Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA EN TECNOLOGÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

SEGUIMIENTO AUTÓNOMO BASADO EN GPS DE UNA PERSONA POR UN DRON

Alumno/Alumna: San Martín Garaluce, Jon

Director/Directora (1): Espinosa Acereda, Jon Koldobika

Curso: 2017-2018

Fecha: Bilbao, 17 de Julio de 2018

RESUMEN

Este trabajo consiste en el desarrollo de un dron "quadcopter" (cuatro motores) capaz de realizar un seguimiento autónomo al usuario mediante el uso del sistema de geoposicionamiento global GPS. Para ello, se desarrollará una aplicación de smartphone sencilla que enviará las coordenadas GPS del usuario al dron a través de ondas de radio. También se desarrollará el software de control del propio vehículo. Para lograr todo esto, el dron estará dotado de lo necesario para conocer su propia posición y, en base a ello, realizar dicho seguimiento.

LABURPENA

Lan honen helburua "quadcopter" (lau motore dituen) drone baten garapena da, erabiltzailearen jarraipen autonomo bat egiteko gai izango dena geoposizionamendu globaleko GPS sistema bidez. Horretarako, smartphonerako aplikazio bakun bat garatuko da zeinak erabiltzailearen GPS koordenatuak droneari bidaliko dizkio irrati-uhinen bidez. Baita ere dronea kontrolatzeko softwarea garatuko da. Hau lortzeko, dronea bere posizioa ezagutzeko behar dituen tresnaz hornituta egongo da eta hauek oinarri izanda aipaturiko jarraipena egingo du.

ABSTRACT

This work consists in the developing of a quadcopter dron (four motors) able to perform an autonomous following of the user by GPS. To do this, a simple smartphone app will be created wich will send the user's GPS coordinates to the dron via radio waves. Also, it will be developed the control software of the vehicle itself. To get all of this, the dron will have all the necessary to knowing its own position and, by that, to make that following.

CONTENIDO

Índice de ilustraciones

Índice de tablas

1 INTRODUCCIÓN

Un dron quadcopter, o cuadricóptero, consiste en un vehículo no tripulado dotado de cuatro hélices que lo levantan y propulsan. El concepto de vehículo aéreo dotado de cuatro hélices existe desde hace tiempo, dado que es un diseño relativamente sencillo y soluciona muchos inconvenientes que sufrían los aviones y helicópteros, como aquellos relacionados con el control del par de torsión y la inestabilidad del rotor de cola. Además, el hecho de no tener que incluir palas que varíen su ángulo como en el caso de los helicópteros, reduce mucho su costo y mantenimiento, así como su diseño. No obstante, los primeros modelos de cuadricópteros eran pilotados, y en seguida aparecieron problemas de estabilidad con la consecuente elevada carga de trabajo para el piloto. No sería hasta los avances en materia de aeronaves no tripuladas y en software, cuando por fin se pudieron desarrollar modelos funcionales de cuadricópteros.

Si nos referimos a un dron en general, como un vehículo aéreo no tripulado, debemos echar la vista atrás hasta la I Guerra Mundial, entre los años 1914 y 1918. En este periodo se desarrollaron los primeros modelos, y es destacable un vehículo aéreo no tripulado controlado por radio que se usaría como blanco aéreo de entrenamiento, además de como defensa contra los Zeppelines.

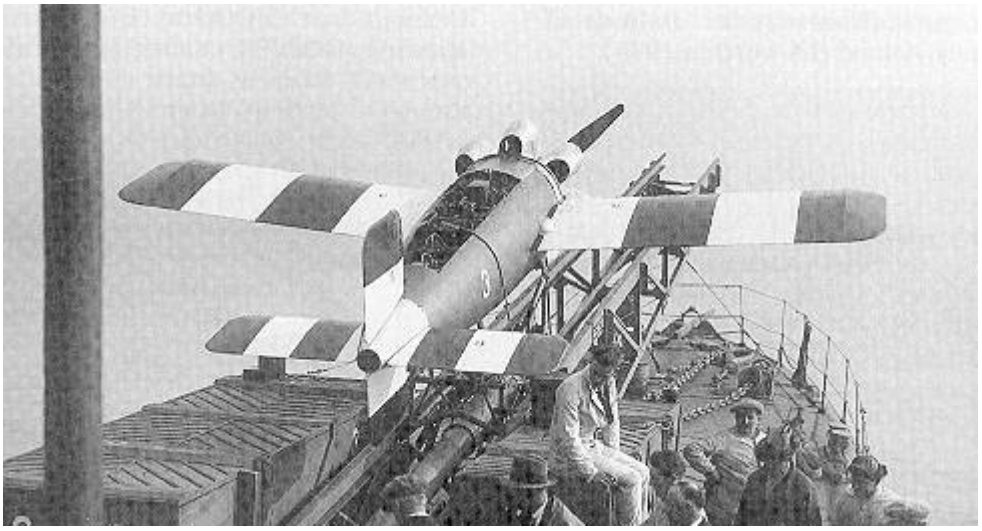


Ilustración 1. Larnyx británico

Gran Bretaña dio pie al desarrollo de blancos aéreos con control completo por radio, y durante el inicio de la II Guerra Mundial comenzaron a producir en cadena el “Queen bee”, avión capaz de transportar 400 kg de carga de guerra. Le siguieron progresivamente EEUU con el “RP4” de Radioplane Company y por último la Alemania nazi de Hitler, creando su V1. Gracias al desarrollo de todos estos modelos, se perfeccionó la tecnología que hoy conocemos como tecnología de control remoto por radio.

Ilustración 2.V1 alemán.

Durante los años 90 se produjo un gran avance en este tipo de vehículos, gracias sobre todo a los diferentes avances en la tecnología. Estos avances hicieron posible el abaratamiento de costes, así como la posibilidad de crear vehículos más versátiles y de tamaños cada vez menores. Como se ha indicado, esta tecnología ha sido siempre principalmente militar, y aunque hoy en día el uso de los UAV (*Unmanned Aerial Vehicle, es decir, Vehículo Aéreo No Tripulado)* en este ámbito sigue siendo elevado, en estos últimos años se ha experimentado un auge en el uso civil de este tipo de vehículos.

Ilustración 3. Repostaje en vuelo de UAV militar. (Fuente: Armada de los Estados Unidos)

Más adelante, se explicará en detalle qué es GPS, pero sería un sistema que permite determinar en toda la Tierra la posición de un objeto con una precisión de hasta centímetros. El sistema fue desarrollado, instalado y empleado por el Departamento de Defensa de los Estados Unidos. Con la mejora y el desarrollo de los drones, se ha logrado ir incorporando funcionalidades GPS a los mismos, desde la memorización de la ruta seguida hasta la programación de la misma. Por ello,

este TFG se centrará en el seguimiento autónomo del usuario.

Primeramente, se va a presentar el contexto en el cual se desarrolla el proyecto, así como un resumen del alcance esperado y los diferentes beneficios (económicos, sociales y técnicos) que ofrecerá este proyecto.

A continuación, se describirán las diferentes partes con las que cuenta un dron en general y nuestro proyecto en particular, para posteriormente realizar un análisis de alternativas y así elegir una solución final.

Seguidamente, se describirá en detalle los pasos tomados en el desarrollo de este TFG, detallando las cuestiones más relevantes y los problemas encontrados, así como el resultado final.

2 CONTEXTO

La razón o inspiración para centrar este TFG en el ámbito de los drones proviene del hecho de que actualmente, tanto el mercado de drones como su utilidad y aplicaciones, están en auge.

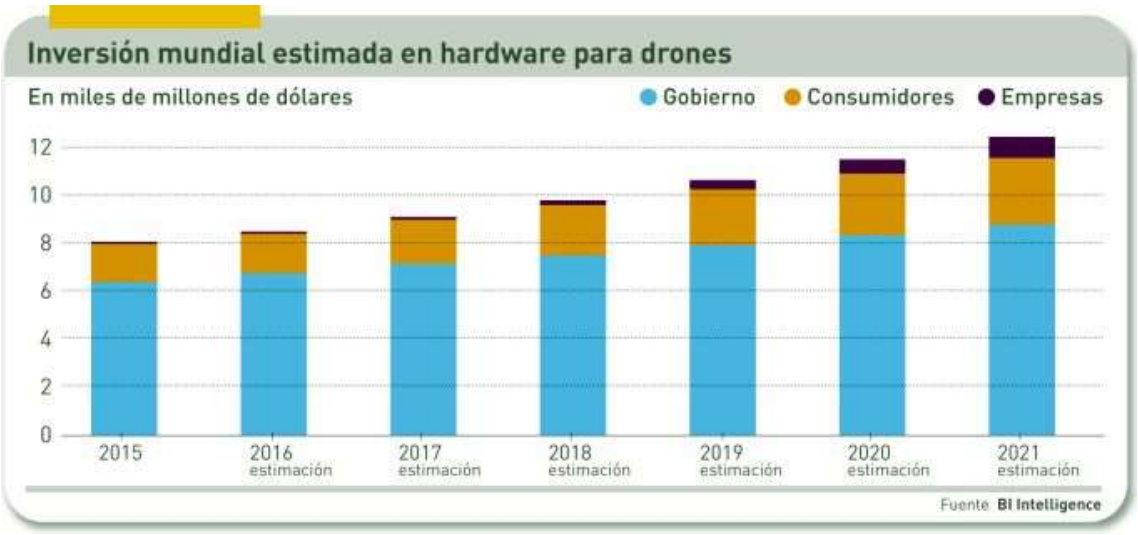


Ilustración 4.Inversión mundial estimada en hardware para drones. (Fuente: BI Intelligence)

Según datos de la consultora especializada en drones Droneii, existe una tendencia para 2018 y 2019 que consiste en el crecimiento de dicho mercado en general, y sobre todo un alza en la inversión en desarrollo de software. También crecen sus usos, de entre los cuales encontramos algunos como: grabaciones en eventos, entrega de paquetes, vigilancia fronteriza, vigilancia de incendios forestales, situaciones de emergencia, búsqueda de personas, etc. Sería en estos últimos aspectos en los que podríamos enfocar la utilidad del desarrollo de este TFG, puesto que un dron capaz de seguir autónomamente a una persona sería de gran ayuda en situaciones de emergencia o contextos de seguridad y vigilancia, así como para el ocio en general.

Si nos fijamos en el ocio, se puede observar una utilidad bastante clara que sería la de la grabación autónoma de un deportista (por ejemplo, un ciclista o corredor). Aunque este TFG no se va a centrar en dicha grabación, sería necesario, para llevarse a cabo, un seguimiento autónomo de dicho deportista basado en GPS. Mientras el deportista sigue una ruta, el dron se movería junto a él siguiendo la misma trayectoria (permitiendo su grabación o incluso la toma de estadísticas y su posterior procesado o envío). Además, dado que la señal de GPS funciona tanto en terreno como en el mar, también podría usarse para seguimiento de embarcaciones de todo tipo. En definitiva, hay una enorme cantidad de usos dentro del ocio, resumidos en todos aquellos en los que el usuario necesite un dron volando a su lado.

En el área de seguridad o emergencias, existen situaciones de peligrosidad en las que sería beneficiosa la existencia de un elemento que siguiera a la policía o a los miembros de los diferentes servicios de emergencias y, por ejemplo, fuera transmitiendo en tiempo real imágenes de la situación o diferentes datos, o incluso ciertas alarmas en momentos dados. Incendios, rescates, tiroteos u otras situaciones de riesgo se verían beneficiadas si incluyéramos a otro “agente” en el aire siguiendo la situación.

3 ALCANCE Y OBJETIVOS

Nuestro objetivo sería, por lo tanto, desarrollar un dron capaz de realizar el seguimiento de un usuario mediante GPS. Más concretamente, este TFG se centrará en desarrollar el software de control del movimiento de un dron capaz de realizar de manera autónoma el seguimiento de una persona mediante GPS. Para visualizar mejor el sistema, se mostrará el siguiente esquema.

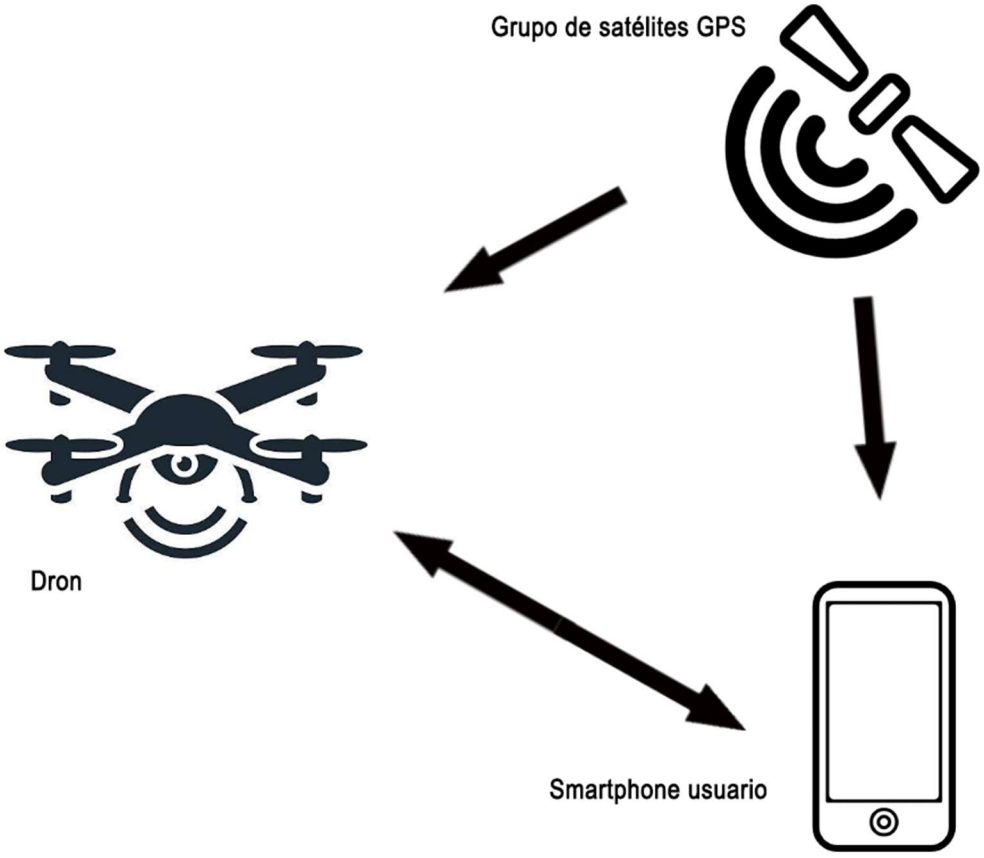


Ilustración 5. Esquema básico del prototipo del sistema

3.1 Software de control

Este software debe realizar lo necesario para que el vehículo realice un seguimiento autónomo del usuario. Para ello, usará hardware específico que se detallará más adelante. Este software debe conocer la posición del usuario, así como la del dron en sí mismo, y en base a ambas calcular el movimiento necesario para conseguir el seguimiento. También deberá estar dotado de lo necesario para que el dron sea estable y no necesite apenas intervención humana.

Por otro lado, debe ser capaz de despegar y aterrizar de forma segura.

3.2 Software de usuario (Aplicación de smartphone).

Como objetivo parcial, tendremos el de desarrollar un software de usuario basado en una aplicación de smartphone.

Para conocer la posición del usuario haremos uso de las capacidades de geoposicionamiento con las que hoy en día todo smartphone cuenta. Por tanto, se creará además una aplicación que obtenga los datos de posicionamiento del usuario y las envíe vía radio al dron. Esta aplicación estará desarrollada en lenguaje Java mediante el IDE (Integrated Development Environment, o Entorno de Desarrollo Integrado) Android Studio.

También, mediante la aplicación, el usuario será capaz de despegar y aterrizar el dron, así como de comenzar o detener el seguimiento cuando desee.

4 BENEFICIOS

4.1 Beneficios económicos

El interés por los drones aumenta cada día, por lo que desarrollar un proyecto relacionado con la materia traería, de entrada, beneficios económicos indiscutibles.

Además, se ha hablado del caso en el que mientras un deportista sigue una ruta, el dron se movería junto a él siguiendo la misma trayectoria, permitiendo su grabación o la toma de estadísticas. En este caso, ya existen drones que siguen al usuario, pero lo realizan mediante reconocimiento y procesado de imágenes, lo que encarece exageradamente su precio. Consiguiendo desarrollar un dron como el de este TFG, lograríamos un producto mucho más barato y por lo tanto asequible, dado que la tecnología GPS disponible es mucho más barata que el procesado de imágenes.

Podemos decir, entonces, que este proyecto nos daría una forma de seguimiento mucho más asequible para cualquier utilidad, provocando esto un beneficio económico destacable.

4.2 Beneficios sociales

A la hora de hablar de los beneficios sociales de este TFG, tenemos que hacer referencia a los distintos usos de los que se ha hablado. Como se ha mencionado, se podrían mejorar los servicios de seguridad o emergencias si se contara con drones capaces de realizar un seguimiento mediante GPS. Por ello, existiría una repercusión directa en la sociedad gracias a dicha mejora de servicios.

Por otro lado, cabría destacar que la simple mejora e investigación en tecnología, y la exploración de los límites de esta, repercute directa e indirectamente en la sociedad de forma beneficiosa.

4.3 Beneficios técnicos

En el aspecto técnico, toda innovación o nuevo desarrollo será positivo. Este TFG también permitirá demostrar la viabilidad técnica del seguimiento autónomo basado en GPS en general, y la del diseño de este TFG en particular. Para ello, mostrará también los problemas encontrados durante el desarrollo, como los relacionados con la cobertura de la señal de radio, precisión del geoposicionamiento, control autónomo de altitud, precisión en la orientación, etc.

5 Componentes y estructura funcional del sistema

5.1 Componentes del dron

Ilustración 6. Dron de uso civil. (Fuente: mundodron.net)

Un dron cuadricóptero o quadcopter actual estaría formado por numerosas partes, de las cuales las más importantes o las más comunes serían los motores, los ESCs (Electronic Speed Controlers), y la unidad de control.

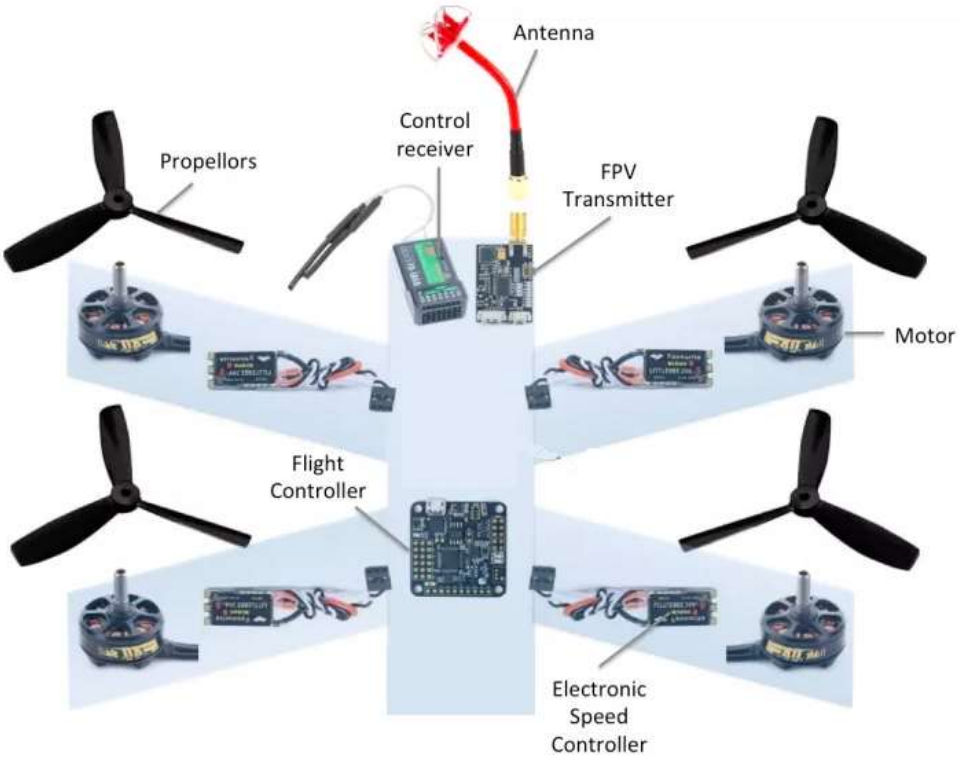


Ilustración 7. Partes básicas de un dron quadcopter. (Fuente: dronezon.com)

5.1.1 Motores

Los motores hacen funcionar el dron; giran las hélices para elevarlo y propulsarlo, pudiendo ser de distintos tamaños, velocidad y potencias.

El movimiento del dron se controla modificando la velocidad (RPM, o revoluciones por minuto) de los motores. Para ello, los cuatro motores se agrupan o asocian entre ellos de cierta forma, teniendo algunos girando en sentido horario y otros en sentido anti-horario.

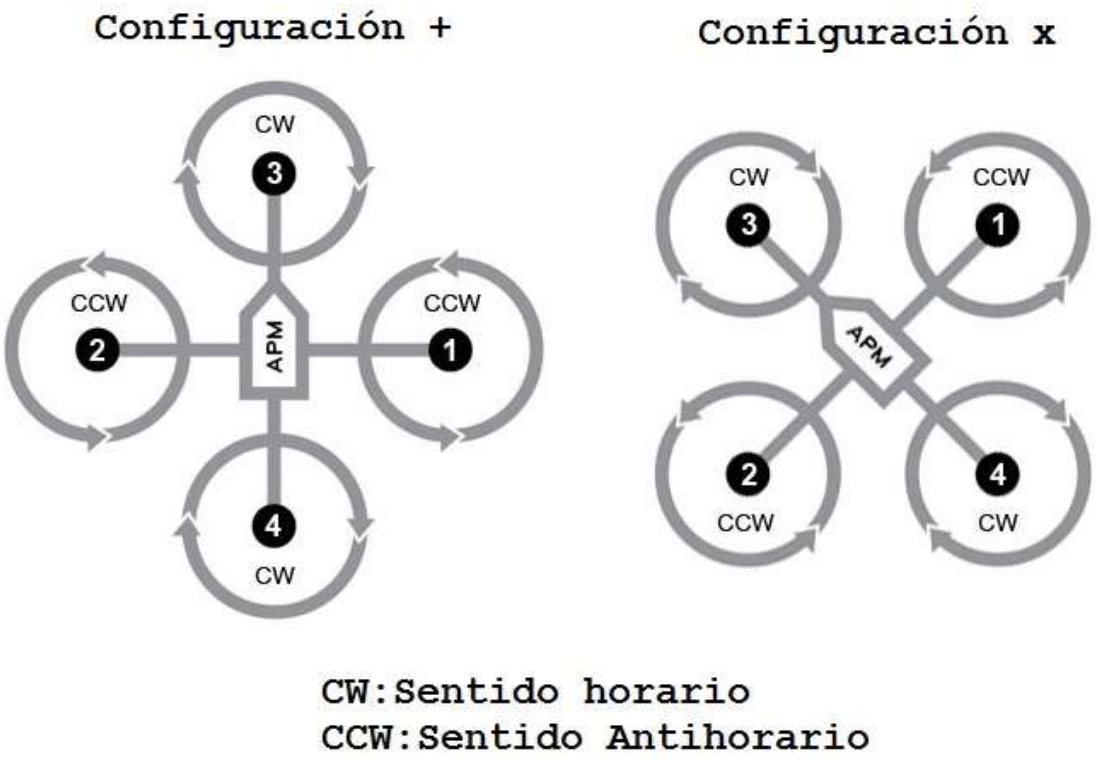


Ilustración 8. Tipos de configuración básica de motores y sentidos de giro. (Fuente: ingenio-triana.blogspot.com)

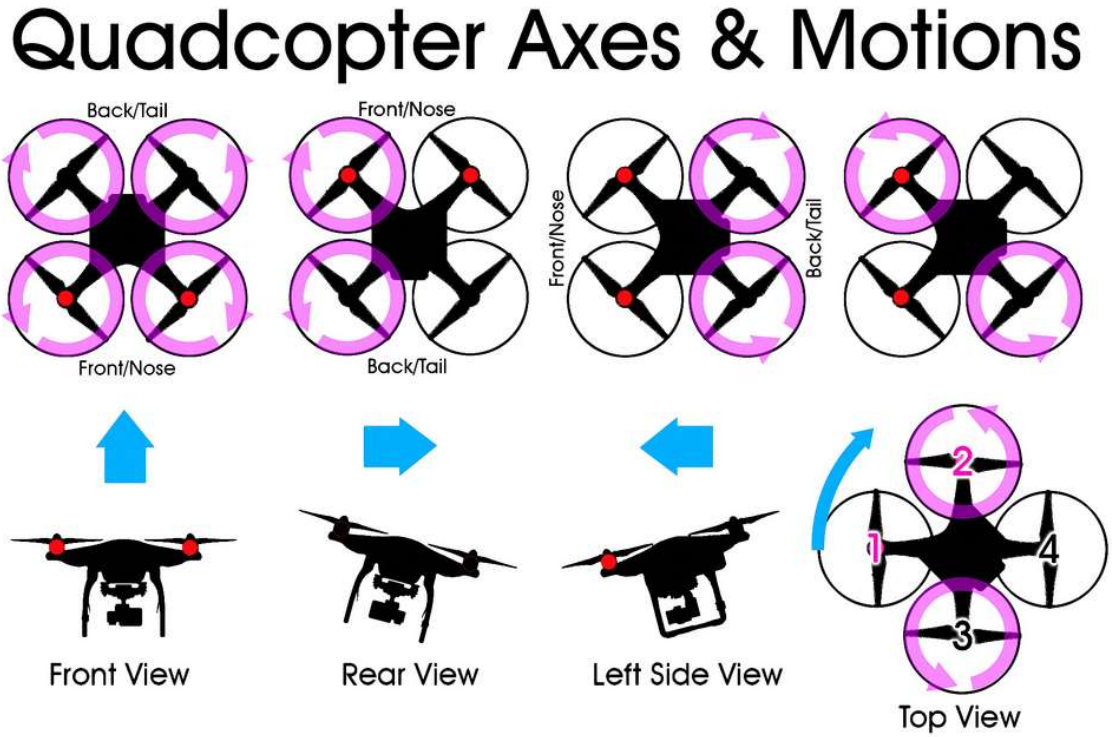


Ilustración 9. Ejemplo de reducción o aumento de RPM en cada motor para control del dron. (Fuente: Ric Stephens)

Si un Quad es simétrico y los motores de este generan la misma fuerza, el sumatorio de fuerzas será por lo tanto únicamente vertical. Si, además, F=m·g, es decir, la fuerza que ejercen los motores en conjunto es igual a su peso, la aeronave se quedará a la misma posición y altura.

Entonces, el sumatorio de las fuerzas ascendientes de cada rotor determina el movimiento del Quad. Por lo tanto, para generar sustentación la potencia tiene que ser como mínimo igual al peso. Cuando simétricamente aparece una compensación entre rotores, se genera una fuerza horizontal. Los quadcópteros son vehículos aéreos que generan los mismos movimientos que estos, solo que al no tener actuadores como en los aviones (alerones), estos generan sustentación de diferente manera. Esto quiere decir que por el simple hecho de que cada rotor gire a una determinada velocidad, se genera un momento, cada uno, de un sentido contrario al de la fuerza que genera el mismo motor. De manera que el sumatorio del momento debe de ser 0, sino la aeronave empezaría a girar sobre sí misma. Por lo tanto, dos motores de la misma diagonal giran en sentido horario y los otros dos en sentido antihorario. Según la mecánica de vuelo, los movimientos más sencillos se basan en aumentar la velocidad del rotor del lado hacia la dirección deseada.

Se entiende mejor con la siguiente imagen, similar a la anterior.

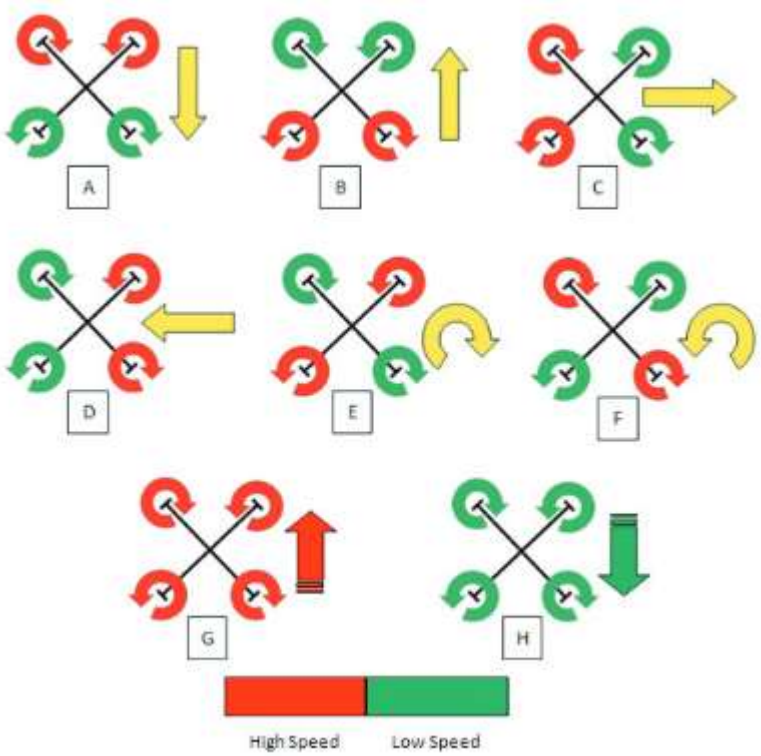


Ilustración 10. Combinación de movimientos posibles en un dron, y su relación con la velocidad de los distintos motores. (Fuente: "Física de un quadcóptero", por Estefanía Mancioc)

Si nos centramos en los motores, podemos distinguir principalmente entre dos tipos, que pueden ser con escobillas (brushed motors) o sin escobillas, (brushless motors). El brushed es un motor más tradicional, menos costoso y no alcanza velocidades relativamente grandes, mientras que el brushless es la última tecnología desarrollada y su precio es mucho más alto.

5.1.1.1 Motor Brushed

Este tipo de motores son cada vez menos utilizados en aeromodelismo, ya que son bastante más pesados que los motores brushless y las escobillas pueden generar chispas. Por otro lado, generan corrientes parasitas que hacen que el motor se caliente mucho más.

La principal ventaja de este motor es la posibilidad de regular la velocidad desde vacío a plena carga. Como se ha indicado anteriormente, estos motores son más baratos, pero no alcanzan grades velocidades.

5.1.1.2 Motor Brushless

Esta clase de motor no emplea escobillas para realizar el cambio de polaridad en el rotor, por lo que desaparecen los problemas que estas generan. Su control es más complicado (requiere ESCs) y son más caros, pero resultan más eficientes y alcanzan mayores velocidades, haciéndolo deseable para este proyecto.

5.1.2 Electronic Speed Controlers (ESCs)

Los reguladores de velocidad o ESC (*Electronic Speed Control)* son los que se encargan de proporcionar electrónicamente las revoluciones necesarias a cada motor/hélice para realizar los movimientos deseados. Cada motor debe tener asociado un ESC, luego son controladores individuales y funcionan de forma individual. Para el caso de los drones, estos ESCs deben ser de respuesta rápida.

Ilustración 11. Ejemplo de ESC. (Fuente: robotshop.com)

5.1.3 Placa microcontroladora o Unidad de Control

Sería el "cerebro" de todo dron. Consta del hardware y software necesario para responder a los comandos del usuario, realizar los cálculos para controlar los motores (mediante los ESCs) y así dotar de movimiento y estabilidad al dron, controlar la altitud, posición, etc. Es la parte en la que más se ha investigado y la que más desarrollo ha sufrido en los últimos años.

Al hablar de placa microcontroladora nos referimos al microcontrolador en sí y al hardware adicional que lleva asociado, así como a la placa física en la que vendría todo ensamblado. Por lo tanto, uno de los elementos más importantes de esta misma sería el microcontrolador, que establecerá la mayoría de las capacidades y funcionalidades que se podrán utilizar.

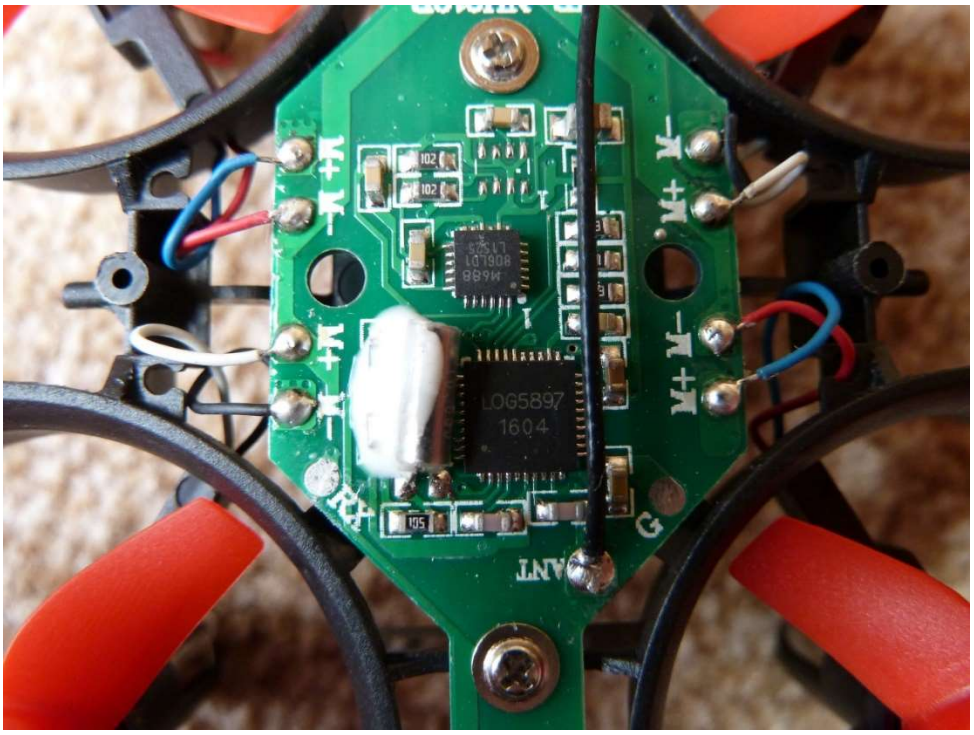


Ilustración 12. Ejemplo unidad de control de dron. (Fuente: rcgroups.com)

5.1.4 IMU (Inertial Measurement Unit)

La unidad de medición inercial o IMU, es un dispositivo electrónico que mide la velocidad, orientación y fuerzas gravitacionales de un aparato mediante el uso de una combinación de acelerómetros y giroscopios e incluso magnetómetros. Mediante esta, se pueden detectar por lo tanto los cambios de una aeronave, en este caso de nuestro dron, en la guiñada (yaw), cabeceo (pitch) y alabeo (roll), mediante el uso de uno o más giroscopios. Es posible también medir el desplazamiento gracias a los acelerómetros.

Son usadas en sistemas de guía inercial instalados en vehículos. Prácticamente todas las aeronaves comerciales o militares poseen una y por supuesto los vehículos espaciales.
Como se ha señalado, la mayoría cuentan con un acelerómetro y un giroscopio, pero algunas poseen también magnetómetros u otros sensores, y varían en cuanto al número de ejes en los que realizar las mediciones.

5.1.4.1 Acelerómetro

Un acelerómetro es un dispositivo que mide la aceleración, es decir, la tasa de cambio de la velocidad de un objeto. Normalmente se basan en 3 sensores para cada eje xyz aunque los hay de 2 ejes o 1. Para realizar la medición usan una masa determinada, relacionada con cada sensor, que por su propio masa ejercerá una fuerza sobre estos al sufrir una aceleración. Al encontrarse en reposo, y con el plano XY paralelo al suelo, en acelerómetro detectará una aceleración de 1 g en el eje Z, puesto que sufrirá la aceleración de la gravedad. Hay que señalar que la aceleración de la gravedad varía según altura y posición, pero de media se acepta 1 g = 9,8 m/s². Si por ejemplo el sistema se encuentra en caída libre, pese a experimentar una aceleración hacia el suelo, el acelerómetro indicará 0 g en el eje Z, dado que está en un marco de referencia en el que no tiene peso (caída libre). Esto es algo a tener en cuenta a la hora de usar este componente.

Los sensores en cada eje, generalmente, contienen placas capacitivas internamente. Al variar las distancias entre placas debido al movimiento de las masas, se crearán diferencias de potencial proporcionales a este.

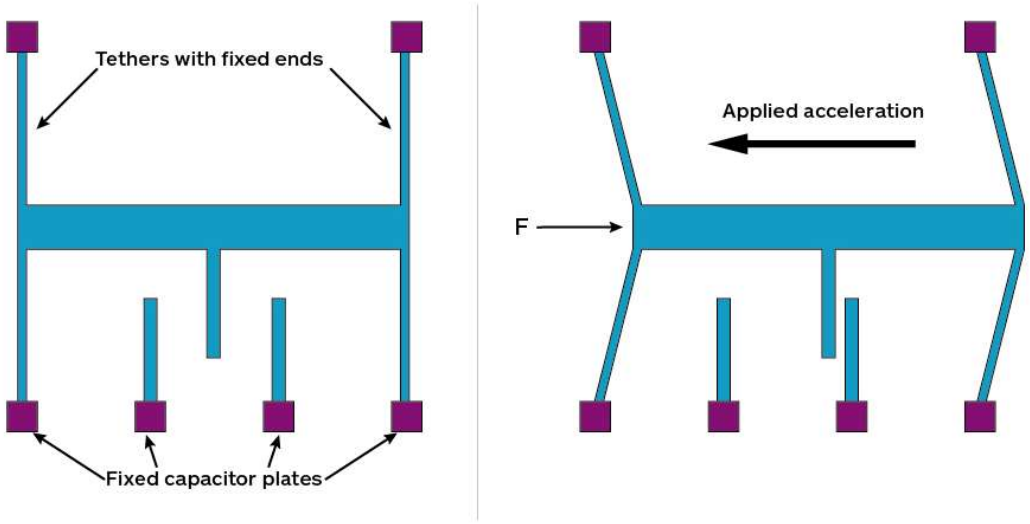


Ilustración 13. Ejemplo de funcionamiento del acelerómetro. (Fuente: globalspec.com)

5.1.4.2 Giroscopio

A diferencia del acelerómetro, el giroscopio no es capaz de medir movimiento longitudinal, sino que mide movimiento angular. Cuando se hace girar el giroscopio, una pequeña masa se desplaza a medida que cambia la velocidad angular. Este movimiento se convierte en señales eléctricas de muy baja corriente que pueden ser amplificadas y leídas por un microcontrolador. Al igual que con el acelerómetro, existen giroscopios de 3 ejes, de menos e incluso de más (6 ejes).

5.1.5 Módulo GPS

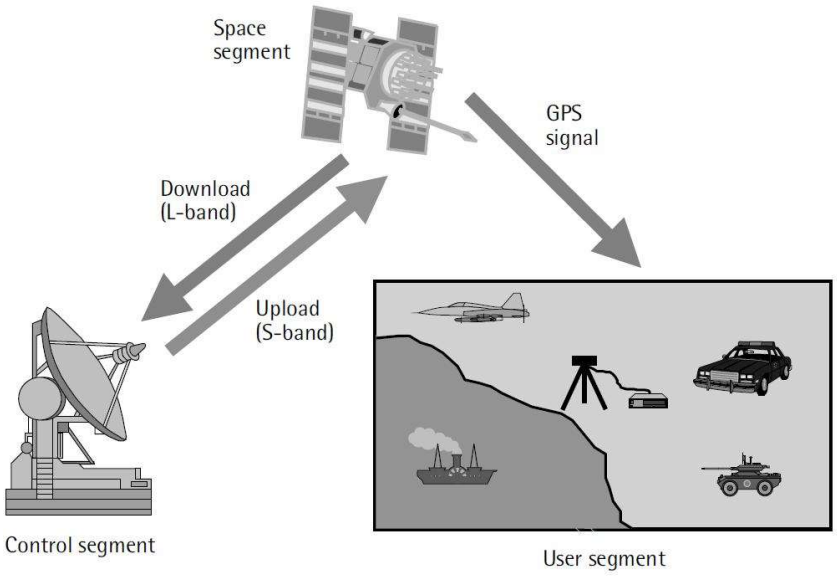
La clave o la originalidad de este proyecto reside en las capacidades de geoposicionamiento con las que contará el dron, valiéndose de la tecnología GPS. Por ello, contará con un módulo GPS. Además, muchos drones comerciales cuentan con ciertas capacidades GPS, como por ejemplo el registro de la ruta seguida. Por ello, se debe explicar en qué consiste dicha tecnología.

5.1.5.1 GPS

El Global Positioning System (GPS), en castellano, Sistema de Posicionamiento Global, es un sistema que permite determinar en toda la Tierra la posición de un objeto (una persona, un vehículo) con una precisión de hasta centímetros. El sistema fue desarrollado, instalado y empleado por el Departamento de Defensa de los Estados Unidos.

GPS sería uno de los GNSS (Global Navigation Satellite System, en castellano, Sistema Global de Navegación por Satélite) que existen actualmente. GLONASS sería la contrapartida rusa, y GALILEO la europea. Todo GNSS consta de 3 segmentos básicos; el segmento espacial, que se

resumiría en los satélites del sistema en órbita, el segmento de control, y el segmento usuario.



GPS segments.

Ilustración 14. Segmentos del sistema GPS. (Fuente: "GNSS Data Processing", Agencia Espacial Europea)

5.1.5.1.1 Segmento espacial

Las funciones principales del segmento espacial son las de generar y transmitir las señales que recibirá el segmento usuario, necesarias para el cálculo de posición, así como guardar y retransmitir los mensajes de navegación del segmento de control. Estas señales y transmisiones están controladas por relojes atómicos altamente estables que están situados en los satélites.
El segmento espacial está formado por constelaciones de satélites con un número determinado para que al menos haya 4 satélites a la vista desde cualquier punto de la tierra, orbitando a una altura de unos 20200 km.

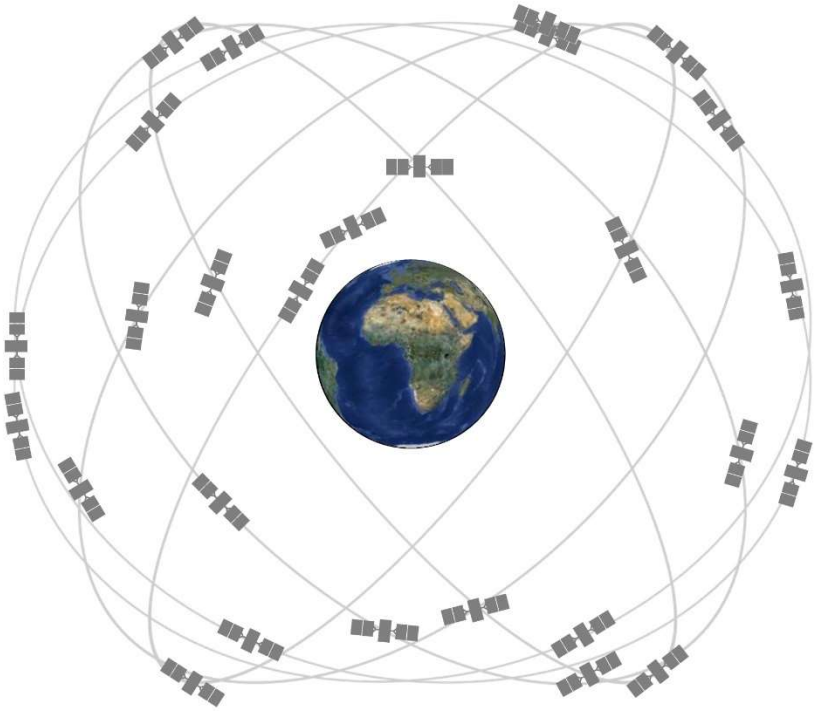


Ilustración 15. Segmento espacial GPS. Red de satélites del sistema. (Fuente: "GNSS Data Processing", Agencia Espacial Europea)

5.1.5.1.2 Segmento de control

El segmento de control de los sistemas GNSS consiste en una red global de instalaciones en tierra que realizan el seguimiento de los satélites GNSS, monitorizan sus transmisiones, ejecutan análisis, y envían comandos y datos a la constelación.

El segmento de control de GPS incluye una estación central maestra, una estación maestra alternativa, 11 antenas de comando y control y 16 sitios de monitorización.

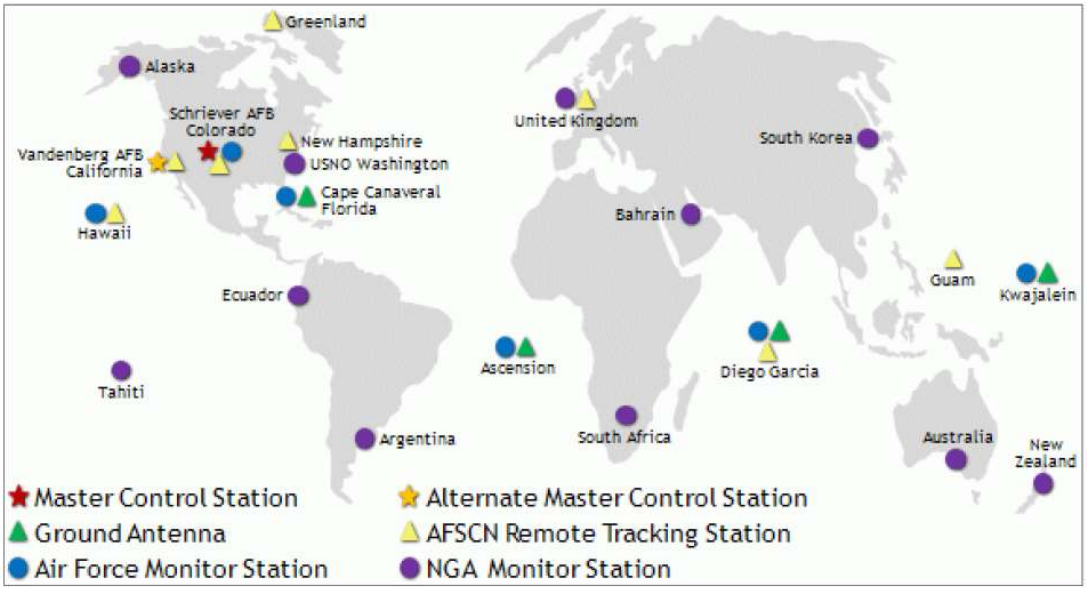


Ilustración 16. Segmento de control de GPS. (Fuente: "GNSS Data Processing", Agencia Espacial Europea)

5.1.5.1.3 Segmento de usuario

Este segmento está compuesto por los receptores GNSS, en este caso receptores GPS. Las funciones principales de este segmento son las de recibir las señales y resolver las ecuaciones de navegación para obtener las coordenadas, entre otras. Los elementos básicos de un receptor genérico GNSS son: una antena con preamplificación, una sección de radiofrecuencia, un microprocesador, un oscilador intermedio, una fuente de alimentación, cierta memoria para almacenar datos y, a veces, un interfaz con el usuario.

5.1.5.1.4 Señales GPS

Las señales GPS son transmitidas en 2 frecuencias en la banda L, que es una banda que va de 1 GHz hasta 2 GHz, y son referidas como Link 1 (L1) y Link 2 (L2). Estas frecuencias están derivadas de una fundamental f_0 = 10,23 MHz, generada por los relojes atómicos a bordo. Tendríamos L1 = 154 * 10,23 MHz = 1575,420 MHz y L2 = 120 * 10,23 MHz = 1227,600 MHz.

En el sistema GPS distinguimos 2 servicios, uno gratuito y público, y otro restringido para usuarios autorizados y entidades militares. Este último sería mucho más preciso que el primero y también estaría fuertemente encriptado. El servicio público se denomina SPS (Standard Positioning Service) y usa simplemente la señal L1, mientras que el sistema restringido utiliza ambas señales y se denomina PPS (Precise Positioning Service). En posteriores modernizaciones se añadió una banda adicional L5.

Finalmente, estas señales estarían compuestas por unos códigos pseudoaleatorios más datos, entre los cuales estarían datos de salud del satélite, el tiempo atómico, su posición en el espacio, etc.

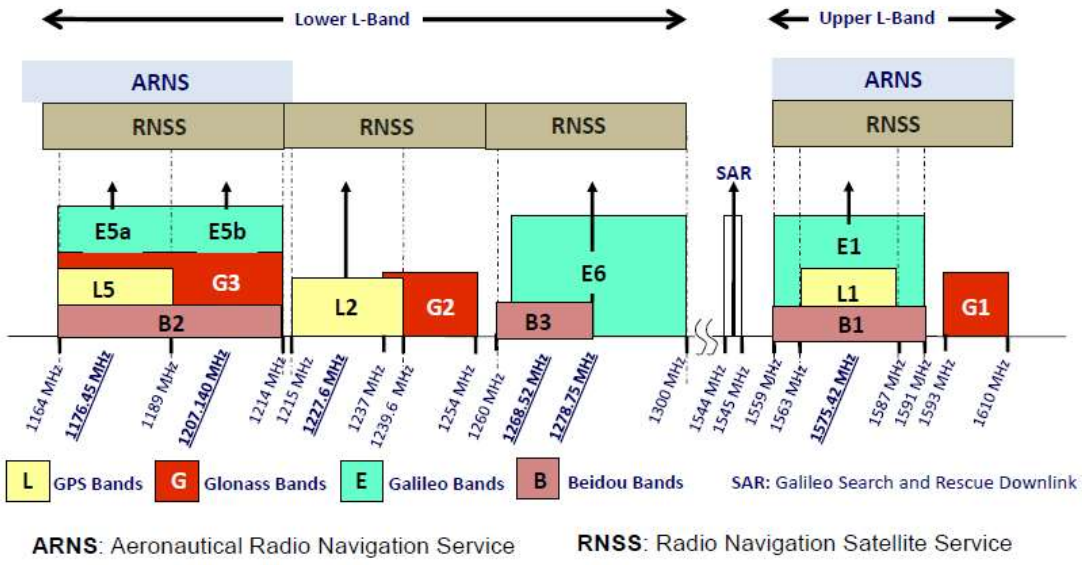


Ilustración 17. Bandas de frecuencias de los sistemas GNSS. (Fuente: "GNSS Data Processing", Agencia Espacial Europea)

5.1.5.1.5 Cálculo de posición

Como se ha indicado, los satélites transmiten su propia posición y la hora de forma precisa, luego el receptor puede estimar la distancia hasta cada satélite mediante la expresión R = c*t, donde R sería la distancia, c la velocidad de la luz y t la diferencia entre la hora recibida y la hora del receptor. Esto es posible gracias a que, pese a la rapidez de las ondas electromagnéticas, a la distancia a la que se encuentran los satélites (unos 20200 km), estas tardan un tiempo en llegar, tiempo que es proporcional a la distancia. Si repetimos este cálculo con al menos 3 satélites, podremos triangular la posición. Sin embargo, nos encontramos con el inconveniente de que, para medir la diferencia de tiempos, en el receptor no contamos con un reloj atómico como aquellos con los que cuentan los satélites, por lo que habrá un error.
Por ello, se toma la hora del usuario como una incógnita, sumándose a las otras tres relativas a la posición en los ejes X, Y y Z. De esta forma, se obtienen 4 incógnitas que necesitarán 4 ecuaciones. Para resolver 4 ecuaciones se necesitan 4 satélites, y se logrará mayor precisión cuanto mayor número de satélites tenga el receptor a la vista.

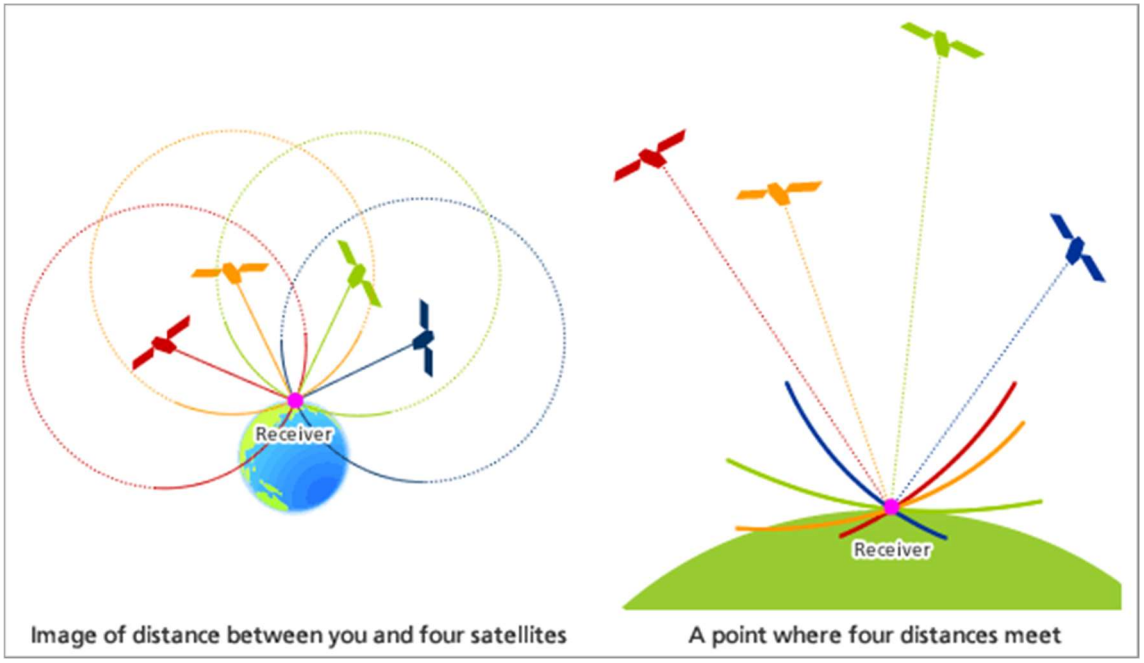


Ilustración 18. Triangulación GPS. (Fuente: "GNSS Data Processing", Agencia Espacial Europea)

5.1.6 Comunicación

Otra de las partes más importantes que constituyen un dron es la comunicación. El conjunto del sistema de un dron consta, normalmente, de un módulo de comunicación que posee el usuario, más hardware de comunicación montado sobre el dron. El módulo del usuario suele ser normalmente un mando "radiocontrol" similar a los usados en coches de radiocontrol u otro tipo de vehículos del estilo. Constaría de 2 "joysticks" o palancas; una para el movimiento de "roll" y "pitch" y otra para controlar la aceleración y el "yaw". También contaría con botones adicionales para otras funciones.

Ilustración 19. Transmisor típico.

En el lado del receptor, es decir, en el dron, normalmente se usan receptores de varios canales para la recepción de cada señal correspondiente. Sin embargo, en modelos más avanzados o con más funcionalidades, se usan transmisores-receptores digitales, para el intercambio de datos más complejos. En este último caso se usarían también protocolos de comunicaciones para dotar de robustez y cierta seguridad a las comunicaciones.

Ilustración 20. Receptor de varios canales.

En nuestro caso, el transmisor será el propio teléfono móvil del usuario, que mediante una sencilla aplicación podrá enviar los comandos necesarios para su funcionamiento. Por otro lado, como receptor necesitaremos un receptor digital, ya que manejaremos datos complejos como coordenadas o comandos.

5.2 Aplicación de smartphone

La forma de enviar las coordenadas GPS del usuario al dron, como se ha tratado con anterioridad, sería mediante una aplicación de smartphone. En estos, el sistema operativo más común y que permite la mayor versatilidad y funcionalidad es Android.

5.2.1 Sistema operativo Android

Android es un sistema operativo basado en el núcleo Linux. Fue diseñado sobre todo para dispositivos móviles con pantalla táctil, como teléfonos inteligentes (smartphones), tabletas y también para relojes inteligentes, televisores e incluso automóviles. Inicialmente fue desarrollado por Android Inc., empresa que Google en principio respaldó económicamente y más tarde, en 2005, compró. Android es el sistema operativo móvil más utilizado del mundo, con una cuota de mercado superior al 80% al año 2017, muy por encima del IOS de Apple.

5.2.2 Aplicaciones Android

Las aplicaciones se desarrollan habitualmente en el lenguaje Java con Android Software Development Kit (Android SDK), pero están disponibles otras herramientas de desarrollo, como un Kit de Desarrollo Nativo para aplicaciones en C o C++, Google App Inventor etc.

El SDK de Android incluye un conjunto de herramientas de desarrollo. Consiste en un depurador de código, biblioteca, un simulador de teléfono, documentación, ejemplos de código y tutoriales. La plataforma integral de desarrollo (IDE, Integrated Development Environment) soportada oficialmente es Android Studio (con el que trabajaremos en este TFG) junto con el complemento ADT (Android Development Tools plugin).

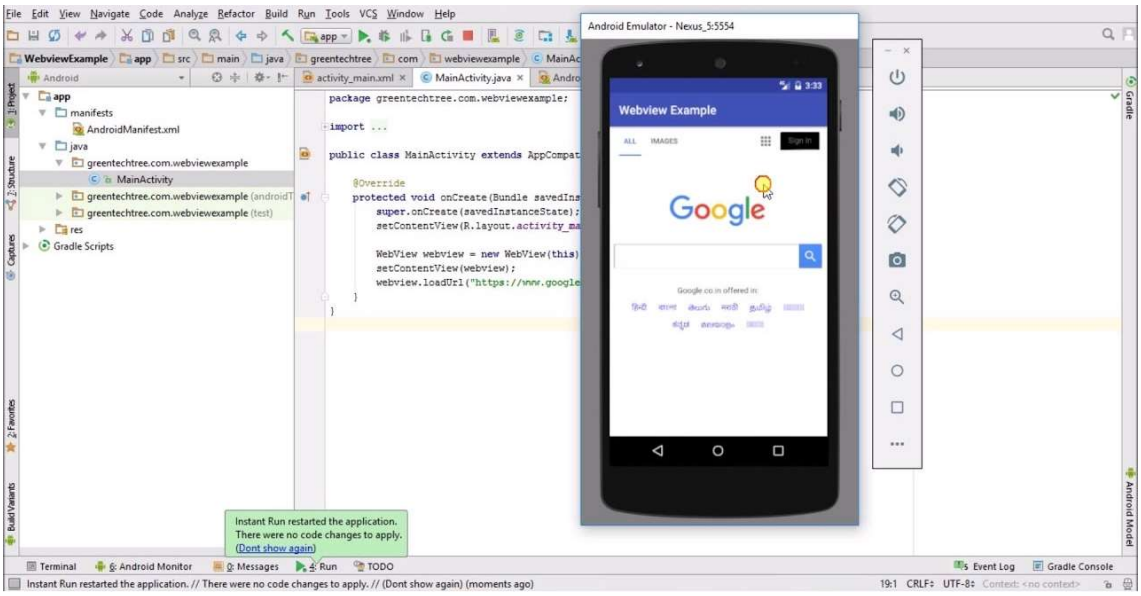


Ilustración 21. Ejemplo de desarrollo en Android Studio.

Las Actualizaciones del SDK se coordinan con el desarrollo general de Android. El SDK soporta también versiones antiguas de Android. De esta forma, si los programadores necesitan instalar aplicaciones en dispositivos ya obsoletos o más antiguos, tendrán lo necesario para hacerlo

Una aplicación Android está compuesta por un conjunto de ficheros empaquetados en formato ".apk" que incluye ficheros ".dex" (ejecutables Dalvik, un código intermedio compilado), recursos, etc.

El desarrollo de aplicaciones para Android no requiere aprender lenguajes relativamente complejos de programación. Todo lo que se necesita es un conocimiento aceptable de Java y estar en posesión del kit de desarrollo de software o SDK provisto por Google el cual se puede descargar gratuitamente.

Todas las aplicaciones están comprimidas en formato APK, que se pueden instalar sin dificultad desde cualquier explorador de archivos en la mayoría de dispositivos.

5.2.3 Java

Al ser Java el lenguaje en el que se desarrollará la App, conviene explicar en qué consiste.

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos, que fue específicamente diseñado para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como WORA, o "write once, run anywhere"), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. Java es uno de los lenguajes de programación más populares, con unos diez millones de usuarios reportados.

Para que todo código sea ejecutable en cualquier plataforma, se hace lo siguiente: se compila el código fuente escrito en lenguaje Java, para generar un código conocido como “bytecode”, que son instrucciones máquina simplificadas específicas de la plataforma Java. Esta pieza está a medio camino entre el código fuente y el código máquina que entiende el dispositivo destino. El bytecode es ejecutado entonces en la máquina virtual (JVM), que es un programa escrito en código nativo de la plataforma destino, que interpreta y ejecuta el código.

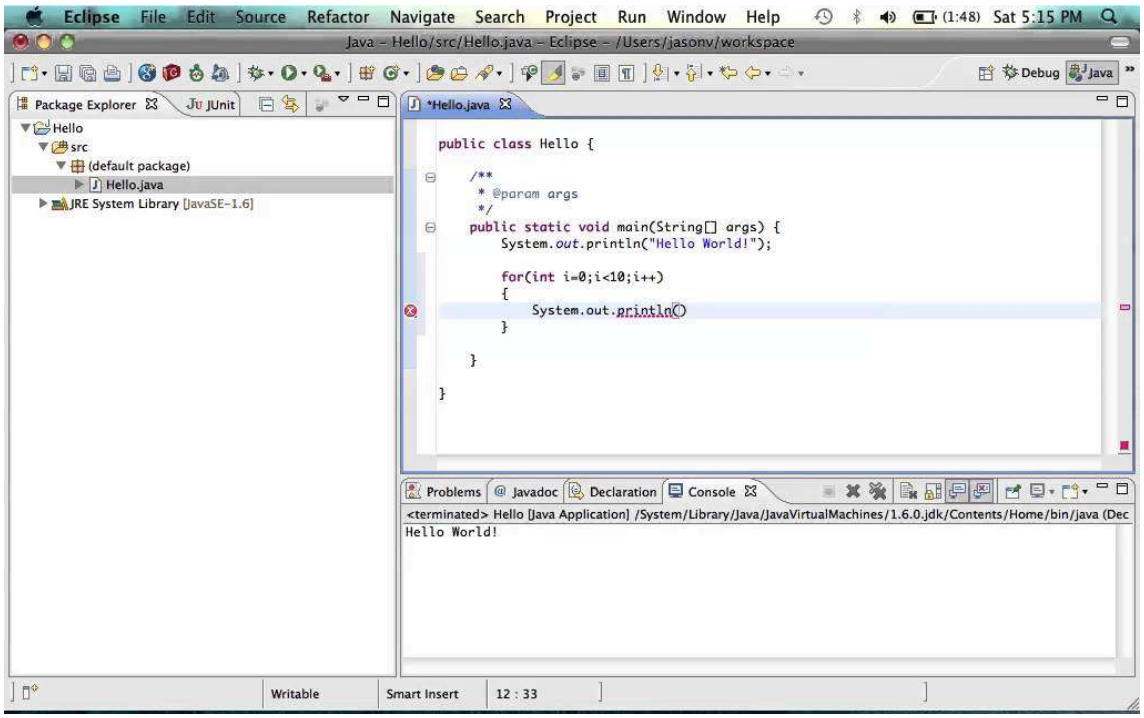


Ilustración 22. Ejemplo de codificación de un "Hello World" en Java en el IDE Eclipse.

5.3 Estructura funcional

Como se ha explicado anteriormente, el sistema completo contaría con una aplicación móvil en el teléfono del usuario, que enviaría constantemente al dron las coordenadas del mismo. En el dron, tendríamos un módulo receptor que recogería estas coordenadas y las entregaría a la unidad de control. Para poder establecer un rumbo concreto, dicha unidad de control recibiría también las coordenadas propias del dron mediante el módulo GPS, y las compararía con las del usuario.

Una vez calculado un rumbo, la unidad de control enviaría las señales correspondientes a los ESCs, para así mover los motores con la velocidad adecuada.

Gracias a la IMU, el dron será estable y se moverá adecuadamente.

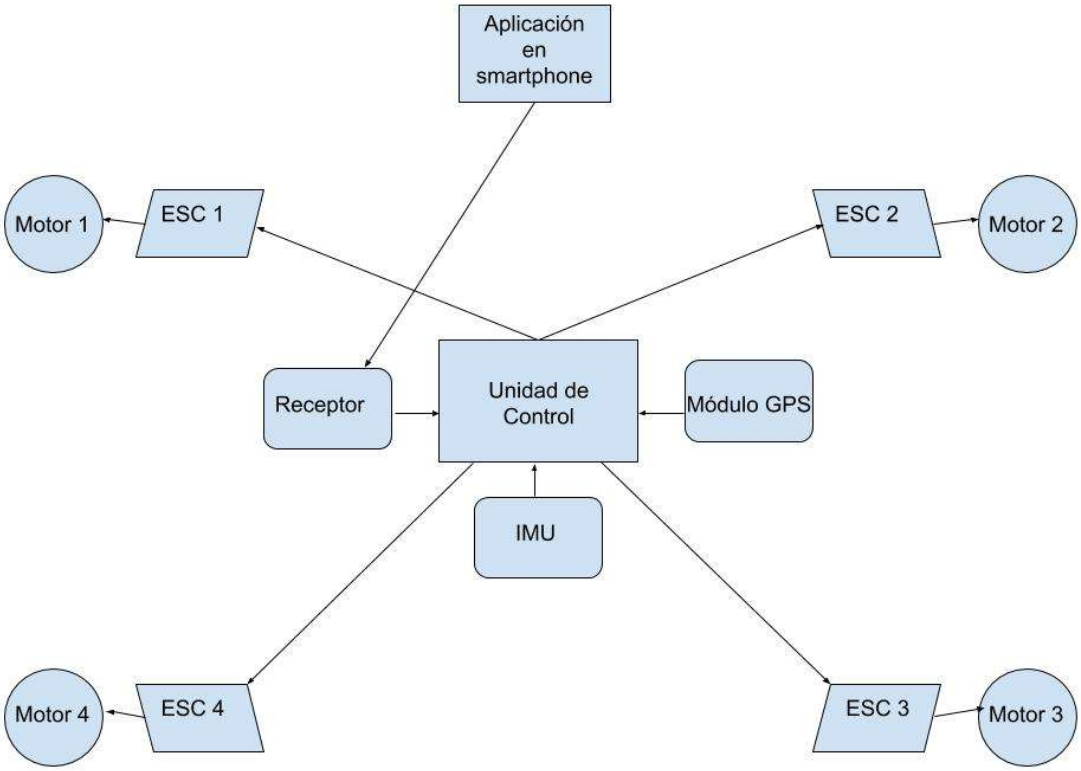


Ilustración 23. Estructura funcional.

6 ANÁLISIS DE ALTERNATIVAS

A la hora de elegir una solución final se deben explorar diferentes alternativas, las cuales se explicarán a continuación.

6.1 Placa microcontroladora

6.1.1 Arduino

Arduino es "una plataforma de prototipos electrónica de código abierto (open-source) basada en hardware y software flexibles y fáciles de usar".
Arduino permite la recepción de entradas desde una variedad de sensores y puede controlar multitud de periféricos. El microcontrolador de la placa se programa en un entorno de desarrollo propio (Arduino IDE) mediante un lenguaje basado en C/C++. Además, dicho IDE se encuentra disponible para casi todas las plataformas (Windows, Linux, Mac).

El hardware Arduino más sencillo consiste en una placa con un microcontrolador y una serie de puertos de entrada y salida y se denominaría "Arduino UNO". Esta opción contiene el microcontrolador "Atmega 328", que posee una velocidad máxima de 20 Mhz. Aunque parezca poco, es suficiente para el proyecto de este TFG e incluso para proyectos más avanzados.

Ilustración 24. Placa Arduino UNO (Fuente: Arduino)

6.1.2 Raspberry Pi

Raspberry Pi es un computador de placa reducida, computador de placa única o computador de placa simple (SBC) de bajo costo. Su software es open source, siendo su sistema operativo oficial una versión adaptada de Debian, denominada Raspbian, aunque permite usar otros sistemas operativos, incluido una versión de Windows 10. Los modelos más potentes contienen un procesador de hasta 1.2 GHz, siendo esta la diferencia más notable con Arduino.

Ilustración 25. Placa Raspberry Pi (Fuente: raspberrypi.org)

6.1.3 Elección final

Una vez presentadas ambas opciones, es hora de seleccionar una de las dos. Hay que señalar que además de estas dos opciones, existen otras tantas basadas también en microcontroladores más otros periféricos, pero sin duda estas dos son las más utilizadas por su versatilidad, y las más adecuadas para el proyecto.

De entre estas dos opciones destacadas, se ha optado finalmente por Arduino, concretamente por el módulo "Arduino UNO", para el dron. Su simplicidad y multitud de periféricos disponibles hacen que la desventaja en potencia respecto a Raspberry Pi sea eludible. Además, se trata de un módulo barato, pero sorprendentemente versátil.

6.2 Comunicación

6.2.1 Comunicación "todo bluetooth"

Como se ha señalado anteriormente, se creará también una aplicación para smartphone que aprovechará las capacidades GPS de éste. De esta forma, simplemente enviaremos las coordenadas GPS del usuario hacia el dron a través de un interfaz radio. Una opción sería establecer una comunicación radio mediante "bluetooth".

6.2.1.1 Bluetooth

Bluetooth es una especificación industrial para "Redes Inalámbricas de Área Personal" (WPAN) creado por Bluetooth Special Interest Group, Inc. Posibilita la transmisión de voz y datos entre distintos dispositivos mediante un enlace por radiofrecuencia en la banda ISM de los 2.4 GHz. Con esta norma se pretende conseguir, entre otras cosas, facilitar las comunicaciones entre equipos móviles, eliminar los cables y conectores entre estos y ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre equipos personales. Esta tecnología tendría un alcance máximo de unos 10 metros.

Los dispositivos que incorporan este protocolo pueden comunicarse entre sí cuando se encuentran dentro de su alcance. Se clasifican como "Clase 1", "Clase 2" o "Clase 3" según su potencia de transmisión.

Tabla 1. Clasificación dispositivos bluetooth en función de la potencia.

Clase	Potencia máxima permitida (mW)	Potencia máxima permitida (dBm)	Alcance (aproximado)
Clase 1	100 mW	20 dBm	~100 metros
Clase 2	2.5 mW	4 dBm	~5-10 metros
Clase 3	1 mW	0 dBm	~1 metro

Los dispositivos con Bluetooth también pueden clasificarse según su capacidad de canal.

Tabla 2. Clasificación dispositivos bluetooth en función de su capacidad de canal.

Versión	**Ancho de banda (BW)**
Versión 1.2	1 Mbit/s
Versión 2.0 + EDR	3 Mbit/s
Versión 3.0 + HS	24 Mbit/s
Versión 4.0	32 Mbit/s

Para utilizar Bluetooth, un dispositivo debe implementar alguno de los perfiles Bluetooth. Un perfil Bluetooth es la especificación de una interfaz de alto nivel para su uso entre dispositivos Bluetooth.

Los perfiles son descripciones de comportamientos generales que los dispositivos pueden utilizar para comunicarse, formalizados para así favorecer un uso unificado, que es el interés principal de esta tecnología. La forma de utilizar las capacidades de Bluetooth se basa, por tanto, en los perfiles que soporta cada dispositivo. Los perfiles permiten la manufactura de dispositivos que se adapten a sus necesidades.

Existen numerosos perfiles bluetooth, que se listan de la siguiente manera.

Lista de perfiles:

- Advanced Audio Distribution Profile (A2DP)
- Audio/Video Remote Control Profile (AVRCP)
- Basic Imaging Profile (BIP)
- Basic Printing Profile (BPP)
- Fax Profile (FAX)
- File Transfer Profile (FTP)
- Hands-Free Profile (HFP)
- Headset Profile (HSP)
- Serial Port Profile (SPP)
- Video Distribution Profile (VDP)
- Wireless Application Protocol Bearer (WAPB)
- Etc…

6.2.1.2 Módulo HC-06

Dado que los smartphones poseen capacidad de comunicación mediante bluetooth, sería interesante aprovecharlo a la hora de comunicarnos desde el teléfono móvil hasta el dron. Para ello, existen diferentes módulos de comunicación bluetooth disponibles para Arduino. El más adecuado, por su sencillez, precio y tamaño, sería el módulo HC-06.

Ilustración 26. Vista frontal y trasera del módulo HC-06. (Fuente: amazon.com)

El módulo HC-06 es un módulo sencillo que actúa como "slave", siendo el módulo HC-05 su contrapartida "master". Como simplemente queremos transmitir datos desde el teléfono móvil hasta nuestro dron, mantendremos el diseño lo más sencillo posible usando un "slave", que sería el HC-06. Posee 4 pines, siendo 2 de ellos para alimentación y los otros 2 para transmisión y recepción serie (Tx y Rx). La alimentación es de 5 V, pero los niveles de tensión de los datos son a 3,3 V. Sobre esto, hay bastante controversia respecto a si se debe o no poner divisores de tensión para acceder a los pines del módulo con Arduino. En base a pruebas realizadas, no parece necesario y no se ve ninguna razón para que a largo plazo puedan surgir problemas, por lo que se conectará directamente los pines al Arduino.

A pesar de todo, nos encontramos con el siguiente problema. Al tratarse de la comunicación con un dron, esta debe ser robusta y tener un buen alcance. Sin embargo, como se ha señalado, mediante bluetooth solo obtendríamos un alcance máximo de como mucho 10 metros. Esto es realmente insuficiente para nuestro proyecto, luego no podemos basarnos en una comunicación totalmente bluetooth.

6.2.2 Comunicación “bluetooth & radio”

Una alternativa para solucionar el problema anterior sería mezclar la sencillez de las comunicaciones bluetooth con la robustez y alcance de las otras comunicaciones vía radio tradicionales. Para conseguir esto, se añadiría un módulo “Arduino UNO” adicional en medio, sirviendo de repetidor entre el teléfono móvil y el dron. De esta forma, usaríamos el módulo HC-06 ya explicado en el lado “smartphone-repetidor”, y un módulo radio adicional en el lado “repetidor-dron”. En el diseño final, el usuario portaría dicho módulo repetidor consigo mismo, pero no tendría que realizar ninguna operación adicional más que el encendido. El resto de operaciones se harían mediante la aplicación para móvil como se ha planteado más atrás.

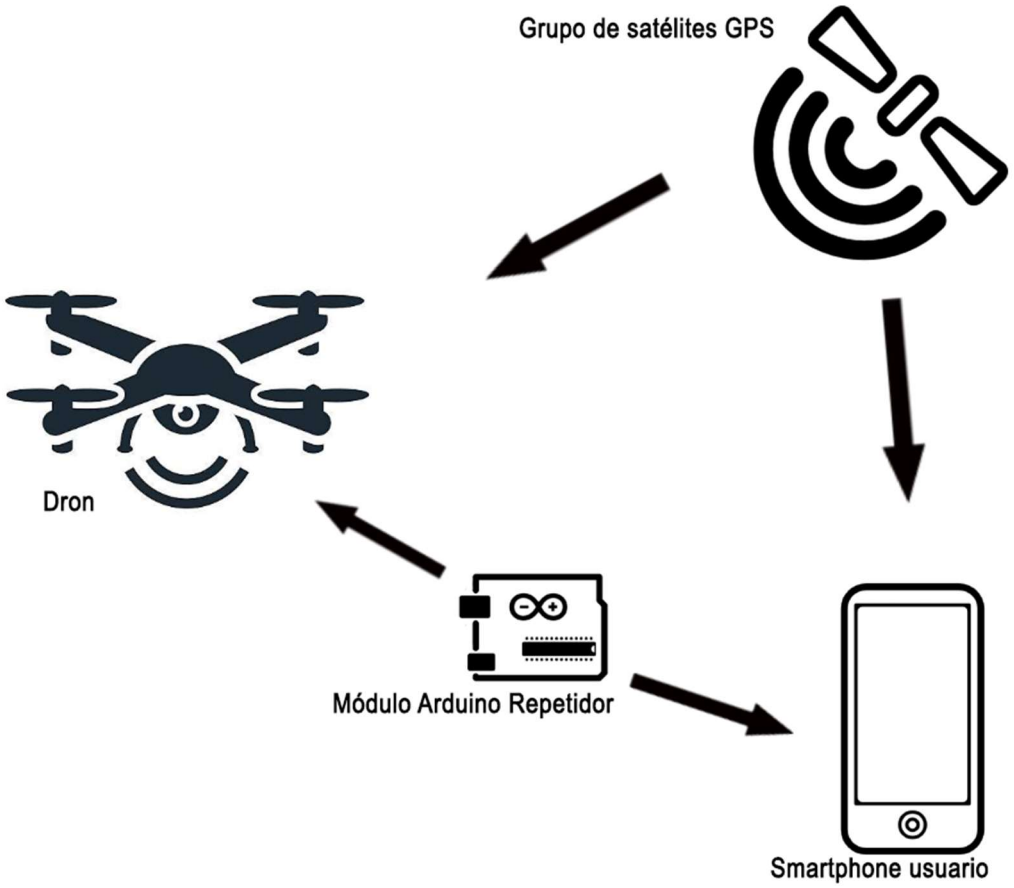


Ilustración 27. Esquema básico de la solución alternativa.

Por ello, pasaremos a explicar varias alternativas radio.

6.2.2.1 Módulos RF 433 MHz

Los módulos de radio frecuencia RF 433 MHz son transmisores/receptores inalámbricos que podemos emplear como forma de comunicación entre procesadores o microcontroladores como Arduino. Este tipo de módulos emisor (FS1000A) y el receptor (XY-MK-5V) se han hecho populares como medio de comunicación, principalmente, por su sorprendente bajo coste.

Operan en una frecuencia de 433 MHz, aunque también existen módulos similares a 315 MHz. Ambas frecuencias pertenecen a bandas libres, por lo que su uso es gratuito y libre. El alcance depende del voltaje con el que alimentemos el módulo y la antena que usemos. Si usamos 5 V y la antena simple del módulo, el alcance difícilmente excederá de los 2 metros. Alimentando a 12 V y con una antena de cobre de 16.5cm el rango en exteriores puede alcanzar 300 metros.
La comunicación es simplex, es decir, usa un canal único y unidireccional, y tienen baja velocidad de transmisión (típicamente 2400bps). La modulación usada es ASK (amplitude shift keying) y no disponen de filtro ni hardware adicional, por lo que si queremos una comunicación robusta tendremos que implementarlo por software.

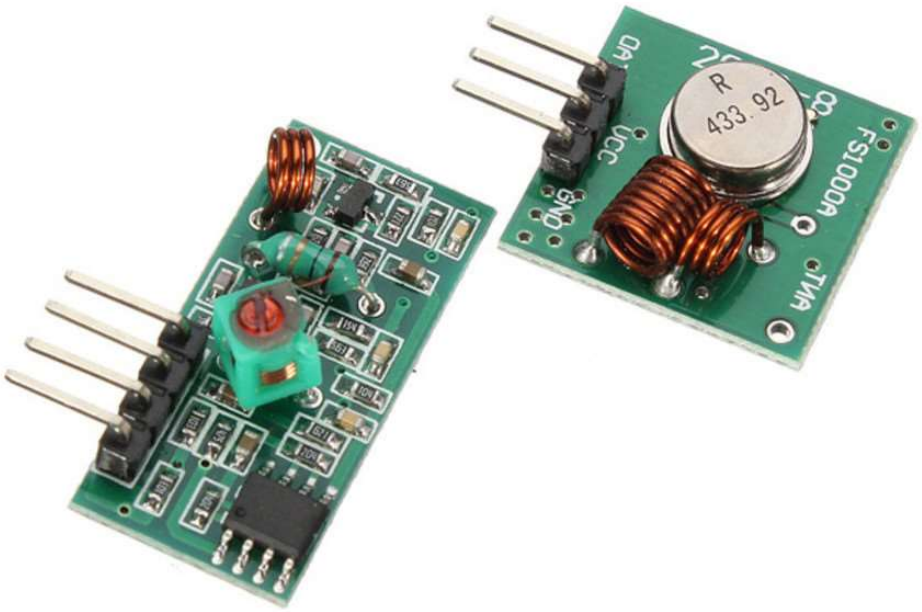


Ilustración 28. Receptor y emisor RF 433 MHz (Fuente: amazon.com)

6.2.2.2 NRF24L01+

El dispositivo NRF2401 integra en un único chip toda la electrónica y bloques funcionales para establecer comunicaciones RF entre dos o más puntos a diferentes velocidades, (hasta 2 Mb/s) con corrección de errores y protocolo de reenvío cuando es necesario, sin intervención de control externo, lo que permite simplificar el diseño y desarrollo de los proyectos. Se trata de un módulo barato y con un alcance relativamente grande sin incluir antenas adicionales (unos 100 m). Además, opera en la banda de 2.4 GHz, por lo que, si se desea añadir una antena adicional para aumentar el alcance, esta es de tamaño reducido. Para añadir, se trata de un módulo de muy bajo consumo y cuenta con librerías propias para Arduino. Hay que añadir que se trata de un transceptor, es decir, tiene capacidades de transmisión y recepción, pero no ambas a la vez.

Ilustración 29. Módulo transceptor NRF24l01+. (Fuente: amazon.com)

Si además se le añade una antena como la de la imagen, el alcance puede aumentarse hasta 1 km. No obstante, con un alcance de 100 m sería suficiente para este proyecto.

6.2.3 Elección final

Finalmente se elige el modelo de comunicación con "repetidor" en medio, puesto que nos permite usar las funcionalidades bluetooth del smartphone del usuario sin comprometer el alcance. Además, no resulta un inconveniente para el usuario llevar el módulo Arduino adicional, puesto que es de tamaño reducido y compacto, y no debe realizar ninguna operación complementaria a las de la aplicación móvil. Para la sección radio se elige el módulo NRF2401 por su alcance y bajo consumo, sumados a su simplicidad tanto en diseño como en uso. En la sección bluetooth, se usará el módulo HC-06.

6.3 Motores

6.3.1 Elección final

Una vez tenidos en cuenta los pros y contras de los dos tipos de motores, se decide emplear un motor bushless, ya que pueden alcanzar velocidades mayores y el mantenimiento resulta menor que en el caso de los motores brushed.

Concretamente se ha elegido el motor “A2212 920KV Brushless Motor”, que tiene una eficiencia del 80%.

No se ha hablado de otro aspecto importante, que serían las hélices o los “propellers”. En este caso, el motor vendría con los propellers “1045 Propeller DJI F450 – Black”.

Ilustración 30. Motor "A2212 920KV Brushless Motor" (Fuente: amazon.com)

6.4 IMU

6.4.1 MPU-6050

Para el desarrollo de este TFG se usará la IMU MPU-6050, que consta de un acelerómetro y un giroscopio. Este dispositivo, es una IMU de seis grados de libertad (6DOF) porque combina un acelerómetro de 3 ejes y un giroscopio de también 3 ejes. El rango del acelerómetro puede ser ajustado a ±2g, ±4g, ±8g y ±16g, y el del giroscopio a ±250 °/s, ±500 °/s, ±1000 °/s y ±2000 °/s. Se trata de un componente relativamente sencillo de usar y de comunicar con nuestro microcontrolador, además de tener un precio muy asequible.

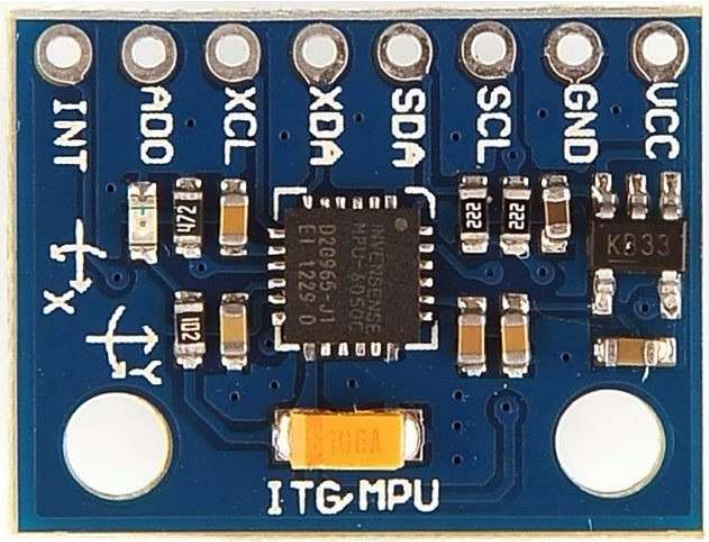


Ilustración 31. MPU-6050. (Fuente: amazon.com)

6.5 ESCs

En este caso, se va a hacer uso del ESC: "30A SimonK ESC (with BEC) for RC Quadcotper", dado que normalmente viene incluido con el motor elegido "A2212 920KV Brushless Motor". Por tanto, no se entrará en detalles.

6.6 Módulo GPS

6.6.1 GY-NEO6MV2

Anteriormente se ha explicado cómo funciona el sistema GPS, por lo que a continuación se explicará el módulo elegido.

De entre infinidad de alternativas para dotar a nuestro dron de capacidad de geoposicionamiento, se ha elegido el módulo GY-NEO6MV2. Se trata de un geolocalizador muy compacto, económico y de muy bajo consumo, puesto que solamente requiere de 3V para su correcto funcionamiento, aunque se le puede alimentar con la salida de 3,3 V que Arduino nos ofrece. Con este módulo se obtienen las coordenadas, en grados, de latitud y longitud, el valor en metros de la altitud a la que nos encontramos, y como plus también se obtiene la hora internacional en formato de 24 horas que el satélite devuelve. El sistema necesita de un cierto tiempo para sincronizarse con los satélites y para su correcto funcionamiento es muy importante utilizarlo en zonas exteriores, algo que no sería problemático para este proyecto, dado que se trata de un dron que volará en exteriores.
El módulo GPS GY-NEO6MV2 viene con una antena de cerámica que permite amplificar las señales recibidas de los satélites con los que se comunica.

Ilustración 32. Módulo GPS gy-neo6mv2. (Fuente: amazon.com)

7 Descripción de la solución propuesta.

7.1 Hardware

7.1.1 Dron

Una vez elegida la arquitectura final y cada componente, se procede a describir la propuesta en su conjunto. A continuación, se presenta el esquema electrónico correspondiente al dron.

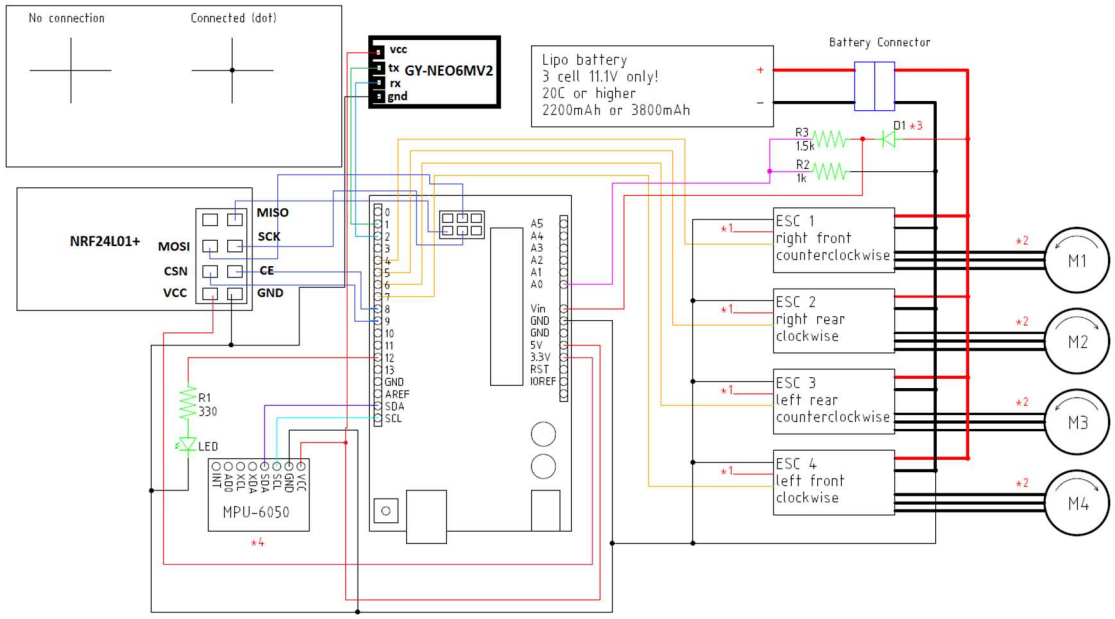


Ilustración 33. Esquema electrónico Dron

Se pueden apreciar los 4 motores conectados a sus respectivos ESCs, y estos a su vez a diferentes pines digitales de la placa Arduino UNO (4,5,6 y 7). Además, tendríamos el módulo GPS GY-NEO6MV2 con sus pines Tx y Rx conectados a los pines digitales 1 y 2 respectivamente, y también la IMU MPU-6050 con sus pines SCL y SDA a los pines SCL y SDA de Arduino.

Por otro lado, tendríamos el módulo transceptor NRF24L01+, cuya conexión es ligeramente especial. Para conectar este módulo es posible usar simplemente pines digitales, pero dada la cantidad de periféricos y pines siendo usados en este proyecto, se hará de una segunda forma.

El módulo NRF24L01+ es capaz de comunicarse mediante el bus SPI, que es un estándar de comunicaciones usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos. El “bus de interfaz de periféricos serie” o bus SPI es un estándar para controlar prácticamente cualquier dispositivo electrónico digital que acepte un flujo de bits serie.

Incluye una línea de reloj, dato entrante, dato saliente y un pin de chip select, que conecta o desconecta la operación del dispositivo con el que uno desea comunicarse, por lo que existe multiplexación. Las ventajas de un bus serie es que se minimizan el número de conductores, pines y el tamaño del circuito integrado. Esto reduce el coste de fabricar, montar y probar la electrónica.

El SPI es un protocolo síncrono. La sincronización y la transmisión de datos se realiza por medio de 4 señales:

- SCLK (Clock): Es el pulso que marca la sincronización. Con cada pulso de este reloj, se lee o se envía un bit. También llamado TAKT (en alemán).
- MOSI (Master Output Slave Input): Salida de datos del Master y entrada de datos al Slave. También llamada SIMO.
- MISO (Master Input Slave Output): Salida de datos del Slave y entrada al Master. También conocida por SOMI.
- SS/Select: Para seleccionar un Slave, o para que el Master le diga al Slave que se active. También llamada SSTE.

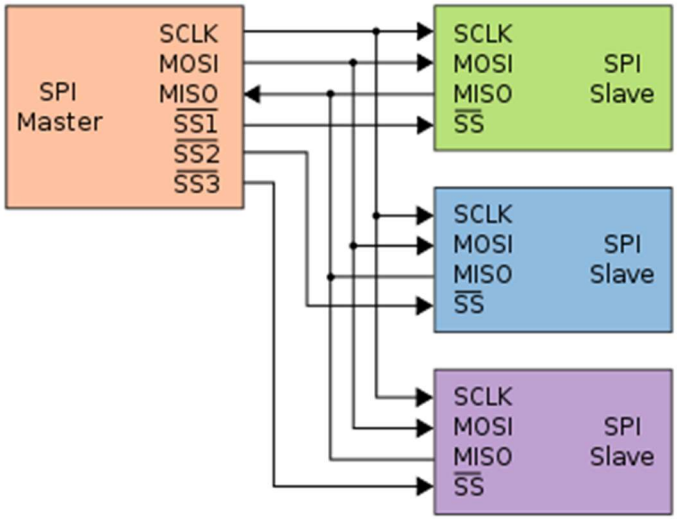


Ilustración 34. Ejemplo de Bus SPI. 1 maestro y 3 esclavos. (Fuente: Wikipedia)

Por otro lado, la placa Arduino posee una entrada ICSP (In Chip Serial Programmer) que tiene acceso a la memoria de programa del AVR (Flash) para poder grabar directamente desde el PC al microcontrolador cualquier programa sin usar el puerto USB. Uno de ellos, el mismo Bootloader de Arduino. ICSP es un conector consistente en 6 señales: MOSI, MISO, SCK, RESET, VCC, GND y además de ser un puerto para programar Arduino, también es el conector de expansión del bus SPI mediante el que podremos comunicarnos con el transceptor NRF24L01+. Se puede considerar el ICSP como un "esclavo" del master del bus SPI del microcontrolador.

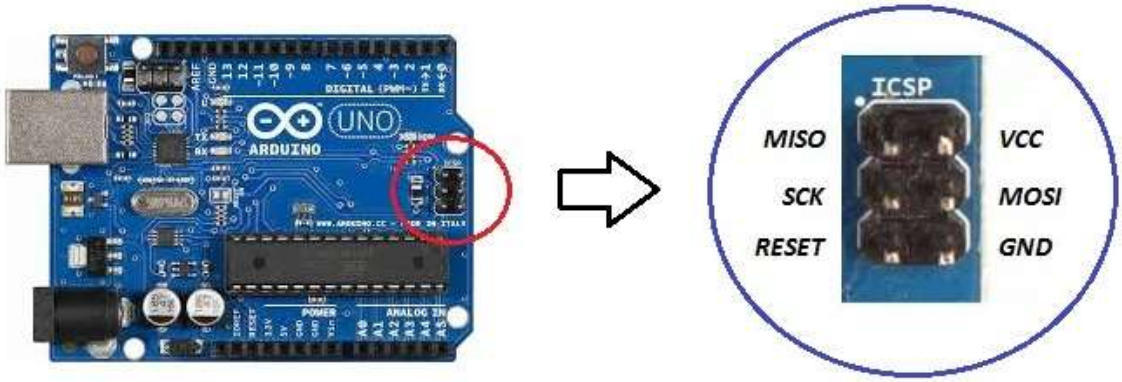


Ilustración 35. Conector ICSP en Arduino UNO

A parte, conectaremos los pines CE y CSN del NRF24L01+ a los pines digitales 8 y 9 del Arduino UNO.

A la hora de las alimentaciones, hay que señalar que el transceptor NRF24L01+ se alimenta con la salida de 3,3 V, mientras que el módulo GPS GY-NEO6MV2, a pesar de necesitar también alimentación a 3,3 V, es posible alimentarlo a 5 V dado que cuenta con un regulador integrado. Por lo tanto, el módulo NRF24L01+ se conectará al pin de alimentación de 3,3 V del Arduino UNO, y el módulo GY-NEO6MV2 junto con la IMU MPU-6050 se conectarán al pin de alimentación de 5 V. En cambio, al necesitar un voltaje y potencia mayor, los motores se alimentarán mediante una batería de 11,1 V.

Mediante esa batería también se alimentará el Arduino UNO. Para ello, usaremos además un divisor de tensión que permitirá, al software de control, conocer la tensión de alimentación en todo momento. Esto es necesario ya que, durante el vuelo, la batería se descargará progresivamente, provocando una caída de potencia en los motores que es necesario compensar. Con el divisor de tensión, conseguiremos una señal de no más de 5 V en la entrada analógica A0 del Arduino UNO. Como protección, tendremos un diodo para controlar la dirección de la corriente.

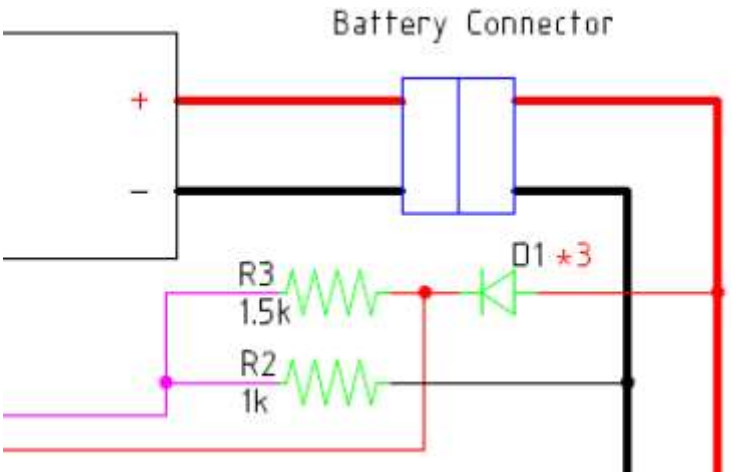


Ilustración 36. Detalle del divisor de tensión y diodo.

Adicionalmente se incluirá un LED conectado al pin digital 12 para indicar el encendido y otra serie de señales.

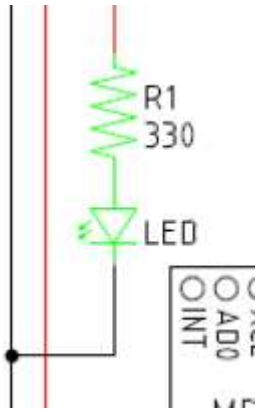


Ilustración 37. Detalle del LED.

7.1.2 Módulo repetidor

Como se ha indicado anteriormente, para solucionar el problema del alcance limitado del bluetooth, se añadiría un módulo "Arduino UNO" adicional en medio, sirviendo de repetidor entre el teléfono móvil y el dron. Usaríamos el módulo HC-06 ya explicado en el lado "smartphone-repetidor", y un módulo radio adicional en el lado "repetidor-dron". El usuario portaría dicho módulo repetidor consigo mismo, sin realizar ninguna operación adicional más que el encendido. El resto de operaciones se harían mediante la aplicación ubicada en el teléfono móvil.

Así pues, el esquemático del módulo intermedio repetidor sería el siguiente.

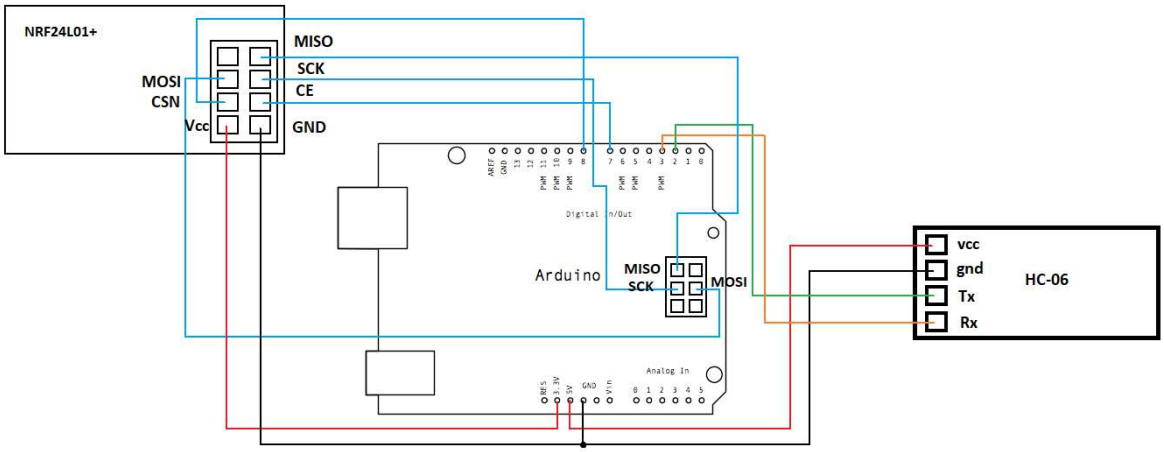


Ilustración 38. Esquemático módulo repetidor.

La conexión del NRF24L01+ sería igual que en el dron, salvo porque en este caso los pines CE y CSN estarían conectados a los pines 7 y 8 del Arduino respectivamente. El módulo HC-06 se conectaría de la siguiente forma: pin TX al pin 2 del Arduino y pin RX al pin 3. Las alimentaciones serían de 5 V para el caso del HC-06 y de 3.3 V para el NRF24L01+. Este módulo intermedio puede ser alimentado por medio de una pila de 9 V, dado que cuenta con regulador.

7.2 Software

Se ha desarrollado dos softwares que se comunican entre sí. Uno sería la aplicación para smartphone que llevaría el usuario, y otro sería el software de control instalado en el dron. Para comunicarse se ha creado un protocolo simple basado en strings (cadenas de caracteres), mediante los cuales se transmiten comandos para realizar acciones, o se envían las coordenadas del usuario.

7.2.1 Protocolo de comunicación

Para enviar un comando, la aplicación de smartphone enviará "COMANDO%", y, a continuación del símbolo de porcentaje, el comando deseado. De entre las acciones disponibles tendremos "ENCENDIDO", para encender el dron, "EMPEZAR", para despegar y comenzar el seguimiento si el dron está en el suelo, o para seguir con el seguimiento si está detenido en el aire, "DETENER", para detener el seguimiento, y "APAGADO" para aterrizar. Al final de cada string se añade el símbolo "#" para indicar el fin del mismo. Por ejemplo, si tenemos el dron en el suelo y queremos comenzar el seguimiento, se enviará lo siguiente: COMANDO%EMPEZAR#

Mientras tanto, la aplicación enviará constantemente las coordenadas del usuario mediante "GEOLOCA%" y las coordenadas después del símbolo de porcentaje. Esto se repetirá cada segundo, dado que, a la velocidad media de una persona cualquiera, no será necesario un refresco de la posición mucho más rápido.

Tabla 3. Combinaciones posibles del protocolo.

TIPO	**DATO**			
COMANDO	ENCENDIDO	EMPEZAR	DETENER	APAGADO
GEOLOCA	*-COORDENADAS-*			

Para una mejor explicación, se expondrá a continuación la codificación del envío y recepción del comando "ENCENDIDO", en lenguaje Java y Arduino respectivamente.

```
IdListo.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {



        MyConexionBT.write("COMANDO%ENCENDIDO#");
        //envio comando para inicialización del dron
        //con esto sale del blucle del setup
    }
});
```

Ilustración 39. Envío de comando "ENCENDIDO".

```
if (radio.available())
  {
   radio.read(&text, sizeof(text));
   emp=strcmp(text,"COMANDO%ENCENDIDO#");
   strcpy(text,"");
  }
```

Ilustración 40. Recepción de comando "ENCENDIDO".

Seguidamente, se mostrará como ejemplo la interpretación de, en este caso, el comando "APAGADO" una vez recibido.

```
emp=strcmp(text,"COMANDO%APAGADO#");
if(emp==0){
  start = 0; //estado aterrizado
  aterriza(); //función de aterrizaje
  strcpy(text,""); //limpio text
```

Ilustración 41. Interpretación del comando "APAGADO".

Como puede verse, una vez recibido cierto comando, se pasa a comprobar cuál es. En este caso, si se recibe "APAGADO", se realizan las acciones correspondientes, de entre las cuales la más importante sería la del aterrizaje del dron.

El resto de la codificación para cada comando se va a obviar puesto que es similar o prácticamente igual.

7.2.2 Envío y recepción de coordenadas GPS

Como se ha explicado con anterioridad, la aplicación de smartphone enviará cíclicamente las coordenadas GPS del usuario al dron, para así realizar el seguimiento. Mientras tanto, la aplicación tiene que estar atenta a las posibles acciones del usuario, es decir, a si el usuario pulsa alguno de los botones presentes en la pantalla para detener el seguimiento autónomo, reanudarlo, aterrizar el dron, etc. Para ello, se crea un thread o hilo que se ejecutará de forma paralela y se encargará de obtener y después enviar las coordenadas GPS del usuario.

```
private class enviaGPS extends AsyncTask<Void, Void, Void> {

    @Override
    protected Void doInBackground(Void... params) {

        while (true){


            String Text = "GEOLOCA%"+ loc1.getLatitude() + "%" + loc1.getLongitude()+"#";

            MyConexionBT.write(Text);

            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            if(isCancelled())
                break;
        }

        return null;
    }
```

Ilustración 42. Envío de coordenadas desde aplicación.

En recepción, es decir, en el dron, se detecta la llegada de coordenadas y se llama a una función especial para su manipulación.

```
else if(start==2){
  emp=strncmp(text,"GEOLOCA%",8);
  if(emp==0){
    //llamar a funcion de geoloc
      calculaGPS();
    strcpy(text,"");
```

Ilustración 43. Llegada de las coordenadas y llamada a función especial.

7.2.3 Cálculo de las diferencias entre coordenadas del dron y del usuario

Una vez que el dron conoce la posición del usuario, debe conocer la suya propia y hallar la diferencia entre ambas. Además, a esta diferencia se le sumará un offset para que el dron siempre se mantenga a cierta distancia del usuario. Este offset dependerá de la posición del dron respecto al usuario.

```
void calculaGPS(){

  strcpy(gpsLOC,&text[8]);

  if(gps.available())
  {
     char clat[20];
     char clon[20];
     char c = gps.read(); //obtenemos el stream de datos del módulo gps
     if (gpsd.encode(c)){  //usamos la librería para traducirlo
      gpsd.f_get_position(&flat, &flon, &age); //obtenemos los datos principales
     }
     // lat y lon vienen juntas separadas por el token "%" desde la app android
     strcpy(clat,strtok(gpsLOC,"%")); //obtenemos los datos del usuario
     strcpy(clon,strtok(NULL,"#"));
     flatu=atof(clat);  //se transforman a floats
     flonu=atof(clon);  //para poder manejarlos
     float difLat,difLon;
     difLat=flat-flatu; //hallamos las diferencias
     difLon=flon-flonu;
```

Ilustración 44. Cálculo de diferencias entre coordenadas.

```
if(difLon<0){ //incluimos un offset
 difLon+=0.00004;
 } //serían como 4 metros
else{
 difLon-=0.00004;
 }

 if(difLat<0){ //el offset
 difLat+=0.00004;
 } //serían como 4 metros
else{
 difLat-=0.00004;
 }
```

Ilustración 45. Cálculo y suma del offset.

Para explicar cómo todo eso se transforma en movimiento, es decir, en un rumbo para realizar el seguimiento, primero se debe explicar de forma más profunda otra serie de cuestiones, como, por ejemplo, cómo se mueve y en qué consiste el código para el movimiento y estabilidad del dron.

7.2.4 Control del dron

Hoy en día la gran mayoría de drones se controlan mediante un controlador PID. Por lo tanto, se debe explicar en qué consiste y cuál es su importancia, además de la razón de peso, que es su uso en este TFG.

7.2.4.1 Controlador PID

Un controlador PID es un mecanismo de control por realimentación ampliamente usado en sistemas de control industrial, que calcula la desviación o error entre un valor medido y un valor deseado. Este tipo de controladores han demostrado ser bastante robustos y es por ello que son utilizados en más del 95% de los procesos industriales en lazo cerrado (que son procesos en los que la salida depende de las consideraciones y correcciones realizadas por la retroalimentación).

El algoritmo del control PID está formado por tres parámetros: el proporcional, el integral, y el derivativo. El primero de ellos, el proporcional, depende del error actual, mientras que el integral depende de valores pasados y, por último, el derivativo es una predicción de los errores futuros. Mediante el ajuste de estas tres variables en el algoritmo de control del PID, el controlador puede realizar una acción de control diseñada para el proceso específico.

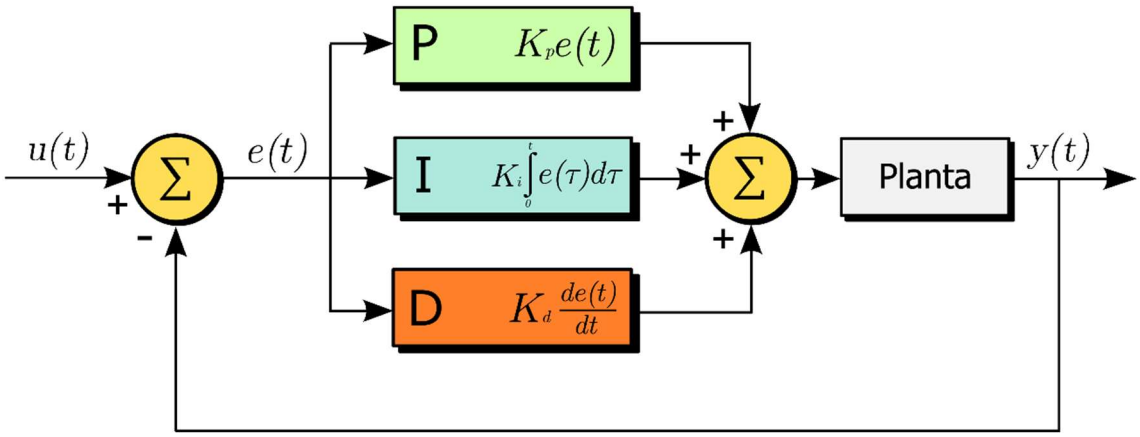


Ilustración 46. Esquema del algoritmo del PID.

Como un sistema PID consiste en que en un proceso se detecte algo para que el controlador actúe en el sistema (todo esto de forma repetida, produciéndose la realimentación), necesitaremos tres componentes. Estos serían: un sensor, un controlador y un actuador.

El sensor determinaría el estado del sistema, mientras que el controlador generaría la señal que gobierna el actuador en función de lo que hubiera detectado el sensor.

7.2.4.1.1 Proporcional

El valor proporcional es el producto entre la señal de error y la constante proporcional o constante de proporcionalidad. Al no considerar el tiempo, la mejor manera de solucionar el error permanente, es incluyendo las variables integral y derivativa. La constante de proporcionalidad, se puede ajustar como el valor de la ganancia del controlador.

En el caso del dron, se podría decir que es el valor encargado de la estabilidad y control. Es el valor más importante e indica el nivel de corrección necesario a aplicar. Mientras más grande sea P, más intentará estabilizarse el dron, pero si se excede en el valor, el aparato se volverá demasiado sensible y provocará oscilaciones incontrolables y exponenciales.

$$P_{\text{sal}} = K_p\, e(t)$$

Ilustración 47. Expresión del valor proporcional.

7.2.4.1.2 Integral

El valor integral es proporcional tanto a la magnitud del error, como a la duración del mismo, y se conoce como la suma de los errores en el tiempo. Indica el número de errores que tenían que haberse corregido previamente, y se multiplica por la constante Ki. Lo que hará la I a grandes rasgos es hacer más progresivo el movimiento de retorno a estabilidad determinado por P.

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

Ilustración 48. Expresión del valor integral.

7.2.4.1.3 Derivativo

Esta variable aparece cuando hay un cambio en el valor absoluto del error (si el error fuera constante, solamente actuarían los modos proporcional e integral). La función de la variable derivativa es mantener el error al mínimo corrigiéndolo proporcionalmente con la misma velocidad que se produce. De esta forma se evita que el error incremente.

$$D_{\text{sal}} = K_d \frac{de}{dt}$$

Ilustración 49. Expresión del valor derivativo.

Las salidas de estos tres términos son sumadas para calcular la salida del controlador PID. Si definimos y(t) como la salida del controlador, la forma final sería:

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

Ilustración 50. Expresión final del algoritmo PID.

7.2.4.2 Implementación del PID en Arduino

Una vez explicado en qué consiste un controlador PID, se debe mostrar cómo se ha implementado en este proyecto en concreto.

Un PID se usa en un dron para mantener su estabilidad y controlar el movimiento. Para ello, existirían 3 PID correspondientes a los 3 posibles movimientos (roll, pitch y yaw). El sensor en un dron correspondería a la IMU, y los actuadores corresponderían al conjunto ESCs-motores. De esta forma, mediante el giroscopio y el acelerómetro de la IMU, detectaríamos movimiento

en alguna de las 3 posibilidades, y el PID de cada una se encargaría de corregirlo.

Hay que decir que más que una corrección del movimiento, el PID se usa para corregir el error entre el estado actual (el "movimiento" del dron) y la orden de control, que puede ser o mantener el dron en esa posición (es decir, cero movimiento en roll, pitch y yaw), o cualquier rumbo deseado (en este caso al menos el pitch o el roll deseados no serán cero).

Como ejemplo, se mostrará a continuación cómo se ha implementado en Arduino la codificación del PID correspondiente al roll:

```
//Roll calculations
pid_error_temp = gyro_roll_input - pid_roll_setpoint;
pid_i_mem_roll += pid_i_gain_roll * pid_error_temp;
if(pid_i_mem_roll > pid_max_roll)pid_i_mem_roll = pid_max_roll;
else if(pid_i_mem_roll < pid_max_roll * -1)pid_i_mem_roll = pid_max_roll * -1;
pid_output_roll = pid_p_gain_roll * pid_error_temp + pid_i_mem_roll + pid_d_gain_roll *
(pid_error_temp - pid_last_roll_d_error);
if(pid_output_roll > pid_max_roll)pid_output_roll = pid_max_roll;
else if(pid_output_roll < pid_max_roll * -1)pid_output_roll = pid_max_roll * -1;
pid_last_roll_d_error = pid_error_temp;
```

Ilustración 51. Ejemplo de implementación del PID del roll.

En la primera línea se calcula el error entre el estado actual (gyro_roll_input) y la orden de control (pid_roll_setpoint). Esto se guarda en la variable pid_error_temp. Después, calcularíamos la variable integral (pid_i_mem_roll), que como sabemos es el producto entre el error y la ganancia I pero acumulándose en el tiempo (de ahí que se sume a la variable con cada ejecución). Más tarde, en una sola línea, se calcularía la salida PID del roll, sumando directamente las 3 componentes; la componente proporcional, que como sabemos es el producto entre el error y la ganancia P, la componente I ya calculada, y la componente D, que sería la ganancia D por la diferencia entre errores (el error actual menos el anterior).

Puede parecer algo complicado, pero no es más que la implementación en código de las expresiones ya explicadas y teniendo en cuenta que trabajamos con valores discretos tomados en momentos también discretos.

Para los casos de roll y yaw, la explicación se obvia, dado que sería exactamente igual.

7.2.5 Creación del rumbo

A continuación, se explicará cómo se traduce la diferencia de coordenadas entre el usuario y el dron, es decir, la posición del dron respecto al usuario, para conseguir el seguimiento autónomo creando el movimiento adecuado para el dron.

Se ha explicado cuál es el mecanismo de control de un dron y también cómo obtenemos la posición del dron respecto al usuario, por lo que solo queda explicar cómo se ha conseguido relacionarlos.

Y es que, si nos fijamos en el problema, vemos que es similar al del control del dron en sí mismo. Tenemos un estado actual (posición del dron) y una orden de control (posición del usuario, hacia donde nos interesa ir), por lo tanto, la diferencia de posiciones ya calculada sería simplemente la diferencia o el error en un sistema PID. En estos sistemas, como ya se ha explicado, se pretende que dicho error sea cero, mediante correcciones proporcionales, integrales y/o

derivativas. Aquí, lo que nos interesa es que la posición del dron y la del usuario sea siempre la misma (sin contar con el offset ya añadido y explicado con anterioridad).

Teniendo todo esto en cuenta, la codificación será igual que para el caso del roll, pitch y yaw.

```
//PID LAT
pid_i_mem_lat += pid_i_gain_lat * difLat;
if(pid_i_mem_lat > pid_max_lat)pid_i_mem_lat = pid_max_lat;
else if(pid_i_mem_lat < pid_max_lat * -1)pid_i_mem_lat = pid_max_lat * -1;

pid_output_lat = pid_p_gain_lat * difLat + pid_i_mem_lat + pid_d_gain_lat
* (difLat - pid_last_lat_d_error);
if(pid_output_lat > pid_max_lat)pid_output_lat = pid_max_lat;
else if(pid_output_lat < pid_max_lat * -1)pid_output_lat = pid_max_lat * -1;


pid_last_lat_d_error = difLat;
```

Ilustración 52. PID para la latitud.

Una vez que tenemos outputs del PID de la latitud y de la longitud, debemos asignar esas correcciones de posición a los setpoints del pitch y el roll, respectivamente. Los setpoints serían las señales de control de los PID correspondientes.

```
pid_roll_setpoint=pid_output_lon;
pid_pitch_setpoint=pid_output_lat;
```

Ilustración 53. Asignación de los setpoints.

De esta forma, ya habríamos traducido esa diferencia de posiciones en movimiento para corregirla.

Por último, los outputs de los PIDs del roll, pitch y yaw, se sumarían con signos determinados para conseguir las señales de salida hacia los ESCs, que controlarán los motores para que finalmente el dron se mueva de forma adecuada.

```
esc_1 = throttle - pid_output_pitch + pid_output_roll - pid_output_yaw;
esc_2 = throttle + pid_output_pitch + pid_output_roll + pid_output_yaw;
esc_3 = throttle + pid_output_pitch - pid_output_roll - pid_output_yaw;
esc_4 = throttle - pid_output_pitch - pid_output_roll + pid_output_yaw;
```

Ilustración 54. Señales finales de los ESCs.

Estas expresiones se entienden de la siguiente manera: si los motores delanteros han de sumar el valor de salida del eje de cabeceo, los motores traseros deberán restarlo. Si a su vez el motor delantero-izquierdo debe sumar el valor de salida del PID que controla el alabeo, el motor delantero-derecho deberá restarlo. Lo mismo ocurre en el caso de la guiñada.

7.2.6 Interfaz de usuario

Todo lo ya explicado sería transparente para el usuario, puesto que este contaría con la aplicación de smartphone ya mencionada y solo tendría que hacer uso de su interfaz para controlar el dron.

Ilustración 55. Interfaz aplicación.

Como se puede observar en la imagen, la interfaz contaría con 5 botones. Para encender el dron, el usuario apretaría "LISTO". Después, para comenzar el seguimiento, pulsaría "COMENZAR SEGUIMIENTO". Para detener el seguimiento, "DETENER SEGUIMIENTO". En este caso, podría volver a pulsar "COMENZAR SEGUIMIENTO" para reanudarlo, o "APAGAR", para que el dron aterrice finalmente. El sistema está provisto de seguridad de tal forma que cualquier combinación de pulsaciones que no concuerde con la explicada, no generará eventos o acciones no deseadas.

Una vez que el dron esté en el suelo tras pulsar "APAGADO", el usuario pulsaría "DESCONECTAR" y la aplicación podrá cerrarse.

7.2.7 Módulo Arduino intermedio

En este caso, no se explicará con detalle este módulo. Su función es tan solo la de repetidor, es decir, recibiría por bluetooth los comandos o coordenadas del usuario, y las enviaría por radio hacia el dron. Por tanto, su codificación es muy sencilla y no merece explicarla.

8 Descripción de tareas

En este apartado se muestran los distintos paquetes de trabajo en los que se ha dividido el proyecto. También se mostrarán los hitos y finalmente el diagrama de Gantt.

8.1 Paquetes de trabajo

Tabla 4. Primer paquete de trabajo.

PT1	Comienzo	Fin	Duración
Gestión del proyecto Seguimiento realizado para comprobar el correcto desarrollo del proyecto	22/02/18	18/07/18	147 días
PT1.1 Propuesta del proyecto	22/02/18	22/02/18	0 días
PT1.2 Gestión y seguimiento del trabajo	22/02/18	18/07/18	147 días

Tabla 5. Segundo paquete de trabajo.

PT2	Comienzo	Fin	Duración
Preparación del proyecto Adquisición de conocimientos y perspectiva necesarios para empezar a desarrollar el proyecto	22/02/18	29/03/18	36 días
PT2.1 Análisis inicial drones	22/02/18	28/02/18	7 días
PT2.2 Estudio situación actual	28/02/18	06/03/18	7 días
PT2.3 Análisis proyectos existentes	06/03/18	16/03/18	11 días
PT2.4Análisis componentes hardware	16/03/18	29/03/18	14 días

Tabla 6. Tercer paquete de trabajo.

PT3	Comienzo	Fin	Duración
Desarrollo del proyecto Fases llevadas a cabo en el desarrollo del proyecto en sí mismo	29/03/18	17/07/18	111 días
PT3.1 Selección de componentes hardware	30/03/18	06/04/18	8 días
PT3.2 Diseño software y hardware	06/04/18	20/04/18	15 días
PT3.3 Desarrollo conexión smartphone-arduino	29/03/18	19/04/18	22 días
PT3.4 Desarrollo conexión dron-arduino y desarrollo software de control	29/03/18	09/07/18	103 días
PT3.5 Unión de ambas partes	10/07/18	16/07/18	7 días

Tabla 7. Cuarto paquete de trabajo.

PT4	Comienzo	Fin	Duración
Documentación y presentación del proyecto Redacción de la documentación del TFG y creación de la presentación oral.	20/06/18	18/07/18	29 días
PT4.1 Documentación del proyecto	20/06/18	14/07/18	25 días
PT4.3 Presentación para el proyecto	14/07/18	17/07/18	4 días

8.2 Hitos del proyecto

A continuación, se muestran los diferentes hitos a alcanzar.

Tabla 8. Hitos del proyecto.

	Hito	Fecha
Hito 1	Comienzo del proyecto	22/02/18
Hito 2	Finalización de la documentación	14/07/18
Hito 3	Pruebas finales	17/07/18
Hito 4	Finalización de la presentación	18/07/18
Hito 5	Finalización del proyecto	18/07/18

8.3 Diagramas de Gantt

A continuación, se muestra el diagrama de Gantt del proyecto, donde se pueden apreciar todos los paquetes de trabajo con sus respectivos subpaquetes, así como las fechas de inicio y fin.

	EDT	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predec
1	**PT1**		**Gestión del proyecto**	**147 días**	**jue 22/02/18**	**mié 18/07/18**	
2	PT1.1		Propuesta del proyecto	0 días	jue 22/02/18	jue 22/02/18	
3	PT1.2		Gestión y seguimiento del trabajo	147 días	jue 22/02/18	mié 18/07/18	
4	PT1.3		Comienzo del proyecto	0 días	jue 22/02/18	jue 22/02/18	
5	**PT2**		**Preparación del proyecto**	**36 días**	**jue 22/02/18**	**jue 29/03/18**	
6	PT2.1		Análisis inicial drones	7 días	jue 22/02/18	mié 28/02/18	
7	PT2.2		Estudio situación actual	7 días	mié 28/02/18	mar 06/03/18	
8	PT2.3		Análisis proyectos existentes	11 días	mar 06/03/18	vie 16/03/18	
9	PT2.4		Análisis componentes hardware	14 días	vie 16/03/18	jue 29/03/18	
10	**PT3**		**Desarrollo del proyecto**	**111 días**	**jue 29/03/18**	**mar 17/07/18**	
11	PT3.1		Selección de componentes hardware	8 días	vie 30/03/18	vie 06/04/18	9
12	PT3.2		Diseño software y hardware	15 días	vie 06/04/18	vie 20/04/18	
13	**PT3.3**		**Desarrollo conexión smartphone-arduino**	**22 días**	**jue 29/03/18**	**jue 19/04/18**	
14	PT3.3.1		Desarrollo software módulo arduino repetidor	2 días	jue 29/03/18	vie 30/03/18	
15	PT3.3.2		Desarrollo aplicación Android	20 días	vie 30/03/18	mié 18/04/18	
16	PT3.3.3		Prueba conexión smartphone-arduino	1 día	mié 18/04/18	mié 18/04/18	

Ilustración 56. Tareas 1.

	EDT	Modo de	Nombre de tarea	Duración	Comienzo	Fin
16	PT3.3.3		Prueba conexión smartphone-arduino	1 día	mié 18/04/18	mié 18/04/18
17	**PT3.4**		**Desarrollo conexión dron-arduino y desarrollo software de control**	**103 días**	**jue 29/03/18**	**lun 09/07/18**
18	PT3.4.1		Aprendizaje sobre PIDs	4 días	jue 29/03/18	dom 01/04/1
19	PT3.4.2		Aprendizaje sobre GPS	5 días	lun 02/04/18	vie 06/04/18
20	PT3.4.3		Aprendizaje en profundidad sobre control de drones	12 días	sáb 07/04/18	mié 18/04/18
21	PT3.4.4		Desarrollo software de control y seguimiento	83 días	mié 18/04/18	lun 09/07/18
22	PT3.5		Unión de ambas partes	7 días	mar 10/07/1	lun 16/07/18
23	PT3.6		Pruebas finales	0 días	mar 17/07/1	mar 17/07/1
24	**PT4**		**Documentación y presentación del proyecto**	**29 días**	**mié 20/06/18**	**mié 18/07/18**
25	PT4.1		Documentación del proyecto	25 días	mié 20/06/18	sáb 14/07/18
26	PT4.2		Finalización de la documentación	0 días	sáb 14/07/18	sáb 14/07/18
27	PT4.3		Presentación para el proyecto	4 días	sáb 14/07/18	mar 17/07/18
28	PT4.4		Finalización de la presentación	0 días	mié 18/07/18	mié 18/07/18
29	PT5		Finalización del proyecto	0 días	mié 18/07/1	mié 18/07/1

Ilustración 57. Tareas 2.

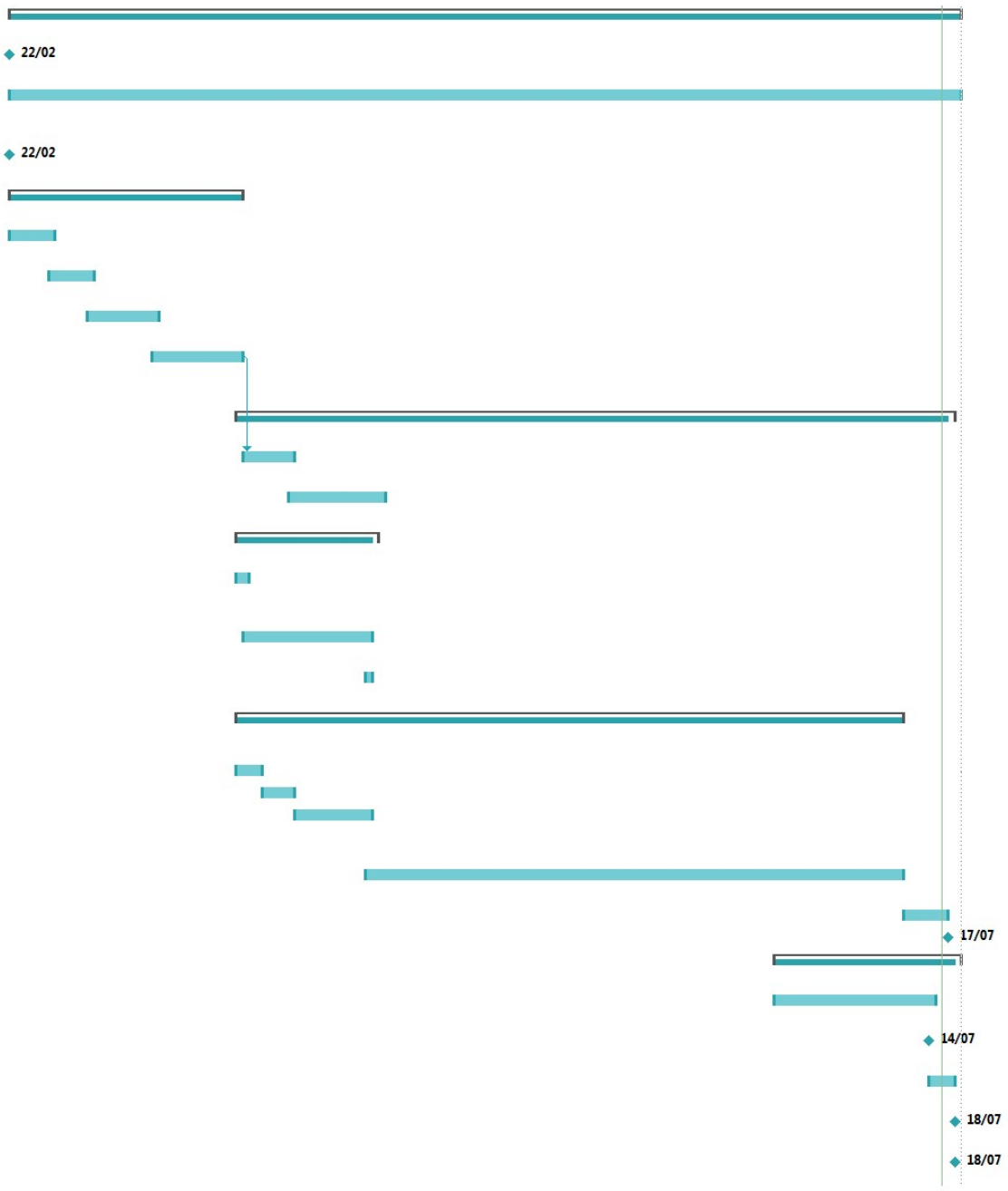


Ilustración 58. Diagrama Gantt.

9 Descargo de gastos

En este apartado se presenta el descargo de gastos del proyecto, en el que se calculan, por un lado, los costes de los recursos humanos empleados y, por otro lado, los costes de los materiales utilizados, tanto el material amortizable como el fungible.

9.1 Recursos humanos

En el apartado de recursos humanos, contamos con dos miembros en el equipo de trabajo, por un lado, el director del proyecto, y por otro lado el que desarrolla el proyecto, el Ingeniero Junior. Estos tienen un coste por hora trabajada diferente y la cantidad de horas que le dedican al proyecto también varía.

En la siguiente tabla, se muestra un desglose de las horas internas de cada integrante del equipo.

Tabla 9. Recursos humanos.

Concepto	€/h	Cantidad (h)	Total (€)
Director del proyecto	60	30	1800
Ingeniero Junior	30	441	13230
Total			**15030**

9.2 Recursos materiales

En las siguientes tablas se determinarán los precios de los materiales utilizados en el desarrollo del proyecto.

9.2.1 Gastos

Tabla 10. Tabla de gastos.

Concepto	Precio (€)	Cantidad	Total (€)
Arduino Uno	20	2	40
mpu 6050	12	1	12
Motores	12	4	48
ESCs	20	4	60
Hélices	3	4	12
Batería	6	1	6
Pila 9V	6	1	6
Cableado	3	1	3
Frame	12	1	12
Hc-06	3	1	3
nrf24l01+	4	2	8
Gps GY-NEO6MV2	20	1	20
SUBTOTAL			**230**

9.2.2 Amortizaciones

Tabla 11. Amortizaciones.

Concepto	Coste inicial (€)	Vida útil (meses)	Uso (meses)	Coste (€)
Ordenador	1200	60	6	120
Office 2016	150	60	2	5
Total				**125**

Además, se hace uso del IDE de Arduino y del IDE Android Studio. Pero, al ser ambos completamente gratuitos, no suponen ningún gasto para el proyecto.

9.3 Resumen descargo de gastos.

Por último, se presenta el resumen del descargo de gastos total, donde se tienen en cuenta todos los gastos mencionados anteriormente.

Tabla 12. Resumen descargo de gastos.

Concepto	Coste
Horas internas	15030
Gastos	230
Amortizaciones	125
Total	**15385**

10 Conclusiones

Como se ha tratado a lo largo de este TFG, se ha desarrollado un dron capaz de realizar un seguimiento autónomo de una persona basado en GPS. Durante el desarrollo del mismo, se han encontrado problemas que se han ido solucionando, tales como problemas con la cobertura, estabilidad, software en sí mismo etc.

Una de las claves del conjunto del sistema es la sencillez de la aplicación de smartphone y el habernos centrado solo en el seguimiento autónomo. No obstante, este proyecto podría ampliarse si se le dotara de más funciones, tales como el envío de ciertos parámetros útiles al usuario u a otro lugar, la toma o reconocimiento de imágenes o cualquier otra habilidad que pudiera ser añadida al dron, potenciando al 100% las capacidades de seguimiento del mismo.

Estas capacidades adicionales y sus potenciales usos aumentarán exponencialmente en los próximos años, debido a la explosión que sufre actualmente el mercado de drones. Por supuesto, junto con dichas capacidades y usos, también aparecerán nuevos beneficios sociales, técnicos y económicos, a parte de los ya mencionados en el apartado correspondiente.

Además, con la futura llegada de Bluetooth 5 (una nueva versión que se encuentra en desarrollo), podríamos eliminar el módulo repetidor intermedio, dado que se espera que con esta nueva tecnología tanto el alcance como la capacidad del enlace aumenten considerablemente. En el caso del alcance, se han logrado rangos de varios cientos de metros en algunos experimentos, algo hasta ahora impensable para las versiones y dispositivos actuales. Por ello, sería interesante actualizar el proyecto como un sistema dron-smartphone, sin necesidad de hardware adicional, puesto que, aunque no resulta relativamente incómodo, sí que puede ser un inconveniente de cara al usuario.

Como conclusión adicional, hay que destacar la sencillez de uso y la versatilidad de los módulos Arduino, así como también la versatilidad de Android y su entorno de desarrollo. Se ha comprobado la gran cantidad de periféricos y módulos adicionales que se puede añadir a Arduino, creando así proyectos más o menos complejos. Además, gracias a esto, cualquier proyecto ya diseñado se puede ampliar para dotarlo de más funcionalidades, incluido el nuestro. En el caso de Android y sus aplicaciones, se ha observado una relativa facilidad para el desarrollo de estas, no necesitando unos amplios conocimientos de programación para lograr una aplicación sencilla y funcional. Este detalle es importante a la hora de crear proyectos como el de este TFG u otros, ya que permite que prácticamente cualquiera pueda crear una aplicación para smartphone, siendo esto muy beneficioso técnica y económicamente.

11 Bibliografía y fuentes de información

Androfast. *Como obtener la ubicación del GPS-Android Studio.*
http://www.androfast.com/2015/07/como-obtener-la-ubicacion-del-gps.html

Android Developers. *Bluetooth.*
https://developer.android.com/guide/topics/connectivity/bluetooth?hl=es-419

Android Developers. *Location.*
https://developer.android.com/things/sdk/drivers/location

Arduino. *¿Qué es Arduino?*
http://ingenio-triana.blogspot.com/2015/12/todo-lo-que-necesitas-saber-sobre-drones.html

Engineering360 News Desk. *Specifying an accelerometer: Function and applications.*
https://insights.globalspec.com/article/1263/specifying-an-accelerometer-function-and-applications

Hemav. *La situación actual del mercado de los drones.*
https://hemav.com/la-situacion-actual-del-mercado-de-los-drones/

Ingenio-Triana. *Todo lo que necesitas saber sobre drones*.
http://ingenio-triana.blogspot.com/2015/12/todo-lo-que-necesitas-saber-sobre-drones.html

Mancioc, E. (2017). *Física de un quadcóptero.* Trabajo. Tárrega: Vedruna Tárrega.
https://www.vedrunacatalunya.cat/recerques/treballs/2017/tarrega/TR_estefania_mancioc.pdf

Mazzone, V. (2002). *Controladores PID.* Apuntes universitarios. Quilmes, Argentina: Universidad Nacional de Quilmes.
http://www.eng.newcastle.edu.au/~jhb519/teaching/caut1/Apuntes/PID.pdf

NaylampMechatronics. *Tutorial básico de uso del módulo hc-06 y hc-05.*
https://www.naylampmechatronics.com/blog/12_Tutorial-B%C3%A1sico-de-Uso-del-M%C3%B3dulo-Bluetooth-H.html

NaylampMechatronics. *Tutorial módulo GPS con Arduino.*
https://naylampmechatronics.com/blog/18_Tutorial-M%C3%B3dulo-GPS-con-Arduino.html

Prometec. *Configurar el módulo bluetooth hc-06.*
https://www.prometec.net/mbock-coche-hc06/

Prometec. *Controlando Arduino con Android.*
https://www.prometec.net/android-bt/

Prometec. *Los módulos de radio nrf2401.*
https://www.prometec.net/nrf2401/

Prometec. *Módulo bluetooth hc-06.*
https://www.prometec.net/bt-hc06/

Prometec. *Usando el MPU6050.*
https://www.prometec.net/usando-el-mpu6050/

Raspberrypi.org. *FAQS.*
https://www.raspberrypi.org/help/faqs/

Starter kit. Learning Arduino. *Connecting and programming nRF24L01 with Arduino and other boards.*
http://starter-kit.nettigo.eu/2014/connecting-and-programming-nrf24l01-with arduino-and-other-boards/

Todrone. *Así está el mapa del mercado global de los drones en 2018.*
https://www.todrone.com/mapa-mercado-global-drones-2018/

Wikipedia. *Android.*
https://es.wikipedia.org/wiki/Android

Wikipedia. *Bluetooth.*
https://es.wikipedia.org/wiki/Bluetooth

Wikipedia. *Controlador PID.*
https://es.wikipedia.org/wiki/Controlador_PID

Wikipedia. *Raspberry Pi.*
https://es.wikipedia.org/wiki/Raspberry_Pi

Wooley, M. (2017). *Bluetooth 5: Go faster. Go further.* Paper. Reino Unido: Bluetooth SIG.
https://www.bluetooth.com/bluetooth-technology/bluetooth5/bluetooth5-paper

12 Anexo 1 – Software codificado

Software de control del dron: *Flight_controller.ino*

```
#include <Wire.h>          //Include the Wire.h library so we can communicate with the gyro.
#include <EEPROM.h>        //Include the EEPROM.h library so we can store information onto the EEPROM

//mis includes//////////////////////////

#include <TinyGPS.h>  //GPS parsing library
#include <SoftwareSerial.h>
#include <SPI.h>      //Serial Peripheral Interface (SPI) library
#include <nRF24L01.h> //libería radio
#include <RF24.h>     //libería radio

/////////////////////////////////////////////////////////////////
//mis variables

int segur=0; // flag/contador para seguridad--->en caso de perder contacto radio por 3 segundos
int conta=0;
SoftwareSerial gps(1,2);//RX 1 TX 2
const byte rxAddr[6] = "00001"; // pipe radio
RF24 radio(8,9);
int emp=1;
char text[150] = {0};
char gpsLOC[150]={0};
TinyGPS gpsd;
float flat, flon, flatu, flonu;
unsigned long age;

float pid_i_mem_lat,pid_output_lat;
float pid_i_gain_lat=100000; //modificar valor
float pid_p_gain_lat=1000000;//modificar valor
float pid_d_gain_lat=100000;//modificar valor
float pid_last_lat_d_error;
float pid_max_lat=500;//modificar valor

float pid_i_mem_lon,pid_output_lon;
float pid_i_gain_lon=100000; //modificar valor
float pid_p_gain_lon=1000000;//modificar valor
float pid_d_gain_lon=100000;//modificar valor
float pid_last_lon_d_error;
float pid_max_lon=500;//modificar valor

float pid_i_mem_alt,pid_output_alt;
float pid_i_gain_alt=0.01; //modificar valor
float pid_p_gain_alt=0.125;//modificar valor
float pid_d_gain_alt=0.09;//modificar valor
float pid_last_alt_d_error;
float pid_max_alt=500;//modificar valor

////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////
//PID gain and limit settings
////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////
float pid_p_gain_roll = 1.3;               //Gain setting for the roll P-controller
float pid_i_gain_roll = 0.04;              //Gain setting for the roll I-controller
float pid_d_gain_roll = 18.0;              //Gain setting for the roll D-controller
int pid_max_roll = 400;                    //Maximum output of the PID-controller (+/-)

float pid_p_gain_pitch = pid_p_gain_roll;  //Gain setting for the pitch P-controller.
float pid_i_gain_pitch = pid_i_gain_roll;  //Gain setting for the pitch I-controller.
float pid_d_gain_pitch = pid_d_gain_roll;  //Gain setting for the pitch D-controller.
int pid_max_pitch = pid_max_roll;          //Maximum output of the PID-controller (+/-)

float pid_p_gain_yaw = 4.0;                //Gain setting for the pitch P-controller. //4.0
float pid_i_gain_yaw = 0.02;               //Gain setting for the pitch I-controller. //0.02
float pid_d_gain_yaw = 0.0;                //Gain setting for the pitch D-controller.
int pid_max_yaw = 400;                     //Maximum output of the PID-controller (+/-)

boolean auto_level = true;                 //Auto level on (true) or off (false)
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////
//Declaring global variables
////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////
byte last_channel_1, last_channel_2, last_channel_3, last_channel_4;
byte eeprom_data[36];
byte highByte, lowByte;
volatile int receiver_input_channel_1, receiver_input_channel_2, receiver_input_channel_3,
receiver_input_channel_4;
int counter_channel_1, counter_channel_2, counter_channel_3, counter_channel_4, loop_counter;
int esc_1, esc_2, esc_3, esc_4;
int throttle, battery_voltage;
int cal_int, start, gyro_address;
int receiver_input[5];
int temperature;
int acc_axis[4], gyro_axis[4];
float roll_level_adjust, pitch_level_adjust;

long acc_x, acc_y, acc_z, acc_total_vector;
unsigned long timer_channel_1, timer_channel_2, timer_channel_3, timer_channel_4, esc_timer,
esc_loop_timer;
unsigned long timer_1, timer_2, timer_3, timer_4, current_time;
unsigned long loop_timer;
double gyro_pitch, gyro_roll, gyro_yaw;
double gyro_axis_cal[4];
float pid_error_temp;
float pid_i_mem_roll, pid_roll_setpoint, gyro_roll_input, pid_output_roll, pid_last_roll_d_error;
float pid_i_mem_pitch, pid_pitch_setpoint, gyro_pitch_input, pid_output_pitch,
pid_last_pitch_d_error;
float pid_i_mem_yaw, pid_yaw_setpoint, gyro_yaw_input, pid_output_yaw, pid_last_yaw_d_error;
float angle_roll_acc, angle_pitch_acc, angle_pitch, angle_roll;
boolean gyro_angles_set;

////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////
//Setup routine
////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////
void setup(){

  //mi setup/////////////////////////////////////////////
  gps.begin(9600);

  radio.begin();
  radio.openReadingPipe(0, rxAddr);
  radio.startListening();
  ////////////////////////////////////////////////

  //Copy the EEPROM data for fast access data.
  for(start = 0; start <= 35; start++)eeprom_data[start] = EEPROM.read(start);
  start = 0;                                                  //Set start back to zero.
  gyro_address = eeprom_data[32];                        //Store the gyro address in the variable.

  Wire.begin();                                             //Start the I2C as master.

  TWBR = 12;                                               //Set the I2C clock speed to 400kHz.

//Arduino (Atmega) pins default to inputs, so they don't need to be explicitly declared as inputs.
  DDRD |= B11110000;                                 //Configure digital poort 4, 5, 6 and 7 as output.
  DDRB |= B00110000;                                 //Configure digital poort 12 and 13 as output.


  //Use the led on the Arduino for startup indication.
  digitalWrite(12,HIGH);                                                    //Turn on the warning led.

//Check the EEPROM signature to make sure that the setup program is executed.
 // while(eeprom_data[33] != 'J' || eeprom_data[34] != 'M' || eeprom_data[35] != 'B')delay(10);

  //The flight controller needs the MPU-6050 with gyro and accelerometer
  //If setup is completed without MPU-6050 stop the flight controller program
  if(eeprom_data[31] == 2 || eeprom_data[31] == 3)delay(10);

set_gyro_registers();                                               //Set the specific gyro registers.
```

```
for (cal_int = 0; cal_int < 1250 ; cal_int ++){            //Wait 5 seconds before continuing.
    PORTD |= B11110000;                                    //Set digital poort 4, 5, 6 and 7 high.
    delayMicroseconds(1000);                               //Wait 1000us.
    PORTD &= B00001111;                                    //Set digital poort 4, 5, 6 and 7 low.
    delayMicroseconds(3000);                               //Wait 3000us.
  }

//Let's take multiple gyro data samples so we can determine the average gyro offset (calibration).
for (cal_int = 0; cal_int < 2000 ; cal_int ++){     //Take 2000 readings for calibration.
 if(cal_int % 15 == 0)digitalWrite(12, !digitalRead(12)); //Change the led status to indicate
//calibration.
    gyro_signalen();                                         //Read the gyro output.
    gyro_axis_cal[1] += gyro_axis[1];                        //Ad roll value to gyro_roll_cal.
    gyro_axis_cal[2] += gyro_axis[2];                        //Ad pitch value to gyro_pitch_cal.
    gyro_axis_cal[3] += gyro_axis[3];                        //Ad yaw value to gyro_yaw_cal.
//We don't want the esc's to be beeping annoyingly. So let's give them a 1000us puls while
//calibrating the gyro.
    PORTD |= B11110000;                                      //Set digital poort 4, 5, 6 and 7 high.
    delayMicroseconds(1000);                                 //Wait 1000us.
    PORTD &= B00001111;                                      //Set digital poort 4, 5, 6 and 7 low.
    delay(3);                                              //Wait 3 milliseconds before the next loop.
  }
  //Now that we have 2000 measures, we need to devide by 2000 to get the average gyro offset.
  gyro_axis_cal[1] /= 2000;                                  //Divide the roll total by 2000.
  gyro_axis_cal[2] /= 2000;                                  //Divide the pitch total by 2000.
  gyro_axis_cal[3] /= 2000;                                  //Divide the yaw total by 2000.


//Mi función para que pueda empezar


while(emp!=0){
  if (radio.available())
    {
     radio.read(&text, sizeof(text));
     emp=strcmp(text,"COMANDO%ENCENDIDO#");
     strcpy(text,"");
    }
    start ++;                                            //While waiting increment start whith every loop.
    //We don't want the esc's to be beeping annoyingly. So let's give them a 1000us puls while
//waiting for the receiver inputs.
    PORTD |= B11110000;                                    //Set digital poort 4, 5, 6 and 7 high.
    delayMicroseconds(1000);                               //Wait 1000us.
    PORTD &= B00001111;                                    //Set digital poort 4, 5, 6 and 7 low.
    delay(3);                                              //Wait 3 milliseconds before the next loop.
    if(start == 125){                                      //Every 125 loops (500ms).
      digitalWrite(12, !digitalRead(12));                                  //Change the led status.
      start = 0;                                                           //Start again at 0.
    }

}


  start = 0;                                                               //Set start back to 0.

  //Load the battery voltage to the battery_voltage variable.
  //65 is the voltage compensation for the diode.
  //12.6V equals ~5V @ Analog 0.
  //12.6V equals 1023 analogRead(0).
  //1260 / 1023 = 1.2317.
  //The variable battery_voltage holds 1050 if the battery voltage is 10.5V.
  battery_voltage = (analogRead(0) + 65) * 1.2317;


  loop_timer = micros();                                     //Set the timer for the next loop.

  //When everything is done, turn off the led.
  digitalWrite(12,LOW);                                    //Turn off the warning led.
}
```

```
void loop(){

  //65.5 = 1 deg/sec
gyro_roll_input = (gyro_roll_input * 0.7) + ((gyro_roll / 65.5) * 0.3); //Gyro pid input is deg/sec.
gyro_pitch_input = (gyro_pitch_input * 0.7) + ((gyro_pitch / 65.5)* 0.3);//Gyro pid input is deg/sec.
gyro_yaw_input = (gyro_yaw_input * 0.7) + ((gyro_yaw / 65.5) * 0.3);    //Gyro pid input is deg/sec.

 //Gyro angle calculations
 //0.0000611 = 1 / (250Hz / 65.5)
 angle_pitch += gyro_pitch * 0.0000611;  //Calculate the traveled pitch angle and add this to the
//angle_pitch variable.
  angle_roll += gyro_roll * 0.0000611;   //Calculate the traveled roll angle and add this to the
//angle_roll variable.

  //0.000001066 = 0.0000611 * (3.142(PI) / 180degr) The Arduino sin function is in radians
  angle_pitch -= angle_roll * sin(gyro_yaw * 0.000001066);                 //If the IMU has yawed
//transfer the roll angle to the pitch angel.
  angle_roll += angle_pitch * sin(gyro_yaw * 0.000001066);                 //If the IMU has yawed
//transfer the pitch angle to the roll angel.

  //Accelerometer angle calculations
  acc_total_vector = sqrt((acc_x*acc_x)+(acc_y*acc_y)+(acc_z*acc_z));       //Calculate the total
//accelerometer vector.

  if(abs(acc_y) < acc_total_vector){                 //Prevent the asin function to produce a NaN
    angle_pitch_acc = asin((float)acc_y/acc_total_vector)* 57.296;    //Calculate the pitch angle.
  }
  if(abs(acc_x) < acc_total_vector){             //Prevent the asin function to produce a NaN
    angle_roll_acc = asin((float)acc_x/acc_total_vector)* -57.296;  //Calculate the roll angle.
  }

  //Place the MPU-6050 spirit level and note the values in the following two lines for calibration.
  angle_pitch_acc -= 0.0;                           //Accelerometer calibration value for pitch.
  angle_roll_acc -= 0.0;                            //Accelerometer calibration value for roll.

  angle_pitch = angle_pitch * 0.9996 + angle_pitch_acc * 0.0004;            //Correct the drift of
//the gyro pitch angle with the accelerometer pitch angle.
  angle_roll = angle_roll * 0.9996 + angle_roll_acc * 0.0004;               //Correct the drift of
//the gyro roll angle with the accelerometer roll angle.

  pitch_level_adjust = angle_pitch * 15;          //Calculate the pitch angle correction
  roll_level_adjust = angle_roll * 15;            //Calculate the roll angle correction

  if(!auto_level){                                //If the quadcopter is not in auto-level mode
    pitch_level_adjust = 0;                      //Set the pitch angle correction to zero.
    roll_level_adjust = 0;                      //Set the roll angle correcion to zero.
  }

////////////////////////////////////////////
//Mi código principal para arrancar los motores/moverse/parar ////////////

 segur++;//contador de seguridad

 if (radio.available())
    {
     segur=0;
     radio.read(&text, sizeof(text));
     if(start==0){
     emp=strcmp(text,"COMANDO%EMPEZAR#");
      if(emp==0){

       // despegue(); //llamo a mi función para despegar
        //start = 2;  //estado en marcha
        start=5; //estado despegue
        angle_pitch = angle_pitch_acc;     //Set the gyro pitch angle equal to the accelerometer
//pitch angle when the quadcopter is started.
        angle_roll = angle_roll_acc;    //Set the gyro roll angle equal to the accelerometer roll
//angle when the quadcopter is started.
        gyro_angles_set = true;          //Set the IMU started flag.
```

```
//Reset the PID controllers for a bumpless start.
        pid_i_mem_roll = 0;
        pid_last_roll_d_error = 0;
        pid_i_mem_pitch = 0;
        pid_last_pitch_d_error = 0;
        pid_i_mem_yaw = 0;
        pid_last_yaw_d_error = 0;

        pid_i_mem_lat=0;
        pid_last_lat_d_error=0;
        pid_i_mem_lon=0;
        pid_last_lon_d_error=0;

        pid_i_mem_alt=0;
        pid_last_alt_d_error=0;

        pid_output_alt=0;

  pid_roll_setpoint=1500; //simulo que es la señal del mando y que pongo los sticks en el centro
  pid_pitch_setpoint=1500;//simulo que es la señal del mando y que pongo los sticks en el centro
        pid_yaw_setpoint=1500;//simulo que es la señal del mando y que pongo los sticks en el centro

        strcpy(text,""); //limpio la variable text
        }
      }
      else if(start==2){
        emp=strncmp(text,"GEOLOCA%",8);  //solo comparo los  primeros caracteres, porque si me ha
//llegado coordenadas éstas son siempre distintas, solo necesito saber que son coordenadas
        if(emp==0){
          //llamar a funcion de geoloc
             calculaGPS();

            strcpy(text,"");

        }else{
          emp= strcmp(text,"COMANDO%DETENER#"); //se detiene el dron pero no aterriza
           if(emp==0){

            strcpy(text,"");
            start = 1; //estado detenido
            pid_roll_setpoint=1500; //simulo que es la señal del mando y detengo el dron
            pid_pitch_setpoint=1500;//simulo que es la señal del mando y detengo el dron
            pid_yaw_setpoint=1500;
            }
          }
      }
     else if(start==1){
        emp=strcmp(text,"COMANDO%APAGADO#");
        if(emp==0){
          start = 6; //estado aterrizado
          //debería llamar a una función para que aterrice suavemente
          //aterriza(); //función mía de aterrizaje
          //

          strcpy(text,""); //limpio text
        }else{
          emp=strcmp(text,"COMANDO%EMPEZAR#"); //si estoy detenido (en el aire) pero quiero seguir
//volando
          if(emp==0){
            start=2;
            strcpy(text,"");//limpio text
          }
        }
      }
      else if(start==5)
      {
       // Serial.print("despegar");
        if(conta<500)
        {
          throttle = 1600;
          conta++;

        }
```

```
        else
        {
          throttle = 1500;
          conta=0;
         // Serial.print("FINdespegar");
          start=2;
        }
      }
      else if(start==6)
      {
        // Serial.print("aterrizar");
        if(conta<500)
        {
          throttle = 1400;
          conta++;

        }
        else
        {
          throttle = 0;
          conta=0;
         // Serial.print("FINaterrizar");
          start=0;
        }
      }

 }

if(segur>=750){  //si pasa 3 segundos (750 loops) sin contacto radio--->aterrizaje emergencia //3
//segundos aproximadamente (porque depende de la duración de la señal a los esc) (estas señales sí se
//envían exactamente cada 4ms)
    pid_roll_setpoint=1500; //simulo que es la señal del mando y detengo el dron
    pid_pitch_setpoint=1500;//simulo que es la señal del mando y detengo el dron
    pid_yaw_setpoint=1500;  //simulo que es la señal del mando y detengo el dron
    //aterriza(); //mi función para aterrizar
    start=6;//estado aterrizar
}

pid_roll_setpoint -= roll_level_adjust;  //Subtract the angle correction from the standardized
//receiver roll input value.
pid_roll_setpoint /= 3.0;  //Divide the setpoint for the PID roll controller by 3 to get angles in
//degrees.


  pid_pitch_setpoint -= pitch_level_adjust;   //Subtract the angle correction from the standardized
//receiver pitch input value.
  pid_pitch_setpoint /= 3.0;   //Divide the setpoint for the PID pitch controller by 3 to get angles
//in degrees.


  calculate_pid();         //PID inputs are known. So we can calculate the pid output.

  //The battery voltage is needed for compensation.
  //A complementary filter is used to reduce noise.
  //0.09853 = 0.08 * 1.2317.
  battery_voltage = battery_voltage * 0.92 + (analogRead(0) + 65) * 0.09853;

  //Turn on the led if battery voltage is to low.
  if(battery_voltage < 1000 && battery_voltage > 600)digitalWrite(12, HIGH);

if(start==2 || start==1){
  throttle = 1500 + pid_output_alt;
//lo cambio por el output del pid altura -> así corrijo la bajada en altura
}
```

```
if (start == 2 || start==1 || start==5 || start==6){
//The motors are started. //añado el posible estado "start=1"
    if (throttle > 1800) throttle = 1800;  //We need some room to keep full control at full throttle.
    esc_1 = throttle - pid_output_pitch + pid_output_roll - pid_output_yaw; //Calculate the pulse for
//esc 1 (front-right - CCW)
    esc_2 = throttle + pid_output_pitch + pid_output_roll + pid_output_yaw; //Calculate the pulse for
//esc 2 (rear-right - CW)
    esc_3 = throttle + pid_output_pitch - pid_output_roll - pid_output_yaw; //Calculate the pulse for
//esc 3 (rear-left - CCW)
    esc_4 = throttle - pid_output_pitch - pid_output_roll + pid_output_yaw; //Calculate the pulse for
//esc 4 (front-left - CW)

    if (battery_voltage < 1240 && battery_voltage > 800){       //Is the battery connected?
      esc_1 += esc_1 * ((1240 - battery_voltage)/(float)3500);   //Compensate the esc-1 pulse for
//voltage drop.
      esc_2 += esc_2 * ((1240 - battery_voltage)/(float)3500);              //Compensate the esc-2
//pulse for voltage drop.
      esc_3 += esc_3 * ((1240 - battery_voltage)/(float)3500);              //Compensate the esc-3
//pulse for voltage drop.
      esc_4 += esc_4 * ((1240 - battery_voltage)/(float)3500);              //Compensate the esc-4
//pulse for voltage drop.
    }

    if (esc_1 < 1100) esc_1 = 1100;                                   //Keep the motors running.
    if (esc_2 < 1100) esc_2 = 1100;                                   //Keep the motors running.
    if (esc_3 < 1100) esc_3 = 1100;                                   //Keep the motors running.
    if (esc_4 < 1100) esc_4 = 1100;                                   //Keep the motors running.

    if(esc_1 > 2000)esc_1 = 2000;                                    //Limit the esc-1 pulse to 2000us.
    if(esc_2 > 2000)esc_2 = 2000;                                    //Limit the esc-2 pulse to 2000us.
    if(esc_3 > 2000)esc_3 = 2000;                                    //Limit the esc-3 pulse to 2000us.
    if(esc_4 > 2000)esc_4 = 2000;                                    //Limit the esc-4 pulse to 2000us.
  }

  else{
    esc_1 = 1000;                                          //If start is not 2 (OR 1) keep a 1000us
pulse for ess-1.
    esc_2 = 1000;                                          //If start is not 2 (OR 1) keep a 1000us
pulse for ess-2.
    esc_3 = 1000;                                          //If start is not 2 (OR 1) keep a 1000us
pulse for ess-3.
    esc_4 = 1000;                                          //If start is not 2 (OR 1) keep a 1000us
pulse for ess-4.
  }

if(micros() - loop_timer > 4050)digitalWrite(12, HIGH); //Turn on the LED if the loop time exceeds
//4050us.

  //All the information for controlling the motor's is available.
  //The refresh rate is 250Hz. That means the esc's need there pulse every 4ms.
  while(micros() - loop_timer < 4000);             //We wait until 4000us are passed.
  loop_timer = micros();                           //Set the timer for the next loop.

  PORTD |= B11110000;                              //Set digital outputs 4,5,6 and 7 high.
  timer_channel_1 = esc_1 + loop_timer;   //Calculate the time of the faling edge of the esc-1 pulse.
  timer_channel_2 = esc_2 + loop_timer;   //Calculate the time of the faling edge of the esc-2 pulse.
  timer_channel_3 = esc_3 + loop_timer;   //Calculate the time of the faling edge of the esc-3 pulse.
  timer_channel_4 = esc_4 + loop_timer;   //Calculate the time of the faling edge of the esc-4 pulse.

//There is always 1000us of spare time. So let's do something usefull that is very time consuming.
//Get the current gyro and receiver data and scale it to degrees per second for the pid calculations.
  gyro_signalen();

  while(PORTD >= 16){         //Stay in this loop until output 4,5,6 and 7 are low.
    esc_loop_timer = micros();                                 //Read the current time.
    if(timer_channel_1 <= esc_loop_timer)PORTD &= B11101111;  //Set digital output 4 to low if the
//time is expired.
    if(timer_channel_2 <= esc_loop_timer)PORTD &= B11011111;  //Set digital output 5 to low if the
//time is expired.
    if(timer_channel_3 <= esc_loop_timer)PORTD &= B10111111;  //Set digital output 6 to low if the
//time is expired.
    if(timer_channel_4 <= esc_loop_timer)PORTD &= B01111111;  //Set digital output 7 to low if the
//time is expired.
  }
}
```

```
//mi función para calcular movimiento
void calculaGPS(){

  strcpy(gpsLOC,&text[8]);

  if(gps.available())
  {
     char clat[20];
     char clon[20];
     char c = gps.read(); //obtenemos el stream de datos del módulo gps
     if (gpsd.encode(c)){  //usamos la librería para traducirlo
    gpsd.f_get_position(&flat, &flon, &age); //obtenemos los datos principales // la variable age no
//se usa pero se debe incluir
     }
     // lat y lon vienen juntas separadas por el token "%" desde la app android
     strcpy(clat,strtok(gpsLOC,"%")); //obtenemos los datos del usuario
     strcpy(clon,strtok(NULL,"#"));
     flatu=atof(clat);  //se transforman a floats
     flonu=atof(clon);  //para poder manejarlos
     float difLat,difLon;
     difLat=flat-flatu; //hallamos las diferencias
     difLon=flon-flonu;


     if(difLon<0){ //incluimos un offset
      difLon+=0.00004;
      } //serían como 4 metros
     else{
      difLon-=0.00004;
      }

      if(difLat<0){ //el offset
      difLat+=0.00004;
      } //serían como 4 metros
     else{
      difLat-=0.00004;
      }

     //CALCULAR PID DE LON Y LAT
//aquí hago los cálculos fundamentales para que el dron sepa cuánto debe moverse y en qué sentido en
//ambos ejes xy
//nota: al no tener magnetómetro, tengo que orientarlo hacia el norte y puede que se acumulen errores

     //PID LAT
     pid_i_mem_lat += pid_i_gain_lat * difLat;
     if(pid_i_mem_lat > pid_max_lat)pid_i_mem_lat = pid_max_lat;
     else if(pid_i_mem_lat < pid_max_lat * -1)pid_i_mem_lat = pid_max_lat * -1;

     pid_output_lat = pid_p_gain_lat * difLat + pid_i_mem_lat + pid_d_gain_lat * (difLat -
pid_last_lat_d_error);
     if(pid_output_lat > pid_max_lat)pid_output_lat = pid_max_lat;
     else if(pid_output_lat < pid_max_lat * -1)pid_output_lat = pid_max_lat * -1;

     pid_last_lat_d_error = difLat;

     //PID LON
     pid_i_mem_lon += pid_i_gain_lon * difLon;
     if(pid_i_mem_lon > pid_max_lon)pid_i_mem_lon = pid_max_lon;
     else if(pid_i_mem_lon < pid_max_lon * -1)pid_i_mem_lon = pid_max_lon * -1;

     pid_output_lon = pid_p_gain_lon * difLon + pid_i_mem_lon + pid_d_gain_lon * (difLon -
pid_last_lon_d_error);
     if(pid_output_lon > pid_max_lon)pid_output_lon = pid_max_lon;
     else if(pid_output_lon < pid_max_lon * -1)pid_output_lon = pid_max_lon * -1;


     pid_last_lon_d_error = difLon;

     //estamos en lat N y lon W así que
     pid_output_lon=pid_output_lon*-1;
     pid_output_lat=pid_output_lat*1;

     pid_roll_setpoint=pid_output_lon; //simulo que es la señal del mando
     pid_pitch_setpoint=pid_output_lat;//simulo que es la señal del mando
  }
}
```

```
//Subroutine for reading the gyro
////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////
void gyro_signalen(){
  //Read the MPU-6050
  if(eeprom_data[31] == 1){
    Wire.beginTransmission(gyro_address);       //Start communication with the gyro.
    Wire.write(0x3B);           //Start reading @ register 43h and auto increment with every read.
    Wire.endTransmission();                         //End the transmission.
    Wire.requestFrom(gyro_address,14);              //Request 14 bytes from the gyro.


    while(Wire.available() < 14);               //Wait until the 14 bytes are received.
    acc_axis[1] = Wire.read()<<8|Wire.read();   //Add the low and high byte to the acc_x variable.
    acc_axis[2] = Wire.read()<<8|Wire.read();   //Add the low and high byte to the acc_y variable.
    acc_axis[3] = Wire.read()<<8|Wire.read();   //Add the low and high byte to the acc_z variable.
 temperature = Wire.read()<<8|Wire.read();  //Add the low and high byte to the temperature variable.
    gyro_axis[1] = Wire.read()<<8|Wire.read();    //Read high and low part of the angular data.
    gyro_axis[2] = Wire.read()<<8|Wire.read();    //Read high and low part of the angular data.
    gyro_axis[3] = Wire.read()<<8|Wire.read();   //Read high and low part of the angular data.
  }

  if(cal_int == 2000){
    gyro_axis[1] -= gyro_axis_cal[1];                //Only compensate after the calibration.
    gyro_axis[2] -= gyro_axis_cal[2];               //Only compensate after the calibration.
    gyro_axis[3] -= gyro_axis_cal[3];              //Only compensate after the calibration.
  }
  gyro_roll = gyro_axis[eeprom_data[28] & 0b00000011];   //Set gyro_roll to the correct axis that was
//stored in the EEPROM.
if(eeprom_data[28] & 0b10000000)gyro_roll*=-1; //Invert gyro_roll if the MSB of EEPROM bit 28 is set.
  gyro_pitch = gyro_axis[eeprom_data[29] & 0b00000011];     //Set gyro_pitch to the correct axis that
//was stored in the EEPROM.
if(eeprom_data[29] & 0b10000000)gyro_pitch*=-1; //Invert gyro_pitch ifthe MSB of EEPROM bit29 is set.
  gyro_yaw = gyro_axis[eeprom_data[30] & 0b00000011];  //Set gyro_yaw to the correct axis that was
//stored in the EEPROM.
  if(eeprom_data[30] & 0b10000000)gyro_yaw*=-1; //Invert gyro_yaw if the MSB of EEPROM bit 30 is set.

  acc_x = acc_axis[eeprom_data[29] & 0b00000011];  //Set acc_x to the correct axis that was stored in
//the EEPROM.
  if(eeprom_data[29] & 0b10000000)acc_x *= -1;     //Invert acc_x if the MSB of EEPROM bit 29 is set.
  acc_y = acc_axis[eeprom_data[28] & 0b00000011];  //Set acc_y to the correct axis that was stored in
//the EEPROM.
  if(eeprom_data[28] & 0b10000000)acc_y *= -1;    //Invert acc_y if the MSB of EEPROM bit 28 is set.
  acc_z = acc_axis[eeprom_data[30] & 0b00000011];  //Set acc_z to the correct axis that was stored in
//the EEPROM.
  if(eeprom_data[30] & 0b10000000)acc_z *= -1;   //Invert acc_z if the MSB of EEPROM bit 30 is set.
}

void calculate_pid(){
  //Roll calculations
  pid_error_temp = gyro_roll_input - pid_roll_setpoint;
  pid_i_mem_roll += pid_i_gain_roll * pid_error_temp;
  if(pid_i_mem_roll > pid_max_roll)pid_i_mem_roll = pid_max_roll;
  else if(pid_i_mem_roll < pid_max_roll * -1)pid_i_mem_roll = pid_max_roll * -1;

pid_output_roll = pid_p_gain_roll * pid_error_temp + pid_i_mem_roll + pid_d_gain_roll *
(pid_error_temp - pid_last_roll_d_error);
  if(pid_output_roll > pid_max_roll)pid_output_roll = pid_max_roll;
  else if(pid_output_roll < pid_max_roll * -1)pid_output_roll = pid_max_roll * -1;

  pid_last_roll_d_error = pid_error_temp;

//Pitch calculations
  pid_error_temp = gyro_pitch_input - pid_pitch_setpoint;
  pid_i_mem_pitch += pid_i_gain_pitch * pid_error_temp;
  if(pid_i_mem_pitch > pid_max_pitch)pid_i_mem_pitch = pid_max_pitch;
  else if(pid_i_mem_pitch < pid_max_pitch * -1)pid_i_mem_pitch = pid_max_pitch * -1;

pid_output_pitch = pid_p_gain_pitch * pid_error_temp + pid_i_mem_pitch + pid_d_gain_pitch *
(pid_error_temp - pid_last_pitch_d_error);
  if(pid_output_pitch > pid_max_pitch)pid_output_pitch = pid_max_pitch;
  else if(pid_output_pitch < pid_max_pitch * -1)pid_output_pitch = pid_max_pitch * -1;
```

```
pid_last_pitch_d_error = pid_error_temp;

  //Yaw calculations
  pid_error_temp = gyro_yaw_input - pid_yaw_setpoint;
  pid_i_mem_yaw += pid_i_gain_yaw * pid_error_temp;
  if(pid_i_mem_yaw > pid_max_yaw)pid_i_mem_yaw = pid_max_yaw;
  else if(pid_i_mem_yaw < pid_max_yaw * -1)pid_i_mem_yaw = pid_max_yaw * -1;

  pid_output_yaw = pid_p_gain_yaw * pid_error_temp + pid_i_mem_yaw + pid_d_gain_yaw * (pid_error_temp
- pid_last_yaw_d_error);
  if(pid_output_yaw > pid_max_yaw)pid_output_yaw = pid_max_yaw;
  else if(pid_output_yaw < pid_max_yaw * -1)pid_output_yaw = pid_max_yaw * -1;

  pid_last_yaw_d_error = pid_error_temp;

//
//CÁLCULOS DE ALT////////////////////////////////////////////////////////////////////////////
//
//son mis cálculos necesarios para corregir la posible variación en altura
//

pid_error_temp = 4096 - acc_z  ; //Añado un offset de 4096 (4096 sería 1g en la escala seleccionada).
                                 //En reposo supuestamente acc_z=4096, de ahí el offset=4096

  pid_i_mem_alt += pid_i_gain_alt * pid_error_temp;
  if(pid_i_mem_alt > pid_max_alt)pid_i_mem_alt = pid_max_alt;
  else if(pid_i_mem_alt < pid_max_alt * -1)pid_i_mem_alt = pid_max_alt * -1;

  pid_output_alt = pid_p_gain_alt * pid_error_temp + pid_i_mem_alt + pid_d_gain_alt * (pid_error_temp
- pid_last_alt_d_error);
  if(pid_output_alt > pid_max_alt)pid_output_alt = pid_max_alt;
  else if(pid_output_alt < pid_max_alt * -1)pid_output_alt = pid_max_alt * -1;

  pid_last_alt_d_error = pid_error_temp;
}

void set_gyro_registers(){
  //Setup the MPU-6050
  if(eeprom_data[31] == 1){
    Wire.beginTransmission(gyro_address); //Start communication with the address found during search.
    Wire.write(0x6B);                     //We want to write to the PWR_MGMT_1 register (6B hex)
    Wire.write(0x00);                   //Set the register bits as 00000000 to activate the gyro
    Wire.endTransmission();               //End the transmission with the gyro.

   Wire.beginTransmission(gyro_address);  //Start communication with the address found during search.
    Wire.write(0x1B);                 //We want to write to the GYRO_CONFIG register (1B hex)
    Wire.write(0x08);                   //Set the register bits as 00001000 (500dps full scale)
    Wire.endTransmission();           //End the transmission with the gyro

   Wire.beginTransmission(gyro_address);  //Start communication with the address found during search.
    Wire.write(0x1C);                     //We want to write to the ACCEL_CONFIG register (1A hex)
    Wire.write(0x10);                   //Set the register bits as 00010000 (+/- 8g full scale range)
    Wire.endTransmission();             //End the transmission with the gyro


//Let's perform a random register check to see if the values are written correct
    Wire.beginTransmission(gyro_address); //Start communication with the address found during search
    Wire.write(0x1B);                     //Start reading @ register 0x1B
    Wire.endTransmission();                   //End the transmission
    Wire.requestFrom(gyro_address, 1);     //Request 1 bytes from the gyro
    while(Wire.available() < 1);           //Wait until the 6 bytes are received
    if(Wire.read() != 0x08){               //Check if the value is 0x08
      digitalWrite(12,HIGH);                //Turn on the warning led
      while(1)delay(10);                    //Stay in this loop for ever
    }

    Wire.beginTransmission(gyro_address); //Start communication with the address found during search
    Wire.write(0x1A);                     //We want to write to the CONFIG register (1A hex)
    Wire.write(0x03);                     //Set the register bits as 00000011 (Set Digital Low Pass
Filter to ~43Hz)
    Wire.endTransmission();                 //End the transmission with the gyro

  }
}
```

Software del módulo repetidor: *mod_rep.ino*

```
#include <SoftwareSerial.h>
#include <SPI.h>  //Serial Peripheral Interface (SPI) library
#include <nRF24L01.h>  //radio libraries
#include <RF24.h>      //

//pines de periféricos
SoftwareSerial ModBluetooth(2, 3); // RX | TX
const byte rxAddr[6] = "00001"; // pipe radio
RF24 radio(7, 8);

//varaibles globales
char cadena[100];


void setup() {

 //inicialización del módulo bluetooth
  ModBluetooth.begin(9600);
  //Serial.begin(9600);

 //inicialización del módulo radio
 radio.begin();
 radio.setRetries(15, 15);
 radio.openWritingPipe(rxAddr);

 radio.stopListening(); //para empezar la transmisión se usa .stopListening()
}

void loop() {

  if (ModBluetooth.available())
    {
        char VarChar;

        VarChar = ModBluetooth.read(); //en cada iteración guarda un caracter

        strcat(cadena,&VarChar);   //los va concatenando
        if(VarChar=='#'){   //cuando detecta el token del final ( "#" )

          //Serial.write(cadena);
          radio.write(&cadena, sizeof(cadena));  //envía la cadena por radio

          strcpy(cadena,"");  //la limpia para volver a usarla después
        }

    }
}
```

Software aplicación de smartphone: *Clase DispositivosBT.java*

```
package com.example.jon_1.controlbt;

        import android.bluetooth.BluetoothAdapter;
        import android.bluetooth.BluetoothDevice;
        import android.content.Intent;
        import android.support.v7.app.AppCompatActivity;
        import android.os.Bundle;
        import android.util.Log;
        import android.view.View;
        import android.widget.AdapterView;
        import android.widget.ArrayAdapter;
        import android.widget.ListView;
        import android.widget.TextView;
        import android.widget.Toast;

        import java.util.Set;

public class DispositivosBT extends AppCompatActivity {

    //1)
    // Depuración de LOGCAT
    private static final String TAG = "DispositivosBT"; //<-<- PARTE A MODIFICAR >->->
    // Declaracion de ListView
    ListView IdLista;
    // String que se enviara a la actividad principal, mainactivity
    public static String EXTRA_DEVICE_ADDRESS = "device_address";

    // Declaracion de campos
    private BluetoothAdapter mBtAdapter;
    private ArrayAdapter mPairedDevicesArrayAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_dispositivos_bt);
    }

    @Override
    public void onResume()
    {
        super.onResume();
        //----------------------------------
        VerificarEstadoBT();

        // Inicializa la array que contendra la lista de los dispositivos bluetooth vinculados
        mPairedDevicesArrayAdapter = new ArrayAdapter(this, R.layout.nombre_dispositivos);//<-<-
PARTE A MODIFICAR >->->
        // Presenta los disposisitivos vinculados en el ListView
        IdLista = (ListView) findViewById(R.id.IdLista);
        IdLista.setAdapter(mPairedDevicesArrayAdapter);
        IdLista.setOnItemClickListener(mDeviceClickListener);
        // Obtiene el adaptador local Bluetooth adapter
        mBtAdapter = BluetoothAdapter.getDefaultAdapter();
        // Obtiene un conjunto de dispositivos actualmente emparejados y agregua a 'pairedDevices'
        Set<BluetoothDevice> pairedDevices = mBtAdapter.getBondedDevices();
        // Adiciona un dispositivos previo emparejado al array
        if (pairedDevices.size() > 0)
        {
            for(BluetoothDevice device : pairedDevices) {

                mPairedDevicesArrayAdapter.add(device.getName() + "\n" + device.getAddress());
            }
        }
    }
```

```
/ Configura un (on-click) para la lista
    private AdapterView.OnItemClickListener mDeviceClickListener = new
AdapterView.OnItemClickListener() {
        public void onItemClick(AdapterView av, View v, int arg2, long arg3) {

            // Obtener la dirección MAC del dispositivo, que son los últimos 17 caracteres en la
vista
            String info = ((TextView) v).getText().toString();
            String address = info.substring(info.length() - 17);

            // Realiza un intent para iniciar la siguiente actividad
            // mientras toma un EXTRA_DEVICE_ADDRESS que es la dirección MAC.
            Intent i = new Intent(DispositivosBT.this, UserInterfaz.class);//<-<- PARTE A MODIFICAR
>->->
            i.putExtra(EXTRA_DEVICE_ADDRESS, address);
            startActivity(i);
        }
    };

    private void VerificarEstadoBT() {
        // Comprueba que el dispositivo tiene Bluetooth y que está encendido.
        mBtAdapter= BluetoothAdapter.getDefaultAdapter();
        if(mBtAdapter==null) {
            Toast.makeText(getBaseContext(), "El dispositivo no soporta Bluetooth",
Toast.LENGTH_SHORT).show();
        } else {
            if (mBtAdapter.isEnabled()) {
                Log.d(TAG, "...Bluetooth Activado...");
            } else {
                //Solicita al usuario que active Bluetooth
                Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
                startActivityForResult(enableBtIntent, 1);

            }
        }
    }
}
```

Software aplicación de smartphone: *clase UserInterfaz.java*

```
package com.example.jon_1.controlbt;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Context;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.os.Handler;
import android.provider.Settings;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import android.os.AsyncTask;

import android.support.v4.app.ActivityCompat;
//import android.support.v7.app.AppCompatActivity;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.UUID;

import android.location.LocationListener;
import android.location.Location;
import android.location.LocationManager;

import android.Manifest;

public class UserInterfaz extends AppCompatActivity {

    //1)
    Button IdComSeg, IdDetenerSeg ,IdDesconectar, IdListo, IdApagar;
    TextView IdBufferIn;
    //-----------------------------------------
    Handler bluetoothIn;
    int th=0;
    final int handlerState = 0;
    private BluetoothAdapter btAdapter = null;
    private BluetoothSocket btSocket = null;
    private StringBuilder DataStringIN = new StringBuilder();
    private ConnectedThread MyConexionBT;
    // Identificador unico de servicio - SPP UUID
    private static final UUID BTMODULEUUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
    // String para la direccion MAC
    private static String address = null;
   Location loc1;
  @Override
    protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_user_interfaz);
    //2)
    //Enlaza los controles con sus respectivas vistas
    IdComSeg = (Button) findViewById(R.id.IdComSeg);
    IdDetenerSeg = (Button) findViewById(R.id.IdDetenerSeg);
    IdDesconectar = (Button) findViewById(R.id.IdDesconectar);
    //IdBufferIn = (TextView) findViewById(R.id.IdBufferIn);

    IdListo = (Button) findViewById(R.id.IdListo);
    IdApagar = (Button) findViewById(R.id.IdApagar);
```

```
if (ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
    ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.ACCESS_FINE_LOCATION,},
1000);
} else {
    locationStart();
}

bluetoothIn = new Handler() {
    public void handleMessage(android.os.Message msg) {
        if (msg.what == handlerState) {
            String readMessage = (String) msg.obj;
            DataStringIN.append(readMessage);

            int endOfLineIndex = DataStringIN.indexOf("#");

            if (endOfLineIndex > 0) {
                String dataInPrint = DataStringIN.substring(0, endOfLineIndex);
             //   IdBufferIn.setText("Dato: " + dataInPrint);//
                DataStringIN.delete(0, DataStringIN.length());
            }
        }
    }
};

btAdapter = BluetoothAdapter.getDefaultAdapter(); // get Bluetooth adapter
VerificarEstadoBT();

// Configuracion onClick listeners para los botones
 // para indicar que se realizara cuando se detecte
 // el evento de Click
final enviaGPS env = new enviaGPS();
 IdComSeg.setOnClickListener(new View.OnClickListener() {
     public void onClick(View v)
     {

         MyConexionBT.write("COMANDO%EMPEZAR#");
         if(th==0) {

            // try {
                 env.execute();
                 th=1;
            // } catch (IllegalStateException e) {
             //    Toast.makeText(getBaseContext(), "Error gps", Toast.LENGTH_SHORT).show();
              //}

         }


     }
 });

IdDetenerSeg.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        //env.cancel(true);
        MyConexionBT.write("COMANDO%DETENER#");

       // MyConexionBT.write("Apagar#");
    }
});

IdDesconectar.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        if (btSocket!=null)
        {
            try {btSocket.close();}
            catch (IOException e)
            { Toast.makeText(getBaseContext(), "Error", Toast.LENGTH_SHORT).show();}
        }
        env.cancel(true);
        finish();
    }
});
```

```
IdListo.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {



            MyConexionBT.write("COMANDO%ENCENDIDO#");
            //envío comando para inicialización del dron
            //con esto sale del blucle del setup
        }
    });

    IdApagar.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {



            MyConexionBT.write("COMANDO%APAGADO#");

        }
    });
}

private void locationStart() {
    LocationManager mlocManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
    Localizacion Local = new Localizacion();
    Local.setMainActivity(this);
    final boolean gpsEnabled = mlocManager.isProviderEnabled(LocationManager.GPS_PROVIDER);
    if (!gpsEnabled) {
        Intent settingsIntent = new Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
        startActivity(settingsIntent);
    }
    if (ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions(this, new
String[]{Manifest.permission.ACCESS_FINE_LOCATION,}, 1000);
        return;
    }
    mlocManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0, 0, (LocationListener)
Local);
    mlocManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0, (LocationListener) Local);
    //mensaje1.setText("Localización agregada");
    //mensaje2.setText("");
}
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
    if (requestCode == 1000) {
        if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            locationStart();
            return;
        }
    }
}


private BluetoothSocket createBluetoothSocket(BluetoothDevice device) throws IOException
{
    //crea un conexion de salida segura para el dispositivo
    //usando el servicio UUID
    return device.createRfcommSocketToServiceRecord(BTMODULEUUID);
}
@Override
public void onResume()
{
    super.onResume();

    if(th==0) {
        //Consigue la direccion MAC desde DeviceListActivity via intent
        Intent intent = getIntent();
        //Consigue la direccion MAC desde DeviceListActivity via EXTRA
        address = intent.getStringExtra(DispositivosBT.EXTRA DEVICE ADDRESS);
        //Setea la direccion MAC
        BluetoothDevice device = btAdapter.getRemoteDevice(address);
```

```
try {
            btSocket = createBluetoothSocket(device);
        } catch (IOException e) {
            Toast.makeText(getBaseContext(), "La creacción del Socket fallo",
Toast.LENGTH_LONG).show();
        }
        // Establece la conexión con el socket Bluetooth.
        try {
            btSocket.connect();
        } catch (IOException e) {
            try {
                btSocket.close();
            } catch (IOException e2) {
            }
        }
        MyConexionBT = new ConnectedThread(btSocket);
        MyConexionBT.start();
    }

}

@Override
public void onPause()
{
    super.onPause();

}

//Comprueba que el dispositivo Bluetooth Bluetooth está disponible y solicita que se active si está
desactivado
private void VerificarEstadoBT() {

    if(btAdapter==null) {
        Toast.makeText(getBaseContext(), "El dispositivo no soporta bluetooth",
Toast.LENGTH_LONG).show();
    } else {
        if (btAdapter.isEnabled()) {
        } else {
            Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(enableBtIntent, 1);
        }
    }
}
private class enviaGPS extends AsyncTask<Void, Void, Void> {

    @Override
    protected Void doInBackground(Void... params) {

        while (true){


            String Text = "GEOLOCA%"+ loc1.getLatitude() + "%" + loc1.getLongitude()+"#";

            MyConexionBT.write(Text);

            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            if(isCancelled())
                break;
        }

        return null;
    }
protected void onProgressUpdate(Integer... values) {

}
@Override
protected void onPreExecute() {
}
```

```
protected void onPostExecute(Boolean result) {
        // if(result)
          //   Toast.makeText(MainHilos.this, "Tarea finalizada!", Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onCancelled() {
       // Toast.makeText(MainHilos.this, "Tarea cancelada!", Toast.LENGTH_SHORT).show();
    }
}
//Crea la clase que permite crear el evento de conexion
private class ConnectedThread extends Thread
{
    private final InputStream mmInStream;
    private final OutputStream mmOutStream;

    public ConnectedThread(BluetoothSocket socket)
    {
        InputStream tmpIn = null;
        OutputStream tmpOut = null;
        try
        {
            tmpIn = socket.getInputStream();
            tmpOut = socket.getOutputStream();
        } catch (IOException e) { }
        mmInStream = tmpIn;
        mmOutStream = tmpOut;
    }

    public void run()
    {
        byte[] buffer = new byte[256];
        int bytes;

        // Se mantiene en modo escucha para determinar el ingreso de datos
        while (true) {
            try {
                bytes = mmInStream.read(buffer);
                String readMessage = new String(buffer, 0, bytes);
                // Envia los datos obtenidos hacia el evento via handler
                bluetoothIn.obtainMessage(handlerState, bytes, -1, readMessage).sendToTarget();
            } catch (IOException e) {
                break;
            }
        }
    }

//Envio de trama
    public void write(String input)
    {
        try {
            mmOutStream.write(input.getBytes());
        }
        catch (IOException e)
        {
            //si no es posible enviar datos se cierra la conexión
            Toast.makeText(getBaseContext(), "La Conexión fallo", Toast.LENGTH_LONG).show();
            finish();
        }
    }
}
public class Localizacion implements LocationListener {
    UserInterfaz mainActivity;

    public UserInterfaz getMainActivity() {
        return mainActivity;
    }

    public void setMainActivity(UserInterfaz mainActivity) {
        this.mainActivity = mainActivity;
    }
```

```
@Override
        public void onLocationChanged(Location loc) {
            // Este metodo se ejecuta cada vez que el GPS recibe nuevas coordenadas
            // debido a la detecci—n de un cambio de ubicacion
            loc.getLatitude();
            loc.getLongitude();
            String Text = "Mi ubicaci—n actual es: " + "\n Lat = "
                    + loc.getLatitude() + "\n Long = " + loc.getLongitude();
            //messageTextView.setText(Text);

            //this.mainActivity.setLocation(loc);
            loc1=loc;

        }

        @Override
        public void onProviderDisabled(String provider) {
            // Este mŽtodo se ejecuta cuando el GPS es desactivado
           // messageTextView.setText("GPS Desactivado");
            Toast.makeText(getBaseContext(), "GPS Desactivado", Toast.LENGTH_LONG).show();
        }

        @Override
        public void onProviderEnabled(String provider) {
            // Este mŽtodo se ejecuta cuando el GPS es activado
           // messageTextView.setText("GPS Activado");
            Toast.makeText(getBaseContext(), "GPS Activado", Toast.LENGTH_LONG).show();
        }

        @Override
        public void onStatusChanged(String provider, int status, Bundle extras) {
            // Este mŽtodo se ejecuta cada vez que se detecta un cambio en el
            // status del proveedor de localizaci—n (GPS)
            // Los diferentes Status son:
            // OUT_OF_SERVICE -> Si el proveedor esta fuera de servicio
            // TEMPORARILY_UNAVAILABLE -> Temp˜ralmente no disponible pero se
            // espera que este disponible en breve
            // AVAILABLE -> Disponible
        }
    }


}
```

Software aplicación de smartphone: *nombre_dispositivos.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">

</TextView>
```

Software aplicación de smartphone: *activity_dispositivos_bt.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.jon_1.controlbt.DispositivosBT">

    <TextView
        android:id="@+id/IdTitulo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:text="DISPOSITIVOS VINCULADOS"
        android:textAlignment="center"
        android:textSize="18sp"
        android:textStyle="bold" />

    <ListView
        android:id="@+id/IdLista"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_below="@+id/IdTitulo"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="12dp" />
</RelativeLayout>
```

Software de aplicación de smartphone: *activity_user_interfaz.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.jon_1.controlbt.UserInterfaz">

<TextView
    android:id="@+id/IdControl"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:text="CONTROL DRON"
    android:textAlignment="center"
    android:textSize="18sp"
    android:textStyle="bold" />
```

```
<Button
    android:id="@+id/IdComSeg"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="COMENZAR SEGUIMIENTO"
    android:textAlignment="center"
    android:textSize="18sp"
    android:textStyle="bold"
    android:layout_marginBottom="56dp"
    android:layout_above="@+id/IdDetenerSeg"
    android:layout_alignParentStart="true" />

<Button
    android:id="@+id/IdDetenerSeg"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="DETENER SEGUIMIENTO"
    android:textAlignment="center"
    android:textSize="18sp"
    android:textStyle="bold"
    android:layout_above="@+id/IdDesconectar"
    android:layout_alignParentStart="true"
    android:layout_marginBottom="100dp" />

<Button
    android:id="@+id/IdDesconectar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="32dp"
    android:text="DESCONECTAR"
    android:textAlignment="center"
    android:textSize="18sp"
    android:textStyle="bold" />

<Button
    android:id="@+id/IdListo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="25dp"
    android:background="@android:color/holo_green_dark"
    android:text="Listo"
    android:textStyle="bold"
    android:layout_alignBaseline="@+id/IdApagar"
    android:layout_alignBottom="@+id/IdApagar"
    android:layout_alignParentStart="true" />

<Button
        android:id="@+id/IdApagar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="23dp"
        android:background="@android:color/holo_red_dark"
        android:text="APAGAR"
        android:textStyle="bold"
        android:layout_below="@+id/IdControl"
        android:layout_alignParentEnd="true"
        android:layout_marginTop="49dp" />
</RelativeLayout>
```

Software aplicación de smartphone: *AndroidManifest.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.jon_1.controlbt">

    <uses-permission android:name="android.permission.BLUETOOTH"/>
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>

    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS"/>
    <uses-permission android:name="android.permission.READ_LOGS"/>
    <uses-permission android:name="android.permission.INTERNET"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="ControlBT"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <activity android:name=".DispositivosBT">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name=".UserInterfaz"></activity>

    </application>

</manifest>
```