Universidad Politécnica de Cartagena

industriales
etsii UPCT

Diseño, construcción y control de un hexacóptero en plataforma de prueba.

Titulación: Ingeniería industrial

Alumno/a: Alberto López Gassó

Director/a/s: Julio José Ibarrrola Lacalle

Cartagena, 9 de Noviembre de 2012

Agradecimientos

En primer lugar quisiera agradecer a Don Julio José Ibarrola Lacalle la oportunidad que me ha brindado para realizar este proyecto, por todas las atenciones, por el tiempo que ha perdido conmigo, y sobre todo por su apoyo y comprensión desde la distancia.

En segundo lugar al profesor Don Francisco Ortiz, que me ha ayudado a llegar hasta aquí, ofreciéndome su apoyo incondicional desde el principio, hace ya varios años. Sin su ayuda no sé si hubiera despertado en mí el interés por la electrónica y robótica.

Al Departamento de Ingeniería de Sistemas, al Departamento de Tecnología Electrónica, a sus profesores y a Don Pablo Martínez por su apoyo durante la realización de este proyecto.

Al personal del grupo de investigación DSIE por permitirme compartir su laboratorio, tanto este curso como el anterior. A Pencho Soto Vallés por ayudarme en el prototipado de PCBs y a Don José Juan Rodrígez Martínez y a Don Andrés Carrillo Casanova por su valiosa ayuda.

Al profesor Mariusz R. Rzasa de la Universidad Politécnica de Opolska, Polonia, por dejarme usar su laboratorio y herramientas durante mi estancia en dicha universidad.

A mi familia y en especial a mi padre que sin su ayuda no habría sido posible llegar hasta aquí, que me ha ayudado en la parte técnica de las tareas de construcción de los prototipos, balancines, adaptadores, ruedas… y por el incondicional apoyo económico para la construcción del hexacóptero.

Tabla de contenido

Índice de ilustraciones

1 Introducción y objetivos

1.1 Visión general: vehículos aéreos no tripulados, UAV.

Un vehículo aéreo no tripulado es una aeronave que vuela sin tripulación a bordo. En los últimos años ha habido un crecimiento exponencial de las aplicaciones civiles, aunque se han usado principalmente con motivos militares. Se conocen como "UAV" (Unmanned Aerial Vehicle) o como *drones.* Este tipo de vehículos pueden ser totalmente autónomos, por control remoto "RPA" (Remote Piloted Aircraft) o sistemas mixtos. Hay una gran variedad de formas, tamaños, configuraciones y características. Para distinguir estos vehículos de los misiles, los *UAV* pueden ser reutilizados y recuperables. Estos vehículos son muy utilizados por los aeromodelistas radiocontrol.

Un multicóptero es básicamente un helicóptero pero con varios ejes y motores independientes. Existen tricópteros, cuadricópteros, hexacópteros, octocópteros... con tres, cuatro, seis y ocho motores respectivamente. Tienen buena estabilidad y capacidad de carga. Las hélices giran en sentidos contrapuestos para evitar que el multicóptero gire sobre su propio eje de forma continuada debido a la inercia de las hélices. Una unidad de control varía la velocidad de giro de cada motor independientemente, así se consiguen todos los movimientos posibles.

Los Estados Unidos e Israel fueron los pioneros en el desarrollo de esta tecnología. La cuota de mercado de los productores Americanos ronda el 60% y se prevé un crecimiento de un 5 al 10% en 2016. "*Northrop Grumman"* and "*General Atomics"* son las empresas dominantes en la industria. En 2006 la cuota de mercado europea era de sólo un 4%.

Actualmente el mercado internacional está dominado por los israelíes y norteamericanos, que llevan una gran ventaja, pues llevan investigando en este sector más de 20 años, según afirmó el director científico de FADA-CATEC Aníbal Ollero. El éxito dentro de las fronteras españolas lo tienen dos pequeñas empresas, UAV Navigation en Madrid y Aerovisión, en Guipúzcoa, que han logrado vender sus productos en el mercado internacional. La empresa vasca, que cuenta sólo con diez empleados, ha vendido un sistema completo compuesto por tres unidades en servicio y el centro de control en tierra al Gobierno de Malasia para controlar la inmigración ilegal.

1.2 Posibles usos civiles

De entre los numerosos usos civiles actuales destacamos:

- **Agrícola:** utilizado por las compañías de semillas y agricultores. Las fotografías aéreas se utilizan para registrar el crecimiento, la calidad y el rendimiento de un cultivo. A menudo se utilizan como imágenes de progreso y pueden ser de calidad suficiente como para poder ampliar y contar los tallos individualmente.
- **Construcción:** las fotografías aéreas permiten registrar el progreso, desde la topografía, la construcción, terminación y entrega. Estas imágenes se pueden usar para informes anuales, portafolios o publicidad.

- **Inspecciones:** usados para el examen de líneas de alta tensión o torres de refrigeración pudiéndose obtener primeros planos de los aisladores y cables así como conocer su estado. Este tipo de imágenes resulta caro y peligroso de obtener mediante métodos tradicionales.
- **Carreteras y trabajos de movimientos de tierras:** muy útiles para informes de progresos con buena calidad.
- **Marketing:** las fotografías aéreas son perfectas para publicitarse, mostrar instalaciones, edificios, campos de golf, parque de atracciones, camping, etc… útiles para causar buena impresión a los clientes.
- **Operaciones mineras:** usadas para hacer presentaciones, portafolios, enseñar el equipo y excavaciones. Muy utilizadas para monitorizar la rehabilitación posterior del terreno.
- **Periodismo:** en los últimos años ha ganado importancia esta modalidad buscando mejores resultados en los reportajes y fotografías pues muestran otro punto de vista.
- **Eventos deportivos:** seguimiento y retransmisión de todo tipo de eventos, retransmisiones en directo, grabación de videoclips de deportes extremos.
- **Policía, bomberos y equipos de rescate:** pueden proporcionar una mejor visión del caso a tratar, ya sea servicios antincendios, de búsqueda o control de fronteras.

1.3 Motivación

Mis primeros contactos con la electrónica se iniciaron durante el curso 2008/2009. Cuando intenté construir una lámpara RGB LED analógica y surgieron las primeras dudas, acudí al profesor Don Francisco Ortiz que me proporcionó una valiosa ayuda con la electrónica, y, aunque nunca pude acabar ese proyecto fue el inicio de mi interés por el mundo de la electrónica. El curso siguiente, durante mi estancia en la Universidad de Sevilla, tuve la oportunidad de formar parte del equipo EsiBot de robótica de la Escuela. Con ellos aprendí muchísimo y algo de experiencia, participando en la liga nacional de robótica durante el curso 2009/2010.

De vuelta a la Universidad Politécnica de Cartagena, en el curso 2010/2011, le planteé a Don Francisco Ortiz mi deseo de participar en la competición nacional de robótica. Con su ayuda, construimos un robot de Sumo para participar en la prueba de Cosmobot en Madrid, donde conseguimos homologar, aunque no pudimos pasar de la fase de grupos.

Ya en 2011 Don Francisco Ortiz me puso en contacto con Don Julio Ibarrola, a quién le expuse mi propósito e ideas, que parecían un poco indefinidas y confusas, pero que fueron tomando forma con el tiempo. A partir de ese momento iniciamos un proceso que ha acabado en éste Proyecto Fin de Carrera y en la realización física de éste prototipo de hexacóptero.

En el proyecto que ha durado algo más de un año, se han realizado más de 340 ensayos datados, 350 páginas de borrador de laboratorio, se han construido 3 prototipos de balancines y 2 completos, se han usado 10 variadores de los motores y quemado 4, rotas 8 hélices, se han empleado muchos conectores de diferentes tipos y se han realizado muchas soldaduras.

El proyecto que empezó en Cartagena siguió en Opole (Polonia), durante mi estancia Erasmus, con la construcción del balancín "uno" y el prototipo, aunque la mayoría de los ensayos se realizaron en Cartagena durante Julio y Agosto de 2012 y la redacción ha tenido lugar en Mannheim (Alemania), donde actualmente estudio Automation Technology en la Universidad Politécnica de dicha ciudad.

1.4 Objeto del proyecto

Dimensionamiento, construcción y control de un prototipo restringido en una plataforma de pruebas.

1.5 Organización de los contenidos

Capítulo primero: se realiza una introducción a las tareas desarrolladas en el proyecto fin de carrera.

Capítulo segundo: se estudia la estructura de los multirotores en general, y en concreto, del hardware seleccionado y todos sus componentes. También se incluye una breve teoría de cómo se mueven en el aire y la legislación vigente en España.

Capítulo tercero: se expone que microcontroladora se ha utilizado para el control así como la placa de sensores IMU utilizada.

Capítulo cuarto: se explica brevemente el software utilizado para compilar y tratar el código.

Capítulo quinto: se presentan las primeras aproximaciones restringidas a un eje que se han realizado, los balancines llamados *"número uno" y "número dos".*

Capítulo sexto: se muestra cómo se utilizó una plataforma móvil que deja libre movimiento en los tres ejes y se han realizado ensayos en condiciones parecidas al vuelo real.

Capítulo séptimo: se exponen las conclusiones y trabajos futuros.

Capítulo octavo: bibliografía utilizada.

Apéndices: se exponen las pruebas en vuelo real que se realizaron, se explican las características de algunos elementos usados, también el código utilizado para cada tarea, así cómo el código completo.

2 Selección, dimensionamiento de los elementos del prototipo.

2.1 Tipos de multirotores y justificación de la elección.

Según la aplicación que se quiera adquirir con el multicóptero se elegirán el número de rotores. En nuestro caso la finalidad es la fotografía/vídeo aérea con cierta calidad. Las propiedades que queremos conseguir son:

- Simplicidad y bajo peso en la estructura.
- Capacidad de transportar una cámara de fotos.
- Fácilmente transportable y manejable.
- Autonomía de unos 10 minutos.

Configuraciones básicas:

- **Tricóptero:** barato y fácil de construir, menos estable, partes móviles en la estructura (cola móvil por un servo), bajo empuje y menos tiempo de vuelo (porque los motores tienen que girar más rápido para mantenerlo en el aire).Ver estructura en la ilustración 1, se puede observar la cola móvil por servo en la ilustración 2.

Ilustración 1: tricóptero

Ilustración 2: detalle cola móvil por servo

- **Cuadricóptero**: más simple mecánicamente que el tricóptero. Posee 1/3 más de empuje pesando casi lo mismo y suelen ser más estables ya que no tienen partes móviles. Tienen más tiempo de vuelo debido a que pueden llevar baterías más

grandes y a que los motores trabajan a menos revoluciones. Todavía no contamos con redundancia: Si un motor falla... cae. Esquema del chasis en la imagen 3.

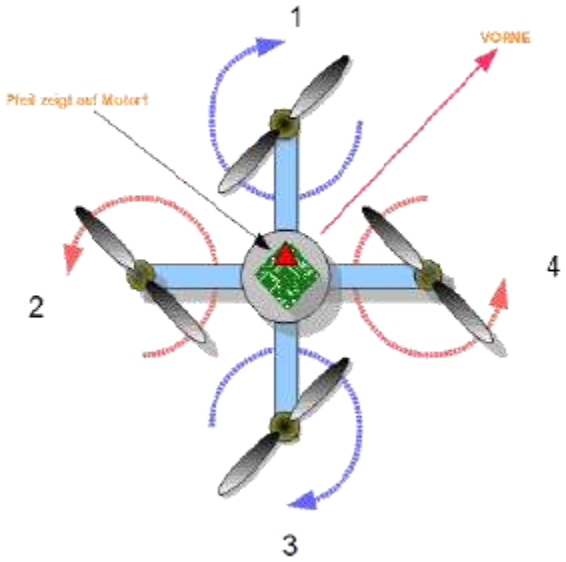


Ilustración 3: cuadricóptero

- **Hexacóptero**: todo lo bueno que tienen los cuadricópteros pero con más potencia y más capacidad de carga. Posee algo de redundancia: Si pierde un motor todavía puede aterrizar, (perdemos el control del ángulo 'yaw'). Tienen el inconveniente de que son más caros y más grandes. Véase la ilustración 4.

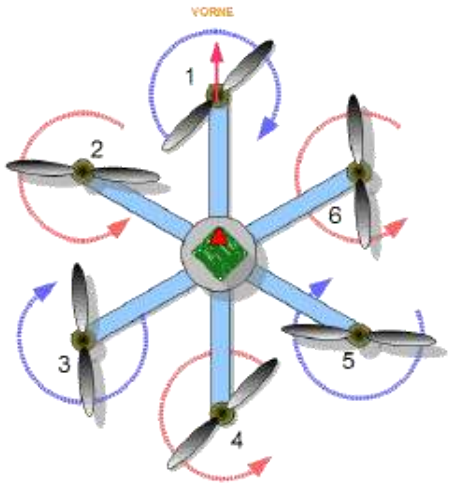


Ilustración 4: hexacóptero

- **Octocóptero**: tienen todo lo bueno de los hexacópteros y además mayor redundancia. Si pierde un motor sigue volando bien. Son muy útiles para asegurar equipos fotográficos de alto precio. Son más caros que los hexacópteros y requieren mucha energía para volar. Véase en la ilustración 5.

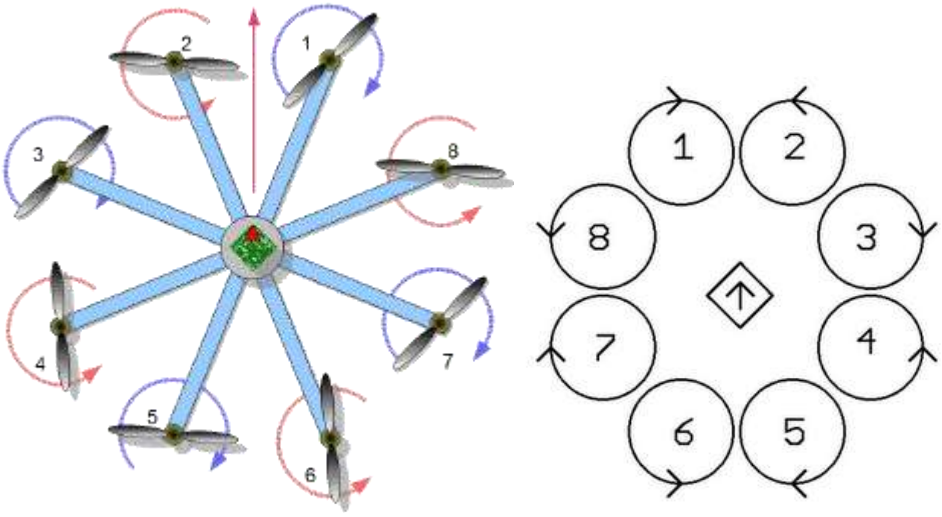


Ilustración 5: octocópteros

Configuraciones compuestas:

- **Y6:** se denomina Y6 a un tricóptero coaxial, es decir 2 motores en cada brazo montados en su mismo eje girando de forma contrapuesta. Funciona igual que un tricóptero, con cola móvil. Las diferencias con un tricóptero son dos: mayor capacidad de carga y redundancia. Véase en la ilustración 6.

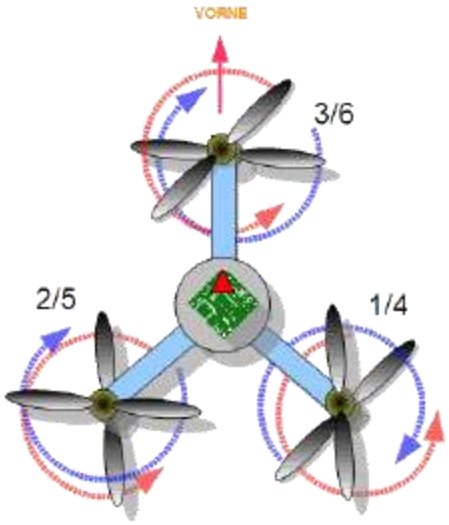


Ilustración 6: hexacóptero Y6

- **Octo-coaxial:** también denominado X8, es un cuadricóptero coaxial con ocho motores. Se consigue una buena redundancia y mayor capacidad de carga que con un cuadricóptero, pero menor que con un octocóptero radial debido al descenso de rendimiento en las hélices.
- **V8:** la principal ventaja es la inmejorable panorámica para la fotografía aérea y su redundancia. Su distribución se puede ver en la ilustración 7. Está totalmente diseñado para la fotografía aérea y es un modelo patentado por una empresa alemana, "Ascending Technologies" la cual comercializa este octocóptero, véase en la ilustración 8, y su precio oscila sobre unos 20 000 euros.

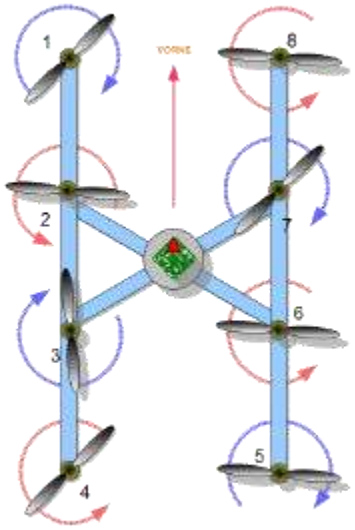


Ilustración 7: octocóptero v8.

Ilustración 8: octocóptero de V8 "Ascending Technologies"

- **Monocóptero:** caso bastante curioso, variante no válida para el uso que se le va a dar en este proyecto. El monocóptero gira sobre su propio eje, ver en la ilustración 9, y cuenta con un único motor en dirección tangente a la trayectoria. Para despegar necesitan de un soporte durante los primero giros hasta que se estabiliza. Su vuelo se asemeja a las semillas voladoras que producen algunos árboles.

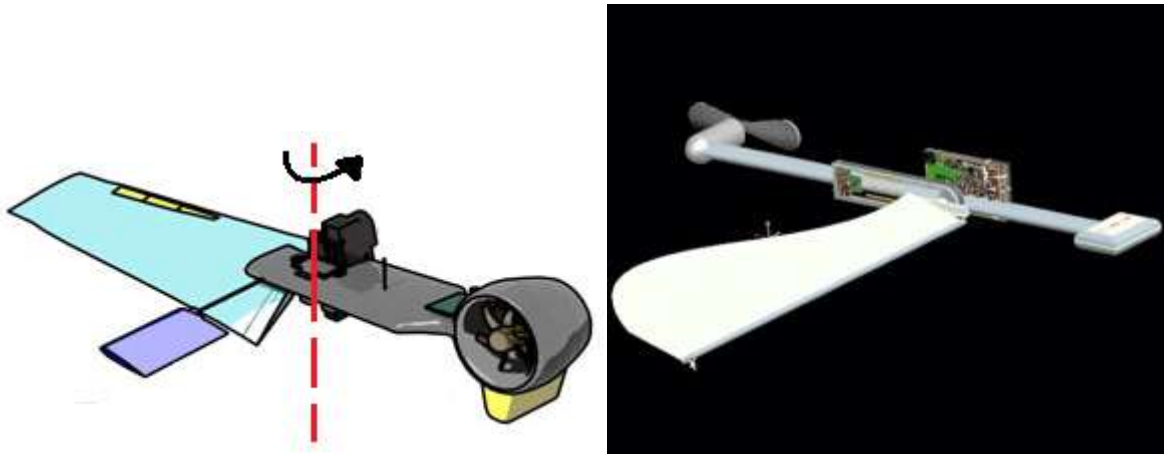


Ilustración 9: monocópteros

Configuraciones en $'\times'$ y en $'+'$:

La configuración en $'\times'$ o en $'+'$ se llaman así por la orientación de los motores delanteros con respecto al tren de aterrizaje y la placa de control. La diferencia entre las distintas configuraciones se puede observar en la ilustración 10.

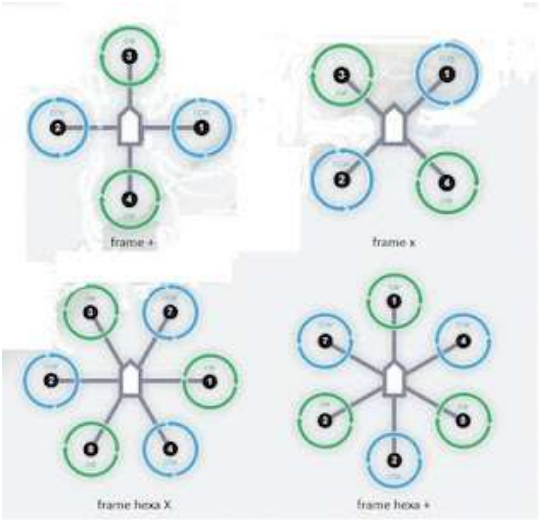


Ilustración 10: orientaciones multicópteros.

Conclusiones:

Descartamos la opción de cuadricóptero para conseguir más capacidad de carga y algo de redundancia. Sobre la opción de elegir un hexacóptero u octocóptero:

Ventajas de un mayor número de Motores:

- Mayor capacidad de carga.
- No es necesario unos motores tan potentes o unas hélices tan grandes para la misma capacidad de carga.
- Redundancia. Muy importante, en caso de rotura de una hélice, choque...

La configuración de seis motores es poco fiable, aunque, probablemente sobreviviría a una parada de un motor, habría que poner ocho motores para tener ambas cualidades: capacidad de carga y redundancia.

A mayor número de motores necesitaremos unas hélices más pequeñas, dan una respuesta más rápida, resultando más estable. Por eso para conseguir estabilidad los motores/hélices han de poder cambiar de velocidad lo más rápidamente posible.

Aunque con seis motores sea más eficiente en un 5 o 10% entre capacidad de carga y peso, elegiremos ocho sólo en el caso de necesitar incrementar la carga o la seguridad, ya que el precio se incrementa en un 30% por un mayor número de motores, ESCs y hélices.

Las configuraciones coaxiales consisten en colocar dos motores contrapuestos en un mismo eje. Tienen como principal ventaja la redundancia. Podemos usar las configuraciones Y6 o X8 equivalentes a un tricóptero y cuadricóptero respectivamente. Aunque son menos eficientes debido a que el aire que llega a la segunda hélice está ya turbulento. Se puede ver en la ilustración 11 un detalle de los motores contrapuestos.

Ilustración 11: detalle motores coaxiales.

Para este proyecto, con objeto de dejar espacio en la frontal para obtener mejores imágenes, la configuración en X es la mejor. Para el prototipo se ha usado la configuración + por inexistencia de la configuración X en el catálogo del distribuidor y por cuestiones económicas. En la ilustración 12 se puede ver la configuración hexa X, que sería la idónea, pero el chasis elegido sólo esta disponible con la configuración +.

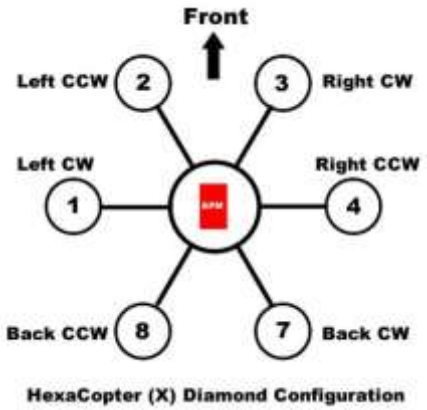


Ilustración 12: hexacóptero configuración X.

2.2 Descripción de la estructura utilizada.

Para la estructura se ha optado por usar una versión comercial debido al alto coste de fabricación y a la dificultad de encontrar algunos componentes como gomas para las juntas, varillas roscadas, etc…

Ilustración 13: estructura utilizada en el prototipo.

Se ha elegido la versión "*SexiCopter*", véase en la ilustración 13, fabricada en fibra de vidrio G10/G11 de 1.5 mm de grosor con tubos de aluminio de 10x10x1mm pintados en rojo y negro, para distinguir la parte frontal. La distancia de motor a motor (*M2M*, Motor to Motor) es de 690mm con un peso de 550 gramos. Cada brazo soporta hasta un kilo de empuje y unas hélices de hasta 12 pulgadas para no chocar entre ellas. El tren de aterrizaje, ver en la figura 14, tiene una altura de 155mm y un espacio porta baterías de 110x90mm.

Ilustración 14: detalle del portabaterías.

El tren de aterrizaje cuenta además con un soporte para cámara. Este tiene la capacidad de ser móvil a través de 2 servos para controlar los dos ángulos 'tilt' y 'pan' de la cámara como se muestra en la ilustración 15.

Ilustración 15: detalle estabilizador para cámara.

2.3 Sistema de propulsión

2.3.1 Calculo del empuje necesario.

Antes de dimensionar los motores y hélices se ha planteado en la tabla 1, cual va a ser el empuje aproximado necesario.

Tabla 1

elemento	cantidad	peso unitario (gramos)	peso (gramos)
Estructura	1	550	550
motores (aproximación)	6	150	900
baterías (aproximación)	2	460	920
ESC	6	40	240
ArduPilotMega	1	45	45
Hélices	6	10	60
cableado	1	50	50
receptor	1	15	15
Carga	1	400	400
TOTAL			3170 gramos

Se estima una carga de entre 150 y 500 gramos. La carga normalmente va a ser una cámara fotográfica con un soporte estabilizador. El empuje mínimo son 3170 gramos.

2.3.2 Motores y hélices.

Los motores que se han usado has sido sin escobillas, normalmente conocidos como "brushless". Estos motores están siendo cada vez más ventajosos, son más baratos de fabricar, pesan mucho menos y requieren menos mantenimiento. La desventaja es que el control es mucho más complejo, pero esto se ha resuelto debido a los controladores de velocidad que transforman la corriente continua en alterna a diferentes frecuencias. Para invertir su giro no se puede hacer por software, y para ello hay que intercambiar dos conductores.

Respecto a las hélices, se han usado de material plástico, aunque también las hay de madera y de fibra de carbono. Las de madera se suelen utilizar para casos que requieran más potencia por motor, las hélices de fibra de carbono son mejores que las de plástico pero notablemente más costosas.

Para el dimensionamiento tomamos como un conjunto Motor + Hélice, ya que cada motor se comporta de manera diferente con diferentes tipos de hélices. Cada motor tiene un tamaño o tipo de hélice ideal u óptima. Si se colocase una hélice muy pequeña el motor se revolucionará en exceso causando efectos negativos y por el contrario, si se coloca una hélice muy grande entonces al motor le faltará fuerza.

Para ello se ha usado el programa "Drive calculator 3.4" que es una herramienta para el análisis de motores Brushless para vehículos aéreos radiocontrol. Este programa contiene una base de datos actualizada con los motores, hélices, ESCs, baterías, estatores y engranajes de transmisión. Así podemos simular el comportamiento de los diferentes elementos.

Para el proceso de elección primero se ha elegido un catálogo de un distribuidor, en este caso *Hobbyking,* un distribuidor de artículos de aeromodelismo y radio control bastante importante y conocido a nivel mundial con almacenes en Hong-Kong, China, Australia, Alemania y Estados Unidos.

De entre los motores brushless disponibles hemos comparado siete de la marca Turnigy, (tabla 2) de menos de 25$ la unidad. Turnigy es una marca asociada a Hobbyking, así conseguimos una alta relación calidad/precio. Estos motores son:

Tabla 2

Modelo	Kv (rpm/V)	peso (g)	precio
Tower Pro 3015	1000	133	$ 17.04
Turnigy 2217 16turn	1105	71	$ 15.38
Turnigy 2217 20turn	860	71	$ 14.04
Turnigy AerodriveXP SK 3548	900	171	$ 24.85
Turnigy AerodriveXP SK 3530	1074	79	$ 17.86
Turnigy 3632	1134	99	$ 19.71
NTM prop drive Series 35-36A	851	112	$ 18.83

Nota: “NTM prop drive Series 35-36A” es la nueva versión de “Turnigy SK 35-36A 910Kv”

De entre las hélices se han comparado sólo dos: **APC 10x5 y APC 10x6**. Los modelos de hélices son quizá las más económicas del mercado actual. Todas las hélices de bajo coste necesitan ser balanceadas, es decir, no son exactamente simétricas en masa y forma. Por lo que precisan de un proceso de calibración previo. Este proceso consiste en colocar la hélice en un eje e ir limando hasta que se quede en equilibrio.

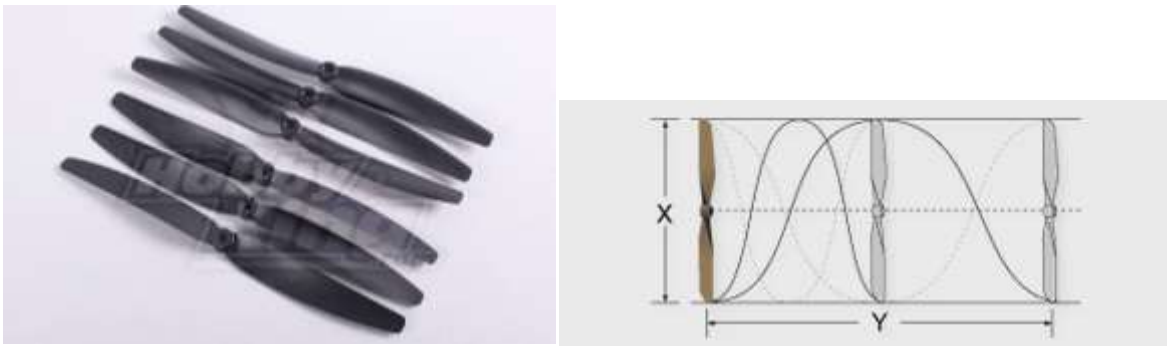


Ilustración 16: hélices y parámetros.

En la nomenclatura de las hélices $X \times Y$ como se puede observar en la ilustración 16, X corresponde a la longitud en pulgadas, e Y a la curvatura, denominada como paso, que es la distancia que recorrería en una vuelta completa. Cuanto mayor sea el paso, debido a que interceptan más líneas de flujo de aire, y en consecuencia mayor cantidad de aire se pondrá en movimiento, mayor fuerza se requerirá por el motor.

2.3.3 Herramienta de cálculo de motores: uso de Drive Calculator

- Con la ayuda de este programa podemos conocer como se comportaría el conjunto de elementos batería + ESC + motor + caja de cambios + hélice. El conexionado de este conjunto se corresponde con el de la ilustración 17.

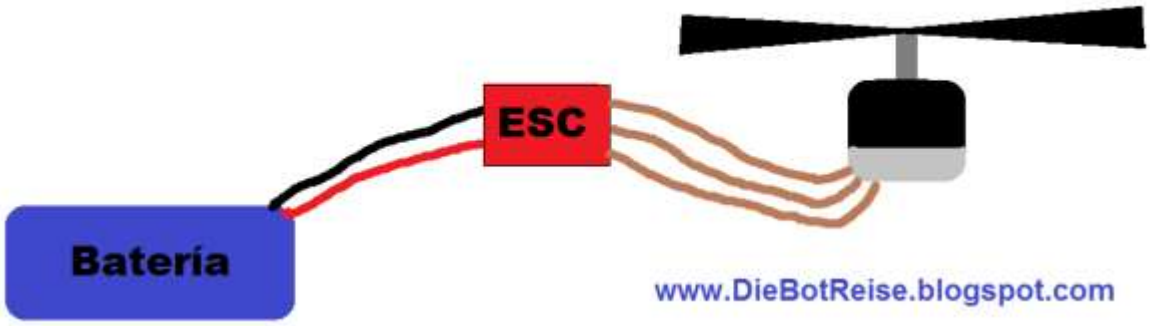


Ilustración 17: motor + Hélice + ESC

- **Baterías:** los parámetros a definir es el número de celdas de la batería que se ha fijado a 3 celdas, es decir 3x3.7 = 11.1Voltios. Al elegir una determinada batería indica el peso correspondiente.
- **ESC:** para el variador de velocidad de motores brushless, el único parámetro que nos interesa es la máxima corriente que permite manejar. Para determinar qué ESC necesitamos basta con elegir el inmediatamente superior a la máxima corriente de consumo de los motores. Al elegir un determinado variador el programa nos indica el peso correspondiente.
- **Motores:** los motores van a ser el parámetro a variar, para así obtener el empuje y rendimiento deseado.
- **Caja de cambios:** no es necesario usar reductora.
- **Hélices:** elegiremos dos hélices diferentes para comparar.

El proceso de diseño consiste en fijar todos los elementos menos los motores. Acto seguido pulsar en Motoren suchen (buscar motores) y saldrá una lista de los motores adecuados a esa configuración y de entre esta lista seleccionaremos los que aparecen en el catálogo del distribuidor.

Realizando una búsqueda con los siguientes parámetros se obtienen 215 motores:

Parámetros:

- Batería: voltaje constante 10.8 Voltios.
- ESC: no registrado.
- Hélice: 10x6 GWS HD.

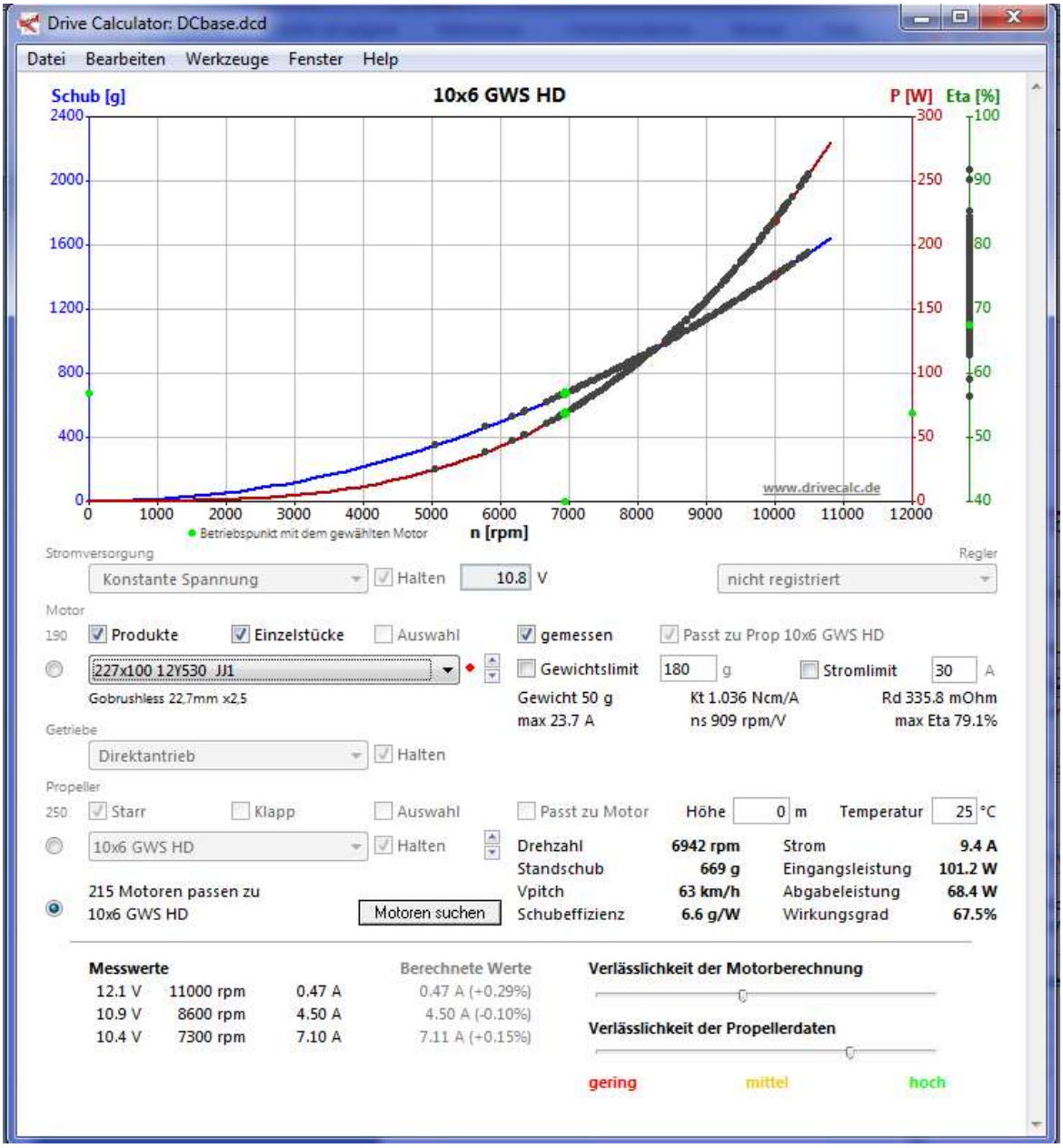


Ilustración 18: drive calculator 1.

De esta primera gráfica se obtiene numerosa información:

La gráfica muestra en puntos grises los motores que se adaptan a nuestros parámetros iniciales, marcando con un punto verde el motor seleccionado en ese momento. Los datos que se obtienen son el empuje (Schub) medido en gramos, Potencia (P) medida en Watios, rendimiento (Eta %) y velocidad (n) en revoluciones por minuto.

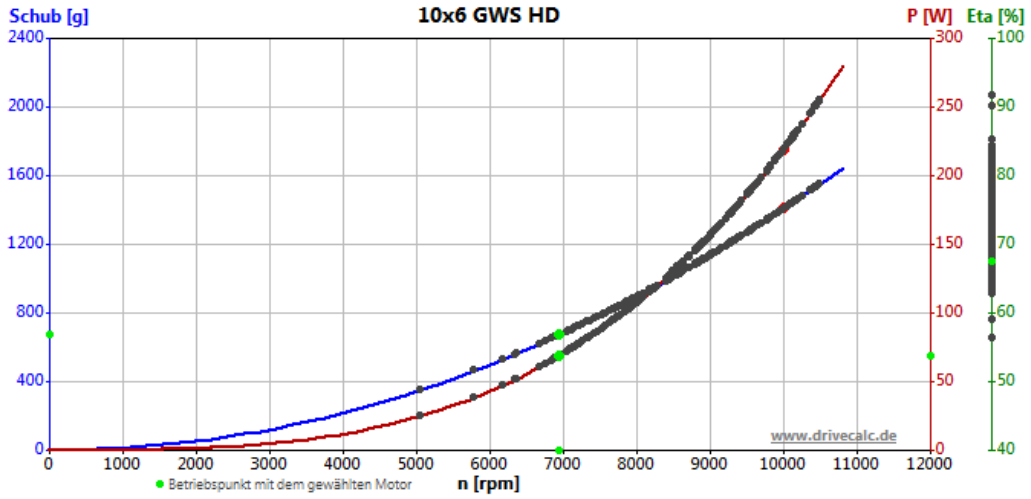


Ilustración 19: drive calculator 2.

Criterios de selección:

- Elegir sólo modelos del catálogo de nuestro distribuidor.
- El empuje de cada motor debe ser alrededor de 1000 gramos, pero no mucho más, ya que la estructura sólo soporta alrededor de 1kg por brazo.
- Limitamos las revoluciones de trabajo a 1000Kv (rpm/V) para evitar vibraciones excesivas.

Eligiendo los motores se elabora la tabla 3 y 4, que es una comparativa con cada hélice a testear:

Hélice APC 10x6:

Tabla 3

Modelo	Kv (rpm/V)	peso (g)	empuje (g)	corriente (A)	eficiencia (%)
Tower Pro 3015	1000	133	1324	23.6	70
Turnigy 2217 16turn	1105	71			
Turnigy 2217 20turn	860	71	851	12.8	76.7
Turnigy AerodriveXP SK 3548	900	171	997	22.3	71.5
Turnigy AerodriveXP SK 3530	1074	79	1064	18.4	69.7
Turnigy 3632	1134	99	1385	26.1	74.3
NTM prop drive Series 35-36A	851	112	921	13.4	76.5

Hélice APC 10x5:

Tabla 4

Modelo	Kv (rpm/V)	peso (g)	empuje (g)	corriente (A)	eficiencia (%)
Tower Pro 3015	1000	133	1324	23.6	70
Turnigy 2217 16turn	1105	71	1148	20.5	77.2
Turnigy 2217 20turn	860	71	851	12.8	76.7
Turnigy AerodriveXP SK 3548	900	171			
Turnigy AerodriveXP SK 3530	1074	79			
Turnigy 3632	1134	99			
NTM prop drive Series 35-36A	**851**	**112**			

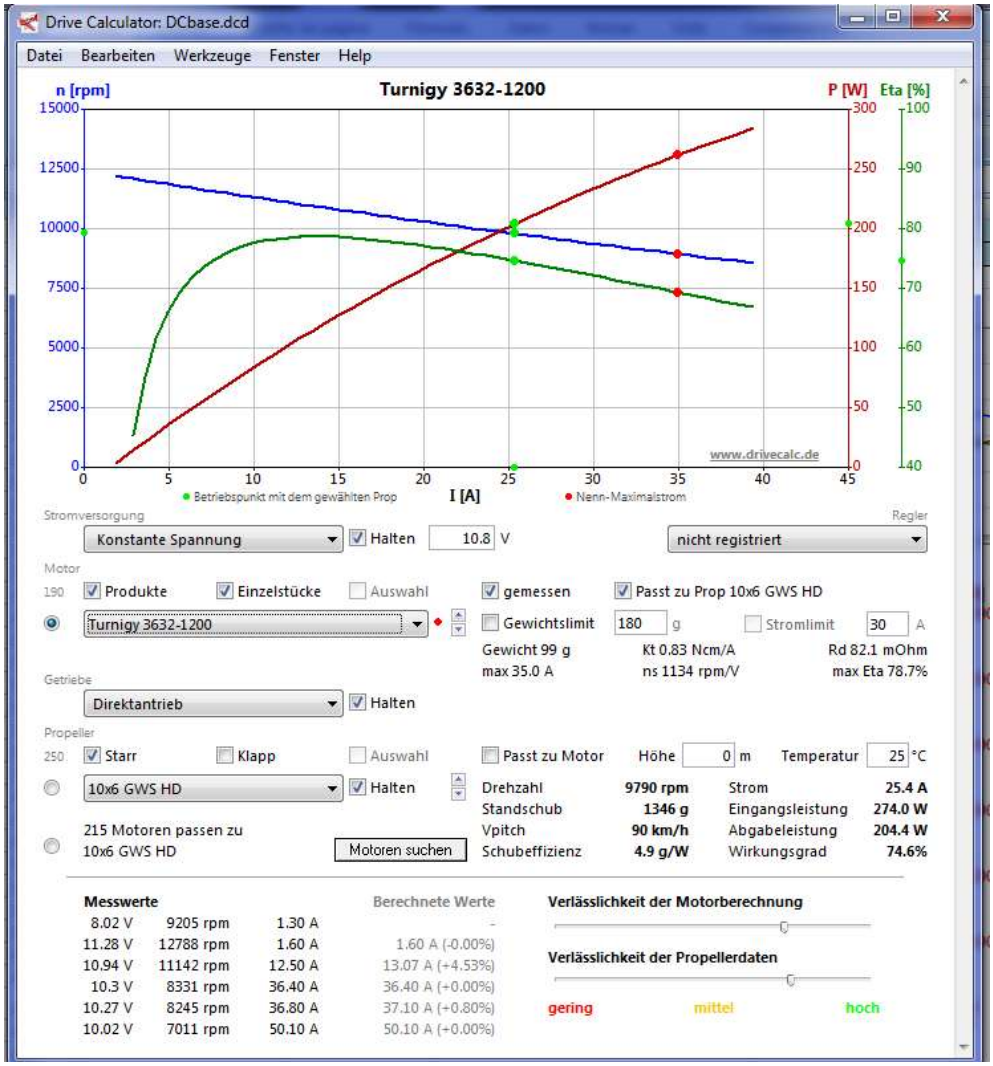


Ilustración 20: drive calcultor 3.

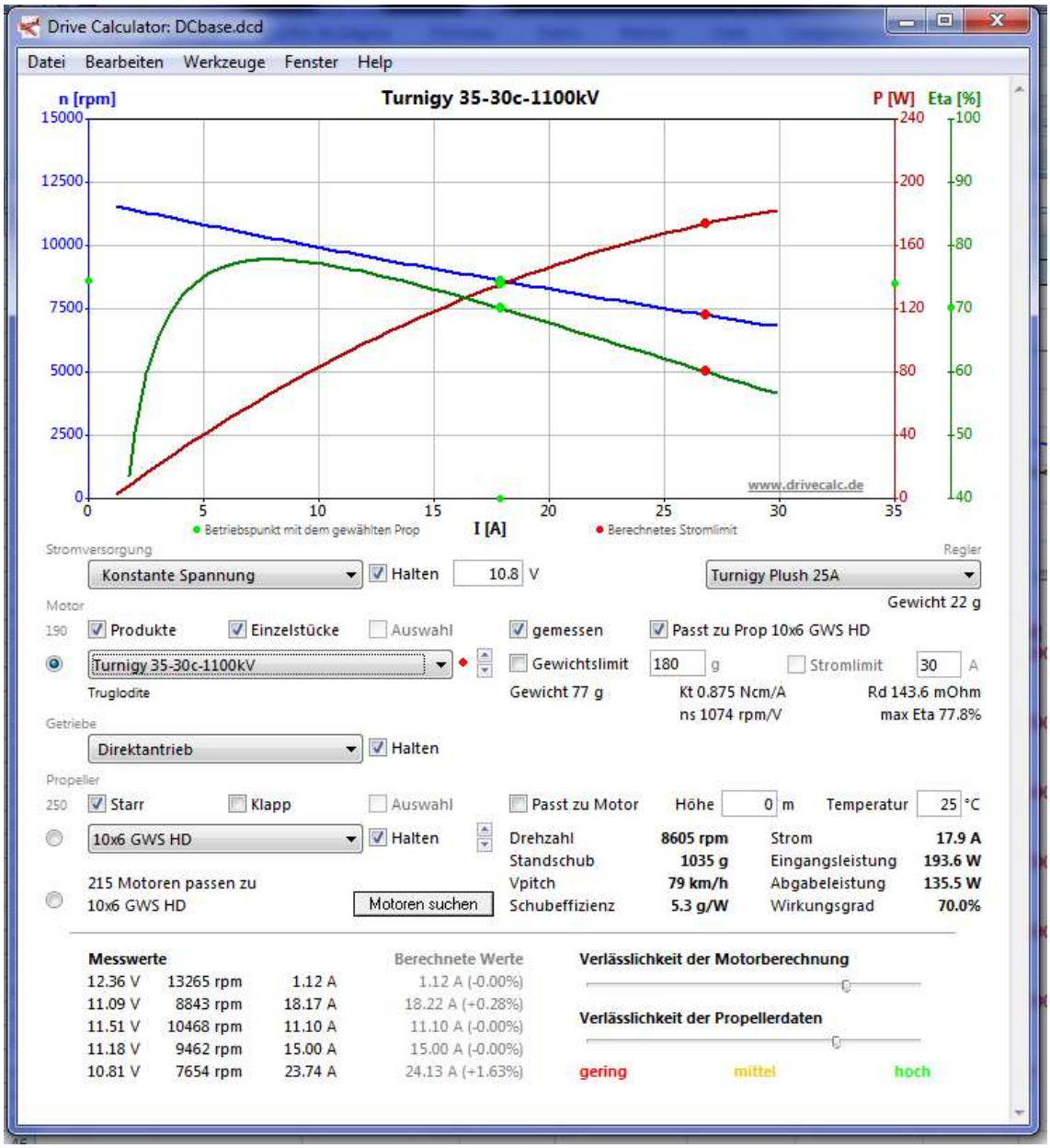


Ilustración 21: drive calculator 4.

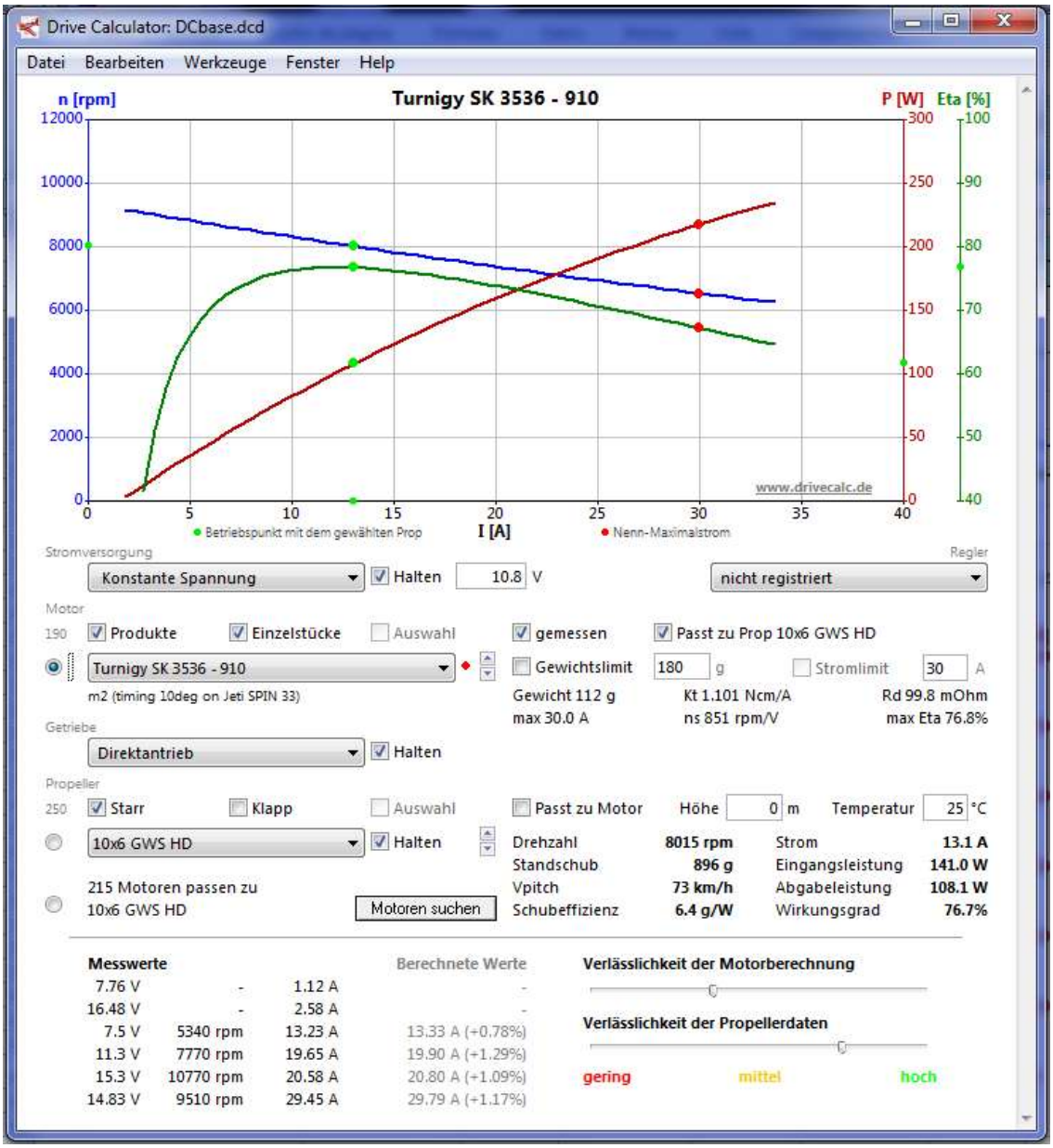


Ilustración 22: drive calculator 5.

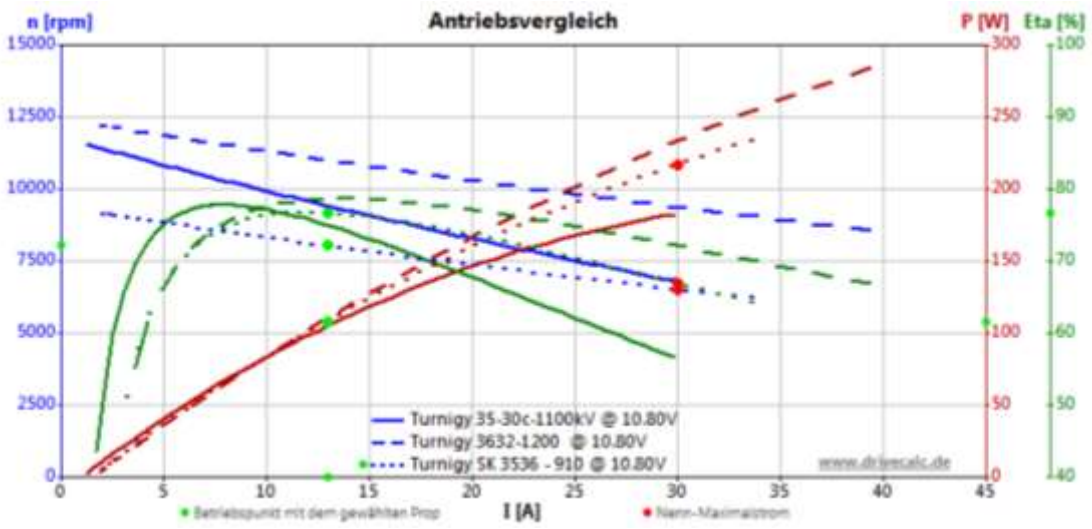


Ilustración 23: Drive calculator 6.

Conclusiones: hasta 20A la curva de potencia de los tres motores es similar. La velocidad del eje sí varía considerablemente entre los tres motores. Para evitar vibraciones se selecciona un motor que trabaje a bajas revoluciones.

Descartamos el modelo 3632, debido a las críticas de los consumidores que apuntan a una mala alineación de los polos magnéticos.

Elegimos el modelo 35-36A por trabajar a menos revoluciones y porque la curva de eficiencia no cae tanto como la otra, por lo que se reduce el calentamiento en el motor.

Ilustración 24: motor seleccionado.

Una vez elegido el motor y hélice analizamos la información:

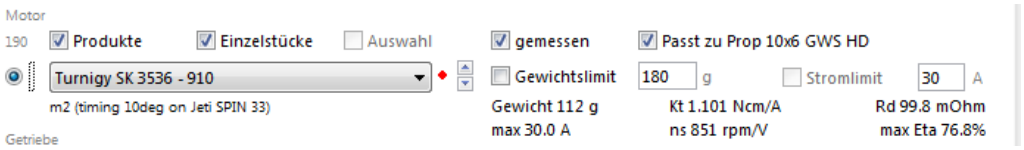


Ilustración 25: drive calculator 7.

Las características experimentales procedentes de una base de datos de este motor han sido:

- Peso: 112 gramos.
- Torque: 0.914 Ncm/A.
- Resistencia interna: 99.8 mOhm.
- Velocidad de giro: 851 rpm/V.
- Corriente de pico: 30A.
- Rendimiento máximo 76.8%.

En la segunda imagen obtenemos los resultados al simular la combinación del motor con la hélice 10x6:

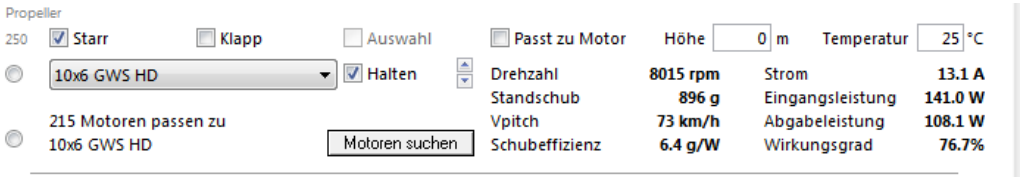


Ilustración 26: Drive calculator 8.

- Velocidad de giro: 8015 rpm.
- Empuje estático: 896 gramos.
- Velocidad pitch: 73 km/h.
- Eficiencia del empuje: 6.4 gramos/W.
- Corriente: 13.1A.
- Potencia de entrada: 141.0W.
- Potencia de salida: 108.1W.
- Eficiencia: 76.7%.

2.3.4 Control de la velocidad de los motores (ESC)

Para controlar estos motores utilizamos los variadores ESC (Electronic Speed Controler), que están diseñados para dirigir a motores sin escobillas *brushles.* Estos pequeños controladores varían la corriente continua procedente de las baterías LiPo a corriente alterna de diferentes frecuencias, controlando así la velocidad de salida del motor. El principal parámetro para dimensionarlos es el amperaje máximo admisible, que debe ser superior a la máxima corriente que van a consumir los motores en continua.

Normalmente, poseen, internamente, un microcontrolador programable para configurar algunos parámetros. Esto depende de cada modelo. En este proyecto se han usado ESCs de la marca Turnigy, solamente programables con las tarjetas programadoras de la misma compañía. Se puede ver en la ilustración 27.

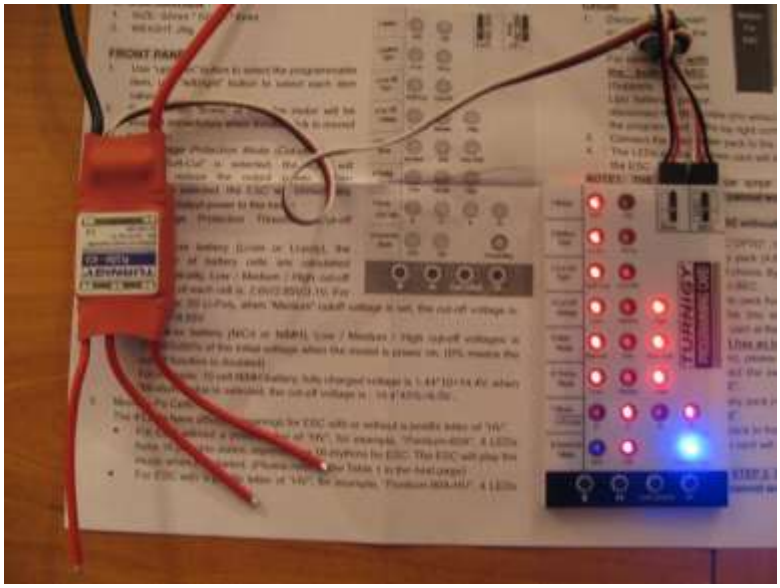


Ilustración 27: tarjeta programadora para ESCs.

Los parámetros variables son:

- Activar el freno (yes/no)
- Tipo de batería (LiPo/NiMh)
- Tipo de Corte ante batería baja (soft-cut/ off-cut)
- Corte por voltaje (Low/Middle/High)
- Mode de arranque (Normal/soft/very soft)
- Temporizador (Low/Middle/High)
- Música en function de las células de la batería (D/C/B/A)
- Modo Governor (on/off)

Realmente los parámetros que vienen por defecto son los mejores y no se necesita cambiar nada, excepto si deseas un arranque super suave, que pueden ser útiles si la batería o el propio ESC no están bien dimensionados para soportar el pico en el arranque.

En la ilustración 28 se muestra el conexionado del ESC:

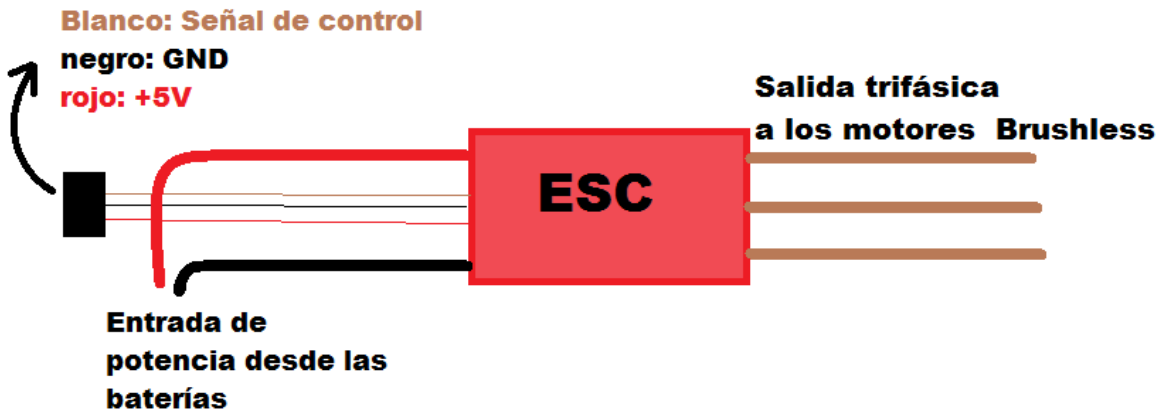


Ilustración 28: conexionado de los ESCs.

Para este proyecto se han usado unos variadores de corriente admisible de 40 A de la marca Turnigy, como los de la imagen 27, ya que los motores seleccionados pueden llegar a consumir 30 A en pico. Muchos de estos controladores reducen también el alto voltaje procedente de las baterías, 11,1 V a 5V para alimentar a la electrónica. Disponemos en este modelo una alimentación para la electrónica de 3A de máximo, más que suficiente para este proyecto. Es importante alimentar la electrónica sólo desde una fuente de 5V, ya que al ser transformadores de tensión diferentes puede alimentar a la electrónica con una pequeña variación de voltaje, creando recirculaciones de corriente indeseadas.

La señal que hay que enviar al variador para controlar la velocidad es igual a la usada para controlar motores servo, véase ilustración 29. Y su frecuencia de refresco es de 20 milisegundos.

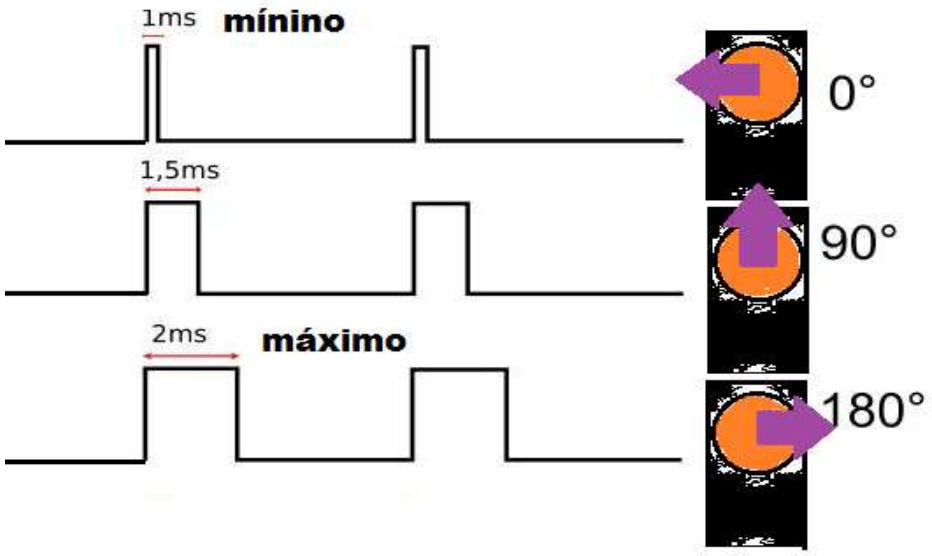


Ilustración 29: señales tipo "servo".

2.3.5 Rango de velocidades de los motores.

Para dirigir el hexacóptero se va a controlar la aceleración mediante el canal tres del emisor de radio. El rango de acción de las emisoras de radio control suele ser (0, 2000). Utilizando una librería <APM_RC.h> para mandar las velocidades a los variadores ESC que usa esta escala. En la siguiente ilustración 30 se detalla el rango de velocidades utilizado.

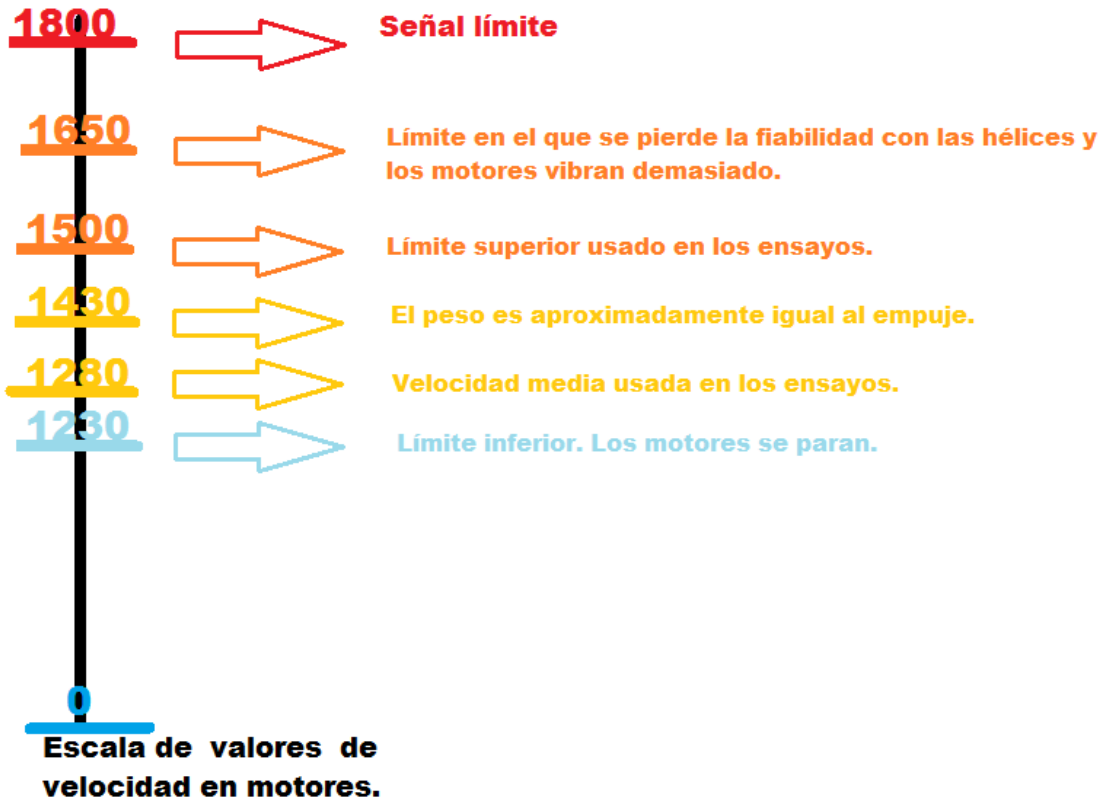


Ilustración 30: rango de velocidad de los motores.

2.4 Baterías y tiempo de vuelo.

Se han usado baterías de polímeros de Litio, conocidas como LiPo. Esta tecnología de baterías es la evolución de las baterías Li-ion y han remplazado a la antiguas NiMh o NiCd, ya que dobla la densidad de energía y aumenta la capacidad de descarga.

Se quiere que el tiempo de vuelo ronde los 10 minutos. Como su cálculo sólo lo podemos saber a partir de datos experimentales, para calcular una estimación a la baja, se procede de la siguiente manera:

La corriente máxima en continuo de cada motor es 13.1A, por lo que los 6 motores van a ser 78.6A.

Se suponen dos casos de baterías: 2 baterías de 4000mAh y 2 baterías de 5000 mAh.

En el primer caso tendremos un total de **8000mAh,** eso significa que con la batería se pueden obtener 8000 mA durante una hora.

$$tiempo\ de\ vuelo\ mínimo = \frac{8000}{1000} Ah \cdot \frac{1}{78.6A} = 0.10178\ h$$

$$tiempo\ de\ vuelo\ mínimo = 0.10178 \cdot 60 = 6.11\ minutos$$

En el segundo caso de **10000 mAh:**

$$tiempo\ de\ vuelo\ mínimo = \frac{10000}{1000} Ah \cdot \frac{1}{78.6A} = 0.1272\ horas$$

$$tiempo\ de\ vuelo\ mínimo = 0.1272 \cdot 60 = 7.63\ minutos$$

Estos tiempos serían los obtenidos en el peor de los casos en los que cada motor trabajase al máximo. Si esto fuese así obtendríamos 896g de empuje en cada motor, un total 5376 gramos de empuje cuando el prototipo pesará menos de 3000 gramos.

Se puede calcular la aceleración máxima de subida:

Suponemos que cada motor da una media de 850 gramos de empuje, el total sería 5100 gramos. Cómo el peso del prototipo va a ser cómo máximo 3000g (sin carga), obtenemos un empuje neto ascendente de 2100gramos.

Según la segunda ley de newton: $F = m_{prototipo} \cdot a$ y sustituyendo la fuerza por $F = m_{empuje} \cdot g$ se obtiene:

$$m_{empuje} \cdot g = m_{prototipo} \cdot a$$

$$a_{neta} = \frac{m_{empuje} \cdot g}{m_{prototipo}} = \frac{2.300 \cdot 9.8}{3.000} = 7.51 \frac{m}{s^2}$$

2.5 Cableado.

El cable utilizado está recubierto de silicona pura, se pueden ver los diferentes grosores en la ilustración 31, y está muy extendido en el campo del aeromodelismo y sus principales características son:

- Para igual amperaje son mucho más flexibles que un cable tradicional.
- Soportan 200 °C.
- Baja resistencia y por tanto alta conductividad.
- Están compuestos por cables de cobre muy finos, entre 0,06 y 0,08 mm, dependiendo del modelo.

Ilustración 31: diferentes grosores de cable AWG.

En el prototipo se han utilizado dos grosores: 12AWG y 16AWG en colores negro y rojo.

Tabla de corrientes máximas admisibles dada por el fabricante:

AWG	Área	Diámetro	Corriente Máxima
20	**0.518mm²**	**0.812mm**	**Hasta 12A**
18	**0.82mm²**	**1.02mm**	**Hasta 20A**
16	**1.31mm²**	**1.29mm**	**Hasta 30A**
14	**2.08mm²**	**1.63mm**	**Hasta 45A**
12	**3.31mm²**	**2.05mm**	**Hasta 70A**
10	**5.26mm²**	**2.59mm**	**Hasta 120A**

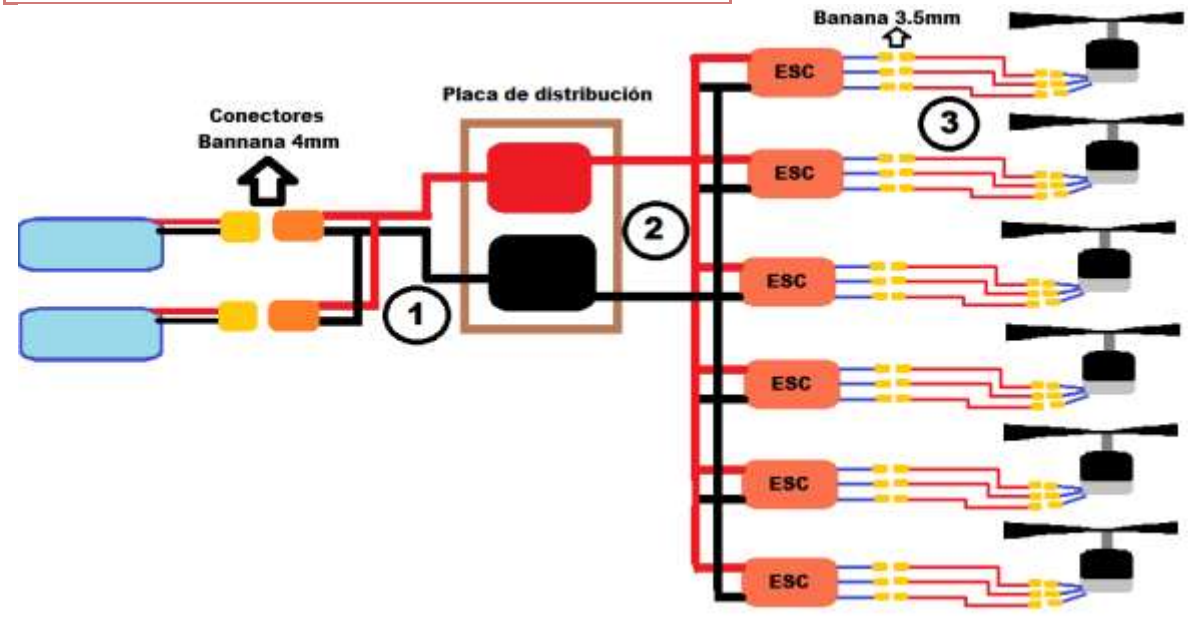


Ilustración 32: diagrama del cableado.

En la ilustración 32 se puede distinguir tres segmentos de cable, marcados con los números 1, 2 y 3. Para el segmento 1, que conecta las baterías con la placa de distribución, la máxima corriente es 60A. El tramo 2, que conecta la placa de distribución con los ESC, se utiliza el cable que llevan los variadores que es del grosor 16AWG. Para el último segmento, número 3, que conecta la salida de los variadores de velocidad con los motores, la máxima corriente es 11'5A y usaremos 16AWG.

Se han usado dos tipos de conectores:

- Conectores "Bannana 3,5mm" que soportan alrededor de 80A.
- Conectores "Bannana 4 mm" soportan hasta 93A aproximadamente y tienen protección contra inversión de la polaridad. Estos conectores son dobles, como se puede ver en la ilustración 33.

Ilustración 33: conectores Banana 4mm.

2.6 Resumen de costes

Uno de los objetivos era desarrollar una plataforma de bajo costo, aunque algunos imprevistos ha hecho, que durante la construcción se han empleado más materiales de lo debido. En la tabla 5 y en la gráfica 34 se muestra cómo se han distribuido los gastos. En los archivos <Materiales.xlsx> se puede encontrar más detalladamente el historial de compras y presupuestos así como el proceso de elección de los materiales.

Tabla 5

Componentes	Coste (Dólares $)	Envío (Dólares $)
Emisor-receptor RC	54,95	
Baterías	128,78	
Motores	131,81	
ESC	189,90	
Hélices	25,66	
Cableado y conectores	25,00	
Electrónica	250,00	29,00
Magnetómetro	44,00	
Chasis	118,00	21,00
Soporte cámara	53,25	
Total	1071,35	

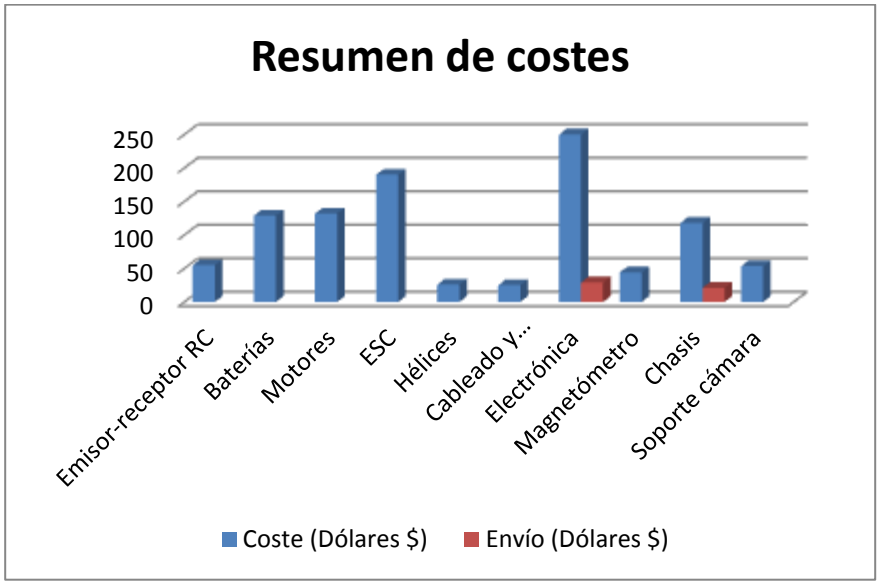


Ilustración 34: gráfica Resumen de costes.

En el caso de que quisiéramos construir un nuevo hexacóptero los costes se reducirían, pues algunos productos han bajado su valor como la electrónica y el chasis, y otros como la radio y las baterías de repuesto serían aprovechadas.

Tabla 6

Componentes	Coste (Dólares $)	Envío (Dólares $)
Emisor-receptor RC	0	
Baterías	60	
Motores	131,81	
ESC	139,9	
Hélices	25,66	
Cableado y conectores	25	
Electrónica	199	29
Magnetómetro	0	
Chasis	110	21
Soporte cámara	35	
Total	776,37	

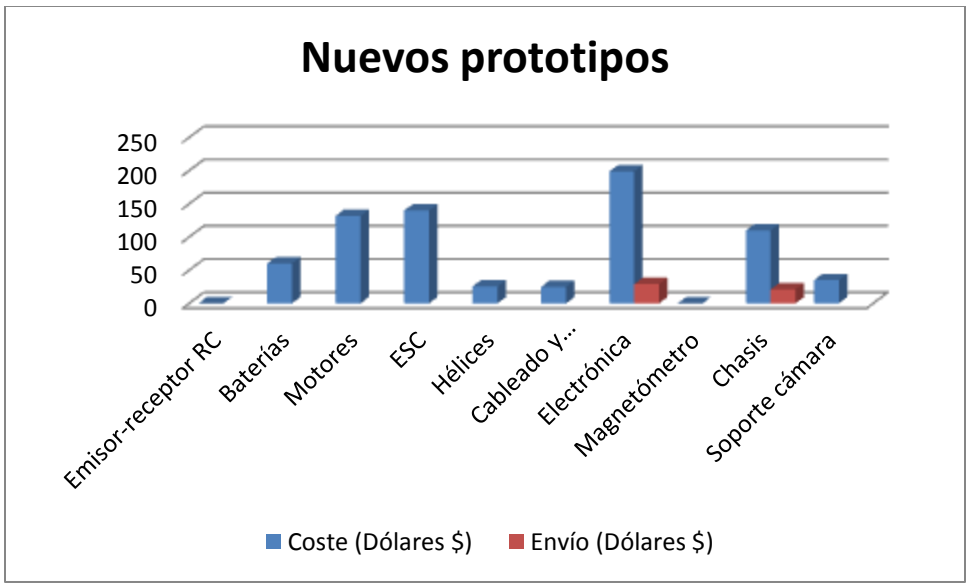


Ilustración 35: gráfica de costes de los nuevos prototipos.

2.7 Comunicaciones inalámbricas

Se van a usar diferentes tipos de transmisión de datos, aunque en el prototipo inicial solo se ha usado el envío de datos por radio, en la tabla 7 se muestran las futuras conexiones de datos.

Tabla 7

Tipo	Frecuencia	Descripción
Radio	2.4 GHz	Turnigy RC control transmitter
Telemetría	433 MHz	Xbee 433MHz module
Video	5.8 GHz	5,8 GHz Audio/video transmitter

Las diferentes frecuencias se han elegido dependiendo de los productos disponibles y de la legislación vigente en cada país respecto al uso civil de frecuencias de radio.

Se ha elegido la frecuencia de 2.4 GHz para la radio por la existencia de un emisor/receptor de radio con altas cualidades y bajo coste. Para la telemetría, las frecuencias más usadas en Radio Control son 900 MHz y 433 MHz dependiendo del país. En Europa la banda de 900 MHz está reservada para teléfonos móviles, aunque si se podría usar 868 MHz como alternativa. Mediante la conexión con Xbee se establece una conexión bilateral, igual que establecemos cuando conectamos por USB la placa.

Ilustración 36: telemetría con Xbee.

Para la transmisión de video se usarían módulos comerciales de fácil uso, que sólo habría que conectarlos. En las ilustración 37 se pueden ver el emisor y el receptor.

Ilustración 37: emisor y receptor de video inalámbricos.

Una alternativa para la telemetría es establecer una conexión de video normal e incluir los datos que queramos dentro de la imagen. Esto se conoce como OSD, del inglés “On Screen Display”. Esto se consigue añadiendo un dispositivo que recibe la señal de video de la cámara, le añade los datos y manda la nueva señal de video al emisor de radio. Se puede ver un ejemplo en la ilustración 38.

Ilustración 38: sistemas OSD.

2.8 Electrónica: microcontrolador y sensores.

Para controlar la aeronave se necesitan, como mínimo, un microcontrolador principal y una placa de sensores llamada IMU, Inercial Measurement Unit (Unidad de medida de Inercia).

2.8.1 Microcontrolador

Se ha usado una placa controladora diseñada para vehículos autónomos aéreos, aviones o helicópteros, llamada AutoPilot. En este caso el modelo es "*ArduPilot Mega 1.4 con ATmega2560*", una placa de libre distribución de hardware y software abierto, conocido también como "Open Source". Esta placa es el producto de un proyecto internacional on-line "Diy drones" (www.diydrones.com).

Desde 2008 un equipo de desarrolladores coopera desinteresadamente en el desarrollo de esta plataforma abierta compartiendo sus conocimientos y experiencias. Los diseñadores del hardware han sido principalmente Chris Anderson y Jordi Muñoz.

El Microcontrolador principal tiene instalado un *bootloader* que permite cagar programas mediante USB por el puerto serie, sin la necesidad de un programador AVR o también llamado "quemador".

En febrero del 2012 salió al mercado un nuevo modelo "APM 2.0" con notables mejoras y ya están trabajando en la versión 3.0 basada en ARM con mucha más potencia de cálculo. En la ilustración 39 vemos la evolución de los distintos modelos de la familia "*ArduPilot*".

The ArduPilot family

Autopilot	ArduPilot (aka "Legacy")	ArduPilotMega APM 1 – 1280	ArduPilotMega APM 1 – 2560	ArduPilotMega APM 2
Date of introduction	Q1 2009	Q1 2010	Q1 2011	Q4 2011
Status	Discontinued	Discontinued	Active	Active
Processors	atmega 328, attiny	atmega 1280, atmega 328	atmega 2560, atmega 328	atmega 2560, atmega 32u2, MPU-6000 DMP processor
Onboard sensors	None. External: Thermopiles or optional ArdulMU	3-axis gyro, 3-axis accel, baro, optional mag	3-axis gyro, 3-axis accel, baro, optional mag	6-axis MPU6000 (gyro+accel), baro, mag, GPS
Datalogging memory	None	2MB	2MB	4MB
Size	30x50x30mm	40x72x20mm	40x72x20mm	40x65x10mm
Assembly required	Lots!	Some soldering	Some soldering	None!

Ilustración 39: evolución de ArduPilot

ArduPilot Mega 2560 es una placa de autopilotaje compatible con Arduino. Esta tiene dos microcontroles de bajo consumo de 8-bit basados en AVR RISC.

- **ATemega 2560 CPU:** es un microcontrolador basado en AVR RISC de 8-bit. Tiene una memoria flash de 256KB ISP, 8KB SRAM, 4KB EEPROM, 86 líneas GPIO, 32 registros de trabajo GP, contador en tiempo real, 6 contador/timer con modo comparativo, PWM, 4 USARTs, 10-bit A/D conversor de 16 canales y interfaz JTAG para debugging on-chip. La placa alcanza una velocidad de procesado de 16 MIPS a 16MHz y opera de 4.5 a 5.5 Voltios. Cada ciclo de instrucción puede generar 1 MIPS por cada MHz, lo que lo hace ser muy potente.
- **ATmega328P CPU:** tiene 32KB de memoria flash ISP con función read-while-write, 1KB EEPROM,2KB SRAM, 23 líneas GPIO, 32 registros de trabajo GP, 3 contador/timer con modo comparativo, interrupciones internas y externas, programable vía serial USART, puerto serial SPI convertidos A/D de 10-bit, watchdog timer con oscilador interno, y cinco modos de ahorro de energía configurables por software.

Ilustración 40: ArduPilot Mega 2560

2.8.2 Placa de sensores: "Ardupilot Mega IMU".

Una unidad de medición inercial o IMU (*Inertial Measurement Unit*) es un dispositivo electrónico que mide e informa acerca de la velocidad, orientación y fuerzas gravitacionales de un aparato, usando una combinación de acelerómetros y giróscopos. La IMU es el componente principal en los sistemas de navegación inerciales en todo tipo de aeronaves, buques, UAVs o misiles.

Está formado normalmente por tres acelerómetros y tres giróscopos colocados en ejes ortogonales entre sí. De esta manera se detectan los cambios de los tres ejes de navegación (Roll, Pitch y Yaw).

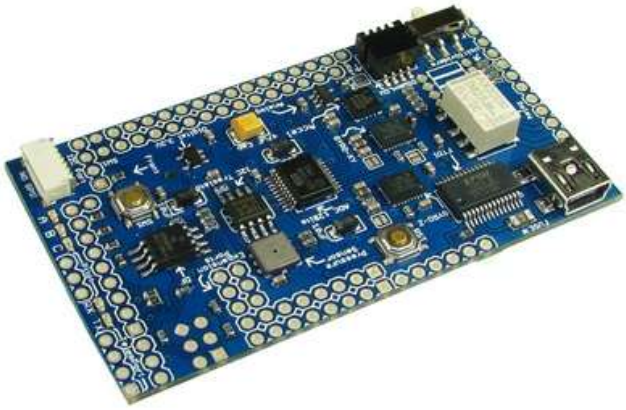


Ilustración 41: placa de sensores IMU.

La placa IMU usada ha sido IMU "Oilpan Rev-H 1.0" cuyas principales características son:

- Acelerómetro ADX330 de tres ejes.
- Sensor de presión barométrica absoluta.
- Giróscopo en los tres ejes.
- Sensor magnetómetro.
- Conversor analógico-digital de 12 bit para aumentar la resolución en la medida de los acelerómetros y giróscopos.
- 16 MB de Data Logger en memoria Flash, Atmel 450B161D 16-Megabit Serial DataFlash chip.
- Switch, interruptor y relé programables.
- FTDI chip, para crear un puerto USB nativo para conectar la placa principal con el ordenador.
- Puerto I2C auxiliar.
- Divisores de voltaje para medir el estado de las baterías.

La parte más importante de la placa IMU son los acelerómetros y giróscopos. El acelerómetro es ADX330 de ANALOG DEVICES y su hoja de características se puede encontrar en la carpeta de archivos anexos, "ADXL330_0.pdf". Para los giróscopos se han utilizado dos, el modelo IDG500 para los ejes X e Y y el ISZ 500 para el eje Z. Ambos están fabricados por Ivensense. Los datasheets completos se pueden encontrar en la carpeta anexa "ISZ-0500.pdf" y "IDG500.pdf"

El conjunto controlador más placa IMU están diseñadas para encajar entre sí quedando un único bloque de 40x72x20mm como se puede ver en la ilustración 42.

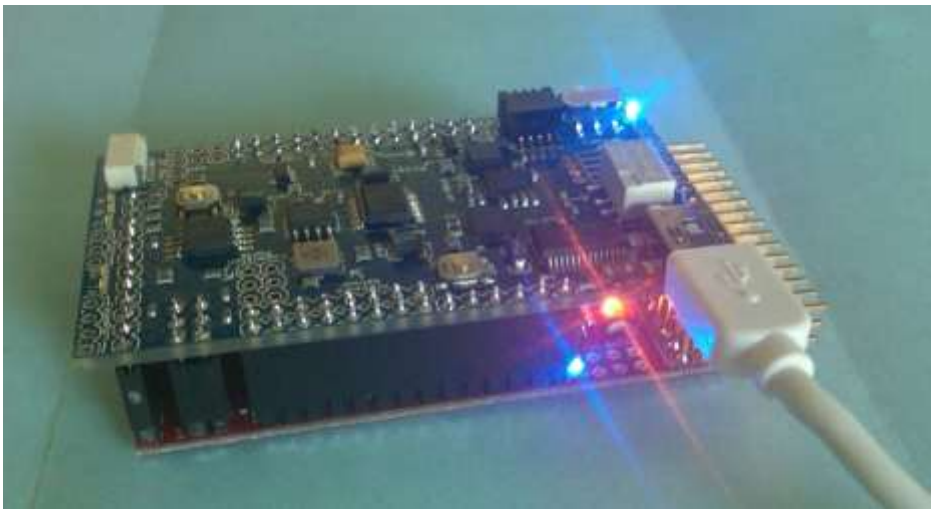


Ilustración 42: conjuto de ArduPilot Mega y sensores.

Se pueden encontrar los archivos de la placa PCB en el anexo. Archivos "ArduPilotMegaShield_F2.sch" y "ArduPilotMegaShield_F2.brd" para el esquemático y la placa respectivamente. A continuación, en la figura 43 se puede ver la distribución de la placa.

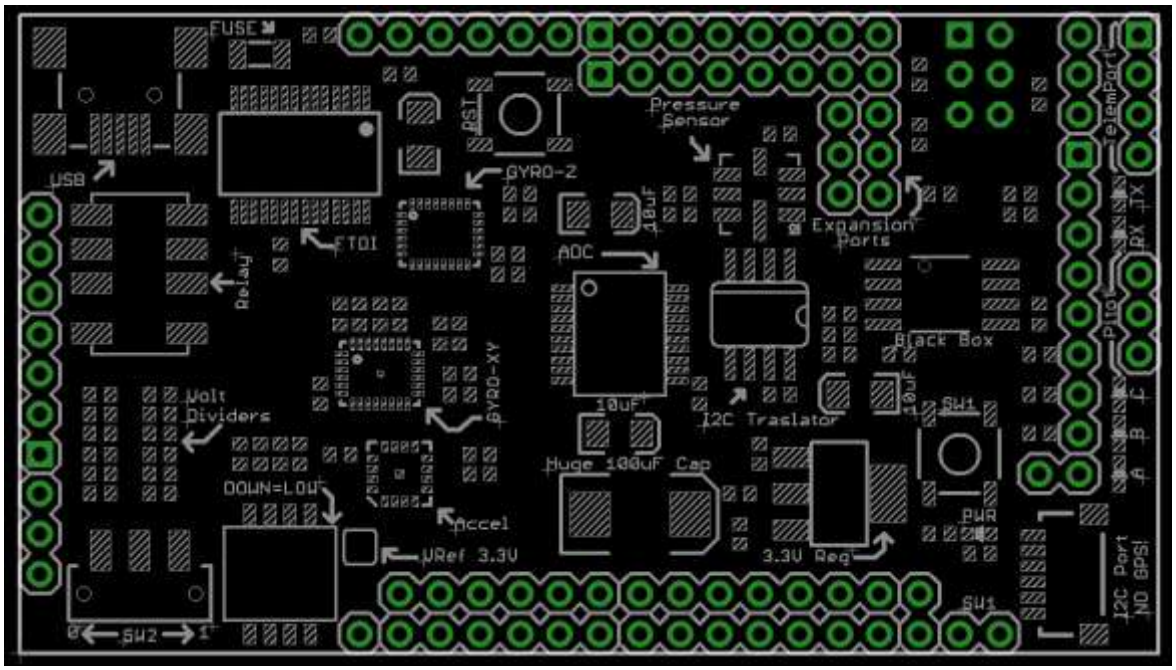


Ilustración 43: distribución de la placa IMU.

2.8.3 Interpretación de los ángulos leídos por el acelerómetro.

El sensor mide la componente de la aceleración de la gravedad en 3 ejes ortogonales entre sí, [X Y Z], como se puede ver en la ilustración 44. Al calibrarlo inicialmente se ajusta de forma que el eje Z coincida con la vertical y que las componentes X e Y sean nulas.

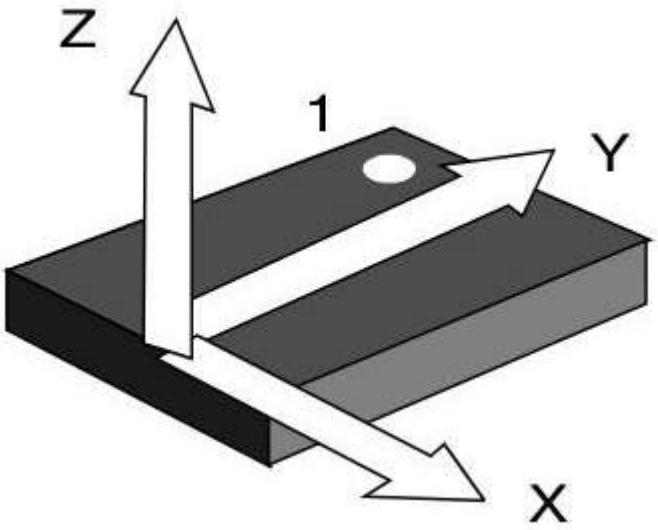


Ilustración 44: ejes en el acelerómetro.

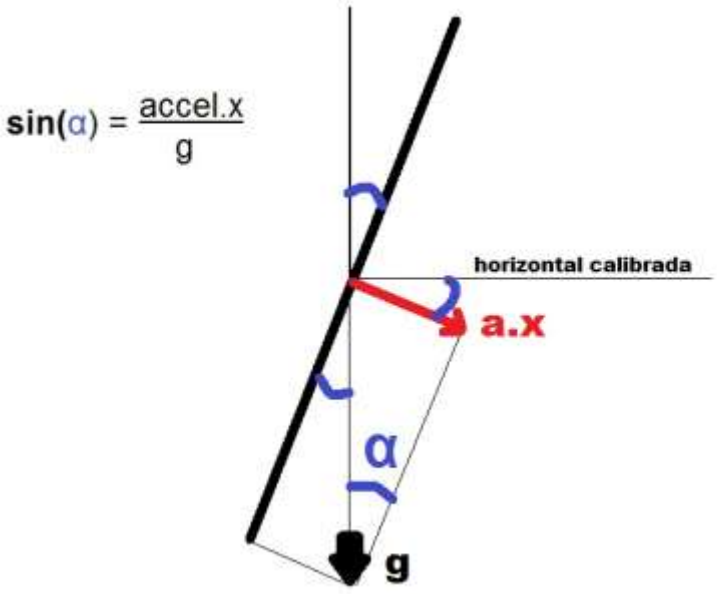


Ilustración 45: cálculo del ángulo.

Se han leído datos correspondientes a unas oscilaciones libres del balancín:

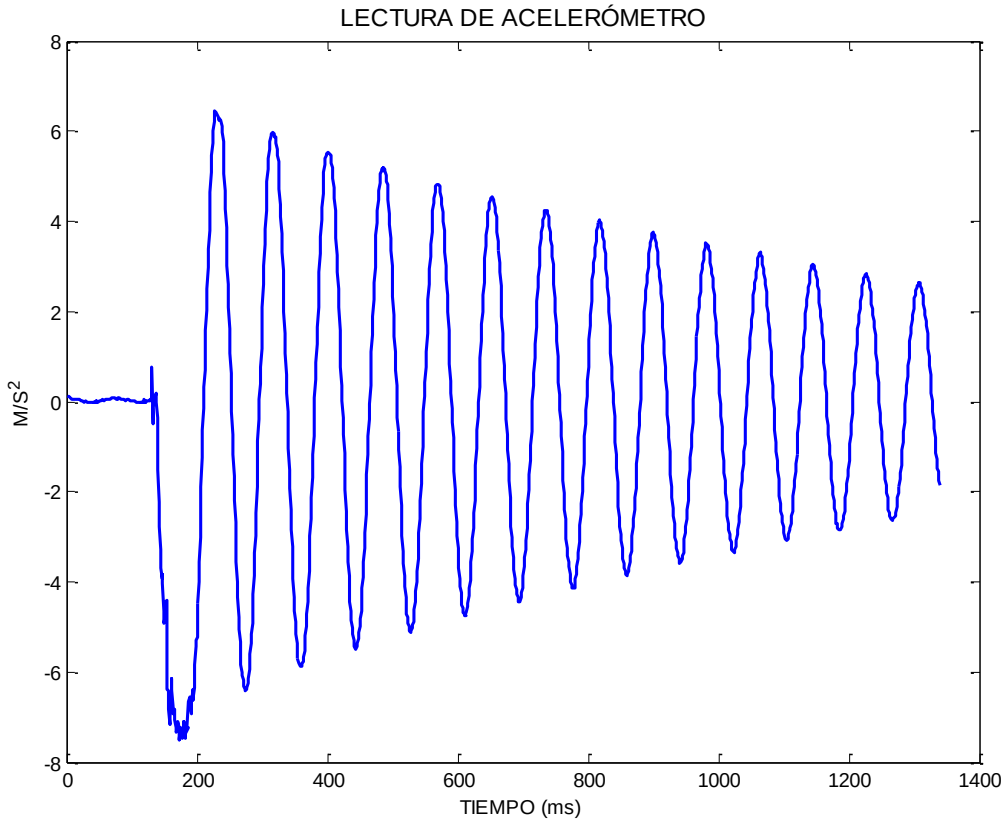


Ilustración 46: lectura de ángulos

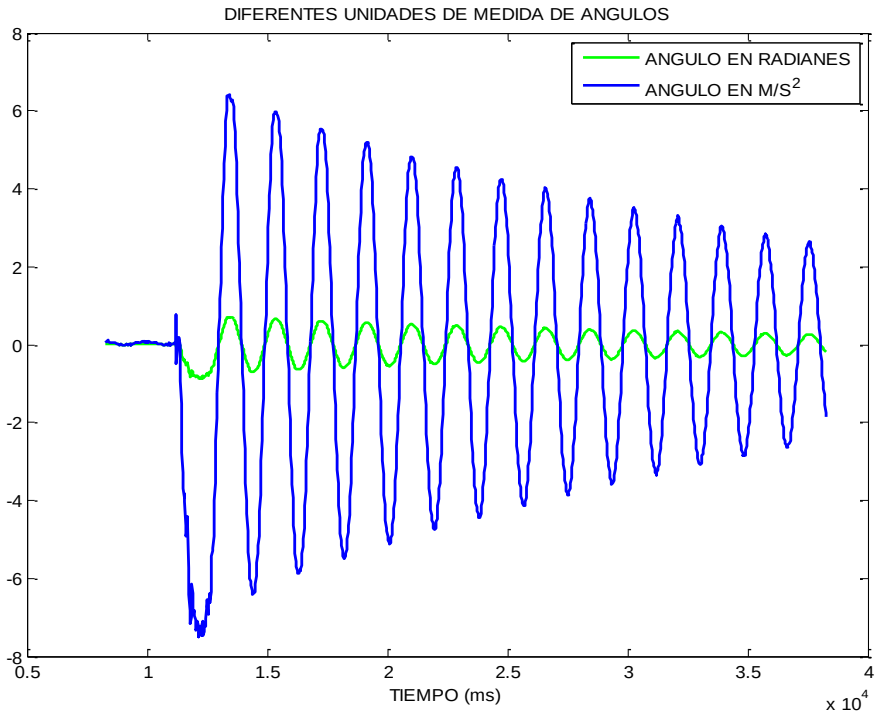
Por trigonometría:

$$\propto = \operatorname{asin}(accel.\frac{x}{g})$$

Usando los siguientes comandos en matlab:

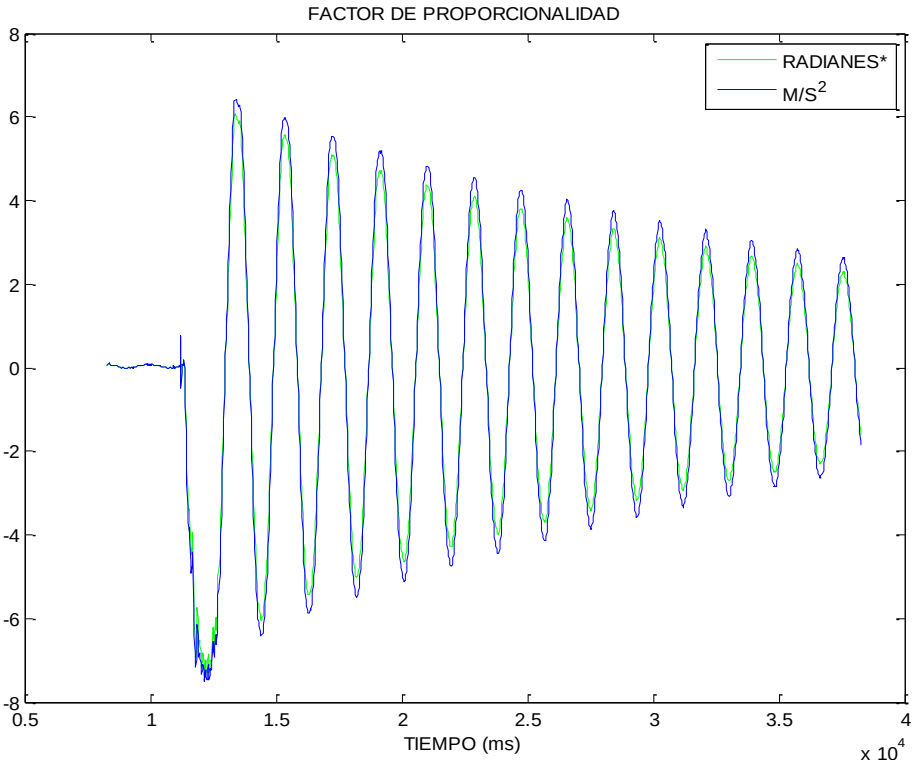
```
for i=1:1792
        angle(i)=asin(d2(i,2)/9.8119);
end
```

```
>> plot(d2(:,7),angle,'g');hold on;
>> plot(d2(:,7),d2(:,2),'b');
```



Multiplicando por el factor proporcional: **7,84/0,9257** hallados en el primer pico, obtenemos la siguiente gráfica.

*plot(d2(:,7),angle*7.84/0.9257,'g');hold on; plot(d2(:,7),d2(:,2),'b');*



Se concluye que el ángulo calculado y el valor medido por el acelerómetro no son linealmente dependientes ya que los picos no coinciden, pero para esta aplicación la aproximación en la medida es suficiente.

2.8.4 Cálculo de la altitud

Para calcular la altura se ha utilizado un sensor de presión absoluta y temperatura modelo BMP085 de Bosch, cuyas características principales se muestran en la tabla 8:

Tabla 8

Resolución en Presión	**1 Pa**
Resolución en Temperatura	0.1 ºC
Rango de Altitud medida	-500m a +9000m (sobre el nivel del mar)
Alimentación	3.3 V
Consumo	30 µA

Desde el barómetro obtenemos los valores de la presión y con la Ecuación Barométrica Internacional que es una fórmula experimental, se calcula la altura referida al nivel del mar.

$$Altitud = 44330 \cdot (1 - \left(\frac{Presión}{Presión_0}\right)^{\left(\frac{1}{5.255}\right)} \text{ metros.} \quad Presión_0 = 101325\ Pa.$$

El valor de $Presión_0$ habría que calibrarlo con nuestro sensor, ya que puede variar dependiendo de las condiciones atmosféricas.

Otra manera de estimar la altura teniendo en cuenta que la presión baja 1hPa cada ocho metros. Es una aproximación no lineal, pero para esta aplicación podría ser válida.Representamos esta ecuación en la gráfica 47.

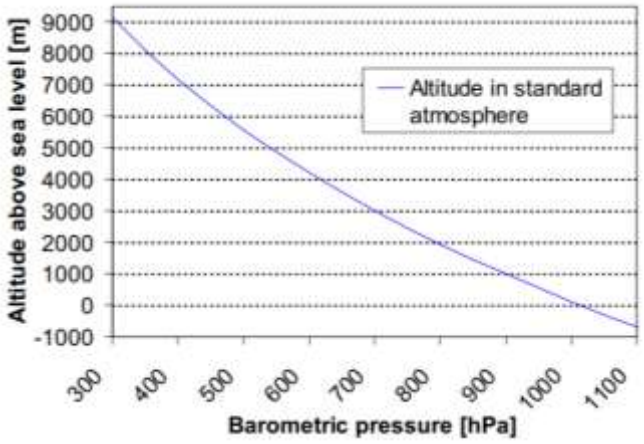


Ilustración 47: gráfica de la ecuación barométrica internacional.

2.9 Parámetros de vuelo

2.9.1 Legislación vigente

Desde el año 1994 en España está vigente el Reglamento de Circulación Aérea Operativa, en el que sólo se contemplan usos militares. Los usos civiles no están regulados todavía.

El 21 de febrero de 2012 se reunieron 12 ejecutivos de las principales empresas españolas del sector y se destacó el problema de la falta de regulación en España y en Europa. La lentitud con la que se legisla en la Unión Europea y toma decisiones está abriendo brecha entre el viejo y el nuevo continente.

Mientras en Estados Unidos el Congreso acaba de aprobar un plan para que en 2015 la operación de UAVs se inserte en el espacio aéreo del país, y así ha emitido el mandato a la FAA (Federación del Aire Americana).

La reciente ley americana, de 14 Febrero de 2012 obliga a la FAA a permitir que los drones ser usados para pequeñas misiones civiles comerciales como monitorizar oleoconductos, seguimiento de cultivos, fotografía aérea o la realización de películas en Hollywood. Por su parte la policía y los servicios de emergencia tienen libertad para utilizar sus dispositivos aéreos. Aunque esta ley tiene algunas limitaciones como el no poder volar a más de 400 pies de la emisora y tener que pasar algunos controles.

2.9.2 Ángulos de aviación

Los ángulos de navegación son ángulos de Euler usados para describir la orientación de una aeronave. 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. Se definen matemáticamente de forma similar a los ángulos de Euler, pero en vez de usar como línea de nodos el corte entre dos planos homólogos (por ejemplo el XY es el homólogo del XY), se utilizan dos planos no homólogos (por ejemplo XY e YZ).

Por ejemplo, en el dibujo adjunto que usa el convenio ZXY, se definen mediante los planos XY (plano perpendicular al eje "Z" usado en la primera rotación) del sistema de referencia, y el plano ZX del sistema móvil (plano perpendicular al eje "Y" de la última rotación).

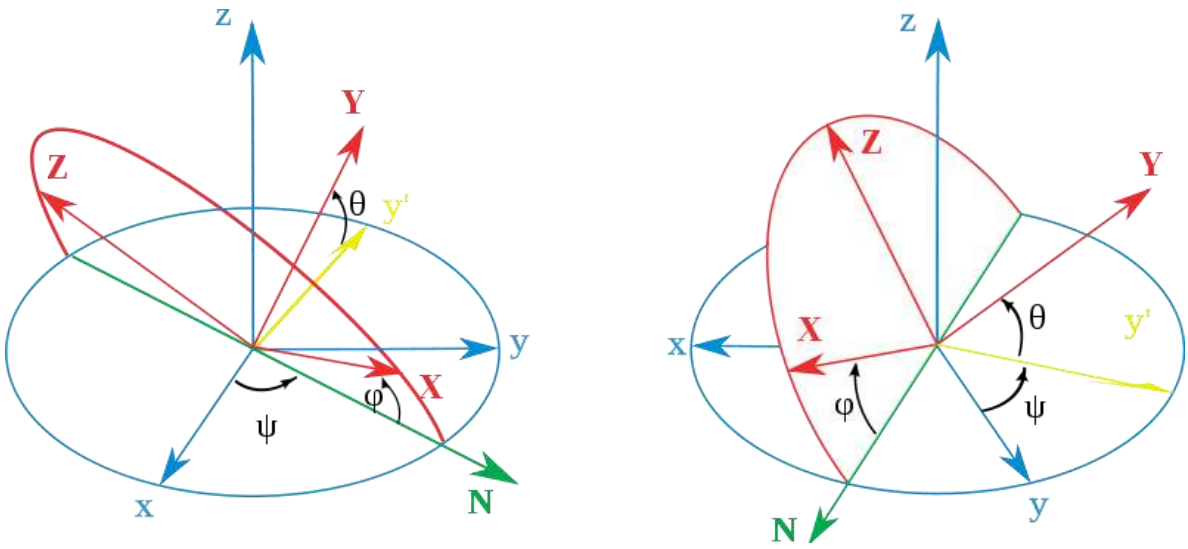


Ilustración 48: ángulos de aviación.

Los ejes de Yaw, Roll y Pitch se traducen al español como eje de guiñada, de alabeo y de cabeceo respectivamente. En este proyecto se van a utilizar, por comodidad, los nombres ingleses. Se muestran en la ilustración 49.

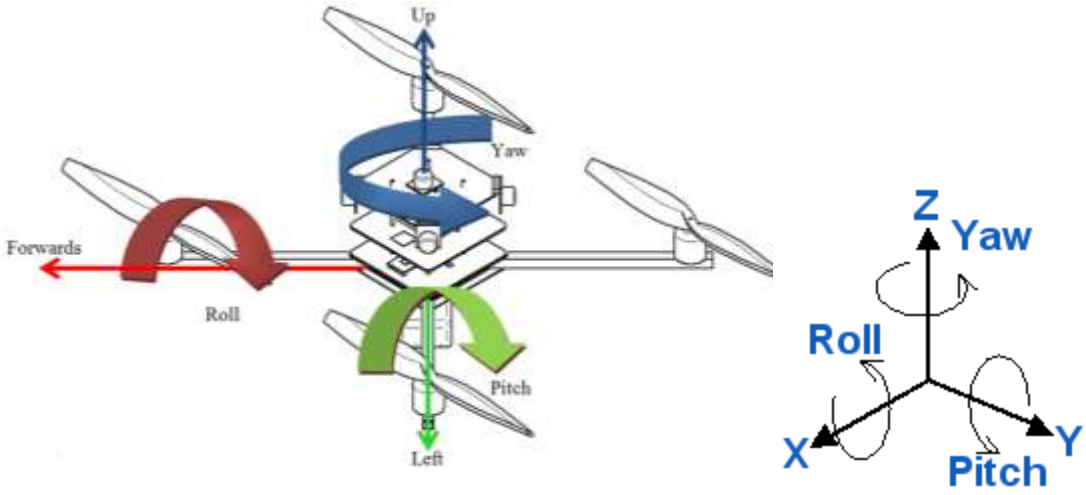


Ilustración 49: Pitch, Roll y Yaw

2.9.3 Cómo vuela un multicóptero

Primero se estudiará el caso del cuadricóptero, con cuatro motores, para luego ver el caso del hexacóptero. Existen 4 parámetros para describir los movimientos de este tipo de vehículo, la altitud, los ángulos Roll y Pitch y el ángulo Yaw. En las gráficas siguientes las flechas rojas indican la potencia del empuje de cada motor.

Cuadricóptero

El control de la altura es relativamente sencillo, basta con variar la potencia de los motores. Como se puede ver en la ilustración 50 igualando las velocidades de los cuatro motores y variando su valor se conseguirá más o menos altura.

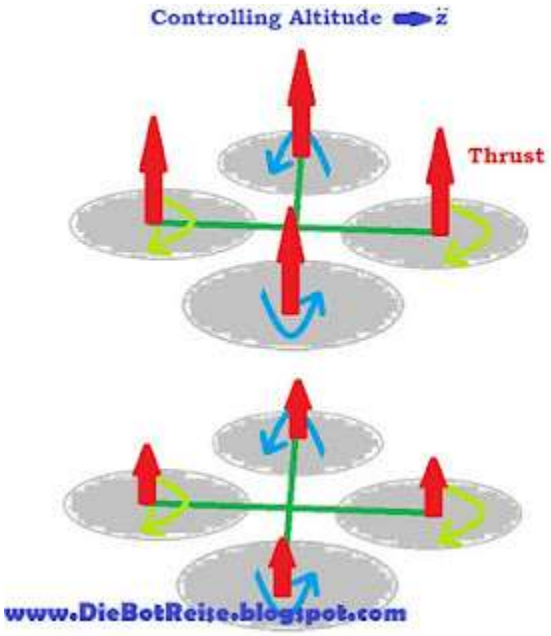


Ilustración 50: control de altura.

Para el control de los ángulos Roll y Pitch se fijan iguales las velocidades de los motores situados sobre el eje que se quiere cambiar y variando las velocidades de los motores perpendiculares de manera desigual, el cuadricóptero se girará en uno y otro sentido como podemos ver en la ilustración 51.

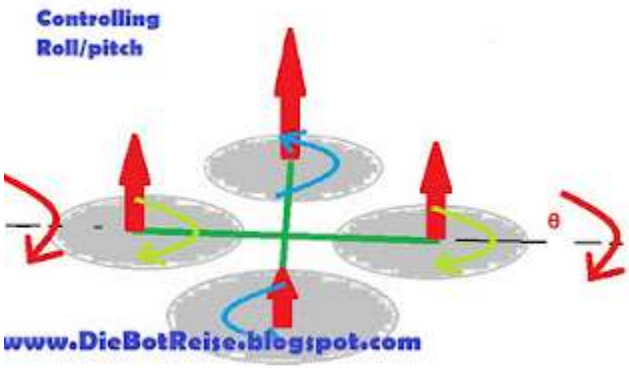


Ilustración 51: control del Pitch y Roll.

Para regular el ángulo Yaw, se fijarán los motores que giren en el mismo sentido, y variando los motores que giran en sentido contrapuesto se girará hacia un lado u a otro.

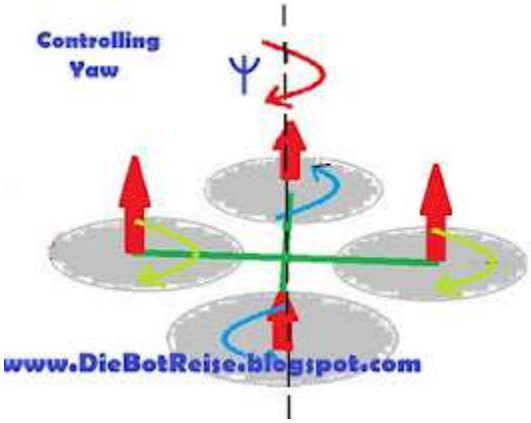


Ilustración 52: control del Yaw.

Hexacóptero

A partir de los 4 parámetros anteriores, Altura y ángulos Pitch, Roll y Yaw, tenemos que distribuir la señal a los 6 motores. El Throttle (acelerador) es un término constante que se va actualizando a 50Hz y a partir de este valor se suma o se restan los errores en los ángulos. En la ilustración 53 se puede ver el esquema, el bloque equivale a una serie de funciones que distribuyen la velocidad.

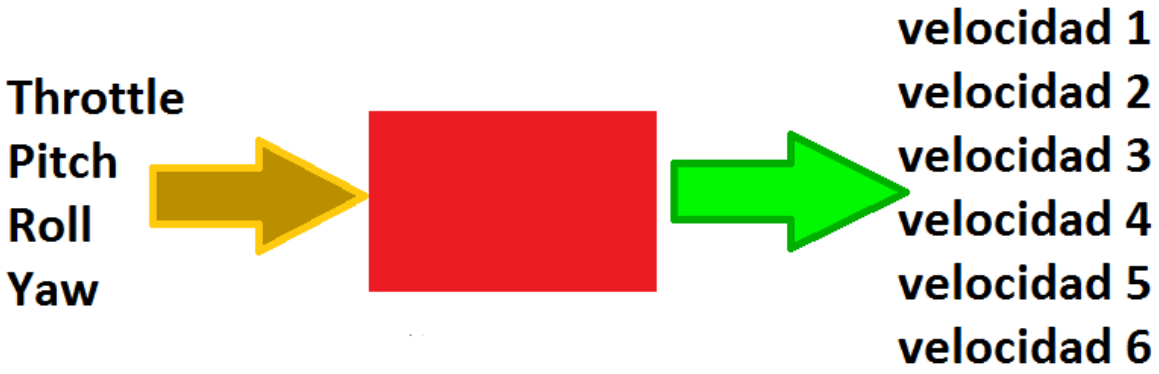


Ilustración 53: distribución de velocidades.

Se han numerado los motores siguiendo el siguiente orden, un motor con el uno, su contrapuesto con el dos, y así sucesivamente (ver en la ilustración 54). Donde los motores 1, 3 y 6 giran en sentido horario (Clockwise), y los 2, 4 y 5 giran en sentido antihorario (Counterclockwise).

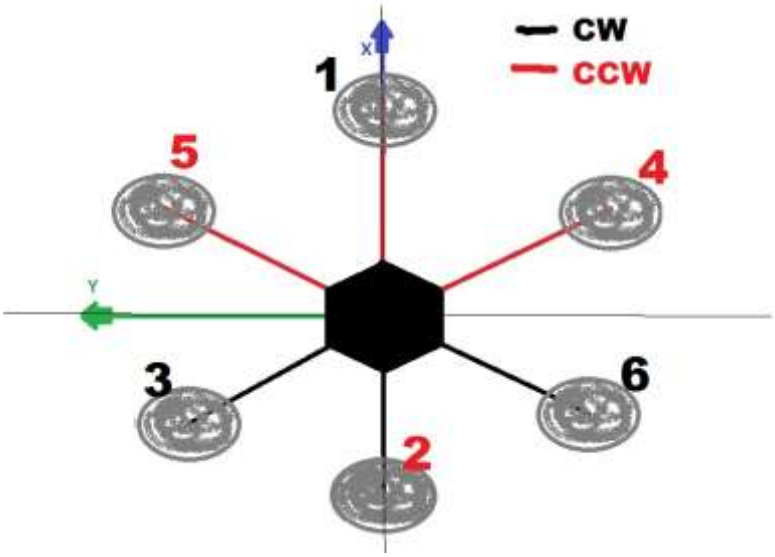


Ilustración 54: numeración de motores.

```
//Motores frontales
 valor1 = throttle            + 1.0* pitch;
 valor5 = throttle + 0.866* roll + 0.5* pitch;
 valor4 = throttle - 0.866* roll + 0.5* pitch
 //Motores traseros
 valor2 = throttle            - 1.0* pitch;
 valor3 = throttle + 0.866* roll - 0.5* pitch;
 valor6 = throttle - 0.866* roll - 0.5* pitch;
```

Para tener en cuenta el ángulo YAW se varía la velocidad de los motores que giran hacia el mismo sentido.

```
//YAW
  //Sentido horario
  valor1 += yaw;
  valor3 += yaw;
  valor6 += yaw;
  //Sentido antihorario
  valor2 -= yaw;
  valor4 -= yaw;
  valor5 -= yaw;
```

3 Software

3.1 Compiladores utilizados

El hardware utilizado es compatible con el entorno Arduino, y debido a su simplicidad se ha usado este programa. El compilador sólo tiene las funciones básicas, pero son suficientes. El entorno de Arduino es un software libre descargable desde la página oficial del proyecto Arduino. En concreto se ha usado la versión "1.00". Este software tiene un monitor serie usado para la transmisión de datos vía puerto serie. Arduino usa un lenguaje de programación C++ con algunas semejanzas al Java.

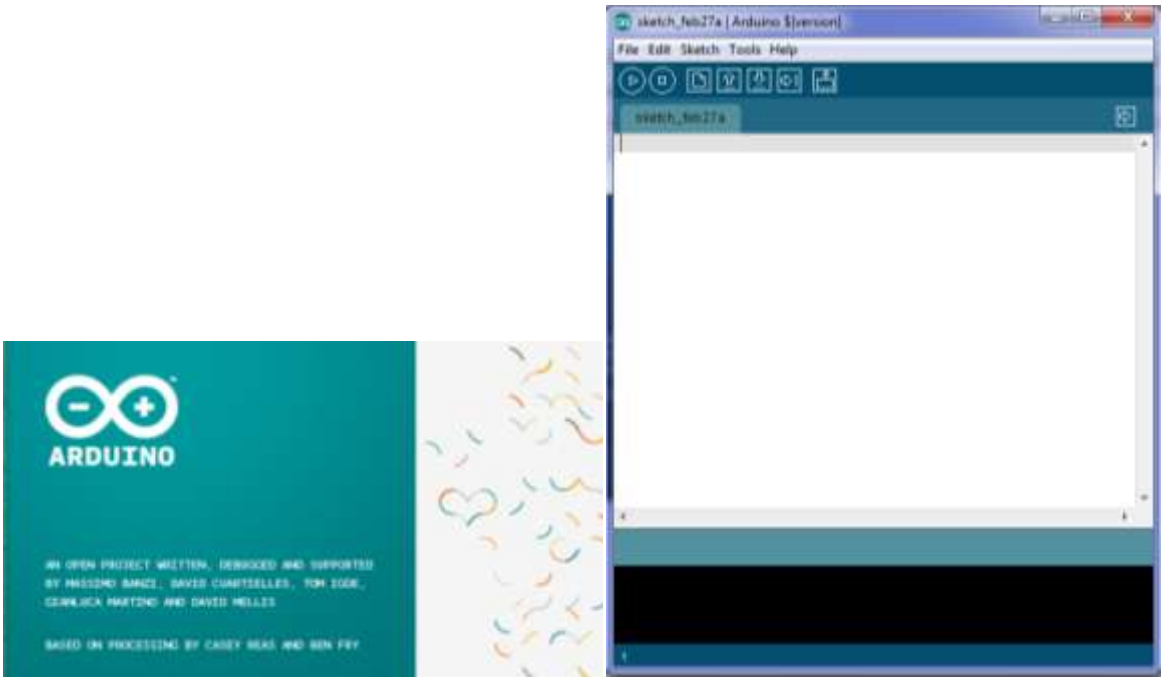


Ilustración 55: captura de pantalla del sotfware Arduino.

Un compilador más completo y también de libre distribución es el "Eclipse IDE for C/C++ Developers". Es un software que posee multitud de herramientas y opciones y podemos programar en distintos lenguajes para casi todos los microcontroladores. Para trabajar con Arduino se configura adecuadamente para AVR/Arduino. Este software se estuvo utilizando durante un tiempo, pero por simplicidad volvimos al entorno Arduino convencional.

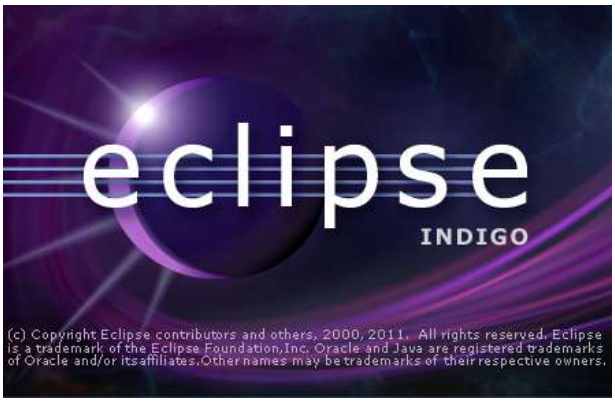


Ilustración 56: alternativa con programa Eclipse.

El programa utilizado para el entorno gráfico ha sido *Processing 1.5.* Este programa permite la creación de imágenes, animaciones e interactuar con ellas. Se puede decir que es el lenguaje equivalente de Arduino para ordenadores.

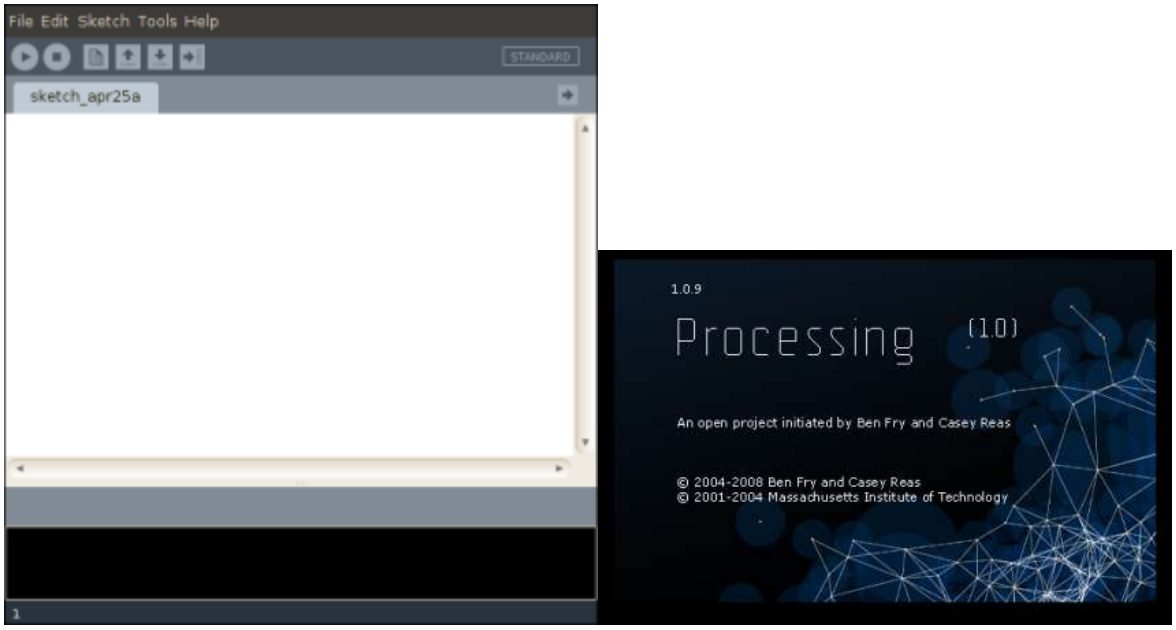


Ilustración 57: captura de pantalla del programa Processing.

3.2 Transmisión y muestra de datos en tiempo real.

Para las primeras pruebas se creó un entorno gráfico para visualizar en tiempo real el estado de los sensores, así como la velocidad de cada motor. Al no contar con telemetría inalámbrica en tiempo real, se dejó de utilizar porque tener el cable serial conectado al helicóptero no tendría mucho sentido para los futuros ensayos reales. En este caso se empezó a utilizar *Datalogs, que* nos permitiría un estudio posterior de cada ensayo, utilizando la memoria Flash de 16MB disponible.

El software para ordenador fue escrito con *Processig,* en la ilustración 58 se puede ver una captura de pantalla. Este programa permitía fácilmente, visualizar el estado de los sensores. Esto se hizo principalmente para las primeras pruebas para ver el sentido de los giróscopos y acelerómetros y su interacción con los motores. En los archivos anexos se puede encontrar el código, así como el ejecutable y un video de su funcionamiento.

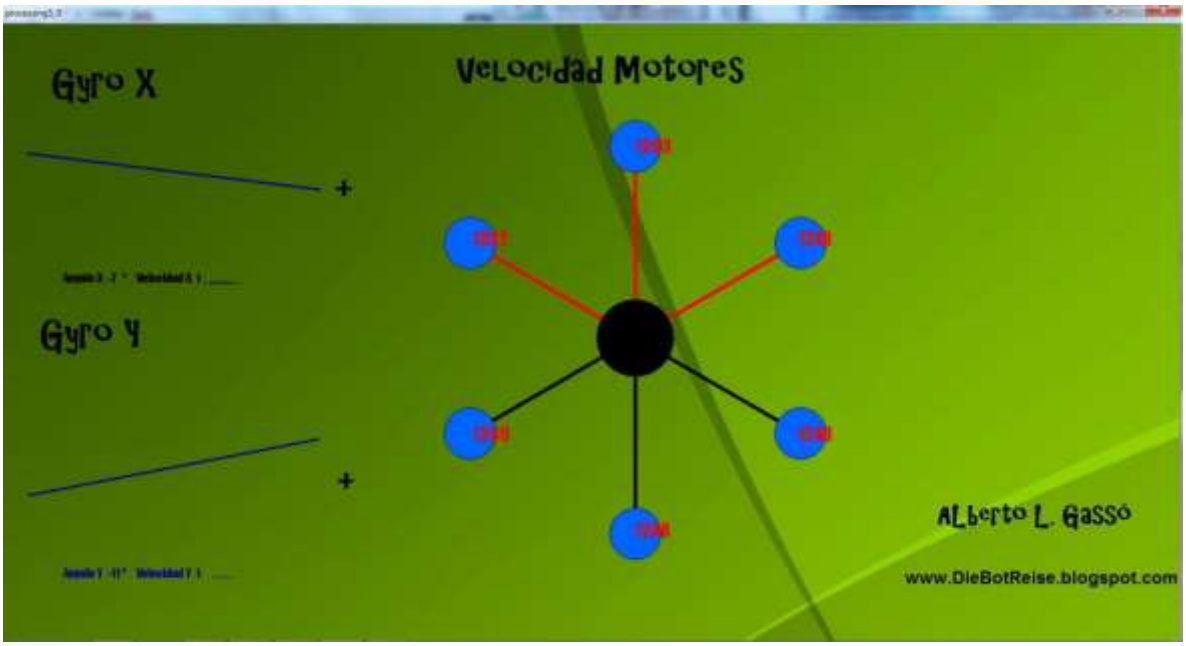


Ilustración 58: muestra de datos en tiempo real.

3.3 Drive Calculator 3.4

Drive Calculator es una herramienta que puede ser usada para el análisis de motores Brushless y es una ayuda para la selección del sistema completo de propulsión de un modelo radiocontrol. El programa puede ser usado para optimizar la combinación de hélices y motores.

El programa usa una base de datos que contiene tablas para motores, baterías, ESCs, reductoras y hélices. El programa, creado en Hannover por el alemán Christian Persson, permite añadir los resultados de tus pruebas y compartirlos a través del "online data Exchange".

3.4 Editor de presentaciones flash: Prezi

Prezi es una aplicación para crear y realizar presentaciones, que usa un solo lienzo en vez de diapositivas tradicionales y separadas. Los textos, imágenes, videos u otros objetos de presentación están colocados en un lienzo infinito y presentados ordenadamente. El lienzo permite a los usuarios crear una presentación no lineal, con conexiones entre diferentes presentaciones, y aplicar un zoom para los detalles del mapa visual.

3.5 Controladores.

3.5.1Reguladores PID para Pitch y Roll.

Un regulador o controlador PID (proporcional, integral y derivativo) es un mecanismo de control retroalimentado en bucle cerrado ampliamente usado en sistemas de control. Un PID calcula el error entre el valor actual del sistema y el valor al que se desea llegar. Además minimiza el error mediante el ajuste de las entradas del proceso.

En el cálculo del regulador PID intervienen tres parámetros distintos. Estos valores se pueden interpretar en función del tiempo. El proporcional P depende del error actual, el integral I depende de la suma de todos los errores pasados, y el derivativo D es la predicción de errores futuros, basándose en la tasa de cambio actual. La suma ponderada de los tres términos permite ajustar el proceso mediante un elemento de control como por ejemplo la velocidad que debe alcanzar un motor. Se adjunta la fórmula genérica de un regulador PID con su correspondiente flujograma en la ilustración 59.

$$Señal = K_p \cdot error + K_i \cdot \int_{t_0}^{t} error(\tau) \cdot d\tau \; + K_d \cdot \frac{d\; error(t)}{dt}$$

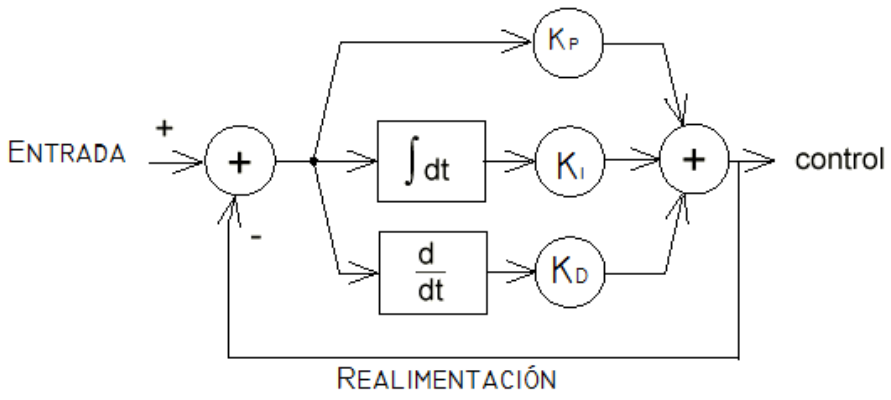


Ilustración 59: diagrama controlador PID.

En este proyecto el parámetro a regular es el ángulo de inclinación del eje Pitch y Roll, para ello realimentaremos con acelerómetros y giróscopos para calcular el error. Usaremos el acelerómetro para calcular el término proporcional multiplicando por la constante K_p y para el integral sumaremos los errores y multiplicará por K_i.

Para el término derivativo usaremos el giróscopo, que nos dará la aceleración angular que multiplicaremos por la constante K_d. Así no tendremos que calcular la derivación numérica que introduce algo de error, sobre todo usando métodos de bajo costo computacional.

$$Señal\ Motor\ (t_i) = K_p \cdot error(t_i) + K_d \cdot giro(t_i) + K_i \cdot \sum_{t_0}^{t_i} error(i)$$

Se han implementado dos controladores PID, uno para cada eje de giro, Pitch y Roll. Cada uno tendrá unos parámetros diferentes, aunque similares y el eje Roll ha necesitado menos componente derivativa.

3.5.2 Regulador PI para el Yaw.

Para el eje Yaw no se ha implementado la parte derivativa por simplicidad y porque con un proporcional-derivativo es más que suficiente. El control de este eje no es tan crítico como los otros dos, y su variación es lenta, por lo que el control es mucho más sencillo.

4 Prototipos de un solo eje.

4.1 Balacín "uno".

4.1.1 Construcción

Se ha construido un prototipo simplificado limitando el movimiento a un solo eje con el objetivo de estudiar el comportamiento que podría tener el prototipo final.

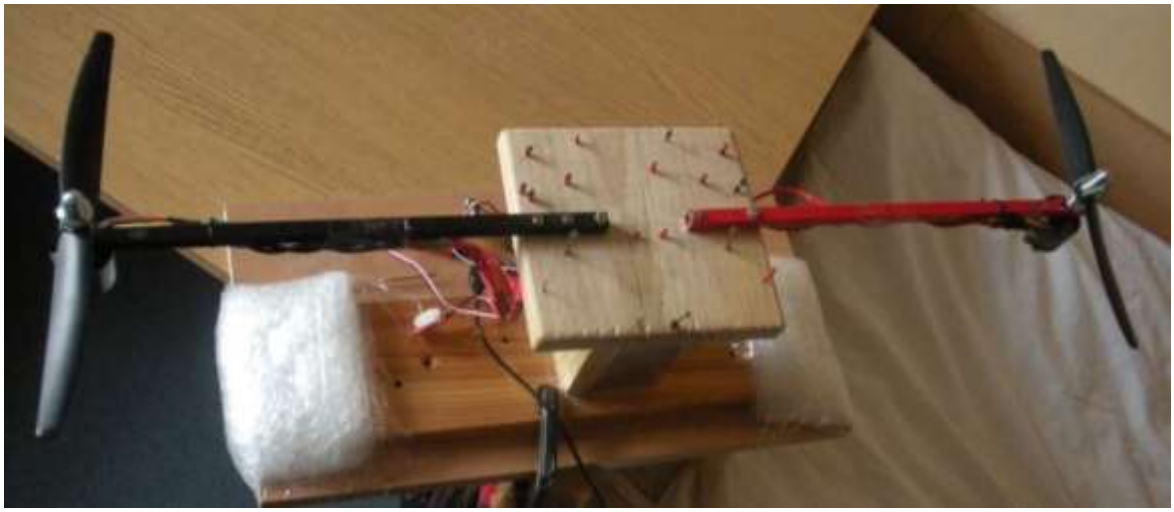


Ilustración 60: balancín uno.

Como se puede observar en la ilustración 60, el "balancín" está compuesto de una plataforma de madera donde se han fijado dos brazos, uno rojo y otro negro, con sus respectivos motores y hélices. En la plataforma de madera se han puesto una serie de fijaciones para que encajen los componentes y no se desplacen.

En las ilustraciones 61 a 64 se muestra los diferentes componentes que se han ido instalando en el *balancín:* dos ESCs, batería LiPo de 4000mAh y placas electrónicas.

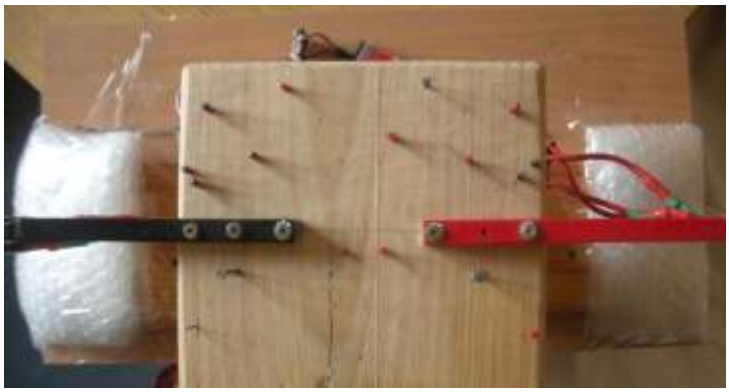


Ilustración 61

Ilustración 62

Ilustración 63

Ilustración 64

El resultado final, se puede observar en la ilustración 65, es un balancín que puede caer a ambos lados hasta un tope protegido con plástico de burbujas de aire para absorber posibles golpes. Los giros de las hélices son contrapuestos, aunque en nuestro modelo simplificado está limitado el giro sobre el eje YAW, se usa esta configuración por similitud al prototipo no simplificado.

El motor del brazo negro gira en el sentido de las agujas del reloj (CW) y el brazo rojo gira en sentido contrario (CCW) y están conectados a los canales 5 y 6 respectivamente (CH5 y CH6). Desde la placa de sensores (IMU) sale un cable USB para cargar los programas y transferir datos de los ensayos. Para medir los ángulos se usa el acelerómetro y para medir la velocidad angular el giróscopo.

Ilustración 65: resultado balancín uno

4.1.2 Ensayos

En este ensayo no se llegaron a conseguir resultados relevantes, debido a varios motivos, principalmente por motivos de seguridad no se pudo trabajar con altas velocidades en los motores, puesto que no teníamos los porta-hélices adecuados. Cuando llegaron las piezas, no se pudieron reanudar los ensayos porque ya se había construido el balancín número dos, y se destruyó el balancín uno.

En la gráfica 66 se muestran los resultados del ensayo 1.6 que se hizo con el balancín número uno.

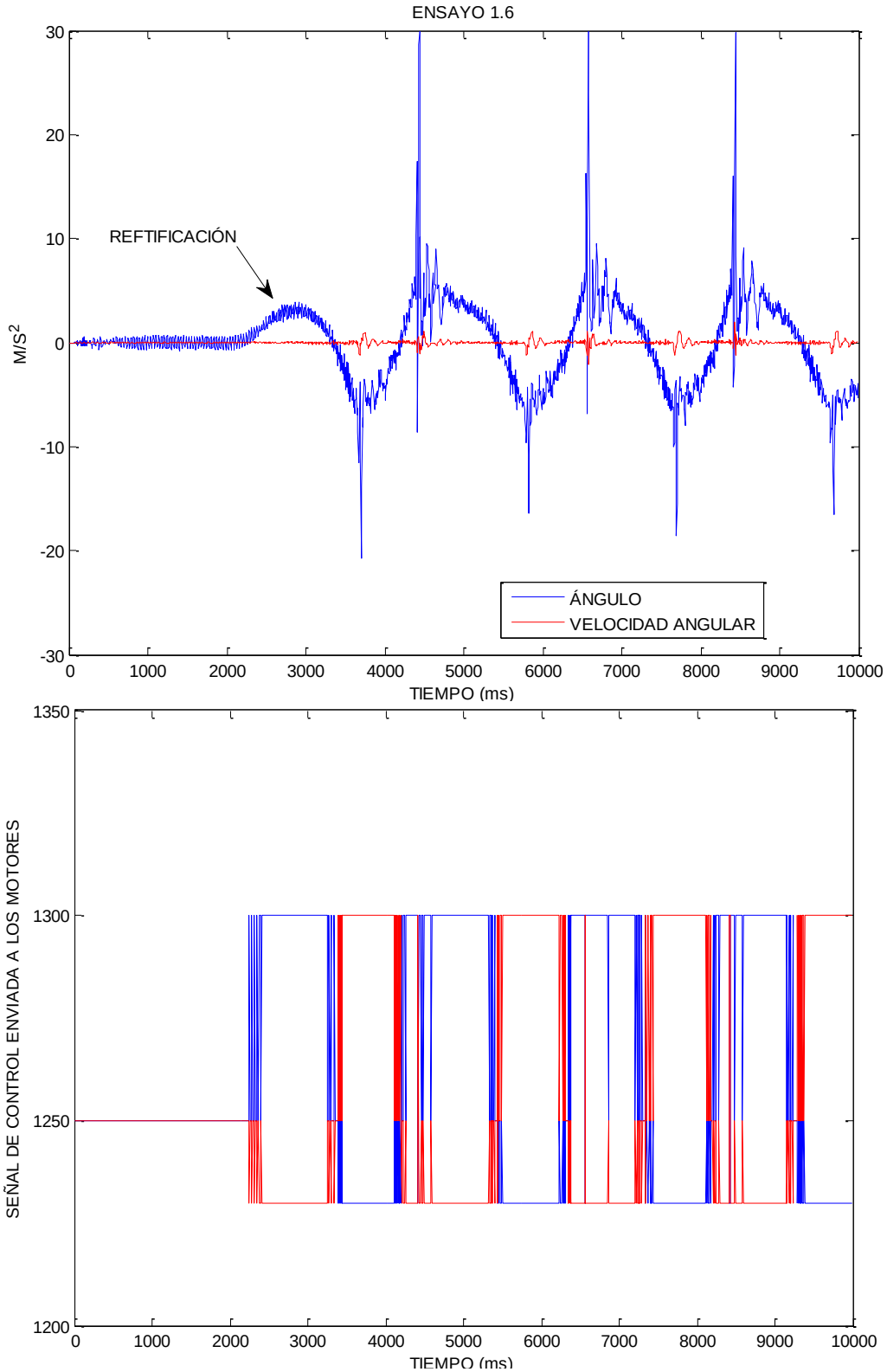


Ilustración 66: ensayo 1.6

En esta ilustración anterior la señal de los motores se satura. Al no poder trabajar en un ancho rango de velocidades, la señal saturaba mucho y el tiempo de reacción no era lo suficientemente pequeño para que estabilizara el balancín y los motores reaccionaban tarde. En los siguientes balancines se consiguió reparar este problema.

4.2 Balancín "dos".

4.2.1 Morfología

Se ha montado un banco de pruebas de manera que se limita el movimiento en un eje, facilitando las pruebas y ensayos. Se ha bloqueado el eje del pitch para tener un grado de libertad y así poder analizar como se comporta el prototipo. Se han sujetado los brazos 1 y 2 para dejar girar libremente el eje roll, de esta manera, y como se ve en las imágenes de 67 a 70, tenemos 4 motores para estabilizar el balancín.

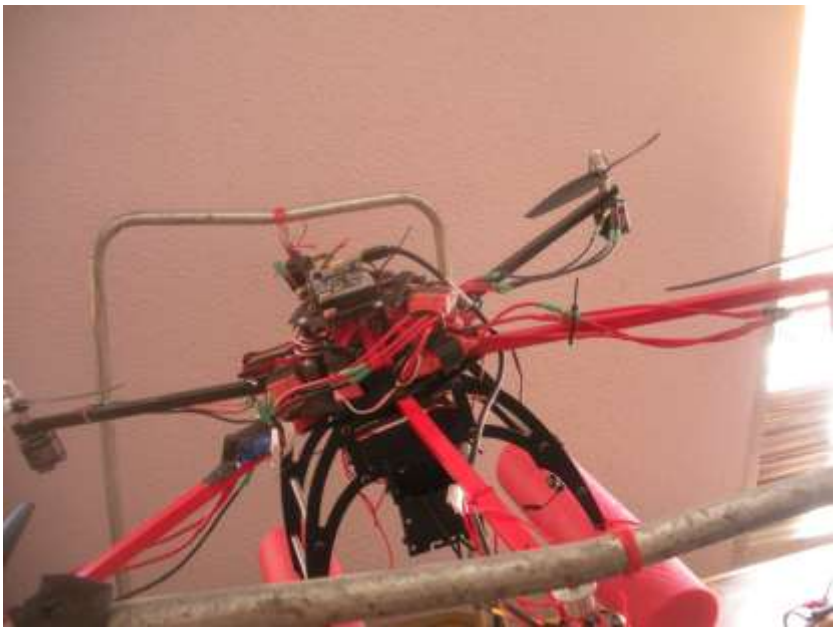


Ilustración 67: balancín dos.

Ilustración 68: balancín dos.

Ilustración 69: balancín dos.

Ilustración 70: balancín dos.

4.2.2 Ensayos

En los primeros ensayos se comprobó que existía mucho ruido debido a vibraciones, como se comprueba en la gráfica siguiente de dos ensayos en parado, uno con motores encendidos (aunque sin movimiento) y otro sin motores.

>> plot(d21(:,7),d21(:,2),'b');hold on; plot(d2(:,7),d2(:,2),'k');

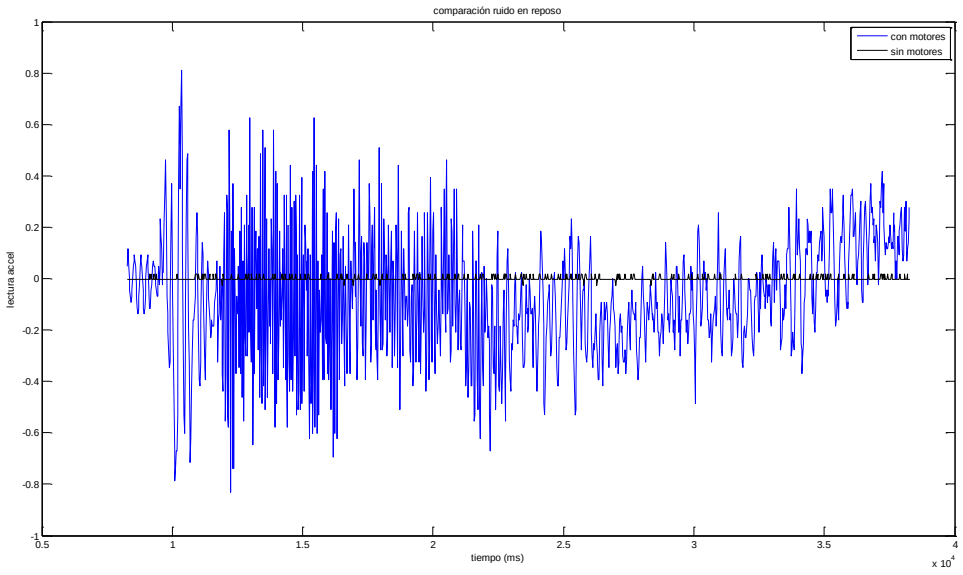


Ilustración 71

Comprobamos que existe demasiado ruido mecánico, por ello colocamos una esponjilla debajo de la placa de sensores para absorber, en la medida de lo posible el ruido mecánico y más adelante se aplicará un filtro.

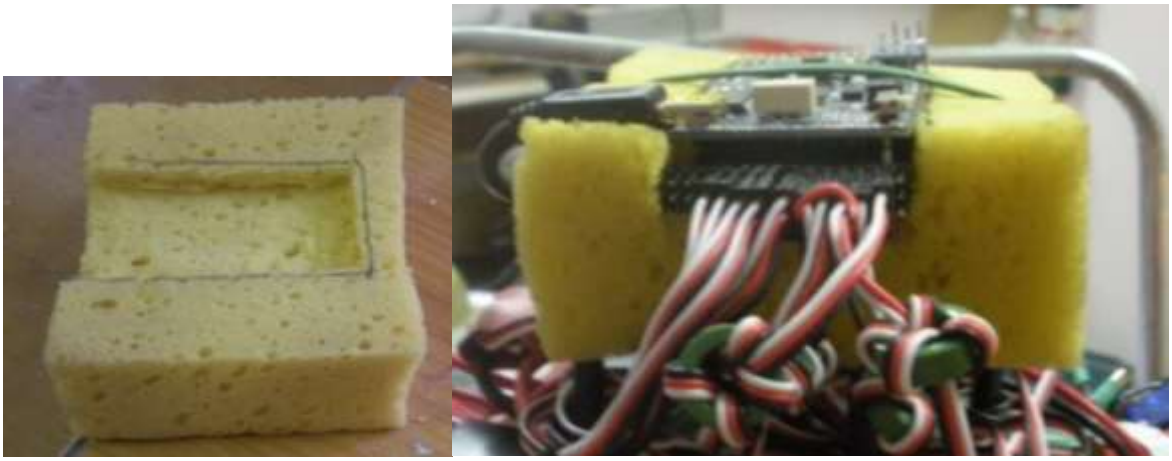


Ilustración 72: esponjillas utilizada como amortiguador.

En estos ensayos se ha usado una matriz de datos con la siguiente distribución en columnas:

Accel.x	Accel.y	Accel.z	Gyro.x	Gyro.y	Gyro.z	Tiempo (ms)	Señal de velocidad motor 1	Señal de velocidad motor 3	Señal de velocidad motor 4	Señal de velocidad motor 5
F	I	L	A		1		D	A	T	O
F	I	L	A		2		D	A	T	O

En el ensayo 2.15 se ha probado el comportamiento ante perturbaciones externas. Estas perturbaciones consistieron en cambiar de posición bruscamente el helicóptero. La señal de referencia es cero durante todo el ensayo. El tiempo de muestreo ha sido aproximadamente de entre 20 y 23 ms. Para el ensayo se ha utilizado un controlador PID cuyos valores han sido:

$$K_{proporcional} = 0.5; K_{derivativo} = 350; K_{integral} = 0.000005;$$

Se puede encontrar en el anexo el archivo de datos: “ensayo 2_15.txt”, el video “ensayo 2_15.avi” el archivo de código “ensayo 2_15.pde”.

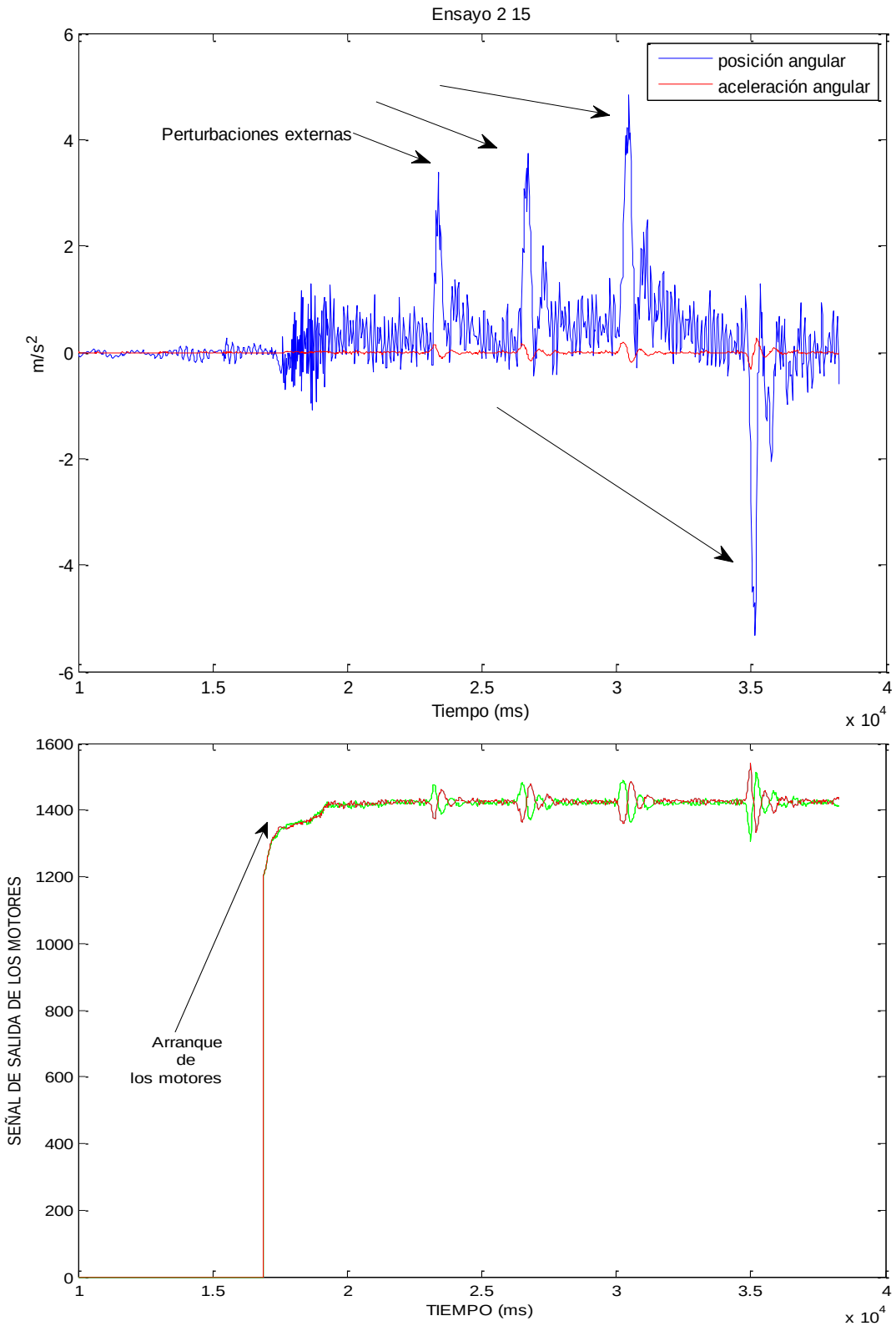


Ilustración 73: ensayo 2.15

En los siguientes ensayos se introducen señales de referencia con el joystick de la radio. Se puede ver en las gráficas 74, 75 y 76 correspondientes a los ensayo 2.47, 2.48 y 2.49. _

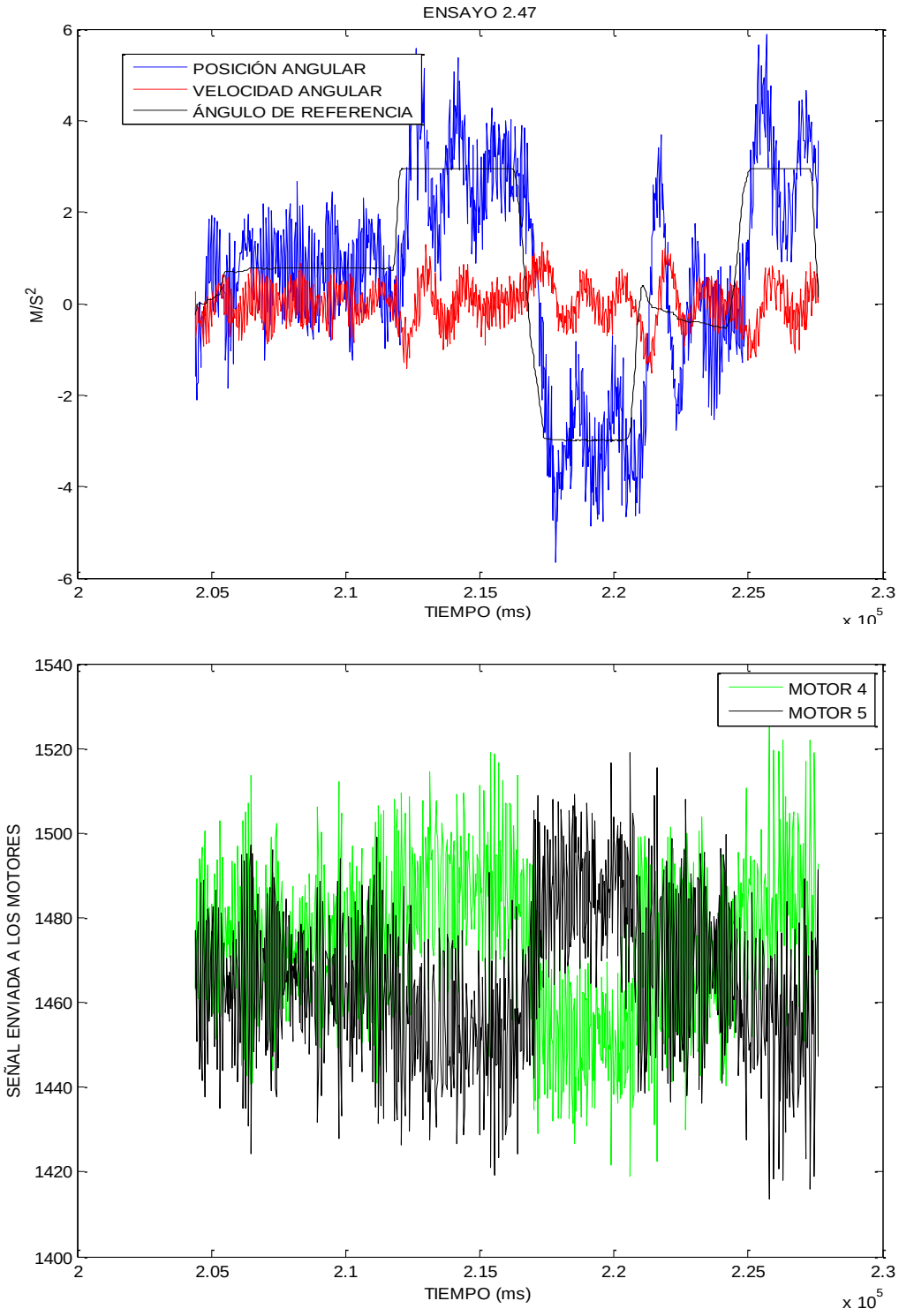


Ilustración 74: ensayo 2.47

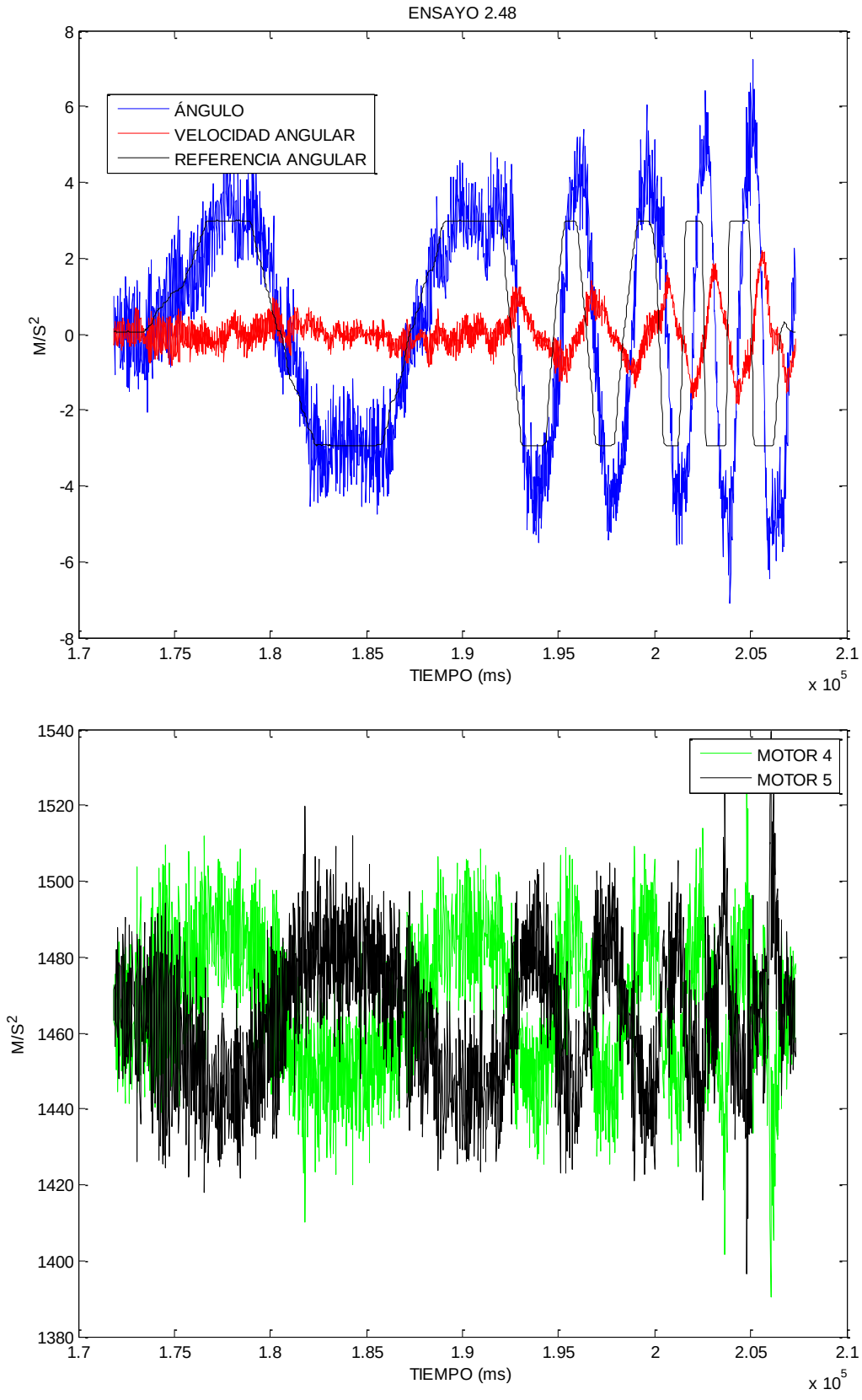


Ilustración 75: ensayo 2.48

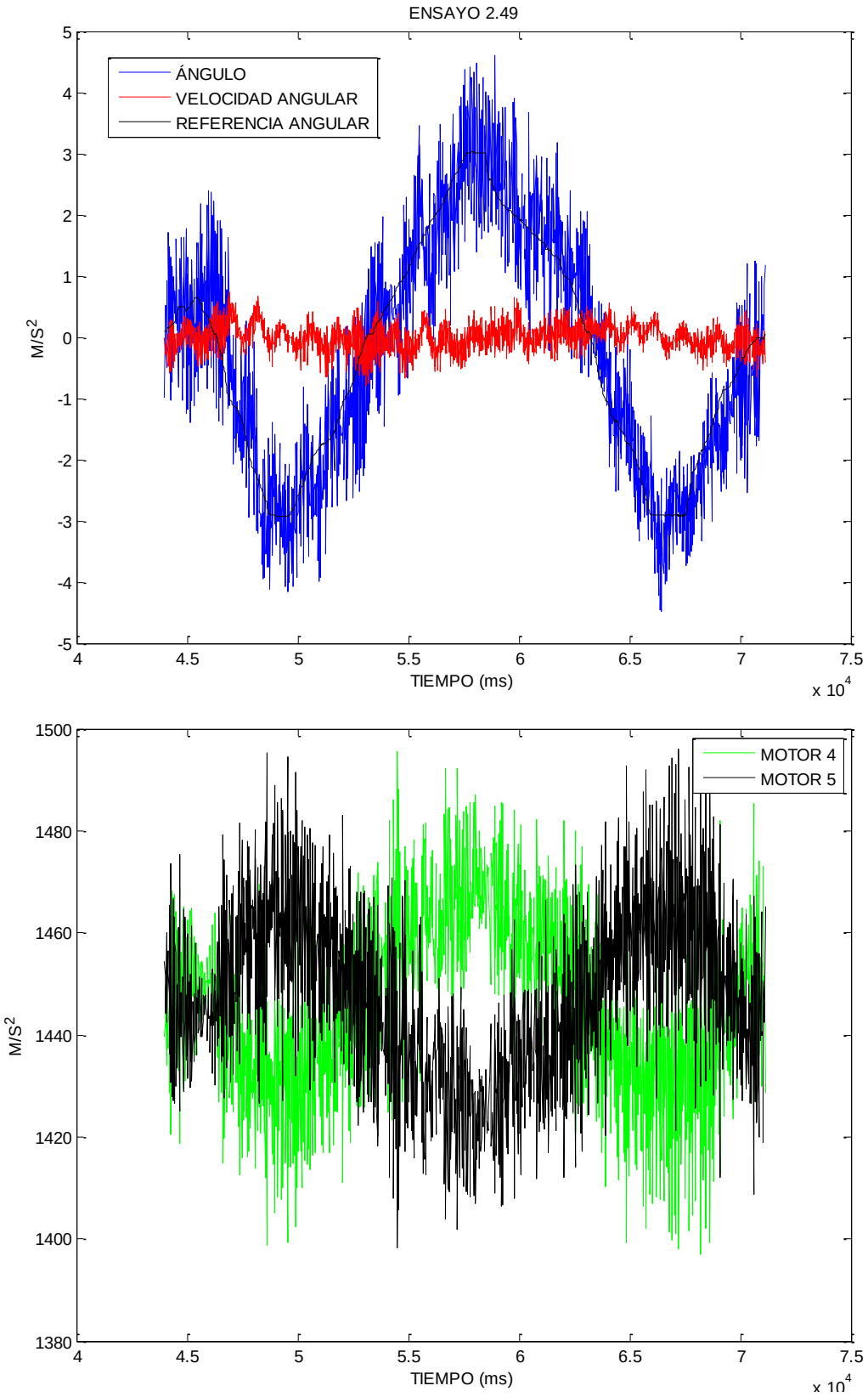


Ilustración 76: ensayo 2.49

5 Control del prototipo completo anclado a una plataforma móvil en los tres ejes, Pitch, Roll y Yaw.

Con la finalidad de testear el comportamiento respecto a ambos ejes, por separado y juntos, se ha utilizado una plataforma móvil. Con esta plataforma se van a poder realizar pruebas de vuelo parecidas a cómo se comportaría el multicóptero en el aire. Este anclaje fue diseñado en 2003 durante la realización de un proyecto fin de carrera[1] para simular el vuelo de un helicóptero RC tradicional. Ahora se ha adaptado para anclarla al hexacóptero.

Ilustración 77: estructura original.

En la imagen 77 se puede ver la estructura original, en la imagen 78 se muestra la plataforma de madera fabricada para sujetar el tren de aterrizaje del helicóptero.

[1] Robles Jiménez, José y Martínez Aparicio, Juan Clemente: DISEÑO, CONSTRUCCIÓN Y PUESTA EN MARCHA DE UN BANCO DE PRUEBAS DE UN HELICÓPTERO RC, PARA SU USO EN CONTROL AUTOMÁTICO.

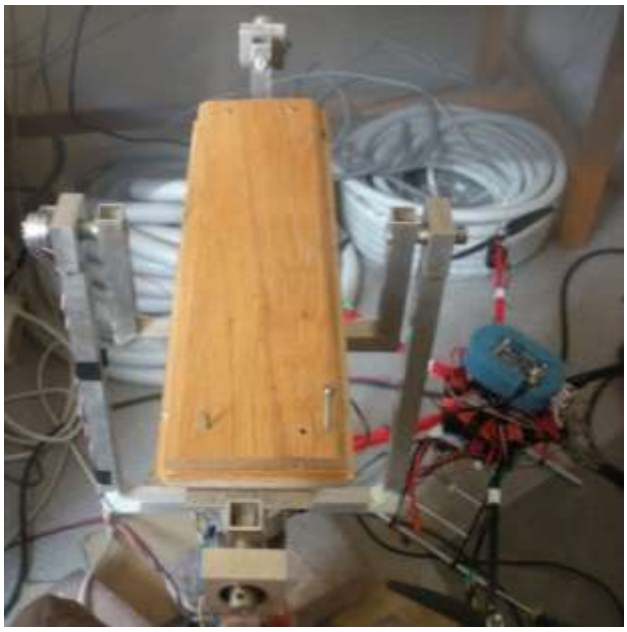


Ilustración 78: adaptador para el hexacóptero.

El resultado final se puede ver en la imagen 79.

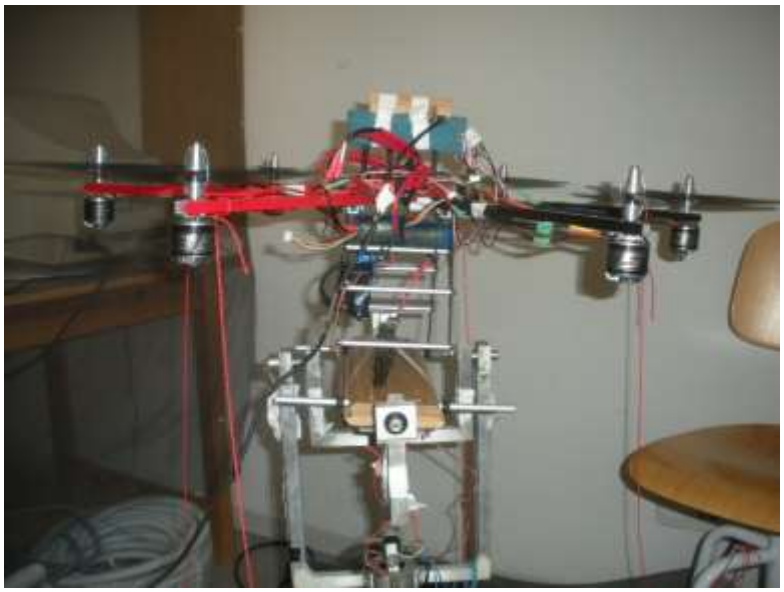


Ilustración 79: resultado.

5.1 Plataforma móvil con movimiento restringido a un eje: Pitch.

Se ha restringido mecánicamente el movimiento en el eje Roll, dejando libre el eje Pitch. El giro con respecto al Yaw queda libre, pues no se va a tratar hasta más adelante. El objetivo en esta fase es estabilizar el hexacóptero y controlar el ángulo de inclinación con respecto a la horizontal para ángulos pequeños. Para ello se ha usado un software parecido al usado en el banco de pruebas anterior, con algunas diferencias.

El ruido en la señal recibida por los sensores es provocado por las vibraciones de la estructura, y en menor medida por el ruido electromagnético generado por los motores. En el ensayo 4.4 de la gráfica 80 se pueden ver estas perturbaciones.

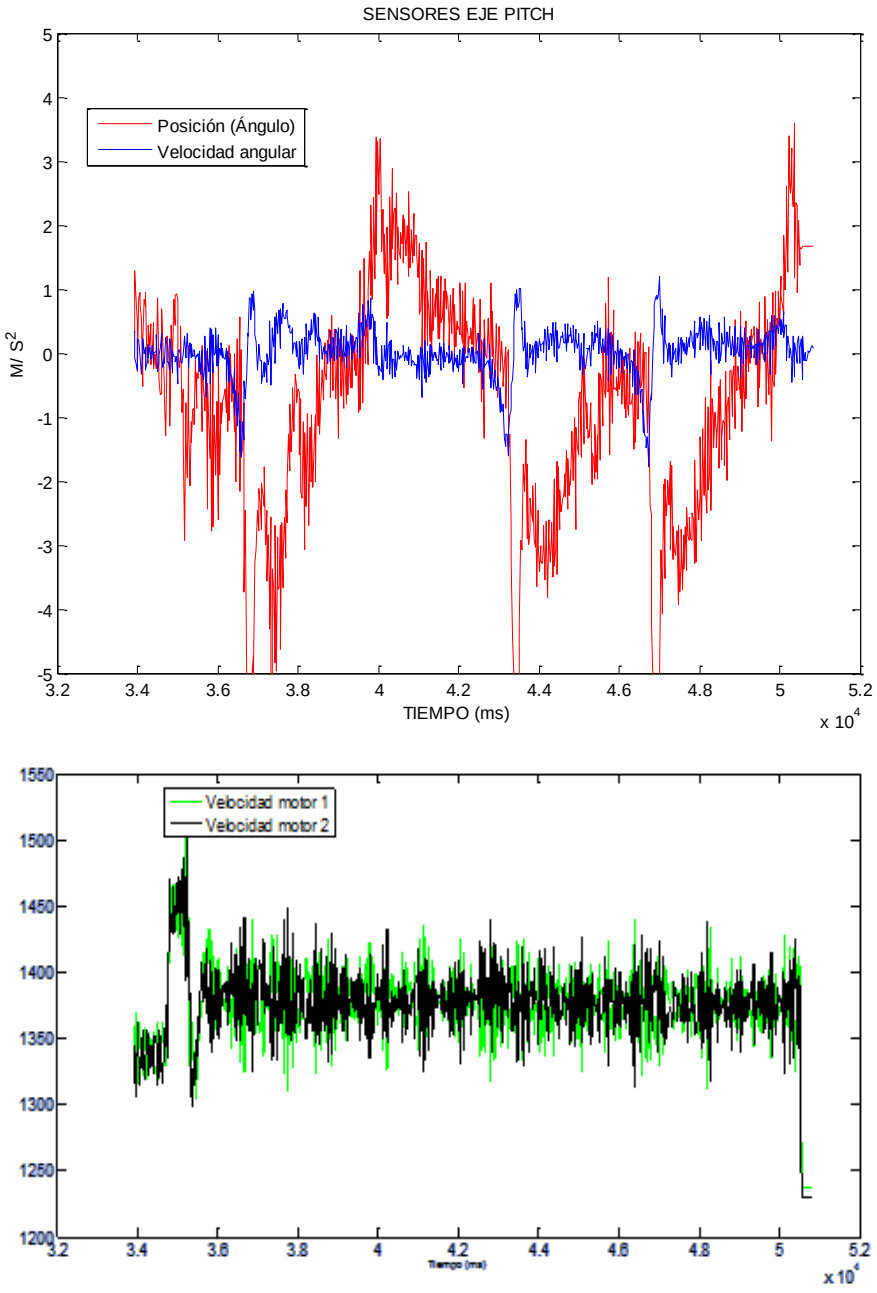


Ilustración 80: ensayo 4.4

Con el objetivo de reducir, en la medida de lo posible el error, se ha añadido un filtro. Para obtener el valor deseado de la constante de tiempo T, se ha usado simulink para simular su efecto.

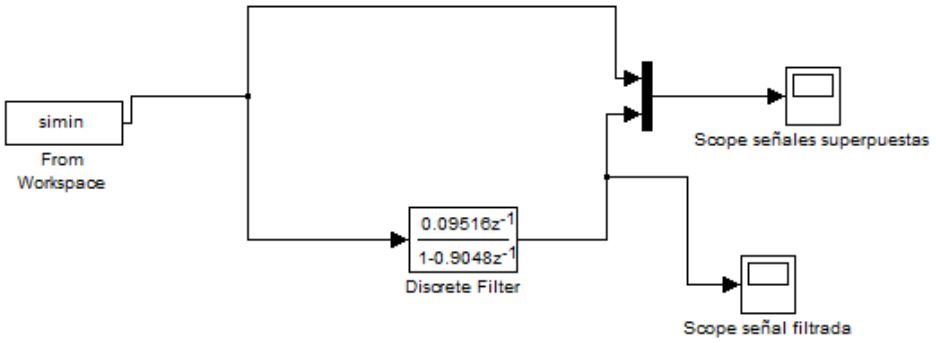


Ilustración 81: filtro con simulink, en matlab.

El filtro utilizado ha sido de primer orden, y su forma continua es:

$$G(s) = \frac{1}{T \cdot s + 1}$$

Con la ayuda de *Matlab* se ha transformado la función de transferencia "G" a tiempo discreto mediante el método "Zero Orden Hold".

```
>>G=tf([1], [0.2  1]);

>> c2d(G,0.02,'zoh');
```

El tiempo de muestreo utilizado en estos ensayos ha sido de aproximadamente 23 ms y una constante de tiempo T= 0,2.

Las líneas de código en C++ utilizadas para ejecutar este filtro han sido:

```
pitchFilter = lastPitchFilter*0.9048 + 0.09516*lastPitch;
```

El filtro añade un retraso de entre 3 y 4 muestras, lo que significan unos 80 ms, este tiempo es notablemente importante, ya que corregir el error a tiempo es crucial teniendo en cuenta la inestabilidad del sistema a tratar.

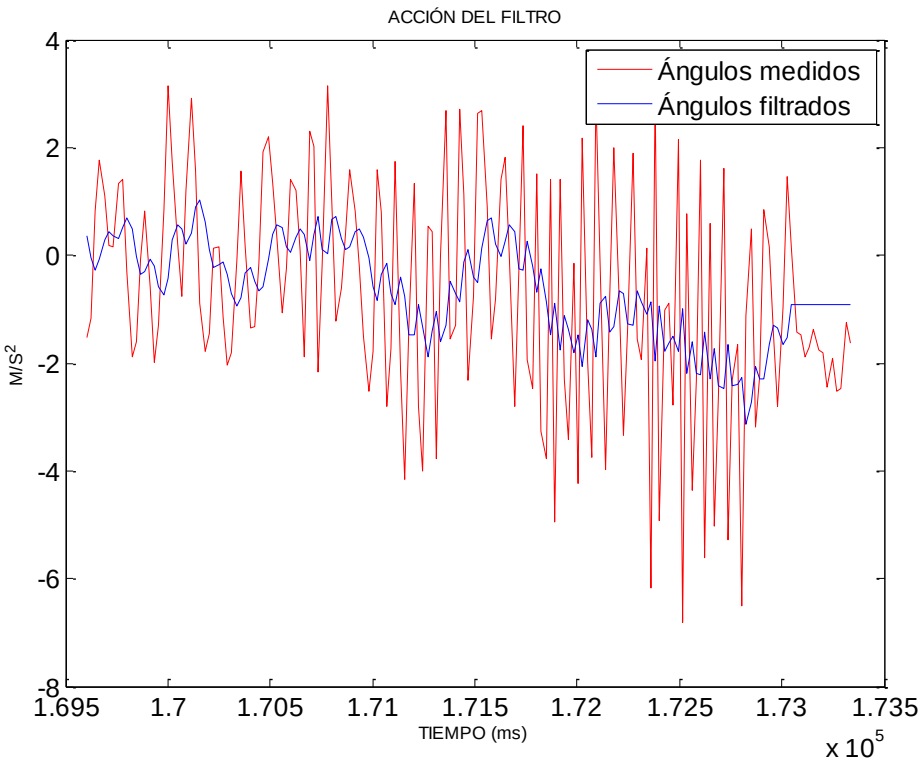


Ilustración 82: filtrado

Aplicando este filtro, véase gráfica 82, y sintonizando el controlador PID se ha llegado a una sintonía que consigue una estabilidad relativa en el cero pero con acelerones muy bruscos que hacían vibrar la estructura y los brazos del hexacóptero. Uno de los mejores resultados logrados se muestra a continuación en el ensayo 5.15, como se puede ver en las gráficas 83 y 84. El código usado es la versión <simulador 5_3.ino> que se puede encontrar en los archivos anexos.

La matriz de datos tiene la forma:

Roll	Pitch	Pitch filtrado	Giro Pitch	Giro Roll	Giro Pitch Filtrado	Tiempo (ms)	Señal de velocidad motor 1	Señal de velocidad motor 2	Señal de velocidad motor 3	Señal de velocidad motor 4	Señal de velocidad motor 5	Señal de velocidad motor 6	Pitch referencia
F	I	L	A		1		D	A	T	O	S		
F	I	L	A		2		D	A	T	O	S		

Para dibujar las gráficas con este tipo de matriz de datos (llamada “d2”) se han usado los siguientes comandos de Matlab para graficar ambas curvas.

```
>> plot(d2(:,7),d2(:,3),'b');hold on; plot(d2(:,7),d2(:,4),'r');

>> figure(1);hold on; plot(d2(:,7),d2(:,14),'LineWidth',2,'Color','k');

>> figure(2); plot(d2(:,7),d2(:,11),'g');hold on; plot(d2(:,7),d2(:,12),'k');
```

- Los parámetros del PID han sido: $K_p = 130; K_i = 0.01; K_d = 400;$ //Pitch
- La señal de referencia ha sido cero durante todo el ensayo.

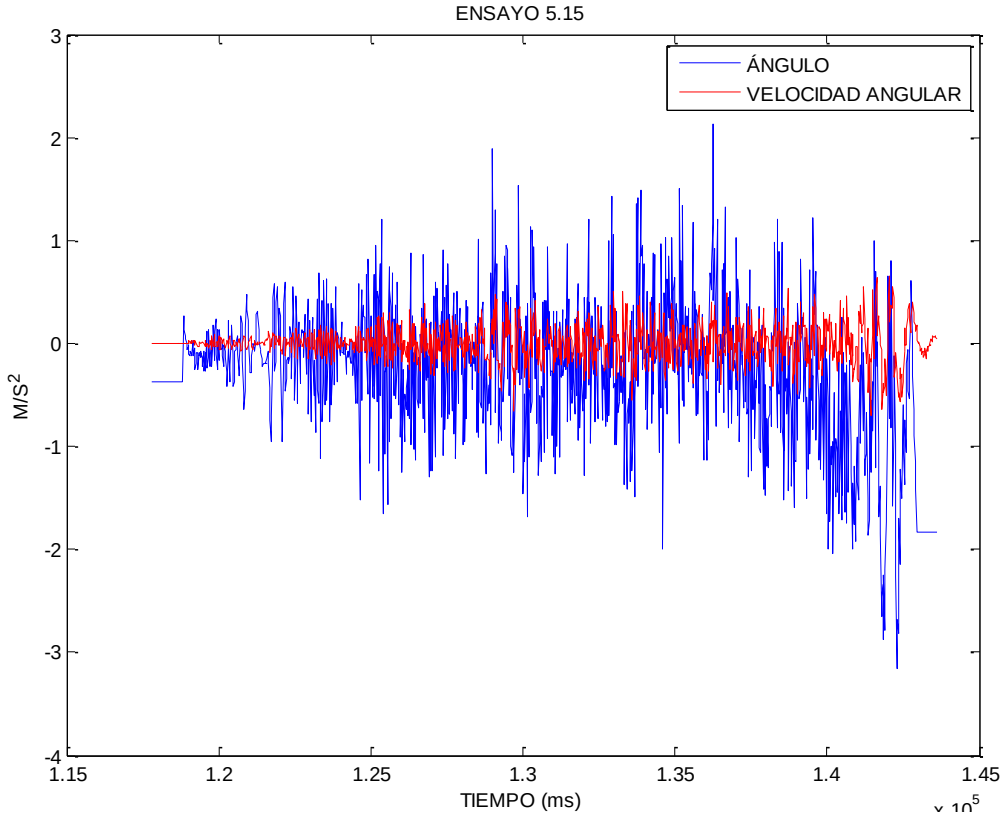


Ilustración 83: ensayo 5.15 (subplot 1)

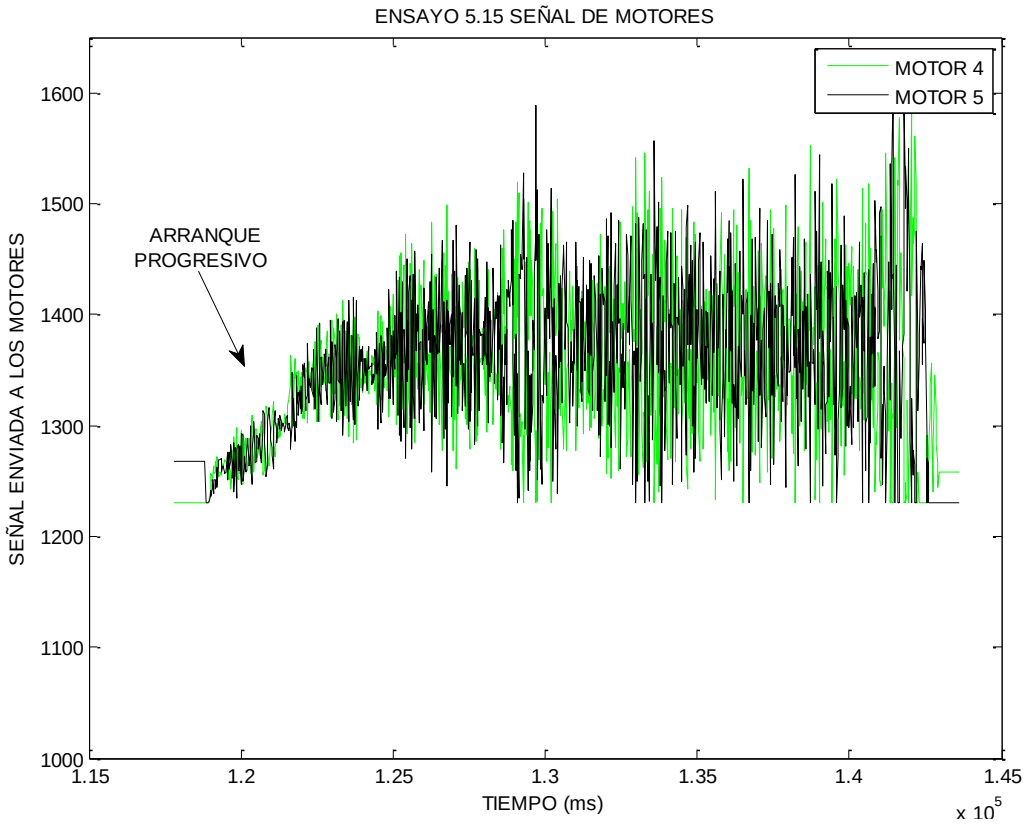


Ilustración 84: ensayo 5.15 (subplot 2)

Para solucionar el retraso en el filtrado y hacer el programa más eficiente, se trató de optimizar el software de Arduino para reducir el tiempo de muestreo.

Al optimizar el software se redujo el tiempo de muestreo desde 22,5 ms hasta 9,5 ms, menos de la mitad, lo que permitió utilizar un filtro más agresivo sin perder capacidad de reacción. El programa anterior tardaba unos 22,5 ms ya que esperaba a actualizar los valores de la radio en cada ciclo, y la radio actualizaba estos valores a 50Hz. Con el nuevo software, el ciclo se repite, independientemente de que hayan llegado valores actualizados o no desde la emisora.

Con estos parámetros se lograron resultados aceptables, y en el ensayo 6.41 que podemos observar en la gráfica 85, sigue una referencia impuesta por radio.

- Los parámetros del PID han sido: $K_p = 50; K_i = 0.075; K_d = 550;$ //Pitch
- La señal de referencia ha sido enviada mediante el joystick de la emisora de radio.

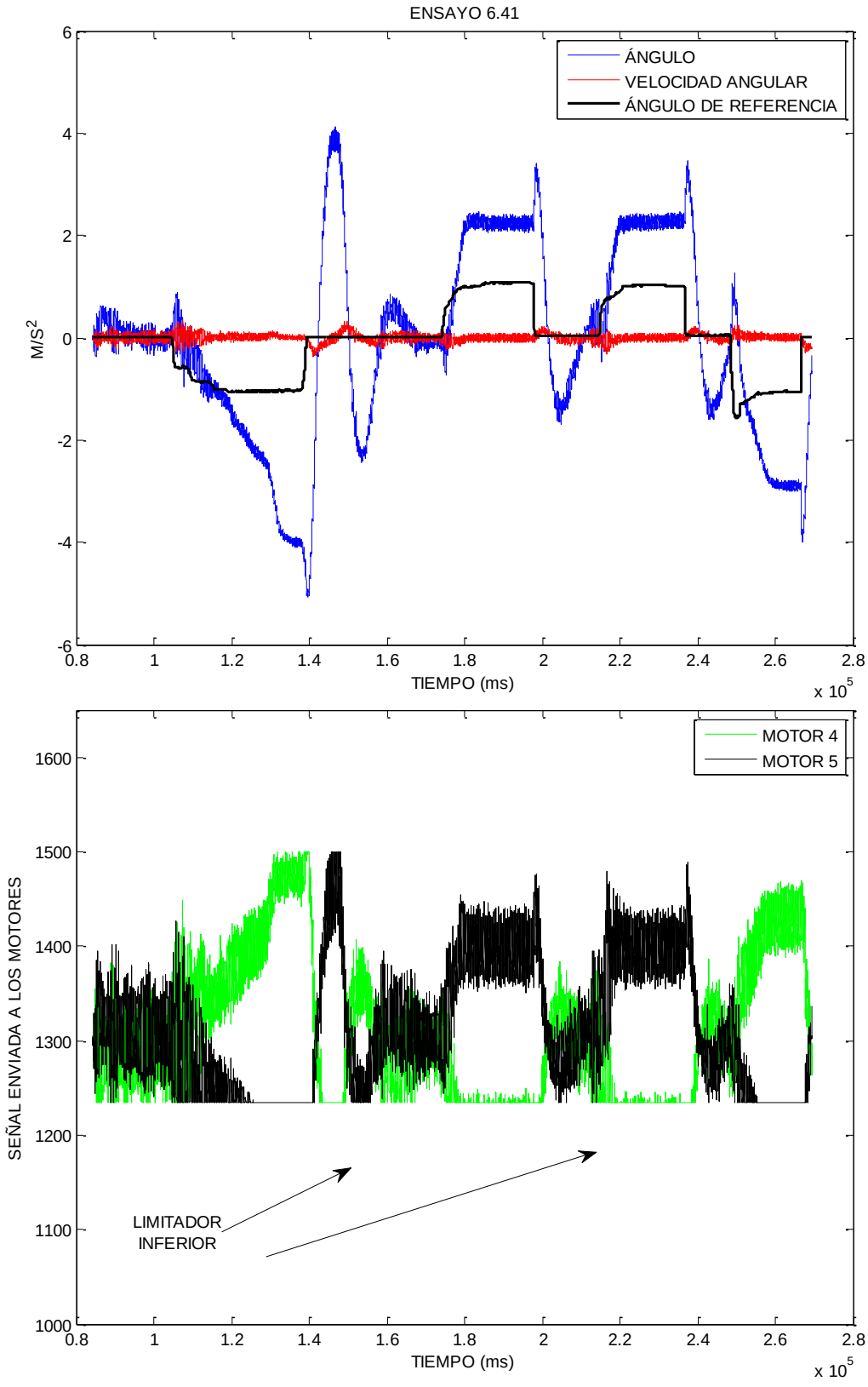


Ilustración 85: ensayo 6.41

En estos casos el limitador inferior funciona porque por debajo de ese umbral se pararían los motores, algo que no debe pasar nunca ya que podría desarmarse el motor y tardar varios segundos en volver a arrancar. También se observan oscilaciones excesivas.

Se puede ver en la gráfica anterior (gráfica 85) que existe bastante error de posición y que se necesita más precisión al seguir el ángulo de referencia. Más adelante, tras varios ensayos, se llegó a la conclusión que las velocidades media a la trabajaban los motores era demasiado alta, y provocaba que la estructura entrara en resonancia, añadiendo mucho ruido. También había problemas de fiabilidad con el sistema de anclaje de las hélices a altas velocidades, existiendo el peligro inminente de que se soltase una hélice y sus consecuencias.

Para solucionar esto se redujeron la velocidad media de los motores (throttle), que antes de esta modificación se mandaba por radio y era variable, y después pasó a ser constante, enviando por radio una señal escalón de tamaño 1280.

Reduciendo la velocidad de los motores cambió bruscamente el modo de comportarse al multicóptero: se eliminaron los acelerones bruscos, se redujo el ruido significativamente y la estructura-soporte dejó de vibrar.

Después de algunos ajustes en los parámetros del PID, se llegó al ensayo 6.53.

- Los parámetros del PID han sido: $K_p = 70; K_i = 0.1; K_d = 500;$ //Pitch
- El acelerador ha estado constante: Throttle = 1270;
- La señal de referencia ya no ha sido enviada mediante el joystick, y se programó unos escalones unitarios positivos y negativos.

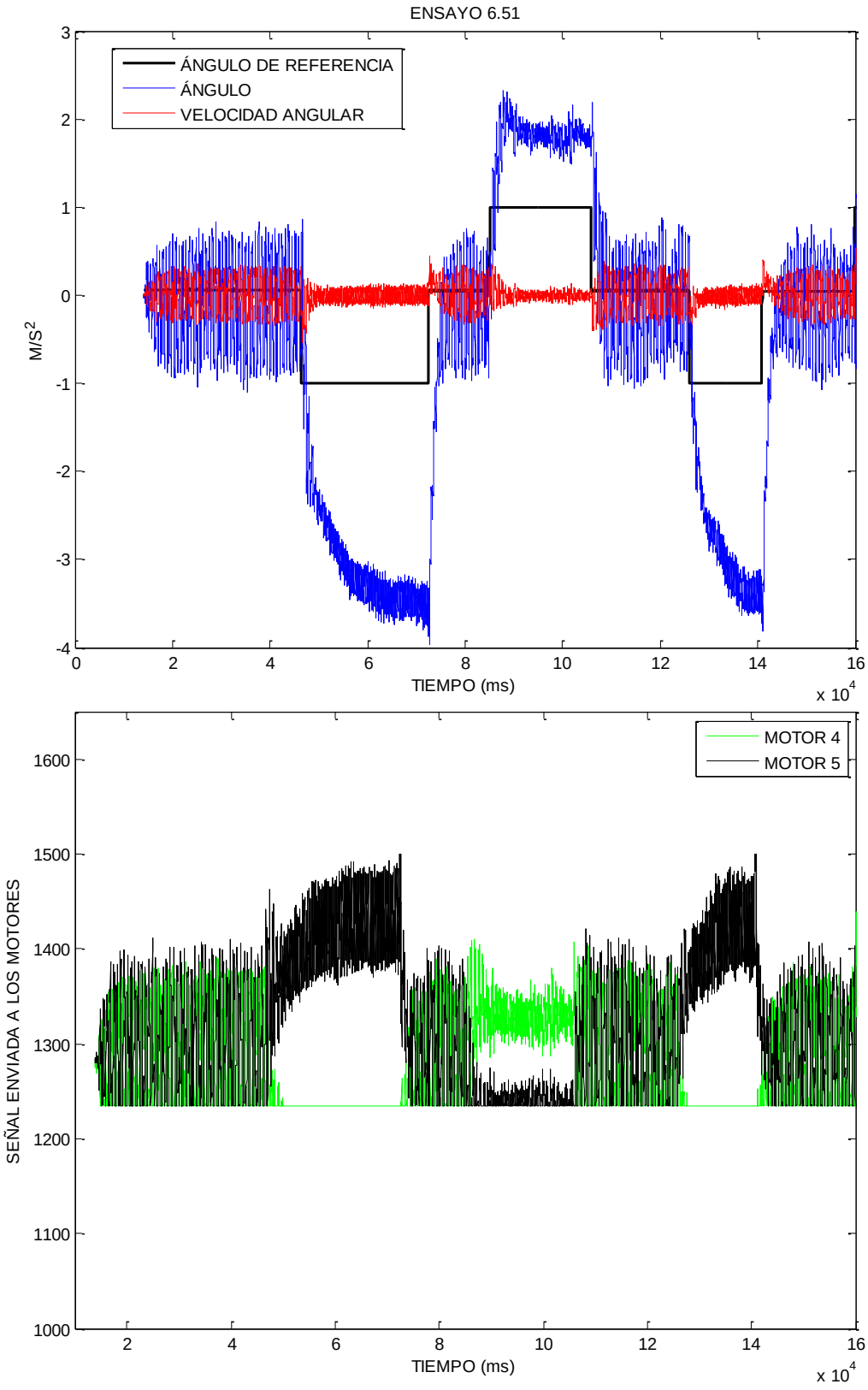


Ilustración 86: ensayo 6.41

Para comprobar el error estacionario, se cambió la señal de referencia a una escalonada variable, obteniendo en el ensayo 6.59:

- Los parámetros del PID han sido: $K_p = 80; K_i = 0.0; K_d = 550;$ //Pitch
- El acelerador ha estado constante: Throttle = 1270;
- La señal de referencia escalones de amplitud variable en ambos sentidos.

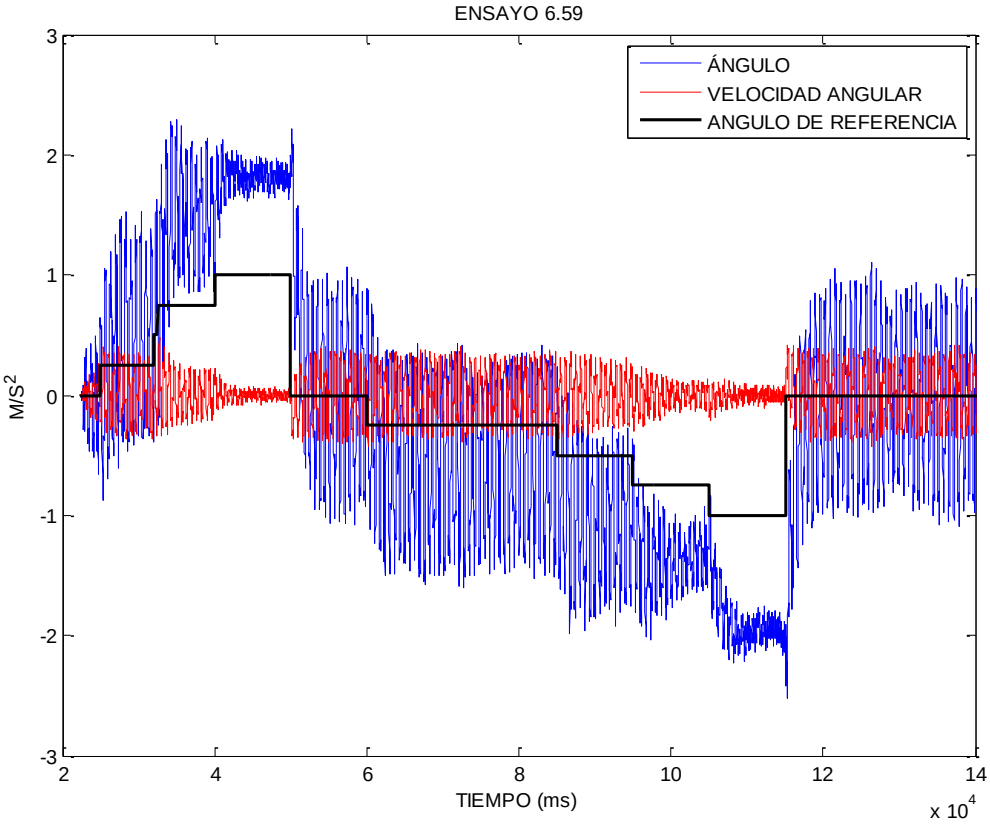


Ilustración 87: ensayo 6.59 (subplot 1)

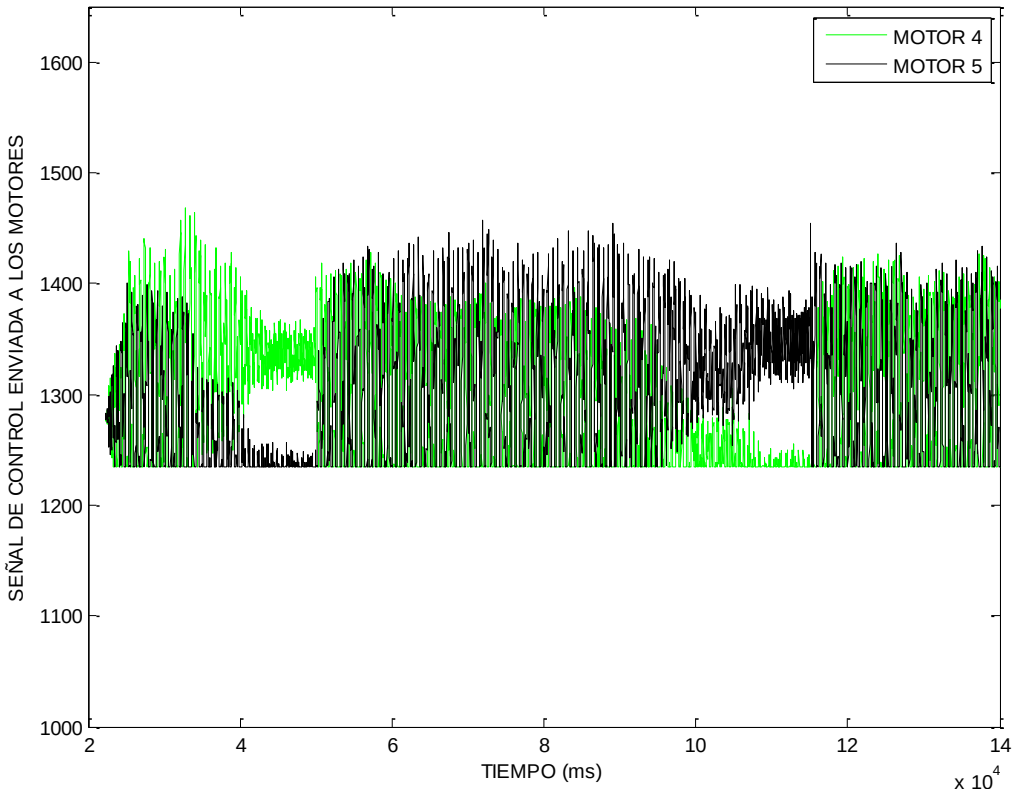


Ilustración 88: ensayo 6.59 (subplot 2)

En el ensayo que se puede ver en la gráfica 87, vemos que el modelo sigue la referencia pero con error estacionario, directamente relacionado con la parte integral, y muchas oscilaciones, relacionados con la parte derivativa y proporcional. Tras unos ensayos de pruebas y ajustes, se ha llegado al ensayo 6.72 (gráfica 89) obteniendo buenos resultados:

- Los parámetros del PID han sido: $K_p = 50; K_i = 0.1; K_d = 400;$ //Pitch
- El acelerador ha estado constante: Throttle = 1270;
- La señal de referencia escalones de amplitud variable en ambos sentidos.

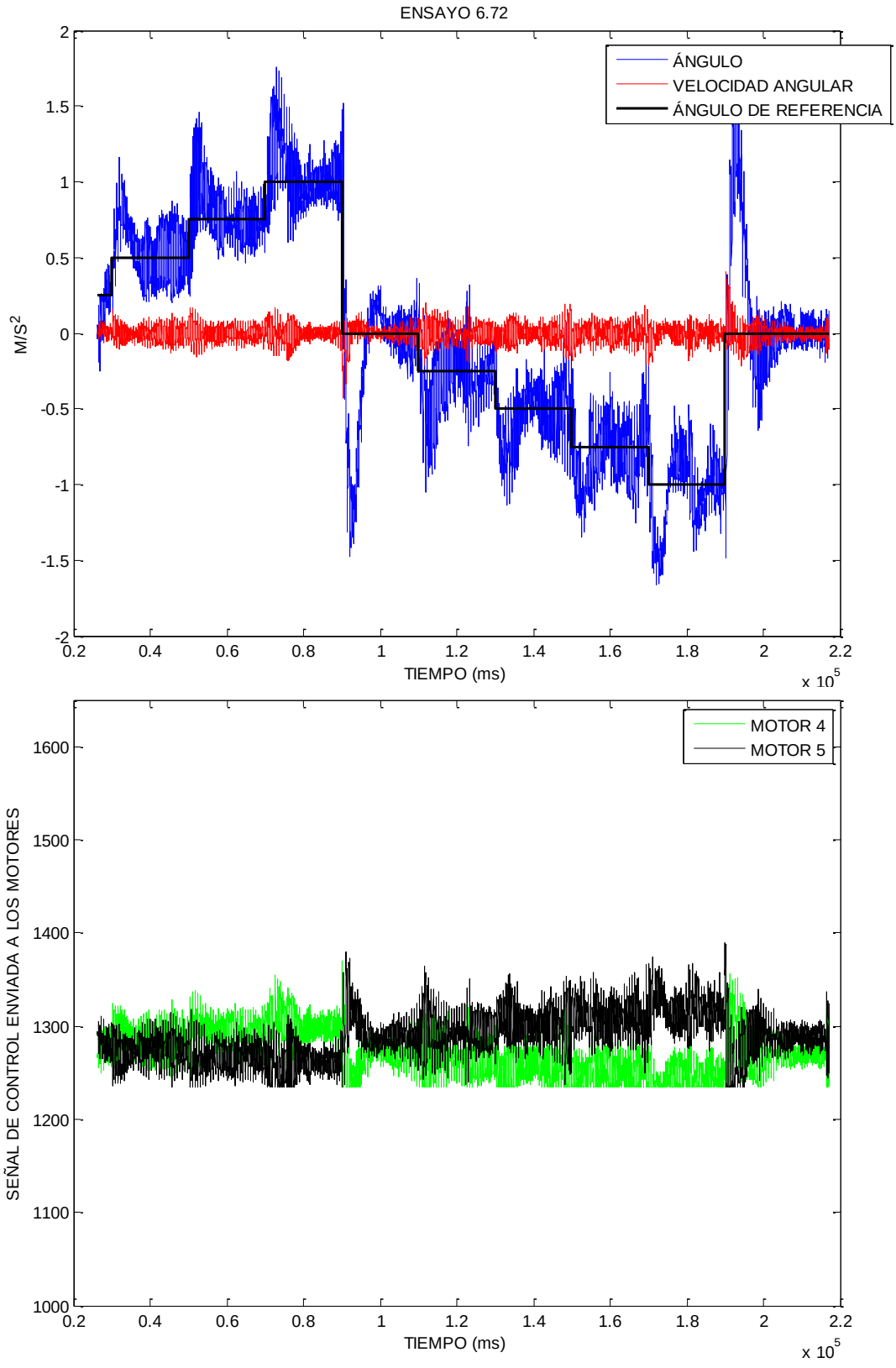


Ilustración 89: ensayo 6.72

El punto débil de esta configuración, véase en las gráficas 88 y 89, son las sobre oscilaciones, para ello se intentó aumentar el derivativo con el objetivo de limar los picos, sobretodo cuando la referencia da saltos bruscos. Los resultados de este ensayo no fueron los esperados, ya que se añadieron muchísimas vibraciones. Se pueden ver a continuación en el ensayo 6.69:

- Los parámetros del PID han sido: $K_p = 50; K_i = 0.1; K_d = 600;$ //Pitch
- El acelerador ha estado constante: Throttle = 1270;
- La señal de referencia escalones de amplitud variable en ambos sentidos.

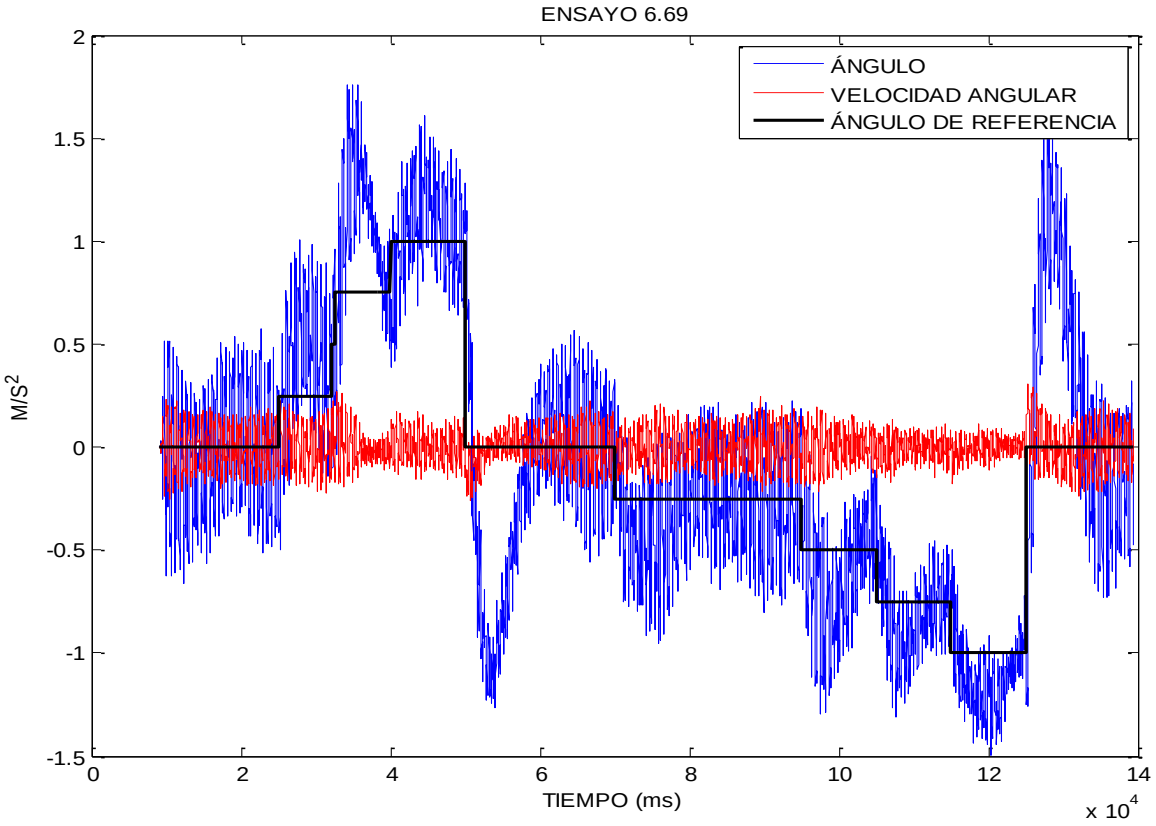


Ilustración 90: ensayo 6.69 (subplot 1)

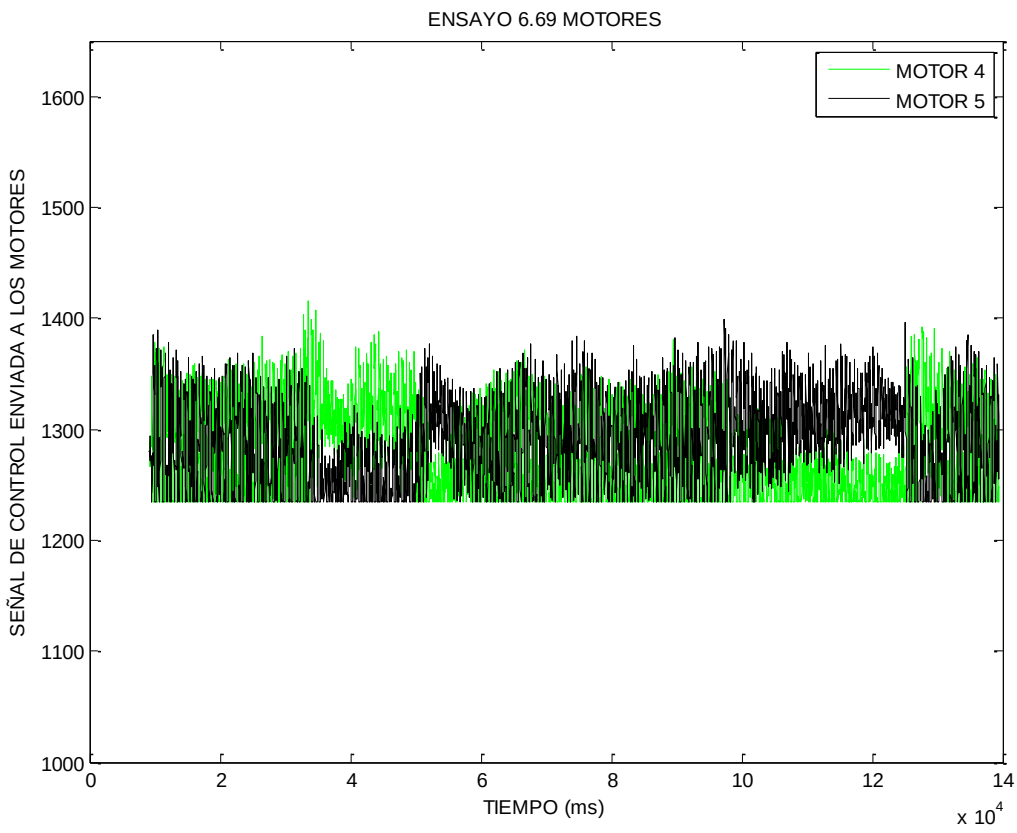


Ilustración 91: ensayo 6.69 (subplot 2)

Tras este ensayo, y puesto que daban mejores resultados, se eligieron como valores del PID los siguientes:

$K_p = 50; K_i = 0.1; K_d = 400;$ para el eje Pitch.

Posteriormente se experimentaron nuevas señales de referencia con diferentes entradas, incluyendo rampas, para estudiar y confirmar su comportamiento. El código usado en los siguientes ensayos ha sido la versión < simulador5_71.ino>.

Rampa de baja pendiente: ensayo 7.79, gráfica 92.

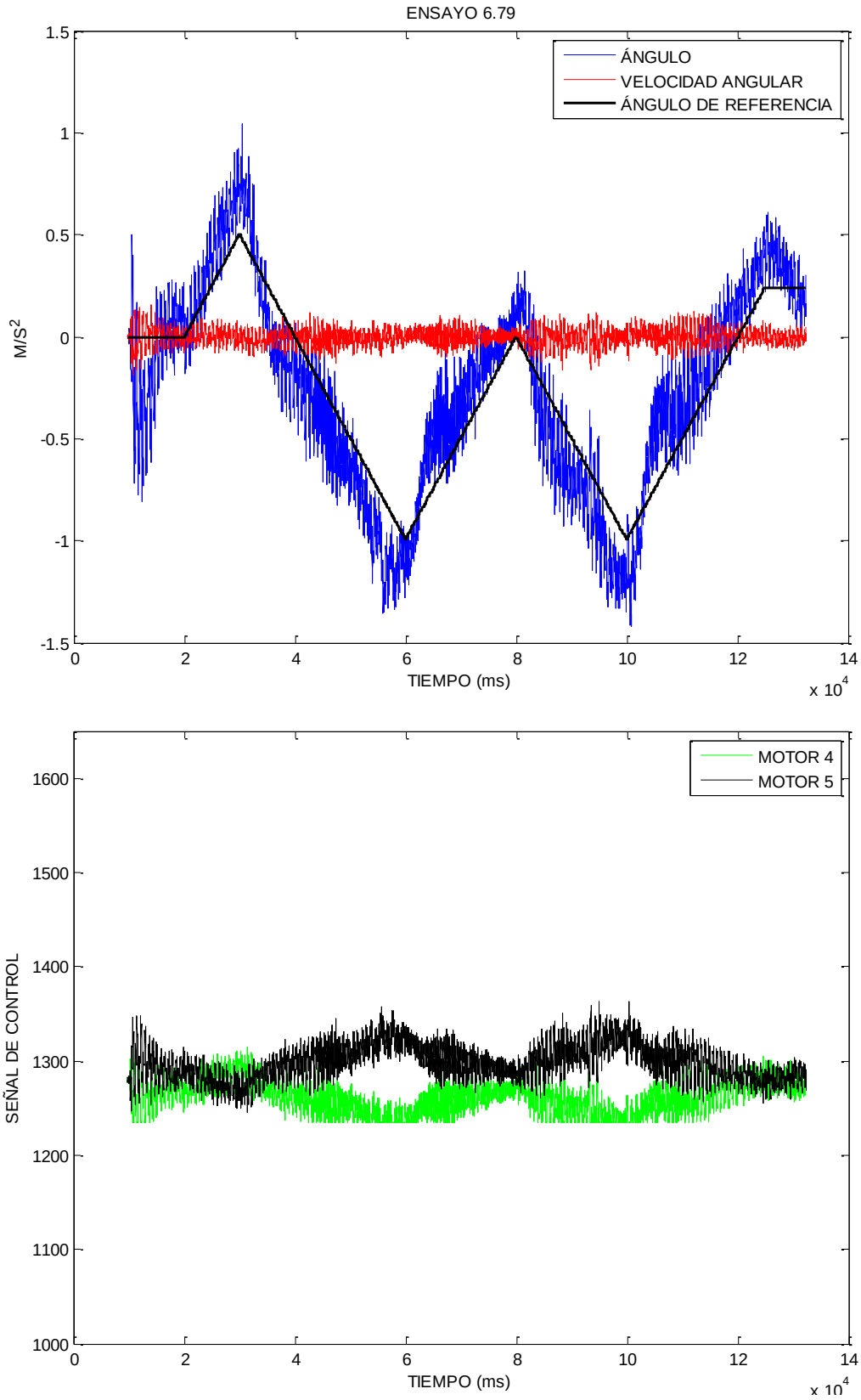


Ilustración 92: ensayo 6.79

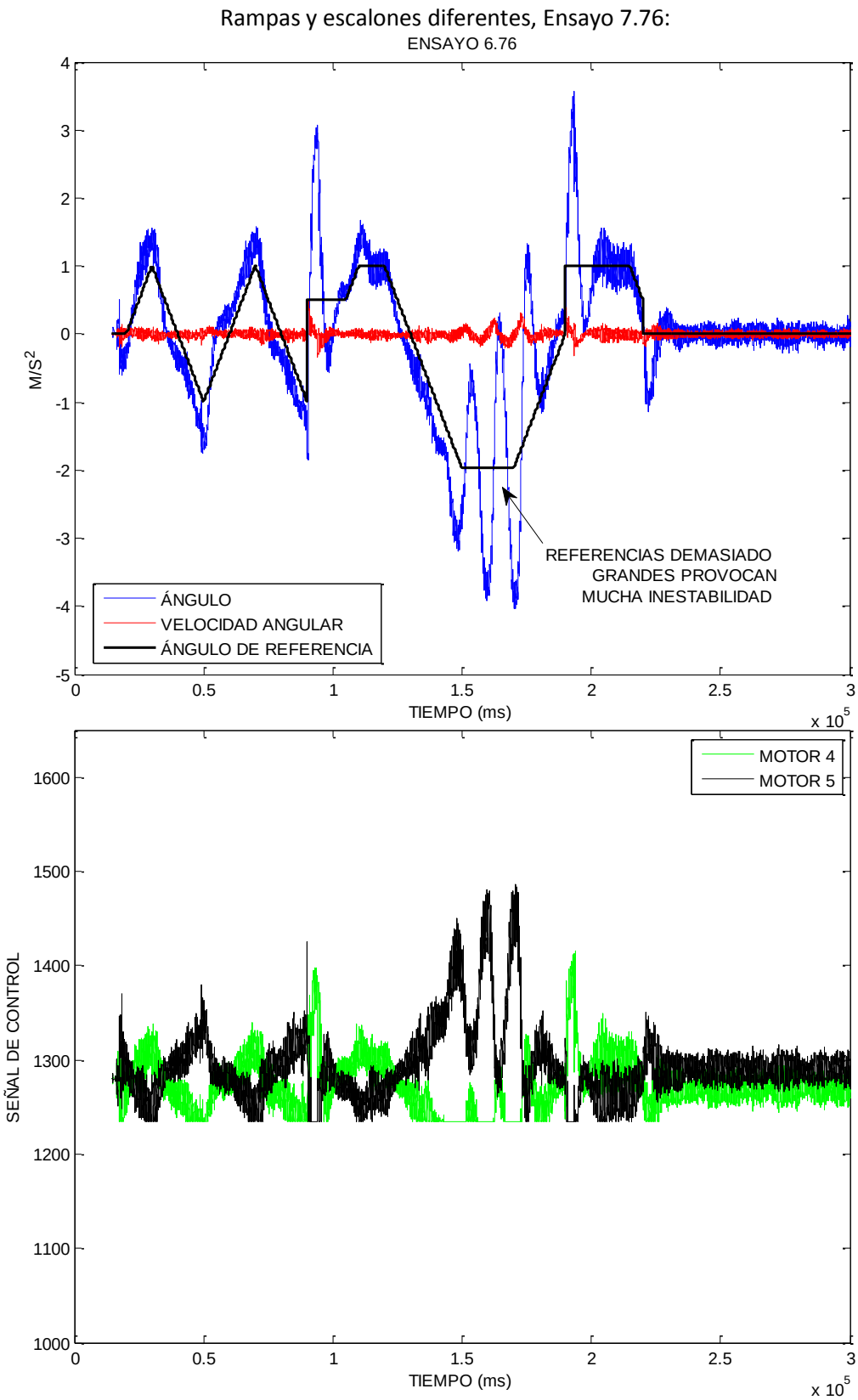


Ilustración 93: ensayo 6.76

Se puede ver en la gráfica 93, que referencias con valor superior a 1 (M/S^2) hacen al sistema oscilar mucho y por encima de 2 (M/S^2) se vuelve inestable.

Para comprobar que no se perdiera la estabilidad en ningún momento a tiempos largos, se realizó un ensayo de duración 1200 segundos en el ensayo 6.71: (gráficas 93 y 94). Los parámetros del PID han sido:

$$K_p = 50; K_i = 0.1; K_d = 400; \quad \text{//Pitch}$$

El acelerador ha estado constante: Throttle = 1270;

La señal de referencia escalones de amplitud variable en ambos sentidos los primeros 140 segundos y después referencia cero hasta 1200 segundos.

Que el ensayo 6.71 durase unos 20 minutos, fue debido a que se usaron bajas velocidades en los motores. A velocidades altas el tiempo máximo de vuelo, se reduciría a unos 10 minutos.

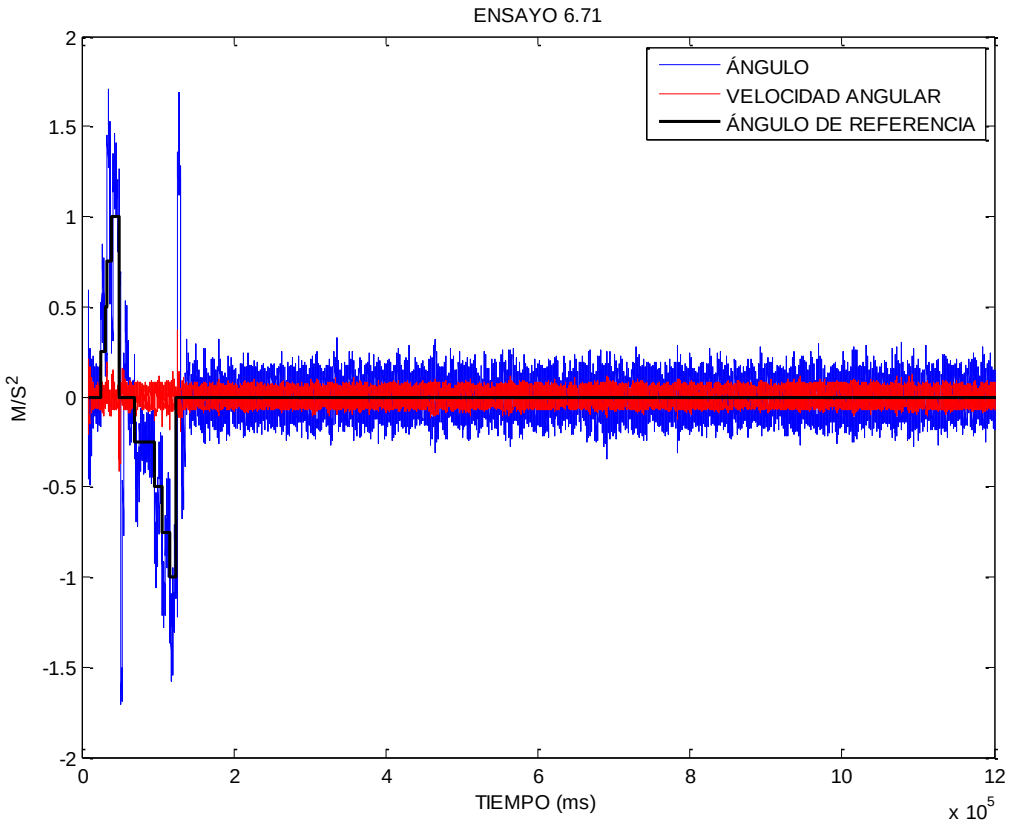


Ilustración 94: ensayo 7.1 (subplot 1)

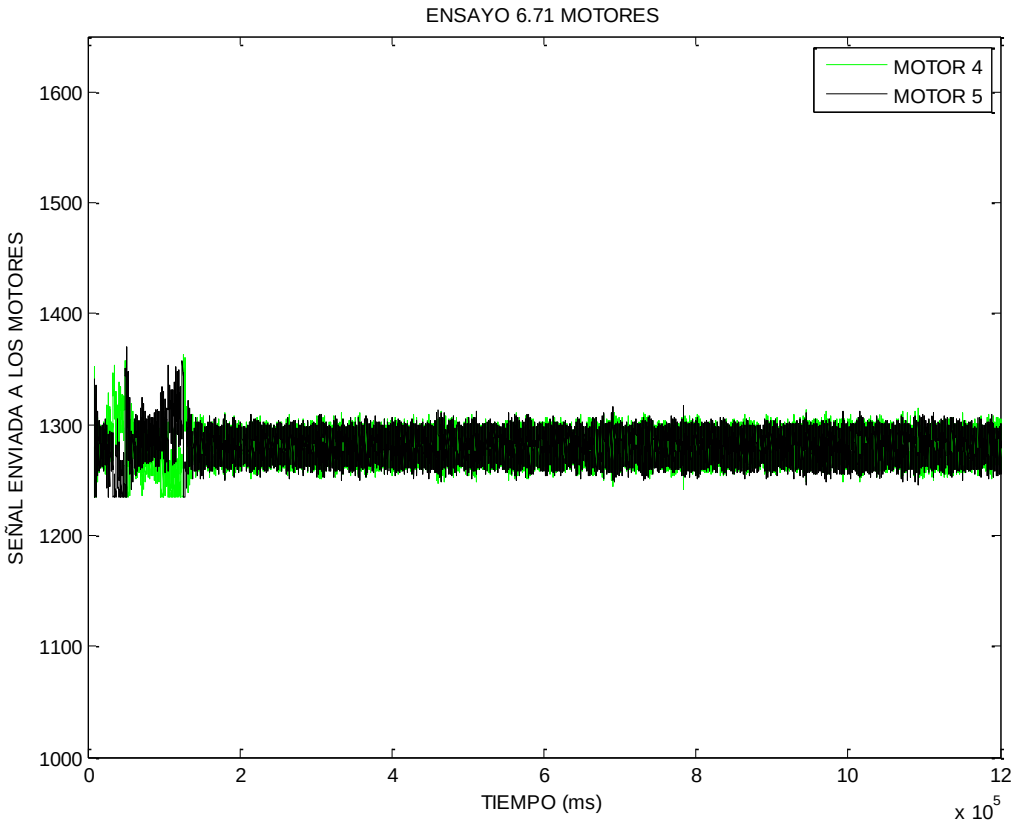


Ilustración 95: ensayo 6.71 (subplot 2)

5.2 Plataforma móvil con movimiento restringido a un eje: Roll.

Para llevar a cabo estos ensayos se han modificados los topes mecánicos colocados en la estructura, restringiendo ahora el eje Pitch para dejar libre el eje Roll. Se han seguido los mismos pasos que en el eje anterior, y para las primeras pruebas se han usado los mismos parámetros que en el otro eje, obteniendo un equilibrio, pero se pueden obtener mejores resultados afinando mejor el PID. Al igual que en eje Pitch se ha usado un eje para filtrar la señal de la posición que nos llega desde el acelerómetro.

El código utilizado ha sido la versión <simulador 7_0.ino>.

La matriz de datos en los siguientes ensayos tiene la forma:

Roll	Pitch	Roll filtrado	Giro Pitch	Giro Roll	Giro Roll Filtrado	Tiempo (ms)	Señal de velocidad motor 1	Señal de velocidad motor 2	Señal de velocidad motor 3	Señal de velocidad motor 4	Señal de velocidad motor 5	Señal de velocidad motor 6	Roll referencia
F	I	L	A		1		D	A	T	O	S		
F	I	L	A		2		D	A	T	O	S		

Los comandos de matlab para graficar con este nuevo modelo de matriz de datos:

```
>> plot(d2(:,7),d2(:,3),'b');hold on; plot(d2(:,7),d2(:,5),'r');

>> figure(1);hold on; plot(d2(:,7),d2(:,14),'LineWidth',2,'Color','k');

>> figure(2); plot(d2(:,7),d2(:,8),'g');hold on; plot(d2(:,7),d2(:,9),'k');
```

Tras unas cuantas pruebas de ajuste se comprueba que los valores del PID que mejor funcionan son aproximadamente $K_p = 60; K_i = 0.1; K_d = 350;$ //Roll, se puede observar en la gráfica 96 los resultados del ensayo 7.16 en el que la entrada son escalones de amplitud diferente.

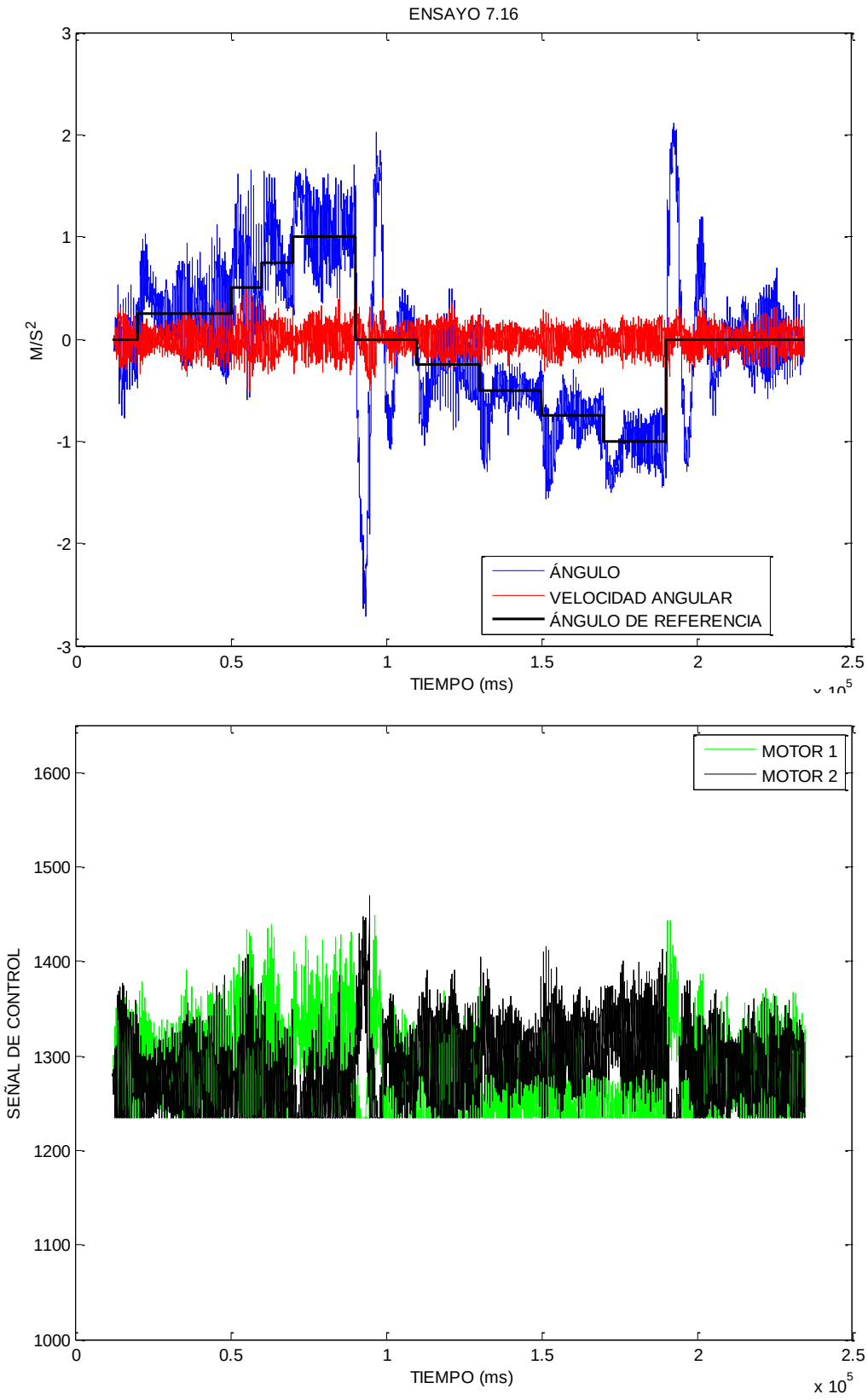


Ilustración 96: ensayo 7.16

Al igual que en el eje anterior se han realizado algunos ensayos con señales de entrada de rampas para observar su comportamiento. En el ensayo 7.22 se ha tomado una rampa:

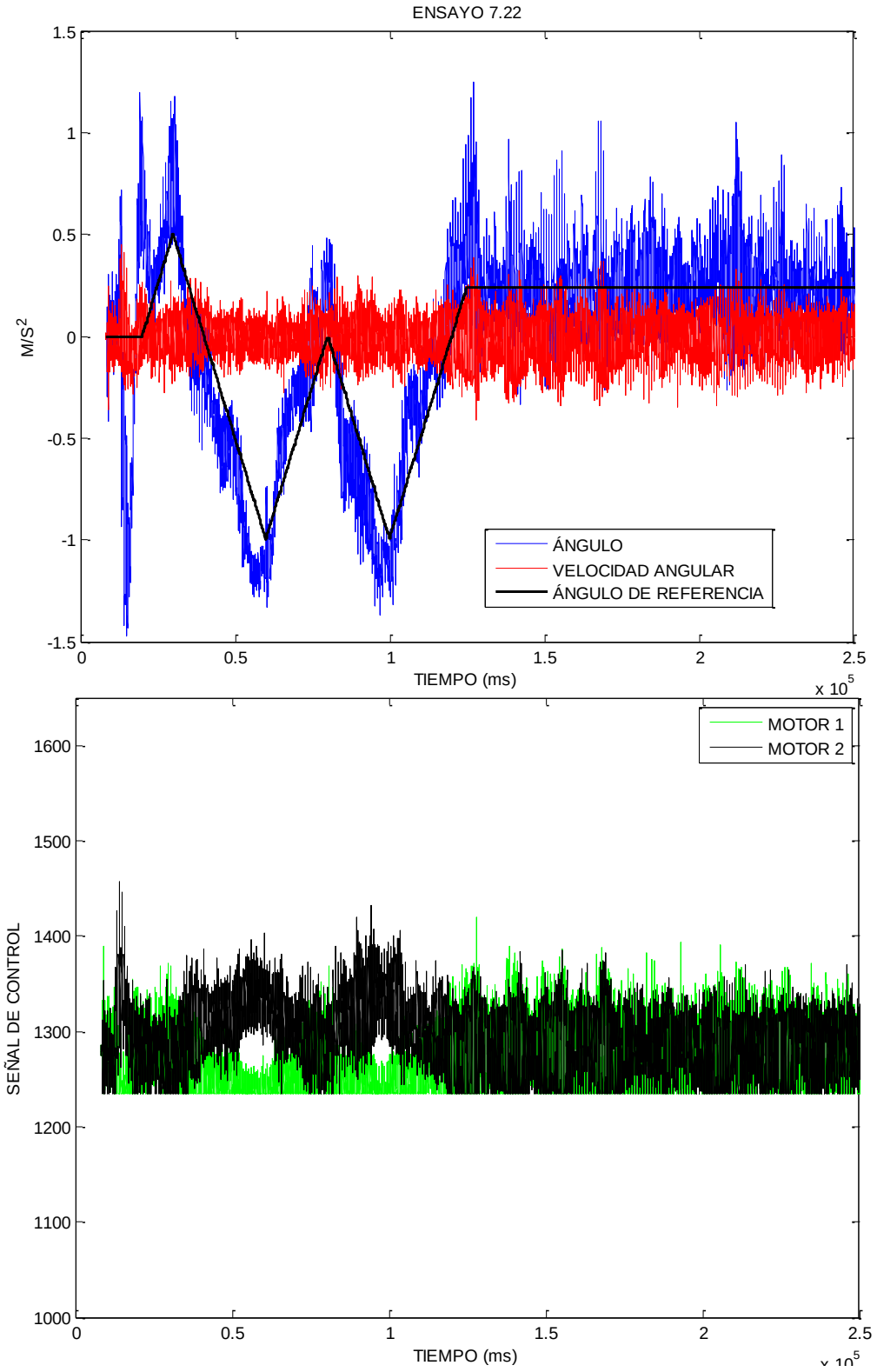


Ilustración 97: ensayo 7.22

En el ensayo 7.23 se han tomado rampas de diferentes pendientes:

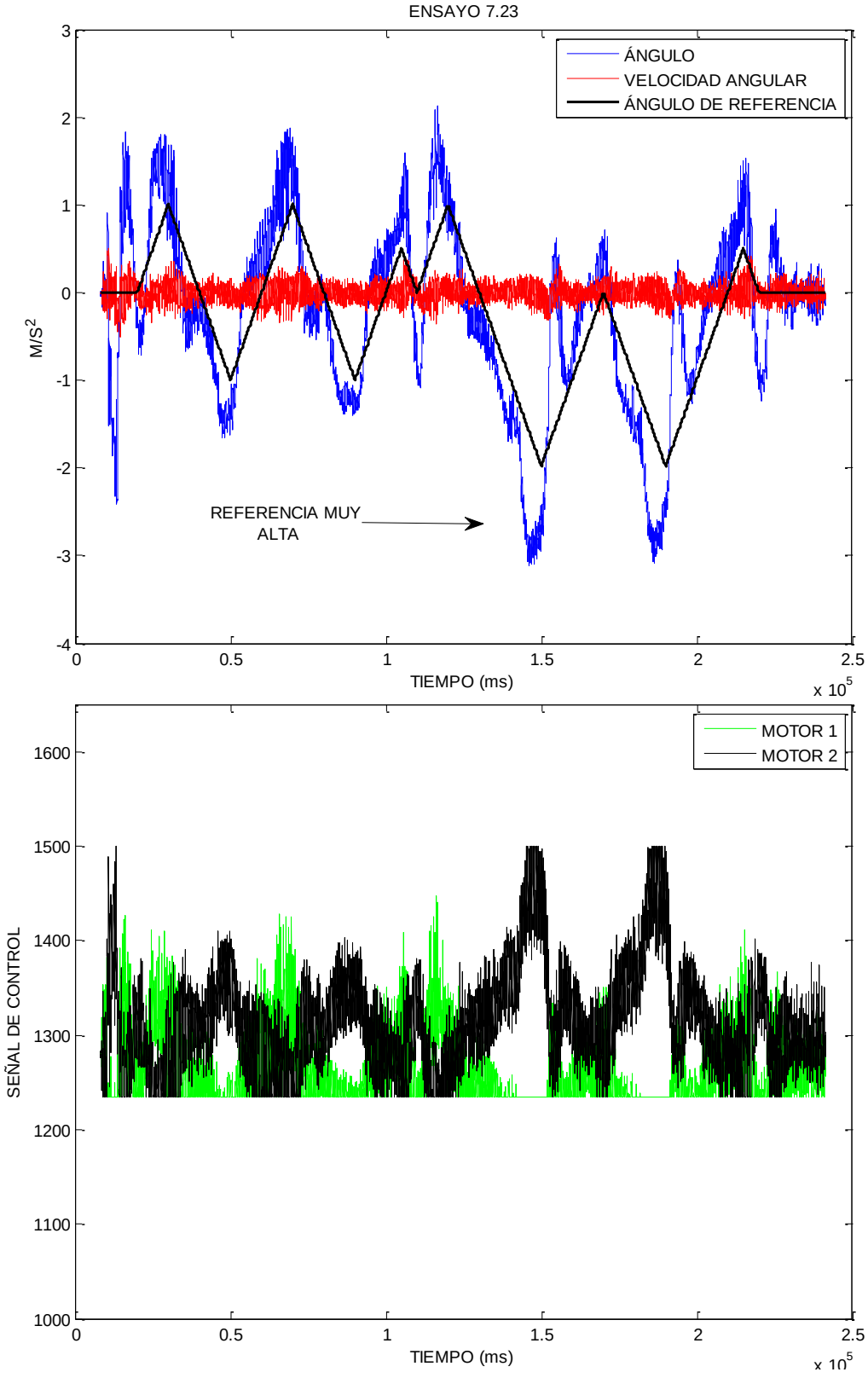


Ilustración 98: ensayo 7.23

En 7.25 se ensayan rampas y escalones conjuntamente:

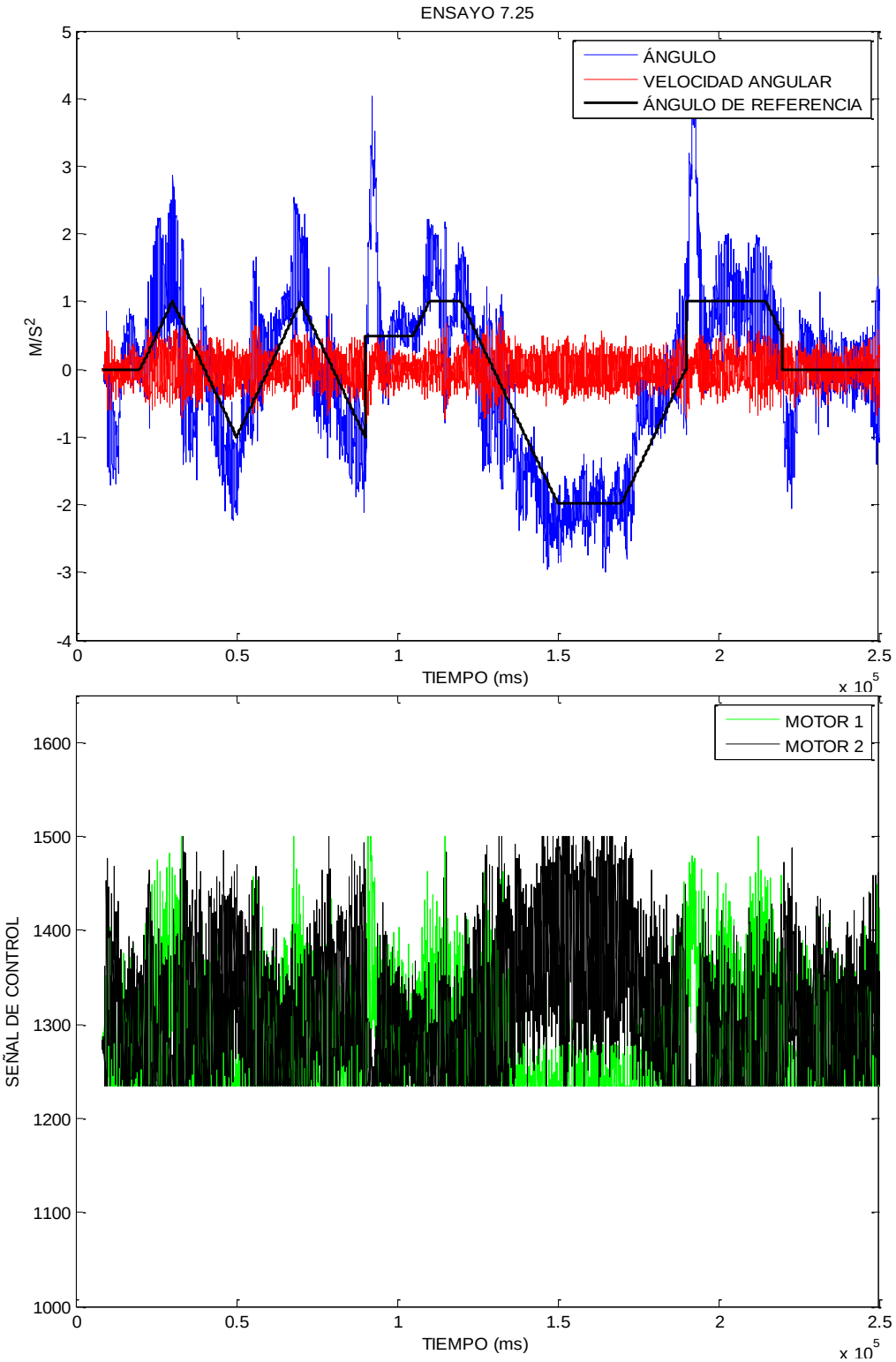


Ilustración 99: ensayo 7.25

Comprobamos que, debido a la anatomía de la estructura, no se puede controlar el eje Roll con la misma precisión que se controla el eje Pitch. Como podemos ver en la ilustración 100 es debido a que la estructura no es simétrica y proporciona mucha más inestabilidad y vibraciones en este eje.

Ilustración 100: estructura para el anclaje.

5.3 Plataforma móvil sin restricciones de movimientos.

Una vez ensayados los dos ejes Pitch y Roll por separado, se han eliminado todos los topes mecánicos y se ha escrito un nuevo software superponiendo el control que se hacía anteriormente por separado, de esta manera cada eje tiene filtro y controlador PID independiente.

Como se puede ver en el ensayo 8.15 el eje Roll vibra muchísimo por la anatomía del tren de aterrizaje. En la figura 101 se puede ver como en el eje Pitch es más rígido que en el eje Roll, (figura 101)

Ilustración 101: tren de aterrizaje

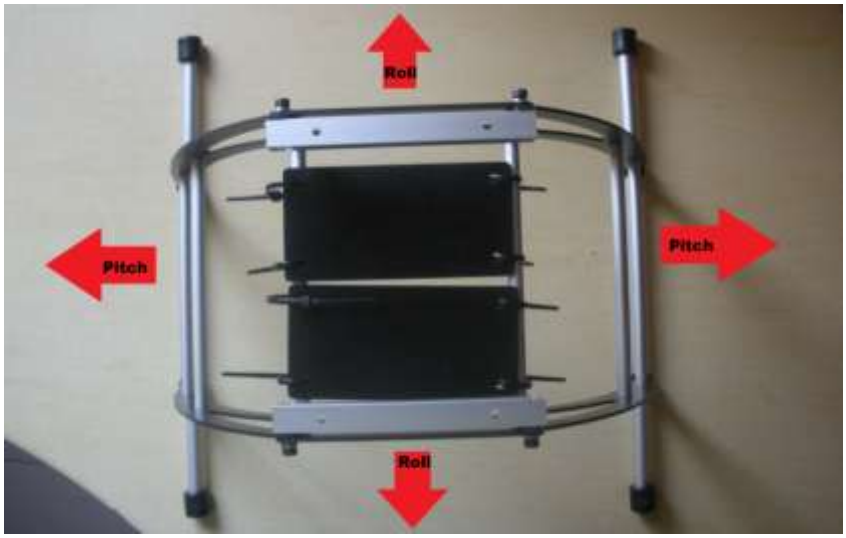


Ilustración 102: vista superior del tren de aterrizaje.

Se ha reforzado la estructura del tren de aterrizaje y se han corregido las vibraciones. Pero se comprobó que la estructura pesa demasiado alterando el modelo físico, además se aleja de la aproximación al modelo real, que es lo que se busca con estos ensayos. (Figura 103)

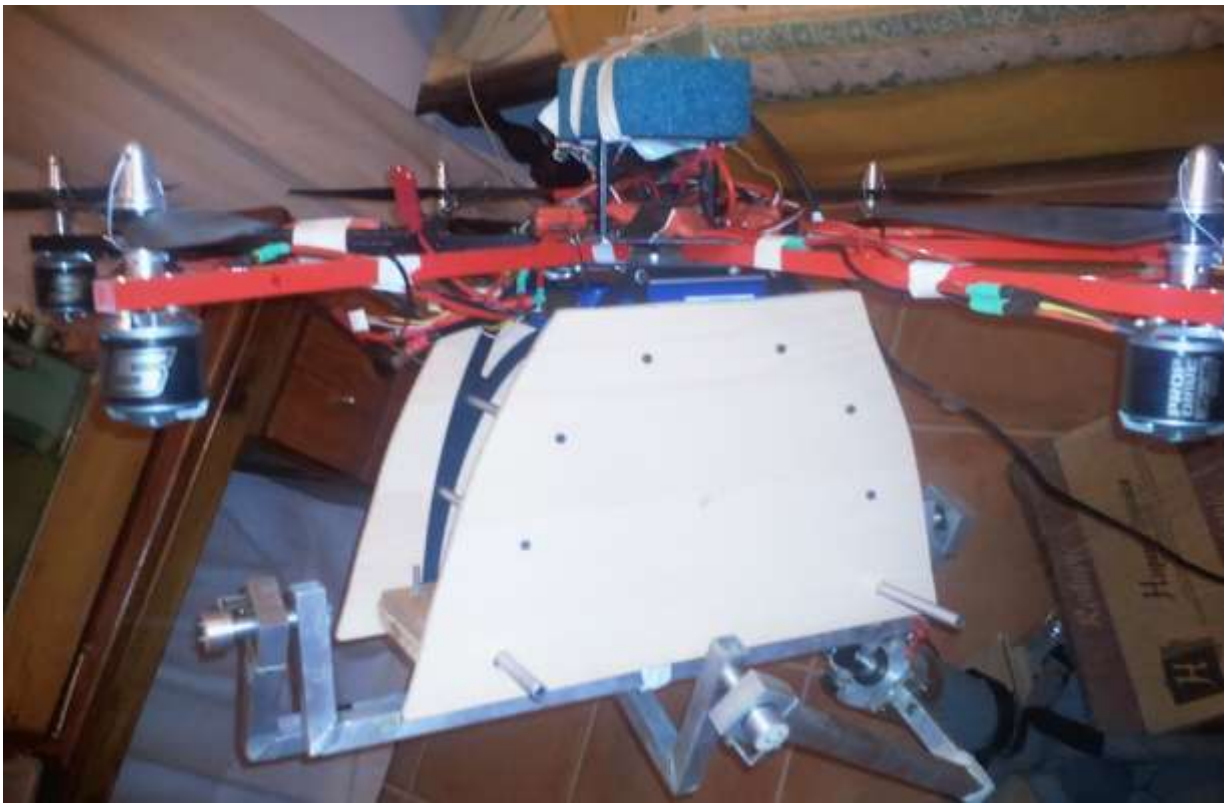


Ilustración 103: refuerzo para el tren de aterrizaje.

5.4 Plataforma móvil sin restricción de movimientos y sin cables conectados al ordenador.

Se han realizado algunas mejoras en el código:

- Nueva estructura de código con prioridad de tareas.
- Almacenado de datos.
- Control PI del ángulo Yaw.
- Comprobación del estado de batería.

5.4.1 Estructura del software

Con motivo de agilizar el software, se ha creado una nueva estructura con prioridades para actualizar más frecuentemente las tareas más importantes como la estabilización, y ejecutar con menos frecuencia las tareas secundarias como la estabilización de la cámara. A continuación se muestran unas tareas con sus correspondientes prioridades:

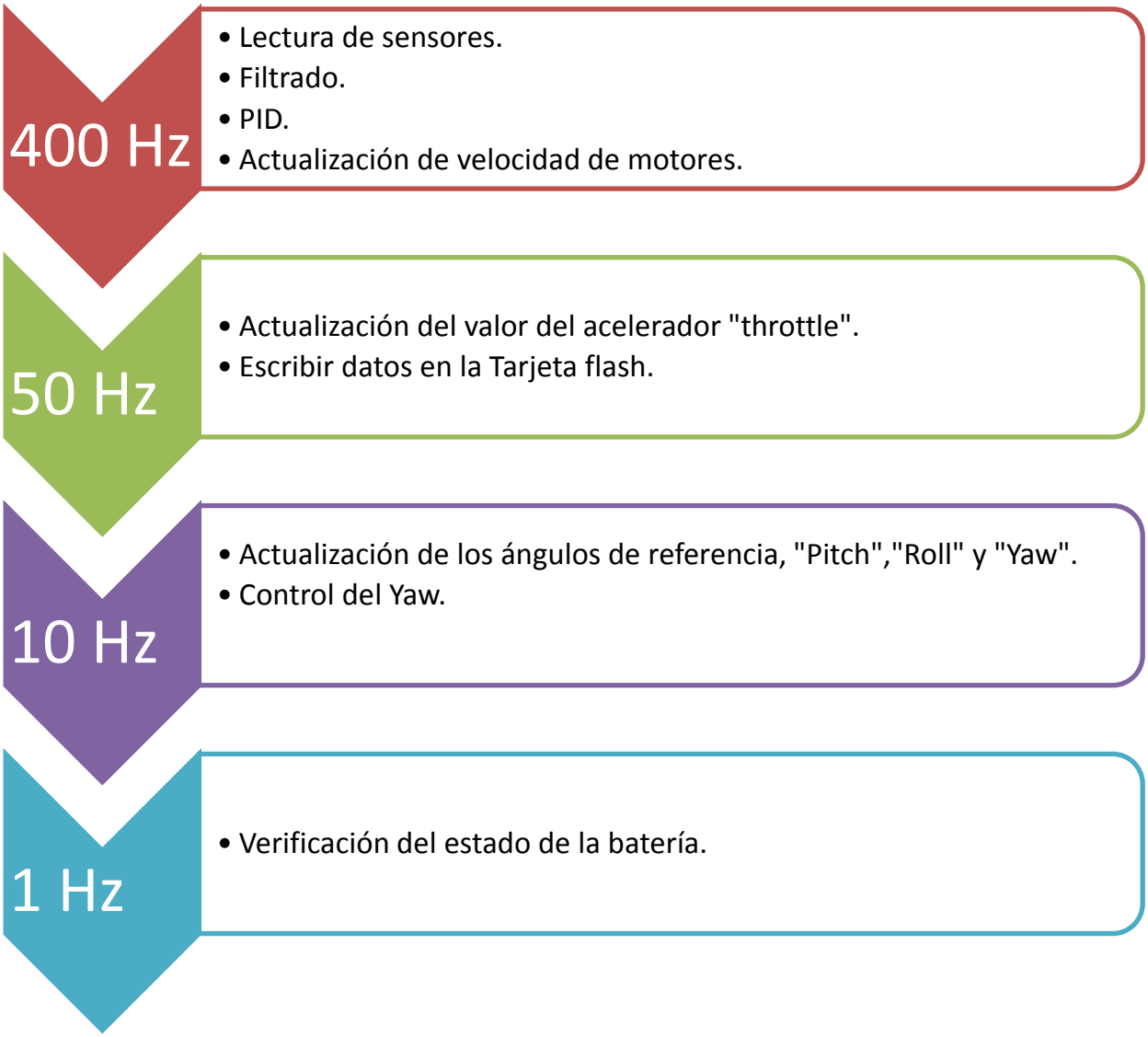


Ilustración 104: prioridades en las tareas.

A continuación en la figura 105 podemos ver el diagrama de flujo correspondiente:

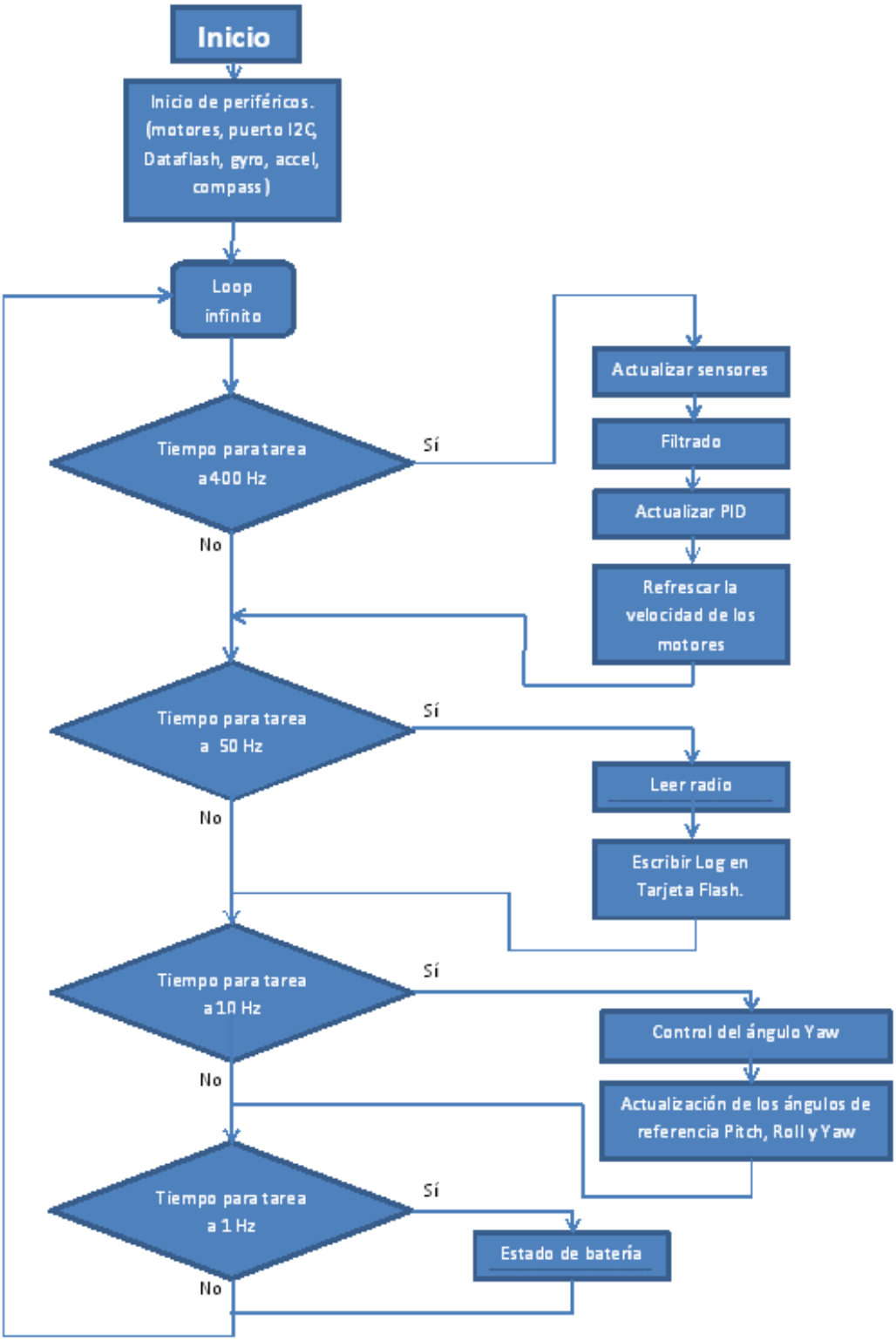


Ilustración 105: diagrama de flujo.

Para ejecutar el programa se ha creado una estructura cuyos rasgos generales es:

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

if ((timer - fast_loopTimer) >= 2500) {    //400 Hz

            fast_loop();
 }
if ((timer - fiftyhz_loopTimer) >= 20000) { // reads all of the necessary trig functions for cameras, throttle, etc.
                    fiftyhz_loopTimer = timer;

                medium_loop();    //10 Hz

                    fiftyhz_loop();    //50 Hz

                    counter_one_herz++;

                    if(counter_one_herz == 50){
                                super_slow_loop();    //1 Hz
                            counter_one_herz = 0;
                    }

            }
}//fin del void loop

 static void fast_loop()
{


}

static void medium_loop()
{
 switch(medium_loopCounter) {

                    //---------------------------------------
                    case 0:
                            medium_loopCounter++;

                break;
```

```
//-----------------------------------------------
                case 1:
                        medium_loopCounter++;

                break;

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


                        break;

                //-----------------------------------------------
                case 3:
                        medium_loopCounter++;


                        break;

                //---------------------------------
                case 4:
                        medium_loopCounter = 0;

                        break;

                default:
                        // Para englobar a todos
                        // ----------------------

                        medium_loopCounter = 0;

                        break;
        }


}//fin medium_loop
```

5.4.2 Memoria flash

Debido a la inexistencia de una conexión inalámbrica bilateral, no podemos mandar telemetría en tiempo real a la estación de tierra. En su lugar usamos un almacenaje en memoria flash de una matriz con parámetros que nos permitan un análisis posterior, descargando tras cada vuelo los datos en el ordenador.

Se almacena una matriz de datos en la que cada columna en el orden mostrado en la tabla inferior y cada fila es un conjunto de datos.

Roll filtrado	**Pitch filtrado**	**Giro Roll filtrado**	**Giro Pitch**	**Tiempo (ms)**	**Señal de velocidad motor 1**	**Señal de velocidad motor 2**	**Señal de velocidad motor 4**	**Señal de velocidad motor 5**	**Señal de referencia Roll**	**Señal de referencia Pitch**	**Compass (°)**	**Acelerador**
F	I	L	A		1		D	A	T	O	S	
F	I	L	A		2		D	A	T	O	S	
F	I	L	A		3		D	A	T	O	S	

Para descargar los datos en el ordenador, hay que conectar el cable USB y abrir el monitor serial. Cuando inicie el programa en el Arduino se leerá el siguiente mensaje:

```
"Bienvenido;
Pulse....
'L' para Leer datos
'B' para Borrar datos

       y después INTRO"
```

Si no hacemos nada se arrancará el programa normalmente. Si pulsamos ‘L’ podremos copiar la matriz de datos en un archivo de texto. Si pulsamos ‘B’ se borrará la memoria y se finaliza el programa.

5.4.3 Control del Yaw con realimentación.

Utilizando el sensor llamado magnetómetro, también conocido como compass (brújula) se puede conocer la orientación debido a que el sensor detecta diferencias entre campos magnéticos. El sensor mide el campo magnético en tres ejes y analizando podemos obtener la orientación. En este proyecto se ha utilizado una librería <AP_compass.h> que calcula la posición. El sensor es el HMC5883L del fabricante Honeywell.

Por su largo uso militar para detectar submarinos, gobiernos como el de USA, Canadá o Australia clasifican los sensores magnetómetros más precisos como material militar, y limitan su distribución.

Sólo se realiza el control del ángulo Yaw, con un controlador PI, cuando las referencias Pitch y Roll son nulas. Esto debido a que cuando variamos los ángulos Pitch o Roll hay que realizar unas correcciones en el ángulo Yaw en función de la inclinación. La función de control del yaw se ha situado en el medium_loop() case 0 para que se ejecute a 10Hz.

En el ensayo C5 se ha marcado una referencia para el ángulo Yaw. En la gráfica 106 se puede ver que existe alguna sobre oscilación pero la consideramos admisible al no ser muy relevante la orientación de este eje.

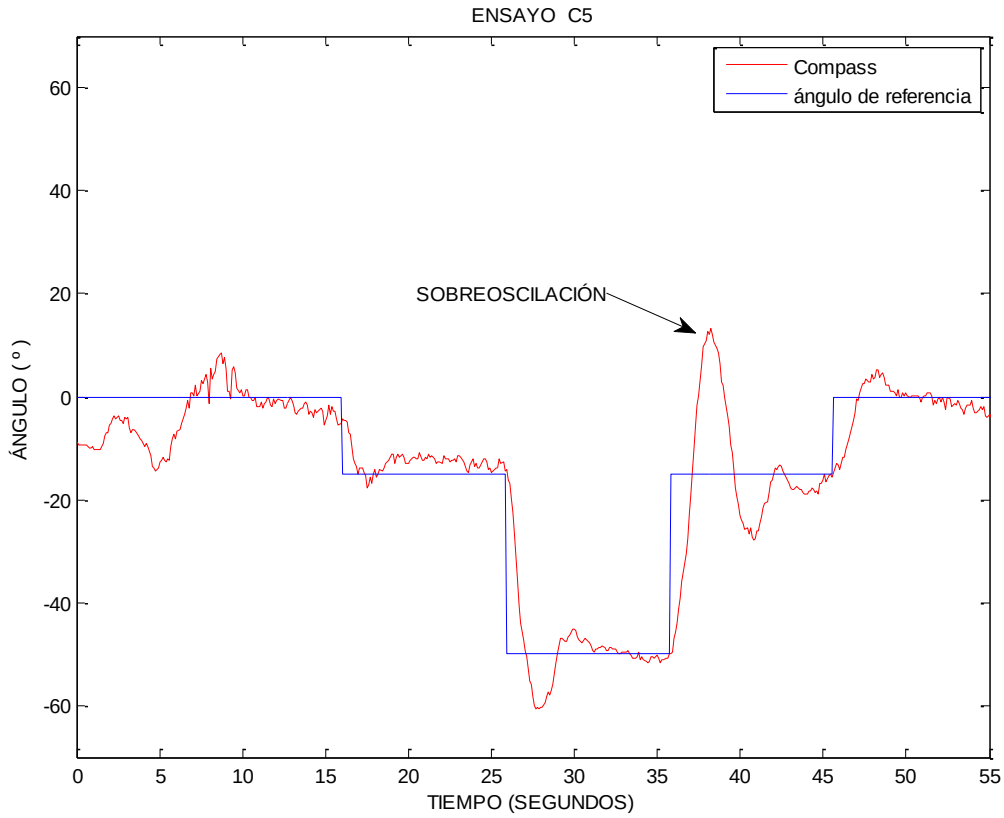


Ilustración 106: control PI sobre el ángulo Yaw.

5.4.4 Comprobación del estado de la batería.

Se ha instalado un divisor de tensión para medir el estado de tensión de la batería. Para ello medimos la tensión de una celda, ya que, idealmente suponemos que el desgaste es uniforme. En caso de que se detecte que el nivel es bajo, activaremos una señal luminosa, que advierta al piloto que ha de terminar el vuelo.

Las baterías del tipo "LiPo" utilizadas por el helicóptero, como ya se ha expuesto anteriormente, están compuestas por 3 celdas. Cada celda tiene una tensión nominal de 3,7V, y para no dañar la batería, esta tensión no debe bajar nunca de los 3V.

Se han instalado unas tiras LED color rojo en la parte inferior de los brazos, fácilmente visibles desde abajo, que sirven como señal luminosa. Están conectados directamente a las baterías para alimentarlos a 12V y para encender o apagar los LED se usará el relé instalado en la placa IMU.

El relé tiene capacidad suficiente para encender los LEDs. Modelo "AXICOM IM23GR" cuyas características principales son:

- Potencia: 60W
- Voltaje: 220V /250 VAC.
- Corriente: 2 / 5A.

Los LED usados son como el de la ilustración 107. Es una tira modulable y adhesiva, de manera que podemos engancharlo a la parte inferior de los brazos del helicóptero.

Las características de estas tiras son:

- 60 led/metro.
- Led tipo smd3528.
- Consumo 0.08 W/led.

Ilustración 107: tira de LED rojos.

Al helicóptero se le han colocado 3 tiras de 18 LEDs cada una, con un consumo total es:

$$Consumo = 18\ Led \cdot 3\ tiras \cdot 0.08\ W/Led = \ 4.32\ W \qquad Amperios = \frac{4.32W}{12V} = 360\ mA$$

El consumo, comparándolo con el de los motores es mínimo, por lo que pueden encenderse siempre, bien para vuelos nocturnos o por motivos estéticos. El nivel de batería se mide con una frecuencia de 1Hz, que es más que suficiente, con el siguiente código:

```
#define BATTERY_PIN 0
#define BATTERY_MIN 3.4 //mínimo voltaje para una celda LIPO

if(analogRead(BATTERY_PIN)*5/1024 >BATTERY_MIN) {

   PORTL ^= B00000100;    //pin donde está conectado el relé
  }
```

5.4.5 Ensayo con Pitch y Roll

Se muestran a continuación los resultados del ensayo 8.13, donde se ha probado la estabilidad del sistema con control autónomo de ambos ejes añadiéndole una señal de referencia al eje Pitch.

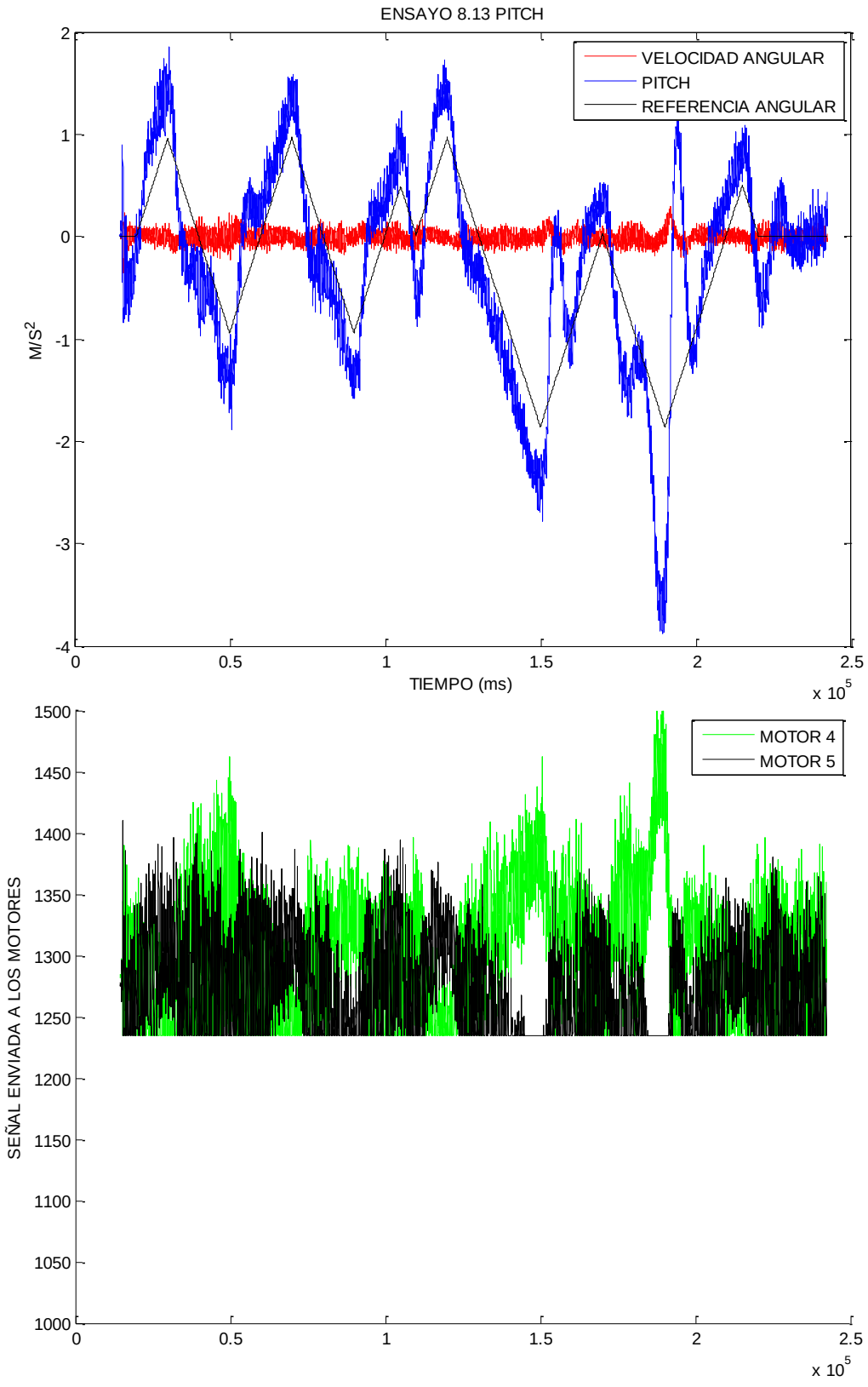


Ilustración 108: ensayo 8.13 Pitch.

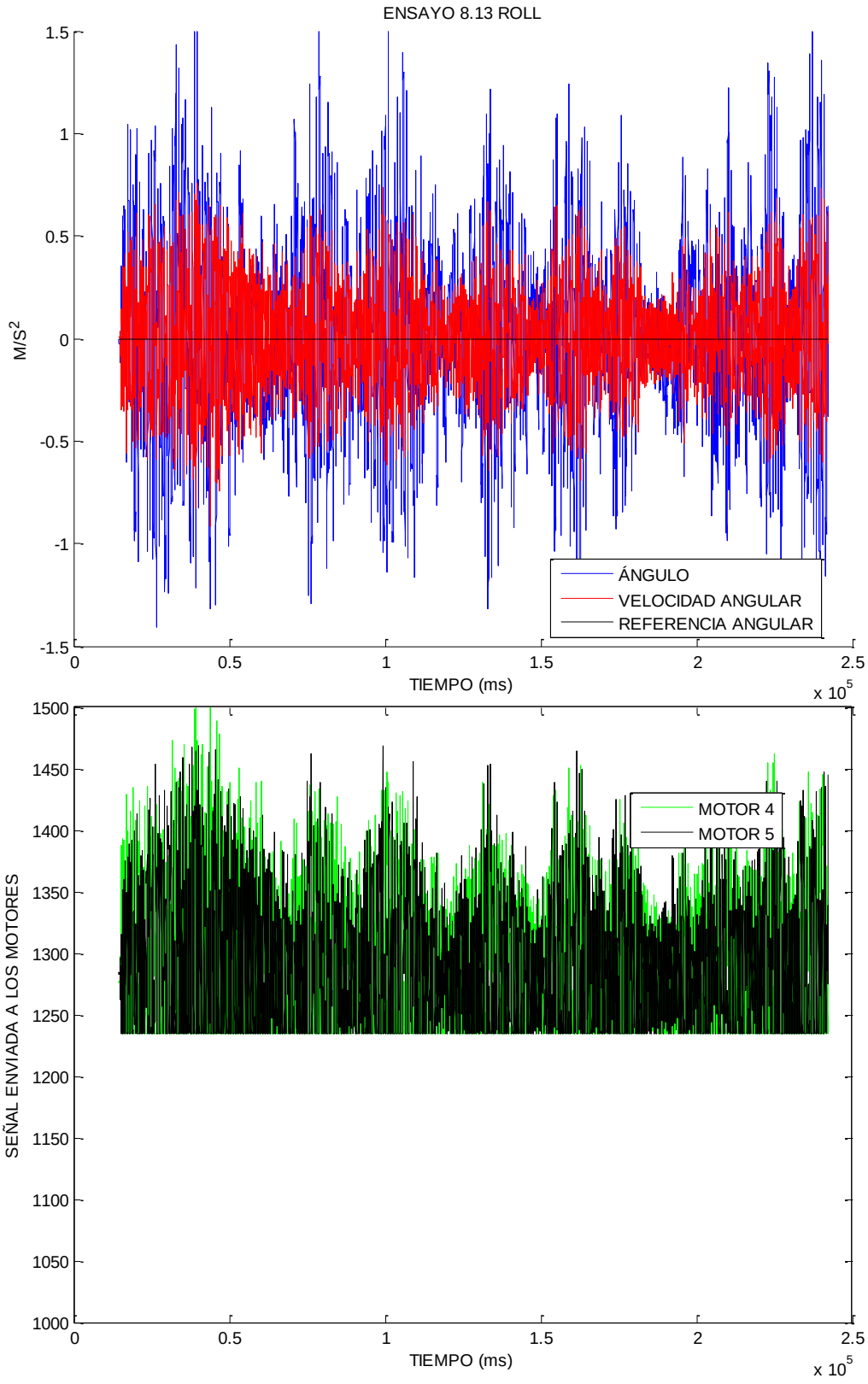


Ilustración 109: ensayo 8.13 Roll.

6 Conclusiones y trabajos futuros.

6.1 Conclusiones

En este proyecto se ha diseñado un hexacóptero mediante el análisis, dimensionamiento y la selección de todos sus componentes. Se ha construido el aparato y se ha calibrado para garantizar su operatividad. Se han utilizado diversas plataformas de sujeción para evitar su rotura durante los experimentos. Finalmente se han sintonizado varios controladores para conseguir un seguimiento de la consigna en cada uno de los ejes por separado y de forma conjunta.

En los últimos ensayos se han llegado a controlar los tres ejes simultáneamente, el sistema de comunicación responde bien, así como el almacenado y transmisión de datos. Muchos de los problemas obtenidos están asociados a la plataforma, y probablemente desaparecerán en los futuros vuelos en aire libre.

El resultado final no fue tan robusto como el esperado para poder mantener el hexacóptero estable en vuelo real. Aunque se realizaron dos pruebas de vuelo que acabaron con el prototipo chocando contra el suelo, afortunadamente sólo se rompieron varias hélices. Las razones por las que el prototipo no funcionó en vuelo real pudieron ser:

- **El modelo físico usado en los test en laboratorio es muy diferente al modelo en el aire**, por lo que los parámetros PID necesitan ser reajustados. En el modelo del laboratorio, el centro de masas está situado por debajo del modelo real, también el eje de giro para ambos ejes está por debajo. La representación de estos ejes se puede ver en la ilustración 110.
- En el laboratorio sólo se ha trabajado con los motores a bajas velocidades, debido a que la estructura vibraba mucho. En vuelo real, se ha de utilizar una velocidad, al menos, suficiente para vencer la fuerza peso del helicóptero.

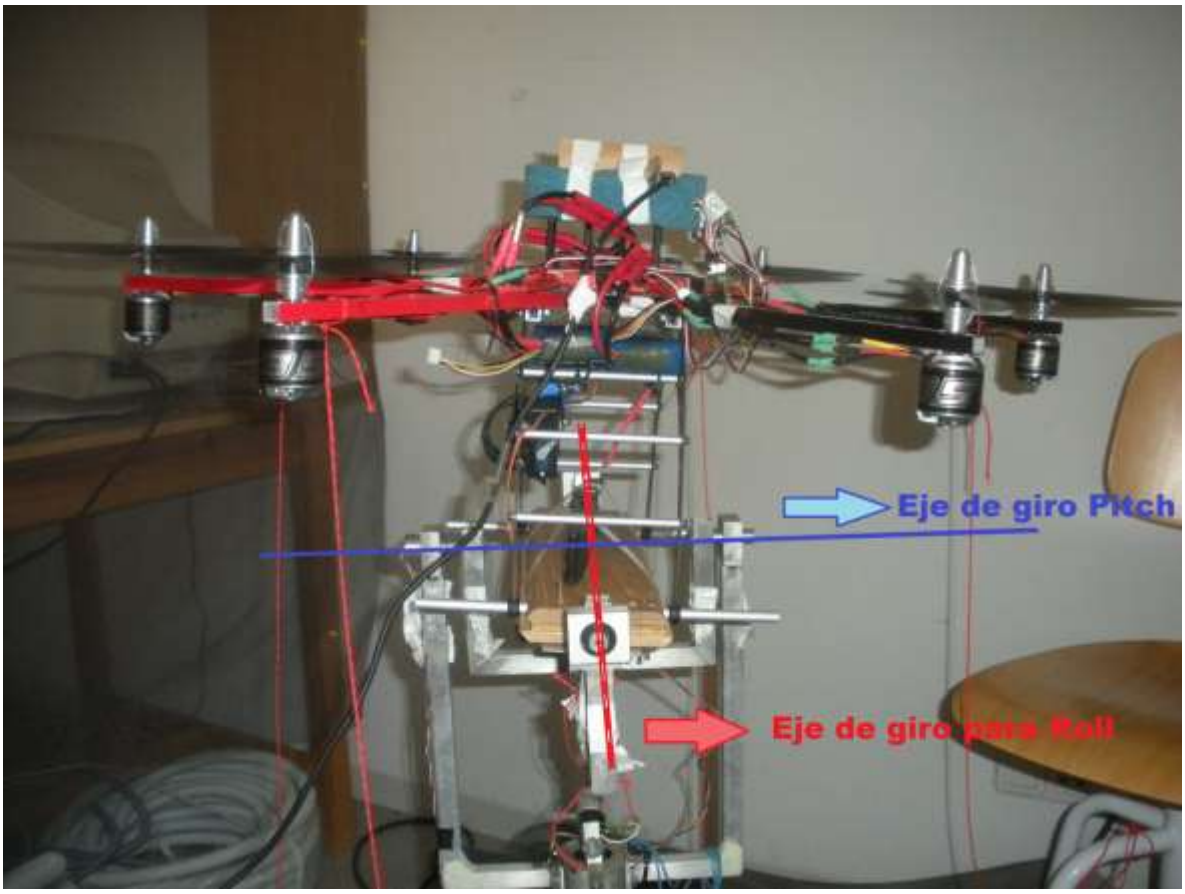


Ilustración 110: ejes de giro

6.2 Trabajos futuros.

Para conseguir un hexacóptero totalmente operativo será necesario continuar con algunas tareas:

- Implementar un código que trabaje con matrices DCM (Direction Cosine Matrix), que representaría en cada momento la orientación del helicóptero respecto a la tierra como una rotación. Esto se podría hacer con matrices o con cuaterniones. Con las matrices de rotación se tendría una adaptación más natural para el control de vuelo que con los cuaterniones.

- Ajustar un mejor control para el eje Yaw que soporte giros de más de 360 grados.

- Añadir telemetría inalámbrica para conocer en tiempo real datos como el estado de la batería, altura, posición, velocidad, consumo actual, etc… Para esto se añadirán dos módulos Xbee para establecer una conexión Serial.

- Incluir un estabilizador al soporte de la cámara de manera que esta se mantenga en la misma posición cuando el helicóptero maniobre. De esta manera se pueden hacer vídeos sin la sensación de mareo al moverse.

- Añadir la transmisión inalámbrica de video en tiempo real, para poder ajustar las imágenes. También se puede usar para controlar el helicóptero con unas gafas con LCD y poder simular el vuelo en primera persona.

- Mejorar el filtrado de la señal y medición del ángulo. Una técnica llamada filtro complementario es un tipo de filtro que tiene en cuenta las señales procedentes del giróscopo y acelerómetro lo que permite converger a un valor más exacto.

7 Bibliografía de Referencia.

7.1 Referencias bibliográficas.

- BOE núm. 130 .Viernes 28 de mayo de 2010. Sec. I, página 46336 y ss.
- European Radio communications Committee (ERC),REPORT 25.
  "European table of frequency allocations and utilizations frequency range 9KHz to 275 GHz.
- Hahn, Brian. Valentine, Daniel T. Essential Matlab for Engineers and Scientists. Cuarta edición. ISBN: 978-0-12-374883-6.
- Goppert, James M. An adaptable, low cost test-bed for unmanned vehicle systems research. Purdue University, Indiana, EEUU. 04/20/2011
- Ogata, Katsuhiko. Ingeniería de Control Moderna. Controles PID e introducción al control robusto. Tercera edición. Pagina 669 y ss. Editorial Prentice Hall. 1998.
- Ogata, Katsuhiko. Sistemas de control en tiempo discreto. Segunda edición. Página 116 y ss. Editorial Prentice Hall. 1996.
- Premerlani, William. Bizard ,Paul. Direction Cosine Matrix IMU: Theory. 5/17/2009.
- Sikiric, Vedran. Control of Quadrocopter. KTH Computer Science and communication, Stockholm, Sweden. 2008. ISS N-1653 - 5715
- Sistema de Observación y Prospectiva Tecnológica (SOPT): "UAS "Unmanned Aurcraft system" Sobre su integración en el espacio aéreo no segregado". Ministerio de defensa.

7.2 Referencias Web

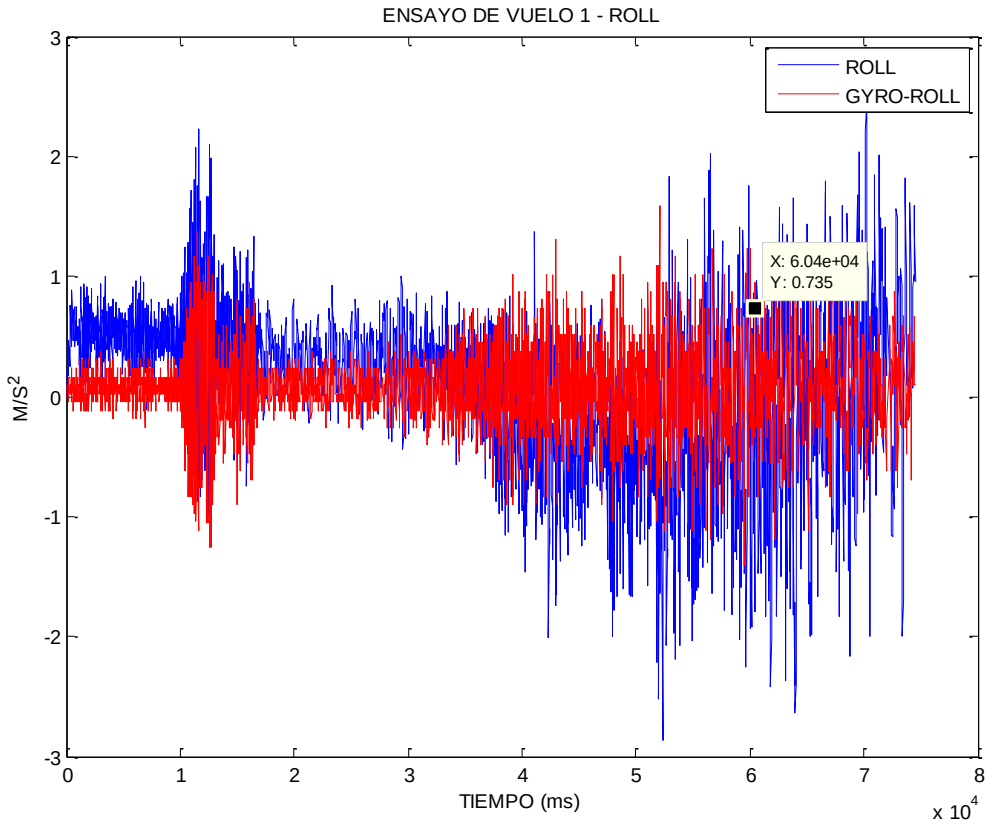
- DIY Drones
  Página oficial del proyecto Open Source "Diy Drones".
  www.diydrones.com
- Arduino Language Reference.
  Lista con las funciones disponibles en las librerías Arduino.
  http://arduino.cc/en/Reference/HomePage
- Wikipedia.
  Enciclopedia libre.
  www.wikipedia.com.
- Hobbyking
  Distribuidor de productos RC.
  www.hobbyking.com.
- Tower Hobbies
  Distribuidor de productos RC.
  http://www.towerhobbies.com/
- Fly News
  Portal de noticias de aviación.
  www.fly-news.es

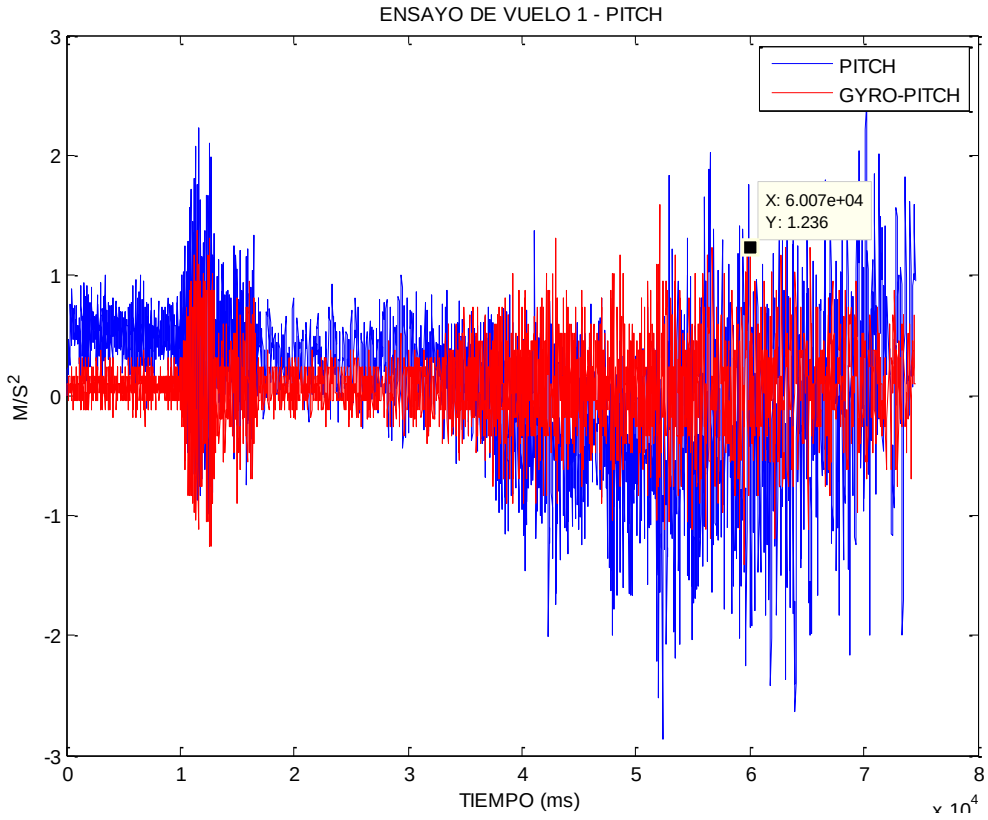
Anexos

Anexo A: Pruebas de vuelo en el aire libre

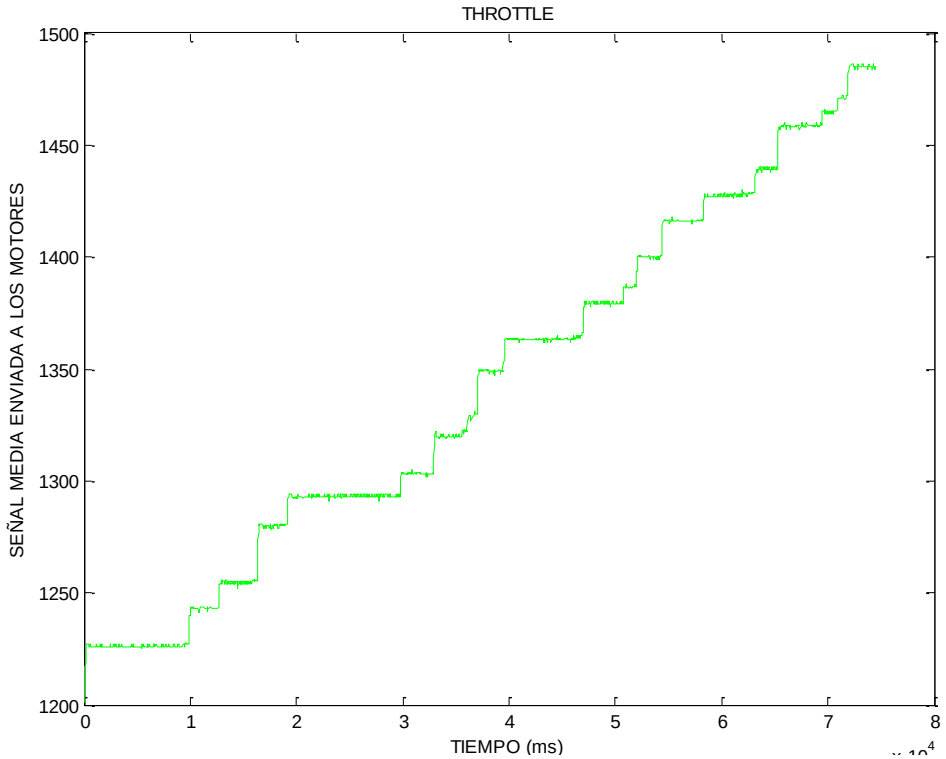
Se realizaron dos pruebas de vuelo de las que se consiguieron sacar algunas conclusiones. Los tests tuvieron lugar sobre un terreno con césped y alejado. Los datos se almacenaron en la memoria flash y luego se descargaron en el ordenador. En la carpeta de anexos se puede encontrar el archivo <prueba de vuelo 1.mat>.

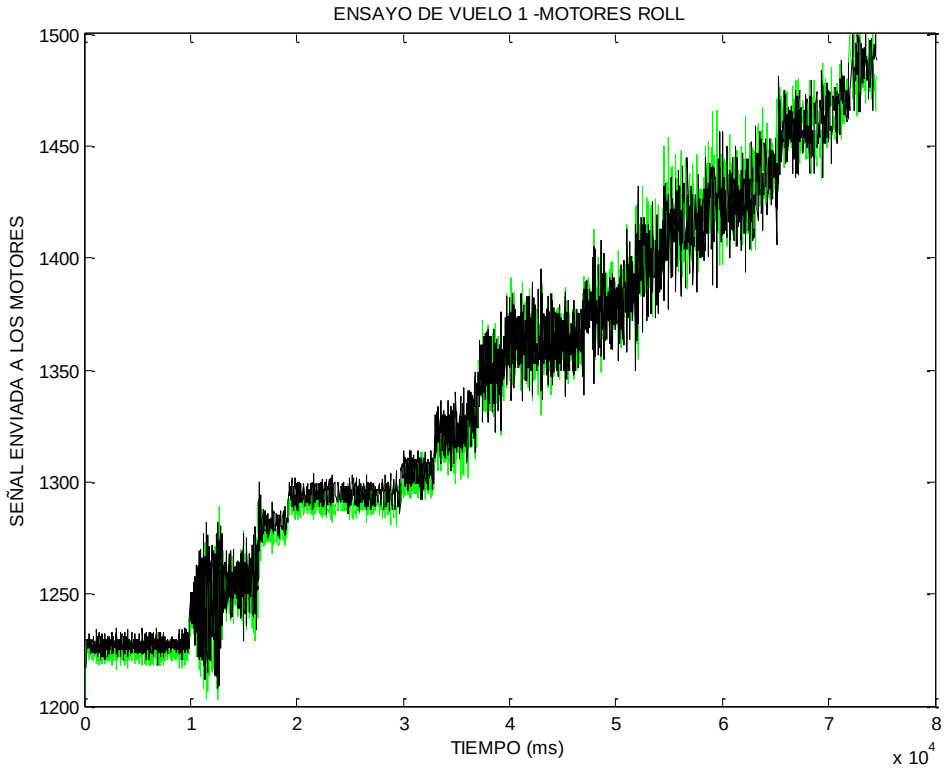
En el primer ensayo se trató de comprobar a que velocidad del acelerador empieza a despegar el hexacóptero.


ENSAYO DE VUELO 1 - ROLL
ROLL
GYRO-ROLL
X: 6.04e+04
Y: 0.735
M/S2
TIEMPO (ms)
x 10^4

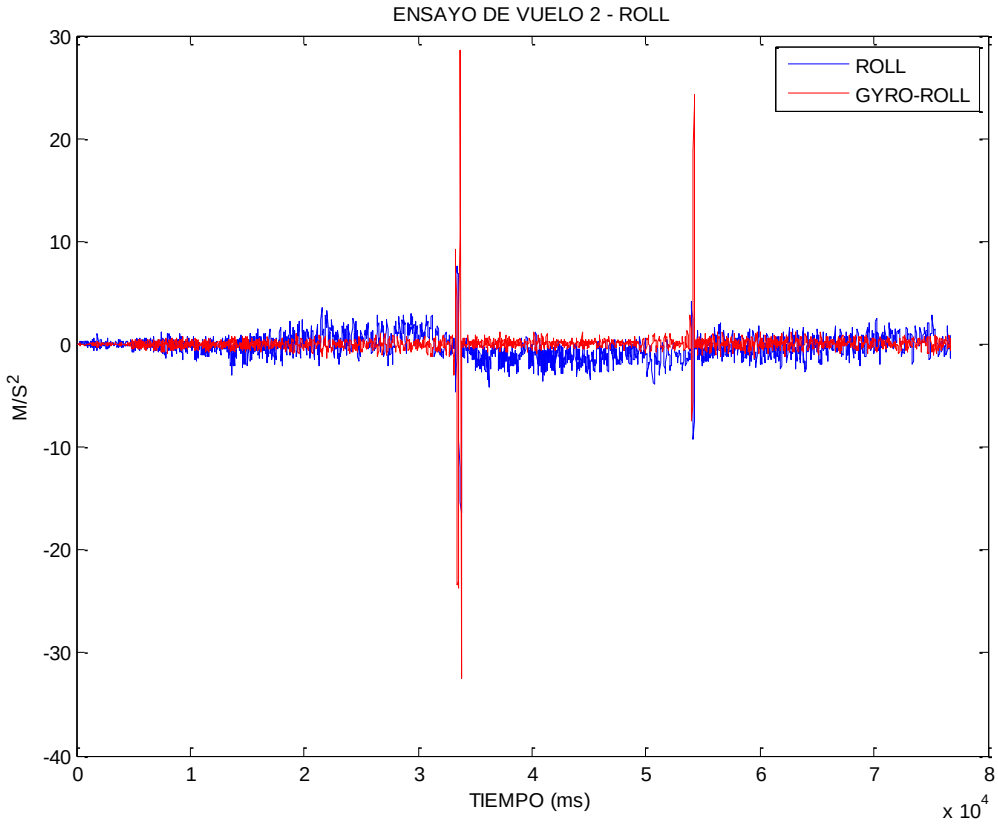

ENSAYO DE VUELO 1 - PITCH
PITCH
GYRO-PITCH
X: 6.007e+04
Y: 1.236
M/S2
TIEMPO (ms)
x 10^4

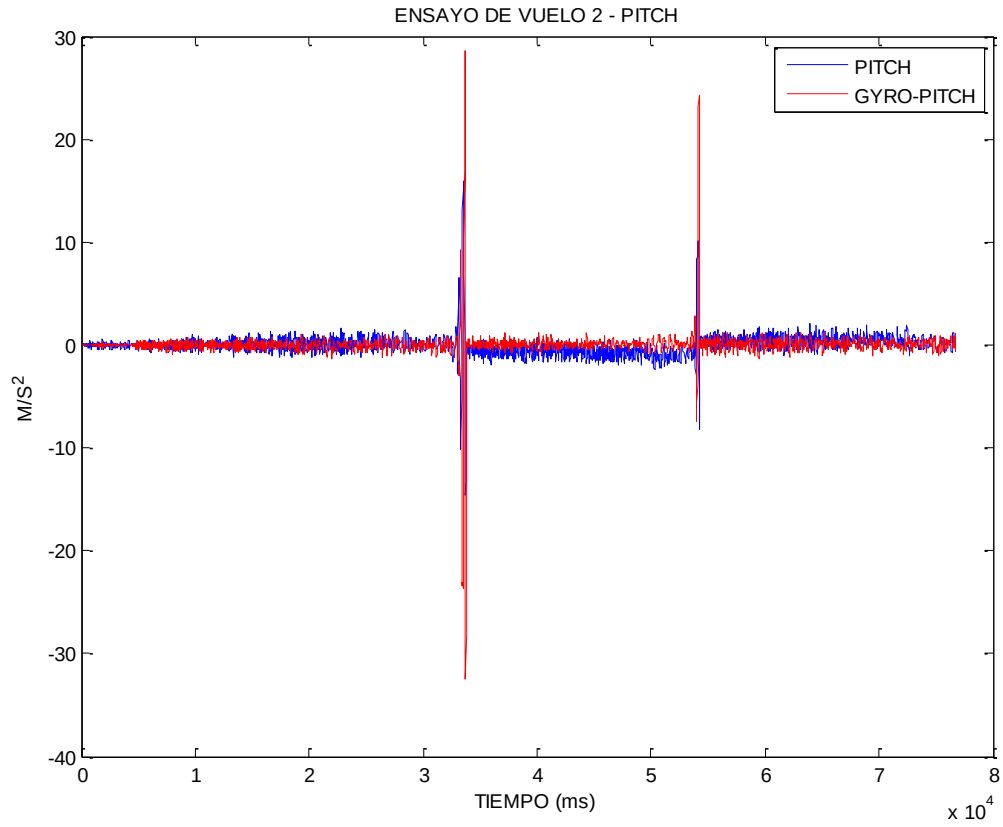
El helicóptero alcanza la velocidad del acelerador de 1495, eso significa que alcanza el limitador fijado en 1500.

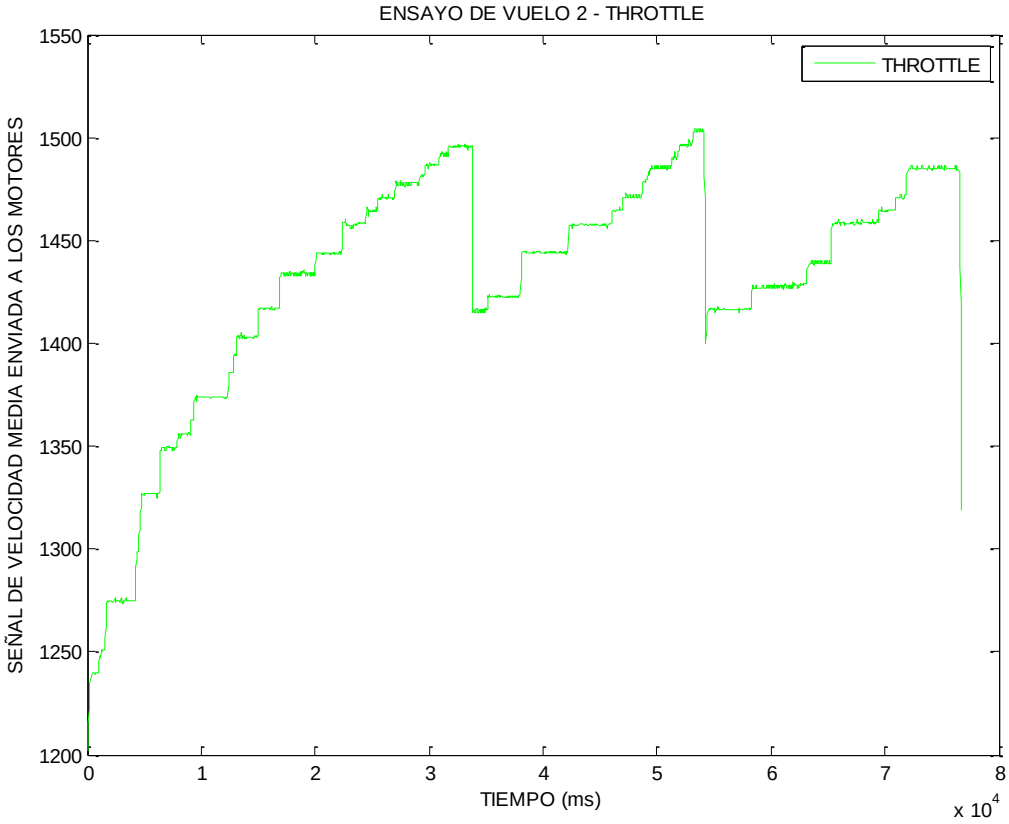


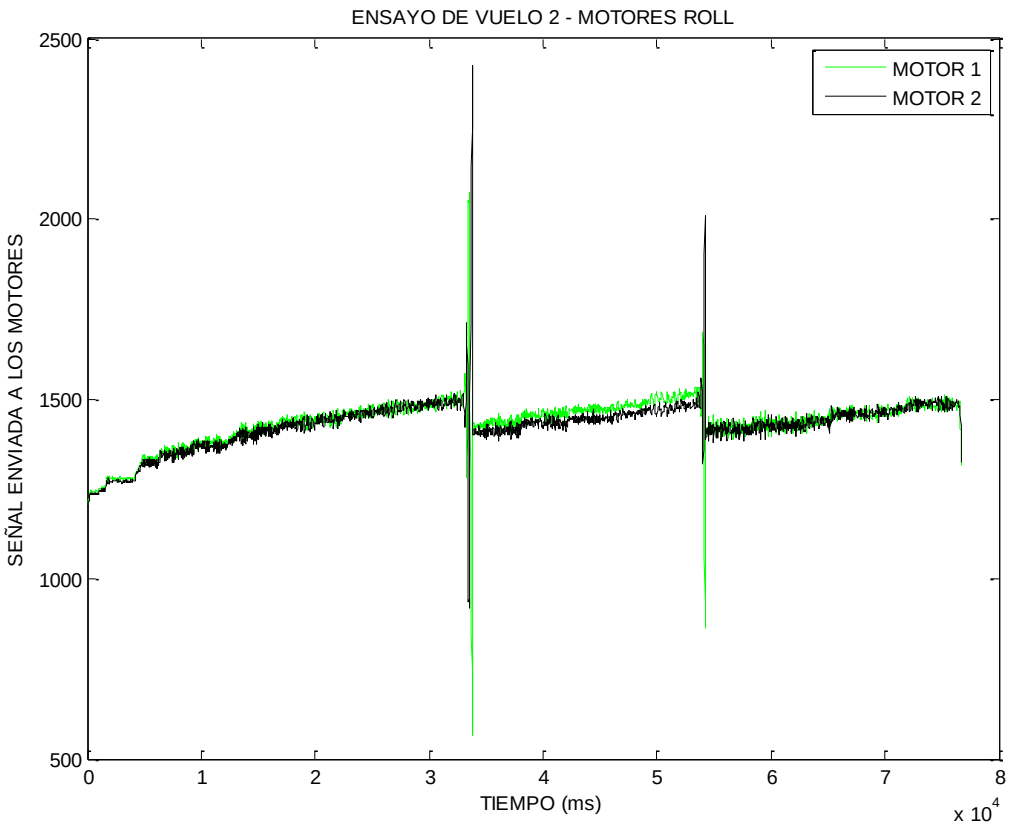


Para el siguiente test, ”ensayo de vuelo 2” se ha cancelado el límite superior de saturación de los motores. El archivo correspondiente a este ensayo es <prueba de vuelo 2.mat>.




ENSAYO DE VUELO 2 - PITCH
PITCH
GYRO-PITCH
M/S2
TIEMPO (ms)
x 10^4


ENSAYO DE VUELO 2 - THROTTLE
THROTTLE
SEÑAL DE VELOCIDAD MEDIA ENVIADA A LOS MOTORES
TIEMPO (ms)
x 10^4



El pico que se observa en la señal de los motores, no es deseable ya que generará mucha intensidad de pico, lo que podría producir que alguna hélice saliese disparada y, además esa intensidad no sería bueno para las baterías ni para los ESCs.

Podemos sacar algunas conclusiones de estas dos pruebas:

- Es necesario un reajuste de los parámetros del PID, aunque en los vídeos no se puede apreciar muy bien, el helicóptero giró bruscamente para ambos lados. Esto se ve claramente en la gráfica anterior, en el segundo 34 cuando el motor 2, en negro, pasa del valor 1000 a 2000 en muy poco tiempo. Los nuevos parámetros del PID han de ser mucho más suaves.
- Con el sistema de escritura en el log actual, sólo se escriben datos cuando el acelerador está pulsado. Debido a esto no se tienen datos de la caída porque el acelerador se desactivó un poco antes.
- Incluir en el log los datos del eje Z, ya que nos pueden ayudar a saber el momento del despegue del suelo.

Anexo B: Características de la placa de sensores IMU

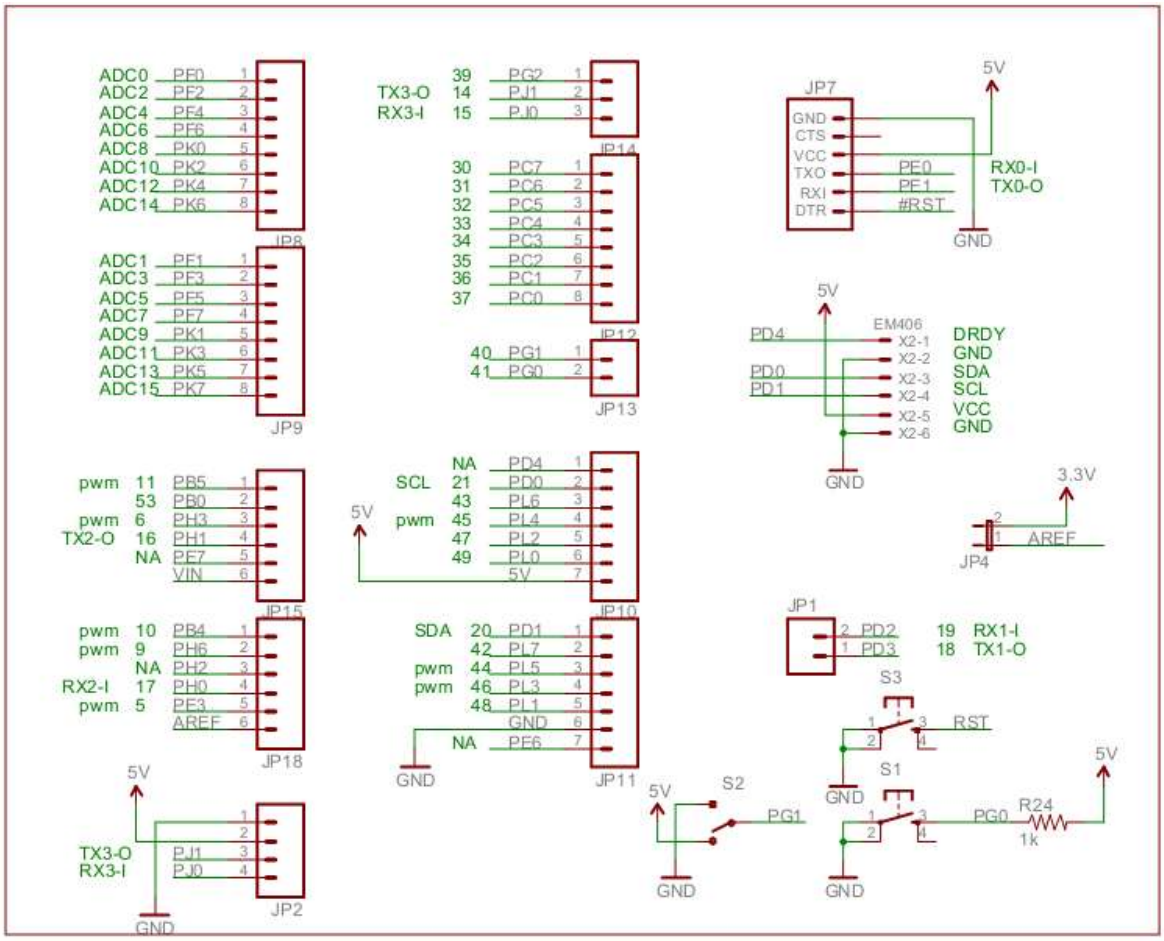
Dimensions: 2.85 x 1.60"

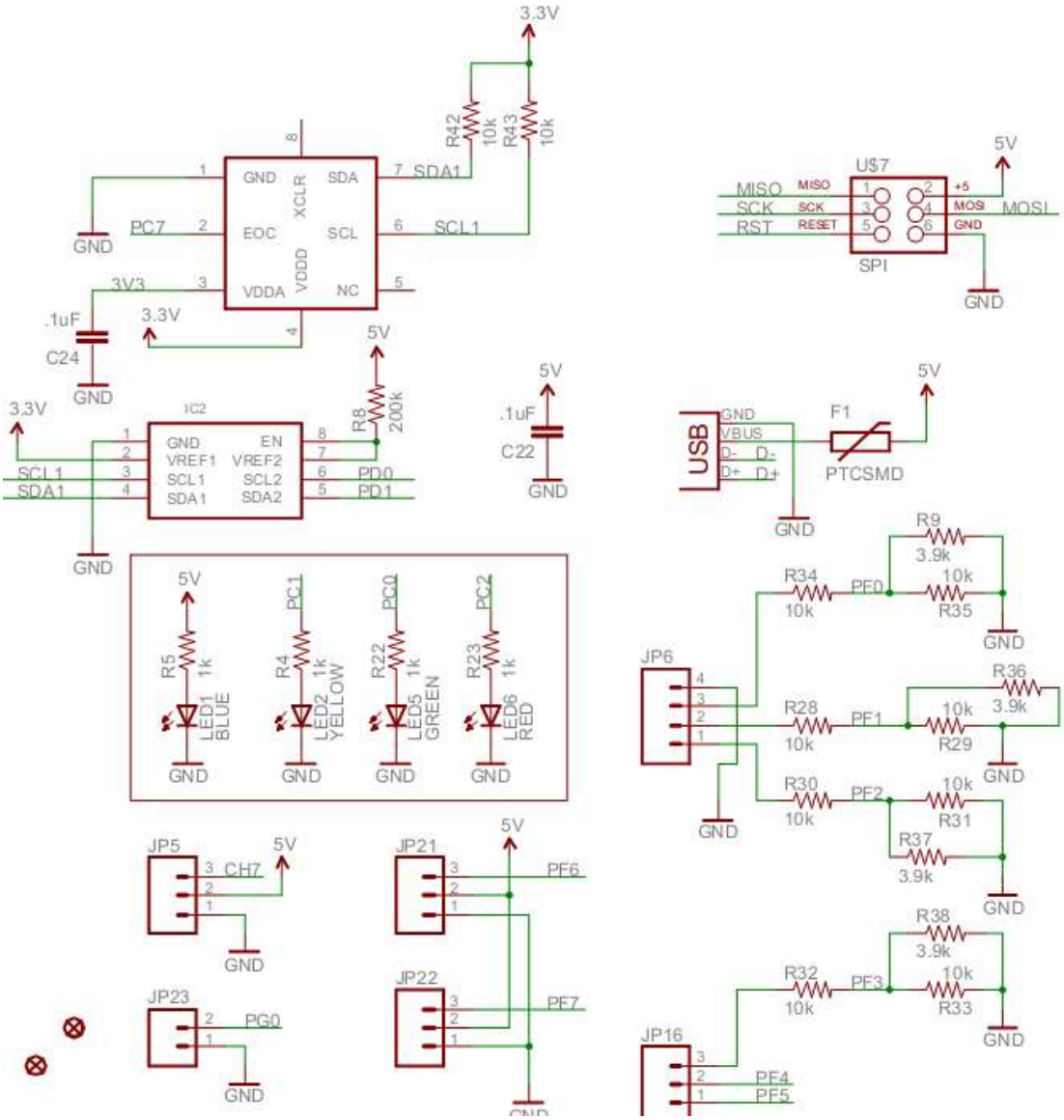
Weight: 0.5 oz. (13g)

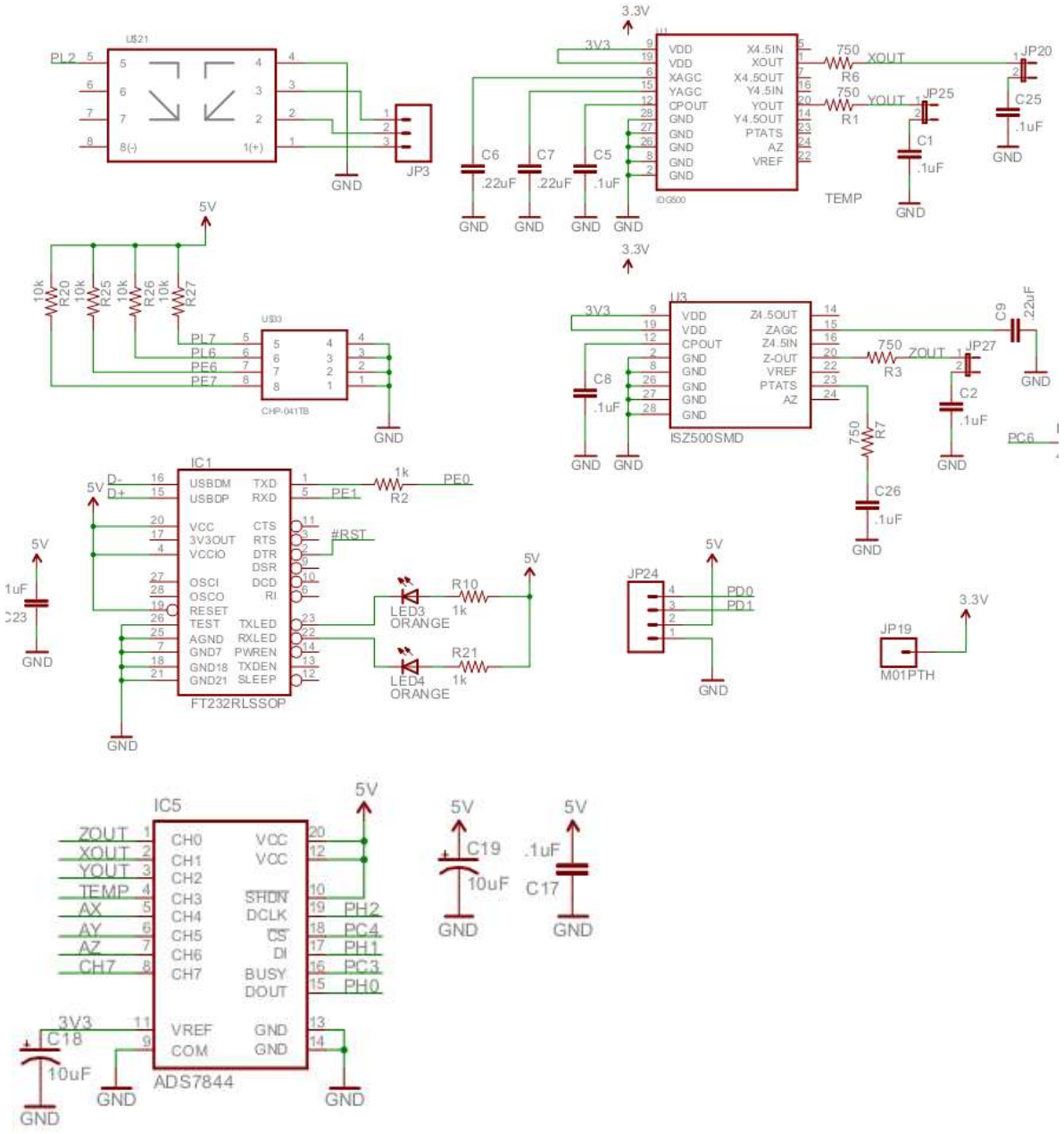
Features:

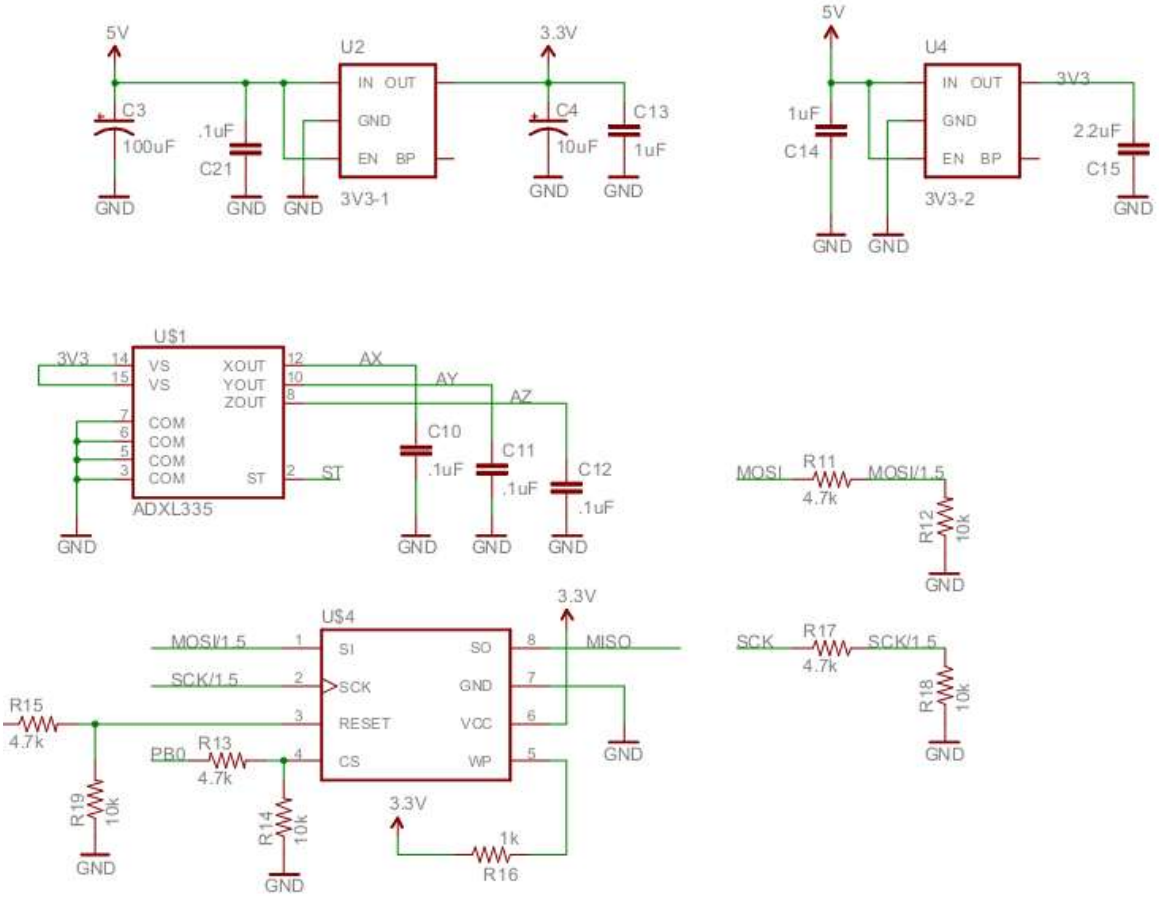
Dual 3.3V regulator (one dedicated for analog sensors!)
Relay switch for cameras, lights or payloads
12-bit ADC for better gyro/accel/airspeed resolution
Built-in 16MB data logger (The Black Box)
Piano DIP switch for servo reverse or user customizable
Built-in FTDI, making the board native USB
Dedicated modem/OSD port
I2C port with incoming "daisy chain board" allowing you to build sensor arrays
Two user-programmable buttons (one momentary, the other slide)
10-Bit analog expansion ports
Reset button
Optional "through hole" voltage dividers
Tons of status LEDs
New vibration resistance Invensense gyros (triple axis)
Analog Devices ADX330 accelerometer
AirSpeed sensor port (optional, sold separately)
Absolute Bosch pressure sensor and temp for accurate altitude

Esquemático es:

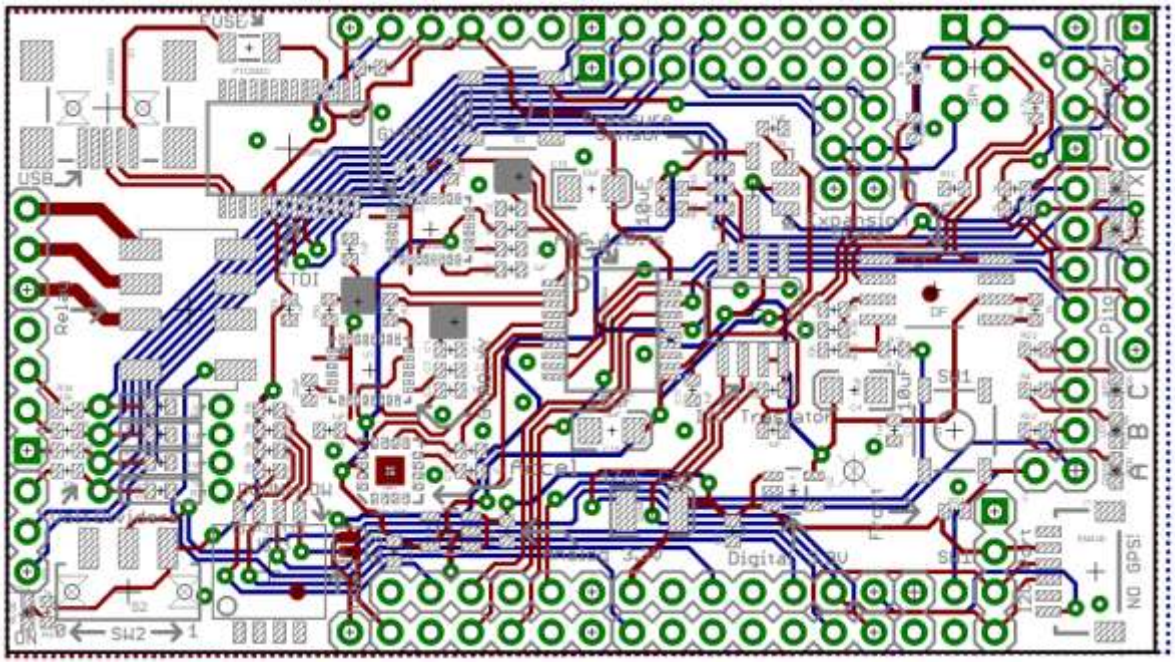



3.3V
R42
10k
R43
10k
GND
XCLR
SDA
SDA1
PC7
EOC
SCL
SCL1
VDDD
3V3
VDDA
NC
.1uF
C24
3.3V
5V
R8
200k
IC2
GND
EN
VREF1
VREF2
SCL1
SCL2
SDA1
SDA2
PD0
PD1
.1uF
C22
5V
R5
1k
LED1
BLUE
PC1
R4
1k
LED2
YELLOW
PC0
R22
1k
LED5
GREEN
PC2
R23
1k
LED6
RED
JP5
CH7
JP23
PG0
JP21
PF6
JP22
PF7
U$7
MISO
SCK
RST
+5
MOSI
GND
SPI
USB
VBUS
D-
D+
F1
PTCSMD
R9
3.9k
R34
10k
PF0
10k
R35
JP6
R36
3.9k
R28
10k
PF1
10k
R29
R30
10k
PF2
10k
R31
R37
3.9k
R38
3.9k
R32
10k
PF3
10k
R33
JP16
PF4
PF5


3.3V
VDD
VDD
XAGC
YAGC
CPOUT
GND
GND
GND
GND
X4.5IN
XOUT
X4.5OUT
Y4.5IN
YOUT
Y4.5OUT
PTATS
AZ
VREF
750
R6
R1
XOUT
YOUT
JP20
JP25
C25
.1uF
C1
.1uF
C6
.22uF
C7
.22uF
C5
.1uF
TEMP
GND
5V
10k
R20
R25
R26
R27
PL7
PL6
PE6
PE7
CHP-041TB
ISZ500SMD
ZAXIS
ZOUT
JP27
C9
.22uF
C8
.1uF
C2
.1uF
R3
R7
C26
.1uF
PC6
IC1
USBDM
USBDP
VCC
3V3OUT
VCCIO
OSCI
OSCO
RESET
TEST
AGND
GND7
GND18
GND21
TXD
RXD
CTS
RTS
DTR
DSR
DCD
RI
TXLED
RXLED
PWREN
TXDEN
SLEEP
FT232RLSSOP
D-
D+
1k
R2
PE0
PE1
#RST
LED3
ORANGE
LED4
ORANGE
R10
R21
1uF
C23
JP24
PD0
PD1
JP19
M01PTH
IC5
ZOUT
XOUT
YOUT
TEMP
AX
AY
AZ
CH7
CH0
CH1
CH2
CH3
CH4
CH5
CH6
CH7
VCC
VCC
SHDN
DCLK
CS
DI
BUSY
DOUT
VREF
COM
GND
GND
PH2
PC4
PH1
PC3
PH0
C19
10uF
C17
.1uF
C18
10uF
ADS7844



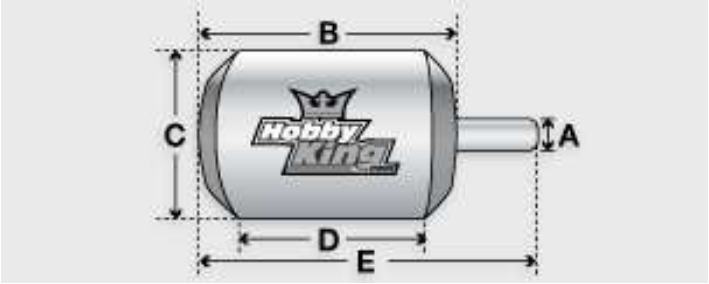
Y en la PCB queda de esta manera:



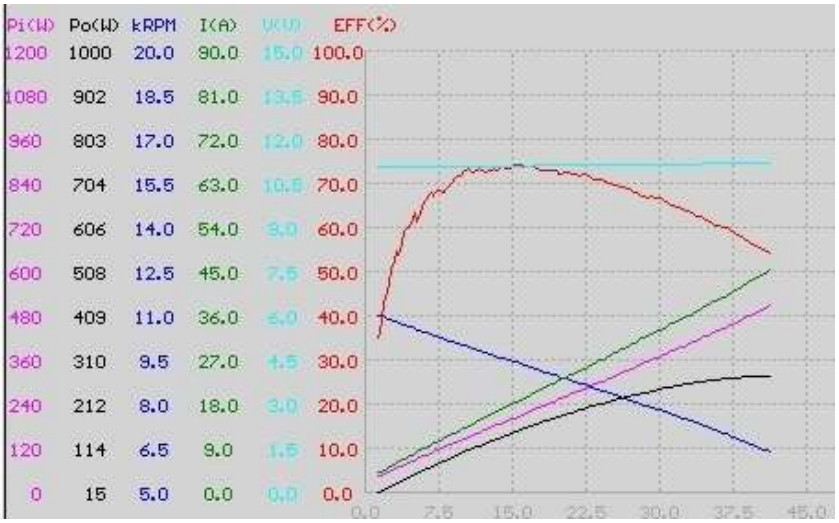
Anexo C: Características de los motores según el fabricante

Model: NTM Prop Drive 35-36A 910
Kv: 910rpm/v
Turns: 10T
Max current: 38A
Rated Power: 350w
Shaft: 4mm
Weight: 117g
Bolt holes: 18.9mm & 25mm
Bolt thread: M3
ESC: 30A
Cell count: 3S Lipoly
Suggested Prop: 10x4
Efficiency: 83%

Kv (rpm/v)	910
Weight (g)	117
Max Current (A)	35
Resistance (mh)	0
Max Voltage (V)	11
Power(W)	350
Shaft A (mm)	4
Length B (mm)	38
Diameter C (mm)	35
Can Length D (mm)	23
Total Length E (mm)	58



3536 910Kv	
Stator Diameter (mm)	¢27.5
Stator thickness (mm)	19
No. of stator arms	12
No. of Magnet Poles	10
Motor Wind (turn)	10T
Motor Wire (mm)	0.22*11
Motor Kv	910
No-Load Current (Io/10V)	1.0 (11.3V)
Max Continuous Current (A)	20A
Max Continuous Power (W)	350W
Weight (G)	121g
Outside Diameter (mm)	35
Shaft Diameter (mm)	4
Body Length (mm)	37
Overall Shaft Length (mm)	59.5



Anexo D: Características de los variadores de velocidad

Se han usado dos variadores debido a que durante la realización del proyecto, el modelo "Sentry" quedó descatalogado y sustituido por la versión "Plush". Como se puede ver las características son muy similares.

Turnigy Plush 40 A
Cont. Current: 40A
Burst Current (<10s): 55A
BEC Mode: Linear
BEC: 5v / 3A
Lipo Cells: 2-6
NiMH: 5-18
Weight: 33g
Size: 55x28x13mm

Turnigy Sentry 40A
Cont. Current: 40A
Burst Current (<10s): 55A
BEC Mode: Switch
BEC: 5v / 3A
Lipo Cells: 2-6
NiMH: 5-18
Weight: 40g
Size: 55x28x15mm
El diagrama de conexionado según fabricante se puede ver en la figura.

Anexo E: Código de los controladores, PID y PI.

```
//Valores de las constantes:
float KpP=50,KdP=400,KiP=0.1;    //PITCH    //50  400 0.1

float KpR=50,KdR=300,KiR=0.1;     //Roll    //60 300 0.1

//PITCH
float pitchSum=0,lastPitchFilter=0,pitchFilter=0;
float giroPitchFilter=0,lastGiroPitchFilter=0;

//Roll
float rollSum=0,lastRoll=0,lastRollFilter=0,rollFilter=0;
float giroRollFilter=0,lastGiroRollFilter=0;
float errorPitch=0,errorRoll=0;
//YAW
float compas=0;  int kyaw=5; int yawi=0;


//Calculamos la señal de los motores a través de la siguiente ecuación:
```

$$//Señal = K_p \cdot error + K_i \cdot \int_{t_0}^{t} error(\tau) \cdot d\tau \; + K_d \cdot \frac{d\, error(t)}{dt}$$

```
pitch    =  accel.y;
roll     =  accel.x;
giroPitch=  gyro.x;
giroRoll = gyro.y;

//A continuación se filtra la señal y las variables filtradas pasan a
nombrarse:
Pitch     → pitchFilter
Roll      → rollFilter
giroPitch → giroPitchFilter
giroRoll  →giroRollFilter


pitchSum +=errorPitch;             //integral
rollSum  += errorRoll;             //integral


errorPitch= yref - pitchFilter;
errorRoll = xref - rollFilter;

//Calculamos el valor para cada uno de los 6 motores.
valor[i] = throttle +(KpR*errorRoll -KdR*giroRollFilter + KiR*rollSum) +
                   +(KpP*errorPitch +KdP*giroPitchFilter + KiP*pitchSum);

//Control del YAW →usamos un controlador Proporcional Integral.
```

```
Yaw = compas - yawref;
yawi=+yaw; //integral

valor[i] +=kyaw*yaw +yawi;

//actualizamos los valores pitch para el controlador
lastPitch = pitch;
lastPitchFilter = pitchFilter;
lastRoll = roll;
lastRollFilter = rollFilter;
```

Anexo F: Código para la escritura del datalog en la memoria flash.

```
#include <DataFlash.h>
#include <I2C.h>
#define HEAD_BYTE1 0xA3
#define HEAD_BYTE2 0x95
#define END_BYTE1 0xA2
#define END_BYTE2 0x4E
#define DF_SLAVESELECT 53

DataFlash_APM1 DataFlash;

long timer=0,timer1=0;

void setup(void)
{
Serial.begin(115200);

menu();         //imprime por menu las opciones para leer o borrar la
memoria del log
I2c.begin();
I2c.timeOut(20);
DataFlash.Init();
DataFlash.StartWrite(1); //empezamos a escribir en la página nº1

funcion_serial();

timer1=micros();

   . . .
  }

void loop(void)
{


escribe_data();


static void escribe_data()
{
  DataFlash.WriteByte(HEAD_BYTE1);
  DataFlash.WriteByte(HEAD_BYTE2);
  DataFlash.WriteInt(rollFilter*10000);        //rollFIlter
  DataFlash.WriteInt(pitchFilter*10000);       //pitch filter
  DataFlash.WriteInt(giroRollFilter*10000);    //gyroRoll
  DataFlash.WriteInt(gyro.y*10000);            //gyroPitch
  DataFlash.WriteLong((long)(millis()-timer)); //time
  DataFlash.WriteInt(valor[1]);     //velocidades motores
  DataFlash.WriteInt(valor[2]);    //motor 1 y 2 para el roll
```

```
  DataFlash.WriteInt(valor[4]);    //motor 4 y 5 para el pitch
  DataFlash.WriteInt(valor[5]);
  DataFlash.WriteInt(xref*100);     //x ref
  DataFlash.WriteInt(yref*100);     //yref
  DataFlash.WriteInt(compas*100);
  DataFlash.WriteInt(throttle);     //throttle
  DataFlash.WriteByte(END_BYTE1);   //2 bytes para los digitos de control
  DataFlash.WriteByte(END_BYTE2);
}

static void funcion_serial()
{
  if(Serial.available()>0){
    byte recived=Serial.read();
    if(recived==76||recived==108){ // L ó l

//LEER
 int  i, tmp_int;
 byte  tmp_byte1, tmp_byte2;
 long  tmp_long;
 Serial.println("empezando a leer...");

 DataFlash.StartRead(1);  // Empezamos a leer desde la primera página

 for(int i;i<10000;i++){  // leemos los primeros 10000 paquetes de datos

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

  if ((tmp_byte1 == HEAD_BYTE1) && (tmp_byte1 == HEAD_BYTE1)){
   // Leer los ints...
   tmp_int = DataFlash.ReadInt();      //*10 000
//Cuando encuentra un espacio significa que hemos llegado al final del
log

if(isSpace(tmp_int)) {
tmp_int = DataFlash.ReadInt();
 if(isSpace(tmp_int)) {
      Serial.println("espacio encontrado");
                break;
    }}



   Serial.print(tmp_int);
   Serial.print(",");
   tmp_int = DataFlash.ReadInt();
   Serial.print(tmp_int);
```

```
  Serial.print(",");
  tmp_int = DataFlash.ReadInt();
  Serial.print(tmp_int);
  Serial.print(",");
  tmp_int = DataFlash.ReadInt();
  Serial.print(tmp_int);
  Serial.print(",");

  // lee 2 longs...
  tmp_long = DataFlash.ReadLong();
  Serial.print(tmp_long);
  Serial.print(",");

  // lee 6 ints...
  tmp_int = DataFlash.ReadInt();  //motores
  Serial.print(tmp_int);
  Serial.print(",");
  tmp_int = DataFlash.ReadInt();
  Serial.print(tmp_int);
  Serial.print(",");
  tmp_int = DataFlash.ReadInt();
  Serial.print(tmp_int);
  Serial.print(",");
  tmp_int = DataFlash.ReadInt();
  Serial.print(tmp_int);
  Serial.print(",");
  tmp_int = DataFlash.ReadInt();   //xref *100
  Serial.print(tmp_int);
  Serial.print(",");
  tmp_int = DataFlash.ReadInt();   //yref *100
  Serial.print(tmp_int);
  Serial.print(",");
  tmp_int = DataFlash.ReadInt();   //COMPAS *100
  Serial.print(tmp_int);
  Serial.print(",");
  tmp_int = DataFlash.ReadInt();  //throttle
  Serial.print(tmp_int);


  // Lee el los digitos de control...
  tmp_byte1 = DataFlash.ReadByte();
  tmp_byte2 = DataFlash.ReadByte();
 }
 Serial.println();
}
Serial.println("");
Serial.println("lectura completada ");
Serial.println("");
```

```
      while(1); //Cuando leemos el Log, finaliza el programa aquí
    }
    else if(recived==98||recived==66){ //B OR b

    //DELETE //borramos la memoria después de cada lectura
    borrar();
    while(1);
    }
  }

}
void dataflash_CS_inactivee()
{
  digitalWrite(DF_SLAVESELECT,HIGH); //disable device
}

void dataflash_CS_activee()
{
  digitalWrite(DF_SLAVESELECT,LOW); //enable device
}

static void borrar()  //Función que borra toda la memoria Flash
{
  dataflash_CS_activee();
 SPI.transfer(0xC7);
 SPI.transfer(0x94);
 SPI.transfer(0x80);
 SPI.transfer(0x9A);
   dataflash_CS_inactivee();
   Serial.println("Borrado de memoria Flash completado");
}
  static void menu() //Muestra por pantalla
  {
    Serial.println("Bienvenido;\n \t\t\t pulse.... \n 'L' para Leer datos
\n 'B' para Borrar datos\n \t\t\t\t\t y despues INTRO");
  }
```

Anexo G: Código usado para el filtro.

```
//FILTER pitch
        pitchFilter=lastPitchFilter*0.9048 + 0.09516*lastPitch;
//filter Roll
        rollFilter=lastRollFilter*0.9048 + 0.09516*lastRoll;

giroRollFilter=lastGiroRollFilter*0.00253 + 0.9975  *lastGiroRoll;    //G
=tf([1],[1.5 1]);c2d(G,0.0038)
giroPitchFilter=giroPitch;

//actualizamos los valores pitch para el filtro
lastPitch=pitch;
lastPitchFilter=pitchFilter;

lastRoll=roll;
lastRollFilter=rollFilter;

lastGiroRoll=giroRoll;
lastGiroRollFilter=giroRollFilter;
```

Anexo H: Código para la lectura de la bacteria

```
#define BATTERY_PIN 0
#define BATTERY_MIN 3.4 //mínimo voltaje para una celda LIPO

if(analogRead(BATTERY_PIN)*5/1024 >BATTERY_MIN) {

   PORTL ^= B00000100;   //pin donde está conectado el relé
  }
```

Anexo I: Código usado para leer valores por la radio

```
#include <APM_RC.h> // ArduPilot Mega RC Library
#define CORTE 1200

APM_RC_APM1 APM_RC;

//Leemos las señales de referencia por la radio
//pitch canal 2
yref=(APM_RC.InputCh(CH_2)-1050);
yref/=210;
yref-=2;
yref=constrain(yref,-1,1);

//Roll canal 2
//xref=(APM_RC.InputCh(CH_2)-1050);
xref/=210;
xref-=2;
constrain(xref,-0.5,0.5);

//Yaw por el canal 4
yaw=APM_RC.InputCh(CH_4)-1050;
yaw/=100;
yaw-=4;

//El acelerador por el canal 3
if(APM_RC.InputCh(CH_3)>CORTE){
throttle = APM_RC.InputCh(CH_3);
}
```

Anexo J: Código completo

```
#include <APM_RC.h>              // ArduPilot Mega RC Library Radio
#include <FastSerial.h>
#include <SPI.h>
#include <Arduino_Mega_ISR_Registry.h>
#include <AP_PeriodicProcess.h>
#include <AP_InertialSensor.h>
#include <AP_IMU.h>
#include <AP_ADC.h>
#include <AP_Math.h>
#include <AP_Common.h>
#include <DataFlash.h>
#include <AP_Compass.h>         // Compass Library
#include <I2C.h>
                          //para dataflash
#define HEAD_BYTE1 0xA3
#define HEAD_BYTE2 0x95
#define END_BYTE1 0xA2
#define END_BYTE2 0x4E
#define DF_SLAVESELECT 53
                       //convierte de radianes a grados  y viceversa
#define ToRad(x) (x*0.01745329252)  // *pi/180
#define ToDeg(x) (x*57.2957795131)  // *180/pi

#define BATTERY_PIN 0
#define BATTERY_MIN 3.4       //mínimo volaje para una celda LiPo

#define A_LED_PIN        35
#define C_LED_PIN        37
#define LED_ON           LOW
#define LED_OFF          HIGH

#define MIN 1235        //evitamos que los motores se paren
#define MAX 1500
#define CORTE 1200 //por debajo de este valor no actualizamos el throttle

Vector3f accel;   //viene de la libreria math.h y es un vector 3D
Vector3f gyro;
FastSerialPort(Serial, 0);

AP_Compass_HMC5843 compass;

Arduino_Mega_ISR_Registry isr_registry;
AP_TimerProcess  adc_scheduler;

APM_RC_APM1 APM_RC;
DataFlash_APM1 DataFlash;
////////////////////////////////////
```

```
                //processing
//int velocidadX=0,velocidadY=0;
//int anguloX=0,anguloY=0;
//int T,W,X,Y;
             //variables MOTORES
int valor1=0;
float throttle=MIN;
float valor[7],valorF[7];
float pitch,roll;
//Serial
int aux=0,aux2=0;
byte recived;
boolean estado=false,datos=true;
int8_t m_opp[7]={0,2,1,4,3,6,5};//matriz que devuelve el #motor contrario

  //CONTROLADOR
float KpP=50,KdP=400,KiP=0.1;    //PITCH    //50  400 0.1

float KpR=50,KdR=300,KiR=0.1;     //Roll    //60 300 0.1

//PITCH
float pitchSum=0,lastPitch=0,lastPitchFilter=0,pitchFilter=0;
float giroPitch=0, lastGiroPitch =0, giroPitchFilter=0,
lastGiroPitchFilter=0;

//Roll
float rollSum=0,lastRoll=0,lastRollFilter=0,rollFilter=0;
float giroRoll=0, lastGiroRoll=0, giroRollFilter=0, lastGiroRollFilter=0,
giroRoll2=0;    //giroRoll lo usamos para hacer la derivación numérica
float errorPitch=0, errorRoll=0;

//YAW
float compas=0;  int kyaw=5; int yawi=0;

//DATALOG
long timer=0,timer1=0;

//INPUTs
float xref=0,yref=0,yaw=0,yawref=0;
//////////////////////////////////////////////////////////////////////

AP_ADC_ADS7844 adc;
AP_InertialSensor_Oilpan oilpan_ins(&adc);
AP_IMU_INS imu(&oilpan_ins,0);


static void flash_leds(bool on)
{
    digitalWrite(A_LED_PIN, on?LED_OFF:LED_ON);
```

```
    digitalWrite(C_LED_PIN, on?LED_ON:LED_OFF);
}

////////////////////// T I M E R S
static uint32_t  fastTimer;
static uint32_t  fast_loopTimer;
static byte    medium_loopCounter;
static uint32_t  fiftyhz_loopTimer;

static byte    slow_loopCounter;
static int16_t   superslow_loopCounter;
static byte    counter_one_herz;
////////////////////

void setup(void)
{
 Serial.begin(115200);
menu();//imprime las opciones para leer/borrar la memoria del log
  I2c.begin();      //comunicación I2C para comunicar con memoria flash
  I2c.timeOut(20);

   if (!compass.init()) {
   Serial.println("compass initialisation failed!");
   while (1) ;
  }

  isr_registry.init();
  adc_scheduler.init(&isr_registry);
  APM_RC.Init(&isr_registry);  // APM Radio inicialización
  APM_RC.enable_out(CH_1);
  APM_RC.enable_out(CH_2);
  APM_RC.enable_out(CH_3);
  APM_RC.enable_out(CH_4);
  APM_RC.enable_out(CH_5);
  APM_RC.enable_out(CH_6);
  APM_RC.enable_out(CH_7);
  APM_RC.enable_out(CH_8);

  APM_RC.OutputCh(1, valor1);
  APM_RC.OutputCh(2, valor1);
  APM_RC.OutputCh(3, valor1);
  APM_RC.OutputCh(4, valor1);
  APM_RC.OutputCh(5, valor1);
  APM_RC.OutputCh(6, valor1);

DataFlash.Init();
DataFlash.StartWrite(1); //empezamos a escribir en la página n°1

```

```
146 imu.init(IMU::COLD_START, delay, flash_leds, &adc_scheduler);
147 imu.init_accel(delay, flash_leds);Serial.print("\n");
148
149 funcion_serial();
150 fast_loopTimer=micros();timer1=micros();
151 compass.set_orientation(AP_COMPASS_COMPONENTS_DOWN_PINS_FORWARD);  // set
152 compass's orientation on aircraft.
153 compass.set_offsets(0,0,0);  // set offsets to account for surrounding
154 interference
155 compass.set_declination(ToRad(0.23));  // set local difference between
156 magnetic north and true north para cartagena/murcia la declinación es
157 0.28
158 }
159
160 void loop(void)
161 {
162 int32_t timer    = micros();
163
164 if ((timer - fast_loopTimer) >= 10000) {    //400 Hz
165
166       fast_loop();
167       fast_loopTimer=micros();
168   }
169 if ((timer - fiftyhz_loopTimer) >= 20000) {
170   fiftyhz_loopTimer = timer;
171
172   medium_loop();    //10 Hz
173
174   fiftyhz_loop();    //50 Hz
175
176   counter_one_herz++;
177
178   if(counter_one_herz == 50){
179    super_slow_loop();    //1 Hz
180    counter_one_herz = 0;
181   }
182                 }
183 }//cierra el void loop
184
185
186
187
188
189 // F U N C I O N E S
190
191 static void fast_loop()
192 {
193  if(estado){
194                 imu.update();
```

```
                accel = imu.get_accel();
        gyro =  imu.get_gyro();
        pitch    =  accel.y;
        roll     =  accel.x;
        giroPitch=  gyro.x;
        giroRoll = gyro.y;


 //FILTER pitch
pitchFilter=lastPitchFilter*0.9048 + 0.09516*lastPitch; //giroRoll2 =
((pitchFilter-lastPitchFilter)*100000)/(micros()-timer1);
timer1=micros(); derivacion numérica
//giroPitchFilter=lastGiroPitchFilter*0.6703 + 0.3297  *lastGiroPitch;

//filter Roll
rollFilter=lastRollFilter*0.9048 + 0.09516*lastRoll;
//giroRollFilter=lastGiroRollFilter*0.07318 + 0.9268  *lastGiroRoll;
//filtro t=0.0038,  G=tf([1],[0.05 1]);c2d(G,0.0038) ensayo d9_10
demasiado suave
//giroRollFilter=lastGiroRollFilter*0.141 + 0.859  *lastGiroRoll;   //
G=tf([1],[0.025 1]);c2d(G,0.0038
giroRollFilter=lastGiroRollFilter*0.00253 + 0.9975  *lastGiroRoll;    //G
=tf([1],[1.5 1]);c2d(G,0.0038)     parece que funciona
//Cuando no tenemos filtro
// rollFilter = roll;        //si activamos o desactivamos los filtros,
hay que activar el actualizar los valores abajo de la funcion
//  pitchFilter = pitch;

        giroPitchFilter=giroPitch;
        pitchSum +=errorPitch;             //integral

        //giroRollFilter=giroRoll;
        rollSum += errorRoll;             //integral
        errorPitch=yref-pitchFilter;
        errorRoll=xref-rollFilter;
```

```
//FRONT      RED
valor[1]=throttle+1.0*(KpR*errorRoll-KdR*giroRollFilter+KiR*rollSum);

valor[5]=throttle+0.5*(KpR*errorRoll-KdR*giroRollFilter+KiR*rollSum) -
0.866*(KpP*errorPitch +KdP*giroPitchFilter + KiP*pitchSum);

valor[4]=throttle+0.5*(KpR*errorRoll-KdR*giroRollFilter+KiR*rollSum)+
0.866*(KpP*errorPitch +KdP*giroPitchFilter + KiP*pitchSum);

//BACK  BLACK
valor[2]=throttle-1.0*(KpR*errorRoll-KdR*giroRollFilter+KiR*rollSum);

valor[3]=throttle-0.5*(KpR*errorRoll-KdR*giroRollFilter+KiR*rollSum)-
0.866*(KpP*errorPitch+KdP*giroPitchFilter+KiP*pitchSum);

valor[6]=throttle-0.5*(KpR*errorRoll-KdR*giroRollFilter+KiR*rollSum)
+0.866* (KpP*errorPitch +KdP*giroPitchFilter + KiP*pitchSum);

//     CONTROL DEL YAW
yaw=compas-yawref; yawi=+yaw;

valor[1] +=kyaw*yaw +yawi;
valor[3] +=kyaw*yaw +yawi;
valor[6] +=kyaw*yaw +yawi;

valor[2] -=kyaw*yaw -yawi;
valor[4] -=kyaw*yaw -yawi;
valor[5] -=kyaw*yaw -yawi;
////////////////////////////////////////////////////
escribe_data();         //////

for(int8_t m=1;m<=6;m++){    //TRIDGE STABILITY PATCH pag 49 hexamotors.
Recalcula las velocidades si alguna de ellas satura.
if(valor[m]>MAX){
valor[m_opp[m]]-=valor[m]-MAX;
valor[m]=MAX;
               }}


for(int8_t i=1;i<=6;i++){//CONSTRAIN --> limites superior e inferior
valor[i] = constrain(valor[i],MIN,MAX);
                         }
```

```
//FILTRO que frena la acelearcion contra la deceleracion
for(int8_t m=1;m<=6;m++){
if(valorF[m]<valorF[m]){   //es porque estamos acelerando
    valorF[m]=(valor[m]+valorF[m])/2;
                } else{                     //no filtramos
          valorF[m]=valor[m];
                                                }}
                                        */

actualiza_motores();

//actualizamos los valores pitch para el filtro
lastPitch=pitch;
//lastGiroPitch=giroPitch;
lastPitchFilter=pitchFilter;
//lastGiroPitchFilter=giroPitchFilter;

lastRoll=roll;
lastGiroRoll=giroRoll;
lastRollFilter=rollFilter;
lastGiroRollFilter=giroRollFilter;


}//cerramos if(estado)

else{   funcion_serial();      //Si el throttle está a cero -->
imprimimos VELOCIDAD CERO a los motores! y calibramos el PID
  for(int8_t i=1;i<=6;i++){
             çAPM_RC.OutputCh(i, 0);
                                }
                }
            /*
if(estado){//imprime datos por Serial.Monitor (ahora ya usamos FLASH)
Serial.printf(" %4.4f , %4.4f , %4.4f , %4.4f , %4.4f , %4.4f ,",
       accel.x, accel.y, rollFilter, gyro.x, gyro.y, pitchFilter);Serial.
print(millis());
          Serial.print(",");Serial.print(valor[1]);
          Serial.print(",");Serial.print(valor[2]);
          Serial.print(",");Serial.print(valor[3]);
          Serial.print(",");Serial.print(valor[4]);
          Serial.print(",");Serial.print(valor[5]);
          Serial.print(",");Serial.print(valor[6]);
          Serial.print(",");Serial.print(xref);
          Serial.print(",");Serial.print(yref);
          Serial.print(",");Serial.print(throttle);
          Serial.print(",");Serial.println(yaw);
          }*/

} //cierra fast_loop
```

```
static void medium_loop()
{
  switch(medium_loopCounter) {

  // Tenemos en cuenta el Compass
  //---------------------------------------
  case 0:
   medium_loopCounter++;
         compass.read();
         compass.calculate(0,0);
         compas=ToDeg(compass.heading);
//imprimimos por puerto serial para DEBUG MODE
         Serial.print(compas);
         Serial.print(",   ");Serial.print(yawref);
         Serial.print(",   ");Serial.println(yaw);

                     break;

  // Leemos valores de las referencias por radio
  //-------------------------------------------------
  case 1:
   medium_loopCounter++;
    yref=(APM_RC.InputCh(CH_2)-1050);
    yref/=210;
    yref-=2;
    yref=constrain(yref,-1,1);    //pitch

    //xref=(APM_RC.InputCh(CH_2)-1050);
    xref/=210;
    xref-=2;´
    constrain(xref,-0.5,0.5);    //roll

   //yaw=APM_RC.InputCh(CH_4)-1050;
     yaw/=100;
     yaw-=4;

   break;


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


   break;


  //-------------------------------------------------
```

```
  case 3:
   medium_loopCounter++;

   break;

  //---------------------------------
  case 4:
   medium_loopCounter = 0;

   break;

  default:
   // ------------------------

   medium_loopCounter = 0;

   break;
 }

}//fin medium_loop


static void fiftyhz_loop(){   //leer datos del acelerador

   if(APM_RC.InputCh(CH_3)>CORTE){
    // throttle = APM_RC.InputCh(CH_3);       //acelerador variable
       throttle=1280;                          //acelerador constante
         estado=true;
       escribe_data(); //ESCRIBIMOS DATOS CADA 50HZ (20ms()
     }else{estado=false;timer=millis();}

      }

static void super_slow_loop()           //loop de un Herz
{

if(analogRead(BATTERY_PIN)*5/1024 >BATTERY_MIN) {

   PORTL ^= B00000100;   //pin donde está conectado el relé
  }

}

void actualiza_motores()
{
for(int8_t i=1;i<=6;i++){    //el throttle está funcionando --> IMPRIMIR
.                              las VELOCIDADES a los motores
APM_RC.OutputCh(i,valor[i]);

```

```
                                          }
}

static void escribe_data()
{
  DataFlash.WriteByte(HEAD_BYTE1);
  DataFlash.WriteByte(HEAD_BYTE2);
  DataFlash.WriteInt(rollFilter*10000);            //rollFIlter
  DataFlash.WriteInt(pitchFilter*10000);            //pitch filter
  DataFlash.WriteInt(giroRollFilter*10000);            //gyroRoll
  DataFlash.WriteInt(gyro.y*10000);            //gyroPitch
  DataFlash.WriteLong((long)(millis()-timer));      //time
  DataFlash.WriteInt(valor[1]);         //velocidades motores
  DataFlash.WriteInt(valor[2]);    //motor 1 y 2 para el roll
  DataFlash.WriteInt(valor[4]);    //motor 4 y 5 para el pitch
  DataFlash.WriteInt(valor[5]);
  DataFlash.WriteInt(xref*100);              //x ref
  DataFlash.WriteInt(yref*100);              //yref
  DataFlash.WriteInt(compas*100);
  DataFlash.WriteInt(throttle);          //throttle
  DataFlash.WriteByte(END_BYTE1);  // 2 bytes de control
  DataFlash.WriteByte(END_BYTE2);



}
static void funcion_serial()
{
  if(Serial.available()>0){
    byte recived=Serial.read();
    if(recived==76||recived==108){ // L OR l

      //LEER
 int  i, tmp_int;
 byte  tmp_byte1, tmp_byte2;
 long  tmp_long;
 Serial.println("empezando a leer...");

 DataFlash.StartRead(1);  // empezamos a leer por página 1

 for(int i;i<10000;i++){  // leemos 1000 packets...

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

  if ((tmp_byte1 == HEAD_BYTE1) && (tmp_byte1 == HEAD_BYTE1)){
   // leemos 4 ints...
   tmp_int = DataFlash.ReadInt();      //*10 000

```

```
471 if(isSpace(tmp_int)) {
472   tmp_int = DataFlash.ReadInt(); if(isSpace(tmp_int)) {
473     Serial.println("espacio encontrado");
474                 break;
475     }}
476    Serial.print(tmp_int);
477    Serial.print(",");
478    tmp_int = DataFlash.ReadInt();
479    Serial.print(tmp_int);
480    Serial.print(",");
481    tmp_int = DataFlash.ReadInt();
482    Serial.print(tmp_int);
483    Serial.print(",");
484    tmp_int = DataFlash.ReadInt();
485    Serial.print(tmp_int);
486    Serial.print(",");
487
488    // leemos 2 longs...
489    tmp_long = DataFlash.ReadLong();
490    Serial.print(tmp_long);
491    Serial.print(",");
492
493    // leemos 6 ints...
494    tmp_int = DataFlash.ReadInt();  //motores
495    Serial.print(tmp_int);
496    Serial.print(",");
497    tmp_int = DataFlash.ReadInt();
498    Serial.print(tmp_int);
499    Serial.print(",");
500    tmp_int = DataFlash.ReadInt();
501    Serial.print(tmp_int);
502    Serial.print(",");
503    tmp_int = DataFlash.ReadInt();
504    Serial.print(tmp_int);
505    Serial.print(",");
506    tmp_int = DataFlash.ReadInt();   //xref *100
507    Serial.print(tmp_int);
508    Serial.print(",");
509    tmp_int = DataFlash.ReadInt();   //yref *100
510    Serial.print(tmp_int);
511    Serial.print(",");
512    tmp_int = DataFlash.ReadInt();   //COMPAS *100
513    Serial.print(tmp_int);
514    Serial.print(",");
515    tmp_int = DataFlash.ReadInt();  //throttle
516    Serial.print(tmp_int);
517
518
519    // leemos los bytes de control...
```

```
520     tmp_byte1 = DataFlash.ReadByte();
521     tmp_byte2 = DataFlash.ReadByte();
522    }
523    Serial.println();
524   }
525   Serial.println("");
526   Serial.println("lectura completada ");
527   Serial.println("");
528
529       while(1);
530     }
531     else if(recived==98||recived==66){ //B OR b
532
533     //DELETE
534     borrar();
535     while(1);
536     }
537    }
538
539  }
540  void dataflash_CS_inactivee()
541  {
542    digitalWrite(DF_SLAVESELECT,HIGH); //disable memoria flash
543  }
544
545  void dataflash_CS_activee()
546  {
547    digitalWrite(DF_SLAVESELECT,LOW); //enable memoria flash
548  }
549
550
```

```
551  static void borrar()
552  {
553   dataflash_CS_activee();
554   SPI.transfer(0xC7);
555   SPI.transfer(0x94);
556   SPI.transfer(0x80);
557   SPI.transfer(0x9A);
558   dataflash_CS_inactivee();
559   Serial.println("Borrado de memoria Flash completado");
560  }
561
562
563  static void menu()
564    {
565      Serial.println("Bienvenido;\n \t\t\t pulse.... \n 'L' para Leer datos
566  \n 'B' para Borrar datos\n \t\t\t\t\t y despues INTRO");
567    }
```