UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Control de un dron de cuatro hélices con sistema Linux empotrado.

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Carbonell Lázaro, Rafael

Tutor: Simó Ten, José Enrique

Curso 2017-2018

Resum

L'objetiu del projecte és l'aplicació de tècniques de control de motors i adquisició d'informació sensorial en sistemes encastats que executen Linux. "Embedded Linux" és una distribució que combina les versatilitats de Linux amb característiques especials de acces al hardware y resposta temporal. En el marc del projecte es realitzarà un prototip de control de un dron amb sensors de processament de senyals i acceleròmetres i el corresponent software de control i comunicació.

Paraules clau: Sistemes Empotrats, Linux, Sistemes Distribuits, Control, Drones.

Resumen

El objetivo del proyecto es la aplicación de técnicas de control de motores y adquisición de información sensorial en sistemas empotrados que ejecutan Linux. "Embedded Linux" es una distribución que combina las versatilidad de Linux con características especiales de acceso al hardware y respuesta temporal. En el marco del proyecto se realizará un prototipo de control de un dron con sensores de procesamiento de señales y acelerómetros y el correspondiente software de control y comunicaciones.

Palabras clave: Sistemas Empotrados, Linux, Sistemas Distribuidos, Control, Drones.

Abstract

The objective of the project is the application of motor control techniques and acquisition of sensory information in embedded systems running Linux. "Embedded Linux" is a distribution that combines the versatility of Linux with special features of hardware access and temporary response. Within the framework of the project, a prototype of a drone control will be made with signal processor sensors and accelerometers and the corresponding control and communications software.

Key words: Embedded Systems, Linux, Distributed Systems, Control, Drones.

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

Este trabajo fin de grado consiste en exponer el trabajo de final de grado que consiste en realizar un programa capaz de controlar la 'estabilidad' de un dron de cuatro hélices funcionando sobre un sistema operativo Linux en una ordenador empotrado.

La particularidad de este programa es que se ejecuta sobre un sistema distribuido de procesos en red (Núcleo de Control). Este se basa en comunicar procesos en un entorno distribuido. Una posible analogía a este concepto sería considerar el programa como un conjunto de bloques de LEGO, entonces nuestro programa estaría formado por pequeños bloques de procesos. Estos pequeños bloques realizarían una tarea y se comunicarían entre ellos a través de un tema en común. De esta forma tendríamos un programa completo y en módulos en el cual no tenemos la necesidad de tener todos los procesos en funcionamiento.

Como se exponía en el primer párrafo, la intención es controlar la estabilidad del dron, y para ello, aplicaremos conceptos de control junto con aspectos de electrónica e informática aplicada a la robótica aérea.

Motivación

La industria de los vehículos aéreos no tripulados es un sector que esta en alza, y que en consecuencia, creará muchos puestos de trabajo en el futuro. Actualmente e históricamente el campo de los UAV ha ido acompañado de la industria militar, aunque en estos últimos años su uso a nivel civil ha ido creciendo.

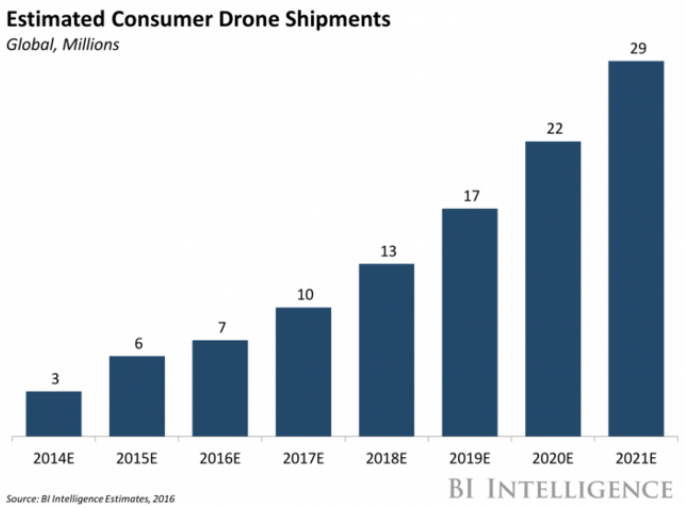


Figura 1.1: Gráfica del consumo comercial estimado de drones basada en el año 2015.

La motivación de este proyecto viene dada de retomar un proyecto anterior, empece cuando era adolescente, cuando se despertó el interés por formar parte de la comunidad 'Hazlo tu mismo', el termino es más conocido en ingles 'Do It Yourself'. Uno de mis proyectos personales con mayor grado de dificultad fue la construcción de un dron.

En aquel momento, empece con el montaje mirando tutoriales por Internet, viendo drones construidos por otra gente, hasta el momento que me lance a la piscina y compre todas piezas pensando en que seria fácil realizar el proyecto, (ingenuo de mi). Después de todo, alguna vez, conseguí volar hacer volar el dron, aunque sufrió algún que otro accidente.

Con el pasó del tiempo, he querido retomar este proyecto. Hablé con mi tutor de sobre el proyecto anterior, y de realizar

y teniendo más de conocimiento sobre las materias de control, sentí la necesidad de adentrarme en el mundo del control en el vuelo, y teniendo el drone ya montado, decidí retomar este proyecto, aunque dándole otro enfoque, realizando mi propio software de vuelo, aprendiendo los pilares básicos de este tipo de aeronaves y aportando mi grano de arena.

Objetivos

El objetivo principal del proyecto es la realización de un software de control de estabilidad para un dron de cuatro hélices sobre un sistema operativo empotrado de propósito general. Este objetivo está relacionado con los siguientes objetivos academicos:

- Aprender sobre aspectos de control básico aplicados.
- Aprender aspectos multidisciplinarios a nivel eléctrico y electrónico.
- Aprender aspectos de tiempo real.
- Aprender más sobre el sistema operativo Linux.

– Conocer el mundo de los sistema distribuidos y empotrados.

Estructura de la memoria

El documento se organiza en dos bloques principales: la parte teórica y la parte práctica. La parte teórica abarca el capitulo 3 y parte del 4, ya que los aspectos de diseño y construcción implican aspectos teóricos de conocer la electrónica y componentes, junto con los distintos protocolos de comunicación. Por otra parte, la parte práctica del proyecto se plantea a partir del capitulo 4 y 5, junto con el apéndice. Estos tratan de explicar la estructura de software empleada para organizar el código y parte del código empleado.

CAPÍTULO 2
Historia

El dron, o también conocido como «drone» en habla anglosajona, proviene del concepto de una aeronave no tripulada, y históricamente, remotamente controlada desde tierra, aunque con el tiempo, ha ido evolucionando a lo largo de su historia adquiriendo nuevas categorías y matices.

En relación con la historia de los drones es más extensa de la que cabe esperar, se remonta a mediados del siglo XIX. La aviación de este tipo de aeronaves ha sido empleado con fines militares hasta a principios de siglo XXI, cuando se popularizó su uso civil. Daremos un repaso a lo largo de la historia para ver su evolución.

Siglo XIX

A lo largo del siglo XIX surgieron los primeros modelos europeos construidos y en funcionamiento, realizados por inventores como Cayley, Du Temple, Langley, entre otros. Cabe destacar uno de los primeros usos de las aeronaves no tripuladas, fue realizado por los austriacos. En 1849, utilizaron 200 globos aerostáticos no tripulados cargados de bombas sobre la ciudad italiana de Venecia.

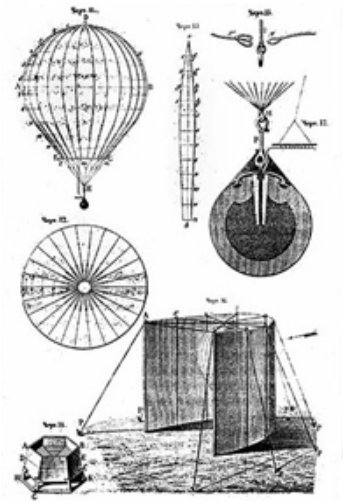


Figura 2.1: Los globos austriacos.

Decadas más tarde, en 1896, Samuel Pierpont Langley construyó varios de aviones no tripulados propulsados a vapor, que realizaron con éxito un tramo sobre el río Potomac, Washington D. C.,Estados Unidos. En 1898, el espionaje aéreo para vigilar al enemigo apareció en la guerra entre Hispanoamericano y Norteamérica cuando los militares de Estados Unidos utilizaban cometas equipadas con una cámara para realizar fotografías del bando enemigo.

En ese mismo año, Nikola Tesla presentó el "Teleautomaton", una maquina naval con la capacidad de avanzar, pararse, virar a la derecha o izquierda e incluso encender o apagar sus luces. Todas estas funcionalidades eran controladas a través de diferentes frecuencias de radio. A Tesla se le atribuye ser el padre de la tele-operación, y en consecuencia, de los sistemas de comunicación de los misiles crucero.

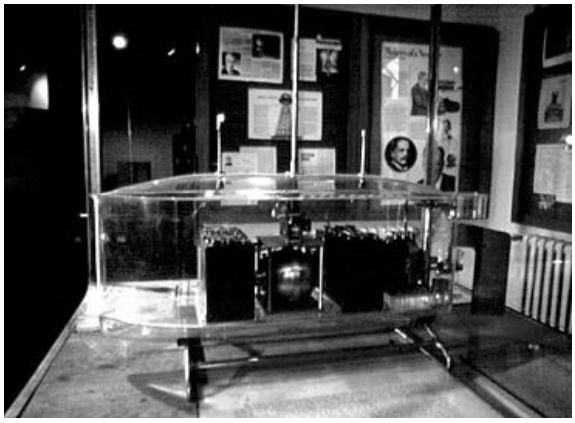


Figura 2.2: "Teleautomaton".

Primeras dos décadas del siglo XX

Esta etapa se caracteriza por desarrollar aeronaves de vuelo con cargas explosivas actuando como un avión kamikaze hacia un blanco determinado, a este tipo se les considero misiles crucero.

Los pioneros en el campo de la aviación fueron los europeos. Fueron los primeros en desarrollar los principio aeronáuticos, que hicieron viable crear aeronaves no tripuladas consideradas como las primeras de la historia. Las naciones no europeas seguían de cerca una progresión común. Aunque a principios de siglo XX, los Estados Unidos les tomaría el relevó.

Para los europeos, la evolución de este tipo de aeronaves se vio truncado al no tener desarrollados motores con insuficiencias de potencia por tener un peso excesivo. En 1903, en los Estados Unidos,con la aparecieron los hnos. Wright y el mecánico del grupo, Charles Taylor, con un motor a modo de propulsor refrigerado por agua, con doce caballos de potencia (CV), cuatro cilindros y ochenta Kg de peso. Con varios vuelos de prueba con su Wright Flyer, fueron capaces de recorrer unos 260 metros en 59 segundos, toda una hazaña para aquella época.

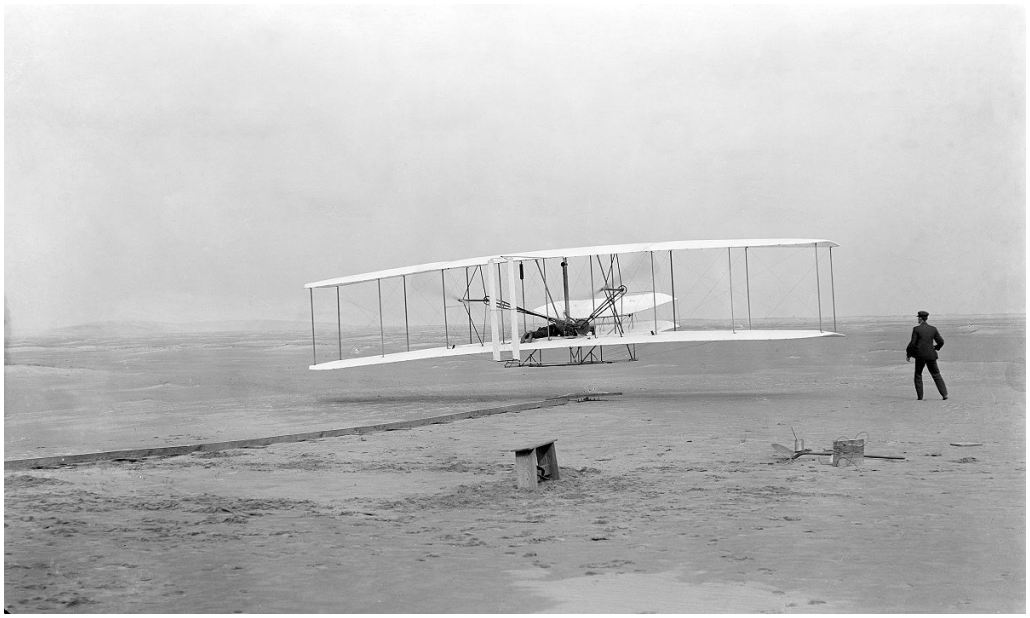


Figura 2.3: Wright Flyer.

En 1909, Elmer Ambrose Sperry fue quien sentó las bases para los sistemas de estabilización automática en una avión no tripulado. Sperry realizó unas experimentos exitosos con giroscopios para servicios navales, que le propiciaron a desenvolver un giro estabilizador. En 1911, respaldado por otro maestro de las aeronaves, Glenn hammond curtiss, mejoraron su invento con un sistema mucho más pequeño, y permitía la estabilización en el plano de vuelo del avión.

A lo largo de la primera guerra mundial (1914-1918), se amplio el uso de aeronaves para vigilancia, de esta forma podían formar mapas de movimientos de los enemigos. Al mismo tiempo, las aeronaves tripuladas se desarrollaron con gran rapidez, en cambio, las no tripuladas perdían fuerza por la falta de desarrollo tecnológico en sistemas de control por radio, estabilidad y navegación automatizada.

En 1914, Sperry obtuvo un galardón en una presentación en Francia por el giro estabilizador. En 1915, se relaciono con un inventor en desarrollos eléctricos, Peter Cooper Hewitt, para llevar a cabo las ideas de comunicación usadas en el Teleautomaton de Tesla en el invento de Sperry. En 1916, se utilizó por primera vez el artefacto de Sperry para la estabilización y navegación de un avión tripulado, (el Hewitt-Sperry Automatic Airplane). La demostración consistía en que el piloto, una vez en el aire, encendía el auto-piloto. Luego, la aeronave describía la trayectoria pre-programada y cuando terminaba el programa, el avión realizaba un picado, entonces el piloto recuperaba el mando de la aeronave en ese momento y aterrizaba en la pista. En 1917, la US Navy fomentó el proyecto dando cinco hidroaviones Curtiss Model N para realizar mas experimentos.

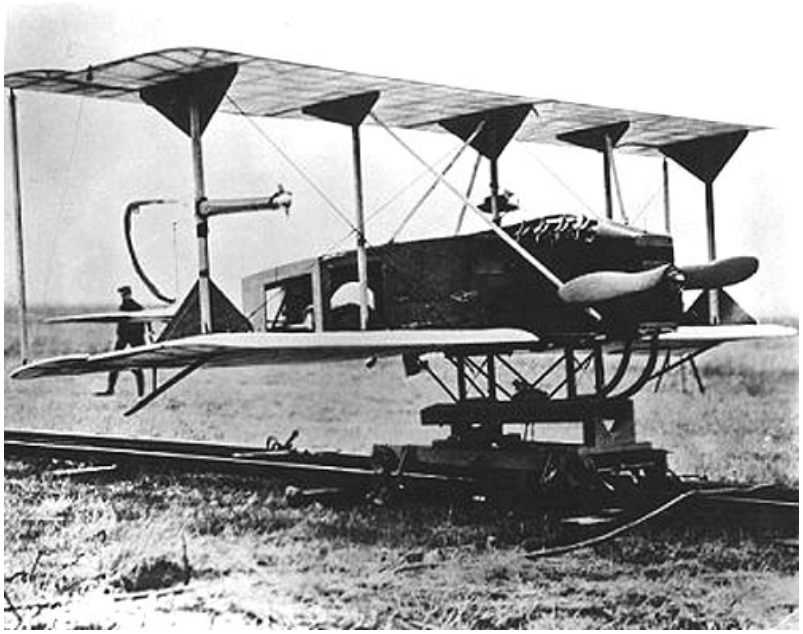


Figura 2.4: Hewitt-Sperry Automatic Airplane.

Al mismo tiempo, Curtiss Aeroplane and Motor company, una empresa estadounidense dedicada a la fabriacación de aviones, emprendió la elaboración de una unidad de navegación aérea para un proyectil no tripulado con la ayuda de una catapulta. La primera prueba con éxito de una aeroplano no tripulado tuvo lugar en marzo de 1918. El artefacto era una aeronave no tripulado de madera, pesaba doscientos setenta kilogramos, incluyendo los explosivos y el motor de propulsión de Ford de cuarenta caballos de fuerza. El método para la elección de la trayectoria de la aeronave equipada con un proyectil se realizaba obteniendo la velocidad del viento y la distancia al blanco. Con estos datos, se calculan las revoluciones necesarias del motor para lograr impactar contra el objetivo. La aeronave mantenía la estabilidad con un giroscopio y un barómetro, contaba las revoluciones dadas por el motor y una vez alcanzadas las deseadas, se desprendía de las alas del fuselaje, cayendo el resto con la bomba en su interior hacia el blanco.

El proyectil de Sperry fue desarrollado con la intención de ser armamento de largo alcance no tripulado, y junto con este fin, encontramos la aeronave llamada Kettering Bug de Charles Kettering construido en 1918 y el Aerial Target de origen británico. Este último fue comenzado en 1914, era una aeronave no tripulada manejada por radiofrecuencia con un motor de treinta y cinco hp de potencia. El Aerial Target sirvió como blanco de entrenamiento y reafirmo la idea planteada por Tesla del uso de señales de radio para la comunicación con la aeronave, por ello, se utilizaron como sistema para guiar a la aeronave hasta su blanco. Por otro lado, la maquina de Charles Kettering, de la empresa General Motors, es un aeronave no tripulado con programa predefinido en tierra. Como ya sabemos, la aeronave contenía una carga explosiva, el mecanismo trataba de volar hasta el objetivo, cuando estaba próximo, se plegaban las alas y caía en picado como si de una bomba aérea se tratase. La activación del mecanismo de plegado se activaba por un reloj pre-programado. No obstante, estos artefactos no tenían la precisión suficiente para ser utilizados con fines bélicos durante la I Guerra Mun-

dial. Aunque si es cierto que sentaron las bases tecnológicas de las aeronaves no tripuladas, todavía quedaba un largo recorrido para mejorarla precisión de este tipo de sistemas.

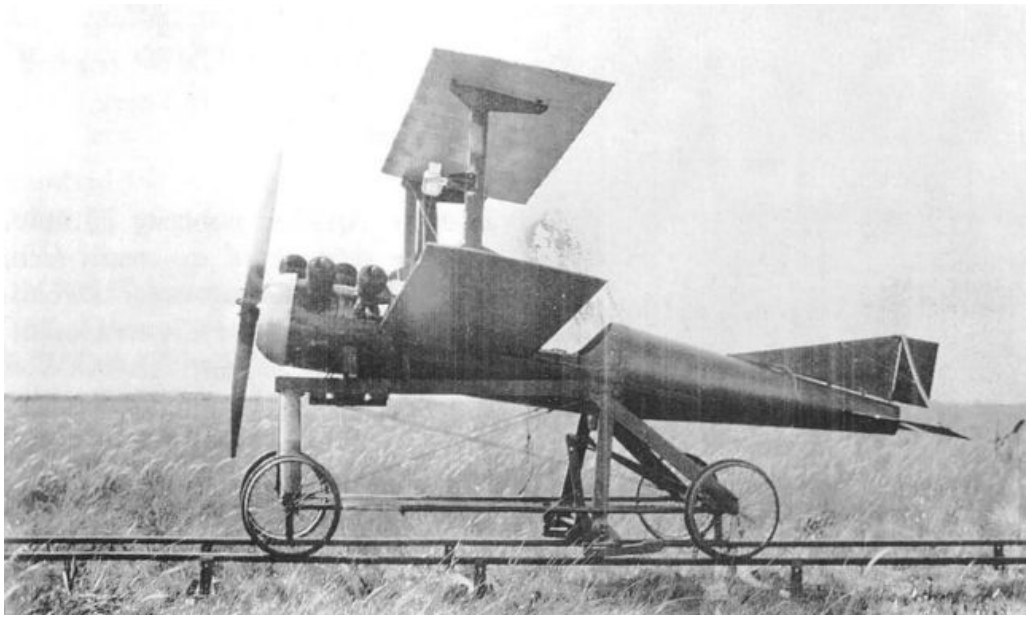


Figura 2.5: Kettering Bug.

Llegados los años 20, la Royal Aircraft Establishment de la armada británica reactivó el desarrollo tecnológico por aeronaves no tripuladas. Estos desarrollaron una aeronave con la capacidad de transportar ciento catorce kilogramos de explosivo, podía recorrer aproximadamente unos cuatrocientos ochenta kilómetros, con una potencia de doscientos hp gracias a su motor de propulsión Armstrong Siddeley Linx, en consecuencia, recibio el nombre de LARYNX (Long Range Gun with Lynx engine). Estaba equipada con un sistema de comunicación por radiofrecuencia, además de disponer de su propio sistema de navegación automático. La primera prueba se realizo con éxito en 1927.

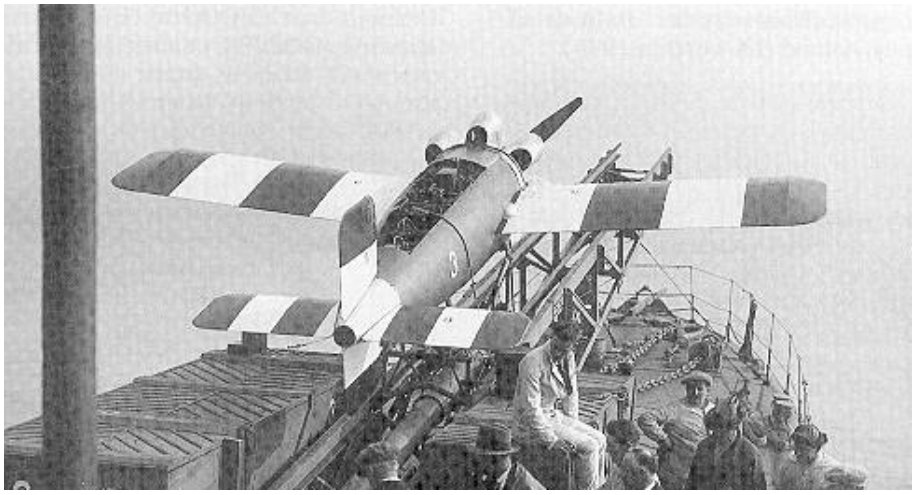


Figura 2.6: LARYNX.

La segunda guerra mundial

Durante la década de los años 30, tenemos a las tres grandes potencias mundiales, como son Gran Bretaña, Estados Unidos y Alemania, desarrollando el concepto de drone como sistemas de vuelo automáticos.

Por una parte esta Gran Bretaña, esta decide abandonar la etapa de los misiles crucero por aeronaves como objetivo aéreo de entrenamiento controlados exclusivamente por radio. Entre los años 1934 y 1943, se construyó un nuevo objetivo, el Queen Bee, un aeronave que surgio a partir del antiguo modelo Fairey Queen. El Queen Bee era empleado como blanco aereo de entrenamiento por el ejercito britanico terreste y la armada britanica.

Al mismo tiempo, la empresa RadioPlane Company desarrolló la serie de modelos RP. En 1935 nace el prototipo RP1, en 1938 se consolida con RP2, en 1939 con unas ligeras modificaciones, el RP3. Tras diversas demostraciones al ejercito, en 1940 el ejercito firma un contrató con RadioPlane Company por su nuevo modelo mejorado RP4, rebautizado por el ejercito con el nombre de Radioplane OQ-2, siendo el primer drone producido en masa en Estados Unidos. Este modelo estaba controlado por radio exclusivamente y era utilizado para el entrenamiento de blanco aéreo de la fuerzas armadas americanas.

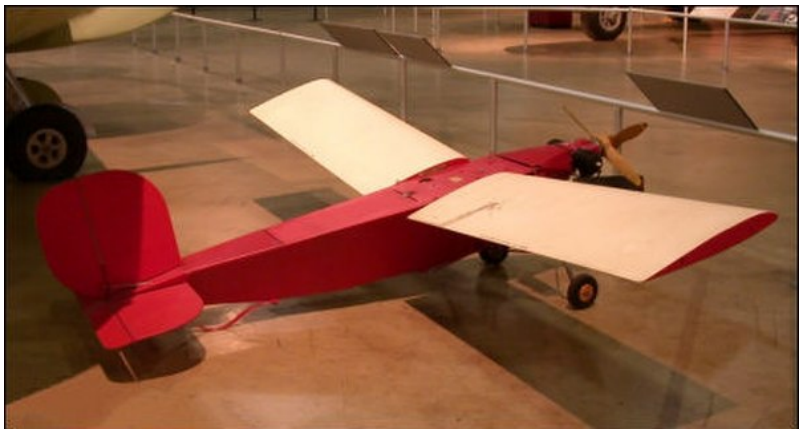


Figura 2.7: Radioplane OQ-2.

A su vez, la Alemania nazi revisó la idea de los misiles crucero con su nueva arma el V1 Vengeance Weapon, con un nuevo sistema de propulsión, un motor de reacción. El sistema de navegación hasta el blanco se había mejorado un poco respecto a los anteriores. Se calculaba la distancia hasta objetivo a través de un barómetro y un anemómetro, estimando la velocidad, la altura y la distancia.

Llegados a este punto, estalla la Segunda Guerra Mundial y por esta razón cambio la visión para la utilización de los drone. La Marina de los Estados Unidos, lanzando una nueva operación llamada .Operación Anvil". Este operación trataba de detectar la ubicación de los bunkers situados en Alemania y parte de la Francia conquistada por los nazis. La intención era destruirlos empleando bombarderos modificados con sistemas de control para radio y llenos de carga explosiva, ya que los bombarderos podrian llevar una carga mayor. Aun así, no disponía de sistema para despegar automáticamente el aparato, y para ello, empleaban pilotos para despegar hasta poner la aeronave a una altura determinada.

Este programa tuvo resultados pésimos, dado que las aeronaves estallaban en pleno vuelo o no alcanzaban su objetivo. En cambio, en Alemania, los científicos nazis quisieron empezar con el programa de desarrollo de misiles. Debido a los resultados pésimos del programa con aviones no tripulados por control remoto.

Por otra parte, Gran Bretaña desarrolla el GB-1 Glide. Una aeronave planeador con la capacidad de escapar a las defensas antiaéreas alemanas y con capacidad de transportar de una carga entre 1000 y 2000 libras de peso. La aeronaves estaba hecha con madera de contrachapado con servomotores que controlan el giro o la elevación por radio. En 1943, se lanzó un ataque con 108 GB-1 Glide a la ciudad alemana de Colonia.

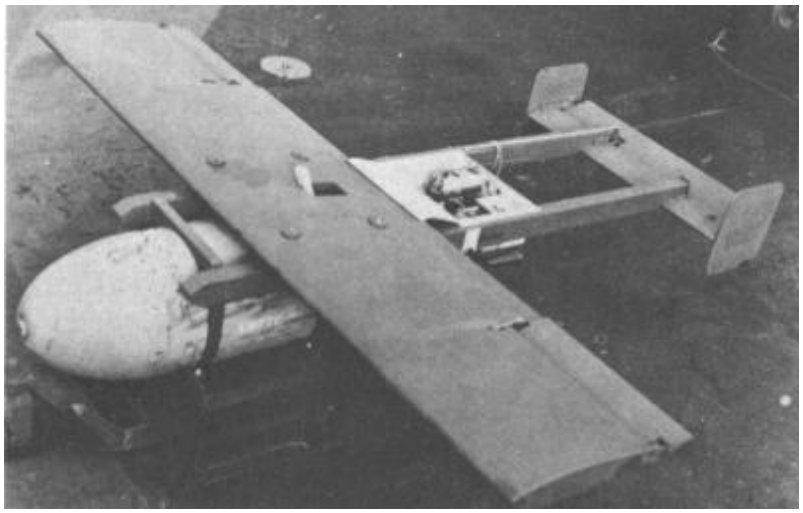


Figura 2.8: GB-1 Glide.

Los años 50

En los años posteriores a la guerra, surgieron varios proyectos paralelos para desarrollar aeronaves no tripuladas para usarlos como blancos para entrenamiento militar. Con la adquisición de Radioplane, a manos de Northrop, desarrollaron uno de estos proyectos con exito. Crearon varias aeronaves, cada cual con mejor, con Shelduck a finales de los 40 o el Falconer a finales de los 50. Ambos eran utilizados como aeronaves no tripuladas para entrenar el blanco, el control estaba basado en linea de visión con control por radio, esta clase de aeronaves recibió el nombre de BTT (Basic Training Target) y sus posteriores hermanos fueron producidos hasta la década de los 80 con métodos de propulsión mas avanzos.

Figura 2.9: Falconer.

El proyecto más importante fue la aeronave que era soltada en el aire llamada Q-2 de la empresa aeronáutica Ryan. Este modelo fue el más notable de la época adquiriendo la distinción de aeronave diana. Posteriormente, en 1951, realizó su primer vuelo con éxito y fue rebautizado como el Firebee, que tenía una autonomía de dos horas de vuelo y podía alcanzar una altura aproximada de 60.000 pies.

El Firebee, como otros UAV, también eran usados en misiones para fotografiar a los enemigos en su territorio y eran controlados por radio en linea de visión desde un puesto de control en tierra o desde un avión que volaba a mayor altitud. Estas aeronaves no tripuladas al tener una envergadura mas reducida que los aviones tripulados, esta característica tenia dos ventajas, tener una menor probabilidad de ser descubiertos en tierras enemigas y evitar los posteriores incidentes diplomáticos.

Por último, tenemos otra peculiaridad del momento, y es el desarrollo de señuelos antiradar llamados Crossbows. Eran lanzados en pleno vuelo desde otro avión sobre el territorio enemigo para confudir los sistemas antiradar.

Figura 2.10: Firebee.

Los años 60

La década de los 60 está marcada por la guerra del Vietnam que fue un hecho histórico que propulso el desarrollo de aeronaves no tripuladas empleadas para el campo de Batalla, aunque todavía seguirán siendo empleadas como blanco.

En la guerra del Vietnam, los drones de esta época consiguieron los sueños de aquellos que imaginaban aeronaves automáticas a principios de siglo. Los sistemas desarrollados a partir de esta guerra fueron cámara de visión en tiempo real en la aeronave para sobrevolar las tierras y selvas vietnamitas y así obtener información del enemigo. El Departamento de Defensa de los EE.UU. tenia intereses por introducir nueva tecnología en el campo de batalla y había mejorado en parte a la introducción en la batalla de sistemas electrónicos de soporte con sensores remotos junto con sistemas informáticos en las bases militares para espiar con esfectividad los movimientos del enemigo así como trazar su situación actual.

El modelo Firebee sufrió varios modificaciones y cambios de nombre. Estas versiones mejoradas eran lanzados desde una nave colmena llamada LockHeed CC-130 Hercules desde la cual se coordinaban las naves lanzadas. Las naves insecto seguían una trayectoria programada o podían ser controlados por radio a través de imágenes de vídeo desde el Hércules. El aterrizaje se producía en tierra y posteriormente recuperado.

Por otra parte, Estados Unidos empezó a enviarlos en misiones de vigilancia a Cuba. Al ver el exito en las operaciones, se utilizaron en nuevos paises de la Guerra Fría, como Corea del norte o China. Por otra parte, en la guerra del Vietnam, se utilizaron los nuevos modelos Firebees apodado como Lightning Bug. Cada vez era mayor el grado de tecnologia introducido en el campo de batalla. Estos aviones estaban equipado con sistemas de radio de largo alcance, que se empleaba como un asistente la localización usando triangulación de las señales de radio al tener una potente comunicación con la aeronave sin apenas interferencias.

Los años 70

Durante los años 70 aumenta el uso de drones para operaciones de vigilancia y reconocimiento del enemigo a causa de la aumento de la tensión de la Guerra Fría. En consecuencia, a este tipo de aeronaves se les exigió mejores requisitos en la fiabilidad del aparato, tanto en el vuelo como en el despegue y el aterrizaje, además de desarrollar nuevos sistemas de comunicación con mayor robustez.

Por una parte, se desarrollaron aeronaves de corto alcance como el LockHeed Aquila. Este proyecto destaca por tener unos requisitos muy ambiciosos para el grado tecnologico del momento. Se basaba en desarrollar una aeronave modular de cuatro piezas que pudiera ser llevada por cuatro soldados. Los requisitos exigidos eran que pudiera volar semi autónoma asistida por radio, ser capaz de localizar los blancos enemigos tanto de día como de noche o de designar blancos mediante un láser para las tropas de artillería terrestres.

Debido al grado tecnológico de la época este proyecto tubo muchos problemas de desarrollo a causa de que los sistemas que componían el avión eran demasia-

do grandes y pesados, además de problemas de consumo excesivo y de tener interferencias entre los diferentes sistemas. Aun así, se consiguió desarrollar un programa viable de vuelo, aunque ahora el problema principal para este tipo de aeronaves, era la recepción del aparato, ya se realizaba con éxito el despeje pero el aterrizaje tanto con paracaídas como con una red de captura causaban daños a la estructura de la aeronave. No obstante, algunos de los sistemas desarrollados se desarrollaron de forma exitosa en la capacidad de reconocimiento de blancos y la vigilancia del enemigo.

Los años 80

Durante los años 80 predomino la serie Canadiar que mejoros las aeronaves de vigilancia aerea asistidos via radiofrecuencia para observar los movimientos enemigos. Estaban equipados con camaras de television e infrarrojas para observación diurna y nocturna de los objetivos. Tenian un diseño de pequeña envergadura para evitar ser detectados. El autopiloto seguia estando basado en un autopiloto asistido por radio.

Esta etapa se caracterizó por la mejora de los sistemas de navegación introduciendo un nuevo sistema autonomo de control en vuelo, la inclusión de nuevos sensores debido a la reducción del tamaño y peso de los sensores junto con la mejora de la precisión. Las aeronaves adquirieron características de puente de comunicaciones en lugares de dificil acceso, guiado láser de objetivos enemigos

Los años 90

La mayor disponibilidad del sistema de posicionamiento global (Global Positioning System; GPS) y de las comunicaciones satélite liberó a los UAS de operar dentro del alcance de la señal de radio y de los sistemas de navegación inexactos basados en giróscopos y datos de aire. De esta forma, junto con los sistemas digitales de control de vuelo (Digital Flight Control System; DFCS), el alcance y la precisión de la navegación mejoraron apreciablemente. Como resultado se desarrollaron sistemas de medio y largo alcance. Los primeros caracterizados por el Seeker de Denel y los últimos por el Gnat de General Atomics.

El Gnat, propulsado por un motor alternativo, se considera el precursor de los actuales sistemas de media altitud/gran autonomía (Medium Altitude Long Endurance; MALE) y elevada altitud/gran autonomía (High Altitude Long Endurance; HALE). Sus actividades operacionales comenzaron a mediados de los años noventa sobre Bosnia y Croacia en tareas de reconocimiento con sensores EO/IR (visible e infrarrojo) y, más tarde, con equipamiento SIGINT. el sistema ha conocido versiones posteriores (A, B y C). Las experiencias con el Gnat pavimentaron el camino para la llegada, a finales de esta década, del MALE UAS Predator y del HALE Global Hawk de Northrop-Grumman, en la década siguiente.

Las operaciones con el Gnat, y las primeras de Predator, permitieron llevar a cabo misiones de reconocimiento a mayores altitudes que en el pasado, lo cual ofrecía protección ante posibles detecciones y fuego desde tierra. Sin embargo al

no ser capaces de detectar a través de las nubes con los sensores eo/Ir que les equipaban, eso les obligaba a tener que descender y ser más vulnerables.

Esta década también fue testigo del desarrollo e introducción en Japón del primer modelo de producción a gran escala de un VTOL (Vertical Take-Off and Landing; aeronave de despegue y aterrizaje vertical): el Yamaha R50, y su modelo mayor, el R.Max. este sistema ha sido muy usado para la siembra de campos de arroz y la fumigación, y ha sido un éxito no sólo por su eficacia en su misión sino también por la colaboración con las autoridades reguladoras, las cuales han facilitado su puesta en operación. Aproximadamente se han llegado a fabricar unos 1.500 sistemas hasta la fecha.

El siglo XXI

En el siglo XXI, la tecnología permitió fabricar aeronaves de mayor tamaño que podían tener una mayor autonomía y mayor alcance de actuación. Con la llegada de los radares de apertura sintética, capaces de hacer lecturas a gran altitud y con gran precisión, además de no ser fácilmente detectables por el enemigo. Con la mejora de esta tecnología aumento el uso de este tipo de aeronaves para fines militares. En particular el General Atomics Predator, el Northrop Global Hawk y el Scan Eagle de Boeing. Otro avance al respecto fue la aparición de misiones de vigilancia aérea empleando este tipo de aeronaves para detectar la existencia de enemigos e incluso estaban equipadas con armamento para posible ataques.

A su vez, sigue habiendo un gran interés en los sistemas capaces de despegar verticalmente a causa de las ventajas que proporciona respecto a las aeronaves de ala fija, debido a que las labores de vigilancia e identificación enemigos junto con la facilidad para el despegue y aterrizaje con respecto a aeronaves de ala fija, las cuales es necesario de pista de aterrizaje o sistema de recuperación como redes.

Pasada la década, la tendencia comercial en el mercado de la robótica estaba en constante crecimiento. Junto con la tecnología cada vez más asequible y de menor tamaño, fomento el desarrollo de aeronaves de carácter comercial al salvar los principales obstáculos que tenia la aviación de radiocontrol.

Figura 2.11: Amazon Prime Air

Por último, cabe destacar que los drones se han empezado a usar para varios mercados como la agricultura, la protección forestal o el reparto de medicinas y paquetes. Estas aeronaves son ideales para el aprendizaje en gran variedad de disciplinas, ya que entrenan la percepción visual, espacial e incluso intelectual si profundizas en sus matemáticas y mecánicas. Con una cámara podemos realizar fotografías aéreas de una parcela, realizar estudios de terreno, es decir, se abre un abanico de posibilidades posibles. Hay grandes empresas que ya apuestan por ello como es Amazon que esta implantando un servicio de distribución de paquetería para sus clientes.

CAPÍTULO 3
Aspectos teóricos de los drones

Vehículos aéreos no tripulados

El criterio para la clasificación de este tipo de vehículos más común es el peso porque tienen relación con la potencia y el tamaño de las aeronaves, además que estas características van unidas con la funcionalidad y la aplicación para la cual fueron concebidas. Daremos un vistazo a la actualidad tanto en el ámbito militar como en el comercial.

Northrop Grumman RQ-4 Global Hawk

Figura 3.1: Northrop Grumman RQ-4 Global Hawk.

General Atomics MQ-9 Reaper

Figura 3.2: General Atomics MQ-9 Reaper.

General Atomics MQ-1 Predator

Figura 3.3: General Atomics MQ-1 Predator.

Boeing Insitu ScanEagle

Figura 3.4: Boeing Insitu ScanEagle.

DJI Agras MG-1S

Figura 3.5: DJI Agras MG-1S.

DJI Mavic pro

Figura 3.6: DJI Mavic pro.

Tabla 3.1: Características

Modelo	Northrop Grumman RQ-4 Global Hawk	General Atomics MQ-9 Reaper	General Atomics MQ-1 Predator
Peso sin carga	6.781 Kg	2.220 Kg	512 Kg
Peso con carga	14.628 Kg	4.760 Kg	1.020 Kg
Longitud	14,5 m	11 m	8,20 m
Altura	4,7 m	3,8 m	2,1 m
Envergadura	39,9 m	20 m	14,8 m
Autonomía	32 h	14 h	24 h
Aplicación	Militar	Militar	Militar

Modelo	Boeing Insitu ScanEagle	DJI Agras MG-1S	DJI Mavic Pro
Peso sin carga	16 Kg	10 Kg	0,734 Kg
Peso con carga	22 Kg	24,8 Kg	0,743 Kg
Longitud	1,65 m	1,471 m	0,083 m
Altura	- m	0,482 m	0,083 m
Envergadura	3,10 m	1,471 m	0,198 m
Autonomía	24 h	0,36 h	0,4 h
Aplicación	Militar	Agrícola	Entretenimiento

Como podemos apreciar en las tablas 3.1., las aeronaves con mayor peso y tamaño son aquellas que su aplicación es militar. Este tipo de aeronaves de gran envergadura se caracterizan por ser naves utilizadas para vigilancia aérea o ataques aéreos a los enemigos, como se viene viendo a lo largo de la historia. Esta no es la razón por la cual queremos desarrollar un drone, nuestra intención se basa en desarrollar un drone de pequeña envergadura para uso comercial. Por ello, nuestra decisión es aproximarnos a un modelo más bien parecido a los que produce la empresa DJI a nivel civil como micro drones de consumo.

Tipos de micro vehículos aéreos

Una vez elegido la envergadura y la aplicación a la cual vamos a enfocar el desarrollo de un drone, debemos conocer un poco los diferentes tipos de drone. Hasta el momento en el trabajo, habíamos visto drones basados en una estructura de avión, es decir, de ala fija y por otra parte, basados en rotores, veremos que hay más diversidad en este tipo de micro aeronaves. Trataremos de abarcar este aspecto según la siguiente clasificación.

· Ala fija
· Ala móvil
  - Insecto
  - Ave
· Rotores
  - Helicóptero
  - Tricóptero, Cuadricóptero, Hexacóptero

Las aeronaves de ala fija se basan en el concepto de avión, que deriva de la idea de una ave con las alas fijas que es impulsada por un motor con una helice o a propulsión, o incluso sin motor como los planeadores. Este grupo puede ser usada para diversión personal, aunque hay nuevas aplicaciones recientes de emplearlas para enviar medicamentos en zonas rurales con difícil acceso con mayor facilidad. En este grupo podemos encontrar desde modelos basados en aeronaves clásicas o en modelos más modernos.

(a) Modelo clásico - Avioneta

(b) Modelo actual - Parrot disco

Figura 3.7: Aeronaves de ala fija

Las aeronaves de ala móvil son la menos comunes y suelen estar inspirados en animales como aves o insectos. Una empresa muy conocida en el sector de la robótica animal es Festo, esta se dedica a desarrollar robots basados en el movimiento de los animales y entre ellos tiene dos robots aéreos basados en una gaviota, una mariposa y un murciélago. Aunque si es cierto que la venta de este tipo de robots todavía no ha llegado al mercado al esta en una etapa muy temprana de desarrollo.

(a) Gaviota

(b) Mariposa

(c) Murciélago

Figura 3.8: Aeronaves de ala móvil

Por último, tenemos las naves con rotos, también conocidas como aeronaves de ala giratoria, porque generan la fuerza empuje para sustentarse debido a unas

palas que giran al estar unidas al eje del motor de la aeronave. Un clásico de este tipo de este tipo de aeronaves es el helicóptero con un gran rotor central y otro pequeño en la cola. Por otra parte, tenemos los tricópteros, cuadricópteros, etc, que son un grupo de aeronaves de vuelo basada en múltiples motores y como mayor maniobrabilidad en el aire pero con mayores consumos.

(a) Cuadricóptero

(b) Octocóptero

(c) Helicóptero

Figura 3.9: Aeronaves basada en rotores

Mecánica de vuelo

Para este apartado consideramos que se debe tener alguna noción previa de geometría, sistemas de coordenadas cartesianas y trigonométria. Para representar el espacio tenemos el sistema cartersiano, que es un sistema basado en un plano de dos dimensiones con dos ejes perpendiculares. En la realidad, para este tipo de aeronaves debemos consideramos más características del entorno, la primera y más fundamental es considerar un sistema de coordenadas basado en tres ejes, que no es más que un sistema de dos coordenadas al añadirle un tercer eje ortogonal a los dos ejes. Como podemos observar en la imagen, para apreciar la existencia del tercer eje se suele representar en perspectiva.

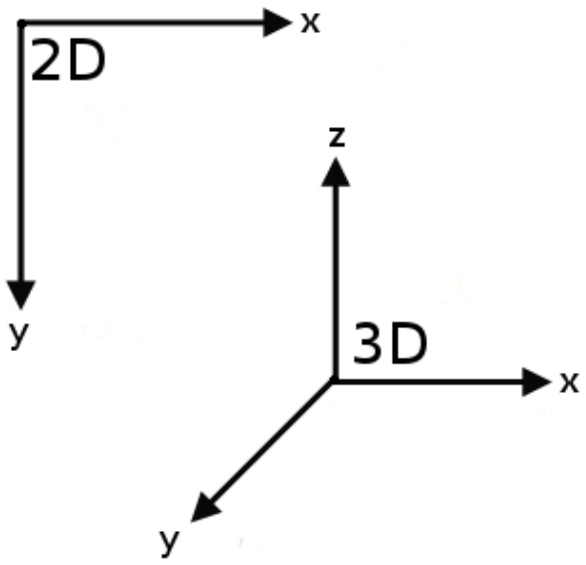


Figura 3.10: Sistemas de coordenadas

En consecuencia, un punto se representa con tres componentes y hay varios sistemas de representación para un objeto en un sistema de coordenadas de tres dimensiones. No entraremos en detalles de su representación, hay varios formas de hacerlo, la que emplearemos es XYZ de Euler.Al igual que en un sistema cartesiano, para movernos por su espacio tenemos dos transformaciones: la rotación y la traslación. Las operaciones matematicas a realizar son muy simirales a las realizadas en dos dimensiones.

La traslación no varia a la forma de operar con respecto dos dimensiones, tenemos un punto en el espacio y para trasladarlo sumamos el vector desplazamiento a el punto p, como resultado obtenemos el punto p'.

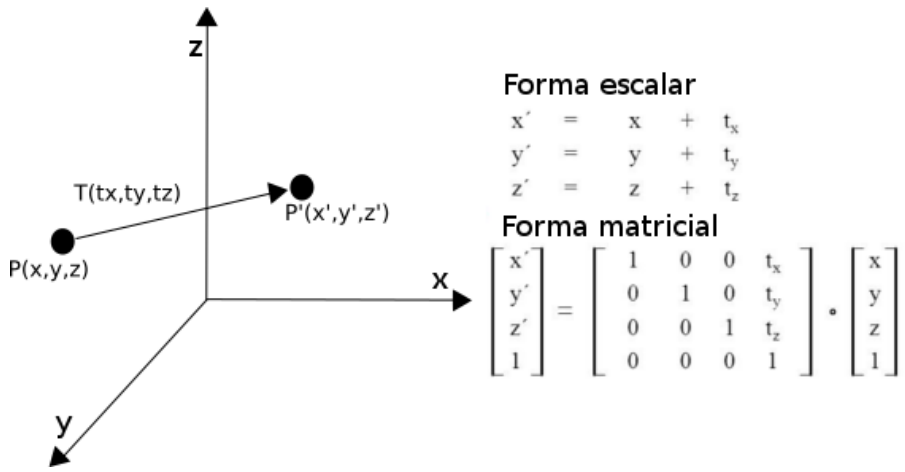
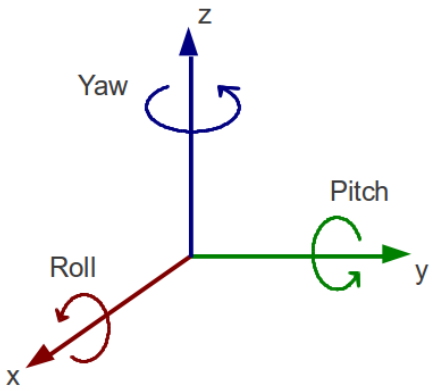


Figura 3.11: La traslación

La rotación en tres dimensiones puede ser intutiva si se tiene como referencia un eje sobre el cual rotamos un cierto angulo, si combinamos las rotaciones de los tres ejes tenemos tres angulos respectivos, conocidos como Roll, Pitch, Yaw,

asociados al eje XYZ de Euler y sus respectivas matrices para realizar el calculo de la nueva orientación.



$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\operatorname{sen}\theta \\ 0 & \operatorname{sen}\theta & \cos\theta \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \operatorname{sen}\theta \\ 0 & 1 & 0 \\ -\operatorname{sen}\theta & 0 & \cos\theta \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\operatorname{sen}\theta & 0 \\ \operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Figura 3.12: La rotación

Llegados a este punto, podemos hablar de los seis grados de libertad en el espacio en tres dimensiones, porque el dron es capaz de moverse en el espacio tanto la traslación en los tres ejes como la rotación en los tres ejes. Como hemos comentando anteriormente, a los angulos asociados a los ejes se les conoce como Roll, Pitch, Yaw (equivalentes en español como Alabeo, Cabeceo, Guiñada) y son angulos asociados a la navegacón de las aeronaves, a los cuales se les asocia las letras griegas θ, ϕ, ψ, respectivamente. Las seis diferentes transformaciones en el espacio son independientes, así que pues obtenemos movimientos con seis grados de libertad.

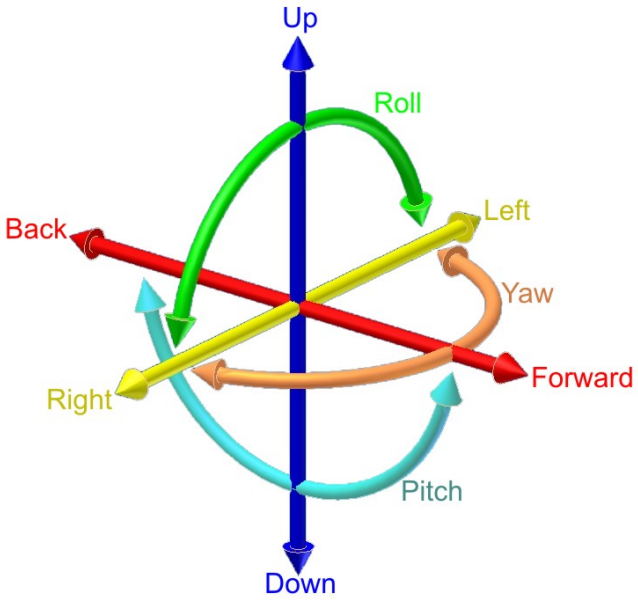


Figura 3.13: Representación de los 6 grados de libertad en el espacio tridimensional

Avanzando en este razonamiento debemos establecer la relación que tiene los grados de libertad con el quadrotor. Mediante métodos de la física clásica podemos obtener una aproximación con las ecuaciones de Euler-Lagrange (REFERENCE) que nos permiten obtener esta relación (ecuaciones 3.1-3.6). Las tres primeras

ecuaciones nos revelan la posición en el plano XYZ con respecto a su centro de gravedad del quadrotor, suponiendo que el centro es puntual y por tanto, paralero al plano Z del suelo, la distancia entre ambos será la altura. Con respecto a θ, ϕ, ψ, tenemos la asociación a sus respectivos angulos Roll, Pitch, Yaw. Por último, el término u corresponde al empuje total del drone.

$$m\,\ddot{x} = -u\,sin\,\theta \tag{3.1}$$

$$m\,\ddot{y} = u\,cos\,\theta\,sin\,\phi \tag{3.2}$$

$$m\,\ddot{z} = u\,cos\,\theta\,cos\,\phi - m\,g \tag{3.3}$$

$$\ddot{\theta} = u_\theta \tag{3.4}$$

$$\ddot{\phi} = u_\phi \tag{3.5}$$

$$\ddot{\psi} = u_\psi \tag{3.6}$$

Partiendo de un modelo de quadrotor en Cruz. De la ecuaciones anteriores, debemos obtener las ganancias para controlar el quadrotor del Roll, Pitch y Yaw, aplicando los empujes necesarios a cada motor. El criterio de signos para el Roll y Pitch positivo son M1 y M2 mientras que el negativo son M3 y M4, indicando M, el motor al cual hace referencia. Con respecto al Yaw, consideramos el giro a izquierda como indica la flecha de la imagen.

$$u_\theta = u_{M2} - u_{M4} \tag{3.7}$$

$$u_\phi = u_{M1} - u_{M3} \tag{3.8}$$

$$u_\psi = u_{M1} - u_{M2} + u_{M3} - u_{M4} \tag{3.9}$$

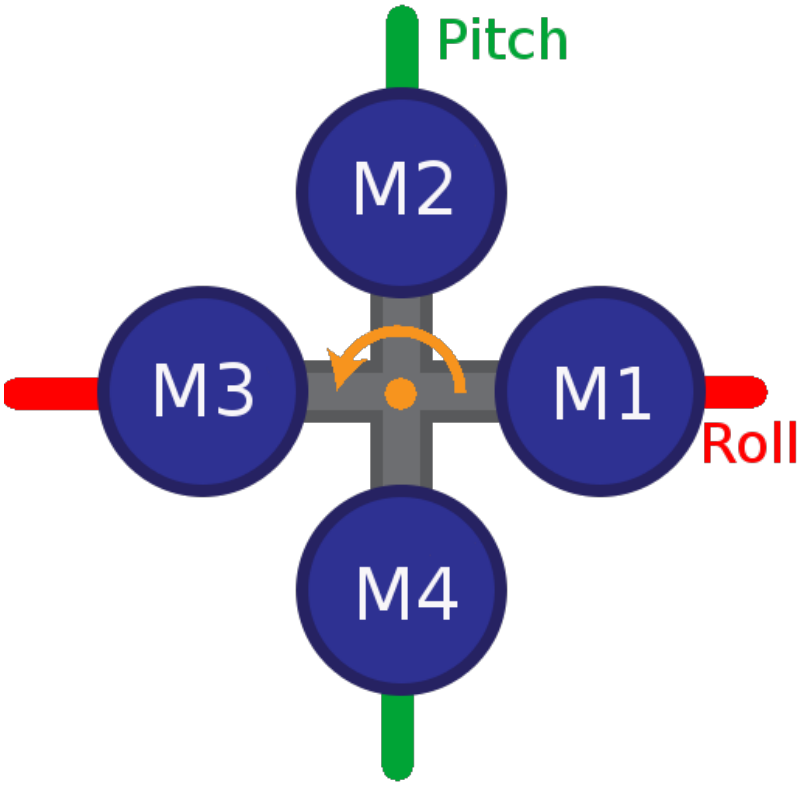


Figura 3.14: Modelo en Cruz de un quadrotor

En cambio, partiendo de un modelo de quadrotor en Equis, que no es mas que rotar 45 grados el modelo anterior. Pues bien esta cambio influye en los ejes del Roll y Pitch, es decir, la mezcla de los empujes de los motores es distinta, ahora todos los motores influyen tanto en el Roll como en el Pitch. El criterio de signo sigue siendo el mismo que en la imagen anterior. Recordamos que θ equivale al Roll y ϕ al Pitch. La elección llevada a cabo es el modelo en Equis.

$$u_\theta = u_{M1} - u_{M2} - u_{M3} + u_{M4} \tag{3.10}$$

$$u_\phi = u_{M1} + u_{M2} - u_{M3} - u_{M4} \tag{3.11}$$

$$u_\psi = u_{M1} - u_{M2} + u_{M3} - u_{M4} \tag{3.12}$$

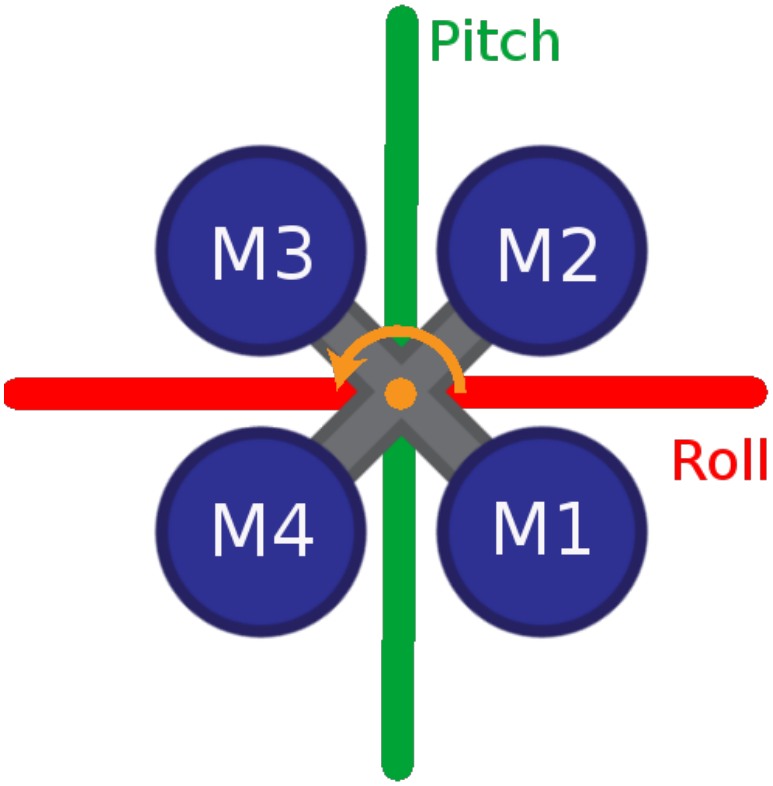


Figura 3.15: Modelo en Equis de un quadrotor

Fundamentos de control

En este apartado trataremos de explicar el control empleado en el quadrotor. Emplearemos el modelo teórico parcial en Cruz explicado anteriormente, ya que hay varios tipos de control tanto en orientación como en posición. En orientación con tener un sensor inercial es suficiente. En posición dependemos de un sensor barometrico junto con un GPS o unas camaras para capturar la posición. Por ello, el control empleado unicamente será un control en orientación.

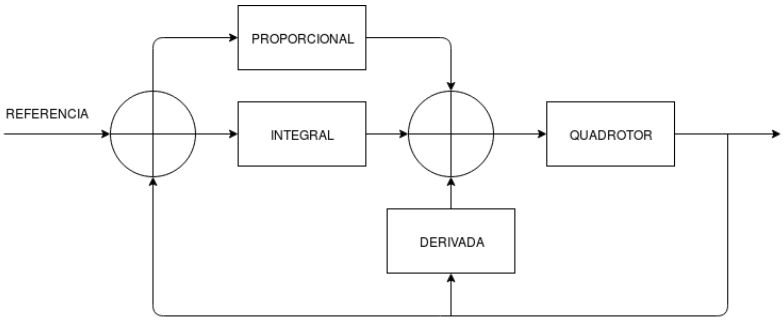


Figura 3.16: Diagrama de bloques de un PID

El control en orientación se basa en controlar la rotación en los tres ejes en el Roll, Pitch y Yaw del quadrotor. En el ámbito del control hay diversas técnicas que podemos emplear desde técnicas de control adaptativo o redes neuronales como técnicas menos sofisticadas como el clásico control PID. Por ello, emplearemos la técnica de PID ya que es una técnica básica fácilmente de implementar.

Como estamos empleando un BeagleBone, al tratarse de un sistema de digital que discretiza la entrada y salida de los sensores y actuadores debemos emplear la discretización de un PID. Como podemos apreciar en la imagen, la derivada se retroalimenta directamente al control esto es debido a que obtenemos el valor en crudo del sensor y este es el empleado directamente para el calculo de la derivada, como si este valor fuese el error, ya que este valor viene a ser la velocidad angular. Las ecuaciones del PID se expresan de la siguiente forma.

$$(E)rror = Referencia - MedidaAngulo \tag{3.13}$$

$$(P)roporcional = KpE \tag{3.14}$$

$$(D)erivada = KdMedidaVelocidadAngular \tag{3.15}$$

$$(I)ntegral = IntegralAnterior + KiE \tag{3.16}$$

$$PID = P + I - D \tag{3.17}$$

Dichas cálculos, al tener tres ejes de rotación a controlar deberá aparecer por triplicado aplicando el control a cada uno de los tres ángulos. Por último, la salida de cada PID deberá ser aplicada a cada motor según la influencia positiva o negativa del motor sobre la rotación del ángulo correspondiente, además hay que tener otro factor en cuenta y es tener un empuje total u_a para los cuatro motores, que en un futuros con otro sensor podríamos utilizarlo para controlar el drone en altitud respecto el suelo. Por ahora, nos bastará en aplicar un cantidad fija para evitar que los valores aplicados a los motores sean inferiores a 0, ya que no tendrían ningún efecto sobre el motor.

$$u_{M1} = u_a + u_\theta - u_\phi - u_\phi \tag{3.18}$$

$$u_{M2} = u_a - u_\theta - u_\phi + u_\phi \tag{3.19}$$

$$u_{M3} = u_a - u_\theta + u_\phi - u_\phi \tag{3.20}$$

$$u_{M4} = u_a + u_\theta + u_\phi + u_\phi \tag{3.21}$$

CAPÍTULO 4
Diseño, construcción y comunicación

Como comentamos en la introducción del trabajo, este proyecto viene dado por uno anterior personal, trataremos de reutilizar piezas que ya teníamos para este proyecto, ademas de dar la razón por la cual unos componentes son compatibles con otros y que componentes hemos añadido para ensamblar el resto del quadrotor. Partimos de un esquema mínimo como podemos ver la imagen, trataremos de explicar cada componente y destinaremos un apartado a ampliar el resto de elementos del quadrotor. Para concluir esta introducción el concepto del titulo de este apartado trata de describir el concepto de ver que componentes tenemos, elegir que componentes son necesarios añadir y si es necesario, como se comunican entre ellos.

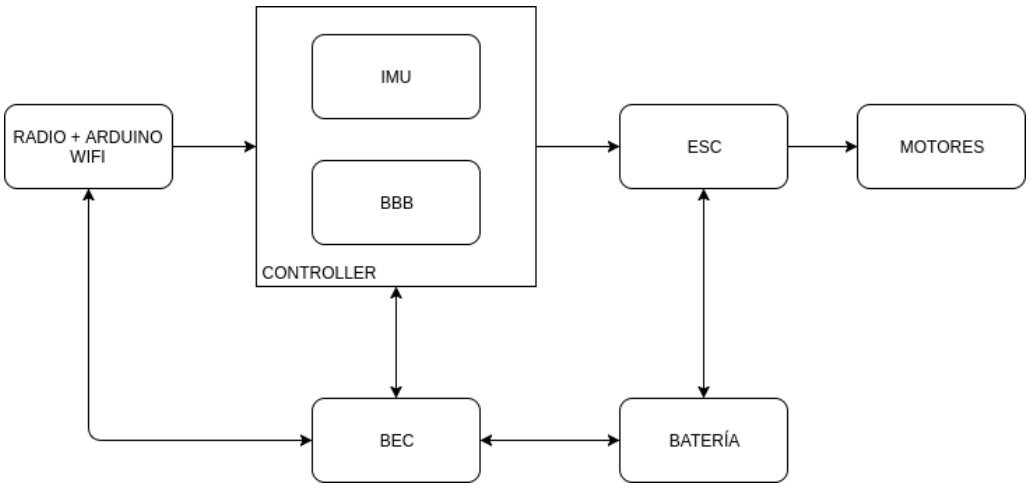


Figura 4.1: Esquema básico de los componentes de un quadrotor

Motor, hélices y chasis

Partimos de la base de un proyecto anterior del cual disponemos de cuatro motores, los cuales nos van a determinar las características del resto de componentes del drone. Los motores eléctricos son los encargados de generar energía

cinética que es transmitida a través de un eje a las hélices y estas son capaces de generar la fuerza de sustentación para mantener el drone en el aire.

El motor eléctrico es un aparato capaz de convertir la energía eléctrica en energía cinética por medio de generar campos electromagnéticos a través de un grupo de bobinas llamado estator, la cual hace que gire un rotor unido a un eje, que es el encargado de transmitir el movimiento cinético. Para los drones comerciales, el tipo mayoritario de motores empleados son los motores sin escobillas o "brushless". Como su propio nombre indica, se caracterizan por no tener escobillas para inducir el movimiento al motor. Estos tienen la propiedad de no tener fricción entre el estator y rotor, y en consecuencia no sufren el desgaste de las escobillas.

Para poder determinar las características de los demás componentes primero debemos determinar las características del motor porque estas afectaran a características de diseño y otros factores que pueden afectar el rendimiento y la eficiencia de un motor. El motor es un motor brushless Turnigy L2210C.

Figura 4.2: Motor brushless Turnigy L2210C

Tabla 4.1: Especificaciones Turnigy L2210C

Dimensiones	28mm x 25mm
Peso	48g
Kv	1200rpm/V
Voltaje	7.2v-11.1v (2s-3s)
Potencia máxima	150w
Amperaje máximo	15.8A
Thrust	700g
ESC	20A
Tamaño hélice	8 pulgadas

El factor más importante a tener en cuenta en la construcción de cualquier multi rotor de cualquier tamaño es la relación entre el empuje y el peso. Esta relación tiene una práctica que se suele llevar a cabo y es poder proporcionar al menos el doble de empuje que el peso total del multirotor. Porque si el empuje proporcionado por los motores es demasiado pequeño, el drone podría no responder con la dinámica deseada, puede que incluso no tenga la fuerza necesaria para despegar. Es decir, vencer a la fuerza de la gravedad.

Por supuesto a mayor relación de empuje peso, el drone tendrá una mejor agilidad, velocidad y aceleración. Incluso se suele apuntar a relaciones más altas de entre tres y cuatro veces mayores, que además de darnos una mayor control, también nos brinda un margen para luego en un futuro poder equiparla con alguna cámara o una batería mas pesada.

Como partimos de unos motores dados, debemos calcular la relación aproximada que vamos a tener para calcular el peso total de drone objetivo que no deberíamos pasar. El calculo para los estos motores podría ser mucho más preciso, hay blogs muy buenos en la red como oscarliang, que sin necesidad de ser un experto en la materia, te pueden dar una idea aproximada para realizar una buena elección de los componentes del multirotor. En nuestro caso, solo tendremos en cuenta el thrust dado por la especificaciones del motor.

Además el empuje (thrust) dado por el motor viene determinado por el tipo de hélice y su tamaño. El tamaño de la hélice se puede calcular según los parámetros del motor, aunque si es cierto que el fabricante también nos da este tipo de información. No entraremos en detalles, porque no somos expertos en este tipo de materia. Por tanto, seguiremos el tamaño de la hélice recomendada en 8 pulgadas.

La relación entre el peso y el empuje se calcula multiplicando por cuatro la fuerza dada, ya que disponemos de cuatro motores, luego se aplica un coeficiente de seguridad porque el thrust dado en las especificaciones es el thrust máximo en condiciones optimas. Se suelen aplicar aplicar dos coeficientes de seguridad, una por la eficiencia esperada del motor con respecto a la helice, elegimos un valor de 0.7, y otro coeficiente de seguridad para la potencia deseada de vuelo, elegiremos a media gas para el vuelo. Por tanto, el peso máximo total estimado debe ser 980 gramos.

$$Peso = (N * T) * Eh * Te = (4 * 700g) * 0.7 * 0.5 = 980g$$

Figura 4.3: Calculo relacion empuje - peso

Donde:
· N - El número de motores
· T - Thrust máximo
· Eh - Eficiencia motor hélices esperada
· Te - Thrust estimado

Por otra parte, tenemos que determinar el tamaño del chasis. Hay diversas formas de realizar este calculo, una de ellas, es obtener la medida de la hélice, y dejar este espacio entre motor y motor, añadiéndole al menos un 1 cm de más para evitar que choquen entre ellas. Por tanto, medimos las hélices de 8 pulgadas,

que equivalen a unos 23 cm. A esta medida le añadimos 3 cm por seguridad, y nos queda una distancia de 26 cm. Como va a ser un drone simétrico en X, ya podemos calcular la distancia entre motor y motor en la diagonal, usando el conocido teorema de Pitágoras, obteniendo aproximadamente 37 cm. La pelota de pinto sirve para cogerlo con seguridad y siempre se aconseja cogerlo con guantes para evitar cortes en las manos.

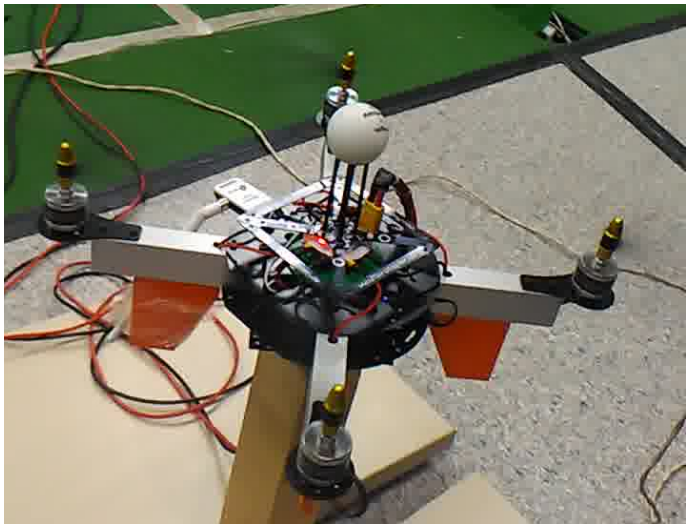


Figura 4.4: Chasis

Bateria y PDB

El sistema de alimentación eléctrico esta compuesto principalmente por dos componentes: la batería y la placa electrónica de distribución (pdb) con un BEC. Todos estos elementos están condicionados por las características eléctricas del motor.

Para empezar, trataremos de explicar la elección de la batería. Como vimos en las especificaciones del motor, el voltaje del motor viene dado en voltios, pero también en un valor seguido de una letra s, la razón es que las baterías de polímero de litio, se clasifican según el número de celdas, con una capacidad de 3.7V, entonces el valor entre paréntesis nos especifica el numero de celdas de la batería que emplea estos motores. Otro factor que influye es el amperaje máximo que puede consumir el motor, este influye en la elección del coeficiente de descarga y la capacidad de la batería, ya que al multiplicar el capacidad por el coeficiente al menos debe de salir un valor superior a la suma del amperaje del máximo de los N motores a emplear.

Figura 4.5: ZIPPY Compacto 3700mAh 3S Lipo 25C

Tabla 4.2: Especificaciones

Dimensiones	146x19x43mm
Peso	264g
Capacidad	3700mAh
Voltaje	3S1P / 3 celdas / 11.1V
Descarga	25C
Enchufe de descarga	HXT4mm

En ultima instancia tenemos la placa de distribución. Esta consiste en una PCB que tiene salidas para conectar los controladores a la batería y también se encarga de regular el voltaje a 5 voltios para la electrónica de control y en algunos casos a 12 voltios para la emisión de la cámara o otro tipo de accesorios. El componente integrado en este tipo de placas se conoce como BEC, este se encarga de transformar la corriente de forma eficiente, además de aislar la electrónica de baja potencia con la electrónica de control de los motores.

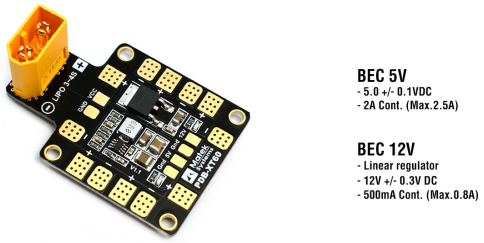


Figura 4.6: PDB Matek v1.1

Tabla 4.3: Especificaciones PDB Matek v1.1

Dimensiones	36mm x 50mm
Voltage	9-18V (3-4S LIPO)
ESC Outputs	6
ESC Amperaje	25A*4 o 15A*6
BEC 5V	2A
BEC 12V	0.5A

ESC

El ESC viende de las siglas en ingles de Electronic speed control, traducido al español como electronica de control de velocidad. Este es un circuito electronico capaz de controlar un motor mediante ordenes recibidas desde otro dispositivo electronico. La comunicación se establece mediante protocolos analogicos como PWM, OneShot125 o protocolos digitales como dshot. Las caracteristicas de este componentes tienen que ir acorde con las caracteristicas del motor. El fabricante suele recomendar un tipo de controlador, como vimos en la tabla y.y, en paginas anteriores, nos recomiendo un controlador de 20 A, este valor viene dado por el amperaje que consume el motor, el cual se sobreestima para evitar llevar al limite la electrónica y tener una mayor durabilidad de los componentes. En cuanto al voltaje, como hemos dicho en la baterías, debe soportar el numero de celdas elegidas. Por tanto, elegimos el controlador Turnigy Multistar 20A Delgado V2 ESC Con BLHeli OPTO 2-6S.

Figura 4.7: Turnigy Multistar 20A Delgado V2 ESC

Tabla 4.4: Especificaciones principales de Turnigy Multistar 20A Delgado V2 ESC

Dimensiones	50mm x 12mm
Voltage	2-6S LIPO
ESC Outputs	6
ESC Voltage	20A
Comunicación	OneShot125
PWM	500 Hz

El pulso por modulación por ancho de pulsos o PWM se emplea para generar señales analogicas a partir de señales digitales. Dada una frecuencia y un voltage total, se aplica un pulso de este voltage durante un porcentaje de tiempo con respecto a la frecuencia. Esta aplicación equivale a multiplicar el porcentaje de tiempo por el voltaje total dando como resultado un porción del voltaje total. Por ejemplo, si aplicamos el 50 % del tiempo a un voltaje de 5V obtendremos como resultado 2.5V.

Aunque en este aspecto si es analógico, si consideramos que son pulsos a una determinada frecuencia, se podría parecer a un protocolo digital, y es que esta es la base de comunicación con el controlador de motor. Básicamente lo que realiza el controlar de motor es medir el tiempo de ancho de pulso con respecto a la frecuencia. El control en PWM se hereda de los servos, que empleaban una frecuencia base de 50Hz, teniendo un ancho de pulso mínimo y máximo de 1ms y 2ms respectivamente. Aunque para el control de un drone, esta aplicación es demasiado lenta y por tanto se aumentó la frecuencia, por esta razón nace protocolos basados en PWM con mayor frecuencia. Oneshot125 se basa en mantener la proporcion, es decir, aumentar la frecuencia manteniendo la relación del 5 al 10 % de la frecuencia base, por ejemplo, si aumentamos a 500Hz, el mínimo y máximo pasarían a ser 125 us y 250 us respectivamente.

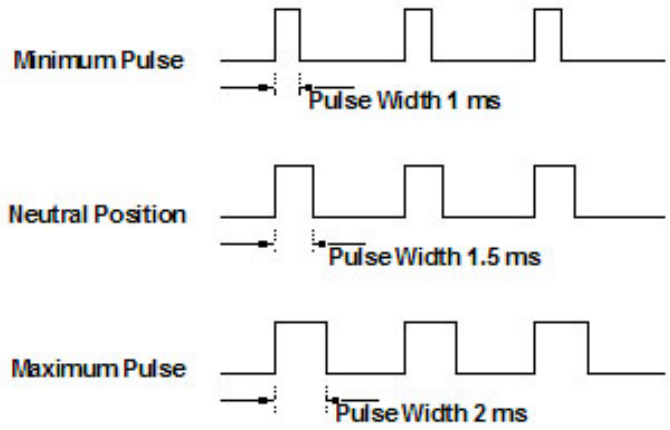


Figura 4.8: PWM 50Hz

IMU

Una unidad de medición inercial o IMU (del inglés inertial measurement unit), es un dispositivo electrónico que mide e informa acerca de la velocidad, orienta-

ción y fuerzas gravitacionales de un aparato, usando una combinación de acelerómetros y giróscopos. La IMU es el componente principal de los sistemas de navegación inercial usados en aviones, naves espaciales, buques y misiles guiados entre otros. En este uso, los datos recolectados por los sensores de una IMU permiten a un computador seguir la posición del aparato, usando un método conocido como navegación por estima.

El MPU-9250 es el dispositivo MotionTracking (seguimiento de movimiento) de 9 ejes de segunda generación de la empresa Invisense para teléfonos inteligentes, tabletas, sensores portátiles y otros mercados de consumo. El MPU-9250, entregado en un paquete QFN de 3x3x1mm, es el dispositivo MotionTracking de 9 ejes más pequeño del mundo e incorpora las últimas innovaciones de diseño de InvenSense, permitiendo un tamaño de chip y consumo de energía dramáticamente reducido, al tiempo que mejora el rendimiento y el costo.

El MPU-9250 es un sistema en paquete (SiP) que combina dos chips: el MPU-6500, que contiene un giroscopio de 3 ejes, un acelerómetro de 3 ejes y un Digital Motion Processor (DMP) integrado, capaz de procesar complejos algoritmos fusión de datos; y el AK8963, la brújula digital de 3 ejes líder del mercado. La comunicación con el sensor inercial MPU-9250 se establece mediante SPI o I2C, para este proyecto emplearemos el I2C.

El I2C (Inter-Integrated Circuit) es un bus de comunicación muy utilizado para comunicar circuitos integrados, uno de sus usos más comunes es la comunicación entre un microcontrolador y sensores periféricos. El I2C es un bus multimaestro (el que inicia la comunicación) es decir permite que haya múltiples maestros y múltiples esclavos en el mismo bus (aunque en este caso se establecen mecanismos de arbitraje de acceso al bus).

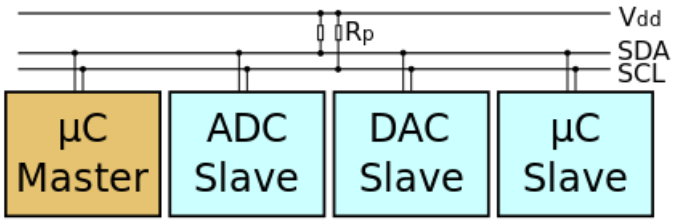


Figura 4.9: I2C

El bus I2C cuenta con dos líneas SDA(datos) y SCL(clock) además de masa. Cada flanco de SCL, marca la aparición de un bit de datos en la línea SDA. Además, hay que tener en cuenta que la línea SDA solo puede cambiar de valor en caso de que la línea SCL este a 0. En I2C hay un bit dominante (0) y otro recesivo (1), al igual que pasa con el bus CAN. Esto es así porque el 0 se consigue forzando la línea a esa tensión, pero por contra el 1 se consigue con pull-up, por lo tanto, en caso de que alguien transmita un 0 y otro un 1, en la línea solo se verá reflejado el 0. Por lo tanto, se define como el valor de reposo del bus como el 1, ya que si alguien quiere empezar a comunicar siempre podrá modificar el estado del bus y los demás se darán cuenta. Otro aspecto a tener en cuenta en I2C es que los maestros o maestro son los únicos que pueden controlar la línea de SCL, eso implica que solo un maestro puede iniciar una transmisión por lo que un Slave tendrá que esperar a que un maestro le pregunte por un dato para poder enviar-

lo. Así en I2C cada dispositivo tiene una dirección de 7 bits, es decir se pueden tener hasta 128 dispositivos conectados al mismo bus, hay que tener en cuenta que existen versiones extendidas de I2C con direccionamiento a 8,10 y 12 bits.

Controladora

La BeagleBone Black (BBB) es una placa programable de bajo coste que encuentra su punto fuerte en su gran versatilidad y en su capacidad para convertirse en un ordenador de bajo consumo de potencia con un sistema operativo en base Linux. El procesado de la BBB incorpora un acelerador hardware de operaciones de punto flotante. Esto es clave para ejecutar de forma eficiente el núcleo de Linux y abre un nuevo abanico de posibilidades de desarrollo de sistemas empotrado sobre Linux. A este tipo de placas se las conoce como SCB (Single Board Computer) o ordenador empotrado.

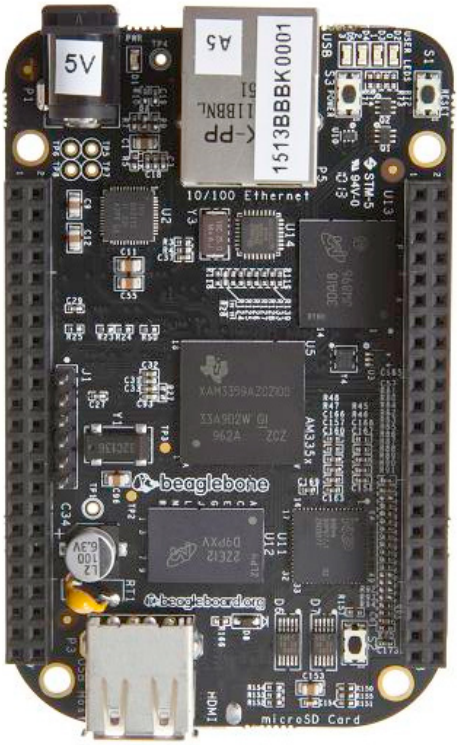


Figura 4.10: BeagleBone Black.

Su procesador ARM-A8 de 1GHz y su memoria RAM DDR3 de 512 MB la convierten en una unidad de proceso realmente potente en relación a su tamaño. Sin embargo, su mayor potencial se encuentra en su capacidad de conexión gracias a sus 92 pines de entrada/salida, compatibles con buses de comunicación como I2C, UART y SPI, y a sus puertos Ethernet, USB y mini HDMI.

La BBB también ofrece una unidad de memoria flash interna de 4GB y la posibilidad de ampliar su capacidad mediante una ranura para tarjetas microSD. Las principales características de la placa podemos observarlas en la siguiente Figura:

	Feature	
Processor	Sitara AM3358BZCZ100 1GHz, 2000 MIPS	
Graphics Engine	SGX530 3D, 20M Polygons/S	
SDRAM Memory	512MB DDR3L 800MHZ	
Onboard Flash	4GB, 8bit Embedded MMC	
PMIC	TPS65217C PMIC regulator and one additional LDO.	
Debug Support	Optional Onboard 20-pin CTI JTAG, Serial Header	
Power Source	miniUSB USB or DC Jack	5VDC External Via Expansion Header
PCB	3.4" x 2.1"	6 layers
Indicators	1-Power, 2-Ethernet, 4-User Controllable LEDs	
HS USB 2.0 Client Port	Access to USB0, Client mode via miniUSB	
HS USB 2.0 Host Port	Access to USB1, Type A Socket, 500mA LS/FS/HS	
Serial Port	UART0 access via 6 pin 3.3V TTL Header. Header is populated	
Ethernet	10/100, RJ45	
SD/MMC Connector	microSD , 3.3V	
User Input	Reset Button Boot Button Power Button	
Video Out	16b HDMI, 1280x1024 (MAX) 1024x768,1280x720,1440x900 ,1920x1080@24Hz w/EDID Support	
Audio	Via HDMI Interface, Stereo	
Expansion Connectors	Power 5V, 3.3V , VDD_ADC(1.8V) 3.3V I/O on all signals McASP0, SPI1, I2C, GPIO(69 max), LCD, GPMC, MMC1, MMC2, 7 AIN(1.8V MAX), 4 Timers, 4 Serial Ports, CAN0, EHRPWM(0,2),XDMA Interrupt, Power button, Expansion Board ID (Up to 4 can be stacked)	
Weight	1.4 oz (39.68 grams)	
Power	Refer to Section 6.1.7	

Figura 4.11: Características del hardware BeagleBone Black.

La BBB normalmente se usa como un "Sistema Linux Empotrado". Como no existe una versión del núcleo de Linux especial para sistemas empotrados, cuando nos referimos a un "Sistema Linux Empotrado" queremos decir simplemente un Linux instalado en un sistema empotrado. El SO que utilizaremos es Linux Debian para ARM con aceleración hardware para operaciones de punto flotante ("hf"). Además Linux en sí mismo no es capaz de tener características de tiempo real, así que el parche adicional Preempt-rt le permite adquirir capacidades de tiempo real. Para ver la diferencia entre tener el parche o no, disponemos de un ejemplo para la lectura de un sensor analógico a través de la BBB, se puede apreciar en las siguientes gráficas (ver apéndice, archivo analog02.cpp). Este ejemplo ha sido proporcionado por el tutor, Joše Simó.

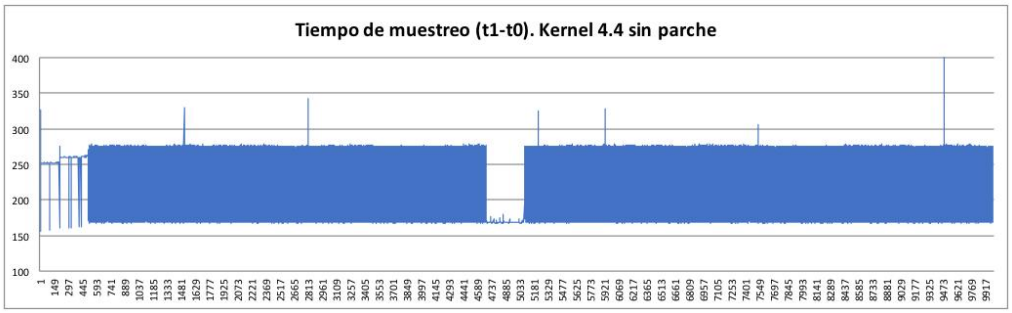


Figura 4.12: Tiempo de acceso al sysfs ADC en Linux sin parche Preempt-rt

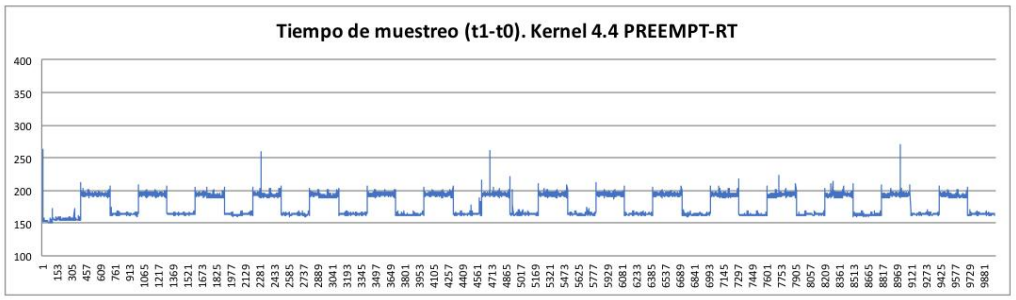


Figura 4.13: Tiempo de acceso al sysfs ADC en Linux con parche Preempt-rt

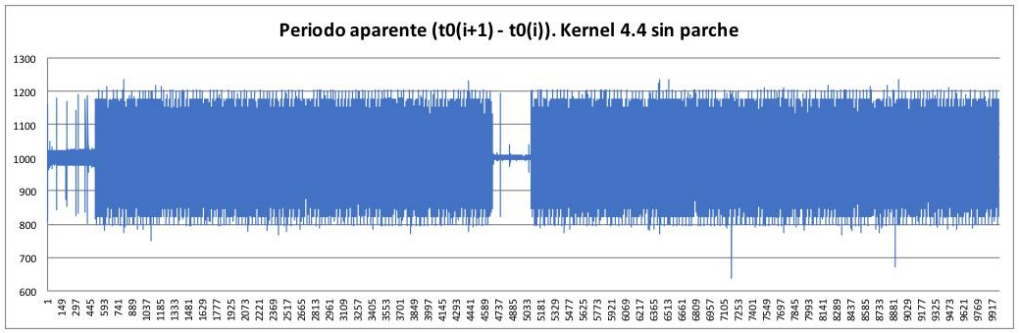


Figura 4.14: Dispersión en el periodo de ejecución (establecido en 1000 us) en un Linux sin parche Preempt-rt

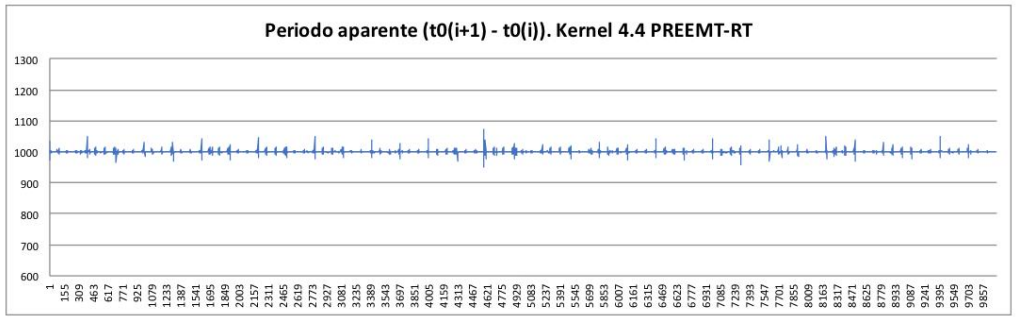


Figura 4.15: Dispersión en el periodo de ejecución (establecido en 1000 us) en un Linux con parche Preempt-rt

Radio y WIFI

La comunicación con el dron se establecerá mediante radiocontrol basada en ondas de radio. Esta técnica consiste en gobernar el dron a distancia por linea visual. Los sistemas de radio control estaban clasificados en dos grandes grupos: amplitud modulada (AM) y frecuencia modulada (FM). Aunque en los últimos 10 años, en el sector del modelismo se ha popularizado el uso del rango de frecuencia de los 2.4 GHz (sobre FM). Ahora pasaremos a explicar por encima esta clasificación.

La modulación de frecuencia se basa en la frecuencia portadora, la cual alterna entres dos frecuencias diferentes. Esta puede basarse en la modulación por posición de pulso (PPM) y la modulación de código de pulso (PCM).

La modulación por posición de pulso es la generación de una onda donde la Amplitud y el ancho son fijos y la posición en variable, es un tipo de modulación en la cual una palabra de M bits es codificada por la transmisión de un único pulso.

La modulación de código de pulso se basa en una aproximación a un formato digital sobre FM. Al transmitir los datos, básicamente se traduce a código binario, donde el 0 corresponde con encender la emisión y el 1 con apagarla.

Una emisora RC para comunicarse con el modelo que controla utiliza una antena y varios canales, con un mínimo de 2 y aumentan la cantidad de funciones según la funcionalidad que tenga incorporado el modelo de radiocontrol. Se caracteriza por tener un numero de canales determinado que condiciona el numero de elementos en el mando, como pueden ser los botones, interruptores, joysticks y potenciometros.

Por una parte, tenemos la emisora de control remoto con un sistema de radio de 2.4 GHz esta se basa en radio de espectro expandido (Spread spectrum radio). Es una técnica de modulación basada en el ensanchamiento de la señal a lo largo de su banda de emisión, y se caracteriza por la ausencia de interferencia con otras radios y tener una respuesta más rápida.

El equipamiento que vamos a usar para el manejo a distancia del dron es una emisora y receptora de la marca y web Hobbyking, véase Figura . Se caracteriza por utilizar los 2.4 GHz y tener seis canales de comunicación, cuatro de ellos para los cuatro joysticks y dos de ellos para potenciometros.

(a) Emisora de Radiocontrol Hobbyking

(b) Receptor de Radiocontrol Hobbyking

Figura 4.16: HobbyKing 2.4Ghz Tx y Rx 6Ch V2

A su vez, tenemos otra forma de comunicarnos con el dron, esta se basa en una antena wifi. El dron se conecta a un punto de acceso wifi donde nos envía toda la información a través de un framework llamado CKMultipeer, que explicaremos posteriormente en el siguiente apartado. Se emplea a grandes rasgos para enviar los parámetros de control y recibir la telemetría del dron. El dispositivo wifi empleado es TP-LINK TL-WN722N Adaptador USB WiFi 802.11n, su elección viene dada por su bajo coste y su compatibilidad con Linux.

Figura 4.17: TP-LINK TL-WN722N Adaptador USB WiFi 802.11n

CAPÍTULO 5
Software

En este capítulo hablaremos del desarrollo del software. Emplearemos dos lenguajes de programación distintos como son C++ y Java, aunque si es cierto que ambos son lenguajes orientados a objectos, esta característica nos ayudara en la estructuración del código. Por otra parte, el otro pilar fundamental es la aplicación de un núcleo de control desarrollo por José Enrique Simó Ten, tutor del TFG. Cabe destacar que el uso de este framework en sistemas de tiempo real esta en desarrollo y comentaremos las distintas problemáticas encontrados.

Núcleo de control

En el contexto del desarrollo de aplicaciones de control basadas en el concepto de "Núcleo de Control" (CK de sus siglas en inglés), CKMultipeer es un elemento fundamental que ofrece acceso transparente a datos en un entorno distribuido y un modelo de programación basado en eventos.

CKMultipeer es un software que sirve para comunicar procesos en un entorno distribuido. Implementa un mecanismo de difusión asíncrono basado en mensajes y posee las siguientes características:

- Comunicación basada en difusión asíncrona.
- Funciona sobre el protocolo basado en conexión TCP/IP.
- No tiene restricciones sobre el tamaño de la carga útil del mensaje.
- Ofrece un reloj sincronizado entre todos lo pares.
- Los mensajes se envían con una marca de tiempo con resolución de microsegundos.
- Los mensajes se envían con una marca de prioridad de entrega.
- Incorpora un servicio de descubrimiento implementado sobre el protocolo sin conexión TCP/UDP. Este protocolo detecta automáticamente los "pares" que forman parte del grupo de comunicación y los agrega para que reciban copia de todos los mensajes.

La comunicación entre los pares se realiza sin la intervención de componentes específicos del tipo "middleware". Toda la funcionalidad necesaria está incluida en el código de los pares a modo de biblioteca pre-compilada o código fuente. El código de CKMultipeer está disponible para construir programas en lenguaje Java o en C++ sobre infraestructura POSIX. Consulte el anexo de este documento para obtener información sobre la distribución de CKMultipeer.

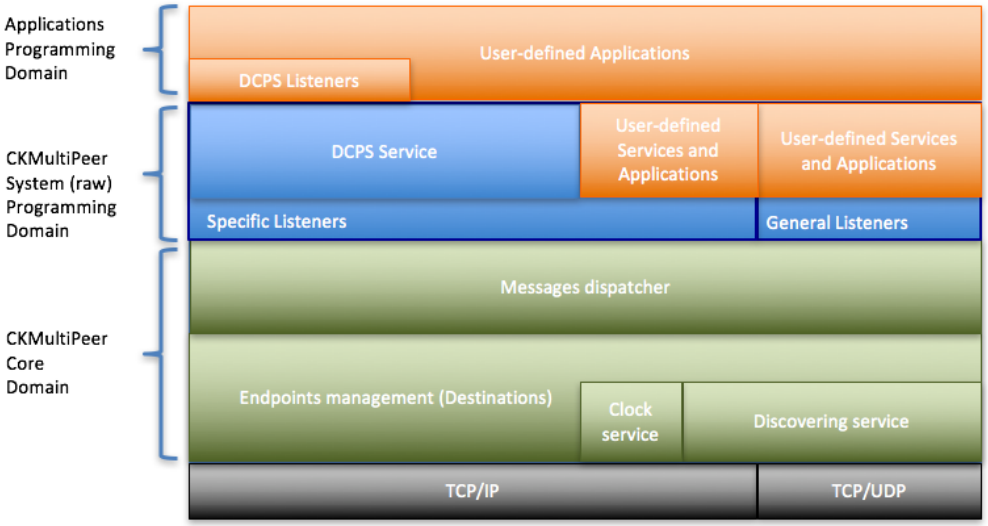


Figura 5.1: Estructura general de CKMultipeer

En la Figura 1 se presenta de forma esquemática la estructura del software CKMultipeer. Observe que el usuario-programador desarrollará sus aplicaciones programando elementos que se conectarán al sistema normalmente en el nivel "Application Programming Domain" o bien excepcionalmente, para el desarrollo de nuevos servicios en el nivel "System Programming Domain". Estas comunicación se realiza mediante una interfaz de publicación y suscripción que consiste en comunicarse a través de temas en común intercambiando mensajes en tiempo real siendo una mensajería rápida y eficiente, además de simplificar el diseño.

Estructura

La estructura software primeramente se planteó en 5 procesos desacoplados: la IMU, la Radio, el algoritmo de control, los motores y el HMI. Los procesos se comunican a través de la librería CKMultipeer. Ahora procederemos a explicar que realiza cada proceso.

En la imagen podemos ver dos grupos destacados por SCB y PC, estas siglas pertenecen al tipo de maquinas que van a ejecutar los diferentes códigos. Por una parte, tenemos el SCB (Single Board Computer - Ordenador de placa reducida) que corresponde a la Beagle Bone Black, y por otra parte, el PC que corresponde al ordenador convencional.

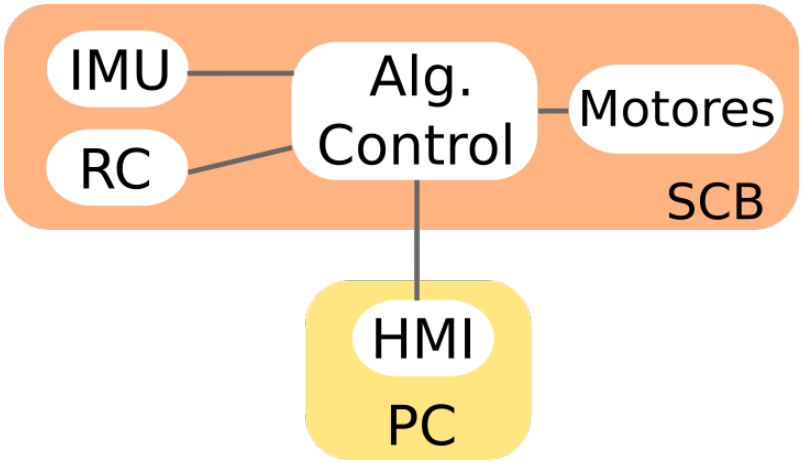


Figura 5.2: Estructura Inicial Planteada

Enumeraremos cada uno de los procesos y luego pasaremos a explicarlos. El proceso principal es el Algoritmo de control. En segundo lugar tenemos el HMI, proceso situado en el PC, este se comunica con la Beagle. Otro proceso es la IMU, este nombre viene dado porque hace referencia al nombre del sensor, inertial measurement unit, conocido en castellano como sensor inercial. Por último la radio, encargado de la lectura de la radio.

Como indicábamos al principio, se planteó la posibilidad de desacoplar el proceso de la actuación de los motores con el algoritmo de control. Este planteamiento generó una serie de problemas, al realizar una serie de pruebas se concluyó que al desacoplar lo, se generaban problemas de retardo, o incluso bloqueo en la aplicación de la actuación a los motores, en consecuencia, se provocaba que la aeronave tenga problemas de control y de seguridad. Por ejemplo, si cae el proceso del algoritmo de control, y el proceso de la actuación tenia dada un referencia, al caerse el proceso de control, los motores seguían en funcionamiento si control, esto generaba problemas de seguridad. En cambio, si la actuación esta acoplada, los motores dejarían junto con el proceso de control. A la vista de los resultados experimentados, la estructura software resultó acoplando la actuación de los motores al algoritmo de control, como podemos ver en la siguiente imagen.

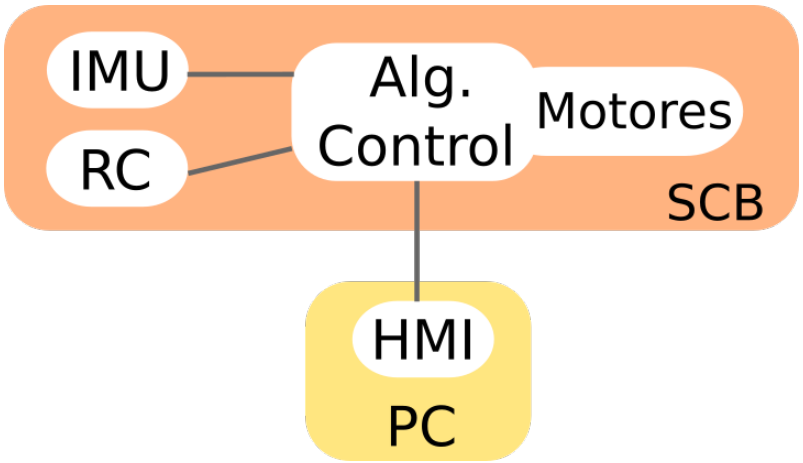


Figura 5.3: Estructura Final Planteada

Ahora trataremos de explicar la siguiente tabla, empezaremos por la relación entre el proceso de la IMU y el algoritmo de control. El proceso de la IMU envía un mensaje al algoritmo de control periódicamente a un frecuencia de 1000 Hz con los ángulos (Roll, Pitch, Yaw) y sus derivadas (DRoll, DPitch, DYaw). Por otra parte, tenemos la relación con el proceso de la radio de control (RC) y el algoritmo de control. El proceso RC envía la información de los 6 canales de la radio a una frecuencia aproximada de 20 Hz.

Tabla 5.1: Tabla de la relación entre procesos y temas

Proceso A	Proceso B	Topic
IMU	Alg. Control	IMUDATA
RC	Alg. Control	RCDATA
HMI	Alg. Control	HMICom

Dirección	Estructura Mensaje
A ->B (Unidireccional)	Roll:DRoll:Pitch:DPitch:Yaw:DYaw"
A ->B (Unidireccional)	Ç1:C2:C3:C4:C5:C6"
A <->B (Bidireccional)	Varios*

Por último, tenemos la relación entre el HMI y el algoritmo de control esta relación de mensajes es bidireccional. Hay varios tipos de estructuras de mensajes y según que mensaje tiene una estructura u otra, lo que si tienen en común es la cabecera. Por tanto, los podemos clasificar en 6 tipos de cabeceras numeradas: (1) establecer modo, (2) enviar parámetros control PID, (3) establecer potencia, (4) solicitar parámetros control PID, (5) latido del corazón del drone, (6) solicitar telemetría.

Para no entrar en demasiado detalle en las estructuras, daremos una pequeña explicación de que información porta cada tipo de paquete. El paquete (1) establecer modo, el HMI envía el modo de actuación del drone, actualmente tres: modo pruebas, modo desarmado y modo vuelo. El paquete (2) enviar parámetros control PID, el HMI envía los parametros de control como son las ganancias del PID, la saturación total de salida y la refencia de orientación. El paquete (3) establecer potencia, el HMI envía la potencia base a aplicar a los motores en conjunto o indidualmente, la acción en consecuencia dependerá del modo del drone. El paquete (4) solicitar parámetros control PID, el HMI envía una solicitud al drone para recibir los parametros actuales del drone. El paquete (5) latido del corazón del drone, el drone envía un ping cada 200 milisegundos para alertar al HMI que esta vivo. El paquete (6) solicitar telemetría, el HMI si esta activada la opción, cada 200 milisegundos, envía una solicitud al drone para recibir el roll, el pitch y el yaw, el drone cuando esta libre, contesta.

Algoritmo de control

Como podíamos ver en la imagen anterior de la estructura final son tres procesos a los cuales esta suscrito el proceso de control, en las siguientes secciones

explicaremos que realiza cada proceso. El proceso del algoritmo de control esta compuesto por un objeto compartido, 3 hilos y 3 temas, cada uno de los temas corresponde a cada proceso acoplado al proceso de control. Con respecto a los 3 hilos, por una parte, tenemos el hilo principal del proceso este se encarga de conectarse al multipeer, suscribir los temas y sus callbacks para recibir la información y depositarla en el objeto compartido. Por otra parte, tenemos un hilo que cada 150 milisegundos envía un mensaje al topic del HMI para notificar que el proceso esta vivo, como consecuencia, si el proceso del HMI esta vivo este le contesta y bloquea el control por parte de la radio de control, es decir, si esta el HMI y la radio, únicamente se hará caso al HMI.

Por último, tenemos el hilo de control que tiene un bucle principal que se encarga de ejecutar las rutinas de control cada cierto tiempo. El bucle de control sigue los siguientes pasos:

- Obtiene el modo de funcionamiento y los ángulos desde el objeto compartido.

- Comprueba si hay nuevos parámetros del HMI, y si los hay, actualiza los parámetros de control.

- Comprueba si esta el HMI o no, si esta, los obtiene del HMI, y sino de la Radio. En ambos casos, obtiene la potencia deseada y los ángulos deseados.

Llegados a este punto, ahora depende del modo elegido.

- Modo desarme: Este modo es el modo por defecto con los motores apagados.

- Modo test: Este modo solo es accesible desde el HMI, y sirve para probar el sentido de los motores.

- Modo Control: Este modo se encarga de realizar un control PD para el Roll, el Pitch y el Yaw. Luego realiza una mezcla de estos valores para la potencia resultante de cada motor.

HMI

El HMI es el programa encargado de enviar los parámetros de control y comprobar los sistemas del drone desde el ordenador a través del núcleo de control. La interfaz esta completamente en Inglés. El programa esta compuesto de 5 pestañas:

- Pestaña Conexión: Esta pestaña es la principal por defecto, en el cual podemos seleccionar la tarjeta de red, el grupo del multipeer y el topic. Por defecto, con los parámetros predefinidos es suficiente, solo hay que seleccionar la interfaz de red. Una vez están configurados los parámetros de la interfaz con el multipeer, pulsamos el botón de conectar y ya estaríamos conectados.

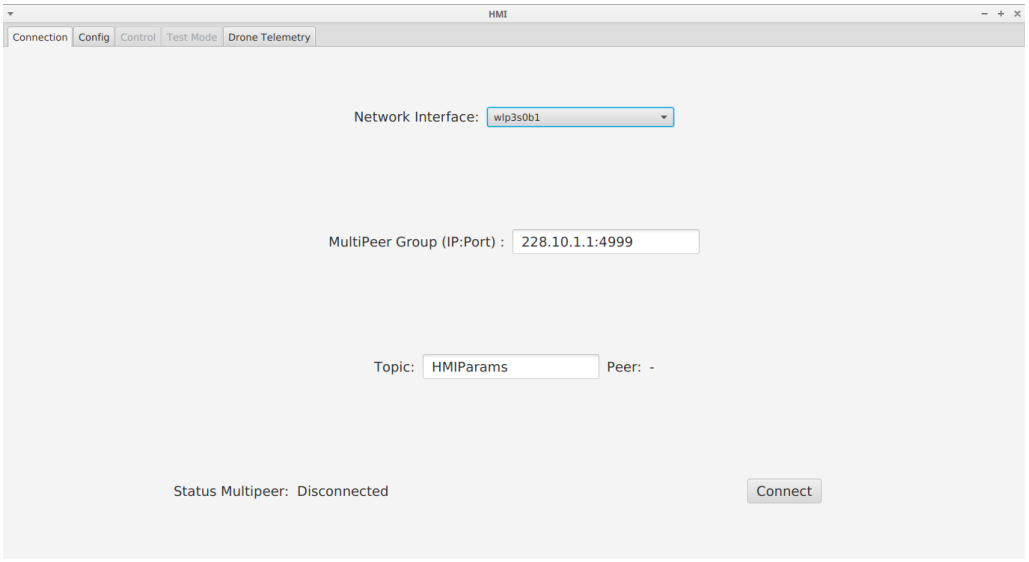


Figura 5.4: Pestaña Conexión

- Pestaña Guardar/Cargar Configuración: Esta pestaña básica tiene dos botones. Load Config para cargar la configuración y Save Config para guardar la configuración actual de los parámetros de control.

Figura 5.5: Pestaña Guardar/Cargar Configuración

- Pestaña Control: La pestaña de control consta de varios grupos de información, por una parte tenemos el más básico, el empuje o thrust que corresponde a la potencia directa dada al motor. Otro grupo correspondería a los modos, de los cuales disponemos el modo de desarmar, volar o test. Por último, las cajas de texto para enviar los parámetros de control del Roll, Pitch y el Yaw. Dispone de un botón de emergencia para desactivar en caso de emergencia. Esta pestaña esta desactivada hasta que se realiza la conexión al Multipeer.

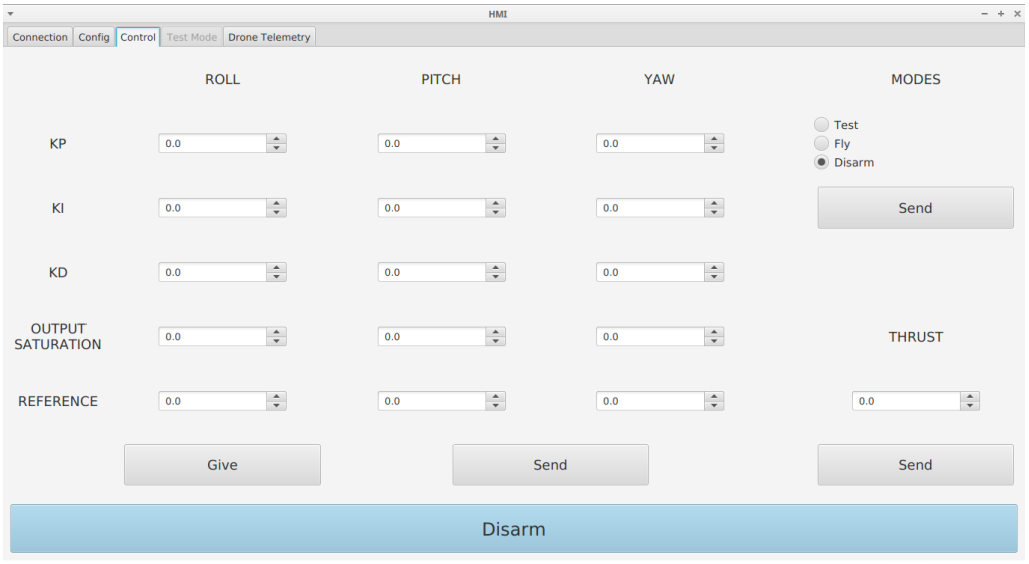


Figura 5.6: Pestaña Control

- Pestaña Test: Esta pestaña sirve para comprobar el funcionamiento de los motores, donde podemos probar los motores uno a uno o todos a la vez, se suele utilizar para comprobar el sentido de giro del motor y comprobar si son muy dispares las potencias de inicio del motor. Esta pestaña esta desactivada hasta que se realiza la conexión al Multipeer y se establece el modo de test desde la pestaña de control.

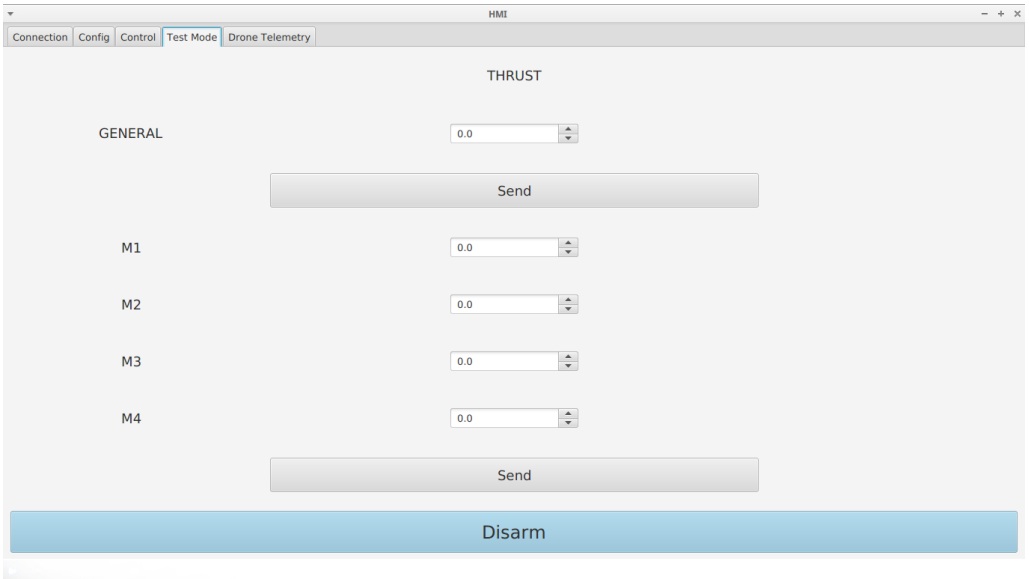


Figura 5.7: Pestaña Test

- Pestaña Telemetría: Esta pestaña consta de dos variables para comprobar si esta conectado el drone y el nombre de cual esta conectado, y también podemos observar algunos de los datos de los sensores en tiempo real.

Figura 5.8: Pestaña Telemetria

IMU

El proceso de la IMU es el encargado de leer el sensor Inercial y publicar la información procesada al tema IMUDATA para informar a todo proceso inscrito. El sensor inercial esta compuesto por un giroscopio, un acelerómetro y magnetómetro en los ejes XYZ. Con estos componentes podemos obtener la orientación del objeto en el espacio. Por otra parte, también viene equipado con un DMP (Digital Motion Processor – Procesador de movimiento), el cual tiene diversas configuraciones para proporcionar información fusionada combinando los valores de los tres componentes nombrados anteriormente.

La configuración empleada puede procesar la información a una frecuencia de 100 Hz aproximadamente, es decir, a un periodo de 10 ms. Este periodo es insuficiente para establecer un control estable para el drone. Para resolver este problemas de periodo emplearemos medidas intermedias crudas del giroscopio empleando un periodo de 1 ms por lectura, con ellos conseguimos 10 valores por cada una medida del DMP. Así que cuando obtenemos una medida del procesador de movimiento, tenemos una medida precisa de la orientación y con el giroscopio estimamos los valores intermedios a partir de la nueva medida del DMP, de esta forma obtenemos medidas cada 1 ms. Tampoco tendremos problemas de deriva y crecimiento del error en la medida de la orientación.

Llegados a este punto, el proceso es simple, primeramente se conecta al multipeer y se suscribe al tema IMUDATA, inicializa el DMP y empieza a obtener lectura del sensor cada 1 ms y publica la información y se duerme hasta la siguiente lectura, como decíamos cada 10 lecturas, lee el DMP corrige la medida actual, y así indefinidamente hasta recibir la señal de apagado del programa. Tiene una opción para guardar la información a modo de depuración en un fichero de texto para consultar sus valores en MATLAB. Véase Apéndice para consultar el código.

En esta imagen podemos ver las lecturas del sensor inercial cuando este esta en estado de reposo obtenidas a través del código en MATLAB. Se puede apreciar que las mediciones tienen algo ruido, este varia entre +-0.06, excepto la derivada del yaw que presenta un mayor ruido, todo este ruido sera compensado con el control.

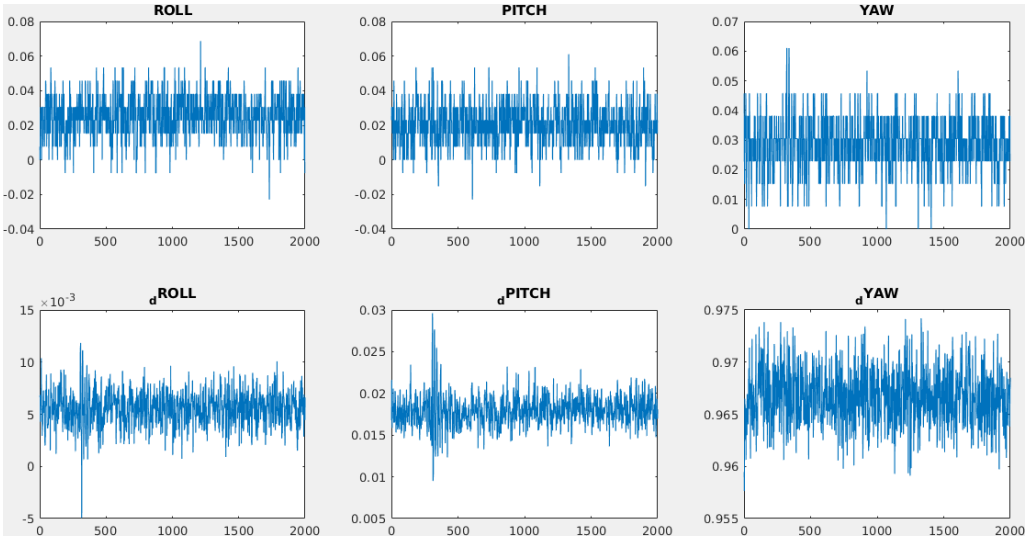


Figura 5.9: Lectura de la IMU

Radio

El proceso de la radio obtiene la información de la radio mediante uart. Luego descodifica la información del serial y la publica en el topic. La publicación es asíncrona, aunque la recepción por puerto serie es cada 20 ms, si han cambiado los valores con respecto al ultimo envió.

La recepción del serial tiene hilo propio para la recepción, el cual ejecuta el siguiente método, este funciona de la siguiente forma, cada vez que se recibe un mensaje, este se recibe si recibe un número de bits equivalente a un número mágico de 8 bits previamente elegido, el arduino envía un paquete con este número al principio, seguido del numero de bytes que ocupa la información, seguido de la información. Por probabilidad, alguna vez este número mágico puede coincidir con un grupo de bits, y no tiene que se necesariamente un paquete, para ello, introducimos unos corchetes en el paquete, de esta forma, reducimos la probabilidad, aninando dos coincidencias. Proseguimos, el paquete esta compuesto por 6 números de 2 bytes cada uno, y por tanto, debemos desempaquetamos de 8 en 8 bits, en grupos de 2 y vamos juntando hasta obtener los 6 números deseados. Cabe destacar, establecemos un pequeño filtro, el cual sino esta entre los valores deseados, cogemos el valor anterior obtenido. Por último, con los valores obtenidos generamos un paquete y los enviamos al algoritmo de control.

Si por algún casual, apagamos la radio, el arduino detecta que esta se ha apagado, y envía por último, los valores por defecto de la radio, dejando así el drone en su posición establece, para un futuro sería conveniente añadir un aterrizaje de emergencia.

```
1 int idx;
2 int canal[6];
3 int antcanal[6];
4 void *readThread(void *param) {
5   for(;;) {
6     MessageUART *m = receiveMessage();
7     if (m!=NULL) {
8       std::string strMessage(m->payLoad);
```

```
9         std::string strDataFrame;
10        if(strMessage.c_str()[0] == '['){
11          //std::cout << "Recibido! ";
12          antcanal[0] = canal[0];
13          antcanal[1] = canal[1];
14          antcanal[2] = canal[2];
15          antcanal[3] = canal[3];
16          antcanal[4] = canal[4];
17          antcanal[5] = canal[5];
18          for(std::size_t i = 1; i <= 12; i = i + 2){
19            idx = (i-1)/2;
20            bitset<8> high(strMessage[i+1]);
21            bitset<8> low(strMessage[i]);
22            bitset<16> join(high.to_ulong() * 0x100 + low.to_ulong());
23            canal[idx] = bitSet16ToInt(join);
24                          if(canal[idx] < 1000 || canal[idx] > 2000)
                                canal[idx] = antcanal[idx];
25          }
26          char buffer[50];
27          sprintf(buffer,"%i:%i:%i:%i:%i:%i",canal[0],canal[1],canal[2],
                canal[3],canal[4],canal[5]);
28          topic->publish((byte*)buffer,strlen(buffer)+1);
29        }
30      }
31      delete m;
32      usleep(10000);
33      }
34    return NULL;
35  }
```

CAPÍTULO 6
Pruebas

El capítulo de pruebas será breve en este trataremos de explicar el montaje resultante y los resultados obtenidos hasta el momento. Con respecto al montaje tenemos una separación de unos 35 mm entre chapa superior e inferior, entre ellas podemos encontrar resguardadas a la beaglebone, el sensor inercial, el arduino leonardo y el receptor radio. Se puede apreciar que en la tapa superior esta el cable de alimentación conectado a la PDB proporcionando energia tanto a los motores como a la beaglebone, además podemos ver como la tarjeta de wifi sobresale del drone para permitir conectividad con el ordenador.

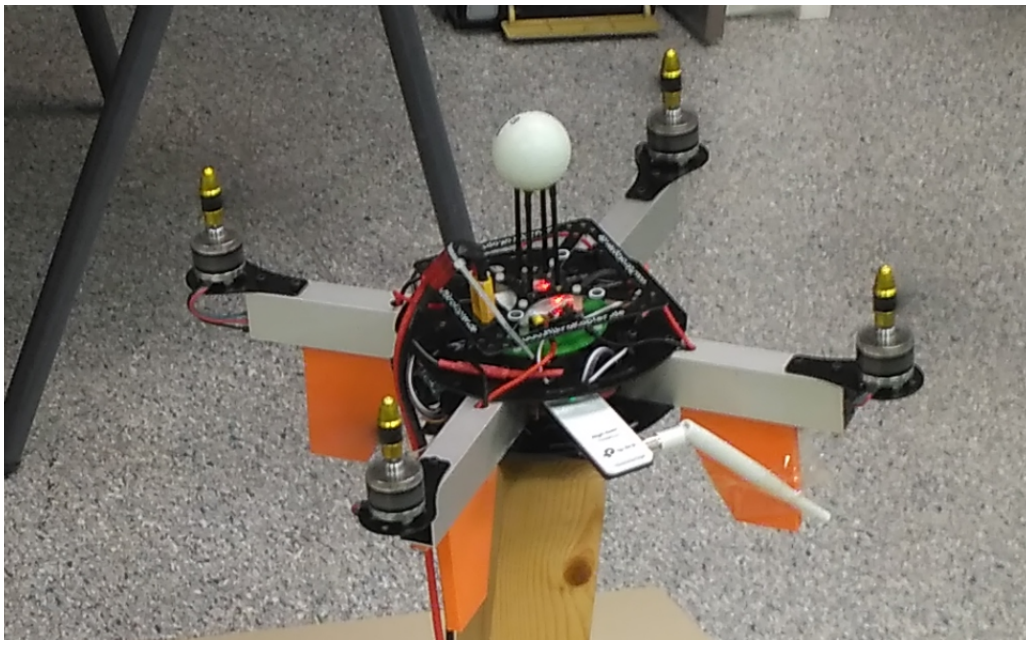


Figura 6.1: Drone

Para la realización de pruebas de control tenemos un plataforma articulada. Esta consiste en un pie de madera sujeto sobre un poste de madera con una tuerca a su extremo, por otro lado, el drone tiene un tornillo con la cabeza redondeada a modo de rótula para permitirle una cierta libertad de movilidad. Como podemos ver en la siguiente imagen, el drone esta en funcionamiento haciendo suficiente fuerza y control como para mantenerse sobre horizontal.

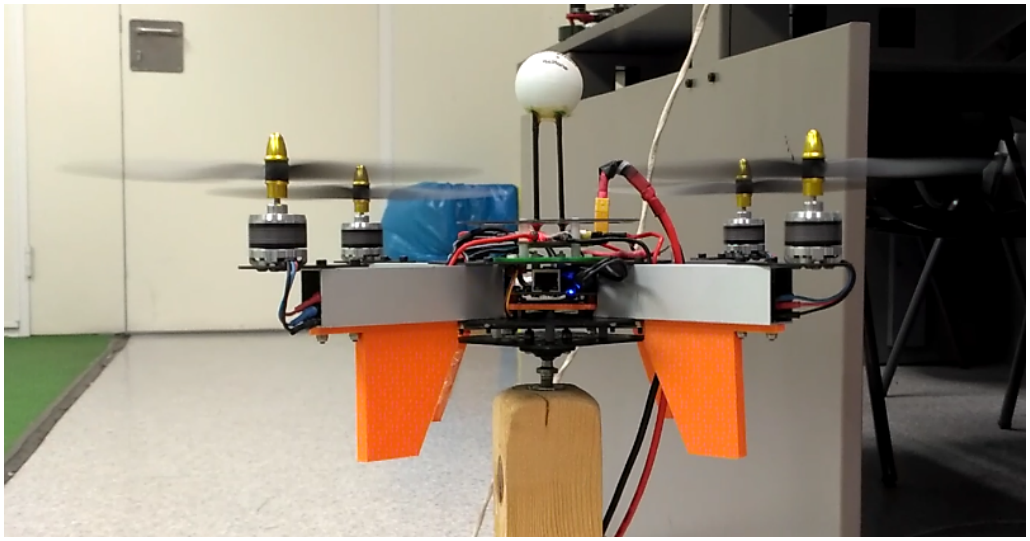


Figura 6.2: Pruebas

La siguiente imagen consiste en variar de 0 a 5 grados la referencia del angulo del Roll, aunque como podemos ver si que hay control en el angulo del Roll y Pitch, el control sobre el angulo del Yaw no tiene perfectamente sintonizados los parámetros y por tanto, genera un pequeña desviación.

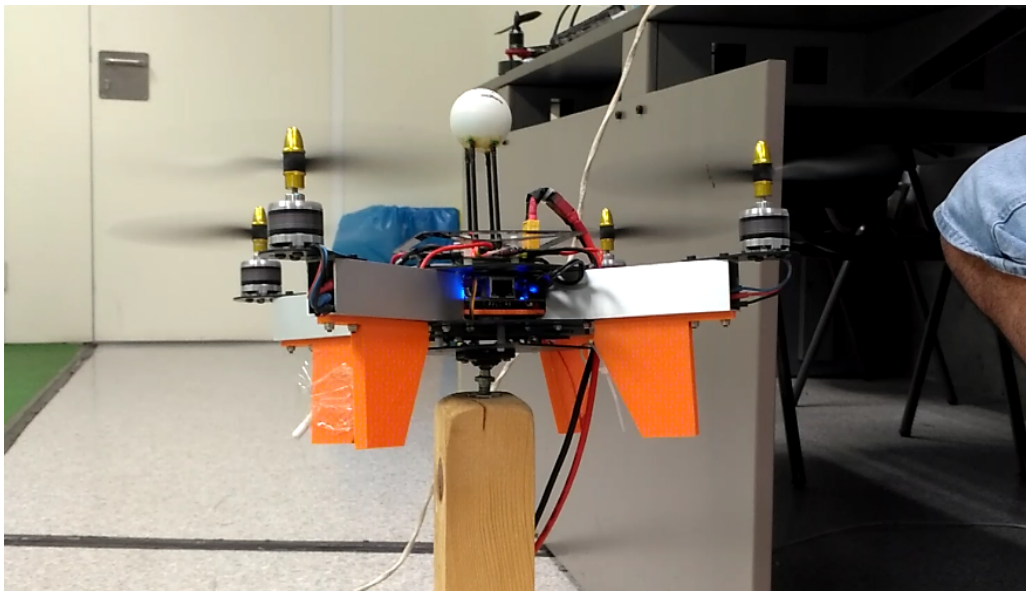


Figura 6.3: Pruebas

Por último, mostrar una imagen completa del puesto de trabajo, también comentar que aunque el control no es total, se puede apreciar que ya hay leyes de control que actúan sobre el drone y en vista al tiempo y dedicación a lo largo de 6 meses, los resultados son mas que satisfactorios, además que el proyecto sigue adelante, para un futuro proyecto de máster o personal.

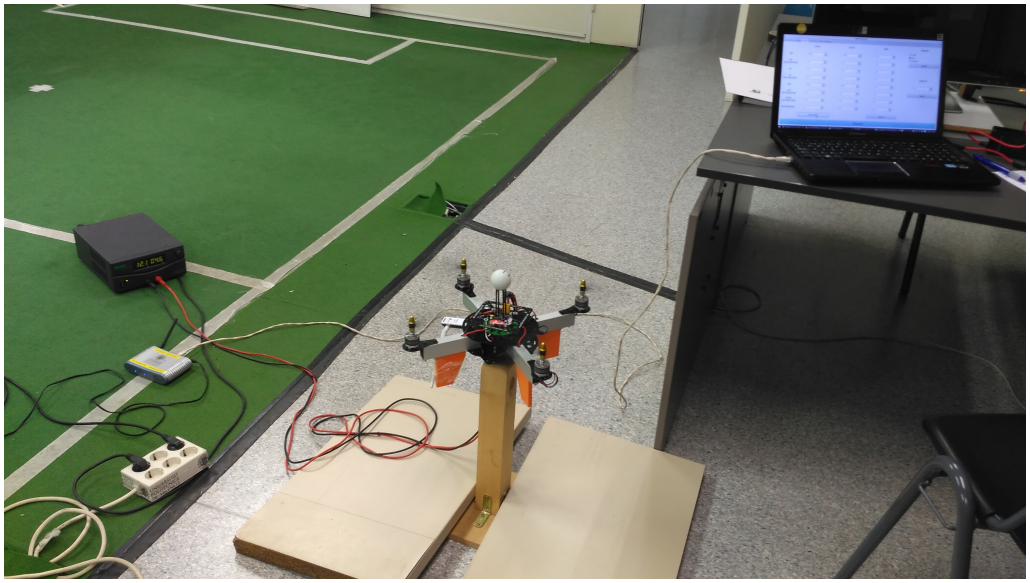


Figura 6.4: Puesto de trabajo

CAPÍTULO 7
Conclusiones y trabajos futuros

Conclusiones

En cuanto al resultado del trabajo parcialmente hemos cumplido objetivos, ya que por una parte hemos sido capaces de llegar a realizar todo el montaje y diseño del drone, tanto a nivel hardware como a nivel software. El control se ha completado sobre un rotula, ya que si que era capaz de sostenerse sobre un rotula y cumplir con la referencia del angulo, aunque si es cierto, que en vuelo habría que pulir ciertos detalles no contemplados ni esperados al inicio del proyecto, como son fallos en el sensor inercial, retardo de tiempo y la falta de planificación del proyecto. Y la falta de seguridad, ya que al no tener una estructura externa que protegiera frente al vuelo, es peligroso volar este tipo de aeronaves, ya que al girar los motores a grandes velocidades puede producir cortes.

Por una parte, hemos logrado desde instalar el sistema operativo con el parche de tiempo de real a la BeagleBone Black desde cero, realizar programas que se comuniquen por completo sobre un framework basado en canales de comunicación como es CKMultipeer, la lectura de sensores y la respuesta de actuadores con la BeagleBone, además de implementar un pequeño protocolo de tiempo de real para la comunicación con Arduino. Teniendo en cuenta que no teníamos conocimiento previo del tema, considerando que venimos de un grado de informática, el aprendizaje desarrollado para la construcción, búsqueda de componentes montaje y puesta en marcha, es más que satisfactoria a nivel personal.

Por otra parte, también se aprende de los errores cometidos en este tipo de proyectos, primero de todo, la planificación con metas semanales o mensuales en el camino del desarrollo ha sido un gran handicap, ya que ha sido determinante, porque al no realizarla para este proyecto, te da la visión para darle el suficiente valor necesario para realizar una planificación en próximos proyectos. Otra conclusión es la dedicación a la documentación, es decir, al no realizar un documentación diaria con los progresos se olvida con facilidad, muchas de las acciones o decisiones realizas para llegar hasta este punto, se olvidan y es mejor documentarlas mientras suceden. Por último, para cerrar esta parte de autoevaluación, habrá que aprender a moderar la ambición personal cuando se piensa en un proyecto de tal envegardura, ya que requiere esfuerzo y dedicación.

Personalmente, agracedecer a mis familiares, a mi tutor, al ai2 por el apoyo recibido. Ha sido una experiencia enriquecedora a nivel personal y nivel multi-

disciplinar, ya que al salir de la zona de confort de la programación y la informática encuentras pequeños retos matemáticos, físicos, mecánicos y electrónicos, apasionantes que hacen del campo de las ingenierías, y sobretodo de los drones, muy apasionante.

Trabajos futuros

Como decíamos en la conclusión, la intención es seguir en el campo de los drones. Por ello, queríamos establecer nuevas metas y avances para futuros trabajos relacionados.

- La primera meta es conseguir el permiso para poder realizar un vuelo en el exterior.
- Realizar un anillo de sensores o usar un sensor tipo LIDAR para evitar tener colisiones contra obstáculos.
- Estudiar nuevos modos, por ejemplo, si falla algún componente actuar en consecuencia para realizar acciones de actuación de emergencia.
- Incluir un GPS para la navegación.
- Mejorar el chasis del drone.
- Implementar un control mas avanzado basado en nuevas técnicas.
- Implementar un control en posición para poder realizar planificación de trayectorias en tres dimensiones.

Bibliografía

[1] Randal Beard. Quadrotor Dynamics and Control Rev 0.1. *Brigham Young University*, 2008-05-05.

[2] Origen y desarrollo de los drones. Publicado el 9 julio, 2015. Consultado a http://drones.uv.es/origen-y-desarrollo-de-los-drones/.

[3] How to choose Motor for Racing Drone & Quadcopter. Updated Mar 2018. Consultado a https://oscarliang.com/quadcopter-motor-propeller/.

[4] MPU-9250 Nine-Axis MEMS MotionTracking™ Device. Consultado a https://www.invensense.com/products/motion-tracking/9-axis/mpu-9250/.

[5] Turnigy Multistar 20A Delgado V2 ESC Con BLHeli OPTO 2-6S. Consultado a https://hobbyking.com/es_es/turnigy-multistar-20a-slim-v2-esc-with-blheli-opto-2-6s.html.

[6] How to choose ESC for Racing Drones and Quadcopters. Consultado a https://oscarliang.com/choose-esc-racing-drones/.

[7] PDB-XT60 w/ BEC 5V & 12V Consultado a http://www.mateksys.com/?portfolio=pdb-xt60.

[8] PWM, OneShot125, OneShot42, Multishot and DSHOT comparison. August 31, 2016. Consultado a https://quadmeup.com/pwm-oneshot125-oneshot42-and-multishot-comparison/.

[9] Historia de los drones. Desde el siglo XIX hasta el 2016: evolución y surgimiento del drone moderno. Consultado a http://eldrone.es/historia-de-los-drones/.

[10] Amazon Prime Air Consultado a https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011.

[11] Drone market shows positive outlook with strong industry growth and trends. Consultado a https://www.businessinsider.com/drone-industry-analysis-market-trends-growth-forecasts-2017-7?IR=T.

[12] Quadcopter frame, lipo battery, motor and propeller size matching table. Consultado a http://discuss.px4.io/t/quadcopter-frame-lipo-battery-motor-and-propeller-size-matching-table/4053.

[13] LiPo Batteries - How to choose the best battery for your drone. Consultado a https://www.dronetrest.com/t/lipo-batteries-how-to-choose-the-best-battery-for-your-drone/1277.

[14] How to choose battery capacity for longer flight time. Consultado a https://oscarliang.com/how-to-choose-battery-for-quadcopter-multicopter/.

[15] How to choose LiPo Battery for Mini Quad, Drones and Quadcopters. Consultado a https://oscarliang.com/lipo-battery-guide/.

APÉNDICE A
Programación

Esta función sirve leer la salida del programa de la imu. El programa de la imu genera archivo con la extensión imuLog para procesar a posteriori con mayor precisión el resultado del sensor inercial.

```
r = []; % Roll
p = []; % Pitch
y = []; % Yaw
dr = []; % Derivative Roll
dp = []; % Derivative Pitch
dy = []; % Derivative Yaw

% Load Files with GUI from current folder / logs folder with files
    extension imuLog
[file,path] = uigetfile('./logs/*.imuLog');
if file == 0
    return
end

% Processing file to array holder
nlines = 0;
fid = fopen(strcat(path,file));
tline = fgetl(fid);
while ischar(tline)
    nlines = nlines + 1;
    line = str2double(strsplit(tline,':'));
    r = [r line(1)];
    p = [p line(2)];
    y = [y line(3)];
    dr = [dr line(4)];
    dp = [dp line(5)];
    dy = [dy line(6)];
    tline = fgetl(fid);
end

% Load content in graphics subplots
fclose(fid);

figure

subplot(2,3,1)
plot(r(1,:))
title('ROLL')

```

```
subplot(2,3,2)
plot(p(1,:))
title('PITCH')

subplot(2,3,3)
plot(y(1,:))
title('YAW')

subplot(2,3,4)
plot(dr(1,:))
title('_dROLL')

subplot(2,3,5)
plot(dp(1,:))
title('_dPITCH')

subplot(2,3,6)
plot(dy(1,:))
title('_dYAW')
```

Este es el código de control que se encarga de realizar el control, recibir los mensajes del HMI para aplicar los parametros de control, enviar un hearbeat para notificar que esta vivo el quadrotor y conectar a la imu para recibir parte de la información sensorial.

```
/**
 * @author Jose Simo (c) ai2-UPV 2016
 *
 * example01.cpp
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <mutex>
#include <time.h>
#include <string>

#include "Net.h"
#include "TopicScope.h"
#include "MultiPeer.h"
#include "MessagePlain.h"

#include "ControlData.h"

#include "RadioListener.h"
#include "IMUListener.h"
#include "HMIListener.h"
#include "PWMuniv.h"

#include "util.h"

using namespace std;
using namespace BBB;

// Used by thread
```

```
34 int frequency = 400; // Hacer una prueba con mayor frecuencia
35 BBB::PWM *esc1PWM;
36 BBB::PWM *esc2PWM;
37 BBB::PWM *esc3PWM;
38 BBB::PWM *esc4PWM;
39
40 std::mutex motores;
41
42 ControlData controlData;
43 Topic* topicHMI;
44
45
46 int done = 0;
47 void sigint_handler(int sig);
48
49 void register_sig_handler()
50 {
51   struct sigaction sia;
52
53   bzero(&sia, sizeof sia);
54   sia.sa_handler = sigint_handler;
55
56   if (sigaction(SIGINT, &sia, NULL) < 0) {
57     perror("sigaction(SIGINT)");
58     exit(1);
59   }
60 }
61
62 void sigint_handler(int sig)
63 {
64     done = 1;
65 }
66
67 int mapInt(int x, int in_min, int in_max, int out_min, int out_max)
68 {
69   return (x - in_min) * (out_max - out_min) / (in_max - in_min) +
       out_min;
70 }
71
72 void *sendHeartBeat(void *param){
73
74     char buffer[200];
75
76   string loadStr = "4:DRONE RAFA";
77   int msgSize = loadStr.length();
78   const char *load = loadStr.c_str();
79   memcpy(buffer,load,msgSize+1);
80
81     int period = 150000; // 150 ms
82     struct timespec start;
83     clock_gettime(CLOCK_MONOTONIC, &start);
84     while(!done){
85   topicHMI->publish((unsigned char *)buffer,msgSize+1);
86   heartTime = getCurrentMicroseconds();
87   sleep_until(&start, period);
88     }
89 }
90
91 float saturacion(float value, float min, float max){
```

```
92    if(value > max) return max;
93    else if(value < min) return min;
94    return value;
95  }
96
97  float minPWM = 4.75f;
98  float maxPWM = 10.0f;
99
100 void setMotor(float esc1Value, float esc2Value, float esc3Value, float
        esc4Value) {
101
102         esc1Value = minPWM + esc1Value * (maxPWM - minPWM);
103         esc2Value = minPWM + esc2Value * (maxPWM - minPWM);
104         esc3Value = minPWM + esc3Value * (maxPWM - minPWM);
105         esc4Value = minPWM + esc4Value * (maxPWM - minPWM);
106
107         esc1PWM->setDutyCycle(esc1Value);
108         esc2PWM->setDutyCycle(esc2Value);
109         esc3PWM->setDutyCycle(esc3Value);
110         esc4PWM->setDutyCycle(esc4Value);
111
112 }
113
114 void *calculatePID(void *param){
115
116     char buffer[400];
117
118     // MODE
119     const int disarmmode = 0;
120     const int flymode = 1;
121     const int testmode = 2;
122
123     int maxD = 30; // MAX Degrees
124     int maxDs = 180; // MAX Degrees per second
125
126     int mode;
127     int lastmode;
128
129     // FLY MODE VARS
130     RadioChannels rcData;
131     HMIParams hmiParams;
132     HMIRef hmiRef;
133     AnglesRPY angles;
134
135     float pid_p_gain_roll = 0;
136     float pid_p_gain_pitch = 0;
137     float pid_p_gain_yaw = 0;
138
139     float pid_p_gain_roll_sat = 0;
140     float pid_p_gain_pitch_sat = 0;
141     float pid_p_gain_yaw_sat = 0;
142
143     float pid_i_gain_roll = 0;
144     float pid_i_gain_pitch = 0;
145     float pid_i_gain_yaw = 0;
146
147     float pid_i_gain_roll_sat = 0;
148     float pid_i_gain_pitch_sat = 0;
149     float pid_i_gain_yaw_sat = 0;
```

```
150
151     float pid_d_gain_roll = 0;
152     float pid_d_gain_pitch = 0;
153     float pid_d_gain_yaw = 0;
154
155     float pid_d_gain_roll_sat = 0;
156     float pid_d_gain_pitch_sat = 0;
157     float pid_d_gain_yaw_sat = 0;
158
159     float pid_main_roll_sat = 0;
160     float pid_main_pitch_sat = 0;
161     float pid_main_yaw_sat = 0;
162
163     float pid_error_temp = 0;
164     float pid_i_mem_roll, pid_roll_setpoint, pid_output_roll,
            pid_p_roll, pid_d_roll = 0;
165     float pid_i_mem_pitch, pid_pitch_setpoint, pid_output_pitch,
            pid_p_pitch, pid_d_pitch = 0;
166     float pid_i_mem_yaw, pid_yaw_setpoint, pid_output_yaw, pid_p_yaw,
            pid_d_yaw = 0;
167
168     //float gyro_roll_input, gyro_pitch_input, gyro_yaw_input = 0;
169     float pid_last_roll_d_error, pid_last_pitch_d_error,
            pid_last_yaw_d_error = 0;
170
171     float e1 = 0;
172     float e2 = 0;
173     float e3 = 0;
174     float e4 = 0;
175
176     float throttle = 0;
177
178     // TEST MODE VARS
179
180     // DISARM MODE VARS
181     long us2ms = 1000;
182     long disarmTime;
183     long disarmTimeDesactive;
184     long disarmTimeDesactiveBase = 2 * us2ms * us2ms; // 10 segundos
185
186     int cont = 0;
187     // Time period bucle
188     int period = 2 * (int)us2ms;
189     struct timespec start;
190     clock_gettime(CLOCK_MONOTONIC, &start);
191     while(!done){
192         lastmode = mode;
193         mode = controlData.getMode();
194
195     angles = controlData.getAngles();
196
197     if(controlData.isUpdated()){
198
199       hmiParams = controlData.getHMIParams();
200
201       pid_p_gain_roll = hmiParams.KPRoll;
202       pid_p_gain_pitch = hmiParams.KPPitch;
203       pid_p_gain_yaw = hmiParams.KPYaw;
204
```

```
205         pid_i_gain_roll = hmiParams.KIRoll;
206         pid_i_gain_pitch = hmiParams.KIPitch;
207         pid_i_gain_yaw = hmiParams.KIYaw;
208 
209         pid_d_gain_roll = hmiParams.KDRoll;
210         pid_d_gain_pitch = hmiParams.KDPitch;
211         pid_d_gain_yaw = hmiParams.KDYaw;
212 
213         pid_main_roll_sat = hmiParams.GRollSat;
214         pid_main_pitch_sat = hmiParams.GPitchSat;
215         pid_main_yaw_sat = hmiParams.GYawSat;
216       }
217 
218       // CHECK MODE RADIO AND BLOCK ARM MODE WHEN DISARM MODE LOCK
219       if(controlData.getStatusHMI() == 0){ // 0 - Libre, no hay control
           hmi
220 
221         rcData = controlData.getRadioChannels();
222 
223         if(rcData.canal3 < 1000) mode = disarmmode;
224         else if(lastmode == disarmmode){
225           if((getCurrentMicroseconds() - disarmTime) >
                disarmTimeDesactive) mode = flymode;
226         } else {
227           mode = flymode;
228 
229           pid_roll_setpoint = mapInt(rcData.canal2, 1000, 2000, -maxD,
                maxD);
230           pid_pitch_setpoint = mapInt(rcData.canal1, 1000, 2000, -maxD,
                maxD);
231           pid_yaw_setpoint = mapInt(rcData.canal1, 1000, 2000, -maxDs,
                maxDs);
232         }
233         controlData.setMode(mode);
234         //cout << "Not Lock" << endl;
235 
236       } else {
237 
238         if((getCurrentMicroseconds() - heartTime) > 200 * us2ms)
              controlData.setUnlock();
239 
240         Thrust t = controlData.getThrust();
241         throttle = t.main;
242         hmiRef = controlData.getRef();
243         pid_roll_setpoint = hmiRef.SetRoll;
244         pid_pitch_setpoint = hmiRef.SetPitch;
245         pid_yaw_setpoint = hmiRef.SetYaw;
246 
247         //cout << "Lock" << endl;
248 
249       }
250 
251 
252       switch(mode) {
253         case disarmmode:
254           //if(lastmode != mode){
255             // Sumar las potencias de los motores, y sacar la media,
256             // Luego aplicar a cada motor la media
257             // Con la media calcular el tiempo de apagado
```

```
258         //}
259         // Aplicar una potencia deseada, siempre que se haya modificido
                , y no sea distinta de 0 para motores
260         e1 = 0.0f;
261         e2 = 0.0f;
262         e3 = 0.0f;
263         e4 = 0.0f;
264         setMotor(0.0f, 0.0f, 0.0f, 0.0f);
265         //char buffer [200];
266         //sprintf (buffer, "%.3f:%.3f:%.3f:%.3f", 0.0f, 0.0f, 0.0f, 0.0
                f);
267         //topicMOTOR->publish((byte*)buffer,strlen(buffer)+1);
268         break;
269
270       case flymode:
271
272         ////////////////////////////
273         // ROLL
274
275         pid_error_temp =(-1*angles.Roll_dot) - pid_roll_setpoint;
276
277         // Integral
278
279         pid_i_mem_roll = pid_i_mem_roll + pid_i_gain_roll *
                pid_error_temp;
280         pid_i_mem_roll = saturacion(pid_i_mem_roll, -pid_main_roll_sat,
                 pid_main_roll_sat);
281
282         // Proporcional
283
284         pid_p_roll = pid_p_gain_roll * pid_error_temp;
285
286         //if(pid_p_roll > pid_p_gain_roll_sat) pid_p_roll =
                pid_p_gain_roll_sat;
287         //else if(pid_p_roll < -pid_p_gain_roll_sat) pid_p_roll = -
                pid_p_gain_roll_sat;
288
289         // Derivada
290
291         pid_d_roll = -pid_d_gain_roll * (pid_error_temp -
                pid_last_roll_d_error);
292
293         //if(pid_d_roll > pid_d_gain_roll_sat) pid_d_roll =
                pid_d_gain_roll_sat;
294         //else if(pid_d_roll < -pid_d_gain_roll_sat) pid_d_roll = -
                pid_d_gain_roll_sat;
295
296         // OUTPUT
297
298         pid_output_roll = pid_p_roll + pid_i_mem_roll + pid_d_roll;
299
300         pid_last_roll_d_error = pid_error_temp;
301
302         ////////////////////////////
303         // PITCH
304
305         pid_error_temp = angles.Pitch_dot - pid_pitch_setpoint;
306
307         // Integral
```

```
308
309         pid_i_mem_pitch = pid_i_mem_pitch + pid_i_gain_pitch *
                pid_error_temp;
310         pid_i_mem_pitch = saturacion(pid_i_mem_pitch, -
                pid_main_pitch_sat, pid_main_pitch_sat);
311
312         // Proporcional
313
314         pid_p_pitch = pid_p_gain_pitch * pid_error_temp;
315
316         //if(pid_p_pitch > pid_p_gain_pitch_sat) pid_p_pitch =
                pid_p_gain_pitch_sat;
317         //else if(pid_p_pitch < -pid_p_gain_pitch_sat) pid_p_pitch = -
                pid_p_gain_pitch_sat;
318
319         // Derivada
320
321         pid_d_pitch = -pid_d_gain_pitch * (pid_error_temp -
                pid_last_pitch_d_error);
322
323         //if(pid_d_pitch > pid_d_gain_pitch_sat) pid_d_pitch =
                pid_d_gain_pitch_sat;
324         //else if(pid_d_pitch < -pid_d_gain_pitch_sat) pid_d_pitch = -
                pid_d_gain_pitch_sat;
325
326         // OUTPUT
327
328         pid_output_pitch = pid_p_pitch + pid_i_mem_pitch + pid_d_pitch;
329
330         pid_last_pitch_d_error = pid_error_temp;
331
332         ////////////////////////////////
333         // YAW
334
335         pid_error_temp = angles.Yaw_dot - pid_yaw_setpoint;
336
337         // Integral
338
339         pid_i_mem_yaw = pid_i_mem_yaw + pid_i_gain_yaw * pid_error_temp
                ;
340         pid_i_mem_yaw = saturacion(pid_i_mem_yaw, -pid_main_yaw_sat,
                pid_main_yaw_sat);
341
342         // Proporcional
343
344         pid_p_yaw = pid_p_gain_yaw * pid_error_temp;
345
346         //if(pid_p_yaw > pid_p_gain_yaw_sat) pid_p_yaw =
                pid_p_gain_yaw_sat;
347         //else if(pid_p_yaw < -pid_p_gain_yaw_sat) pid_p_yaw = -
                pid_p_gain_yaw_sat;
348
349         // Derivada
350
351         pid_d_yaw = -pid_d_gain_yaw * (pid_error_temp -
                pid_last_yaw_d_error);
352
353         //if(pid_d_yaw > pid_d_gain_yaw_sat) pid_d_yaw =
                pid_d_gain_yaw_sat;
```

```
354         //else if(pid_d_yaw < -pid_d_gain_yaw_sat) pid_d_yaw = -
                pid_d_gain_yaw_sat;
355
356         // OUTPUT
357
358         pid_output_yaw = pid_p_yaw + pid_i_mem_yaw + pid_d_yaw;
359
360         pid_last_yaw_d_error = pid_error_temp;
361
362
363         ////////////////////////////////
364
365         // MAIN SATS
366
367         pid_output_roll  = saturacion(pid_output_roll, -
                pid_main_roll_sat, pid_main_roll_sat);
368         pid_output_pitch = saturacion(pid_output_pitch, -
                pid_main_pitch_sat, pid_main_pitch_sat);
369         pid_output_yaw   = saturacion(pid_output_yaw, -
                pid_main_yaw_sat, pid_main_yaw_sat);
370
371         e1 = throttle + pid_output_pitch - pid_output_roll -
                pid_output_yaw;
372         e2 = throttle - pid_output_pitch - pid_output_roll +
                pid_output_yaw;
373         e3 = throttle - pid_output_pitch + pid_output_roll -
                pid_output_yaw;
374         e4 = throttle + pid_output_pitch + pid_output_roll +
                pid_output_yaw;
375
376         e1 = saturacion(e1, 0.0f, 1.0f);
377         e2 = saturacion(e2, 0.0f, 1.0f);
378         e3 = saturacion(e3, 0.0f, 1.0f);
379         e4 = saturacion(e4, 0.0f, 1.0f);
380
381         setMotor(e1, e2, e3, e4);
382         break;
383
384       case testmode:
385         if(lastmode == disarmmode){
386           e1 = 0.0f;
387           e2 = 0.0f;
388           e3 = 0.0f;
389           e4 = 0.0f;
390           setMotor(0.0f, 0.0f, 0.0f, 0.0f);
391           controlData.setThrustAll(0.0f);
392         }
393         Thrust t = controlData.getThrust();
394         setMotor(t.m1, t.m2, t.m3, t.m4);
395         break;
396     }
397
398     cont++;
399     if(cont > 160) {
400
401       //cout << "-------------------------------------------------------" <<
              endl;
402       //cout << "Thrust: " << throttle << endl;
```

```
403         //cout << "E: " << e1 << " " << e2  << " " << e3 << " " << e4 <<
                endl;
404         cout << "************************" << endl;
405         cout << "MODE: " << mode << endl;
406         cout << "************************" << endl;
407         cout << "ROLL" << endl;
408         cout << " KP:" << pid_p_gain_roll << " KD:" << pid_d_gain_roll <<
                endl;
409         cout << " Set:" << pid_roll_setpoint << " A.RD:" << angles.
                Roll_dot << endl;
410         cont = 0;
411       }
412
413       sleep_until(&start, period);
414       }
415 }
416
417
418
419 void printUsage() {
420         printf("Usage: -n NetworkInterfaceName -t TopicName -p period -h
                -s -r \n");
421         printf("\t -n NetworkInterfaceName: Set the network interface to
                bind the MultiPeer communications infrastructure \n");
422         printf("\t\t By default, if there are many interfaces available,
                a menu to choose it is displayed. \n");
423         printf("\t\t By default, if there is only one interface available
                , it is choosen. \n");
424         printf("\t -g xx.xx.xx.xx:pppp: Peers Group identification. Set
                the IP and port of the dicovering service.");
425         printf("\t\t By default, UDP discovering in the group
                228.10.1.1:4999");
426         printf("\t\t It can be used a TCP IP and port if a \"
                MPDiscoveringServiceDaemon\" is running there.");
427         printf("\t -t TopicName: Select the name of the topic (dcps). \n"
                );
428         printf("\t -h Prints this help. \n");
429         printf("\t -s MultiPeer silent mode (disable Log messages). \n");
430 }
431
432 int main(int argc, char **argv) {
433
434   pthread_t threadHeartBeat;
435   pthread_t threadPID;
436   pthread_attr_t attr;
437   pthread_attr_init(&attr);
438
439   string topicRCName = "RCDATA";
440   string topicIMUName = "IMUDATA";
441
442   string topicHMIName = "HMIParams";
443   string networkInterface = "";
444   string groupIPPortStr = "";
445
446   /*** Parse command line example ***/
447   opterr = 0;
448   char c;
449   bool endLoop = false;
450   while ((c = getopt (argc, argv, "n:hs")) != -1) {
```

```
451     switch (c) {
452       case 'n':
453         //sets the desired network interface name to bind multipeer
              communications infrastructure.
454             networkInterface = string(optarg);
455             break;
456           case 'g':
457         ////Set the discovering group IP and port
458             groupIPPortStr = string(optarg);
459             break;
460           case 'h':
461             printUsage();
462             return 1;
463             break;
464           case 's':
465             Log::setActive(false);
466             break;
467           case '?':
468             if (optopt == 'c')
469                 fprintf (stderr, "Option-%c requires an argument.\n",
                      optopt);
470             else if (isprint (optopt))
471                 fprintf (stderr, "Unknown option '-%c'.\n", optopt);
472             else
473                 fprintf (stderr, "Unknown option character '\\x%x'.\n",
                      optopt);
474             printUsage();
475             return 1;
476           default:
477             endLoop = true;
478             break;
479       }
480       if (endLoop == true) break;
481   }
482
483   ////////////////////////////
484   ///Multipeer initialization
485
486   // If the interface is not selected by the command line option,
487   // ask to the user to select one
488   if (networkInterface.size() == 0) {
489     //networkInterface = Net::menuUserSelectNetworkInterface();
490     map<string, string> myMap = Net::getInterfaces();
491         auto it = myMap.find("wlx7c8bca0ae363");
492         if (it != myMap.end()) networkInterface = "wlx7c8bca0ae363";
493     else{
494       auto it = myMap.find("eth0");
495           if (it != myMap.end()) networkInterface = "eth0";
496       //else{} AUTOCHOOSE
497     }
498   }
499
500   // Setup a MultiPeer Configuration Object
501   MultiPeerConf mpCfg;
502   mpCfg.setInterfaceName(string(networkInterface));
503   //Configure the IP and port of the discovering service
504   IPPort *discoveringIPPport = IPPort::parseIPPort(groupIPPortStr);
505   if (discoveringIPPport != NULL) {
```

```
506     mpCfg.setDiscoveringAddress(discoveringIPPport->getStringAddress())
            ;
507     mpCfg.setDiscoveringPort(discoveringIPPport->getPort());
508   }
509   //You can change also other properties p.e. the Initial Port:
510   //mpCfg.setInitialPort(9000);
511
512   //Build the communication artifacts
513   MultiPeer mp(&mpCfg);
514   TopicScope scope(&mp);
515
516
517     //PWM objects
518   esc1PWM = new PWM(PWM::P9_22);
519   esc1PWM->setFrequency(frequency);           //50Hz
520   esc1PWM->setDutyCycle(minPWM);
521   esc1PWM->setPolarity(PWM::ACTIVE_LOW);
522   esc1PWM->run();
523
524         esc2PWM = new PWM(PWM::P9_16);
525   esc2PWM->setFrequency(frequency);           //50Hz
526   esc2PWM->setDutyCycle(minPWM);
527   esc2PWM->setPolarity(PWM::ACTIVE_LOW);
528   esc2PWM->run();
529
530         esc3PWM = new PWM(PWM::P9_14);
531   esc3PWM->setFrequency(frequency);           //50Hz
532   esc3PWM->setDutyCycle(minPWM);
533   esc3PWM->setPolarity(PWM::ACTIVE_LOW);
534   esc3PWM->run();
535
536         esc4PWM = new PWM(PWM::P9_21);
537   esc4PWM->setFrequency(frequency);           //50Hz
538   esc4PWM->setDutyCycle(minPWM);
539   esc4PWM->setPolarity(PWM::ACTIVE_LOW);
540   esc4PWM->run();
541
542         // Radio Control
543   Topic *topicRC = scope.lookup(topicRCName);
544   RadioListener radioListener(&controlData);
545   topicRC->addListener(&radioListener);
546
547         // MOTORES
548   //topicMOTOR = scope.lookup(topicMOTORName);
549   //MotorListener motorListener(&controlData);
550   //topicMOTOR->addListener(&motorListener);
551
552         // IMU
553   Topic *topicIMU = scope.lookup(topicIMUName);
554   IMUListener ImuListener(&controlData);
555   topicIMU->addListener(&ImuListener);
556
557         // HMIParams
558   topicHMI = scope.lookup(topicHMIName);
559   HMIListener hmiListener(&controlData, topicHMI);
560   topicHMI->addListener(&hmiListener);
561
562   register_sig_handler();
563   //Application loop
```

```
564    //// void
565
566            pthread_create(&threadHeartBeat,&attr,sendHeartBeat,NULL);
567            pthread_create(&threadPID,&attr,calculatePID,NULL);
568
569            char buffer[200];
570    //This loop avoids the process termination
571    while(!done){
572                usleep(1000000); // 1 seg
573            }
574
575    pthread_join(threadHeartBeat,NULL);
576      pthread_join(threadPID,NULL);
577
578      delete esc1PWM;
579    delete esc2PWM;
580    delete esc3PWM;
581    delete esc4PWM;
582
583    printf("\nExit. \n");
584    return EXIT_SUCCESS;
585 }
```

Este código se encarga de leer la radio de control a distancia con un arduino Leonardo, que procesa las señales PWM de entrada y serializa esta información a la beaglebone.

```
1  #include <EEPROM.h>
2  #include <EEPROMAnything.h>
3  #include <SerialSendAnything.h>
4  #include <SimpleKalmanFilter.h>
5
6  /*
7   SimpleKalmanFilter(e_mea, e_est, q);
8   e_mea: Measurement Uncertainty
9   e_est: Estimation Uncertainty
10  q: Process Noise
11 */
12
13 int e_mea = 20;
14 int e_est = 20;
15 float q = 0.15;
16
17 SimpleKalmanFilter canal_1_filter(e_mea, e_est, q);
18 SimpleKalmanFilter canal_2_filter(e_mea, e_est, q);
19 SimpleKalmanFilter canal_3_filter(e_mea, e_est, q);
20 SimpleKalmanFilter canal_4_filter(e_mea, e_est, q);
21 SimpleKalmanFilter canal_5_filter(e_mea, e_est, q);
22 SimpleKalmanFilter canal_6_filter(e_mea, e_est, q);
23
24 bool debug = false;
25 bool debug_raw = false;
26 bool debug_normal = false;
27
28 byte numero_magico = 0xAF;
29 int numero_canales = 6;
30 long carga_util = numero_canales*sizeof(int) + 2;
31
32 // Gestion del tiempo
```

```
33 long tiempo_max_respuesta = 40000; // 40 ms para estar seguros,
       preguntar
34 long tiempo_actual;
35 long tiempo_ultima_impresion;
36 long tiempo_ultima_rutina;
37 long tiempo_actual_rutina_hardware;
38 long tiempo_actual_rutina0;
39 long tiempo_actual_rutina1;
40
41
42 // Valores para la normalizacion de los canales
43 int canal_margen = 10;
44 int canal_normal_max = 2000;
45 int canal_normal_mid = 1500;
46 int canal_normal_min = 1000;
47
48 float canal_1_estimated_value;
49 int canal_1_estado;
50 int canal_1_valor;
51 int canal_1_min;
52 int canal_1_mid;
53 int canal_1_max;
54 int canal_1_normal; // Valor normalizado entre 1000 y 2000
55 int canal_1_ant_normal;
56 long tiempo_1;
57
58 float canal_2_estimated_value;
59 int canal_2_estado;
60 int canal_2_valor;
61 int canal_2_min;
62 int canal_2_mid;
63 int canal_2_max;
64 int canal_2_normal; // Valor normalizado entre 1000 y 2000
65 int canal_2_ant_normal;
66 long tiempo_2;
67
68 float canal_3_estimated_value;
69 int canal_3_estado;
70 int canal_3_valor;
71 int canal_3_min;
72 int canal_3_max;
73 int canal_3_normal; // Valor normalizado entre 1000 y 2000
74 int canal_3_ant_normal;
75 long tiempo_3;
76
77 float canal_4_estimated_value;
78 int canal_4_estado;
79 int canal_4_valor;
80 int canal_4_min;
81 int canal_4_mid;
82 int canal_4_max;
83 int canal_4_normal; // Valor normalizado entre 1000 y 2000
84 int canal_4_ant_normal;
85 long tiempo_4;
86
87 float canal_5_estimated_value;
88 int canal_5_valor;
89 int canal_5_min;
90 int canal_5_max;
```

```
91  int canal_5_normal; // Valor normalizado entre 1000 y 2000
92  int canal_5_ant_normal;
93  long tiempo_5;
94
95  float canal_6_estimated_value;
96  int canal_6_valor;
97  int canal_6_min;
98  int canal_6_max;
99  int canal_6_normal; // Valor normalizado entre 1000 y 2000
100 int canal_6_ant_normal;
101 long tiempo_6;
102
103 void setup() {
104
105   tiempo_ultima_rutina = micros();
106   tiempo_ultima_impresion = tiempo_ultima_rutina;
107
108   PCICR  |= (1 << PCIE0); // Activo interrupcion por PC hardware 0.
109   PCMSK0 |= (1 << PCINT4); // Pin Digital 8
110   PCMSK0 |= (1 << PCINT5); // Pin Digital 9
111   PCMSK0 |= (1 << PCINT6); // Pin Digital 10
112   PCMSK0 |= (1 << PCINT7); // Pin Digital 11
113
114   pinMode(PD0, INPUT); // Pin Digital 3
115   attachInterrupt(digitalPinToInterrupt(3), rutinaInterrupcion0, CHANGE
          ); // // Activo interrupcion por hardware 0.
116
117   pinMode(PD1, INPUT); // Pin Digital 2
118   attachInterrupt(digitalPinToInterrupt(2), rutinaInterrupcion1, CHANGE
          ); // // Activo interrupcion por hardware 1.
119
120   EEPROM_readAnything(0, canal_1_min);
121   EEPROM_readAnything(1, canal_1_mid);
122   EEPROM_readAnything(2, canal_1_max);
123
124   EEPROM_readAnything(3, canal_2_min);
125   EEPROM_readAnything(4, canal_2_mid);
126   EEPROM_readAnything(5, canal_2_max);
127
128   EEPROM_readAnything(6, canal_3_min);
129   EEPROM_readAnything(7, canal_3_max);
130
131   idleDelay(1000);
132   EEPROM_readAnything(8, canal_4_min);
133   EEPROM_readAnything(9, canal_4_mid);
134   EEPROM_readAnything(10, canal_4_max);
135
136   EEPROM_readAnything(11, canal_5_min);
137   EEPROM_readAnything(12, canal_5_max);
138
139   EEPROM_readAnything(13, canal_6_min);
140   EEPROM_readAnything(14, canal_6_max);
141
142   canal_4_mid = 1540;
143
144   canal_1_min = canal_1_min + 20;
145   canal_1_max = canal_1_max - 20;
146
147   canal_4_min = canal_4_min + 20;
```

```
148   canal_4_max = canal_4_max - 20;
149
150   canal_2_min = canal_2_min + 100;
151   canal_2_max = canal_2_max - 100;
152
153   canal_3_max = canal_3_max - 100;
154
155   Serial1.begin(115200);
156   while (!Serial1) {}
157
158   if(debug){
159     Serial.begin(115200);
160     while (!Serial) {}
161
162     Serial.println("--------------------");
163     Serial.println("CANAL MIN MID MAX");
164
165     Serial.print("1");
166     Serial.print("\t");
167     Serial.print(canal_1_min);
168     Serial.print(" ");
169     Serial.print(canal_1_mid);
170     Serial.print(" ");
171     Serial.print(canal_1_max);
172     Serial.println();
173
174     Serial.print("2");
175     Serial.print("\t");
176     Serial.print(canal_2_min);
177     Serial.print(" ");
178     Serial.print(canal_2_mid);
179     Serial.print(" ");
180     Serial.print(canal_2_max);
181     Serial.println();
182
183     Serial.print("3");
184     Serial.print("\t");
185     Serial.print(canal_3_min);
186     Serial.print(" ");
187     Serial.print("----");
188     Serial.print(" ");
189     Serial.print(canal_3_max);
190     Serial.println();
191
192     Serial.print("4");
193     Serial.print("\t");
194     Serial.print(canal_4_min);
195     Serial.print(" ");
196     Serial.print(canal_4_mid);
197     Serial.print(" ");
198     Serial.print(canal_4_max);
199     Serial.println();
200
201     Serial.print("5");
202     Serial.print("\t");
203     Serial.print(canal_5_min);
204     Serial.print(" ");
205     Serial.print("----");
206     Serial.print(" ");
```

```
207       Serial.print(canal_5_max);
208       Serial.println();
209
210       Serial.print("6");
211       Serial.print("\t");
212       Serial.print(canal_6_min);
213       Serial.print(" ");
214       Serial.print("____");
215       Serial.print(" ");
216       Serial.print(canal_6_max);
217       Serial.println();
218
219       Serial.println("--------------------");
220     }
221
222     idleDelay(10000);
223   }
224
225   void loop() {
226     tiempo_actual = micros();
227     if(tiempo_actual - tiempo_ultima_rutina < tiempo_max_respuesta){
228       canal_1_estimated_value = canal_1_filter.updateEstimate(
              canal_1_valor);
229       canal_2_estimated_value = canal_2_filter.updateEstimate(
              canal_2_valor);
230       canal_3_estimated_value = canal_3_filter.updateEstimate(
              canal_3_valor);
231       canal_4_estimated_value = canal_4_filter.updateEstimate(
              canal_4_valor);
232       canal_5_estimated_value = canal_5_filter.updateEstimate(
              canal_5_valor);
233       canal_6_estimated_value = canal_6_filter.updateEstimate(
              canal_6_valor);
234       enviarMensajeS1();
235     } else {
236       Serial1.write(numero_magico);
237       Serial_SendAnything(&Serial1, carga_util);
238
239       Serial1.write("[");
240       Serial_SendAnything(&Serial1, canal_normal_mid);
241       Serial_SendAnything(&Serial1, canal_normal_mid);
242       Serial_SendAnything(&Serial1, canal_normal_min);
243       Serial_SendAnything(&Serial1, canal_normal_mid);
244       Serial_SendAnything(&Serial1, canal_5_normal);
245       Serial_SendAnything(&Serial1, canal_6_normal);
246       Serial1.write("]");
247     }
248
249     if(debug_raw){
250       Serial.print(canal_1_valor);
251       Serial.print(",");
252       Serial.print(canal_2_valor);
253       Serial.print(",");
254       Serial.print(canal_3_valor);
255       Serial.print(",");
256       Serial.print(canal_4_valor);
257       Serial.print(",");
258       Serial.print(canal_5_valor);
259       Serial.print(",");
```

```
260       Serial.print(canal_6_valor);
261       Serial.println();
262     }
263
264     waitUntil(40); // loop cada 10 ms
265 }
266
267 void enviarMensajeS1(){
268
269     canal_1_valor = (int) canal_1_estimated_value;
270     canal_2_valor = (int) canal_2_estimated_value;
271     canal_3_valor = (int) canal_3_estimated_value;
272     canal_4_valor = (int) canal_4_estimated_value;
273     canal_5_valor = (int) canal_5_estimated_value;
274     canal_6_valor = (int) canal_6_estimated_value;
275
276     canal_1_ant_normal = canal_1_normal;
277     canal_2_ant_normal = canal_2_normal;
278     canal_3_ant_normal = canal_3_normal;
279     canal_4_ant_normal = canal_4_normal;
280     canal_5_ant_normal = canal_5_normal;
281     canal_6_ant_normal = canal_6_normal;
282
283     if(canal_1_valor > canal_1_max) canal_1_normal = canal_normal_max;
284     else if(canal_1_valor < canal_1_min) canal_1_normal =
            canal_normal_min;
285     else if(canal_1_valor > canal_1_mid) canal_1_normal = map(
            canal_1_valor, canal_1_mid + 1, canal_1_max, canal_normal_mid +
            1, canal_normal_max);
286     else canal_1_normal = map(canal_1_valor, canal_1_min, canal_1_mid,
            canal_normal_min, canal_normal_mid);
287
288
289     if(canal_2_valor > canal_2_max) canal_2_normal = canal_normal_max;
290     else if(canal_2_valor < canal_2_min) canal_2_normal =
            canal_normal_min;
291     else if(canal_2_valor > canal_2_mid) canal_2_normal = map(
            canal_2_valor, canal_2_mid + 1, canal_2_max, canal_normal_mid +
            1, canal_normal_max);
292     else canal_2_normal = map(canal_2_valor, canal_2_min, canal_2_mid,
            canal_normal_min, canal_normal_mid);
293
294     if(canal_3_valor > canal_3_max) canal_3_normal = canal_normal_max;
295     else if(canal_3_valor < canal_3_min) canal_3_normal =
            canal_normal_min;
296     else canal_3_normal = map(canal_3_valor, canal_3_min, canal_3_max,
            canal_normal_min - 100, canal_normal_max);
297
298     if(canal_4_valor > canal_4_max) canal_4_normal = canal_normal_max;
299     else if(canal_4_valor < canal_4_min) canal_4_normal =
            canal_normal_min;
300     else if(canal_4_valor > canal_4_mid) canal_4_normal = map(
            canal_4_valor, canal_4_mid + 1, canal_4_max, canal_normal_mid +
            1, canal_normal_max);
301     else canal_4_normal = map(canal_4_valor, canal_4_min, canal_4_mid,
            canal_normal_min, canal_normal_mid);
302
303     if(canal_5_valor > canal_5_max) canal_5_normal = canal_normal_max;
```

```
304   else if(canal_5_valor < canal_5_min) canal_5_normal =
          canal_normal_min;
305   else canal_5_normal = map(canal_5_valor, canal_5_min, canal_5_max,
          canal_normal_min, canal_normal_max);
306
307   if(canal_6_valor > canal_6_max) canal_6_normal = canal_normal_max;
308   else if(canal_6_valor < canal_6_min) canal_6_normal =
          canal_normal_min;
309   else canal_6_normal = map(canal_6_valor, canal_6_min, canal_6_max,
          canal_normal_min, canal_normal_max);
310
311   if(comprobarEnvio()){
312     Serial1.write(numero_magico);
313     Serial_SendAnything(&Serial1, carga_util);
314
315     Serial1.write("[");
316     Serial_SendAnything(&Serial1, canal_1_normal);
317     Serial_SendAnything(&Serial1, canal_2_normal);
318     Serial_SendAnything(&Serial1, canal_3_normal);
319     Serial_SendAnything(&Serial1, canal_4_normal);
320     Serial_SendAnything(&Serial1, canal_5_normal);
321     Serial_SendAnything(&Serial1, canal_6_normal);
322     Serial1.write("]");
323   }
324
325   if(debug_normal){
326   //      VALORES NORMALIZADOS
327     Serial.print(canal_1_normal);
328     Serial.print(",");
329     Serial.print(canal_2_normal);
330     Serial.print(",");
331     Serial.print(canal_3_normal);
332     Serial.print(",");
333     Serial.print(canal_4_normal);
334     Serial.print(",");
335     Serial.print(canal_5_normal);
336     Serial.print(",");
337     Serial.print(canal_6_normal);
338     Serial.println();
339   }
340 }
341
342
343 void idleDelay(long idle_delay){
344   long time_delay = millis();
345   while(millis() - time_delay < idle_delay){}
346 }
347
348 void waitUntil(long idle_delay){
349   idle_delay = idle_delay * 1000;
350   while(micros() - tiempo_actual < idle_delay){}
351 }
352
353 bool comprobarEnvio(){
354   return true;
355   if(canal_1_normal < canal_1_ant_normal - canal_margen ||
          canal_1_normal > canal_1_ant_normal + canal_margen) return true;
356   if(canal_2_normal < canal_2_ant_normal - canal_margen ||
          canal_2_normal > canal_2_ant_normal + canal_margen) return true;
```

```
357   if(canal_3_normal < canal_3_ant_normal - canal_margen ||
          canal_3_normal > canal_3_ant_normal + canal_margen) return true;
358   if(canal_4_normal < canal_4_ant_normal - canal_margen ||
          canal_4_normal > canal_4_ant_normal + canal_margen) return true;
359   if(canal_5_normal < canal_5_ant_normal - canal_margen ||
          canal_5_normal > canal_5_ant_normal + canal_margen) return true;
360   if(canal_6_normal < canal_6_ant_normal - canal_margen ||
          canal_6_normal > canal_6_ant_normal + canal_margen) return true;
361   return false;
362 }
363
364 void rutinaInterrupcion0(){
365   tiempo_actual_rutina0 = micros();
366   tiempo_ultima_rutina = tiempo_actual_rutina0;
367
368   if(PIND & (1 << PD0)) tiempo_5 = tiempo_actual_rutina0;
369   else canal_5_valor = tiempo_actual_rutina0 - tiempo_5;
370 }
371
372 void rutinaInterrupcion1(){
373   tiempo_actual_rutina1 = micros();
374   tiempo_ultima_rutina = tiempo_actual_rutina1;
375
376   if(PIND & (1 << PD1)) tiempo_6 = tiempo_actual_rutina1;
377   else canal_6_valor = tiempo_actual_rutina1 - tiempo_6;
378 }
379
380 ISR(PCINT0_vect){
381   tiempo_actual_rutina_hardware = micros();
382   tiempo_ultima_rutina = tiempo_actual_rutina_hardware;
383
384   if(PINB & (1 << PCINT4)){
385     if(canal_1_estado == 0){
386       canal_1_estado = 1;
387       tiempo_1 = tiempo_actual_rutina_hardware;
388     }
389   }
390   else if (canal_1_estado == 1){
391     canal_1_estado = 0;
392     canal_1_valor = tiempo_actual_rutina_hardware - tiempo_1;
393   }
394
395   if(PINB & (1 << PCINT5)){
396     if(canal_2_estado == 0){
397       canal_2_estado = 1;
398       tiempo_2 = tiempo_actual_rutina_hardware;
399     }
400   }
401   else if (canal_2_estado == 1){
402     canal_2_estado = 0;
403     canal_2_valor = tiempo_actual_rutina_hardware - tiempo_2;
404   }
405
406   if(PINB & (1 << PCINT6)){
407     if(canal_3_estado == 0){
408       canal_3_estado = 1;
409       tiempo_3 = tiempo_actual_rutina_hardware;
410     }
411   }
```

```
412   else if (canal_3_estado == 1){
413     canal_3_estado = 0;
414     canal_3_valor = tiempo_actual_rutina_hardware - tiempo_3;
415   }
416
417   if(PINB & (1 << PCINT7)){
418     if(canal_4_estado == 0){
419       canal_4_estado = 1;
420       tiempo_4 = tiempo_actual_rutina_hardware;
421     }
422   }
423   else if (canal_4_estado == 1){
424     canal_4_estado = 0;
425     canal_4_valor = tiempo_actual_rutina_hardware - tiempo_4;
426   }
427 }
```

analog02.cpp

```
1  /*
2   * analog02.c
3   * Author: Jose Simo (2016)
4   * Universitat Politecnica de Valencia. AI2-DISCA
5   * Creative Commons.
6   */
7
8  #include <unistd.h>
9  #include <iostream>
10 #include <fstream>
11 #include <string>
12 #include <sstream>
13
14 #include "util.h"
15
16 using namespace std;
17 using namespace BBB;
18
19 //base path of ADC sysfs.
20 #define LDR_PATH "/sys/bus/iio/devices/iio:device0/in_voltage"
21
22 void setRealTimeCurrentThread()
23 {
24   // If pid equals zero, the scheduling policy and attributes
25   // of the calling thread will be set.
26   int my_pid = 0;
27   struct sched_param param;
28   int retval;
29   int low_priority, high_priority;
30
31   // Get min and max priority values of SCHED_FIFO policy
32   high_priority = sched_get_priority_max(SCHED_FIFO);
33   if (high_priority == -1) {
34     perror("Error in sched_get_priority_max"); _exit(-1);
35   }
36   low_priority = sched_get_priority_min(SCHED_FIFO);
37   if (low_priority == -1) {
38     perror("Error in sched_get_priority_min"); _exit(-1);
39   }
40
```

```
41    // Change to SCHED_FIFO and medium priority.
42    param.sched_priority = 50; //(high_priority + low_priority) / 2;
43    retval = sched_setscheduler(my_pid, SCHED_FIFO, &param);
44    if (retval == -1) {
45      perror("Error in sched_setscheduler"); _exit(-1);
46    }
47  }
48  /**
49   *
50   */
51  void openAnalog(fstream &fs, int number){
52     stringstream ss;
53     ss << LDR_PATH << number << "_raw";
54     fs.open(ss.str().c_str(), fstream::in);
55  }
56  /**
57   *
58   */
59  int readAnalog(fstream &fs) {
60    int sample;
61      fs >> sample;
62      fs.seekg(0);
63      return sample;
64  }
65  /**
66   *
67   */
68  void closeAnalog(fstream &fs){
69      fs.close();
70  }
71
72  /**
73   *
74   */
75  int main() {
76      cout << "Start ADC sampling..." << endl;
77
78      setRealTimeCurrentThread();
79
80      unsigned long data_buffer_t0[10000];
81      unsigned long data_buffer_t1[10000];
82      int data_buffer_value[10000];
83
84      fstream adc_channel0;
85      openAnalog(adc_channel0, 0); //Channel 0 pin P9.39
86
87      struct timespec ts;
88      clock_gettime(CLOCK_MONOTONIC, &ts);
89
90      //Sample at 1kHz (10 seconds)
91    for (int i=0; i<10000; i++) {
92      unsigned long  t0 = getCurrentMicroseconds();
93      int value = readAnalog(adc_channel0);
94      unsigned long  t1 = getCurrentMicroseconds();
95      data_buffer_t0[i] = t0;
96      data_buffer_t1[i] = t1;
97      data_buffer_value[i] = value;
98      //
99      sleep_until(&ts, 1000);
```

```
100     }
101
102     //Dump data to file
103       ofstream datafile;
104       datafile.open ("data03.txt");
105
106     for (int i=0; i<10000; i++) {
107       datafile << data_buffer_t0[i] << " " << data_buffer_t1[i] << " " <<
              data_buffer_value[i] << endl;
108     }
109     datafile.close();
110
111     cout << "ADC sampling finished" << endl;
112     return 0;
113 }
```

APÉNDICE B
Simulador

Como trabajos previos, realizamos un control en posición en un simulador para preparar este proyecto para sentar las bases de conocimiento necesarias para llevarlo a cabo. Aquí dejamos una muestra del trabajo llevado a cabo con una imagen de una simulación con MATLAB y el código de control.

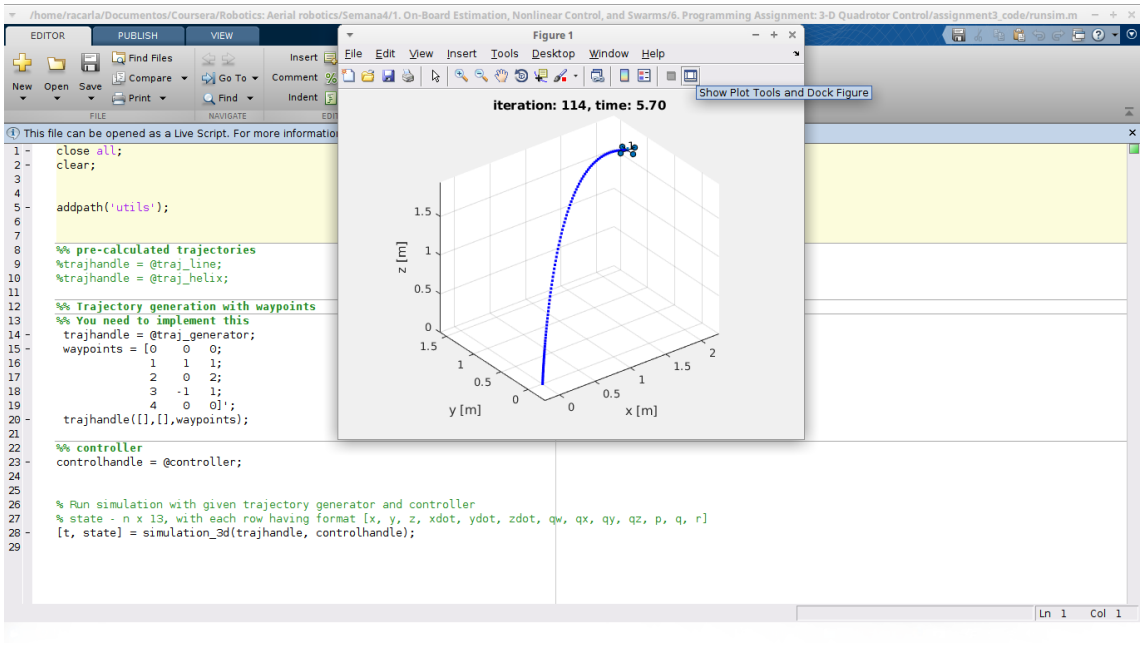


Figura B.1: Simulación de un quadrotor.

```
function [F, M] = controller(t, state, des_state, params)
%CONTROLLER  Controller for the quadrotor
%
%   state: The current state of the robot with the following fields:
%   state.pos = [x; y; z], state.vel = [x_dot; y_dot; z_dot],
%   state.rot = [phi; theta; psi], state.omega = [p; q; r]
%
%   des_state: The desired states are:
%   des_state.pos = [x; y; z], des_state.vel = [x_dot; y_dot; z_dot],
%   des_state.acc = [x_ddot; y_ddot; z_ddot], des_state.yaw,
%   des_state.yawdot
%
%   params: robot parameters

```

```
15 %   Using these current and desired states, you have to compute the
       desired
16 %   controls
17
18 % Thrust
19 F = 0;
20
21 % Moment
22 M = zeros(3,1);
23
24 % Constants:
25 I = params.I
26 g = params.gravity;
27 m = params.mass;
28
29 % Parameters:
30 % control
31   % axys
32     % proporcinal
33     Kp_1 = 22; % 12 -18
34     Kp_2 = 22; % 12 -18
35     Kp_3 = 4;
36     % derivative
37     Kd_1 = 0.05;
38     Kd_2 = 0.05;
39     Kd_3 = 450;
40   % angles
41     % proporcinal
42     Kp_phi   = 100 * Kp_1;
43     Kp_theta = 100 * Kp_2;
44     Kp_psi   = 10 * Kp_3;
45     % derivative
46     Kd_phi   = 0;
47     Kd_theta = 0;
48     Kd_psi   = 70;
49 % axys
50 r1 = state.pos(1);
51 r2 = state.pos(2);
52 r3 = state.pos(3);
53 r1_dot = state.vel(1);
54 r2_dot = state.vel(2);
55 r3_dot = state.vel(3);
56
57 r1_T = des_state.pos(1);
58 r2_T = des_state.pos(2);
59 r3_T = des_state.pos(3);
60 r1_dot_T = des_state.vel(1);
61 r2_dot_T = des_state.vel(2);
62 r3_dot_T = des_state.vel(3);
63 r1_ddot_T = des_state.acc(1);
64 r2_ddot_T = des_state.acc(2);
65 r3_ddot_T = des_state.acc(3);
66 % angles
67 phi   = state.rot(1);
68 theta = state.rot(2);
69 psi   = state.rot(3);
70 p     = state.omega(1);
71 q   = state.omega(2);
72 r     = state.omega(3);
```

```
73 % desired angles - omega
74 p_des = 0;
75 q_des = 0;
76 r_des = des_state.yawdot;
77
78
79 % Desired accels
80
81 r1_ddot_des = r1_ddot_T + Kd_1 * (r1_dot_T - r1_dot) + Kp_1 * (r1_T -
       r1);
82
83 r2_ddot_des = r2_ddot_T + Kd_2 * (r2_dot_T - r2_dot) + Kp_2 * (r2_T -
       r2);
84
85 r3_ddot_des = r3_ddot_T + Kd_3 * (r3_dot_T - r3_dot) + Kp_3 * (r3_T -
       r3);
86
87 % Desired angles
88
89   % needed variables
90   psi_T = des_state.yaw;
91
92 phi_des = (1/g) * ((r1_ddot_des * sin(psi_T)) - (r2_ddot_des * cos(
       psi_T)));
93
94 theta_des = (1/g) * ((r1_ddot_des * cos(psi_T)) + (r2_ddot_des * sin(
       psi_T)));
95
96 psi_des = des_state.yaw;
97
98
99 % u2
100
101 u2 = [
102   Kp_phi   * (phi_des   - phi)   + Kd_phi   * (p_des - p);
103   Kp_theta * (theta_des - theta) + Kd_theta * (q_des - q);
104   Kp_psi   * (psi_des   - psi)   + Kd_psi   * (r_des - r)
105 ];
106
107 % Thrust and moment
108
109 F = m * (g + r3_ddot_des);
110
111 M = I * u2;
112
113 end
```