UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DISEÑO E IMPLEMENTACIÓN DE UN QUADCOPTER BASADO EN MICROCONTROLADOR ARDUINO

AUTOR: VÍCTOR RAMOS VICEDO

TUTOR: RAÚL ESTEVE BOSCH

Curso Académico: 2013-14

Índice del trabajo

MEMORIA

PRESUPUESTO

PLANOS

MEMORIA

Introducción y objetivos

Desde hace unos años, y propiciados por el continuo desarrollo de la electrónica, se han puesto de moda unos aparatos vanguardistas un tanto peculiares: los ***V**ehículos **A**éreos no **T**ripulados*. **VANT** por sus siglas en español, **UAV/UAS** por sus siglas en inglés ***U**nmanned **A**erial **V**ehicle/**U**nmanned **A**erial **S**ystem* o comúnmente conocidos como **DRONES***.*

Varios nombres para definir un mismo concepto: Un vehículo aéreo que no tiene piloto o, de tenerlo, no está en el interior del aparato. La habilidad y los sentidos del aeronauta han sido sustituidos por sensores electrónicos de gran precisión, que consiguen una maniobrabilidad prácticamente perfecta, como son los acelerómetros o los giroscopios, de los que hablaremos más adelante.

Al principio, estos Drones estaban pilotados por una persona desde un lugar remoto, teniendo acceso a los mismos datos de aviación que tendría de estar físicamente en el dispositivo, pero con la ventaja de estar lejos en caso de que corriese algún peligro. Actualmente este sistema de control se sigue manteniendo sobretodo en el sector militar, pero va proliferando más el control autónomo del aparato propiciado por el avance en la tecnología GPS (*Global Positioning System)*. Así un Drone puede despegar, realizar su misión y aterrizar, todo sin intervención humana.

Entre los diversos tipos de estos dispositivos destaca uno por su excepcional maniobrabilidad y variedad de entornos de uso, el **Quadcopter**, que prácticamente es conocido en todo el mundo también por su posibilidad de construcción casera. Básicamente es un helicóptero con 4 motores equidistantes horizontalmente del centro.

Los objetivos del presente proyecto son, en primer lugar, diseñar un chasis apto para el montaje de un Quadcopter con el programa de diseño 3D Autodesk Inventor®, buscando y seleccionando los componentes óptimos que necesita, y en segundo lugar, implementar el código necesario para la comunicación y el control de estabilidad del aparato mediante la plataforma ARDUINO con un dispositivo a distancia.

Este proyecto pretende ser una especie de guía de fabricación para tener las pautas de cómo fabricar un quadcopter, explicando los pasos y ensayos que se han efectuado para una propuesta con unos componentes determinados, y no la fabricación de uno físicamente, por el alto coste que tiene. Por esta razón se diseñará y fabricará una especie de balancín para poder efectuar ensayos de los componentes, y, en un futuro para ajustar el PID necesario sin necesidad de hacer pruebas con el quadcopter.

La memoria va a estar dividida en 4 capítulos.

En el primero se introducirá el concepto de quadcopter y se explicará detalladamente su funcionamiento y los componentes necesarios para su fabricación, presentando varias opciones en cada uno explicando sus ventajas e inconvenientes.

El segundo tratará del dimensionamiento del chasis y la selección de componentes justificadamente, para más tarde, con las características físicas de esos componentes dimensionar el sistema de propulsión necesario para el movimiento.

Durante el tercer capítulo se desarrollará todo el código de programación que necesita el quadcopter, tanto el código de control de estabilidad y respuesta del microcontrolador como el

de comunicación por bluetooth, y además se diseñará y fabricará un sistema basado en un balancín para estudiar posibles valores de constantes del regulador PID.

El 4º capítulo será un resumen/conclusión del proyecto realizado.

Índice

1) Vehículos aéreos no tripulados.

1.1) Definición

Un vehículo aéreo no tripulado (**VANT**, o **UAV** por sus siglas en inglés) es, como ya se indicó en la introducción, un aparato volador que no contiene un piloto en su interior, bien porque está siendo pilotado por control remoto o bien porque tiene en su programación todo lo necesario para llevar a cabo sus instrucciones sin intervención humana.

En un principio fueron diseñados para el sector militar, ya que se podía aprovechar al máximo el hecho de que no había ninguna persona físicamente en la aeronave. Así, se podían efectuar operaciones de entrada en espacios aéreos peligrosos sin temer más que por la suma de unos 30 millones de € que pueden alcanzar cada uno de estos aparatos. Además, a parte de la ventaja de la seguridad, tenemos que el volumen necesario que necesita el aparato es mucho menor que el que necesita uno que deba albergar una persona, lo que repercute también en una mejor maniobrabilidad.

No obstante no todo es perfecto, y tantas ventajas no podían esconder menos desventajas. A medida que avanzamos en la era tecnológica damos más autonomía a las máquinas para quitarnos trabajo a los humanos, lo que a priori parece perfecto, pero hay que tener en cuenta que aún tiene que pasar mucho tiempo hasta que una maquina pueda tener el mismo juicio que una persona. Si ya no es por ética, pensemos en la cantidad de *hackers*[1] que hay por el mundo, y si pueden conseguir entrar a los lugares más seguros de internet, ¿muy difícil les será acceder al control de aeronaves no tripuladas militares?

El VANT más antiguo del que se tiene constancia se desarrolló al final de la primera guerra mundial, y su utilidad era simplemente la de entrenar a los operarios de la artillería antiaérea. No obstante, hasta el final del siglo XX no empiezan a aparecer DRONES que pueden operar con total autonomía controlados sólo por radio.

Fig 1. General Atomics MQ-9 Reaper (Predator B)

[1] Un cracker es una persona que rompe algún sistema de seguridad informático. Comúnmente se les conoce como hackers, pero en realidad esto es una equivocación, pues los hackers se encargan de evaluar los sistemas de seguridad para protegerse contra los crackers.

1.2) Tipos

En cuanto a la clasificación de los UAV podemos efectuarla conforme a tres criterios: su misión, y su techo[2] y alcance máximo y su morfología.

En cuanto a su misión distinguimos los siguientes tipos:

1. **Blanco**: Como el primer UAV desarrollado al final de la primera guerra mundial, se utilizan para simular objetivos voladores.
2. **Reconocimiento**: Se encargan de obtener y enviar información militar, como por ejemplo imágenes aéreas de una base militar enemiga. En este tipo destacan los MUAV[3] en forma de avión o helicóptero.
3. **Combate (UCAV[4])**: Los sustitutos de los pilotos de combate en misiones que pueden resultar muy peligrosas.
4. **Logística**: Diseñados simplemente para llevar carga.
5. **Investigación y desarrollo**: Para probar los nuevos sistemas que están en investigación y desarrollo.
6. **UAV comerciales y civiles**: Los más conocidos y que se pueden conseguir fácilmente por internet. Para su uso civil y como entretenimiento (El UAV objeto de este proyecto se incluye en esta categoría).

En cuanto a su altura y alcance máximo tenemos los siguientes tipos:

Categoría	Acrónimo	Alcance(km)	Altitud máx.(m)	Carga máx.(kg)
Micro	Micro	<10	250	<5
Mini	Mini	<10	300	<30
Alcance cercano	CR	10-30	3000	150
Alcance corto	SR	30-70	3000	200
Alcance medio	MR	70-200	5000	1250
Altitud baja	LADP	>250	9000	350
Autonomía media	MRE	>500	8000	1250
Autonomía alta Altitud baja	LALE	>500	3000	<30
Autonomía media Altitud media	MALE	>500	14000	1500
Autonomía media Altitud alta	HALE	>2000	20000	12000

[2] Cuando hablamos del "Techo" de un vehículo aéreo nos referimos a la altitud máxima que puede alcanzar en vuelo. Cuando hablamos del "Techo" de un vehículo aéreo nos referimos a la altitud máxima que puede alcanzar en vuelo.
[3] *Micro Unmanned Aerial Vehicle*, o en español Micro vehículo aéreo no tripulado.
[4] *Unmanned Combat Air Vehicle*, o en español *Vehículo aéreo de combate no tripulado*.

Estratosférico	STRATO	>2000	30000	ND
EXO-Estratosférico	EXO	ND	>30000	ND

Tabla 1. Clasificación por altura y alcance

En cuanto a su morfología distinguimos 4 tipos:

1. **Helicópteros**: Una morfología mundialmente conocida, con un rotor en la parte superior y uno de cola para compensar el par del rotor que lo haría girar sin dar vueltas. Tiene una alta maniobrabilidad y puede quedarse en vuelo estacionario y volar verticalmente.
2. **Aviones**: También quien no sabe qué es un avión. Dos alas horizontales en torno al fuselaje que lo dotan de la sustentación necesaria al ir a una determinada velocidad por diferencia de presiones. Puede ir a altas velocidades y llevar cargas elevadas, pero no tiene la posibilidad del vuelo estacionario ni tiene tanta maniobrabilidad como un helicóptero.
3. **Dirigibles**: Mundialmente conocidos también durante la segunda guerra mundial, estos aparatos vuelan por un principio muy básico de diferencia de densidades. El helio que contienen es menos denso que el aire del exterior y por eso flotan. Luego para maniobrarlos es sencillo con un motor para cada eje de movimiento. No obstante la carga condiciona mucho su capacidad de vuelo y su velocidad y maniobrabilidad son muy precarias.
4. **Multirrotores**: Parecidos a los helicópteros, pero tienen varios rotores verticales en puntos equidistantes del centro horizontalmente, y variando las velocidades de giro de cada uno de los motores se consigue una maniobrabilidad sorprendente. Esto hace que sean las estrellas de los vuelos en interior, pero no son aptos para volar a grandes altitudes (El UAV objeto de este proyecto se incluye en esta categoría).

Fig 2. Avión RC

Fig 3. Helicóptero RC

Fig 4. Dirigible RC

Fig 5. Quadcopter

1.3) Multirrotores

Como ya indicamos en los apartados precedentes, el UAV objeto de este proyecto se incluye dentro del grupo de los multirrotores, clase que corresponde fundamentalmente a la clasificación Micro-Mini en cuanto a techo y alcance de vuelo.

Fig 6. Trirotor

Un multirrotor es un aparato cuya sustentación está producida por el giro de múltiples rotores equidistantes del centro geométrico del mismo, y con distintas combinaciones de velocidades de giro se consigue una amplia maniobrabilidad.

Dentro del grupo de los multirrotores se dividen varios grupos en función del número concreto de rotores, habiendo comúnmente desde trirotores hasta decarrotores. Obviamente, cuantos más motores más estabilidad y más fuerza de propulsión, lo que conlleva más capacidad de carga. No obstante la mayor parte de estos aparatos son de 4 rotores, incluido el que vamos a diseñar.

Fig 7. Decarotor

1.3.1) Dinámica

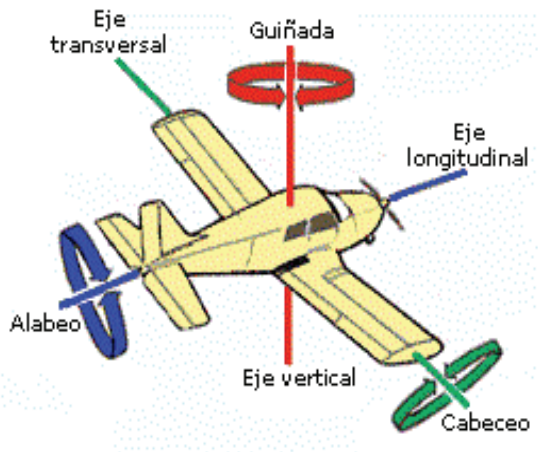


Fig 8. Giros aviación

En aviación hay 3 giros principales: El alabeo, el cabeceo y la guiñada. Con nuestro quadcopter pasa lo mismo, pero a diferencia de los aviones, que consiguen estos movimientos por la posición de alerones y timones, nuestro aparato los consigue sólo variando las velocidades de giro de los motores:

1. **Alabeo y cabeceo**. Se consiguen girando a distintas velocidades los pares opuestos de motores (motores 1-3 y 0-2 balanceo; 1,2 y 0-3 cabeceo) (Fig 10.)

Al girar varios motores con velocidades de giro distintas se descentra el eje de los mismos con respecto a la posición de equilibrio horizontal, en la que todos tienen sus ejes verticales, produciendo que el empuje ahora tenga una componente en la dirección hacia la que se han desviado. Veamos un ejemplo en 2 dimensiones:

Fig 9. Explicación alabeo/cabeceo

En reposo (Fig 9.1) los dos motores giran igual, y por tanto tienen el mismo empuje, que sólo se invierte en compensar la fuerza con la que la gravedad tira del aparato hacia el suelo. En el momento en el que el motor derecho gira más rápidamente que el izquierdo el sistema dinámico se desequilibra, produciendo que haya un mayor empuje a la derecha que a la izquierda del centro de gravedad, y originando un momento torsor que hace girar el aparato (Fig 9.2).

Esto hace que la resultante de la fuerza de los empujes no sea vertical y se compense con el peso, sino que hay una componente horizontal que tira del quadcopter hacia la izquierda hasta que se vuelvan a equilibrar los motores. Así, solo es necesario compensar el empuje para que el aparato no ascienda ni descienda y tenemos un desplazamiento lateral. Si aplicamos este supuesto a las 3 dimensiones con 4 motores tenemos el mismo resultado.

2. Guiñada. La guiñada es el giro con respecto al eje vertical del quadcopter. Se consigue por el principio de conservación del momento cinético al disponer 2 rotores en cada sentido de giro. En un quadcopter, y en general en cualquier multirrotor la mitad de los rotores giran en un sentido y la otra mitad en el otro. De esta forma, con todos los rotores girando a la misma velocidad se consigue equilibrar el momento cinético y mantener el aparato recto (Fig 10.1). Si no estuvieran así dispuestos empezaría a girar sobre sí mismo y sería imposible dirigirlo.

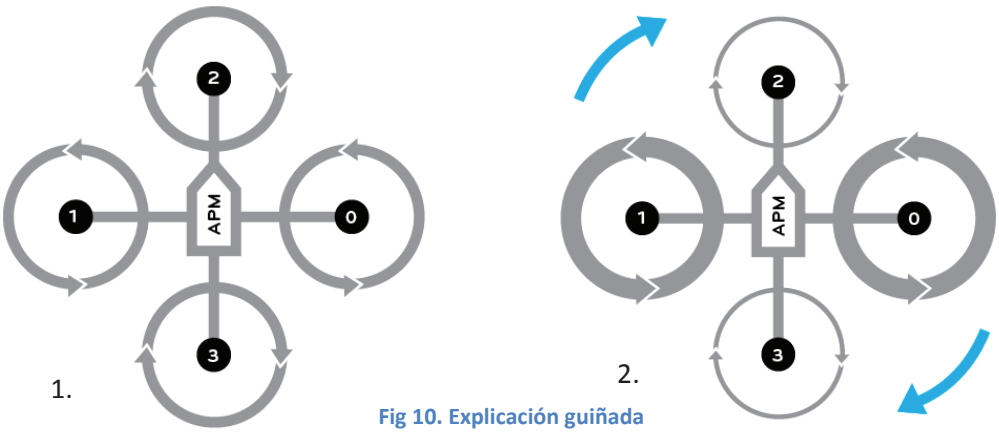


Fig 10. Explicación guiñada

Si lo que queremos es hacer "guiñar" el quadcopter, lo que tenemos que conseguir es que el par de motores que gira en un sentido gire más rápido que el que lo hace en el otro. Así el sumatorio de momentos no es 0 y hay un momento resultante en uno de los dos sentidos de giro. Ahí tenemos la guiñada (Fig 10.2). Aquí la solución está en decelerar el otro par de motores en la misma magnitud que se acelera el primero para no variar la resultante y que el aparato se mantenga suspendido en el aire sin variar su altitud.

Podemos advertir ya el complejo control de velocidades que hay que tener sobre los 4 motores para conseguir una estabilidad y maniobrabilidad perfecta, ya que la ventaja de controlarlo todo con sólo variar 4 valores de tensión al final se convierte en la desventaja de tener que tener una precisión del orden de milivoltios si no queremos pasar mucho tiempo en el taller de reparaciones.

1.3.2) Estructura y componentes

Ya hemos visto cómo consigue moverse nuestro quadcopter sólo con sus 4 motores. Ahora vamos a ver cómo se consigue obtener en cada momento el valor preciso de tensión que hay que enviar a cada motor para evitar acabar impactando contra el suelo accidentalmente.

Lo primero que tenemos que tener en cuenta es que la perfección no existe. Si tuviésemos que volar nuestro Drone en un espacio cerrado con el aire absolutamente quieto y sin ningún estímulo, y lo hubiésemos conseguido fabricar completamente simétrico respecto a sus dos planos de simetría centrales, y los motores fueran a responder de una forma perfectamente teórica, podríamos programar el procesador con una lista de combinaciones de velocidades de giro y nunca tendríamos ningún accidente. Sin embargo esto no es posible dado la gran cantidad de variables que intervienen en el proceso y la imperfección de los componentes, así que se debe diseñar un sistema de control que elabore una respuesta en función de la inclinación del aparato.

En este apartado vamos a describir la estructura funcional de nuestro quadcopter, los componentes físicos necesarios para formarlo y varias alternativas a cada uno con sus ventajas e inconvenientes.

1.3.2.1) Estructura funcional

Empecemos por lo importante: Los **motores**. Como hemos dicho, 4 motores con velocidades de giro independientes que controlar. ¿Quién las controla? La **placa procesadora**. Es sobre la que programamos el código que convertirá los datos de entrada en valores de tensión para los motores. Pero, ¿qué datos de entrada? En principio tenemos datos de entrada de dos orígenes distintos: Los **sensores dinámicos** y el **control manual**. Los sensores dinámicos serán los ojos que indiquen a nuestra placa la orientación espacial del quadrotor en todo momento, y el control manual será la interfaz con la persona que lo esté pilotando.

Ahora bien, esa persona no puede volar con el quadcopter (al menos con el nuestro), así que los datos tienen que ser enviados inalámbricamente mediante el **módulo de comunicaciones**. Toda esta electrónica la vamos a tener que alimentar con algo, que va a ser una **batería**. Y ya sólo nos queda algo con lo que unirlo todo, sujetarlo y evitar que se rompa al primer golpe: el **chasis**.

Con esto ya tenemos nuestro quadcopter. Todavía falta escoger cada componente particularmente, pero a groso modo no hay más. Sólo estos 7.

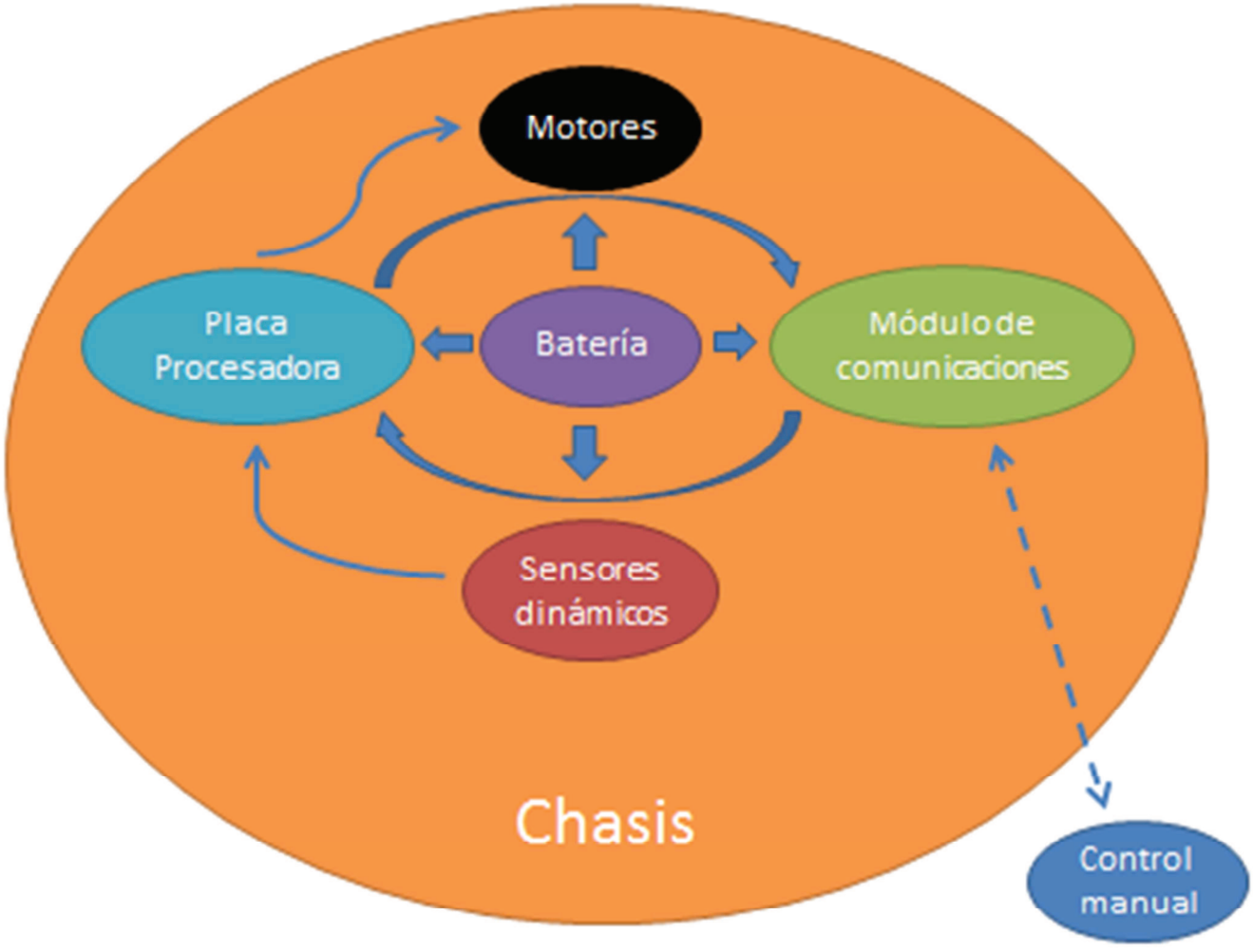


Fig 11. Esquema funcional

1.3.2.2) Descripción de componentes

Ya tenemos claros los componentes que componen nuestro aparato. Ahora vamos a describir cada uno de ellos y a ver distintas alternativas en cuanto a modelos comerciales.

a) **Motores**:

Los motores son el aparato locomotor de nuestro quadcopter. Son los únicos actuadores del sistema para conseguir todos los movimientos, y de su buen dimensionamiento depende que el aparato pueda volar o se quede en el suelo, y en el primer caso, la agilidad y capacidad de carga que tenga.

Hay muchos tipos de motores: eléctricos, térmicos, a reacción, turbinas de vapor y muchos puntos suspensivos. Pero evidentemente, para nuestro sistema ligero con un bajo consumo de energía, la mejor opción son los motores eléctricos.

Un motor eléctrico transforma la energía eléctrica en energía mecánica en forma de par de torsión. Su funcionamiento es muy sencillo: La corriente eléctrica circula por un bobinado de hilo de cobre enrollado en la parte móvil del motor llamada rotor, que está conectado al eje motriz, produciendo un campo magnético cuya polaridad se opone a la de un imán permanente situado en la carcasa que hace de estator, generando una fuerza de repulsión entre las parejas de polos que hace que el eje gire. Cuando cada polo del rotor se está acercando a su opuesto en el estator un sistema de escobillas cambia la polaridad del rotor, volviendo a generar el empuje que lo mantiene girando indefinidamente.

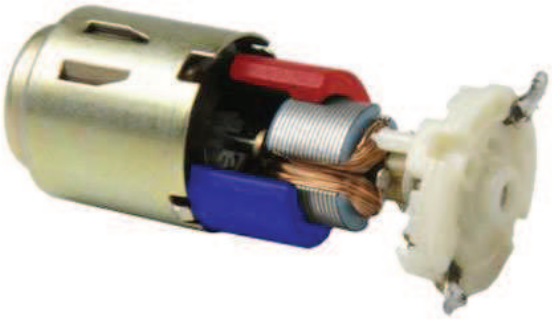


Fig 12. Motor DC

También hay otro tipo de motores eléctricos que no llevan escobillas conocidos como motores *Brushless*. Tanto su estator como su rotor están formados por bobinados de cobre. Cuando los bobinados del estator son recorridos por corriente alterna, se generan campos magnéticos secuencialmente en los bobinados, produciendo un campo giratorio, que induce una corriente en el bobinado del rotor que, al estar en cortocircuito, tiende a girar a una velocidad proporcional a los pares de polos y a la frecuencia de la corriente que recorre el estator.

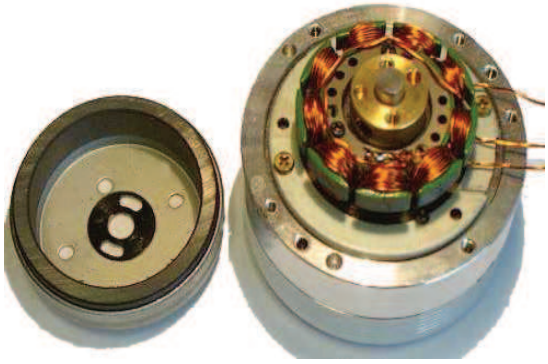


Fig 13. Motor Brushless

La diferencia principal entre los dos motores es, como hemos mencionado, el tipo de corriente. El motor DC requiere corriente continua para funcionar, que es la más común en este tipo de trabajos por ser la necesaria para el resto de componentes como la placa o los

sensores. Por el contrario el motor Brushless necesita corriente alterna para generar su campo giratorio al carecer de escobillas, lo que complica el asunto y obliga a utilizar unos controles electrónicos que envían corriente trifásica alterna a los motores a partir de corriente continua y una señal de control que viene de la placa. También hay que añadir que al no tener escobillas no tiene la desventaja del rozamiento y consecuente desgaste que estas producen, por lo que estos motores alternos tienen una durabilidad más elevada.

En cuanto a la potencia y velocidad de giro hay que decir que en el motor Brushless es bastante más elevada, ya que en el DC viene limitada por la velocidad de conmutación de las escobillas. También es importante tener en cuenta que la implementación del control del sistema con motores Brushless es bastante más compleja que con motores DC en los que simplemente se controla la tensión que reciben mediante transistores.

Para terminar queda lo importante: el coste. Y es que aquí radica la principal diferencia. Un motor DC con la hélice incluida nos puede costar unos 5€ más o menos. Si necesitamos 4, con 20€ tenemos la propulsión de nuestro quadrotor. Ahora bien, un motor Brushless, con la hélice puede alcanzar fácilmente los 30€. Si le sumamos los 15€ que puede valer el control electrónico necesario ya son 45€. Por 4 que necesitamos son 180€ que nos gastamos ya solamente en la propulsión, así que es un punto clave a considerar para no salirnos de presupuesto.

A continuación tenemos una tabla a modo de resumen del apartado de motores, con los principales puntos fuertes y débiles de cada tipo de motor.

	Motor DC	**Motor Brushless**
Durabilidad	Menor	Más alta
Potencia y velocidad	Menor	Más alta
Consumo	Mayor	Menor
Implementación del control	Sencilla	Compleja
Coste	≈5€	>45€

Tabla 2. Resumen de motores

b) Sensores dinámicos

Ahora que ya tenemos los actuadores de nuestro Drone necesitamos los sensores para conocer en todo momento su orientación espacial y que el microcontrolador elabore las señales de corrección correctas.

La forma que tenemos de pintarle el mundo real a un procesador es mediante sensores que transformen luz, fuerzas o temperatura en señales eléctricas.

Ciertamente la variedad de sensores que hay hoy en día es casi infinita, desde sensores de temperatura hasta detectores de infartos en prendas de ropa futuristas, así que puestos a imaginar podríamos crear un robot inteligente volador con la suficiente cantidad de sensores y claro, la programación adecuada. Pero como no es nuestro caso vamos a “regalarle” a nuestro quadcopter un par de sensores imprescindibles y suficientes para el objetivo que le queremos

dar de ser capaz de sustentarse en el aire y responder correctamente a las órdenes de movimiento. Estos sensores son el acelerómetro y el giroscopio o giróscopo.

Son sensores dinámicos que usualmente se utilizan juntos (incluso a veces están en un mismo circuito integrado). La diferencia principal entre ambos es que mientras el acelerómetro detecta las aceleraciones en los 3 ejes, el giroscopio detecta los giros, mejor dicho, las derivadas de los giros, porque lo que mide es la velocidad de rotación del mismo sobre los 3 ejes. Veámoslos por separado:

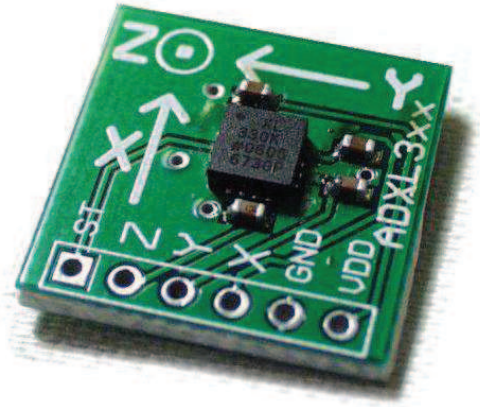


Fig 14. Acelerómetro

Un acelerómetro detecta las aceleraciones en los 3 ejes y devuelve los valores correspondientes en forma de variaciones en la tensión (analógicos) o por comunicación serial (digitales). Su funcionamiento es muy sencillo: En los piezoeléctricos, los usados en electrónica, se comprime un retículo cristalino piezoeléctrico por efecto de alguna aceleración, y eso produce una carga eléctrica proporcional a la fuerza aplicada que se transforma y se envía al microcontrolador.

Así, si el acelerómetro está en caída libre no detectará ninguna aceleración, pero sólo con que esté en reposo por ejemplo encima de una mesa, ya detectará una aceleración de $-9.81\ \frac{m}{s^2}$ en el eje perpendicular a la superficie de la mesa. Así, teniendo en cuenta la trigonometría correspondiente, sólo con los valores del acelerómetro se puede saber si el quadcopter está en reposo, si está en caída libre o si está ascendiendo. No obstante, para un control más preciso y para facilidad en el futuro al implementar el control PID, se incluye también un giroscopio.

Un giroscopio es un objeto en forma de disco, montado en un soporte cardánico, que puede medir la velocidad de rotación con respecto a 3 ejes por el principio de conservación del momento angular.

Fig 15. Giroscopio

La explicación al efecto del giroscopio es bastante compleja y se sale de los objetivos del presente proyecto, por lo que nos quedaremos simplemente con que es un circuito integrado que nos permite medir los giros y devolverlos, al igual que el acelerómetro, en forma de valores de tensión en el caso de ser analógico o por comunicación serial en caso de ser digital[5].

Entre las opciones comerciales podemos escoger entre acelerómetros y giroscopios digitales o analógicos como hemos indicado, y entre circuitos separados o un solo integrado con los dos sensores. Variedad de marcas modelos y precios hay una barbaridad.

[5] *Ver Giroscopio en http://es.wikipedia.org/wiki/Gir%C3%B3scopo*

A continuación se presenta una tabla con varias opciones con ventajas y desventajas:

	ADXL345	ADXL193	ITG-3200	LPY503AL	MPU 6050
Acelerómetro	X	X			x
Giroscopio			X	X	x
Ejes	3	1	3	2	3
Digital	X		X		x
Analógico		X	X	X	
Coste	17.95€	29.95€	24.95€	29.95€	39.95€

Tabla 3. Resumen de sensores

Viendo la gran variedad de precios se ve como hay mucha variación en modelos con funcionalidades parecidas, de lo que se infiere que la calidad de los mismos debe hacer variar mucho el precio.

A parte de estos dos podemos añadir cuantos queramos a nuestro quadcopter, como una brújula, un sensor GPS y un sensor de proximidad para evitar la colisión con obstáculos. Aquí es cuestión de imaginación, pericia a la hora de programar y bolsillo.

c) Placa procesadora

Ya tenemos los sensores y los actuadores de nuestro sistema. Ahora falta lo más importante: el cerebro. Alguien que se encargue de recopilar todos los datos suministrados por los sensores, los procese y elabore con ellos una respuesta lo más adecuada posible para conseguir los movimientos deseados. Este componente es la placa. Bueno, en realidad no la placa, solo el microcontrolador que lleva integrado.

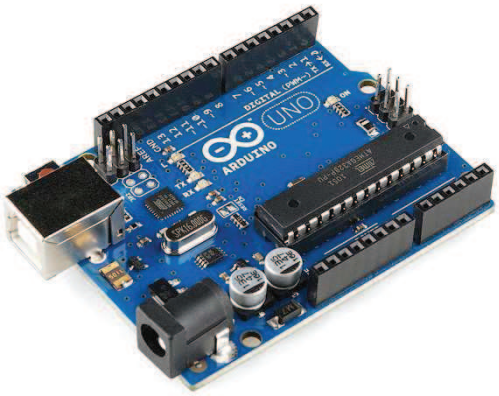


Fig 16. Placa Arduino

La placa es una especie de esqueleto que facilita enormemente la programación del microcontrolador al llevar integrados los demás componentes que, de utilizar el micro suelto, habría que diseñar e implementar, y el trabajo sería mucho más costoso. Procesadores y placas hay en el mercado para aburrir, así que para este proyecto vamos a utilizar la mundialmente conocida: Placa Arduino.

El proyecto Arduino surgió allá por el 2005 en el Instituto de Diseño Interactivo IVREA de Italia, cuando Massimo Banzi y Hernando Barragan se dieron cuenta de la necesidad de obtener una plataforma más moderna, barata y sencilla de programar para enseñar a sus alumnos.

Por aquel entonces se utilizaba en todo el mundo un microcontrolador llamado Basic Stamp que costaba unos 76 €, lo que resultaba muy caro para los estudiantes. Los creadores trabajaban con un lenguaje de programación de código abierto y de fácil utilización basado en JAVA, el processing, y empezaron a buscar cómo hacer un "processing" para el hardware.

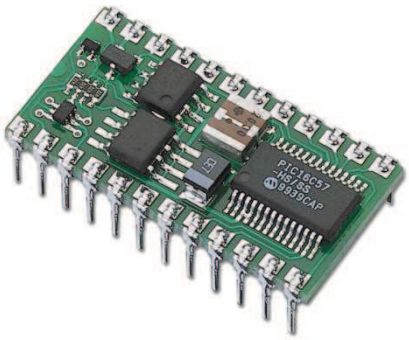


Fig 17. Basic Stamp

Arduino tiene dos grandes puntos a favor más: Uno es que es compatible tanto con Windows como con Mac y con Linux, por lo que es apto para cualquier plataforma. La segunda es que tanto en internet como en el mismo programa de ordenador que sirve para programar la placa, aparecen ejemplos de programas que, tan sólo con transmitirlos a la placa ya sólo queda conectar los periféricos a esta para que funcione.

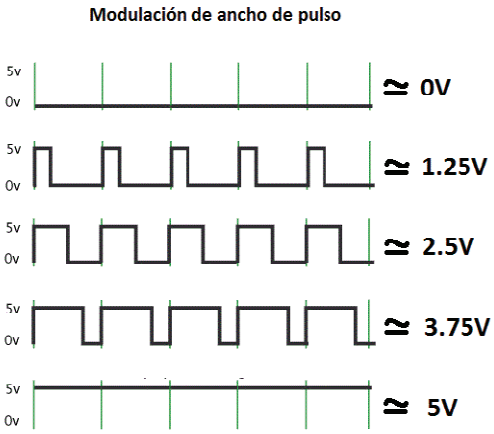


Fig 18. PWM

Así empezaron a desarrollar las placas y en poco tiempo se convirtieron en las más utilizadas en el mundo para programación de pequeños proyectos.

La placa Arduino consiste en una especie de circuito integrado que tiene una serie de entradas y salidas digitales de 5 voltios y otras tantas entradas analógicas. Casi ningún modelo tiene salidas analógicas propiamente dichas, pero puede simularlas mediante un técnica denominada Modulación de ancho de pulso (PWM por sus siglas en inglés **P**ulse **W**idth **M**odulation), que consiste en modificar el ciclo de trabajo de una señal cuadrada (de las salidas digitales de 5 V) de forma que se obtiene cualquier valor de tensión en la salida entre 0 y 5 voltios.

La placa dispone de un terminal de alimentación para poder enchufarla a una batería o a una toma de corriente de 9 voltios, y un conector USB para poder conectarla a un ordenador directamente. También tiene disponibles una serie de módulos acoplables denominados *Shields*, que encajan encima de la placa otorgándole funcionalidades extra como posicionamiento GPS o comunicación inalámbrica.

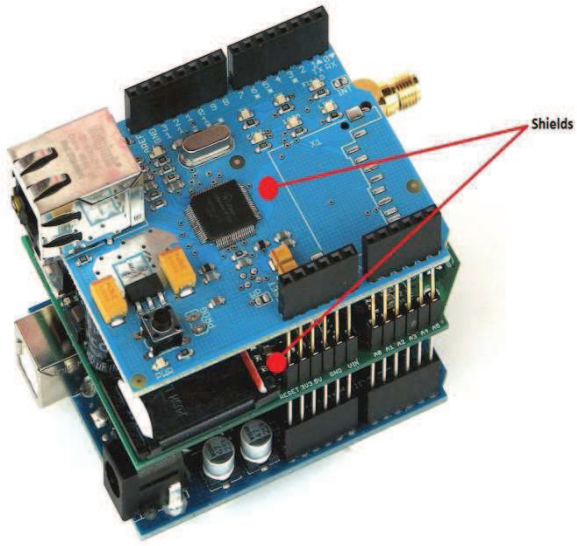


Fig 19. Shields

En cuanto a los modelos comerciales son varios los que ofrece Arduino. A continuación mostramos 4 opciones que podemos

considerar al escoger la placa:

Uno: La placa más conocida por tener las funcionalidades básicas. Tiene un microcontrolador ATmega328, con 1 KB de memoria EEPROM[6] , 32 KB de memoria flash[7] y 2 KB de SRAM[8]. Posee 14 pines digitales de entrada/salida, de los cuales 6 pueden usarse como salidas analógicas por PWM, y 6 pines de entrada analógica. Su voltaje de alimentación es de unos 9V y sus entradas y salidas tienen un voltaje máximo de operación de 5 voltios y una intensidad máxima de 40 mA. Sus dimensiones son de 6,9 x 5,34 cm. Precio: **20.00 €**

Due: Es casi el doble de larga que la **Uno**, y tiene un microcontrolador AT91SAM3X8E, con bastante más memoria Flash y SRAM: 512 KB y 96 KB respectivamente, pero sin memoria EEPROM. Tiene 54 pines digitales de E/S de los cuales 12 pueden usar PWM, 12 entradas analógicas y este modelo si tiene 2 salidas analógicas mediante 2 conversores DAC. Su voltaje de alimentación es también de 9V , el de operación es de 3.3 V y su corriente máxima en cada pin es de 130 mA. Sus dimensiones son de 10,16 x 5.34 cm. Precio: **36.00 €**

Mega: Es de un tamaño similar a la **Due**, pero en funcionalidad es más una versión mejorada de la **Uno**. Su microcontrolador es el ATmega2560, una versión mejorada del ATmega328, con 256 KB de memoria Flash, 8 KB de SRAM y 4 KB de EEPROM. Posee 54 pines de entrada/salida digital, de los cuales 15 permiten PWM, 16 entradas analógicas y, al igual que el **Uno** no tiene salidas analógicas. Su voltaje de alimentación es de 9V como los anteriores, y su voltaje máximo operativo en los pines es de 5V también como el primero, a 40 mA de corriente máxima. Precio: **39.00 €**

Micro: Es una de las placas más compactas de Arduino, también con menos prestaciones que las dos anteriores. Tiene un microcontrolador ATmega32u4 desarrollado con la colaboración de una empresa estadounidense dedicada a la venta de componentes electrónicos llamada **Adafruit**. El microcontrolador dota a la placa de 32KB de memoria Flash, 2.5 KB de SRAM y 1 KB de EEPROM. Posee 20 pines de E/S digitales, de los cuales 7 pueden usar PWM, 12 entradas analógicas y tampoco tiene salidas analógicas. Su voltaje de alimentación también es de 9V y su voltaje máximo en los pines es de 5V, con 40mA de corriente máxima. Vamos, básicamente es la versión compacta de la Arduino **Uno**, con 4,8 x 1,8 cm. Precio: **18.00 €**

A continuación tenemos una tabla resumen con las características principales de los 4 modelos que acabamos de explicar.

	Flash	SRAM	EEPROM	E/S Dig	PWM	V máx.	I máx.	Entradas Analogic.	Precio
Uno	32 KB	2 KB	1 KB	14	6	5 V	40 mA	5	20 €
Due	512 KB	96 KB	----------	54	12	5 V	130 mA	12	36 €
Mega	256 KB	8 KB	8 KB	54	15	3,3 V	40 mA	16	39 €
Micro	32 KB	2,5 KB	1 KB	20	7	5 V	40 mA	5	18 €

Tabla 4. Resumen de placas Arduino

[6] *Electrically Erasable Programmable Read-Only Memory,* o en español *Memoria de sólo lectura programable y borrable eléctricamente.*

[7] Las memorias flash son la evolución de las EEPROM, siendo más rápidas que estas.

[8] *Static Random Access Memory*, o en español *Memoria Estática de Acceso Aleatorio*. Puede mantener los datos sin necesidad de refresco mientras está alimentada.

Ahora que tenemos la placa tengamos en cuenta que no es un componente como los motores o los sensores que simplemente con conectarlos debidamente ya realizan su función. La placa hay que programarla mediante un código en un lenguaje muy parecido al C, de lo que hablaremos en el capítulo 3.

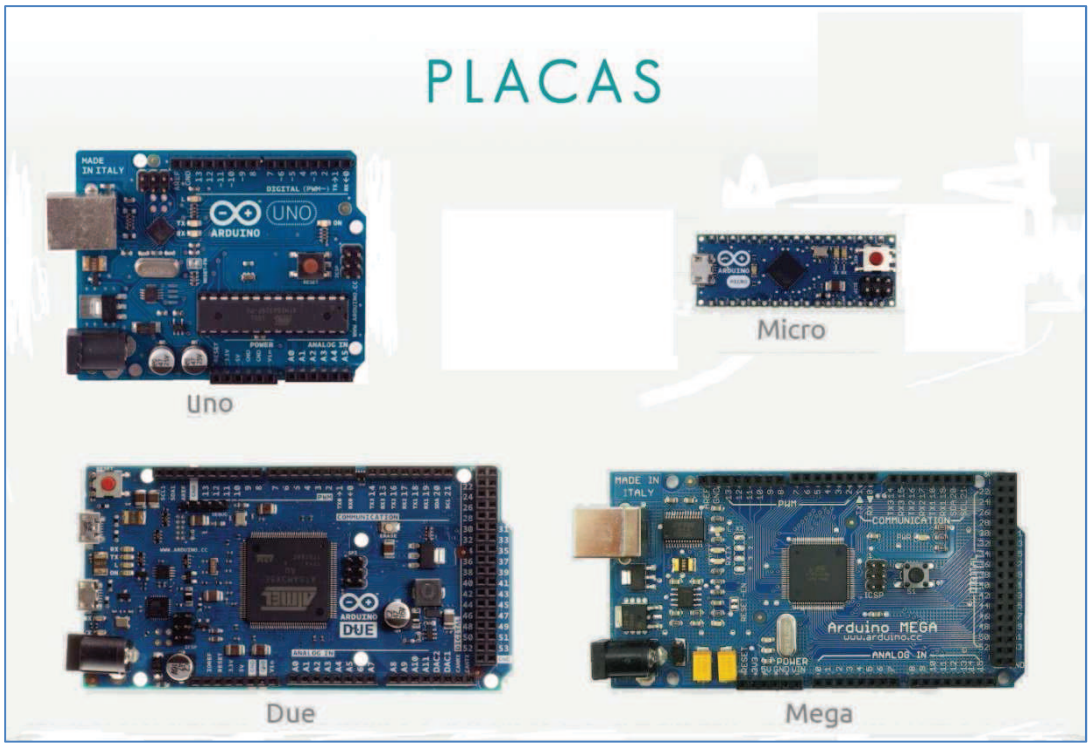


Fig 20. Placas Arduino

d) Módulo de comunicaciones

En cuarto lugar tenemos el módulo de comunicaciones. Ya que nuestro dispositivo va a ser móvil, tenemos que conseguir enviarle las órdenes a distancia sin ningún tipo de cables. Para esto la idea es implementar una interfaz de control en un dispositivo a distancia con el que se comunicará mediante un sistema inalámbrico para lo que veremos varias opciones.

Como vimos en el apartado anterior Arduino tiene unos apliques llamados **Shields** con los que se le pueden añadir funcionalidades. Bueno, pues entre estas funcionalidades extra hay sistemas de comunicación inalámbricos de los que vamos a ver 3 ejemplos a parte de otro método que no requiere Shields:

Infrarrojos: Nuestra primera opción es la más sencilla de implementar, pues simplemente necesitamos un receptor infrarrojo en nuestro circuito (en este caso un VS1838) y un emisor infrarrojo en el control que utilicemos. Este sistema no necesita Shields, por lo que habría que utilizar una pequeña placa de prototipos o soldarlo a la circuitería. La dinámica es sencilla: El control emite una señal de una frecuencia infrarroja que no es visible al ojo humano (como

alguna radiación solar). El receptor detecta esta onda y el procesador efectúa una u otra acción en función de la programación.

Como hemos dicho la radiación solar tiene una parte infrarroja, así que a menos que volemos el quadcopter de noche tenemos un problema porque el receptor estaría siempre detectando la solar. Afortunadamente hay una solución sencilla si se tiene en cuenta que la frecuencia de nuestro emisor va a estar en torno a 38kHz, y es programar en el código un filtro paso banda, o adquirir receptores infrarrojos que traigan esta funcionalidad integrada.

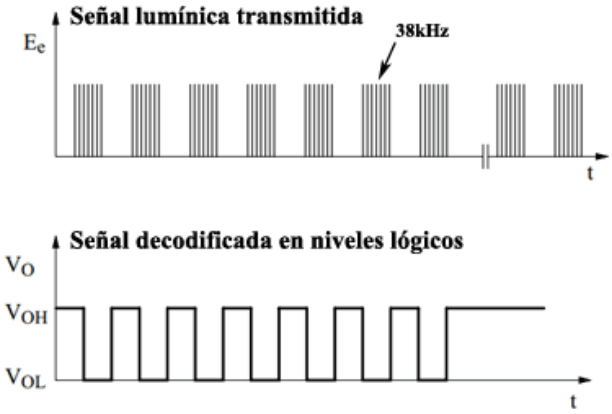


Fig 21. Decodificación de señal infrarroja

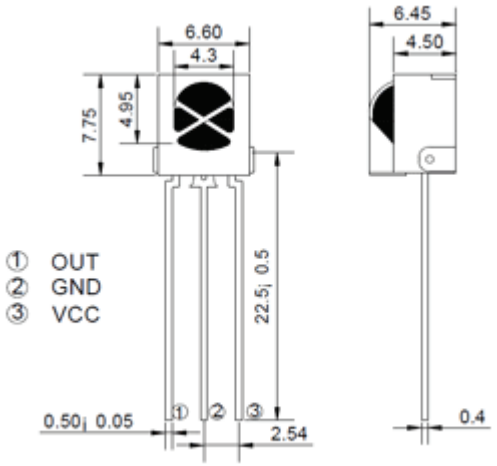


Fig 22. Receptor Infrarrojo VS1838

Así, el emisor enviará una serie de pulsos en una combinación precisa que el procesador interpretará, pero siempre en frecuencia de 38kHz.

El VS1838 ya trae, como decíamos, un filtro paso banda en esa frecuencia, por lo que no tenemos que implementarlo. Tiene 3 patillas de las que dos se conectan a Vcc y Masa de unos 5 V y la tercera se conecta a una de las entradas digitales del Arduino.

La ventaja principal de este sistema es que es muy barato, pero las desventajas son muchas: Muchas interferencias, poco alcance, necesidad de tener visibilidad del receptor. **Precio: 1€**

Bluetooth: La siguiente opción que tenemos para la comunicación es el sistema Bluetooth. De alcance es equivalente al anterior, pero con la ventaja de que no necesita tener visión sobre el receptor para comunicarse con él.

El sistema Bluetooth utiliza la técnica FHSS (en español Espectro ensanchado por saltos de frecuencia) para comunicarse, que consiste básicamente en dividir un intervalo de frecuencia en 79 canales de 1MHz cada uno y transmitir la señal mediante una combinación de canales conocida tanto por el emisor como por el receptor.

Ha sido la tecnología inalámbrica de corto alcance más utilizada hasta la llegada del Wi-Fi por su versatilidad y uso sencillo. Se usaba en móviles, teclados y ratones inalámbricos, manos libres etc.

Su ventaja principal es el precio también, ya que no requiere componentes caros, y la capacidad de poder estar en contacto emisor y receptor sin visión directa uno del otro. Su desventaja es, como con los infrarrojos el rango, ya que alcanza como máximo unos 10-15 metros, reducibles por paredes intermedias.

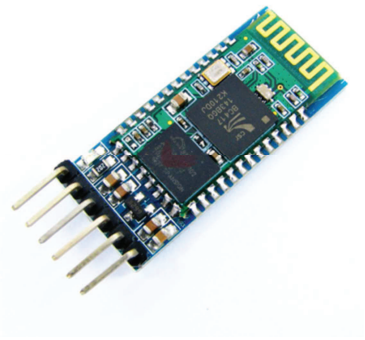


Fig 23. Módulo Bluetooth HC-05

El sistema bluetooth se puede acoplar a la placa Arduino mediante un Shield o directamente por circuitería con un módulo bluetooth HC-05, siendo la primera opción más cara pero más robusta también. **Precio Shield: 26€. Precio Módulo: 10€.**

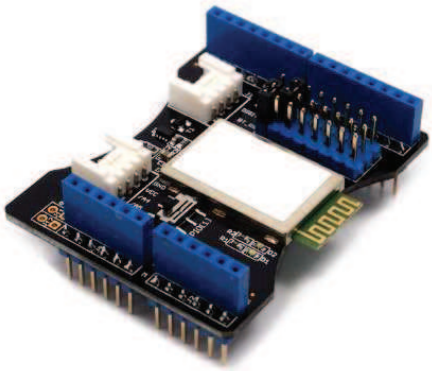


Fig 24. Shield Bluetooth

Wi-Fi: Quien no va a conocer esta tecnología siendo nuestro principal punto de conexión con el internet que tanto queremos. Y es que desde más o menos el año 2000 el Wi-Fi está presente en nuestra vida en cualquier sitio: Móviles, portátiles, Smart-Tv, etc…

Aunque comúnmente se cree que el nombre "Wi-Fi" viene de *Wireless Fidelity* (Fidelidad inalámbrica), en realidad la verdad es bastante decepcionante, pues surgió simplemente de una petición de la WECA (Empresa creadora del Wi-Fi) a la corporación Interbrand a quien contrató para que les diera un nombre atractivo y sencillo de recordar.

Esta tecnología, al igual que el **bluetooth** utiliza la técnica FHSS para transmitir, usualmente en la frecuencia de 2.4 GHz. Para Arduino existen módulos Wi-Fi en forma de Shields.

Entre las ventajas del Wi-Fi tenemos la capacidad, al igual que el **bluetooth**, de conexión sin vista directa del objetivo. También su mucho mayor alcance con respecto a este de 300 m frente a los 15 del **BT**, pero por desgracia no comparte su ventaja del bajo precio, ya que un **Shield Wi-Fi para Arduino** nos puede costar unos **72 €.**

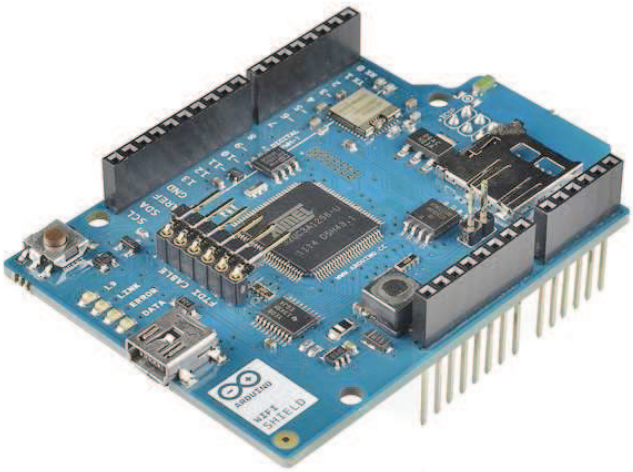


Fig 25. Shield Wi-Fi

XBee: Nuestra última opción para formar el sistema de comunicación inalámbrico es el módulo XBee de MaxStream.

Estos módulos son dispositivos inalámbricos que tienen su propio protocolo de comunicación por radiofrecuencia. Son pequeños, con bajo consumo y según el modelo pueden tener alcances de hasta ¡24 km!

Cada módulo tiene 9 entradas y salidas, siendo las entradas analógicas y digitales. El consumo de corriente del **XBee** es de menos de 50 mA cuando está en funcionamiento y menos de 10 mA cuando está en reposo. Ahora llega el principal inconveniente. Y es que a diferencia de los anteriores, que el emisor puede ser un móvil que ya tiene bluetooth o Wi-Fi, no tiene XBee integrado, así que el control necesitará 2 módulos, uno para la placa Arduino y otro para conectarlo al dispositivo emisor. A parte necesitamos un Shield para acoplar el XBee a Arduino. Así que tenemos 2 módulos cuyo precio para un modelo con alcance de unos 1500m (nuestro Quadcopter difícilmente sobrepasará esa distancia en vuelo) es de unos **33 €** cada uno, más el Shield para el acoplamiento con la placa que vale unos **20 €**. Total: 2x33€+20€= **86€.**

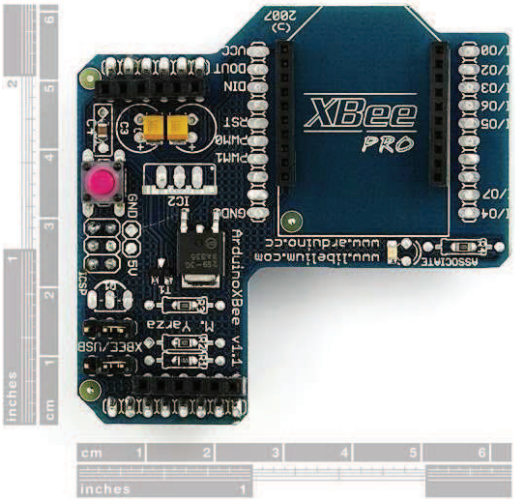


Fig 26. Shield XBee sin módulo

Fig 27. Shield XBee con módulo XBee

A continuación tenemos una tabla resumen con las opciones mencionadas.

	Alcance	Dificultad de implementación	Comunicación directa desde móvil	Coste
Infrarrojos	5 metros	Baja	Si	**1 €**
Bluetooth	15 metros	Baja	Si	**10-26€**
Wi-Fi	300 metros	Media	Si	**72€**
XBee	1500 metros	Alta	No	**86€**

Tabla 5. Resumen sistema comunicación

e) Batería

Ahora que tenemos todos los componentes necesarios en nuestro quadcopter para su funcionamiento nos falta algo fundamental: la alimentación. Para ello vamos a necesitar un elemento mundialmente conocido que se utiliza en prácticamente cualquier aparato electrónico portátil: la batería.

Una batería es un dispositivo que básicamente almacena carga eléctrica para liberarla posteriormente a medida que se necesite. Las baterías se cargan y descargan con corriente continua, por lo que si necesitamos corriente alterna debemos disponer un conversor adicional.

Antes del avance en el desarrollo de las baterías se utilizaban unos elementos parecidos llamados pilas, que aún se siguen usando en muchos aparatos electrónicos. La diferencia principal entre una pila y una batería es que, mientras esta última se puede recargar conectándola a una fuente de corriente, la pila produce electricidad en un proceso irreversible y hay que eliminarla cuando se agota. El problema con esto es que los componentes de las pilas son altamente contaminantes, y su eliminación presenta un problema cuando no se hace adecuadamente (desgraciadamente bastante a menudo).

Por el contrario, las baterías generan electricidad por medio de reacciones reversibles, así que pueden reutilizarse muchas veces antes de que tengan que eliminarse, lo que resulta en un ahorro económico y medioambiental. Consisten en una o más celdas electroquímicas cada una con un cátodo (electrodo positivo +), un ánodo (electrodo negativo -) y electrolitos que permiten a los iones moverse entre los electrodos y producir así la diferencia de potencial.

A continuación vamos a ver 4 tipos de baterías según su composición interna:

Níquel-Cadmio:

Las baterías de **Ni-Cd** son utilizadas en el ámbito doméstico e industrial. Están formadas por un cátodo de hidróxido de níquel y un ánodo compuesto de cadmio, con un electrolito de hidróxido de potasio. Producen un voltaje de 1,2 V por celda, y tienen una densidad de energía de 50 Wh/kg[9], con una intensidad máxima de 1 A. Pueden funcionar en un amplio rango de temperaturas y admiten sobrecargas, pero por el contrario tienen un alto efecto memoria[10] y el cadmio es muy contaminante, por lo que están en desuso.

Níquel-Hidruro metálico:

Este segundo tipo es similar a las de Ni-Cd pero sin el contaminante cadmio. Producen el mismo voltaje (1,2 V por celda), pero tienen una capacidad de carga mayor, 80 Wh/kg, y tienen una intensidad máxima de salida de hasta 2,8 A. También son mejores en cuanto a

[9] Vatios por hora por cada kilogramo. Es una unidad de densidad energética puesto que W·h son unidades de energía.

[10] El efecto memoria es un problema que tienen algunas baterías y consiste en que si se interrumpe la carga de la batería antes de que esta llegue al 100% la próxima vez que se cargue sólo lo podrá hacer hasta el punto en el que se interrumpió la vez anterior. Este efecto se da principalmente en baterías de Níquel-Cadmio y Níquel-Hidruro metálico.

que se encuentran menos afectadas por el efecto memoria que las anteriores. No obstante tienen la desventaja de que no soportan bien el frío extremo, reduciendo la potencia eficaz que pueden entregar.

Ion Litio:

Las baterías de **Li-ion** son de las más modernas. Están formadas por un ánodo de grafito y un cátodo de óxido de cobalto, trifilina (LiFePO4) u óxido de manganeso. Producen un voltaje variable por celda, dependiente de la carga de la batería, variando desde unos 4,2 V a plena carga hasta unos 2,7 V casi descargada, siendo el voltaje nominal de 3,7 V. Tienen una densidad de energía de 115 Wh/kg y una intensidad máxima de unos 2,8 A. No les afecta prácticamente el efecto memoria, y pueden recargarse sin estar completamente agotadas sin ningún perjuicio, pero tienen la desventaja de que no soportan bien las descargas completas, por lo que suelen llevar circuitería adicional para conocer la carga de la batería y evitar su descarga completa.

Polímero de litio:

Las baterías **Li-PO** son una variación de las Li-ion, teniendo una densidad de energía y una tasa de descarga mayores. Tienen una tensión nominal de 3,7V por celda, y se dañan irreparablemente si la tensión cae por debajo de 3 V. Se suelen agrupar en serie varias celdas para obtener voltajes mayores, nombrándose con un número seguido de la letra S según las celdas que la compongan:

Li-PO 1S: una celda. 3,7 V

Li-PO 2S: dos celdas. 7,4 V

Li-PO 3S: tres celdas. 11,1 V

Li-PO 4S: cuatro celdas. 14,8 V

Al adquirir una batería tenemos que tener en cuenta la tensión y la corriente que necesitamos para alimentar todos los elementos simultáneamente, y también la capacidad energética que queremos, pues de ello depende el poder volar el quadcopter durante 2 minutos o durante 20. Para ello debemos ver un dato de las especificaciones de la batería con unidades mAh (mili amperios hora). Ese dato indica los miliamperios que puede suministrar la batería durante una hora, por lo que si nuestro consumo fuera del doble de intensidad, lo podríamos mantener durante media hora. De este dato depende principalmente el precio de una batería.

Además no debemos olvidar que queremos volar el aparato, por lo que cualquier gramo extra de peso lastra la movilidad, así que tenemos que tener en cuenta el peso de la batería.

En este apartado no podemos poner varias opciones de baterías como los anteriores, pues la variedad en cuanto a tensiones, intensidades, capacidad y precio

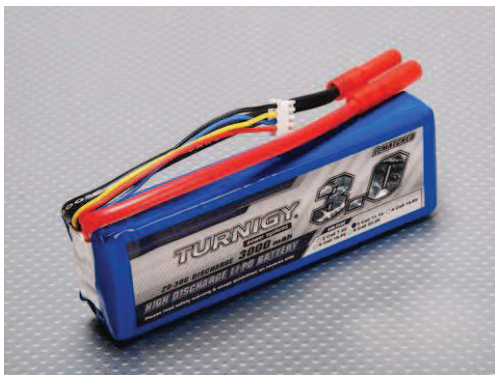


Fig 28. Batería Li-Po Turnigy 3000 mAh

es inmensa, y hasta que seleccionemos definitivamente el resto de componentes no sabremos con seguridad los requerimientos que necesita nuestra batería.

f) Chasis

En cuanto al chasis del quadcopter decir que es el soporte que mantiene unidos los anteriores componentes y fijos en sus posiciones y evita que el aparato se convierta en un amasijo de escombros con el primer golpe.

Para el caso actual de un quadcopter vamos a necesitar que tenga 4 brazos y que sea lo más simétrico posible para no sobresolicitar un motor por encima de los otros en reposo[11]. Dicho esto cualquier estructura en X con algo que le sirva de tren de aterrizaje y sea mínimamente resistente nos serviría.

En cuanto a los materiales hay desde chasis caseros hechos en madera de balsa, barata y de procesamiento sencillo, hasta chasis comerciales hechos en fibra de carbono ultra resistente, pasando por los hechos en plástico por impresoras 3D.

Para hacernos una idea a continuación tenemos una tabla resumen con características de 3 posibles opciones con sus correspondientes fotografías.

	Diseño propio	Fabricación propia	Resistencia	Coste
Madera de balsa	x	x	Baja	<10€
Polietileno en impresora 3D	x	-	Media	50€
Fibra de carbono comercial	-	-	Alta	142€

Tabla 6. Resumen de opciones de chasis

Fig 29. Chasis Comercial *Tarot*

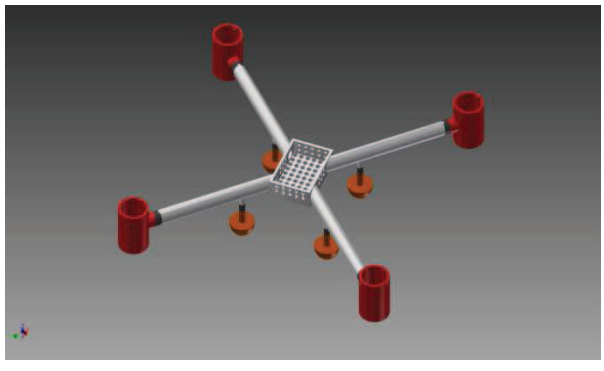


Fig 30. Chasis diseñado por Ordenador

[11] Cuando hablamos de estado de reposo en el quadcopter nos referimos al aparato suspendido en el aire sin realizar ningún movimiento.

g) Control remoto

El último "componente" de nuestro quadcopter es el control remoto, y va entre comillas porque en realidad es un componente externo al aparato sin el cual no podemos transmitirle las órdenes que debe cumplir.

Para ello de las numerosas opciones que hay hemos seleccionado 3. Control mediante una emisora de radio, control mediante un ordenador y control mediante un dispositivo Android.

1) Emisora de radio

La emisora de radio es una opción común entre los amantes del aeromodelismo por su facilidad de uso al no tener que programarla, sólo hay que tener en cuenta los datos que va a enviar al receptor al programar este último y el calibrarla antes del primer vuelo. Se puede adquirir fácilmente por internet pero su precio suele ser bastante elevado y tiene la pega que necesariamente la comunicación ha de ser por radio, lo que nos deja sin opciones de escoger en el sistema de comunicación. **Precio: 85-2000€**

2) Ordenador

Esta segunda opción es más versátil en el sentido en el que no obliga a la comunicación a ser de una manera determinada, sino que depende del emisor que acoplemos al ordenador. Es una opción muy económica si tenemos en cuenta que no tenemos que comprar el PC ya que solo hay que comprar el emisor, pero tiene la pega de la poca movilidad de este, y a diferencia de la emisora, hay que diseñar un programa informático que sirva de interfaz para el control. **Precio (según sensor bluetooth, wi-fi, radio…): 5-30€**

3) Dispositivo Android

La última propuesta es programar una aplicación en cualquier dispositivo Android, que es una plataforma abierta. De esta forma podemos aprovecharnos por un lado de los elementos de transmisión que estos dispositivos llevan ya integrados, y por otro de su alta movilidad en el caso de Smartphones o Tablets. En este caso no tenemos que adquirir nada si ya tenemos el dispositivo, así que la mayor ventaja es: **Precio: 0€**

Fig 31. Emisora Radio FX-22

Fig 32. Ordenador y dispositivos Android

2) Diseño del chasis y selección de componentes

2.1) Planificación

En el presente capítulo vamos a obtener una estructura o chasis para el quadcopter y vamos a dejar claro los componentes que vamos a escoger para nuestro diseño. Así, al finalizar esta parte tendremos definida la parte física del aparato, es decir, sólo faltará la parte de software para que funcione.

Primero obtendremos una aproximación de componentes a utilizar. A continuación, en base al espacio necesario a ocupar por esos componentes obtendremos un primer diseño del chasis. De las opciones mencionadas en el capítulo anterior para obtener un chasis se va a optar por la opción de diseño e impresión en 3D, ya que uno de los objetivos del presente proyecto es demostrar las competencias adquiridas durante la carrera, y el diseño asistido por ordenador es una de ellas.

Será entonces cuando, con el peso de los componentes sumados a los del chasis tendremos el peso total a mover por los motores, y con ese dato dimensionaremos estos últimos. Ahora rediseñaremos el modelo del chasis para ajustarlo definitivamente a los motores y ya tendremos la estructura finalizada.

En este capítulo se mostrarán imágenes de los modelos 3D, pero no los planos. Estos se pueden encontrar en el documento **Planos** adjunto a la memoria.

2.2) Componentes electrónicos

Vamos a seleccionar los componentes internos de mayor a menor tamaño.

En primer lugar tenemos la placa procesadora, que de las opciones mostradas de Arduino vamos a escoger la placa Arduino UNO. Es la más estándar, en principio las salidas nos darán intensidad suficiente con ayuda de controles electrónicos de velocidad en el caso de motores Brushless o con transistores en el caso de motores DC. Además su precio es muy asequible y vamos a adquirir una físicamente para hacer pruebas. En cuanto al acelerómetro y giroscopio la mejor opción sería adquirir sendos circuitos independientes, pero para poder adquirir uno para hacer pruebas vamos a optar por un integrado de los dos, acelerómetro y giroscopio, el MPU-6050. Devuelve datos digitales por comunicación serial y tiene un precio muy económico. Y ya que realizaremos los ensayos y la implementación del PID para ese acelerómetro, para nuestra propuesta de quadcopter utilizaremos el mismo.

En cuanto a la batería ya hemos hablado de las múltiples opciones que hay en el mercado, así que estimaremos un peso y unas dimensiones, que son los únicos datos que nos interesan, y cuando tengamos todos los componentes obtendremos, en función de la demanda de electricidad, una batería adecuada.

Por último el sistema de comunicación que utilizaremos será el bluetooth, por un lado por el bajo precio y por otro porque tanto ordenadores como dispositivos Android tienen la

posibilidad de emitir por bluetooth. Entre utilizar Shield o módulo escogemos el módulo por su menor precio y peso. Optamos por el módulo bluetooth HC-05.

Así, hasta ahora tenemos:

Componente	Modelo	Peso	Dimensiones
Placa Arduino	Arduino UNO	40 g	19 x 43 cm
Acelerómetro	MPU-6050	2 g	2.3 x 2.3 cm
Batería	¿¿¿???	400 g	15.4 x 4.5 x 3 cm
Módulo Bluetooth	HC-05	4 g	1.7 x 4 cm

Tabla 7. Resumen componentes finales

Tengamos en cuenta que la interconexión entre estos elementos requiere de cableado y de una pequeña placa de prototipos para mantener quieto el acelerómetro y módulo bluetooth, así que añadiremos un margen de peso para ello de unos 50 g, con lo que tenemos un peso total de componentes de 496≈500 gramos.

2.3) Primer diseño del chasis

Ahora en primer lugar vamos a realizar el modelado 3D de los componentes anteriores para situarlos virtualmente en el chasis y así dimensionarlo correctamente. Para ello vamos a utilizar el software comercial Autodesk Inventor, en su versión para estudiantes, que es un programa usado en varias ocasiones durante la carrera que nos permite crear un modelo virtual con el que interactuar sin necesidad de fabricar nada físicamente.

Primero vamos a obtener el modelo de la placa Arduino UNO, midiendo con un pie de rey los componentes más significativos y plasmándolos en el boceto, para luego extruirlos cada uno una distancia determinada, con lo que llegamos a la siguiente pieza:

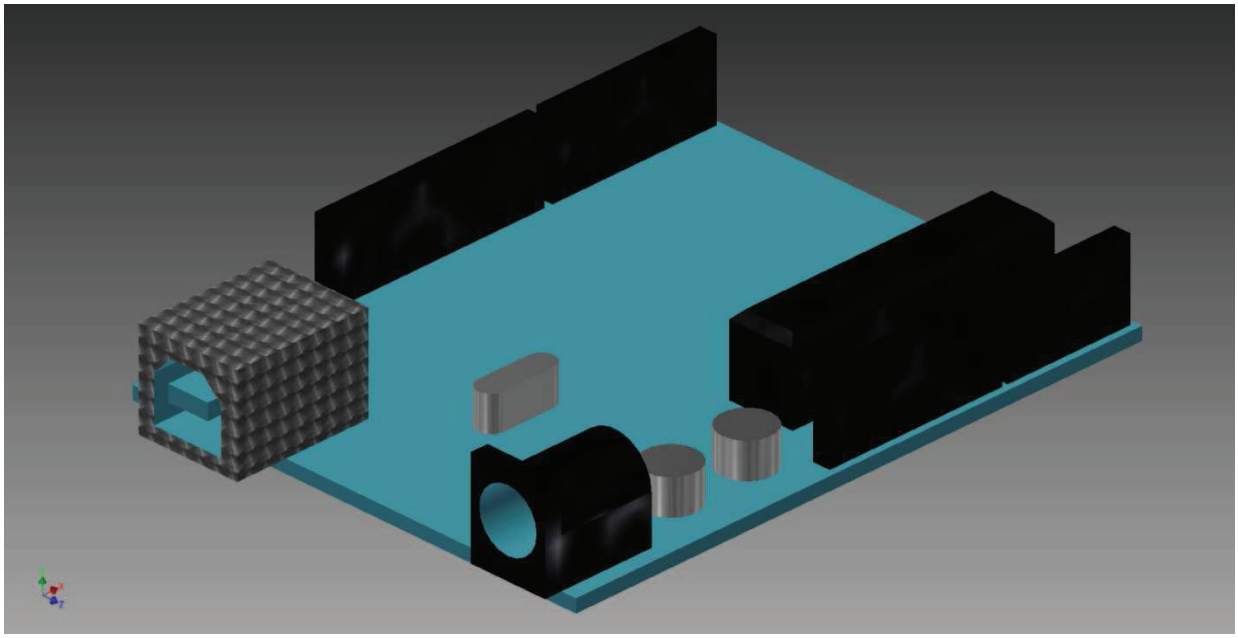


Fig 33. Modelo 3D Arduino UNO

Seguidamente hacemos lo mismo con el acelerómetro, el módulo bluetooth y la placa de prototipos.

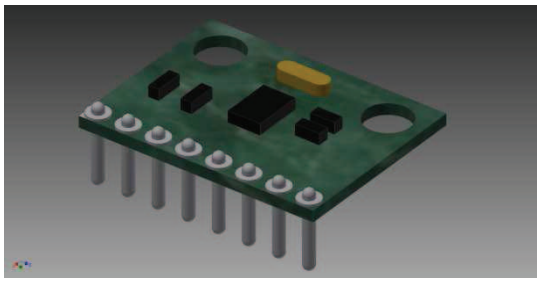


Fig 34. Modelo 3D Acelerómetro

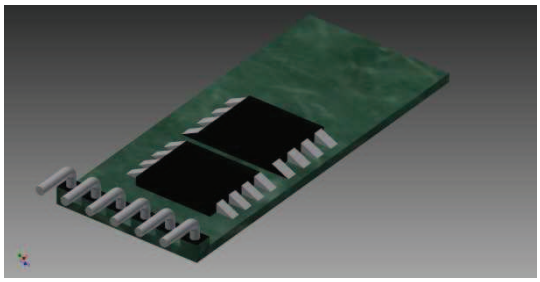


Fig 35. Modelo 3D Módulo Bluetooth

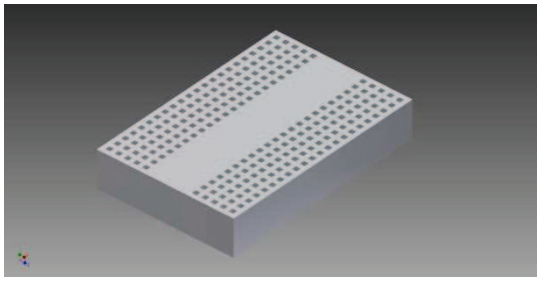


Fig 36. Modelo 3D Placa de prototipos

Ahora ensamblamos los 3 anteriores en un solo fichero de ensamblaje, pues irán juntos en la estructura para mantenerlos inmóviles.

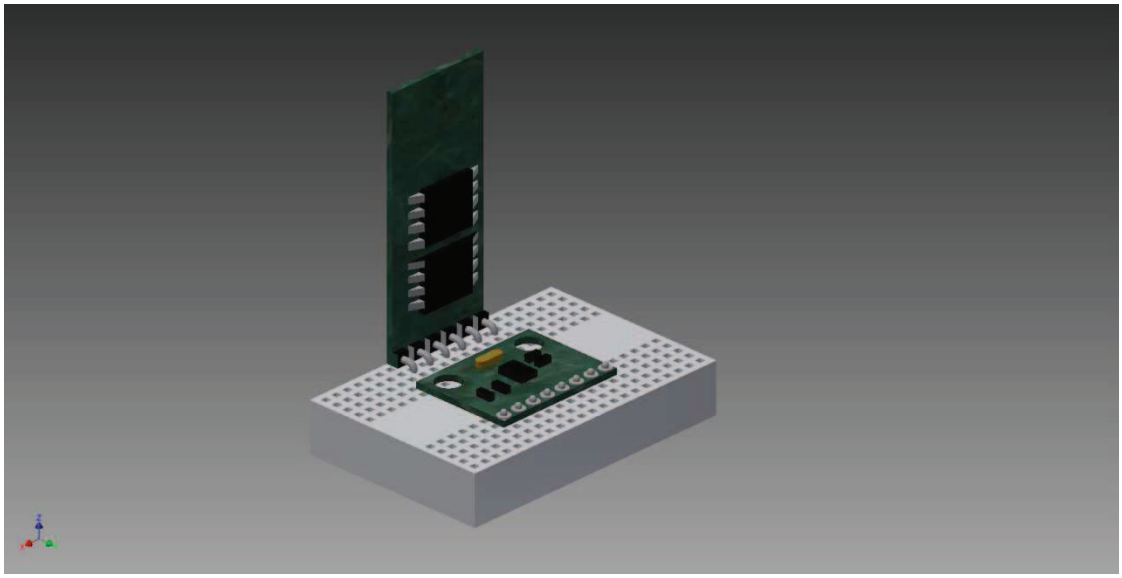


Fig 37. Ensamblaje 3D Acelerómetro, Bluetooth y protoboard

Por último vamos a hacer un modelo simple de la batería aproximándola a un prisma:

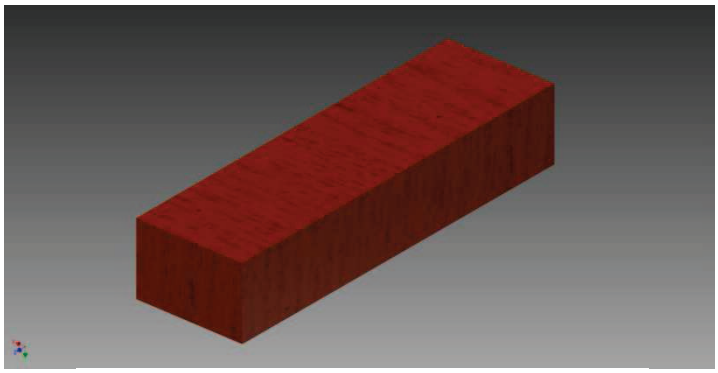


Fig 38. Modelo 3D Batería

Ahora que ya tenemos modelados los distintos componentes del quadcopter vamos a realizar un modelo de la parte central del chasis, es decir, la que albergará los componentes que acabamos de obtener. Dicha parte debe tener unas dimensiones para admitir al menos la placa Arduino, que es el elemento más amplio después de la batería, y que por su importancia y fragilidad debe ir bien resguardado.

La batería, al ser un elemento que no precisa de mayor cuidado, vamos a situarla por debajo del chasis, sujeta por algún tipo de cinta, para conseguir así también bajar el centro de masas de la estructura, lo que ayudará posteriormente a conseguir la estabilidad. El resto de componentes irán situados sobre la placa Arduino porque será el punto de mayor movimiento al tener el centro de masas bajo, y así el acelerómetro funcionará mejor.

De este modo hemos diseñado una estructura central para los componentes ligera al no hacer paredes macizas, lo que también ayuda a la ventilación, pero hay que tener cuidado con la lluvia en caso de volarlo en el exterior, ya que los circuitos estarán completamente expuestos.

Hemos dejado 4 agujeros laterales con terminales exteriores por los que discurrirá el cableado hacia los motores, y donde luego se unirán los brazos del quadcopter, y una pieza en forma de T que se puede poner y quitar, para separar la placa del protoboard. También hemos añadido 4 patas en diagonal a la estructura central para que sirvan de tren de aterrizaje de la estructura, y eviten que la batería toque el suelo.

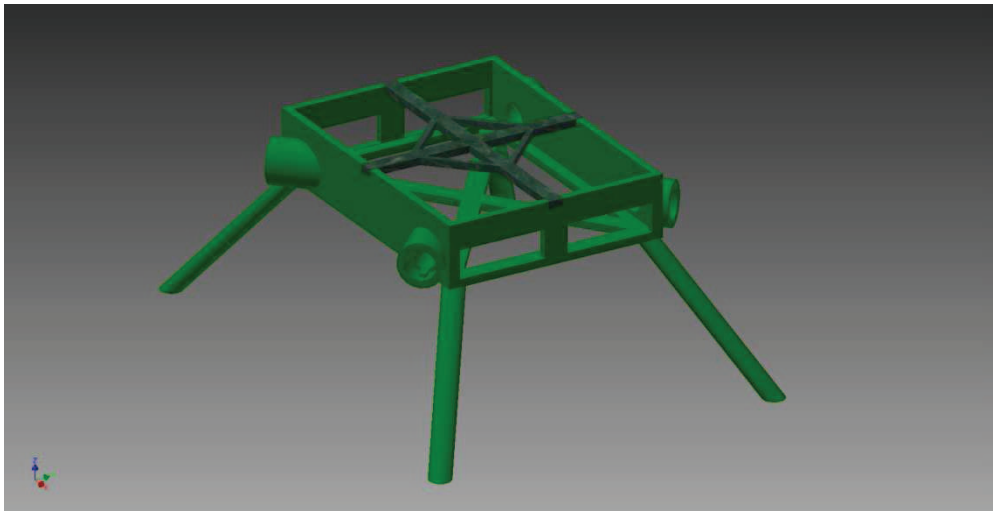


Fig 39. Estructura central del chasis

Este diseño lo hemos hecho en base a las medidas de la placa Arduino, de modo que debería encajar perfectamente, pero vamos a ensamblar la estructura central, la placa y los demás componentes para ver el resultado.

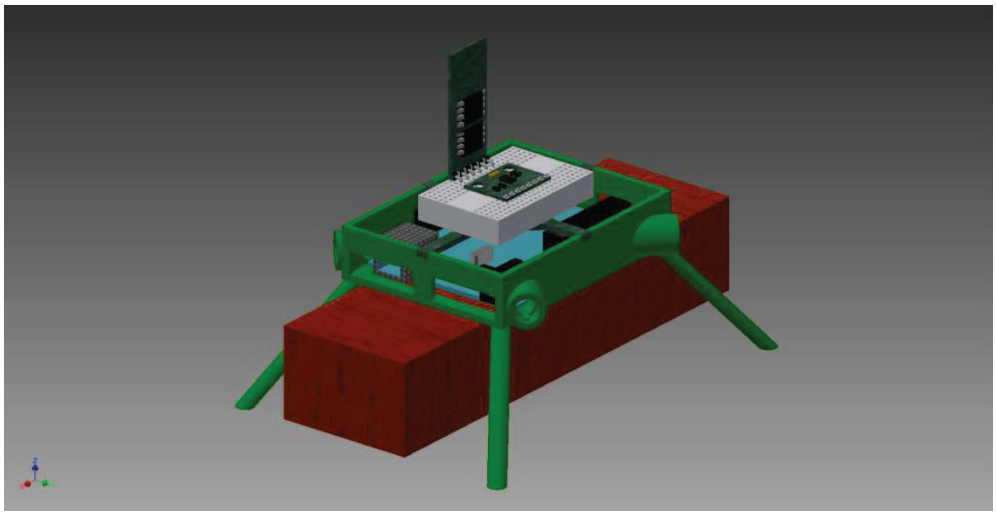


Fig 40. Ensamblaje central

Vemos que el ensamble queda bastante centrado y compacto, así que lo damos por bueno y ahora vamos a diseñar los brazos.

La distancia de estos puede variar cuanto queramos, teniendo en cuenta que cuanto más largos sean, la estabilidad será mayor, pero también el peso y el precio de la estructura, y la respuesta dinámica será más lenta. Así que optamos por diseñar unos brazos de 10 cm, lo que nos da una estructura de aproximadamente 30 cm de diámetro, que era más o menos el tamaño buscado.

En cuanto al perfil de los brazos tenemos innumerables opciones. En T, en doble T, tubulares, en U… En este caso las solicitaciones para dichos brazos son muy bajas, por lo que dimensionarlos según Estados Límite Últimos como hemos visto en asignaturas de materiales y construcción parece un poco absurdo.

No obstante como tenemos que escoger algún perfil hemos optado por uno tubular, que permite discurrir los cables por dentro y

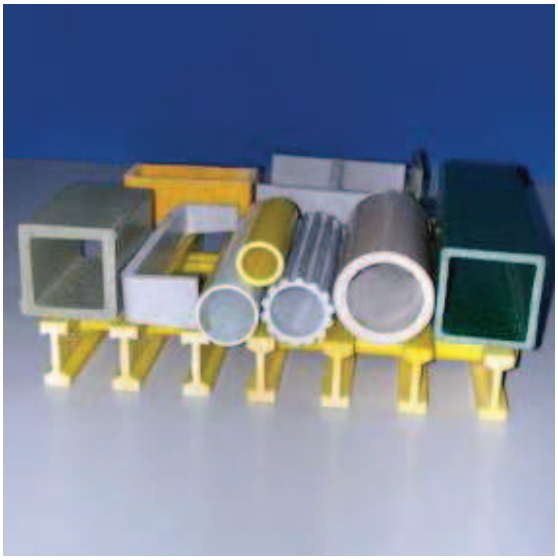


Fig 41. Tipos de perfiles

tiene una resistencia a momento torsor mayor que los otros perfiles, lo que nos interesa pues si los brazos llegaran a torcerse los ejes de los motores ya no serían perpendiculares al plano del quadcopter y quedarían descentrados, afectando al movimiento de este.

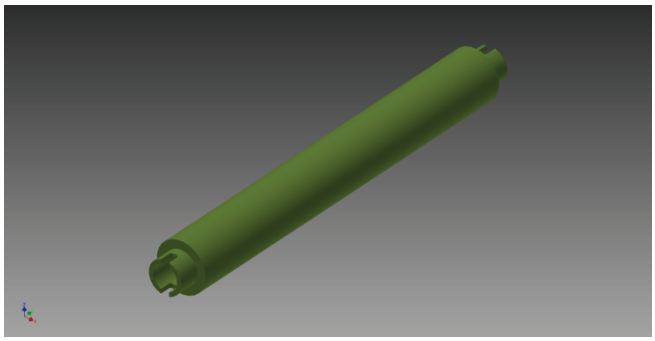


Fig 42. Brazo

También hemos añadido a los extremos del brazo dos terminales para facilitar la unión de este a la estructura central y a los soportes de los motores en el otro extremo.

Ahora vamos a ver la estructura central con los componentes y los brazos:

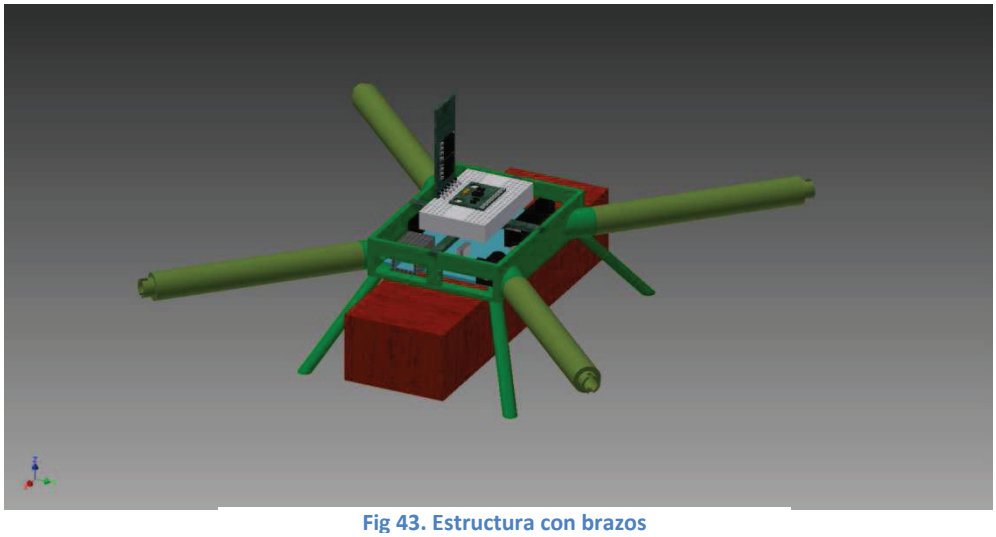


Fig 43. Estructura con brazos

Por último sólo queda diseñar los soportes para los 4 motores que irán en los extremos de los brazos. Para ello los vamos a hacer con forma de cilindro hueco abiertos por arriba, y con un agujero inferior por donde discurren los cables. No obstante hay que recordar que este modelado es una primera aproximación pues todavía no conocemos la morfología de los motores. Cuando la conozcamos ya ajustaremos estos soportes.

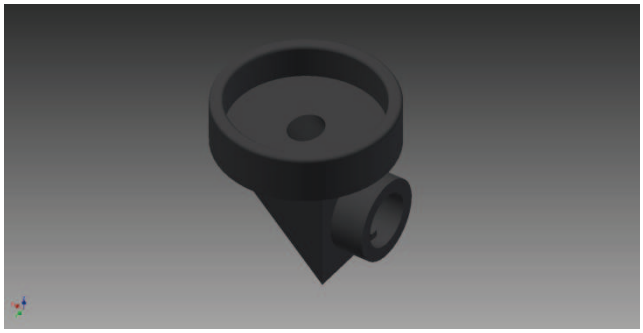


Fig 44. Soporte de motores

Ahora unimos los soportes de motores a lo que teníamos y ya tenemos la primera estructura de nuestro quadcopter terminada.

Seguidamente vamos a añadir a cada uno de los componentes sus propiedades físicas, como el material o la densidad para que el programa procese y nos devuelva datos como su peso o la situación del centro de masas, información importante para el diseño de los motores y la validación del modelo.

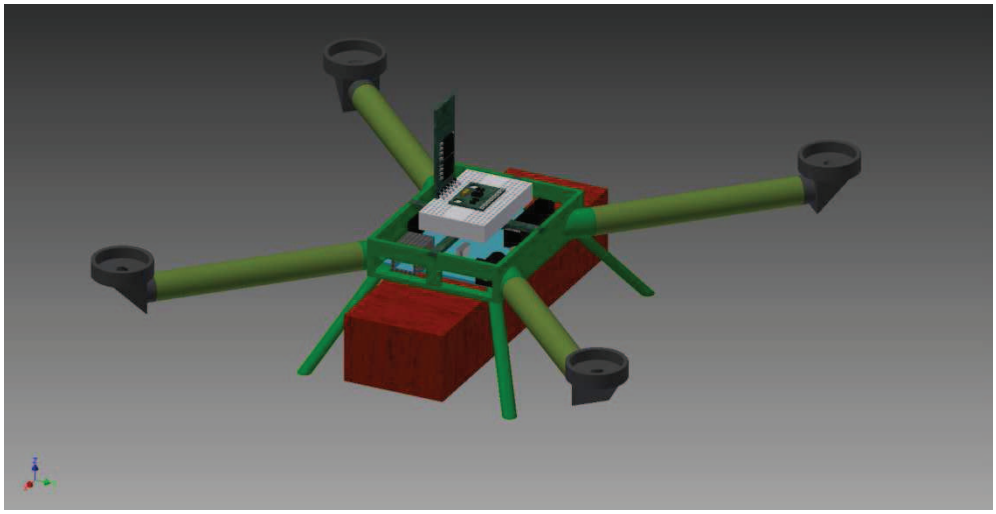


Fig 45. 1º Modelo terminado

De la estructura conocemos el material, que va a ser en principio Polietileno de alta densidad, un polímero plástico que se puede aplicar en una impresora 3D , con una densidad de unos 950 kg/m^3, o sea menor que la del agua.

Tiene la pega de que a temperaturas mayores de 50 ºC, y con luz directa a temperaturas ordinarias, se produce la degradación de sus moléculas, pero como en principio el quadcopter es para vuelos interiores y, en el caso de exteriores no van a ser muy prolongados, esto no va a ser un problema. En las propiedades del modelo introduciremos la densidad. El programa ya se encargará de obtener el peso en función del volumen.

Para los componentes electrónicos en cambio no podemos obtener una densidad, así que vamos a introducir su peso y el programa lo situará en el centro geométrico de cada pieza, por lo que a efectos prácticos nos sirve igual.

También vamos a aprovechar para obtener la inercia del sistema con respecto a los 3 ejes correspondientes a los dos pares de brazos, y al eje vertical que pasa por el centro.

Con esto obtenemos que el peso del conjunto es de **521 g**, y la posición del centro de masas es la que se muestra en la Fig 46.

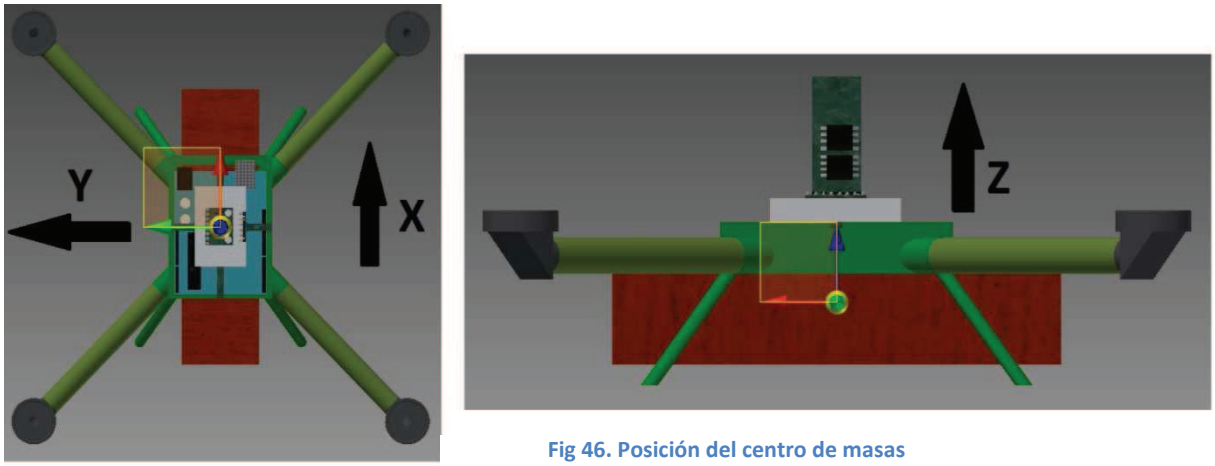


Fig 46. Posición del centro de masas

Posición del CDG: **-0,069i + 0,064j - 9.328k mm.**

Con esto vemos que la estructura está dimensionada correctamente al tener el centro de gravedad[12] completamente centrado en la vertical, con lo que conseguimos que todos los motores trabajen igual para mantener la posición de reposo. Además lo tenemos 9,328 mm por debajo de la base, lo que afecta positivamente a la estabilidad. Ahora con el peso podemos pasar a buscar unos motores adecuados.

2.3) Selección y validación de propulsión

Ahora que ya tenemos una estimación de la carga a mover por los motores (**521 gramos**) ya podemos empezar a buscar en el mundo comercial unos que tengan suficiente empuje como para levantarla y con bastante empuje como para poder moverse ágilmente.

Para ello lo primero es decidir qué clase de motores queremos entre las dos opciones vistas en el apartado anterior, si DC o Brushless. En general un motor Brushless es mejor que uno DC por tener más potencia y menos desgaste, y aunque sea un poco más caro vale la pena invertir en estos motores más modernos y potentes, así que vamos a diseñar el aparato con ellos.

Para empezar vamos a aprovecharnos de la gran herramienta que tenemos a nuestra disposición que es internet y buscar páginas web donde vendan material de aeromodelismo, que es un área donde se puede encontrar gran variedad de motores eléctricos del tamaño y potencia que queremos.

Buscando en la web hemos encontrado una página americana dedicada exclusivamente a los motores Brushless para aeromodelismo con páginas completas de detalles sobre ellos, por lo que vamos a utilizarla para buscar nuestros propulsores.

www.cobramotorsusa.com

[12] El centro de gravedad de un cuerpo es el punto donde situaríamos una masa puntual de igual valor que el cuerpo en cuestión para que la suma de momentos de las infinitas masas puntuales que forman el cuerpo fueran iguales al momento originado por dicha masa. El centro de gravedad coincide con el centro de masas en cualquier entorno con gravedad uniforme, y coincide también con el centro geométrico en el caso de que la densidad del cuerpo sea también uniforme.

Ya tenemos de donde obtener los motores, pero tenemos que definir ciertos criterios para escoger los que mejor satisfacen la necesidad planteada.

¿Qué criterios seguimos para buscar nuestros motores?

La respuesta viene en parte del resultado del apartado anterior, y es **521 g.** Sí, el peso que debemos levantar debe ser el primer filtro para buscar estos motores. Como tendremos 4 motores ese peso se divide entre ellos, correspondiéndole a cada uno:

$$E_m = \frac{P_{est}}{4} = \frac{521}{4} = 130.25\ g$$

E_m= Empuje del motor

P_{est}= Peso de la estructura

Así al menos cada motor debe levantar 130.25 g de la estructura, pero no sólo está el peso de la estructura, sino también el peso del propio motor y de su control electrónico de velocidad, así que modificamos la fórmula anterior:

$$E_m = \frac{P_{est}}{4} + P_{m+}P_{CE} = \frac{521}{4} + P_{m+}P_{CE} = 130.25 + P_m + P_{CE}\ g$$

P_m= Peso del motor

P_{CE}= Peso del control electrónico

El problema ahora es que no conocemos ni P_m ni P_{CE} así que tenemos que dejarlos en función e ir iterando probando con varios motores. No obstante aún hay un último aspecto a tener en cuenta, pues si buscamos un motor que cumpla perfectamente con E_m invertiremos todo el empuje en compensar el peso total del quadcopter. Si recordamos las primeras clases de física sabremos que el peso es una fuerza, y las unidades de fuerza son newtons (N), que son $kg \cdot \frac{m}{s^2}$. Los kg es la masa del quadcopter, pero los $\frac{m}{s^2}$ son la aceleración de la gravedad (9.81 $kg \cdot \frac{m}{s^2}$), lo que significa que con ese empuje lo único que conseguiremos será compensar la aceleración de la gravedad, pero no mover el aparato arriba y abajo, y sin eso al mínimo desplazamiento lateral se nos irá al suelo. Así que antes de ponernos a buscar nos falta un último parámetro por añadir a nuestra ecuación, y es la aceleración vertical que queremos alcanzar con nuestro quadcopter. En nuestro caso hemos decidido que nuestro quadcopter debe poder ascender con una aceleración de igual valor a la de la gravedad, o lo que es lo mismo, debe poder sustentarse en el aire lastrando una carga igual a su peso. Así que la fórmula que nos queda es:

$$E_m = \frac{P_{est}}{4} + P_{m+}P_{CE} + E_{ex} = 2 \cdot \left(\frac{521}{4} + P_{m+}P_{CE}\right) = 2 \cdot (130.25 + P_m + P_{CE})\ g$$

E_{ex}= Empuje extra

Para ayudarnos con el proceso iterativo vamos a utilizar una hoja de Excel con 4 columnas.

En la primera pondremos el motor + hélice + Control electrónico de velocidad correspondiente a los datos de la fila. En la segunda y la tercera ingresaremos los pesos del motor y del control electrónico respectivamente. El control electrónico será el más barato que soporte la

intensidad necesaria para el motor escogido, es decir, el correspondiente a un motor cuya intensidad máxima es de 25 A será el ESC Cobra 33 A.

En la cuarta ingresaremos el empuje de ese motor con la hélice escogida y en la quinta el precio. En la sexta nos aparecerá el empuje resultante de la resta entre el que tiene el motor y el que necesitamos mediante la fórmula anterior. De forma que si nos sobra empuje aparecerá en verde, y si nos falta aparecerá en rojo.

El siguiente criterio es el empuje que sobra por euro que cuesta. Es decir, ordenaremos los modelos según una séptima columna que mostrará el valor del empuje que nos sobra dividido entre el precio del conjunto motor + ESC correspondiente, de forma que el motor que escogeremos será el que más empuje extra nos dé por € o el que a mismo empuje extra cueste menos.

Después de buscar entre la gran variedad de opciones del fabricante nos hemos dado cuenta que tiene motores realmente potentes, ya que nos sobra con la serie más baja de motores que tiene. Hemos introducido 6 modelos en la tabla de Excel realizada obteniendo lo siguiente:

Tabla 8. Selección de motor

Motor + ESC + Hélice	Peso Motor (g)	Peso Control electrónico (g)	Empuje (g)	Precio Motor + ESC (€)	Empuje resultante	Empuje por €
C-2221/12 + ESC 33 A + APC 8x4-E	88	37,1	1378	60	867,3	14,455
C-2217/16 + ESC 22 A + APC 8x4-E	73	24,2	1160	50	705,1	14,102
C-2213/18 + ESC 22 A + GWS 8x4x3-DD	61	24,2	840	48	409,1	8,522916667
C-2208/34 + ESC 11 A + APC 8x4-E	**47**	**9,9**	**627**	**42**	**252,7**	**6,016666667**
C-2204/32 + ESC 11 A + GWS 6x3x3-DD	**22,5**	**9,9**	**499**	**37**	**173,7**	**4,694594595**
C-2202/70 + ESC 11 A + GWS 8x4-DD	15	9,9	266	34	-44,3	-1,302941176

De la tabla vemos que, menos uno de los motores, los demás cumplen con el requerimiento de empuje y algunos con creces, a costa también del precio.

Podríamos quedarnos con el C-2204/32 y el C-2208/34 (en negrita) como finalistas por cumplir y no dispararse de precio, pero contando que ya hemos tenido en cuenta el margen del doble de empuje en cálculo del empuje, y que con 4 de estos motores con sus respectivos ESC el presupuesto se nos dispara, vamos a escoger el más económico de los dos, el C-2204/32 con el Control electrónico de velocidad de 11 A y hélice GWS 6x3x3-DD.

Fig 47. Motor Cobra C-2204/32

Número de polos magnéticos	14
Valor Kv	1960 RPM por Voltio
Resistencia por fase	0.153 Ω
Corriente de máxima potencia	9,42 Amps
Potencia máxima con 2 celdas Li-Po	90 W
Potencia máxima con 3 celdas Li-Po	130 W
Peso	22,5 g
Diámetro exterior	27,0 mm
Longitud cuerpo motor	14,2 mm
Longitud total del eje	21,8 mm
Frecuencia de PWM	8 KHz
Precio	21 €

Tabla 9. Especificaciones motor C-2204/32

Fig 48. ESC Cobra 11 A

Peso	9.9 g
Corriente máxima	11 A
Voltaje operativo	6 a 12 V
Celdas Li-Po admitidas	2 o 3
Celdas Ni-XX admitidas	De 6 a 10
Celdas Li-Fe admitidas	2 o 3
Tamaño	30 x 18 x 10
Precio	16 €

Tabla 10. Especificaciones ESC 11 A

Fig 49. Hélice GWS 6x3x3

Tamaño	3 hojas de 6x3 cm
RPM Max	16667 rpm
Precio	1,40 €

Tabla 11. Especificaciones Hélice GWS 6x3x3

Hay que tener en cuenta al adquirir las hélices que para compensar el momento cinético del quadcopter al volar hay que hacer girar dos hélices en un sentido y las otras dos en otro. Por tanto tenemos que pedir dos hélices levógiras y dos dextrógiras.

Ahora que ya tenemos los motores y los ESC tenemos que buscar una batería que pueda dar la intensidad demandada y sea lo más barata y ligera posible.

Para buscar un modelo adecuado de batería vamos a utilizar una web americana dedicada a proyectos caseros de electrónica y radiocontrol:

www.hobbyking.com

A continuación debemos buscar una batería que tenga una tasa de descarga que permita tener en marcha todos los componentes y los 4 motores a máximo rendimiento a la vez. Para ello contamos la máxima intensidad con los 4 motores a plena potencia, que será 4 veces la potencia de cada uno: 9,42 x 4=**37.68 A.** A esta intensidad deberíamos sumarle la consumida por los otros elementos, pero como es del orden de miliamperios no necesitamos considerarla, no obstante tenemos margen al normalizar el valor a **40 A**.

El segundo punto a tener en cuenta ya zanjada la intensidad es la tensión. En el capítulo 1 hemos visto los tipos de baterías que podemos adquirir, y en este caso por su gran capacidad de carga y otras ventajas ya mencionadas vamos a optar por una batería de polímero de litio (Li-Po). Como decíamos antes, estas baterías están compuestas de celdas, siendo la tensión de cada una de ellas de 3.7 V, así que ahora debemos escoger el número de celdas de nuestra batería conectadas en serie para obtener más tensión. Cuantas más celdas más tensión, pero también más precio y peso. En las especificaciones del ESC y del motor Brushless nos aparece que se admiten baterías tanto de 2 como de 3 celdas, así que en principio cogeríamos la de 2 por ser más barata. No obstante debemos pensar que también debemos alimentar la placa Arduino, quien a su vez alimentará los otros 2 módulos, y aunque esta placa puede ser alimentada de 7 a 12 voltios, lo que nos permitiría utilizar 2 o 3 celdas, es más conveniente tener un margen de tensión para prever que la batería se descargue, así que definitivamente vamos a buscar una batería de 3 celdas con al menos una tasa de descarga de 40 amperios, o **Li-Po 3S,** de más o menos 400 gramos, que son los que hemos utilizado para el dimensionamiento de los motores, aunque con el margen de seguridad que hemos tomado en todas las operaciones difícilmente será esto un problema.

La tasa de descarga viene dada por un parámetro consistente en un número y una letra C mayúscula. Con ese número se puede obtener la intensidad máxima que produce una batería cada hora al multiplicar ese valor por la capacidad. Así, si tenemos una batería de 4000 mAh 40C, la intensidad máxima por hora que puede suministrar es de: (4000*40)/1000=160 A.

Como no tenemos altas necesidades en cuanto a una elevada tasa de descarga, vamos a buscar baterías con bajo índice de descarga pero que tengan alta capacidad.

Finalmente, después de ver varias secciones con índices de carga distintos nos hemos decidido por la **ZIPPY Flightmax 5000mAh 3S1P 20C.** Realizando la operación anterior de la tasa de descarga obtenemos que la batería puede entregar una intensidad máxima de: 5000·20/1000= **100 A**, con lo que tenemos más que suficiente.

Fig 50. Batería ZIPPY Flightmax 4000mAh 3S1P 40C

Capacidad	5000 mAh
Celdas en serie	3
Celdas en paralelo	1
Tensión	11,1 V
Tasa de descarga	20C
Peso	418g
Dimensiones	137x47x40
Precio	24 €

Tabla 12. Especificaciones Batería

Si nos fijamos en la tabla 7 donde resumimos los componentes para empezar el dimensionamiento al principio de este capítulo vemos que no nos hemos ido mucho con respecto a las dimensiones y al peso de la batería.

Con 5000 mAh de batería, si estuviésemos continuamente a la máxima potencia (9.42 A x 4) tendríamos una intensidad de 37,68 A. Si dividimos la capacidad entre este dato obtenemos la duración de la batería:

$$T_{bat} = \frac{Capacidad}{Intensidad} = \frac{5000\ mAh}{37.68\ A} \cdot \frac{1\ A}{1000\ mA} = 0.133\ h\ \approx \mathbf{8\ min}.$$

Lo que es mucho tiempo contando que es a pleno rendimiento, si utilizásemos sólo la mitad de la potencia podríamos estar unos 15 minutos, que no está nada mal.

Para concluir la definición de estos últimos componentes vamos a sumar todos los pesos y a compararlos con los 4 empujes para validar la propulsión:

Peso		Empuje	
Estructura y componentes sin batería	121	Motor 1	499
4 Motores	90	Motor 2	499
4 ESC	36	Motor 3	499
batería	418	Motor 4	499
TOTAL	665	TOTAL	1996
Coeficiente de seguridad	**3,001503759**		

Tabla 13. Balance pesos

Con esto vemos que tenemos 3 veces el empuje necesario para mantener en el aire el quadcopter, lo que nos da una alta maniobrabilidad.

Ahora que ya tenemos perfectamente definidos el resto de componentes es hora de volver al modelo 3D y ajustar el chasis a los nuevos elementos.

2.4) Diseño final del chasis

Para acabar de ajustar la estructura tenemos que adaptarla a los elementos modificados, que son la batería, los motores y los ESC con los que no habíamos contado.

En el caso de la batería el tamaño no ha variado mucho respecto a nuestra hipótesis inicial, y como está colocada en la parte inferior sin ninguna ranura no es necesaria hacer ninguna modificación. No obstante con respecto a los motores debemos modificar los soportes extremos para que encajen con su forma perfectamente, y tenemos que buscar un sitio también a los ESC que se acaban de subir al barco.

Antes de ponernos a modificar vamos a obtener los modelos 3D tanto de los motores como de los ESC para obtener un mejor ajuste. De estos últimos no hay problema porque no tienen que ir fijados al chasis como los motores, sólo sujetos, y además su forma es sencilla de desarrollar. De los primeros en cambio, necesitamos un modelo muy aproximado porque tendremos que dimensionar los agujeros para el atornillado y, con 1 mm que nos vayamos ya no ajustará.

Como no tenemos acceso a ningún plano de los motores vamos a obtener las cotas a partir de las que ya tenemos por especificaciones y por extrapolación en unas fotografías de la planta y el perfil obtendremos las demás.

Fig 51. Medidas del motor

Esto no deja de ser una aproximación, pero es la manera más acertada de obtener las medidas sin tener el motor físicamente. Como vemos en la imagen el diámetro del eje de la Planta es de 8,565 mm, y en el perfil de 7,271 mm, pero sabemos que ese diámetro es de 3 mm por las especificaciones, así que ya tenemos que la relación entre la medida de la foto y la medida real es de $R_{planta} = \frac{Real}{Foto} = \frac{3}{8.565} = \mathbf{0.35}$ para la planta y de $R_{perfil} = \frac{Real}{Foto} = \frac{3}{7.271} = \mathbf{0.413}$

Ahora sólo tenemos multiplicar cada cota por el coeficiente correspondiente a su vista y obtendremos las cotas reales.

Diámetro del eje: 3 mm

Diámetro del cuerpo del motor: 27 mm

Longitud total del eje: 21.8 mm

Longitud del cuerpo del motor: 14.2 mm

Longitud de la patilla: R_{planta} x 9.713 = 3.4 mm

Diámetro del agujero de la patilla: R_{planta} x 5.969 = 2.09 mm

Espesor de la patilla: $R_{perfil}\ x\ 3.218 = 1.33\ mm$

Distancia del perímetro del motor al centro del agujero de la patilla: R_{planta} x 3.167 = 1.11 mm

Con estas medidas ya podemos empezar con el modelo 3D del motor, y al terminarlo así es como nos quedan este y el ESC:

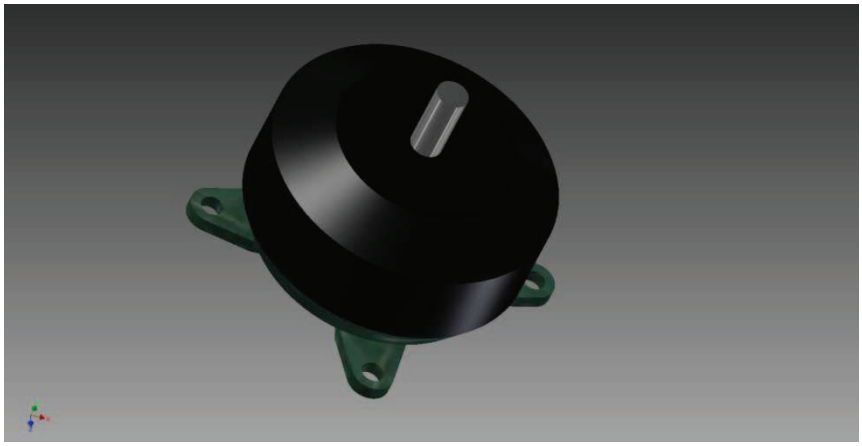


Fig 52. Modelo 3D del motor

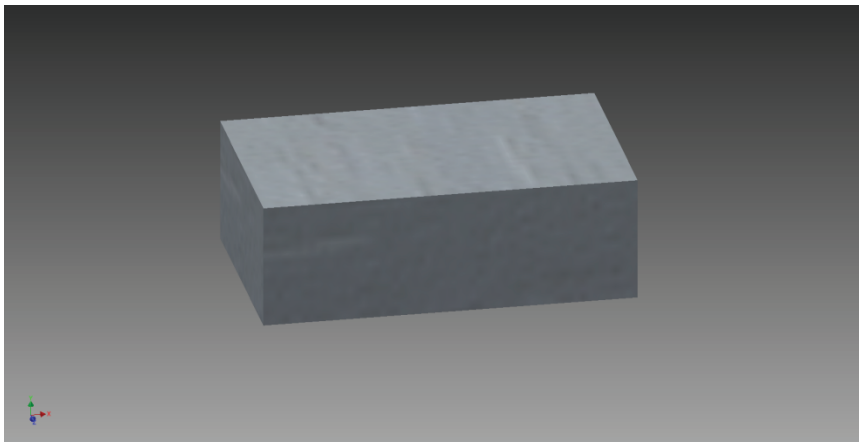


Fig 53. Modelo 3D del ESC

Seguidamente toca rediseñar por completo el soporte de los motores con una superficie plana para atornillarlos y cambiar el hueco por el que discurre el cable pues del motor sale por un lateral. En cuanto a la localización de los ESC parece que la mejor opción es situarlos en los brazos del quadcopter, así que el perfil tubular que pensamos ya no nos sirve, necesitamos un perfil plano para situar el control electrónico.

A diferencia del primer diseño, esta vez vamos a incluir el soporte del motor, el brazo y la estructura central en un mismo modelo para así también evitarnos material en las juntas. Hemos adaptado los agujeros de la estructura central también al modificar el acople que tenía para los antiguos brazos tubulares, y así ha quedado el conjunto:

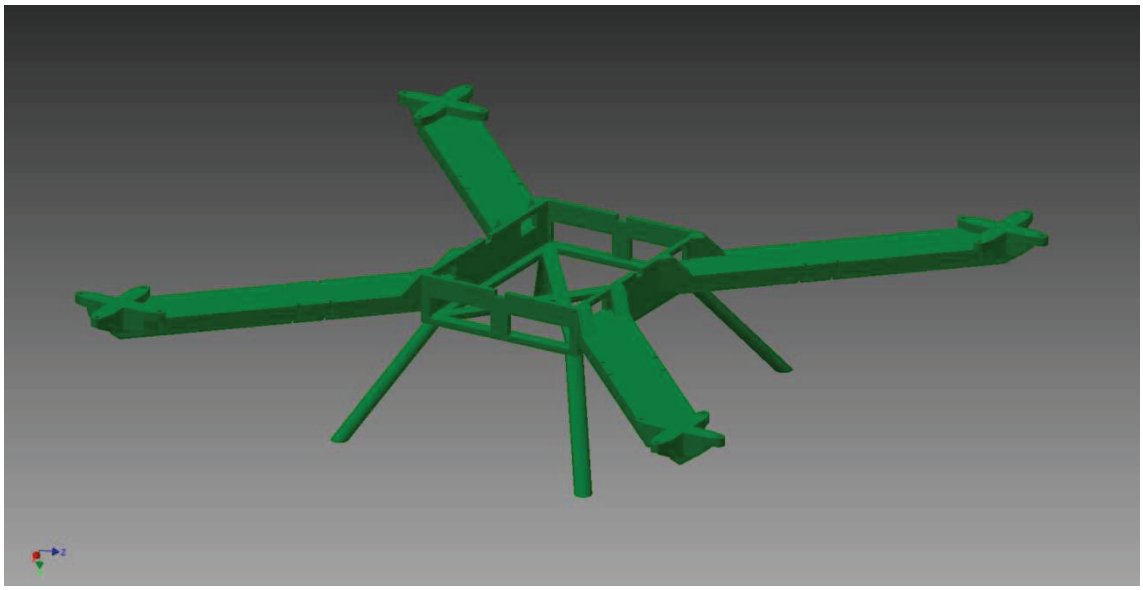


Fig 54. Chasis final

Como se puede apreciar están los soportes, brazos y central todo en una misma estructura que puede imprimir íntegramente una impresora 3D sin tener que recurrir a uniones. Hemos hecho unas muescas en los brazos en la zona donde irá el ESC para disponer un par de gomas elásticas que sujetarán el mismo al brazo impidiendo su movimiento.

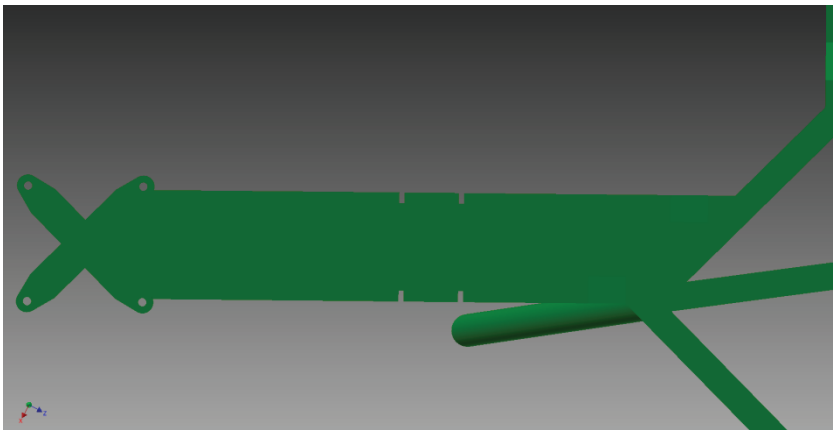


Fig 55. Muescas para ESC

Ya ha llegado el momento de ensamblar todos los componentes incluyendo motores y ESC a la estructura nueva.

Hemos tenido que alargar las patas del tren de aterrizaje por el mayor grosor final de la batería. Además la cruz que sujeta la protoboard ahora tiene unos ganchos en las puntas para impedir que la caja que contiene la placa Arduino se abra por flexión de los brazos nuevos más pesados.

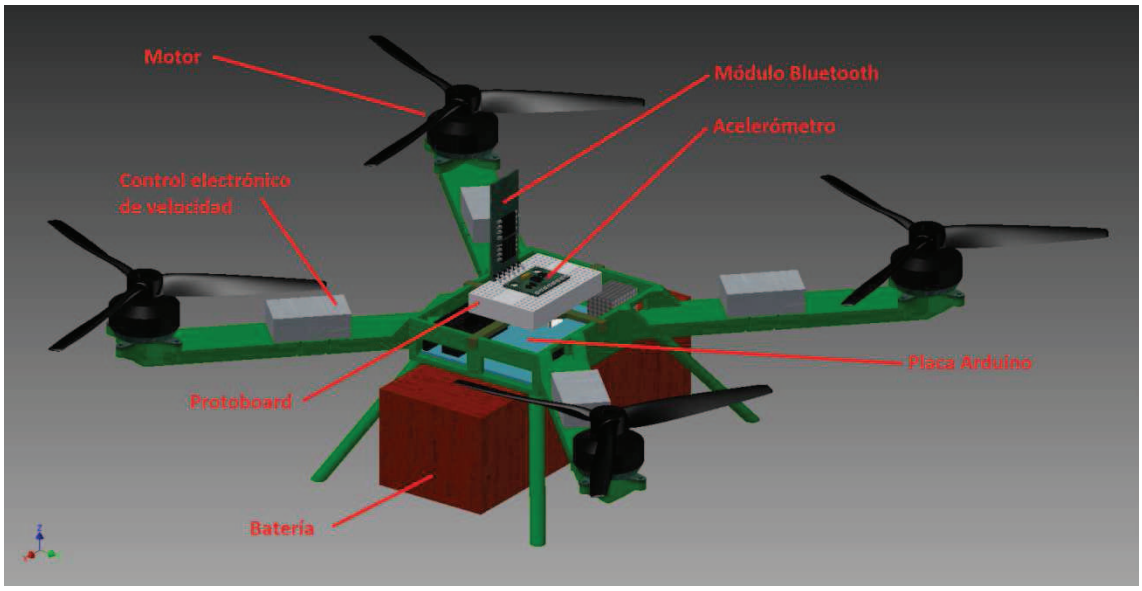


Fig 56. Ensamblaje final

Hemos añadido cuatro hélices parecidas a las que hemos obtenido en el dimensionamiento de la propulsión, con la única intención de tener una imagen final del modelo

En las propiedades físicas del modelo nos aparece ahora que la masa total es de 679 gramos, sólo 14 gramos de diferencia con respecto a lo calculado en la tabla 13, contando que hemos añadido las hélices, por lo que deducimos que hemos diseñado el modelo correctamente.

En cuanto al centro de masas se sitúa ahora en la posición (0,030i - 0.207j – 5.919k) mm, siendo aceptable por estar situado en la vertical central de nuestro modelo, prácticamente igual que en el del primer diseño. Con esto concluimos el capítulo 2 *Diseño del chasis y selección de componentes.*

En la página siguiente aparece una imagen apaisada del modelo del quadcopter renderizado, y en el documento **Planos** adjunto a la memoria se pueden encontrar los planos detallados de las piezas y las especificaciones de los componentes utilizados.

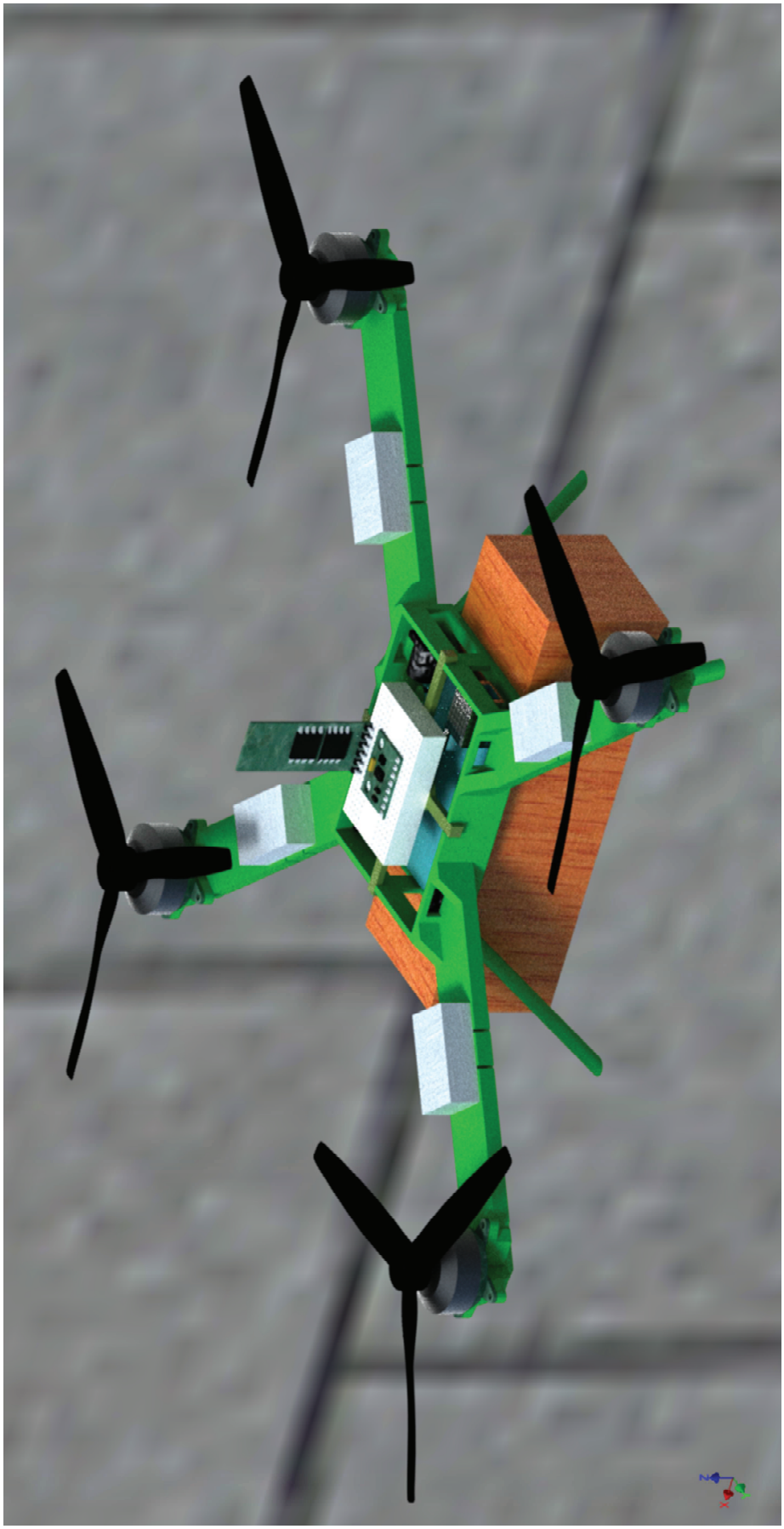


Fig 57. Renderizado de diseño finalizado

3) Implementación del control de estabilidad

3.1) Introducción

En este tercer capítulo nos vamos a centrar en lo último que queda por diseñar de nuestro quadcopter: el software.

El único componente programable del quadcopter es el procesador, integrado en la placa Arduino; pues tanto el acelerómetro como el módulo bluetooth ya llevan incorporada la programación adecuada para realizar su función, y lo único que haremos con ellos es leer datos de una manera determinada.

En el primer apartado desarrollaremos el código encargado de obtener datos del acelerómetro y el módulo bluetooth y procesarlos para elaborar una respuesta con la máxima velocidad de respuesta y manteniendo el error de posición=0. Para ello implementaremos un controlador PID, con parte proporcional necesariamente y parte integral necesaria para mantener el error de posición=0. Con respecto a la parte derivativa decidiremos si utilizarla o no en el mismo apartado.

Para probar las diversas opciones que seguramente obtengamos de la elaboración del PID, y ya que no vamos a construir el quadcopter por razones económicas, diseñaremos, obtendremos los planos y fabricaremos un balancín de ensayos donde probaremos nuestro regulador. El balancín sólo nos permitirá el estudio con respecto a uno de los ejes, así que cuando decidamos cual utilizar obtendremos una aproximación para adecuarlo al control de los 3 ejes. Estará construido con madera de balsa por su bajo precio y facilidad de proceso y tendrá un motor en cada extremo y el acelerómetro que ya hemos adquirido en su eje de rotación.

En realidad hay todavía un elemento más programable que es la aplicación de control móvil, pero como veremos en su correspondiente apartado, la idea del desarrollo de una aplicación Android que hubo en un principio ha quedado desestimada por el tiempo que llevaría aprender el lenguaje de programación JavaScript necesario para programar en Android. Por tanto durante el presente capítulo, en el apartado de comunicación sólo se desarrollará el trozo de código encargado de configurar el módulo bluetooth y ajustar los datos que vienen de la aplicación de control al resto del programa.

Finalmente elaboraremos el código final que compondrá el programa uniendo los fragmentos anteriores adecuando las variables y terminando con la declaración de las mismas y de las librerías necesarias.

3.2) Código de estabilidad y simulación

Como dijimos en la programación del capítulo, en este primer apartado vamos a obtener el regulador PID encargado de controlar la respuesta de los motores ante las señales de control del acelerómetro y las órdenes del dispositivo móvil.

Para ello primero vamos a diseñar y fabricar el balancín que utilizaremos para ensayar los reguladores.

3.2.1) Diseño y fabricación del balancín de ensayos

Para diseñar el balancín vamos a utilizar el mismo software que ya usamos en el capítulo anterior para obtener el chasis del quadcopter.

Así la estructura que necesitamos es muy simple, un brazo que pueda albergar en sus extremos 2 motores equidistantes del centro de balanceo del mismo y soportado por una base que permita solamente ese balanceo, impidiendo el resto de movimientos.

En cuanto al material no necesitamos complicarnos mucho pues las solicitaciones de carga del brazo y del soporte van a ser muy bajas y lo único que necesitamos es que tenga suficiente rigidez para que el brazo no se doble, así que el material vamos a escogerlo en función de la facilidad de fabricación y el precio.

La mejor opción, ya que nos permite la fabricación propia en un taller simple y además resulta muy barata es la madera de balsa. Esta madera recibe el nombre por el árbol del que se obtiene, el Balso, que crece en la selva subtropical del ecuador. La principal ventaja de esta madera es que es la más ligera que se conoce, con una densidad de 0.1 a 0.15 g/cm^3, más ligera incluso que el corcho. Es muy fácil de trabajar al comercializarse en láminas de unos pocos milímetros que se cortan fácilmente con un serrucho de marquetería o una dremel. No tiene mucha resistencia y se puede doblar con la fuerza adecuada, pero el bajo precio que tiene lo compensa.

Debemos tener en cuenta que al estar trabajando con una lámina de madera (que finalmente será de 3mm de espesor), vamos a necesitar disponer piezas cruzadas para sostener motores, reforzar el brazo y tener un hueco por el que discurra el eje de balanceo.

Fig 58. Madera de balsa

Las uniones entre piezas las vamos a realizar en principio con adhesivo polivinílico, más conocido como cola blanda, idóneo para el trabajo con materiales porosos como nuestra madera de balsa. No obstante en las zonas en las que necesitemos mayor rigidez haremos ranuras del grosor de la lámina en las que encajen las otras piezas.

Vamos a ponernos en el diseño ya. Primero

diseñamos el brazo que va a albergar los motores circulares en los extremos. En el centro hemos hecho una zona más ancha porque ahí es donde irá la placa de prototipos que sostendrá el acelerómetro. Los motores tienen un diámetro de 12mm y un largo de cuerpo de 15mm, así que dispondremos dos agujeros circulares de 12mm donde se sujetarán. Además hemos hecho 4 agujeros rectangulares donde se inserta un nervio inferior para darle rigidez que veremos más adelante.

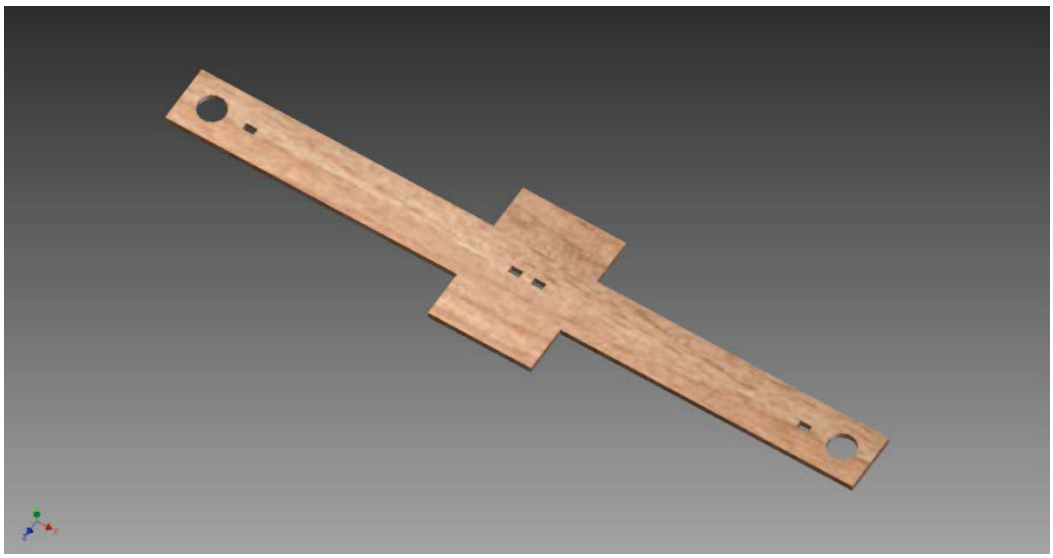


Fig 59. Brazo del balancín

Ahora vamos a diseñar el nervio inferior que hemos mencionado y que dará la rigidez al brazo que compensará la flexibilidad de la madera. En los extremos hemos dispuesto una ranura que servirá de sujeción al motor, y dos más pequeñas a cada lado de la anterior por donde discurrirá un pequeño cable o cinta para fijar el motor. También hemos hecho los 4 puntos que se insertarán en los huecos del brazo para quedar más estable, y un agujero circular en el centro que servirá de cuenca deslizante para el eje de balanceo, y dos huecos a sus lados para poner 2 barras de refuerzo que veremos a continuación.

Fig 60. Nervio longitudinal

Para mejorar la sujeción de la parte central del brazo que sostiene la protoboard con el acelerómetro, y para disponer más guías para el eje, vamos a colocar dos refuerzos laterales paralelos al nervio, que van a sujetar dos barras transversales que dan rigidez al conjunto central.

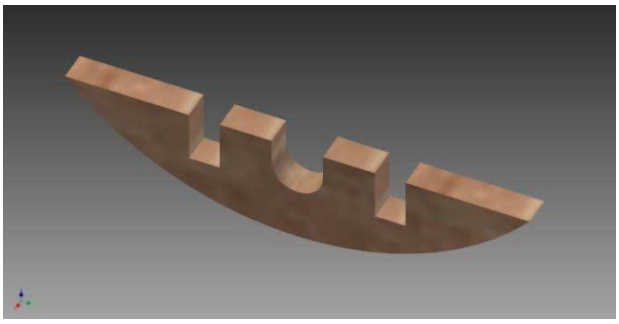


Fig 61. Refuerzo lateral

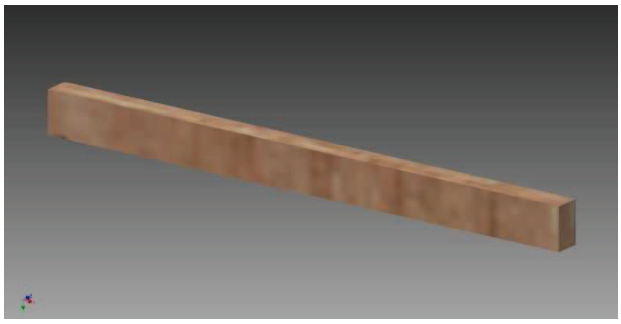


Fig 62. Barra transversal

A continuación tenemos 2 vistas diferentes del resultado final del brazo del balancín.

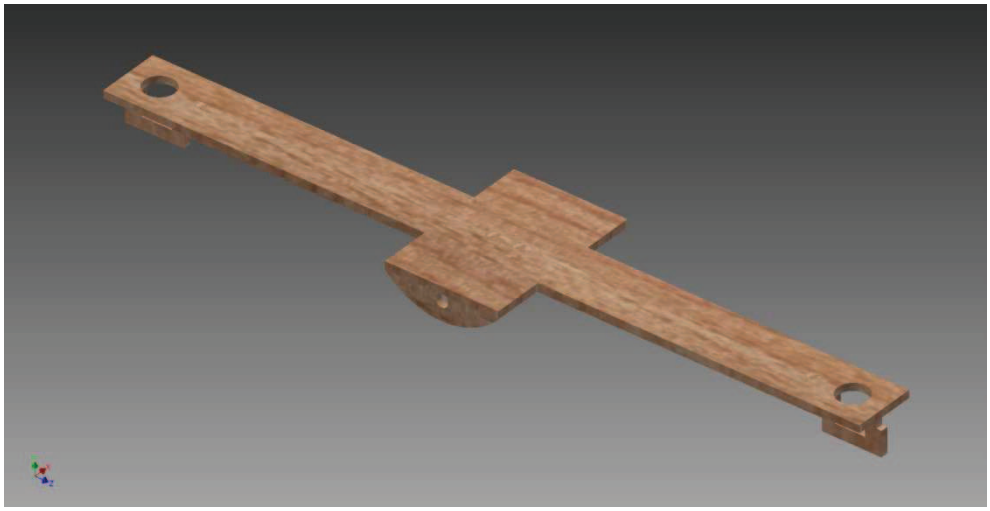


Fig 63. Vista superior brazo

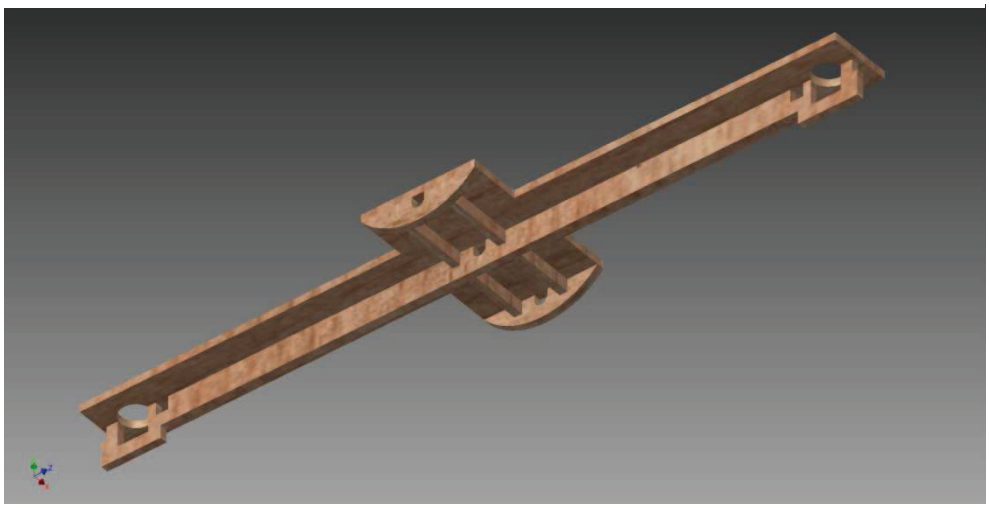


Fig 64. Vista inferior brazo

Ahora que tenemos el brazo sólo nos falta el soporte, que no necesita demasiada explicación ni exactitud en el diseño, pues la única condición que debe cumplir es la de sujetar el eje que a su vez sujetará el brazo por el centro, así que simplemente vamos a mostrar el modelo del soporte y el modelo del balancín terminado.

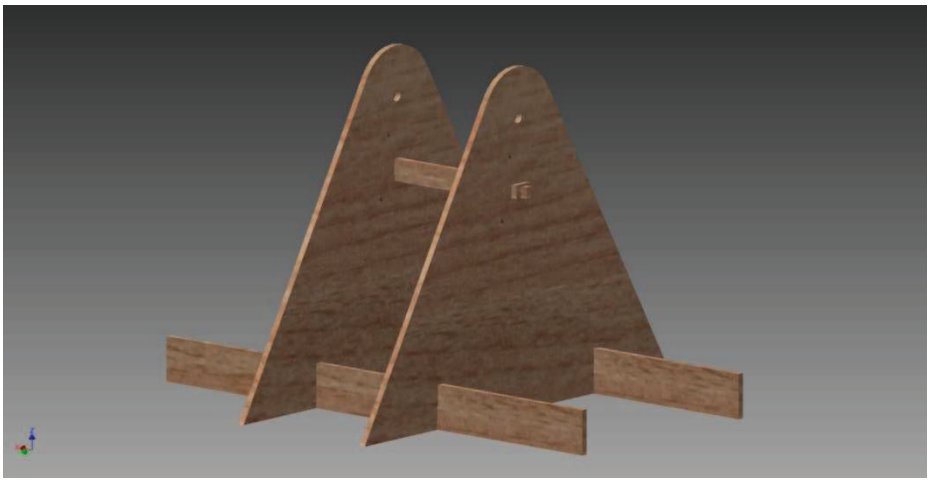


Fig 65. Soporte balancín

Fig 66. Balancín vista superior

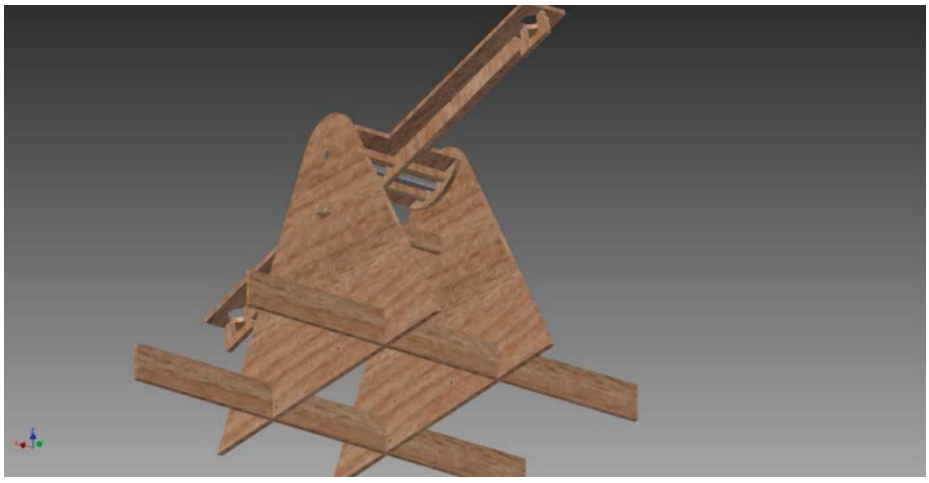


Fig 67. Balancín vista inferior

Ahora que tenemos el diseño vamos a colocar las piezas en un plano a escala real de tamaño A3, que imprimiremos y colocaremos sobre una lámina de madera de balsa con un papel de calco en medio, y vamos a repasar las piezas con un lápiz para que se calquen en la madera gracias al papel de calco.

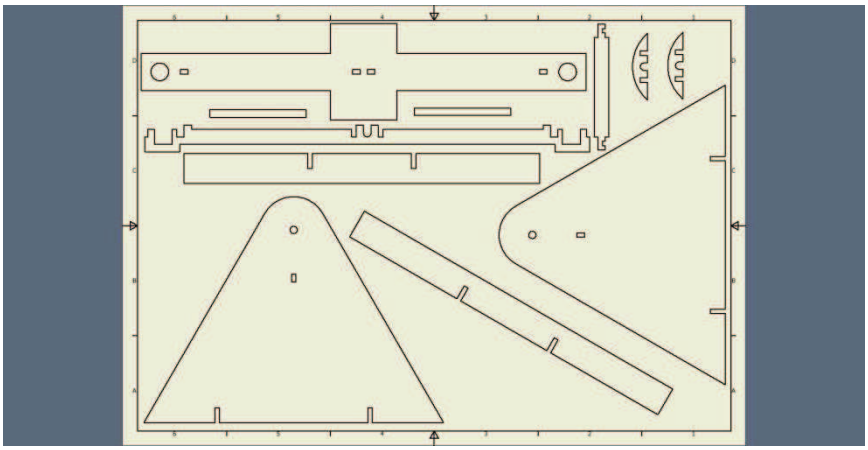


Fig 68. Plano de las piezas del balancín

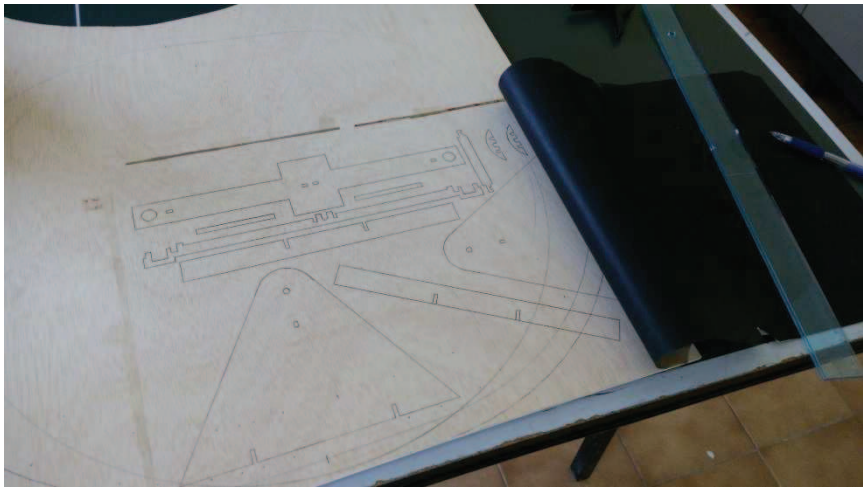


Fig 69. Plano calcado a la lámina de madera de balsa

Fig 70. Recorte de las piezas complicadas con una Dremel

Para completar nuestro sistema de ensayos nos falta lo más importante: la electrónica y los motores.

Como hemos visto en el capítulo anterior el precio de todos los componentes del quadcopter era bastante elevado, así que no se va a fabricar entero en este proyecto. En vez de eso, y puesto que para elaborar el control de estabilidad se necesitan pruebas con un modelo real, hemos diseñado el balancín, y vamos a ponerle 2 pequeños motores DC de 350 mA de corriente máxima, lo suficiente para hacer balancearse el brazo. También el acelerómetro que decidimos anteriormente, el MPU-6050, y conectaremos todo a la placa Arduino UNO que también hemos adquirido.

No obstante todavía nos falta la electrónica que ha de comunicar la placa con los motores, pues estos necesitan cerca de 10 veces más intensidad de la que puede suministrar cada pin del Arduino. Para solucionar este problema hemos optado por utilizar transistores.

Un transistor bipolar de unión, o BJT (por sus siglas en inglés) es un elemento electrónico semiconductor que produce una señal de salida en función de otra de entrada. Se utilizan como interruptores o como amplificadores de señal, y hay de dos tipos, NPN y PNP. En nuestro caso los vamos a utilizar para amplificar nuestra señal de salida del pin de Arduino que llega a 40 mA hasta los 350 mA que requiere cada motor a máxima potencia. Para esto hemos adquirido 2 transistores (uno por cada motor) **BC338**.

Fig 71. Transistor BJT

Necesitaremos otro elemento en nuestro circuito de control por estar trabajando con motores, y es un diodo situado paralelo a cada motor en sentido inverso, para dejar pasar la corriente de retorno del motor y proteger así al transistor de una corriente inversa. Además necesitaremos dimensionar las resistencias de base para ajustar la ganancia, pero eso lo haremos más adelante. Por ahora el circuito que tenemos es el que sigue:

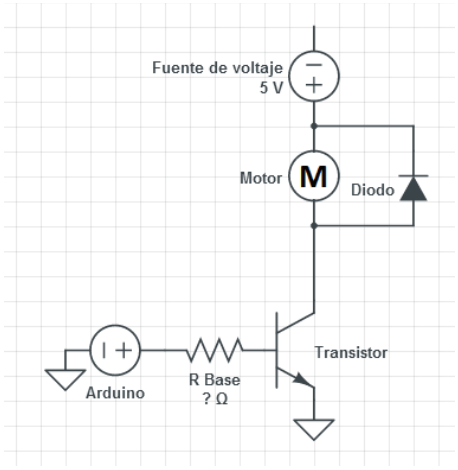


Fig 72. Circuito ensayo

Hemos hecho ya las conexiones con unas resistencias estimadas de unos 100 kΩ, y el resultado es:

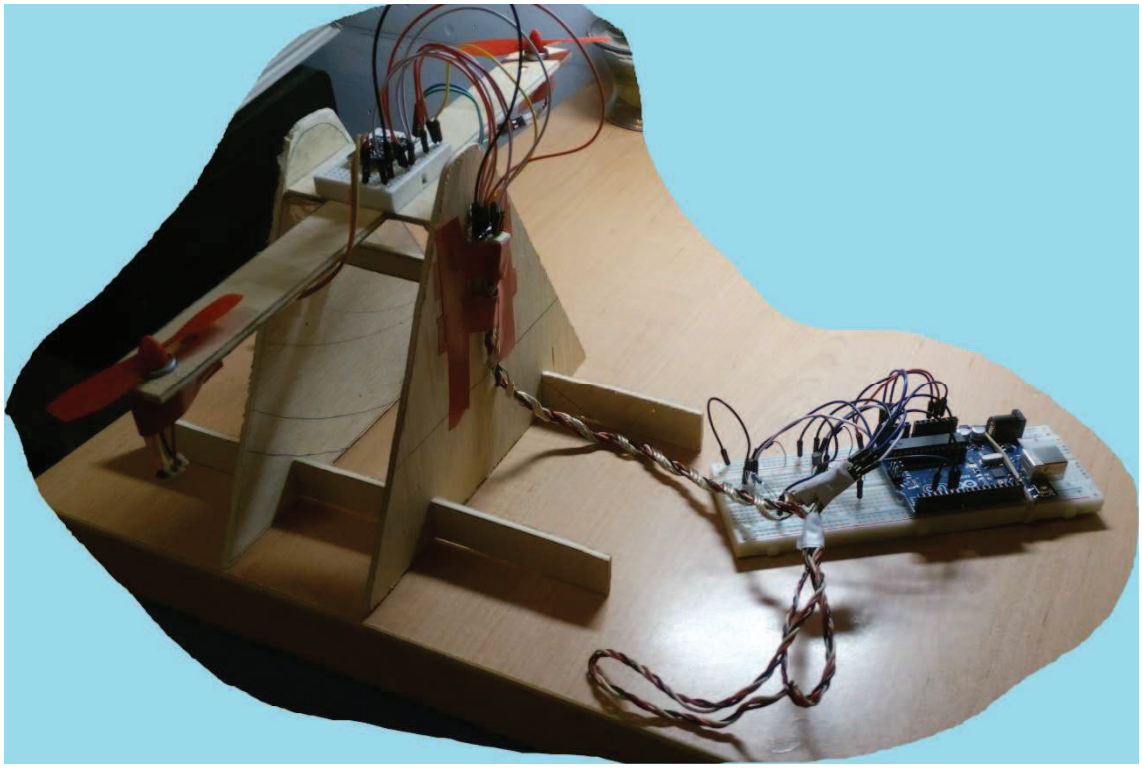


Fig 73. Sistema de ensayo terminado

Posteriormente hemos hecho varios ensayos para determinar la resistencia necesarias para ajustar la intensidad requerida en la base del transistor, que ha resultado ser de 55 Ω para que la corriente de base sea de 36.9 mA, dejando margen a los 40 mA que no han de rebasarse en cada pin de Arduino, para 5V en la salida PWM.

Con esta intensidad en base obtenemos una corriente en colector de 230 mA, y probando el sistema con estos parámetros hemos visto que los motores tienen suficiente fuerza para mover el balancín, así que nuestro circuito queda definido perfectamente y ya podemos ponernos con el código de control.

3.2.2) Implementación PID

Ya tenemos correctamente conectado nuestro sistema con los motores y el acelerómetro en sus pines correspondientes. Así los pines 5 y 6 digitales del Arduino están conectados a las bases de los dos transistores. Los pines A5 y A4 del Arduino se conectan a los pines SCL y SDA del acelerómetro respectivamente y el pin INT del acelerómetro sale un cable hasta el pin digital 2 de Arduino. Vcc y GND del acelerómetro se conectan a 3.3V y a GND del Arduino, y del pin 5V del Arduino sale un cable que va a ser la alimentación de los motores conforme se vio en el gráfico del circuito.

Para empezar debemos conectar la placa Arduino al ordenador, instalar los drivers y el programa y ya podemos empezar.

En este apartado vamos a diseñar un programa que mantenga el balancín en horizontal y lo haga con una respuesta lo más rápida y estable posible, así que necesitaremos los datos correspondientes al eje del acelerómetro coincidente con el eje del brazo, que en este caso es el eje Y. Así, la variación en la lectura de aceleración del eje Y ocasionada por la gravedad nos permitirá saber el ángulo del brazo y enviar órdenes a los motores en consecuencia. En el apartado de comunicación añadiremos un trozo de código que cambiará el concepto de horizontal al Arduino, y así controlaremos los movimientos.

Vamos a empezar declarando las librerías que vamos a necesitar. Una librería es un archivo que contiene una serie de funciones para dotar de funcionalidades extra a nuestro sketch.

Para poder utilizar el acelerómetro con sus comandos por comunicación serie vamos a necesitar 3 librerías que son MPU6050.h, Wire.h y I2Cdev.h. Por tanto nuestro código empezará por:

```
#include <MPU6050.h>

#include <Wire.h>

#include <I2Cdev.h>
```

También debemos declarar que tenemos un acelerómetro conectado y denominarlo de alguna manera. En este caso vamos a llamarlo “mpu”, y el comando para declararlo es el MPU6050. Por tanto el código sigue:

```
MPU6050 mpu;
```

Ahora viene el turno de la declaración de las variables que vamos a utilizar.

El acelerómetro devuelve los datos de las aceleraciones y los giros en valores enteros de 16 bits. 3 aceleraciones y 3 giros, por tanto 6 variables de 16 bits que debemos declarar. Las vamos a llamar **ax**, **ay**, **az**, **gx**, **gy** y **gz** (Es obvio qué dato albergará cada una). Los enteros de 16 bits se declaran con el comando *int16_t "NOMBREVARIABLE", "NOMBREOTRAVARIABLE";*

También declaramos una variable entera de 8 bits llamada "estado".

Por tanto la declaración de variables queda:

```
int16_t  ax, ay, az, gx, gy, gz;

int estado=1;
```

Con esto termina nuestra etapa de declaración. Ahora vamos a las dos funciones imprescindibles de Arduino: setup() y loop().

En **setup()** van a ir todas las instrucciones correspondientes a la configuración del programa, y solo se van a ejecutar una vez. En cambio, las instrucciones que estén en **loop()** van a ejecutarse secuencialmente ininterrumpidamente. Delante de cada función vamos a poner un *void*, pues son funciones que no devuelven ningún valor, y detrás dos corchetes {} que engloban la función.

Así nuestra función de configuración queda:

```
void setup()                    Declaramos la función setup.
{                               Empezamos a redactar las instrucciones de la función.
  Wire.begin();                 Comando necesario para la comunicación serial con el mpu.
  Serial.begin(9600);           Iniciamos la comunicación serial con una tasa de 9600 baudios.
  pinMode(5, OUTPUT);           Declaramos el pin digital 5 como salida.
  pinMode(6, OUTPUT);           Declaramos el pin digital 6 como salida.
  mpu.initialize();             Inicializamos el acelerómetro.
mpu.setSleepEnabled(false);     Nos aseguramos de que el acelerómetro no está en estado sleep.
}
```

Con esto ya tenemos la configuración de nuestro programa. Ahora vamos con la función loop, que es la que estará ejecutándose ininterrumpidamente.

```
void loop()                     Declaramos la función loop.
{                               Empezamos a redactar las instrucciones de la función.
  if(Serial.available()>0){     Si hay datos para leer por el puerto serial entonces:
  estado= Serial.read();        Leo el primer dato y lo guardo en la variable "estado".
  Serial.flush();               Borro lo que hay en el buffer del puerto serie.
  }                             Termino las iteraciones de la condición anterior.
```

```
if (estado==1){                    Si he leído un 1 en la variable estado entonces:
mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
                                   Leo los valores de aceleración y giros y los guardo en las variables que
                                   he declarado al principio.
  Serial.print(ax);                Muestro por el monitor serial el valor de la aceleración en x
  Serial.print("  ");              Muestro un espacio en blanco en el monitor serial.
  Serial.print(ay);                Muestro por el monitor serial el valor de la aceleración en y.
  Serial.print("  ");              Muestro un espacio en blanco en el monitor serial.
  Serial.print(az);                Muestro por el monitor serial el valor de la aceleración en z.
  Serial.print("  ");              Muestro un espacio en blanco en el monitor serial.
  Serial.print(gx);                Muestro por el monitor serial el valor del giro en x.
  Serial.print("  ");              Muestro un espacio en blanco en el monitor serial.
  Serial.print(gy);                Muestro por el monitor serial el valor del giro en y.
  Serial.print("  ");              Muestro un espacio en blanco en el monitor serial.
  Serial.println(gz);              Muestro por el monitor serial el valor del giro en z y salto de línea.
  Delay(50)}                       Hago una pausa de 50 ms para dar tiempo al acelerómetro a tomar el
                                   siguiente lote de datos.
}                                  Termino las instrucciones de la función loop.
```

Con esto tenemos ya la función terminada, así que vamos a probar el código con el brazo horizontal.

Hemos hecho un ensayo de 10 segundos con el brazo horizontal, y hemos tomado 100 series de valores correspondientes a los 5 segundos centrales (100 valores a 50ms de periodo de muestreo). Hemos introducido esos valores en MATLAB y formado con ellos una gráfica.

En la gráfica vemos como hay cierta interferencia en la toma de datos tanto en las aceleraciones como en los giros, así que vamos a tener que filtrar las señales de alguna manera. Los colores corresponden el azul al eje X (eje del brazo), el rojo al eje Y (perpendicular al X y con el que forma el plano del brazo) y el verde al eje Z (vertical).

Como podemos ver, estando horizontal en la gráfica de las aceleraciones aparecen los valores en eje X y eje Y prácticamente despreciables, mientras que los valores del eje Z son mucho mayores, ya que está detectando la aceleración de la gravedad. En cuanto a los giros vemos como los valores de Y y de Z se mantienen constantes con holgura, pues deberían ser 0, pero los valores en X varían erróneamente, por lo que debe estar descalibrada la lectura en ese eje. No obstante no nos afecta pues solo queremos el giro en el eje Z para la guiñada.

-1924	-1460	14848	-447	73	42
-1816	-1432	14852	-289	127	29
-1812	-1336	14776	-376	79	64
-1828	-1408	14724	-181	105	114
-1828	-1332	14868	-183	105	110
-1804	-1320	14964	-450	88	31
-1884	-1300	14792	-437	82	38
-1692	-1260	14840	-560	97	9
-1852	-1360	14740	-326	110	45
-1832	-1424	14840	-171	79	83
-1748	-1384	14788	-469	98	25
-1744	-1388	14800	-388	92	48
-1744	-1412	14780	-498	84	29
ax	**ay**	**az**	**gx**	**gy**	**gz**

Fig 74. Muestreo de datos

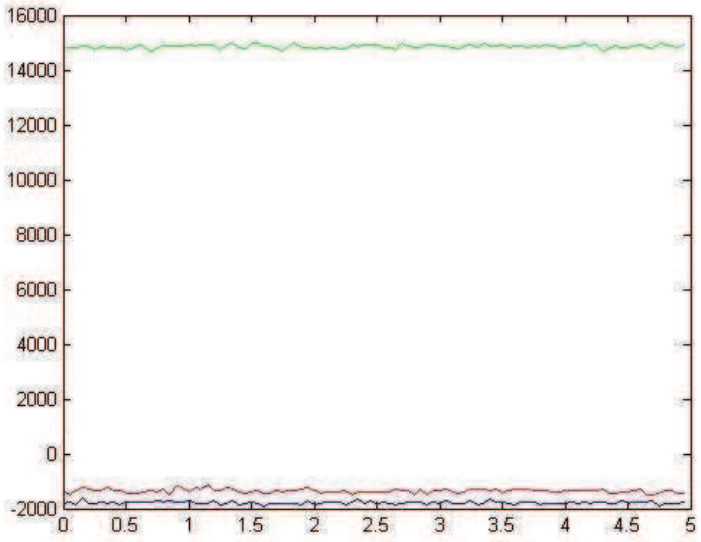


Fig 75. Gráfica aceleraciones

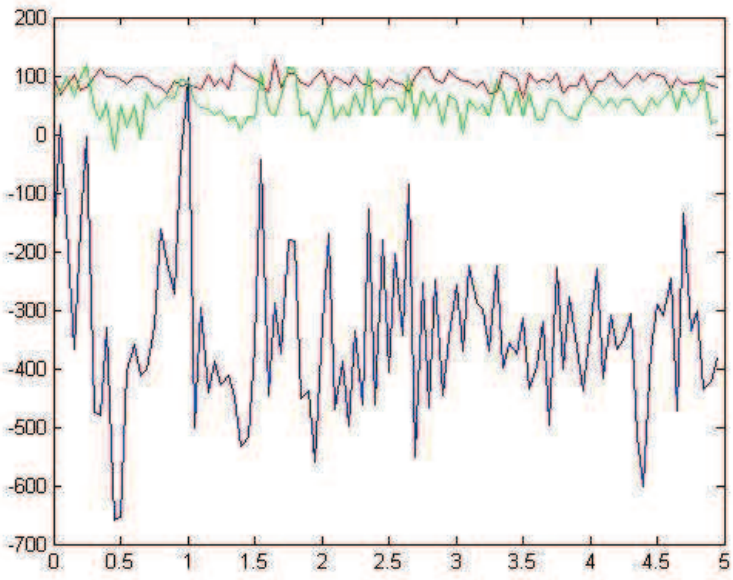


Fig 76. Gráfica giros

Vamos ahora a probar distintos métodos de filtrado en Matlab con los datos obtenidos. Más tarde implementaremos ese filtro en el código de Arduino.

Después de varias pruebas el filtro más simple con mejor resultado ha sido un filtro paso-bajo IIR de orden 1. La función de transferencia del mismo es:

$$H(s) = \frac{1-\alpha}{1-\alpha s^{-1}}$$

Que corresponde a la ecuación en diferencias:

$$y[n] = (1-\alpha)x[n] + \alpha y[n-1]$$

Siendo x[] la sucesión de datos sin filtrar y y[] la sucesión filtrada. Tenemos que variar el parámetro α, que refleja el porcentaje de dato que corresponde al dato filtrado anterior, para obtener la mejor respuesta. Respecto a esto mostramos varias opciones del estudio realizado con una nueva serie de datos esta vez moviendo el brazo para ver la velocidad de respuesta sobre el eje X:

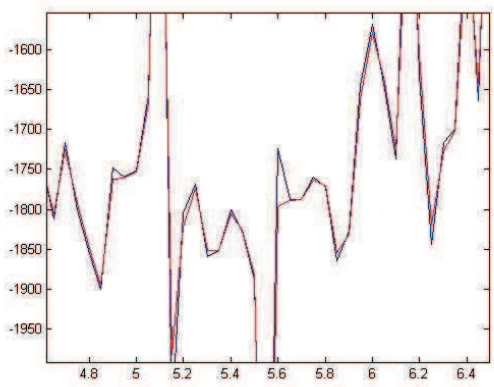


Fig 77. Filtro con α=0.3

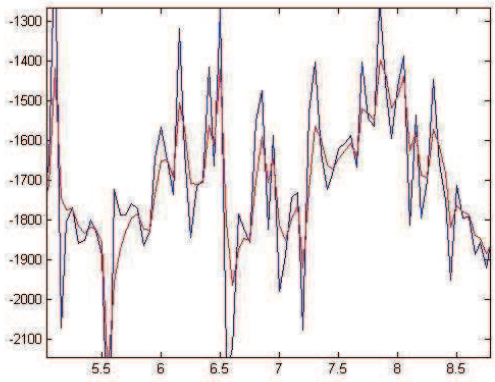


Fig 78. Filtro con α=0.5

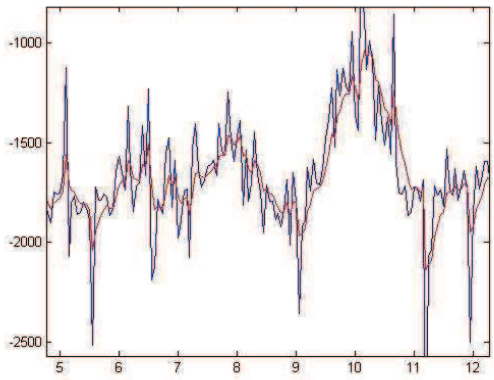


Fig 79. Filtro con α=0.7

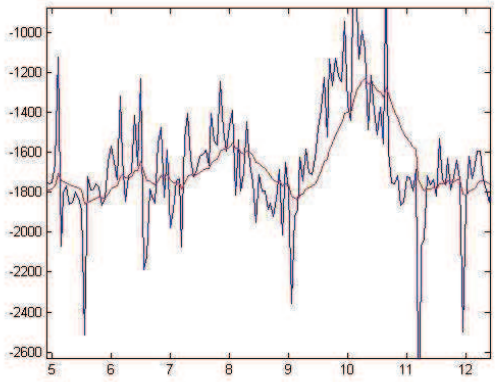


Fig 80. Filtro con α=0.9

Como podemos ver en las gráficas anteriores cuanto mayor es el parámetro α menor es la variación en los datos, pero mayor es el retardo de la señal, teniendo en algunos momentos de la gráfica de $\alpha = 0{,}9$ retardos de hasta 300 ms, así que finalmente vamos a optar por definir $\alpha = 0{,}7$.

Como en el balancín sólo vamos a trabajar con el eje longitudinal al mismo, es decir el eje Y, vamos a definir el código ya para el balancín con el filtro para ese eje. También vamos a operar el dato de salida sabiendo que devuelve un valor de 0 cuando no detecta aceleración, es decir, cuando el brazo forma un ángulo de 0º con la horizontal, y de unos 14800 cuando el brazo está completamente vertical, para mostrar por el monitor serial directamente el ángulo que forma el brazo con la horizontal. Simplemente dividiendo el dato por 14800 y multiplicándolo por 90 tenemos el ángulo en grados.

Declaramos dos variables nuevas, la ayy y la ayy2, que son el valor filtrado en el momento y el valor filtrado en el momento anterior respectivamente. Así el código queda:

```
#include <MPU6050.h>
#include <Wire.h>
#include <I2Cdev.h>
MPU6050 mpu;
int16_t  ay, ax, az, gx, gy, gz;
float ayy;
float alfa=0.8;
float ayy2=0;
int estado=1, angulo;
void setup()
{
    Wire.begin();
    Serial.begin(9600);
    pinMode(5, OUTPUT);
    pinMode(6, OUTPUT);
    mpu.initialize();
mpu.setSleepEnabled(false);
}

void loop()
{

    if(Serial.available()>0){
    estado= Serial.read();
    Serial.flush();
    }
    if (estado==1){
        ayy2=ayy;
    mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
        ayy=(1-alfa)*ay+alfa*ayy2;
        angulo=ayy/164.44;
                Serial.print(ayy);
                Serial.print("  ");
                Serial.print(angulo);
                Serial.println("  ");
    }

}
```

Fig 81. Código con filtro

Ahora que ya tenemos la entrada de datos filtrados debemos implementar el código que controle los motores en función del ángulo. Por la disposición del acelerómetro un ángulo positivo deberá contrarrestarse con mayor potencia en el motor del pin 5, y un ángulo negativo se compensará con mayor potencia en el motor 6.

Para el control vamos a declarar 3 variables más: p, m y n.

p: constante ponderadora del efecto del balanceo. Tipo float.

m: salida del motor del pin 5. Tipo entera de 8 bits.

n: salida del motor del pin 6. Tipo entera de 8 bits.

Ya sabemos cómo se declaran las variables, así que no lo vamos a volver a explicar.

El siguiente paso es implementar el código que debe controlar los motores. Como tenemos 2 actuadores para variar sólo una variable (el ángulo) debemos añadir una condición más al sistema. Esa condición será que la media de los valores de los motores debe ser **175**, un punto medio entre el máximo **255** y **100** que es el valor para el cual el motor se mueve con poca potencia. Para ello escribiremos las siguientes líneas de código:

```
m= (angulo*80/30)+175;          Le doy a m el valor de potencia del motor 5.

m=constrain(m,100,255);         Limito m para que la potencia máxima y mínima
                                esté entre -30º y 30º.

n=255-m;                        Le doy a n el valor que le corresponde para
                                cumplir la condición de que la media de los
                                motores sea 200.
```

Con esto a tenemos el control básico terminado. El acelerómetro leerá el valor de la proyección de la aceleración de la gravedad en el eje del brazo y el microcontrolador lo transformará a grados. Más tarde operará esos grados y elaborará una respuesta en función de ellos.

Hemos probado el sistema y pese a que consigue estabilizarse en horizontal a groso modo, tarda mucho en hacerlo y se balancea mucho. Para mejorar esto tenemos que implementar un controlador PID.

Un controlador PID es un pedazo de código que se basa en el error de la posición actual, en el pasado del error y en el futuro del error para elaborar una respuesta más adecuada. La explicación es la siguiente:

El error actual es obvio, la diferencia entre la posición actual y el punto de consigna.

El pasado del error no es más que la integral de los errores que ha habido.

El futuro del error tiene en cuenta la pendiente con que evoluciona el error para anticiparse.

Estos tres componentes se suman en proporciones distintas según el ajuste para elaborar la respuesta más adecuada. Las constantes de proporcionalidad son la k_p, la k_i y la k_d para la parte proporcional, integral y derivativa respectivamente.

La parte proporcional es sencilla de obtener porque es una simple resta, pero las otras dos vamos a tener que hacer un ajuste.

Para obtener la parte integral vamos a efectuar un sumatorio, que es prácticamente equivalente en un sistema como este que no es continuo, sino discreto con un periodo de muestreo de 50 ms. Así que definiremos una variable nueva llamada h que almacenará el valor de la suma de los errores. Con esta parte podemos conseguir un error de posición nulo a costa de la velocidad de reacción.

En cuanto a la parte derivativa la conseguiremos obteniendo la pendiente de la recta de la progresión del error en el momento actual y el muestreo inmediatamente anterior. La variable que albergará esa pendiente la llamaremos f. La parte derivativa es un poco peligrosa pues es muy sensible al ruido, que abunda en nuestro aparato. Por eso lo más probable es que prescindamos de este componente y solamente elaboremos un PI.

Para implementar el PID lo primero va a ser declarar las variables. Para la variable h que va almacenar la suma del error vamos a utilizar un tipo **long**, más grande que un int, pues la suma del error puede hacerse muy grande. Para la f que almacena la pendiente nos servirá con un **int.** También declararemos las variables mpi y npi que serán las de salida del PID y las constantes del pid.

```
long h=0;

int f=0, mpi, npi;

float kp, ki, kd;
```

No necesitamos declarar una variable que almacene el valor anterior para la parte derivativa, pues ya lo tenemos para el filtro paso bajo, la **ayy2**. Así el PID en código queda:

```
h=h+ayy;                        Calculo la suma de los errores.

f=(ayy-ayy2)/0.05;              Calculo la pendiente del error.

mpi= kp*ayy+ki*h+kd*f;                Aplico el PID.

mpi=constrain(mpi,0,255);       Limito la salida del motor.

npi=255-mpi;                    El otro motor hará lo opuesto.
```

Cuando se tenga el quadcopter montado se deberá utilizar el balancín de ensayos para obtener el PID deseado, en función de querer más estabilidad o mayor maniobrabilidad. En el anexo se explica el procedimiento para el ajuste mediante el método heurístico.

Ya tenemos implementado el PID, y con esto concluye la programación del control del balancín. A continuación modificaremos el código del mismo para adecuarlo al quadcopter, que debe controlar el movimiento en 3 ejes.

La diferencia del código del quadcopter con respecto al del balancín es que hay que contemplar las variaciones con respecto a los 3 ejes, por lo que tenemos que añadir varias cosas:

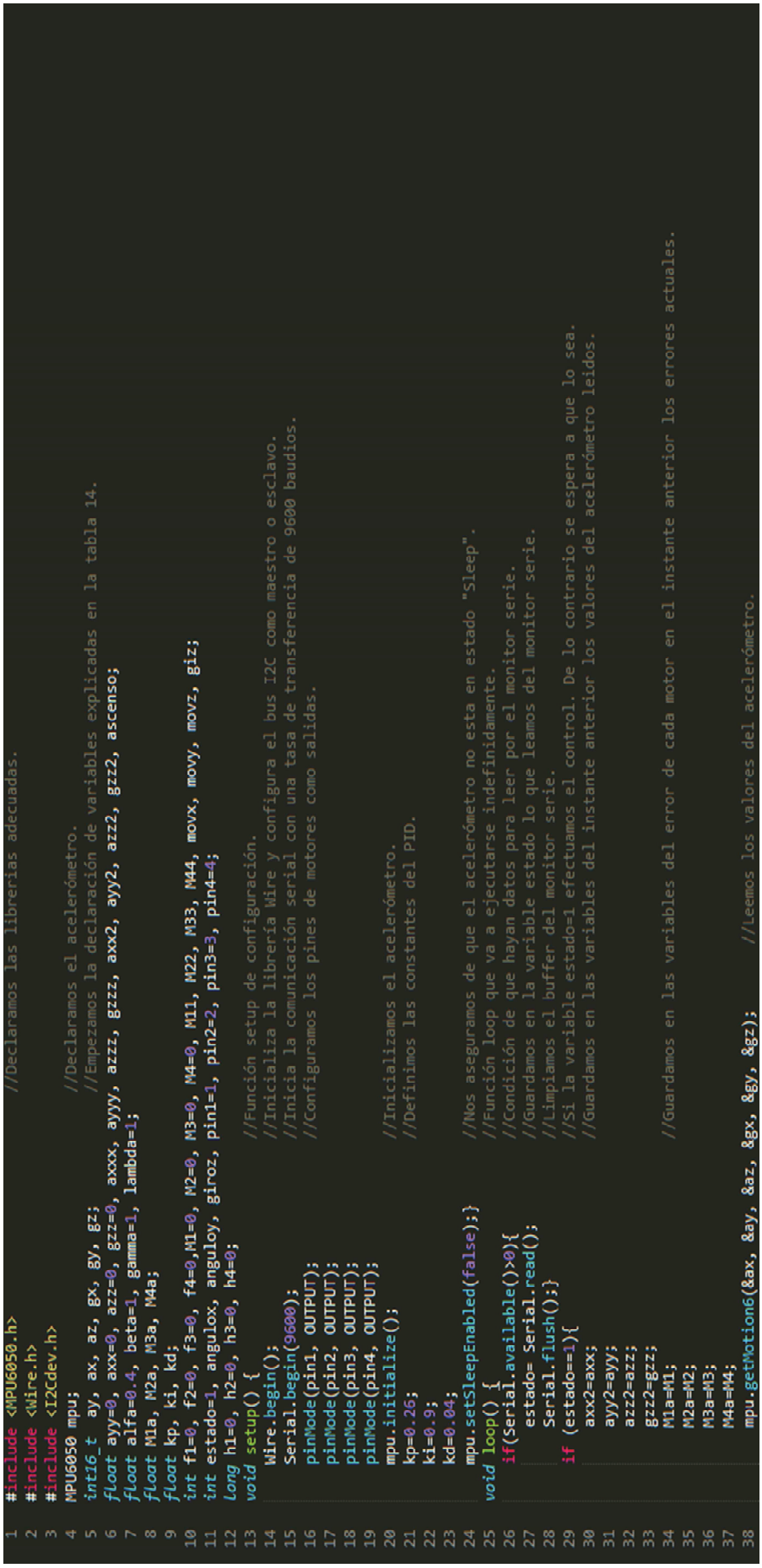
En primer lugar tenemos que filtrar las otras 3 señales que vamos a tratar, que son la aceleración en el eje X para el cabeceo, la aceleración en el eje Z para ascenso/descenso vertical y el giro en el eje Z para la guiñada. Las variables serán ax, az y gz, y las filtradas serán axx, azz y gzz respectivamente. También declararemos angulox, anguloy, ascenso y giroz para el cambio de escala.

Ahora la variación de potencia a cada motor no se da sólo por un suceso que es el balanceo en un eje, sino que interviene balanceo, cabeceo, guiñada y ascenso/descenso, por lo que tenemos que contemplar todos los movimientos posibles y ponderarlos. Esto lo haremos definiendo si aumentando la potencia de un motor se consigue uno u otro sentido en cada movimiento. Así por ejemplo aumentando cualquier motor se consigue ascenso, por lo que este parámetro tendrá un signo '-' delante. En cambio un aumento en los motores 2 o 4 producirá un giro positivo respecto al eje Z, así que llevarán un '-', pero los 1 o 3 producirán el sentido contrario y por eso llevarán un '+'. Para ponderarlos vamos a declarar las variables beta, gamma y lambda, que ponderarán el efecto del cabeceo/balanceo, guiñada y ascenso/descenso respectivamente. Además en el quadcopter no buscaremos estar siempre horizontales, sino que desde el dispositivo controlador se enviarán datos con los movimientos deseados, y habrá que considerarlos todos.

Esto lo haremos mediante la diferencia entre el punto de consigna que viene del móvil y la situación actual para cada movimiento posible (en el código se entenderá más fácilmente). El resultado de esta resta será axxx, ayyy, azzz y gzzz. También necesitaremos los coeficientes del PID h y f para cada movimiento (Teniendo también coeficientes k_p, k_i, y k_d distintos para cada movimiento se mejoraría el control, pero ya que no podemos ajustarlos por no tenerlo fabricado no tiene sentido una mejora tan perfecta). M1, M2, M3 y M4 son las sumas ponderadas de los errores que debe corregir cada uno de los motores, y M11, M22, M33 y M44 son las señales a enviar a los motores post-PID. Necesitaremos además M1p, M2a, M3a y M4a para almacenar la suma de errores en el instante anterior.

Declararemos también pin1, pin2, pin3 y pin4 que serán a los que se conectará cada motor. Ahora los dejamos como 1, 2, 3 y 4, pero deberán cambiarse si los ESC de los motores Brushless se conectan a otros pines. Como la explicación del código va a ser difícil porque se complica con 4 ejes, a continuación aparece una captura del código comentado en cada línea y una tabla con la explicación de cada variable utilizada.

Con esto terminamos este apartado, en el que hemos programado la función que controla los motores a partir de los datos que vienen del acelerómetro en el balancín, estimado un juego de constantes del PID del balancín que se pueden usar en el quadcopter y por último hemos definido completamente el código necesario para un quadcopter con cualquiera de los 4 movimientos posibles.



```
#include <MPU6050.h>                    //Declaramos las librerias adecuadas.
#include <Wire.h>
#include <I2Cdev.h>
MPU6050 mpu;                            //Declaramos el acelerómetro.
int16_t  ay, ax, az, gx, gy, gz;        //Empezamos la declaración de variables explicadas en la tabla 14.
float ayy=0, axx=0, azz=0, gzz=0, axxx, ayyy, azzz, gzzz, axx2, ayy2, azz2, gzz2, ascenso;
float alfa=0.4, beta=1, gamma=1, lambda=1;
float M1a, M2a, M3a, M4a;
float kp, ki, kd;
int f1=0, f2=0, f3=0, f4=0,M1=0, M2=0, M3=0, M4=0, M11, M22, M33, M44, movx, movy, movz, giz;
int estado=1, angulox, anguloy, giroz, pin1=1, pin2=2, pin3=3, pin4=4;
long h1=0, h2=0, h3=0, h4=0;
void setup() {                          //Función setup de configuración.
    Wire.begin();                       //Inicializa la librería Wire y configura el bus I2C como maestro o esclavo.
    Serial.begin(9600);                 //Inicia la comunicación serial con una tasa de transferencia de 9600 baudios.
    pinMode(pin1, OUTPUT);              //Configuramos los pines de motores como salidas.
    pinMode(pin2, OUTPUT);
    pinMode(pin3, OUTPUT);
    pinMode(pin4, OUTPUT);
    mpu.initialize();                   //Inicializamos el acelerómetro.
    kp=0.26;                            //Definimos las constantes del PID.
    ki=0.9;
    kd=0.04;
    mpu.setSleepEnabled(false);}        //Nos aseguramos de que el acelerómetro no esta en estado "Sleep".
void loop() {                           //Función loop que va a ejecutarse indefinidamente.
  if(Serial.available()>0){             //Condición de que hayan datos para leer por el monitor serie.
      estado= Serial.read();            //Guardamos en la variable estado lo que leamos del monitor serie.
      Serial.flush();}                  //Limpiamos el buffer del monitor serie.
  if (estado==1){                       //Si la variable estado=1 efectuamos el control. De lo contrario se espera a que lo sea.
    axx2=axx;                           //Guardamos en las variables del instante anterior los valores del acelerómetro leidos.
    ayy2=ayy;
    azz2=azz;
    gzz2=gzz;
    M1a=M1;                             //Guardamos en las variables del error de cada motor en el instante anterior los errores actuales.
    M2a=M2;
    M3a=M3;
    M4a=M4;
    mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);     //Leemos los valores del acelerómetro.
```

Fig 82. Código de control 1º Parte

```
39  axx=(1-alfa)*ax+alfa*axx2;                    //Filtramos los datos del acelerómetro mediante un filtro paso-bajo.
40  ayy=(1-alfa)*ay+alfa*ayy2;
41  azz=(1-alfa)*az+alfa*azz2;
42  gzz=(1-alfa)*gz+alfa*gzz2;
43  angulox=axx/164.44;                           //Convertimos los datos para obtener grados en los ejes X e Y.
44  anguloy=ayy/164.44;
45  ascenso=azz/14800;                            //Convertimos el dato del eje Z para tenerlo en unidades de gravedad(1G, 1,2G..).
46  giroz=gzz-70;                                 //Ponemos a 0 el valor del giro en Z, que por gráfica hemos visto que tiene un offset en sobre 70.
47  axxx=angulox-movx;                            //Calculamos el error entre el estado actual y la orden del dispositivo de control.
48  ayyy=anguloy-movy;
49  azzz=ascenso-movz;
50  gzzz=giroz-giz;
51  M1=beta*axxx-beta*ayyy-lambda*azzz+gamma*gzzz;          //Calculamos y ponderamos el error que le corresponde a cada motor en función de sus movimientos.
52  M2=beta*axxx+beta*ayyy-lambda*azzz-gamma*gzzz;
53  M3=-beta*axxx+beta*ayyy-lambda*azzz+gamma*gzzz;
54  M4=-beta*axxx-beta*ayyy-lambda*azzz-gamma*gzzz;
55  h1=h1+M1;                                     //Añadimos el error actual a la variable del sumatorio para la parte integral del PID.
56  h2=h2+M2;
57  h3=h3+M3;
58  h4=h4+M4;
59  f1=(M1-M1a)/0.05;                   //Calculamos la pendiente de la recta tangente a la progresión del error.
60  f1=(M2-M2a)/0.05;
61  f1=(M3-M3a)/0.05;
62  f1=(M4-M4a)/0.05;
63  M11= kp*M1+ki*h1+kd*f1;             //Aplicamos el PID para obtener la señal que se debe enviar a cada motor.
64  M22= kp*M2+ki*h2+kd*f2;
65  M33= kp*M3+ki*h3+kd*f3;
66  M44= kp*M4+ki*h4+kd*f4;
67  M11=constrain(M11,0,255);           //Limitamos la señal que se puede enviar a cada motor.
68  M22=constrain(M22,0,255);
69  M33=constrain(M33,0,255);
70  M44=constrain(M44,0,255);
71  analogWrite(pin1,M11);              //Escribimos las señales en cada motor.
72  analogWrite(pin2,M22);
73  analogWrite(pin3,M33);
74  analogWrite(pin4,M44);
75  delay(50);}                         //Hacemos una pausa de muestreo de 50 ms.
76  }
```

Fig 83. Código de control 2º Parte

3.3) Comunicación

En este apartado configuraremos la conexión con el módulo bluetooth HC-05 para su comunicación con un dispositivo Android.

Puesto que no vamos a elaborar el programa en Android hemos hecho la mayor parte de operaciones en la programación de la placa Arduino, por lo que el móvil sólo va a tener que enviar 4 datos, que serán el cabeceo, el balanceo, la guiñada y el ascenso/descenso.

En un principio teníamos la intención de enviar valores variables para tener mayor maniobrabilidad, pero parece ser que el módulo bluetooth no permite esto, así que el control desde el móvil va a realizarse con variables booleanas. Es decir, el móvil tendrá 8 botones, correspondientes a los 2 sentidos posibles de cada uno de los 4 movimientos. Así, enviaremos datos de 8 bits, en los que cada bit referenciara un sentido de un movimiento.

Por orden hemos definido los movimientos:

Ejex+///Ejex-///Ejey+///Ejey-///Giroz+///Giroz-///Ascenso///Descenso

Debido a esto vamos a tener que definir desde un principio las variables que declaramos en el programa de control *movx, movy, movz y giz.*

Hemos decidido que el cabeceo y balanceo van a ser de 30º, el ascenso y descenso será de ±0.5G, y el giro en Z será de 90º/s. Así que definimos las variables en el código:

```
lecbt=0;

movx=±30;

movy=±30;

movz=1±0.5;
```

Como el giz no está ajustado a º/s porque no era un movimiento crítico, lo ajustaremos ahora. En la datasheet del acelerómetro nos dice que devuelve 131 por cada º/s que giramos, así que si queremos girar a 90º/s el valor que tenemos que poner en giz es 131*90:

```
giz=±11790;
```

El problema es cómo enviar un solo valor que actúe sobre 8 sentidos distintos.

La solución es operando por bit. Cuando el Arduino reciba el número de 8 bits(que irá de 0 a 255) lo procesará comparando el bit correspondiente al movimiento con una máscara, y con un if efectuará o no la acción. Veamos un ejemplo.

Suponiendo que se quiere ascender a la vez que inclinarse en dirección al eje x. La aplicación móvil generará un 10000010, que es un 130 en decimal. Cuando al Arduino le llegue ese número hará 8 comparaciones (una por cada sentido y movimiento), y si el bit de esa acción está activo la realizará. En nuestro caso:

```
lecbt = bt.read();
bt.flush();

if(lecbt & 128){
        movx=30;}
else{
        movx=0;}

if(lecbt & 64){
        movx=-30;}
else{
        movx=0;}
if(lecbt & 32){
        movy=30;}
else{
        movy=0;}
if(lecbt & 16){
        movx=-30;}
else{
        movx=0;}
if(lecbt & 8){
        giz=11790;}
else{
        giz=0;}
if(lecbt & 4){
        giz=-11790;}
else{
        giz=0;}
if(lecbt & 2){
        movz=1.5;}
else{
        movz=0;}
if(lecbt & 1){
        movz=0.5;}
else{
        movz=0;}}
```

Como las únicas condiciones que se habrán cumplido son la de 128(10000000) y 2(00000010), movx=30 y movz=1.5. El resto serán 0.

3.4) Recopilación

Ahora que ya tenemos el código de control y el de comunicación sólo nos queda juntarlos los dos declarando todas las variables y dejar perfectamente definido el código de la placa Arduino de nuestro quadcopter.

Este código deberá ser introducido en el programa de Arduino cuya URL para descargarlo aparece en el Anexo, para posteriormente enviarlo por USB a la Placa Arduino.

A continuación se muestra el diagrama de flujo correspondiente a nuestro programa.

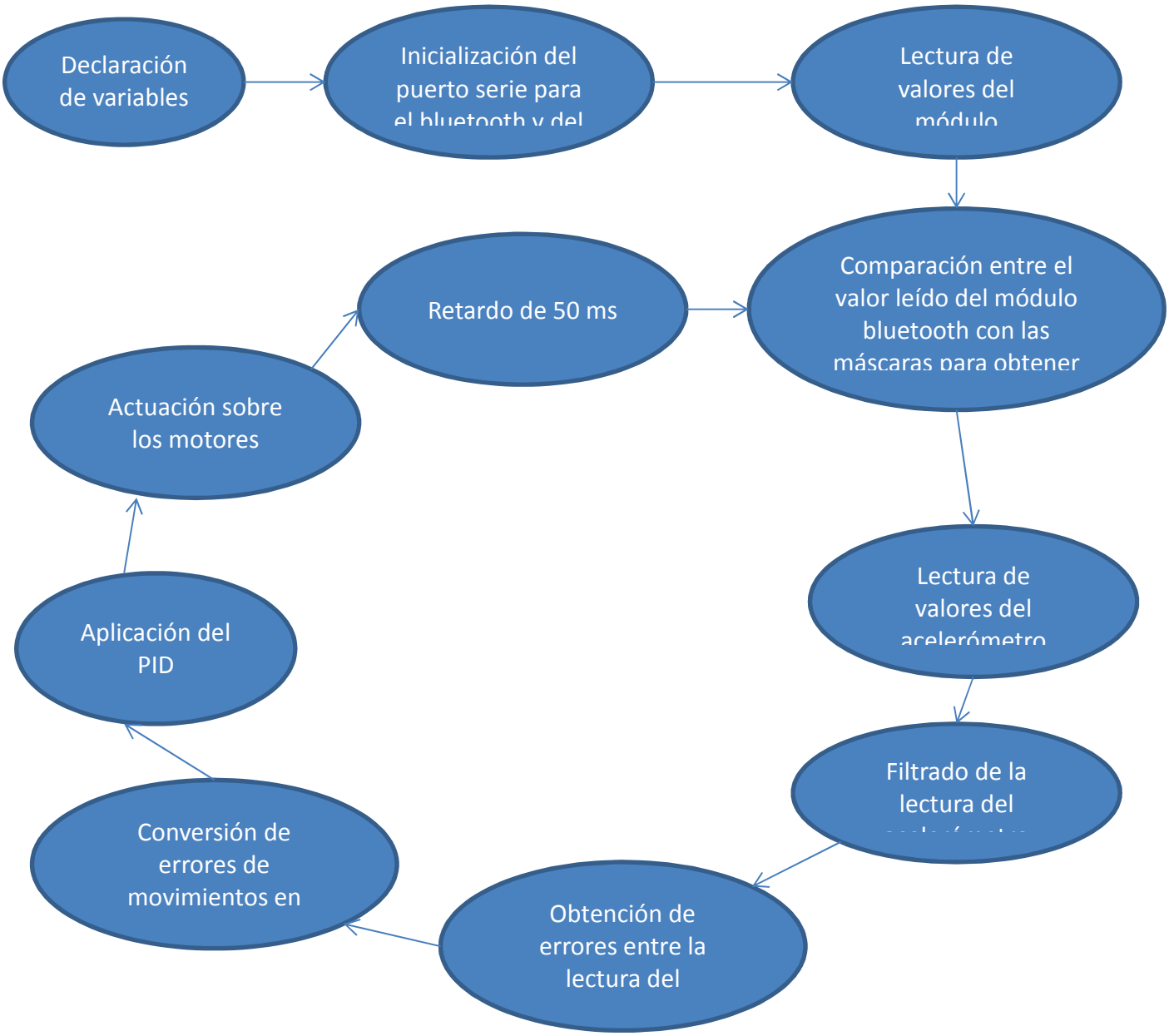


Fig 84. Diagrama de flujo del código

4) Resumen y conclusión.

Durante el presente proyecto se ha conseguido lo expuesto en los objetivos, que ha sido diseñar un Quadcopter basado en un microcontrolador Arduino. Se ha descrito los componentes posibles, escogido los más adecuados planteando los criterios de selección y más tarde diseñando la estructura en 3D. A continuación se ha diseñado, fabricado y programado también un sistema de pruebas para experimentar diversos valores de constantes del controlador PID, evitando con esto tener que probarlo directamente en un hipotético quadcopter que, de estar mal ajustados podrían ocasionar la rotura del mismo. Para finalizar también se ha implementado el código completo que sería capaz de controlar el aparato, a falta de algún retoque y ajuste del PID.

S ha intentado desarrollar el primer objetivo de forma que se abarcasen la mayor cantidad posible de áreas que han aparecido durante la carrera. Por eso obtener la estructura con un programa de diseño asistido por ordenador en vez de escoger una comercial, o programar el código en Arduino en vez de comprar un microcontrolador preprogramado. También se ha entrado en el área de electrónica para la circuitería, de automática para el PID y utilizado el software MATLAB visto en varias asignaturas para obtener las gráficas de lectura del acelerómetro y realizar el filtrado óptimo.

Gracias al amplio detalle en cuanto al desarrollo del quadcopter sería posible y sencillo para un tercero sin amplios conocimientos de ingeniería el adquirir los componentes correctos, fabricar los otros, obtener el PID y montar el aparato sólo con este proyecto. Además como mejora del mismo se ha considerado ya la posibilidad de añadir nuevos componentes como control por GPS utilizando la base de datos de google maps incluyendo los desarrollos 3D de edificios, y sensores de ultrasonidos, con lo que se podría conseguir el vuelo autónomo evitando obstáculos y llegaría a cualquier lugar sólo con unas coordenadas.

En general ha sido gratificante realizar el presente proyecto por poner en práctica conocimientos de la carrera que hasta que no se ponen en práctica no se comprenden a la perfección. Ha servido además para entender conceptos que hasta ahora simplemente se habían memorizado sin entender el porqué. Considero una muy buena práctica la realización de TFG o PFC para poner en práctica lo aprendido y tener una idea de cómo va a ser nuestro futuro trabajo como Ingenieros.

Bibliografía

A continuación aparecen las webs y los libros a los que se ha accedido para consultas o búsqueda de componentes

WEBS:

http://es.wikipedia.org/wiki/Veh%C3%ADculo_a%C3%A9reo_no_tripulado

http://es.wikipedia.org/wiki/Veh%C3%ADculo_a%C3%A9reo_de_combate_no_tripulado

http://www.iuavs.com/pages/aplicaciones_y_usos

http://es.wikipedia.org/wiki/Ejes_del_avi%C3%B3n

http://en.wikipedia.org/wiki/Brushless_DC_electric_motor

https://www.sparkfun.com/

http://www.arduino.cc/

http://vimeo.com/18390711

http://gizmologia.com/2014/03/hardware-novatos-arduino

http://es.kioskea.net/contents/69-como-funciona-bluetooth

http://es.wikipedia.org/wiki/Bater%C3%ADa_el%C3%A9ctrica

http://www.cobramotorsusa.com

http://www.hobbyking.com

http://es.wikipedia.org/wiki/Proporcional_integral_derivativo

http://www.infouas.com/nuevos-pasos-hacia-la-regulacion-del-uso-de-uav-civiles-en-espana/

http://www.multicopters.es/

http://control-pid.wikispaces.com/

LIBROS:

Apuntes de informática industrial. Ors, Martí, Pérez, Gracia, Saiz. 2011

Feedback and control for everyone. P. Albertos. 2011

Anexo

1) Hojas de especificaciones de componentes:

Motor brushless Cobra C-2204/32:

http://www.cobramotorsusa.com/2204-32.html

Control de velocidad Cobra 11A ESC w/2A Linear BEC:

http://www.cobramotorsusa.com/esc11.html

Batería ZIPPY Flightmax 5000mAh 3S1P 20C hardcase pack:

http://www.hobbyking.com/hobbyking/store/__14083__ZIPPY_Flightmax_5000mAh_3S1P_20C_hardcase_pack.html

Acelerómetro MPU-6050:

http://www.cdiweb.com/datasheets/invensense/PS-MPU-6000A.pdf

Placa Arduino UNO:

http://arduino.cc/en/Main/arduinoBoardUno

Módulo bluetooth HC-05:

http://www.electronicaestudio.com/docs/istd016A.pdf

Transistor BC-338 40:

http://pdf.datasheetcatalog.com/datasheet/motorola/BC338-16.pdf

2) Legislación

La vicepresidenta del gobierno Soraya Sáez de Santamaría dijo el pasado 10 de junio de 2014 en el consejo de ministros que próximamente se iba a legislar el uso de Drones.

Aún no habían detalles concretos, pero sí que se confirmó que dicha legislación iría relacionada con el peso y el tamaño de los Drones, y que no estaría permitido que sobrevolaran zonas pobladas sin autorización.

En el borrador de la ley se indica que los pilotos de Drones deberán tener contacto visual con el aparato si pesa entre 25 y 250 kg, no siendo necesario si pesa menos, pero siempre volando a una altitud máxima de 120 m. También aparece la intención de que los pilotos necesiten un certificado de navegación aérea para Drones de hasta 25 kg o una licencia de piloto profesional si su masa está entre 25 y 150 kg.

Todo esto está todavía un poco en el aire y habrá que esperar a nuevos comunicados del gobierno para asegurarnos de no estar incumpliendo ninguna ley al volar nuestro quadcopter de forma civil.

3) Ajuste PID

Para el ajuste del PID se necesitará tener el software de programación de Arduino que se puede encontrar en la siguiente web:

http://arduino.cc/en/Main/Software#toc2

Una vez instalado introduciremos el código del balancín que aparece a continuación y lo enviaremos a Arduino.

```
#include <MPU6050.h>
#include <Wire.h>
#include <I2Cdev.h>
MPU6050 mpu;
int16_t ay, ax, az, gx, gy, gz;
float ayy;
float alfa=0.8;
float ayy2=0;
int estado=1, angulo;
long h=0;
int f=0, mpi, npi;
float kp, ki, kd;
void setup() {
        Wire.begin();
        Serial.begin(9600);
        pinMode(5, OUTPUT);
        pinMode(6, OUTPUT);
        mpu.initialize();
        mpu.setSleepEnabled(false);
        kp= 0;
        ki= 0;
        kd= 0;
}
void loop(){
        if(Serial.available()>0{
                estado=Serial.read();
                Serial.flush();}
        if (estado==1){
```

```
                ayy2=ayy;
                mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
                ayy=(1-alfa)*ay+alfa*ayy2;
                h=h+ayy;
                f= (ayy-ayy2)/0.05;
                mpi= kp*ayy+ki*h+kd*f;
                mpi=constrain(mpi,0,255);
                npi= 255-mpi;
                analogWrite(5, mpi);
                analogWrite(6, npi);
                delay(50);}
}
```

Una vez introducido el código se van a variar las constantes kp, ki y kd, que están inicialmente a 0 hasta sintonizar correctamente el PID.

1) Aumentar kp hasta obtener oscilaciones estables.

2) Aumentar ki hasta anular el error estacionario, aunque haya sobreoscilación, y disminuir entonces la kp hasta que se consigan oscilaciones estables. En este punto decidiremos qué dinámica queremos para el quadcopter, con una ganancia más baja para movimientos más suaves y más alta para movimientos más bruscos.

3) Aumentar la kd hasta obtener la velocidad de respuesta deseada.

Una vez ajustado el PID del balancín podremos extrapolarlo al quadcopter y ajustar un poco las constantes para adecuarlo al aparato.

4) Variables utilizadas

Variable	Explicación
ax	Lectura de aceleración en el eje X del acelerómetro
ay	Lectura de aceleración en el eje Y del acelerómetro
az	Lectura de aceleración en el eje Z del acelerómetro
gz	Lectura del giro en el eje Z del acelerómetro
axx	Valor de la aceleración en x después de pasarla por el filtro paso bajo
ayy	Valor de la aceleración en y después de pasarla por el filtro paso bajo
azz	Valor de la aceleración en z después de pasarla por el filtro paso bajo
gzz	Valor del giro en z después de pasarla por el filtro paso bajo
axx2	Valore de la aceleración en X en el instante anterior filtrada
ayy2	Valore de la aceleración enY en el instante anterior filtrada
azz2	Valore de la aceleración en Z en el instante anterior filtrada
gzz2	Valore del giro en Z en el instante anterior filtrado
axxx	Error en el angulo que el eje x forma con el punto de consigna
ayyy	Error en el angulo que el eje y forma con el punto de consigna
azzz	Error en el angulo que el eje z forma con el punto de consigna
gzzz	Error en el giro que se produce con respecto al punto de consigna
h1	Sumatorio de errores del motor 1
h2	Sumatorio de errores del motor 2
h3	Sumatorio de errores del motor 3
h4	Sumatorio de errores del motor 4
f1	Pendiente de recta de errores del motor 1
f2	Pendiente de recta de errores del motor 2
f3	Pendiente de recta de errores del motor 3
f4	Pendiente de recta de errores del motor 4
kp	Constante proporcional del PID
ki	Constante integral del PID
kd	Constante derivativa del PID
estado	Variable para comprobar el estado del puerto serie
angulox	angulo que el eje x forma con la horizontal
anguloy	angulo que el eje y forma con la horizontal
ascenso	fuerza que sufre el aparato verticalmente
giroz	giro que sufre el aparato alrededor del eje z
pin1	pin donde se conectará el ESC del motor 1
pin2	pin donde se conectará el ESC del motor 2
pin3	pin donde se conectará el ESC del motor 3
pin4	pin donde se conectará el ESC del motor 4
M1	Suma ponderada de errores atribuidos al motor 1
M2	Suma ponderada de errores atribuidos al motor 2
M3	Suma ponderada de errores atribuidos al motor 3
M4	Suma ponderada de errores atribuidos al motor 4
M11	Señal a enviar al motor 1
M22	Señal a enviar al motor 2
M33	Señal a enviar al motor 3
M44	Señal a enviar al motor 4
movx	Orden del dispositivo de control para girar el eje x
movy	Orden del dispositivo de control para girar el eje y
movz	Orden del dispositivo de control para ascender o descender
giz	Orden del dispositivo de control para guiñar
alfa	Parámetro para el filtro paso bajo
beta	Ponderación del cabeceo/alabeo
gamma	Ponderación de la guiñada
lambda	Ponderación del ascenso/descenso

5) Código completo

```
#include <MPU6050.h>

#include <Wire.h>

#include <I2Cdev.h>

#include <SoftwareSerial.h>

MPU6050 mpu;

int16_t  ay, ax, az, gx, gy, gz;

float ayy=0, axx=0, azz=0, gzz=0, axxx, ayyy, azzz, gzzz, axx2, ayy2, azz2,
gzz2, ascenso;

float alfa=0.4, beta=1, gamma=1, lambda=1;

float M1a, M2a, M3a, M4a;

float kp, ki, kd;

int f1=0, f2=0, f3=0, f4=0,M1=0, M2=0, M3=0, M4=0, M11, M22, M33, M44, movx,
movy, movz, giz;

int estado=1, angulox, anguloy, giroz, pin1=1, pin2=2, pin3=3, pin4=4, lecbt;

long h1=0, h2=0, h3=0, h4=0;

SoftwareSerial bt(5, 6);

void setup() {

    Wire.begin();

    Serial.begin(9600);

    pinMode(pin1, OUTPUT);

    pinMode(pin2, OUTPUT);

    pinMode(pin3, OUTPUT);

    pinMode(pin4, OUTPUT);

    mpu.initialize();

    bt.begin(9600);

    kp=0.26;

    ki=0.9;

    kd=0.04;

    mpu.setSleepEnabled(false);}

void loop() {

    if(bt.available()>0){
```

```
lecbt = bt.read();
bt.flush();
if(lecbt & 128){
    movx=30;}
else{
    movx=0;}
if(lecbt & 64){
    movx=-30;}
else{
    movx=0;}
if(lecbt & 32){
    movy=30;}
else{
    movy=0;}
if(lecbt & 16){
    movx=-30;}
else{
    movx=0;}
if(lecbt & 8){
    giz=11790;}
else{
    giz=0;}
if(lecbt & 4){
    giz=-11790;}
else{
    giz=0;}
if(lecbt & 2){
    movz=1.5;}
else{
    movz=0;}
if(lecbt & 1){
    movz=0.5;}
```

```
    else{
        movz=0;}}
    if(Serial.available()>0){
        estado= Serial.read();
        Serial.flush();}
    if (estado==1){
        axx2=axx;
        ayy2=ayy;
        azz2=azz;
        gzz2=gzz;
        M1a=M1;
        M2a=M2;
        M3a=M3;
        M4a=M4;
        mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
        axx=(1-alfa)*ax+alfa*axx2;
        ayy=(1-alfa)*ay+alfa*ayy2;
        azz=(1-alfa)*az+alfa*azz2;
        gzz=(1-alfa)*gz+alfa*gzz2;
        angulox=axx
        anguloy=ayy
        ascenso=azz
        giroz=gzz-70;
        axxx=angulox-movx;
        ayyy=anguloy-movy;
        azzz=ascenso-movz;
        gzzz=giroz-giz;
        M1=beta*axxx-beta*ayyy-lambda*azzz+gamma*gzzz;
        M2=beta*axxx+beta*ayyy-lambda*azzz-gamma*gzzz;
        M3=-beta*axxx+beta*ayyy-lambda*azzz+gamma*gzzz;
        M4=-beta*axxx-beta*ayyy-lambda*azzz-gamma*gzzz;
        h1=h1+M1;
```

```
        h2=h2+M2;
        h3=h3+M3;
        h4=h4+M4;
        f1=(M1-M1a)
        f1=(M2-M2a)
        f1=(M3-M3a)
        f1=(M4-M4a)
        M11= kp*M1+ki*h1+kd*f1;
        M22= kp*M2+ki*h2+kd*f2;
        M33= kp*M3+ki*h3+kd*f3;
        M44= kp*M4+ki*h4+kd*f4;
        M11=constrain(M11,0,255);
        M22=constrain(M22,0,255);
        M33=constrain(M33,0,255);
        M44=constrain(M44,0,255);
        analogWrite(pin1,M11);
        analogWrite(pin2,M22);
        analogWrite(pin3,M33);
        analogWrite(pin4,M44);
        delay(50);}
}
```

PRESUPUESTO

Presupuesto

En el presente apartado se valorará y contabilizará el coste monetario en el que se incurriría si se llevase a cabo el proyecto. Para esto vamos a tener en cuenta el coste de la mano de obra (en este caso de un ingeniero en prácticas), el coste de amortización de los equipos y programas utilizados, y el coste de los componentes necesarios para la construcción del aparato. Obtendremos estos presupuestos parciales y luego generaremos el presupuesto de ejecución material y finalmente el presupuesto de desarrollo.

Índice

1) Presupuestos parciales

1.1) Presupuesto parcial de mano de obra

Como coste de mano de obra se contará solamente la del ingeniero que desarrolla el proyecto. El sueldo medio de un ingeniero industrial en prácticas se estima en 5 €/h

Presupuesto parcial de mano de obra

Concepto	Unidad	Precio unitario	Cantidad	Total
Ingeniero Industrial	Horas	5€/h	300 h	1500 €
Mano de obra indirecta (10%)				150 €
TOTAL				**1650 €**

1.2) Presupuesto parcial de amortización de equipos

Presupuesto parcial de amortización de equipos

Concepto	Precio	Periodo de amortización	Periodo utilizado	Coste
Ordenador i7	700 €	5 años	1 mes	11,667 €
MATLAB 2013	2.000 €	5 años	1 semana	7,7 €
Autodesk Inventor 2014	9.000 €	5 años	3 semanas	103,85 €
TOTAL				**123,217 €**

1.3) Presupuesto parcial de componentes del quadcopter

Presupuesto parcial de componentes del quadcopter

Concepto	Precio	Cantidad	Total
Impresión 3D Estructura	70 €	1	70 €
Placa Arduino UNO	20 €	1	20 €
Acelerómetro MPU-6050	40 €	1	40 €
Módulo Bluetooth HC-05	10 €	1	10 €
Motor Brushless Cobra C-2204/32	21 €	4	84 €
ESC Cobra 11 A	16 €	4	64 €
Hélice GWS 6x3x3	1 €	4	6 €
Batería ZIPPY Flightmax 4000mAh 3S1P 40C	24 €	1	24 €
Cableado	2 €	1	2 €
TOTAL			**319,55 €**

2) Presupuesto total

Presupuesto total

Concepto	Precio
Presupuesto parcial de mano de obra	1600,00 €
Presupuesto parcial de amortización de equipos	123,22 €
Presupuesto parcial de componentes del quadcopter	319,55 €
Presupuesto de ejecución material	**2042,77€**
Gastos generales (5%)	102,14 €
Beneficio industrial (6%)	122,57 €
Presupuesto total de ejecución	**2267,48 €**
I.V.A. (21%)	476,17 €
Presupuesto base de licitación	**2743,65 €**

PLANOS

Planos

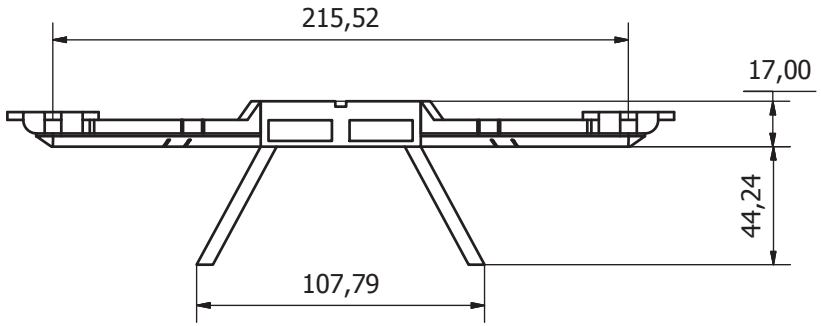
Las páginas siguientes corresponden a los planos obtenidos de los desarrollos 3D de la estructura del quadcopter y del balancín de ensayos.

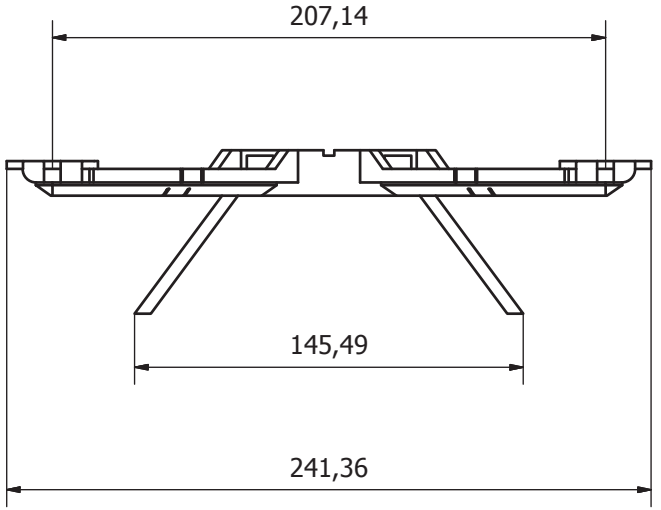
Puesto que el primero está diseñado para generarse por impresión 3D, no es posible obtener planos de despiece, así que solo aparece el plano general de la estructura. No obstante se adjunta la dirección URL en la que se puede descargar un archivo con el modelo 3D de la estructura y de la cruz que sujeta la placa de prototipos en formatos stl y parasolid listos para su impresión. (La contraseña para descomprimir el fichero es *vicravi*)

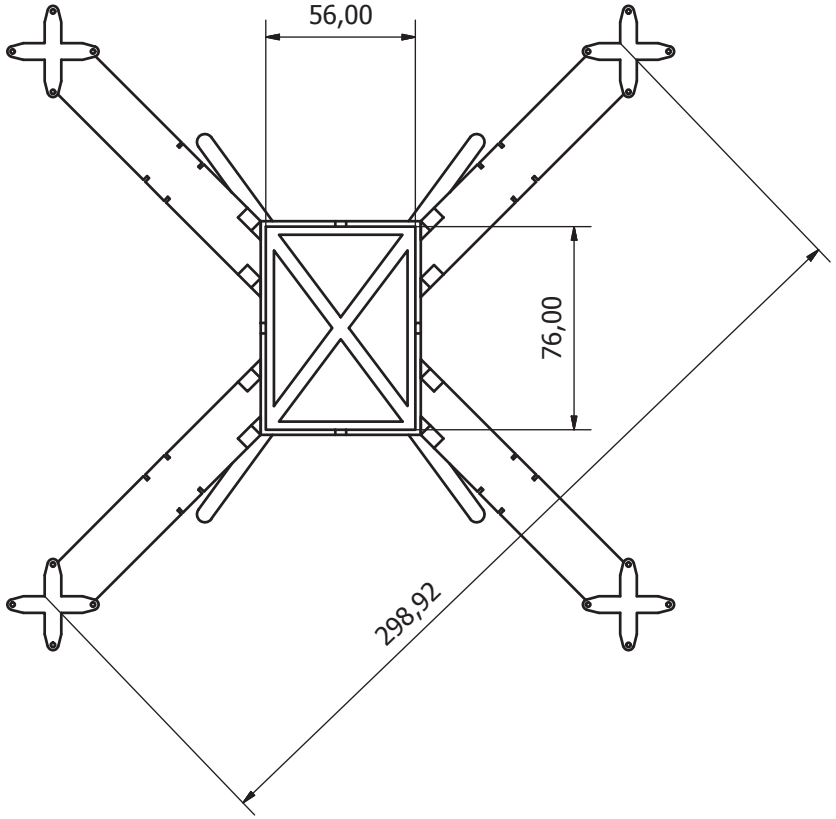
https://drive.google.com/file/d/0B5A22Z9_4f6SdEtfc1dfNVY4cGc/edit?usp=sharing

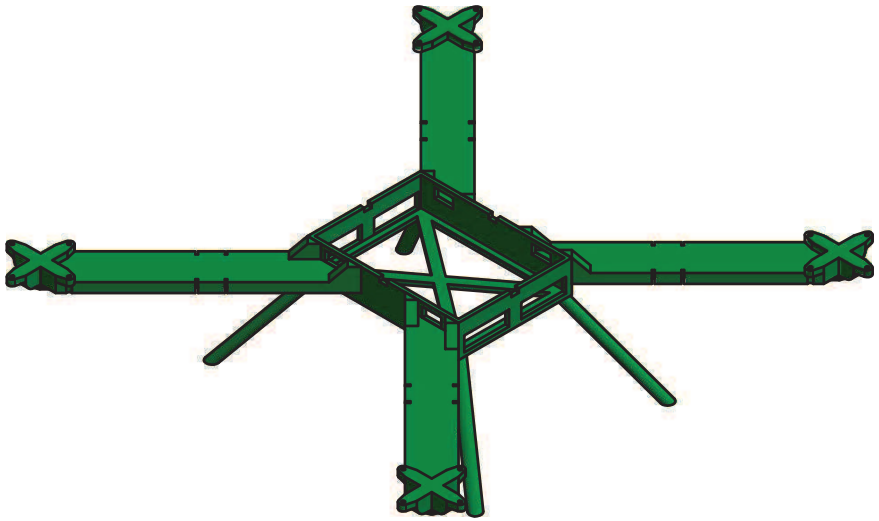
En cuanto al balancín, puesto que se fabrica en madera de balsa, se han obtenido los planos de conjunto en A3 y despiece detallados en A4, terminando con un plano que contiene todas las piezas (duplicadas las necesarias) en tamaño A3 con el que se han fabricado cada una de las partes.

Índice



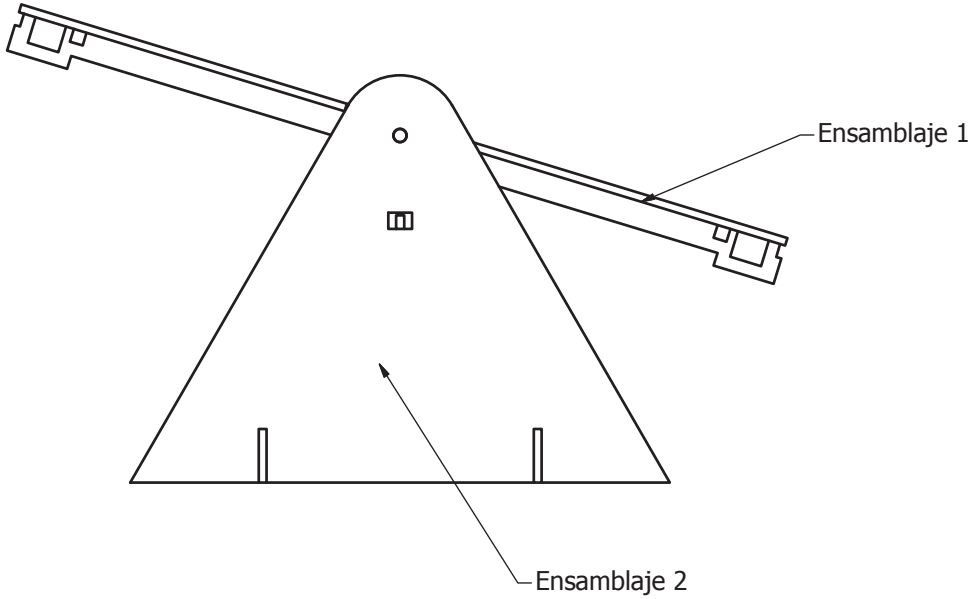


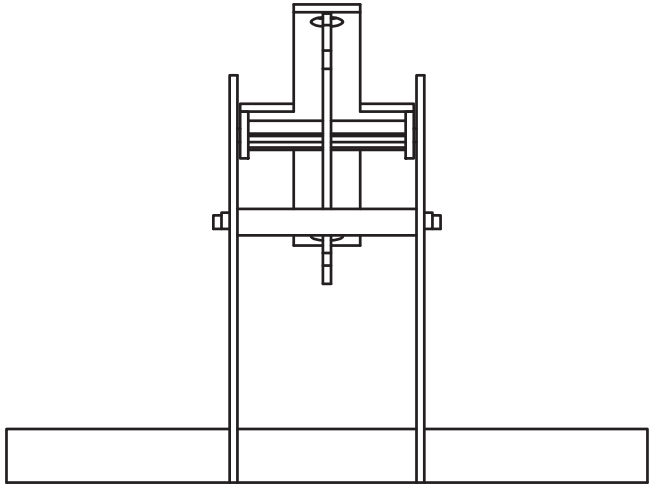


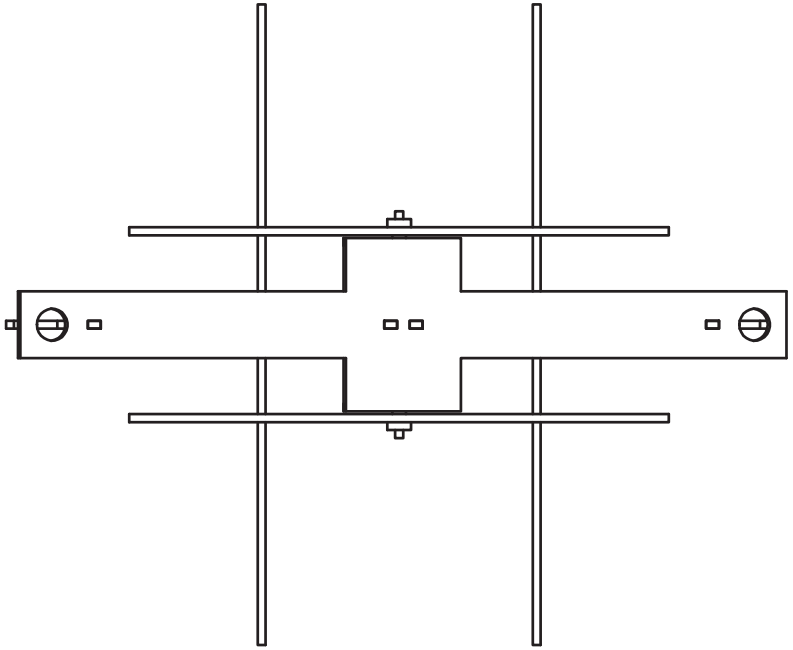


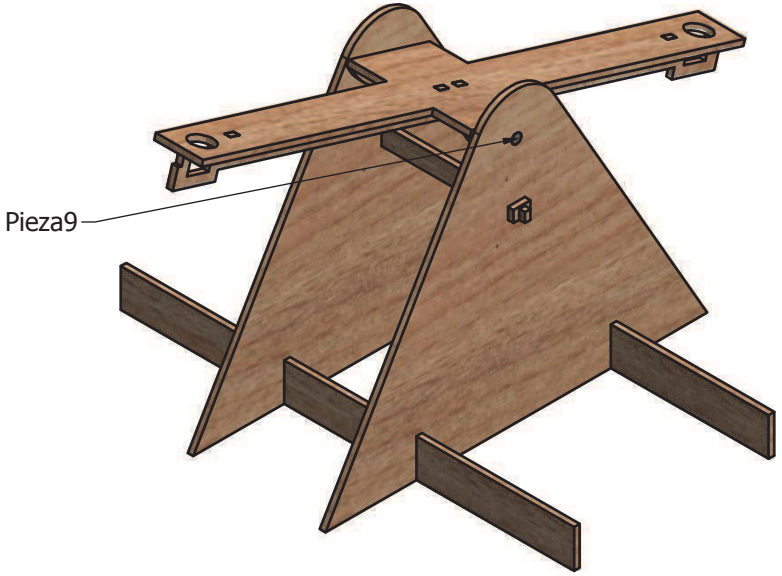
Diseñado por	Comprobado por	Cotas en	Escala	Fecha	
Victor		mm	1:2	19/06/2014	

PLANOS	Estructura del Quadcopter	
	Plano estructura	Hoja 1 / 1



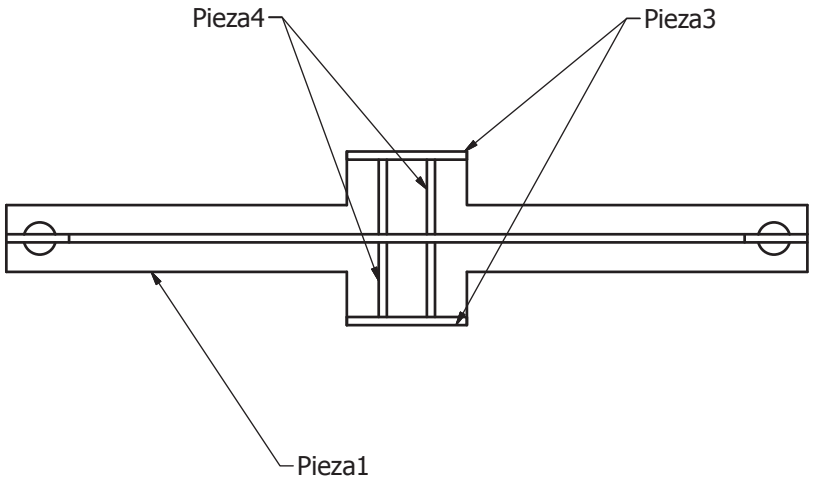


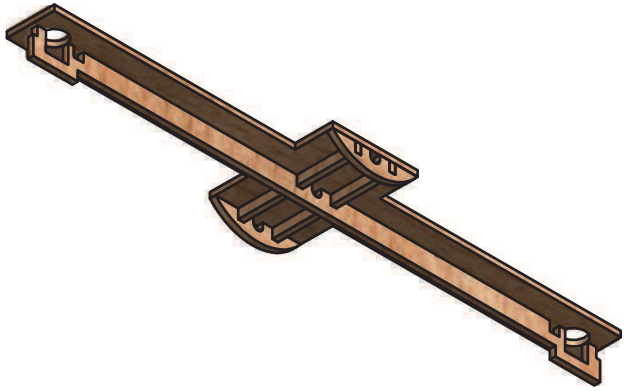


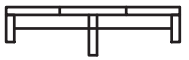


Diseñado por	Comprobado por	Cotas en	Escala	Fecha	
Victor		mm	1:2	19/06/2014	

PLANOS	Balancín de ensayo de PID	
	Modelo global	Hoja 1 / 12

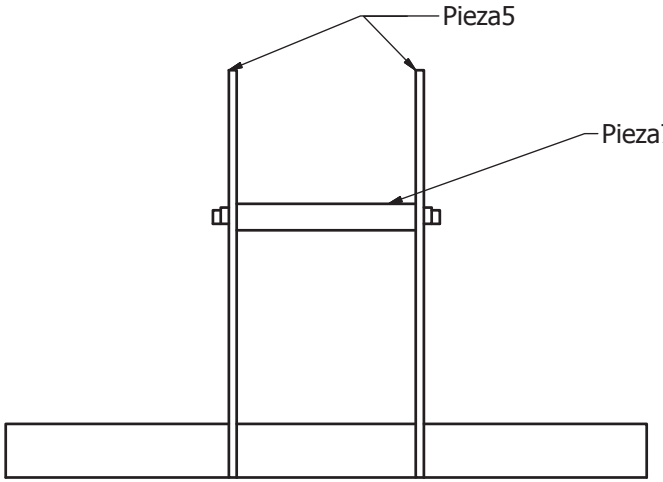


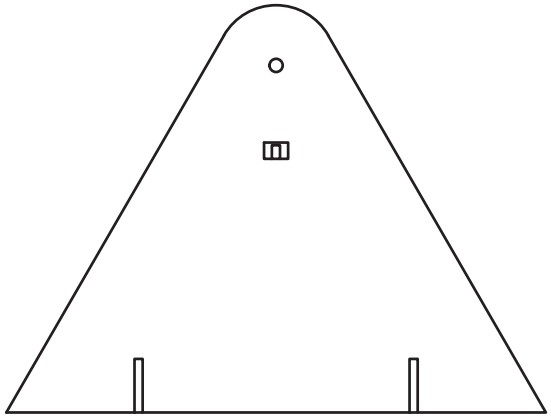


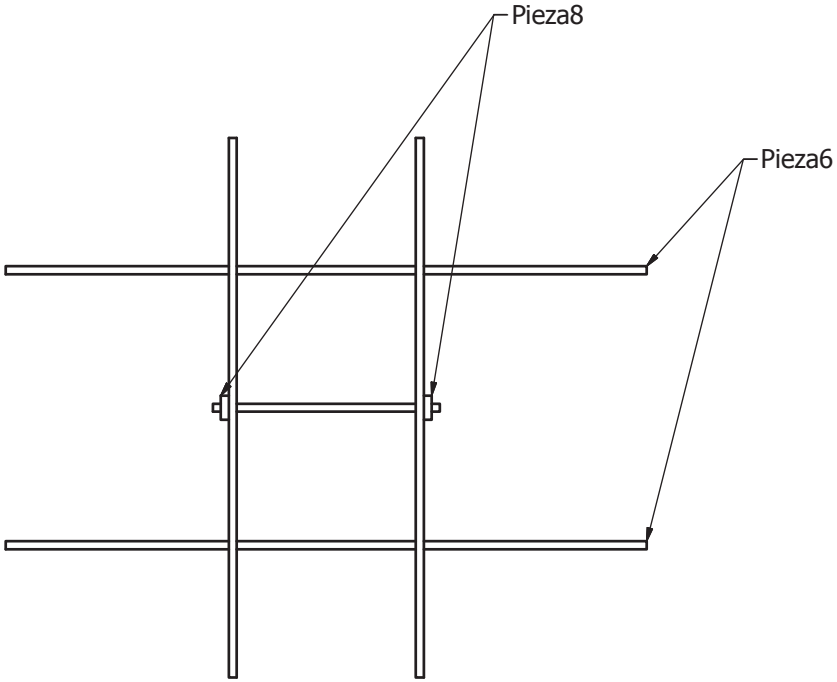


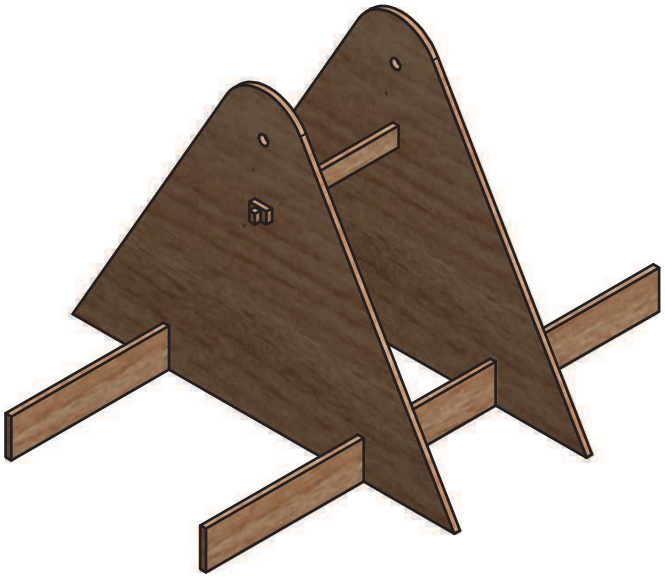
Diseñado por	Comprobado por	Cotas en	Escala	Fecha	
Victor		mm	1:2	19/06/2014	

PLANOS	Balancín de ensayo de PID	
	Ensamblaje 1	Hoja 2 / 12



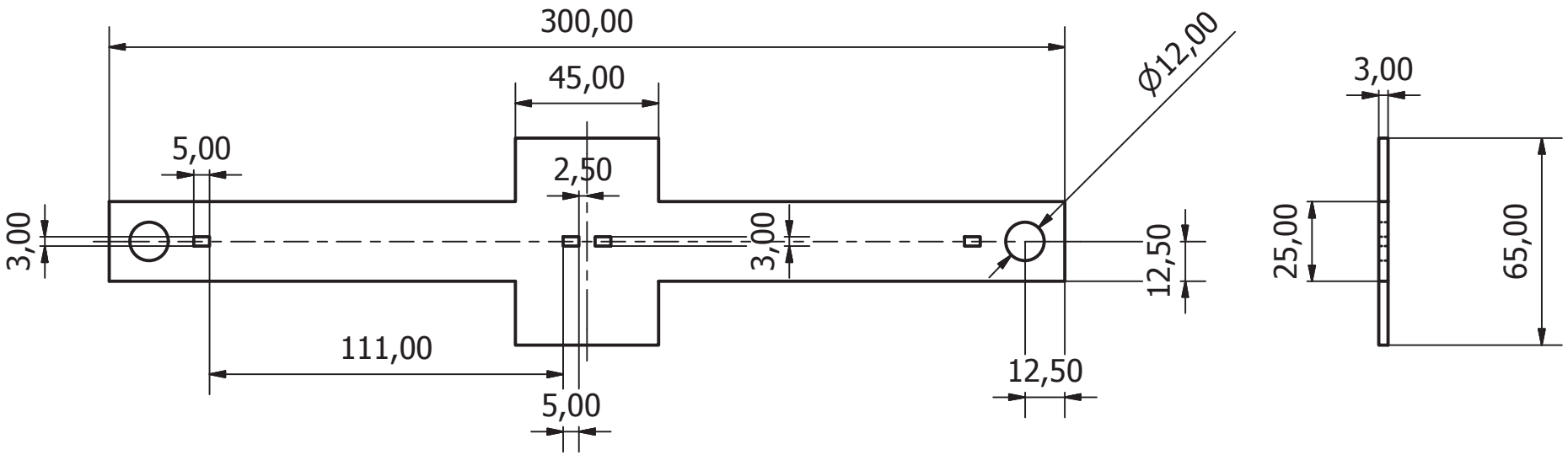


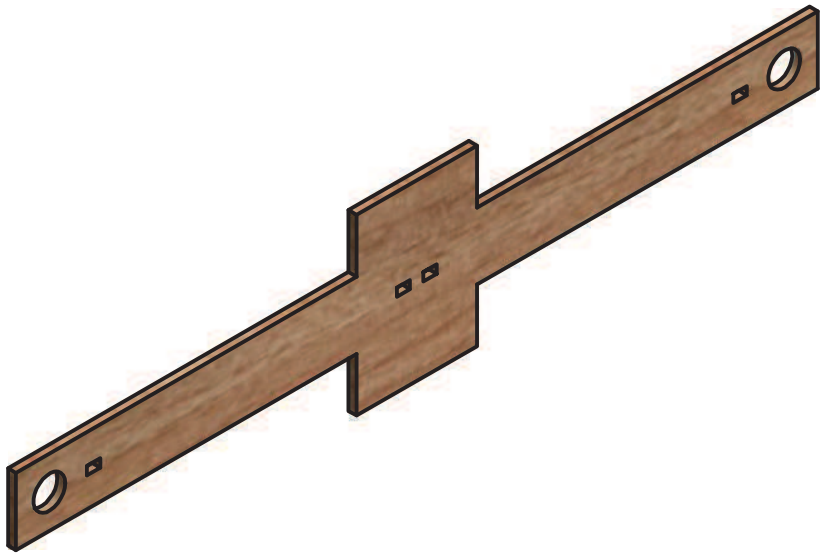




Diseñado por	Comprobado por	Cotas en	Escala	Fecha	
Victor		mm	1:2	19/06/2014	

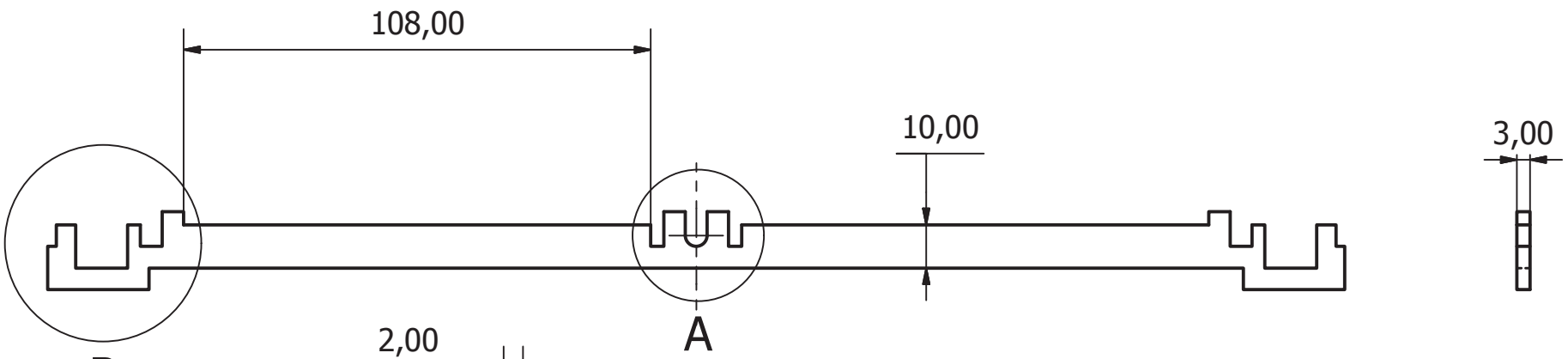
PLANOS	Balancín de ensayo de PID	
	Ensamblaje 2	Hoja 3 / 12

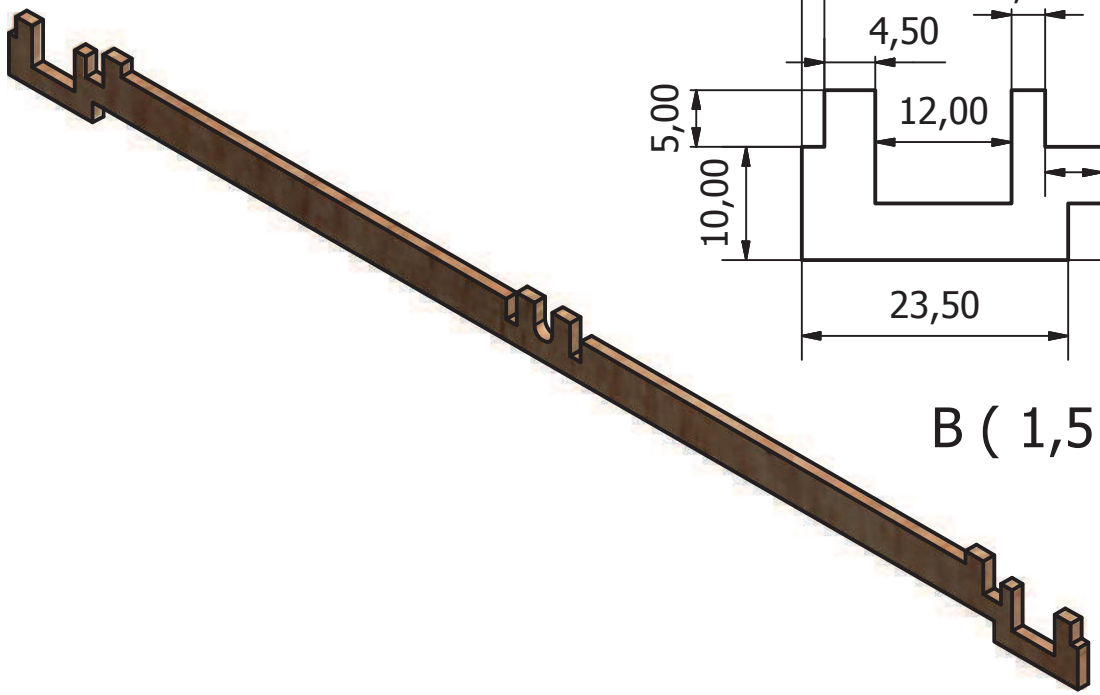




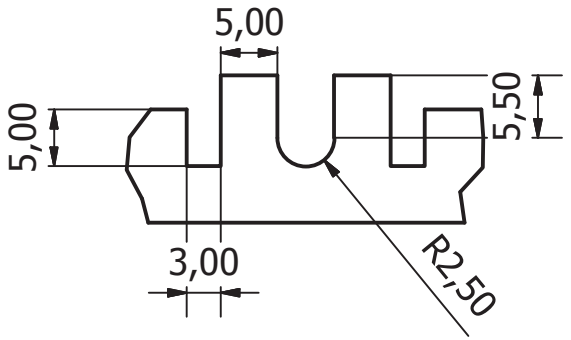
Diseñado por	Comprobado por	Cotas en	Escala	Fecha	
Victor		mm	1:2	19/06/2014	

PLANOS	Balancín de ensayo de PID		
	Pieza1		Hoja 4 / 12





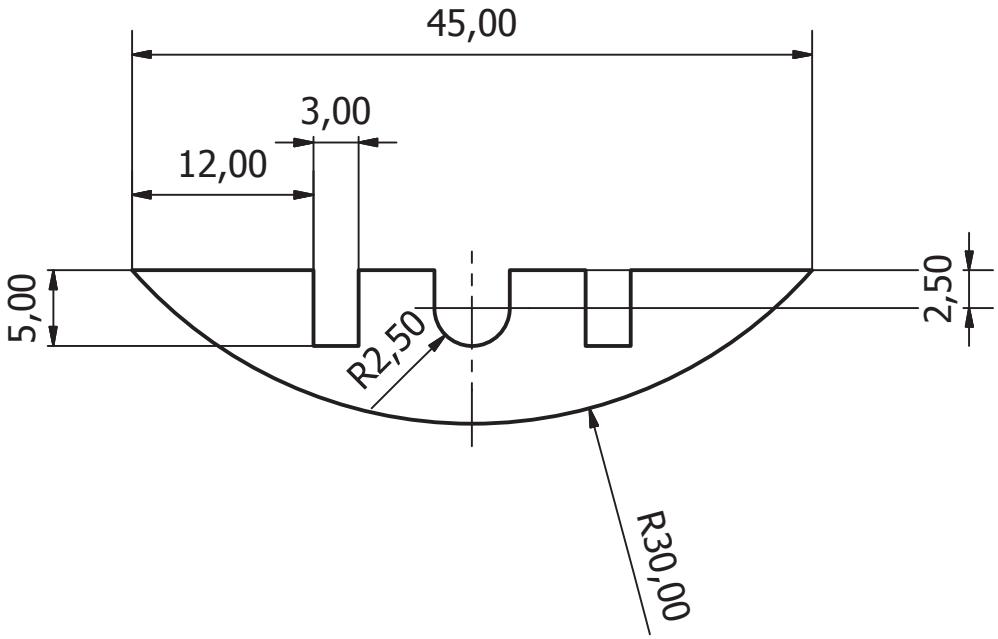
B (1,5 : 1)

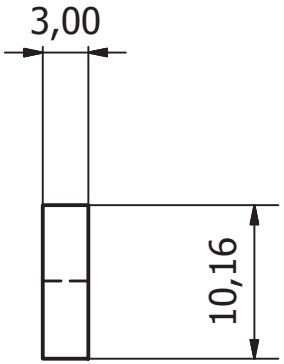


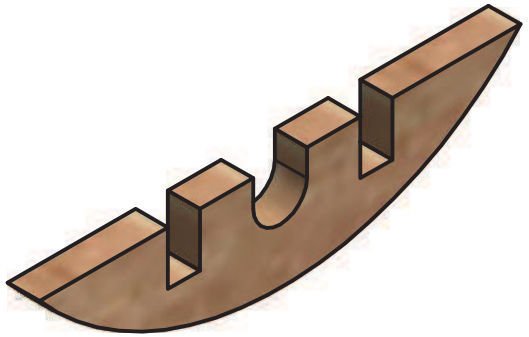
A (1.5 : 1)

Diseñado por	Comprobado por	Cotas en	Escala	Fecha	
Victor		mm	1:1.5	19/06/2014	

PLANOS	Balancín de ensayo de PID		
	Pieza2		Hoja 5 / 12

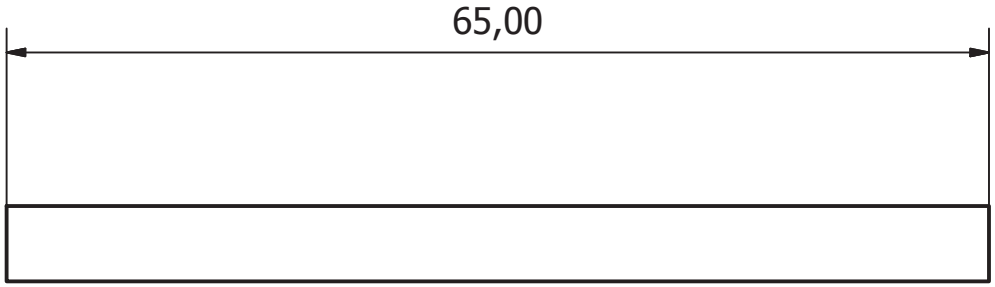


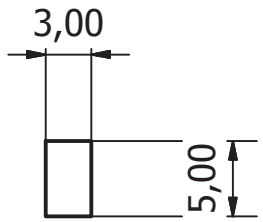


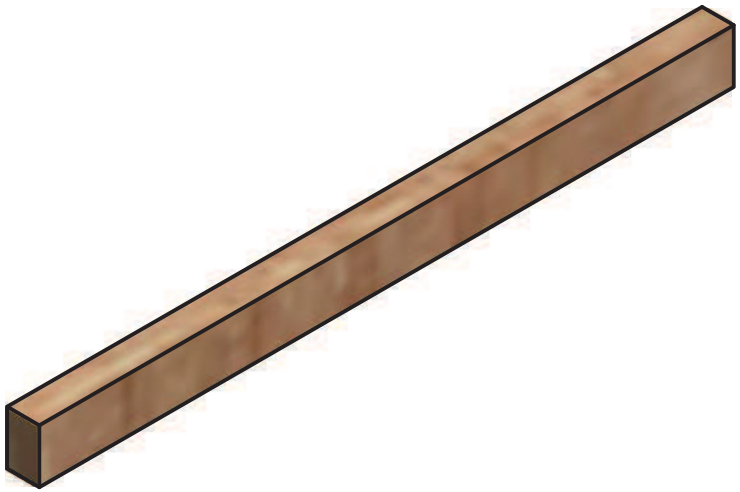


Diseñado por	Comprobado por	Cotas en	Escala	Fecha	
Victor		mm	2:1	19/06/2014	

PLANOS	Balancín de ensayo de PID		
	Pieza3		Hoja 6 / 12

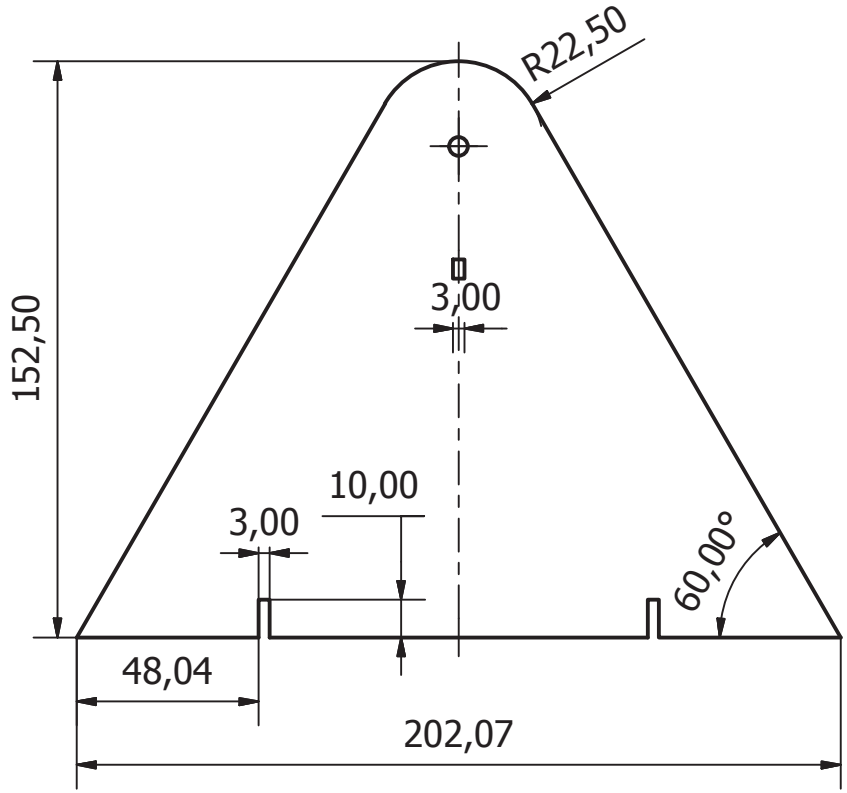


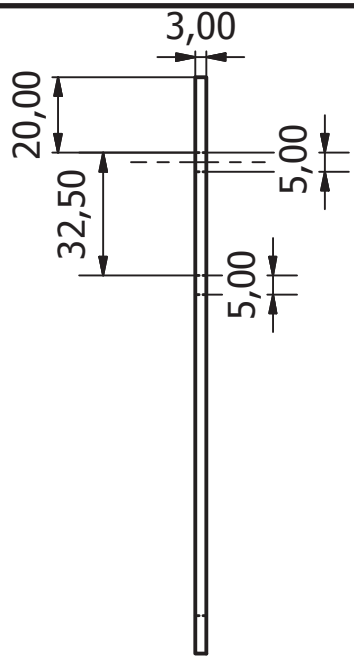


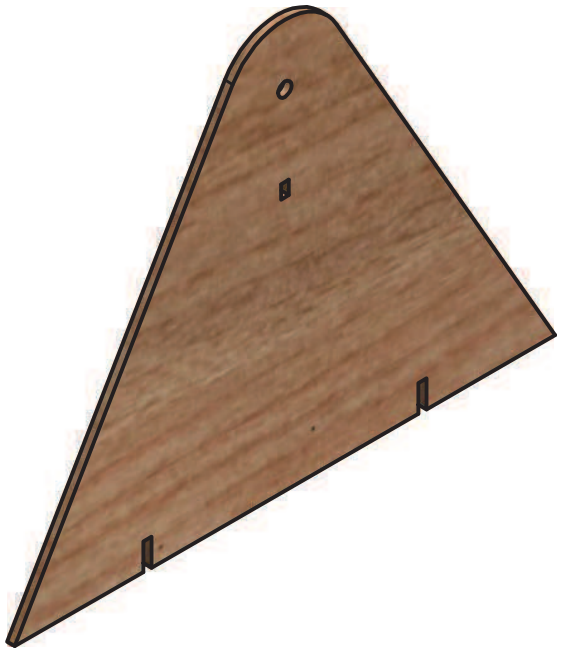


Diseñado por	Comprobado por	Cotas en	Escala	Fecha	
Victor		mm	2:1	19/06/2014	

PLANOS	Balancín de ensayo de PID		
	Pieza4		Hoja 7 / 12

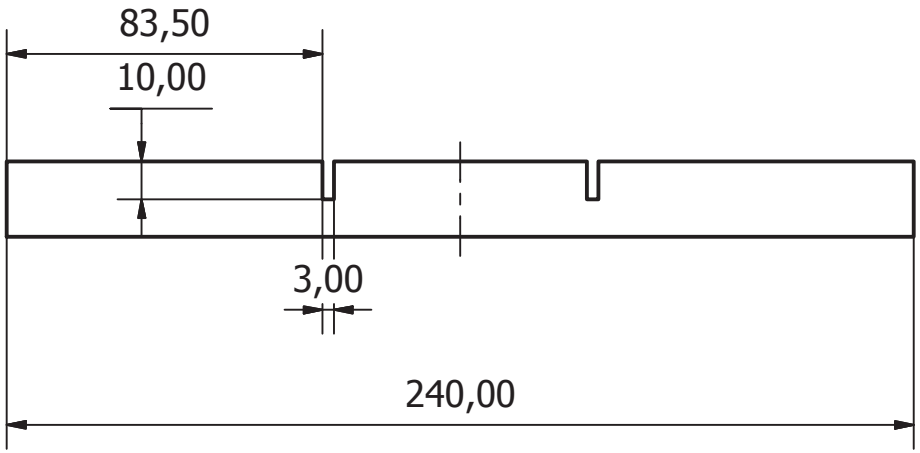


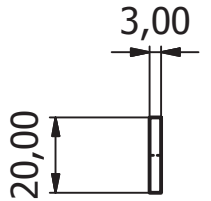


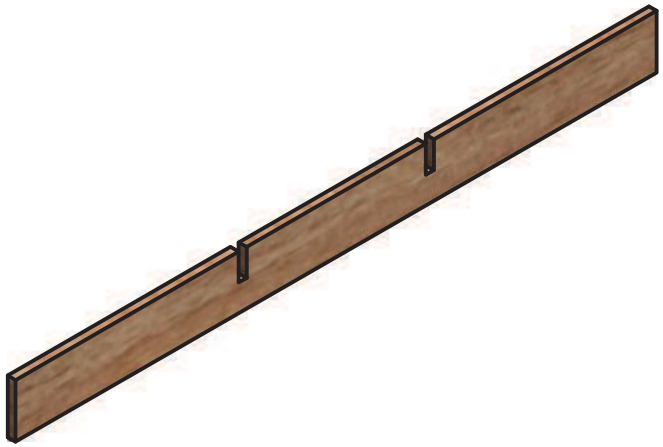


Diseñado por	Comprobado por	Cotas en	Escala	Fecha	
Victor		mm	1:2	19/06/2014	

PLANOS	Balancín de ensayo de PID		
	Pieza5		Hoja 8 / 12

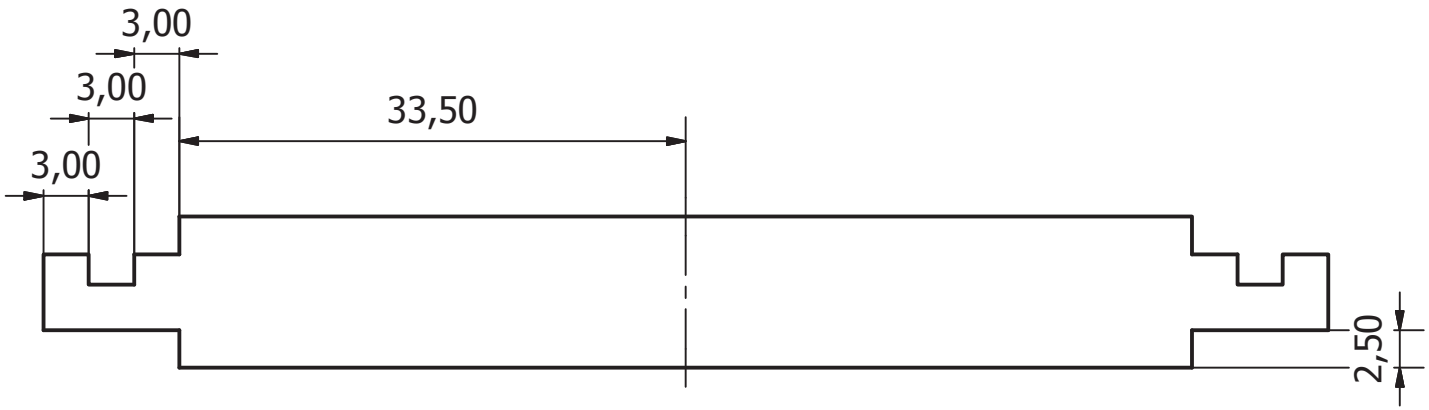


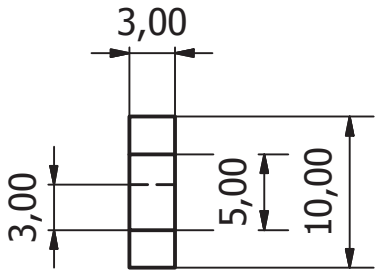


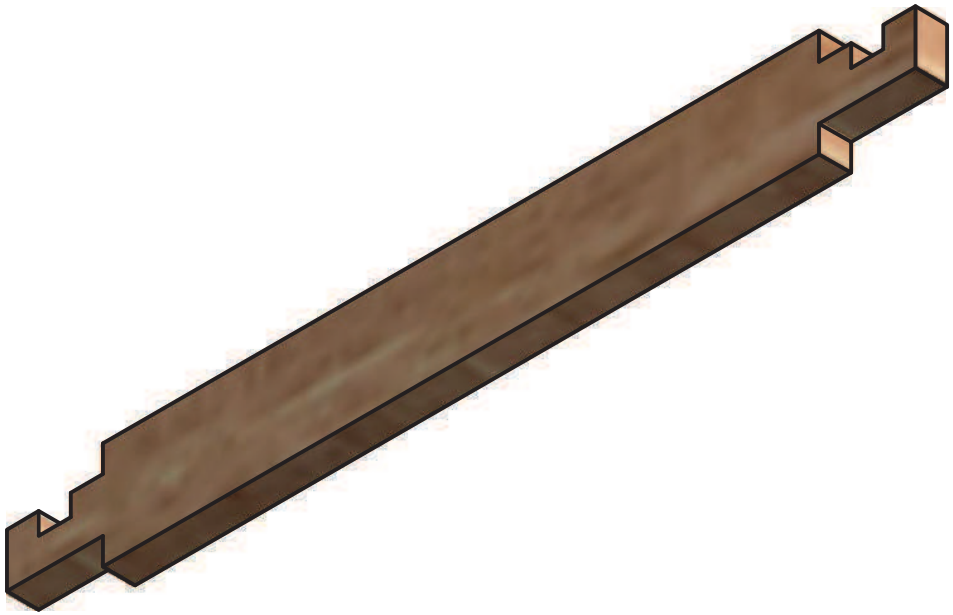


Diseñado por	Comprobado por	Cotas en	Escala	Fecha	
Victor		mm	1:2	19/06/2014	

PLANOS	Balancín de ensayo de PID		
	Pieza6		Hoja 9 / 12

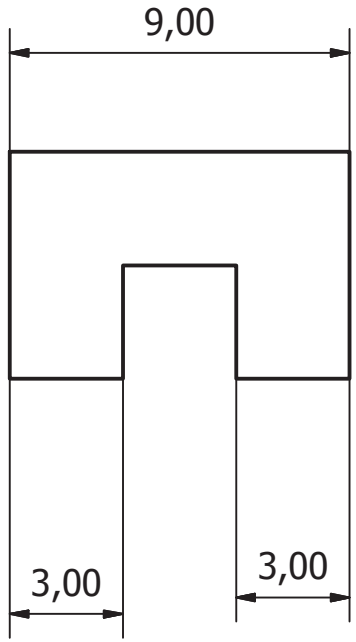


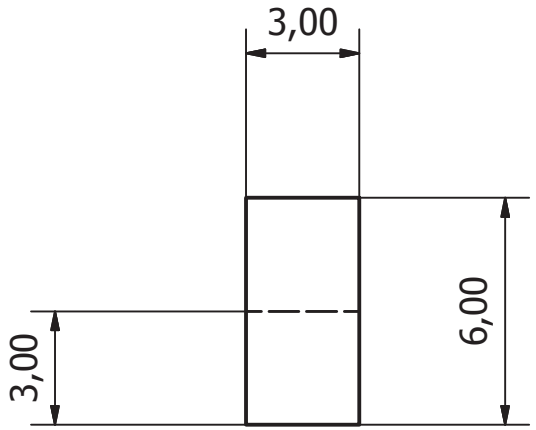


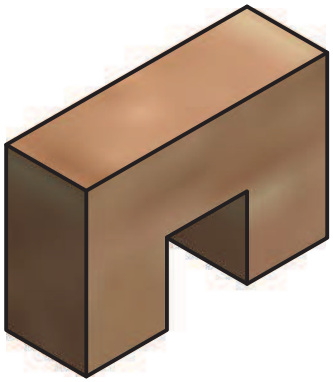


Diseñado por	Comprobado por	Cotas en	Escala	Fecha	
Victor		mm	2:1	19/06/2014	

PLANOS	Balancín de ensayo de PID		
	Pieza7		Hoja 10 / 12

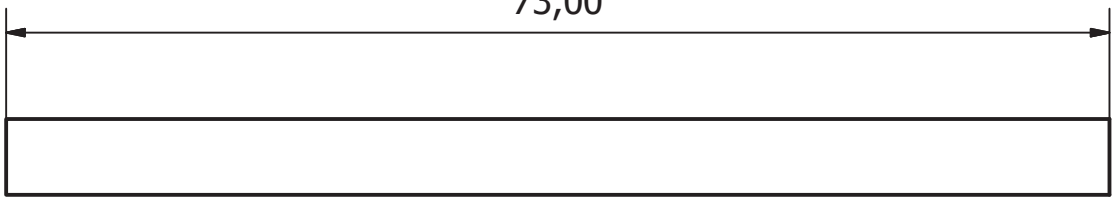


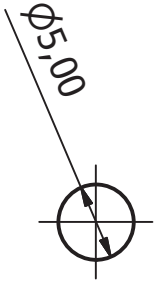


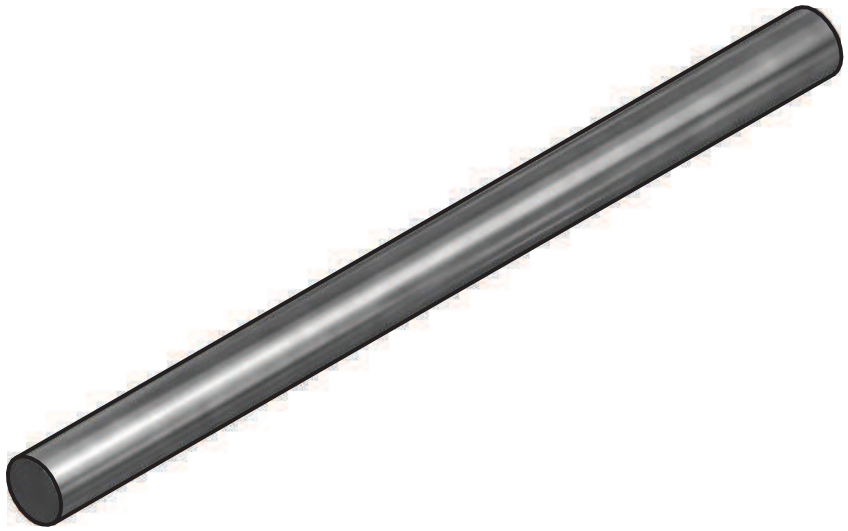


Diseñado por	Comprobado por	Cotas en	Escala	Fecha	
Victor		mm	5:1	19/06/2014	

PLANOS	Balancín de ensayo de PID		
	Pieza8		Hoja 11 / 12







Diseñado por	Comprobado por	Cotas en	Escala	Fecha	
Victor		mm	2:1	19/06/2014	

PLANOS	Balancín de ensayo de PID		
	Pieza9		Hoja 12 / 12