DISEÑO E IMPLEMENTACIÓN DE UNA PLATAFORMA MULTIROTOR BASADA EN ARDUINO

Antonio Ventosa Cutillas

Documento creado con LaTeX

A mis padres,
A mi hermano,
A mi abuela.

A Manuel Vargas y Manuel Gil, mis tutores, por guiarme, brindarme la oportunidad de realizar este proyecto y por su desinteresada y magnífica colaboración en el transcurso de este trabajo.

A mi familia, mis padres, mi hermano y mi abuela, por apoyarme en todo momento.

A María José, por estar conmigo.

A mis amigos, "Los Limonchelos", Fiti, Rocío, Julio, Inma, Juan, Bea, David, Ana, Nachito, Juanki, por estar siempre ahí, en los buenos y malos momentos.

A mis compañeros del departamento, Jesús, Guillermo, Ramón y Manolo, por ayudarme con todas mis dudas y trabajar siempre con una sonrisa, haciendo que ir a trabajar sea divertido.

Gracias a todos,

Antonio Ventosa Cutillas

Índice general

Índice de figuras

Lista de abreviaturas

AHRS	Attitude and Heading Reference Systems o Sistemas de Referencia de Actitud y Rumbo
GPS	Global Positioning System
IMU	Inertial Measurement Unit
PD	Proporcional - Derivativo
PI	Proporcional - Integral
PID	Proporcional - Integral - Derivativo
PWM	Pulse Width Modulation
UAV	Unmanned Aerial Vehicle

Capítulo 1

Introducción

1.1. Historia

Un quadrotor o cuadricóptero se puede definir como una aeronave que se eleva y se desplaza por el movimiento de cuatro motores colocados en los extremos de una estructura en forma de cruz. Dispone de cuatro motores con sus respectivas palas dispuestas en el mismo plano y cuyo ángulo permanece fijo durante el movimiento. En estos vehículos se varía la velocidad de los motores para controlar la estabilidad y realizar movimientos.

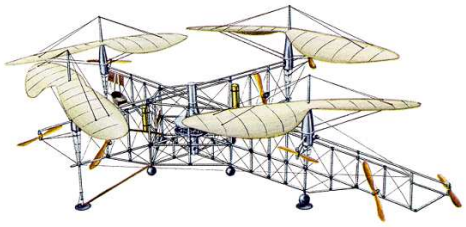

Figura 1.1: Oehmichen No.2.

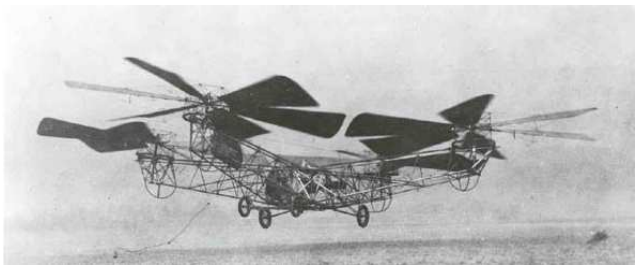

Figura 1.2: Bothezat.

El origen del quadrotor se puede establecer en la decada de 1920, cuando Etienne Oehmichen experimentó con diseños de helicópteros. Entre ellos se encontraba el Oehmichen No.2, un helicóptero que tenía cuatro rotores y ocho hélices, todos impulsados por un solo motor. El Oehmichen No.2 utilizaba rotores bipala en los extremos de los cuatro brazos. El ángulo de estas cuchillas se podía variar por deformación. Cinco de las hélices, girando en el plano horizontal, estabilizaban la máquina lateralmente. Otra hélice montada en la nariz se utilizaba para la dirección. El par restante de hélices fueron para la propulsión hacia adelante. La aeronave presentaba un alto grado de estabilidad y capacidad de control para su época.

En 1922, el Dr. George de Bothezat e Ivan Jerome desarrollaron un vehículo con cuatro rotores de seis palas en el extremo de una estructura en forma de X. Se utilizaron dos pequeñas hélices de paso variable para controlar la orientación y el ángulo de guiñada. A pesar de su viabilidad tenía poca potencia, sólo se levantaba cinco metros, era mecánicamente complejo y susceptible de problemas de fiabilidad.

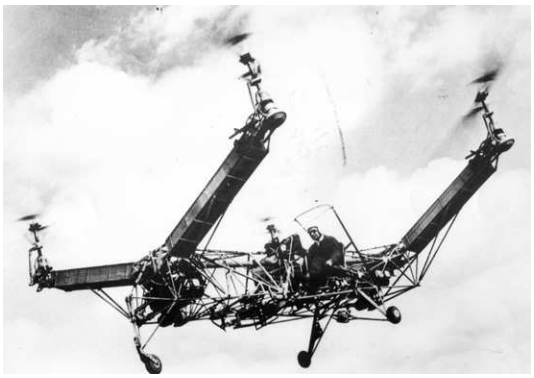

Figura 1.3: Convertawings Model A Quadrotor.

Figura 1.4: Curtiss-Wright VZ-7.

La empresa Convertawings desarrolló en 1956 un prototipo de quadrotor civil y militar de mayor tamaño que disponía de dos motores que accionaban cuatro rotores. Este modelo no utilizaba rotor

de cola y el control se obtenía mediante la variación de la velocidad de cada rotor, estableciéndose de esta forma el diseño de los actuales quadrotor.

Dos años más tarde la compañía Curtis-Wright diseñó un quadrotor de uso militar para el ejército americano, el Curtiss-Wright VZ-7. Se trataba también de un vehículo aéreo cuyo control se obtenía mediante la variación de la velocidad de cada rotor.

Tras el VZ-7, el ejército americano siguió investigando el uso de los quadrotor como vehículo militar, realizando contratos con compañías como Chrysler con el VZ-6 o el VZ-8 de Airgeep.

En la actualidad el ejercito de Estados Unidos está buscando aeronaves con gran capacidad de transporte de pasajeros y armamento, fruto de ello es el proyecto en desarrollo con la empresa Boeing, el Bell Boeing Quad TiltRotor.

Figura 1.5: Bell Boeing Quad TiltRotor.

Además de los desarrollos militares, diversas empresas han desarrollado quadrotors de menor tamaño usados como aparatos de radiocontrol o para tareas de vigilancia. Un ejemplo de quadrotor de radiocontrol es el AR. de Parrot o el Arducopter, en el que se basa el quadrotor realizado para este proyecto, desarrollado por la comunidad de internet DIY Drones.

Figura 1.6: Parrot AR-Drone.

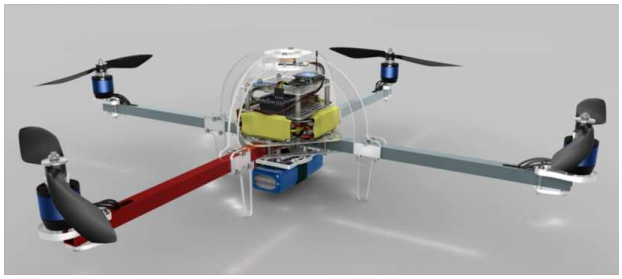

Figura 1.7: Arducopter.

1.2. Objetivo

El objetivo del presente proyecto es documentar todo el proceso realizado para el montaje y programación de un quadrotor, para el cual se ha tomado como referencia un proyecto creado por la comunidad DIY DRONES basado en la plataforma Arduino. Este proceso comenzó con la compra del material necesario para el montaje, el cual se describe en el apartado 2. Una vez obtenido el material, se pasó a realizar experimentos para comprobar el funcionamiento de todas las partes. También se comprobó el funcionamiento una vez realizado el montaje completo y con la programación especificada en el proyecto de DIY DRONES. Una vez comprobado esto, se pasó a programar el quadrotor con un código específico realizado en el laboratorio, aunque basado en el original.

Todos estos experimentos realizados y sus resultados son los que se tratan de exponer en este Proyecto Fin de Carrera, el cual se estructura de la siguiente manera:

En el primer capítulo se hará una introducción. Ya se ha hablado sobre la historia de los quadrotor y más adelante se describirán los usos, ventajas y desventajas, así como el funcionamiento mecánico del mismo.

En el capítulo dos se describen todos los componentes utilizados en el montaje del quadrotor. En esta parte no se ha tenido en cuenta la estructura, la cual está compuesta por varillas de aluminio y placas de metacrilato. Esta estructura se ha realizado de forma artesanal.

En el capítulo tres se presentará el diseño realizado, complementando las fotos del quadrotor con planos en los que se indican las medidas más importantes. También se tratará en este caítulo sobre diversos sistemas de control implementados en el quadrotor. En este apartado hay que tener en cuenta la estructura del quadrotor, comentada anteriormente. Como se dijo, esta estructura se ha realizado de forma artesanal, por lo que se ha diseñado un sistema de control suficientemente robusto, que sea capaz de compensar las pequeñas asimetrías que pueda presentar ésta.

En el cuarto capítulo se estudiará un sistema de posicionamiento por ultrasonidos. Este sistema responde a la necesidad de posicionar el quadrotor en un punto fijo durante el vuelo para poder ajustar de forma precisa los controladores de estabilización de los ángulos. Al momento de la escritura de este documento, este sistema de posicionamiento aún se encuentra en desarrollo.

En el último capítulo se establecerán las conclusiones obtenidas en el desarrollo de este proyecto y propondré algunas posibles mejoras, algunas de ellas ya en desarrollo.

Como anexo a estos capítulos se encuentra el código de programación del microcontrolador presente en el sistema de posicionamiento por ultrasonidos.

1.3. Usos, ventajas y desventajas

Los principales usos que tienen actualmente los quadrotor son militares o policiales, desarrollos para investigación académica y usos comerciales.

En el aspecto militar el principal uso que tienen las plataformas tipo quadrotor es la vigilancia y observación de zonas de difícil acceso. Esta es una gran ventaja en este tipo de aparatos, puesto que son capaces de realizar tareas difíciles o peligrosas sin poner en riesgo la vida de seres humanos. Otro uso militar se comentó en el apartado primero, donde la empresa Boeing está desarrollando un quadrotor capaz de desplazar gran cantidad de material o personas.

Figura 1.8: Fotograma de un video realizado por la policía mediante un quadrotor durante los disturbios en Estambul de junio de 2013.

Desde el punto de vista comercial, los principales usos de los quadrotor son como aparatos de radio-control para aficionados y como aparatos de vigilancia para empresas de seguridad debido a la gran capacidad de maniobra que presentan. A parte de estos usos también se está trabajando en el uso de quadrotor para realizar grabaciones de películas y retransmisiones de eventos deportivos.

En cuanto al uso académico, se están realizando multitud de proyectos de investigación en torno a los quadrotor puesto que en estos se pueden probar una gran variedad de aspectos relacionados con la ingeniería. Entre estos aspectos se encuentran la ingeniería eléctrica, de control, mecánica o sistemas de navegación entre otros.

El uso de los quadrotor está en auge debido a las ventajas que presentan respecto a los helicópteros tradicionales ya que los quadrotor no requieren conexiones mecánicas para variar el ángulo de las palas del rotor a medida que giran. Esto simplifica el diseño y mantenimiento del aparato. Además, el uso de cuatro rotores permite que cada rotor pueda tener un diámetro menor que el rotor de helicóptero equivalente, lo que les permite tener menos energía cinética durante el vuelo. Esto reduce el daño causado en caso de que los rotores golpeen algo. Otra ventaja de los quadrotor es que son relativamente baratos y están disponibles en una gran variedad de tamaños.

Como desventaja, los quadrotor requieren un gasto eléctrico elevado al poseer cuatro motores. Esto hace que la autonomía de vuelo sea relativamente corta. Es posible aumentar dicha autonomía a costa de un aumento de peso del conjunto debido a las baterías.

1.4. Descripción del funcionamiento

Como se ha comentado al comienzo del capítulo, las plataformas tipo quadrotor disponen de cuatro motores coplanarios en los que las hélices tienen un ángulo de ataque fijo. Con esta disposición es necesario que dos motores giren en sentido horario y los dos opuestos lo hagan en sentido antihorario. Esto es debido a los pares de rotación que generan los motores, de forma que el par generado por los motores que giran en un sentido pueda ser contrarrestado por los motores que giran en el sentido contrario. El sentido de giro de las hélices se puede observar en el esquema de la figura 1.9.

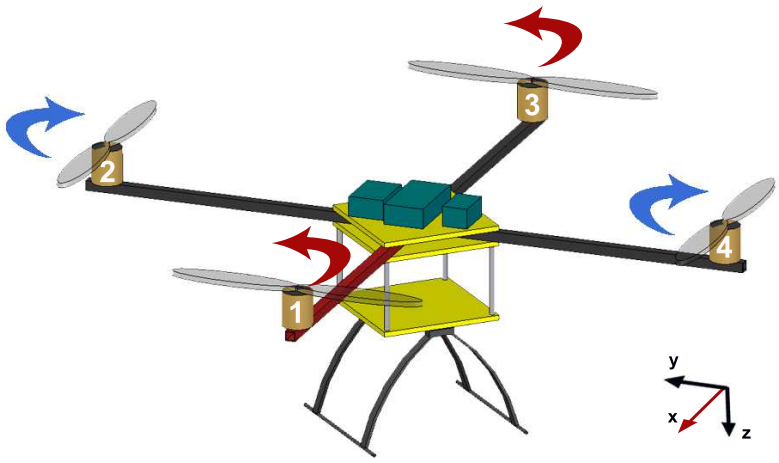

Figura 1.9: Sentido de giro de las hélices.

Los quadrotor, al no variar la posición de los motores y las hélices, realizan los movimientos de rotación y traslación aumentando o disminiyendo la velocidad de algún motor. Para realizar esta operación se describen una serie de ángulos de referencia o ángulos de navegación. Los ángulos de navegación, llamados en matemáticas ángulos de Tait-Bryan, son tres coordenadas angulares que definen un triedro rotado desde otro que se considera el sistema de referencia. Estos ángulos son:

- **Cabeceo (Pitch)**

  Es el ángulo que se obtiene cuando se produce una rotación en torno all eje formado por los motores 2 y 4 (eje Y) de la figura 1.9, por lo que para modificar este ángulo será necesario actuar sobre los motores 1 y 3. Como consecuencia de la variación de este ángulo se obtiene un movimiento a través de la proyección horizontal del eje formado por los motores 1 y 3.

- **Alabeo (Roll)**

  Es el ángulo que se obtiene cuando se produce una rotación en torno al eje formado por los motores 1 y 3 (eje X), por lo que para modificar este ángulo será necesario actuar sobre los motores 2 y 4. Como consecuencia de la variación de este ángulo se obtiene un movimiento a través de la proyección horizontal del eje formado por los motores 2 y 4.

- **Guiñada (Yaw)**

  Es el ángulo que se obtiene cuando el quadrotor realiza una rotación alrededor de un eje vertical perpendicular al plano formado por los motores (eje Z). Para modificar este ángulo se puede

actuar sobre cualquier motor, puesto que un desequilibrio en los pares de rotación producidos por los motores supondrá un movimiento de giro en torno al eje Z.

La modificación de estos ángulos puede deberse a dos motivos, por control de estabilización o control de desplazamiento lateral.

En el primer caso, lo que se pretende obtener es mantener el quadrotor totalmente horizontal, de forma que no se produzca ningún movimiento lateral. Es decir, se actúa sobre los motores para poner a cero la medida del ángulo de navegación que difiera de dicho valor.

En el segundo caso ocurre lo contrario. Lo que se desea es crear un ángulo distinto de cero para producir un desplazamiento lateral del quadrotor. Estos desplazamientos se consiguen actuando sobre el roll y el pitch de la siguiente forma:

- **Movimiento en el eje X**

  Se consigue aumentando o disminuyendo la velocidad de los motores 3 y 4. En el caso de la figura 1.10, si se aumenta la velocidad del motor 3 y/o disminuye la velocidad del motor 4, se produce una modificación del pitch dando un valor positivo del ángulo y produciendo un desplazamiento en sentido negativo del eje X.

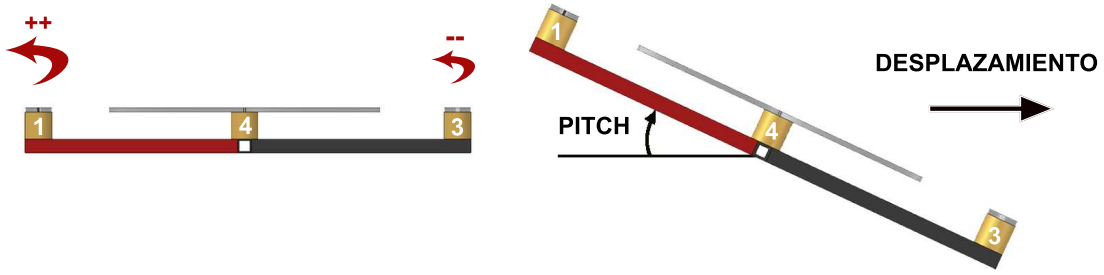

Figura 1.10: Desplazamiento en el eje X.

- **Movimiento en el eje Y**

  Se consigue aumentando o disminuyendo la velocidad de los motores 1 y 2. En el caso de la figura 1.11, si se aumenta la velocidad del motor 1 y/o disminuye la velocidad del motor 2, se produce una modificación del roll dando un valor negativo del ángulo y produciendo un desplazamiento en sentido negativo del eje Y.

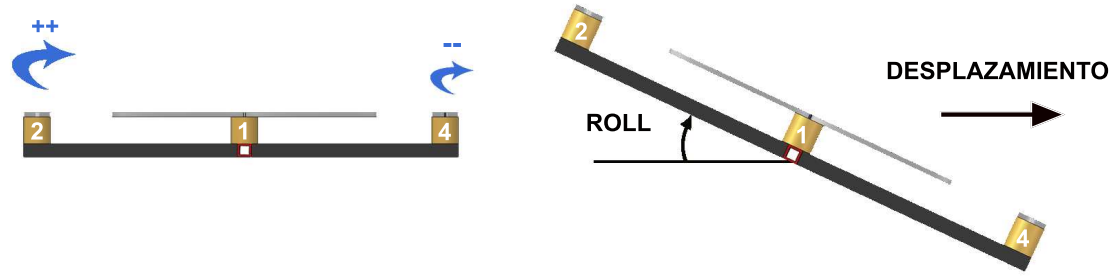

Figura 1.11: Desplazamiento en el eje Y.

Capítulo 2

Descripción de las partes que componen el quadrotor

En este capítulo se va a realizar una breve descripción de todos los elementos físicos que componen el quadrotor, es decir, toda la parte hardware del quadrotor. Estos componentes se dividen en motores, variadores, placa de control Ardupilot, sensores, sistema de comunicación y baterias.

2.1. Motores

Figura 2.1: Motor Axi 2217/16 Gold Line.

2.1.1. Descripción

Los motores utilizados en el quadrotor son de la marca Model Motors, concretamente el modelo Axi 2217/16 Gold Line. Se trata de un motor brushless trifásico de 12 polos, con una relación de velocidad de 1050 RPM/V. Este motor pesa 69,5g por lo que es ideal para su uso en vehículos aéreos, ya que es capaz de producir un empuje cercano a los 13 Newton con un peso tan reducido. Puede soportar una corriente máxima de 22 amperios durante 60 segundos, siendo la corriente de funcionamiento entre 10 y 18 amperios. Con este rango de corriente se consigue obtener un rendimiento del 83 %. Todas las características de estos motores se detallan en la siguiente tabla.

Alimentación	2s y 3s Lipo
RPM/V	1050
Eficiencia máxima	83 %
Corriente para eficiencia máxima	10-18A
Corriente máxima	22A/60s
Resistencia interna	120mΩ
Dimensiones	27,7x35 mm
Diámetro del eje	3,17 mm
Peso	69,5g

Tabla 2.1: Características de los motores Axi 2217/16 Gold Line

En estos motores se han montado hélices de composite de 10”x 4.5”siguiendo las recomendaciones del fabricante.

Para conocer las características de los motores empleados es necesario realizar una serie de experimentos. El proceso utilizado para realizar estos experimentos y sus resultados se observan en las Curvas Características que se comentan a continuación.

2.1.2. Curvas Características

La curvas características son un conjunto de curvas que representan las relaciones existentes entre las distintas variables de explotación de los motores. Las más usuales son las curvas Par - Intensidad, Velocidad - Intensidad y Velocidad - Par. Dado que en nuestro caso no medimos intensidad sino porcentaje de actuación de la señal de control que se envía a los motores, y tampoco medimos el par sino el empuje que producen los motores, las curvas representadas serán Empuje - Acción de control, Velocidad - Acción de control y Velocidad - Empuje.

Curva Caraterística Estática

La curva caraterística estática es la curva que relaciona las variables del motor cuando éste se encuentra en régimen permanente.

- **Curva Empuje - Acción de control**

  Para obtener los resultados de esta gráfica, se va aumentando la acción de control y se mide el par de empuje generado por el motor.

  Podemos observar, tanto en esta gráfica como en las siguientes, como el porcentaje de la acción de control sólo alcanza el 90 % puesto que a partir de ese punto se alcanza la máxima intensidad que puede soportar el motor.

  En estas gráficas también se puede observar la aproximación de la curva con una serie de polinomios de segundo, tercer y cuarto grado.

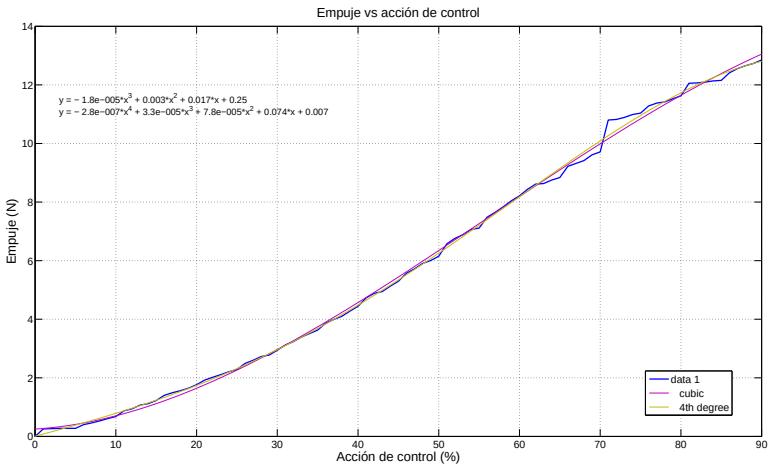

Figura 2.2: Curva Empuje - Acción de control.

- **Curva Velocidad - Acción de control**

  En esta gráfica se va aumentando la acción de control y se mide la velocidad de giro de los motores. Este valor de la velocidad se obtiene mediante una medida generada por los variadores en una unidad desconocida, la cual no es necesario conocer puesto que la referencia utilizada a la hora de controlar el quadrotor es el valor del empuje[1]. Una vez establecida esta referencia en empuje, se realiza un cálculo interno mediante la ecuación de cuarto grado obtenida en la gráfica 15, obteniéndose el valor de la velocidad necesaria en cada motor.

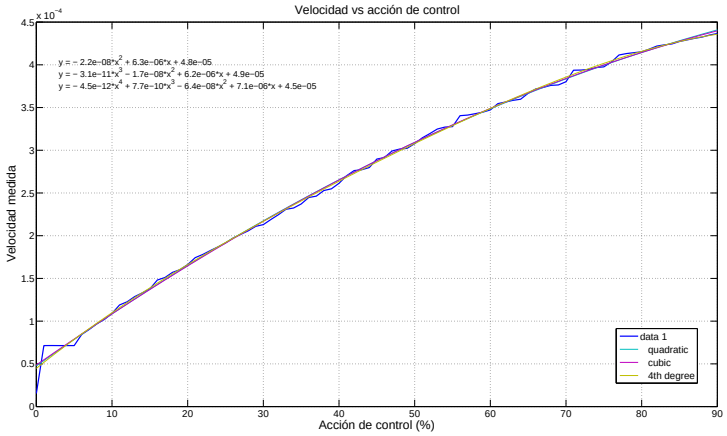

Figura 2.3: Curva Velocidad - Acción de control.

- **Curva Velocidad - Empuje**

  Esta curva se obtiene al fusionar los datos de las dos curvas anteriores y es la que realmente nos interesa puesto que es la que se utiliza para obtener la referencia de velocidad que se envía a los motores a partir del empuje necesario en cada momento.

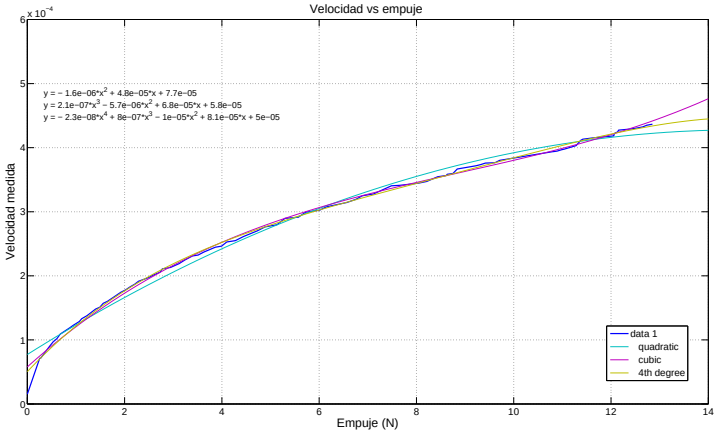

Figura 2.4: Curva Velocidad - Empuje.

[1]Se puede calcular el valor de la velocidad en rpm teniendo en cuenta que se realiza un control trapezoidal de la velocidad utilizando pulsos de tensión contraelectromotriz.

Curva Característica Dinámica

La curva característica dinámica es la curva que relaciona las variables del motor cuando este se encuentra en régimen transitorio. Para ello se somete al sistema a una excitación de tipo escalón al 65 % de acción de control y se mide la velocidad de los motores, obteniéndose las gráficas siguientes. En la tercera gráfica se representa la respuesta al escalón y la curva aproximada que se ha utilizado para realizar los cálculos.

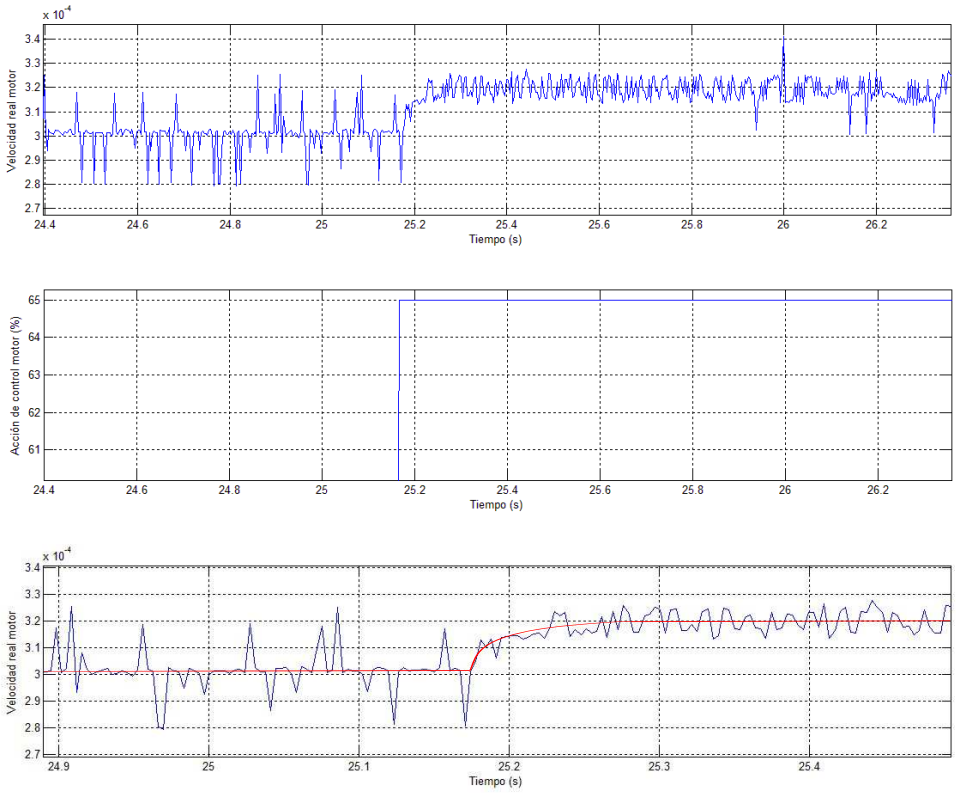

Figura 2.5: Respuesta ante escalón y aproximación de la curva.

A partir de estas gráficas se obtienen los siguientes parámetros:

- **Ganancia estática**

  Se denomina ganancia estática de un sistema lineal estable a la relación entre su salida y su entrada cuando ambas se han estabilizado.

  $K = 4 * 10{-}6$

- **Tiempo de subida**

  Es el tiempo que se tarda en pasar del 10 % al 90 % del valor en régimen permanente.

  $t_r = 0{,}0522$ s

- **Tiempo de establecimiento**

  Es el tiempo que tarda el motor en alcanzar el régimen permanente.

  $t_s = 0{,}087$ s

- **Constante de tiempo**

  Es el tiempo que tarda el motor en alcanzar el 63 % del valor en régimen permanente.

  $\tau = 0{,}01$ s

- **Tiempo de retardo**

  Es el tiempo que transcurre desde que se envía la señal de control hasta que el motor responde.

  $t_d = 0{,}009$ s

2.1.3. Zonas de funcionamiento no lineal

- **Zona de saturación** Dado que por seguridad no se ha sobrepasado el 90 % de la acción de control, se ha asumido que es ese punto donde los motores alcanzan la zona de saturación. Si se observan las gráficas 2.2 y 2.3 se puede ver que este punto se alcanza para unos valores de 12.83 N y $4.36{\cdot}10^{-4}$ unidades de velocidad del variador, que como se comentó anteriormente es una unidad de medida desconocida.

- **Zona muerta**

  Al ir aplicando la señal de control al motor, éste no comienza a moverse hasta que se alcanza un valor cercano al 5 % de la acción de control, por lo que se considera toda la zona anterior a dicho valor como la zona muerta. Si observamos la figura 2.6, se observa que la lectura de velocidad no es cero. Este es un valor mínimo que generan los sensores que equivale a cero puesto que físicamente los motores no se mueven.

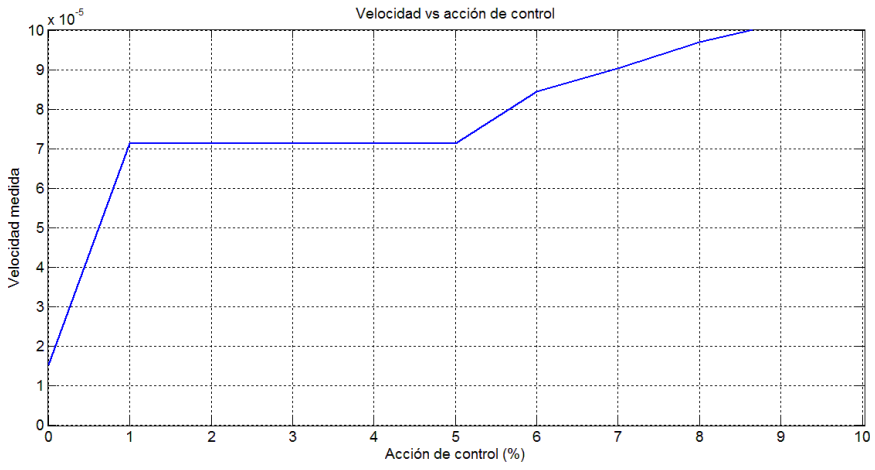

Figura 2.6: Zona muerta (Velocidad frente acción de control).

2.2. Variadores

Figura 2.7: Variador YGE-30i.

El variador es un elemento que convierte la corriente continua de la batería en un patrón de frecuencia y voltaje variables para controlar la velocidad del motor, dado que se están usando motores brushless.

Los variadores utilizados son de la marca YGE. La característica principal de los variadores montados en el quadrotor es que se comunican con la placa de control a través del bus I^2C, además de la posibilidad de utilizar una entrada PWM. El interés en usar comunicación por I^2C responde a dos aspectos principalmente, es una comunicación más rápida que por PWM y es bidireccional, por lo que se puede obtener información de la velocidad de los motores a través de los variadores.

El modelo de variadores utilizado es el YGE-30i cuyas características se muestran en la siguiente tabla:

Alimentación	2s a 4s Lipo
Corriente máxima	30A
Protección por sobretemperatura	SI
Protección por sobrecorriente	SI
I^2C compatible	SDA cable rojo. SCL cable negro
Dimensiones	42 x 24 x 6 mm
Peso	11g (sin cables)

Tabla 2.2: Características de los variadores YGE-30i

Estos variadores tienen un sistema para mostrar fallos mediante el parpadeo de un led. El código para mostrar dichos fallos es el siguiente:

- 2 parpadeos: Fallo por bajo voltaje de alimentación.
- 3 parpadeos: Fallo por sobre temperatura.
- 4 parpadeos: Fallo por sobre corriente.
- 5 parpadeos: El variador no recibe señal de control.
- 6 parpadeos: Fallo al iniciar.

En condición normal de funcionamiento el led permanece fijo en situación de espera y parpadea una vez cuando está operando correctamente.

2.3. Inertial Measurement Unit (IMU)

La unidad de medición inercial o IMU (del inglés *Inertial Measurement Unit*) es un dispositivo electrónico que usa acelerómetros y giróscopos para obtener valores de orientación, velocidad, aceleración y fuerzas de gravedad entre otros.

La IMU usada en el quadrotor es el modelo foxtrap 2.2 de DIYDrones, se encuentra integrada en una placa junto a otros sensores y componentes y posee las siguientes características:

- Regulador dual de 3.3V para sensores analógicos.
- Relé para cámaras, luces y otras cargas.
- 12 bit ADC para mejorar la resolución de los sensores.
- Logger integrado de 16 Mb de memoria flash.
- Chip FTDI para soporte de conexión USB.
- Bus I^2C.
- Puertos de expansión analógicos de 10 bits.
- Botón de reset.
- Leds de status.
- Giróscopo de tres ejes resistente a las vibraciones.
- Acelerómetro de tres ejes ADX330.
- Sensor de presión Bosch.
- Magnetómetro de tres ejes.

2.3.1. Situación de los sensores

La situación de alguno de los sensores y dispositivos comentados de la IMU se indican en la figura 2.8:

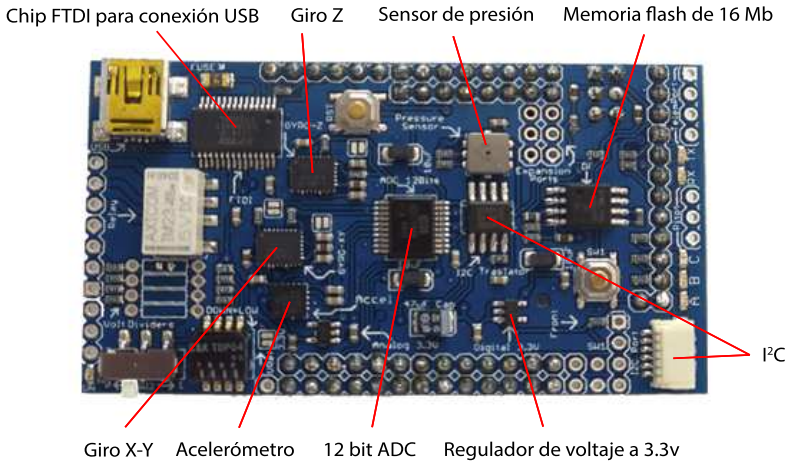

Figura 2.8: Imagen de la IMU mostrando la situación de los sensores.

2.3.2. Giro

Orientación de los ejes y medida de la orientación

Los ejes de giro de la IMU están orientados según se observa en la siguiente figura:

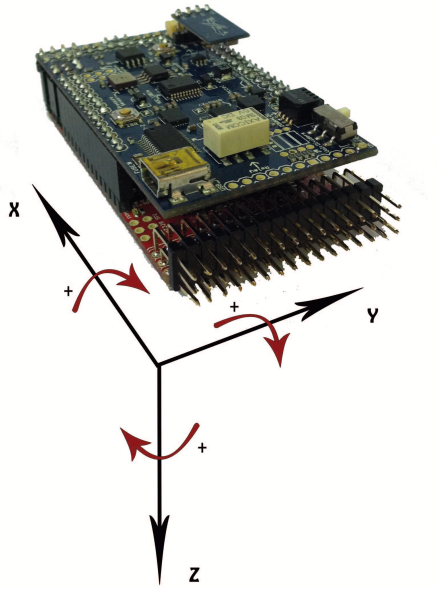

Figura 2.9: Sentido positivo de los ángulos según los ejes obtenidos.

En esta figura se puede observar la IMU montada sobre la placa de control Arduino. Esta disposición se ha obtenido en base a experimentos realizados sobre el conjunto IMU-Placa Arduino. En estos experimentos se hizo girar la placa primero en sentido positivo y a continuación en sentido negativo. Esta acción se realizó en los tres ejes, obteniendo los valores representados en la figura 2.10.

En este experimento se observa cómo al girar la placa según los ejes indicados en la figura 2.9 (primero giro en sentido positivo y segundo en sentido negativo), la IMU envía giros positivos y negativos para los tres ejes, confirmándose así el sentido elegido de los ejes de giro.

También podemos observar en esta figura una deriva en la medida del ángulo yaw. Esta deriva puede deberse a un proceso de calibración del magnetómetro en busca del norte magnético. El proceso de arranque del quadrotor es suficientemente largo para que la calibración termine antes de comenzar a volar, y por lo tanto, no afecta al vuelo del quadrotor.

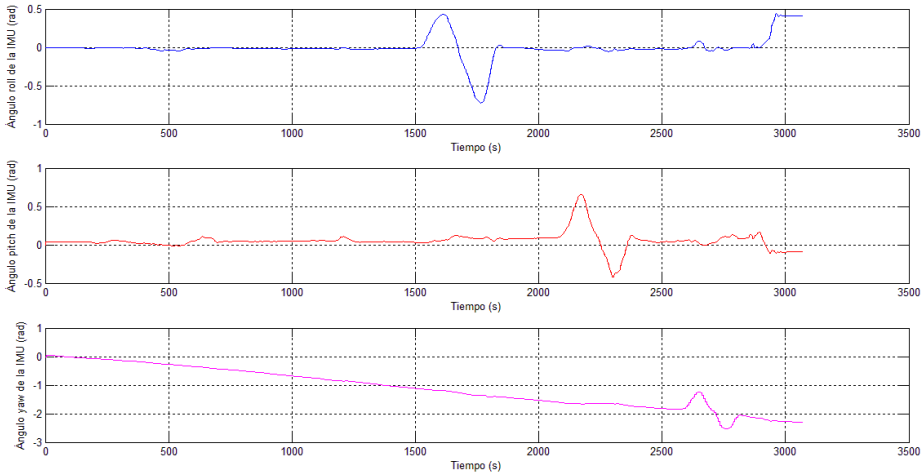

Figura 2.10: Resultado experimental al aplicar giros a la placa Arduino.

2.3.3. Velocidad angular

Para obtener los valores de la IMU para la velocidad se hizo el mismo experimento anterior. En la figura 2.11 se pueden observar los resultados. En dicha figura vemos que al realizar el giro en un eje, primero en sentido positivo y después en sentido negativo, obtenemos una curva con pendiente positiva seguida de una pendiente negativa. Esto corresponde a la velocidad del primer movimiento en el que hemos girado la placa 90 grados en sentido positivo. En la figura 2.12 podemos observar en detalle a que movimiento corresponde cada parte de la gráfica. El sistema de referencia al que están referidas estas medidas es el propio de la IMU, representado en la figura 2.9.

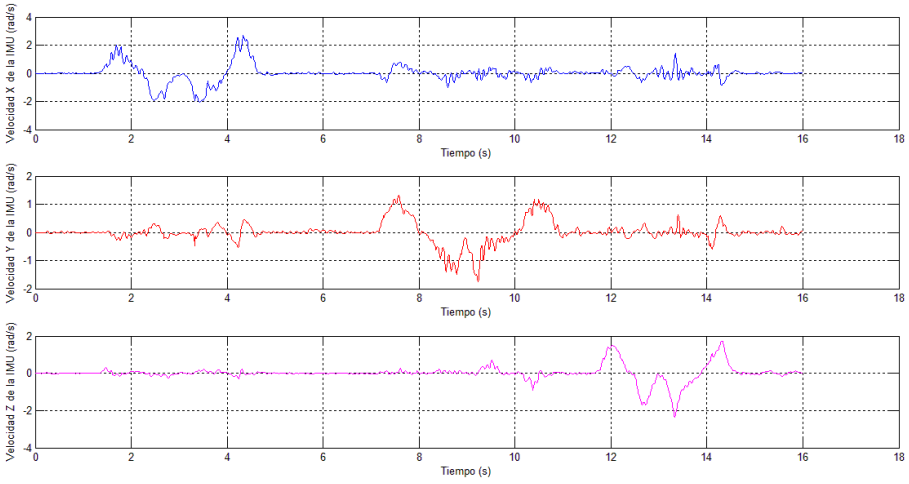

Figura 2.11: Componentes del vector ω registrados por la IMU al aplicar giros a la placa Arduino.

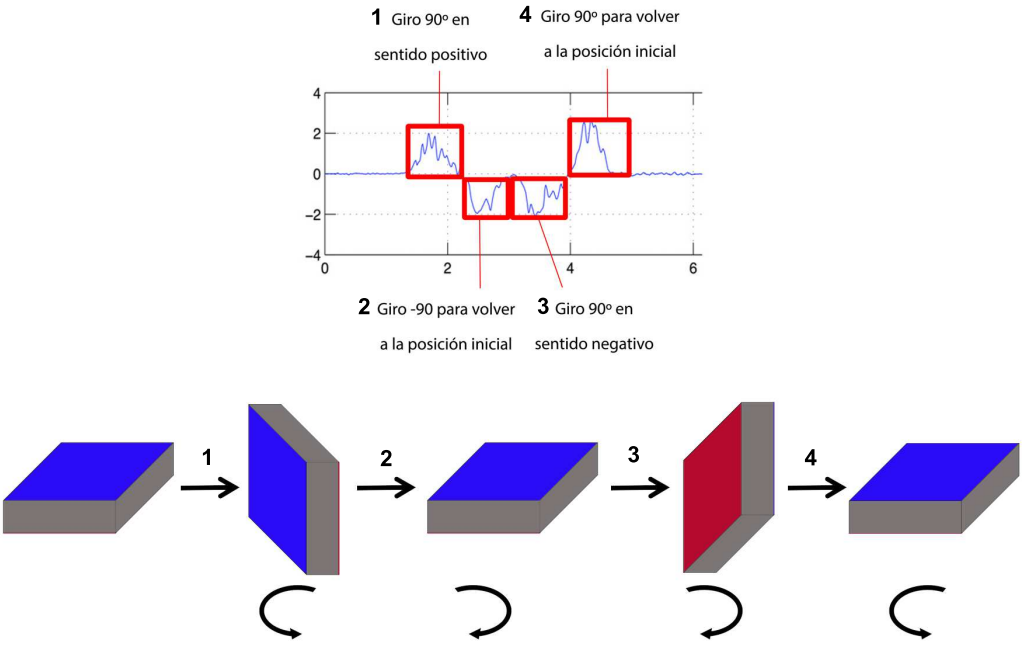

Figura 2.12: Detalle de la velocidad en el eje X.

2.3.4. Aceleración lineal

Para obtener la medida de la aceleración se ha realizado un experimento distinto. En este caso se ha realizado un movimiento hacia delante (sentido positivo del eje) y hacia atrás (sentido negativo del eje) en cada uno de los ejes, obteniéndose los valores representados en la figura 2.13.

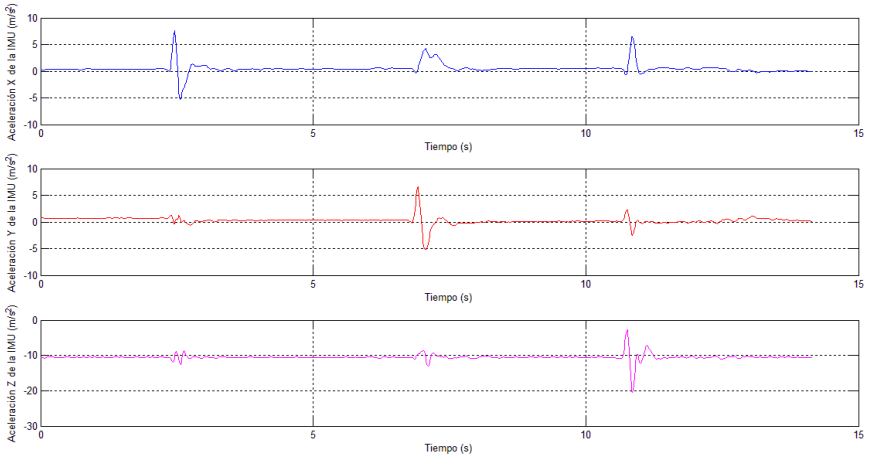

Figura 2.13: Aceleración registrada por la IMU al aplicar giros a la placa Arduino.

Como podemos observar, la IMU ha registrado valores positivos en el movimiento hacia delante y negativos hacia atrás, confirmando nuevamente el sentido elegido de los ejes indicados en la figura 2.9.

2.4. Ardupilot

La placa utilizada para realizar el control del quadrotor es una modificación de la marca Arduino denominada Ardupilot Mega. Esta placa utiliza un microcontrolador ATmega2560 que funciona a una velocidad de 16MHz, dispone de 16 entradas analógicas, 40 entradas/salidas digitales y una memoria flash de 256Kb. En esta placa también se encuentra el conector para el GPS.

La placa Ardupilot es capaz de establecer comunicaciones mediante el bus I^2C comentado anteriormente. Concretamente esta comunicación se utiliza para la conexión con los variadores y el magnetómetro.

Para alimentar el Ardupilot Mega es necesario una fuente de 5 voltios, que en nuestro caso obtenemos del regulador de tensión existente en uno de los variadores. También es posible utilizar una fuente de alimentación externa de 7 a 11 voltios, como una batería Lipo 2s de 7.4V , si eliminamos la soldadura del jumper SJ1.

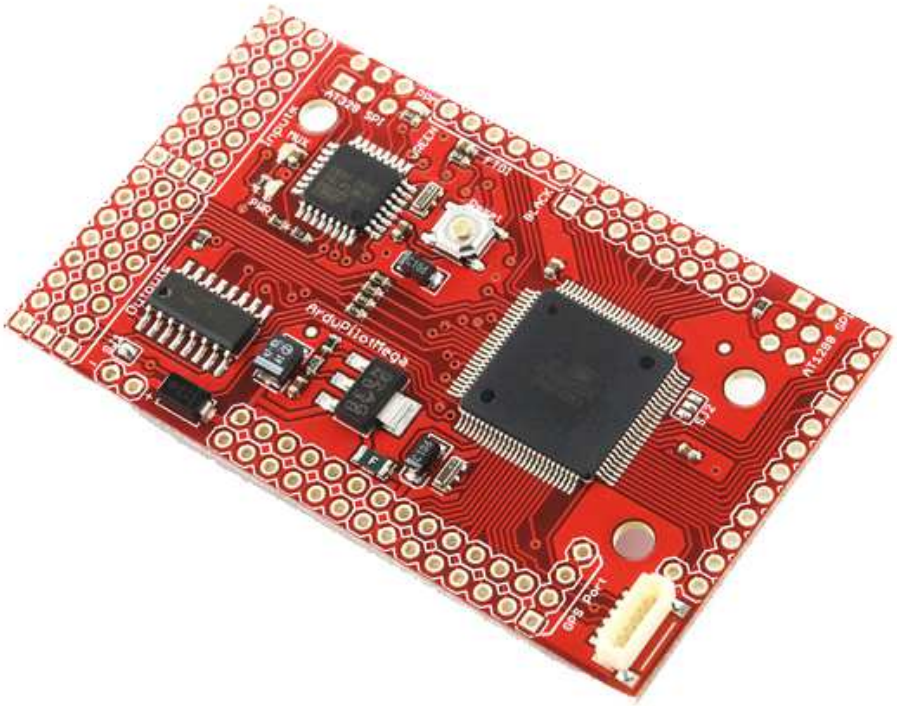

Figura 2.14: Placa Ardupilot Mega.

2.5. Sensor de altura

Figura 2.15: Sonar XL-MaxSonar®-EZ0 de MaxBotix®.

El sensor de altura utilizado en el quadrotor es el modelo XL-MaxSonar®-EZ0 de MaxBotix®. Este sensor de ultrasonidos es capaz de medir distancias desde 20 centímetros a 7 metros. Tiene una resolución de 1 cm y es capaz de autocalibrarse ante cambios de temperatura, humedad, voltaje y ruido ambiental y eléctrico. El sensor está conectado a la placa Ardupilot a través de su salida analógica (Pin 3 del sensor), la cual suministra la medida con un factor de escalado de Vcc/1024 por centímetro. Esta medida está limitada por el hardware a 700 cm cuando se alimenta el sensor con 5v.

Algunas de las características del sensor se detallan a continuación:

- Resolución de 1 cm.
- Tasa de lectura de 10Hz.
- Frecuencia de trabajo de 42 KHz.
- Posibilidad de realizar la lectura del sensor mediante medida analógica del voltaje, serie o ancho de pulso.
- Rango de 20 a 700 cm. Las medidas fuera de ese rango dan como resultado el extremo del rango por el cual saturan.
- Alimentación entre 3.3v y 5.5v.
- Consumo medio de 3.4 mA.
- Temperatura de funcionamiento desde 0 a 65°C.
- Calibración automática en tiempo real. (Cambios de voltaje, temeratura, humedad y ruido ambiente).

En la figura 2.16 se puede observar un experimento realizado para probar el sonar. En dicho experimento se puso un objeto delante del sensor, se alejó hasta una distancia de unos seis metros y luego se acercó. Esta medida se obtuvo mediante la salida analógica del sensor sin realizar ningún filtrado.

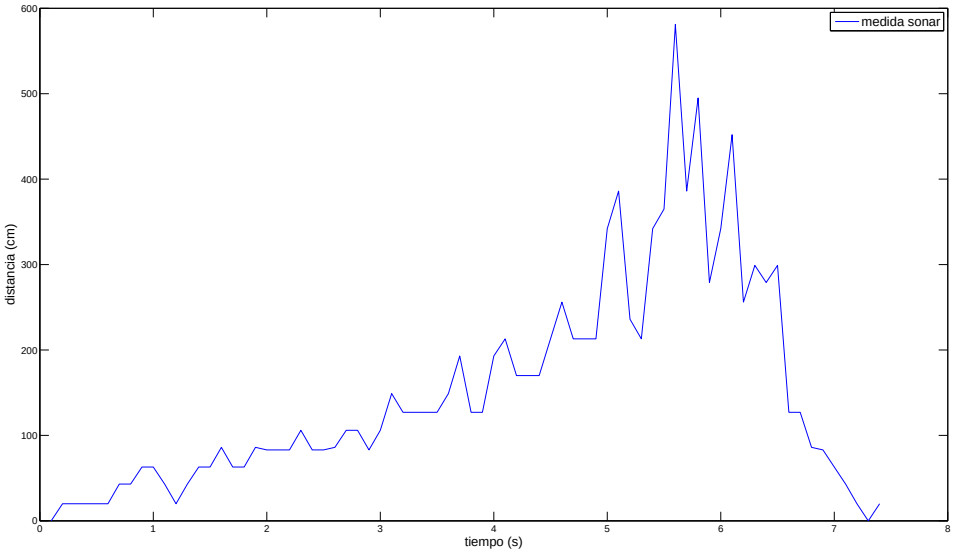

Figura 2.16: Prueba de medida del sonar XL-MaxSonar®-EZ0 de MaxBotix®.

En la figura 2.17 se muestra el resultado de un nuevo experimento. En este caso se realizó el mismo procedimiento, pero se obtuvo la medida mediante ancho de pulso. En este caso, además, la medida obtenida es la media de nueve medidas, por lo que se obtienen unos resultados menos ruidosos que en el caso anterior.

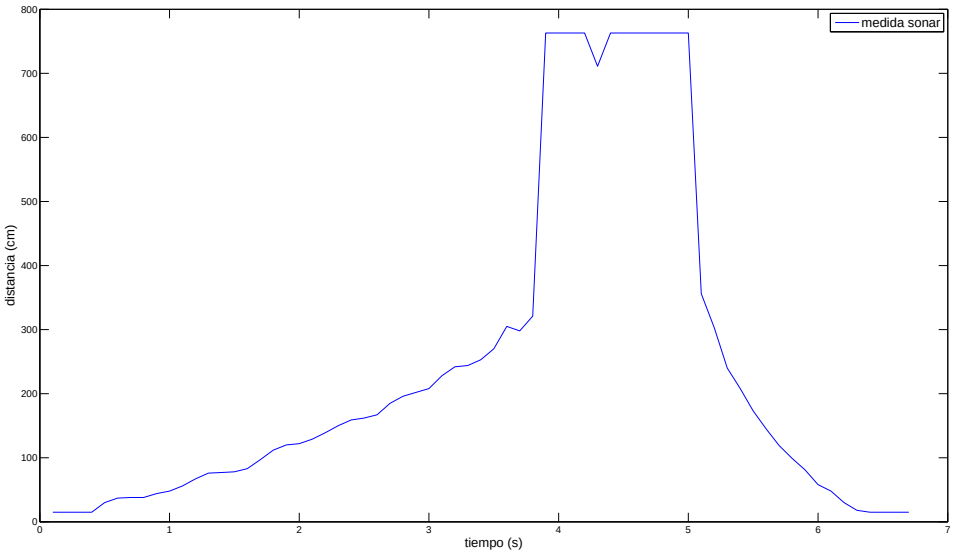

Figura 2.17: Prueba de medida del sonar XL-MaxSonar®-EZ0 de MaxBotix®.

2.6. GPS

Figura 2.18: Módulo GPS.

El GPS utilizado en el quadrotor es el Mediatek GPTA010. Se trata de un GPS de reducidas dimensiones capaz de conectarse a 66 satélites. Dado que la placa Ardupilot incorpora un magnetómetro que proporciona una medida bastante precisa del norte magnético, el GPS fusiona sus medidas con la medida de éste para obtener una medida de la posición más precisa.

Alimentación	5V
Consumo	42 mA
Sensibilidad	-165 dBm
Dimensiones	16 x 16 x 6 mm

Tabla 2.3: Características del GPS GTPA010.

Para probar el GPS se realizó un experimento desplazándolo al aire libre de forma que la posición inicial fuese aproximadamente la misma que la posición final. El resultado se observa en la figura 2.19, donde se representa el desplazamiento medido en coordenadas x e y del GPS.

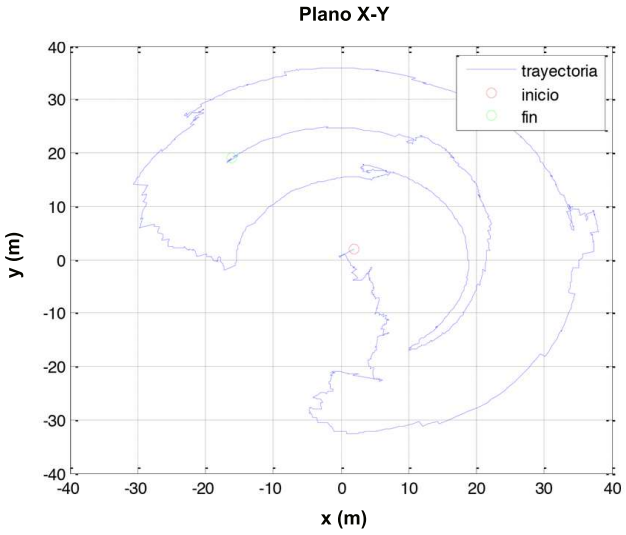

Figura 2.19: Medidas obtenidas por el GPS.

2.7. Comunicaciones

2.7.1. XBee

Para la comunicación entre el quadrotor y la estación de tierra, se usa la tecnología Zigbee. Zigbee es un protocolo de comunicación inalámbrico basado en el estándar de comunicaciones para redes inalámbricas IEEE 802.15.4. Este protocolo permite que dispositivos electrónicos de bajo consumo puedan realizar comunicaciones inalámbricas. Las comunicaciones Zigbee se realizan en la banda libre de 2.4GHz. En nuestra aplicación en particular, para poder establecer una comunicación entre dos módulos es necesario configurar uno de ellos como coordinador y el otro como dispositivo final (End Device). El coordinador tiene la función de crear la red, establecer el canal de comunicaciones y el identificador de la red (PAN ID). El dispositivo final o End Device tiene que configurarse con el mismo nombre de la red creada por el coordinador, a través del cual enviará información al ordenador o a otros end device si los hubiera. Al no tener que desempeñar las funciones del coordinador consumen menos que éstos, por lo que es aconsejable su uso montado en el quadrotor, ya que estarán alimentados por baterías.

Los módulos de comunicaciones utilizados en el quadrotor son concretamente los Xbee PRO Series 1 de la marca Digi International. Algunas de sus características se muestran a continuación:

Alcance en interior	60 m
Alcance en exterior	750 m
Potencia de transmisión	10 mW (10 dBm)
Tasa de transferencia por radio	250000 bps
Tasa de transferencia por puerto serie	1200 bps - 250 kbps
Sensibilidad de receptor	-100 dBm
Tensión de alimentación	2.8 - 3.4 V
Corriente del transmisor	150 mA
Corriente del receptor	55 mA
Frecuencia de trabajo	2.4 GHz
Dimensiones	2.438 x 3.294 cm
Temperatura de funcionamiento	-40 a 85°C

Tabla 2.4: Características de los Xbee Pro Series 1

Figura 2.20: Xbee situado en el quadrotor (End Device).

Figura 2.21: Xbee situado en el ordenador (coordinador).

A través de el Xbee se transmite información sobre las medidas de la IMU, estado de funcionamiento en el que se encuentra el quadrotor, medidas del GPS, medidas del sonar y velocidad de los motores.

2.7.2. Emisora

Con el objetivo de poder imponer referencias de empuje vertical y movimientos en los tres ángulos, roll, pitch y yaw, se decidió instalar un receptor en el quadrotor para una emisora. El modelo elegido fue el 9x8c-V2 para la emisora de nueve canales de la marca Turnigy que se observa en la figura 2.22. La instalación de una emisora responde a la necesidad de realizar un guiado manual del quadrotor. Esta situación es transitoria puesto que el objetivo final es que el quadrotor vuele de forma independiente.

Figura 2.22: Emisora Turnigy 9x y receptor.

La comunicación de la emisora con la placa Ardupilot y de ésta con los variadores se establece mediante señales PWM, dejando la comunicación por Xbee exclusivamente para lectura de telemetría.

El receptor dispone de ocho canales de comunicación para PWM conectados directamente a la placa Ardupilot. Aunque están todos conectados, sólo se usan los pines del 1 al 4 (roll, pitch, empuje y yaw) y el 6 (selección del modo de vuelo). El resto de canales están libres para futuras aplicaciones, como el control de una cámara.

En la imagen inferior observamos en la primera fila de pines del Ardupilot a qué canales del receptor han de estar conectados (Pines rojos). Estos pines están marcados del 1 al 8. El pin 5 es el único que no está conectado. En esta primera fila podemos ver también los cuatro pines, correspondientes a las salidas PWM, marcados del 1 al 4 según el variador del motor al que hay que conectarlos (Pines azules). También podemos ver los dos pines de alimentación del Ardupilot (Pines amarillos) y los del receptor. Dado que el receptor funciona en un rango de tensión de 4.5 a 6.5V, podemos alimentarlo directamente desde la placa Ardupilot.

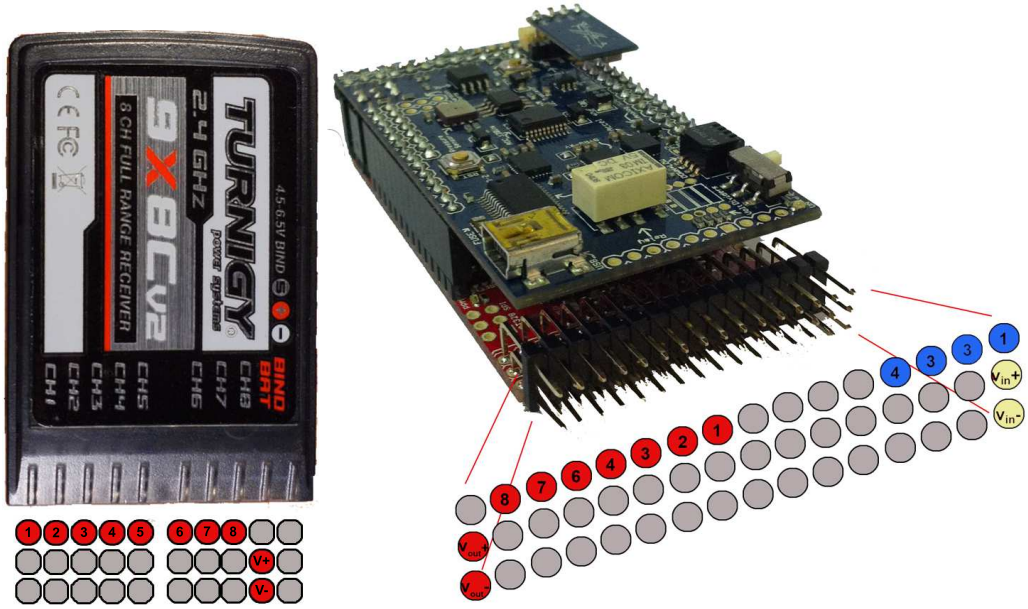

Figura 2.23: Conexionado al receptor de la emisora.

Una vez realizadas esta conexiones, podemos enviar a través de los cuatro canales antes mencionados la referencia de ángulos y empuje que queremos obtener. Esta referencia llega a través del receptor al Ardupilot y éste se encarga de calcular la señal PWM que tiene que llegar al variador.

Como se comentó en apartados anteriores, la comunicación entre la placa Ardupilot y los variadores se realiza a través del bus I^2C. Para utilizar la emisora es necesario utilizar la comunicación a través de PWM. Por esta razón, el uso de la emisora para controlar el vuelo es una situación transitoria, la cual sólo servirá para realizar ajustes de los controladores.

Para poder usar la emisora es necesario calibrarla antes. La calibración consiste en definir las posiciones neutras y límites de cada stick. Para realizar la calibración se usa un programa llamado *Mission Planner*. En este programa se puede observar la asignación que tiene cada canal del receptor con cada uno de los sticks de la emisora y modificar los valores neutral y límite de todos ellos. En este programa también es posible configurar algunos botones para establecer distintos modos de vuelo. Estos modos de vuelo sirven para activar distintos sistemas de control definidos en el quadrotor, tales como control de altura, estabilización o posicionamiento por GPS.

2.8. Baterías

Las baterías utilizadas en el quadrotor son el modelo 5.0 3s 5000mAh 25-35C de la marca Turnigy. Ésta es una batería de polímero de litio (Lipo), las cuales se caracterizan por su elevada densidad de energía, reducido peso y elevada tasa de descarga.
Las baterías lipo están formadas por celdas de 3.7 voltios unidas en serie o paralelo. La utilizada en el quadrotor es 3s por lo que está formada por 3 celdas en serie para obtener una tensión total de 11.1 voltios.

Figura 2.24: Batería.

Una característica importante de estas baterías es su tasa de descarga, es decir, la cantidad de amperios/h que puede suministrar de forma segura. Este dato nos lo proporciona el valor 25-35C, lo que significa que podemos demandar una intensidad de $5Ah*25 = 125A$ de forma continua o $5Ah*35 = 175A$ de pico durante 10 segundos. Dado que los cuatro motores del quadrotor funcionando a la vez consumen un máximo de 90 amperios cuando lo hacen a máxima velocidad, esta batería es adecuada para su uso en nuestro montaje.

Algunas de las características de las baterías son:

Capacidad	5000 mAh
Configuración	3S1P
Tensión nominal	11.1v
Tensión de funcionamiento	12.6v
Tasa de descarga constante	25C
Tasa de descarga (pico)	35C durante 10 segundos
Dimensiones	146 x 50 x 25 mm
Peso	412 gramos
Conector de carga	JST-XH
Conector de descarga	4mm bullet connector

Tabla 2.5: Características de las baterías Turnigy 5.0

2.9. Placa de distribución de potencia y señal de control

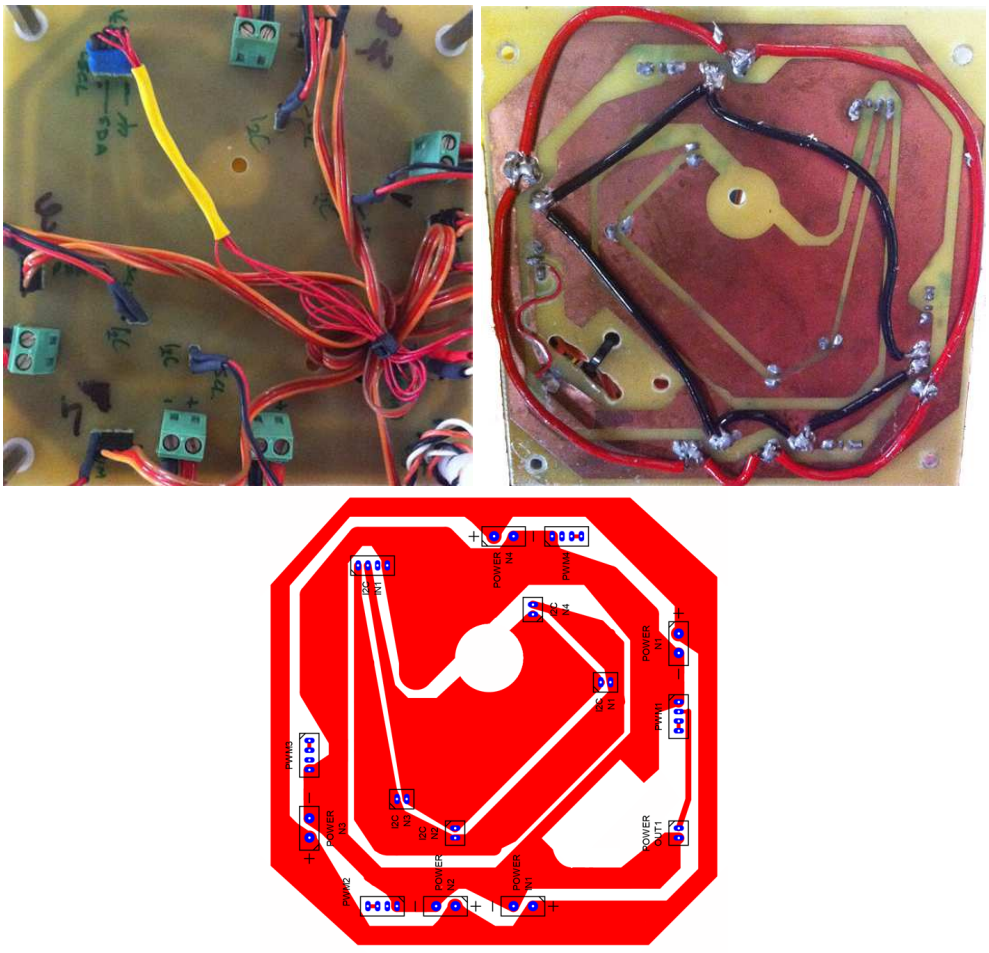

Figura 2.25: Placa de distribución de potencia y señal de control.

Esta placa, diseñada especialmente para el quadrotor, distribuye la señal de potencia a los variadores y conecta al Ardupilot con ellos a través del bus I^2C. Esta placa también alimenta la placa Ardupilot mediante una salida de 5 voltios obtenida de uno de los variadores. Dado que por esta placa va a circular una intensidad grande, se ha decidido reforzar las pistas de alimentación de los motores con cables conectados directamente a los conectores de cada variador. Esto puede observarse en la segunda imagen de la figura 2.25.

Esta placa también sirve para realizar la distribución de la señal PWM de la emisora cuando ésta se utiliza para controlar el quadrotor.

Si se observa la figura 2.25, se puede ver que la placa diseñada dispone de:

- Un conector de salida PWM, uno I^2C y uno de alimentación a cada motor.
- Un conector de entrada de tensión conectado a la batería (marcado como POWER IN1).
- Un conector de enlace I^2C con la placa Ardupilot (marcado como I2C IN1).
- Un conector de alimentación de 5V para la placa Ardupilot (marcado como POWER OUT1).

Capítulo 3

Diseño y sistema de control del quadrotor

3.1. Diseño

En este apartado se puede observar el montaje final del quadrotor. Para dicho montaje se han utilizado la mayor parte de los elementos descritos en el apartado anterior, además se han utilizado varillas cuadradas de aluminio de 1 cm de lado para los brazos del quadrotor.
También se han utilizado placas de metacrilato para proporcionar rigidez a la estructura y servir de soporte a los elementos anteriores.

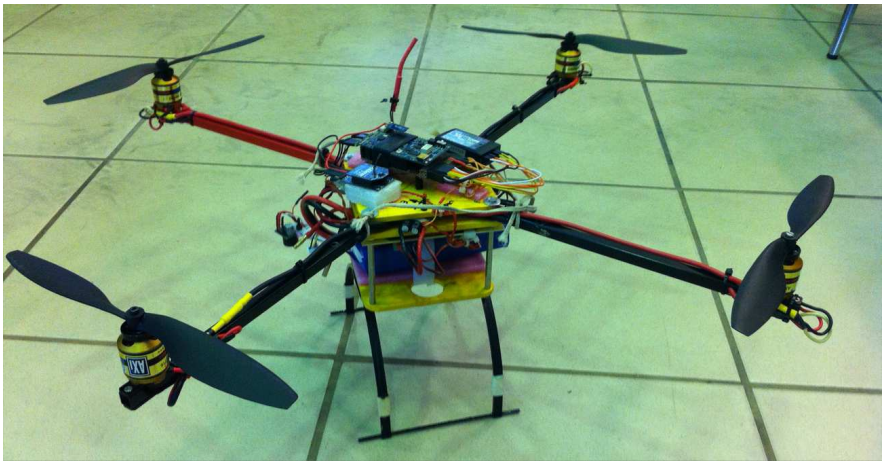

Figura 3.1: Montaje final del quadrotor.

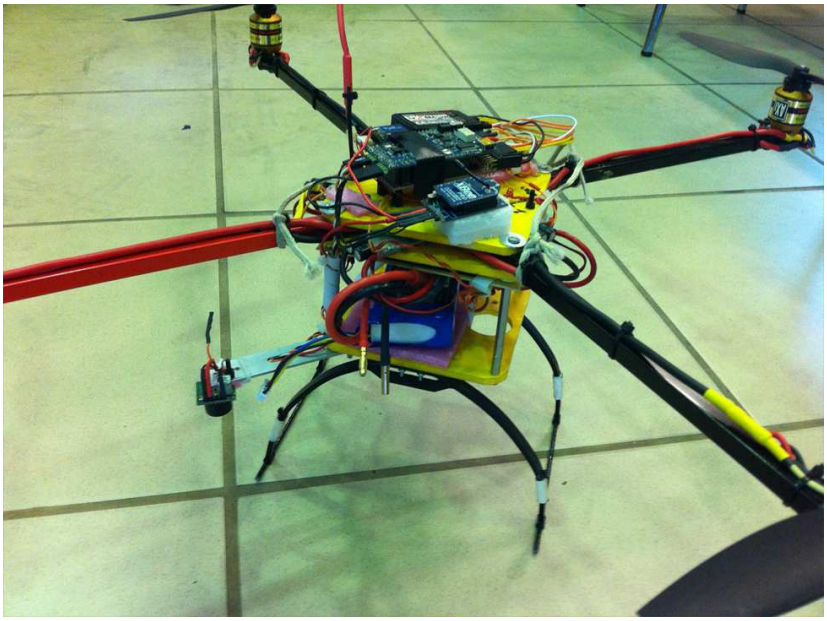

Figura 3.2: Montaje final del quadrotor.

Algunas de las medidas más importantes del quadrotor se observan en los siguientes planos.

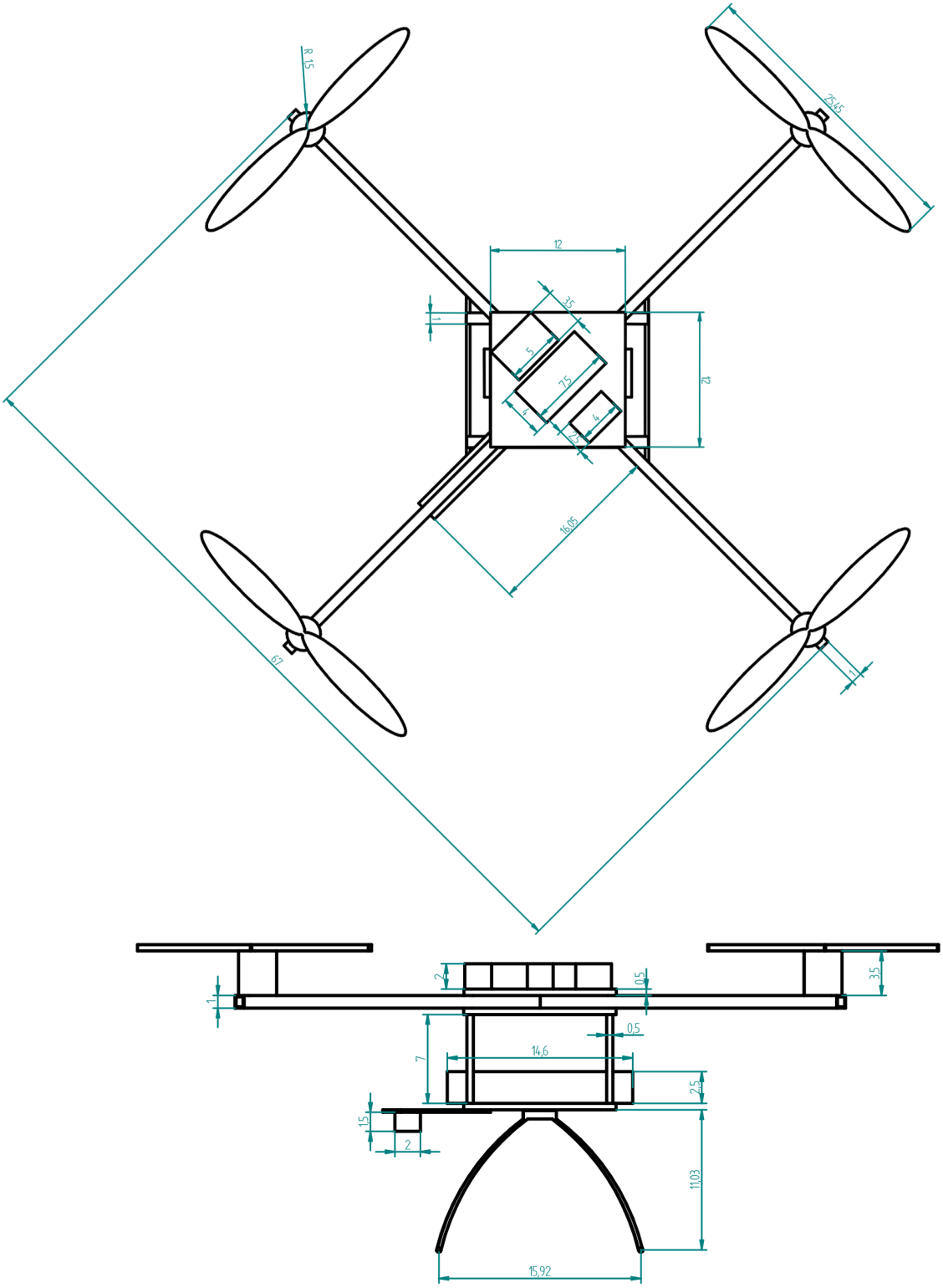

Figura 3.3: Vista en planta y alzado del quadrotor. (Todas las cotas en cm)

El peso total del quadrotor es de 1581 gramos, distribuidos de la siguiente forma:
- Peso electrónica: 100g (Placa de conexiones) + 65g (Ardupilot) + 40 (Receptor emisora + Xbee)
- Batería: 410g
- Estructura: 936g
- Emisor de ultrasonidos: 30g

Dado que el empuje total del quadrotor es de 52 Newtons, la relación empuje/peso que se obtiene es de 3,25:1 la cual es una relación muy buena para un quadrotor de vuelo no acrobático.

3.2. Arquitectura del software de control

En este aparatado se va a comentar la parte software del quadrotor. Se empezará con una descripción de la estructura utilizada en el código de programación del Ardupilot y de cada uno de los estados de funcionamiento en los que puede encontrarse el quadrotor.
El código programado en la placa Ardupilot se encuentra en el apéndice A de este documento.

3.2.1. Estructura del código de control

El código implementado en el quadrotor tiene la estructura básica utilizada en el entorno de desarrollo de cualquier plataforma Arduino. Dicha estructura se compone de una declaración de variables, una función de configuración llamada *setup*() y un bucle principal llamado *loop*().
En la declaración de variables se declaran las variables globales, puertos de comunicaciones y algunas funciones.
En la función *setup*() se realiza la configuración de las comunicaciones, se inicializan algunas variables y se inicializan los sensores y variadores.
La función *loop*() está formada por varios bucles dependiendo de la frecuencia de trabajo a la que se necesite realizar las operaciones. Estos bucles o *loops* son:

- $fast\_loop()$;

  Este bucle opera a una frecuencia de 100 Hz. En él se lee la velocidad medida por los variadores, se envían los datos a través de la telemetría y se lee el AHRS o sistema de referencia de actitud y rumbo. El AHRS fusiona la medida realizada por giróscopos, acelerómetros y magnetómetros para proporcionar datos de la orientación en los tres ejes del espacio. Además puede usar las medidas del GPS para mejorar la estabilidad de los giróscopos. Teniendo las medidas anteriores, se calculan las actuaciones sobre los motores y se envían las señales de control.

- $fifty\_hz\_loop()$;

  Este es un bucle que opera a 50 Hz, en el cual sólo se realiza la medida de altura mediante el sonar comentado en el apartado 2.5.

- *medium_loop*(); **y** *slow_loop*();

  En el *medium_loop* se actualizan las medidas del GPS a una frecuencia de 10 Hz las cuales se envían a través de la telemetría en el *slow_loop* a una frecuencia de 3,33 Hz.

3.2.2. Diagrama de estados

En este apartado se pueden observar los estados programados en el quadrotor. En la tabla 3.1 se muestra una breve reseña de la acción que se realiza en cada estado, así como la condición que se debe cumplir para la transición al siguiente estado. La transición de los estados 2 a 6 se encuentra temporizada, la del estado 9 está sujeta a una posición concreta del quadrotor, mientras que para abandonar los demás estados es necesario enviar un comando a través del Xbee. El comando a enviar corresponde al siguiente estado al que se desea pasar, siempre que sea posible pasar a dicho estado. Se puede observar también en el siguiente diagrama que los estados 5 y 8 no están representados, puesto que corresponden a los estados en los que se realiza el control de posición por GPS (apartado 3.3.4) y control de velocidad lineal (apartado 3.3.5). Estos estados se programaron en un principio, pero debido al funcionamiento poco preciso que presentaban se decidió no incluirlos en la actual programación.

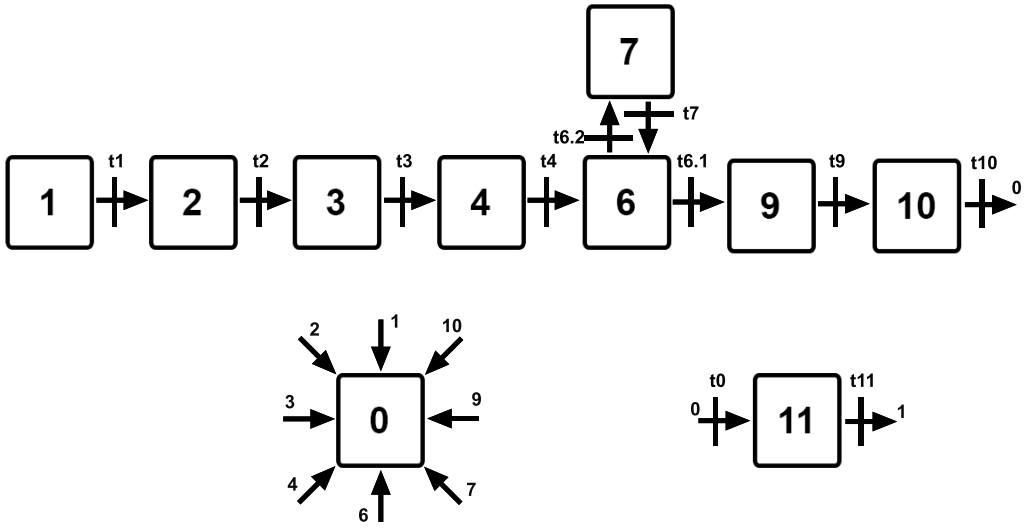

Figura 3.4: Diagrama de estados programados en el quadrotor.

Estado	Descripción	Transición	Condición
1	Reposo	t1	Envío comando ("2")
2	Arranque de los motores en bucle abierto	t2	1 segundo
3	Control de velocidad de los motores	t3	1 segundo
4	Aumento de la velocidad de los motores	t4	0.1 segundos
6	Control de estabilización y altura (Despegue)	t6.1 t6.2	Envío comando ("9") Envío comando ("7")
7	Escalón de altura	t7	Sin condición
9	Aterrizaje	t9	llegar a 40 cm de altura
10	Descenso de la velocidad hasta parada	t10	0.75 segundos
11	Lectura y borrado de datos de la memoria flash	t11	Envío comando ("1")
0	Parada de emergencia	t0	Envío comando ("y","n")

Tabla 3.1: Estados programados en el quadrotor y condiciones para la transición entre ellos.

3.2.3. Descripción de los estados

- **Estado 1**

  En el primer estado se parte de una situación de reposo. Los motores no giran, ya que la señal de control es cero, y se realiza la inicialización de las variables de los controladores. Para pasar al siguiente estado es necesario enviar un comando por Xbee.

- **Estado 2**

  En este estado se arrancan los motores a una velocidad baja sin que el Ardupilot realice ninguna función de control, ya que en este estado no existe realimentación de la medida de velocidad. Esto se consigue mediante la aplicación del 10 % de la señal de acción sobre los motores. Este estado se encuentra temporizado, de forma que se pasa al estado 3 al transcurrir un segundo.

- **Estado 3**

  En el tercer estado los motores siguen girando a una velocidad correspondiente al 10 % de la señal de actuación, aunque, en este caso, sí se realiza un control de la velocidad, produciéndose una realimentación de la medida de ésta. El estado tres también se encuentra temporizado, pasando al estado cuatro al pasar un segundo.

- **Estado 4**

  En este estado se realiza un aumento de la velocidad de giro de los motores en forma de rampa hasta alcanzar un empuje ligeramente inferior al peso de forma que el quadrotor no llegue a despegar. Interesa abandonar este estado de forma rápida, puesto que los motores están produciendo un empuje cercano al peso y se encuentra en situación de despegue. Por esta razón este estado está temporizado a 0.75 segundos.

- **Estado 6**

  En el estado 6 se cierran los lazos de control correspondientes a la altura y a los ángulos, roll, pitch y yaw. En este estado se establece una altura objetivo de un metro como altura inicial forzando de esta forma que el quadrotor despegue. Este estado no se encuentra temporizado, siendo necesario enviar un comando desde el ordenador para pasar a otro estado.

- **Estado 7**

  Para entrar en este estado es necesario enviar un comando por Xbee. Cuando esto ocurre, se realiza un aumento de 10 cm en la variable que almacena la altura objetivo a la que tiene que situarse el quadrotor. Con este estado obtenemos un aumento de la altura de vuelo del quadrotor. Este estado se programó para observar el comportamiento del control de altura ante cambios en la referencia de altura. Aunque no tiene mucho uso en el funcionamiento normal del quadrotor, se ha mantenido programado. Este estado se abandona de forma automática al actualizarse la variable de altura objetivo.

- **Estado 9**

  Este estado corresponde a la primera fase de la operación de aterrizaje del quadrotor. En este estado se mantiene el control de altura en el cual se fija una altura objetivo de 70 cm. Una vez se

alcanza esta altura, se disminuye el empuje a un valor ligeramente inferior al peso del quadrotor de forma que éste descienda gradualmente hasta una altura de 40 cm, momento en el cual se pasa al siguiente estado.

- **Estado 10**

  Alcanzada una altura de 40 cm se pasa a la segunda fase de la operación de aterrizaje, donde se realiza un descenso de velocidad en forma de rampa temporizada en 0.75 segundos. En este momento se paran los motores y se pasa al estado cero.

- **Estado 0**

  En este estado se realiza una parada de emergencia. En él se paran los motores y se pregunta al usuario si desea guardar los datos almacenados en la memoria flash de la placa Ardupilot. Estos datos corresponden a las medidas realizadas por la IMU, sensor de altura, estado que se está ejecutando en cada momento y el valor de todas las variables usadas en los controladores. Tras este estado se ejecuta el estado 11.

- **Estado 11**

  En este estado se realiza la lectura de los datos almacenados en la memoria flash si se ha contestado que sí en el estado 0. En todo caso, en este estado, se pregunta al usuario si desea borrar los datos almacenados en la memoria flash. En caso de contestar de forma afirmativa se realiza dicho borrado en este mismo estado. Para abandonar este estado es necesario enviar un comando a través del Xbee.

3.3. Estrategias de control

En este apartado se van a comentar las distintas estrategias de control implementadas en el quadrotor, las cuales se pueden observar en el siguiente esquema de control. En él se puede observar que los bloques referidos al sistema de control y al sistema quadrotor, se han dividido en dos subsistemas según hagan refrencia a la traslación o a la rotación del quadrotor. Aunque se traten como subsitemas por separado, están relacionados puesto que un cambio en los ángulos de rotación producirá una traslación y viceversa.

Como puede observarse también, las salidas de los controladores generan unos pares de rotación referentes al roll (U_ϕ), pitch (U_θ) y yaw (U_ψ) y un par de empuje (U_T) a partir de los ángulos, la posición y la velocidad de referencia.

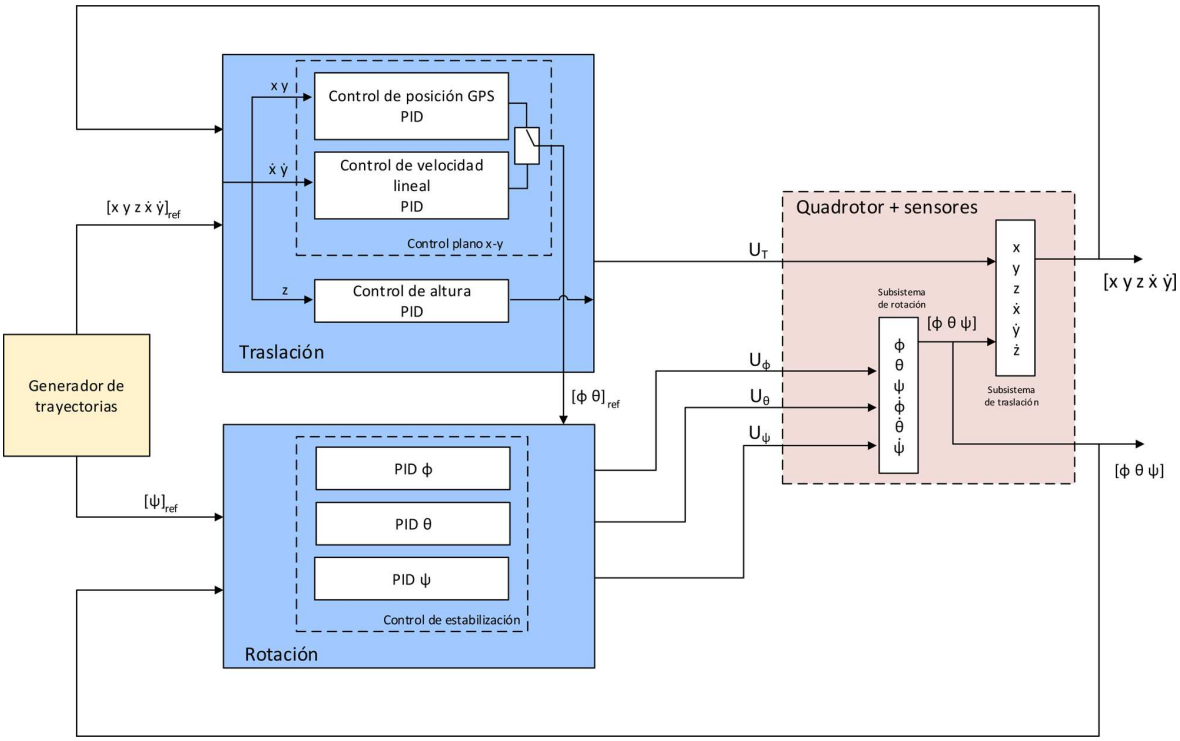

Figura 3.5: Esquema de estrategias de control.

3.3.1. Control de velocidad de los motores

En el control a bajo nivel de velocidad de los motores se controla la velocidad de rotación de los mismos, la cual, dado que la técnica de control empleada en este caso es trapezoidal, se obtiene a través de los variadores. Se comentó anteriormente que dicha medida de velocidad se encuentra en unidades desconocidas, lo cual no es necesario para realizar el control.

A la salida del control de traslación y rotación se obtienen unos pares de rotación y empuje que

posteriormente serán convertidos a fuerzas a aplicar por cada motor. Para realizar la conversión de fuerza a velocidad angular es necesario un conocimiento previo tanto del modelado del motor como de la geometría de las hélices. Una vez obtenida la variable de control, se utiliza un contolador proporcional integral (PI) con anti windup, cuya señal de salida se filtra mediante un filtro de primer orden. Este proceso se puede observar en el esquema de control de la figura 3.6.

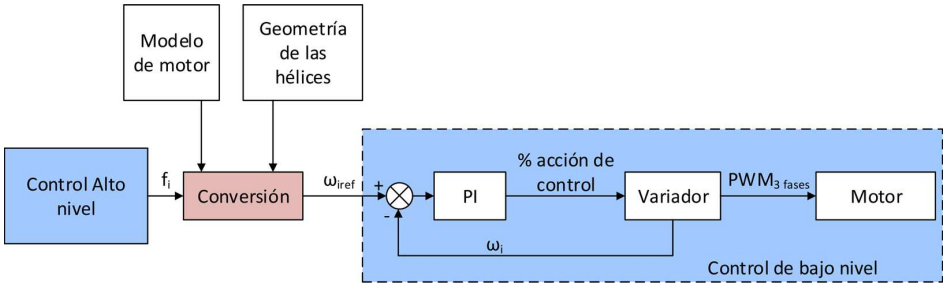

Figura 3.6: Esquema de control de velocidad de los motores.

La velocidad de giro de los motores se encuentra limitada por la cantidad de corriente que pueden soportar, por lo que la señal de control se encuentra limitada a un 90 %. Como se comentó en el apartado 2.1.3 la señal de control mínima para que los motores comiencen a girar es de un 5 %.

3.3.2. Control de estabilización

En este apartado de control se controla el sistema de referencia de actitud y rumbo o AHRS, utilizando como referencia los valores de los tres ángulos de navegación, roll, pitch y yaw. Para ello se utilizan las medidas realizadas por la IMU y los valores de variación temporal de dichos ángulos, también proporcionados por la placa Ardupilot.

Para este control se utiliza un controlador proporcional integral derivativo (PID) con anti windup.

Con este controlador se obtienen los pares a aplicar en cada ángulo que junto con el empuje necesario en cada momento determinan las fuerzas de referencia de cada motor. Una vez obtenida esta fuerza, se utilizan las curvas características representadas en el apartado 2.1.2 para calcular la velocidad de giro de cada motor y el porcentaje de acción de control que la placa Ardupilot envía a los variadores.

3.3.3. Control de altura

Para realizar el control de altura se utilizan las medidas obtenidas por el sonar descrito en el apartado 2.5. Estas medidas se filtran mediante un filtro de primer orden. Dado que la altura que se quiere calcular es la altura del centro de masas del quadrotor, es necesario hacer una transformación de la medida realizada por el sonar, dado que éste se encuentra situado a 14 cm, en dirección positiva del eje x, de dicho centro de masas. Esta medida filtrada y corregida es la que se usa para calcular la respuesta del controlador de altura.

Para el control de altura se utiliza un controlador proporcional integral derivativo (PID) con anti windup.

Al igual que en el control de estabilización, los pares calculados mediante el controlador junto con el empuje sirven para calcular las fuerzas a aplicar en cada motor y mediante las curvas características de los motores se obtiene la velocidad de referencia y el porcentaje de acción de control.

3.3.4. Control de posición por GPS

El control de posición se basa en las medidas realizadas por el GPS comentado en el apartado 2.6. Como se puede leer en dicho apartado, las medidas del GPS se fusionan con las medidas del magnetómetro para obtener una medida más precisa de la posición.

Para el control de posición mediante GPS se utiliza un controlador proporcional integral derivativo (PID). La señal de control obtenida se filtra mediante un filtro de primer orden y se le aplica un control anti windup.

3.3.5. Control de velocidad lineal

Mediante este apartado se realiza el control de la velocidad del quadrotor en los ejes móviles x e y de la IMU, la cual se obtiene mediante integración numérica de la medida que ésta proporciona de la aceleración.

Para este control se utiliza un controlador proporcional integral derivativo (PID) cuya señal se filtra mediante un filtro de primer orden.

Este control se basa en el envío del sentido positivo o negativo del eje de referencia en el que deseamos mover el quadrotor. Dichas referencias se envían al quadrotor mediante telemetría a través del teclado del ordenador.

Aunque este apartado de control se encuentra programado, no se ha llegado a utilizar finalmente en el quadrotor debido al funcionamiento poco preciso que presenta. Dicho funcionamiento se debe a que el valor de la velocidad lineal se obtiene mendiante integración numérica de la aceleración, ya que no se dispone de medida directa. Esta integración acumula un error que produce el mal funcionamiento del control.

Capítulo 4

Posicionamiento por ultrasonidos

A la hora de realizar experimentos con el quadrotor, uno de los principales inconvenientes que nos encontramos es que éste presenta derivas en la posición. Para poder realizar los ajustes necesarios de los distintos controladores de los que disponemos, es necesario realizar pruebas de vuelo en el interior del laboratorio. Por esto es imprescindible que el quadrotor no realice este movimiento horizontal y permanezca en una determinada posición. Por este motivo se ha decidido realizar un control de posición mediante balizas estacionarias basadas en ultrasonidos.

Para poder realizar el posicionamiento de un objeto en el espacio es necesario medir la distancia a cuatro puntos. Por ello se ha diseñado un sistema donde el emisor de ultrasonidos se encuentra montado en el quadrotor y los cuatro receptores están ubicados en el suelo. De esta forma midiendo las distancias desde el emisor hasta los cuatro receptores es posible posicionar el quadrotor en el espacio.

Debido a que comercialmente los medidores de ultrasonidos están fabricados con el emisor y receptor en el mismo dispositivo, se ha decidido aprovechar unos emisores y receptores sueltos que se encontraban en el laboratorio para fabricar nuestro propio dispositivo de medida.
Siguiendo las especificaciones del fabricante de los ultrasonidos se han montado los siguientes circuitos electrónicos:

4.1. Emisor

El emisor utilizado es el MA40S4S de la marca *muRata*. Para excitar dicho emisor es necesario realizar el circuito formado por los componentes que se especifican en el diagrama de la figura 4.1. Con este circuito se obtiene una señal de excitación como la observaba en la figura 4.2, la cual genera un tren de pulsos a 42 KHz durante un milisegundo cada 24 milisegundos.

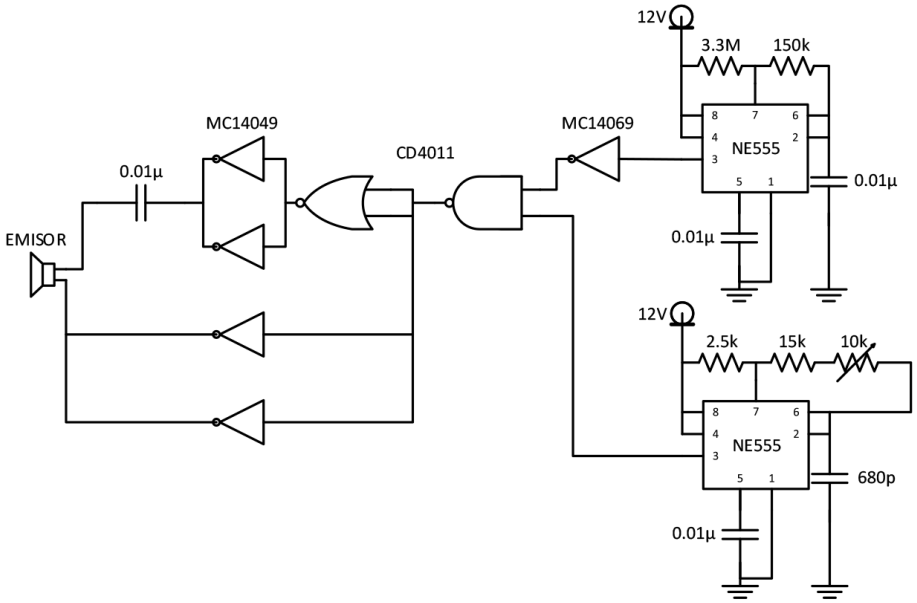

Figura 4.1: Diagrama del circuito emisor de ultrasonidos.

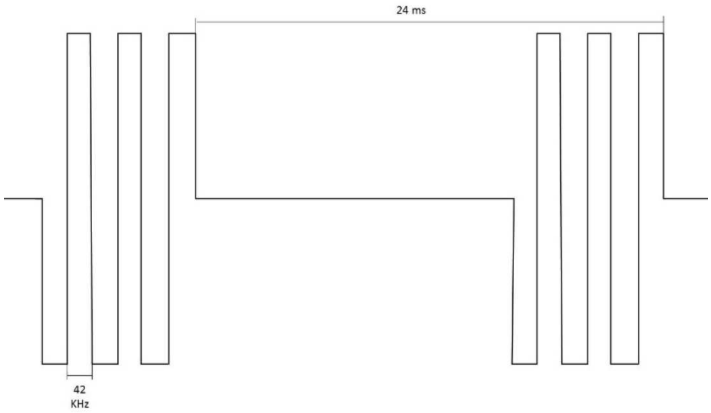

Figura 4.2: Señal de excitación producida por el emisor.

El circuito emisor está alimentado a 12 voltios, produciendo una señal de excitación de 24 voltios de amplitud, ya que los componentes existentes en el emisor producen una señal desde -12 a +12 voltios.

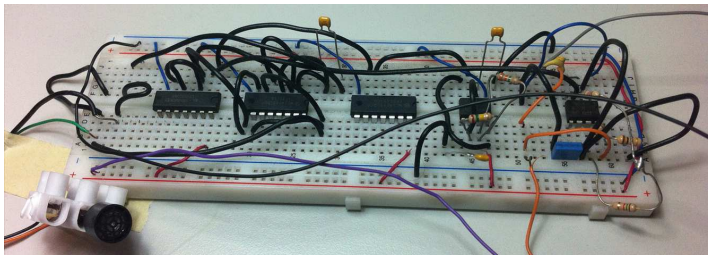

Figura 4.3: Montaje del emisor en placa de prototipos.

Como se ha comentado anteriormente, el emisor está montado en el quadrotor, por lo que para el diseño final se ha conseguido reducir el tamaño de la placa a unas dimensiones de 6 x 7 cm., obteniéndose la placa de la figura 4.4.

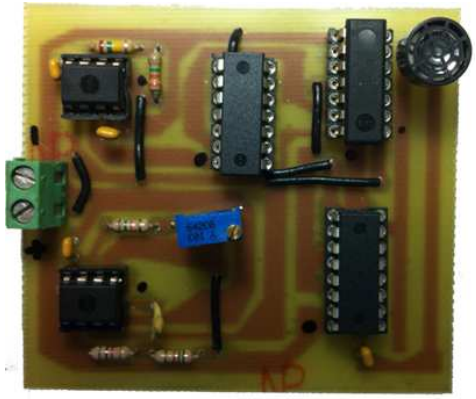

Figura 4.4: Montaje del emisor en placa impresa.

4.2. Receptor

Para el receptor se ha usado el modelo MA40B8R de la marca *muRata*. Debido a que la señal obtenida con el sensor es muy débil, es necesario amplificarla hasta un valor adecuado para poder realizar medidas sobre ella. Para esto se ha realizado el circuito cuyo diagrama se representa en la figura 4.5.

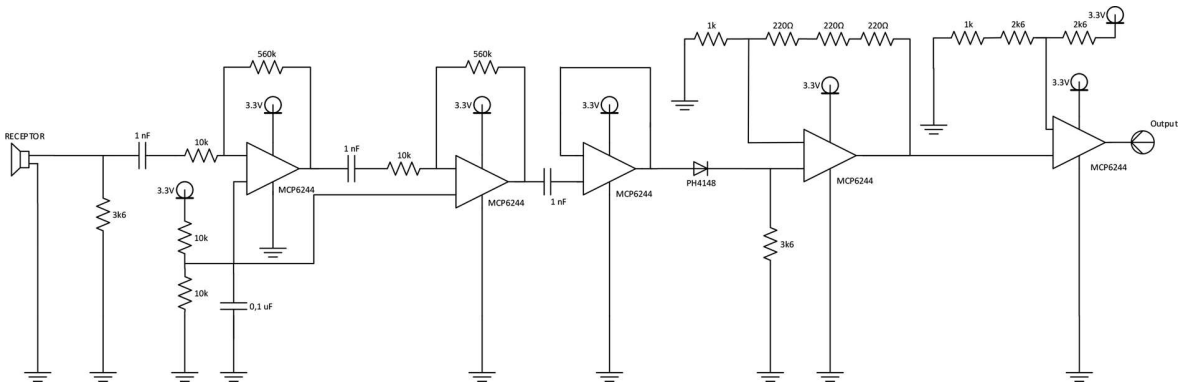

Figura 4.5: Diagrama del circuito receptor de ultrasonidos.

Este circuito se alimenta mediante una batería de litio de 3.7V. Dicha batería es la encargada de alimentar también la placa electrónica con la que se realiza la medida de distancia, de la cual se hablará en el siguiente apartado.

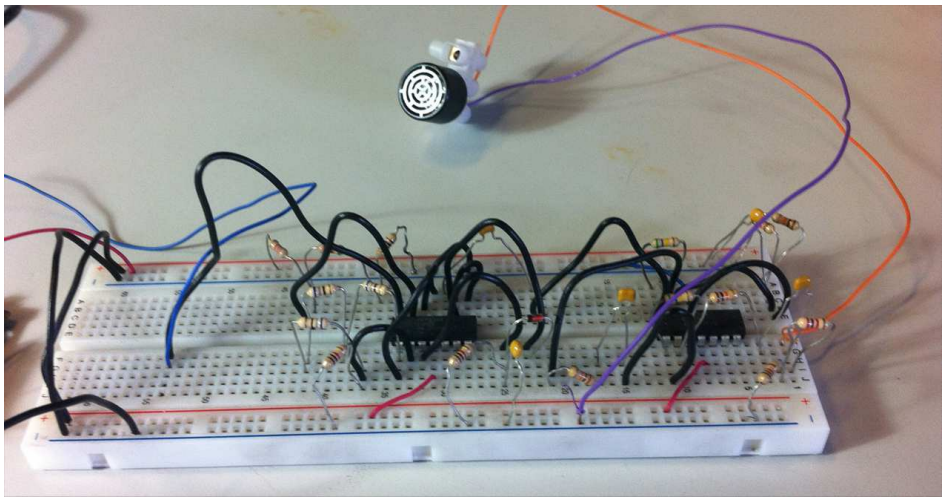

Figura 4.6: Montaje del receptor en placa de prototipos.

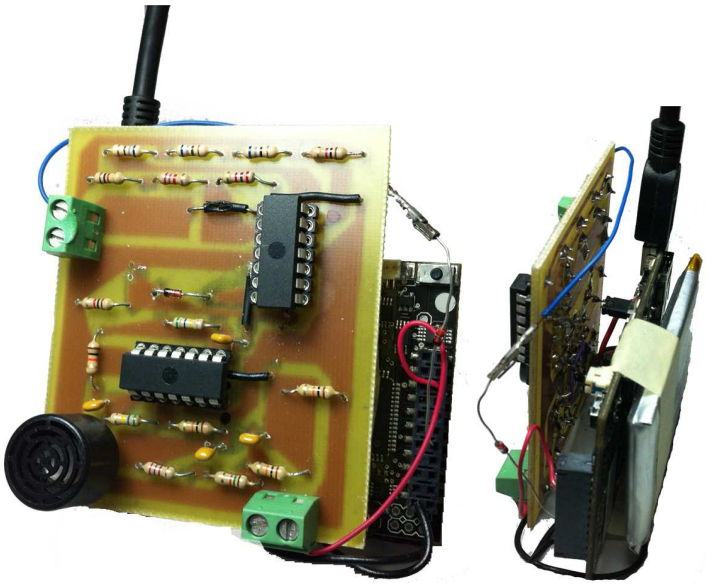

Figura 4.7: Montaje del receptor en placa impresa conectado a la placa Waspmote.

La señal obtenida del receptor se puede observar en la figura 4.8. Esta señal emite un pulso de 1 ms con una frecuencia de 24 ms de periodo siempre que el emisor y el receptor se encuentren a una distancia fija uno del otro.

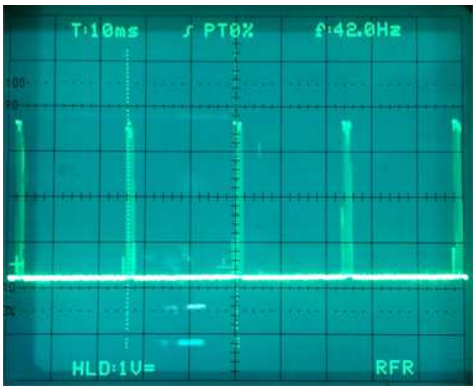

Figura 4.8: Señal de salida generada en el receptor.

4.3. Lectura de distancia

Como se ha comentado antes, mediante el montaje hecho en el receptor, se obtiene una señal cuadrada de 24 ms de periodo que se encuentra en estado alto 1 ms cuando no varía la distancia entre emisor y receptor. Cuando el emisor varía su posición respecto al receptor, esta señal varía su periodo. Esto se debe a que la señal emitida es una onda sonora que se transmite a través del aire a la velocidad del sonido, por lo que al variar la distancia entre emisor y receptor variará también el tiempo entre pulsos de la señal. Midiendo este tiempo somos capaces de saber cuanto ha variado la distancia entre emisor y receptor.
Para realizar la medida de tiempos entre flancos, se ha programado el microcontrolador de una placa de la marca *Libelium* llamada *Waspmote*. Esta placa tiene la característica principal de poder establecer conexiones inalámbricas entre varias de ellas y un receptor situado en un pc mediante adaptadores Xbee como los mencionados en el apartado 2.7.1. Así se puede crear una red con los cuatro receptores de ultrasonidos para obtener sus medidas y poder calcular la posición del quadrotor.

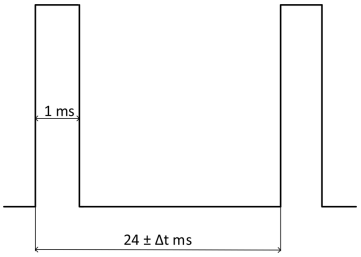

Figura 4.9: Señal producida por el receptor.

Dado que este sistema no proporciona medida directa de distancia, sino variaciones de ésta, es necesario partir de un valor inicial conocido, que se establecerá en función de la distancia de cada receptor al emisor en el momento del despegue del quadrotor. De esta forma, el valor de la distancia se actualizará, a partir de ese valor inicial, mediante la siguiente ecuación:

$$\text{Distancia nueva } (m) = \text{Distancia antigua } (m) \pm \Delta t(s) * 343(m/s)$$

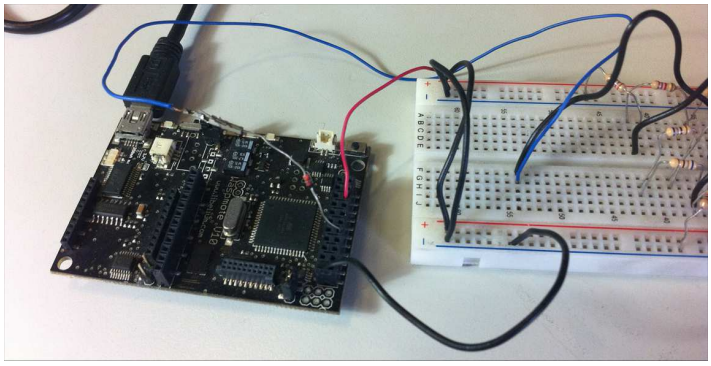

Figura 4.10: Placa Waspmote.

En la figura 4.10 se puede observar la placa *Waspmote*, la cual genera los 3.3 voltios que alimentan el receptor de ultrasonidos.

El código de programación utilizado en el *Waspmote* se encuentra en el apéndice B.

Para comprobar que las lecturas del *Waspmote* son correctas, se realizó un experimento con una señal generada por un generador de señales, obteniéndose la gráfica de la figura 4.11. Observando dicha gráfica se puede comprobar que la lectura de tiempos se está realizando de forma correcta.

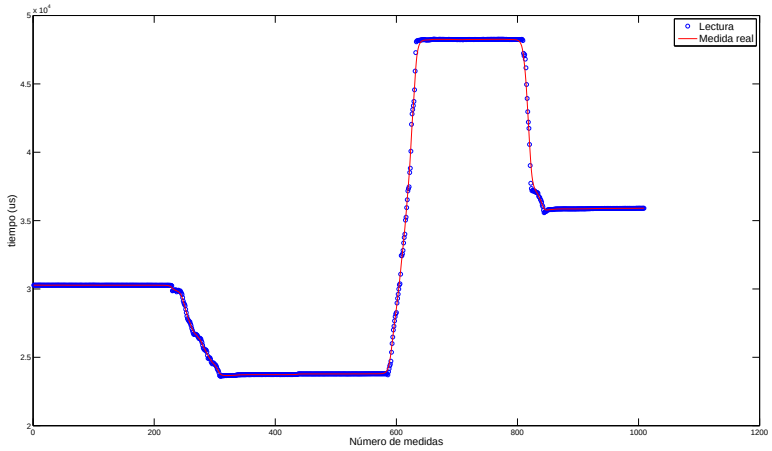

Figura 4.11: Lectura del Waspmote de una señal obtenida con un generador de señales.

En la figura 4.12 se puede observar la lectura de tiempos realizada sobre una señal obtenida mediante el sensor de ultrasonidos. Como se puede ver, esta lectura es menos precisa debido a que la señal obtenida por el sensor de ultrasonidos es más ruidosa que la proporcionada por el generador de señales.

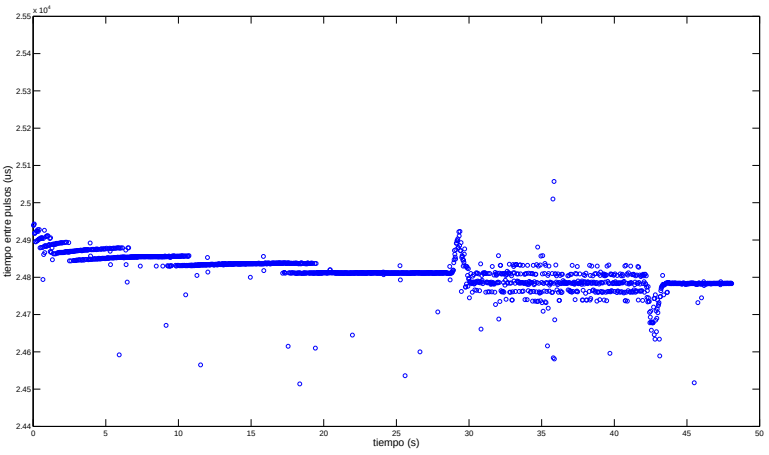

Figura 4.12: Lectura del Waspmote de una señal obtenida con el sensor de ultrasonidos.

Para calcular la distancia entre emisor y receptor se usa la medida realizada por el *Waspmote* filtrada, obteniéndose la gráfica de la figura 4.13. Como se comentó anteriormente, es necesario proporcionar un valor inicial de distancia para generar la medida. En el caso del experimento realizado, se comenzó con una distancia entre emisor y receptor de unos 50 centímetros.

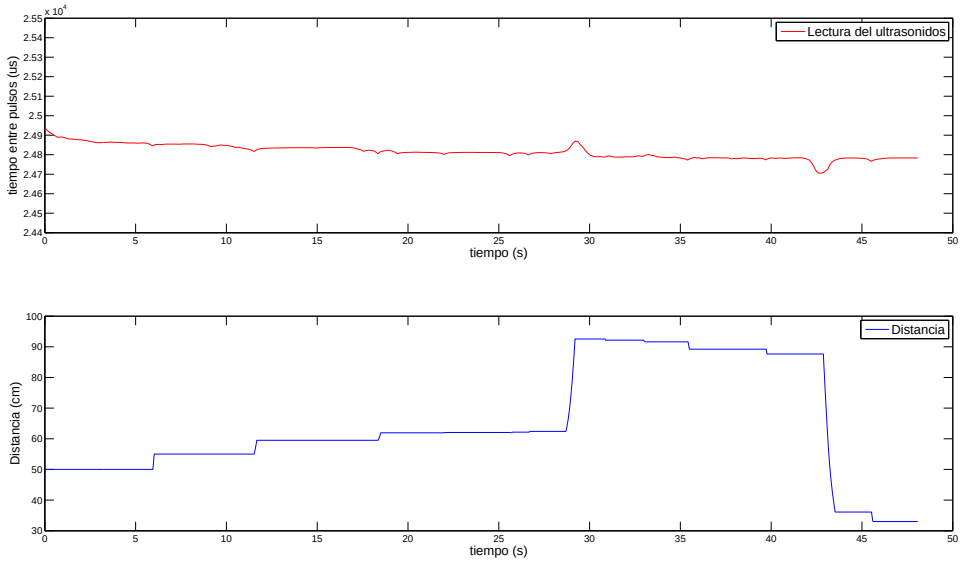

Figura 4.13: Lectura del Waspmote filtrada y transformada en distancia.

Después de realizar numerosos experimentos, los resultados obtenidos no proporcionan una precisión aceptable. Esto puede ser debido a una señal demasiado ruidosa y a la necesidad de realizar un mejor montaje del emisor y receptor de ultrasonidos. Dado que las pruebas realizadas con un generador de señales producían unos resultados óptimos, es posible descartar la lectura de los datos, por parte de la placa waspmote, como causa del problema de precisión.

4.4. Posicionamiento 3D

Para poder posicionar el quadrotor en el espacio tridimensional es necesario medir la distancia respecto a cuatro puntos fijos. De esta forma, como ya se comentó anteriormente, se dispone de un emisor de ultrasonidos en el quadrotor y de cuatro receptores situados en una posición fija conocida. El método utilizado es similar al usado para el sistema de posicionamiento por GPS. Este método consiste en trazar una esfera, cuyo radio será la distancia medida al quadorotor, alrededor del primer receptor. De esta forma la posición del quadrotor estará situada en la superficie de dicha esfera. Si se traza otra esfera alrededor del segundo receptor, la posición del quarotor se encontrará en el círculo que forma la intersección de las dos esferas. Al considerar un tercer receptor, la posición queda reducida a los dos puntos en los que interseccionan las tres esferas. Por este motivo es necesario un cuarto receptor que limite la posición a un sólo punto.

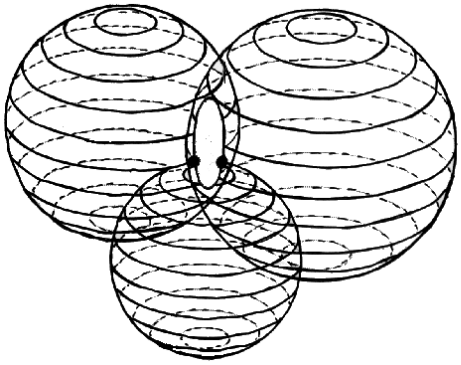

Figura 4.14: Esferas que delimitan la posición en el espacio del quadrotor.

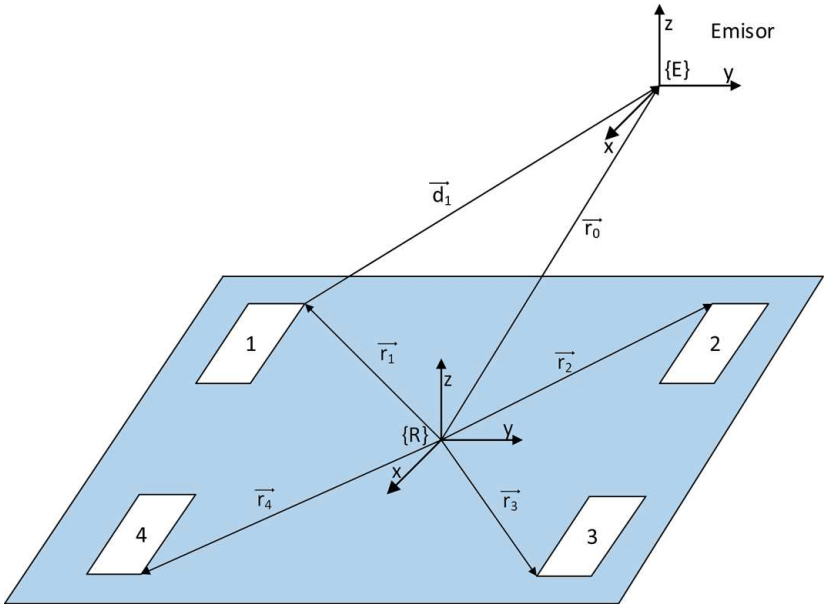

Figura 4.15: Esquema y sistemas de referencia para el cálculo de ecuaciones de la posción.

En la figura 4.15 podemos observar el esquema del problema a resolver, donde se indican los sistemas de referencia solidarios al quadrotor y al conjunto de receptores de ultrasonidos. El objetivo es calcular el valor del vector $\vec{r_0}$ que marca la posición del quadrotor en el espacio. Para ello se utilizarán dos vectores conocidos, el vector $\vec{r_i}$ que determina la posición del receptor i respecto al sistema de referencia R y el vector $\vec{d_i}$ del cual se conoce el módulo puesto que es la medida que se obtiene del ultrasonidos. En el montaje realizado se ha posicionado cada receptor a la misma distancia del sistema de referencia R, por lo que los vectores $\vec{r_i}$ tendrán todos el mismo módulo.

$$||\vec{r_1}|| = ||\vec{r_2}|| = ||\vec{r_3}|| = ||\vec{r_4}|| = D$$

Realizando una composición de vectores obtenemos, $\vec{r_i} = \vec{r_0} + \vec{d_i}$, de donde:

$$||\vec{r_1}||^2 = ||\vec{r_0}||^2 + 2\vec{r_0}\vec{d_1} + ||\vec{d_1}||^2 \quad (1)$$

$$||\vec{r_2}||^2 = ||\vec{r_0}||^2 + 2\vec{r_0}\vec{d_2} + ||\vec{d_2}||^2 \quad (2)$$

$$||\vec{r_3}||^2 = ||\vec{r_0}||^2 + 2\vec{r_0}\vec{d_3} + ||\vec{d_3}||^2 \quad (3)$$

$$||\vec{r_4}||^2 = ||\vec{r_0}||^2 + 2\vec{r_0}\vec{d_4} + ||\vec{d_4}||^2 \quad (4)$$

restando (1) - (2):

$$\vec{r_0}(\vec{d_1} - \vec{d_2}) = \frac{||\vec{d_2}||^2 - ||\vec{d_1}||^2}{2} \quad (5)$$

restando (2) - (3):

$$\vec{r_0}(\vec{d_2} - \vec{d_3}) = \frac{||\vec{d_3}||^2 - ||\vec{d_2}||^2}{2} \quad (6)$$

restando (3) - (4):

$$\vec{r_0}(\vec{d_3} - \vec{d_4}) = \frac{||\vec{d_4}||^2 - ||\vec{d_3}||^2}{2} \quad (7)$$

sabiendo que:

$$\vec{d_1} = (x_1 - x_0; y_1 - y_0; z_1 - z_0)$$

$$\vec{d_2} = (x_2 - x_0; y_2 - y_0; z_2 - z_0)$$

$$\vec{d_3} = (x_3 - x_0; y_3 - y_0; z_3 - z_0)$$

$$\vec{d_4} = (x_4 - x_0; y_4 - y_0; z_4 - z_0)$$

obtenemos:

$\vec{d_1} - \vec{d_2} = (x_1 - x_2; y_1 - y_2; z_1 - z_2)$

$\vec{d_2} - \vec{d_3} = (x_2 - x_3; y_2 - y_3; z_2 - z_3)$

$\vec{d_3} - \vec{d_4} = (x_3 - x_4; y_3 - y_4; z_3 - z_4)$

Introduciendo esto en (5), (6) y (7) obtenemos el siguiente sistema de ecuaciones:

$$x_0(x_1 - x_2) + y_0(y_1 - y_2) + z_0(z_1 - z_2) = \frac{||\vec{d_2}||^2 - ||\vec{d_1}||^2}{2}$$

$$x_0(x_2 - x_3) + y_0(y_2 - y_3) + z_0(z_2 - z_3) = \frac{||\vec{d_3}||^2 - ||\vec{d_2}||^2}{2}$$

$$x_0(x_3 - x_4) + y_0(y_3 - y_4) + z_0(z_3 - z_4) = \frac{||\vec{d_4}||^2 - ||\vec{d_3}||^2}{2}$$

cuya solución proporciona la posición del quadrotor en el espacio:

$$x_0 = \frac{\begin{vmatrix} \frac{||\vec{d_2}||^2 - ||\vec{d_1}||^2}{2} & y_1 - y_2 & z_1 - z_2 \\ \frac{||\vec{d_3}||^2 - ||\vec{d_2}||^2}{2} & y_2 - y_3 & z_2 - z_3 \\ \frac{||\vec{d_4}||^2 - ||\vec{d_3}||^2}{2} & y_3 - y_4 & z_3 - z_4 \end{vmatrix}}{\begin{vmatrix} x_1 - x_2 & y_1 - y_2 & z_1 - z_2 \\ x_2 - x_3 & y_2 - y_3 & z_2 - z_3 \\ x_3 - x_4 & y_3 - y_4 & z_3 - z_4 \end{vmatrix}}$$

$$y_0 = \frac{\begin{vmatrix} x_1 - x_2 & \frac{||\vec{d_2}||^2 - ||\vec{d_1}||^2}{2} & z_1 - z_2 \\ x_2 - x_3 & \frac{||\vec{d_3}||^2 - ||\vec{d_2}||^2}{2} & z_2 - z_3 \\ x_3 - x_4 & \frac{||\vec{d_4}||^2 - ||\vec{d_3}||^2}{2} & z_3 - z_4 \end{vmatrix}}{\begin{vmatrix} x_1 - x_2 & y_1 - y_2 & z_1 - z_2 \\ x_2 - x_3 & y_2 - y_3 & z_2 - z_3 \\ x_3 - x_4 & y_3 - y_4 & z_3 - z_4 \end{vmatrix}}$$

$$z_0 = \frac{\begin{vmatrix} x_1 - x_2 & y_1 - y_2 & \frac{||\vec{d_2}||^2 - ||\vec{d_1}||^2}{2} \\ x_2 - x_3 & y_2 - y_3 & \frac{||\vec{d_3}||^2 - ||\vec{d_2}||^2}{2} \\ x_3 - x_4 & y_3 - y_4 & \frac{||\vec{d_4}||^2 - ||\vec{d_3}||^2}{2} \end{vmatrix}}{\begin{vmatrix} x_1 - x_2 & y_1 - y_2 & z_1 - z_2 \\ x_2 - x_3 & y_2 - y_3 & z_2 - z_3 \\ x_3 - x_4 & y_3 - y_4 & z_3 - z_4 \end{vmatrix}}$$

En la figura 4.16 se observa la disposición que se ha comentado. En esta figura también se puede observar que la transmisión de las medidas de distancia a través de las placas *Waspmote* se realizará de forma inalámbrica mediante Xbee. Las medidas llegarán al receptor de Xbee colocado en el ordenador y éste se encargará de realizar los cálculos de la posición en la que se encuentra el quadrotor. Una vez realizados estos cálculos, enviará a través de la telemetría las referencias de posición a la placa Ardupilot. Este proceso aún no se ha desarrollado por lo que permanece pendiente como trabajo futuro.

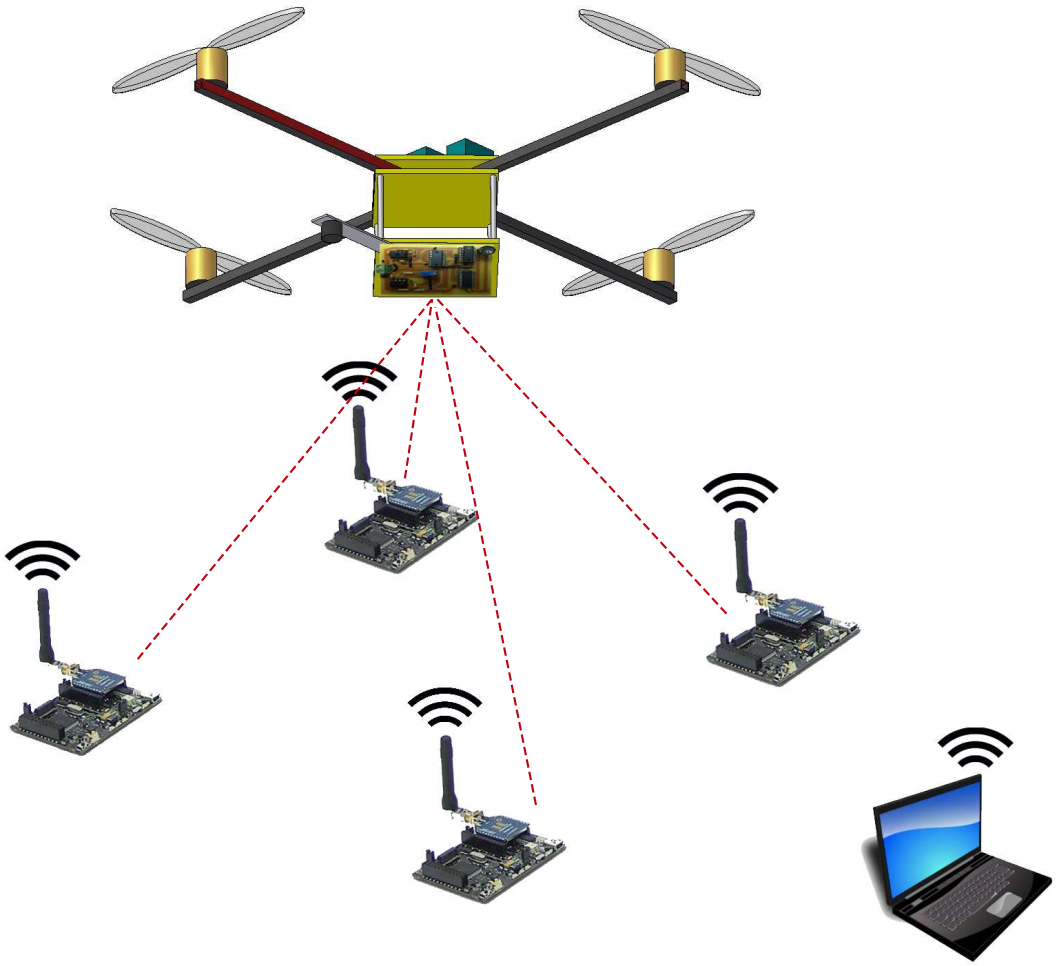

Figura 4.16: Esquema para posicionamiento tridimensional.

Con la inclusión del ultrasonidos en el sistema, es necesario modificar el esquema de control mostrado en la figura 3.5, detallándose algo más la equipación sensorial del quadrotor, obteniéndose el esquema de la figura 4.17. En este esquema, el bloque del controlador es igual que el anterior, donde se observaban los sistemas de control en rotación y traslación. En este caso, al introducir un sistema de control por ultrasonidos, habrá que añadir un nuevo bloque al control en el plano x-y, por lo que se podrá elegir entre el control de posición por GPS, el de velocidad lineal y el control por ultrasonidos. En este esquema también es posible observar que el sonar y el sensor de ultrasonidos se encuentran metidos en un mismo bloque. Se ha considerado así para resaltar el hecho de que sólo uno funciona en cada momento, es decir, cuando el sistema de ultrasonidos proporcione medidas de posición, el sonar

se encontrará desactivado y viceversa. Esto es debido a que el sonar y el sistema de ultrasonidos usan la misma frecuencia de excitación y se producen interferencias en las medidas de ambos.

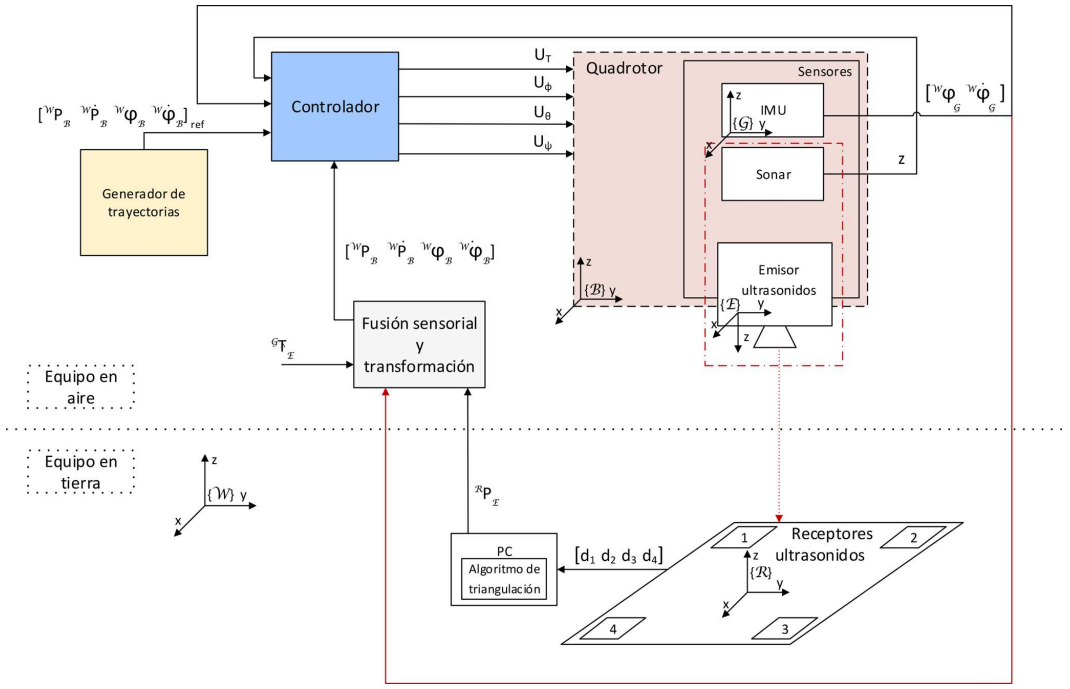

Figura 4.17: Esquema de control con sistema de posicionamiento por ultrasonidos.

La notación utilizada en el esquema para los sistemas de referencia es la siguiente: sistema asociado a la IMU ($\mathcal{G}$), emisor de ultrasonidos ($\mathcal{E}$), receptor ($\mathcal{R}$), quadrotor ($\mathcal{B}$) y sistema de referencia inercial, denotado como ($\mathcal{W}$).

Otros símbolos representados en el esquema son:

- d_i: distancia medida por el receptor i-ésimo.
- ${}^{\mathcal{B}}\mathbf{p}_{\mathcal{A}}$: es el vector de posición del sistema de referencia $\mathcal{A}$ respecto del sistema de referencia $\mathcal{B}$.
- ${}^{\mathcal{B}}\dot{\mathbf{p}}_{\mathcal{A}}$: es el vector de velocidades del sistema de referencia $\mathcal{A}$ respecto del sistema de referencia $\mathcal{B}$.
- ${}^{\mathcal{B}}\boldsymbol{\varphi}_{\mathcal{A}}$: es el vector de ángulos *roll*, *pitch* y *yaw* del sistema de referencia $\mathcal{A}$ respecto al sistema de referencia $\mathcal{B}$.
- ${}^{\mathcal{B}}\dot{\boldsymbol{\varphi}}_{\mathcal{A}}$: es el vector de variación temporal de los ángulos *roll*, *pitch* y *yaw* del sistema de referencia $\mathcal{A}$ respecto al sistema de referencia $\mathcal{B}$.
- ${}^{\mathcal{B}}\hat{\mathbf{T}}_{\mathcal{A}}$: es la estimación de la relación entre el sistema de referencia $\mathcal{A}$ y el sistema de referencia $\mathcal{B}$.

Capítulo 5

Conclusiones y mejoras

En esta memoria se ha documentado todo el proceso de construcción de un quadrotor, el cual era el objetivo principal de este Proyecto Fin de Carrera.

En este proceso de construcción se han seleccionado y comprado elementos mecánicos, eléctricos y electrónicos acorde al tamaño del diseño que se quería obtener, se han fabricado otros elementos, como la placa de distribución de potencia y señal y se han diseñado y ajustado controladores PD, PI y PID.

Después de todo este trabajo, se han realizado pruebas de vuelo obteniéndose resultados muy positivos en el aspecto de estabilización y control de altura. Sin embargo, los controles de posición por GPS y velocidad lineal deben mejorarse. En el caso del control de velocidad lineal ya se comentó la causa del funcionamiento poco preciso al describir el control en el apartado 3.3.5. En el caso del control de posición por GPS es necesario un ajuste más fino de los parámetros del controlador. Además de estos sistemas para el control de posición, se está estudiando la posibilidad de utilizar una cámara de video para complementar los sistemas de posicionamiento. La implementación y ajuste de estos sistemas queda como trabajo futuro.

Tras la construcción del quadrotor y de las pruebas realizadas, se ha podido comprobar que una de las principales características de este tipo de vehículos aéreos es la maniobrabilidad y estabilidad de vuelo. Es debido a estas características que en la actualidad se están realizando gran cantidad de proyectos con estos vehículos, consiguiéndose grandes resultados a la hora de realizar formaciones e interacciones entre varios quadrotor o utilizando éstos para montar estructuras sencillas.

Durante el desarrollo de este Proyecto también se ha podido comprobar la causa del auge en la investigación de estos vehículos, ya que para su desarrollo es necesario investigar en diversos aspectos de la ingeniería. Estos aspectos abarcan la ingeniería mecánica, ingeniería eléctrica, ingeniería electrónica, teoría de control de vuelo y navegación y sobre todo la ingeniería automática, destacando los campos de diseño y ajuste de controladores, sistemas de control por visión o sistemas en tiempo real.

En el momento en el que se terminó de escribir este Proyecto Fin de Carrera, ya se estaban realizando algunos trabajos de mejora. Entre ellos, se está continuando con el posicionamiento por ultrasonidos del cual se habló en el apartado 4. También se encuentra en estudio un sistema que permita realizar la grabación de imágenes o video mediante una cámara fijada a una estructura balancín capaz de contrarrestar los movimientos del quadrotor, de forma que la imagen permanezca siempre horizontal con el ajuste visual deseado.

Capítulo 6

Bibliografía

- Proyecto Arducopter, http://code.google.com/p/arducopter/

- Guilherme Vianna Raffo. Modelado y control de un helicóptero quadrotor. Tesis doctoral, Universidad de Sevilla 2007.

- Wikipedia, http://es.wikipedia.org/

- Página oficial del proyecto Arduino, http://www.arduino.cc

- Libelium Comunicaciones Distribuidas S.L., http://www.libelium.com/es/products/waspmote/

- Lucio Di Jasio. Programming 16-Bit PIC Microcontrollers in C: Learning to Fly the PIC 24. Ed. Newnes

- Luis Fernando Castaño. Lenguaje C: Herramienta de ingeniería. Universidad de Sevilla

- Jonay Tomás Toledo Carrillo. Estrategias de control para sistemas multivariables no lineales : aplicación a un helicóptero de 4 rotores. Tesis doctoral, Universidad de la Laguna, 2007.

- Jesús María González Villagómez. Realimentación visual para el control de un vehículo aéreo cuatrimotor no tripulado. Proyecto final de carrera, Universidad de Sevilla 2010.

- Ramn A. Garca, Jess G. Villagmez, Guilherme V. Raffo, Manuel G. Ortega, Manuel Vargas, Francisco R. Rubio. Plataforma para experimentación de controladores basada en sistema aeropropulsado de cuatro rotores coplanarios. Universidad de Sevilla

- Proyecto Starmac, University of Stanford/Berkeley. http://hybrid.eecs.berkeley.edu/starmac/

- Josh Villbrandt. The Quadrotors Coming of Age. University of Southern California

- Quad-Rotor Autonomous Helicopter Eschews Gps in Favor of Lasers by Lambert Varias. http://technabob.com/blog/2009/10/16/quad-rotor-autonomous-helicopter/

- Herramienta para la simulación integrada de subsistemas en un equipo quadrotor. Ramón Andrés García Rodrguez. Proyecto Fin de Carrera, Universidad de Sevilla, 2010.

- Diseño y construcción de un quadrotor de motores orientables. Alejandro Suárez Fernández-Miranda. Universidad de Sevilla, 2012.

- Multirotor Aerial Vehicles. Modeling Estimation and Control of Quadrotor by Robert Mahony, Vijay Kumar and Peter Corke. IEEE Robotics & Automation Magazine. Vol 19, No. 13, September 2012.

- Build Your Own Quadrotor by Hyon Lim, Jaeman Park, Daewon Lee and H.J. Kim. IEEE Robotics & Automation Magazine. Vol 19, No. 13, September 2012.

Apéndice A

Código Ardupilot

En el apéndice A se muestra el código programado en la placa Ardupilot. Este código es una modificación del código creado por la comunidad de internet DIY Drones, en el que se han adaptado y añadido partes a nuestro quadrotor, como la programación de los controladores, los estados comentados en el apartado 3.2.2, el envío de información a través de Xbee o algún otro cambio de menor importancia.

Cómo se comentó en el apartado 3.2.1, el código está estructurado en tres funciones principales.

- **Declaración de variables**

  Aquí se declaran las variables globales, puertos de comunicaciones y algunas funciones

- **Setup()**

  En esta función se realiza la configuración de las comunicaciones, se inicializan variables, sensores y variadores

- **loop()**

  La función *loop*() está formada por varios bucles dependiendo de la frecuencia de trabajo a la que se neceseite realizar las operaciones. Estos bucles o loops son:

  - *fast_loop*();
    Este bucle opera a una frecuencia de 100 Hz. En él se lee la velocidad medida por los variadores, se envían los datos a través de la telemetría y se lee el AHRS o sistema de referencia de actitud y rumbo. El AHRS fusiona la medida realizada por giróscopos, acelerómetros y magnetómetros para proporcionar datos de la orientación en los tres ejes del espacio. Además puede usar las medidas del GPS para mejorar la estabilidad de los giróscopos. Teniendo las medidas anteriores, se calculan las actuaciones sobre los motores y se envían las señales de control.

  - *fifty_hz_loop*();
    Este es un bucle que opera a 50 Hz, en el cual sólo se realiza la medida de altura mediante el sonar comentado en el apartado 2.5.

  - *medium_loop*(); **y** *slow_loop*();
    En el *medium_loop* se actualizan las medidas del GPS a una frecuencia de 10 Hz las cuales se envían a través de la telemetría en el *slow_loop* a una frecuencia de 3,33 Hz.

El código programado en la placa es el siguiente:

Archivo Arducopter.pde

```
#define THISFIRMWARE "ArduCopter V2.0.50 Beta"

////////////////////////////////////////////////////////////////////////////////
// Header includes
////////////////////////////////////////////////////////////////////////////////

// AVR runtime
#include <avr/io.h>
#include <avr/eeprom.h>
#include <avr/pgmspace.h>
#include <math.h>
```

```
// Libraries
#include <FastSerial.h>
#include <AP_Common.h>
//#include <APM_RC.h>           // ArduPilot Mega RC Library
#include <AP_GPS.h>           // ArduPilot GPS library
#include <Wire.h>                     // Arduino I2C lib
#include <SPI.h>                      // Arduino SPI lib
#include <DataFlash.h>        // ArduPilot Mega Flash Memory Library
#include <AP_ADC.h>           // ArduPilot Mega Analog to Digital Converter Library
//#include <APM_BMP085.h>      // ArduPilot Mega BMP085 Library
#include <AP_Compass.h>       // ArduPilot Mega Magnetometer Library
#include <AP_Variador_mio.h>     // ArduPilot Mega Variador Library
#include <AP_Math.h>          // ArduPilot Mega Vector/Matrix math Library
#include <AP_IMU.h>           // ArduPilot Mega IMU Library
#include <AP_DCM.h>           // ArduPilot Mega DCM Library
//#include <APM_PI.h>                 // PI library
//#include <RC_Channel.h>      // RC Channel Library
#include <AP_RangeFinder.h>     // Range finder library
//#include <AP_OpticalFlow.h> // Optical Flow library
#include <ModeFilter.h>
//#include <AP_Relay.h>           // APM relay
//#include <GCS_MAVLink.h>     // MAVLink GCS definitions
#include <memcheck.h>
#include <ControlAltura_mio.h>
#include <ControlRoll.h>
#include <ControlPitch.h>
#include <ControlYaw.h>
#include <ControlVelocidad_mio.h>
#include <ControlPosicion_mio.h>

// Configuration
#include "defines.h"
#include "config.h"

// Local modules
#include "Parameters.h"
//#include "GCS.h"

////////////////////////////////////////////////////////////////////////////////
// Serial ports
////////////////////////////////////////////////////////////////////////////////
//
// Note that FastSerial port buffers are allocated at ::begin time,
// so there is not much of a penalty to defining ports that we don't
// use.
//
FastSerialPort0(Serial);        // FTDI/console
FastSerialPort1(Serial1);       // GPS port
FastSerialPort3(Serial3);       // Telemetry port

////////////////////////////////////////////////////////////////////////////////
// Parameters
////////////////////////////////////////////////////////////////////////////////
//
// Global parameters are all contained within the 'g' class.
//
static Parameters      g;


////////////////////////////////////////////////////////////////////////////////
// prototypes
//static void update_events(void);


////////////////////////////////////////////////////////////////////////////////
// Sensors
////////////////////////////////////////////////////////////////////////////////
//
// There are three basic options related to flight sensor selection.
//
// - Normal flight mode.  Real sensors are used.
// - HIL Attitude mode.  Most sensors are disabled, as the HIL
//   protocol supplies attitude information directly.
// - HIL Sensors mode.  Synthetic sensors are configured that
//   supply data from the simulation.
//

// All GPS access should be through this pointer.
```

```
static GPS          *g_gps;

#if HIL_MODE == HIL_MODE_DISABLED

        // real sensors
        AP_ADC_ADS7844          adc;
    AP_Compass_HMC5843      compass(Parameters::k_param_compass);
    AP_Variador_mio *Vari = new AP_Variador_mio[4];

#endif // HIL MODE

#if HIL_MODE != HIL_MODE_ATTITUDE
        #if HIL_MODE != HIL_MODE_SENSORS
                // Normal
                AP_IMU_Oilpan imu(&adc, Parameters::k_param_IMU_calibration);
        #else
                // hil imu
                AP_IMU_Shim imu;
        #endif
        // normal dcm
        AP_DCM  dcm(&imu, g_gps);
#endif

////////////////////////////////////////////////////////////////////////////////
// SONAR selection
////////////////////////////////////////////////////////////////////////////////
//
ModeFilter sonar_mode_filter;

#if SONAR_TYPE == MAX_SONAR_XL
        AP_RangeFinder_MaxsonarXL sonar(&adc, &sonar_mode_filter);//(SONAR_PORT, &adc);
#else
    #error Unrecognised SONAR_TYPE setting.
#endif

// agmatthews USERHOOKS
////////////////////////////////////////////////////////////////////////////////
// User variables
////////////////////////////////////////////////////////////////////////////////
#ifdef USERHOOK_VARIABLES
#include USERHOOK_VARIABLES
#endif

////////////////////////////////////////////////////////////////////////////////
// Global variables
////////////////////////////////////////////////////////////////////////////////

/* Radio values
Channel assignments
1       Ailerons (rudder if no ailerons)
2       Elevator
3       Throttle
4       Rudder (if we have ailerons)
5       Mode - 3 position switch
6       User assignable
7       trainer switch - sets throttle nominal (toggle switch), sets accels to Level (hold > 1 second)
8       TBD
*/

static int16_t          sonar_alt;

// System Timers
// --------------
static uint32_t         fast_loopTimer;         // Time in miliseconds of main control loop

static uint32_t fast_fast_loopTimer;    // Time in miliseconds of main control loop

static byte medium_loopCounter; // Counters for branching from main control loop to slower loops

static uint32_t  fiftyhz_loopTimer;

static byte     slow_loopCounter;
static int16_t  superslow_loopCounter;
static byte             simple_timer;           // for limiting the execution of flight mode thingys


//static float  dTnav;          // Delta Time in milliseconds for navigation computations
//static uint32_t         nav_loopTimer;                  // used to track the elapsed ime for GPS nav
```

```
static byte                             counter_one_herz;
static bool                             GPS_enabled     = false;
//static bool                   new_radio_frame;

//AP_Relay relay;


////////////////////////////////////////////////////////////////////////////////

// Variables de Guillermo:

// Variables auxiliares para telemetria:
#define  CHAR_LEN  15
int C = 10;


// DEFINICION DE DURACION DE FASES TEMPORIZADAS:


#define     TIEMPO_PREV          1

#define     TIEMPO_VELBC         1

#define     TIEMPO_RAMP          0.1

#define     TIEMPO_ELEV          0.1

#define     TIEMPO_APAG          0.75

#define     TIEMPO_DESC          0.5

#define     TIEMPO_RESET         2000

unsigned long TI_FASE_ACT = 0;
unsigned long TI_FASE_SIG = 0;
unsigned long TI_RESET = 0;

// DESPEGUE y ATERRIZAJE:

// Altura objetivo para el despegue:
//#define     H_OBJ_DESP    80
// Altura objetivo para el aterrizaje:
//#define     H_OBJ_ATER    34

// Numero de baterias LiPo que lleva el quadrotor:

#define N_BATERIAS 1


// Limitaciones de cada motor:

#define  VEL_MIN   8.4438e-5             // Velocidad minima del motor (velocidad minima en zona muerta)
#define  VEL_MAX   4.3668e-4             // Velocidad maxima del motor (para 90% de actuacion)

#define  ACT_MIN   6.                     // Actuacion minima del motor en %(por encima de zona muerta)
#define  ACT_MAX   90.                    // Actuacion maxima del motor en %

#define  EMP_MIN   0.4018                // Empuje minimo del motor en N (por encima de zona muerta)
#define  EMP_MAX   12.838                // Empuje maximo del motor en N

// Bandera para control de velocidad:
#define          CV      true


// Periodo de ejecucion del "fast loop" en s:

#define TM_fastloop 0.015

// Control esclavo de velocidad de los motores:

#define     TI_CV_     35.e-3           /* Tiempo integral */
#define     TM_CV_     TM_fastloop      /* Tiempo de muestreo */
#define     KP_CV_     2.3540e5         /* Ganancia proporcional */
#define     TINTAW_CV_    0.07          /* Tiempo integral del anti wind up */
#define     TAU_AF_CV_    15.e-3        /* Constante de tiempo de filtro adicional para senal de control */
```

```
#define   V_MIN_REF_  VEL_MIN  // Ref min para la velocidad del variador (velocidad minima en zona muerta)
#define   V_MAX_REF_  VEL_MAX  // Ref max para la velocidad del variador (vel max para 90% de actuacion)
#define   PORC_MIN_   ACT_MIN                  // Porcentaje minimo de actuacion en %
#define   PORC_MAX_   ACT_MAX                  // Porcentaje maximo de actuacion en %


// Control maestro de estabilizacion:

#define L  0.31                                 // Longitud de los brazos en m
#define kappa  0.0451                           // Parametro kappa en Nm/N
#define MASA  2.                                 // Masa del quadrotor en Kg

#define gravedad 10.48

ControlRoll       CRoll;
ControlPitch      CPitch;
ControlYaw        CYaw;

float TauRoll = 0.;                                 // Par de roll en Nm

#if N_BATERIAS == 3
  float KP_EST_R = 1.8;
#elif N_BATERIAS == 2
  float KP_EST_R = 1.5;
#elif N_BATERIAS == 1
  float KP_EST_R = 1.32;
#else
  float KP_EST_R = 1.21;
#endif

float   TI_EST_R  =  0.75;                      /* Tiempo integral */
float   TD_EST_R  =  0.75/4.;                   /* Tiempo derivativo */

#define   TM_EST_R_    TM_fastloop              /* Tiempo de muestreo */
#define   TINTAW_EST_R_   0.07                  /* Tiempo integral del anti wind up */
#define ROLL_MIN_REF_ -0.75                     // Referencia minima para el angulo roll en rad (-20 )
#define ROLL_MAX_REF_ 0.75                      // Referencia maxima para el angulo roll en rad (+20 )
#define T_MIN_ROLL_ -3.8552                         // Par de roll minimo en Nm
#define T_MAX_ROLL_ 3.8552                         // Par de roll maximo en Nm

float TauPitch = 0.;                                // Par de pitch en Nm

#if N_BATERIAS == 3
  float KP_EST_P = 1.8;
#elif N_BATERIAS == 2
  float KP_EST_P = 1.5;
#elif N_BATERIAS == 1
  float KP_EST_P = 1.32;
#else
  float KP_EST_P = 1.21;
#endif

float   TI_EST_P  =  0.75;                      /* Tiempo integral */
float   TD_EST_P  =  0.75/4.;                   /* Tiempo derivativo */

#define   TM_EST_P_    TM_fastloop              /* Tiempo de muestreo */
#define   TINTAW_EST_P_   0.07                  /* Tiempo integral del anti wind up */
#define PITCH_MIN_REF_ -0.75                      // Referencia minima para el angulo pitch en rad (-20 )
#define PITCH_MAX_REF_ 0.75                       // Referencia maxima para el angulo pitch en rad (+20 )
#define T_MIN_PITCH_  -3.8552          // Par de pitch minimo en Nm
#define T_MAX_PITCH_  3.8552          // Par de pitch maximo en Nm

float TauYaw = 0.;                                  // Par de yaw en Nm

float   KP_EST_Y  =  1.5;                       /* Ganancia proporcional  PARA NINGUNA/UNA/DOS BATERIAS*/
float   TI_EST_Y  =  1.;                        /* Tiempo integral */
float   TD_EST_Y  =  1./4.;                     /* Tiempo derivativo */

#define   TM_EST_Y_    TM_fastloop              /* Tiempo de muestreo */
#define   TINTAW_EST_Y_   0.07                  /* Tiempo integral del anti wind up */
#define YAW_MIN_REF_  -15.                       // Referencia minima para el angulo yaw en rad
#define YAW_MAX_REF_  15.                        // Referencia maxima para el angulo yaw en rad
#define T_MIN_YAW_  -1.1217    // Par de yaw minimo en Nm
#define T_MAX_YAW_  1.1217    // Par de yaw maximo en Nm

// Control de altura:

ControlAltura_mio    CAltura;
```

```
float T = 0.;                                   // Fuerza de empuje en N

#define ALTURA_REF 140.

#if N_BATERIAS == 3
  float KP_CA = 0.02;
#elif N_BATERIAS == 2
  float KP_CA = 0.016;
#elif N_BATERIAS == 1
  float KP_CA = 0.014;
#else
  float KP_CA = 0.011;
#endif

float   TI_CA  =  6.8;                          /* Tiempo integral */
float   TD_CA  =  6.8/4.;                       /* Tiempo derivativo */

#define   TM_CA_    TM_fastloop                 /* Tiempo de muestreo */
#define   TINTAW_CA_  0.07        /* Tiempo integral del anti wind up */
#define   TAU_F_H_  0.2             /* Constante de tiempo de filtro para la medida de la altura */
#define   TAU_F_REF_  6.          /* Constante de tiempo de filtro para la referencia de altura */

#define  H_MIN_REF_  0.                  // Referencia minima de altura en cm
#define  H_MAX_REF_  300.                // Referencia maxima de altura en cm
#define  T_MIN_ALT_  1.6           // Fuerza de empuje minima en N
#define  T_MAX_ALT_  52.           // Fuerza de empuje maxima en N

#if N_BATERIAS == 3
  #define  T_OBJ  30.
#elif N_BATERIAS == 2
  #define  T_OBJ  27.
#elif N_BATERIAS == 1
 #define  T_OBJ  21
#else
  #define  T_OBJ  19.
#endif

// Control de posicion 2D:

ControlPosicion_mio     CPosX;

float       posx = 0.;                          // Posicion deseada en el eje local X

#if N_BATERIAS == 3
  float KP_POSX = 0.105;
#elif N_BATERIAS == 2
  float KP_POSX = 0.0845;
#elif N_BATERIAS == 1
  float KP_POSX = 0.0719;
#else
  float KP_POSX = 0.0583;
#endif

float   TI_POSX  =  30.8;                       /* Tiempo integral */
float   TD_POSX  =  7.7;                /* Tiempo derivativo */


#define   TM_POSX_    TM_fastloop               /* Tiempo de muestreo */
#define   TINTAW_POSX_  0.07                    /* Tiempo integral del anti wind up */

#define   TAU_AF_POSX  0.5  /* Constante de tiempo de filtro adicional para senal de control */

#define   TAU_F_POSX_  0.    //Cte de tiempo de filtro para la medida de la posicion en el eje local X
#define   TAU_F_POSXREF_  6.  //Cte de tiempo de filtro para la referencia de posicion en el eje local X


#define  POSX_MIN_REF_  -1000.                  // Referencia minima de posicion en el eje local X en m
#define  POSX_MAX_REF_  1000.                   // Referencia maxima de posicion en el eje local X en m

#define  FX_MIN  -T_MAX_ALT_*sin(PITCH_MAX_REF_)  // F min de control de posicion en el eje local X en N
#define  FX_MAX  T_MAX_ALT_*sin(PITCH_MAX_REF_)  // F max de control de posicion en el eje local X en N

ControlPosicion_mio     CPosY;

float       posy = 0.;                          // Posicion deseada en el eje local Y

#if N_BATERIAS == 3
  float KP_POSY = 0.105;
#elif N_BATERIAS == 2
```

```
  float KP_POSY = 0.0845;
#elif N_BATERIAS == 1
  float KP_POSY = 0.0719;
#else
  float KP_POSY = 0.0583;
#endif

float   TI_POSY  =  30.8;                       /* Tiempo integral */
float   TD_POSY  =  7.7;                        /* Tiempo derivativo */


#define   TM_POSY_    TM_fastloop                /* Tiempo de muestreo */
#define   TINTAW_POSY_  0.07                     /* Tiempo integral del anti wind up */

#define   TAU_AF_POSY  0.5   /* Constante de tiempo de filtro adicional para senal de control */

#define   TAU_F_POSY_  0.   /* Cte de tiempo de filtro para la medida de la posicion en el eje local Y */
#define   TAU_F_POSYREF_  6.   /* Cte de tiempo de filtro para la ref de posicion en el eje local Y */

#define  POSY_MIN_REF_  -1000.                  // Referencia minima de posicion en el eje local Y en m
#define  POSY_MAX_REF_  1000.                   // Referencia maxima de posicion en el eje local Y en m

#define  FY_MIN  -T_MAX_ALT_*sin(ROLL_MAX_REF_)  // F min de control de posicion en el eje local X en N
#define  FY_MAX  T_MAX_ALT_*sin(ROLL_MAX_REF_)  // F max de control de posicion en el eje local X en N

// angulos de Euler:

float roll_corr;
float pitch_corr;
float yaw_corr;

// Derivada temporal de los angulos de Euler:

Vector3f rpy_p;

// Velocidades lineales:

Vector3f  veloc;

// Aceleraciones lineales:

Vector3f  accel;  // Vector de aceleraciones lineales expresado respecto a los ejes inerciales

// Lectura de altura del sonar por I2c:

static int16_t          sonar_I2C_alt = 0;

// Medida de altura corregida:

float                   med_alt = 0.;

// Estado del QR:

uint8_t     ESTADO = 4;
uint8_t     ESTADO_ANT = 4;

// Banderas de caminos distintos entre estados:

bool flag_control2D = false;

// Localizaciones GPS:

struct localizacion{

  float latit;
  float longi;
  float XN;
  float YE;

};

struct localizacion base = {0.,0.,0.,0.};
struct localizacion point = {0.,0.,2.,2.};

// Cabeceras de paquetes de telemetria:

#define HEAD_BYTE1 0xA3
#define HEAD_BYTE2 0x95
```

```
#define CHECK_SUM1 0xA2
#define CHECK_SUM2 0x4E

bool guardado_DF = false;
bool guardar_DF = false;
bool borrar_DF = false;

long n_paq_DF = 0;

// Temporizadores personalizados:

long unsigned timer_mio;
long unsigned timer_telemetria;
long unsigned timer_telemetria_lenta;
float period = 0.;
float period_lento = 0.;

////////////////////////////////////////////////////////////////////////////////
// Top-level logic
////////////////////////////////////////////////////////////////////////////////

void setup()
{
        memcheck_init();

        init_ardupilot_reducido();

        inicializacion_variables();

        inicializacion_controladores();

        init_variadores();

        timer_mio = micros();
        timer_telemetria = micros();
        timer_telemetria_lenta = micros();
        TI_FASE_ACT = micros();
        TI_RESET = micros();

}

static void init_ardupilot_reducido()
{

        bool flag = false;
        char caracter = '0';

        Serial.begin(SERIAL0_BAUD, 128, 128);

        #if GPS_PROTOCOL != GPS_PROTOCOL_IMU
          Serial1.begin(38400, 128, 16);
        #endif

        adc.filter_result = true;
        adc.Init();

        // Do GPS init
        g_gps = &g_gps_driver;
        g_gps->init();                  // GPS Initialization
        g_gps->callback = delay;

        Wire.begin();

        if(g.compass_enabled){
                compass.set_orientation(MAG_ORIENTATION);  // set compass's orientation on aircraft
                dcm.set_compass(&compass);
                compass.init();
                compass.get_offsets();  // load offsets to account for airframe magnetic interference
        }

        GPS_enabled = false;

        #if HIL_MODE == HIL_MODE_DISABLED
        // Read in the GPS
        for (byte counter = 0; ; counter++) {
                g_gps->update();
                if (g_gps->status() != 0){
                        GPS_enabled = true;
                        break;
```

```
                }

                if (counter >= 2) {
                        GPS_enabled = false;
                        break;
            }
        }
        #else
                GPS_enabled = true;
        #endif

        // lengthen the idle timeout for gps Auto_detect
        // ---------------------------------------------
        g_gps->idleTimeout = 20000;

        imu.init_gyro(delay);

        DataFlash.Init();

        Serial3.begin(57600);

        Serial3.println("Desea borrar los datos de la memoria flash?: [y/n] ");
        while(flag==false){
          caracter = Serial3.read();
          if(caracter == 'y'){
            Serial3.println("Borrando datos...");
            erase_logs();
            flag = true;
          }else if(caracter == 'n'){
            Serial3.println("No borrar datos");
            flag = true;
          }
        }

        DataFlash.StartWrite(1);
}

void inicializacion_variables()
{

       CAltura.T_conmu = 0;
       CAltura.T_obj = T_OBJ;

}

void inicializacion_controladores()
{

  // Controlador de posicion en el eje local X:

  CPosX.TM = TM_POSX_;

  CPosX.TI = TI_POSX;
  CPosX.KP = KP_POSX;
  CPosX.TD = TD_POSX;

  CPosX.TINTAW = TINTAW_POSX_;
  CPosX.TAU_AF = TAU_AF_POSX;

  CPosX.TAU_F_POS = TAU_F_POSX_;
  CPosX.TAU_F_POSREF = TAU_F_POSXREF_;

  CPosX.POS_MIN = POSX_MIN_REF_;
  CPosX.POS_MAX = POSX_MAX_REF_;
  CPosX.F_MIN = FX_MIN;
  CPosX.F_MAX = FX_MAX;

  CPosX.actualiza_coef();

  // Controlador de posicion en el eje local Y:

  CPosY.TM = TM_POSY_;

  CPosY.TI = TI_POSY;
  CPosY.KP = KP_POSY;
  CPosY.TD = TD_POSY;

  CPosY.TINTAW = TINTAW_POSY_;
  CPosY.TAU_AF = TAU_AF_POSY;
```

```
CPosY.TAU_F_POS = TAU_F_POSY_;
CPosY.TAU_F_POSREF = TAU_F_POSYREF_;

CPosY.POS_MIN = POSY_MIN_REF_;
CPosY.POS_MAX = POSY_MAX_REF_;
CPosY.F_MIN = FY_MIN;
CPosY.F_MAX = FY_MAX;

CPosY.actualiza_coef();

// Controlador de altura:

CAltura.TM = TM_CA_;

CAltura.TI = TI_CA;
CAltura.KP = KP_CA;
CAltura.TD = TD_CA;

CAltura.TINTAW = TINTAW_CA_;

CAltura.TAU_F_H = TAU_F_H_;
CAltura.TAU_F_REF = TAU_F_REF_;

CAltura.H_MIN = H_MIN_REF_;
CAltura.H_MAX = H_MAX_REF_;
CAltura.T_MIN = T_MIN_ALT_;
CAltura.T_MAX = T_MAX_ALT_;

CAltura.actualiza_coef();

// Controlador de roll:

CRoll.TM = TM_EST_R_;

CRoll.TI = TI_EST_R;
CRoll.KP = KP_EST_R;
CRoll.TD = TD_EST_R;

CRoll.TINTAW = TINTAW_EST_R_;

CRoll.ANG_MIN = ROLL_MIN_REF_;
CRoll.ANG_MAX = ROLL_MAX_REF_;
CRoll.T_MIN = T_MIN_ROLL_;
CRoll.T_MAX = T_MAX_ROLL_;

CRoll.actualiza_coef();

// Controlador de pitch:

CPitch.TM = TM_EST_P_;

CPitch.TI = TI_EST_P;
CPitch.KP = KP_EST_P;
CPitch.TD = TD_EST_P;

CPitch.TINTAW = TINTAW_EST_P_;

CPitch.ANG_MIN = PITCH_MIN_REF_;
CPitch.ANG_MAX = PITCH_MAX_REF_;
CPitch.T_MIN = T_MIN_PITCH_;
CPitch.T_MAX = T_MAX_PITCH_;

CPitch.actualiza_coef();

// Controlador de yaw:

CYaw.TM = TM_EST_Y_;

CYaw.TI = TI_EST_Y;
CYaw.KP = KP_EST_Y;
CYaw.TD = TD_EST_Y;

CYaw.TINTAW = TINTAW_EST_Y_;

CYaw.ANG_MIN = YAW_MIN_REF_;
CYaw.ANG_MAX = YAW_MAX_REF_;
CYaw.T_MIN = T_MIN_YAW_;
CYaw.T_MAX = T_MAX_YAW_;
```

```
  CYaw.actualiza_coef();

  // Controladores de velocidad de bajo nivel:

  for(int i=0;i<4;i++){

    Vari[i].TM = TM_CV_;

    Vari[i].TI = TI_CV_;
    Vari[i].KP = KP_CV_;

    Vari[i].TINTAW = TINTAW_CV_;
    Vari[i].TAU_AF = TAU_AF_CV_;

    Vari[i].V_MIN = V_MIN_REF_;
    Vari[i].V_MAX = V_MAX_REF_;
    Vari[i].PORC_MIN = PORC_MIN_;
    Vari[i].PORC_MAX = PORC_MAX_;

    Vari[i].actualiza_coef();
  }
}

static void inicializacion_GPS()
{

        if(GPS_enabled){

          bool flag = false;
          int n = 0;
          int N = 0;
          float suma_latit = 0.;
          float suma_longi = 0.;

          while(flag==false && N<100){

            update_GPS();
            read_AHRS();

            if(g_gps->latitude!=0){

              flag = true;
              suma_latit += (float)g_gps->latitude/ 1.0e7;
              suma_longi += (float)g_gps->longitude/ 1.0e7;
              N++;

            }else if(flag==false){

              if(n==20){
                n = 0;
                Serial3.print("No GPS!! \tNumero de satelites: ");
                Serial3.println(g_gps->num_sats,DEC);
              }else{
                n++;
              }
            }
            delay(100);
          }
          base.latit = suma_latit/N;
          base.longi = suma_longi/N;
        }
}

void init_variadores() {

  int i=0;
  int j=0;

  for(i=0;i<4;i++){

    Vari[i].numero_vari = i+1;

    for(j=0;j<34;j++){
      delay(10);
      Vari[i].accion_control = 0;
      Vari[i].write();
    }
  }
```

```
}

void loop()
{
                int32_t timer                   = micros();

                if((timer - fast_fast_loopTimer) >= 5000) {
                  fast_fast_loopTimer           = timer;
                  fast_fast_loop();
                }


        // We want this to execute fast
        // ----------------------------
              if ((timer - fast_loopTimer) >= TM_fastloop*1e6){
                //PORTK |= B00010000;
                fast_loopTimer          = timer;
                // Execute the fast loop
                // ---------------------
                fast_loop();
        }
        //PORTK &= B11101111;

        if ((timer - fiftyhz_loopTimer) >= 20000) {
                fiftyhz_loopTimer               = timer;
                //PORTK |= B01000000;

                // perform 10hz tasks
                medium_loop();

                // Stuff to run at full 50hz, but after the loops
                fifty_hz_loop();

                counter_one_herz++;

                if(counter_one_herz == 50){
                        super_slow_loop();
                        counter_one_herz = 0;
                }
        }
}

static void fast_fast_loop()
{
          lee_telemetria();
}

// Main loop

static void fast_loop()
{
        timer_mio = micros();

        read_AHRS();
        read_AHRS();

        // Calculo de las derivadas temporales de los angulos de Euler:

        roll_corr = dcm.roll;                           // Medida del roll en rad procedente de la IMU
        pitch_corr = dcm.pitch;                         // Medida del pitch en rad procedente de la IMU
        yaw_corr = dcm.yaw;                             // Medida del yaw en rad procedente de la IMU

        roll_corr = vueltas(roll_corr,1);
        pitch_corr = vueltas(pitch_corr,2);
        yaw_corr = vueltas(yaw_corr,3);

        // Velocidades y aceleraciones:
    // Vector de velocidades angulares procedente de la IMU expresado respecto a los ejes moviles
        Vector3f  gyro = dcm.get_gyro();
  // Vector de aceleraciones lineales procedente de la IMU expresado respecto a los ejes moviles
        Vector3f  accelg = dcm.get_accel();

        float c1 = cos(roll_corr);
        float s1 = sin(roll_corr);

        float c2 = cos(pitch_corr);
        float s2 = sin(pitch_corr);
```

```
        float c3 = cos(yaw_corr);
        float s3 = sin(yaw_corr);

        rpy_p.x = gyro.x + s2*s1/c2*gyro.y + s2*c1/c2*gyro.z;
        rpy_p.y = c1*gyro.y - s1*gyro.z;
        rpy_p.z = s1/c2*gyro.y + c1/c2*gyro.z;

        Vector3f glocal(-s2*gravedad, s1*c2*gravedad, c1*c2*gravedad);

        Vector3f accel_libre = accelg + glocal;

        accel.x = c2*accel_libre.x + s2*s1*accel_libre.y + s2*c1*accel_libre.z;
        accel.y = c1*accel_libre.y - s1*accel_libre.z;
        accel.z = -s2*accel_libre.x + c2*s1*accel_libre.y + c2*c1*accel_libre.z;

        veloc += accel*TM_fastloop;

        CRoll.angk = roll_corr;
        CRoll.der_angk = rpy_p.x;

        CPitch.angk = pitch_corr;
        CPitch.der_angk = rpy_p.y;

        CYaw.angk = yaw_corr;
        CYaw.der_angk = rpy_p.z;

//el numero es la distancia del sonar dividido entre raiz de 2 en cm
        med_alt = (float)sonar_alt - 30.4056*(s2+c2*s1);

        CAltura.hk = med_alt;

        CPosX.posk = point.XN*c3 + point.YE*s3;
        CPosY.posk = -point.XN*s3 + point.YE*c3;

        // Medida de velocidad de variadores:
        read_variadores();

        // write out the servo PWM values
        // ------------------------------

        output_motors_disarmed();

        envia_telemetria_DF();
}

float vueltas(float ang, int i)
{

  static float roll_ant;
  static int roll_n;
  static float pitch_ant;
  static int pitch_n;
  static float yaw_ant;
  static int yaw_n;

  float result;
  float ang_ant;
  float n;

  switch(i)
  {
    case 1:
            ang_ant = roll_ant;
            n = roll_n;
            break;

    case 2:
            ang_ant = pitch_ant;
            n = pitch_n;
            break;

    case 3:
            ang_ant = yaw_ant;
            n = yaw_n;
            break;

    default:
            break;
  }
```

```
  if(abs(ang-ang_ant)>(3.14159/2.)){
    if(ang>ang_ant){
      n--;
    }else{
      n++;
    }
  }

  result = ang + n*2.*3.14159;

  switch(i)
  {
    case 1:
            roll_ant = ang;
            roll_n = n;
            break;

    case 2:
            pitch_ant = ang;
            pitch_n = n;
            break;

    case 3:
            yaw_ant = ang;
            yaw_n = n;
            break;

    default:
            break;
  }

  return(result);

}

void read_variadores()
{
  int i;
  uint32_t result = 0;

  for (i=0;i<4;i++)
  {
    Vari[i].read();
    result = Vari[i].medida;
    Vari[i].velocity = (float)(1./((float)(result)));
  }

}

void lee_telemetria(){

        int j;
        int resultado = 0;
        int char_len = 0;
        char ascii[CHAR_LEN];

        // Se esperan entradas por la telemetria:

           if(Serial3.available()>0){
            char_len = Serial3.available();
            ascii[0] = (char)(Serial3.read());

            if(ascii[0]>='0' && ascii[0]<='9'){
              ESTADO = (uint8_t)(ascii[0] - '0');

            }else if(ascii[0] == 'q'){
              TI_EST_R*=1.1;
              TD_EST_R = TI_EST_R/4;
              CRoll.Set_TD(TD_EST_R);
              CRoll.Set_TI(TI_EST_R);
            }else if(ascii[0] == 'a'){
              TI_EST_R/=1.1;
              TD_EST_R = TI_EST_R/4;
              CRoll.Set_TD(TD_EST_R);
              CRoll.Set_TI(TI_EST_R);
            }else if(ascii[0] == 'w'){
              KP_EST_R*=1.1;
              CRoll.Set_KP(KP_EST_R);
```

```
}else if(ascii[0] == 's'){
  KP_EST_R/=1.1;
  CRoll.Set_KP(KP_EST_R);
}else if(ascii[0] == 'e'){
  TD_EST_R*=1.1;
  TI_EST_R = TD_EST_R*4;
  CRoll.Set_TI(TI_EST_R);
  CRoll.Set_TD(TD_EST_R);
}else if(ascii[0] == 'd'){
  TD_EST_R/=1.1;
  TI_EST_R = TD_EST_R*4;
  CRoll.Set_TI(TI_EST_R);
  CRoll.Set_TD(TD_EST_R);

}else if(ascii[0] == 'r'){
  TI_POSY*=1.1;
  TD_POSY = TI_POSY/4;
  CPosY.Set_TI(TI_POSY);
  CPosY.Set_TD(TD_POSY);
}else if(ascii[0] == 'f'){
  TI_POSY/=1.1;
  TD_POSY = TI_POSY/4;
  CPosY.Set_TI(TI_POSY);
  CPosY.Set_TD(TD_POSY);
}else if(ascii[0] == 't'){
  KP_POSY*=1.1;
  CPosY.Set_KP(KP_POSY);
}else if(ascii[0] == 'g'){
  KP_POSY/=1.1;
  CPosY.Set_KP(KP_POSY);
}else if(ascii[0] == 'y' && ESTADO!=0 && ESTADO!=11){
  TD_POSY*=1.1;
  TI_POSY = TD_POSY*4;
  CPosY.Set_TI(TI_POSY);
  CPosY.Set_TD(TD_POSY);
}else if(ascii[0] == 'y' && ESTADO == 0){
  Serial3.println("Guardando datos ...");
  guardar_DF = true;
  guardado_DF = false;
  ESTADO = 11;
}else if(ascii[0] == 'y' && ESTADO == 11){
  Serial3.println("Borrando datos ...");
  borrar_DF = true;
}else if(ascii[0] == 'h'){
  TD_POSY/=1.1;
  TI_POSY = TD_POSY*4;
  CPosY.Set_TI(TI_POSY);
  CPosY.Set_TD(TD_POSY);

}else if(ascii[0] == 'u'){
  TI_EST_Y*=1.1;
  TD_EST_Y = TI_EST_Y/4;
  CYaw.Set_TD(TD_EST_Y);
  CYaw.Set_TI(TI_EST_Y);
}else if(ascii[0] == 'j'){
  TI_EST_Y/=1.1;
  TD_EST_Y = TI_EST_Y/4;
  CYaw.Set_TD(TD_EST_Y);
  CYaw.Set_TI(TI_EST_Y);
}else if(ascii[0] == 'i'){
  KP_EST_Y*=1.1;
  CYaw.Set_KP(KP_EST_Y);
  resultado = C;
}else if(ascii[0] == 'k'){
  KP_EST_Y/=1.1;
  CYaw.Set_KP(KP_EST_Y);
}else if(ascii[0] == 'o'){
  TD_EST_Y*=1.1;
  TI_EST_Y = TD_EST_Y*4;
  CYaw.Set_TI(TI_EST_Y);
  CYaw.Set_TD(TD_EST_Y);
}else if(ascii[0] == 'l'){
  TD_EST_Y/=1.1;
  TI_EST_Y = TD_EST_Y*4;
  CYaw.Set_TI(TI_EST_Y);
  CYaw.Set_TD(TD_EST_Y);

}else if(ascii[0] == 'z'){
  TI_CA*=1.1;
```

```
            TD_CA = TI_CA/4;
            CAltura.Set_TD(TD_CA);
            CAltura.Set_TI(TI_CA);
          }else if(ascii[0] == 'x'){
            TI_CA/=1.1;
            TD_CA = TI_CA/4;
            CAltura.Set_TD(TD_CA);
            CAltura.Set_TI(TI_CA);
          }else if(ascii[0] == 'c'){
            KP_CA*=1.1;
            CAltura.Set_KP(KP_CA);
          }else if(ascii[0] == 'v'){
            KP_CA/=1.1;
            CAltura.Set_KP(KP_CA);
          }else if(ascii[0] == 'b'){
            TD_CA*=1.1;
            TI_CA = TD_CA*4;
            CAltura.Set_TI(TI_CA);
            CAltura.Set_TD(TD_CA);
          }else if(ascii[0] == 'n' && ESTADO!=0 && ESTADO!=11){
            TD_CA/=1.1;
            TI_CA = TD_CA*4;
            CAltura.Set_TI(TI_CA);
            CAltura.Set_TD(TD_CA);

          }else if(ascii[0] == 'n' && ESTADO == 0){
            Serial3.println("No guardar datos");
            guardar_DF = false;
            guardado_DF = true;
            ESTADO = 11;
          }else if(ascii[0] == 'n' && ESTADO == 11){
            Serial3.println("No borrar datos");
            borrar_DF = false;

          }else if(ascii[0] == ' '){
            posx += 10.;
          }else if(ascii[0] == '-'){
            posx -= 10.;
          }else if(ascii[0] == ','){
            posy += 10.;
          }else if(ascii[0] == '.'){
            posy -= 10.;

          }else if(ascii[0] == 'm'){
            CAltura.hrefk+=10.;

          }else if(ascii[0] == 'p'){
            CAltura.hrefk-=50.;
          }
        }
}

void envia_telemetria()
{

  int i;

  Serial3.println(med_alt/100.,2);

  Serial3.println(CRoll.angk,8);
  Serial3.println(CPitch.angk,8);
  Serial3.println(CYaw.angk,8);
  Serial3.println(rpy_p.x,8);
  Serial3.println(rpy_p.y,8);
  Serial3.println(rpy_p.z,8);

  Serial3.println(CRoll.angrefk,8);
  Serial3.println(CRoll.der_angrefk,8);

  Serial3.println(CPitch.angrefk,8);
  Serial3.println(CPitch.der_angrefk,8);

  Serial3.println(CYaw.angrefk,8);
  Serial3.println(CYaw.der_angrefk,8);

  Serial3.println(CAltura.hrefk/100.,2);

  Serial3.println(T,4);
  Serial3.println(TauRoll,4);
```

```
  Serial3.println(TauPitch,4);
  Serial3.println(TauYaw,4);

  for(i=0;i<4;i++){
    Serial3.println(Vari[i].f,4);
  }

  for(i=0;i<4;i++){
    Serial3.println(Vari[i].sp_velocity*1e4,4);
  }

  for(i=0;i<4;i++){
    Serial3.println(Vari[i].accion_control,4);
  }

  Serial3.println(accel.x,4);
  Serial3.println(accel.y,4);

  Serial3.println(CPosX.posk,4);
  Serial3.println(CPosY.posk,4);

  Serial3.println(posx,4);
  Serial3.println(posy,4);

  Serial3.println(CPosX.Freal,4);
  Serial3.println(CPosY.Freal,4);

  Serial3.println(KP_POSX,4);
  Serial3.println(KP_POSY,4);

  Serial3.println(TI_POSX,2);
  Serial3.println(TI_POSY,2);

  Serial3.println(TD_POSX,2);
  Serial3.println(TD_POSY,2);

  Serial3.println((float)ESTADO);

  period_lento = (float)((float)((micros()-timer_telemetria_lenta))/1000);
  Serial3.println(period_lento*1000);
  timer_telemetria_lenta = micros();

}

void envia_telemetria_imprescindible()
{
  if (ESTADO!=0 && ESTADO!=11){

    Serial3.println(KP_POSX,4);
    Serial3.println(KP_POSY,4);

    Serial3.println(TI_POSX,2);
    Serial3.println(TI_POSY,2);

    Serial3.println(TD_POSX,2);
    Serial3.println(TD_POSY,2);

    Serial3.println((float)g_gps->num_sats, 2);
    Serial3.println((float)g_gps->latitude / 1.0e7, 7);
    Serial3.println((float)g_gps->longitude / 1.0e7, 7);

    Serial3.println((float)ESTADO);

    period_lento = (float)((float)((micros()-timer_telemetria_lenta))/1000);
    Serial3.println(period_lento*1000);

    timer_telemetria_lenta = micros();
  }

}

void envia_telemetria_DF()
{

  int j;

  if(ESTADO!=0 && ESTADO!=11){

    DataFlash.WriteByte(HEAD_BYTE1);
```

```
    DataFlash.WriteByte(HEAD_BYTE2);

    DataFlash.WriteLong((long)med_alt);

    DataFlash.WriteLong((long)(CRoll.angk*1.e8));
    DataFlash.WriteLong((long)(CPitch.angk*1.e8));
    DataFlash.WriteLong((long)(CYaw.angk*1.e8));
    DataFlash.WriteLong((long)(rpy_p.x*1.e8));
    DataFlash.WriteLong((long)(rpy_p.y*1.e8));
    DataFlash.WriteLong((long)(rpy_p.z*1.e8));

    DataFlash.WriteLong((long)(CRoll.angrefk*1.e8));
    DataFlash.WriteLong((long)(CRoll.der_angrefk*1.e8));

    DataFlash.WriteLong((long)(CPitch.angrefk*1.e8));
    DataFlash.WriteLong((long)(CPitch.der_angrefk*1.e8));

    DataFlash.WriteLong((long)(CYaw.angrefk*1.e8));
    DataFlash.WriteLong((long)(CYaw.der_angrefk*1.e8));

    DataFlash.WriteLong((long)CAltura.hrefk);

    DataFlash.WriteLong((long)(T*1.e4));
    DataFlash.WriteLong((long)(TauRoll*1.e4));
    DataFlash.WriteLong((long)(TauPitch*1.e4));
    DataFlash.WriteLong((long)(TauYaw*1.e4));

    for(j=0;j<4;j++){
      DataFlash.WriteLong((long)(Vari[j].f*1.e4));
    }
    for(j=0;j<4;j++){
      DataFlash.WriteLong((long)(Vari[j].sp_velocity*1.e8));
    }
    for(j=0;j<4;j++){
      DataFlash.WriteLong((long)(Vari[j].accion_control*1.e4));
    }

    DataFlash.WriteLongLong((long long)(base.longi*1.e9));
    DataFlash.WriteLongLong((long long)(base.latit*1.e8));

    DataFlash.WriteLongLong((long long)(point.longi*1.e9));
    DataFlash.WriteLongLong((long long)(point.latit*1.e8));

    DataFlash.WriteLong((long)(CPosX.posk*1.e4));
    DataFlash.WriteLong((long)(CPosY.posk*1.e4));

    DataFlash.WriteLong((long)(posx*1.e4));
    DataFlash.WriteLong((long)(posy*1.e4));

    DataFlash.WriteLong((long)(CPosX.Freal*1.e4));
    DataFlash.WriteLong((long)(CPosY.Freal*1.e4));

    DataFlash.WriteLong((long)(KP_EST_R*1.e4));
    DataFlash.WriteLong((long)(KP_EST_P*1.e4));

    DataFlash.WriteLong((long)(TI_EST_R*1.e4));
    DataFlash.WriteLong((long)(TI_EST_P*1.e4));

    DataFlash.WriteLong((long)(TD_EST_R*1.e4));
    DataFlash.WriteLong((long)(TD_EST_P*1.e4));

    DataFlash.WriteInt((int)(ESTADO));

    period = (float)((float)((micros()-timer_telemetria))/1000);
    DataFlash.WriteLong((long)(period*1000));

    DataFlash.WriteByte(CHECK_SUM1);      // 2 bytes of checksum (example)
    DataFlash.WriteByte(CHECK_SUM2);

    timer_telemetria = micros();

    n_paq_DF++;
  }
}

void envia_telemetria_GPS()
{

  if (ESTADO!=0 && ESTADO!=11){
```

```
    Serial3.println(base.latit, 7);
    Serial3.println(base.longi, 7);

    Serial3.println(point.latit, 7);
    Serial3.println(point.longi, 7);

    Serial3.println((float)g_gps->num_sats, 2);

    Serial3.println(point.XN,8);
    Serial3.println(point.YE,8);

    Serial3.println(CPosX.posk,8);
    Serial3.println(CPosY.posk,8);

    Serial3.println((float)ESTADO);

    period_lento = (float)((float)((micros()-timer_telemetria_lenta))/1000);
    Serial3.println(period_lento*1000);
    timer_telemetria_lenta = micros();

  }
}

static void medium_loop()
{
        static bool flag;
        float yaw;

        // This is the start of the medium (10 Hz) loop pieces
        // -----------------------------------
        switch(medium_loopCounter) {

                // This case deals with the GPS and Compass
                //-----------------------------------------
                case 0:
                        medium_loopCounter++;
                        if(GPS_enabled){

                                update_GPS();

                                if(g_gps->latitude!=0){

                                    point.latit = (float) g_gps->latitude / 1.0e7;
                                    point.longi = (float) g_gps->longitude/ 1.0e7;

                                    convert_GPS(&base, &point);

                                }

                        }

                        #if HIL_MODE != HIL_MODE_ATTITUDE        // don't execute in HIL mode
                                if(g.compass_enabled){
                                        compass.read(); // Read magnetometer
                                        // Calculate heading
                                        compass.calculate(dcm.get_dcm_matrix());
                                        compass.null_offsets(dcm.get_dcm_matrix());
                                }
                        #endif

                        // auto_trim, uses an auto_level algorithm

                        // record throttle output
                        // ------------------------------
                        break;

                // This case performs some navigation computations
                //------------------------------------------------
                case 1:
                        medium_loopCounter++;
                        break;

                // command processing
                //--------------------
                case 2:
                        medium_loopCounter++;

                        break;
```

```
                // This case deals with sending high rate telemetry
                //-----------------------------------------------------
                case 3:
                        medium_loopCounter++;
                        break;

                // This case controls the slow loop
                //----------------------------------
                case 4:
                        medium_loopCounter = 0;
                        #if HIL_MODE != HIL_MODE_ATTITUDE
                        #endif

                        slow_loop();
                        break;

                default:
                        // this is just a catch all
                        // ------------------------
                        medium_loopCounter = 0;
                        break;
        }
}

// stuff that happens at 50 hz
// ---------------------------
static void fifty_hz_loop()
{
        if(g.sonar_enabled){
                sonar_alt = sonar.read();
        }
}

static void slow_loop()
{
        // This is the slow (3 1/3 Hz) loop pieces
        //----------------------------------------
        switch (slow_loopCounter){
                case 0:
                        slow_loopCounter++;
                        superslow_loopCounter++;

                        if(superslow_loopCounter > 1200){
                                #if HIL_MODE != HIL_MODE_ATTITUDE
                                #endif
            }
                        break;

                case 1:
                        slow_loopCounter++;

                        envia_telemetria_GPS();

                        break;

                case 2:
                        slow_loopCounter = 0;

                        #if MOTOR_LEDS == 1

                        #endif

                        break;

                default:
                        slow_loopCounter = 0;
                        break;

        }
}

static void update_GPS(void)
{
        g_gps->update();
}

static void read_AHRS(void)
{
```

```
        dcm.update_DCM_fast();
}

static void convert_GPS(struct localizacion *base, struct localizacion *point)
{

  float a = 6378137.;
  float b = 6356752.3142;
  float f = (a-b)/a;

  float phi1 = (base->latit)*3.14159/180.;
  float phi2 = (point->latit)*3.14159/180.;

  float Le = ((point->longi) - (base->longi))*3.14159/180.;

  float U1 = atan((1-f)*tan(phi1));
  float U2 = atan((1-f)*tan(phi2));

  float lambda = Le;
  float lambdap = 2.*3.14159;

  float sinsigma, cossigma, sigma, sinalfa, cos2alfa, cos2sigmam, A, B, u2, deltasigma, s, alfa1, alfa2;

  while(abs(lambda-lambdap) > 1.e-12){
    sinsigma = sqrt((cos(U2)*sin(lambda))*(cos(U2)*sin(lambda)) + (cos(U1)*sin(U2)
     - sin(U1)*cos(U2)*cos(lambda))*(cos(U1)*sin(U2) - sin(U1)*cos(U2)*cos(lambda)));
    cossigma = sin(U1)*sin(U2) + cos(U1)*cos(U2)*cos(lambda);
    sigma = atan2(sinsigma, cossigma);

    if(sigma == 0)
    {
      break;
    }

    sinalfa = cos(U1)*cos(U2)*sin(lambda)/sinsigma;
    cos2alfa = 1 - sinalfa*sinalfa;
    cos2sigmam = cossigma - 2*sin(U1)*sin(U2)/cos2alfa;
    C = (f/16.)*cos2alfa*(4.+f*(4.-3.*cos2alfa));
    lambdap = lambda;
    lambda = Le + (1-C)*f*sinalfa*(sigma + C*sinsigma*(cos2sigmam
     + C*cossigma*(-1 + 2*cos2sigmam*cos2sigmam)));
  }

  if(sigma != 0)
  {

    u2 = cos2alfa*(a*a - b*b)/(b*b);
    A = 1+(u2/16384.)*(4096.+u2*(-768.+u2*(320.-175.*u2)));
    B = (u2/1024.)*(256.+u2*(-128.+u2*(74.-47.*u2)));
    deltasigma = B*sinsigma*(cos2sigmam + (B/4.)*(cossigma*(-1+2*cos2sigmam*cos2sigmam)
    - (B/6.)*cos2sigmam*(-3.+4.*sinsigma*sinsigma)*(-3.+4.*cos2sigmam*cos2sigmam)));
    s = b*A*(sigma-deltasigma);

    alfa1 = atan2(cos(U2)*sin(lambda), cos(U1)*sin(U2) - sin(U1)*cos(U2)*cos(lambda));
    alfa2 = atan2(cos(U1)*sin(lambda), -sin(U1)*cos(U2) + cos(U1)*sin(U2)*cos(lambda));

    point->XN = s*cos(alfa2);
    point->YE = s*sin(alfa2);

  }else{

    point->XN = 0.;
    point->YE = 0.;

  }
}

/*void lee_sonar_i2c()
{

   // Leo el dato disponible:

   // step 3: instruct sensor to return a particular echo reading
   Wire.beginTransmission(0x70); // transmit to device #112
   Wire.send(0x02);              // sets register pointer to echo #1 register (0x02)
   Wire.endTransmission();       // stop transmitting

   // step 4: request reading from sensor
   Wire.requestFrom(0x70, 2);    // request 2 bytes from slave device #112
```

```
    // step 5: receive reading from sensor
  if(2 <= Wire.available())    // if two bytes were received
  {
    sonar_I2C_alt = Wire.receive();  // receive high byte (overwrites previous reading)
    sonar_I2C_alt = sonar_I2C_alt << 8;    // shift high byte to be high 8 bits
    sonar_I2C_alt |= Wire.receive(); // receive low byte as lower 8 bits
    Wire.endTransmission();      // stop transmitting

    // Envio la orden para leer otra vez:

    // step 1: instruct sensor to read echoes
    Wire.beginTransmission(0x70); // transmit to device #112 (0x70)
    // the address specified in the datasheet is 224 (0xE0)
    // but i2c adressing uses the high 7 bits so it's 112 (0x70)
    Wire.send(0x00);             // sets register pointer to the command register (0x00)
    Wire.send(0x51);             // command sensor to measure in cm (0x51)
    // use 0x52 for ping microseconds
    Wire.endTransmission();      // stop transmitting

  }
  Wire.endTransmission();      // stop transmitting
}
*/

void lectura_DF(){

        int       i, tmp_int;
        byte      tmp_byte1, tmp_byte2;
        long      tmp_long;
        long long tmp_longlong;
        int       mili = 1;

        DataFlash.StartRead(1);  // We start reading from page 1

        Serial3.println("Reading initialized");

        for (i = 1; i < n_paq_DF-2; i++){

                tmp_byte1 = DataFlash.ReadByte();
                delay(mili);
                tmp_byte2 = DataFlash.ReadByte();
                delay(mili);

                if ((tmp_byte1 == HEAD_BYTE1) && (tmp_byte2 == HEAD_BYTE2)){

                        // Read 44 longs...

                        tmp_long = DataFlash.ReadLong();
                        Serial3.println((float)tmp_long,2);
                        delay(mili);

                        tmp_long = DataFlash.ReadLong();
                        Serial3.println((float)tmp_long/1.e8,8);
                        delay(mili);
                        tmp_long = DataFlash.ReadLong();
                        Serial3.println((float)tmp_long/1.e8,8);
                        delay(mili);
                        tmp_long = DataFlash.ReadLong();
                        Serial3.println((float)tmp_long/1.e8,8);
                        delay(mili);
                        tmp_long = DataFlash.ReadLong();
                        Serial3.println((float)tmp_long/1.e8,8);
                        delay(mili);
                        tmp_long = DataFlash.ReadLong();
                        Serial3.println((float)tmp_long/1.e8,8);
                        delay(mili);
                        tmp_long = DataFlash.ReadLong();
                        Serial3.println((float)tmp_long/1.e8,8);
                        delay(mili);

                        tmp_long = DataFlash.ReadLong();
                        Serial3.println((float)tmp_long/1.e8,8);
                        delay(mili);
                        tmp_long = DataFlash.ReadLong();
                        Serial3.println((float)tmp_long/1.e8,8);
                        delay(mili);

                        tmp_long = DataFlash.ReadLong();
```

```
Serial3.println((float)tmp_long/1.e8,8);
delay(mili);
tmp_long = DataFlash.ReadLong();
Serial3.println((float)tmp_long/1.e8,8);
delay(mili);

tmp_long = DataFlash.ReadLong();
Serial3.println((float)tmp_long/1.e8,8);
delay(mili);
tmp_long = DataFlash.ReadLong();
Serial3.println((float)tmp_long/1.e8,8);
delay(mili);

tmp_long = DataFlash.ReadLong();
Serial3.println((float)tmp_long,2);
delay(mili);

tmp_long = DataFlash.ReadLong();
Serial3.println((float)tmp_long/1.e4,4);
delay(mili);
tmp_long = DataFlash.ReadLong();
Serial3.println((float)tmp_long/1.e4,4);
delay(mili);
tmp_long = DataFlash.ReadLong();
Serial3.println((float)tmp_long/1.e4,4);
delay(mili);
tmp_long = DataFlash.ReadLong();
Serial3.println((float)tmp_long/1.e4,4);
delay(mili);

tmp_long = DataFlash.ReadLong();
Serial3.println((float)tmp_long/1.e4,4);
delay(mili);
tmp_long = DataFlash.ReadLong();
Serial3.println((float)tmp_long/1.e4,4);
delay(mili);
tmp_long = DataFlash.ReadLong();
Serial3.println((float)tmp_long/1.e4,4);
delay(mili);
tmp_long = DataFlash.ReadLong();
Serial3.println((float)tmp_long/1.e4,4);
delay(mili);

tmp_long = DataFlash.ReadLong();
Serial3.println((float)tmp_long/1.e8,8);
delay(mili);
tmp_long = DataFlash.ReadLong();
Serial3.println((float)tmp_long/1.e8,8);
delay(mili);
tmp_long = DataFlash.ReadLong();
Serial3.println((float)tmp_long/1.e8,8);
delay(mili);
tmp_long = DataFlash.ReadLong();
Serial3.println((float)tmp_long/1.e8,8);
delay(mili);

tmp_long = DataFlash.ReadLong();
Serial3.println((float)tmp_long/1.e4,4);
delay(mili);
tmp_long = DataFlash.ReadLong();
Serial3.println((float)tmp_long/1.e4,4);
delay(mili);
tmp_long = DataFlash.ReadLong();
Serial3.println((float)tmp_long/1.e4,4);
delay(mili);
tmp_long = DataFlash.ReadLong();
Serial3.println((float)tmp_long/1.e4,4);
delay(mili);

tmp_longlong = DataFlash.ReadLongLong();
Serial3.println((float)tmp_longlong/1.e9,9);
delay(mili);
tmp_longlong = DataFlash.ReadLongLong();
Serial3.println((float)tmp_longlong/1.e8,8);
delay(mili);

tmp_longlong = DataFlash.ReadLongLong();
Serial3.println((float)tmp_longlong/1.e9,9);
delay(mili);
```

```
                    tmp_longlong = DataFlash.ReadLongLong();
                    Serial3.println((float)tmp_longlong/1.e8,8);
                    delay(mili);

                    tmp_long = DataFlash.ReadLong();
                    Serial3.println((float)tmp_long/1.e4,4);
                    delay(mili);
                    tmp_long = DataFlash.ReadLong();
                    Serial3.println((float)tmp_long/1.e4,4);
                    delay(mili);

                    tmp_long = DataFlash.ReadLong();
                    Serial3.println((float)tmp_long/1.e4,4);
                    delay(mili);
                    tmp_long = DataFlash.ReadLong();
                    Serial3.println((float)tmp_long/1.e4,4);
                    delay(mili);

                    tmp_long = DataFlash.ReadLong();
                    Serial3.println((float)tmp_long/1.e4,4);
                    delay(mili);
                    tmp_long = DataFlash.ReadLong();
                    Serial3.println((float)tmp_long/1.e4,4);
                    delay(mili);

                    tmp_long = DataFlash.ReadLong();
                    Serial3.println((float)tmp_long/1.e4,4);
                    delay(mili);
                    tmp_long = DataFlash.ReadLong();
                    Serial3.println((float)tmp_long/1.e4,4);
                    delay(mili);

                    tmp_long = DataFlash.ReadLong();
                    Serial3.println((float)tmp_long/1.e4,4);
                    delay(mili);
                    tmp_long = DataFlash.ReadLong();
                    Serial3.println((float)tmp_long/1.e4,4);
                    delay(mili);

                    tmp_long = DataFlash.ReadLong();
                    Serial3.println((float)tmp_long/1.e4,4);
                    delay(mili);
                    tmp_long = DataFlash.ReadLong();
                    Serial3.println((float)tmp_long/1.e4,4);
                    delay(mili);

                    // Read 1 int...

                    tmp_int = DataFlash.ReadInt();
                    Serial3.println((float)tmp_int,2);
                    delay(mili);

                    // Read 1 long...

                    tmp_long = DataFlash.ReadLong();
                    Serial3.println(tmp_long);
                    delay(mili);

                    // Read the checksum...
                    tmp_byte1 = DataFlash.ReadByte();
                    delay(mili);
                    tmp_byte2 = DataFlash.ReadByte();
                    delay(mili);
                }
                delay(10*mili);
        }

        guardado_DF = true;

        n_paq_DF = 0;

        Serial3.println("Reading completed");
}

void erase_logs()
{

        Serial3.println("Erasing logs...");
```

```
        float porc = 0.;
        int porc_ant = 0;
        int aux;

        Serial3.print("0%____");

        for(int j = 1; j < 4096; j++){
                DataFlash.PageErase(j);
                porc = (float)j/4096.*100.;
                aux = (int)porc/10;
                if((aux - porc_ant)>=1 && (aux%1==0)){
                  Serial3.print(aux*10,DEC);
                  Serial3.print("%____");
                }
                porc_ant = aux;
        }
        Serial3.println("100%");
        borrar_DF = false;
        Serial3.println("Logs_erased");
}
```

Archivo variador.cpp

En este archivo se realizan los cálculos del control de velocidad de los motores.

```
#include "Variador.h"

Variador::Variador(AP_Var::Key key, int I, uint32_t mm, float acc, float vel, float sp_vel, float ff) :
    _group(key, PSTR("VARIADOR_")), numero_vari(I), medida(mm), accion_control(acc), velocity(vel), sp_velo
    {

                reset();

        }

void Variador::reset()
{
                S_motor = 0;
                SC_eq = 0;
                umk = 0;
                umk1 = S_motor-SC_eq;
                ufmk = 0;
                ufmk1 = 0;
                emk = 0;
                emk1 = 0;
                IntAntiWindup = 0;
                Urealns = 0;
                Wmk = 0;
                Wrefmk = 0;

                //out_of_range = false;

                TI  = 35.e-3;                   /* Tiempo integral */
                TM  =  10.e-3;                  /* Tiempo de muestreo */
                KP  =  2.3540e5;        /* Ganancia proporcional*/
                TINTAW = 0.07;                  /* Tiempo integral del anti wind up */
                /* Constante de tiempo de filtro adicional para se al de control */
                TAU_AF = 10.e-3;

                coef0=(KP*(TI+TM))/TI;          /* Constante del controlador lineal PI */
                coef1=KP;                                       /* Constante del controlador lineal PI */

                coef0f=TAU_AF/(TAU_AF+TM);      /* Constante del filtro adicional del controlador lineal */
                coef1f=TM/(TAU_AF+TM);          /* Constante del filtro adicional de controlador lineal */

                V_MIN = 8.4438e-5;
                V_MAX = 4.3668e-4;
                PORC_MIN = 6;
                PORC_MAX = 90;
}

void Variador::calculo_PI()
{

        Wrefmk = sp_velocity;
```

```
        Wmk = velocity;

        if (Wrefmk<V_MIN)
        {
                Wrefmk = V_MIN;
        }
        else if (Wrefmk>V_MAX)
        {
                Wrefmk = V_MAX;
        }
        emk = Wrefmk - Wmk;
                /* Ecuacin del PI */
                umk = umk1 + coef0*emk - coef1*emk1;
        /* Filtro adicional de la seal de control del PI */
                ufmk = coef0f*ufmk1 + coef1f*umk;
        /* Calculo de la seal de control nosaturada */
                Urealns = SC_eq + ufmk;
        /* Anti wind up */
                if ((Urealns-IntAntiWindup)>PORC_MAX){
                        S_motor = PORC_MAX;
                }else if ((Urealns-IntAntiWindup)<PORC_MIN){
                        S_motor = PORC_MIN;
                }else{
                        S_motor = Urealns - IntAntiWindup;
                }
                /* Actualizacin de la integral del anti wind up */
                IntAntiWindup += TM/TINTAW*(Urealns - IntAntiWindup - S_motor);

        umk1 = umk;
        emk1 = emk;
        ufmk1 = ufmk;

}

void Variador::actualiza_coef()
{
                coef0=(KP*(TI+TM))/TI;          /* Constante del controlador lineal PI */
                coef1=KP;                                       /* Constante del controlador lineal PI */

                coef0f=TAU_AF/(TAU_AF+TM);      /* Constante del filtro adicional del controlador lineal */
                coef1f=TM/(TAU_AF+TM);          /* Constante del filtro adicional de controlador lineal */
}

bool
Variador::read()
{
    return true;
}

bool
Variador::write()
{
    return true;
}
```

Archivo ControlAltura.cpp

En este archivo se realizan los cálculos del control de altura.

```
#include "ControlAltura.h"

ControlAltura::ControlAltura(AP_Var::Key key) :
    _group(key, PSTR("CONTROLADOR_ALTURA_"))
    {
                reset();

        }

void ControlAltura::reset(){

        Treal = 0;
        Teq = 0;
        Tk = 0;
        Tk1 = Treal-Teq;
        ek = 0;
```

```
        ek1 = 0;
        ek2 = 0;
        IntAntiWindup = 0;
        Trealns = 0;
        hk = 0;
        hfk = 0;
        hfk1 = 0;
        hrefk = 0;
        hreffilk = 0;
        hreffilk1 = 0;

        //out_of_range = false;

        TI  =  8;                       /* Tiempo integral */
        TD  =  2;                       /* Tiempo derivativo */
        TM  =  10.e-3;      /* Tiempo de muestreo */
        KP  =  1.6;         /* Ganancia proporcional */
        TINTAW = 0.07;          /* Tiempo integral del anti wind up */
        TAU_F_H = 10.e-3;       /* Constante de tiempo de filtro para la medida de la altura */
        TAU_F_REF = 3.;         /* Constante de tiempo de filtro para la referencia de altura */
        coef0=(KP*(TD*TI+TM*TI+TM*TM))/(TI*TM);          /* Constante del controlador lineal PID */
        coef1=(KP*(TM + 2*TD))/TM;                          /* Constante del controlador lineal PID */
        coef2=KP*TD/TM;                                     /* Constante del controlador lineal PID */
        H_MIN = 20;
        H_MAX = 120;
        T_MIN = 50;
        T_MAX = 100;

        h_conmu_escalon = 0;
        h_obj_escalon = 0;
        h_conmu_desp = 0;
        h_obj_desp = 0;
        h_conmu_ater = 0;
        h_obj_ater = 0;

        T_conmu = 0;
        T_obj = 0;

}

void ControlAltura::calculo_PID()
{

        hreffilk = TAU_F_REF/(TAU_F_REF+TM)*hreffilk1 + TM/(TAU_F_REF+TM)*hrefk;
        hfk = TAU_F_H/(TAU_F_H+TM)*hfk1 + TM/(TAU_F_H+TM)*hk;

        if (hreffilk<H_MIN)
        {
                hreffilk = H_MIN;
        }
        else if (hreffilk>H_MAX)
        {
                hreffilk = H_MAX;
        }

        ek = hreffilk - hfk;

                /* Ecuacin del PID */
                Tk = Tk1 + coef0*ek - coef1*ek1 + coef2*ek2;
                /* Filtro adicional de la seal de control del PID */
                /*Tfk = coef0f*Tfk1 + coef1f*Tk;*/
                /*Tfk = Tk;*/
                /* Calculo de la seal de control nosaturada */
                //Trealns = Teq + Tfk;
                Trealns = Teq + Tk;
                /* Anti wind up */
                if ((Trealns-IntAntiWindup)>T_MAX){
                        Treal = T_MAX;
                }else if ((Trealns-IntAntiWindup)<T_MIN){
                        Treal = T_MIN;
                }else{
                        Treal = Trealns - IntAntiWindup;
                }
                /* Actualizacin de la integral del anti wind up */
                IntAntiWindup += TM/TINTAW*(Trealns - IntAntiWindup - Treal);

        /*}
        else{
                out_of_range = true;
```

```
        }*/

        Tk1 = Tk;
        ek2 = ek1;
        ek1 = ek;
        //Tfk1 = Tfk;

        hreffilk1 = hreffilk;
        hfk1 = hfk;

}

void ControlAltura::actualiza_coef()
{
                //coef0=(KP*(TI+TM))/TI;        /* Constante del controlador lineal PI */
                //coef1=KP;                     /* Constante del controlador lineal PI */
                /* Constante del controlador lineal PID */
                coef0=(KP*(TD*TI+TM*TI+TM*TM))/(TI*TM);
                coef1=(KP*(TM + 2*TD))/TM;      /* Constante del controlador lineal PID */
                coef2=KP*TD/TM;         /* Constante del controlador lineal PID */

                /* Constante del filtro adicional del controlador lineal */
                //coef0f=TAU_AF/(TAU_AF+TM);
                /* Constante del filtro adicional de controlador lineal */
                //coef1f=TM/(TAU_AF+TM);

}
```

Archivo ControlAngulo.cpp

En este archivo se realizan los cálculos del control de estabilización.

```
#include "ControlAngulo.h"

ControlAngulo::ControlAngulo(AP_Var::Key key) :
    _group(key, PSTR("CONTROLADOR_ANGULO_"))
    {
                reset();

        }

void ControlAngulo::calculo_PID()
{

        if (angrefk<ANG_MIN)
        {
                angrefk = ANG_MIN;
        }
        else if (angrefk>ANG_MAX)
        {
                angrefk = ANG_MAX;
        }

        ek = angrefk - angk;
        dk = der_angrefk - der_angk;

                /* Ecuacin del PID */
                /*Tk = Tk1 + coef0*ek - coef1*ek1 + coef2*(dfk-dfk1);*/

                Tk = Tk1 + coef0*ek - coef1*ek1 + coef2*(dk-dk1);

                //Tk = coef1*ek + coef2*dk;

                /* Filtro adicional de la seal de control del PID */
                //Tfk = coef0f*Tfk1 + coef1f*Tk;
                /*Tfk = Tk;*/

                /* Calculo de la seal de control nosaturada */
                /*Trealns = Teq + Tfk;*/
                Trealns = Teq + Tk;
                /* Anti wind up */
                Treal = Trealns-IntAntiWindup;
                if (Treal>T_MAX){
                        Treal = T_MAX;
                }else if (Treal<T_MIN){
```

```
                Treal = T_MIN;
            }
            /* Actualizacin de la integral del anti wind up */
            IntAntiWindup += TM/TINTAW*(Trealns - IntAntiWindup - Treal);

        /*else{
            out_of_range = true;
        }*/

        Tk1 = Tk;
        dk1 = dk;
        ek1 = ek;

}

void ControlAngulo::actualiza_coef()
{

            coef0=(KP*(TI+TM))/TI;          /* Constante del controlador lineal PID */
            coef1=KP;                                   /* Constante del controlador lineal PID */
            coef2=KP*TD;                              /* Constante del controlador lineal PID */
            /* Constante del filtro adicional del controlador lineal */
            //coef0f=TAU_AF/(TAU_AF+TM);
            /* Constante del filtro adicional de controlador lineal */
            //coef1f=TM/(TAU_AF+TM);

}

void ControlAngulo::reset()
{

            Treal = 0;
            Teq = 0;
            Tk = 0;
            Tk1 = Treal-Teq;
            ek = 0;
            ek1 = 0;
            dk = 0;
            dk1 = 0;
            IntAntiWindup = 0;
            Trealns = 0;
            angk = 0;
            angrefk = 0;

            //out_of_range = false;

            TI  =  8;                       /* Tiempo integral */
            TD  =  2;                       /* Tiempo derivativo */
            TM  =  10.e-3;      /* Tiempo de muestreo */
            KP  =  2.3540e5;    /* Ganancia proporcional */
            TINTAW = 0.07;          /* Tiempo integral del anti wind up */
             /* Constante de tiempo de filtro adicional para se al de control */
            //TAU_AF = 10.e-3;

            coef0=(KP*(TI+TM))/TI;  /* Constante del controlador lineal PID */
            coef1=KP;                           /* Constante del controlador lineal PID */
            coef2=KP*TD;                      /* Constante del controlador lineal PID */
            /* Constante del filtro adicional del controlador lineal */
            //coef0f=TAU_AF/(TAU_AF+TM);
            /* Constante del filtro adicional de controlador lineal */
            //coef1f=TM/(TAU_AF+TM);

            ANG_MIN = 0;
            ANG_MAX = 0;
            T_MIN = 50;
            T_MAX = 100;

            ang_conmu_reg = 0;
            ang_obj_reg = 0;
}
```

Archivo ControlPosicion.cpp

En este archivo se realizan los cálculos del control de posición por GPS.

```
#include "ControlPosicion.h"

ControlPosicion::ControlPosicion(AP_Var::Key key) :
    _group(key, PSTR("CONTROLADOR_POSICION_2D_"))
    {
            reset();
        }

void ControlPosicion::reset()
{

            Freal = 0;
            Feq = 0;
            Fk = 0;
            Fk1 = 0;
            Ffk = 0;
            Ffk1 = 0;
            ek = 0;
            ek1 = 0;
            ek2 = 0;
            IntAntiWindup = 0;
            Frealns = 0;
            posk = 0;
            posfk = 0;
            posfk1 = 0;
            posrefk = 0;
            posreffilk = 0;
            posreffilk1 = 0;

            TI  =  8;                       /* Tiempo integral */
            TD  =  2;                       /* Tiempo derivativo */
            TM  =  10.e-3;      /* Tiempo de muestreo */
            KP  =  1.6;         /* Ganancia proporcional */
            TINTAW = 0.07;          /* Tiempo integral del anti wind up */
            TAU_AF = 10.e-3;    /* Constante de tiempo de filtro adicional para se al de control */
            TAU_F_POS = 10.e-3;     /* Constante de tiempo de filtro para la medida de la posicin */
            TAU_F_POSREF = 3.;      /* Constante de tiempo de filtro para la referencia de posicin */

            coef0=(KP*(TD*TI+TM*TI+TM*TM))/(TI*TM); /* Constante del controlador lineal PID */
            coef1=(KP*(TM + 2*TD))/TM;                              /* Constante del controlador lineal PID */
            coef2=KP*TD/TM;                                         /* Constante del controlador lineal PID */

            coef0f=TAU_AF/(TAU_AF+TM);      /* Constante del filtro adicional del controlador lineal */
            coef1f=TM/(TAU_AF+TM);          /* Constante del filtro adicional de controlador lineal */

            POS_MIN = -10;
            POS_MAX = 10;
            F_MIN = -3.14;
            F_MAX = 3.14;

}

void ControlPosicion::calculo_PID()
{

        posreffilk = TAU_F_POSREF/(TAU_F_POSREF+TM)*posreffilk1 + TM/(TAU_F_POSREF+TM)*posrefk;
        posfk = TAU_F_POS/(TAU_F_POS+TM)*posfk1 + TM/(TAU_F_POS+TM)*posk;

        if (posreffilk<POS_MIN)
        {
                posreffilk = POS_MIN;
        }
        else if (posreffilk>POS_MAX)
        {
                posreffilk = POS_MAX;
        }

        ek = posreffilk - posfk;

                /* Ecuacin del PID */

                Fk = Fk1 + coef0*ek - coef1*ek1 + coef2*ek2;

                /* Filtro adicional de la se al de control del PID */

                Ffk = coef0f*Ffk1 + coef1f*Fk;
                //Ffk = Fk;
```

```
                /* Calculo de la se al de control nosaturada */

                Frealns = Feq + Ffk;

                /* Anti wind up */
                if ((Frealns-IntAntiWindup)>F_MAX){
                        Freal = F_MAX;
                }else if ((Frealns-IntAntiWindup)<F_MIN){
                        Freal = F_MIN;
                }else{
                        Freal = Frealns - IntAntiWindup;
                }
                /* Actualizacion de la integral del anti wind up */
                IntAntiWindup += TM/TINTAW*(Frealns - IntAntiWindup - Freal);

        Fk1 = Fk;
        ek2 = ek1;
        ek1 = ek;
        Ffk1 = Ffk;

        posreffilk1 = posreffilk;
        posfk1 = posfk;

}

void ControlPosicion::actualiza_coef()
{

                coef0=(KP*(TD*TI+TM*TI+TM*TM))/(TI*TM); /* Constante del controlador lineal PID */
                coef1=(KP*(TM + 2*TD))/TM;          /* Constante del controlador lineal PID */
                coef2=KP*TD/TM;                  /* Constante del controlador lineal PID */

                coef0f=TAU_AF/(TAU_AF+TM);       /* Constante del filtro adicional del controlador lineal */
                coef1f=TM/(TAU_AF+TM);   /* Constante del filtro adicional de controlador lineal */

}
```

Archivo ControlVelocidad.cpp

En este archivo se realizan los cálculos del control de velocidad lineal.

```
#include "ControlVelocidad.h"

ControlVelocidad::ControlVelocidad(AP_Var::Key key) :
    _group(key, PSTR("CONTROLADOR_VELOCIDAD_LINEAL_"))
    {
                reset();

        }

void ControlVelocidad::calculo_PD()
{

        if (vrefk<V_MIN)
        {
                vrefk = V_MIN;
        }
        else if (vrefk>V_MAX)
        {
                vrefk = V_MAX;
        }

        ek = vrefk - vk;
        dk = der_vrefk - der_vk;

        /*if (vrefk<V_MIN)
        {
                vrefk = V_MIN;
        }
        else if (vrefk>V_MAX)
        {
                vrefk = V_MAX;
        }*/

                /* Ecuacin del PD */
```

```
                //Fk = coef0*ek + coef1*dk;

                /*Fk = Fk1 + coef0*ek - coef1*ek1 + coef2*(dk-dk1);*/
                Fk = coef1*ek + coef2*dk;

                /* Filtro adicional de la se al de control del PID */
                Ffk = coef0f*Ffk1 + coef1f*Fk;
                //Ffk = Fk;

                /* Calculo de la se al de control nosaturada */
                /*angrealns = angeq + angfk;*/
                Frealns = Feq + Ffk;

                if (Frealns>F_MAX){
                        Freal = F_MAX;
                }else if (Frealns<F_MIN){
                        Freal = F_MIN;
                }else{
                        Freal = Frealns;
                }

        Fk1 = Fk;
        dk1 = dk;
        ek1 = ek;
        Ffk1 = Ffk;

}

void ControlVelocidad::actualiza_coef()
{

                //coef0 = KP;           /* Constante del controlador lineal PID */
                //coef1 = KP*TD;        /* Constante del controlador lineal PID */

                coef0=(KP*(TI+TM))/TI;          /* Constante del controlador lineal PID */
                coef1=KP;                               /* Constante del controlador lineal PID */
                coef2=KP*TD;                            /* Constante del controlador lineal PID */

                coef0f=TAU_AF/(TAU_AF+TM);      /* Constante del filtro adicional del controlador lineal */
                coef1f=TM/(TAU_AF+TM);          /* Constante del filtro adicional de controlador lineal */

}

void ControlVelocidad::reset()
{

                Freal = 0;
                Feq = 0;
                Fk = 0;
                Fk1 = 0;
                Ffk = 0;
                Ffk1 = 0;
                ek = 0;
                ek1 = 0;
                dk = 0;
                dk1 = 0;
                IntAntiWindup = 0;
                Frealns = 0;
                vk = 0;
                der_vk = 0;
                vrefk = 0;
                der_vrefk = 0;
                TI  =  8;                       /* Tiempo integral */
                TD  =  2;                       /* Tiempo derivativo */
                TM  =  10.e-3;      /* Tiempo de muestreo */
                KP  =  1.6;         /* Ganancia proporcional */
                TINTAW = 0.07;          /* Tiempo integral del anti wind up */
                  /* Constante de tiempo de filtro adicional para se al de control */
                TAU_AF = 10.e-3;

                //coef0 = KP;                           /* Constante del controlador lineal PID */
                //coef1 = KP*TD;                      /* Constante del controlador lineal PID */

                coef0=(KP*(TI+TM))/TI;          /* Constante del controlador lineal PID */
                coef1=KP;                               /* Constante del controlador lineal PID */
                coef2=KP*TD;                            /* Constante del controlador lineal PID */

                coef0f=TAU_AF/(TAU_AF+TM); //Cte del filtro adicional del controlador lineal
```

```
        coef1f=TM/(TAU_AF+TM);  //Cte del filtro adicional de controlador lineal

        V_MIN = -10;
        V_MAX = 10;
        F_MIN = -3.14;
        F_MAX = 3.14;

}
```

Apéndice B

Código ultrasonidos

A continuacion se encuentra el codigo programado en los microcontroladores de las placas Waspmote. Estas placas son las encargadas de proporcionar las medidas de distancia entre los emisores y receptores de ultrasonidos. El proceso para obtener dicha distancia se comento en el apartado 4.3.

```
// Declaracion de variables

long tiempo1 = 0;
long tiempo2 = 0;
int intnum = 0;

// Interrupcion por overflow del timer1

ISR(TIMER1_OVF_vect)
{
  TCNT1 = 0;
}

// Declaracion de la funcion de configuracion

void setup()
{
  // Pin salida 3.3v ON

  PWR.setSensorPower(SENS_3V3,SENS_ON);

  // Pin DIGITAL0 de entrada

  pinMode(DIGITAL0, INPUT);

  // Configuracion timer 1

  TIMSK1 = 0x01;
  TCCR1A = 0x00;
  TCNT1 = 0x0000;
  TCCR1B = 0xC2;

  // Configuracion timer 3

  TIMSK3 = 0x01;
  TIMSK3 |= (1 << OCIE3A);
  TCCR3A = 0x00;
  TCNT3 = 0x0000;
  TCCR3B = 0x02;
  OCR3A = 25000;

  // Interrupcion asociada a pin DIGITAL0 (4)

  attachInterrupt(4, interrupcion, HIGH);

  // Inicio de la comunicacion USB

  USB.begin();
```

```
  // Activacion de las interrupciones

  sei();

  // Esperar 5 segundos

  delay(5000);

  // Reseteo de la variable tiempo1

  tiempo1 = 0;

  // Reseteo del timer 1

  TCNT1 = 0;
}

// Declaracion de la funcion principal

void loop ()
{
  if (intnum == 2)
  {
    delay(14);
    intnum = 5;
  }
}

// Declaracion de la funcion de interrupcion

void interrupcion()
{
  if (intnum <=2)
  {
    intnum++;
  }
 if (TCNT1 > 15000)
 {
   tiempo1 = TCNT1;
   TCNT1 = 0;
 }
}

// Interrupcion por comparacion del timer 3

ISR(TIMER3_COMPA_vect)
{
  USB.println(tiempo1);
  TCNT3 = 0;
}
```