

Títol: Cuadricóptero Arduino por control remoto Android
Autor: Xabier Legasa Martín-Gil
Data: 3 de Juliol de 2012
Director: Pere Marés Martí
Dep. del director: Eng.Sistemes, Automàtica i Inf.Ind.
Titulació: Enginyeria Informática
Centre: Facultat d'Informàtica de Barcelona (FIB)
Universitat: Universitat Politècnica de Catalunya (UPC)
BarcelonaTech

Cuadricóptero

Arduino por control remoto Android



Proyecto Final de Carrera
Xabier Legasa Martín-Gil
Ingeniería Informática
Facultad de Informática de Barcelona (FIB)
UPC BarcelonaTech
2011/2012

Índice

1. Lista de Figuras.....	6
2. Glosario.....	8
3. Introducción.....	9
3.1. Presentación.....	10
3.1.1. Descripción general del proyecto.....	10
3.2. Motivación personal.....	10
3.3. Objetivo.....	11
3.3.1. Objetivos del proyecto.....	11
3.3.2. Objetivos personales.....	11
4. Estado del Arte.....	13
4.1. Shriquette project.....	14
4.2. Mikuadricoptero.....	14
4.3. NG UAVP.....	14
4.4. Aeroquad.....	14
4.5. DiyDrones - ArduPilot.....	15
4.6. Openpilot.....	15
4.7. Parrot AR Drone.....	15
4.8. Análisis de la oportunidad.....	17
5. Arquitectura del sistema cuadricóptero.....	18
5.1. Hardware. Diseño.....	18
5.1.1. Configuración.....	20
5.1.2. Frame.....	21
5.1.3. Motores.....	21
5.1.3.1. Brushed Motors (DC Motors).....	22
5.1.3.2. Brushless Motors.....	22
5.1.4. Batería.....	26
5.1.5. ESC (variador).....	28
5.1.6. PWD.....	30
5.1.7. Regla “watts per pound”.....	33
5.1.8. Microcontrolador.....	34
5.1.8.1. Hardware libre.....	34
5.1.8.2. Arduino.....	34

5.1.8.2.1. ¿Qué es arduino realmente?.....	35
5.2.8.2.2. ¿Por qué arduino?.....	35
5.1.9. Batería 9V.....	39
5.1.10. IMU.....	39
5.1.11. Acelerómetro.....	40
5.1.12. Giroscopio.....	41
5.1.13. I ² C.....	41
5.1.14. Módulo Bluetooth.....	42
5.1.15. Otros.....	43
5.1.16. Peso.....	43
6. Software. Diseño.....	55
6.1. Android.....	56
6.2. Arduino.....	58
7.1. Lectura de la IMU.....	61
7.1.1. Filtro de Kalman.....	61
7.2. Comunicación Android - Arduino.....	64
7.2.1. Amario.....	64
7.2.2. Android APP.....	65
7.2.3. Arduino Firmware.....	69
8. Algoritmo de estabilización. Bases.....	71
8.1. Controlador PID.....	71
8.1.2. Proporcional.....	72
8.1.3. Integral.....	72
8.1.3. Integral.....	72
8.1.4. Derivativo	73
8.2. PID en Arduino.....	73
9. Conclusiones.....	75
9.1. Objetivos conseguidos.....	75
9.2. Desviaciones de planificación.....	75
9.3. Valoración económica.....	75
10. Planificación.....	78
11. Posibilidades de trabajo futuro.....	80
11.1. Prueba del algoritmo de estabilización.....	80
11.2. Aeroquad Software.....	80

12. Referencias.....	83
13. Bibliografía.....	85

1. Lista de Figuras

Figura 1. AR Drone	13
Figura 2. Esquema general interconexión de dispositivos.....	16
Figura 3. Sentido de rotación de los cuatro motores.....	18
Figura 4. Motor de Corriente Continua (DC motor / Brushed motor).....	19
Figura 5. Diferencia entre motores con y sin escobillas.....	20
Figura 6. Motor sin escobillas (Brushless DC motor).....	20
Figura 7. Hacker Style Brushless Outrunner 20-28M.	22
Figura 8. Batería ZIPPY Flightmax 2200mAh 3S1P 20C.....	23
Figura 9. Ejemplo de ESC.....	25
Figura 10. Único ESC controlando 2 motores brushless a una misma velocidad.....	26
Figura 11. Esquema de conexión de un ESC.....	27
Figura 12. PWM.....	28
Figura 13. Turnigy AE-20A Brushless ESC.....	29
Figura 14. Logo Open Hardware.....	31
Figura 15. Logo Arduino.....	32
Figura 16. Arduino Uno.....	34
Figura 17. Yaw Pitch Roll.....	36
Figura 18. Yaw Pitch Roll.....	36
Figura 19. Sparkfun IMU Digital Combo Board - ITG3200/ADXL345.....	37
Figura 20. Parte posterior de la IMU ITG3200/ADXL345.....	39
Figura 21. “Bluetooth Modem - BlueSMiRF Silver” de sparkfun.....	39
Figura 22. Izquierda: Protoshield Arduino. Derecha: Arduino UNO.....	42
Figura 23. Esquema de conexión con IMU y Bluetooth.....	42
Figura 24. Esquema de conexión con interruptor.....	44
Figura 25. Esquema de conexión definitivo.....	45
Figura 26. arduino Uno y protoshield acabado.....	46
Figura 27. Protoshield acabado.....	46
Figura 28. Protoshield sobre Arduino Uno.....	47
Figura 29. Protoshield sobre Arduino Uno.....	48
Figura 30. IMU y Bluetooth en la placa Protoshield.....	48
Figura 31. Protoshield acoplado al micro.....	49

Figura 32. Motor brushless con hélice, anclado.....	50
Figura 33. Conexión entre motor y ESC.....	50
Figura 34. Cableado eléctrico de la parte inferior interior.....	51
Figura 35. Arduino instalado dentro del frame.....	51
Figura 36. Arduino instalado dentro del frame.....	52
Figura 37. Logo Android.....	54
Figura 38. Logo Arduino.....	55
Figura 39. IDE Arduino.....	56
Figura 40. Ejemplo Arduino.....	57
Figura 41. Serial Monitor.....	58
Figura 42. Filtro de Kalman.....	59
Figura 43. Ángulos de Euler.....	61
Figura 44. Hegalari app. Menú.....	62
Figura 45. Seekbar que controlará la velocidad de rotación.....	63
Figura 46. FourMotorActivity.....	63
Figura 47. Esquema PID.....	68

2. Glosario

Arduino

Plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de usar. Se creó para artistas, diseñadores, aficionados y cualquiera interesado en crear entornos u objetos interactivos.

Android

Sistema operativo para plataformas móviles como smartphones, tablets..

UAV

(Unmanned aerial vehicle). Vehículo Aéreo no tripulado. Sistema autónomo que puede operar sin intervención humana alguna durante su funcionamiento.

Aeronave RC

Vehículo aéreo controlado remotamente.

Multirotor o Multicóptero

Helicóptero con más de dos rotores.

3. Introducción

Un cuadricóptero, cuadrirotor o cuadrotor es un helicóptero que consta de cuatro rotores mediante los cuales se propulsa. Estos rotores se sitúan por lo general en las extremidades de una cruz sus hélices giran en un mismo plano. El control de movimiento del aparato se consigue, ajustando la velocidad de rotación de cada hoja.

La razón principal de que los cuadricópteros hayan aparecido hace relativamente poco, puede atribuirse a varios factores. El obstáculo principal es la necesidad de calcular la velocidad de las hélices en tiempo real con tal de mantenerlo estable. Esto hace que el algoritmo de control sea complejo: El sistema tiene que retroalimentarse con datos sobre su estado actual y actuar en consecuencia al momento. Sin embargo, a medida que la tecnología ha ido avanzando, la precisión y carga de computación necesarias han dejado de ser una barrera. Gracias a la tecnología más precisa de sensores y procesadores que existe hoy en día, es posible hacer volar estos aparatos. De ahí el auge que han experimentado estos sistemas en los últimos tiempos.

3.1. Presentación

3.1.1. Descripción general del proyecto

El proyecto descrito en este documento consiste en la realización del diseño e implementación de un sistema físico cuadricóptero. El aparato tendrá que ser capaz de comunicarse remotamente para que sea posible controlar su dirección. Para ello, se habrán de analizar cuales son las alternativas existentes, tanto respecto al hardware como al software se refiere.

3.2. Motivación personal

La idea de este proyecto surgió del cúmulo de varias circunstancias. Me encontraba a falta de un año de acabar la carrera y me apetecía realizar un PFC que fuera diferente. Algo que me forzara a aprender, pero que me permitiera a la vez aplicar parte de lo que he aprendido todos estos años. Un amigo francés, me envió la URL de la plataforma Arduino[1] y empecé a interesarme más por la electrónica. Al poco tiempo, otro amigo me envió un video de un micro-cuadricóptero controlado por radio control. Tras darle vueltas al tema, me decidí a juntar los conceptos Arduino+Android+Cuadricóptero y me puse manos a la obra.

3.3. Objetivo

El objetivo principal del presente proyecto es el diseño de un sistema físico cuadricóptero de experimentación. El sistema constará de cuatro rotores que estarán controlados desde un microcontrolador Arduino el cual a su vez recibirá información de los sensores necesarios para una correcta estabilización de vuelo del cuadricóptero.

Por otra parte se dotará al sistema de un control remoto inalámbrico desde un smartphone Android. En el proyecto se distinguen objetivos más generales y otros más específicos:

3.3.1. Objetivos del proyecto

- Generar la plataforma cuadricóptero, haciendo uso de un microcontrolador y sensores adecuados
- El cuadricóptero ha de ser capaz de comunicarse con una aplicación que corra en el sistema operativo Android.

3.3.2. Objetivos personales

- Aprendizaje sobre vehículos RC(control remoto).
- Afrontar un problema que combine hardware + software.
- Profundizar los conocimientos de electrónica de los que dispongo.
- Aprendizaje del entorno Arduino
 - Lenguaje de programación Arduino (basado en Wiring)
 - IDE Arduino (basado en Processing)
- Profundizar en el diseño de aplicaciones Android.

- Utilización del IDE Eclipse en conjunto con la SDK de Android.
 - Sensores intrínsecos en teléfonos Android.
- Aprender el funcionamiento básico de diferentes elementos electrónicos.

4. Estado del Arte

Es necesaria una fase inicial de análisis para determinar el alcance, objetivo, requisitos del proyecto. También habrá que hacer una selección del hardware y herramientas de software del que se hará uso.

En este capítulo se va a abordar las fase previa al desarrollo. Se va a poner la atención en diferentes cuadricópteros existentes que nos ayudará durante el análisis y desarrollo de nuestra idea.

Existen una cantidad reseñable de proyectos que afrontan el reto de construir multicópteros o helicópteros de múltiples motores. Dependiendo del enfoque desde donde los miremos, se pueden dividir de varias maneras:

Amateur vs Comercial

Por una lado, existen proyectos amateur que sin ánimo de lucro. Se trata más bien de compartir experiencias entre amantes de este tipo de aparatos. En otros en cambio, el enfoque comercial es claro. Se suelen vender cuadricópteros enteros o partes del mismo. Además, en muchos de ellos se ve un claro desarrollo continuo del producto con el objetivo de mejorarlo.

Open source vs Propietario

Por otro lado, podríamos separar entre proyectos de código abierto y propietario. Existen varias comunidades, que desarrollan software de código abierto, orientadas a determinadas plataformas. Otras en cambio no publican su código. Pero no solo eso. Al tratarse de aparatos físicos, también los esquemas de alguna(s) parte(s) del hardware pueden hacerse públicas o no. Un claro ejemplo de hardware open-source es Arduino. Se explica más adelante en este documento.

Atendiendo a estas dos maneras de clasificación, obtenemos 4 combinaciones posibles. Se ha realizado un análisis de los proyectos más reseñables de diferentes combinación poniendo especial atención en proyectos de código abierto:

4.1. Shriquette project

Iniciado a finales de 2008 en Alemania, Shriquette project[3] es un proyecto amateur que surge con el ánimo de profundizar en conocimientos de programación, electrónica y control. El código se publica bajo un tipo de licencia Creative Commons.

El primer modelo es un tricóptero, pero le siguen otros modelos con un número mayor de rotores. Dispone de un giroscopio de tres ejes y un acelerómetro.

4.2. Mikuadricoptero

Mikuadricoptero[6] es otro ejemplo de un proyecto individual, en este caso escrito en castellano, que describe el diseño de un cuadricóptero.

4.3. NG UAVP

NG UAVP[9] es un proyecto de RC de código abierto que se inicia en 200. Se apoya en una comunidad de desarrolladores.

4.4. Aeroquad

AeroQuad[4] es un proyecto de código abierto de hardware y software que se dedica a la construcción de cuadricópteros controlados remotamente. Es un proyecto muy vivo, con una gran comunidad detrás.

4.5. DiyDrones - ArduPilot

Al estilo del anterior, se trata de un proyecto de código abierto. Abarca un gran número diferente de dispositivos, desde multirotores y aviones hasta vehículos terrestres. Se basa en ArduPilot, un sistema de pilotaje remoto compatible con la plataforma Arduino. Implementa un hardware llamado APM_2 en el cual se puede correr diferentes programas de código abierto específicos para Arduino.

4.6. Openpilot

Openpilot[7] es otra iniciativa que se inicia en 2010, muy del estilo de Aeroquad y DiyDrones. Como su propio nombre indica, se trata de código abierto, y está pensado para diferentes tipos de vehículos aéreos.

Cabe destacar que son países como EEUU o Alemania donde he encontrado más proyectos que desarrollen cuadricópteros. Además, es claramente en EEUU donde la faceta comercial está más marcada. El modelo de negocio suele ser parecido: Se encuentran claramente orientados a la venta de hardware específico que ellos fabrican, para el cual desarrollan el software. Este código lo abren, permitiendo que los usuarios lo modifiquen y mejoren a su gusto. Existe sin embargo un proyecto de especial interés, llevado a cabo por una empresa Francesa, el cual paso a detallar más profundamente.

4.7. Parrot AR Drone

Proyecto reciente de la empresa Francesa Parrot[8]. En el año 2010 presentaron su cuadricóptero AR Drone[9] en una conferencia de Las vegas: Un multirotor con cubierta de plástico y espuma de aproximadamente 30 centímetros de largo.



Figura 1. AR Drone

Tiene dos micro-cámaras incorporadas que permiten grabar video. Es este hecho el hace que a menudo presenten el producto como “The Flying Video Game”. Se controla mediante una aplicación para smartphones que está disponible tanto para dispositivos iOS / como para Android. La comunicación es vía Wi-Fi. Internamente, contiene hardware propio que incluye dispositivos habituales como son el acelerómetro, giroscopio, altímetro de ultrasonidos que permite controlar la altitud,...

El software corre sobre el sistema operativo **Linux**. Es un producto completamente comercial y han decidido mantener el código como propietario. Sin embargo, dispone de una API con la que se permite programar hasta cierto punto el comportamiento del drone. Tiene la habilidad de reconocer objetos 3D y es compatible con juegos que introducen realidad aumentada.

En cuanto a las especificaciones más técnicas, dispone de un microcontrolador ARM9 468 MHz embebido con 128 Megabytes de RAM. Dispone de comunicación mediante Wi-Fi y USB. Un acelerómetro de 3 ejes, dos giroscopios y un altímetro ultrasónico. La estructura está fabricada en fibra de carbono. Posee 4 motores eléctricos de 15 watt de tipo brushless y una batería recargable de 1000 miliAmperios de Litio que proporciona 11.1 voltios. Con un peso de entre 380/420 gramos es capaz de volar durante 12 minutos con una velocidad de 5 metros/segundo o lo que es lo mismo, 18 km/h.

Su precio a fecha de 1 de Junio de 2012 es de 300€.

4.8. Análisis de la oportunidad

Si bien existen bastantes proyectos en el mundo de los cuadricópteros, existen muy pocos que sean controlados mediante un smartphone. La tendencia hacia el uso masivo de este tipo de teléfonos permiten reducir costes en el ámbito de los dispositivos controlados por control remoto, ya que no hace necesario el uso de otros sistemas de transmisión más caros como son el radio control. Si hablamos de cuadricópteros controlados mediante móvil, hay que destacar que *Parrot AR Drone* ha implementado esta idea recientemente. Sin embargo, no se ofrecen esquemas del hardware y el código es propietario, por lo que se habrá de analizar y ver cual es la mejor manera de llevar a cabo el proyecto.

La idea inicial es, realizar el desarrollo basándonos en plataformas abiertas y con un coste económico lo más reducido posible.

5. Arquitectura del sistema cuadricóptero

Se ha realizado un análisis para determinar qué elementos serán necesarios para llevar a cabo proyecto. Podemos diferenciar claramente el trabajo realizado entre aspectos que tienen que ver con el hardware o la estructura física del aparato y por otro lado, todo lo que tiene que ver con el tema lógico o de software. Vamos a ver en más detalle el camino recorrido.

5.1. Hardware. Diseño.

Se ha llevado a cabo un estudio de todos y cada uno de los componentes físicos que compondrán el cuadricóptero. Antes de entrar a explicar más en profundidad cada uno de ellos, conviene antes hacer un repaso rápido para poder entender cual es la relación de cada uno de ellos con el resto.

Comenzamos por el **frame** o armazón que será el soporte físico donde irán anclados todos los demás componentes. Tendrá forma de cruz y en cada uno de los extremos irá anclado un **motor**. Habrá que determinar qué tipo de motor se utiliza y cómo va a ser controlado. En este caso se ha optado por la utilización de motores brushless (sin escobillas). Cada uno de ellos será controlado por un **ESC** (Electronic Speed Controller) o también llamado variador, cuyo funcionamiento se explica más abajo en este capítulo. El ‘cerebro’ del sistema será un **microcontrolador** Arduino. Este será el encargado de mandar sobre los ESCs que serán los encargados últimos de mover los motores.

Se ha decidido separar en dos la alimentación del sistema. Por un lado hemos añadido una **pila** que se encargará de alimentar el microcontrolador y todos los componentes unidos a él. Por otro lado se encuentra una **batería** mayor que será la encargada de alimentar los motores. Esta corriente sin embargo, ha de pasar por los ESCs primero que serán los encargados de administrar electricidad a los motores. O si se quiere decir de otra manera, los motores se alimentan de electricidad a través de los ESCs.

Anclados al microcontrolador, tenemos también una **IMU** (Inertial Measurement Unit) que consta de un **acelerómetro** y un **giroscopio** en una único board o tablero, ambos de tres ejes. La filosofía y funcionamiento de estos aparatos se explica más adelante en este capítulo, pero a groso modo, son los

sensores necesarios para saber la inclinación en todo momento del sistema. Para realizar la conexión entre el micro y el smartphone, también hemos añadido a la placa un **módulo bluetooth**.

Las **hélices** unidas a los motores crearan la propulsión necesaria.

He creado el siguiente esquema (figura 2) que sirve para hacerse una idea de la interconexión de elementos con más facilidad. El esquema se completa después pero sirve para ver el micro por un lado (con la IMU y el módulo Bluetooth) y los ESCs y motores por otro.

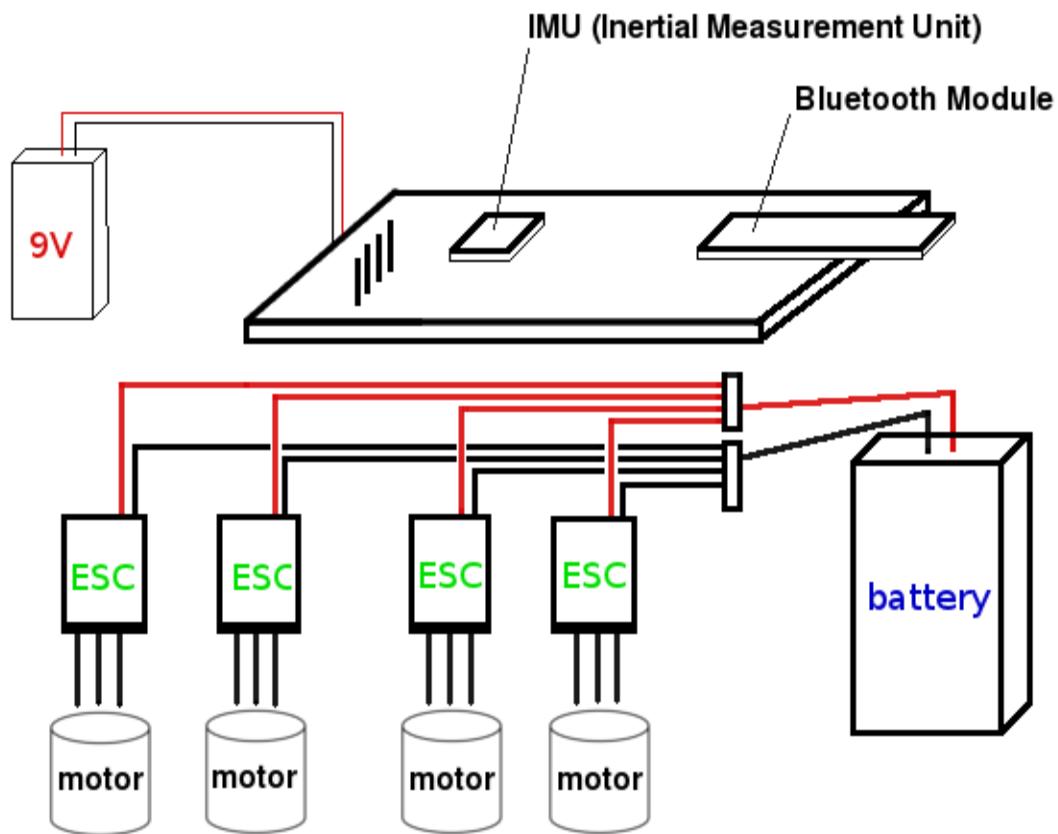


Figura 2. Esquema general interconexión de dispositivos.

La lista por tanto queda de la siguiente manera:

- Frame.
- 4 Motores sin escobillas.
- 4 ESCs (Electronic Speed Controller).
- Microcontrolador Arduino.
- IMU (Inertial Measurement Unit).
- Modulo Bluetooth.
- Bateria (para micro Arduino).
- Batería (motores).
- 4 Helices.

5.1.1. Configuración

Desde el inicio de este documento se ha hablado de un sistema cuadricóptero. Antes sin embargo, se planteó la posibilidad de diseñar un multirotor con un número diferente de motores diferente a cuatro. Un tricóptero habría sido una posibilidad muy válida. Este tipo de sistemas hacen necesario el uso de un servo en la parte posterior de su estructura. El servo sería el equivalente a la cola de un pájaro o de un pez. Se descartó porque implica una dificultad extra a la hora del montaje y la estabilidad de este sistema no es mayor que un cuadricóptero, por ejemplo. También se contempló la posibilidad de montar un hexacóptero, o cualquier sistema con más de 4 motores. Se descartó porque el coste económico aumenta.

Tomada pues, la decisión de construir un sistema cuadricóptero, la siguiente decisión a tomar, es si se utilizará una distribución en “x” o en “+”. A la hora de controlar los motores, la configuración en “+” implica una mayor sencillez, por lo que se ha optado por esta opción. Un motor estará situado en la parte delantera y otro en la trasera.

El sentido de giro de cada hélice es de vital importancia. Haremos girar dos de las hojas en sentido horario (CW) y otras dos de ellas en sentido contrario(CCW) como en la figura 3. Si todas giraran en las agujas del reloj, por ejemplo, el cuadricóptero comenzaría a girar en sentido inverso continuamente.

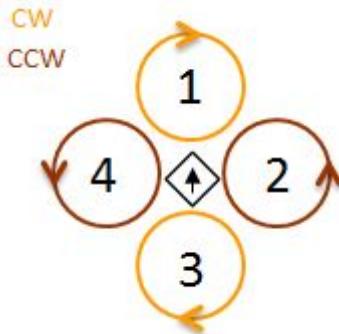


Figura 3. Sentido de rotación de los cuatro motores.

Pasemos por tanto a analizar los elementos necesarios para un cuadricóptero configurado en +.

5.1.2. Frame

Existen muchas posibilidades a la hora de elegir frame. Habrá que decidir respecto al material, tamaño y forma del mismo. Para ello tendremos que tomar en cuenta criterios como el peso, la aerodinámica y el precio. En cuanto al peso, está claro que un material demasiado pesado sería negativo, puesto que tendríamos que aumentar la fuerza para lograr el mismo resultado de propulsión. Si nos vamos al otro extremo y elegimos un material demasiado ligero corremos el riesgo de que este, sea demasiado frágil. Existen sin embargo materiales que aun siendo ligeros, siguen siendo igual de fuertes que los primeros o más. También hay que tener en cuenta que una mayor flexibilidad del cuadricóptero evitará posibles daños en su estructura en caso de caída. El ejemplo perfecto de material ligero, duro y flexible es la fibra de carbono y derivados o la fibra de carbono y vidrio. Sin embargo, se ha elegido un frame (semi ensamblado de fábrica) de contrachapado de madera por varias razones. La principal es el precio. Como se ha dicho, la fibra de carbono es un material más adecuado pero es bastante más caro. Al final del presente documento hay un resumen de precios. El frame tiene 3mm de grosor y pesa 195 gramos.

5.1.3. Motores

La elección de los motores es una decisión importante. Existen dos tipos. Los motores con escobillas (brushed motors o DC motors) y los motores sin escobillas llamados, brushless motors o brushless DC motors.

5.1.3.1. Brushed Motors (DC Motors)

Según la wikipedia, “el motor de *corriente continua* es una máquina que convierte la energía eléctrica continua en mecánica, provocando un movimiento rotatorio” [10]. Su popularidad y uso descendió mucho con la llegada de los motores de corriente alterna, aunque se siguen utilizando en determinadas industrias. La principal característica del motor de corriente continua es la posibilidad de regular la velocidad desde vacío a plena carga. Se compone principalmente de dos partes, un estator que da soporte mecánico al aparato y tiene un hueco en el centro generalmente de forma cilíndrica. En el estator además se encuentran los polos, que pueden ser de imanes permanentes o devanados con hilo de cobre sobre núcleo de hierro. El rotor es generalmente de forma cilíndrica, también devanado y con núcleo, al que llega la corriente mediante dos escobillas. Requieren mucho mantenimiento.



Figura 4. Motor de Corriente Continua (DC motor / Brushed motor).

5.1.3.2. Brushless Motors

También llamados Brushless DC motors, son motores que carecen de colector y escobillas o carbones. En vez de funcionar en DC, funcionan con una señal trifásica que aunque idealmente debería de tener una forma sinusoidal, en la práctica son pulsos haciendo que la señal sea una continua.

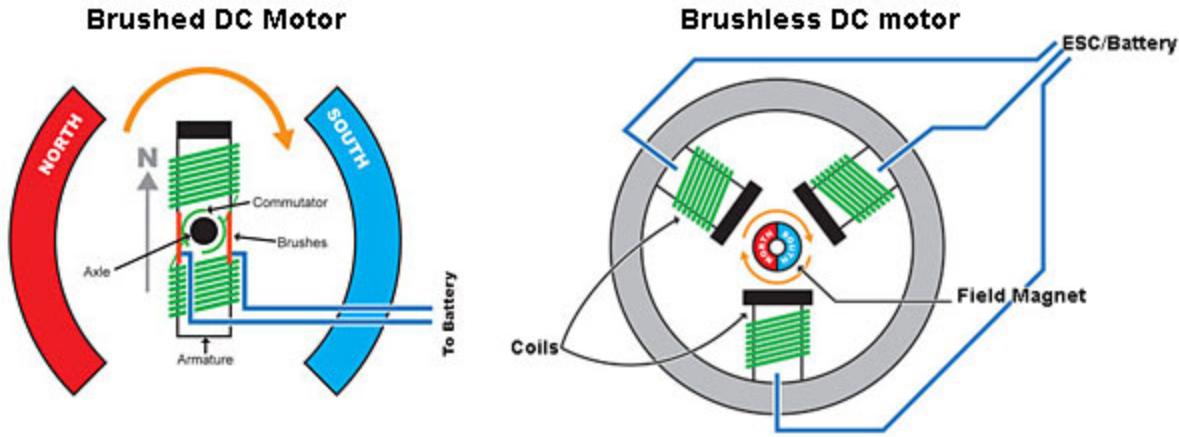


Figura 5. Diferencia entre motores con y sin escobillas.

Como se puede apreciar en la figura 5, en los motores brushless, las bobinas (“coils” en inglés) rodean los imanes. Es el ESC, como se explicará más adelante, el encargado de activar estas bobinas consecutivamente, haciendo que el rotor gire. El motor dispone de sensores que son capaces de detectar la orientación del rotor en cada momento, para así poder activar y desactivar las bobinas en el momento adecuado. Los imanes del centro por tanto, son atraídos por la polaridad de un campo magnético generado en las bobinas, las cuales reciben pulsos en un patrón específico. Si queremos que el motor gire mas rápido, simplemente hacemos girar el campo magnético secuencial a mayor velocidad. O lo que es lo mismo, tendremos que aumentar la frecuencia de los pulsos. Más adelante se explica como los ESCs generan estos pulsos.



Figura 6. Motor sin escobillas (Brushless DC motor).

Cada uno de los dos tipos de motor que hemos visto tiene sus pros y sus contras.

Motores CON escobillas

- + Control simple y sencillo. No hace falta ningún tipo de controlador.
- + Pueden operar en situaciones extremas, al no haber elementos electrónicos.
- Precisan mantenimiento periódico.
- Disipación del calor pobre, debido a la construcción interna del rotor.
- Rango de velocidad menor, debido a limitaciones de las escobillas
- Las escobillas generan ruido causando Interferencias Magnéticas Eléctricas (EMI)

Motores SIN escobillas

- + Comutación electrónica basada en sensores de posición VS cambio mecánico
- + Mantenimiento menor al no haber escobillas.
- + Mayor eficiencia.
- + Disipa mejor el calor.
- + Mayor rango de velocidades. No hay limitaciones mecánicas por escobillas.
- + Se genera un menor ruido eléctrico (EMI)
- Algo mas caros.
- Los ESCs se hacen necesarios.

De la tabla anterior se concluye que merece la pena utilizar motores sin escobillas, aunque para ello sea necesaria la incorporación de ESCs. Son muchas las ventajas que conseguiremos comparado con los inconvenientes. Utilizaremos motores brushless por tanto.

Dentro de la familia de los brushless, existen dos tipos diferentes. De tipo *Inrunner* y *Outrunner*. En los motores outrunner, los imanes están situados en el exterior de la estructura. Por tanto, se puede ver como la parte exterior del motor, gira. Sucede lo contrario en los motores inrunner. Los imanes están en el interior de la estructura, y por tanto se ve que lo único que gira es el eje.

Los motores **outrunner** giran mucho más despacio y el par es mucho mayor. La mayor ventaja es el hecho de que no es necesario una caja de cambios, lo que los hace mas silenciosos. Son ligeramente menos eficientes que los inrunner, pero es tan pequeña la diferencia que no debiera de ser un factor determinante a la hora de hacer la elección.

Los motores **inrunner** por su parte, son más eficientes cuanto más rápido gira el motor y en general son más eficientes que los outrunner. Necesitan de un elemento adicional entre el motor y la hélice. La parte negativa de los motores inrunner es que estas partes adicionales pueden y suelen dar problemas.

Se ha decidido utilizar motores brushless outrunner por su mayor fiabilidad. El modelo que se ha elegido es un “hacker Style Brushless Outrunner 20-28M”.



Figura 7. Hacker Style Brushless Outrunner 20-28M.

Las características de este motor son:

Dimensions: 28x28mm

Rating: **1050kv**

Battery Config: **3 Lipo Cells**

Shaft(eje): 3.175mm

Weight(peso): 43gr

Standard Current: **4-12A**

Max Current: **15A**

'rpm' se refiere al número de rotaciones completadas cada minuto por un cuerpo que gira alrededor de un eje. kv es el número de rpm-s que un motor girará, por cada volt aplicado. Más adelante, se ha elegido una batería de 3 celdas (11.1V) con lo que calculamos ya el número de giros completos que hará el eje de nuestro motor por minuto (revoluciones por minuto de nuestro sistema).

$$kv = \text{rpm} / V$$

$$\text{rpm} = kv * V = 1050kv * (\text{Voltios}_\text{batería})$$

$$\text{rpm} = kv * V = 1050kv * 11.1V = \mathbf{11655 \text{ rpm}}$$

5.1.4. Batería

La capacidad de una batería se mide por el ratio de descarga. El ratio de descarga C de una batería, es la máxima cantidad de corriente que puede proporcionar. Como ejemplo:

1300ma (1.3A) 12C battery can deliver (1.3A x 12) 15.6A

Hemos elegido la batería que se muestra en la figura 7.



Figura 8. Batería ZIPPY Flightmax 2200mAh 3S1P 20C.

Spec.

Capacity: **2200mAh**

Voltage: 3S1P / 3 Cell / **11.1v**

Discharge: **20C** Constant / 25-30C Burst

Weight: 180g (including wire, plug & shrink wrap)

Dimensions: 102x37x24mm

Balance Plug: JST-XH

Discharge wire: 8cm high strand count soft silicon wire. 12AWG

Discharge plug: XT60

Se trata de una batería de 3 celdas(3S) de 2200mAh que es lo mismo que decir que es capaz de dar 2200mA (o 2.2 Amperios) en una hora. 20 C es el ratio de descarga o la máxima cantidad de corriente que puede proporcionar. Podemos calcular la corriente total (I)

2200ma (2.2A) 20C battery can deliver $I = (2.2A \times 20) = 44A$

Si dividimos los 44A entre los 4 motores que tenemos, tenemos **11A**. Vemos que esto encaja perfectamente con la especificación de los motores: Standard Current: 4-12A. Max Current: 15A

Podemos calcular los watt que tendrá cada motor y los caballos que eso supone como curiosidad:

$$\text{watts} = \text{Voltios} * \text{Corriente} = 11.1 \text{ v} * 11 \text{ amps} = 122.1$$

$$[1 \text{ horse power} = 746 \text{ watts} \rightarrow 122.1 / 746 = 0.1367 \text{ horses}]$$

5.1.5. ESC (variador)

Un ESC (Electronic Speed Controller) o en castellano variador, es un circuito electrónico cuyo objetivo es variar la velocidad de un motor eléctrico pudiendo jugar también el papel de freno dinámico. Utilizaremos los ESCs para controlar nuestros motores brushless. Cada ESC proporcionará a cada motor una señal eléctrica trifásica que generará la rotación de las hojas.

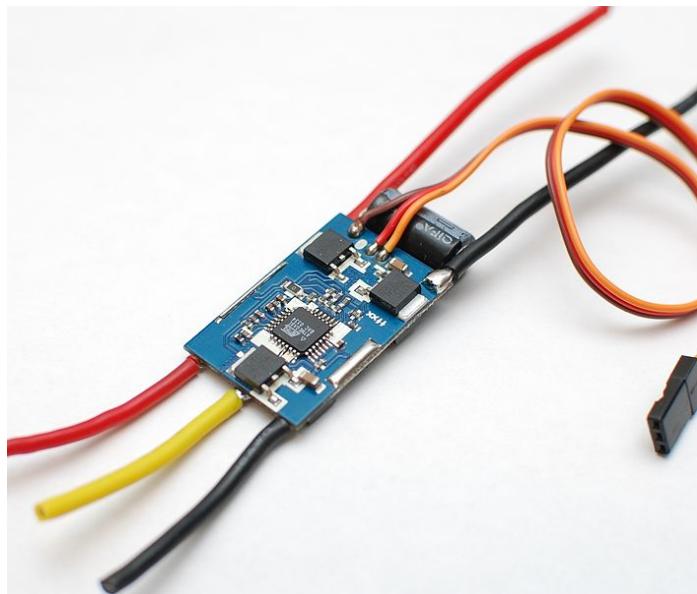
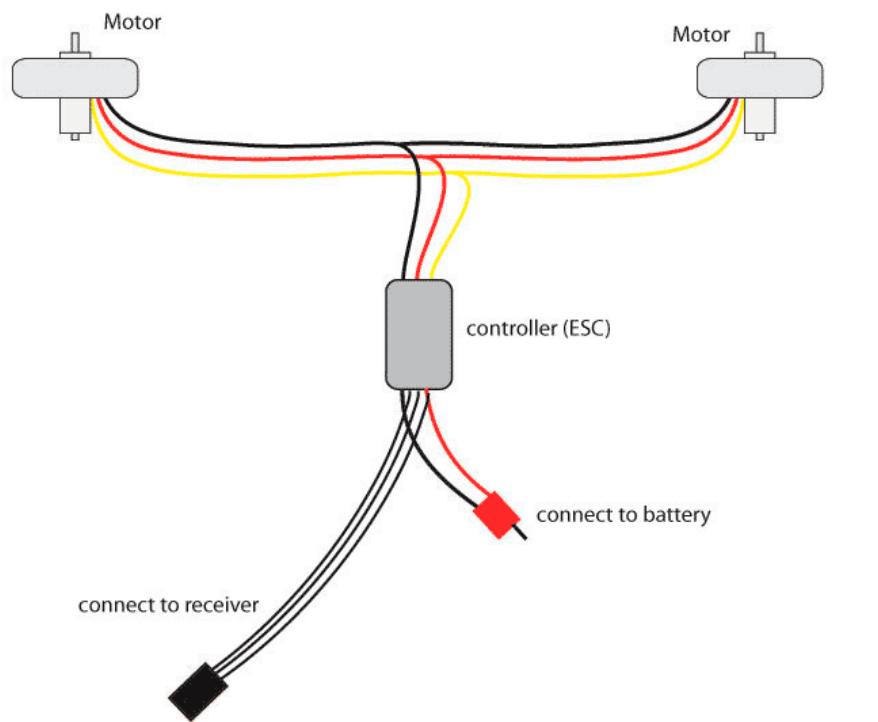


Figura 9. Ejemplo de ESC.

Un ESC al fin y al cabo es un controlador PWM[11] para motores eléctricos. El concepto de PWM se explica más adelante en este mismo apartado.

Por un lado, un ESC (figura 8), dispone de dos cables por los cuales se alimenta y que irán conectados a la batería. Por otro lado tendrá los tres cables que irán al motor. Además, dispone de 3 cables, que irán conectados al microcontrolador. Es mediante estos tres cables por los que el micro le indica al ESC a qué velocidad quiere que el motor gire. Si quisieramos controlar dos motores a la misma velocidad, sería suficiente con tener un ESC como en la figura 9. El cable donde se indica “connect to receiver” iría conectado al microcontrolador.



www.PteroWorks.com

Figura 10. Un solo ESC controlando 2 motores brushless a una misma velocidad.

En nuestro caso necesitamos controlar 4 motores y cada uno con una velocidad diferente, por lo que el esquema anterior no nos es válido. Conectaremos cada ESC independientemente al microcontrolador y a un motor cada uno. He redibujado uno de los esquemas anteriores, dejando un solo ESC para una mayor claridad.

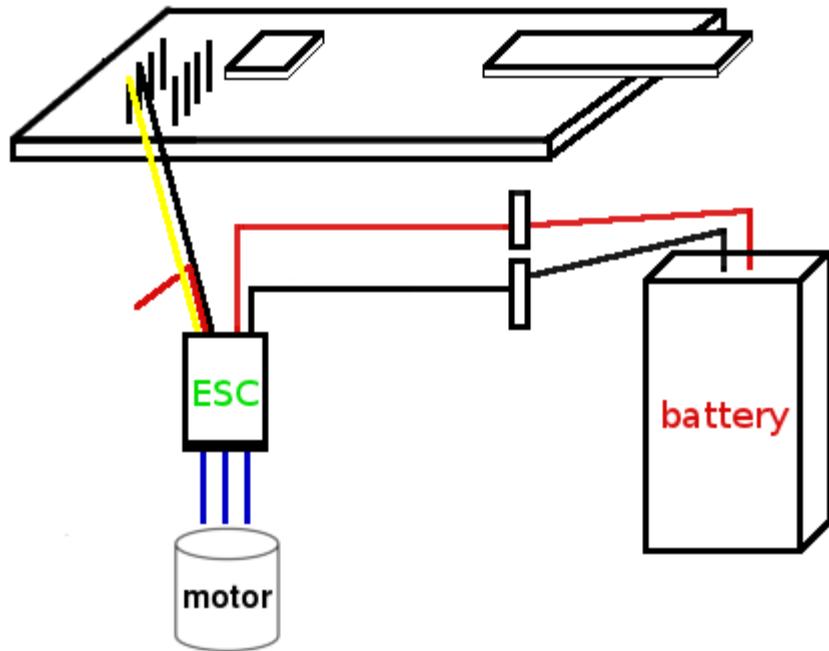


Figura 11. Esquema de conexión de un ESC.

Ya sabemos que los cables azules de la figura 10 son la señal trifásica que hacen girar el motor. También que los dos cables de la parte superior derecha del ESC van hacia la batería. ¿Pero qué pasa con los otros 3? Sirven para comunicar micro y ESCs. El cable rojo central se puede utilizar para alimentar el microcontrolador pero nosotros lo alimentaremos con una pila aparte por lo que quedará al aire, como se puede ver en la figura 10. Para entender los otros dos cables, tenemos que introducir primero el concepto de PWD que es el tipo de señal que el ESC recibe por esos dos cables.

5.1.6. PWD

La modulación por ancho de pulsos (también conocida como PWM, siglas en inglés de pulse-width modulation) es una potente técnica para controlar circuitos analógicos mediante una salida digital del microcontrolador. Consiste en modificar el *ciclo de trabajo* de una señal periódica (una senoidal o una cuadrada, por ejemplo) para transmitir información a través de un canal de comunicaciones. El *ciclo de trabajo* de una señal periódica es el ancho relativo de su parte positiva en relación con el período. Expresado matemáticamente:

$$D = \frac{\tau}{T}$$

D es el ciclo de trabajo

T es el tiempo en que la función es positiva (ancho del pulso)

T es el período de la función

Una imagen aclaratoria viene a continuación. La figura 11 muestra 5 señales PWM diferentes. La primera indica una señal de salida con un ciclo de trabajo del 0%. La señal se mantiene constante a 0 voltios siempre. La segunda señal trabaja al 25% por lo que un cuarto del tiempo la señal valdrá 5v. Las siguientes señales trabajan al 50%, 75% y 100% respectivamente siguiendo esa misma idea. Cuando, por ejemplo, el suministro es de 9V y el ciclo de trabajo es un 10%, la señal analógica resultante será de 0.9 V. Siguiendo esta lógica, en la figura 11, suponiendo que tenemos un suministro de 5V, las señales analogicas resultantes serían: a) 0v b) $5 \times 0.25 = 1.25$ c) $5 \times 0.5 = 2.5$ v c) $5 \times 0.75 = 3.75$ v d) $5 \times 1 = 5$ v

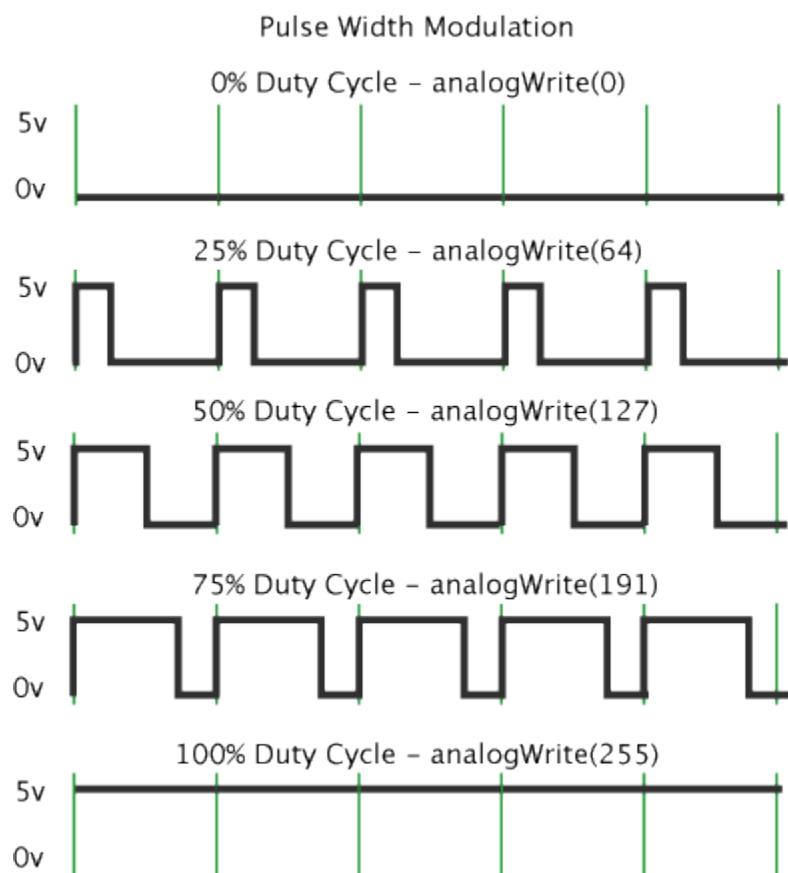


Figura 12. PWM.

Un concepto importante a tener en cuenta para que la técnica PWM funcione, es la frecuencia (medida en Hercios) con la que los cambios de señal descritos anteriormente se dan. Imaginemos que estamos controlando una lámpara mediante la técnica de PWM. Si mandamos una señal de 9v durante 5

segundos y después otra de 0V durante 5 segundos estaríamos ante un ciclo de trabajo del 50% pero está claro que nuestra bombilla estaba completamente apagada los primeros 5 segundos y apagada otros 5. Si lo que queremos es que la bombilla alumbe lo correspondiente a 4.5V (a la mitad de su capacidad), hay que conseguir que el cambio en la señal sea lo suficientemente rápido, respecto al tiempo de respuesta.

Los ESCs generalmente aceptan una señal PWM a 50 Hz (ciclos por segundo), cuya amplitud de pulso, varía de 1 ms a 2 ms. Cuando el pulso es de 1 ms a 50 Hz, el ESC responde apagando el motor conectado a su salida. Una amplitud de pulso de 1.5 ms hará que el ciclo de trabajo sea del 50% lo que moverá el motor a media velocidad. Para que el motor trabaje a su máxima velocidad, el pulso tendrá que ser de 2 ms.

Para el sistema cuadricóptero, hemos elegido el modelo “Turnigy AE-20A Brushless ESC” (figura 10).

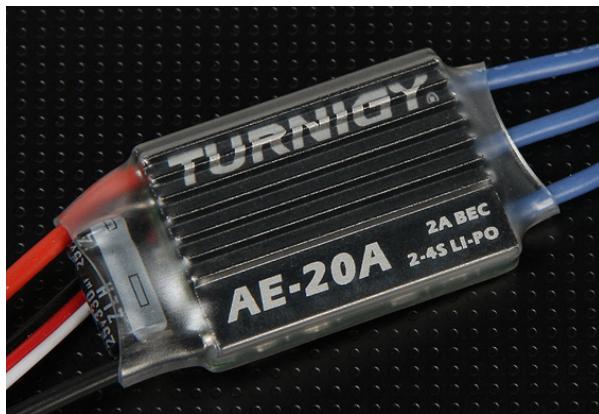


Figura 13. Turnigy AE-20A Brushless ESC.

Los ESCs de 10 amp se recomiendan para cuadricópteros por debajo de 1kg. Nosotros hemos elegido un ESC de 20A en previsión de que nuestro sistema pueda superar este peso en el futuro (sensores extra, cámara digital,...).

Especificaciones técnicas:

Output: Continuous 20A, burst 25A up to 10 seconds.

Input Voltage: 2-4 cells lithium battery or 5-12 cells NiMH battery.

BEC: Linear 2A @ 5V

Control Signal Transmission: Optically coupled system.

Max Speed:

2 Pole: 210,000rpm

6 Pole: 70,000rpm

12 Pole: 35,000rpm

Size: 50mm (L) * 26mm (W) * 12mm (H).

Weight: 19g.

5.1.7. Regla “watts per pound”

Existe una regla llamada “watts per pound” o “vatio por libra”, que determina que un multirotor necesita 50w/lbs (50 vatios por cada libra de peso) para poder desplazar su propio peso. Aunque se trata de una aproximación general, hemos comprobado que con los elementos que disponemos, efectivamente cumplimos esta regla.

$$\text{Vatio} = \text{Voltio} * \text{Corriente}$$

$$\text{watts} = \text{Voltios} * \text{Corriente} = 11.1 \text{ v} * 11 \text{ amps} = 122.1$$

$$1 \text{ pound} \rightarrow 0.45359237 \text{ kg}$$

$$x \text{ pounds} \rightarrow 1 \text{ Kg}$$

Nos da 2.2 pounds

$$50 \text{ w} \rightarrow 1 \text{ pound}$$

$$x \rightarrow 2.2 \text{ pounds}$$

Son necesarios 110w y tenemos 122w, por lo que cumplimos la regla. Hay que señalar que hemos hecho el cálculo suponiendo que el cuadricóptero pesa 1 kg, lo cual nos da un cierto margen de tranquilidad.

5.1.8. Microcontrolador

5.1.8.1. Hardware libre



Figura 14. Logo Open Hardware.

Open source hardware (OSHW) se refiere al conjunto de dispositivos diseñados con la misma filosofía que el software FOSS(FOSS - Free and open source software). *Open source hardware* es parte del movimiento **Open Source**, que se aplica como concepto. El término significa que la información sobre el hardware está accesible fácilmente. Tanto el diseño del hardware se refiere (planos mecánicos, esquemas, PCB, código HDL y disposición del circuito integrado) como a software que utiliza el hardware. Ambos son llevados a cabo con el enfoque de software libre abierto y gratuito. Hemos elegido una plataforma que sigue esta filosofía.

5.1.8.2. Arduino

Arduino[1] es una plataforma de prototipado electrónico basada en el principio del hardware y software libre. Al igual que Firefox fué en su día el caballo de batalla del software libre, Arduino también lo está siendo en el mundo del Hardware Open Source. El proyecto nace en Italia en 2005. El auge y desarrollo que está teniendo esta propuesta es reseñable. Importantes empresas a nivel mundial llevan ya tiempo interesándose y colaborando con el proyecto. Un buen ejemplo es el acuerdo de colaboración al que han firmado Google y Gran Bretaña[12] en mayo del presente año 2012, para la compra de kits de iniciación Arduino y la formación de profesores para enseñar utilizando Arduino en los colegios.

Se ha buscado desde el principio desarrollar el sistema con una plataforma que fuese de código abierto. Arduino tiene todos los ingredientes para ser apetecible para alguien que busque profundizar sus conocimientos en el mundo de la electrónica. Es por eso que existe una gran comunidad de artistas, diseñadores y aficionados creando proyectos constantemente con este microcontrolador, lo que hace del mismo una empresa muy viva. Vamos a ver qué es eso que lo hace tan especial.



Figura 15. Logo Arduino.

Quien mejor para presentar la plataforma que ellos mismos. Mediante estos dos puntos que vienen a continuación en nararaja, se explica en su web[1] que es Arduino en realidad.

5.1.8.2.1. ¿Qué es arduino realmente?

Arduino puede tomar información del entorno a través de sus pines de entrada de toda una gama de sensores y puede afectar aquello que le rodea controlando luces, motores y otros actuadores. El microcontrolador en la placa Arduino se programa mediante el [lenguaje de programación Arduino](#) (basado en [Wiring](#)) y el entorno de desarrollo Arduino (basado en [Processing](#)). Los proyectos hechos con Arduino pueden ejecutarse sin necesidad de conectar a un ordenador, si bien tienen la posibilidad de hacerlo y comunicar con diferentes tipos de software

5.2.8.2.2. ¿Por qué arduino?

Hay muchos otros microcontroladores y plataformas con microcontroladores disponibles para la computación física. Parallax Basic Stamp, BX-24 de Netmedia, Phidgets, Handyboard del MIT, y muchos otros ofrecen funcionalidades similares. Todas estas herramientas organizan el complicado trabajo de programar un microcontrolador en paquetes fáciles de usar. Arduino, además de simplificar el

proceso de trabajar con microcontroladores, ofrece algunas ventajas respecto a otros sistemas:

- **Asequible** - Las placas Arduino son más asequibles comparadas con otras plataformas de microcontroladores. La versión más cara de un módulo de Arduino puede ser montada a mano, e incluso ya montada cuesta bastante menos de 60€
- **Multi-Plataforma** - El software de Arduino funciona en los sistemas operativos Windows, Macintosh OSX y Linux. La mayoría de los entornos para microcontroladores están limitados a Windows.
- **Entorno de programación simple y directo** - El entorno de programación de Arduino es fácil de usar para principiantes y lo suficientemente flexible para los usuarios avanzados. Pensando en los profesores, Arduino está basado en el entorno de programación de Procesing con lo que el estudiante que aprenda a programar en este entorno se sentirá familiarizado con el entorno de desarrollo Arduino.
- **Software ampliable y de código abierto**- El software Arduino se publica bajo una licencia libre y preparado para ser ampliado por programadores experimentados. El lenguaje puede ampliarse a través de librerías de C++, y si se está interesado en profundizar en los detalles técnicos, se puede dar el salto a la programación en el lenguaje AVR C en el que está basado. De igual modo se puede añadir directamente código en AVR C en tus programas si así lo deseas.
- **Hardware ampliable y de Código abierto** - Arduino está basado en los microcontroladores ATMEGA168, ATMEGA328 y ATMEGA1280. Los planos de los módulos están publicados bajo licencia Creative Commons, por lo que diseñadores de circuitos con experiencia pueden hacer su propia versión del módulo, ampliéndolo u optimizándolo. Incluso usuarios relativamente inexpertos pueden construir la versión para placa de desarrollo para entender cómo funciona y ahorrar algo de dinero.

Las razones por las que se ha elegido Arduino para este proyecto casan muy bien con la descripción del punto anterior:

- Económico.
- Open Source, gran potencial.
- Curva de aprendizaje pequeña.
- En plena expansión.

El modelo concreto que se ha utilizado es el Arduino Uno de la figura 16.

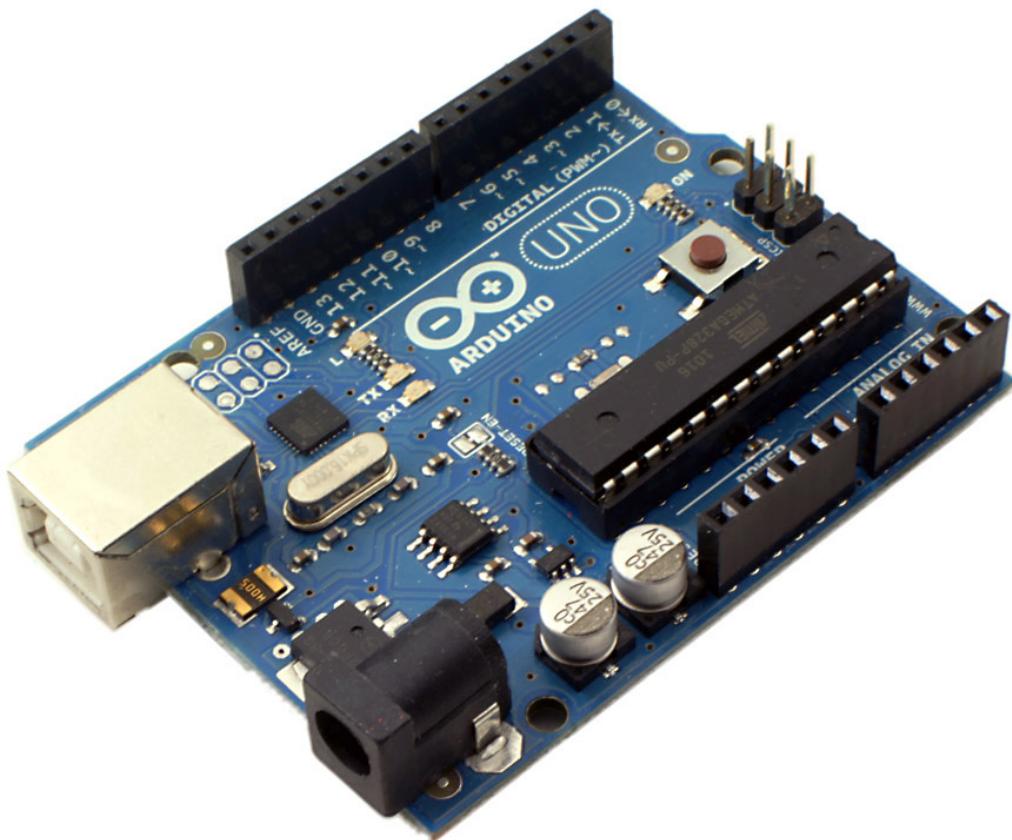


Figura 16. Arduino Uno.

Las especificaciones técnicas son:

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

Dispone de un procesador ATMEL ATmega328[13] a 16 MHz. Memoria Flash de 32 KB y 1 KB de memoria EEPROM (memoria que no se borra al cortar el suministro eléctrico). En cuanto a los pines, operan a 5 V. Tiene 14 digitales de los cuales 6 pueden actuar como salida de tipo PWM. A cuatro de ellos conectaremos nuestros ESCs. También hay 6 inputs analógicos a los cuales conectaremos los sensores. Por último, tenemos los pines 0(RX) y 1(TX) (Receive y Transmit respectivamente) que permiten una comunicación TTL en serie. Los utilizaremos para comunicarnos con el módulo Bluetooth. Hacia el final del apartado Hardware se adjunta un esquema con todas las conexiones. Antes en cambio, vamos a ver los dispositivos que van unidos al micro Arduino: El IMU y el módulo bluetooth.

5.1.9. Batería 9V

Se ha añadido una batería de 9V para alimentar el microcontrolador. Cumple con las especificaciones respecto a los límites de entrada de voltaje. (Recommended Input voltage: 7-12V). El micro tiene un regulador que rebaja este voltaje hasta 5V que es valor en el que opera.

5.1.10. IMU

IMU (Inertial Measurement Unit) o unidad de medición inercial es un dispositivo electrónico que mide e informa acerca de la velocidad, orientación y fuerzas gravitacionales de un aparato, usando una combinación de acelerómetros y giroscopos. La IMU funciona detectando la actual tasa de aceleración usando uno o más acelerómetros, y detecta los cambios en atributos rotacionales como son el yaw pitch y roll (figuras 15 y 16) mediante giroscopios.

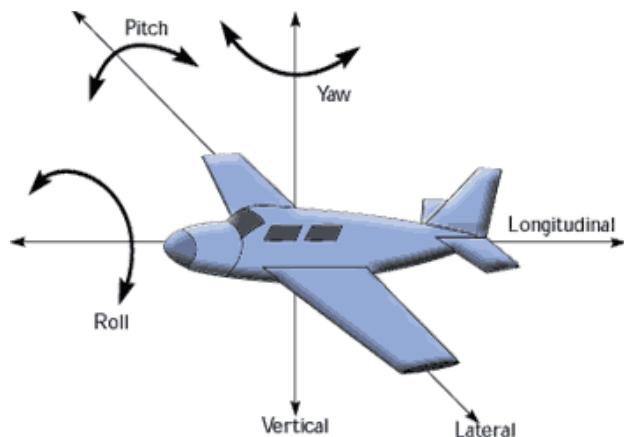


Figura 17. Yaw Pitch Roll.

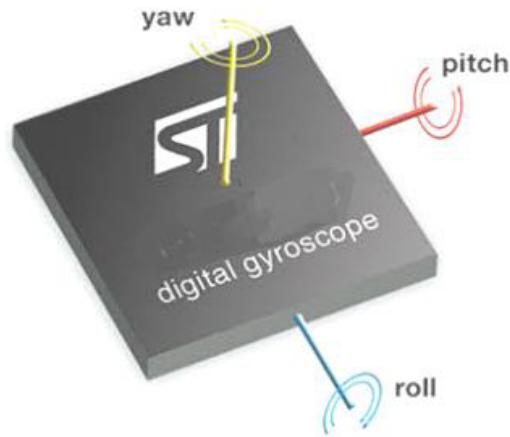


Figura 18. Yaw Pitch Roll.

En nuestro cuadricóptero se hará uso del **IMU Digital Combo Board - 6 Degrees of Freedom ITG3200/ADXL345** de sparkfun[14] que se muestra en la figura 17. Consta de un acelerómetro *ADXL345* y de un giroscopio *ITG3200* que se explican a continuación.

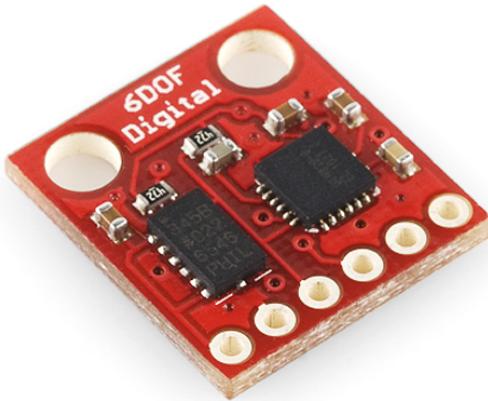


Figura 19. Sparkfun IMU Digital Combo Board - ITG3200/ADXL345.

5.1.11. Acelerómetro

El acelerómetro es un instrumento que mide aceleraciones. Lo que nos interesa es la aceleración de coordenadas (cambio de la velocidad del dispositivo en el espacio) pero esto presenta varios problemas. Para entender un acelerómetro, imaginemos una pelota de tenis “encerrada” en una especie de dado gigante. Este dado será nuestro acelerómetro, donde las paredes del mismo serán capaces de medir la fuerza que aplicamos sobre ellas. Si movemos el dado hacia la izquierda, la pelota chocará contra la pared izquierda. La fuerza que mediremos en esta pared será la manera de medir la aceleración. Si movemos el dado en diagonal, la pelota hará fuerza en dos paredes en vez de una, pero siguiendo la misma idea. Pero qué pasa si dejamos quieto nuestro dado en el suelo? Debido a la gravedad, la pared inferior medirá una fuerza, ¡y sin embargo no hay cambio de velocidad en el dado (acelerómetro)!!. Otro ejemplo claro es que un dado en caída gravitacional libre hacia el centro de la Tierra medirá un valor de cero, ya que, a pesar de que su velocidad es cada vez mayor, está en un marco de referencia en el que no tiene peso. El acelerómetro por tanto, mide todas las aceleraciones excepto las causadas por la gravedad.

El acelerómetro que incluye nuestro IMU es un ADXL345[15]. Un pequeño aparato de bajo consumo, que mide la aceleración de 3 ejes, y alta resolución (13-bits) que mide hasta ± 16 g. La salida digital del acelerómetro está disponible mediante interfaz SPI o I²C.

5.1.12. Giroscopio

El giroscopio es un dispositivo que mide la orientación, basándose en los principios del momento angular. Típicamente, consiste en un objeto que gira sobre sí mismo, y cuyo eje es libre de cambiar de orientación libremente. Cuando se somete el giroscopio a un momento de fuerza que tiende a cambiar la orientación del eje de rotación, el objeto que giraba sobre sí mismo cambiará de orientación para intentar seguir en su dirección «intuitiva».

Nuestra IMU incluye un *ITG3200* [16]. Vamos a ver como se comunica la IMU con el micro.

5.1.13. I²C

La IMU se comunica con arduino mediante un bus de comunicaciones en serie llamado I²C. Su nombre viene de Inter-Integrated Circuit (Circuitos Inter-Integrados). La velocidad de transmisión es de 100 kbits por segundo. Es un sistema muy usado para comunicar periféricos o (sensores en nuestro caso) con el microcontrolador en un sistema integrado. La principal característica de I²C es que utiliza dos líneas para transmitir la información. Una para datos y otra para la señal del reloj. Una tercera línea es necesaria para la masa. El nombre que reciben las líneas son: SDA (datos), SCL (reloj) y GND (ground - tierra).

En la parte posterior de la IMU podemos ver efectivamente las señales SDA y SCL y GND. Tendremos que tener cuidado de no darle más de 3.3 V para no quemarlo.

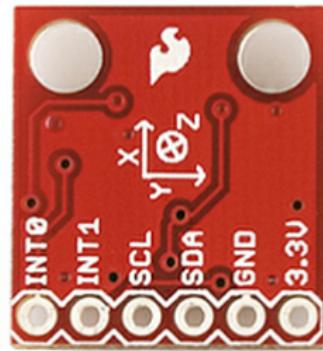


Figura 20. Parte posterior de la IMU ITG3200/ADXL345.

5.1.14. Módulo Bluetooth

En la primera fase del proyecto, se analizó si era mejor hacer la comunicación mediante WiFi o bluetooth. Pienso que ambas opciones son válidas, pero se decidió coger el camino del bluetooth por motivos económicos. Existe un módulo bluetooth con un coste de 32€ y no se encontró en ese momento ninguna opción Wi-Fi tan económica. Por comparación, existe un módulo acoplable[25] para arduino, que permite una comunicación WiFi. Su coste sin embargo, es de 71€, lo que hace esta opción, 39€ más cara que la opción bluetooth.

Utilizaremos por tanto un módulo bluetooth para realizar la comunicación con el smartphone. Haremos servir el “Bluetooth Modem - BlueSMiRF Silver”[15] de sparkfun de la figura X.



Figura 21. “Bluetooth Modem - BlueSMiRF Silver” de sparkfun.

BlueSMiRF Silver utiliza un modulo RN-42 que se comunica en serie (RX/TX). Funciona con rangos desde 2400 hasta 115200 bps (bits por segundo) de transmisión. Podemos alimentarlo con 3.3 V hasta 6V.

5.1.15. Otros

Además, han sido necesarios otros elementos que aun, siendo secundarios son imprescindibles como el proto-shield, un conmutador, el adaptador para conectar batería de 9 V con Arduino, el arnés de energía (cable que distribuye la energía desde la batería hasta los ESCs y un conmutador)

5.1.16. Peso

Con la siguiente configuración hemos calculado el peso total del sistema multirotor que quedaría de la siguiente manera: * Todos los pesos están en gramos.

Elemento	Peso unidad	Cantidad	Sub Total
Frame	195 gr	-	195
Motor	43 gr	x4	172
ESC	19 gr	x4	76
Batería	180 gr	-	180
Bateria arduino(9v)	47 gr	-	47
jack batería	8 gr	-	8
Microcontrolador	30 gr	-	30
Hélices	7gr	x4	28
IMU	8 gr	-	8
Bluetooth	8 gr	-	8

Arnés de energía	50 gr	-	50
Conmutador	8 gr		8
Total			810 gr

El sistema cuadricóptero nos da 810 gr. Para asegurarnos de que tiene potencia suficiente como para volar, se han hecho calculos con 1 Kg, dando un margen suficiente si se quisieran añadir otros elementos (una cámara, por ejemplo).

5.2. Hardware. Implementación.

En el presente capítulo se trata de mostrar como se ha llevado a cabo la implementación del sistema cuadricóptero diseñado en el apartado anterior. El primer paso ha sido dotar al microcontrolador de los sensores y el bluetooth. Se muestra un esquema de conexiones y fotos, una vez terminado el trabajo. El segundo paso ha sido montar uno por uno los distintos elementos del sistema. También se muestran fotos de todo este proceso.

5.2.1. Micro

Hemos utilizado un protoshield (figura 20) del mismo tamaño que el micro arduino. Soldaremos la IMU y el módulo bluetooth a este protoshield, que después acoplaremos al micro Arduino. De esta manera tendremos todos los dispositivos en una misma placa y en caso de necesidad, podríamos acoplarla a otro micro Arduino si el nuestro sufriera algún daño por cualquier circunstancia.

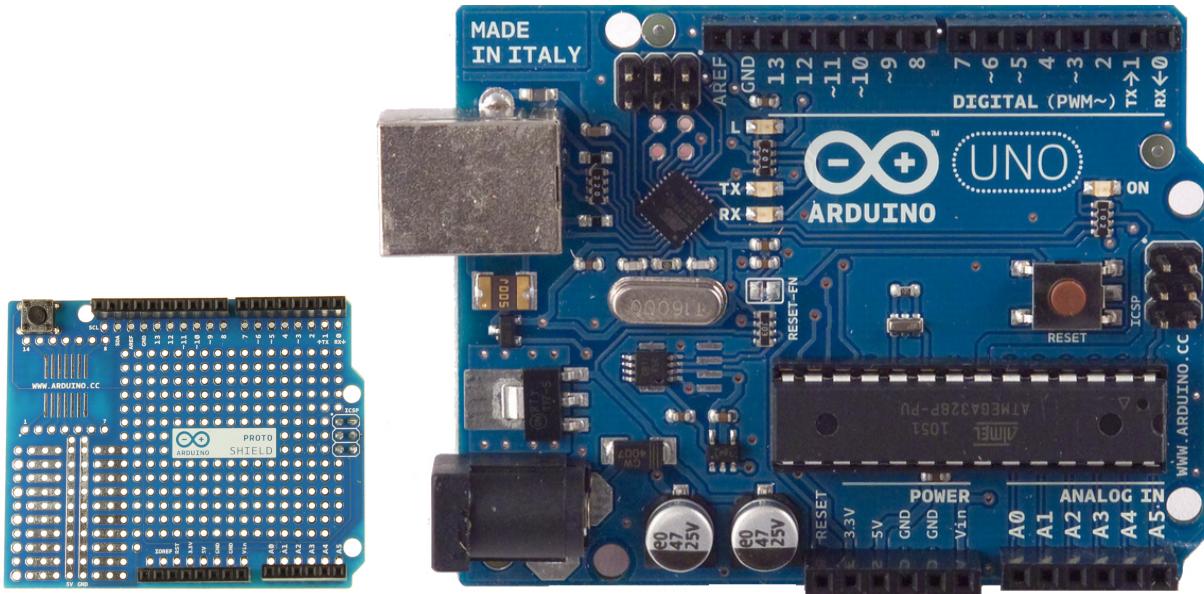


Figura 22. Izquierda: Protoshield Arduino. Derecha: Arduino UNO

La figura 21 muestra un esquema de conexiones de la IMU y el módulo Bluetooth BlueSMiRF con el microcontrolador.

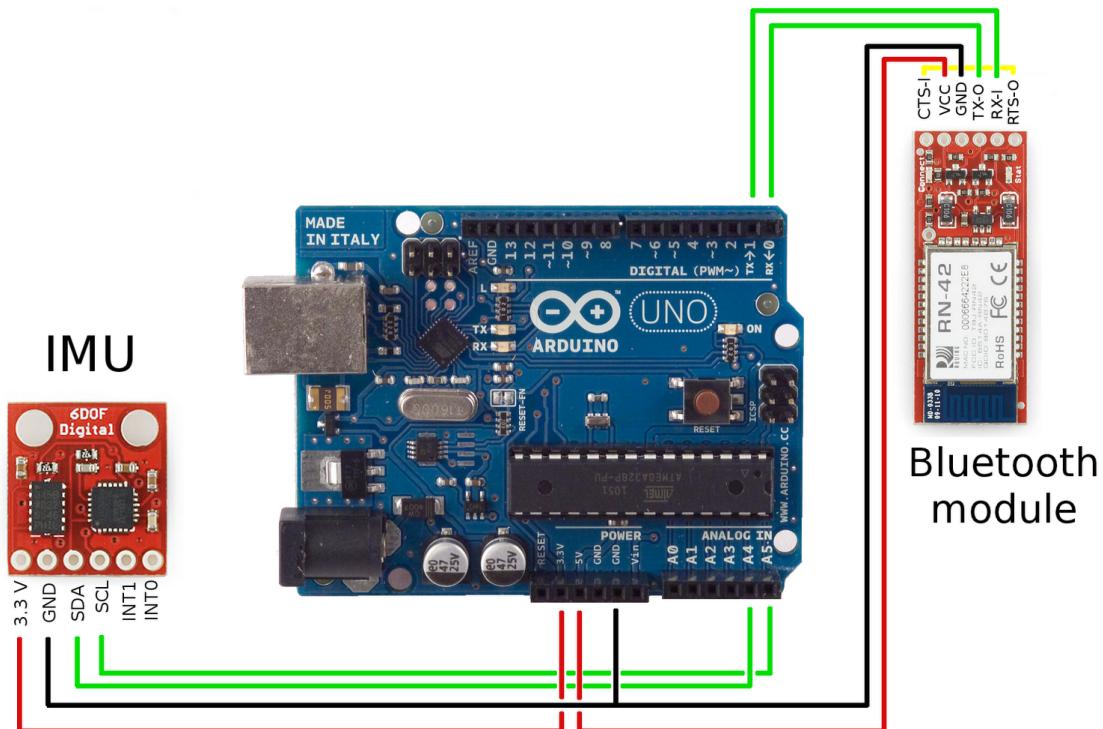


Figura 23. Esquema de conexión con IMU y Bluetooth.

En cuanto a la IMU, esta debe ser alimentada con 3.3 V porque así se especifica en su ficha técnica. Utilizaremos los pines de entrada analógicos A4 y A5 de Arduino para recibir las señales SDA (Datos) y SCL (Reloj) respectivamente.

El módulo bluetooth por su parte se alimentará mediante 5 V. La señal TX-O (transmit - output) del bluetooth irá conectado al pin RX (Receive) de Arduino. De manera similar, la señal RX-I(receive-input) del bluetooth irá conectada al pin TX (Transmit) de Arduino. En este esquema faltan dos cosas por añadir todavía.

La primera es el **conmutador** que hemos tenido que añadir para dar o dejar de dar corriente al módulo bluetooth. La razón es evitar problemas a la hora de descargar el código de programación al micro. Esto lo haremos conectando directamente un cable, entre el PC y Arduino, que crea una comunicación en serie. Si en el momento de descargar el código, el bluetooth está también enviando datos en serie al micro, no podremos descargar el código correctamente. La solución que se ha buscado es por tanto (1) desactivar la alimentación del bluetooth (conmutador OFF), (2) descargar el código al micro Arduino desde nuestro ordenador y por último, (3) volver a alimentar el bluetooth (conmutador ON). La figura 22 muestra el esquema tras añadir el conmutador.

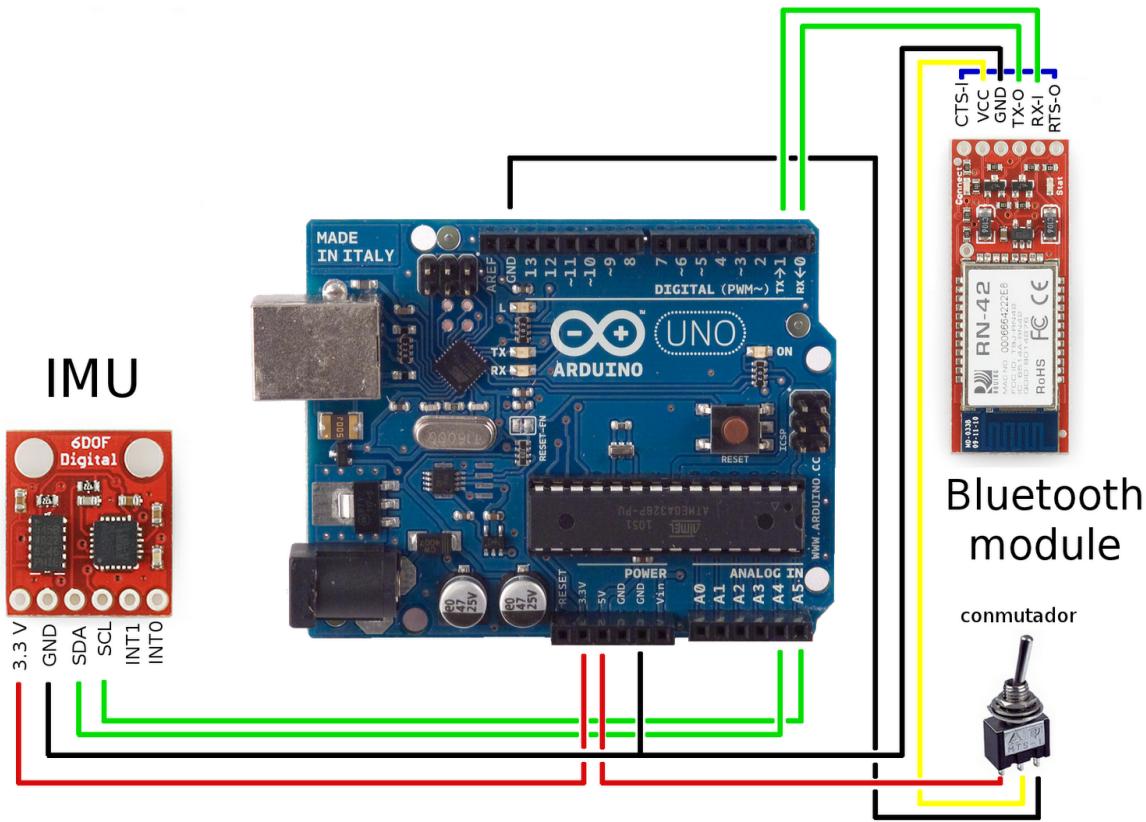


Figura 24. Esquema de conexión con conmutador.

El segundo elemento que nos quedaba por añadir a la placa son los cables que salen de los ESCs. Serán 4 (uno por motor, mas sus respectivas masa o tierra) y los conectaremos a los pines de salidas digitales. Habremos de tener cuidado a la hora de elegir los pins, ya que el modelo Arduino Uno[16], no permite a todos los pines digitales actuar como salidas PWM. La documentación de Arduino lo especifica sin dar lugar a errores:

PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the [analogWrite\(\)](#) function.

Vamos a usar los pines 6, 9, 10 y 11. El esquema definitivo del micro se muestra en la figura 23.

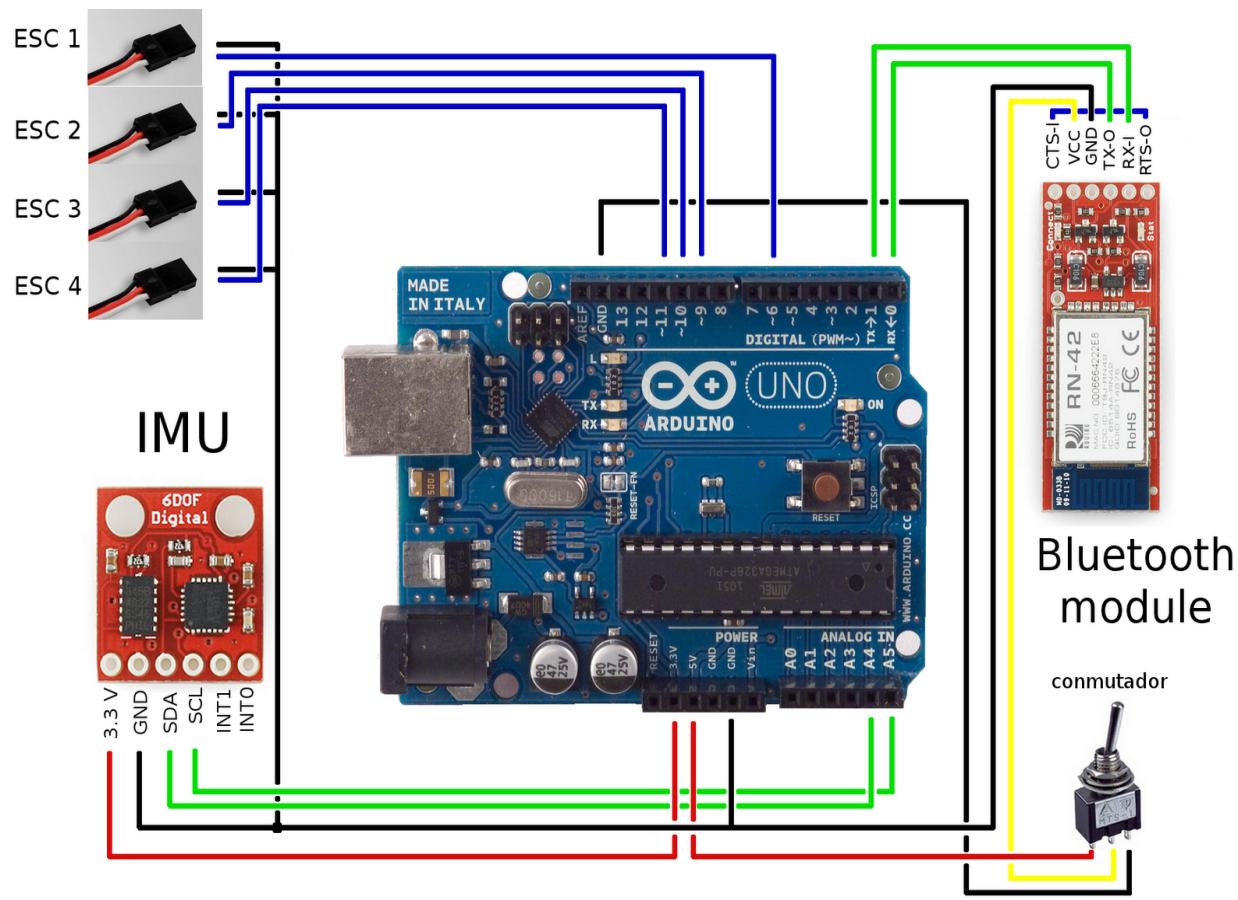


Figura 25. Esquema de conexión definitivo.

Hasta aquí el proceso de montaje del micro. Vamos a ver algunas fotos del resultado:

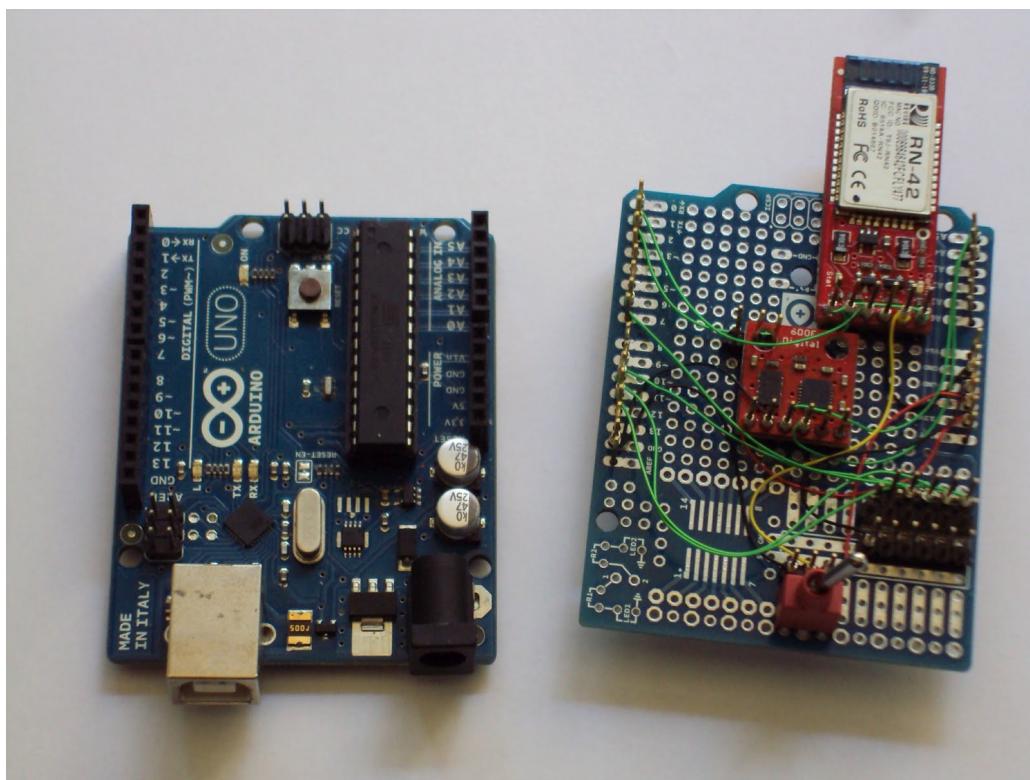


Figura 26. arduino Uno y protoshield acabado.

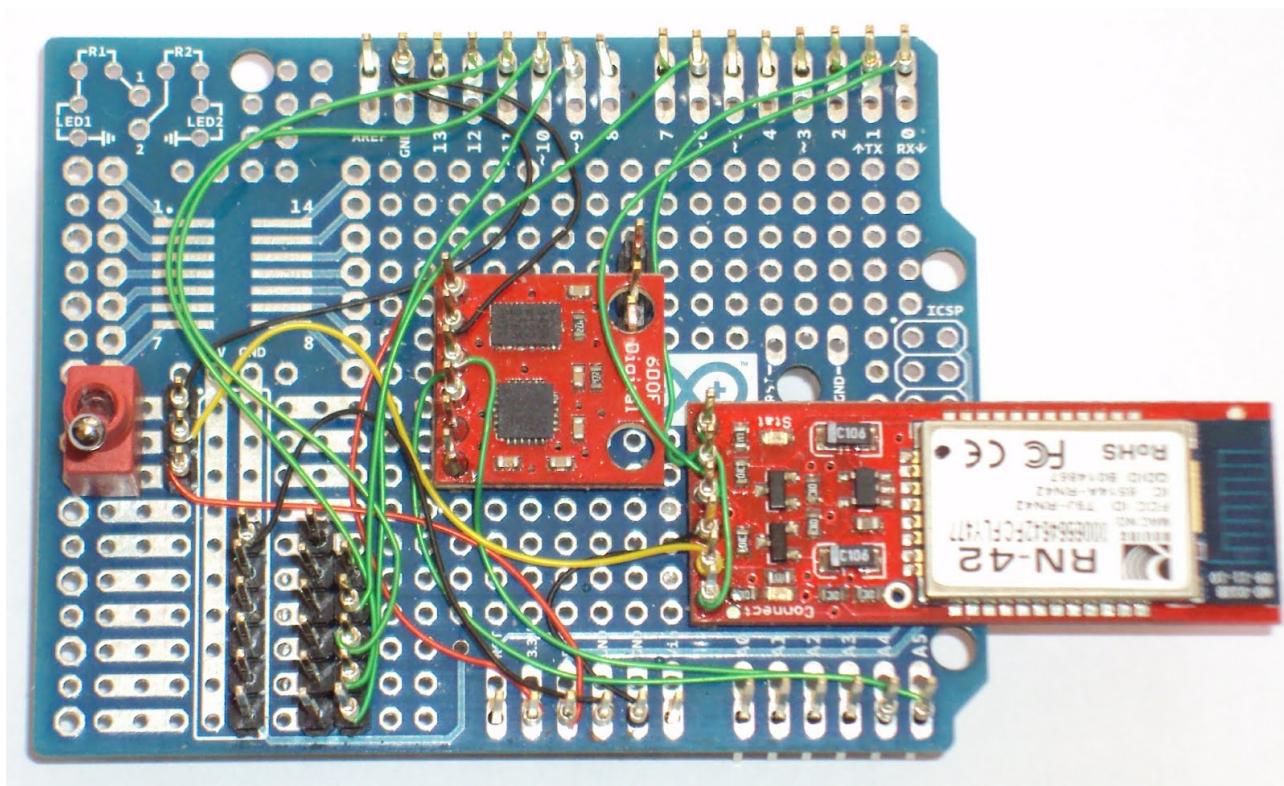


Figura 27. Protoshield acabado.

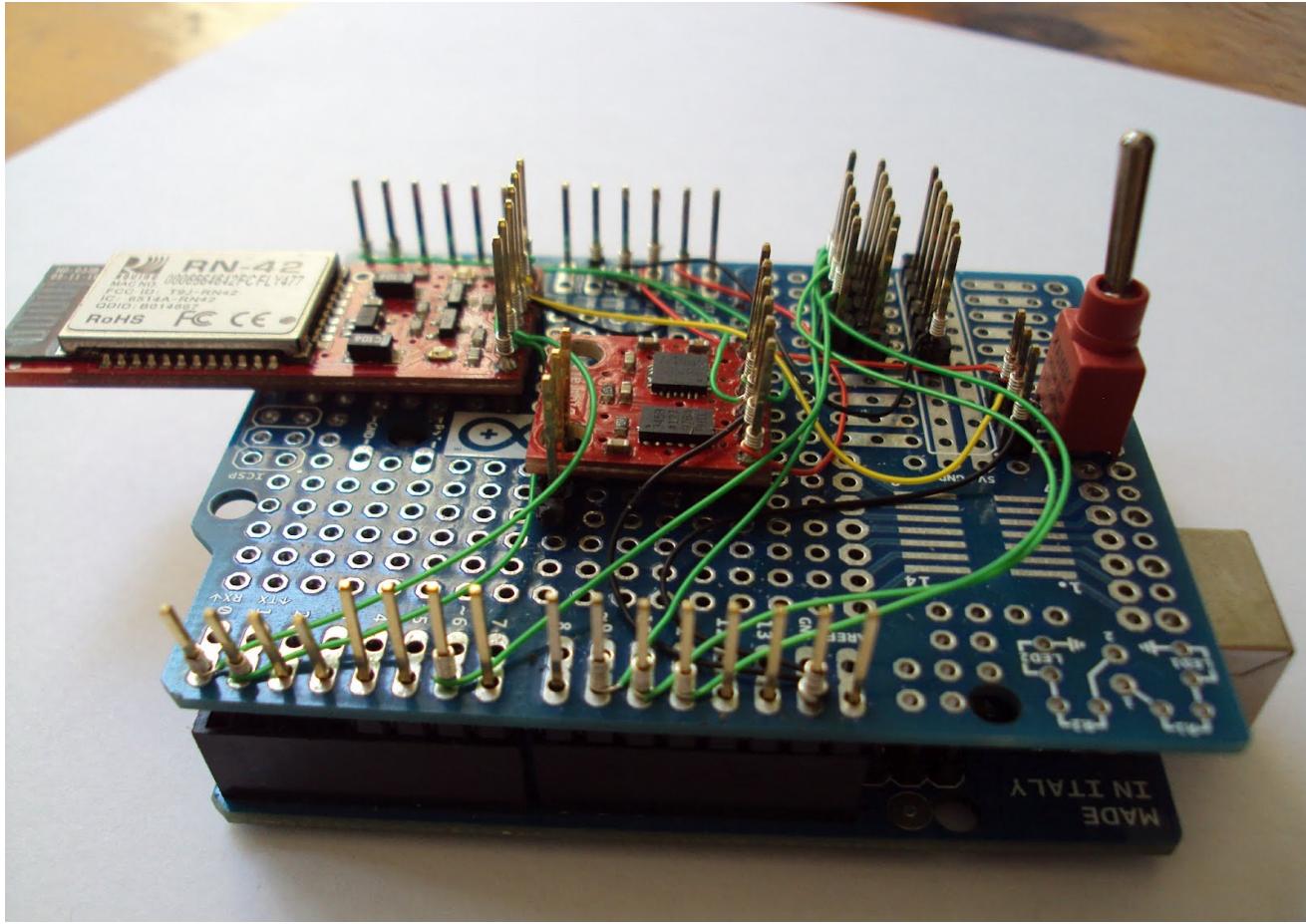


Figura 28. Protoshield sobre Arduino Uno.

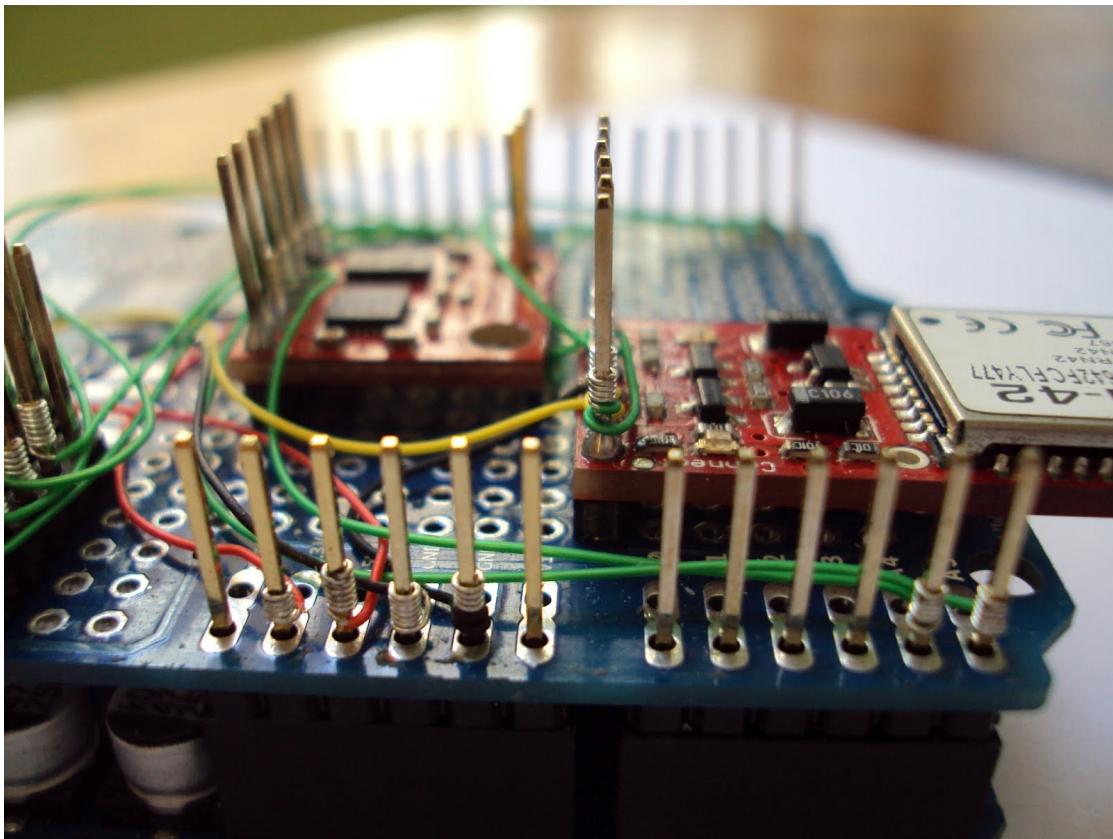


Figura 29. Protoshield sobre Arduino Uno.

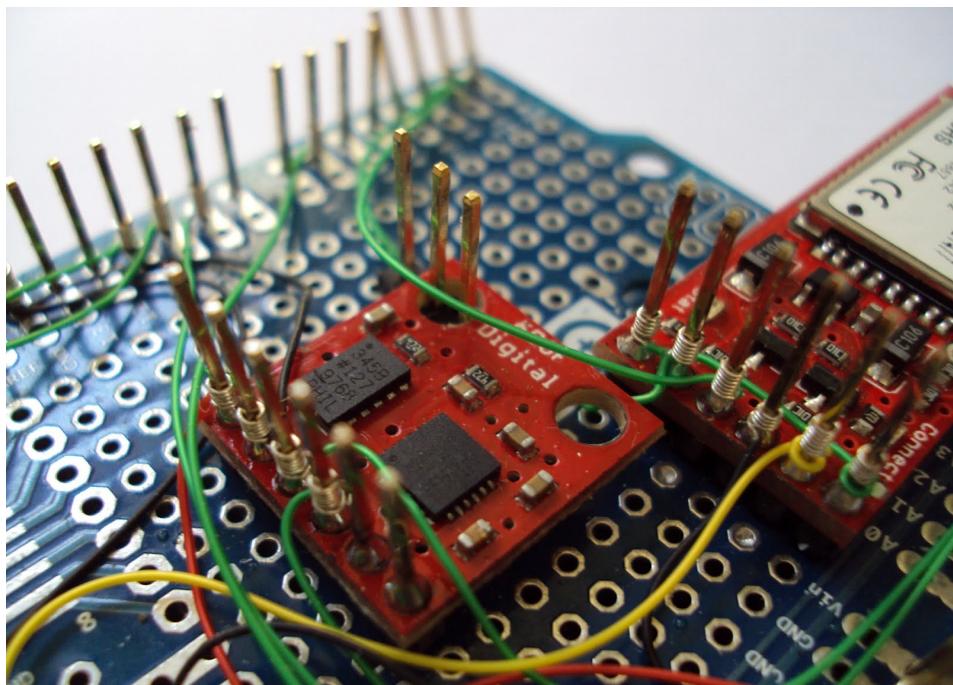


Figura 30. IMU y Bluetooth en la placa Protoshield.

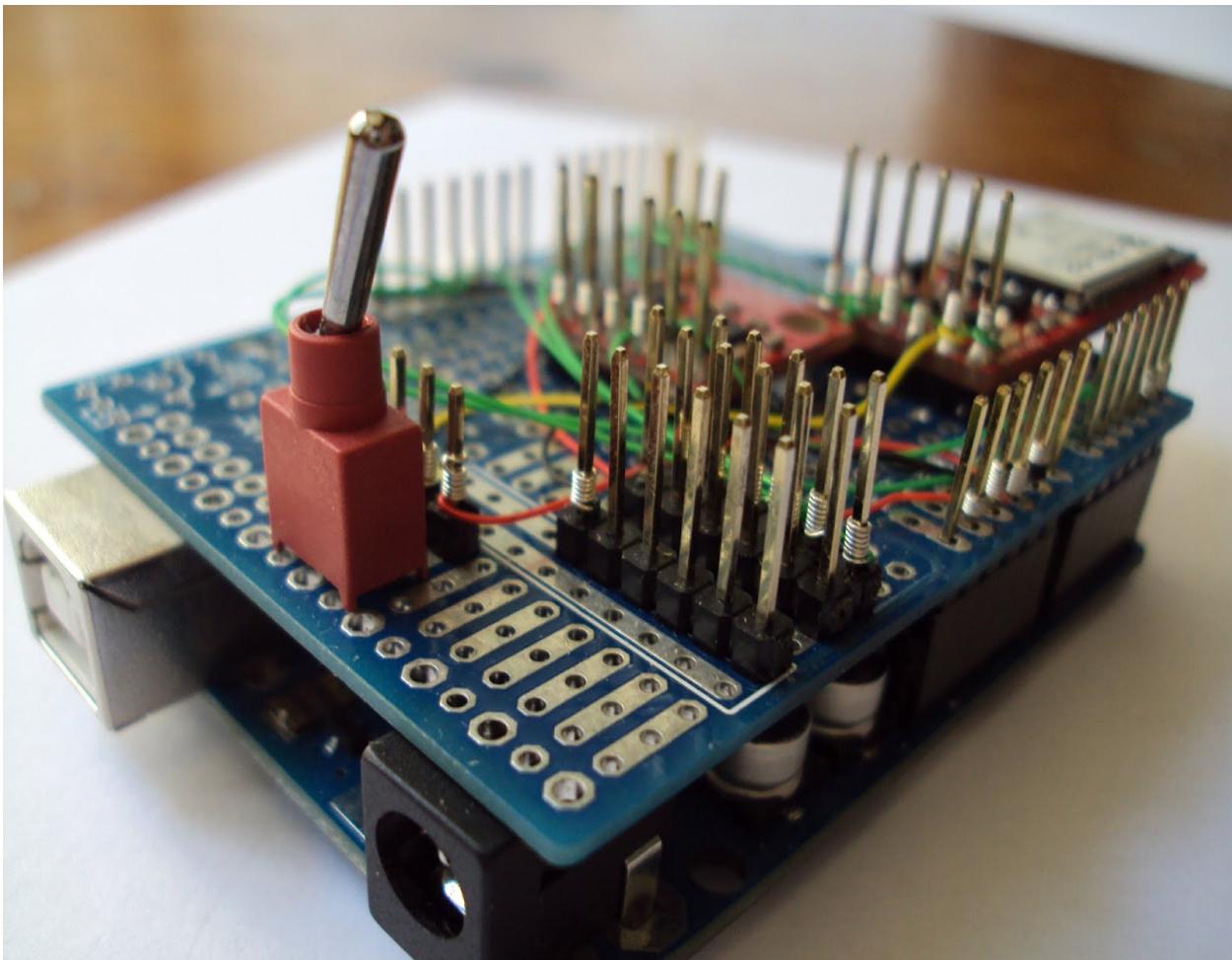


Figura 31. Protoshield acoplado al micro.

Vamos a ver ahora el proceso de montaje paso a paso del resto del sistema multirotor. En primer lugar montaremos el frame y a partir de ahí, iremos añadiendo los elementos que se seleccionaron en el capítulo anterior.

Los motores (con sus respectivas hélices) van anclados a cada extremo del frame.



Figura 32. Motor brushless con hélice, anclado.

Hemos colocado los ESCs a media altura de cada brazo de la estructura (figura 1). Por un lado conectados a los motores, por otro a la batería y por último al microcontrolador (figura 4).

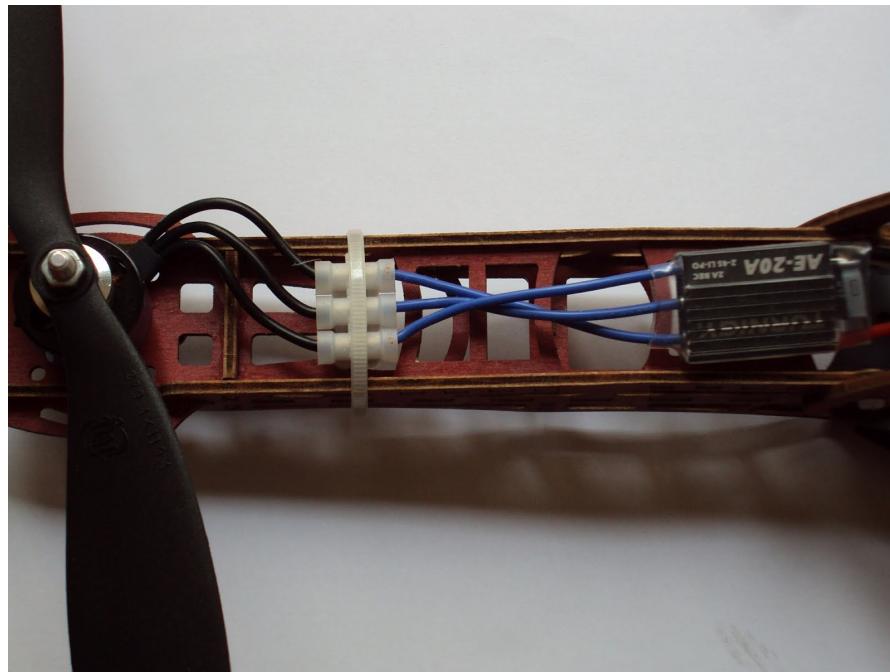


Figura 33. Conexión entre motor y ESC.

Hemos puesto los cables que alimentan los ESCs en medio de las dos estructuras de la base del frame. La figura 5 muestra el cableado del interior de las dos bases.

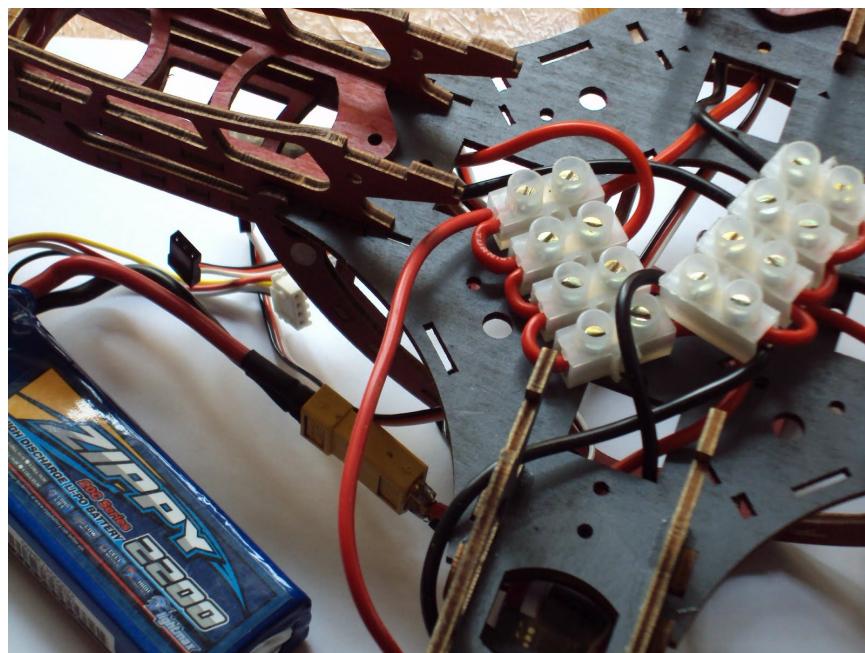


Figura 34. Cableado eléctrico de la parte inferior interior.

El micro y la pila que lo alimenta lo pondremos en el centro.

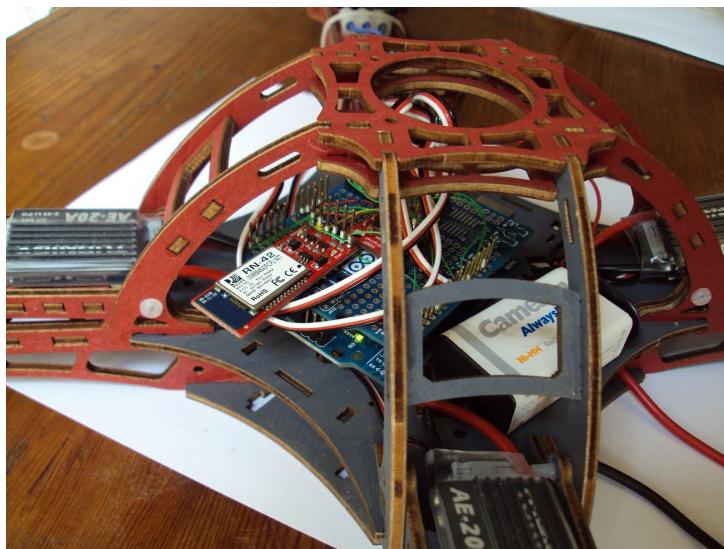


Figura 35. Arduino instalado dentro del frame.

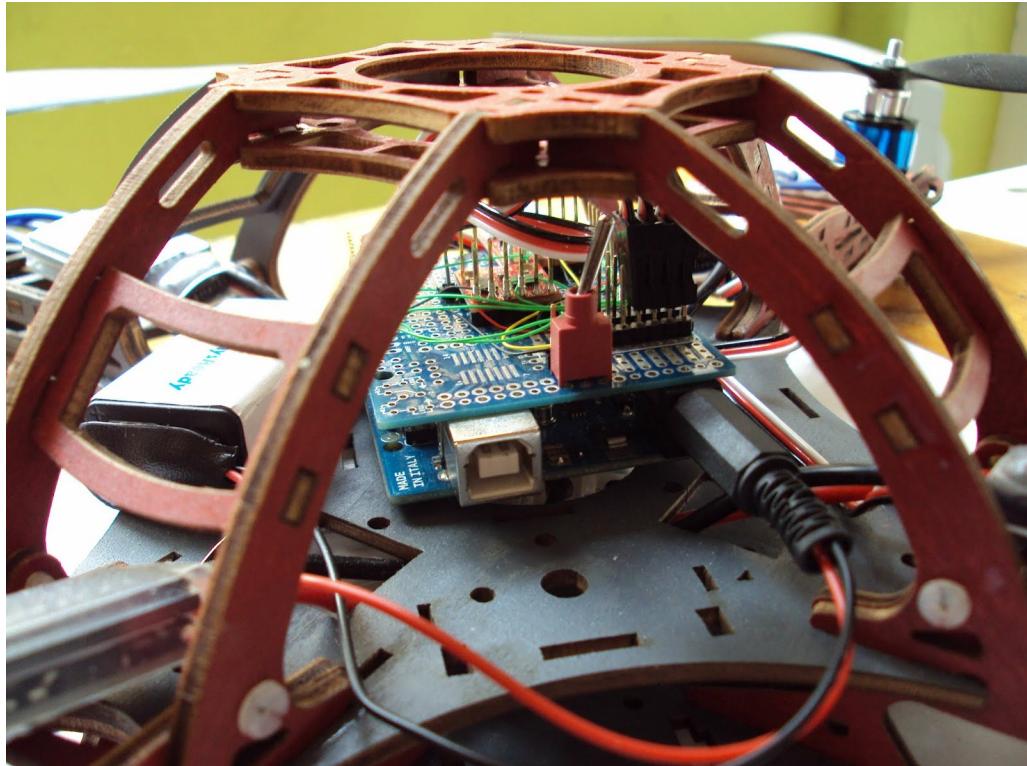


Figura 36. Arduino instalado dentro del frame.

La batería la hemos colocado en la parte inferior del sistema, con un sistema de velcro, para poder cambiarla fácilmente. (Habrá que extraerla para recargarla).

6. Software. Diseño

Todo lo que tiene que ver con la parte lógica del proyecto, se ha dividido en dos capítulos. En este primero, llamado “Software. Diseño” se intenta explicar las razones por las cuales se han elegido determinadas herramientas y el uso que se hará de ellas. En el siguiente capítulo, llamado “Software. Implementación”, explicaremos el código que se ha generado.

Antes de entrar en el diseño, hay que separar claramente entre dos cuestiones principales. La primera es la referente a la **comunicación** que hay que montar entre el microcontrolador arduino y un smartphone Android. Esta comunicación ha de ser posible en ambos sentidos.

Por otra parte, está el tema de la estabilización del sistema. Se ha realizado un trabajo de análisis sobre este problema. Se este capítulo se sientan las bases del control PID para un trabajo de futuro ([capítulo ?](#)). También se analizan diferentes proyectos donde se ha dado solución a este problema de alguna manera u otra. Se verá, que habría de ser posible hacer una reutilización de una parte del código amoldando la solución a nuestro hardware. Pero vamos a ver primero el diseño que se ha realizado en la parte de la comunicación.

6.1. Android

Un resumen de explicando qué es Android sería el siguiente:

“Android es un [sistema operativo móvil](#) basado en [Linux](#), que junto con aplicaciones [middleware](#) está enfocado para ser utilizado en [dispositivos móviles](#) como [teléfonos inteligentes](#), [tabletas](#) y otros dispositivos. Es desarrollado por la [Open Handset Alliance](#), la cual es liderada por [Google](#).

Las unidades vendidas de teléfonos inteligentes con Android se ubican en el primer puesto en los [Estados Unidos](#), en el segundo y tercer trimestres de [2010](#), con una cuota de mercado de 43,6% en el tercer trimestre. A nivel mundial alcanzó una cuota de mercado del 50,9% durante el cuarto trimestre de 2011, más del doble que el segundo sistema operativo (iOS de iPhone) con más cuota.

Tiene una gran comunidad de desarrolladores escribiendo aplicaciones para extender la funcionalidad de los dispositivos. A la fecha, se han sobrepasado las 600.000 aplicaciones (de las cuales, dos tercios son gratuitas) disponibles para la tienda de aplicaciones oficial de Android: [Google Play](#). Existe la posibilidad de obtener software externamente. Los programas están escritos en el [lenguaje de programación Java](#).

[...] Google liberó la mayoría del código de Android bajo la [licencia Apache](#), una licencia [libre](#) y de [código abierto](#).

La estructura del sistema operativo Android se compone de aplicaciones que se ejecutan en un [framework](#) Java de aplicaciones orientadas a objetos sobre el núcleo de las [bibliotecas](#) de Java en una máquina virtual [Dalvik](#) con [compilación en tiempo de ejecución](#). Las bibliotecas escritas en [lenguaje C](#) incluyen un administrador de [interfaz gráfica](#) (surface manager), un framework [OpenCore](#), una [base de datos](#) relacional [SQLite](#), una Interfaz de programación de [API](#) gráfica [OpenGL ES 2.0 3D](#), un motor de renderizado [WebKit](#), un motor gráfico [SGL](#), [SSL](#) y una [biblioteca estándar de C Bionic](#). El sistema operativo está compuesto por 12 millones de líneas de código, incluyendo [XML](#), C, Java y [C++](#).



Figura 37. Logo Android.

Utilizaremos el sistema operativo Android, entre otras opciones disponibles(iOS, Blackberry), por varias razones. En primer lugar por tratarse de la plataforma que está sufriendo una mayor expansión en los últimos años.

En segundo lugar, el hecho de que yo personalmente dispusiera de un teléfono Android hacia más práctico el desarrollo. De esa manera no ha sido necesario el uso de ningún simulador. A la vez que he ido desarrollando el proyecto, también he cursado una asignatura en la que se trabajaba con esta plataforma. Estas horas han sido bien invertidas, porque ese conocimiento se ha aplicado en este proyecto.

En tercer lugar, las herramientas de desarrollo que existen cuando se programa para android son muy amplias y la curva de aprendizaje no es demasiado elevada. Por ejemplo, existe la posibilidad de crear aplicaciones usando el IDE ([Integrated development environment](#) o Entorno de Desarrollo) Eclipse con el cual ya estaba familiarizado.

En último lugar, Android se programa en JAVA [18], lo cual viene muy en concordancia con mi background hasta ahora.

6.2. Arduino

En la sección del microcontrolador ya hemos explicado qué es Arduino y el porqué de su utilización. Pero no hemos visto como se implementa realmente un programa.

El modo de trabajo suele ser el siguiente: Se conecta el micro al PC mediante un cable de serie. Se abre la IDE (entorno de desarrollo) Arduino. Se carga el programa que se desee y se descarga al micro. Despues, se desconecta el cable. En el momento en que se alimente arduino de alguna otra manera,(en nuestro caso, con una pila de 9V), se ejecutará el programa cargado.

Mediante un par de ejemplos simples vamos a mostrar como se pueden leer datos de un sensor, o enviar consignas a un periférico a la vez que mostrar el IDE de Arduino.



Figura 38. Logo Arduino.

Algunos ejemplos

La figura X muestra un código sencillo con el que se ilumina y apaga un LED intermitentemente. En la parte superior del código, vemos como se han definido las variables. Despues viene la función **setup()** que se ejecutará una sola vez. Aquí es donde se hacen las configuraciones iniciales. En este caso definimos el pin “led” (al cual hemos asignado el valor 13) como OUTPUT (salida). Despues viene la función **loop()** que se ejecutará automáticamente una y otra vez mientras el micro esté alimentado. Lo que hace este sencillo ejemplo es simplemente escribir un valor “HIGH” en el pin 13. Como hemos definido el pin 13 como de salida, HIGH será una constante que vale 5V.

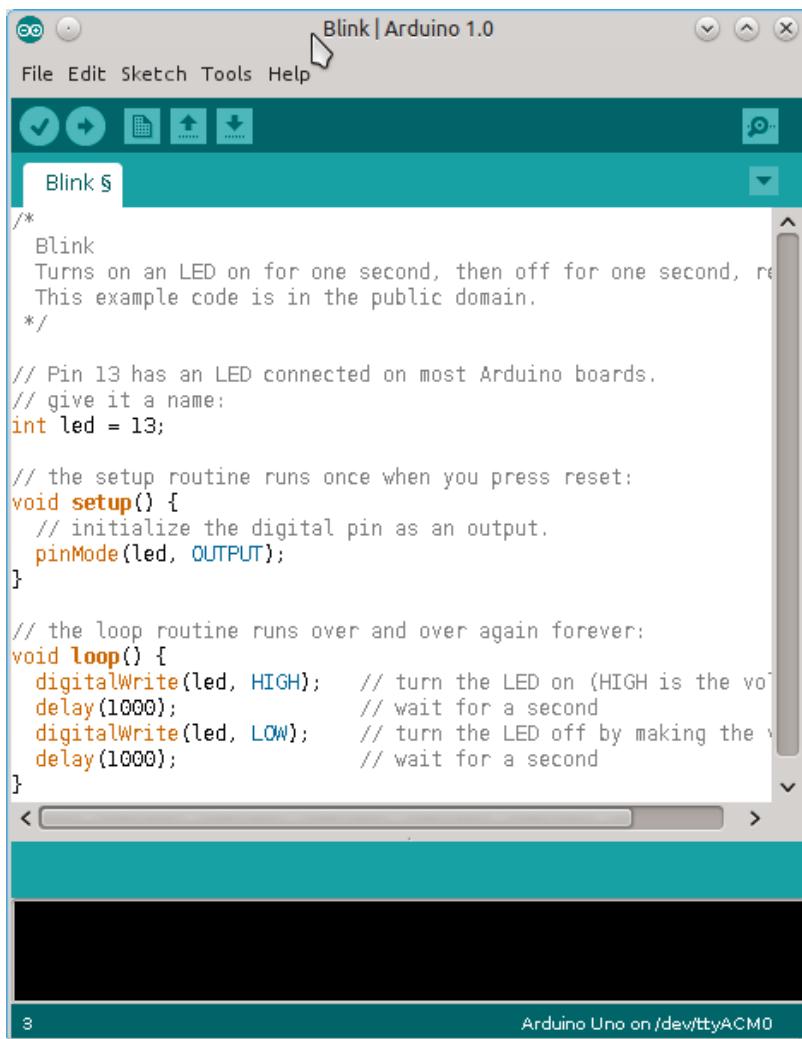
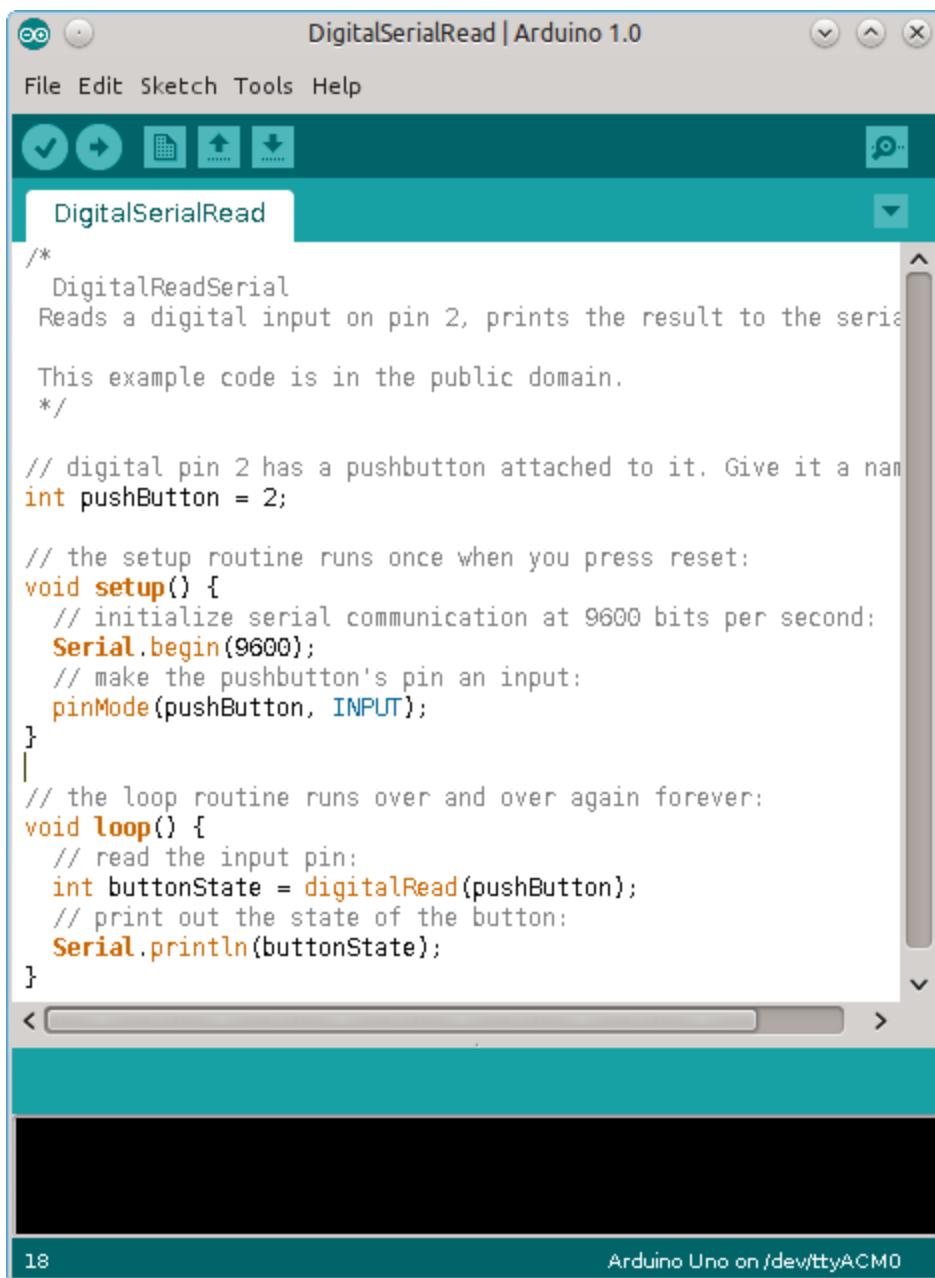


Figura 39. IDE Arduino.

El IDE de Arduino es bastante sencillo e intuitivo y no es objetivo de este documento explicar su funcionamiento al completo pero sí daremos un par de pinceladas. Hay un ícono de especial interés, en la parte superior derecha llamado “Serial Monitor” que abre una ventana nueva. Todos los datos que llegan por comunicación serie desde el micro, se irán mostrando en esta vista. Es decir, que si conectamos un sensor al micro y hacemos que envié sus lecturas a través del puerto serie al pc podremos ver estos datos en la ventana de la figura X. En las líneas del próximo ejemplo, se envía en serie (`Serial.println()`) el estado de un botón, que no es más que la entrada del pin 2 definida como un INPUT del sistema. En la función `setup()` se establece que la comunicación en serie se hará a 9600 baudios (`Serial.begin(9600)`). El baudio [19] es una unidad de medida usada en telecomunicaciones, que representa la cantidad de veces que cambia el estado de una señal en un periodo de tiempo, tanto para señales digitales como para señales analógicas. Es importante resaltar que no se debe confundir el baud rate o velocidad en baudios con el bit.

rate o velocidad en bits por segundo, ya que cada evento de señalización (símbolo) transmitido puede transportar uno o más bits. Sólo cuando cada evento de señalización (símbolo) transporta un solo bit coinciden la velocidad de transmisión de datos baudios y en bits por segundo.



The screenshot shows the Arduino IDE interface with the title bar "DigitalSerialRead | Arduino 1.0". The menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with icons for file operations. The main window displays the "DigitalSerialRead" sketch code. The code reads a digital input on pin 2 and prints the result to the serial monitor. It includes comments explaining the setup and loop routines. The bottom status bar shows "18" on the left and "Arduino Uno on /dev/ttyACM0" on the right.

```
/*
  DigitalReadSerial
  Reads a digital input on pin 2, prints the result to the serial monitor.

  This example code is in the public domain.
*/

// digital pin 2 has a pushbutton attached to it. Give it a name:
int pushButton = 2;

// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
  // make the pushbutton's pin an input:
  pinMode(pushButton, INPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input pin:
  int buttonState = digitalRead(pushButton);
  // print out the state of the button:
  Serial.println(buttonState);
}
```

Figura 40. Ejemplo Arduino.

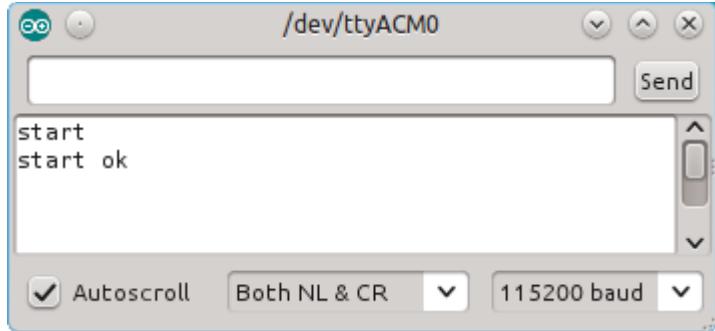


Figura 41. Serial Monitor.

Después de haber mostrado estos dos ejemplos, debería de haber quedado clara la filosofía que sigue Arduino: Definir pines digitales o analógicos como de entrada o de salida, e ir leyendo o escribiendo en ellos.

7. Software. Implementación

El presente capítulo se divide en dos partes. En la primera se explica que herramienta se ha utilizado para hacer las lecturas de la IMU y poder calcular los ángulos de Euler del sistema cuadricóptero en cada momento. En la segunda, se explicará cómo se ha logrado montar la comunicación entre Android y Arduino.

7.1. Lectura de la IMU

7.1.1. Filtro de Kalman

El filtro de Kalman es un algoritmo que utiliza una serie de medidas recogidas de una señal en el tiempo que contienen ruido e imperfecciones. Con estos datos, trata de producir una nueva variable que de una información más precisa que los datos medidos inicialmente. La idea es eliminar el ruido, produciendo una señal más limpia. El proceso consta de dos etapas. En una primera fase de predicción, se

hace una estimación del estado actual de la variable y de su nivel de incertidumbre. Una vez llega la siguiente medición (que estará corrupta, debido a un cierto error y/o ruido), se hace una estimación del valor correspondiente utilizando una media ponderada, dando mayor peso, a los valores cercanos. Al tratarse de un algoritmo recursivo, se puede ejecutar en tiempo real, siendo necesario solamente, la medición actual y el estado anterior. No hace falta más información que esa. El ejemplo de la figura X muestra en rojo el resultado del filtro kalman.

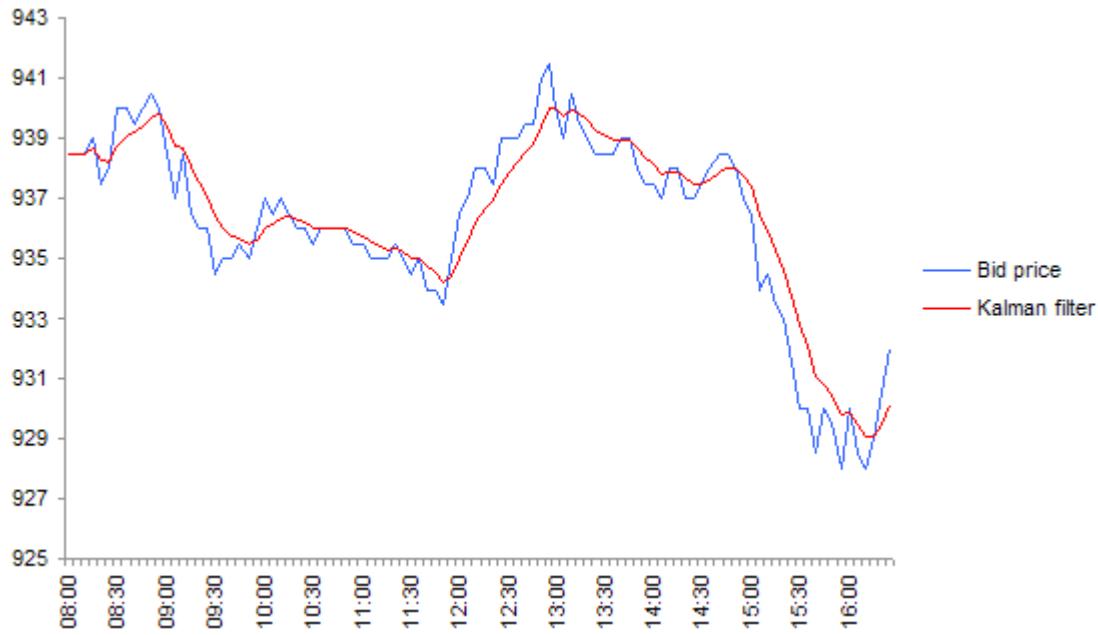


Figura 42. Filtro de Kalman.

En el caso del cuadricóptero, es muy interesante el uso de este tipo de filtros, para evitar que las perturbaciones en las lecturas de los sensores debidas a la vibración, afecten a la estabilidad del aparato. He encontrado en la web de starlino.com una implementación de un algoritmo simplificado del filtro de Kalman para una IMU de 5 grados de libertad [22]. (5 DOF). Voy a utilizar una librería para Arduino que está basada en el código anterior, pero que ha sido modificada para trabajar con el acelerómetro ADXL345 y el giroscopio ITG3200 que trae nuestra IMU.

Es una librería de código abierto con Licencia GNU 3, para Arduino, llamada “Free Six IMU” [20], diseñada por el Italiano Fabio Varesano [21]. En una primera implementación, la IMU simplemente envía sus lecturas. Es la segunda implementación la que nos interesa; La lógica del algoritmo está dentro del código arduino, con lo que es posible saber la orientación de la IMU (y por ende, del multirotor) en tiempo real. Hemos copiado la librería a nuestro PC para hacer uso de ella. Hemos ejecutado el siguiente

código de prueba que envía los ángulos de Euler que calcula el micro, por el puerto serie, como se muestra en la figura X.

```
/*
IMU Test Euler angles

*/
#include <FreeSixIMU.h>
#include <FIMU_ADXL345.h>
#include <FIMU_ITG3200.h>
#include <Wire.h>

float angles[3]; // yaw pitch roll

// Set the FreeSixIMU object
FreeSixIMU sixDOF = FreeSixIMU();

void setup() {
    Serial.write("setup begin.");
    Serial.begin(115200);
    Wire.begin();
    delay(5);
    sixDOF.init(); //begin the IMU
    delay(5);
    Serial.write("setup end.");
}

void loop() {
    Serial.write("Euler Angles: ");
    sixDOF.getEuler(angles);
    Serial.print(angles[0]);
    Serial.print(" | ");
    Serial.print(angles[1]);
    Serial.print(" | ");
    Serial.println(angles[2]);
    delay(100);
}
```

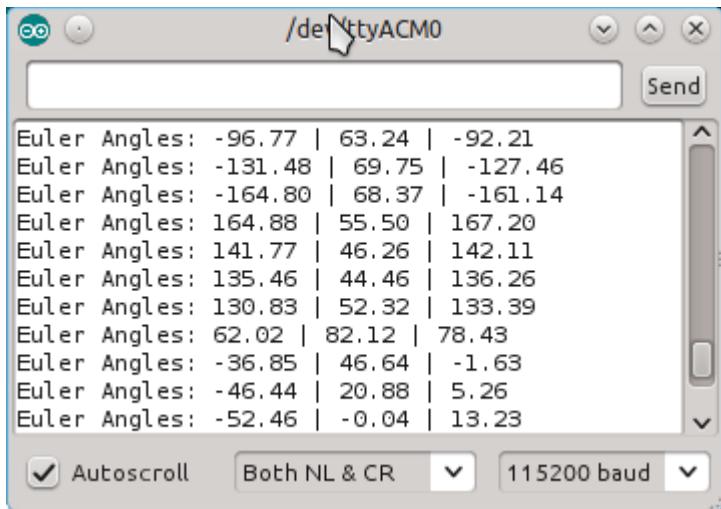


Figura 43. Ángulos de Euler.

7.2. Comunicación Android - Arduino

Desde el momento en el que se tomó la decisión de comunicar ambos dispositivos mediante bluetooth, se ha ido buscando una solución que fuera aplicable a este contexto. Vamos a utilizar una herramienta llamada Amario.

7.2.1. Amario

Amarino[24] es un toolkit os erie de librerías, para conectar móviles que funcionan bajo el sistema operativo Android, con un microcontrolador Arduino, mediante bluetooth.

Amarino is a toolkit to connect Android-driven mobile devices with Arduino microcontrollers via Bluetooth

Amarino consiste en una aplicación para Android y una librería para Arduino, que hacen posible la comunicación vía Bluetooth. La idea que hay detrás es ofrecer una conexión entre un

smartphone y un microcontrolador Arduino de forma transparente. De esta manera, se pueden programar eventos en el smartphone que vayan informando a Arduino de manera periódica. Por ejemplo, se podría informá a Arduino de los valores que va tomando un sensor específico de nuestro móvil.

7.2.2. Android APP

Hemos creado una aplicación para Android llamada “*Hegalari*” (pájaro). El menú de nuestra app, tiene 3 botones. El último de ellos es el que nos interesa.

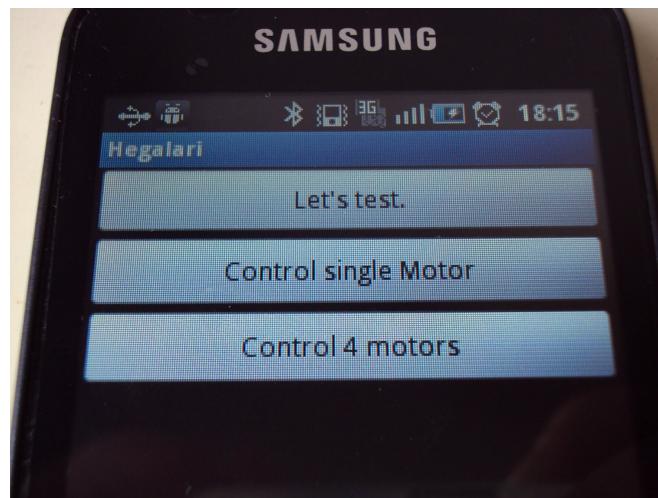


Figura 44. Hegalari app. Menú.

“Control 4 motors” nos llevará a una pantalla que contiene dos objetos “SeekBar” (figuras X y X). Mediante la barra superior, indicaremos la potencia que recibirán los motores.

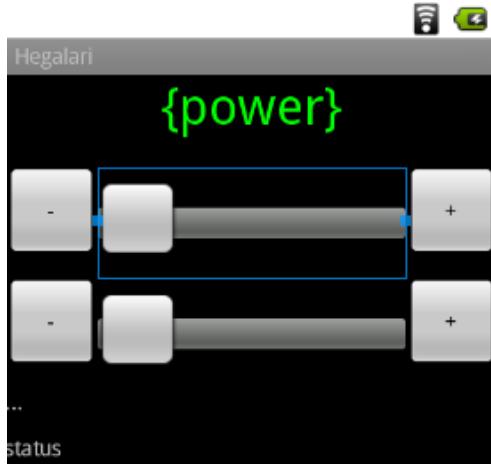


Figura 45. Seekbar que controlará la velocidad de rotación.



Figura 46. *FourMotorActivity*.

Echémosle un vistazo a la clase de Java “*FourMotorActivity.java*” que hay detrás del layout de la figura X. La clase implementa la clase “Activity”, que no es más que un tipo de objeto estándar de Android con un layout propio, donde el usuario puede interactuar. No mostraremos en este documento todo el código de la activity, ya que se encuentra en el Anexo I, pero sí que vamos a explicar y ver algunas líneas importantes del código.

En primer lugar definimos una clase ArduinoReceiver (que crearemos más abajo) y la registramos como un receiver del sistema. Esto servirá para recibir feedback desde arduino. Después nos conectamos al dispositivo Bluetooth cuya dirección MAC tenemos que proporcionar. Cuando la activity termina, hay que acordarse de desconectarse y des-registrar el receiver. Si se quiere pasar alguna variable a Arduino, utilizaremos la función sendDataToArduino().

```
package eus.xabi.ocell;

import at.abraxas.amarino.Amarino;
import at.abraxas.amarino.AmarinoIntent;

public class FourMotorActivity extends Activity implements
    OnSeekBarChangeListener {

    // Dirección MAC del módulo Blue SMiRF Bluetooth
    private static final String DEVICE_ADDRESS = "00:06:66:46:42:FC";
```

```

// Creamos el objeto Receiver
private ArduinoReceiver arduinoReceiver = new ArduinoReceiver();

public void onCreate(Bundle savedInstanceState) {
    [...]
    // Connect to Bluetooth module
    Amarino.connect(this, DEVICE_ADDRESS);
    [...]
}

@Override
protected void onStart() {

    // Registraremos nuestro Receiver para poder “escuchar” a Arduino
    registerReceiver(arduinoReceiver, new IntentFilter(
        AmarinoIntent.ACTION_RECEIVED));
}

@Override
protected void onStop() {
    super.onStop();
    // stop Amarino's background service, we don't need it any more
    Amarino.disconnect(this, DEVICE_ADDRESS);
    // do never forget to unregister a registered receiver
    unregisterReceiver(arduinoReceiver);
}

private void updatePower() {
    Amarino.sendDataToArduino(this, DEVICE_ADDRESS, 'f', power);
}

/**
 * ArduinoReceiver is responsible for catching broadcasted Amarino events.
 *
 * It extracts data from the intent and updates the graph accordingly.
 */
public class ArduinoReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {

```

```
    TextView tvStatus = (TextView) findViewById(R.id.tvStatus);
```

```

tvStatus.setText("Connected.");
String data = null;

// the device address from which the data was sent, we don't need it
// here but to demonstrate how you retrieve it
final String address = intent
        .getStringExtra(AmarinoIntent.EXTRA_DEVICE_ADDRESS);

// the type of data which is added to the intent
final int dataType = intent.getIntExtra(
        AmarinoIntent.EXTRA_DATA_TYPE, -1);

// we only expect String data though, but it is better to check if
// really string was sent
// later Amarino will support differnt data types, so far data comes
// always as string and
// you have to parse the data to the type you have sent from
// Arduino, like it is shown below
if (dataType == AmarinoIntent.STRING_EXTRA) {
    data = intent.getStringExtra(AmarinoIntent.EXTRA_DATA);

    if (data != null) {
        // mValueTV.setText(data);

        try {
            // since we know that our string value is an int number
            // we can parse it to an integer
            // final int sensorReading = Integer.parseInt(data);
            // mGraph.addDataPoint(sensorReading);
            // showIMU(sensorReading, sensorReading);

            final String sensorReading = data;
            showIMU(sensorReading, sensorReading);

        } catch (NumberFormatException e) {

        }
    }
}
}

```

7.2.3. Arduino Firmware

Por otra parte el código que correrá en Arduino también tiene que ser capaz de “escuchar”. Antes de mostrar el código que hemos creado, lo explicamos brevemente: Primero se incluyen las librerías necesarias y despues en la sección de configuración (función setup()) se registra **controlMotors** como una función que se ejecutará cada vez que a Arduino le llegue una función con el tag “f” (en este caso). En esa función lo que hacemos es mapear el valor que hemos recibido en el rango que nos interesa y escribirlo en los motores. Pasemos a ver el código.

```
/*
Hegalari v 0.3
Control a quadricopter from Android
author: Xabier Legasa Martin-Gil - June 2012
*/
#include <MeetAndroid.h>
#include <Servo.h>
#include <Wire.h>
#include <FreeSixIMU.h>
#include <FIMU_ADXL345.h>
#include <FIMU_ITG3200.h>

MeetAndroid meetAndroid; // declare MeetAndroid so that we can call functions with it.
int fullRangeMotor = 150; // Afterwards set to 180
Servo motor1;
Servo motor2;
Servo motor3;
Servo motor4;
int pinmotor1 = 6;
int pinmotor2 = 9;
int pinmotor3 = 10;
int pinmotor4 = 11;
int val = 0; // variable we get from Android
int androidval;
float angles[3]; // yaw pitch roll
// Set the FreeSixIMU object
FreeSixIMU sixDOF = FreeSixIMU();

void setup()
{
    // initialize serial communication
    Serial.begin(115200);
    Serial.println("start");
    androidval = 0;
    meetAndroid.registerFunction(controlMotors, 'f'); // f = four_motors

    Wire.begin();
    delay(5);
    sixDOF.init(); //begin the IMU
    delay(5);

    // initialize motors
```

```

motor1.attach(pinmotor1); // attaches the servo on pin 9 to the servo object
motor2.attach(pinmotor2);
motor3.attach(pinmotor3);
motor4.attach(pinmotor4);

Serial.println("start ok");
}

void loop()
{
    meetAndroid.receive(); // you need to keep this in your loop() to receive events

    updateMotors();

    sixDOF.getEuler(angles);
    sendIMUdata();

    delay(20);
}

void sendIMUdata()
{
    // for now, send just first angle.
    meetAndroid.send(angles[0]);
}

/*
 * Whenever the servoControl app changes the power value
 * this function will be called
 */
void controlMotors(byte flag, byte numOfValues)
{
    //Serial.println("controlServo bat");
    androidval = meetAndroid.getInt();
    //Serial.println("controlServo bi");
    val = map(androidval, 0, 1023, 0, fullRangeMotor);
    //Serial.println(val, DEC);
    //Serial.println("controlServo hiru");
}

void updateMotors()
{
    // at this point all motors are running at same speed.
    motor1.write(val);
    motor2.write(val);
    motor3.write(val);
    motor4.write(val);
}

```

8. Algoritmo de estabilización. Bases.

8.1. Controlador PID

La familia de controladores PID, es una estructura de control mediante realimentación, que calcula la desviación entre un valor medido y un valor que se quiere obtener, haciendo una corrección en consecuencia. Este tipo de controladores han demostrado ser robustos y es por eso que son utilizados en más del 95% de los procesos industriales en lazo cerrado[27].

El algoritmo de control calcula tres parámetros diferentes: el Proporcional, el Integral y el Derivativo. El Proporcional, es directamente proporcional al error actual, el Integral hace una corrección del error acumulado en el tiempo (integral del error) y el Derivativo determina la reacción del tiempo en el que el error se produce.

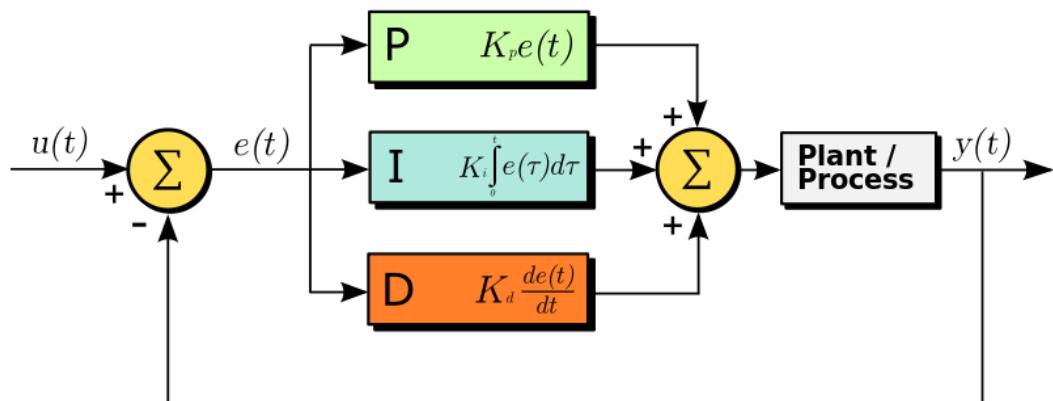


Figura 47. Esquema PID.

Las variables que nos interesan son:

- PV ([process variable](#)) o process value). Valor medido.
- SP (setpoint). Valor deseado. $u(t)$
- MV (manipulated variable). Valor de entrada al sistema.
- E (Error). Diferencia entre valor medido(PV) y el deseado (SP)

Un ejemplo típico con el que se explica un controlador PID es cuando queremos mantener una caldera a una temperatura determinada. Por ejemplo, queremos mantenerla a 100°C (SP) pero en un

momento determinado el termómetro nos dice que está a, 106°C(PV). El error será de 6°C (e) que cuantifica si el agua está muy caliente o muy fría. Una vez calculado este error, el sistema calculará cuánto hay que abrir/cerrar la válvula de agua(MV). Si es necesario mucho más calor, habrá que abrir mucho la válvula. O todo lo contrario si solo hace falta calentarla un poco. Esto sería un ejemplo de Control **Proporcional**. (un PID donde no hay ni I ni D). Podríamos abrir la válvula más y más cada vez si no obtenemos el resultado deseado. Esto sería un Control **Integral**. El controlador puede querer ajustarse también, con el objetivo de anticiparse a futuras oscilaciones. Esto sería el método del **Derivativo**.

El valor de entrada al sistema (MV) se calcula por tanto como la **suma** del valor Proporcional, Integral y Derivado. Es importante decir que K_p K_i y K_d son constantes que habremos de ajustar en nuestro sistema. Vamos a verlos por separado con un poco más de detalle.

$$u(t) = MV(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{d}{dt} e(t)$$

donde

K_p : Proportional gain, a tuning parameter

K_i : Integral gain, a tuning parameter

K_d : Derivative gain, a tuning parameter

e : Error = $SP - PV$

t : Time or instantaneous time (the present)

8.1.2. Proporcional

El valor proporcional es el producto entre la constante proporcional K_p y el error ($SP-PV$).

$$P_{out} = K_p e(t)$$

Un K_p demasiado grande llevaría al sistema a ser inestable. Uno demasiado pequeño, tiene como consecuencia, un error grande. La parte proporcional no considera el tiempo, por lo tanto, la mejor manera de solucionar el error permanente y hacer que el sistema contenga alguna componente que tenga en cuenta la variación respecto al tiempo, es incluyendo y configurando las acciones integral y derivativa.

8.1.3. Integral

El valor correspondiente al control Integral, es proporcional tanto a la magnitud del error, como a la duración del mismo. Es la suma de los errores en el tiempo e indica el cúmulo de errores que tendrían que haberse corregido previamente. Este error acumulado se multiplica por la constante Ki.

$$I_{\text{out}} = K_i \int_0^t e(\tau) d\tau$$

El control integral se utiliza para obviar el inconveniente del offset (desviación permanente de la variable con respecto al punto de consigna) de la banda proporcional.

8.1.4. Derivativo

La acción derivativa actúa cuando hay un cambio en el valor absoluto del error. La función de la acción derivativa es mantener el error al mínimo corrigiéndolo proporcionalmente con la misma velocidad que se produce; de esta manera evita que el error se incremente. El error se deriva con respecto al tiempo y se multiplica por una constante D y luego se suma a las señales anteriores (P+I).

$$D_{\text{out}} = K_d \frac{d}{dt} e(t)$$

En nuestro ejemplo de la caldera, planteado anteriormente, la acción derivada es adecuada cuando hay retraso entre el movimiento de la válvula de control y su repercusión a la variable controlada.

8.2. PID en Arduino

“**PID Library**” es una librería[23] para Arduino que implementa el control PID. Existe, a su vez, una librería llamada “**PID Autotune Library**” cuya misión es automatizar el máximo posible el ajuste de las constantes Kp, Ki y Kd que la librería necesita.

Existe otra implementación muy interesante llamada “BBCC: Bare Bones (PID) Coffee Controller” [26] que aunque inicialmente fue usado para controlar la temperatura de una maquina de cafe, es en realidad, un PID de uso general. Entender estas dos implementaciones es útil porque muchos de los proyectos de código abierto hacen uso de una de estas dos soluciones. La función updatePID() de esta última librería es una implementación del PID sencilla de entender.

```
float updatePID(float targetTemp, float curTemp)
```

```

{

    // these local variables can be factored out if memory is an issue,
    // but they make it more readable
    double result;
    float error;
    float windupGuard;

    // determine how badly we are doing
    error = targetTemp - curTemp;

    // the pTerm is the view from now, the pgain judges
    // how much we care about error we are this instant.
    pTerm = pgain * error;

    // iState keeps changing over time; it's
    // overall "performance" over time, or accumulated error
    iState += error;

    // to prevent the iTerm getting huge despite lots of
    // error, we use a 'windup guard'
    // (this happens when the machine is first turned on and
    // it can't help be cold despite its best efforts)

    // not necessary, but this makes windup guard values
    // relative to the current iGain
    windupGuard = WINDUP_GUARD_GAIN / igain;

    if (iState > windupGuard)
        iState = windupGuard;
    else if (iState < -windupGuard)
        iState = -windupGuard;
    iTerm = igain * iState;

    // the dTerm, the difference between the temperature now
    // and our last reading, indicated the "speed,"
    // how quickly the temp is changing. (aka. Differential)
    dTerm = (dgain * (curTemp - lastTemp));

    // now that we've used lastTemp, put the current temp in
    // our pocket until for the next round
    lastTemp = curTemp;
}

```

```
// the magic feedback bit  
return pTerm + iTerm - dTerm;  
}
```

9. Conclusiones

9.1. Objetivos conseguidos

Se ha conseguido los objetivos con éxito. Se ha realizado el diseño e implementación del sistema cuadricóptero satisfactoriamente. También se ha conseguido establecer una conexión entre la aplicación Android y el micro. La aplicación es capaz de enviar las variables que el usuario envía y en este momento Arduino tambien es capaz de enviar datos de vuelta (su orientación). A partir de aquí, es relativamente sencillo modificar o ampliar esta funcionalidad.

9.2. Desviaciones de planificación

A lo largo del proyecto han ido surgiendo complicaciones imprevistas que han hecho cambiar el orden de alguna tarea planificada. Por ejemplo en relación al montaje del cuadricoptero, el tiempo de llegada de los componentes no fue igual en todos los casos: Se decidió acertadamente comprar el módulo Bluetooth, aparte y en seguida, para poder trabajar en la comunicación con Android. Fué una decisión acertada, porque el resto de componentes llegó bastante tarde. Sin el módulo bluetooth no se podría haber avanzado y el proyecto se habría estancado.

9.3. Valoración económica

El cálculo total del proyecto se va a calcular respecto a do factores. Coste del material y coste por las horas de trabajo realizadas. La siguiente tabla muestra el precio de cada componente utilizado. Todos los precios están en €.

Elemento	Proveedor	Precio unidad(€)	Cantidad	Total
Frame	hobbyking.com	11.17	-	11.17
Motor	hobbyking.com	9.25	x4	37
ESC	hobbyking.com	6.70	x4	26.8
Batería	hobbyking.com	6.70	-	6.70
Bateria arduino(9v)	tienda local	9	-	9
jack batería	bcn cybernetics.com	1.5	-	1.5
Microcontrolador	cooking-hacks.com	20	-	20
Hélices(5x)	hobbyking.com	2.97	-	2.97
IMU	sparkfun.com	51.66	-	51.66
Bluetooth	sparkfun.com	31.78	-	31.78
Arnés de energía	tienda local	2	-	2
Comutador	diotronic.com	3	-	3
Cargador Batería	hobbyking.com	8	-	8
SUB Total				€ 211

Los componentes se han comprado en su mayoría on-line. Otros elementos más pequeños, en tiendas de electrónica de Barcelona. La relación queda de esta manera:

cooking-hacks.com	Arduino
sparkfun.com	IMU
sparkfun.com	Blue SMiRF Bluetooth
hobbyKing.com	frame, motores, ESCs, Batería, hélices, Cargador
Tiendas Barcelona	Arnés de energía
Diotronic BCN	Comutador

El mayor de los paquetes (hobbyking.com), que venía de Singapur, tenía un precio marcado en la caja de 161\$ (123€ aprox.). En la aduana de barajas, se aplicó un I.V.A. al 18% de 22.27€, un importe por el despacho de €14.40 y el I.V.A. del despacho €2.59. Hubo que pagar a su llegada la nada despreciable cantidad de **€39.26** para poder recibirla. Esto representó un gasto no esperado de casi un 20% más del valor total de los componentes hasta el momento. De haber sabido esta circunstancia, se habría buscado una alternativa como por ejemplo hacer los pedidos en partes más pequeñas (paquetes de menos de 45 Euros no pagan ni I.V.A. ni porte, y paquetes entre 45 y 150 pagan I.V.A. pero no porte). El precio de los materiales por tanto queda en $211+39.26 = \textbf{250.26 €}$. Esto supone que estos costes inesperados son ahora el 15.6% del precio total del material.

Aún y todo, la conclusión que se saca es que hemos sido capaces de construir un cuadricóptero más barato que el proyecto mas similar de los que habíamos analizado al principio (AR Parrot, que valía 300€). El aeroquad AR Parrot dispone sin embargo de algunos componentes, como por ejemplo 2 cámaras integradas con lo que las comparaciones reales no son posibles.

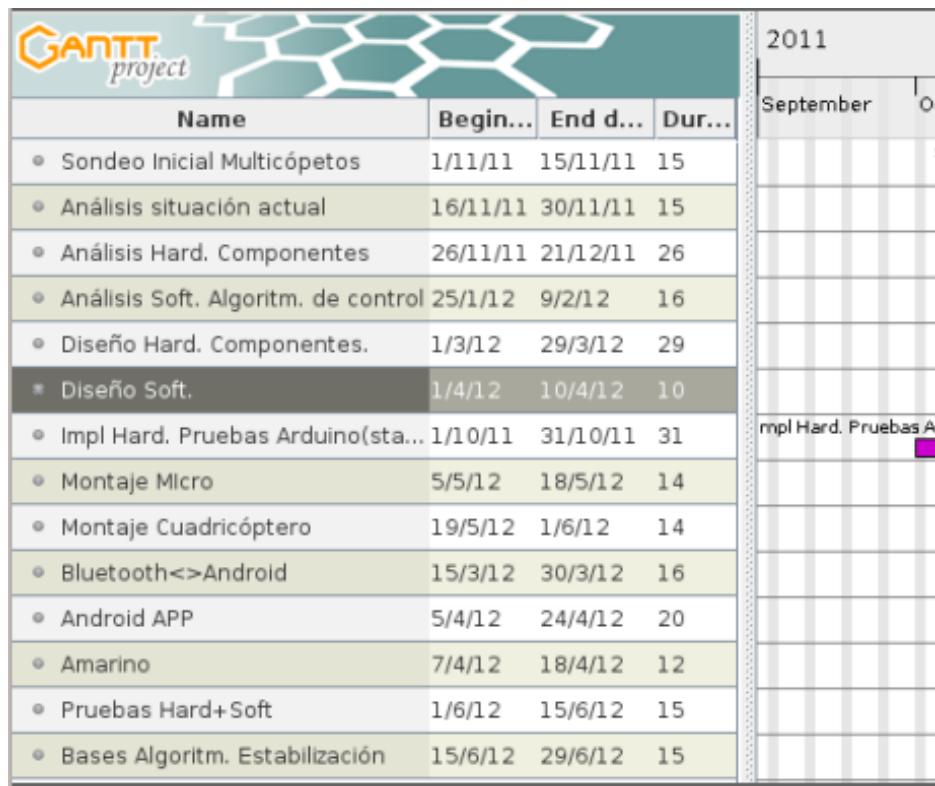
Si tenemos en cuenta que un Analista cobra 28.000 € anuales [28] y que durante un año se trabaja una media de 1.700 horas, el sueldo por hora de un analista sería de 16.4€/hora. Por la misma regla de tres, un Ingeniero Informático que cobra 24.000 €[29] anuales cobra por hora 14.1 €.

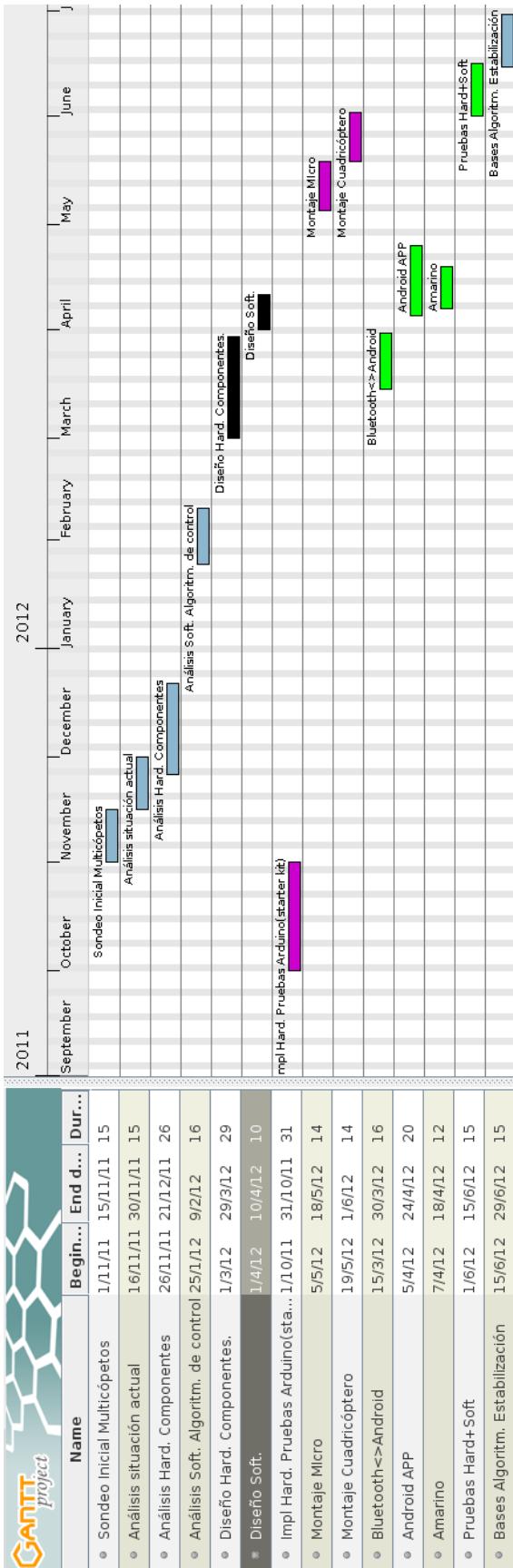
	Horas	Precio €/hora	Total €
<i>Estudio previo</i>	100	16.4 €	1640 €
<i>Analisis y Diseño</i>	175	16.4 €	2870 €
<i>Implementación Hard</i>	80	14.1 €	1128 €
<i>Implementación Soft. Comunicación</i>	160	14.1 €	2256 €
<i>Memoria</i>	85 h	16.4 €	1394 €
<i>Total sin material</i>	600 h	-	9288 €
<i>Total con material</i>			9538.26 €

10. Planificación

El proyecto ha sido llevado a cabo en diferentes etapas consecutivas pero, dada la naturaleza del proyecto, se han realizado algunas en paralelo. Esto es debido por una parte a la clara distinción entre tareas de hardware (montaje del cuadricóptero) y por otra, tareas lógicas como la programación, por ejemplo. Además, se ha tenido que tener en cuenta los períodos que van desde la compra on-line de un componente, hasta que este se recibía.

A continuación, el diagrama de Gantt, con la planificación que se ha seguido. Se muestra una primera captura de pantalla, para más claridad y el gráfico entero a continuación.





11. Posibilidades de trabajo futuro.

11.1. Prueba del algoritmo de estabilización.

Una vez que hemos diseñado y construido el sistema y cuadricóptero, un trabajo futuro podría ir encaminado a probar algoritmos de control utilizando alguno de las librerías de control PID descritas anteriormente. Existe varios proyectos de los mencionados en el capítulo 4 (Estado del arte), que publican el código con algún tipo de licencia abierta. El código de “Aeroquad”, por ejemplo, ha sido utilizado en otros proyectos como “scout UAV”, para hacer volar su cuadricóptero. Una posible linea a seguir por tanto, sería intentar entender este código y modificarlo para hacerlo funcionar con nuestro hardware. Hay que destacar que la migración de este código no es una tarea trivial. Esto se debe a que el código de Aeroquad está pensado para poderse usar con diferentes tipos de microcontroladores Arduino en combinación con diferentes placas que contienen a su vez diferentes sensores. Habrá que adaptar el código a nuestras necesidades.

11.2. Aeroquad Software.

Vamos a dar una idea de por donde va el software que utiliza el proyecto Aeroquad. El código tiene un fichero de configuración (*UserConfiguration.h*) que permite configurar diferentes modelos del microcontrolador Arduino y diferentes configuraciones (en x o en +) así como definir diferentes sensores,etc... .

El archivo principal que se ejecuta es “AeroQuad.ino”. En la función *setup()* de ese archivo es donde se hacen todas las configuraciones. Se inicializan las variables oportunas dependiendo de las líneas comentadas/descomentadas del mencionado archivo.

En el bucle principal *loop()* que se ejecutará constantemente, se hacen lecturas de los sensores disponibles y se dan las órdenes correspondientes. He realizado un pequeño pseudocódigo de esta función principal:

```

función loop()

    coger_tiempo();
    medir_sensores_importantes(); // giroscopio y acelerómetro
    si(tiempo multiplo de 100 Hz){
        medir_resto_sensores();
        calcular_angulos();
        controlar_motores();
    }
    si(tiempo multiplo de 50 Hz){
        medir_comandos_de_piloto()
    }
    si(tiempo multiplo de 10 Hz){
        medir_barómetro();
        medir_telemetría();
        medir_voltage_batería();
        // ...
    }
    actualizar_tiempo();
end

```

Como se puede ver, la lógica es realizar con mas frecuencia las tareas mas críticas (lectura de sensores) y con menos, las que no son tan importantes.

A la hora de adecuar el código a nuestro sistema cuadricóptero, serían dos los aspectos importantes a tener en cuenta.

En primer lugar, tendríamos que prestar atención a los sensores configurados y además, ver donde es necesario cambiar los puertos de entrada de estos. Aeroquad trabaja con una placa que venden en su web donde cada sensor esta conectado a un determinado pin de Arduino. Este NO es nuestro caso, por tanto habría que cambiar los puertos de entradada, como los de la IMU por ejemplo.

El segundo paso para poder utilizar este software seria cambiar completamente la función “***medir_comandosde_piloto()***” para poder leer instrucciones por bluetooth desde el smartphone. Sin

embargo, este trabajo ya está realizado como hemos demostrado en el capítulo 7.2.3 (Arduino Firmware). La adaptación del código se basaría por tanto, en la configuración de los sensores y pines, mayormente.

Despues del trabajo de configuración, solo quedaría decidir los valores constantes del PID que se implementa. Tendremos que proteger bien el cuadricoptero para poder hacer las pruebas correspondientes e ir mejorando estos valores, hasta conseguir una estabilidad óptima del aparato.

12. Referencias.

* Links válidos a 1 de Julio de 2012.

- [1] Arduino. Main Site. <http://www.arduino.cc/>.
- [2] Android. Main site. <http://www.android.com>.
- [3] Shredquette Project. Main site <http://shredquette.blogspot.com/>.
- [4] Aeroquad. Main site. <http://www.aeroquad.com>.
- [5] DiyDrones. Main site. <http://www.diydrones.com/>
- [6] Mikuadricóptero. Main site. <https://sites.google.com/site/mikuadricoptero/>
- [7] OpenPilot. Main site. <http://www.openpilot.org>
- [8] Parrot. Company Main Site. <http://www.parrot.com>
- [9] NG UAVP. Main site. <http://ng.uavp.ch/moin/FrontPage>
- [10] Wikipedia. Motor con escobillas http://es.wikipedia.org/wiki/Motor_de_corriente_continua
- [11] PWM. Pulse Width Modulation http://en.wikipedia.org/wiki/Pulse-width_modulation
- [12] BBC. “Google funds computer teachers and Raspberry Pis in England”. Noticia del 23 Mayo de 2012. <http://www.bbc.com/news/technology-18182280>
- [13] ATmega328. Datasheet. <http://www.atmel.com/Images/doc8161.pdf>
- [14] Sparkfun. Main site. <http://www.sparkfun.com>
- [15] ADXL345. Acelerómetro. [Datasheet.](#)
<http://www.sparkfun.com/datasheets/Sensors/Accelerometer/ADXL345.pdf>
- [16] ITG3200. Giroscopio. <http://www.sparkfun.com/datasheets/Sensors/Gyro/PS-ITG-3200-00-01.4.pdf>
- [15] Sparkfun. Bluetooth Modem - BlueSMiRF Silver. <http://www.sparkfun.com/products/10269>
- [16] Arduino. Especificaciones modelo Arduino Uno. <http://arduino.cc/en/Main/ArduinoBoardUno>
- [17] Wikipedia. Android. <http://es.wikipedia.org/wiki/Android>
- [18] Wikipedia. JAVA. [http://en.wikipedia.org/wiki/Java_\(programming_language\)](http://en.wikipedia.org/wiki/Java_(programming_language))
- [19] Wikipedia. Baudio. <http://es.wikipedia.org/wiki/Baudio>
- [20] Fabio Varesano. Implementación FreeSixIMU. <http://www.varesano.net/blog/fabio/my-first-6-dof-imu-sensors-fusion-implementation-adxl345-itg3200-arduino-and-processing>
- [21] Fabio Varesano. Main site. <http://www.varesano.net/>
- [22] Starlino.com. Implementación Kalman. http://www.starlino.com/imu_kalman_arduino.html
- [23] Arduino. PID library. <http://arduino.cc/playground/Code/PIDLibrary>
- [24] Amarino. Main site. <http://www.amarino-toolkit.net/>
- [25] Sparkfun. WiFly shield. <http://www.sparkfun.com/products/9954>

- [26] Arduino. PIDForEspresso library.
<http://www.arduino.cc/playground/Main/BarebonesPIDForEspresso>
- [27] K.J. Åström & T.H. Hägglund, «New tuning methods for PID controllers,» Proceedings of the 3rd European Control Conference, p.2456–62.
- [28] Infojobs. Salario Analista. <http://salarios.infojobs.net/resultados.cfm?sueldo=Analista>
- [29] Infojobs. Salario Ingeniero Informático. http://salarios.infojobs.net/resultados.cfm?sueldo=ingeniero+inform%C3%A1tico&o_id=2

13. Bibliografía.

1. **Diydrones** [online] <http://diydrones.com>
2. **Aeroquad** [online] <http://aeroquad.com>
3. **ScoutUAV**. [online] <http://www.scoutuav.com>
4. **Shrediquette**. [online] <http://shredquette.blogspot.com.es/>
5. **Openpilot**. [online] <http://www.openpilot.org>
6. **ArduCopter**. [online] <http://code.google.com/p/arducopter/>
7. **Airhacks** [online] <http://airhacks.org/>
8. **Ted**. [online] <http://www.ted.com>
9. **Instructables**. [online] <http://www.instructables.com>
10. **Sparkfun**. [online] <http://www.sparkfun.com/>
11. **Bricogeek**. [online] <http://www.bricogeek.com>
12. **Hobbyking** [online] <http://www.hobbyking.com>
13. **Buildcircuit** [online] <http://www.buildcircuit.com>
14. **jeremyblum** [online] <http://www.jeremyblum.com>
15. **Adalfruit** [online] <http://www.adafruit.com>
16. **Samshield electronic designs**. [online] <http://samshieldesigns.blogspot.com.es/>
17. **Arduino** [online] <http://www.arduino.cc>
18. **Android** [online] <http://www.android.com/>
19. Modelling and control of mini-flying machines By Pedro Castillo, Rogelio Lozano, Alejandro E. Dzul
20. Wikipedia [online] <http://en.wikipedia.org/wiki/Multirotor>
21. A bot travel Blog. [online] <http://abottravel.blogspot.com.es/>
22. Varesano [online] <http://www.varesano.net>