetria = TELE_1;}
                    else if((Canal[7] > -16.6) && (Canal[7] <  16.6)){ Modo_Control =
                    EMPUJE; ModoPerturbaciones = Perturbaciones_Seleccionada; InfoTelemetria
                    = TELE_0;}
                    else if((Canal[7] >  16.6) && (Canal[7] <  50.0)){ Modo_Control =
                    ANGULOS_3; ModoPerturbaciones = NO_CORREGIR_PERTURBACIONES;
                    InfoTelemetria = TELE_2;}
                    else if((Canal[7] >  50.0) && (Canal[7] <  83.2)){ Modo_Control =
                    ANGULOS_4; ModoPerturbaciones = NO_CORREGIR_PERTURBACIONES;
                    InfoTelemetria = TELE_3;}
                    else if((Canal[7] >  83.2)){ Modo_Control = EMPUJE; ModoPerturbaciones =
                    NO_CORREGIR_PERTURBACIONES; InfoTelemetria = TELE_0;}
*/                  break;
            }
```

```
        }
    }
}

void CLK_Coordinador(){
    Semaphore_post(SEMAPHORE_Coordinador);
}

//....Identificacion......//
void Identificacion(UArg arg0, UArg arg1){
    UInt Key;
    tpLecturas_IMU Lecturas_IMU;
    int16_t Aux;
//  uint32_t Rpm;

    GPIO_enableInt(QUAD_BOARD_RPM);

    while(1){
        Semaphore_pend(SEMAPHORE_Identificacion, BIOS_WAIT_FOREVER);

        Aux = *((int16_t *)Datos + nDatos_leidos);
        PWM_setDuty(PWM0, PuntoTrabajo_motor + Aux/2 + Pulso_minimo_PWM_motor);
        PWM_setDuty(PWM2, PuntoTrabajo_motor - Aux/2 + Pulso_minimo_PWM_motor);
        nDatos_leidos++;

        Leer_servidor_Lecturas_IMU(&Lecturas_IMU);

        Key = Hwi_disable();
//      Rpm = (uint32_t)(Frecuencia_CPU / Ticks_por_RPS *60);
        Hwi_restore(Key);

        UART_write(UART_BT_TELEMETRIA, &Lecturas_IMU, 14);
//      UART_write(UART_BT_TELEMETRIA, &Rpm, sizeof(Rpm));

        if((nDatos_leidos == nDatos_Identifiacion) || (Estado_Sistema == ESPERA)){
            Clock_stop(CLOCK_Identificacion);
            Semaphore_reset(SEMAPHORE_Identificacion, 0);
            nDatos_leidos = 0;

            Memory_free(NULL, Datos, nDatos_Identifiacion*2);
            PWM_setDuty(PWM0, Pulso_minimo_PWM_motor);
            PWM_setDuty(PWM1, Pulso_minimo_PWM_motor);
            PWM_setDuty(PWM2, Pulso_minimo_PWM_motor);
            PWM_setDuty(PWM3, Pulso_minimo_PWM_motor);

            Estado_Sistema = ESPERA;

            GPIO_write(QUAD_BOARD_LED_RED, 1);
            GPIO_write(QUAD_BOARD_LED_GREEN, 1);
            GPIO_write(QUAD_BOARD_LED_BLUE, 0);
        }
    }
}

void CLK_Identificacion(){
    Semaphore_post(SEMAPHORE_Identificacion);
}
#ifdef Sensor_RPM
    void ISR_GPIO_RPM(UArg arg0){
        static uint32_t Tick_anterior = 0;

        Ticks_por_RPS =  Clock_getTicks() - Tick_anterior;
    }
#endif
//....Altura.............//

void ISR_GPIO_US(UArg arg0){
    UInt Key;
```

```
    GPIO_clearInt(QUAD_BOARD_ECHO);

    if(GPIO_read(QUAD_BOARD_ECHO) == 0){
        Key = Hwi_disable();
        Timer_stop(US_Timer);
        Altura_US_mm = (Timer_getPeriod(US_Timer) - Timer_getCount(US_Timer))/80*0.340/2 ;
        //microsegondos
        Hwi_restore(Key);
    }else{
        Key = Hwi_disable();
        Timer_setPeriodMicroSecs(US_Timer, (uint32_t)Max_pulso_us);
        Timer_start(US_Timer);
        Hwi_restore(Key);
    }
}

void ISR_Timer_US(){

    if(Timer_getPeriod(US_Timer) == 800){
        GPIO_write(QUAD_BOARD_TRIGG, 0);
    }
    else{
        Altura_US_mm = 0xFF;
    }
}

void Calculo_Altura(UArg arg0, UArg arg1){
    UInt Key;

//  uint32_t Presion_Inicial = 0;
//  tpLecturasBarometro LecturasBarometro;
    tpLecturasBarometro_BMP280 LecturasBarometro_BMP280;
    float32_t Temperatura = 0;
/*
    Iniciar_Barometro(I2C_PRINCIPAL, Direccion_Barometro, &LecturasBarometro);

    Iniciar_Medida_Temp_Barometro(I2C_PRINCIPAL, Direccion_Barometro, &LecturasBarometro);
    Task_sleep(5);
    Leer_Temp_Barometro(I2C_PRINCIPAL, Direccion_Barometro, &LecturasBarometro);
    Iniciar_Medida_Temp_Barometro(I2C_PRINCIPAL, Direccion_Barometro, &LecturasBarometro);
    Task_sleep(26);
    Leer_Presion_Barometro(I2C_PRINCIPAL, Direccion_Barometro, &LecturasBarometro);

    Presion_Inicial = LecturasBarometro.Presion;
*/
    Iniciar_Barometro_BMP280(I2C_PRINCIPAL, &Barometro_BMP280, &LecturasBarometro_BMP280);

    Clock_start(CLOCK_Calculo_Altura);
    while(1){
        Semaphore_pend(SEMAPHORE_Calculo_Altura, BIOS_WAIT_FOREVER);
        //....US....//
        GPIO_write(QUAD_BOARD_TRIGG, 1);
        Timer_setPeriodMicroSecs(US_Timer, (uint32_t)Pulso_arranque_us);
        Key = Hwi_disable();
        Timer_start(US_Timer);
        Hwi_restore(Key);

        Leer_Barometro_BMP280(I2C_PRINCIPAL, &Barometro_BMP280, &LecturasBarometro_BMP280);
        Temperatura = Conversion_Temperatura(&LecturasBarometro_BMP280);
        Altura_Presion_mm = Conversion_Altura(Temperatura, &LecturasBarometro_BMP280);

/*
        //...Barometro....//
        Iniciar_Medida_Temp_Barometro(I2C_PRINCIPAL, Direccion_Barometro, &LecturasBarometro);
        Task_sleep(5);
        Leer_Temp_Barometro(I2C_PRINCIPAL, Direccion_Barometro, &LecturasBarometro);
        Iniciar_Medida_Temp_Barometro(I2C_PRINCIPAL, Direccion_Barometro, &LecturasBarometro);
```

```
        Task_sleep(26);
        Leer_Presion_Barometro(I2C_PRINCIPAL, Direccion_Barometro, &LecturasBarometro);

        Altura_Presion_mm = 4433000 * ( 1 - pow(LecturasBarometro.Presion / Presion_Inicial
        , 1/5.255));
*/
    }
}

void CLK_Calculo_Altura(){
    Semaphore_post(SEMAPHORE_Calculo_Altura);
}

void Control(UArg arg0, UArg arg1){

#ifdef IMU_MPU6050
    tpLecturas_IMU Lecturas_IMU_Control;
#endif
#ifdef IMU_MPU9250
    tpLecturas_9DOF_IMU Lecturas_9DOF_IMU_Control;
#endif
    tpLecturas_Giroscopo Lecturas_Giroscopo_control;

    tpLecturas_Brujula Lecturas_Brujula_control;

    tpAHRS AHRS = {
        .DCM_matriz = {1, 0, 0, 0, 1, 0, 0, 0, 1},
        .DCM = {3, 3, (float32_t *)AHRS.DCM_matriz},
        .Kp_Roll_Pitch = Kp_ROLLPITCH,
        .Ki_Roll_Pitch = Ki_ROLLPITCH,
        .Kp_Yaw = Kp_YAW,
        .Ki_Yaw = Ki_YAW,
        .Periodo_Muestreo = PERIODO_Control / 1000.0
    };

    tpTelemetria_YPR Telemetria_YPR = {
            .Inicio = START_FRAME,
            .Final = FINAL_FRAME
        };

    tpTelemetria_Control Telemetria_Control = {
        .Inicio = START_FRAME,
        .Final = FINAL_FRAME
    };

    float32_t Ref_matriz[4] = {0, 0, 0, 0};
    arm_matrix_instance_f32 Ref = {4, 1, Ref_matriz};

    float32_t Accion_matriz[4] = {0, 0, 0, 0};
    arm_matrix_instance_f32 Accion = {4, 1, Accion_matriz};

    float32_t Variables_medidas_matriz[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    arm_matrix_instance_f32 Variables_medidas = {10, 1, Variables_medidas_matriz};

    float32_t Variables_predichas_matriz[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    arm_matrix_instance_f32 Variables_predichas = {10, 1, Variables_predichas_matriz};

    float32_t Variables_estimadas_matriz[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    arm_matrix_instance_f32 Variables_estimadas = {10, 1, Variables_estimadas_matriz};

    float32_t Perturbaciones_estimadas_matriz[4] = {0, 0, 0, 0};
    arm_matrix_instance_f32 Perturbaciones_estimadas = {4, 1,
    Perturbaciones_estimadas_matriz};

    float32_t Perturbaciones_calculadas_matriz[4] = {0, 0, 0, 0};
    arm_matrix_instance_f32 Perturbaciones_calculadas = {4, 1,
    Perturbaciones_calculadas_matriz};
```

```
    float32_t Aux_Matriz[100];
    arm_matrix_instance_f32 Aux = {10, 10, Aux_Matriz};

    float32_t Aux2_Matriz[100];
    arm_matrix_instance_f32 Aux2 = {10, 10, Aux2_Matriz};

    float32_t Var_Est_Aux[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

#ifdef Filtro_Perturbaciones

    float32_t Estado_filtro_Per_0[4*num_etapas_Filtro_Per];
    float32_t Estado_filtro_Per_1[4*num_etapas_Filtro_Per];
    float32_t Estado_filtro_Per_2[4*num_etapas_Filtro_Per];
    float32_t Estado_filtro_Per_3[4*num_etapas_Filtro_Per];

    arm_biquad_casd_df1_inst_f32 Filtro_Per_0 = {num_etapas_Filtro_Per, Estado_filtro_Per_0,
    (float32_t *)Coeficientes_Filtro_Pre_Valores};
    arm_biquad_casd_df1_inst_f32 Filtro_Per_1 = {num_etapas_Filtro_Per, Estado_filtro_Per_1,
    (float32_t *)Coeficientes_Filtro_Pre_Valores};
    arm_biquad_casd_df1_inst_f32 Filtro_Per_2 = {num_etapas_Filtro_Per, Estado_filtro_Per_2,
    (float32_t *)Coeficientes_Filtro_Pre_Valores};
    arm_biquad_casd_df1_inst_f32 Filtro_Per_3 = {num_etapas_Filtro_Per, Estado_filtro_Per_3,
    (float32_t *)Coeficientes_Filtro_Pre_Valores};

    //Inicializamos el filtro
    arm_fill_f32(0.0, Estado_filtro_Per_0, 4*num_etapas_Filtro_Per);
    arm_fill_f32(0.0, Estado_filtro_Per_1, 4*num_etapas_Filtro_Per);
    arm_fill_f32(0.0, Estado_filtro_Per_2, 4*num_etapas_Filtro_Per);
    arm_fill_f32(0.0, Estado_filtro_Per_3, 4*num_etapas_Filtro_Per);
#endif

    while(1){
        Semaphore_pend(SEMAPHORE_Control, BIOS_WAIT_FOREVER);

#ifdef IMU_MPU6050
        Leer_servidor_Lecturas_IMU(&Lecturas_IMU_Control);
#endif
#ifdef IMU_MPU9250
        Leer_servidor_Lecturas_IMU_9DOF(&Lecturas_9DOF_IMU_Control);
#endif
        Leer_servidor_Lecturas_Giroscopo(&Lecturas_Giroscopo_control);
        Leer_servidor_Lecturas_Brujula(&Lecturas_Brujula_control);
        Leer_servidor_Referencia(Ref_matriz, NULL);
        Ref_matriz[2] = Posicion_inicial + Ref_matriz[2]; //Giro no absoluto +- posicion
        inicial;

        Leer_servidor_DCM((float32_t*)AHRS.DCM_matriz);
        Leer_servidor_RPY(&AHRS.Roll, &AHRS.Pitch, &AHRS.Yaw);

        Leer_servidor_Perturbaciones_Estimadas(Perturbaciones_estimadas.pData);

//.....Sensado..Variables.....//
        //Ajuste posicion inicial//
        Variables_medidas.pData[1] = Normalizar_Grados(CONVERTIR_A_GRADOS(AHRS.Pitch));
        Variables_medidas.pData[3] = Normalizar_Grados(CONVERTIR_A_GRADOS(AHRS.Roll));
        Variables_medidas.pData[5] = Normalizar_Grados(CONVERTIR_A_GRADOS(AHRS.Yaw));

#ifdef IMU_MPU9250
        Variables_medidas.pData[0] = (float32_t)Lecturas_9DOF_IMU_Control.Valor.y_vel /
        IMU9250.Sensibilidad_Giroscopo;
        Variables_medidas.pData[2] = (float32_t)Lecturas_9DOF_IMU_Control.Valor.x_vel /
        IMU9250.Sensibilidad_Giroscopo;
        Variables_medidas.pData[4] = (float32_t)Lecturas_9DOF_IMU_Control.Valor.z_vel /
        IMU9250.Sensibilidad_Giroscopo;
#endif

#ifdef IMU_MPU6050
        Variables_medidas.pData[0] = (float32_t)Lecturas_IMU_Control.Valor.y_vel /
```

```
        IMU6050.Sensibilidad_Giroscopo;
        Variables_medidas.pData[2] = (float32_t)Lecturas_IMU_Control.Valor.x_vel /
        IMU6050.Sensibilidad_Giroscopo;
        Variables_medidas.pData[4] = (float32_t)Lecturas_IMU_Control.Valor.z_vel /
        IMU6050.Sensibilidad_Giroscopo;
#endif

#ifdef GYRO_L3G4200
        Variables_medidas.pData[0] = (float32_t)Lecturas_Giroscopo_control.Valor.y_vel /
        Giroscopo_L3G4200.Sensibilidad_Giroscopo;
        Variables_medidas.pData[2] = (float32_t)Lecturas_Giroscopo_control.Valor.x_vel /
        Giroscopo_L3G4200.Sensibilidad_Giroscopo;
        Variables_medidas.pData[4] = (float32_t)Lecturas_Giroscopo_control.Valor.z_vel /
        Giroscopo_L3G4200.Sensibilidad_Giroscopo;
#endif

        if(Modo_Control == ANGULOS_3){
            switch(ModoPerturbaciones){
                case NO_CORREGIR_PERTURBACIONES:
                case CORREGIR_PERTURBACIONES:
                case INTEGRAR_PERTURBACIONES_ESTIMADAS:
                    Variables_medidas.pData[9] = Variables_predichas.pData[9];
                break;
                case INTEGRAR_PERTURBACIONES:
                    Variables_medidas.pData[9] = Ref.pData[3];
            }
        }
        Escribir_servidor_Variables_Estado_Medidas(Variables_medidas.pData);

//Estimar variables estado

        Aux.numRows = Variables_medidas.numRows;
        Aux.numCols = Variables_predichas.numCols;
        arm_mat_sub_f32(&Variables_medidas, &Variables_predichas, &Aux);

        Aux2.numRows = 10;
        Aux2.numCols = 1;
        arm_mat_mult_f32(&Lo_per, &Aux, &Aux2);

        arm_mat_add_f32(&Variables_predichas, &Aux2, &Variables_estimadas);

#ifdef Estimador_Parcial //Optimizar multiplicando solo los estimados
        Variables_estimadas.pData[0] = Variables_medidas.pData[0];
        Variables_estimadas.pData[1] = Variables_medidas.pData[1];
        Variables_estimadas.pData[2] = Variables_medidas.pData[2];
        Variables_estimadas.pData[3] = Variables_medidas.pData[3];
        Variables_estimadas.pData[4] = Variables_medidas.pData[4];
        Variables_estimadas.pData[5] = Variables_medidas.pData[5];

#endif
        Escribir_servidor_Variables_Estado_Estimadas(Variables_estimadas.pData);

//Estimar perturbacion
        switch(ModoPerturbaciones){
            case NO_CORREGIR_PERTURBACIONES:
            case CORREGIR_PERTURBACIONES:
/*
                Aux.numRows = Variables_medidas.numRows;
                Aux.numCols = Variables_predichas.numCols;
                arm_mat_sub_f32(&Variables_medidas, &Variables_predichas, &Aux);
*/
                Aux2.numRows = Aux2.numRows;
                Aux2.numCols = Aux2.numCols;
                arm_mat_mult_f32(&Lp, &Aux, &Aux2);

                Aux.numRows = Aux.numRows;
                Aux.numCols = Aux.numCols;
                arm_mat_add_f32(&Perturbaciones_estimadas, &Aux2, &Aux);
```

```
                //A 0 la estimacion de Empuje
//              Aux.pData[3] = 0;

#ifndef Filtro_Perturbaciones
                arm_copy_f32(Aux.pData, Perturbaciones_estimadas.pData,
                sizeof(Perturbaciones_estimadas_matriz)/sizeof(float32_t));
#else
                //      Filtar_Perturbacion
                arm_biquad_cascade_df1_f32(&Filtro_Per_0, &Aux.pData[0],
                &Perturbaciones_estimadas.pData[0], 1);
                arm_biquad_cascade_df1_f32(&Filtro_Per_1, &Aux.pData[1],
                &Perturbaciones_estimadas.pData[1], 1);
                arm_biquad_cascade_df1_f32(&Filtro_Per_2, &Aux.pData[2],
                &Perturbaciones_estimadas.pData[2], 1);
                arm_biquad_cascade_df1_f32(&Filtro_Per_3, &Aux.pData[3],
                &Perturbaciones_estimadas.pData[3], 1);
#endif

            break;
            case INTEGRAR_PERTURBACIONES:

                Perturbaciones_estimadas.pData[0] += Ki * (Ref.pData[0] -
                Variables_medidas.pData[1]);
                Perturbaciones_estimadas.pData[1] += Ki * (Ref.pData[1] -
                Variables_medidas.pData[3]);
                Perturbaciones_estimadas.pData[2] += Ki * (Ref.pData[2] -
                Variables_medidas.pData[5]);
                Perturbaciones_estimadas.pData[3] += Ki * (Ref.pData[3] -
                Variables_medidas.pData[9]);
            break;

            case INTEGRAR_PERTURBACIONES_ESTIMADAS:

                Perturbaciones_estimadas.pData[0] += Ki_EST * (Variables_predichas.pData[1]
                - Variables_medidas.pData[1]);
                Perturbaciones_estimadas.pData[1] += Ki_EST * (Variables_predichas.pData[3]
                - Variables_medidas.pData[3]);
                Perturbaciones_estimadas.pData[2] += Ki_EST * (Variables_predichas.pData[5]
                - Variables_medidas.pData[5]);
                Perturbaciones_estimadas.pData[3] += Ki_EST * (Variables_predichas.pData[9]
                - Variables_medidas.pData[9]);
            break;
        }

                //Prealimentar perturbaciones conocidas
                Perturbaciones_calculadas.pData[0] = Perturbaciones_estimadas.pData[0];// +
                Pert_Fuerza_Bateria*sin(Variables_estimadas.pData[1]*PI/180.0);
                Perturbaciones_calculadas.pData[1] = Perturbaciones_estimadas.pData[1];// +
                Pert_Fuerza_Bateria*sin(Variables_estimadas.pData[3]*PI/180.0);
                Perturbaciones_calculadas.pData[2] = Perturbaciones_estimadas.pData[2];
                Perturbaciones_calculadas.pData[3] = Perturbaciones_estimadas.pData[3];

         Escribir_servidor_Perturbaciones_Estimadas(Perturbaciones_estimadas.pData);

//Accion
        switch(ModoPerturbaciones){
            case NO_CORREGIR_PERTURBACIONES:
                Aux.numRows = K_pre_4.numRows;
                Aux.numCols = Ref.numCols;

                Aux2.numRows = K_4.numRows;
                Aux2.numCols = Variables_estimadas.numCols;

                switch(Modo_Control){
                    case ANGULOS_4:
                        arm_mat_mult_f32(&K_pre_4, &Ref, &Aux);
                        arm_mat_mult_f32(&K_4, &Variables_estimadas, &Aux2);
```

```
                    arm_mat_sub_f32(&Aux, &Aux2, &Accion);
                break;

                case ANGULOS_3:
                    arm_mat_mult_f32(&K_pre_3, &Ref, &Aux);
                    arm_mat_mult_f32(&K_3, &Variables_estimadas, &Aux2);
                    arm_mat_sub_f32(&Aux, &Aux2, &Accion);
                break;

                case EMPUJE:
                    Accion_matriz[0] =  Ref.pData[3]; //
                    Accion_matriz[1] =  Ref.pData[3]; //
                    Accion_matriz[2] =  Ref.pData[3]; //
                    Accion_matriz[3] =  Ref.pData[3]; //
                break;
            }
        break;

        case CORREGIR_PERTURBACIONES:
        case INTEGRAR_PERTURBACIONES:
        case INTEGRAR_PERTURBACIONES_ESTIMADAS:

            Aux.numRows = K_4.numRows;
            Aux.numCols = Variables_estimadas.numCols;

            Aux2.numRows = Accion.numRows;
            Aux2.numCols = Aux.numCols;

            switch(Modo_Control){
                case ANGULOS_4:
                    arm_mat_mult_f32(&K_pre_4, &Ref, &Accion);
                    arm_mat_mult_f32(&K_4, &Variables_estimadas, &Aux);

                    arm_mat_sub_f32(&Accion, &Aux, &Aux2);

                    Aux.numRows = La.numRows;
                    Aux.numCols = Perturbaciones_calculadas.numCols;
                    arm_mat_mult_f32(&La, &Perturbaciones_calculadas, &Aux);

                    arm_mat_sub_f32(&Aux2, &Aux, &Accion);
                break;

                case ANGULOS_3:
                    arm_mat_mult_f32(&K_pre_3, &Ref, &Accion);
                    arm_mat_mult_f32(&K_3, &Variables_estimadas, &Aux);

                    arm_mat_sub_f32(&Accion, &Aux, &Aux2);

                    Aux.numRows = La.numRows;
                    Aux.numCols = Perturbaciones_calculadas.numCols;
                    arm_mat_mult_f32(&La, &Perturbaciones_calculadas, &Aux);

                    arm_mat_sub_f32(&Aux2, &Aux, &Accion);
                break;
                case EMPUJE:
                    Accion_matriz[0] =  Ref.pData[3]; //
                    Accion_matriz[1] =  Ref.pData[3]; //
                    Accion_matriz[2] =  Ref.pData[3]; //
                    Accion_matriz[3] =  Ref.pData[3]; //
                break;
            }
        break;

    }

    if(Accion.pData[0] < Accion_Minima )
        Accion.pData[0] = Accion_Minima;
    if(Accion.pData[1] < Accion_Minima )
```

```
            Accion.pData[1] = Accion_Minima;
        if(Accion.pData[2] < Accion_Minima )
            Accion.pData[2] = Accion_Minima;
        if(Accion.pData[3] < Accion_Minima )
            Accion.pData[3] = Accion_Minima;

        if(Accion.pData[0] > Accion_Maxima )
            Accion.pData[0] = Accion_Maxima;
        if(Accion.pData[1] > Accion_Maxima )
            Accion.pData[1] = Accion_Maxima;
        if(Accion.pData[2] > Accion_Maxima )
            Accion.pData[2] = Accion_Maxima;
        if(Accion.pData[3] > Accion_Maxima )
            Accion.pData[3] = Accion_Maxima;

//Aplicar U
        PWM_setDuty(PWM0, (uint32_t)Accion.pData[0] + Pulso_minimo_PWM_motor);
        PWM_setDuty(PWM1, (uint32_t)Accion.pData[1] + Pulso_minimo_PWM_motor);
        PWM_setDuty(PWM2, (uint32_t)Accion.pData[2] + Pulso_minimo_PWM_motor);
        PWM_setDuty(PWM3, (uint32_t)Accion.pData[3] + Pulso_minimo_PWM_motor);

//Predecir estado
        arm_mat_mult_f32(&F, &Variables_estimadas, &Variables_predichas);

        Aux.numRows = G.numRows;
        Aux.numCols = Accion.numCols;
        arm_mat_mult_f32(&G, &Accion, &Aux);

        Aux2.numRows = Variables_predichas.numRows;
        Aux2.numCols = Aux.numCols;
        arm_mat_add_f32(&Variables_predichas, &Aux, &Aux2);

        Aux.numRows = Gp.numRows;
        Aux.numCols = Perturbaciones_calculadas.numCols;
        arm_mat_mult_f32(&Gp, &Perturbaciones_calculadas, &Aux);

        arm_mat_add_f32(&Aux2, &Aux, &Variables_predichas);


//Telemetria
        switch(ModoTelemetria){
            case TELEMETRIA_YPR:
                Telemetria_YPR.Yaw = (int16_t)(AHRS.Yaw * 10.0);
                Telemetria_YPR.Pitch = (int16_t)(AHRS.Pitch * 10.0);
                Telemetria_YPR.Roll = (int16_t)(AHRS.Roll * 10.0);

                UART_write(UART_BT_TELEMETRIA, &Telemetria_YPR, sizeof(Telemetria_YPR)-1);
            break;
            case TELEMETRIA_CONTROL:
                Telemetria_Control.InfoTelemetria = InfoTelemetria;
                //Referencia
                Var_Est_Aux[0] = Ref_matriz[0] / Angulo_Max_Q16;
                Var_Est_Aux[1] = Ref_matriz[1] / Angulo_Max_Q16;
                Var_Est_Aux[2] = Ref_matriz[2] / Angulo_Max_Q16;
                Var_Est_Aux[3] = Ref_matriz[3] / 1000.0;
                arm_float_to_q15( Var_Est_Aux, (q15_t *)Telemetria_Control.Referencia, 4);

                //Accion
                Telemetria_Control.Accion[0] = Accion_matriz[0];
                Telemetria_Control.Accion[1] = Accion_matriz[1];
                Telemetria_Control.Accion[2] = Accion_matriz[2];
                Telemetria_Control.Accion[3] = Accion_matriz[3];

                //Var_Est
                Var_Est_Aux[0] = Variables_estimadas.pData[0] / Velocidad_Max_Q16;
                Var_Est_Aux[1] = Variables_estimadas.pData[1] / Angulo_Max_Q16;
                Var_Est_Aux[2] = Variables_estimadas.pData[2] / Velocidad_Max_Q16;
                Var_Est_Aux[3] = Variables_estimadas.pData[3] / Angulo_Max_Q16;
```

```
                Var_Est_Aux[4] = Variables_estimadas.pData[4] / Velocidad_Max_Q16;
                Var_Est_Aux[5] = Variables_estimadas.pData[5] / Angulo_Max_Q16;
                Var_Est_Aux[6] = Variables_estimadas.pData[6] / F_Max_Q16;
                Var_Est_Aux[7] = Variables_estimadas.pData[7] / F_Max_Q16;
                Var_Est_Aux[8] = Variables_estimadas.pData[8] / F_Max_Q16;
                Var_Est_Aux[9] = Variables_estimadas.pData[9] / F_Max_Q16;
                arm_float_to_q15( Var_Est_Aux, (q15_t *)Telemetria_Control.Variables_Estado,
                10);

                //Perturbaciones
                Var_Est_Aux[0] = Perturbaciones_calculadas_matriz[0] / F_Max_Q16;
                Var_Est_Aux[1] = Perturbaciones_calculadas_matriz[1] / F_Max_Q16;
                Var_Est_Aux[2] = Perturbaciones_calculadas_matriz[2] / F_Max_Q16;
                Var_Est_Aux[3] = Perturbaciones_calculadas_matriz[3] / F_Max_Q16;
                arm_float_to_q15( Var_Est_Aux, (q15_t *)Telemetria_Control.Perturbaciones, 4);

                //Altura
                Telemetria_Control.Altura_Barometrica = Altura_Presion_mm;
                Telemetria_Control.Altura_US = Altura_US_mm;

#ifdef IMU_MPU9250
                Mailbox_pend(Buzon_Lecturas_IMU, &Lecturas_9DOF_IMU_Control, BIOS_NO_WAIT);
                //Lectura ACCEL
                Telemetria_Control.Acel[0] = Lecturas_9DOF_IMU_Control.Valor.x_acel;
                Telemetria_Control.Acel[1] = Lecturas_9DOF_IMU_Control.Valor.y_acel;
                Telemetria_Control.Acel[2] = Lecturas_9DOF_IMU_Control.Valor.z_acel;

                //Lectura GYRO
                Telemetria_Control.Gyro[0] = Lecturas_9DOF_IMU_Control.Valor.x_vel;
                Telemetria_Control.Gyro[1] = Lecturas_9DOF_IMU_Control.Valor.y_vel;
                Telemetria_Control.Gyro[2] = Lecturas_9DOF_IMU_Control.Valor.z_vel;

                //Lectura COMPASS
                Telemetria_Control.Magnetics[0] = Lecturas_9DOF_IMU_Control.Valor.x_mag;
                Telemetria_Control.Magnetics[1] = Lecturas_9DOF_IMU_Control.Valor.y_mag;
                Telemetria_Control.Magnetics[2] = Lecturas_9DOF_IMU_Control.Valor.z_mag;
#endif

#ifdef IMU_MPU6050
                Mailbox_pend(Buzon_Lecturas_IMU, &Lecturas_IMU_Control, BIOS_NO_WAIT);

                //Lectura ACCEL
                Telemetria_Control.Acel[0] = Lecturas_IMU_Control.Valor.x_acel;
                Telemetria_Control.Acel[1] = Lecturas_IMU_Control.Valor.y_acel;
                Telemetria_Control.Acel[2] = Lecturas_IMU_Control.Valor.z_acel;

                //Lectura GYRO
                Telemetria_Control.Gyro[0] = Lecturas_IMU_Control.Valor.x_vel;
                Telemetria_Control.Gyro[1] = Lecturas_IMU_Control.Valor.y_vel;
                Telemetria_Control.Gyro[2] = Lecturas_IMU_Control.Valor.z_vel;

#ifdef GYRO_L3G4200
                Telemetria_Control.Gyro[0] = Lecturas_Giroscopo_control.Valor.x_vel;
                Telemetria_Control.Gyro[1] = Lecturas_Giroscopo_control.Valor.y_vel;
                Telemetria_Control.Gyro[2] = Lecturas_Giroscopo_control.Valor.z_vel;
#endif
                //Lectura Brujula
                Telemetria_Control.Magnetics[0] = Lecturas_Brujula_control.Valor.Magnetismo_x;
                Telemetria_Control.Magnetics[1] = Lecturas_Brujula_control.Valor.Magnetismo_y;
                Telemetria_Control.Magnetics[2] = Lecturas_Brujula_control.Valor.Magnetismo_z;
#endif
                UART_write(UART_BT_TELEMETRIA, &Telemetria_Control,
                sizeof(Telemetria_Control)-1);
            break;
        }

    }
}
```

```
void CLK_Control(){
    Semaphore_post(SEMAPHORE_Control);
}
```

```c
/*
 *  ======== QUAD_BOARD.c ========
 *  This file is responsible for setting up the board specific items for the
 *  QUAD_BOARD board.
 */

#include <stdint.h>
#include <stdbool.h>
#include <inc/hw_memmap.h>
#include <inc/hw_types.h>
#include <inc/hw_ints.h>
#include <inc/hw_gpio.h>

#include <driverlib/gpio.h>
#include <driverlib/sysctl.h>
#include <driverlib/i2c.h>
#include <driverlib/ssi.h>
#include <driverlib/uart.h>
#include <driverlib/udma.h>
#include <driverlib/pin_map.h>

#include <xdc/std.h>
#include <xdc/cfg/global.h>
#include <xdc/runtime/Error.h>
#include <xdc/runtime/System.h>
#include <ti/sysbios/family/arm/m3/Hwi.h>

#include "QUAD_board.h"


#ifndef TI_DRIVERS_UART_DMA
#define TI_DRIVERS_UART_DMA 0
#endif

/*
 *  ============================== DMA ===============================
 */
#if defined(__TI_COMPILER_VERSION__)
#pragma DATA_ALIGN(dmaControlTable, 1024)
#elif defined(__IAR_SYSTEMS_ICC__)
#pragma data_alignment=1024
#elif defined(__GNUC__)
__attribute__ ((aligned (1024)))
#endif
static tDMAControlTable dmaControlTable[32];
static bool dmaInitialized = false;

/* Hwi_Struct used in the initDMA Hwi_construct call */
static Hwi_Struct hwiStruct;

/*
 *  ======== dmaErrorHwi ========
 */
static Void dmaErrorHwi(UArg arg)
{
    System_printf("DMA error code: %d\n", uDMAErrorStatusGet());
    uDMAErrorStatusClear();
    System_abort("DMA error!!");
}

/*
 *  ======== QUAD_BOARD_initDMA ========
 */
void QUAD_BOARD_initDMA(void)
{
    Error_Block eb;
    Hwi_Params  hwiParams;
```

```
    if (!dmaInitialized) {

        Error_init(&eb);

        Hwi_Params_init(&hwiParams);
        Hwi_construct(&(hwiStruct), INT_UDMAERR, dmaErrorHwi,
                      &hwiParams, &eb);
        if (Error_check(&eb)) {
            System_abort("Couldn't create DMA error hwi");
        }

        SysCtlPeripheralEnable(SYSCTL_PERIPH_UDMA);
        uDMAEnable();
        uDMAControlBaseSet(dmaControlTable);

        dmaInitialized = true;
    }
}

/*
 *  ============================== General ==============================
 */
/*
 *  ======== QUAD_BOARD_initGeneral ========
 */
void QUAD_BOARD_initGeneral(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
}

/*
 *  ============================== GPIO ==============================
 */
/* Place into subsections to allow the TI linker to remove items properly */
#if defined(__TI_COMPILER_VERSION__)
#pragma DATA_SECTION(GPIOTiva_config, ".const:GPIOTiva_config")
#endif

#include <ti/drivers/GPIO.h>
#include <ti/drivers/gpio/GPIOTiva.h>

/*
 * Array of Pin configurations
 * NOTE: The order of the pin configurations must coincide with what was
 *       defined in QUAD_BOARD.h
 * NOTE: Pins not used for interrupts should be placed at the end of the
 *       array.  Callback entries can be omitted from callbacks array to
 *       reduce memory usage.
 */
GPIO_PinConfig gpioPinConfigs[] = {
    /* Input pins */
    /* QUAD_BOARD_GPIO_SW1 */
    GPIOTiva_PF_4 | GPIO_CFG_IN_PU | GPIO_CFG_IN_INT_RISING,
    /* QUAD_BOARD_GPIO_SW2 */
    GPIOTiva_PF_0 | GPIO_CFG_IN_PU | GPIO_CFG_IN_INT_RISING,
    /* QUAD_BOARD_ECHO */
    GPIOTiva_PD_6 | GPIO_CFG_INPUT | GPIO_CFG_IN_INT_BOTH_EDGES,
    /* QUAD_BOARD_RPM */
    GPIOTiva_PA_5 | GPIO_CFG_INPUT | GPIO_CFG_IN_INT_RISING,

    /* Output pins */
    /* QUAD_BOARD_TRIGG */
    GPIOTiva_PD_7 | GPIO_CFG_OUT_STD | GPIO_CFG_OUT_STR_LOW  | GPIO_CFG_OUT_LOW,
```

```
    /* QUAD_BOARD_LED_RED */
    GPIOTiva_PF_1 | GPIO_CFG_OUT_STD | GPIO_CFG_OUT_STR_HIGH | GPIO_CFG_OUT_LOW,
    /* QUAD_BOARD_LED_BLUE */
    GPIOTiva_PF_2 | GPIO_CFG_OUT_STD | GPIO_CFG_OUT_STR_HIGH | GPIO_CFG_OUT_LOW,
    /* QUAD_BOARD_LED_GREEN */
    GPIOTiva_PF_3 | GPIO_CFG_OUT_STD | GPIO_CFG_OUT_STR_HIGH | GPIO_CFG_OUT_LOW,
    /* QUAD_BOARD_POWER_33 */
    GPIOTiva_PD_0 | GPIO_CFG_OUT_STD | GPIO_CFG_OUT_STR_HIGH | GPIO_CFG_OUT_HIGH,
    /*QUAD_BOARD_SPI_CE */
    GPIOTiva_PA_6 | GPIO_CFG_OUT_STD | GPIO_CFG_OUT_STR_HIGH | GPIO_CFG_OUT_LOW,
    /*QUAD_BOARD_SPI_CSN */
    GPIOTiva_PA_3 | GPIO_CFG_OUT_STD | GPIO_CFG_OUT_STR_HIGH | GPIO_CFG_OUT_HIGH,

};

/*
 * Array of callback function pointers
 * NOTE: The order of the pin configurations must coincide with what was
 *       defined in QUAD_BOARD.h
 * NOTE: Pins not used for interrupts can be omitted from callbacks array to
 *       reduce memory usage (if placed at end of gpioPinConfigs array).
 */
GPIO_CallbackFxn gpioCallbackFunctions[] = {
    NULL,  /* QUAD_BOARD_GPIO_SW1 */
    NULL,  /* QUAD_BOARD_GPIO_SW2 */
    NULL,  /* QUAD_BOARD_ECHO */
    NULL,  /* QUAD_BOARD_RPM */
};

/* The device-specific GPIO_config structure */
const GPIOTiva_Config GPIOTiva_config = {
    .pinConfigs = (GPIO_PinConfig *) gpioPinConfigs,
    .callbacks = (GPIO_CallbackFxn *) gpioCallbackFunctions,
    .numberOfPinConfigs = sizeof(gpioPinConfigs) / sizeof(GPIO_PinConfig),
    .numberOfCallbacks = sizeof(gpioCallbackFunctions)/sizeof(GPIO_CallbackFxn),
    .intPriority = (~0)
};

/*
 *  ======== QUAD_BOARD_initGPIO ========
 */
void QUAD_BOARD_initGPIO(void)
{
    /* QUAD_BOARD_GPIO_SW2 - PF0 requires unlocking before configuration */
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= GPIO_PIN_0;
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_0);

    HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTD_BASE + GPIO_O_CR) |= GPIO_PIN_6;
    GPIOPinTypeGPIOInput(GPIO_PORTD_BASE, GPIO_PIN_6);

    HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTD_BASE + GPIO_O_CR) |= GPIO_PIN_7;
    GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, GPIO_PIN_7);

    /* Initialize peripheral and pins */
    GPIO_init();
}

/*
 *  ============================== I2C ===============================
 */
/* Place into subsections to allow the TI linker to remove items properly */
#if defined(__TI_COMPILER_VERSION__)
#pragma DATA_SECTION(I2C_config, ".const:I2C_config")
#pragma DATA_SECTION(i2cTivaHWAttrs, ".const:i2cTivaHWAttrs")
#endif
```

```
#include <ti/drivers/I2C.h>
#include <ti/drivers/i2c/I2CTiva.h>

/* I2C objects */
I2CTiva_Object i2cTivaObjects[QUAD_BOARD_I2CCOUNT];

/* I2C configuration structure, describing which pins are to be used */
const I2CTiva_HWAttrs i2cTivaHWAttrs[QUAD_BOARD_I2CCOUNT] = {
    {I2C0_BASE, INT_I2C0, ~0 /* Interrupt priority */},
    {I2C2_BASE, INT_I2C2, ~0 /* Interrupt priority */}
};

const I2C_Config I2C_config[] = {
    {&I2CTiva_fxnTable, &i2cTivaObjects[0], &i2cTivaHWAttrs[0]},
    {&I2CTiva_fxnTable, &i2cTivaObjects[1], &i2cTivaHWAttrs[1]},
    {NULL, NULL, NULL}
};

/*
 *  ======== QUAD_BOARD_initI2C ========
 */
void QUAD_BOARD_initI2C(void)
{
    /* I2C1 Init */
    /* Enable the peripheral */
    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C2);

    /* Configure the appropriate pins to be I2C instead of GPIO. */
    GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
    GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);
    GPIOPinConfigure(GPIO_PB3_I2C0SDA);
    GPIOPinConfigure(GPIO_PB2_I2C0SCL);


    /* Configure the appropriate pins to be I2C instead of GPIO. */
    GPIOPinTypeI2CSCL(GPIO_PORTE_BASE, GPIO_PIN_4);
    GPIOPinTypeI2C(GPIO_PORTE_BASE, GPIO_PIN_5);
    GPIOPinConfigure(GPIO_PE4_I2C2SCL);
    GPIOPinConfigure(GPIO_PE5_I2C2SDA);

    I2C_init();
}

/*
 *  ============================== PWM ===============================
 */
/* Place into subsections to allow the TI linker to remove items properly */
#if defined(__TI_COMPILER_VERSION__)
#pragma DATA_SECTION(PWM_config, ".const:PWM_config")
#pragma DATA_SECTION(pwmTivaHWAttrs, ".const:pwmTivaHWAttrs")
#endif

#include <ti/drivers/PWM.h>
#include <ti/drivers/pwm/PWMTiva.h>
#include <driverlib/pwm.h>

PWMTiva_Object pwmTivaObjects[QUAD_BOARD_PWMCOUNT];

/* PWM configuration structure */
const PWMTiva_HWAttrs pwmTivaHWAttrs[QUAD_BOARD_PWMCOUNT] = {
        {
            PWM0_BASE,
            PWM_OUT_0,
            PWM_GEN_MODE_DOWN | PWM_GEN_MODE_DBG_RUN
        },
        {
```

```
            PWM0_BASE,
            PWM_OUT_1,
            PWM_GEN_MODE_DOWN | PWM_GEN_MODE_DBG_RUN
        },
        {
            PWM0_BASE,
            PWM_OUT_2,
            PWM_GEN_MODE_DOWN | PWM_GEN_MODE_DBG_RUN
        },
        {
            PWM0_BASE,
            PWM_OUT_3,
            PWM_GEN_MODE_DOWN | PWM_GEN_MODE_DBG_RUN
        }
};

const PWM_Config PWM_config[] = {
    {&PWMTiva_fxnTable, &pwmTivaObjects[0], &pwmTivaHWAttrs[0]},
    {&PWMTiva_fxnTable, &pwmTivaObjects[1], &pwmTivaHWAttrs[1]},
    {&PWMTiva_fxnTable, &pwmTivaObjects[2], &pwmTivaHWAttrs[2]},
    {&PWMTiva_fxnTable, &pwmTivaObjects[3], &pwmTivaHWAttrs[3]},
    {NULL, NULL, NULL}
};

/*
 *  ======== QUAD_BOARD_initPWM ========
 */
void QUAD_BOARD_initPWM(void)
{
    /* Enable PWM peripherals */
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);

    /*
     * Enable PWM output on GPIO pins.  Board_LED1 and Board_LED2 are now
     * controlled by PWM peripheral - Do not use GPIO APIs.
     */
    GPIOPinConfigure(GPIO_PB4_M0PWM2);
    GPIOPinConfigure(GPIO_PB5_M0PWM3);
    GPIOPinConfigure(GPIO_PB6_M0PWM0);
    GPIOPinConfigure(GPIO_PB7_M0PWM1);
    GPIOPinTypePWM(GPIO_PORTB_BASE, GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7);

    PWM_init();
}

/*
 *  ============================== UART ==============================
 */
/* Place into subsections to allow the TI linker to remove items properly */
#if defined(__TI_COMPILER_VERSION__)
#pragma DATA_SECTION(UART_config, ".const:UART_config")
#pragma DATA_SECTION(uartTivaHWAttrs, ".const:uartTivaHWAttrs")
#endif

#include <ti/drivers/UART.h>
#if TI_DRIVERS_UART_DMA
#include <ti/drivers/uart/UARTTivaDMA.h>

/* UART objects */
UARTTivaDMA_Object uartTivaObjects[QUAD_BOARD_UARTCOUNT];

/* UART configuration structure */
const UARTTivaDMA_HWAttrs uartTivaHWAttrs[QUAD_BOARD_UARTCOUNT] = {
        {/* QUAD_BOARD_UART0 */
            UART0_BASE,
            INT_UART0,
            ~0,         /* Interrupt priority */
            UDMA_CH8_UART0RX,
```

```
            UDMA_CH9_UART0TX,
        },
        {/* QUAD_BOARD_UART1 */
            UART1_BASE,
            INT_UART1,
            ~0,         /* Interrupt priority */
            UDMA_CH22_UART1RX,
            UDMA_CH23_UART1TX,
        },
        {/* QUAD_BOARD_UART4 */
            UART5_BASE,
            INT_UART5,
            ~0,         /* Interrupt priority */
            UDMA_CH6_UART5RX,
            UDMA_CH7_UART5TX,
        },
        {/* QUAD_BOARD_UART7 */
            UART7_BASE,
            INT_UART7,
            ~0,         /* Interrupt priority */
            UDMA_CH20_UART7RX,
            UDMA_CH21_UART7TX,
        }
};

const UART_Config UART_config[] = {
        {
            &UARTTivaDMA_fxnTable,
            &uartTivaObjects[0],
            &uartTivaHWAttrs[0]
        },
        {
            &UARTTivaDMA_fxnTable,
            &uartTivaObjects[1],
            &uartTivaHWAttrs[1]
        },
        {
            &UARTTivaDMA_fxnTable,
            &uartTivaObjects[2],
            &uartTivaHWAttrs[2]
        },
        {
            &UARTTivaDMA_fxnTable,
            &uartTivaObjects[3],
            &uartTivaHWAttrs[3]
        },
    {NULL, NULL, NULL}
};

#else
#include <ti/drivers/uart/UARTTiva.h>

/* UART objects */
UARTTiva_Object uartTivaObjects[QUAD_BOARD_UARTCOUNT];
unsigned char uartTivaRingBuffer[32];
unsigned char uartTivaRingBuffer1[32];
unsigned char uartTivaRingBuffer5[32];
unsigned char uartTivaRingBuffer7[32];

/* UART configuration structure */
const UARTTiva_HWAttrs uartTivaHWAttrs[QUAD_BOARD_UARTCOUNT] = {
    {/* QUAD_BOARD_UART0 */
        .baseAddr = UART0_BASE,
        .intNum = INT_UART0,
        .intPriority = ~0,
        .flowControl = UART_FLOWCONTROL_NONE,
        .ringBufPtr  = uartTivaRingBuffer,
        .ringBufSize = sizeof(uartTivaRingBuffer)
```

```
    },
    {/* QUAD_BOARD_UART1 */
        .baseAddr = UART1_BASE,
        .intNum = INT_UART1,
        .intPriority = ~0,
        .flowControl = UART_FLOWCONTROL_NONE,
        .ringBufPtr  = uartTivaRingBuffer1,
        .ringBufSize = sizeof(uartTivaRingBuffer1)
    },
    {/* QUAD_BOARD_UART4 */
        .baseAddr = UART5_BASE,
        .intNum = INT_UART5,
        .intPriority = ~0,
        .flowControl = UART_FLOWCONTROL_NONE,
        .ringBufPtr  = uartTivaRingBuffer5,
        .ringBufSize = sizeof(uartTivaRingBuffer5)
    },
    {/* QUAD_BOARD_UART7 */
        .baseAddr = UART7_BASE,
        .intNum = INT_UART7,
        .intPriority = ~0,
        .flowControl = UART_FLOWCONTROL_NONE,
        .ringBufPtr  = uartTivaRingBuffer7,
        .ringBufSize = sizeof(uartTivaRingBuffer7)
    }
};

const UART_Config UART_config[] = {
        {
            &UARTTiva_fxnTable,
            &uartTivaObjects[0],
            &uartTivaHWAttrs[0]
        },
        {
            &UARTTiva_fxnTable,
            &uartTivaObjects[1],
            &uartTivaHWAttrs[1]
        },
        {
            &UARTTiva_fxnTable,
            &uartTivaObjects[2],
            &uartTivaHWAttrs[2]
        },
        {
            &UARTTiva_fxnTable,
            &uartTivaObjects[3],
            &uartTivaHWAttrs[3]
        },
    {NULL, NULL, NULL}
};
#endif /* TI_DRIVERS_UART_DMA */

/*
 *  ======== QUAD_BOARD_initUART ========
 */
void QUAD_BOARD_initUART(void)
{
    /* Enable and configure the peripherals used by the uart. */
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART1);
    GPIOPinConfigure(GPIO_PB0_U1RX);
    GPIOPinConfigure(GPIO_PB1_U1TX);
    GPIOPinTypeUART(GPIO_PORTB_BASE, GPIO_PIN_0 | GPIO_PIN_1);
/*
```

```
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART5);
    GPIOPinConfigure(GPIO_PE4_U5RX);
    GPIOPinConfigure(GPIO_PE5_U5TX);
    GPIOPinTypeUART(GPIO_PORTE_BASE, GPIO_PIN_4 | GPIO_PIN_5);
*/
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART7);
    GPIOPinConfigure(GPIO_PE1_U7TX);
    GPIOPinConfigure(GPIO_PE0_U7RX);
    GPIOPinTypeUART(GPIO_PORTE_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    /* Initialize the UART driver */

#if TI_DRIVERS_UART_DMA
    QUAD_BOARD_initDMA();
#endif
    UART_init();
}

/*
 *  ============================== Watchdog ==============================
 */
/* Place into subsections to allow the TI linker to remove items properly */
#if defined(__TI_COMPILER_VERSION__)
#pragma DATA_SECTION(Watchdog_config, ".const:Watchdog_config")
#pragma DATA_SECTION(watchdogTivaHWAttrs, ".const:watchdogTivaHWAttrs")
#endif

#include <ti/drivers/Watchdog.h>
#include <ti/drivers/watchdog/WatchdogTiva.h>

/* Watchdog objects */
WatchdogTiva_Object watchdogTivaObjects[QUAD_BOARD_WATCHDOGCOUNT];

/* Watchdog configuration structure */
const WatchdogTiva_HWAttrs watchdogTivaHWAttrs[QUAD_BOARD_WATCHDOGCOUNT] = {
    /* QUAD_BOARD_WATCHDOG0 with 1 sec period at default CPU clock freq */
    {WATCHDOG0_BASE, INT_WATCHDOG, ~0 /* Interrupt priority */, 80000000},
};

const Watchdog_Config Watchdog_config[] = {
    {&WatchdogTiva_fxnTable, &watchdogTivaObjects[0], &watchdogTivaHWAttrs[0]},
    {NULL, NULL, NULL},
};

/*
 *  ======== QUAD_BOARD_initWatchdog ========
 *
 * NOTE: To use the other watchdog timer with base address WATCHDOG1_BASE,
 *       an additional function call may need be made to enable PIOSC. Enabling
 *       WDOG1 does not do this. Enabling another peripheral that uses PIOSC
 *       such as ADC0 or SSI0, however, will do so. Example:
 *
 *       SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
 *       SysCtlPeripheralEnable(SYSCTL_PERIPH_WDOG1);
 *
 *       See the following forum post for more information:
 *
 http://e2e.ti.com/support/microcontrollers/stellaris_arm_cortex-m3_microcontroller/f/471/p/176487/654390.aspx#654390
 */
void QUAD_BOARD_initWatchdog(void)
{
    /* Enable peripherals used by Watchdog */
    SysCtlPeripheralEnable(SYSCTL_PERIPH_WDOG0);

    /* Initialize the Watchdog driver */
    Watchdog_init();
}
```

```
/*
 *  ============================== SPI ===============================
 */
/* Place into subsections to allow the TI linker to remove items properly */
#if defined(__TI_COMPILER_VERSION__)
#pragma DATA_SECTION(SPI_config, ".const:SPI_config")
#pragma DATA_SECTION(spiTivaDMAHWAttrs, ".const:spiTivaDMAHWAttrs")
#endif

#include <ti/drivers/SPI.h>
#include <ti/drivers/spi/SPITivaDMA.h>

/* SPI objects */
SPITivaDMA_Object spiTivaDMAObjects[QUAD_BOARD_SPICOUNT];
#if defined(__TI_COMPILER_VERSION__)
#pragma DATA_ALIGN(spiTivaDMAscratchBuf, 32)
#elif defined(__IAR_SYSTEMS_ICC__)
#pragma data_alignment=32
#elif defined(__GNUC__)
__attribute__ ((aligned (32)))
#endif
uint32_t spiTivaDMAscratchBuf[QUAD_BOARD_SPICOUNT];

/* SPI configuration structure */
const SPITivaDMA_HWAttrs spiTivaDMAHWAttrs[QUAD_BOARD_SPICOUNT] = {
    {
        SSI0_BASE,
        INT_SSI0,
        ~0,         /* Interrupt priority */
        &spiTivaDMAscratchBuf[0],
        0,
        UDMA_CHANNEL_SSI0RX,
        UDMA_CHANNEL_SSI0TX,
        uDMAChannelAssign,
        UDMA_CH10_SSI0RX,
        UDMA_CH11_SSI0TX
    }
};

const SPI_Config SPI_config[] = {
    {&SPITivaDMA_fxnTable, &spiTivaDMAObjects[0], &spiTivaDMAHWAttrs[0]},
    {NULL, NULL, NULL},
};

/*
 *  ======== QUAD_BOARD_initSPI ========
 */
void QUAD_BOARD_initSPI(void)
{
    /* SPI0 */
    SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI0);

    /* Need to unlock PF0 */
    GPIOPinConfigure(GPIO_PA2_SSI0CLK);
//    GPIOPinConfigure(GPIO_PA3_SSI0FSS);
    GPIOPinConfigure(GPIO_PA4_SSI0RX);
    GPIOPinConfigure(GPIO_PA5_SSI0TX);

    GPIOPinTypeSSI(GPIO_PORTA_BASE, GPIO_PIN_2 | /* GPIO_PIN_3 | */
                                    GPIO_PIN_4 | GPIO_PIN_5);

    QUAD_BOARD_initDMA();
    SPI_init();
}
```

```c
/** ============================================================================
 *  @file       QUAD_BOARD.h
 *
 *  @brief      QUAD_BOARD Board Specific APIs
 *
 *  The QUAD_BOARD header file should be included in an application as
 *  follows:
 *  @code
 *  #include <QUAD_BOARD.h>
 *  @endcode
 *
 *  ============================================================================
 */
#ifndef QUAD_BOARD_H
#define QUAD_BOARD_H

#ifdef __cplusplus
extern "C" {
#endif

/* LEDs on QUAD_BOARD are active high. */
#define QUAD_BOARD_LED_OFF (0)
#define QUAD_BOARD_LED_ON  (1)

/*!
 *  @def    QUAD_BOARD_GPIOName
 *  @brief  Enum of GPIO names on the QUAD_BOARD dev board
 */
typedef enum QUAD_BOARD_GPIOName {
    QUAD_BOARD_SW1 = 0,
    QUAD_BOARD_SW2,
    QUAD_BOARD_ECHO,
    QUAD_BOARD_RPM,

    QUAD_BOARD_TRIGG,
    QUAD_BOARD_LED_RED,
    QUAD_BOARD_LED_BLUE,
    QUAD_BOARD_LED_GREEN,
    QUAD_BOARD_POWER_33,
    QUAD_BOARD_SPI_CE,
    QUAD_BOARD_SPI_CSN,

    QUAD_BOARD_GPIOCOUNT
} QUAD_BOARD_GPIOName;

/*!
 *  @def    QUAD_BOARD_I2CName
 *  @brief  Enum of I2C names on the QUAD_BOARD dev board
 */
typedef enum QUAD_BOARD_I2CName {
    QUAD_BOARD_I2C0 = 0,
    QUAD_BOARD_I2C2,
    QUAD_BOARD_I2CCOUNT
} QUAD_BOARD_I2CName;

/*!
 *  @def    QUAD_BOARD_PWMName
 *  @brief  Enum of PWM names on the QUAD_BOARD dev board
 */
typedef enum QUAD_BOARD_PWMName {
    QUAD_BOARD_PWM0 = 0,
    QUAD_BOARD_PWM1,
    QUAD_BOARD_PWM2,
    QUAD_BOARD_PWM3,

    QUAD_BOARD_PWMCOUNT
} QUAD_BOARD_PWMName;
```

```
/*!
 *  @def    QUAD_BOARD_UARTName
 *  @brief  Enum of UARTs on the QUAD_BOARD dev board
 */
typedef enum QUAD_BOARD_UARTName {
    QUAD_BOARD_UART0_USB = 0,
    QUAD_BOARD_UART5_BT_TELEMETRIA,
    QUAD_BOARD_UART1_BT_MANDO,
    QUAD_BOARD_UART7_AUX,

    QUAD_BOARD_UARTCOUNT
} QUAD_BOARD_UARTName;

/*!
 *  @def    QUAD_BOARD_WatchdogName
 *  @brief  Enum of Watchdogs on the QUAD_BOARD dev board
 */
typedef enum QUAD_BOARD_WatchdogName {
    QUAD_BOARD_WATCHDOG0 = 0,

    QUAD_BOARD_WATCHDOGCOUNT
} QUAD_BOARD_WatchdogName;

/*!
 *  @brief  Initialize board specific DMA settings
 *
 *  This function creates a hwi in case the DMA controller creates an error
 *  interrrupt, enables the DMA and supplies it with a uDMA control table.
 */

typedef enum QUAD_BOARD_SPIName {
    QUAD_BOARD_SPI0 = 0,

    QUAD_BOARD_SPICOUNT
} QUAD_BOARD_SPIName;


extern void QUAD_BOARD_initDMA(void);

/*!
 *  @brief  Initialize the general board specific settings
 *
 *  This function initializes the general board specific settings.
 *  This includes:
 *     - Flash wait states based on the process
 *     - Disable clock source to watchdog module
 *     - Enable clock sources for peripherals
 */
extern void QUAD_BOARD_initGeneral(void);

/*!
 *  @brief  Initialize board specific GPIO settings
 *
 *  This function initializes the board specific GPIO settings and
 *  then calls the GPIO_init API to initialize the GPIO module.
 *
 *  The GPIOs controlled by the GPIO module are determined by the GPIO_PinConfig
 *  variable.
 */
extern void QUAD_BOARD_initGPIO(void);

/*!
 *  @brief  Initialize board specific I2C settings
 *
 *  This function initializes the board specific I2C settings and then calls
 *  the I2C_init API to initialize the I2C module.
 *
 *  The I2C peripherals controlled by the I2C module are determined by the
```

```
 *  I2C_config variable.
 */
extern void QUAD_BOARD_initI2C(void);

/*!
 *  @brief  Initialize board specific PWM settings
 *
 *  This function initializes the board specific PWM settings and then calls
 *  the PWM_init API to initialize the PWM module.
 *
 *  The PWM peripherals controlled by the PWM module are determined by the
 *  PWM_config variable.
 */
extern void QUAD_BOARD_initPWM(void);

/*!
 *  @brief  Initialize board specific UART settings
 *
 *  This function initializes the board specific UART settings and then calls
 *  the UART_init API to initialize the UART module.
 *
 *  The UART peripherals controlled by the UART module are determined by the
 *  UART_config variable.
 */
extern void QUAD_BOARD_initUART(void);

/*!
 *  @brief  Initialize board specific Watchdog settings
 *
 *  This function initializes the board specific Watchdog settings and then
 *  calls the Watchdog_init API to initialize the Watchdog module.
 *
 *  The Watchdog peripherals controlled by the Watchdog module are determined
 *  by the Watchdog_config variable.
 */
extern void QUAD_BOARD_initWatchdog(void);

/*!
 *  @brief  Initialize board specific SPI settings
 *
 *  This function initializes the board specific SPI settings and then calls
 *  the SPI_init API to initialize the SPI module.
 *
 *  The SPI peripherals controlled by the SPI module are determined by the
 *  SPI_config variable.
 */
extern void QUAD_BOARD_initSPI(void);

#ifdef __cplusplus
}
#endif

#endif /* __QUAD_BOARD_H */
```

```c
/*
 * Sensores.h
 *
 *  Created on: 11/5/2015
 *      Author: Ruben
 */

#ifndef QUADROTOR_V1_3_1_SENSORES_H_
#define QUADROTOR_V1_3_1_SENSORES_H_

#include <ti/drivers/I2C.h>

#include "arm_math.h"
#include "Parametros.h"
#include "math.h"

#define CONVERTIR_A_RADIANES(DEGS) PI/180.0*DEGS
#define CONVERTIR_A_GRADOS(RADS) 180.0/PI*RADS

float32_t Normalizar_Grados(float32_t Grados);

typedef enum {DLPF_CFG_NO =-1, DLPF_CFG_0, DLPF_CFG_1, DLPF_CFG_2, DLPF_CFG_3, DLPF_CFG_4,
DLPF_CFG_5, DLPF_CFG_6, DLPF_CFG_7}tpDLPF_CFG;
typedef enum {Gain_Gyro_250, Gain_Gyro_500, Gain_Gyro_1000, Gain_Gyro_2000}tpGanancia_Gyro;
typedef enum {DLPF_CFG_GYRO_NO = -1, DLPF_CFG_GYRO_0, DLPF_CFG_GYRO_1, DLPF_CFG_GYRO_2,
DLPF_CFG_GYRO_3, DLPF_CFG_GYRO_4, DLPF_CFG_GYRO_5, DLPF_CFG_GYRO_6,
DLPF_CFG_GYRO_7}tpDLPF_CFG_GYRO;
typedef enum {Gain_Acel_2G, Gain_Acel_4G, Gain_Acel_8G, Gain_Acel_16G}tpGanancia_Acel;
typedef enum {DLPF_CFG_ACEL_NO =-1, DLPF_CFG_ACEL_0, DLPF_CFG_ACEL_1, DLPF_CFG_ACEL_2,
DLPF_CFG_ACEL_3, DLPF_CFG_ACEL_4, DLPF_CFG_ACEL_5, DLPF_CFG_ACEL_6,
DLPF_CFG_ACEL_7}tpDLPF_CFG_ACEL;

//............RECEPTOR............................//
typedef struct{
    uint16_t Canal_PWM[8];
    uint8_t Voltaje_Bat[4];
    uint8_t Error_conexion;
}tpLectura_Radio;

typedef struct {
    float32_t Rango_Salida[2];   //min max
    uint16_t Rango_Entrada[2]; //min max
}tpCalibracion_canal_PWM;

typedef tpCalibracion_canal_PWM tpCalibracion_Receptor[numCanales];

//..........Barometro.............................//

//Parametros barometro
#define Direccion_Barometro 0x77
#define Bar_Reg_Eprom_Barometro 0xAA //0xAA to 0xBF donde se hallan los parametros de
calibracion
#define Bar_Reg_leer_temp 0xF4
#define Bar_leer_Temp 0x2E
#define Bar_leer_Presion 0xF4
#define Bar_Reg_MSB 0xF6
#define Bar_Reg_LSB 0xF7
#define Bar_Reg_XLSB 0xF8

//..........9DOF_IMU.............................//
typedef union{
    struct{
    uint8_t x_acel_l;   //Los ponemos al reves (LOW y HIGH) dado que
    uint8_t x_acel_h;   //La arquitectura usa LITTLE Endian
    uint8_t y_acel_l;
    uint8_t y_acel_h;
    uint8_t z_acel_l;
    uint8_t z_acel_h;
```

```
    uint8_t temp_l;
    uint8_t temp_h;

    uint8_t x_vel_l;
    uint8_t x_vel_h;
    uint8_t y_vel_l;
    uint8_t y_vel_h;
    uint8_t z_vel_l;
    uint8_t z_vel_h;

    uint8_t x_mag_l;
    uint8_t x_mag_h;
    uint8_t y_mag_l;
    uint8_t y_mag_h;
    uint8_t z_mag_l;
    uint8_t z_mag_h;
  }Reg;

  struct{
        int16_t x_acel;
        int16_t y_acel;
        int16_t z_acel;

        int16_t temp;

        int16_t x_vel;
        int16_t y_vel;
        int16_t z_vel;

        int16_t x_mag;
        int16_t y_mag;
        int16_t z_mag;
  }Valor;
}tpLecturas_9DOF_IMU;

//.........IMU.................................//
typedef union{ //tpLecturas_IMU
    struct{
    uint8_t x_acel_l;   //Los ponemos al reves (LOW y HIGH) dado que
    uint8_t x_acel_h;   //La arquitectura usa LITTLE Endian
    uint8_t y_acel_l;
    uint8_t y_acel_h;
    uint8_t z_acel_l;
    uint8_t z_acel_h;

    uint8_t temp_l;
    uint8_t temp_h;

    uint8_t x_vel_l;
    uint8_t x_vel_h;
    uint8_t y_vel_l;
    uint8_t y_vel_h;
    uint8_t z_vel_l;
    uint8_t z_vel_h;

  }Reg;

  struct{
        int16_t x_acel;
        int16_t y_acel;
        int16_t z_acel;

        int16_t temp;

        int16_t x_vel;
        int16_t y_vel;
        int16_t z_vel;
```

```
  }Valor;
}tpLecturas_IMU;

typedef struct{ //tpCalibracion_IMU
    int16_t Rango_Acel_x[2]; // Min Max
    int16_t Media_Acel_x;
    float32_t Des_est_Acel_x;
    int16_t Rango_Acel_y[2];
    int16_t Media_Acel_y;
    float32_t Des_est_Acel_y;
    int16_t Rango_Acel_z[2];
    int16_t Media_Acel_z;
    float32_t Des_est_Acel_z;

    int16_t Media_Temp;
    float32_t Des_est_Temp;

    int16_t Rango_Vel_x[2];
    int16_t Media_Vel_x;
    float32_t Des_est_Vel_x;
    int16_t Rango_Vel_y[2];
    int16_t Media_Vel_y;
    float32_t Des_est_Vel_y;
    int16_t Rango_Vel_z[2];
    int16_t Media_Vel_z;
    float32_t Des_est_Vel_z;

    float32_t Correccion_Alineamiento_matriz[9];
    arm_matrix_instance_f32 Correccion_Alineamiento;

    float32_t Giro[3];

}tpCalibracion_IMU;


//.........BRUJULA.....................//

typedef union{ //tpLecturas_Brujula
    struct{
        uint8_t Magnetismo_x_l; //tpLecturas_Brujula
        uint8_t Magnetismo_x_h; //La arquitectura usa LITTLE Endian
        uint8_t Magnetismo_y_l;
        uint8_t Magnetismo_y_h;
        uint8_t Magnetismo_z_l;
        uint8_t Magnetismo_z_h;
    }Reg;
    struct{
        int16_t Magnetismo_x;
        int16_t Magnetismo_y;
        int16_t Magnetismo_z;
    }Valor;
}tpLecturas_Brujula;

typedef struct{
    int16_t Media_Magnetismo_x;
    float32_t Des_est_Mag_x;

    int16_t Media_Magnetismo_y;
    float32_t Des_est_Mag_y;

    int16_t Media_Magnetismo_z;
    float32_t Des_est_Mag_z;

    float32_t Offset_matriz[3];
    arm_matrix_instance_f32 Offset;
    float32_t Transformada_matriz[9];
    arm_matrix_instance_f32 Transformada;
```

```
}tpCalibracion_Brujula;

//.......BAROMETRO.................//

typedef struct{//tpLecturasBarometro
        uint32_t UP;
        uint16_t UT;

        uint32_t Presion;
        float Temperatura;

        //Parametros de calibracion
        int16_t AC1;
        int16_t AC2;
        int16_t AC3;
        uint16_t AC4;
        uint16_t AC5;
        uint16_t AC6;
        int16_t B1;
        int16_t B2;
        int16_t MB;
        int16_t MC;
        int16_t MD;
}tpLecturasBarometro;

typedef struct{//tpLecturasBarometro_BMP280

    uint32_t Presion;
    uint32_t Temperatura;

    uint16_t dig_T1;
    uint16_t dig_T2;
    uint16_t dig_T3;

    uint16_t dig_P1;
    uint16_t dig_P2;
    uint16_t dig_P3;
    uint16_t dig_P4;
    uint16_t dig_P5;
    uint16_t dig_P6;
    uint16_t dig_P7;
    uint16_t dig_P8;
    uint16_t dig_P9;
}tpLecturasBarometro_BMP280;

//...........GIROSCOPO.................//

typedef union{ //tpLecturas_Giroscopo
    struct{

        uint8_t x_vel_l;
        uint8_t x_vel_h;
        uint8_t y_vel_l;
        uint8_t y_vel_h;
        uint8_t z_vel_l;
        uint8_t z_vel_h;
    }Reg;
    struct{

        int16_t x_vel;
        int16_t y_vel;
        int16_t z_vel;
    }Valor;
}tpLecturas_Giroscopo;

//.............................IMU_9250.....................................//
typedef struct{
    uint8_t Direccion_IMU;
```

```
    uint8_t Direccion_MAG;
    uint8_t SMPLRT_DIV;
    float32_t Sensibilidad_Giroscopo;
    float32_t Sensibilidad_Acel;
    float32_t Sensibilidad_Brujula;

    tpDLPF_CFG_GYRO DLPF_CFG_GYRO;
    tpGanancia_Gyro Ganancia_Gyro;
    tpDLPF_CFG_ACEL DLPF_CFG_ACEL;
    tpGanancia_Acel Ganancia_Acel;

}tpIMU9250;

#define DIR_0_IMU_MPU9250 0b1101000
#define Dir_1_IMU_MPU9250 0b1101001
#define Dir_MAG_MPU9250 0x0C


#define IMU_MPU9250_INT_PIN_CFG 0x37
#define IMU_MPU9250_SMPLRT_DIV 0x19
#define IMU_MPU9250_USER_CTRL 0x6A
#define IMU_MPU9250_PWR_MGMT_1 0x6B
#define IMU_MPU9250_MAG_CTL1 0x0A
#define IMU_MPU9250_MAG_HXL 0x03

bool Iniciar_IMU_MPU9250(I2C_Handle I2C, tpIMU9250 IMU92520);
bool Leer_IMU_MPU9250(I2C_Handle I2C, tpIMU9250 IMU92520, tpLecturas_9DOF_IMU
*Lecturas_9DOF_IMU);

//.............................IMU_6050.....................................//
typedef struct{
    uint8_t Direccion;
    uint8_t SMPLRT_DIV;
    float32_t Sensibilidad_Giroscopo;
    float32_t Sensibilidad_Acel;

    tpDLPF_CFG DLPF_CFG;
    tpGanancia_Gyro Ganancia_Gyro;
    tpGanancia_Acel Ganancia_Acel;

}tpIMU6050;


#define Dir_0_IMU_MPU6050 0b1101000
#define Dir_1_IMU_MPU6050 0b1101001

#define IMU_MPU6050_SMPLRT_DIV 0x19
#define IMU_MPU6050_CONFIG 0x1A
#define IMU_MPU6050_GYRO_CONFIG 0x1B
#define IMU_MPU6050_ACCEL_CONFIG 0x1C

#define IMU_MPU6050_PWR_MGMT_1 0x6B
#define IMU_MPU6050_ACCEL_XOUT_H 0x3B

bool Iniciar_IMU_MPU6050(I2C_Handle I2C, tpIMU6050 IMU6050);
bool Leer_IMU_MPU6050(I2C_Handle I2C, tpIMU6050 IMU6050, tpLecturas_IMU *Lecturas_IMU);

//.............................GYRO_ITG3200.................................//
#define Dir_Gir_0 0b01101000
#define Dir_Gir_1 0b01101001
#define Reg_DLPF_FS 0x16
#define Reg_TEMP_OUT_H 0x1B

bool Iniciar_Giroscopio_ITG3200(I2C_Handle I2C, uint8_t Dir_Giroscopo, uint16_t
Frecuencia_muestreo, uint8_t Filtro);
bool Leer_Giroscopio_ITG3200(I2C_Handle I2C, uint8_t Dir_Giroscopo, tpLecturas_Giroscopo
*Lecturas_Giroscopo);
```

```
//.........................GYRO_L3G4200.....................................//
#define Dir_0_L3G4200 0x68
#define Dir_1_L3G4200 0x69

#define L3G4200_WHO_I_AM 0x0F
#define L3G4200_CTRL_REG1 0x20
#define L3G4200_CTRL_REG2 0x21
#define L3G4200_CTRL_REG3 0x22
#define L3G4200_CTRL_REG4 0x23
#define L3G4200_CTRL_REG5 0x24
#define L3G4200_REFERENCE 0x25
#define L3G4200_OUT_TEMP 0x26
#define L3G4200_STATUS_REG 0x27
#define L3G4200_OUT_X_L 0x28

#define L3G4200_FIFO_CTRL_REG 0x2E

typedef struct{
    uint8_t Direccion;
    float32_t Sensibilidad_Giroscopo;
    enum{dps_250 = 0, dps_500, dps_2000}Ganancia;
    enum{ODR_100_Hz = 0, ODR_200_Hz, ODR_400_Hz, ODR_800_Hz}ODR;
    enum{Bypass = 0, FIFO, Stream, Bypass_to_Stream, Stream_to_FIFO}Modo;
    enum{LPF1_0 = 0, LPF1_1, LPF1_2, LPF1_3}BW_LPF;
    enum{HPF_No_Filtro = 0, HPF_Filtro = 1 }HPF_activar;
    enum{Filtrado_LPF = 0, Filtrado_HPF, Filtrado_LPF2}Modo_Filtro;
    enum{HPF_0 = 0, HPF_1, HPF_2, HPF_3, HPF_4, HPF_5, HPF_6, HPF_7}BW_HPF;
    enum{HPF_Normal_mode = 0, HPF_Reference, HPF_Normal, HPF_Autoreset}HPF_modo;
    enum{BDU_Continuo = 0, BDU_No_continuo}BDU;
    enum{BLE_Big_Endian = 0, BLE_Little_Endian}BLE;
}tpGiroscopo_L3G4200;


bool Iniciar_Giroscopo_L3G4200(I2C_Handle I2C, tpGiroscopo_L3G4200 Giroscopo_L3G4200);
bool Leer_Giroscopo_L3G4200(I2C_Handle I2C, tpGiroscopo_L3G4200 Giroscopo_L3G4200,
tpLecturas_Giroscopo *Lecturas_Giroscopo);

//.............BRUJULA_HMC5883L............//
#define HMC5883L_DIR 0x1E
#define HMC5883L_CONFIG_A 0x00
#define HMC5883L_CONFIG_B 0x01
#define HMC5883L_MODE 0x02
#define HMC5883L_DATA_OUTPUT_X 0x03
#define HMC5883L_SATUS 0x09
/*
#ifdef SENSIBILIDAD_MAG
    const uint16_t Magnitud_HMC5883L[8] = {1370, 1090, 820, 660, 440, 390, 330, 230};
#endif
*/
typedef struct{
    float32_t Angulo_Rotacion;
    enum {MEDIA_1, MEDIA_2, MEDIA_4, MEDIA_8}Muestras_Media;
    enum {ODR_0_75_Hz, ODR_1_5_Hz, ODR_3_Hz, ODR_7_5_Hz, ODR_15_Hz, ODR_30_Hz, ODR_75_Hz}ODR;
    enum {Normal, Bias_Positivo, Bias_Negativo}Modo_Medida;
    enum {Gauss_0_88, Gauss_1_3, Gauss_1_9, Gauss_2_5, Gauss_4, Gauss_4_7, Gauss_5_6,
    Gauss_8_1}Ganancia;
    enum {I2C_400_Khz, I2C_3400_Khz}Velocidad_I2C;
    enum {Continuo, Simple, Idle}Modo_Operacion;
    enum {S_0 = 1370, S_1 = 1090, S_2 = 820, S_3 = 660, S_4 = 440, S_5 = 390, S_6 = 330, S_7
    = 230}Sensibilidad;
}tpHMC5883L;

bool Iniciar_Brujula_HMC5883L(I2C_Handle I2C, tpHMC5883L HMC5883L);
bool Leer_Brujula_HMC5883L(I2C_Handle I2C, tpHMC5883L HMC5883L, tpLecturas_Brujula
*Lecturas_Brujula);

//...BAROMETRO.............//
bool Iniciar_Barometro(I2C_Handle I2C, uint8_t Dir_Barometro, tpLecturasBarometro
```

```
*LecturasBarometro);
bool Iniciar_Medida_Temp_Barometro(I2C_Handle I2C, uint8_t Dir_Barometro,
tpLecturasBarometro *LecturasBarometro);
bool Leer_Temp_Barometro(I2C_Handle I2C, uint8_t Dir_Barometro, tpLecturasBarometro
*LecturasBarometro);
bool Iniciar_Medida_Presion_Barometro(I2C_Handle I2C, uint8_t Dir_Barometro,
tpLecturasBarometro *LecturasBarometro);
bool Leer_Presion_Barometro(I2C_Handle I2C, uint8_t Dir_Barometro, tpLecturasBarometro
*LecturasBarometro);

/*...BMP280..*/
#define BMP_280_DIR_0 0x76
#define BMP_280_DIR_1 0x77

#define BMP_280_calibracion_T1
#define BMP_280_press_msb 0xF7
#define BMP_280_ctrl_meas 0xF4

typedef enum {x1, x2, x4, x8, x16}OverSampling_BMP280;

typedef struct{
    uint8_t Direccion;
    enum {Mode_Sleep = 0, Mode_Forced, Mode_Normal}Modo;
    OverSampling_BMP280 Oversampling_Presion;
    OverSampling_BMP280 Oversampling_Temperatura;
    enum {Filtro_off = 0, Filtro_2, Filtro_4, Filtro_8, Filtro_16}Filtro_BMP280;
    enum {ms05 = 0, ms625, ms125, ms250, ms500, ms1000, ms2000, ms4000}t_sampling;
}tpBarometro_BMP280;

bool Iniciar_Barometro_BMP280(I2C_Handle I2C, tpBarometro_BMP280 *Barometro_BMP280,
tpLecturasBarometro_BMP280 *LecturasBarometro_BMP280);
bool Leer_Barometro_BMP280(I2C_Handle I2C, tpBarometro_BMP280 *Barometro_BMP280,
tpLecturasBarometro_BMP280 *LecturasBarometro_BMP280);
float32_t Conversion_Temperatura(tpLecturasBarometro_BMP280 *LecturasBarometro_BMP280);
float32_t Conversion_Altura(float32_t Temperatura, tpLecturasBarometro_BMP280
*LecturasBarometro_BMP280);


#endif /* QUADROTOR_V1_3_1_SENSORES_H_ */
```

```c
/*
 * Sensores.c
 *
 *  Created on: 11/5/2015
 *      Author: Ruben
 */
#include <ti/drivers/I2C.h>
#include "Sensores.h"

float32_t Normalizar_Grados(float32_t Grados){

    /*
    Grados += 180;
    Grados = fmodf(Grados, 360);
    Grados -= 180;
     */
    if( Grados > 180.0 ){ Grados -= 360;}
    else if( Grados < -180.0 ){ Grados += 360;}

    return Grados;
}

bool Iniciar_IMU_MPU9250(I2C_Handle I2C, tpIMU9250 IMU92520){
    I2C_Transaction I2C_Transmision;
    uint8_t bufferEscritura[6] ={IMU_MPU9250_PWR_MGMT_1, 0, 0, 0, 0, 0}; //Ponemos a cero el
    registro Reg_Power_Managent_1 para arrancar;
    bool TransmisionOK;

    I2C_Transmision.slaveAddress = IMU92520.Direccion_IMU;
    I2C_Transmision.writeBuf = bufferEscritura;
    I2C_Transmision.writeCount = 2;
    I2C_Transmision.readBuf = NULL;
    I2C_Transmision.readCount = 0;

    TransmisionOK = I2C_transfer(I2C, &I2C_Transmision);

    bufferEscritura[0] = IMU_MPU9250_SMPLRT_DIV;
    bufferEscritura[1] = IMU92520.SMPLRT_DIV;
    bufferEscritura[2] = IMU92520.DLPF_CFG_GYRO;
    bufferEscritura[3] = IMU92520.Ganancia_Gyro << 3;
    if (IMU92520.DLPF_CFG_GYRO == -1){
        bufferEscritura[3] |= 0b00000011;
    }
    bufferEscritura[4] = IMU92520.Ganancia_Acel << 3;
    bufferEscritura[5] = IMU92520.DLPF_CFG_ACEL;
    if (IMU92520.DLPF_CFG_ACEL == -1){
        bufferEscritura[3] |= 0b00000001;
    }

    I2C_Transmision.writeCount = 5;
    TransmisionOK |= I2C_transfer(I2C, &I2C_Transmision);


    bufferEscritura[0] = IMU_MPU9250_INT_PIN_CFG;
    bufferEscritura[1] = 0x02;

    I2C_Transmision.writeCount = 2;
    TransmisionOK |= I2C_transfer(I2C, &I2C_Transmision);

    //brujula

    bufferEscritura[0] = IMU_MPU9250_MAG_CTL1;
    bufferEscritura[1] = 0x16;

    I2C_Transmision.writeCount = 2;
    I2C_Transmision.slaveAddress = IMU92520.Direccion_MAG;
    TransmisionOK |= I2C_transfer(I2C, &I2C_Transmision);
```

```
    return(TransmisionOK);
}

bool Leer_IMU_MPU9250(I2C_Handle I2C, tpIMU9250 IMU92520, tpLecturas_9DOF_IMU
*Lecturas_9DOF_IMU){
    I2C_Transaction I2C_Transmision;

    uint8_t bufferEscritura[] = {IMU_MPU6050_ACCEL_XOUT_H};
    uint8_t bufferLectura[14];
    bool TransmisionOK;

    I2C_Transmision.slaveAddress = IMU92520.Direccion_IMU;
    I2C_Transmision.writeBuf = bufferEscritura;
    I2C_Transmision.writeCount = 1;
    I2C_Transmision.readBuf = bufferLectura;
//  I2C_Transmision.readBuf = &(Lecturas_IMU->Reg.x_acel_h);
    I2C_Transmision.readCount = 14;

    TransmisionOK = I2C_transfer(I2C, &I2C_Transmision);

    Lecturas_9DOF_IMU->Reg.x_acel_h=bufferLectura[0];
    Lecturas_9DOF_IMU->Reg.x_acel_l=bufferLectura[1];
    Lecturas_9DOF_IMU->Reg.y_acel_h=bufferLectura[2];
    Lecturas_9DOF_IMU->Reg.y_acel_l=bufferLectura[3];
    Lecturas_9DOF_IMU->Reg.z_acel_h=bufferLectura[4];
    Lecturas_9DOF_IMU->Reg.z_acel_l=bufferLectura[5];

    Lecturas_9DOF_IMU->Reg.temp_h=bufferLectura[6];
    Lecturas_9DOF_IMU->Reg.temp_l=bufferLectura[7];

    Lecturas_9DOF_IMU->Reg.x_vel_h=bufferLectura[8];
    Lecturas_9DOF_IMU->Reg.x_vel_l=bufferLectura[9];
    Lecturas_9DOF_IMU->Reg.y_vel_h=bufferLectura[10];
    Lecturas_9DOF_IMU->Reg.y_vel_l=bufferLectura[11];
    Lecturas_9DOF_IMU->Reg.z_vel_h=bufferLectura[12];
    Lecturas_9DOF_IMU->Reg.z_vel_l=bufferLectura[13];

    I2C_Transmision.slaveAddress = IMU92520.Direccion_MAG;
    I2C_Transmision.readCount = 6;
    I2C_Transmision.readBuf = &Lecturas_9DOF_IMU->Reg.x_mag_l;

    bufferEscritura[0] = IMU_MPU9250_MAG_HXL;

    TransmisionOK |= I2C_transfer(I2C, &I2C_Transmision);
    return(TransmisionOK);
}

bool Iniciar_IMU_MPU6050(I2C_Handle I2C, tpIMU6050 IMU6050){
    I2C_Transaction I2C_Transmision;
    uint8_t bufferEscritura[5] ={IMU_MPU6050_PWR_MGMT_1, 0, 0, 0, 0}; //Ponemos a cero el
    registro Reg_Power_Managent_1 para arrancar;
    bool TransmisionOK;

    I2C_Transmision.slaveAddress = IMU6050.Direccion;
    I2C_Transmision.writeBuf = bufferEscritura;
    I2C_Transmision.writeCount = 2;
    I2C_Transmision.readBuf = NULL;
    I2C_Transmision.readCount = 0;

    TransmisionOK = I2C_transfer(I2C, &I2C_Transmision);

    bufferEscritura[0] = IMU_MPU6050_SMPLRT_DIV;
    bufferEscritura[1] = IMU6050.SMPLRT_DIV;
    bufferEscritura[2] = IMU6050.DLPF_CFG;
    bufferEscritura[3] = IMU6050.Ganancia_Gyro << 3;
    bufferEscritura[4] = IMU6050.Ganancia_Acel << 3;

    I2C_Transmision.writeCount = 5;
```

```
    TransmisionOK |= I2C_transfer(I2C, &I2C_Transmision);
    return(TransmisionOK);
}

bool Leer_IMU_MPU6050(I2C_Handle I2C, tpIMU6050 IMU6050, tpLecturas_IMU *Lecturas_IMU){
    I2C_Transaction I2C_Transmision;

    uint8_t bufferEscritura[] = {IMU_MPU6050_ACCEL_XOUT_H};
    uint8_t bufferLectura[14];
    bool TransmisionOK;

    I2C_Transmision.slaveAddress = IMU6050.Direccion;
    I2C_Transmision.writeBuf = bufferEscritura;
    I2C_Transmision.writeCount = 1;
    I2C_Transmision.readBuf = bufferLectura;
//  I2C_Transmision.readBuf = &(Lecturas_IMU->Reg.x_acel_h);
    I2C_Transmision.readCount = 14;

    TransmisionOK = I2C_transfer(I2C, &I2C_Transmision);

    Lecturas_IMU->Reg.x_acel_h=bufferLectura[0];
    Lecturas_IMU->Reg.x_acel_l=bufferLectura[1];
    Lecturas_IMU->Reg.y_acel_h=bufferLectura[2];
    Lecturas_IMU->Reg.y_acel_l=bufferLectura[3];
    Lecturas_IMU->Reg.z_acel_h=bufferLectura[4];
    Lecturas_IMU->Reg.z_acel_l=bufferLectura[5];

    Lecturas_IMU->Reg.temp_h=bufferLectura[6];
    Lecturas_IMU->Reg.temp_l=bufferLectura[7];

    Lecturas_IMU->Reg.x_vel_h=bufferLectura[8];
    Lecturas_IMU->Reg.x_vel_l=bufferLectura[9];
    Lecturas_IMU->Reg.y_vel_h=bufferLectura[10];
    Lecturas_IMU->Reg.y_vel_l=bufferLectura[11];
    Lecturas_IMU->Reg.z_vel_h=bufferLectura[12];
    Lecturas_IMU->Reg.z_vel_l=bufferLectura[13];

    //Comprobar a pasar el puntero de lectura la direccion de Lecturas_IMU

    return(TransmisionOK);
}

////////////////////////////////////////////////////////////////////////////////////////////
///////////
bool Iniciar_Brujula_HMC5883L(I2C_Handle I2C, tpHMC5883L HMC5883L){
    I2C_Transaction I2C_Transmision;
    uint8_t bufferEscritura[4] = {0, 0, 0, 0};


    I2C_Transmision.slaveAddress = HMC5883L_DIR;
    I2C_Transmision.writeBuf = bufferEscritura;
    I2C_Transmision.writeCount = 4;
    I2C_Transmision.readBuf = NULL;
    I2C_Transmision.readCount = 0;

    bufferEscritura[0] = HMC5883L_CONFIG_A;
    bufferEscritura[1] = (HMC5883L.Muestras_Media << 4) | (HMC5883L.ODR << 2) |
    HMC5883L.Modo_Medida;
    bufferEscritura[2] = HMC5883L.Ganancia << 5;
    bufferEscritura[3] = (HMC5883L.Velocidad_I2C << 7) | HMC5883L.Modo_Operacion;

    return(I2C_transfer(I2C, &I2C_Transmision));
}

bool Leer_Brujula_HMC5883L(I2C_Handle I2C, tpHMC5883L HMC5883L, tpLecturas_Brujula
*Lecturas_Brujula){
    I2C_Transaction I2C_Transmision;
```

```
    uint8_t bufferEscritura[] = {HMC5883L_DATA_OUTPUT_X};
    uint8_t bufferLectura[6];
    bool TransmisionOK;

    I2C_Transmision.slaveAddress = HMC5883L_DIR;
    I2C_Transmision.writeBuf = bufferEscritura;
    I2C_Transmision.writeCount = 1;
    I2C_Transmision.readBuf = bufferLectura;
    I2C_Transmision.readCount = 6;

    TransmisionOK = I2C_transfer(I2C, &I2C_Transmision);

    Lecturas_Brujula->Reg.Magnetismo_x_h = bufferLectura[0];
    Lecturas_Brujula->Reg.Magnetismo_x_l = bufferLectura[1];
    Lecturas_Brujula->Reg.Magnetismo_z_h = bufferLectura[2];
    Lecturas_Brujula->Reg.Magnetismo_z_l = bufferLectura[3];
    Lecturas_Brujula->Reg.Magnetismo_y_h = bufferLectura[4];
    Lecturas_Brujula->Reg.Magnetismo_y_l = bufferLectura[5];

    return(TransmisionOK);
}

////////////////////////////////////////////////////////////////////////////////////////////
///////////
bool Iniciar_Giroscopo_L3G4200(I2C_Handle I2C, tpGiroscopo_L3G4200 Giroscopo_L3G4200){
    I2C_Transaction I2C_Transmision;
    bool TransmisionOK;
    uint8_t bufferEscritura[6];

    bufferEscritura[0] = 0x80 | L3G4200_CTRL_REG1;
    bufferEscritura[1] = (Giroscopo_L3G4200.ODR<<6 | Giroscopo_L3G4200.BW_LPF<<4 | 0x0F);
    bufferEscritura[2] = (Giroscopo_L3G4200.HPF_modo << 4 | Giroscopo_L3G4200.BW_HPF);
    bufferEscritura[3] = 0;
    bufferEscritura[4] = (Giroscopo_L3G4200.BDU << 7 | Giroscopo_L3G4200.BLE << 6 |
    Giroscopo_L3G4200.Ganancia << 4);
    bufferEscritura[5] = ( 1 << 6 | Giroscopo_L3G4200.HPF_activar << 4 |
    Giroscopo_L3G4200.Modo_Filtro );


    I2C_Transmision.slaveAddress = Giroscopo_L3G4200.Direccion;
    I2C_Transmision.writeBuf = bufferEscritura;
    I2C_Transmision.writeCount = 6;
    I2C_Transmision.readBuf = NULL;
    I2C_Transmision.readCount = 0;

    TransmisionOK = I2C_transfer(I2C, &I2C_Transmision);

    bufferEscritura[0] = L3G4200_FIFO_CTRL_REG;
    bufferEscritura[1] = Giroscopo_L3G4200.Modo << 5;

    I2C_Transmision.slaveAddress = Giroscopo_L3G4200.Direccion;
    I2C_Transmision.writeBuf = bufferEscritura;
    I2C_Transmision.writeCount = 2;
    I2C_Transmision.readBuf = NULL;
    I2C_Transmision.readCount = 0;

    TransmisionOK |= I2C_transfer(I2C, &I2C_Transmision);

    return(TransmisionOK);
}

bool Leer_Giroscopo_L3G4200(I2C_Handle I2C, tpGiroscopo_L3G4200 Giroscopo_L3G4200,
tpLecturas_Giroscopo *Lecturas_Giroscopo){
    I2C_Transaction I2C_Transmision;
    uint8_t bufferEscritura[] = {L3G4200_OUT_X_L};

    I2C_Transmision.slaveAddress = Giroscopo_L3G4200.Direccion;
    I2C_Transmision.writeBuf = bufferEscritura;
```

```
    I2C_Transmision.writeCount = 1;
    I2C_Transmision.readBuf = &(Lecturas_Giroscopo->Reg.x_vel_l);
    I2C_Transmision.readCount = 6;

    return( I2C_transfer(I2C, &I2C_Transmision));
}
////////////////////////////////////////////////////////////////////////////////
bool Iniciar_Barometro(I2C_Handle I2C, uint8_t Dir_Barometro, tpLecturasBarometro
*LecturasBarometro){
    I2C_Transaction I2C_Transmision;
    uint8_t bufferLectura[22];
    uint8_t bufferEscritura[] = {Bar_Reg_Eprom_Barometro};
    bool TransmisionOK;

    I2C_Transmision.slaveAddress = Dir_Barometro;
    I2C_Transmision.writeBuf = bufferEscritura;
    I2C_Transmision.writeCount = 1;
    I2C_Transmision.readBuf = bufferLectura;
    I2C_Transmision.readCount = 22;

    TransmisionOK = I2C_transfer(I2C, &I2C_Transmision);

    LecturasBarometro->AC1 = bufferLectura[0]<<8  | bufferLectura[1];
    LecturasBarometro->AC2 = bufferLectura[2]<<8  | bufferLectura[3];
    LecturasBarometro->AC3 = bufferLectura[4]<<8  | bufferLectura[5];
    LecturasBarometro->AC4 = bufferLectura[6]<<8  | bufferLectura[7];
    LecturasBarometro->AC5 = bufferLectura[8]<<8  | bufferLectura[9];
    LecturasBarometro->AC6 = bufferLectura[10]<<8 | bufferLectura[11];
    LecturasBarometro->B1  = bufferLectura[12]<<8 | bufferLectura[13];
    LecturasBarometro->B2  = bufferLectura[14]<<8 | bufferLectura[15];
    LecturasBarometro->MB  = bufferLectura[16]<<8 | bufferLectura[17];
    LecturasBarometro->MC  = bufferLectura[18]<<8 | bufferLectura[19];
    LecturasBarometro->MD  = bufferLectura[20]<<8 | bufferLectura[21];

    return(TransmisionOK);
}

bool Iniciar_Medida_Temp_Barometro(I2C_Handle I2C, uint8_t Dir_Barometro,
tpLecturasBarometro *LecturasBarometro){
    I2C_Transaction I2C_Transmision;
    uint8_t bufferEscritura[] = {Bar_Reg_leer_temp, Bar_leer_Temp};

    I2C_Transmision.slaveAddress = Dir_Barometro;
    I2C_Transmision.writeBuf = bufferEscritura;
    I2C_Transmision.writeCount = 2;
    I2C_Transmision.readBuf = NULL;
    I2C_Transmision.readCount = 0;

    return(I2C_transfer(I2C, &I2C_Transmision));
}

bool Leer_Temp_Barometro(I2C_Handle I2C, uint8_t Dir_Barometro, tpLecturasBarometro
*LecturasBarometro){
    I2C_Transaction I2C_Transmision;
    uint8_t bufferLectura[2];
    uint8_t bufferEscritura[] = {Bar_Reg_MSB};
    bool TransmisionOK;
    int32_t X1;
    int32_t X2;
    int32_t B5;


    I2C_Transmision.slaveAddress = Dir_Barometro;
    I2C_Transmision.writeBuf = bufferEscritura;
    I2C_Transmision.writeCount = 1;
    I2C_Transmision.readBuf = bufferLectura;
    I2C_Transmision.readCount = 2;
```

```
    TransmisionOK = I2C_transfer(I2C, &I2C_Transmision);

    LecturasBarometro->UT = bufferLectura[0]<<8 | bufferLectura[1];

    X1 = ((LecturasBarometro->UT - LecturasBarometro->AC6) * LecturasBarometro->AC5) >> 15;
    X2 = LecturasBarometro->MC << 11 / ( X1 + LecturasBarometro->MD);
    B5 = X1 + X2;
    LecturasBarometro->Temperatura = (B5 + 8) / 160.0;

    return(TransmisionOK);
}

bool Iniciar_Medida_Presion_Barometro(I2C_Handle I2C, uint8_t Dir_Barometro,
tpLecturasBarometro *LecturasBarometro){
    I2C_Transaction I2C_Transmision;
    uint8_t bufferEscritura[] = {Bar_Reg_leer_temp, Bar_leer_Presion};

    I2C_Transmision.slaveAddress = Dir_Barometro;
    I2C_Transmision.writeBuf = bufferEscritura;
    I2C_Transmision.writeCount = 2;
    I2C_Transmision.readBuf = NULL;
    I2C_Transmision.readCount = 0;

    return(I2C_transfer(I2C, &I2C_Transmision));
}

bool Leer_Presion_Barometro(I2C_Handle I2C, uint8_t Dir_Barometro, tpLecturasBarometro
*LecturasBarometro){
    I2C_Transaction I2C_Transmision;
    uint8_t bufferLectura[3];
    uint8_t bufferEscritura[] = {Bar_Reg_MSB};
    bool TransmisionOK;

    int32_t X1;
    int32_t X2;
    int32_t X3;
    int32_t B3;
    int32_t B4;
    int32_t B5;
    int32_t B6;
    int32_t B7;

    I2C_Transmision.slaveAddress = Dir_Barometro;
    I2C_Transmision.writeBuf = bufferEscritura;
    I2C_Transmision.writeCount = 1;
    I2C_Transmision.readBuf = bufferLectura;
    I2C_Transmision.readCount = 3;

    TransmisionOK = I2C_transfer(I2C, &I2C_Transmision);

    LecturasBarometro->UT = (bufferLectura[0]<<16 | bufferLectura[1] | bufferLectura[2])>>5;

    X1 = ((LecturasBarometro->UT - LecturasBarometro->AC6) * LecturasBarometro->AC5) >> 15;
    X2 = LecturasBarometro->MC << 11 / ( X1 + LecturasBarometro->MD);
    B5 = X1 + X2;

    B6 = B5 - 4000;
    X1 = (LecturasBarometro->B2 * (B6*B6 << 12 )) << 11;
    X2 = LecturasBarometro->AC2 * B6 << 11;
    X3 = X1 + X2;
    B3 = ((LecturasBarometro->AC1*4 + X3) << 5) / 4;
    X1 = LecturasBarometro->AC3 * B6 << 13;
    X2 = (LecturasBarometro->B1 * (B6*B6 << 12 )) << 16;
    X3 = (X1 + X2 + 2) / 4;

    B4 = LecturasBarometro->AC4 * (unsigned long)(X3 + 32768) << 15;
    B7 = ((unsigned long)LecturasBarometro->UP -B3 ) * (5000 >> 3);
```

```
    if (B7 < 0x80000000) {
        LecturasBarometro->Presion = B7 * 2 / B4;
    }else{
        LecturasBarometro->Presion = (B7 /B4) * 2;
    }

    X1 = (LecturasBarometro->Presion << 8) * (LecturasBarometro->Presion << 8);
    X1 = (X1 * 3038) << 16;
    X2 = (7357 * LecturasBarometro->Presion ) << 16;
    LecturasBarometro->Presion = LecturasBarometro->Presion + (X1 - X2 + 3791) << 4;

    return(TransmisionOK);
}
////////////////////////////////////////////////////////////////////////////
bool Iniciar_Giroscopio_ITG3200(I2C_Handle I2C, uint8_t Dir_Giroscopo, uint16_t
Frecuencia_muestreo, uint8_t Filtro){
    I2C_Transaction I2C_Transmision;
    uint8_t bufferEscritura[2] = {Reg_DLPF_FS, 0};

    bufferEscritura[1] = ((0x03<<3) | (Filtro & 0x7));

    I2C_Transmision.slaveAddress = Dir_Giroscopo;
    I2C_Transmision.writeBuf = bufferEscritura;
    I2C_Transmision.writeCount = 2;
    I2C_Transmision.readBuf = NULL;
    I2C_Transmision.readCount = 0;
/*
    I2C_transfer(I2C, &I2C_Transmision);

    bufferEscritura[0] = Reg_SMPRT_DIV;
    bufferEscritura[1] = 8000/Frecuencia_muestreo - 1;

    I2C_transfer(I2C, &I2C_Transmision);

    bufferEscritura[0] = Reg_CONFIG;
    bufferEscritura[1] = (Filtro & 0x07);
*/
    return(I2C_transfer(I2C, &I2C_Transmision));
}
//bool Sensibilidad_Giroscopio_ITG3200(I2C_Handle I2C , uint8_t Dir_Giroscopo,
tpRangoGiroscopo_ITG3200 RangoGiroscopo);
bool Leer_Giroscopio_ITG3200(I2C_Handle I2C, uint8_t Dir_Giroscopo, tpLecturas_Giroscopo
*Lecturas_Giroscopo){
    I2C_Transaction I2C_Transmision;
    uint8_t bufferEscritura[] = {Reg_TEMP_OUT_H};
    uint8_t bufferLectura[8];
    bool TransmisionOK;

    I2C_Transmision.slaveAddress = Dir_Giroscopo;
    I2C_Transmision.writeBuf = bufferEscritura;
    I2C_Transmision.writeCount = 1;
    I2C_Transmision.readBuf = bufferLectura;
//  I2C_Transmision.readBuf = &(Lecturas_IMU->Reg.x_acel_h);
    I2C_Transmision.readCount = 8;

    TransmisionOK = I2C_transfer(I2C, &I2C_Transmision);

    Lecturas_Giroscopo->Reg.x_vel_h=bufferLectura[2];
    Lecturas_Giroscopo->Reg.x_vel_l=bufferLectura[3];
    Lecturas_Giroscopo->Reg.y_vel_h=bufferLectura[4];
    Lecturas_Giroscopo->Reg.y_vel_l=bufferLectura[5];
    Lecturas_Giroscopo->Reg.z_vel_h=bufferLectura[6];
    Lecturas_Giroscopo->Reg.z_vel_l=bufferLectura[7];

    return(TransmisionOK);
}

bool Iniciar_Barometro_BMP280(I2C_Handle I2C, tpBarometro_BMP280 *Barometro_BMP280,
```

```
tpLecturasBarometro_BMP280 *LecturasBarometro_BMP280){
    I2C_Transaction I2C_Transmision;
    uint8_t bufferEscritura[] = {BMP_280_ctrl_meas, 0, 0};
    uint8_t bufferLectura[24];
    bool TransmisionOK;

    I2C_Transmision.slaveAddress = Barometro_BMP280->Direccion;
    I2C_Transmision.writeBuf = bufferEscritura;
    I2C_Transmision.writeCount = 3;
    I2C_Transmision.readBuf = NULL;
    I2C_Transmision.readCount = 0;

    bufferEscritura[1] = (Barometro_BMP280->t_sampling << 5) |
    (Barometro_BMP280->Filtro_BMP280 << 2);
    bufferEscritura[2] = (Barometro_BMP280->Oversampling_Temperatura << 5) |
    (Barometro_BMP280->Oversampling_Presion<< 2) | (Barometro_BMP280->Modo);

    TransmisionOK = I2C_transfer(I2C, &I2C_Transmision);

    bufferEscritura[0] = BMP_280_ctrl_meas;

    I2C_Transmision.writeCount = 1;
    I2C_Transmision.readBuf = bufferLectura;
    I2C_Transmision.readCount = 24;

    TransmisionOK = I2C_transfer(I2C, &I2C_Transmision);

    LecturasBarometro_BMP280->dig_T1 = bufferLectura[1]<<8 |bufferLectura[0];
    LecturasBarometro_BMP280->dig_T2 = bufferLectura[3]<<8 |bufferLectura[2];
    LecturasBarometro_BMP280->dig_T3 = bufferLectura[5]<<8 |bufferLectura[4];

    LecturasBarometro_BMP280->dig_P1 = bufferLectura[7]<<8 |bufferLectura[6];
    LecturasBarometro_BMP280->dig_P2 = bufferLectura[9]<<8 |bufferLectura[8];
    LecturasBarometro_BMP280->dig_P3 = bufferLectura[11]<<8 |bufferLectura[10];
    LecturasBarometro_BMP280->dig_P4 = bufferLectura[13]<<8 |bufferLectura[12];
    LecturasBarometro_BMP280->dig_P5 = bufferLectura[15]<<8 |bufferLectura[14];
    LecturasBarometro_BMP280->dig_P6 = bufferLectura[17]<<8 |bufferLectura[16];
    LecturasBarometro_BMP280->dig_P7 = bufferLectura[19]<<8 |bufferLectura[18];
    LecturasBarometro_BMP280->dig_P8 = bufferLectura[21]<<8 |bufferLectura[20];
    LecturasBarometro_BMP280->dig_P9 = bufferLectura[23]<<8 |bufferLectura[22];


    return TransmisionOK;
}
bool Leer_Barometro_BMP280(I2C_Handle I2C, tpBarometro_BMP280 *Barometro_BMP280,
tpLecturasBarometro_BMP280 *LecturasBarometro_BMP280){
    I2C_Transaction I2C_Transmision;
    uint8_t bufferEscritura[] = {BMP_280_ctrl_meas};
    uint8_t bufferLectura[6];
    bool TransmisionOK;

    I2C_Transmision.slaveAddress = Barometro_BMP280->Direccion;
    I2C_Transmision.writeBuf = bufferEscritura;
    I2C_Transmision.writeCount = 1;
    I2C_Transmision.readBuf = bufferLectura;
    I2C_Transmision.readCount = 6;

    TransmisionOK = I2C_transfer(I2C, &I2C_Transmision);

    LecturasBarometro_BMP280->Presion = bufferEscritura[0]<<12 | bufferEscritura[1]<<8 |
    bufferEscritura[2]>>3 ;
    LecturasBarometro_BMP280->Temperatura = bufferEscritura[3]<<12 | bufferEscritura[4]<<8 |
    bufferEscritura[5]>>3 ;

    return TransmisionOK;
}

float32_t Conversion_Temperatura(tpLecturasBarometro_BMP280 *LecturasBarometro_BMP280){
```

```
    float32_t var1, var2;

    var1 = (LecturasBarometro_BMP280->Temperatura/16384.0 -
    LecturasBarometro_BMP280->dig_T1/1024.0) * LecturasBarometro_BMP280->dig_T2;
    var2 = ( (LecturasBarometro_BMP280->Temperatura/131072.0 -
    LecturasBarometro_BMP280->dig_T1/8192.0) *
    (LecturasBarometro_BMP280->Temperatura/131072.0 -
    LecturasBarometro_BMP280->dig_T1/8192.0) ) * LecturasBarometro_BMP280->dig_T3 ;

    return ((var1 + var2) / 5120.0);
}

float32_t Conversion_Altura(float32_t Temperatura, tpLecturasBarometro_BMP280
*LecturasBarometro_BMP280){
    float32_t var1, var2, p;

    var1 = Temperatura * 2560.0 - 64000.0;
    var2 = var1 * var1 * LecturasBarometro_BMP280->dig_P6 / 32768.0;
    var2 = var2 + var1 * LecturasBarometro_BMP280->dig_P5 * 2.0;
    var2 = var2/4.0 + LecturasBarometro_BMP280->dig_P4 * 65536.0;
    var1 = (LecturasBarometro_BMP280->dig_P3 * var1 * var1 / 524288.0 +
    LecturasBarometro_BMP280->dig_P2 * var1) / 524288.0;
    var1 = (1.0 + var1 / 32768.0)*LecturasBarometro_BMP280->dig_P1;

    if (var1 == 0.0)    return 0; // avoid exception caused by division by zero

    p = 1048576.0 - LecturasBarometro_BMP280->Presion;
    p = (p - var2/4096.0) * 6250.0 / var1;
    var1 = LecturasBarometro_BMP280->dig_P9 * p * p / 2147483648.0;
    var2 = p * LecturasBarometro_BMP280->dig_P8 / 32768.0;
    p = p + (var1 + var2 + LecturasBarometro_BMP280->dig_P7) / 16.0;

    return p;



}
```

```
/*
 * Servidores.h
 *
 *  Created on: 17/5/2015
 *      Author: Ruben
 */

#ifndef QUADROTOR_V1_3_1_SERVIDORES_H_
#define QUADROTOR_V1_3_1_SERVIDORES_H_

#include "Sensores.h"
#include "arm_math.h"

void Iniciar_Servidores();

void Leer_servidor_Lecturas_IMU(tpLecturas_IMU *Lecturas_IMU);
void Escribir_servidor_Lecturas_IMU(tpLecturas_IMU *Lecturas_IMU);

void Leer_servidor_Lecturas_IMU_9DOF(tpLecturas_9DOF_IMU *Lecturas_9DOF_IMU);
void Escribir_servidor_Lecturas_IMU_9DOF(tpLecturas_9DOF_IMU *Lecturas_9DOF_IMU);

void Leer_servidor_Lecturas_Giroscopo(tpLecturas_Giroscopo *Lecturas_Giroscopo);
void Escribir_servidor_Lecturas_Giroscopo(tpLecturas_Giroscopo *Lecturas_Giroscopo);

void Leer_servidor_Lecturas_Brujula(tpLecturas_Brujula *Lecturas_Brujula);
void Escribir_servidor_Lecturas_Brujula(tpLecturas_Brujula *Lecturas_Brujula);

void Leer_servidor_DCM(float32_t* DCM);
void Escribir_servidor_DCM(float32_t* DCM);

void Leer_servidor_RPY(float32_t *Roll, float32_t *Pitch, float32_t *Yaw);
void Escribir_servidor_RPY(float32_t *Roll, float32_t *Pitch, float32_t *Yaw);

void Leer_servidor_quaternios(float32_t* q);
void Escribir_servidor_quaternios(float32_t* q);

void Leer_servidor_Variables_Estado_Medidas(float32_t* Variables_Estado_Medidas);
void Escribir_servidor_Variables_Estado_Medidas(float32_t* Variables_Estado_Medidas);

void Leer_servidor_Variables_Estado_Estimadas(float32_t* Variables_Estado_Estimadas);
void Escribir_servidor_Variables_Estado_Estimadas(float32_t* Variables_Estado_Estimadas);

void Leer_servidor_Perturbaciones_Estimadas(float32_t* Perturbaciones_Estimadas);
void Escribir_servidor_Perturbaciones_Estimadas(float32_t* Perturbaciones_Estimadas);
void Resetear_servidor_Perturbaciones_Estimadas();

void Leer_servidor_Referencia(float32_t* Referencia, int16_t* Referencia_Entero);
void Escribir_servidor_Referencia(float32_t* Referencia, int16_t* Referencia_Entero);
float32_t* Direccion_servidor_Referencia();

#endif /* QUADROTOR_V1_3_1_SERVIDORES_H_ */
```

```c
/*
 * Servidores.c
 *
 *  Created on: 17/5/2015
 *      Author: Ruben
 */
#include "Servidores.h"
#include "arm_math.h"
#include <ti/sysbios/gates/GateMutexPri.h>

static volatile GateMutexPri_Handle SERVIDOR_Datos_IMU;
static volatile GateMutexPri_Handle SERVIDOR_Datos_9DOF_IMU;
static volatile GateMutexPri_Handle SERVIDOR_Datos_Giroscopo;
static volatile GateMutexPri_Handle SERVIDOR_Datos_Brujula;
static volatile GateMutexPri_Handle SERVIDOR_DCM;
static volatile GateMutexPri_Handle SERVIDOR_YPR;
static volatile GateMutexPri_Handle SERVIDOR_Quaternio;
static volatile GateMutexPri_Handle SERVIDOR_Variables_Estado_Medidas;
static volatile GateMutexPri_Handle SERVIDOR_Variables_Estado_Estimadas;
static volatile GateMutexPri_Handle SERVIDOR_Perturbaciones_Estimadas;
static volatile GateMutexPri_Handle SERVIDOR_Referencia;

static tpLecturas_IMU Lectura_Almacenada_IMU = { 0, 0, 0, 0, 0, 0, 0};
static tpLecturas_9DOF_IMU Lecturas_9DOF_IMU_Almacenada = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
static tpLecturas_Giroscopo Lecturas_Almacenada_Giroscopo = {0, 0, 0, 0};
static tpLecturas_Brujula Lectura_Almacenada_Brujula = { 0, 0, 0};
static float32_t DCM_Almacenada[9] = { 0, 0, 0, 0, 0, 0, 0, 0, 0};
static float32_t Roll_Almacenado = 0;
static float32_t Pitch_Almacenado = 0;
static float32_t Yaw_Almacenado= 0;
static float32_t q_almacenado[4] = {0, 0, 0, 0};
static float32_t Variables_Estado_Medidas_Almacenadas[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
static float32_t Variables_Estado_Estimadas_Almacenadas[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
static float32_t Perturbaciones_Estimadas_Almacenadas[4] = {0, 0, 0, 0};
static float32_t Referencia_Almacenada[4] = {0, 0, 0, 0};
static int16_t Referencia_Almacenada_Entero[4] = {0, 0, 0, 0};

void Iniciar_Servidores(){
    SERVIDOR_Datos_IMU = GateMutexPri_create(NULL, NULL) ;
    SERVIDOR_Datos_9DOF_IMU = GateMutexPri_create(NULL, NULL) ;
    SERVIDOR_Datos_Giroscopo = GateMutexPri_create(NULL, NULL) ;
    SERVIDOR_Datos_Brujula = GateMutexPri_create(NULL, NULL) ;
    SERVIDOR_DCM = GateMutexPri_create(NULL, NULL);
    SERVIDOR_YPR = GateMutexPri_create(NULL, NULL) ;
    SERVIDOR_Quaternio = GateMutexPri_create(NULL, NULL) ;
    SERVIDOR_Variables_Estado_Medidas = GateMutexPri_create(NULL, NULL) ;
    SERVIDOR_Variables_Estado_Estimadas = GateMutexPri_create(NULL, NULL) ;
    SERVIDOR_Perturbaciones_Estimadas = GateMutexPri_create(NULL, NULL);
    SERVIDOR_Referencia = GateMutexPri_create(NULL, NULL) ;
}

void Leer_servidor_Lecturas_IMU(tpLecturas_IMU *Lecturas_IMU){
    IArg Key;

    Key = GateMutexPri_enter(SERVIDOR_Datos_IMU);
    memcpy(Lecturas_IMU, &Lectura_Almacenada_IMU, sizeof(tpLecturas_IMU));
    GateMutexPri_leave(SERVIDOR_Datos_IMU, Key);
}

void Escribir_servidor_Lecturas_IMU(tpLecturas_IMU* Lecturas_IMU){
    IArg Key;

    Key = GateMutexPri_enter(SERVIDOR_Datos_IMU);
    memcpy(&Lectura_Almacenada_IMU, Lecturas_IMU, sizeof(tpLecturas_IMU));
    GateMutexPri_leave(SERVIDOR_Datos_IMU, Key);
}

void Leer_servidor_Lecturas_IMU_9DOF(tpLecturas_9DOF_IMU *Lecturas_9DOF_IMU){
```

```
    IArg Key;

    Key = GateMutexPri_enter(SERVIDOR_Datos_9DOF_IMU);
    memcpy(Lecturas_9DOF_IMU, &Lecturas_9DOF_IMU_Almacenada, sizeof(tpLecturas_9DOF_IMU));
    GateMutexPri_leave(SERVIDOR_Datos_9DOF_IMU, Key);
}
void Escribir_servidor_Lecturas_IMU_9DOF(tpLecturas_9DOF_IMU *Lecturas_9DOF_IMU){
    IArg Key;

    Key = GateMutexPri_enter(SERVIDOR_Datos_9DOF_IMU);
    memcpy(&Lecturas_9DOF_IMU_Almacenada, Lecturas_9DOF_IMU, sizeof(tpLecturas_IMU));
    GateMutexPri_leave(SERVIDOR_Datos_9DOF_IMU, Key);
}

void Leer_servidor_Lecturas_Giroscopo(tpLecturas_Giroscopo *Lecturas_Giroscopo){
    IArg Key;

    Key = GateMutexPri_enter(SERVIDOR_Datos_Giroscopo);
    memcpy(Lecturas_Giroscopo, &Lecturas_Almacenada_Giroscopo, sizeof(tpLecturas_Giroscopo));
    GateMutexPri_leave(SERVIDOR_Datos_Giroscopo, Key);
}
void Escribir_servidor_Lecturas_Giroscopo(tpLecturas_Giroscopo *Lecturas_Giroscopo){
    IArg Key;

    Key = GateMutexPri_enter(SERVIDOR_Datos_Giroscopo);
    memcpy(&Lecturas_Almacenada_Giroscopo, Lecturas_Giroscopo, sizeof(tpLecturas_Giroscopo));
    GateMutexPri_leave(SERVIDOR_Datos_Giroscopo, Key);
}

void Leer_servidor_Lecturas_Brujula(tpLecturas_Brujula *Lecturas_Brujula){
    IArg Key;

    Key = GateMutexPri_enter(SERVIDOR_Datos_Brujula);
    memcpy(Lecturas_Brujula, &Lectura_Almacenada_Brujula, sizeof(tpLecturas_Brujula));
    GateMutexPri_leave(SERVIDOR_Datos_Brujula, Key);
}

void Escribir_servidor_Lecturas_Brujula(tpLecturas_Brujula* Lecturas_Brujula){
    IArg Key;

    Key = GateMutexPri_enter(SERVIDOR_Datos_Brujula);
    memcpy(&Lectura_Almacenada_Brujula, Lecturas_Brujula, sizeof(tpLecturas_Brujula));
    GateMutexPri_leave(SERVIDOR_Datos_Brujula, Key);
}

void Leer_servidor_DCM(float32_t* DCM){
    IArg Key;

    Key = GateMutexPri_enter(SERVIDOR_DCM);
    arm_copy_f32(DCM_Almacenada, DCM, sizeof(DCM_Almacenada)/sizeof(float32_t));
    GateMutexPri_leave(SERVIDOR_DCM, Key);
}

void Escribir_servidor_DCM(float32_t* DCM){
    IArg Key;

    Key = GateMutexPri_enter(SERVIDOR_DCM);
    arm_copy_f32(DCM, DCM_Almacenada, sizeof(DCM_Almacenada)/sizeof(float32_t));
    GateMutexPri_leave(SERVIDOR_DCM, Key);
}

void Leer_servidor_RPY(float32_t *Roll, float32_t *Pitch, float32_t *Yaw){
    IArg Key;

    Key = GateMutexPri_enter(SERVIDOR_YPR);
    *Roll = Roll_Almacenado;
    *Pitch = Pitch_Almacenado;
    *Yaw = Yaw_Almacenado;
```

```
    GateMutexPri_leave(SERVIDOR_YPR, Key);
}

void Escribir_servidor_RPY(float32_t *Roll, float32_t *Pitch, float32_t *Yaw){  IArg Key;

    Key = GateMutexPri_enter(SERVIDOR_YPR);
    Roll_Almacenado = *Roll;
    Pitch_Almacenado = *Pitch;
    Yaw_Almacenado = *Yaw;
    GateMutexPri_leave(SERVIDOR_YPR, Key);
}

void Leer_servidor_quaternios(float32_t* q){
    IArg Key;

    Key = GateMutexPri_enter(SERVIDOR_Quaternio);
    arm_copy_f32(q_almacenado, q, sizeof(q_almacenado)/sizeof(float32_t));
    GateMutexPri_leave(SERVIDOR_Quaternio, Key);
}
void Escribir_servidor_quaternios(float32_t* q){
    IArg Key;

    Key = GateMutexPri_enter(SERVIDOR_Quaternio);
    arm_copy_f32(q, q_almacenado, sizeof(q_almacenado)/sizeof(float32_t));
    GateMutexPri_leave(SERVIDOR_Quaternio, Key);
}


void Leer_servidor_Variables_Estado_Medidas(float32_t* Variables_Estado_Medidas){
    IArg Key;

    Key = GateMutexPri_enter(SERVIDOR_Variables_Estado_Medidas);
    arm_copy_f32(Variables_Estado_Medidas_Almacenadas, Variables_Estado_Medidas,
    sizeof(Variables_Estado_Medidas_Almacenadas)/sizeof(float32_t));
//  memcpy(Variables_Estado_Medidas, Variables_Estado_Medidas_Almacenadas,
sizeof(Variables_Estado_Medidas_Almacenadas));
    GateMutexPri_leave(SERVIDOR_Variables_Estado_Medidas, Key);
}

void Escribir_servidor_Variables_Estado_Medidas(float32_t* Variables_Estado_Medidas){
    IArg Key;

    Key = GateMutexPri_enter(SERVIDOR_Variables_Estado_Medidas);
    memcpy(Variables_Estado_Medidas_Almacenadas, Variables_Estado_Medidas,
    sizeof(Variables_Estado_Medidas_Almacenadas));
    GateMutexPri_leave(SERVIDOR_Variables_Estado_Medidas, Key);
}

void Leer_servidor_Variables_Estado_Estimadas(float32_t* Variables_Estado_Estimadas){
    IArg Key;

    Key = GateMutexPri_enter(SERVIDOR_Variables_Estado_Estimadas);
    memcpy(Variables_Estado_Estimadas, Variables_Estado_Estimadas_Almacenadas,
    sizeof(Variables_Estado_Estimadas_Almacenadas));
    GateMutexPri_leave(SERVIDOR_Variables_Estado_Estimadas, Key);
}

void Escribir_servidor_Variables_Estado_Estimadas(float32_t* Variables_Estado_Estimadas){
    IArg Key;

    Key = GateMutexPri_enter(SERVIDOR_Variables_Estado_Estimadas);
    memcpy(Variables_Estado_Estimadas_Almacenadas, Variables_Estado_Estimadas,
    sizeof(Variables_Estado_Estimadas_Almacenadas));
    GateMutexPri_leave(SERVIDOR_Variables_Estado_Estimadas, Key);
}

void Leer_servidor_Perturbaciones_Estimadas(float32_t* Perturbaciones_Estimadas){
    IArg Key;
```

```
    Key = GateMutexPri_enter(SERVIDOR_Perturbaciones_Estimadas);
    arm_copy_f32(Perturbaciones_Estimadas_Almacenadas, Perturbaciones_Estimadas,
    sizeof(Perturbaciones_Estimadas_Almacenadas)/sizeof(float32_t));
    GateMutexPri_leave(SERVIDOR_Perturbaciones_Estimadas, Key);
}

void Escribir_servidor_Perturbaciones_Estimadas(float32_t* Perturbaciones_Estimadas){
    IArg Key;

    Key = GateMutexPri_enter(SERVIDOR_Perturbaciones_Estimadas);
    arm_copy_f32(Perturbaciones_Estimadas, Perturbaciones_Estimadas_Almacenadas,
    sizeof(Perturbaciones_Estimadas_Almacenadas)/sizeof(float32_t));
    GateMutexPri_leave(SERVIDOR_Perturbaciones_Estimadas, Key);
}

void Resetear_servidor_Perturbaciones_Estimadas(){
    IArg Key;

    Key = GateMutexPri_enter(SERVIDOR_Perturbaciones_Estimadas);
    arm_fill_f32(0, Perturbaciones_Estimadas_Almacenadas,
    sizeof(Perturbaciones_Estimadas_Almacenadas)/sizeof(float32_t));
    GateMutexPri_leave(SERVIDOR_Perturbaciones_Estimadas, Key);
}

void Leer_servidor_Referencia(float32_t* Referencia , int16_t* Referencia_Entero){
    IArg Key;

    Key = GateMutexPri_enter(SERVIDOR_Referencia);
    memcpy(Referencia, Referencia_Almacenada, sizeof(Referencia_Almacenada));
    memcpy(Referencia_Entero, Referencia_Almacenada_Entero,
    sizeof(Referencia_Almacenada_Entero));
    GateMutexPri_leave(SERVIDOR_Referencia, Key);
}

void Escribir_servidor_Referencia(float32_t* Referencia , int16_t* Referencia_Entero){
    IArg Key;

    Key = GateMutexPri_enter(SERVIDOR_Referencia);
    memcpy(Referencia_Almacenada, Referencia, sizeof(Referencia_Almacenada));
    memcpy(Referencia_Almacenada_Entero, Referencia_Entero,
    sizeof(Referencia_Almacenada_Entero));
    GateMutexPri_leave(SERVIDOR_Referencia, Key);
}

float32_t* Direccion_servidor_Referencia(){
    return Referencia_Almacenada;
}
```

```c
/*
 * Transmisores.h
 *
 *  Created on: 5/10/2015
 *      Author: Ruben_User
 */

#ifndef QUADROTOR_V9_2_TRANSMISORES_H_
#define QUADROTOR_V9_2_TRANSMISORES_H_

#include <ti/drivers/SPI.h>
#include "Parametros.h"

typedef struct{
    SPI_Handle SPI;
    unsigned int PIN_CE;
    unsigned int PIN_CSN;
    unsigned int PIN_IRQ;
}tp_nRF24L01;

bool Iniciar_nRF24L01(tp_nRF24L01 nRF24L01, uint8_t RX, uint8_t Canal, uint8_t Payload);
void Activar_nRF24L01(tp_nRF24L01 nRF24L01);
void Desactivar_nRF24L01(tp_nRF24L01 nRF24L01);

bool Flush_nRF24L01(tp_nRF24L01 nRF24L01, uint8_t RX);

bool Escribir_Registro_nRF24L01(tp_nRF24L01 nRF24L01, uint8_t Reg, uint8_t Dato);
bool Leer_Registro_nRF24L01(tp_nRF24L01 nRF24L01, uint8_t Reg, uint8_t* Dato);

//bool WritePayLoad_nRF24L01(tp_nRF24L01 nRF24L01, uint8_t* PayLoad, size_t n_PayLoad );
//bool ReadPayLoad_nRF24L01(tp_nRF24L01 nRF24L01, uint8_t* PayLoad, size_t n_PayLoad );

bool MandarByte_nRF24L01(tp_nRF24L01 nRF24L01, uint8_t Dato);
bool RecibirByte_nRF24L01(tp_nRF24L01 nRF24L01, uint8_t* Dato);

#endif /* QUADROTOR_V9_2_TRANSMISORES_H_ */
```

```c
/*
 * Transmisores.c
 *
 *  Created on: 7/10/2015
 *      Author: Ruben_User
 */

#include <ti/drivers/SPI.h>
#include <ti/sysbios/Knl/Clock.h>
#include "Transmisores.h"
#include "Parametros.h"
#include <ti/drivers/GPIO.h>

bool Iniciar_nRF24L01(tp_nRF24L01 nRF24L01, uint8_t RX, uint8_t Canal, uint8_t Payload){
    uint8_t Dato;
    uint32_t systick = 0;
    bool Transmision_OK;

    Transmision_OK = Escribir_Registro_nRF24L01(nRF24L01, 0x00 , (0b00001010 | (0x01 &
    RX))); // Inicio Transmisor, CRC, NO Int
    systick = Clock_getTicks();
    while(Clock_getTicks() < systick+2);
    Transmision_OK &= Escribir_Registro_nRF24L01(nRF24L01, 0x05 , Canal);
    Transmision_OK &= Escribir_Registro_nRF24L01(nRF24L01, 0x11 , Payload);
    Transmision_OK &= Flush_nRF24L01(nRF24L01, true);
    Transmision_OK &= Leer_Registro_nRF24L01(nRF24L01, 0x00, &Dato);

     return (Transmision_OK);
}

void Activar_nRF24L01(tp_nRF24L01 nRF24L01){
    GPIO_write(nRF24L01.PIN_CE, 1);
}

void Desactivar_nRF24L01(tp_nRF24L01 nRF24L01){
    GPIO_write(nRF24L01.PIN_CE, 0);
}

bool Flush_nRF24L01(tp_nRF24L01 nRF24L01, uint8_t RX){
    SPI_Transaction SPI_Transmision;
    uint8_t Buffer_Tx = 0xE1;
    bool Transmision_OK;

    Buffer_Tx += RX;

    SPI_Transmision.count = 1;
    SPI_Transmision.txBuf = &Buffer_Tx;
    SPI_Transmision.rxBuf = NULL;

    GPIO_write(nRF24L01.PIN_CSN, 0);
    Transmision_OK = (SPI_transfer(nRF24L01.SPI, &SPI_Transmision));
    GPIO_write(nRF24L01.PIN_CSN, 1);

    return(Transmision_OK);
}

bool Escribir_Registro_nRF24L01(tp_nRF24L01 nRF24L01, uint8_t Reg, uint8_t Dato){
    SPI_Transaction SPI_Transmision;
    uint8_t Buffer_Tx[2];
    bool Transmision_OK;
    uint8_t Estado_Anterior;

    Estado_Anterior = GPIO_read(nRF24L01.PIN_CE);

    Buffer_Tx[0] = 0x20 | Reg;
    Buffer_Tx[1] = Dato;

    SPI_Transmision.count = 2;
```

```
    SPI_Transmision.txBuf = Buffer_Tx;
    SPI_Transmision.rxBuf = NULL;

    GPIO_write(nRF24L01.PIN_CE, 0);
    GPIO_write(nRF24L01.PIN_CSN, 0);
    Transmision_OK = (SPI_transfer(nRF24L01.SPI, &SPI_Transmision));
    GPIO_write(nRF24L01.PIN_CSN, 1);
    GPIO_write(nRF24L01.PIN_CE, Estado_Anterior);

    return(Transmision_OK);
}

bool Leer_Registro_nRF24L01(tp_nRF24L01 nRF24L01, uint8_t Reg, uint8_t* Dato){
    SPI_Transaction SPI_Transmision;
    bool Transmision_OK;
    uint8_t Buffer_Tx[2] = {0, 0};
    uint8_t Buffer_Rx[2];

    Buffer_Tx[0] = (Reg & 0x1F);

    SPI_Transmision.count = 2;
    SPI_Transmision.txBuf = Buffer_Tx;
    SPI_Transmision.rxBuf = Buffer_Rx;

    GPIO_write(nRF24L01.PIN_CSN, 0);
    Transmision_OK =  SPI_transfer(nRF24L01.SPI, &SPI_Transmision);
    GPIO_write(nRF24L01.PIN_CSN, 1);
    *Dato = Buffer_Rx[1];

    return(Transmision_OK);
}
/*
bool WritePayLoad_nRF24L01(tp_nRF24L01 nRF24L01, uint8_t* PayLoad, size_t n_PayLoad){
    SPI_Transaction SPI_Transmision;

    PayLoad[0] = 0b10100000;
    SPI_Transmision.count = n_PayLoad;
    SPI_Transmision.txBuf = PayLoad;
    SPI_Transmision.rxBuf = NULL;

    GPIO_write(nRF24L01.PIN_CSN, 0);
    return (SPI_transfer(nRF24L01.SPI, &SPI_Transmision));
    GPIO_write(nRF24L01.PIN_CSN, 1);
}

bool ReadPayLoad_nRF24L01(tp_nRF24L01 nRF24L01, uint8_t* PayLoad, size_t n_PayLoad){
    SPI_Transaction SPI_Transmision;

    PayLoad[0] = 0b01100001;
    SPI_Transmision.count = n_PayLoad;
    SPI_Transmision.rxBuf = PayLoad;
    SPI_Transmision.txBuf = NULL;

    GPIO_write(nRF24L01.PIN_CSN, 1);
    return (SPI_transfer(nRF24L01.SPI, &SPI_Transmision));
}
*/
bool MandarByte_nRF24L01(tp_nRF24L01 nRF24L01, uint8_t Dato){
    SPI_Transaction SPI_Transmision;
    bool Transmision_OK;
    uint8_t Tx[2] = {0b10100000, 0};

    Tx[1] = Dato;
    GPIO_write(nRF24L01.PIN_CSN, 0);

    SPI_Transmision.count = 2;
    SPI_Transmision.txBuf = Tx;
    SPI_Transmision.rxBuf = NULL;
```

```
    GPIO_write(nRF24L01.PIN_CSN, 0);

    Transmision_OK =  SPI_transfer(nRF24L01.SPI, &SPI_Transmision);

    GPIO_write(nRF24L01.PIN_CSN, 1);

    return(Transmision_OK);
}

bool RecibirByte_nRF24L01(tp_nRF24L01 nRF24L01, uint8_t* Dato){
    SPI_Transaction SPI_Transmision;
    bool Transmision_OK;
    uint8_t Tx[2] = {0b01100001, 0};
    uint8_t Rx[2] = {0, 0};

    SPI_Transmision.count = 2;
    SPI_Transmision.txBuf = Tx;
    SPI_Transmision.rxBuf = Rx;

    GPIO_write(nRF24L01.PIN_CSN, 0);

    Transmision_OK =  SPI_transfer(nRF24L01.SPI, &SPI_Transmision);
    *Dato = Rx[1];

    GPIO_write(nRF24L01.PIN_CSN, 1);

    return(Transmision_OK);
}
```