Universidad
Carlos III de Madrid

TRABAJO FIN DE GRADO

Técnicas de control de vuelo sobre un cuadricóptero

García Muñoz, Francisco Javier

Director:	*Jesús García Herrero*
Tutor:	*Abdulla Hussein Abdulrahman Al-Kaff*
Titulación:	*Ingeniería Electrónica Industrial y Automática*
Universidad:	*Universidad Carlos III de Madrid* *Escuela Politécnica Superior* *Avda. de la Universidad, 30 28911 Leganés, España*

Febrero de 2015

Quiero dedicar el presente proyecto a mis padres, tíos y hermano por el apoyo, fuerza y aguante aportado a lo largo de toda la carrera.

También quiero agradecer la ayuda recibida por parte de mis dos profesores Jesús García y Abdulla Hussein, y de la Universidad Carlos III de Madrid en general.

ÍNDICE GENERAL:

ÍNDICE DE FIGURAS:

ÍNDICE DE TABLAS:

Capítulo 1: Introducción

Un sistema cuadricóptero o quadcopter es un derivado de los helicópteros, que consta de 4 rotores para poder despegar y realizar determinados movimientos. También hay otro tipo de multicópteros que tienen tres (tricóptero) o más de cuatro rotores, aunque nos vamos a centrar en el cuadricóptero con el que vamos a trabajar. También son conocidos como drones ("abejorros" en inglés). Nos podemos encontrar con dos tipos de cuadricópteros dependiendo de la configuración de sus ejes:

- Cuadricóptero en configuración aspa o equis(X): en esta configuración los motores están situados hacia Noroeste, Noreste, Sureste y Suroeste según la enumeración de la figura. Es el sistema con el que vamos a trabajar a lo largo del proyecto.

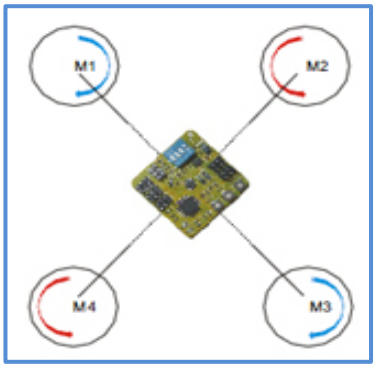


Figura 1. Cuadricóptero configuración X

- Cuadricóptero en configuración cruz (+): en esta configuración los motores están situados hacia Norte, Este, Oeste y Sur, según la enumeración de la siguiente figura. Esta configuración es más inestable que la anterior, pero también es posible realizar vuelos con éxito.

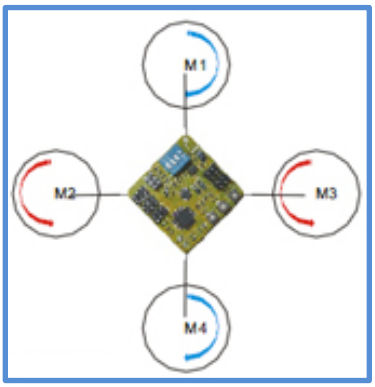


Figura 2. Cuadricóptero configuración +

Para poder controlar el cuadricóptero y evitar caídas, hay que conseguir que cada par de hélices giren en el mismo sentido. El principal problema de estos es calcular la velocidad en tiempo real, para lograr un movimiento estable.

1.1 Presentación

1.1.1 Descripción general del proyecto

El proyecto descrito consiste en que el cuadricóptero sea capaz de seguir una trayectoria de manera autónoma calculando previamente un algoritmo. Para ello habrá que analizar todas las alternativas posibles en cada momento del vuelo.

1.2 Motivación personal

La idea de este proyecto surgió consultando los diferentes trabajos disponibles en el tablón de la Universidad, y queriendo trabajar en diferentes áreas de lo aprendido a lo largo de la carrera me decanté por este ya que vi que tendría que programar en lenguaje C# muy similar a los vistos en la carrera(C y C++), realizar un sistema de control y aplicar diferentes temas de Física para poder realizar el proyecto de manera exitosa.

También me decidí por este proyecto porque es un tema que está actualmente en plena expansión y me forzaría a aprender, además de ser un trabajo bastante práctico y dinámico con los vuelos realizados.

1.3 Objetivos

El presente proyecto tiene por objetivo calcular dos algoritmos en los que el cuadricóptero AR Drone 2.0 sea capaz de realizar un movimiento autónomo a través de una serie de puntos (X, Y) que se le introducirán al programa antes de despegar, y una vez realizado el movimiento vuelva a aterrizar. El primer algoritmo realizará los movimientos inclinando la parte delantera del drone, mientras que el segundo realizará un movimiento de inclinación frontal y lateral dependiendo de las condiciones del punto de destino. El control será en lazo abierto.

1.3.1 Objetivos del proyecto

- Lograr estabilizar el vuelo del cuadricóptero con el control manual.
- Conseguir que el drone realice un vuelo autónomo con la aplicación LSIDrone en el programa Microsoft Visual Studio 2013 con el lenguaje de programación C#.
- Calcular los algoritmos necesarios y pedidos para realizar un control autónomo.
- Obtener los diferentes errores al realizar las trayectorias una vez desarrollado el software.

1.3.2 Objetivos personales

- Aprender diferentes temas relacionados con RC (control remoto) y aeronaves.
- Profundizar en los diferentes conocimientos de Programación de los que dispongo.
- Aprender a programar en C# con Microsoft Visual Studio 2013.
- Resolver diferentes problemas de Física y Mecánica de manera práctica.
- Manejo básico de archivos .DLL (Dinamic Link Library).

1.4 Contenido de la memoria

En este primer capítulo se ha explicado de manera sencilla y con conceptos básicos qué es un sistema cuadricóptero y sus distintas configuraciones dependiendo de la posición de los ejes, también se han expuesto las motivaciones y objetivos que nos han llevado a realizar el presente proyecto con ganas y esfuerzo.

En el capítulo 2 de esta memoria entramos más a fondo en comprender qué es un UAV y conocer la historia del cuadricóptero con el que vamos a trabajar el AR Drone 2.0, el cual será descompuesto completamente por partes y especificado todo en el capítulo 3.

El desarrollo del software, que es lo más importante del presente trabajo vendrá desarrollado en el capítulo 4 desde el punto de partida en que empezamos hasta el trabajo final donde se mostrarán los algoritmos calculados, las distintas configuraciones y modos de vuelo posibles y la modificación de la interfaz. Posteriormente se incluye la mayoría de código utilizado en la programación de nuestro software.

Una vez realizado el trabajo final tendremos que medir los errores y sacar conclusiones de las trayectorias seguidas, lo cual viene detallado en el capítulo 5 donde se analizan todos los resultados.

En el capítulo 6 se incluyen las conclusiones y trabajos futuros relacionados con el presente proyecto.

El estudio del presupuesto y la organización del trabajo se exponen en el capítulo 7 con su respectivo diagrama de Gantt explicativo.

Por último tendremos diferentes anexos donde se detallan las referencias bibliográficas consultadas (apéndice A), diferentes manuales útiles para hacer funcionar y comprender el programa (apéndice B) y un glosario con diferentes términos utilizados en la descripción del presente proyecto (apéndice C).

Capítulo 2: Estado del arte

En este capítulo se va a explicar de forma clara y concisa los diferentes tipos y aplicaciones de los vehículos aéreos no tripulados. También vamos a conocer más a fondo la historia sobre el cuadricóptero con el que vamos a trabajar a lo largo del proyecto, el AR Drone 2.0 de la conocida marca francesa Parrot.

2.1 Vehículos aéreos no tripulados

Los vehículos aéreos no tripulados son también conocidos como VANTs en español, o en inglés UAVs (Unmanned Aerial Vehicle), y son básicamente aviones que no requieren de un piloto humano a bordo. Estos son controlados manualmente a través de un mando por un piloto en tierra, o de forma autónoma con su correspondiente programación gracias a la electrónica de la que dispone.

2.1.1 Historia

Las primeras ideas de vehículos aéreos no tripulados surgieron hace muchos años alrededor de 1916 al entrar Estados Unidos en la Primera Guerra Mundial. Durante estos años se encargó al ingeniero e inventor Charles Franklin Kettering (1876-1858) de construir un “torpedo volador”, para atacar las defensas alemanas en el frente occidental, y por otro lado Elmer Ambrose Sperry (1860-1930) desarrolló un aparato similar al anterior para la Marina de los Estados Unidos, con el fin de combatir la amenaza submarina germana. Finalmente ninguno de estos dos aparatos se utilizó en la Guerra y quedaron en simples diseños e ideas para el futuro.

Sin embargo a lo largo de la Segunda Guerra Mundial, los norteamericanos desarrollaron el “Proyecto Afrodita”, con el fin de utilizar bombarderos con sistema de piloto automático a control remoto, para intentar destruir las bombas voladoras alemanas. El ingeniero norteamericano Clarence Leonard Johnson (1910-1990) predijo que el futuro pertenecía a los aviones sin piloto a bordo.

En la década de los años cincuenta la empresa Ryan Aeronautical Co comenzó a desarrollar los Firebees, una familia de UAVs utilizada durante la guerra de Vietnam para reconocimiento aéreo y espionaje principalmente. La Marina de los Estados Unidos realizó un proyecto de helicópteros autónomos, que fracasó debido a problemas electrónicos y de control. Israel a

finales de la década de los sesenta continuó con el proyecto de la Marina de los EEUU con resultados muy notables, con lo que Estados Unidos volvió a interesarse nuevamente en diseñar y construir aviones aéreos no tripulados.

En 1991 estos vehículos pasaron a la historia ya que desarrollaron su misión de forma exitosa, dado que un UAV lanzado desde Wisconsin observó a tropas iraquíes en la isla de Failaka (Kuwait) en el Golfo Pérsico, siendo estas tropas atacadas, lo que supuso la rendición de los iraquíes al ver aproximarse dicho avión no tripulado.

En los conflictos bélicos de Bosnia-Herzegovina y Kosovo en la década de los noventa hubo un punto de inflexión en el desarrollo y diseño de UAVs, ya que se mostró un gran interés por parte del director de la Agencia Central de Inteligencia (CIA) de los Estados Unidos James Woolsey, y el comandante de las Fuerzas Aéreas Aliadas y de los Estados Unidos en Europa, el general John P. Humper. El resultado de esto fue el UAV Gnat 750, evolucionando al Predator MQ-1, y posteriormente en el MQ-9 Reaper, siendo este último el primer vehículo aéreo no tripulado capaz de realizar vigilancias de larga duración y de gran altitud. En esta época otros países como Reino Unido, Francia y Alemania desarrollaron sus propios vehículos aéreos.

En la primera década del año 2000, los vehículos aéreos no tripulados fue cuando alcanzaron su mayor protagonismo. El evento que captó mayor atención fue en agosto de 2008, cuando en un conflicto entre Georgia y Rusia, un caza ruso derribó un UAV georgiano (fabricado por israelíes). En Iraq, el Ejercito de los Estados Unidos despliega sus UAVs para detectar explosivos, y a Estados Unidos le ha surgido una gran preocupación ya que los iraquíes han logrado interferir la transmisión de video de sus UAVs.

La década en la que nos encontramos actualmente tiene por objetivo conseguir crear un vehículo aéreo no tripulado totalmente autónomo, es decir reemplazar por completo la intervención del operador. En la actualidad destaca el diseño y la construcción de un Black Hawk sin piloto para el año 2015, un UAV adaptado a una plataforma aérea veinte veces más pesada.

Resumiendo las primeras aplicaciones de UAVs fueron militares, y después con el gran desarrollo de la electrónica se han creado elementos utilizados para ocio, o incluso para realizar tareas de mensajería, donde algunas empresas tienen problemas de legalidad, ya que el espacio aéreo regulado exige unos estándares de control y seguridad muy exigentes, y ni Estados Unidos ni Europa permiten el vuelo de aviones totalmente autónomos y UAVS, en general. Los drones militares también son conocidos como UCAV (Unmanned Combat Aerial Vehicle).

2.1.2 Clasificación

Para lograr clasificar de forma entendible los UAVs, podemos hacerlo atendiendo a diez tipos de configuraciones o usos, siendo la siguiente:

- Por tipo de misión: podemos encontrarnos con distintos vehículos dependiendo de la misión para la que han sido creados:
  I. Reconocimiento u observación.
  II. Investigación.
  III. Salvamento.
  IV. Anti-incendios.
  V. Combate.
  VI. Transporte.
  VII. Blancos aéreos.
- Por origen de misión:
  I. Civil.
  II. Militar.
- Por tamaño: dependiendo del tamaño del drone o UAV podemos clasificarlo en cuatro tamaños:
  I. Grandes.
  II. Medianos.
  III. Pequeños.
  IV. Micro UAV.
- Por la forma de obtener la sustentación:
  I. Más pesados que el aire.
  II. Más ligeros que el aire.
  III. Híbridos
- Por la forma de despegue:
  I. Desde una pista.
  II. Lanzados a mano.
  III. Lanzados con medios mecánicos.
- Por su motor:
  I. Alternativo.
  II. Turbinas.
  III. Eléctrico.
- Por el origen del diseño:
  I. Desde cero.
  II. UAV modificado.
- Por la duración de la misión:
  I. Larga duración.
  II. Media duración.

III. Corta duración.

- Por cota de vuelo:
  I. Alta o muy alta cota.
  II. Media cota.
  III. Baja cota.
- Por el tipo de control.
  I. Autónomo y adaptativo.
  II. Monitorizado.
  III. Supervisado.
  IV. Autónomo no adaptativo o pre-programado.
  V. Dirigido por un operador.

Cabe destacar que hay diferentes tamaños y que dependiendo de él, la autonomía y agilidad del UAV varía. A mayor tamaño tendremos más autonomía de vuelo dado que las hélices son más grandes por lo que se consigue mantener con una rotación más lenta y por lo tanto con una mayor eficiencia, además constan de baterías con más capacidad. Sin embargo si son de menor tamaño la autonomía es menor, pero su agilidad al volar es mucho mayor.

A continuación se muestran algunos ejemplos de UAVs con diferentes tipos de uso:

i. ***Boeing J-UCAV X-45 A***, es un vehículo de combate, perteneciente a los Estados Unidos.
ii. ***Drone FOX-C8-HD AltiGator***, es un vehículo cuyo origen de misión es civil. Su origen es francés.
iii. ***Micro-UAV Mosquito***, es uno de los vehículos más pequeños de todos los clasificados. Pertenece a la Industria Aeroespacial israelí.
iv. ***NASA Helios***, es un UAV de motor eléctrico desarrollado por la NASA.
v. ***Ion Tiger UAV***, es un vehículo de origen de diseño "delicado" propulsado con hidrógeno líquido capaz de volar hasta 24 horas.
vi. ***V-Star: UAV VTOL***, es un UAV hibrido que se caracteriza por alimentarse gracias a unas turbinas de gas, lo que hace que sea mucho más eficiente. El nombre VTOL (Vertical and Take-Off Landing) se debe a que es un vehículo que despega y aterriza verticalmente.
vii. ***Diamond DA42***, es un UAV que tiene el tamaño de una avioneta convencional y su despegue se realiza desde una pista. Puede transportar hasta 2000 kg. , volar a casi 10 km. de altura y es capaz de alcanzar una velocidad de 351 km/h.
viii. ***RQ-11 Raven***, es un UAV que podemos clasificar por cota de vuelo muy baja. Son lanzados a mano, y su uso es principalmente militar.

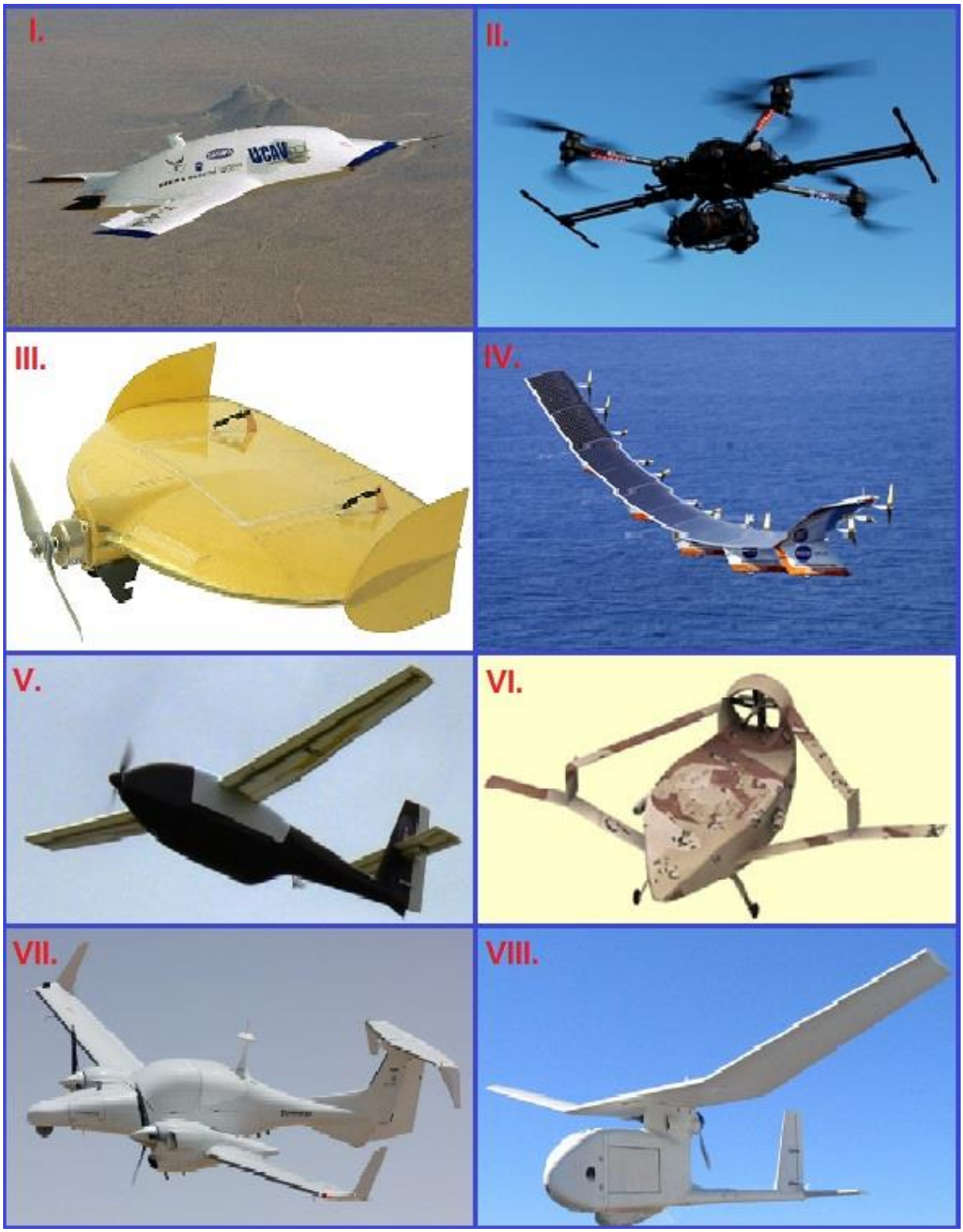


Figura 3. Ejemplo de diferentes UAVs

2.2 Cuadricóptero Parrot AR Drone:

2.2.1 Historia

El primer cuadricóptero de la empresa francesa Parrot fue lanzado en 2010 en Las Vegas en la Feria Internacional de Consumidores Electrónicos, con el nombre de AR Drone. Se podía pilotar desde dispositivos con iOS o Android. Tras liberar la empresa los applets de control de código abierto aparecieron diferentes aplicaciones para ser controlado con PC y demás.

Dos años después en 2012, Parrot anunció la segunda versión, el AR Drone 2.0 con notables ventajas respecto al anterior. Esta última versión, que es la tenemos en la actualidad, es capaz de grabar en alta definición a 720 píxeles, y cuenta con la opción de compartir videos directamente. También se mejoró la resistencia a impactos, el control y estabilidad del drone gracias a los sensores de ultrasonido y magnetómetro 3D, siendo capaz de realizar flips con el control absoluto del que dispone.

La siguiente versión de este cuadricóptero se desconoce cuándo será lanzada, aunque se prevé que la empresa Parrot no tardará demasiado y pondrá a la venta el esperado Bebop Drone a comienzos del año 2015 con muchas mejoras y mayores prestaciones que sus predecesores.

2.2.2 Modelos

Desde el lanzamiento del AR Drone 2.0 en 2012, han surgido mejoras como aumento de la capacidad de la batería a 1500 mAh con respecto a los 1000 mAh iniciales, y se ha creado un GPS Flight Recorder, conocido como “caja negra”, capaz de grabar el vuelo y las coordenadas por las que ha pasado el drone. También se le han cambiado los colores de hélices y carcasas para distintas misiones y gustos de cada piloto.

Los distintos modelos de AR Drone 2.0 con los que contamos en la actualidad, siendo las características principales idénticas, son:

- AR Drone 2.0 Classic.
- AR Drone 2.0 Elite Edition.
- AR Drone 2.0 GPS Edition.
- AR Drone 2.0 Power Edition.

Las características tanto mecánicas como electrónicas vendrán explicadas detalladamente en el capítulo 3, donde se explicará a fondo todos los elementos de nuestro UAV.

Figura 4. Cuadricóptero AR Drone 2.0

2.3 Modelo dinámico del cuadricóptero AR Drone 2.0

El modelo dinámico de nuestro cuadricoptero lo mostramos en la figura 5. Con el controlador inercial que lleva equipado podemos conocer diferentes parámetros y tener una visión global del sistema, teniendo en cuenta que modelaremos y programaremos nuestro cuadricóptero sabiendo que los ángulos de pitch, roll, yaw y gaz serán entradas en nuestro controlador, mientras que los ángulos reales obtenidos y las velocidades en los distintos ejes serán las salidas. Nuestras entradas, dependiendo de los puntos de la trayectoria, llegarán a la función Navigate (Controlroll, Controlpitch, Controlyaw, Controlgaz) y comenzará el movimiento obteniendo la salida que podremos observar en nuestra interfaz.

Gracias a este modelo podemos programar diferentes comportamientos autónomos modelando cada uno de los controles por separado o conjuntamente, que en nuestro caso será implementado con un sistema de control en lazo abierto. El análisis completo de este modelado vendrá detallado a partir del apartado 4.4 en el capítulo 4 de la presente memoria.

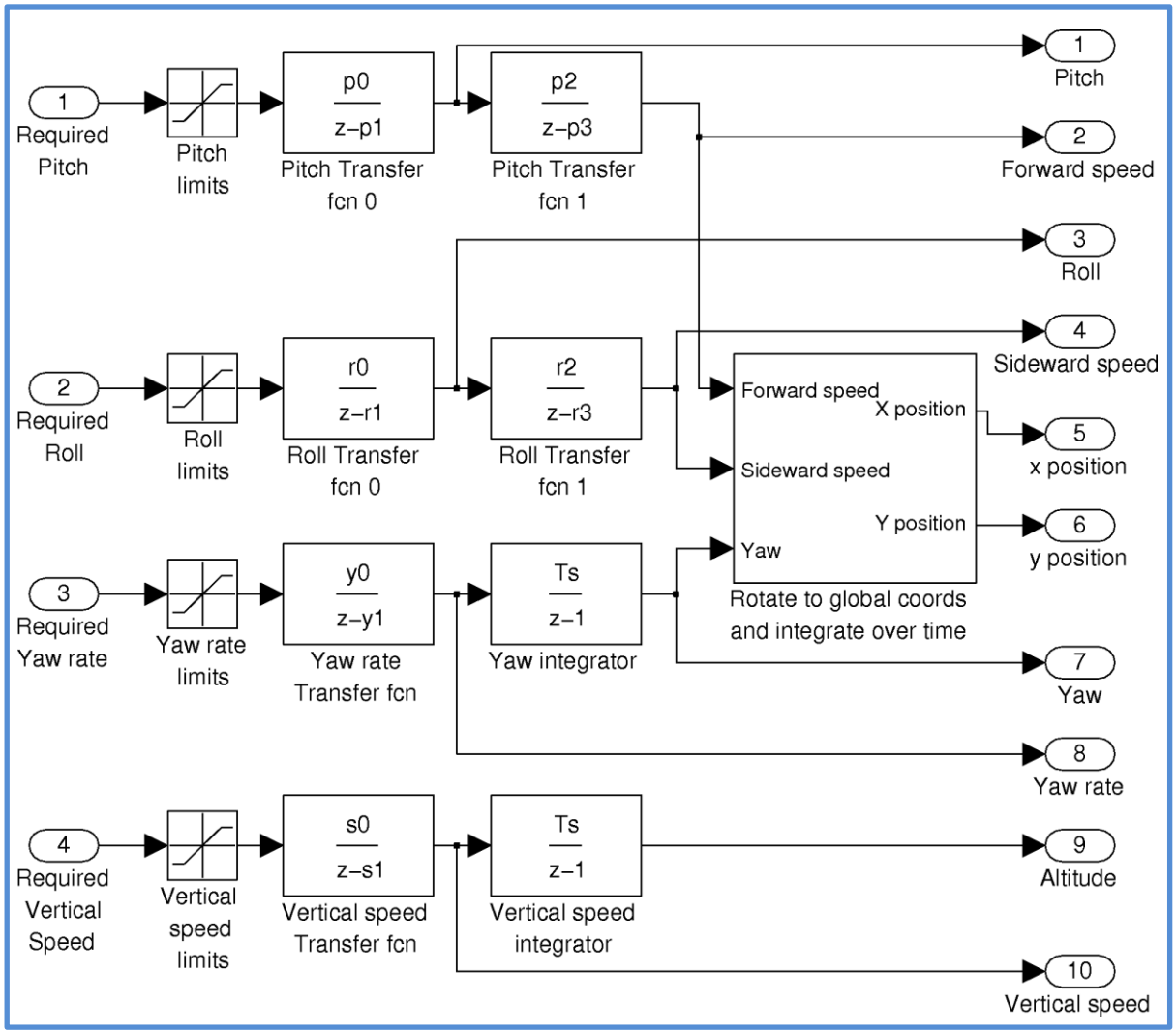


Figura 5. Modelo dinámico de nuestro cuadricóptero

2.4 Normativa drones en España

La Agencia Estatal de Seguridad Aérea (AESA) es la entidad responsable de controlar que el uso de aeronaves tripuladas por control remoto en España se realice en el ámbito de la ley y la seguridad. El uso de este tipo de aparatos es reciente y por ello, con el objetivo de evitar mal entendidos y posibles incidentes, AESA quiere aclarar en qué circunstancias y condiciones se pueden usar los drones y en cuáles no, y qué consecuencias tiene hacerlo en este último caso. Véase [16].

2.4.1 ¿Se pueden usar drones en España? ¿Qué se puede hacer con un dron con la actual regulación en España?

En España no está permitido el uso de drones para aplicaciones civiles (para uso militar existe una normativa que permite su operación exclusivamente en espacio aéreo segregado). Es decir, no está permitido, y nunca lo ha estado, el uso de aeronaves pilotadas por control remoto con fines comerciales o profesionales (salvo en recintos completamente cerrados, incluyendo techo).

Actualmente se pueden utilizar drones para realización de trabajos aéreos como son:

- actividades de investigación y desarrollo.
- tratamientos aéreos, fitosanitarios y otros que supongan esparcir sustancias en el suelo o la atmósfera, incluyendo actividades de lanzamiento de productos para extinción de incendios.
- levantamientos aéreos.
- observación y vigilancia aérea incluyendo filmación y actividades de vigilancia de incendios forestales.
- publicidad aérea, emisiones de radio y TV.
- operaciones de emergencia, búsqueda y salvamento.
- y otro tipo de trabajos especiales no incluidos en la lista anterior.

Aunque en un primer momento, y hasta que no esté aprobada la reglamentación definitiva, las operaciones que se pueden realizar se limitan a zonas no pobladas y al espacio aéreo habilitado.

2.4.2 El uso de drones por particulares para fines deportivos o de recreo

La actividad del aeromodelismo la regula la Real Federación Aeronáutica de España y además, cada Comunidad Autónoma y cada Municipio puede tener su regulación propia sobre esta práctica deportiva o lúdica, aunque siempre deben respetar la legislación aeronáutica general.

Los aeromodelos vuelan por debajo de los 100 metros de altura y no pueden volar sobre núcleos urbanos ni sobre grupos de población (playas, conciertos, las calles de cualquier ciudad, etc…). Deben volar en zonas habilitadas para ello. Lo contrario, puede suponer sanciones y se debe denunciar.

Por tanto los particulares que posean un drone del tipo que sea, sólo podrán usarlo en las zonas habilitadas para ello conforme a la normativa que regula las actividades de aeromodelismo. Deben consultar la normativa de su municipio o comunidad autónoma. En ningún caso podrán utilizarlos para una actividad profesional o con carácter comercial.

2.4.3 Vuelo de drones en recintos privados

Los recintos completamente cerrados (un pabellón industrial o deportivo, un centro de convenciones, un domicilio particular) no están sujetos a la jurisdicción de AESA, al no formar parte del espacio aéreo. Los titulares de esos recintos pueden decidir si autorizan el vuelo de drones en su interior y en qué condiciones. Un estadio de fútbol no tiene la consideración de recinto cerrado, a menos que su cubierta cubra la totalidad de su superficie.

Cabe destacar que en el tema de sanciones, no hay una cuantía predeterminada sino que dicha sanción ira en proporción al daño causado o riesgo incurrido. También en caso de daños a terceras personas se puede sancionar por vía penal o civil, como en cualquier otra actividad.

2.4.4 Requisitos para poder grabar exteriores

La nueva normativa permite para una aeronave de hasta 25 Kg, grabar en exteriores, pero ha de hacerse de día y en condiciones meteorológicas visuales, en zonas fuera de aglomeraciones de edificios en ciudades, pueblos o lugares habitados o de reuniones de personas al aire libre, en espacio aéreo no controlado, dentro del alcance visual del piloto, a una distancia de éste no mayor de 500 m. y a una altura sobre el terreno no mayor de 400 pies (es decir, como máximo 120 m. sobre el terreno).

Para conseguir la habilitación como operador de drones para realizar este tipo de trabajos el régimen establecido es de comunicación previa y declaración responsable, por lo que no es necesario un permiso o autorización, tan solo un acuse de recibo una vez que presente en el Registro de AESA la declaración responsable junto con la documentación exigida, cuyo acuse de recibo le habilitará como operador de drones.

Independientemente de estar habilitado como operador de drones es necesario recordar que para la realización de fotografías o filmaciones con cualquier tipo de aeronaves, tripuladas o no, es necesario obtener una autorización específica de AESA, para ese tipo de actividad, en virtud de una Orden de Presidencia del Gobierno de 14 de Marzo de 1957.

2.4.5 Requisitos de un drone para volar legalmente

Todos los drones, sin excepción, deben de llevar fijada en su estructura una placa de identificación en la que deberá constar, de forma legible y a simple vista, la identificación de la aeronave, mediante la designación específica, número de serie si es el caso, nombre de la empresa operadora y los datos para contactar con la misma.

Además, los que pesen más de 25kg deben estar inscritos en el Registro de Matrícula de Aeronaves de AESA y disponer de un certificado de aeronavegabilidad. Los que pesen menos, no tendrán que cumplir estos dos requisitos.

La placa es responsabilidad exclusivamente del operador (adquirirla o elaborarla, grabar o hacer grabar la información requerida y fijarla a la aeronave) y en ella deberá constar "de forma legible a simple vista e indeleble" la información que se especifica. Por tanto, el tamaño de la placa ha de ser el necesario para que incluya toda la información requerida y ésta pueda leerse a simple vista.

Capítulo 3: Descripción general

En este capítulo vamos explicar a fondo todos los elementos y características de los que consta el cuadricóptero Parrot con el que hemos trabajado, como la interfaz utilizada y la comunicación entre el drone y el PC. También se plantearán los principales movimientos que se han estudiado del cuadricóptero para poder entender mejor el funcionamiento de nuestro drone.

3.1 Especificaciones técnicas Parrot AR Drone 2.0

A continuación se van a mostrar todos los elementos internos y externos que componen nuestro cuadricóptero de la marca francesa Parrot. Vamos a descomponerlo para ver los elementos con los que trabajar y las opciones de las que se disponen antes de empezar a programar nuestro software.

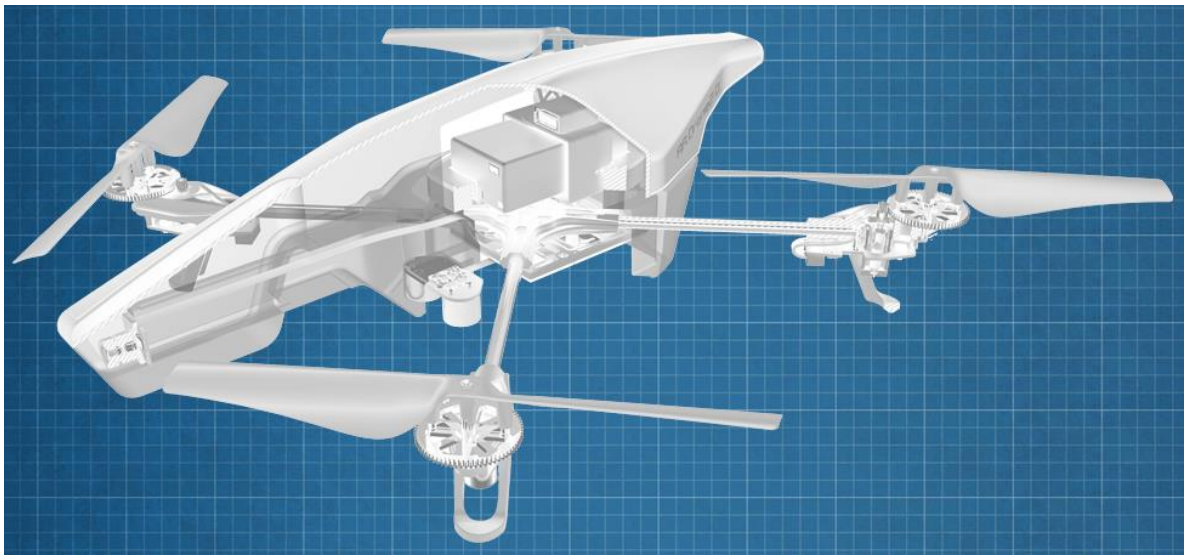


Figura 6. Estructura interna del AR Drone 2.0

3.1.1 Controlador

Las características principales de la placa controladora de nuestro cuadricóptero son:

- Arquitectura: AR M Cortex A8.
- Procesador: OMAP 3630.
- Velocidad del procesador: 1GHz.
- Memoria RAM: DDR de 128 MB a 200 MHz.
- Memoria: entrada para pendrive o GPS Flight Recorder.
- Sistema operativo: Windows.

3.1.2 Sensores

La placa de navegación incluye los sensores para la estabilización del AR Drone 2.0:

- Sensores de ultrasonidos para medir los cambios de altitud.
- Sensor de altitud para correcciones.
- Acelerómetro digital de 3 ejes para monitorizar los movimientos de posición.
- Giróscopo de 2 ejes y giróscopo preciso piezoeléctrico para medir el giro y cabeceo.
- Magnetómetro de 3 ejes para la orientación.

Estos elementos combinados permiten estabilizar el aparato.

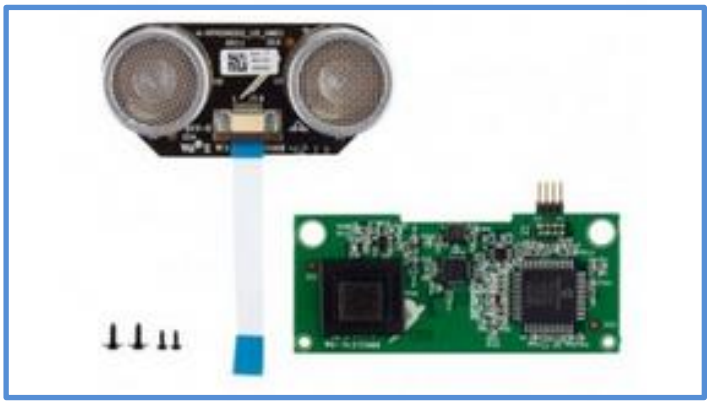


Figura 7. Sensores del AR Drone 2.0

3.1.3 Motores

Los motores de los que dispone nuestro cuadricóptero son motores de rotor interno ("inrunner") sin escobillas, y estos motores se caracterizan por:

- Potencia: 15 W.
- Revoluciones por minuto: 28000 rpm, en vuelo estático
- Velocidad correspondiente en la hélice: 3300 rpm.
- Rango de velocidades: 10.350 y 41.400 rpm.
- Velocidad de desplazamiento: 5 m/s; 18 km/h.
- Alimentación de cada motor: 5 V

El motor está conectado a su controlador electrónico, diseñado para el AR Drone 2.0. Un microcontrolador de baja potencia de 8 bits y un convertidor analógico-digital (CAD) de 10 bits controlan la velocidad del motor.

Figura 8. Motores del AR Drone 2.0

3.1.4 Cámaras

El AR Drone 2.0 lleva integradas dos cámaras para poder observar por donde estamos volando nuestro cuadricóptero. También nos permiten realizar fotografías o grabar en video el vuelo realizado, ya que cuenta con un procesador de video DSP a 800 MHz el cual permite codificar el video en alta definición HD (High Definition) a alta velocidad. Las cámaras son las siguientes:

- Cámara frontal HD: ésta es la encargada de mostrar las imágenes en alta definición a 720 pixeles a 30 fps (frames por segundo), además tiene un objetivo angular de 92º en diagonal.

Figura 9. Cámara frontal del AR Drone 2.0

- Cámara vertical QVGA: esta cámara tiene peor definición que la anterior, pero suficiente para la misión que tiene que cumplir. Es la encargada de medir la velocidad del drone respecto al suelo, y lo hace a 60 fps.

Figura 10. Cámara vertical del AR Drone 2.0

3.1.5 Baterías

La batería de alta calidad es de polímero de litio (LiPo) con 3 celdas. Incluye un módulo de circuito de protección (PCM) que protege la batería frente a sobrecarga, sobredescarga y cortocircuitos. La batería está protegida con una carcasa rígida y cumple con la normativa de seguridad UL2054. Posee dos conectores: un conector de descarga para alimentar al Ar Drone 2.0 y otro conector de carga.

Podemos encontrarnos según la capacidad dos tipos de baterías originales, la convencional de 1000 mAh (12 minutos de vuelo aproximadamente) y la de alta densidad de 1500 mAh (18 minutos de vuelo).

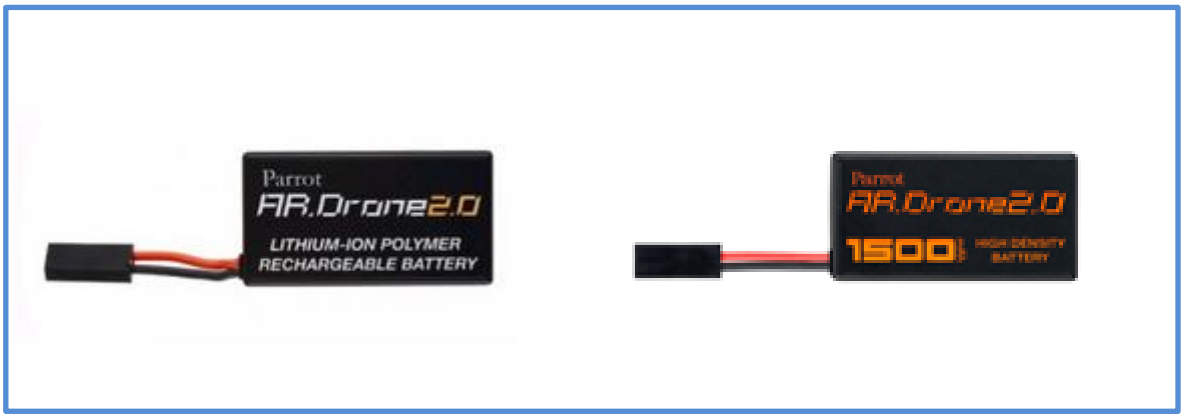


Figura 11. Baterías del AR Drone 2.0

3.1.6 Comunicaciones

La comunicación entre el cuadricóptero y el portátil se realiza a través de una red inalámbrica Wi-Fi, donde el controlador de a bordo del drone es el servidor y el PC portátil es el cliente de la comunicación. Cualquier computadora con sistema operativo Windows y tarjeta de red inalámbrica puede utilizarse. Para este proyecto se ha utilizado un portátil Hp Pavilion Dv5, con Windows 8.

La tecnología Wi-Fi de nuestro cuadricóptero es 802.11 b/g/n, la cual permite desplazar nuestro cuadricóptero una mayor distancia desde nuestra posición.

3.2 Elementos mecánicos Parrot Ar Drone 2.0

El resto de elementos que conforman el cuadricóptero y se han querido mencionar para tener una visión global, como son los actuadores, carcasas y estructura, son los siguientes:

I. Cruz central: es la base del cuadricóptero donde se montan las hélices y los motores. Está hecha de plástico de alta calidad PA 66 con 4 tubos de fibra de carbono cruzada integrados.
II. Hélices: son las encargadas de mantener el vuelo e impulsar nuestro drone del suelo. Fueron diseñadas por un equipo que ganó el concurso de mini-UAV de la Armada Francesa. Garantizan un bajo consumo de energía.

III. Engranajes y ejes: son los encargados de conectar las hélices con los motores. Los engranajes están fabricados con plástico de alta resistencia, y los ejes de acero inoxidable.

IV. Casco para interiores: está hecho de polipropileno expandido (PEE) lo que hace dotar al cuadricóptero de agilidad y solidez.

V. Casco para exteriores: está fabricado del mismo material que el casco de interiores. Con esta protección tenemos menos resistencia al viento pudiendo realizar un pilotaje más dinámico.

VI. Estructura inferior: fabricada también de polipropileno expandido (PEE). Es la base del cuadricóptero donde se monta toda la electrónica y demás elementos.

Las medidas del cuadricóptero con la carcasa interior son 517 mm y 420 g de peso, mientras que para exterior sin la carcasa son 451 mm y 380 g, como se muestra en la siguiente imagen:

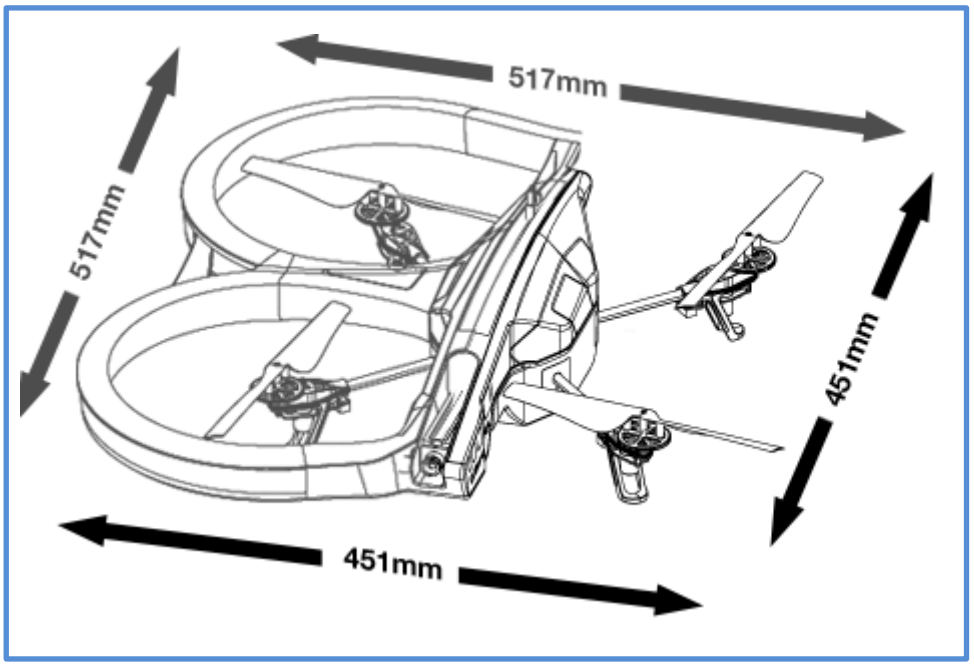


Figura 12. Medidas del AR Drone 2.0

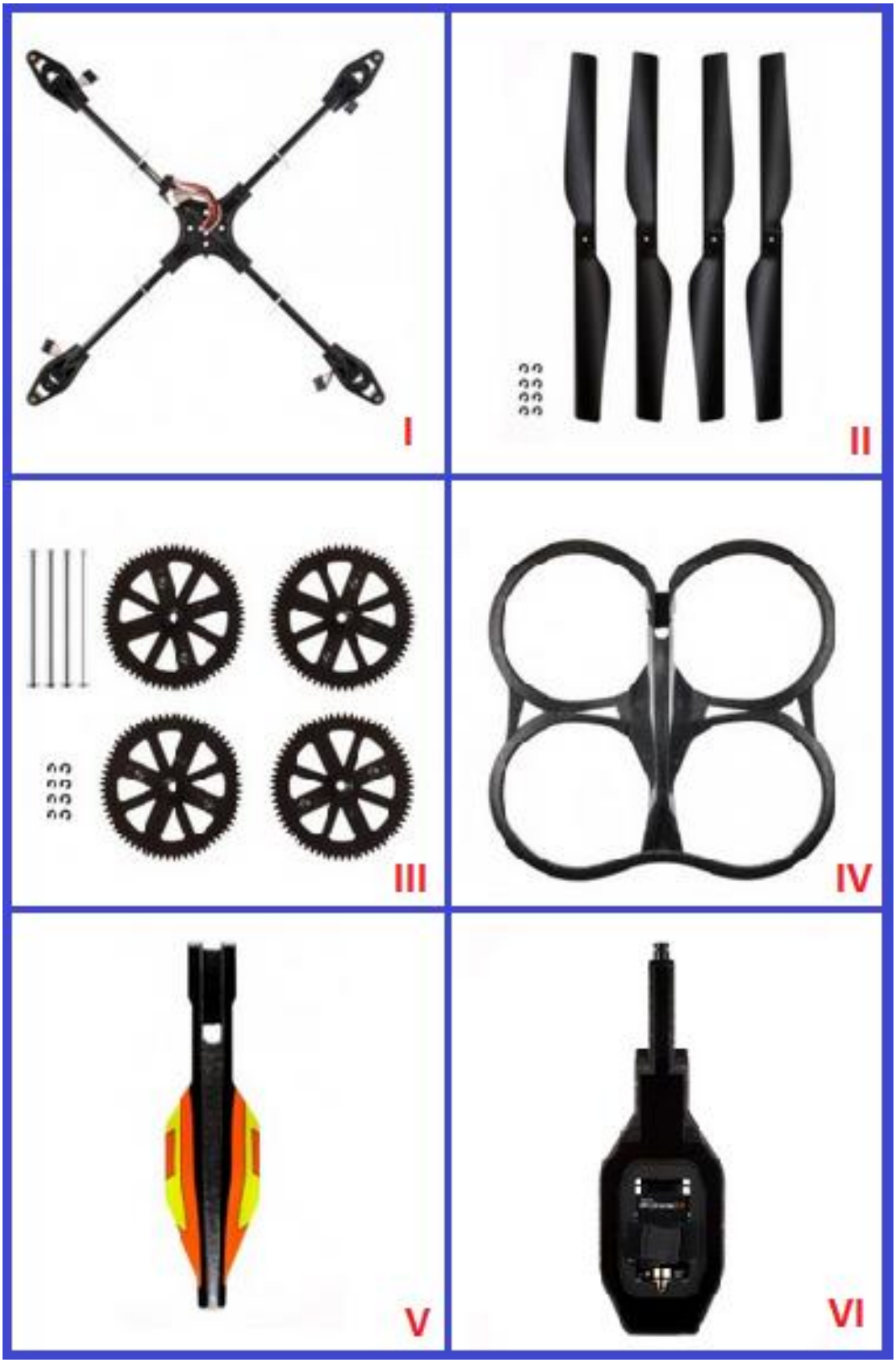


Figura 13. Componentes y actuadores del AR Drone 2.0

3.3 Ángulos de navegación

Los ángulos de navegación son los ángulos que permitirán realizar el movimiento de nuestro cuadricóptero, y básicamente son un tipo de ángulos de Euler para describir la orientación de un objeto en 3 dimensiones, los cuáles explicaremos a continuación relacionados con aeroplanos como cuadricópteros o helicópteros.

Podemos distinguir tres tipos de ángulos respecto a aeroplanos los cuáles son dirección o guiñada (yaw), elevación o cabeceo (pitch) y ángulo de alabeo (roll). Son rotaciones intrínsecas, relativas al sistema móvil las cuáles nos permiten describir una maniobra.

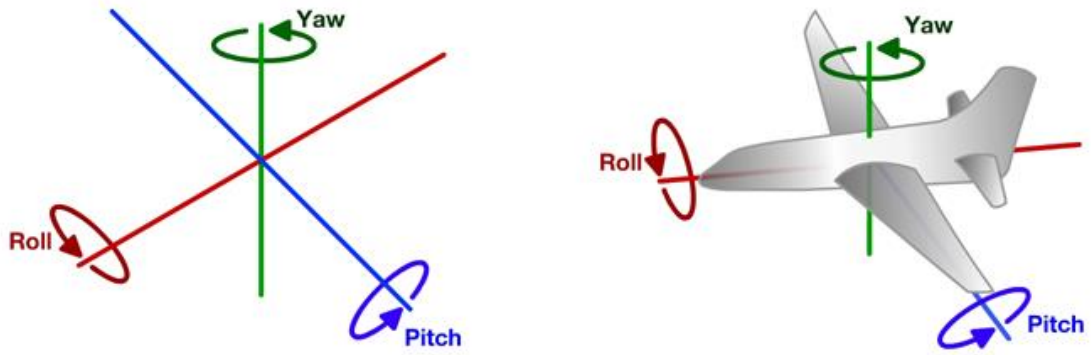


Figura 14. Ángulos de navegación de un cuadricóptero

- Pitch→es una inclinación del eje correspondiente al morro-cola de un cuadricóptero en este caso. Hace que nuestro UAV se desplace hacia el frente.
- Roll→es una inclinación del eje correspondiente al ala-ala de un cuadricóptero en este caso. Hace que nuestro UAV se desplace lateralmente.
- Yaw→ es una rotación alrededor del eje vertical perpendicular al avión, es decir perpendicular a nuestros ejes X e Y. Hace que nuestro UAV gire sobre sí mismo.

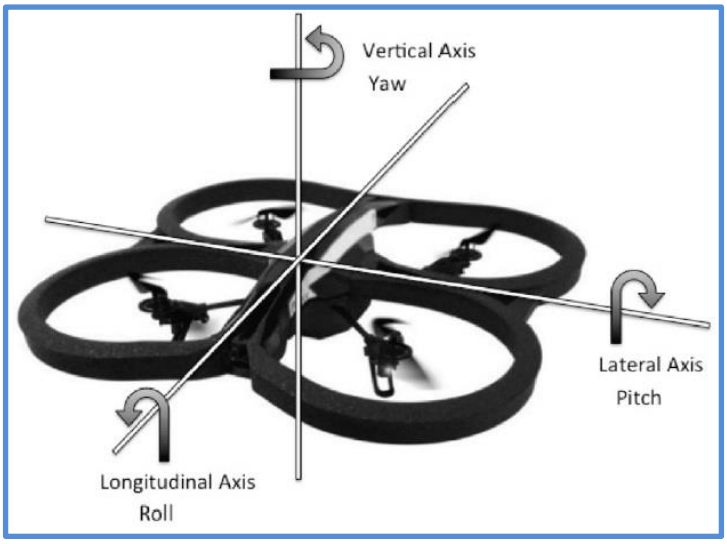


Figura 15. Ejes del AR Drone 2.0

Estos ángulos los podemos observar mejor en la siguiente imagen, donde se aprecia la cámara delantera de nuestro drone con sus ejes y rotaciones:

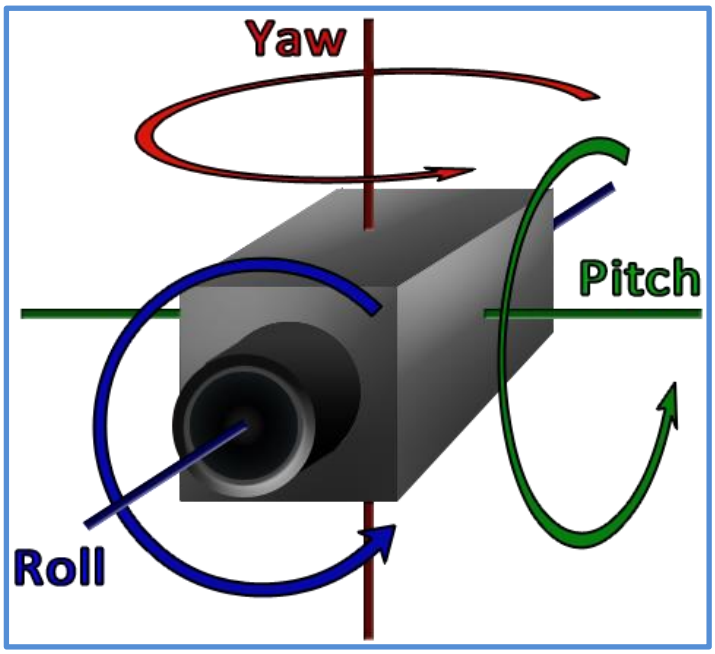


Figura 16. Ángulos expresados en la cámara delantera del AR Drone 2.0

A la hora de realizar un movimiento, tenemos que variar las diferentes velocidades lineales de nuestro sistema, de forma que para realizar los movimientos siguientes:

- Avance→aplicaremos una velocidad en X **negativa**, y tendrá movimiento de pitch con valor **negativo**. La variable que modificaremos será **dVx**.
- Retroceso→aplicaremos una velocidad en X **positiva**, y tendrá movimiento de pitch con valor **positivo**. La variable que modificaremos será **dVx**.
- Desplazamiento lateral hacia su derecha → aplicaremos una velocidad en Y **positiva**, y tendrá movimiento de roll con valor **positivo**. La variable que modificaremos será **dVy**.
- Desplazamiento lateral hacia su izquierda→ aplicaremos una velocidad en Y **negativa**, y tendrá movimiento de roll con valor **negativo**. La variable que modificaremos será **dVy**.
- Rotación hacia la derecha→aplicaremos a nuestra variable **dYaw** un valor **sumando**.
- Rotación hacia la derecha→aplicaremos a nuestra variable **dYaw** un valor **restando**.
- Ascenso vertical→aplicaremos a la variable de altura **dH** un valor **sumando**.
- Descenso vertical→ aplicaremos a la variable de altura **dH** un valor **restando**.

En la interfaz disponemos de unos datos que nos ayudan a saber el estado en que se encuentra el drone dependiendo de los datos anteriores, como son la orientación gracias al giróscopo del que dispone el cuadricóptero, altitud en que se encuentra, la inclinación de los ángulos pitch y roll, el ángulo de rotación yaw y la velocidad de los rotores para un ascenso o descenso vertical gaz.

Figura 17. Datos del vuelo en nuestra interfaz LSIDrone

3.3.1 Comportamiento rotores en los distintos movimientos

Para poder mover nuestro UAV debemos originar cambios de velocidad en los rotores, acelerando unos y desacelerando otros dependiendo del movimiento deseado. El comportamiento de los 4 rotores al realizar los movimientos anteriores y su funcionamiento, teniendo en cuenta que un cuadricóptero tiene 6 grados de libertad (3 actuados y 3 no actuados) con estabilizadores de control IMU, lo detallamos a continuación en imágenes:

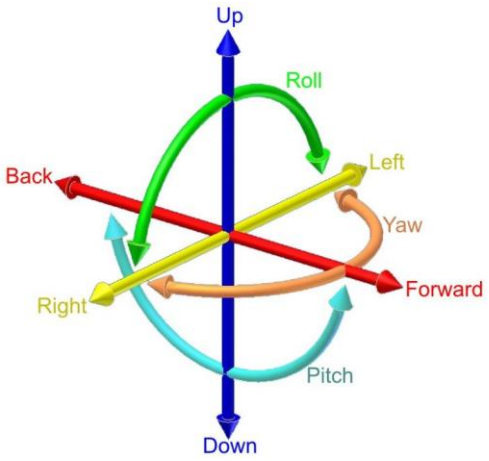


Figura 18. Grados de libertad de un cuadricóptero

- Hover o modo flotante: para que nuestro cuadricóptero se mantenga en el aire de manera estática, los 4 rotores deben mantenerse a la misma velocidad angular y en la dirección que se muestra en la siguiente figura. Es decir, el rotor 2 y 4 girarán en el sentido de las agujas del reloj (CW), mientras que los rotores 1 y 3 al contrario de las agujas del reloj (CCW). Esto es para compensar el par de fuerza generado, ya que en el caso de que todos los rotores giraran en el mismo sentido, nuestro cuadricóptero empezaría a dar vueltas al despegar.

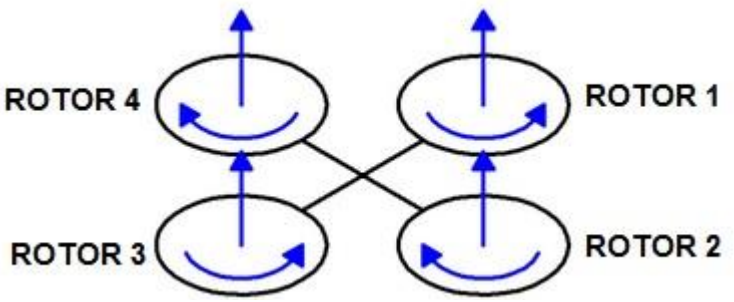


Figura 19. Comportamiento rotores en modo Hover

- Movimiento con pitch: es el que produce un movimiento de avance o retroceso de nuestro cuadricóptero. Tenemos que tener en cuenta que para avanzar nuestro ángulo de pitch será negativo con un valor comprendido entre 0 y -1 con velocidad lineal en X positiva, mientras que para retroceder nuestro ángulo de pitch será positivo y estará comprendido entre 0 y 1 con velocidad lineal en X negativa según la asignación de los ejes para este proyecto. El intervalo (-1,+1) es el valor máximo que puede alcanzar nuestro ángulo pitch para este movimiento, aunque al programar la trayectoria éste estará limitado entre (-0.5,+0.5).

  Para realizar un movimiento de avance los rotores 1 y 4 se mantienen a una velocidad angular igual, y los rotores traseros 2 y 3 (flecha verde) aumentan la velocidad angular lo que permite una inclinación del morro de nuestro cuadricóptero. Para retroceder los movimientos serán a la inversa, los rotores 1 y 4 aumentarán su velocidad, y los rotores 3 y 4 se mantendrán a las mismas revoluciones por minuto.

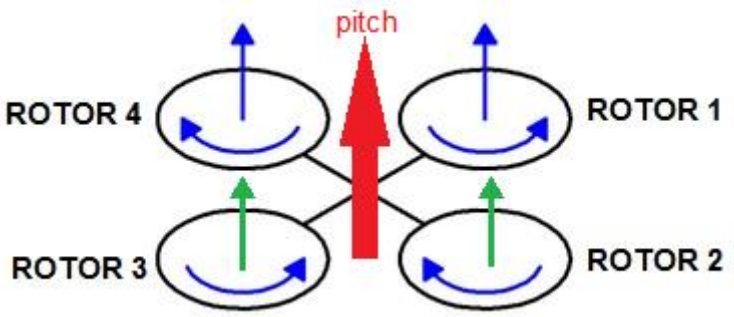


Figura 20. Comportamiento rotores con pitch

- Movimiento con roll: es el que produce un movimiento de desplazamiento lateral de nuestro cuadricóptero hacia su derecha o izquierda.

  Para realizar un movimiento de desplazamiento hacia la derecha, los rotores 1 y 2 se mantienen a una velocidad angular igual, y los rotores traseros 3 y 4 (flecha verde) aumentan la velocidad angular lo que permite una inclinación lateral del drone. Para movernos hacia la izquierda se haría justamente a la inversa, es decir dejando los rotores 3 y 4 mantendrían una velocidad que sería inferior a la de los rotores 1 y 2.

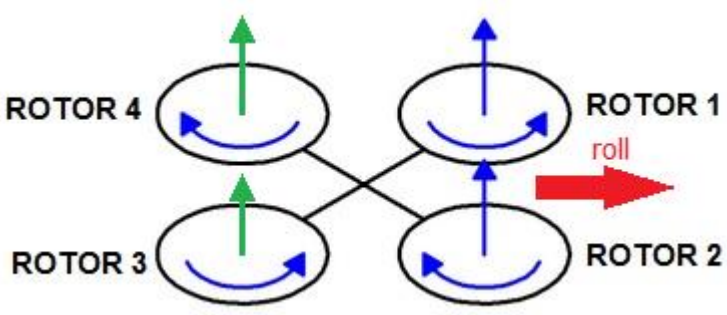


Figura 21. Comportamiento rotores con roll

- Movimiento con yaw: la variación de este ángulo nos permite rotar el cuadricóptero sobre sí mismo cambiando la orientación. Para este movimiento aumentaremos la velocidad de 2 rotores opuestos entre sí los cuáles serán 2-4, 1-3 y disminuiremos la de los otros 2 rotores respectivamente. En la imagen siguiente se expone el comportamiento de los rotores para hacer una rotación hacia la derecha del drone, en el que se aceleran los rotores con sentido horario (CW) y se desaceleran los rotores con sentido anti-horario (CCW). Con la flecha verde indicamos un aumento de la velocidad para los rotores 2 y 4 en este caso.

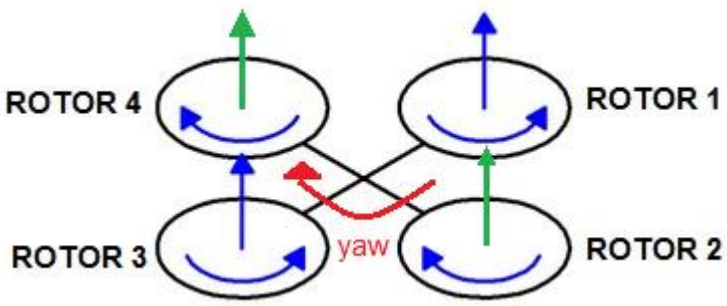


Figura 22. Comportamiento rotores con yaw

- Movimiento con throttle: es el movimiento de ascenso o descenso del cuadricóptero en el eje z verticalmente, es el encargado de controlar la altitud. Aquí los 4 rotores deben de tener la misma velocidad de forma que para ascender habrá que aumentar la velocidad de los cuatro rotores, y en consecuencia para descender habrá que reducir la velocidad de los rotores. En la siguiente imagen mostramos el comportamiento de los rotores al aumentar la altura de nuestro drone. Las flechas en color verde en la siguiente imagen indican un aumento de velocidad en todos los rotores.

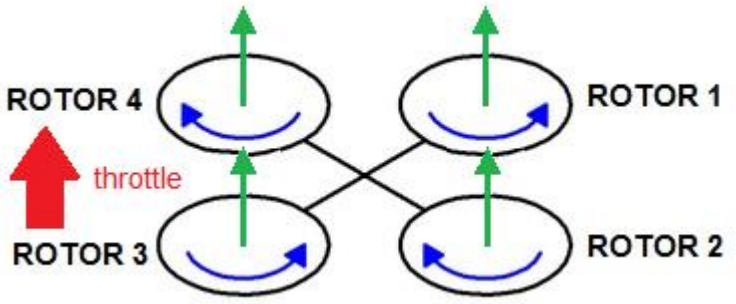


Figura 23. Comportamiento rotores con throttle

3.4 Interfaz en C#:

El proyecto se ha realizado en el conocido programa Microsoft Visual Studio 2013 con el lenguaje de programación C# como ya se ha comentado anteriormente. Para ello se ha partido de un software basado en el SDK distribuido por Parrot realizado por un equipo de desarrolladores, y proporcionado en la Universidad Carlos III de Madrid por parte de Abdulla Hussein Al-Kaff una versión mejorada del software inicial de Parrot. Este código fuente nos permite modificar cualquier parámetro de nuestro cuadricóptero para realizar los movimientos que deseemos; también se nos permite adaptar la interfaz a nuestro gusto, cambiando por ejemplo los mandos de control o eliminar parámetros que no nos interesen para nuestro proyecto. Para nuestro proyecto se ha reutilizado la mayoría de código y se han añadido todas las características necesarias para cumplir con las expectativas planteadas. Todos los algoritmos y modificaciones realizadas se explicarán detalladamente en el capítulo 4 del presente proyecto.

A continuación se muestra una imagen de la interfaz LSIDrone utilizada al comienzo del trabajo:

Figura 24. Interfaz LSIDrone inicial

Capítulo 4: Desarrollo del software

En el presente capítulo se va a explicar el desarrollo del software en el que se ha trabajado, por lo que es uno de los puntos más importantes del trabajo. Se van a mostrar los pasos seguidos desde el punto de partida hasta el trabajo final, que es conseguir que el cuadricóptero sea capaz de realizar un vuelo autónomo siguiendo una serie de puntos que le pasaremos al programa.

A lo largo del proyecto se ha modificado el software en muchas ocasiones hasta obtener resultados correctos, pero solo se expondrá aquí la solución desarrollada para solventar dichos problemas.

Para comprender el trabajo realizado a lo largo del proyecto podemos dividir este capítulo en varias partes, que explicaremos en los siguientes apartados, las cuales son:

- Familiarización y estabilización del cuadricóptero.
- Modificación de la interfaz para poder mover el cuadricóptero con la botonera de forma manual.
- Cálculo y desarrollo de los algoritmos para seguir una trayectoria.
- Implementación del software y programación de los algoritmos con todas las condiciones necesarias para su control determinado.

4.1 Punto de partida

Una vez instalado el programa LSIDrone y establecida la comunicación con el AR Drone 2.0, obtenemos la interfaz de trabajo. Es un software de licencia libre derivado del original suministrado por Parrot AR.DRONE.CONTROL.NET, habiendo sido mejorado como se ha comentado anteriormente, ya que se le han incluido botones para poder ser controlado por el usuario desde la propia interfaz, y un sistema de control que se encarga de no sobrepasar ciertos valores de roll, pitch, yaw y gaz para que nuestro cuadricóptero no se revolucione demasiado. Nuestro objetivo ha sido mejorar aún más esta interfaz para poder realizar vuelos autónomos. Tanto en la interfaz de Parrot como la de la Universidad, la reproducción del video obtenido por las cámaras del drone está habilitada.

Los creadores del SDK distribuido por Parrot son Julien Vinel, Thomas Endres y Steve Hobley. Esta interfaz nos permite modificar cualquier parámetro del cuadricóptero como puede ser la velocidad, altitud y orientación entre otros.

A continuación mostraremos la interfaz original de Parrot, la cual simplemente se instaló para hacer una comparativa y un estudio inicial del sistema. Los pasos para instalar y compilar el programa se encuentran en los anexos I y III al final de esta memoria.

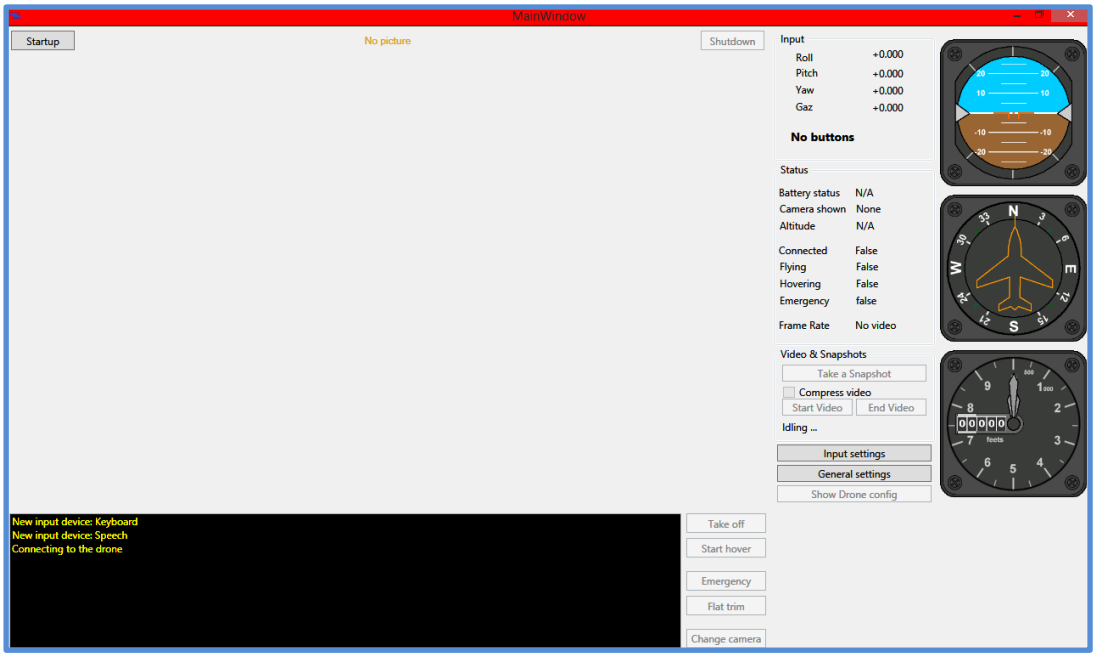


Figura 25. Interfaz original de Parrot

Como podemos observar la interfaz LSIDrone es más completa que la de Parrot, ya que tenemos una botonera a la derecha la cual nos permite avanzar, retroceder, rotar y desplazar nuestro cuadricóptero en lugar de realizarlo por teclado, aunque no eran muy precisas al comienzo. Además también tenemos datos extras que nos proporcionan mayor información del vuelo.

Para poder variar el comportamiento de nuestro cuadricóptero tenemos diferentes variables privadas declaradas de tipo float, que son parámetros del sistema de control. Estos parámetros son:

- dH→con este variaremos la altura del cuadricóptero, que por defecto es 1000 mm.
- dYaw→es el encargado de girar un determinado número de grados la orientación de nuestro cuadricóptero.
- dVx→con este parámetro podremos desplazar el cuadricóptero hacia adelante si le asignamos velocidad negativa, o hacia atrás si le establecemos velocidad positiva.
- dVy→con este parámetro podremos desplazar lateralmente el cuadricóptero hacia su derecha con velocidad positiva, o hacia su izquierda con velocidad negativa.

Adicionalmente tenemos diferentes modos o comportamientos previamente definidos que nos permiten realizar diferentes acciones, estos son:

- Takeoff ()→se encarga de despegar el cuadricóptero.
- Land ()→se utiliza para aterrizar nuestro cuadricóptero.
- FlatTrim ()→esta instrucción es la encargada de poner a cero todos los valores de los sensores inerciales en posición horizontal. Es muy importante tener en cuenta que antes de despegar, el cuadricóptero debe tener los sensores a cero para que despegue perpendicularmente al suelo y se mantenga estable.
- Emergency ()→este modo se utiliza para situaciones de emergencia y se encarga de detener los cuatro motores de forma que nuestro cuadricóptero dejará de funcionar y se detendrá. Es importante no utilizarlo si la situación no lo requiere ya que se puede dañar nuestro dispositivo al caer al suelo, dado que no es un aterrizaje lo que hace.
- EnterHoverMode ()→mediante esta instrucción nuestro cuadricóptero entrará en modo flotante, es decir se mantendrá estable en el aire a velocidad cero en x, y, z. Este modo es muy importante al cambiar de instrucción, tanto en el modo de vuelo manual como en el autónomo.
- LeaveHoverMode ()→con esta instrucción nuestro cuadricóptero abandonará el modo flotante explicado anteriormente para volver a moverse.

El uso correcto de estos parámetros lo explicaremos detalladamente en el siguiente apartado. Cabe destacar que estos modos los tenemos en nuestra interfaz a través de botones también, por lo que si en alguna situación anómala necesitamos de su uso, podremos utilizarlos rápidamente.

4.2 Modificación de interfaz

En este apartado vamos a explicar cómo se ha estabilizado el vuelo del cuadricóptero ante los diferentes problemas iniciales y qué botones se han añadido para potenciar nuestro programa.

4.2.1 Estabilización del vuelo

Al comienzo del proyecto se tuvieron muchos problemas con el vuelo inicial ya que era descontrolado y acababa estrellándose con las paredes o el techo de la habitación al despegar, o al presionar cualquier botón. Con el estudio y conocimiento del programa se observó tras varias pruebas que la programación de la interfaz inicial no era la correcta en determinados parámetros.

A continuación vamos a explicar el desarrollo seguido hasta conseguir un vuelo manual correcto con ayuda de la botonera.

4.2.1.1 *Despegue y aterrizaje*

El desarrollo de esta función la tenemos en el botón **btnControlTakeLand,** el cuál utilizaremos para despegar y aterrizar el drone. La programación de la función de los botones se hace en la función principal MainForm () dentro del proyecto LSIDroneInterface {}, al igual que el resto de botones.

Tenemos que tener en cuenta que si el drone no está volando tiene que despegar al presionar el presente botón, y para ello debe resetear los valores de roll, pitch, yaw y la altitud. De realizar esto se encarga la función explicada anteriormente FlatTrim (). Esto se hace para que al despegar no lo haga con los valores memorizados en un vuelo anterior dentro de la misma compilación, y porque suponemos y obligamos a que el cuadricóptero despegue en llano desde el suelo. Tras el reinicio de los valores anteriores el drone despegara con ayuda de la función Takeoff (), y este se mantendrá a una determinada altura especificada por la variable dH en milímetros, que por defecto va a ser 1 metro (1000 mm). Una vez que nuestro cuadricóptero ha despegado queremos que se mantenga detenido en el aire con un vuelo estable, y para ello entraremos en el modo flotante con la función EnterHoverMode (), explicado con anterioridad en el apartado 4.1. Con estas especificaciones se consiguió despegar el drone de manera exitosa.

Por el contrario si nuestro drone está volando y queremos aterrizarlo, tenemos que llamar a la función Land () pulsando nuevamente el botón, y hará que aterrice.

La programación completa del botón **btnControlTakeLand** es la siguiente:

```
private void btnControlTakeLand_Click(object sender, EventArgs e)
{
        if (!droneControl.IsFlying)
        {
           FlatTrim();
           Takeoff();
           dH = 1000; dVx = 0.0f; dVy = 0.0f;
           dYaw = (float)clsARDroneGeneralParams.clsYaw;
           EnterHoverMode();
         }
        else
        {
           Land();
        }
 }
```

4.2.1.2 *Avance y retroceso*

Para realizar los movimientos de avance y retroceso tendremos dos botones que son **btnForward** y **btnBack** respectivamente.

Tenemos que tener en cuenta que para realizar estos movimientos vamos a variar la velocidad lineal en el eje X del cuadricóptero, dVx, de manera que si le asignamos velocidad negativa avanzará hacia delante con respecto a si mismo, y de igual forma si le asignamos velocidad positiva el drone retrocederá.

Aparte para que el cuadricóptero responda adecuadamente debemos saber que estos movimientos tienen ángulo pitch que variará según la velocidad que le pongamos, de forma que a mayor velocidad mayor será el ángulo o inclinación. Para ello debemos asignar el tipo de dato lógico o booleano en verdadero, cuyo control se realizara llamando a la función **tmrTestControl ().** La velocidad que se ha asignado por defecto es 500 mm/s a una altura de 1000 mm.

La programación completa del botón **btnForward** es la siguiente:

```
private void btnForward_Click(object sender, EventArgs e)
{
      LeaveHoverMode();
      dH = 1000;
      dVx = -500.0f;
      dVy = 0.0f;
      dYaw = (float)clsARDroneGeneralParams.clsYaw;
      UpdateUIAsync("Avanzando...");
      boolpitch = true;
      boolroll = false;
      boolyaw = false;
      tmrTestControl.Start();

}
```

La programación completa del botón **btnBack** es la siguiente:

```
private void btnBack_Click(object sender, EventArgs e)
{
       LeaveHoverMode();
       dH = 1000;
       dVx = +500.0f;
       dVy = 0.0f;
       dYaw = (float)clsARDroneGeneralParams.clsYaw;
       UpdateUIAsync("Retrocediendo...");
       boolpitch = true;
       boolroll = false;
       boolyaw = false;
       tmrTestControl.Start();
}
```

4.2.1.3 *Desplazamiento lateral hacia derecha e izquierda*

Para realizar los movimientos de desplazamiento lateral hacia derecha e izquierda tendremos dos botones que son **btnYRight** y **btnYLeft** respectivamente.

Tenemos que tener en cuenta que para realizar estos movimientos vamos a variar la velocidad en el eje Y del cuadricóptero, dVy, de manera que si le asignamos velocidad negativa avanzará hacia la izquierda con respecto a si mismo, de igual forma si le asignamos velocidad positiva el drone se desplazara hacia la derecha.

Aparte para que el cuadricóptero responda adecuadamente debemos saber que estos movimientos tienen ángulo roll que variará según la velocidad que le pongamos, de forma que a mayor velocidad mayor será el ángulo o inclinación lateral. Para ello debemos asignar el booleano de roll en verdadero, cuyo control se realizara llamando a la función **tmrTestControl(),** como en el avance y retroceso**.** Las velocidades por defecto son 500 mm/s, a una altura igual que en el anterior punto de 1000 mm.

La programación completa del botón **btnYRight** es la siguiente:

```
private void btnYRight_Click(object sender, EventArgs e)
{
      LeaveHoverMode();
      dH = 1000;
      dVx = 0.0f;
      dVy = +500.0f;
      dYaw = (float)clsARDroneGeneralParams.clsYaw;
      boolpitch = false;
      boolroll = true;
      boolyaw = false;
      UpdateUIAsync("Desp. lateral hacia la derecha");
      tmrTestControl.Start();
}
```

La programación completa del botón **btnYLeft** es la siguiente:

```
private void btnYLeft_Click(object sender, EventArgs e)
{
      LeaveHoverMode();
      dH = 1000;
      dVx = 0.0f;
      dVy = -500.0f;
      dYaw = (float)clsARDroneGeneralParams.clsYaw;
      boolpitch = false;
      boolroll = true;
      boolyaw = false;
      UpdateUIAsync("Desp. lateral hacia la izquierda");
      tmrTestControl.Start();
}
```

4.2.1.4 *Rotación hacia derecha e izquierda*

Para realizar los movimientos de rotación tendremos dos botones que son **btnYawLeft** y **btnYawRight** respectivamente.

Tenemos que tener en cuenta que para realizar estos movimientos vamos a variar el ángulo de yaw en el eje perpendicular al cuadricóptero que consideraremos eje Z, cuya variable a modificar es dYaw.

Si a la variable dYaw le sumamos un determinado ángulo, nuestro drone girará en el sentido de las agujas del reloj visto desde planta, y de igual forma si le restamos girará en sentido contrario. Para ello debemos asignar el booleano de yaw en verdadero, cuyo control se realizara llamando a la función **tmrTestControl ()**. Los giros por defecto son de 90º, estando el drone situado a una distancia de 1000 mm del suelo.

La programación completa del botón **btnYawLeft** es la siguiente:

```
private void btnYawLeft_Click(object sender, EventArgs e)
{
      LeaveHoverMode();
      dH = 1000; dVx = 0.0f; dVy = 0.0f;
      dYaw = dYaw - 90.0f;
      UpdateUIAsync("Girando 90º a la izquierda...");
      boolpitch = false;
      boolroll = false;
      boolYaw = true;
      tmrTestControl.Start();
}
```

La programación completa del botón **btnYawRight** es la siguiente:

```
private void btnYawRight_Click(object sender, EventArgs e)
{
       LeaveHoverMode();
       dH = 1000; dVx = 0.0f; dVy = 0.0f;
       dYaw = dYaw + 90.0f;
       UpdateUIAsync("Girando 90º a la derecha...");
       boolpitch = false;
       boolroll = false;
       boolYaw = true;
       tmrTestControl.Start();
}
```

4.2.2 Creación de los botones movimiento vertical

Una vez estabilizado el vuelo, se han diseñado dos botones adicionales a nuestra interfaz para poder mover el cuadricóptero verticalmente. Los botones se llaman **btnUp** para ascender verticalmente, y **btnDown** para descender. La función de estos será producir en el drone un ascenso o descenso en la altitud de 500 mm, por defecto, cada vez que sea presionado.

En este caso cambiaremos la variable dH, es decir modificaremos la posición de nuestro drone en el eje Z, de tal forma que si le sumamos un valor ascenderá, y descenderá al restarle un valor.

Estos botones son idénticos a los anteriores en cuanto a características y diseños, y su declaración y creación se ha hecho siguiendo los siguientes pasos:

i. Dentro del proyecto LSIDroneInterface tendremos que ir a la siguiente ruta:

LSIDroneInterface → MainForm.cs → MainForm.Designer.cs → MainForm

En este último archivo haremos doble click para abrir el diseño de nuestra interfaz, en la cual podremos modificar y añadir diferentes botones o etiquetas de acuerdo a nuestras necesidades.

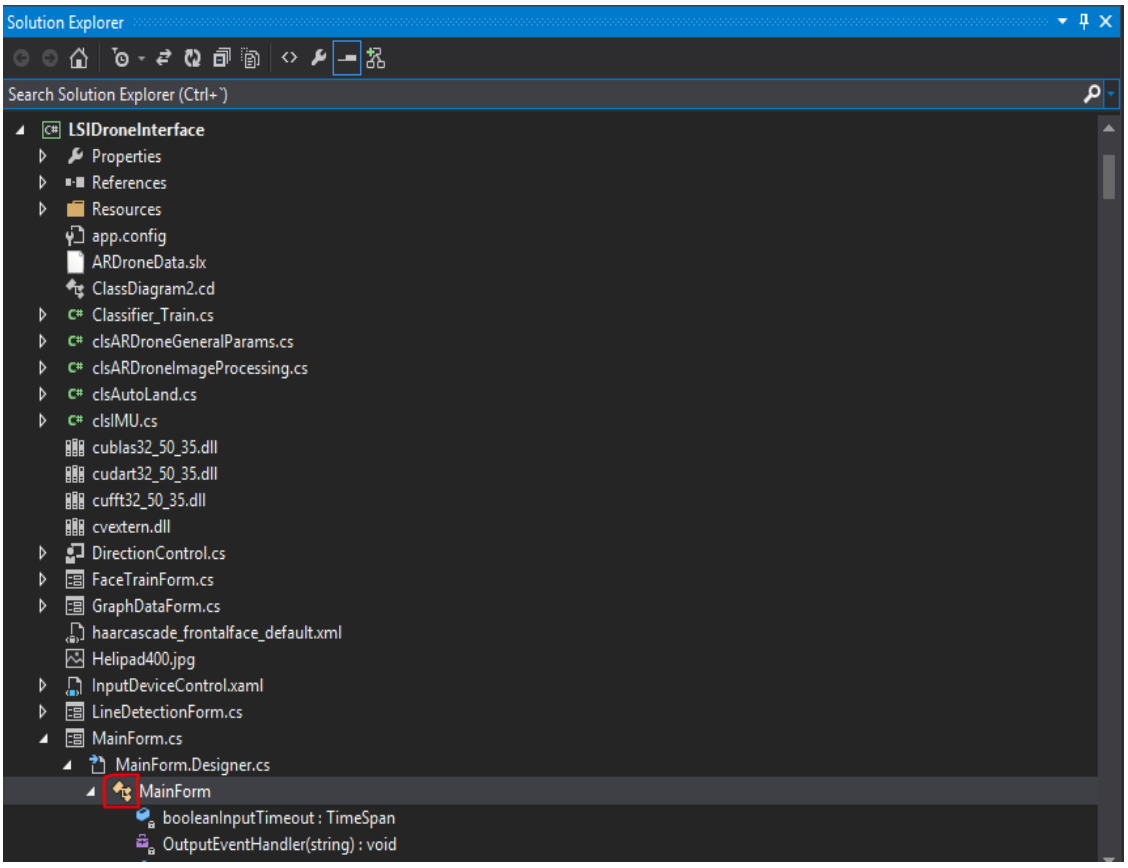


Figura 26. Acceso al Mainform del proyecto

ii. En la función principal, tendremos que declarar los dos botones en la función *InitializeComponent()*:

```
this.btnUp = new System.Windows.Forms.Button();
this.btnDown = new System.Windows.Forms.Button();
```

iii. Dentro de esta misma función tendremos que definir los nombres, colores, tamaños y tipo de botón que deseamos, que en este caso será de estilo PopUp el cual nos permite cliquear estos botones.

```
//
// btnUp
//
this.btnUp.BackColor = System.Drawing.Color.Navy;
this.btnUp.FlatAppearance.MouseDownBackColor =
        System.Drawing.Color.Cyan;
this.btnUp.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
this.btnUp.Location = new System.Drawing.Point(1186, 564);
this.btnUp.Name = "btnUp";
this.btnUp.Size = new System.Drawing.Size(75, 23);
this.btnUp.TabIndex = 146;
this.btnUp.Text = "Up";
this.btnUp.UseVisualStyleBackColor = false;
this.btnUp.Click += new System.EventHandler(this.btnUp_Click);
//
// btnDown
//
this.btnDown.BackColor = System.Drawing.Color.Navy;
this.btnDown.FlatAppearance.MouseDownBackColor =
        System.Drawing.Color.Cyan;
this.btnDown.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
this.btnDown.Location = new System.Drawing.Point(1186, 593);
this.btnDown.Name = "btnDown";
this.btnDown.Size = new System.Drawing.Size(75, 23);
this.btnDown.TabIndex = 147;
this.btnDown.Text = "Down";
this.btnDown.UseVisualStyleBackColor = false;
this.btnDown.Click += new System.EventHandler(this.btnDown_Click);
//
// btnAutonomousFlight
//
```

iv. Lo siguiente será añadir el control al proyecto mediante la siguiente instrucción:

```
this.Controls.Add(this.btnDown);
this.Controls.Add(this.btnUp);
```

v. Y por último, de forma automática el programa nos declara los botones al final, de la siguiente forma:

```
private System.Windows.Forms.Button btnUp;
private System.Windows.Forms.Button btnDown;
```

Lo siguiente será programar los botones, y para ello hacemos doble click en el botón deseado, dentro de la interfaz, la cual abriremos en el archivo *MainForm.cs [Design]**. Esto hará que se abra y declare nuestro botón en el proyecto, donde especificaremos sus parámetros.

La programación completa del botón **btnUp** es la siguiente:

```
private void btnUp_Click(object sender, EventArgs e)
{
      LeaveHoverMode();
      dH = dH + 500;
      dVx = 0.0f;
      dVy = 0.0f;
      UpdateUIAsync("Subiendo 0,5 metros...");
      boolpitch = false;
      boolroll = false;
      boolyaw = false;
      tmrTestControl.Start();
}
```

La programación completa del botón **btnDown** es la siguiente:

```
private void btnDown_Click(object sender, EventArgs e)
{
    LeaveHoverMode();
    dH = dH - 500;
    dVx = 0.0f;
    dVy = 0.0f;
    UpdateUIAsync("Bajando 0,5 metros...");
    boolpitch = false;
    boolroll = false;
    boolyaw = false;
    tmrTestControl.Start();
 }
```

4.2.3 Creación botones para vuelo autónomo

La última modificación que tendremos que hacer en la interfaz es crear los botones de vuelo autónomo. Al pulsar estos botones nuestro cuadricóptero realizará el vuelo autónomo que hayamos programado. El desarrollo de los algoritmos se expondrá en los apartados 4.5 y 4.6.

El botón para el movimiento del algoritmo 1 se va a llamar **btnAutonomousFlight** y se creará de forma similar a los botones definidos para mover el drone verticalmente enseñado en el apartado 4.2.2, con la única diferencia que este tendrá una imagen para hacerlo más vistoso y llamativo, que es la siguiente:

Figura 27. Imagen botón vuelo autónomo 1

Los pasos para el diseño del botón serán similares a los anteriores, aunque tendremos que modificar las propiedades para poder insertar la imagen. La definición de nuestro botón acorde a los pasos explicados con anterioridad será la siguiente:

i. Abrimos la interfaz.

ii.
```
this.btnAutonomousFlight = new System.Windows.Forms.Button();
```

iii.

```
// btnAutonomousFlight
//
this.btnAutonomousFlight.BackColor = System.Drawing.Color.Transparent;
this.btnAutonomousFlight.BackgroundImage =
((System.Drawing.Image)(resources.GetObject("btnAutonomousFlight.BackgroundImag
e")));
this.btnAutonomousFlight.BackgroundImageLayout =
        System.Windows.Forms.ImageLayout.Zoom;
this.btnAutonomousFlight.FlatAppearance.MouseDownBackColor =
        System.Drawing.Color.Black;
this.btnAutonomousFlight.FlatAppearance.MouseOverBackColor =
        System.Drawing.Color.Silver;
this.btnAutonomousFlight.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
this.btnAutonomousFlight.ForeColor = System.Drawing.Color.Black;
this.btnAutonomousFlight.Location = new System.Drawing.Point(617, 529);
this.btnAutonomousFlight.Name = "btnAutonomousFlight";
this.btnAutonomousFlight.Size = new System.Drawing.Size(104, 107);
this.btnAutonomousFlight.TabIndex = 148;
this.btnAutonomousFlight.UseVisualStyleBackColor = false;
this.btnAutonomousFlight.Click += new
        System.EventHandler(this.btnAutonomousFlight_Click);
        //
```

iv.

```
this.Controls.Add(this.btnAutonomousFlight);
```

v.

```
private System.Windows.Forms.Button btnAutonomousFlight;
```

vi. La imagen deseada para el botón deberemos añadirla a la siguiente ruta, donde se encuentra nuestro proyecto:

C:\Users\Fran\Desktop\LSIDroneInterface Fco Javier\LSIDroneInterface\Resources

vii. Lo siguiente será asignarla al botón deseado, y para ello dentro de la interfaz nos posicionamos con el ratón sobre el botón y se nos abrirán las propiedades abajo a la derecha como se muestra a continuación.

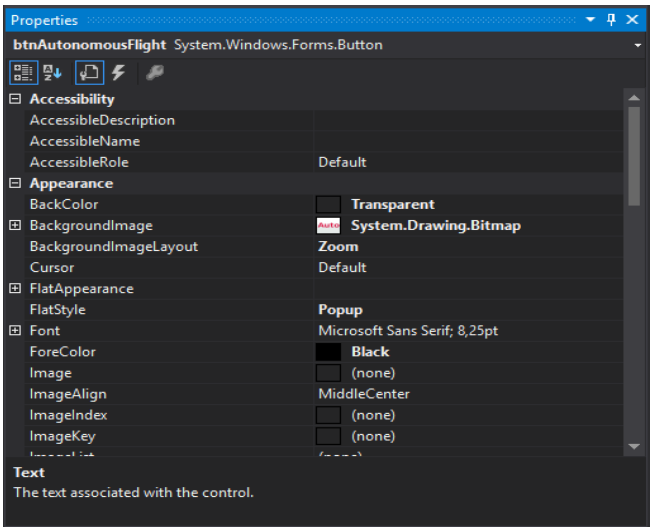


Figura 28. Acceso a propiedades de los botones

viii. En el apartado de apariencia seleccionamos BackgroundImage y cargamos la imagen que hemos añadido con anterioridad, de forma que nuestro botón estará listo para ser programado con las instrucciones de control deseadas.

Para crear el botón de vuelo autónomo para el algoritmo 2 que contendrá un movimiento de pitch con roll conjuntamente, los pasos son idénticos a estos anteriores expuestos al comienzo de este apartado 4.2.3. La programación de este movimiento irá desarrollada en el control 3 de nuestro sistema **tmrTestControl3**.El botón se llamará **buttonAutonomousFlight2** y lo podremos reconocer por la siguiente imagen:

Figura 29. Imagen botón vuelo autónomo 2

Una vez que hemos ordenado los botones en las posiciones de la pantalla que queremos simplemente arrastrándolos, la interfaz final de nuestro proyecto queda como se muestra a continuación:

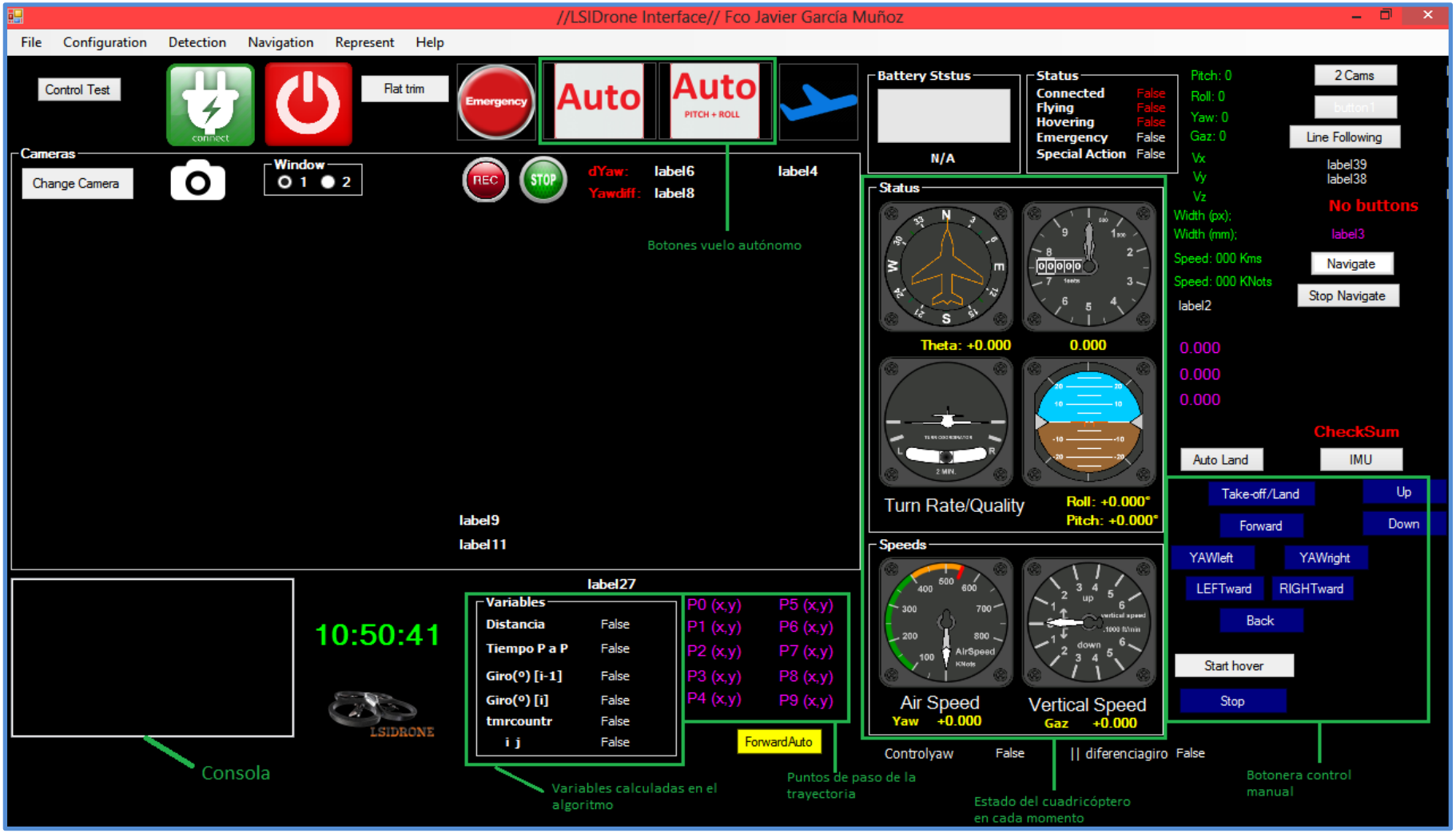


Figura 30. Interfaz final del proyecto

4.3 Configuración del vuelo manual

Para poder realizar un vuelo manual, con la botonera situada a la derecha en nuestra interfaz, sin perder el control del cuadricóptero se debe llamar a la función **tmrTestControl ()** siempre, como se ha indicado en la programación de los botones.

Nosotros queríamos tener un control con estos botones de forma que al pulsar un botón el drone realizara ese movimiento durante un determinado tiempo, es decir que entrara en modo flotante pasados unos segundos. De esta forma podremos controlar de forma correcta el vuelo, ya que al pulsar un botón cualquiera no realizará ese movimiento hasta que se estrelle o cambiemos de instrucción. Hemos querido hacer un control similar a la aplicación creada por Parrot para smartphones, de forma que al pulsar un botón realizará ese movimiento y después entrará en modo flotante hasta nueva instrucción.

Para poder detener nuestro cuadricóptero y entrar en modo flotante hemos tenido que modificar la función de control añadiendo un contador. Se ha establecido que al pulsar un botón, si no se pulsa otro antes, tras 4 segundos entrará el HoverMode (). Los pasos han sido los siguientes:

i. Hemos declarado una variable de tipo entero con valor inicial 0, llamada **tmrcountr_manual.**

ii. Dentro del control hemos declarado la instrucción **tmrcountr_manual= tmrcountr_manual + 1,** es decir que cada vez que el control se encuentre activado y trabajando nuestro contador sume 1.

iii. La condición establecida ha sido que cuando el contador supere un determinado tiempo(3 segundos) entre a **HoverMode ()**, y se reinicie a cero la variable para que al ser llamado nuestro control de nuevo por otro botón pueda volver a contar desde cero.

```
if (tmrcountr_manual >= 6)
{
      EnterHoverMode();
      tmrcountr = 0;
}
```

iv. Para lograr detener el drone al pulsar un botón pasados 3 segundos de forma automática, la condición ha sido que el contador debe ser mayor o igual que 8 por el intervalo declarado en nuestra función de control. El intervalo de la función **tmrTestControl ()** es de 500 milisegundos, es decir cada vez nuestro contador suma 1

en el control han pasado 500 milisegundos, lo que hace que para llegar a los 3 segundos que queremos que se detenga tendrá que contar 8. A continuación vemos cómo se declaran e inicializan los parámetros de nuestra función de tipo DispatcherTimer:

→Declaración:

```
private DispatcherTimer tmrTestControl;
```

→Inicialización dentro de la función **private void InitializeTimers ()**, donde podemos observar el intervalo:

```
tmrTestControl = new DispatcherTimer();
tmrTestControl.Interval = new TimeSpan(0, 0, 0, 0, 500);//200
tmrTestControl.Tick += new EventHandler(tmrTestControl_Tick);
```

4.4 Sistema de control en lazo abierto

El sistema de control utilizado para seguir una trayectoria en nuestros algoritmos 1 y 2 será en lazo abierto. Un sistema en lazo abierto se caracteriza porque un determinado proceso actúa sobre la señal de entrada, que en nuestro caso son los puntos y sus condiciones, y se obtiene una señal de salida que es independiente y no influye sobre la señal de entrada. La salida está basada en la entrada, pero no hay realimentación para poder ajustarse ante determinadas perturbaciones en nuestro sistema. En nuestro sistema, el controlador dependiendo de los puntos que le entren calcula los parámetros necesarios para desplazar el cuadricóptero hacia ese punto, y tras llegar a ese punto vuelve a calcular los nuevos parámetros con el siguiente punto sin comparar si está realmente en ese punto o no, aunque matemáticamente sí debería estar.

Los sistemas en lazo abierto se caracterizan por:

- Dependen de la variable de tiempo, que es con la que controlamos las distintas acciones y procesos.
- La salida es independiente de la entrada.
- Sencillos y de fácil concepto.

- Su estabilidad puede ser afectada por perturbaciones externas ya que son muy sensibles.
- La precisión depende de la calibración del sistema, por lo que debemos realizar un sistema tras el estudio lo más exacto posible con muchas pruebas para comprender su funcionamiento.

La exactitud de estos sistemas depende de su programación previa, y hay que prever las relaciones entre los diferentes componentes del sistema para intentar que la salida sea la esperada y alcance el valor previsto calculado teóricamente.

Nuestro sistema en lazo abierto tiene la siguiente estructura:

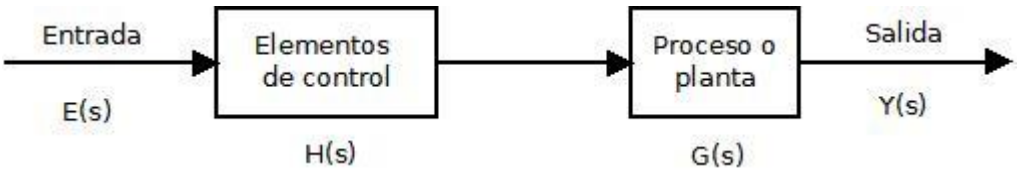


Figura 31. Sistema de control en lazo abierto

De acuerdo a la imagen anterior, los elementos de nuestro sistema se corresponden:

- Entrada→corresponde con el punto deseado al que tiene que desplazarse el drone.
- Elementos de control→programación del control y cálculo de los parámetros.
- Proceso o planta→corresponde con el movimiento de nuestro cuadricóptero con la función Navigate () que es la encargada de ello, de acuerdo a los parámetros calculados en el paso anterior.
- Salida→corresponde con el punto real al que se ha desplazado. Este valor tendremos que compararlo con la entrada para medir el error. Esto será detallado en el capítulo 5 de la presente memoria.

4.5 Algoritmo 1

En este apartado se va a explicar cómo se ha calculado el algoritmo para poder seguir una trayectoria con nuestro cuadricóptero, y la configuración de los ejes y giros. Se verán todos los casos posibles que nos podemos encontrar al ir de punto de punto dependiendo de su posición. En este algoritmo nuestro cuadricóptero seguirá la trayectoria únicamente con ángulo de navegación pitch o cabeceo, es decir con la parte delantera.

Partiremos de la posición inicial (0,0) en el suelo y la orientación inicial será 0 también, es decir el drone seguirá los puntos respecto a la orientación inicial que se encuentre indiferentemente de si se encuentra mirando al norte o no, de forma que el eje X coincidirá siempre con el eje morro-cola del cuadricóptero y la cámara frontal del drone al comienzo del vuelo. Los puntos de paso se marcarán respecto a la posición del UAV antes de despegar y los giros en cada caso dependerán siempre del punto anterior.

Dado que los ejes se encuentran desacoplados, se ha hecho coincidir para relacionar ángulo de pitch con velocidad (dVx) y posición en X, y ángulo de roll con velocidad (dVy) y posición en Y. Este sistema de coordenadas será fijo respecto al drone, pero los puntos cardinales marcados no y al momento de cargar la trayectoria la orientación podrá ser cualquiera. En la imagen siguiente se muestra cómo se han establecido los ejes para el cálculo.

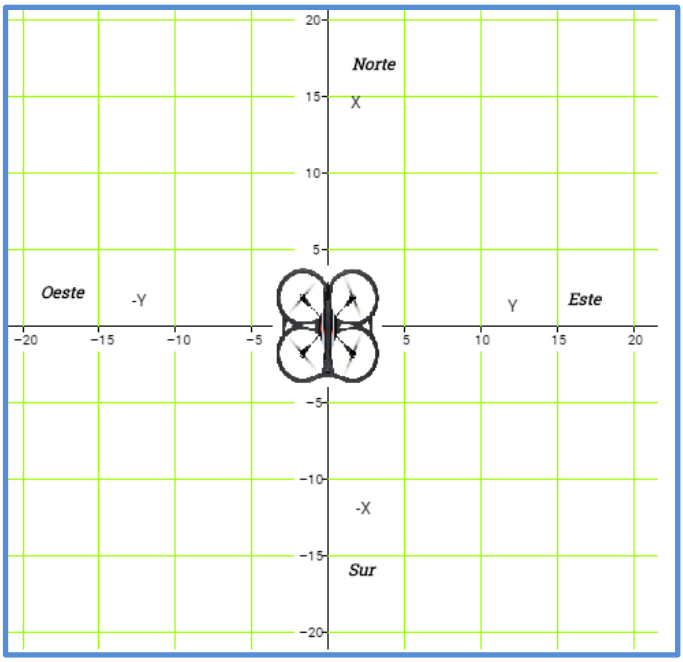


Figura 32. Asignación de ejes algoritmo 1

Para desplazar nuestro cuadricóptero de un punto a otro siempre realizaremos un giro previo dependiendo de la posición del punto siguiente, y después avanzará aplicando una velocidad en X negativa acuerdo a los ángulos de navegación del cuadricóptero establecido. La programación completa del algoritmo se realiza en el control **tmrTestControl2()**.

Para el cálculo del algoritmo hemos declarado una serie de variables que variarán al cambiar de punto como son:

- distancia[i]→ la distancia que hay entre dos puntos en línea recta, y que tendrá que recorrer nuestro cuadricóptero. Este parámetro será general para todos los casos.

- tiempoflying[i]→el tiempo que estará volando de punto a punto sabiendo que nos desplazaremos a 500 mm/s. Este parámetro será general para todos los casos.
- beta[i]→el ángulo que hay entre dos puntos respecto al eje Y, β. Este parámetro será general para todos los casos.
- giro[i]→el giro que se hará para orientar el drone hacia el punto siguiente, dependiendo del caso en el que nos encontremos.

En total podremos encontrarnos con 9 casos posibles dependiendo de las condiciones de los puntos, que explicaremos a continuación.

4.5.1 Secuencia del movimiento y código algoritmo 1

La secuencia que se seguirá tras el despegue de forma generalizada para recorrer punto a punto la trayectoria es la siguiente:

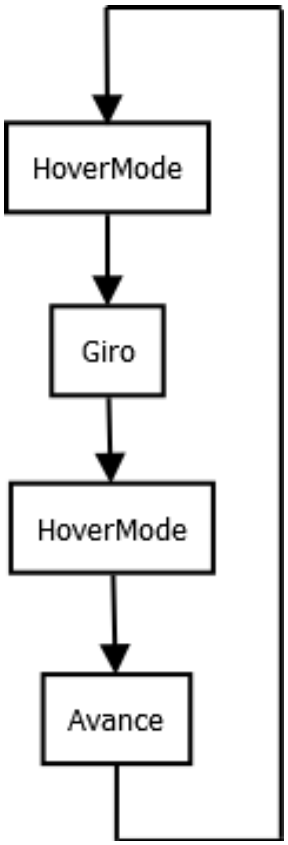


Figura 33. Secuencia del movimiento algoritmo 1

La programación se realiza dentro del bucle de control **while** para este algoritmo 1, que finaliza cuando nuestro programa ha alcanzado el último punto. Mostramos la secuencia para el caso 1, aunque tiene la misma estructura para todos los casos posibles:

```
///Caso 1, entra en la condición y activa el contador
if (x[i] < x[i - 1] && y[i] > y[i - 1])
{
    timerACC1++;
    giro[i] = 90.0f - beta[i];
    labelgiro.Text = giro[i].ToString();
    ///3.14 intervalos de control para realizar el giro
    if (timerACC1 >= 8 && timerACC1 < 11.14)
    {
        LeaveHoverMode();
        dH = 1000;
        dVx = 0.0f; dVy = 0.0f;
        dYaw = giro[i] - diferenciagiro;
        UpdateUIAsync("Girando caso 1...");
        boolpitch = false;
        boolroll = false;
        boolYaw = true;
        tmrTestControl2.Start();
    }
    ///2 segundos detenemos el cuadricóptero
    else if (timerACC1 >= 11.14 && timerACC1 < 16)
    {
        EnterHoverMode();
    }
    ///Sale del modo Hover y nuestro cuadricóptero empieza a avanzar hasta que
        se cumple el tiempo volando calculado en el algoritmo dependiendo de la
        distancia
    else if (timerACC1 >= 16 && timerACC1 < (16 + tiempoflying[i]))
    {
        btnForwardAuto.PerformClick();
    }
    ///Tras llegar al punto destino se detiene el drone y empieza al análisis
        del siguiente punto
    else if (timerACC1 >= (16 + tiempoflying[i]))
    {
        EnterHoverMode();
        goto sumamospunto;
    }
    break;
}
```

4.5.2 Cálculo matemático de las variables empleadas

A continuación detallaremos el cálculo de los parámetros:

- La distancia total que hay de punto a punto en línea recta, calculada en metros.

$$distancia[i] = \sqrt{(x[i] - x[i-1])^2 - (y[i] - y[i-1])^2}$$

- Una vez calculada la distancia, calcularemos el tiempo que tardará en ir de un punto a otro, a velocidad 500 mm/s como hemos dicho anteriormente.

$$velocidad = \frac{espacio}{tiempo} \rightarrow tiempo = \frac{espacio}{velocidad}$$

  El tiempo lo calcularemos en segundos, y lo multiplicaremos por 2 ya que el intervalo de nuestro control es cada 500 ms, de forma que así lo tendremos en segundos cada vez que aumente nuestro contador. La distancia la multiplicamos por 1000 para obtenerla en milímetros y poder dividir por la velocidad, que está en mm/s.

$$tiempoflying[i] = 2\left(\frac{1000.\,distancia[i]}{500}\right)$$

- Para calcular el ángulo existente entre cada punto utilizaremos la siguiente función trigonométrica.

$$beta[i] = \tan^{-1}\left(\frac{x[i] - x[i-1]}{y[i] - y[i-1]}\right)$$

  Lo pasamos a grados, multiplicando por 180 y dividiendo por π, de forma que nuestro ángulo β queda finalmente así:

$$beta[i] = \left(\tan^{-1}\left(\frac{x[i] - x[i-1]}{y[i] - y[i-1]}\right)\right)\left(\frac{180}{\pi}\right)$$

El ángulo beta calculado lo mostramos gráficamente a continuación dependiendo de las condiciones de cada punto, pero siempre será la arcotangente entre X e Y:

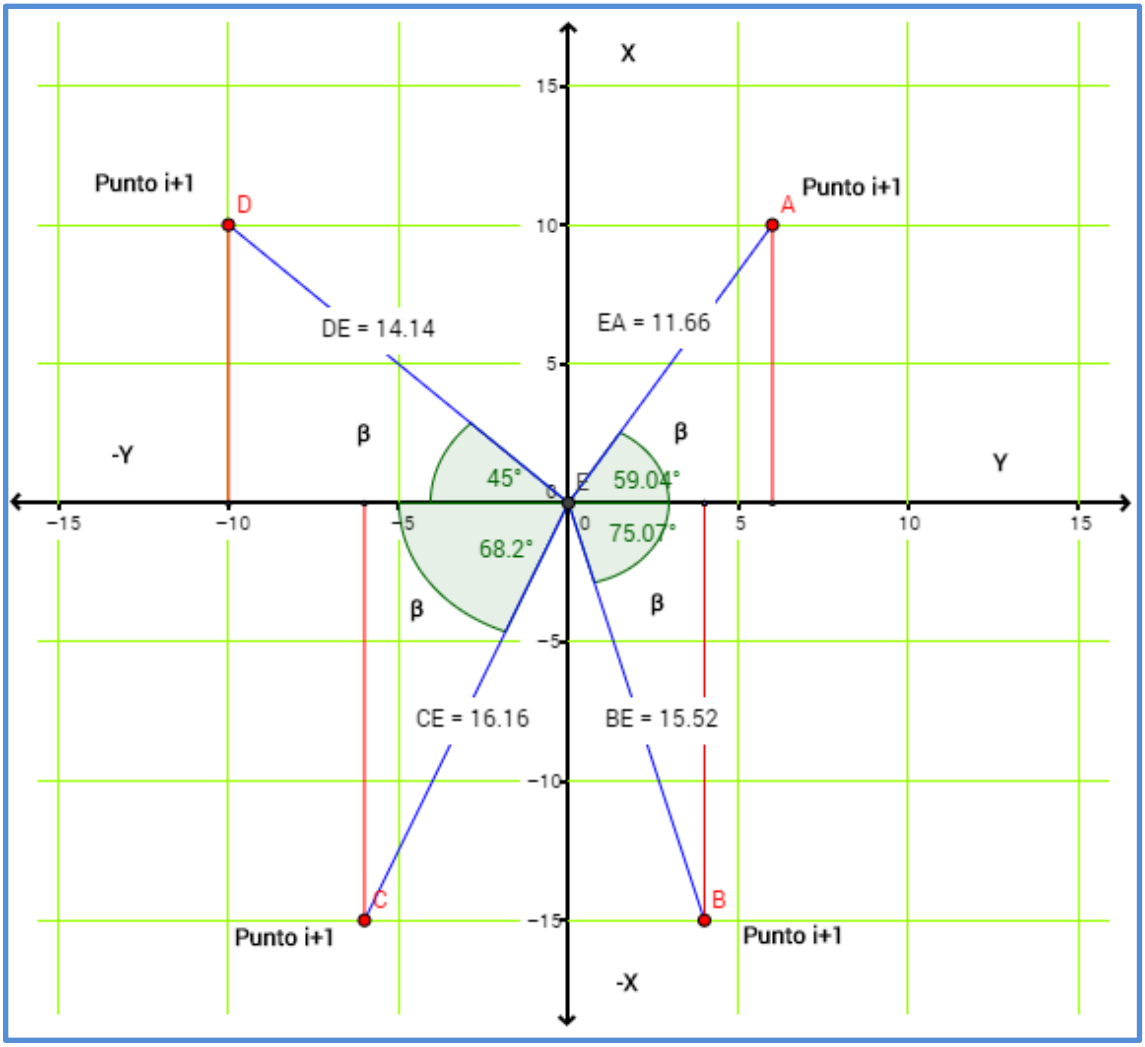


Figura 34. Cálculo ángulo β

4.5.3 Casos posibles. Análisis entre dos puntos

Para configurar el giro como hemos dicho con anterioridad tendremos 9 casos posibles, dependiendo de la posición en que se encuentre el punto siguiente i respecto al anterior i-1, que son los siguientes:

CASO	x[i] respecto a x[i-1]	y[i] respecto a y[i-1]
1	<	>
2	>	>
3	<	<
4	>	<
5	=	=
6	<	=
7	>	=
8	=	<
9	=	>

Tabla 1. Análisis de los puntos en el algoritmo 1

El giro que debe realizar nuestro drone en cada caso es el siguiente, aunque en determinados casos (si un punto x o y es igual al anterior) no tendremos en cuenta el ángulo beta calculado:

- Caso 1: $dYaw = giro[i] = 90 - beta[i]$
- Caso 2: $dYaw = giro[i] = 90 - beta[i]$
- Caso 3: $dYaw = giro[i] = -90 - beta[i]$
- Caso 4: $dYaw = giro[i] = -90 - beta[i]$
- Caso 5: $dYaw = giro[i] = 0$
- Caso 6: $dYaw = giro[i] = +180$

- Caso 7: $dYaw = giro[i] = 0$
- Caso 8: $dYaw = giro[i] = -90$
- Caso 9: $dYaw = giro[i] = +90$

4.6 Algoritmo 2

Se ha calculado otro algoritmo para poder seguir una trayectoria que no sea mueva solamente con la parte delantera del drone, es decir que tenga ángulos pitch y roll conjuntamente. En este nuevo algoritmo nuestro cuadricóptero podrá ir de punto a punto con ángulos de navegación cabeceo (inclinación del morro del cuadricóptero) y alabeo (inclinación del eje ala-ala del cuadricóptero) conjuntamente. En este algoritmo nuestro sistema tendrá movimiento de pitch y roll, ya que en vehículos aéreos no tripulados las posibilidades para ir de punto a punto son muchas.

En este nuevo algoritmo no tendremos que realizar giros de yaw ya que seguiremos los puntos siempre con la orientación inicial en que despega nuestro drone. En este algoritmo el eje morro-cola siempre coincidirá con el eje X. No tiene por qué estar orientado hacia el Norte, sino que en este caso hacemos coincidir el eje X con el Norte que será el morro del drone para comprenderlo.

La configuración de los ejes y su justificación será la misma que en el algoritmo anterior, apartado 4.5:

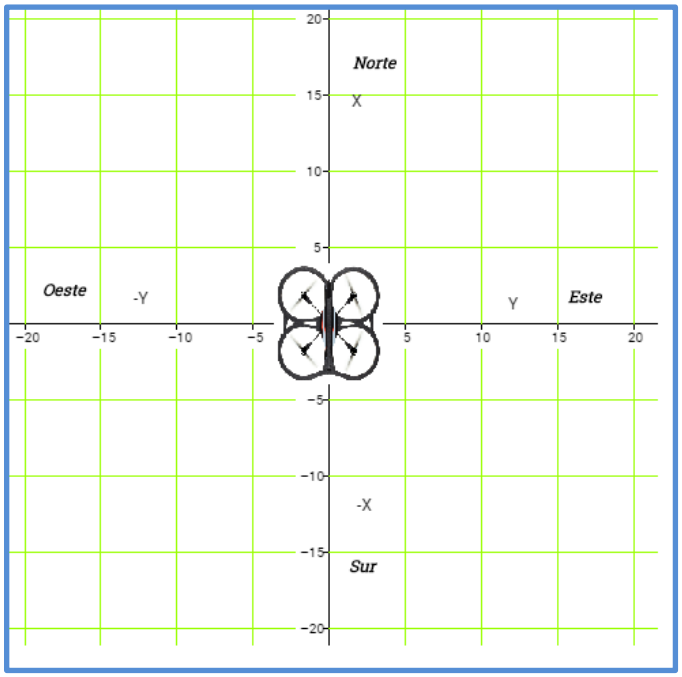


Figura 35. Asignación de ejes algoritmo 2

De forma que un avance en X siempre será desplazarnos hacia el Norte, un avance en –X será desplazarnos hacia el Sur, un avance en Y será desplazarnos hacia el Este y por último un avance en –Y será desplazarnos hacia el Oeste. Estos puntos cardinales no son absolutos, sino que son respecto al drone.

4.6.1 Secuencia del movimiento y código algoritmo 2

La secuencia que se seguirá tras el despegue de forma generalizada para recorrer punto a punto la trayectoria es la siguiente:

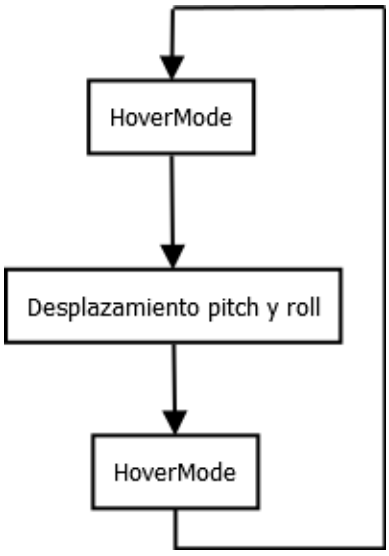


Figura 36. Secuencia del movimiento algoritmo 2

La programación se realiza dentro del bucle de control **while** para este algoritmo 2 también, que finaliza cuando nuestro programa ha alcanzado el último punto del array. Mostramos la secuencia para el caso 1, aunque tiene la misma estructura para todos los casos posibles:

```
///CASO 1
if (x[j] < x[j - 1] && y[j] > y[j - 1])
{
    timerACC1++;
    ///Abandonamos el modo Hover y nos desplazamos hacia el punto con pitch(x) y
        roll (y) dependiendo de las condiciones del punto en que nos encontremos
    if (timerACC1 >= 8 && timerACC1 < (8 + tiempoflying[j]))
    {
        LeaveHoverMode();
        dH = 1000;
        ///Multiplicamos las velocidades por 1000 para obtenerlo en
            milímetros/segundo, en todos los casos
        dVx = -((x[j] - x[j - 1]) * 1000 / tiempoflying[j]);
        dVy = ((y[j] - y[j - 1]) * 1000 / tiempoflying[j]);
        dYaw = 0.0f;
        UpdateUIAsync("Movimiento caso 1...");
        boolpitch = true;
        boolroll = true;
        boolYaw = false;
        tmrTestControl3.Start();
    }
    ///Tras llegar al punto destino se detiene 2 segundos el drone y empieza al
        análisis del siguiente punto
    else if (timerACC1 >= (8 + tiempoflying[j]) && timerACC1 < (12 +
        tiempoflying[j]))
    {
        EnterHoverMode();
        goto sumamospunto3;
    }
    break;
}
```

El cálculo de este algoritmo lo hemos realizado en un nuevo control llamado **tmrTestControl3()**, y recorreremos los arrays de puntos con la variable j. En este nuevo control tendremos que descomponer la velocidad en los ejes X e Y, cuyo módulo hemos establecido que será también de 500 mm/s, es decir nuestro drone irá de punto a punto con una velocidad |v|=500 mm/s.

Las variables que hemos declarado para poder calcular el algoritmo y realizar el control son las siguientes:

- distancia[j]→será el módulo de la distancia entre los dos puntos.
- tiempoflying[j]→al igual que en el algoritmo anterior será el tiempo que tardará en llegar de un punto a otro a una |v|=500 mm/s.
- beta[j]→igual que en el algoritmo anterior es el ángulo que forman dos puntos con respecto al eje Y, aunque en este caso no trabajaremos con este parámetro y sólo lo tendremos como información en nuestra interfaz.

Tenemos que tener en cuenta que una velocidad es la derivada de la distancia respecto al tiempo:

$$x = \int V_x . d_t \rightarrow V_x = \frac{d_x}{d_t} \rightarrow V_x = \frac{\Delta x}{\Delta t}$$

$$y = \int V_y . d_t \rightarrow V_y = \frac{d_y}{d_t} \rightarrow V_y = \frac{\Delta y}{\Delta t}$$

4.6.2 Cálculo matemático de las variables empleadas

A continuación detallamos y explicamos el cálculo del algoritmo:

- En primer lugar calcularemos la distancia entre los dos puntos por Pitágoras de la siguiente forma:

$$distancia[j] = \sqrt{(x[j] - x[j-1])^2 - (y[j] - y[j-1])^2}$$

- El tiempo que tardará en ir de un punto a otro lo calculamos de la misma manera que en el algoritmo anterior en el apartado 4.5.2, multiplicando por 2 ya que el control es cada 500 ms y multiplicando por 1000 para pasar las unidades a mm y no tener errores.

$$tiempoflying[j] = 2\left(\frac{1000. distancia[j]}{500}\right)$$

- El ángulo beta[j] que tendremos como información, para saber hacia dónde nos desplazamos, se calcula también exactamente igual que en el caso anterior.

$$beta[j] = \left(\tan^{-1}\left(\frac{x[j] - x[j-1]}{y[j] - y[j-1]}\right)\right)\left(\frac{180}{\pi}\right)$$

- Por último calcularemos la velocidad en X e Y que tendremos que aplicar a nuestro vuelo, teniendo en cuenta que habrá movimientos pitch y roll, relacionado con los ángulos de navegación.

$$V_x = \frac{\Delta x}{\Delta t} \rightarrow V_x = \frac{(x[j] - x[j-1])}{tiempoflying[j]}$$

$$V_y = \frac{\Delta y}{\Delta t} \rightarrow V_y = \frac{(y[j]-y[j-1])}{tiempoflying[j]}$$

Podemos comprobar que el cálculo es correcto calculando el módulo de la velocidad descompuesta en los ejes de la siguiente manera:

$$|V| = \sqrt{v_x^2 + v_y^2}$$

La configuración de las velocidades la exponemos a continuación, teniendo en cuenta que como hemos detallado con anterioridad tendremos movimientos de roll y pitch, y que los puntos cardinales son solo una orientación en este caso:

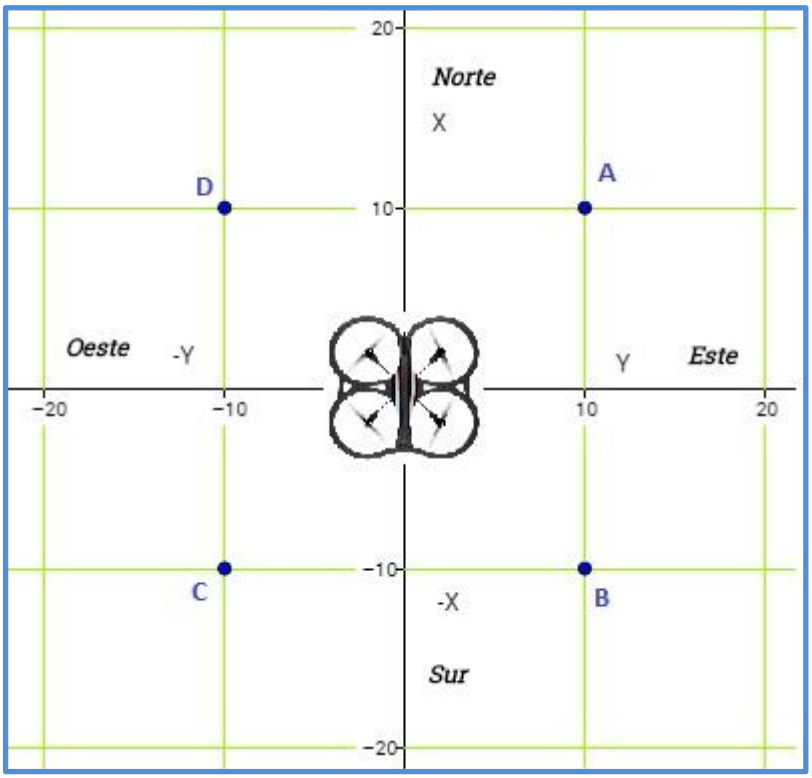


Figura 37. Análisis de velocidades en los ejes algoritmo 2

- Desplazamiento hacia el punto A:
  Vx→negativa
  Vy→positiva

- Desplazamiento hacia el punto B:
  Vx→positiva
  Vy→positiva

- Desplazamiento hacia el punto C:
  Vx→positiva
  Vy→negativa

- Desplazamiento hacia el punto D:
  Vx→negativa
  Vy→negativa

4.6.3 Casos posibles. Análisis entre dos puntos

A continuación explicamos todos los casos posibles que podremos encontrarnos al ir de punto a punto, ya que dependiendo de las condiciones de estos en algunos casos no tendremos movimiento de roll o de pitch.

CASO	x[j] respecto a x[j-1]	y[j] respecto a y[j-1]	pitch	roll	Vx	Vy
1	<	>	true	true	+	+
2	>	>	true	true	-	+
3	<	<	true	true	+	-
4	>	<	true	true	-	-
5	=	=	false	false	0	0
6	<	=	true	false	+	0
7	>	=	true	false	-	0
8	=	<	false	true	0	-
9	=	>	false	true	0	+

Tabla 2. Análisis de puntos y velocidades en el algoritmo 2

4.7 Función Navigate( )

Esta es la función más importante de todo el software ya que es la que le ordena al cuadricóptero moverse dependiendo de unos parámetros que se le introducen, teniendo en cuenta las variables utilizadas para calcular los algoritmos. Esta función es llamada siempre en el control, que en nuestro caso se le llamará tanto en el control manual, en el control del algoritmo 1 y en el control del algoritmo 2.

Se caracteriza por los siguientes 4 parámetros:

Navigate (Controlroll, Controlpitch, Controlyaw, Controlgaz);

Cada vez que esta función es llamada en el control varía el comportamiento de nuestro cuadricóptero y su movimiento debido a los ángulos de navegación. Además tenemos los operadores lógicos booleano para programar distintos modos de vuelo. Estos operadores con los que trabajamos, los cuáles asignaremos a true o false dependiendo si queremos ese determinado movimiento o no, son:

- boolroll
- boolpitch
- boolyaw

Los valores del control de estos ángulos dentro de la función Navigate tienen como valores máximos un rango entre (-1,1), aunque generalmente estarán limitados en nuestros algoritmos entre (-0.05, 0.05) tanto Controlroll como Controlpitch. Son valores de tipo float, y sus unidades son en radianes. El objetivo y misión de esta función Navigate() es relacionar las entradas angulares pitch y roll con la respuesta esperada de las velocidades en nuestro plano XY. En el caso de estar asignados a 0 no tendrá ese determinado movimiento o inclinación.

A continuación se detallará el funcionamiento de cada uno de los controles de nuestro sistema, que como hemos comentado anteriormente entra al control con un intervalo de 500 milisegundos.

4.7.1 Controlador proporcional P

Tenemos que tener en cuenta que nuestro sistema estará regulado por un controlador proporcional P. En este tipo de controladores la señal de error se amplifica antes de aplicarla a la planta o proceso de nuestro sistema en lazo abierto. Para amplificar la señal utilizamos una constante de proporcionalidad Kp. En nuestro sistema tendremos 4 controles que regular los cuáles son los pertenecientes a la función Navigate(), que es la encargada de mover y actuar nuestro cuadricóptero. Estos controles son Controlroll, Controlpitch, Controlyaw y Controlgaz, y cada uno tendrá una constante de proporcionalidad. Dichas constantes se han establecido

con los siguientes valores de acuerdo a la respuesta esperada de nuestro sistema y que mejor lo hace funcionar:

$KPPitch = 0.0005\text{f};$ $KPRoll = 0.0005\text{f};$ $KPH = 0.0005\text{f};$ $KPYaw = 0.005555\text{f};$

Estos valores se han calculado a partir de la salida deseada Y(s), con respecto a la señal de error o entrada al controlador E(s). En estos controladores la señal de salida es directamente proporcional a la desviación. La función de transferencia de un controlador proporcional P es la siguiente:

$Y(s) = K_P . E(s)$, donde Y(s) es la señal de control (roll, pitch, yaw, gaz).

$K_P = \frac{Y(s)}{E(s)}$, la ganancia que debemos aplicar al controlador para obtener la respuesta deseada.

Tenemos que tener en cuenta que nuestro controlador va a tener un cierto error siempre debido al tiempo de respuesta que no se puede compensar. Este error será acentuado para acciones en las que los intervalos de tiempo sean cortos, es decir en acciones en que la señal de entrada sea rápida. Presentará una trayectoria exponencial hasta alcanzar el valor o salida deseada. Ocurre en general en los controladores proporcionales.

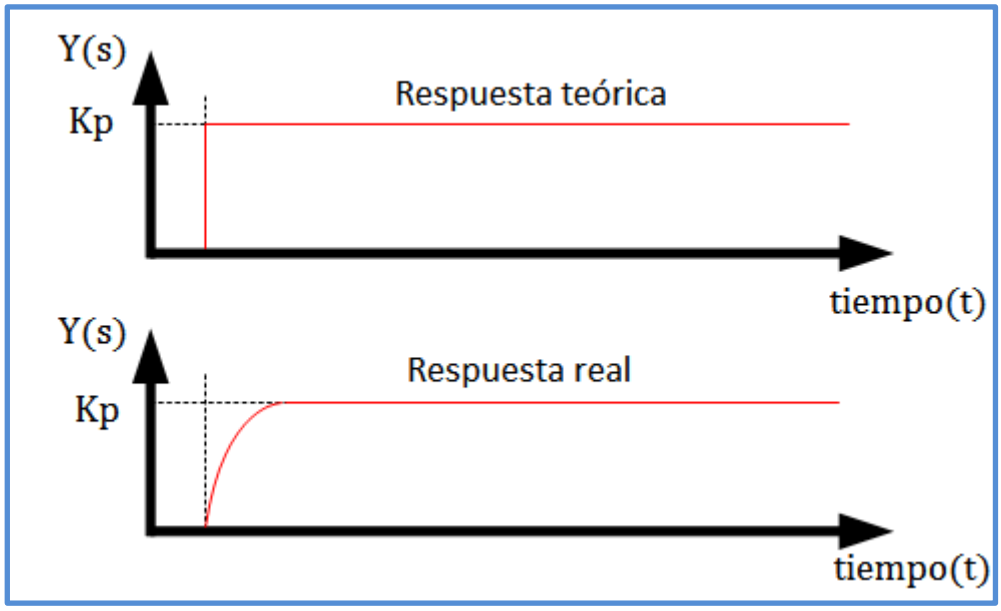


Figura 38. Tiempo de respuesta controlador P

4.7.2 Función Navigate( ) tmrTestControl

Para el movimiento manual nuestro sistema calculará primeramente el control en pitch y roll, que dependerán del botón que se haya pulsado en cada momento. Este será igual en los 3 controles y es la diferencia de velocidades en X e Y respectivamente multiplicadas por la constante de proporcionalidad:

$$Controlpitch = KPPitch\ x\ (dVx - clsARdroneGeneralParams.clsVx)$$

$$Controlroll = KPRoll\ x\ (dVy - clsARdroneGeneralParams.clsVy)$$

Para poder controlar la altura del cuadricóptero utilizaremos Controlgaz, el cual se controla igual que los anteriores con un controlador proporcional P de la siguiente manera:

$$Controlgaz = KPH\ x\ (dH - clsARdroneGeneralParams.clsintAltitude)$$

Lo siguiente será controlar las rotaciones alrededor del eje vertical perpendicular al drone o giros mediante Controlyaw, que definimos como:

$$Yawdiff = (dYaw - clsARdroneGeneralParams.clsYaw\ )$$

$$Controlyaw = (KPYaw\ x\ Yawdiff)$$

Los valores $clsARdroneGeneralParams$ son los valores que tiene el cuadricóptero en ese momento en el IMU, de forma que calculamos una diferencia entre el valor actual y el valor de entrada.

Los valores de las constantes de nuestro controlador proporcional P que son de tipo float, serán las siguientes, sabiendo que si las cambiamos alteraremos y modificaremos el funcionamiento de nuestro sistema:

$$KPPitch = 0.0005\text{f};\quad KPRoll = 0.0005\text{f};\quad KPH = 0.0005\text{f};\quad KPYaw = 0.005555\text{f};$$

Una vez calculados los valores anteriores procedemos a analizarlos y a limitar los valores para que se adecúen al comportamiento que deseamos:

- Controlpitch: para el caso en que el operador lógico este asignado a verdadero los límites estarán truncados entre -0.05 y +0.05. Sin embargo si el booleano es falso los límites estarán entre -0.02 y +0.02.

```
if (boolpitch == true)
{
    if (Controlpitch > 0.05f) { Controlpitch = 0.05f; }
    if (Controlpitch < -0.05f) { Controlpitch = -0.05f; }
}
if (boolpitch == false)
{
    if (Controlpitch > 0.02f) { Controlpitch = 0.02f;}
    if (Controlpitch < -0.02f) { Controlpitch = -0.02f;}
}
```

- Controlroll: para el movimiento de alabeo o roll, el control es exactamente igual que en pitch y tiene los mismos límites de acuerdo al operador lógico.
- Controlgaz: el encargado de mantener la altura del cuadricóptero tendrá los límites entre -0.9 y +0.9, dependiendo de los parámetros utilizados para su control.
- Controlyaw: para controlar la dirección a la que apunta o el giro de nuestro cuadricóptero utilizamos este control. La variable Yawdiff comentada anteriormente es la diferencia entre el giro que deseamos y la orientación que tiene el cuadricóptero en ese momento en el giróscopo.

  Si el giro que queremos realizar es menor que 0º le sumamos 360º, y lo siguiente será calcular Yawdiff. Tras esto, analizamos Yawdiff que es el giro total que debe hacer nuestro cuadricóptero, de manera que siempre girará hacia la derecha (CW) o izquierda (CCW) por donde tenga que recorrer menos grados, es decir por el camino más corto y rápido. La programación del control es la siguiente:

```
if (dYaw < 0)
{
      dYaw = dYaw + 360;
}
Yawdiff = (dYaw - float)(clsARDroneGeneralParams.clsYaw));
if (Yawdiff < -180)
{
      Yawdiff = Yawdiff + 360;
}
if (Yawdiff <= 180)
{
     Controlyaw = (float)Math.Round((KPYaw * Yawdiff), 5);
}
if (Yawdiff > 180)
{
     Yawdiff = -(360 - Yawdiff);
     Controlyaw = (float)Math.Round((KPYaw * Yawdiff), 5);
}
```

Una vez obtenido el valor de cada variable de control, llamamos a la función Navigate que es la encargada de mover el cuadricóptero dependiendo de sus valores y el análisis realizado con anterioridad.

Navigate(Controlroll, Controlpitch, Controlyaw, Controlgaz);

Por último se ha establecido como se ha comentado anteriormente que cada orden tendrá una duración de 3 segundos, es decir 6 pulsos de control debido al intervalo que es de 500 ms. También en ese tiempo se puede pulsar otro botón y volver a empezar el contador, quedando anulada la orden anterior. Tras esos 3 segundos el drone entra en modo flotante (HoverMode) y se detiene en el aire. Para esto utilizamos el contador declarado **tmrcountr_manual** de tipo entero comentado anteriormente.

4.7.3 Función Navigate( ) tmrTestControl2

Para el algoritmo 1, en primer lugar nuestro sistema calcula el control en pitch y roll, los cuáles vienen definidos por las velocidades en x e y respectivamente multiplicados por su constante proporcional como indicamos a continuación:

$$Controlpitch = KPPitch\ x\ (dVx - clsARdroneGeneralParams.clsVx)$$

$$Controlroll = KPRoll\ x\ (dVy - clsARdroneGeneralParams.clsVy)$$

Para controlar la altitud de nuestro cuadricóptero utilizamos el Controlgaz, el cual se calcula:

$$Controlgaz = KPH\ x\ (dH - clsARdroneGeneralParams.clsintAltitude)$$

El control de los giros se realiza de la siguiente forma, aunque este lo explicamos detalladamente en el apartado siguiente 4.8:

$$Controlyaw = (dYaw\ x\ KPYaw)$$

Los valores de las constantes de nuestro controlador proporcional P de tipo float, serán las siguientes, sabiendo que si las cambiamos alteraremos y modificaremos el funcionamiento de nuestro sistema:

$$KPPitch = 0.0005\text{f};\quad KPRoll = 0.0005\text{f};\quad KPH = 0.0005\text{f};\quad KPYaw = 0.005555\text{f};$$

Tras estos cálculos analizamos sus valores:

- Controlpitch: si el booleano de pitch es verdadero, asignamos un Controlyaw igual a cero, esto es para que mientras el drone este avanzando no realice giros y tenga el menor error posible. Además en este caso (boolpitch=true) los valores estarán limitados entre -0.05 y +0.05 radianes. Sin embargo si es booleano es falso los valores los limitaremos entre -0.02 y +0.02, es decir su movimiento será mínimo en pitch.

- Controlroll: en este caso el control es similar al control de pitch si el booleano es verdadero lo limitamos entre -0.05 y +0.05, y si es falso entre -0.02 y +0.02 radianes. Como en este algoritmo no vamos a desplazarnos con roll en ningún caso lo asignaremos a 0 durante todo el control.

- Controlgaz: para controlar la altura los límites de este control estarán entre -0.9 y +0.9 radianes. Para este caso no tenemos operador lógico ya que el control de la altura siempre estará calculándose dependiendo de los parámetros que tengamos en cada momento. Estará asignado a 0 a lo largo de toda la trayectoria para que no varíe su altitud.

- Controlyaw: en este algoritmo 1, este control de giro es el más importante. Empezamos a analizar los giros negativos a los cuáles le sumaremos 360º hasta que ese valor sea positivo:

```
if (dYaw < 0)
{
    dYaw = dYaw + 360;
}
```

Si el giro es menor o igual a 180º(caso normal), se calcula el control multiplicando por la proporcional como hemos detallado con anterioridad:

```
if (dYaw <= 180)
{
      Controlyaw = (float)Math.Round((KPYaw *
      dYaw), 5);
}
```

Si el giro tras el análisis anterior fuera mayor que 180º, transformamos ese ángulo. Esto se hace para que si el giro a realizar es mayor que 180º el cuadricóptero realice el giro por donde menos grados tenga que recorrer, es decir en sentido contrario a las agujas del reloj (CCW).

```
if (dYaw > 180)
{
      dYaw = -(360 - dYaw);
      Controlyaw = (float)Math.Round((KPYaw *
      dYaw), 5);
}
```

Una vez calculados los 4 controles de cada movimiento, estos valores entran a la función Navigate y comienza el movimiento:

Navigate(Controlroll, Controlpitch, Controlyaw, Controlgaz);

El análisis de la trayectoria a seguir vendrá detallado con diagramas de flujo en el apartado 4.9.1 del presente capítulo.

4.7.4 Función Navigate( ) tmrTestControl3

El control para este algoritmo en el que nuestro cuadricóptero tendrá movimiento de cabeceo (pitch) y alabeo (roll) conjuntamente es igual que el del apartado 4.7.2 con la única diferencia que nuestro Controlyaw va a estar asignado a cero siempre, dado que en este control el drone no va a realizar giros.

El análisis de la trayectoria a seguir vendrá detallado con diagramas de flujo en el apartado 4.9.2 del presente capítulo.

4.8 Configuración de los giros y ajuste del tiempo algoritmo 1

Para realizar los giros en el algoritmo 1, el cual está programado en el **tmrtestControl2** como hemos dicho anteriormente, tendremos que modificar el Controlyaw que será llamado por la función Navigate.

El valor que le entrará a la función será una velocidad angular que tendrá unidades de (rad/intervalo de control). Nuestro intervalo de control será 500 ms, que es el tiempo que transcurre para entrar nuevamente nuestro sistema al control. Estas unidades las pasamos a (º/intervalo de control) con la siguiente conversión para ver los grados que gira en un segundo:

$$\omega\left(\frac{º}{intervalo}\right) = \omega\left(\frac{rad}{intervalo}\right) x \frac{360}{2\pi}$$

1 rad = 57.2957 º

Para nuestro caso crítico que son giros de 180º lo máximo que debe girar, hemos observado que girando a mayor velocidad el drone tiene un mejor comportamiento. Esto es debido a que cuanto más tiempo está el cuadricóptero fuera del modo Hover el error es mayor y el balanceo del cuadricóptero se acentúa, por lo que intentamos reducirlo asignando un valor de Controlyaw=1, es decir a 57.29 (º/intervalo) . Para conseguir esto tendremos que modificar la constante de proporcionalidad KPYaw. Al hacer el ajuste, la ganancia de nuestro controlador

tendrá un valor de 0.00555f, la cuál será constante para cualquier giro tras haber comprobado que se comporta correctamente para todos los ángulos que queramos girar independientemente del caso en el que nos encontremos. Sabemos que nuestro giro en cada caso va a ser dYaw (en unidades de grados º), que es la variable que va a entrar al control.

$$Controlyaw = (dYaw\ x\ KPYaw)$$

Con esto podemos calcular el tiempo que necesita para girar, el cuál va a ser constante para todos los giros. A continuación exponemos la demostración:

$$Controlyaw = (dYaw\ x\ KPYaw) = 180\ x\ 0.00555 = 0.99\ \frac{rad}{intervalo} = 57.2957\ \frac{º}{intervalo}$$

$$tiempo = \frac{dYaw}{Controlyaw} = \frac{180\ º}{57.2957\ º/intervalo} = 3.14\ intervalos$$

$$3.14\ intervalos.\frac{0.5\ segundos}{1\ intervalo} = 1.57\ segundos$$

Ese tiempo es el que va a tardar en realizar cualquier giro dentro de nuestro algoritmo, de modo que al programar nuestra secuencia de control con los contadores queda asignado a ese valor constante, 3.14 intervalos o pulsos de control. Podemos asegurar que para giros menores nuestro cuadricóptero se comporta correctamente a menor velocidad angular, y siempre con este tiempo calculado.

El ángulo theta que nos muestra la orientación del drone gracias al giroscopio es un parámetro general de nuestro drone, el cual reconocemos como **clsARDroneGeneralParams.clsYaw,** y podemos acceder a él para modificar sus propiedades u operar con él. En nuestra interfaz podemos identificarlo con la siguiente imagen, de forma que siempre sabremos hacia qué punto cardinal apunta nuestro drone en cada instante.

Figura 39. Giroscopio interfaz

4.9 Implementación del software en C#

Como se ha comentado anteriormente el trabajo se ha realizado en el lenguaje de programación C# corriendo en Windows, que es muy similar a Java. Este lenguaje está orientado a objetos desarrollado por Microsoft como parte de su plataforma .NET, y su sintaxis deriva C y C++. Su creador fue Anders Heljsberg, y su idea principal es combinar la potencia de lenguajes como C++ con la sencillez de otros como Visual Basic.

En la interfaz de nuestra aplicación tenemos una consola que nos indica el movimiento que está realizando nuestro cuadricóptero en todo momento; estos mensajes se indican en la programación de la secuencia de control con el método **UpdateUIAsync(**"Mensaje"**)**.

Aparte también se ha creado una clase **GroupBox** donde se indica mediante leyendas o etiquetas los valores de las variables necesarias calculadas para los algoritmos, de forma que nos ayuda a conocer el estado y el movimiento que debe de hacer el drone. En esta tendremos información como el tiempo que necesita para desplazarse de punto a punto, la distancia entre dos puntos, el giro a realizar actual y anterior, y el valor de la variable (i o j) con que recorremos nuestro array de puntos. Para ello utilizamos la instrucción siguiente **NameLabel.Text = Variable.ToString().** Podremos modificar todos los parámetros y propiedades de dichas etiquetas en la interfaz sin compilar.

Por último tendremos en nuestra interfaz también mediante etiquetas los puntos en X,Y que se cargarán al sistema y que deberá seguir nuestra trayectoria.

A continuación se muestra una imagen capturada de nuestra interfaz en funcionamiento con los datos anteriores:

Figura 40. Estado movimiento, variables y puntos de los algoritmos

4.9.1 Diagrama del algoritmo 1 calculado

En este apartado se mostrará el diagrama de flujo de nuestro algoritmo 1 desarrollado para seguir una trayectoria. El código de programación completo de este control vendrá detallado en el anexo V de esta memoria.

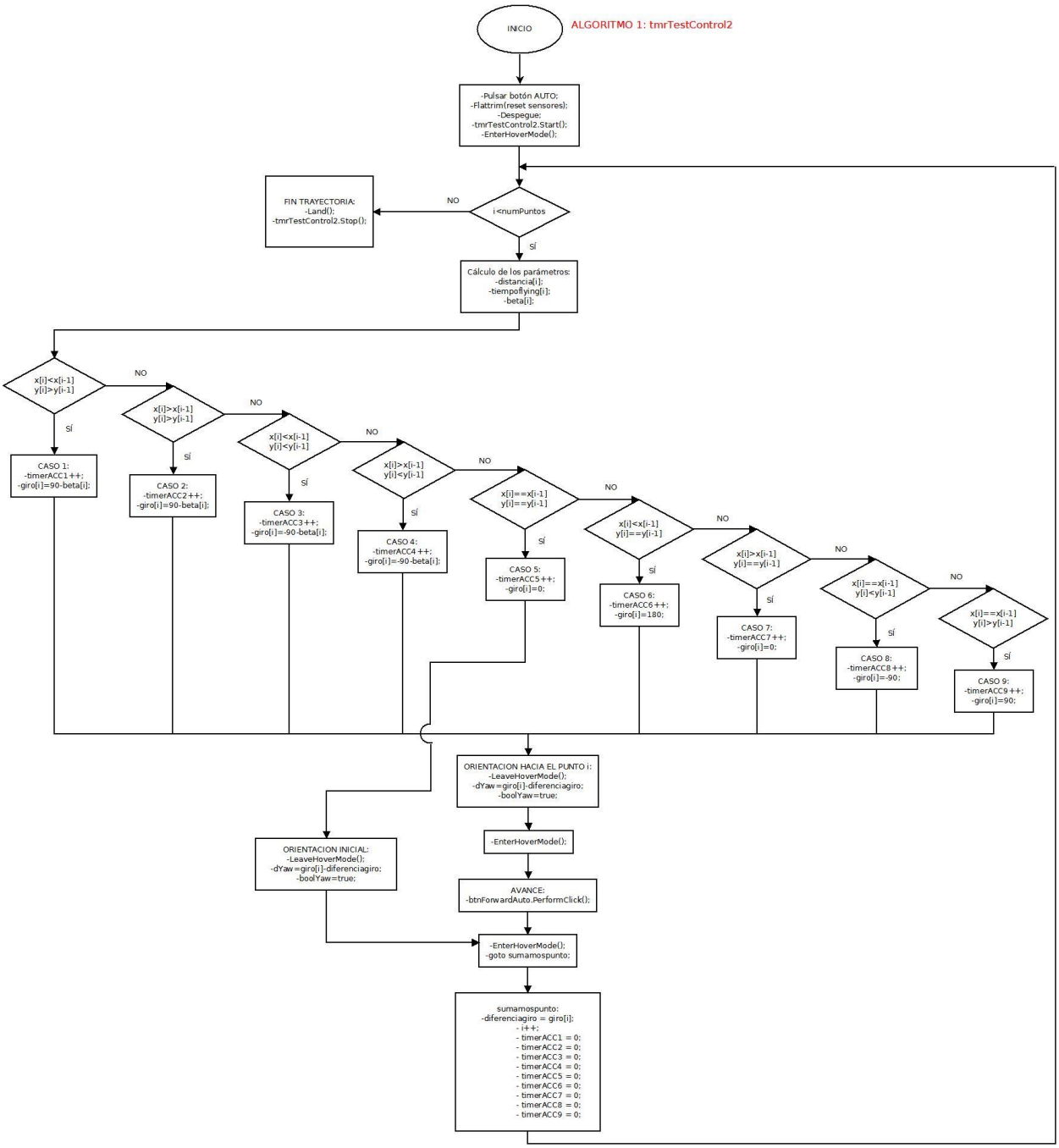


Figura 41. Diagrama de flujo algoritmo 1

4.9.2 Diagrama del algoritmo 2 calculado

En este apartado se mostrará el diagrama de flujo de nuestro algoritmo 2 desarrollado para seguir una trayectoria. El código de programación completo de este control vendrá detallado en el anexo VI de esta memoria.

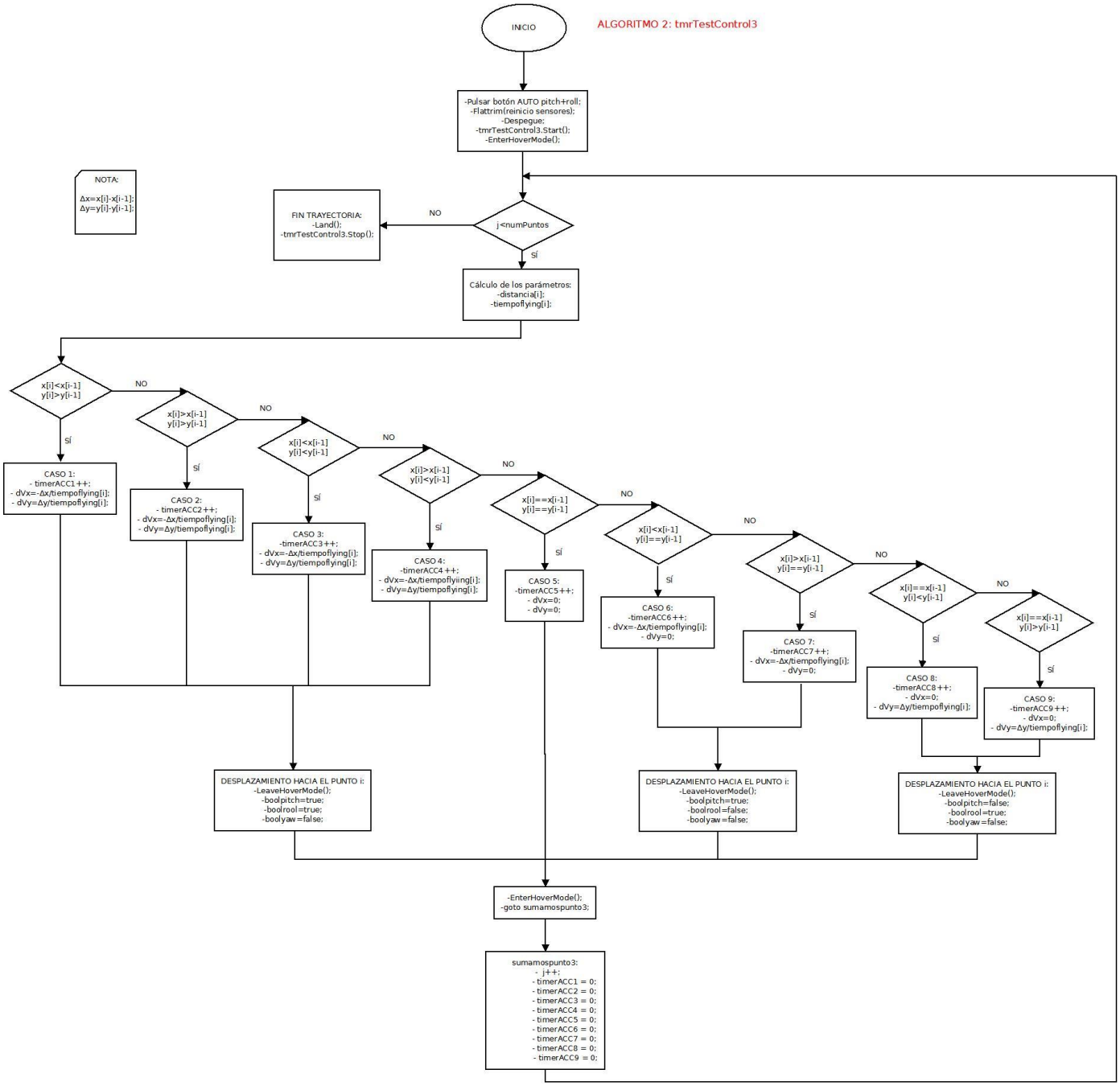


Figura 42. Diagrama de flujo algoritmo 2

Capítulo 5: Análisis de resultados

En este capítulo vamos a mostrar los resultados obtenidos al simular el software desarrollado a lo largo del proyecto y analizaremos también los errores en cada trayectoria. Las trayectorias de puntos se han elegido al azar, aumentando en cada una de ellas la complejidad del movimiento y el número de puntos a seguir, teniendo en cuenta que cuanto mayor sea el número de puntos cargados mayor será el error acumulado en nuestro vuelo.

También se van a exponer en este capítulo los errores obtenidos en instrucciones básicas del vuelo manual, el cual está desarrollado en el control manual, **tmrTestControl**.

En cada trayectoria se realizará un vídeo, grabado con una cámara externa, para ver si el vuelo ha pasado por los puntos de manera correcta y poder calcular el desvío.

Deberemos tener en cuenta que nuestro primer punto siempre va a ser el (0,0), que es el que tenemos nada más despegar el drone verticalmente a una altura constante de 1000 mm a lo largo de la trayectoria. El último punto de regreso también será el (0,0) que es donde se calculará el desvío que ha tenido en la trayectoria.

En total se van a realizar seis trayectorias en lugares cerrados como los pasillos de la Universidad o salón de casa donde hay una contaminación de redes Wi-Fi.

Cabe destacar que todas las pruebas realizadas se han realizado con la batería del AR Drone 2.0 a la mayor carga posible y nunca por debajo del 40%, para así evitar errores derivados en su respuesta.

Los vídeos realizados con las trayectorias para los algoritmos 1(**tmrTestControl2**) y 2(**tmrTestControl3**) realizadas tanto en interior como en exterior se subirán al canal de Youtube con la siguiente dirección y serán analizados en el presente capítulo 5: https://www.youtube.com/channel/UCvLPsv2sbaeqY7nhzZGXcpw .

5.1 Errores vuelo manual

Estos datos se han calculado tras haber realizado muchas pruebas en nuestro control y haber obtenido diferentes conclusiones para el modelo final. Se obtuvieron datos finales como el intervalo de tiempo de nuestro control que finalmente se determinó que nuestro sistema entrara al control cada 500 ms, y se fijó una velocidad de desplazamiento de 500 mm/s. De esta forma nuestro sistema era lo más estable posible y tenía menos error.

Para calcular los errores relativo y absoluto en cada medida realizada utilizaremos las siguientes fórmulas:

- **Error absoluto**: es una desviación entre el valor medido y el valor que se toma como real, en valor absoluto dicha diferencia.

$$\varepsilon_{abs} = |Valor_{medido} - Valor_{real}|$$

- **Error relativo**: es el cociente entre el error absoluto y el valor real o teórico de la medida en cuestión. Este nos da una idea más exacta de la precisión a la hora de comparar dos o más medidas. Se suele dar en tanto por ciento %.

$$\varepsilon_r = \frac{\varepsilon_{abs}}{Valor_{real}} = \frac{|Valor_{medido} - Valor_{real}|}{Valor_{real}}$$

$$\varepsilon_r(\%) = \frac{|Valor_{medido} - Valor_{real}|}{Valor_{real}} x100$$

5.1.1 Despegue:

Valor teórico	Valor medido 1	Error 1		Valor medido 2	Error 2	
0.5 metros	0.44 metros	0.06	12.00%	0.46 metros	0.04	8.00%
1 metro	0.95 metros	0.05	5.00%	0.92 metros	0.08	8.00%
1.5 metros	1.33 metros	0.17	11.33%	1.41 metros	0.09	6.00%
2 metros	1.86 metros	0.14	7.00%	1.88 metros	0.12	6.00%

Tabla 3. Medidas error al despegar

Error calculado = 7.91%

5.1.2 Desplazamiento eje X: pitch

Valor teórico	Valor medido 1	Error 1		Valor medido 2	Error 2	
1.5 metros	1.38 metros	0.12	8 %	1.42 metros	0.08	5.33%
2 metros	1.78 metros	0.22	11 %	1.84 metros	0.16	8 %
3 metro	2.92 metros	0.08	2.66%	2.95 metros	0.05	1.66%
-1 metros	-0.77 metros	0.23	23 %	-0.7 metros	0.3	30 %

Tabla 4. Medidas error al movernos en X

Como podemos observar cuanto mayor son las distancias cometemos menos error, esto es debido al tiempo de respuesta que necesita el cuadricóptero para posicionarse adecuadamente y comenzar a avanzar, de manera que a menor distancia y tiempo el error será mayor.

Error calculado = 9.8%

5.1.3 Desplazamiento eje Y: roll

Valor teórico	Valor medido 1	Error 1		Valor medido 2	Error 2	
1.5 metros dcha.	1.48 metros	0.02	1.33%	1.44 metros	0.06	4 %
3 metro dcha.	2.85 metros	0.15	5 %	2.9 metros	0.1	3.33%
-1 metros izqda.	-1.12 metros	0.12	12 %	-1.1 metros	0.1	10%
-2 metros izqda.	-1.85 metros	0.15	7.5 %	-1.9 metros	0.1	5 %

Tabla 5. Medidas error al movernos en Y

Error calculado = 6.02%

5.1.4 Rotación alrededor del eje Z: yaw

Valor teórico	Valor medido 1	Error 1		Valor medido 2	Error 2	
+45º	45.6º	0.6	1.33%	41º	4	8.88%
+90º	91º	1	1.11%	89º	1	1.11%
+135º	130º	5	3.7%	128º	7	5.18%
+180º	174º	6	3.33%	182º	2	1.11%
-45º	-46º	1	2.22%	-43º	2	4.44%
-135º	-142º	7	5.18%	-140º	5	3.70%
-180º	-182º	2	1.11%	-177º	3	1.66%

Tabla 6. Medidas error al rotar alrededor de Z

Error calculado = 3.14%

5.1.5 Ascenso/Descenso:

Valor teórico	Valor medido 1	Error 1		Valor medido 2	Error 2	
+0.5 metros	+0.43 metros	0.07	14%	+0.50 metros	0	0%
+1 metro	+1.10 metro	0.1	10%	+0.95 metros	0.05	5%
-1.5 metros	-1.40 metros	0.1	6.66%	-1.51 metros	0.01	0.66%
-2 metros	-2.15 metros	0.15	7.5%	-1.95 metros	0.05	2.5%

Tabla 7. Medidas error al ascender/descender

Error calculado = 5.79%

Los errores de este apartado 5.1 se han medido en el salón de casa, por eso las medidas a lo largo del eje x (movimiento con pitch) e y (movimiento con roll) nunca sobrepasan los 3 metros. Debido a esto nos encontramos con un error mayor. Se puede asegurar que a lo largo del desarrollo del proyecto se realizaron diferentes pruebas en los pasillos de la Universidad con medidas de hasta 14 metros, donde el error era muy bajo. Debido a esto se ajustaron diferentes parámetros finales como válidos, por ejemplo el intervalo de control que se estableció a 500 ms, o la velocidad que se determinó a 500 mm/s entre otros.

5.2 Trayectorias realizadas en interior

A continuación analizaremos las trayectorias realizadas con vuelos en interior. En las gráficas que se muestran el punto inicial P0 coincidirá con el punto final, que es donde mediremos el error y lo que se ha desviado tanto en X como en Y.

5.2.1 Trayectoria 1: (0,0), (8,0), (0,0) en interior

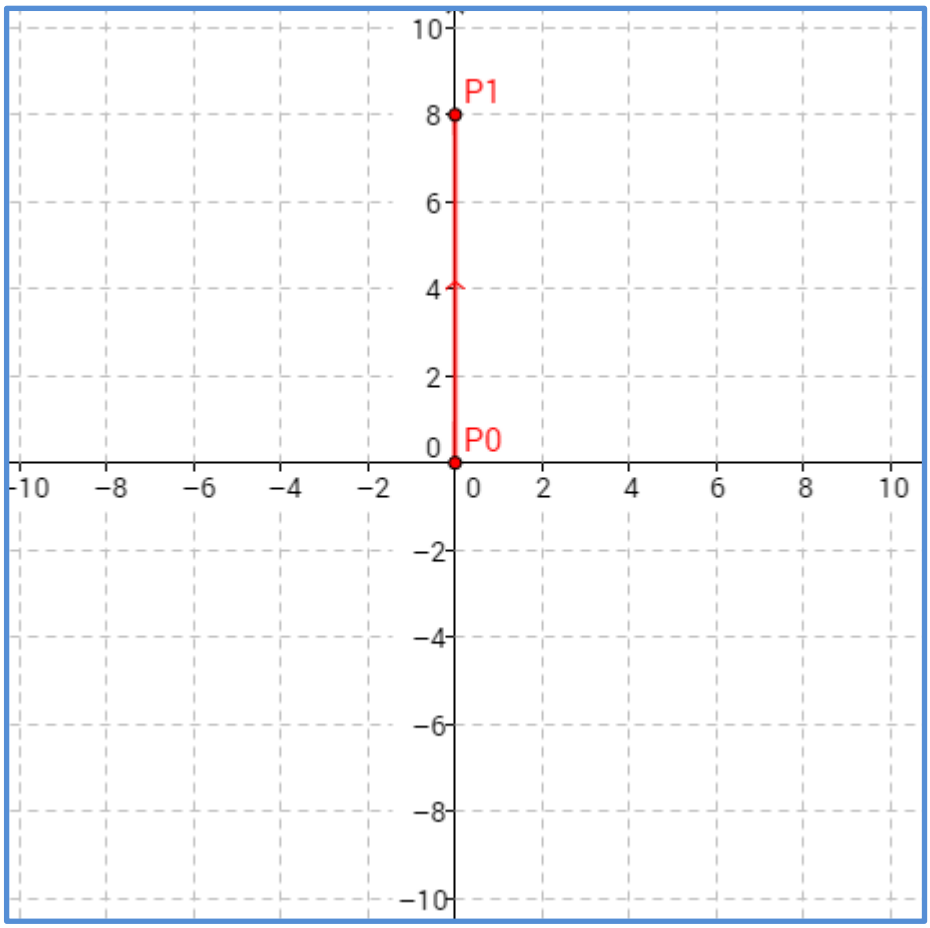


Figura 43. Trayectoria 1 en interior

5.2.1.1 *Algoritmo de control 1: tmrtestControl2*

Desplazamiento i→i+1	CASO (pág.52)	Punto deseado(x,y)	Navigate()				Punto destino medido(x,y)
			roll	pitch	yaw	gaz	
1→2	**7**	(0,0)→(8,0)	0	-0.05	0	0	-
2→3	**6**	(8,0)→(0,0)	0	-0.05	0.9999	0	**(-0.3,1)**

Tabla 8. Medidas trayectoria 1, algoritmo 1 en interior

Error calculado en X= 0.3 metros; Error calculado en Y= 1 metros;

http://youtu.be/7DrTnniMIWI

5.2.1.2 *Algoritmo de control 2: tmrTestControl3*

Desplazamiento i→i+1	CASO (pág.58)	Punto deseado(x,y)	Navigate()				Punto destino medido(x,y)
			roll	pitch	yaw	gaz	
1→2	**7**	(0,0)→(8,0)	0	-0.05	0	0	-
2→3	**6**	(8,0)→(0,0)	0	0.05	0	0	**(-0.6,-0.4)**

Tabla 9. Medidas trayectoria 1, algoritmo 2 en interior

Error calculado en X= 0.6 metros; Error calculado en Y= 0.4 metros;

http://youtu.be/AYwo9HlyYGo

5.2.2 Trayectoria 2: (0,0), (-1,0), (-2,2), (3,2), (2,0), (0,0) en interior

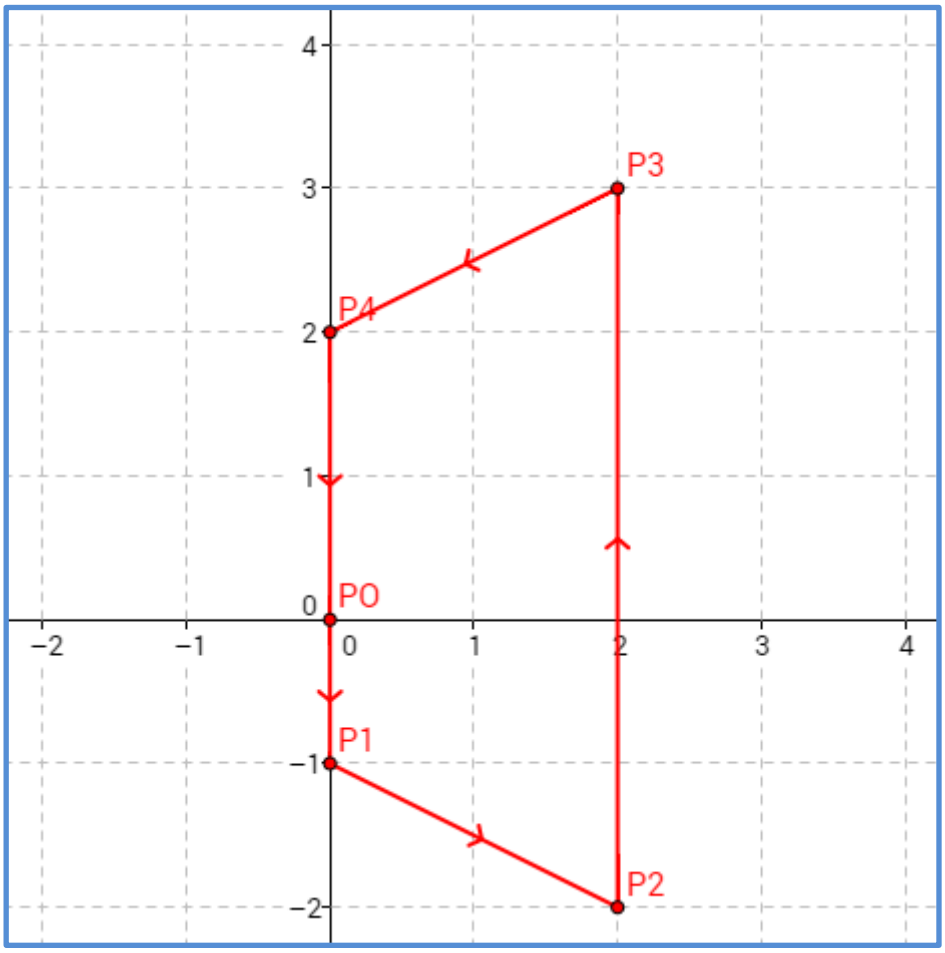


Figura 44. Trayectoria 2 en interior

5.2.2.1 *Algoritmo de control 1: tmrtestControl2*

Desplazamiento i→i+1	CASO (pág.52)	Punto deseado(x,y)	Navigate()				Punto destino medido(x,y)
			roll	pitch	yaw	gaz	
1→2	**6**	(0,0)→(-1,0)	0	-0.05	0.9999	0	-
2→3	**1**	(-1,0)→(-2,2)	0	-0.05	-0.3520	0	-
3→4	**7**	(-2,2)→(3,2)	0	-0.05	-0.6469	0	-
4→5	**3**	(3,2)→(2,0)	0	-0.05	-0.6469	0	-
5→6	**6**	(2,0)→(0,0)	0	-0.05	-0.3520	0	**(1.4,1.2)**

Tabla 10. Medidas trayectoria 2, algoritmo 1 en interior

Error calculado en X= 1.4 metros ; Error calculado en Y= 1.2 metros;

http://youtu.be/9CM1uuXeRDE

5.2.3 Trayectoria 3: (0,0), (4,1), (0,2), (0,0) en interior

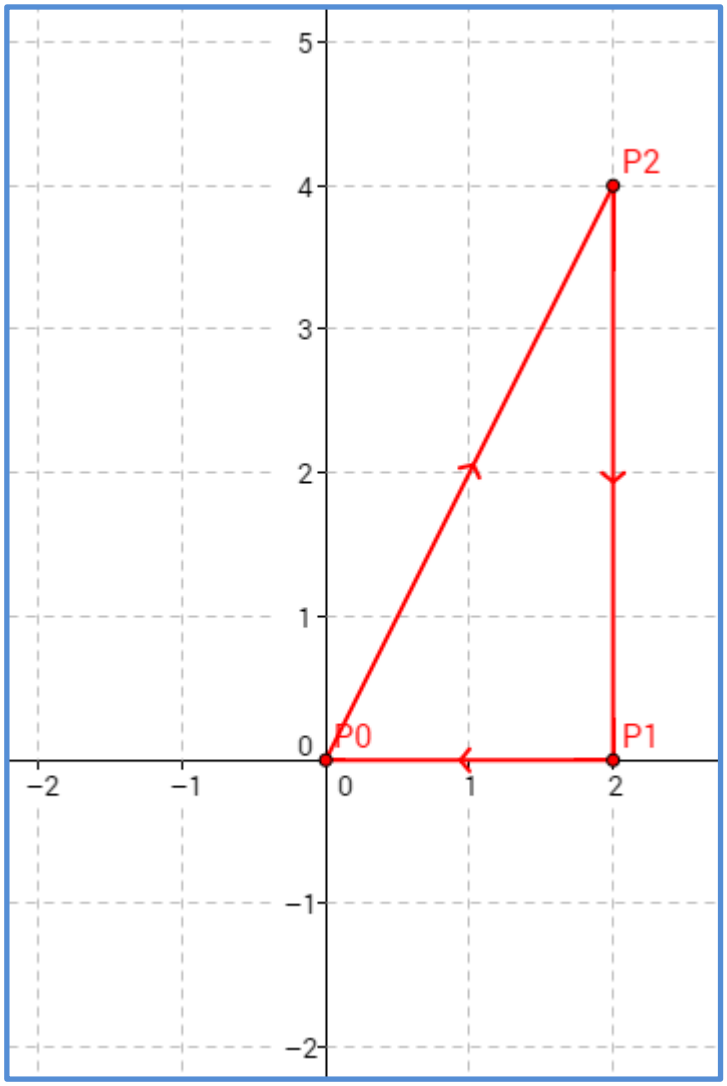


Figura 45. Trayectoria 3 en interior

5.2.3.1 *Algoritmo de control 1: tmrtestControl2*

Desplazamiento i→i+1	CASO (pág.52)	Punto deseado(x,y)	Navigate()				Punto destino medido(x,y)
			roll	pitch	yaw	gaz	
1→2	**2**	(0,0)→(4,1)	0	-0.05	0.0779	0	-
2→3	**1**	(4,1)→(0,2)	0	-0.05	0.8439	0	-
3→4	**8**	(0,2)→(0,0)	0	-0.05	0.5779	0	**(-0.3,1.2)**

Tabla 11. Medidas trayectoria 3, algoritmo 1 en interior

Error calculado en X= 0.3 metros; Error calculado en Y= 1.2 metros;

http://youtu.be/WA-SNl8tPsA

5.2.4 Trayectoria 4: (0,0), (3,0), (3,2), (0,3), (0,0) en interior

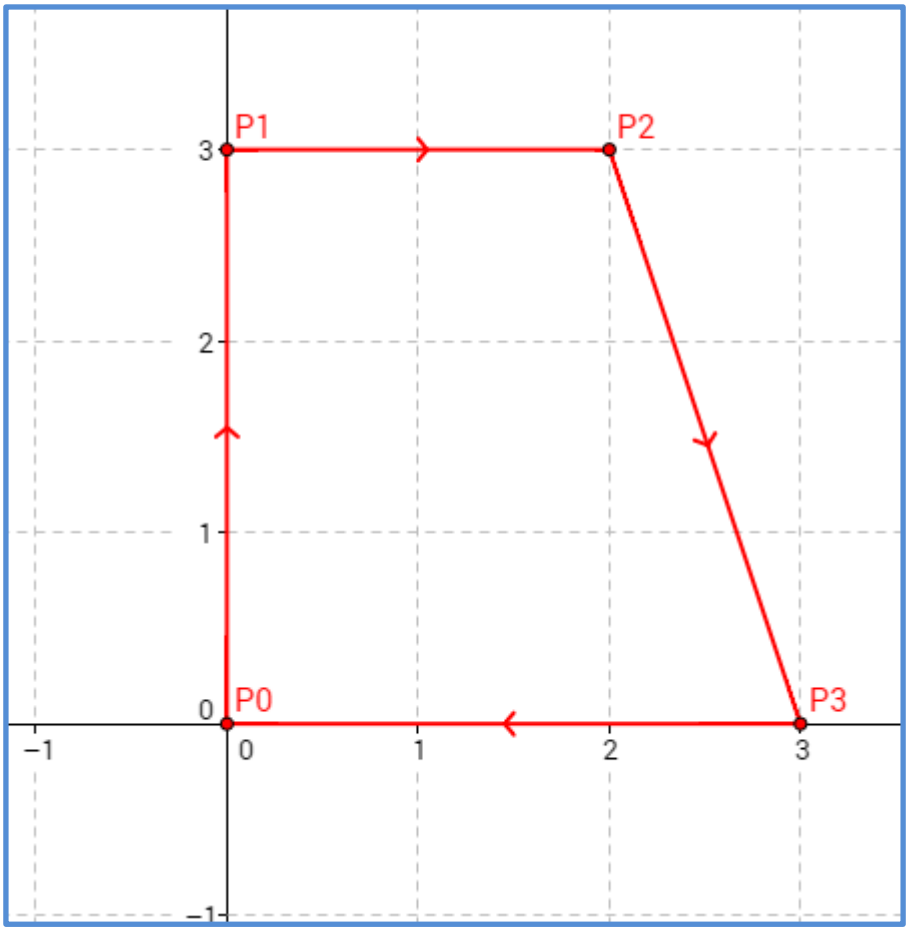


Figura 46. Trayectoria 4 en interior

5.2.4.1 *Algoritmo de control 1: tmrtestControl2*

Desplazamiento i→i+1	CASO (pág.52)	Punto deseado(x,y)	Navigate()				Punto destino medido(x,y)
			roll	pitch	yaw	gaz	
1→2	**7**	(0,0)→(3,0)	0	-0.05	0	0	-
2→3	**9**	(3,0)→(3,2)	0	-0.05	0.4999	0	-
3→4	**1**	(3,2)→(0,3)	0	-0.05	0.3975	0	-
4→5	**8**	(0,3)→(0,0)	0	-0.05	0.6023	0	**(0,-0.8)**

Tabla 12. Medidas trayectoria 4, algoritmo 1 en interior

Error calculado en X= 0 metros; Error calculado en Y= 0.8 metros;

http://youtu.be/CEHztrAhoJE

5.2.5 Trayectoria 5: (0,0), (0,-2), (-2,-3), (-4,-3), (-4,0) en interior

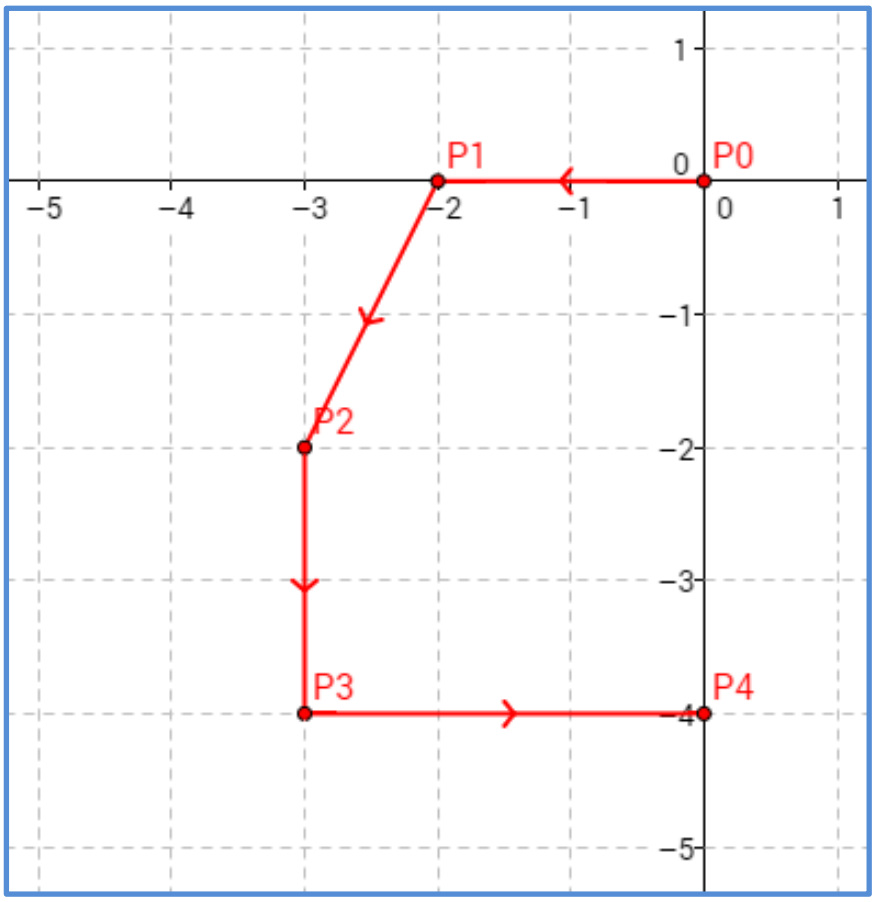


Figura 47. Trayectoria 5 en interior

5.2.5.1 *Algoritmo de control 1: tmrtestControl2*

Desplazamiento i→i+1	CASO (pág.52)	Punto deseado(x,y)	Navigate()				Punto destino medido(x,y)
			roll	pitch	yaw	gaz	
1→2	**8**	(0,0)→(0,-2)	0	-0.05	-0.4999	0	-
2→3	**3**	(0,-2)→(-2,-3)	0	-0.05	-0.3523	0	-
3→4	**6**	(-2,-3)→(-4,-3)	0	-0.05	-0.1475	0	-
4→5	**9**	(-4,-3)→(-4,0)	0	-0.05	-0.4999	0	**(-4.2,0)**

Tabla 13. Medidas trayectoria 5, algoritmo 1 en interior

Error calculado en X= 0.2 metros; Error calculado en Y= 0 metros;

http://youtu.be/DjFtRl-VtQ4

5.2.6 Trayectoria 6: (0,0), (6,0), (6,-4) en interior

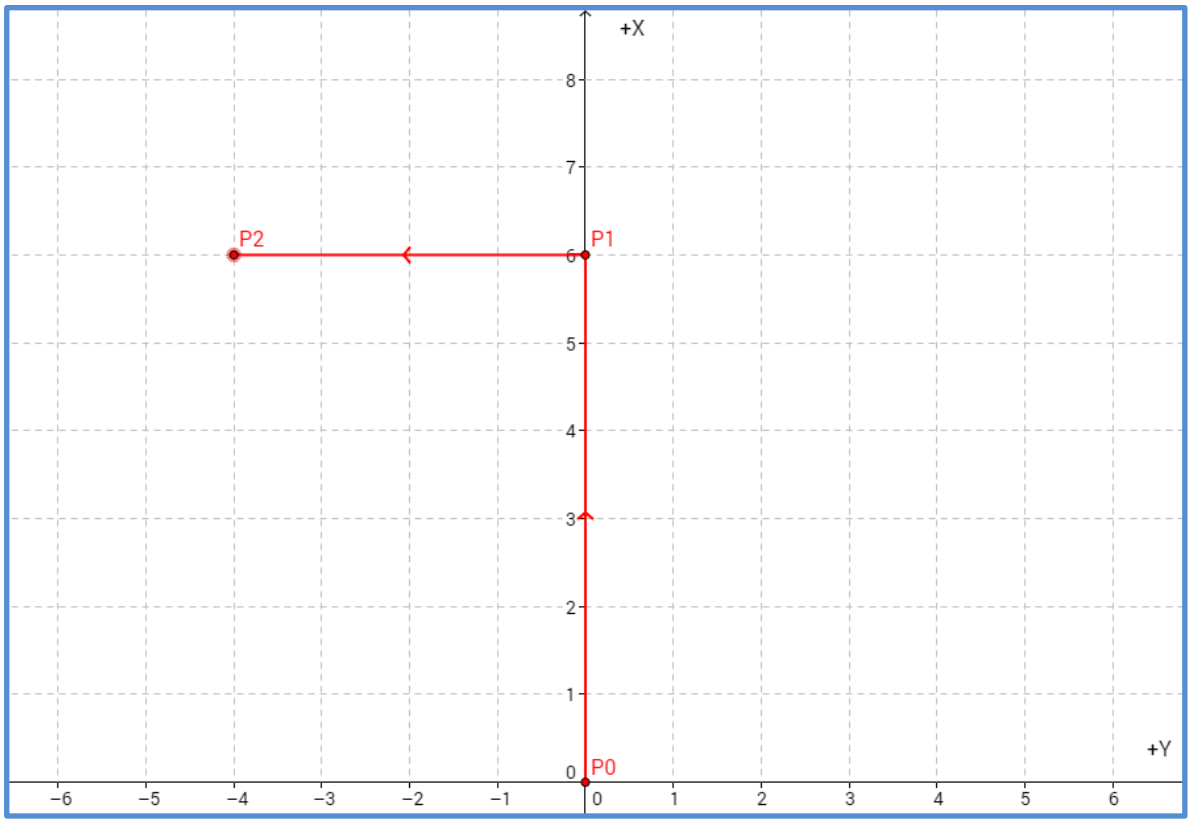


Figura 48. Trayectoria 6 en interior

5.2.6.1 *Algoritmo de control 1: tmrtestControl2*

Desplazamiento i→i+1	CASO (pág.52)	Punto deseado(x,y)	Navigate()				Punto destino medido(x,y)
			roll	pitch	yaw	gaz	
1→2	**7**	(0,0)→(6,0)	0	-0.05	0	0	**(6,0)**
2→3	**8**	(6,0)→(6,-4)	0	-0.05	-0.4999	0	**(4.9,-4.7)**

Tabla 14. Medidas trayectoria 6, algoritmo 1 en interior

Error calculado en X= 1.1 metros; Error calculado en Y= 0.7 metros;

http://youtu.be/RXEAXZNnw1E

5.3 Trayectorias realizadas en exterior

Las trayectorias realizadas en exterior y que fueron grabadas para analizar se realizaron el día 20 de Febrero de 2015, con las siguientes condiciones meteorológicas:

Condiciones meteorológicas actuales

Estación: Madrid / Getafe el 20 Feb 2015, 14:00

Cielo: **10º** sensación térmica **8º** **Viento:** 11 km/h S

Humedad: 54% **Presión:** 1021 hPa **Visibilidad:** 10 km

Podemos observar que hizo un viento de más de 11 km/h lo que dificultó mucho los vuelos haciendo a nuestro sistema muy inestable. En estas trayectorias apreciaremos que los giros sí son realizados correctamente, pero los avances con forward son muy distintos a como tendrían que ser debido a la fuerza del viento

5.3.1 Trayectoria 1: (0,0), (5,0), (5,-3), (0,-3), (0,0) en exterior

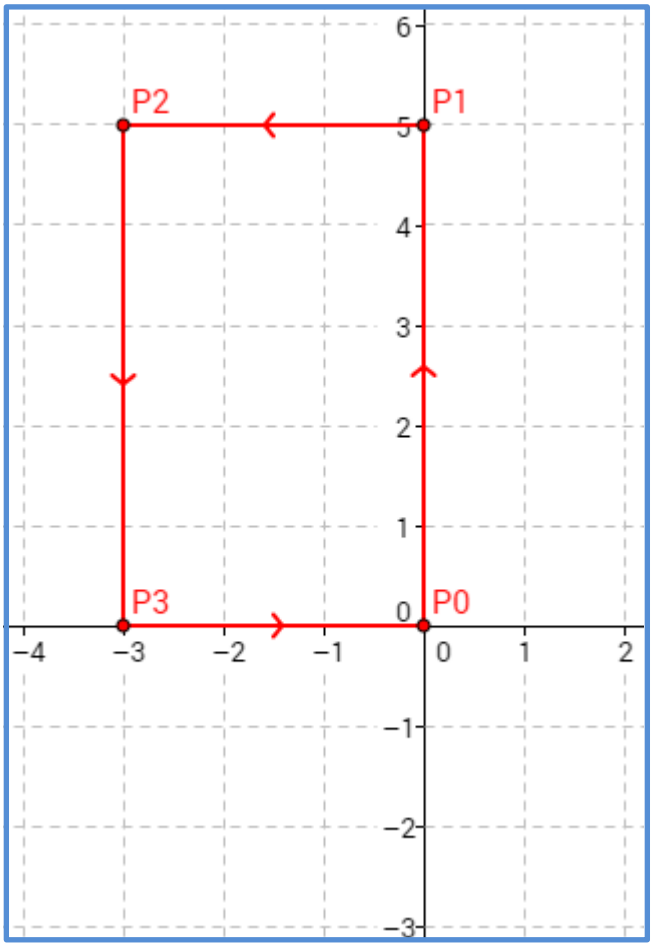


Figura 49. Trayectoria 1 en exterior

5.3.1.1 *Algoritmo de control 1: tmrtestControl2*

Desplazamiento i→i+1	CASO (pág.52)	Punto deseado(x,y)	Navigate()				Punto destino medido(x,y)
			roll	pitch	yaw	gaz	
1→2	**7**	(0,0)→(5,0)	0	-0.05	-0.4999	0	-
2→3	**8**	(5,0)→(5,-3)	0	-0.05	-0.4999	0	-
3→4	**6**	(5,-3)→(0,-3)	0	-0.05	-0.4999	0	-
4→5	**9**	(0,-3)→(0,0)	0	-0.05	-0.4999	0	**(-3.2,-1.8)**

Tabla 15. Medidas trayectoria 1, algoritmo 1 en exterior

Error calculado en X= 1.8 metros; Error calculado en Y= 3.2 metros;

http://youtu.be/2yhuWYMpNqo

5.3.2 Trayectoria 2: (0,0), (2,2), (2,4), (0,6), (-4,6), (-4,0), (0,0) en exterior

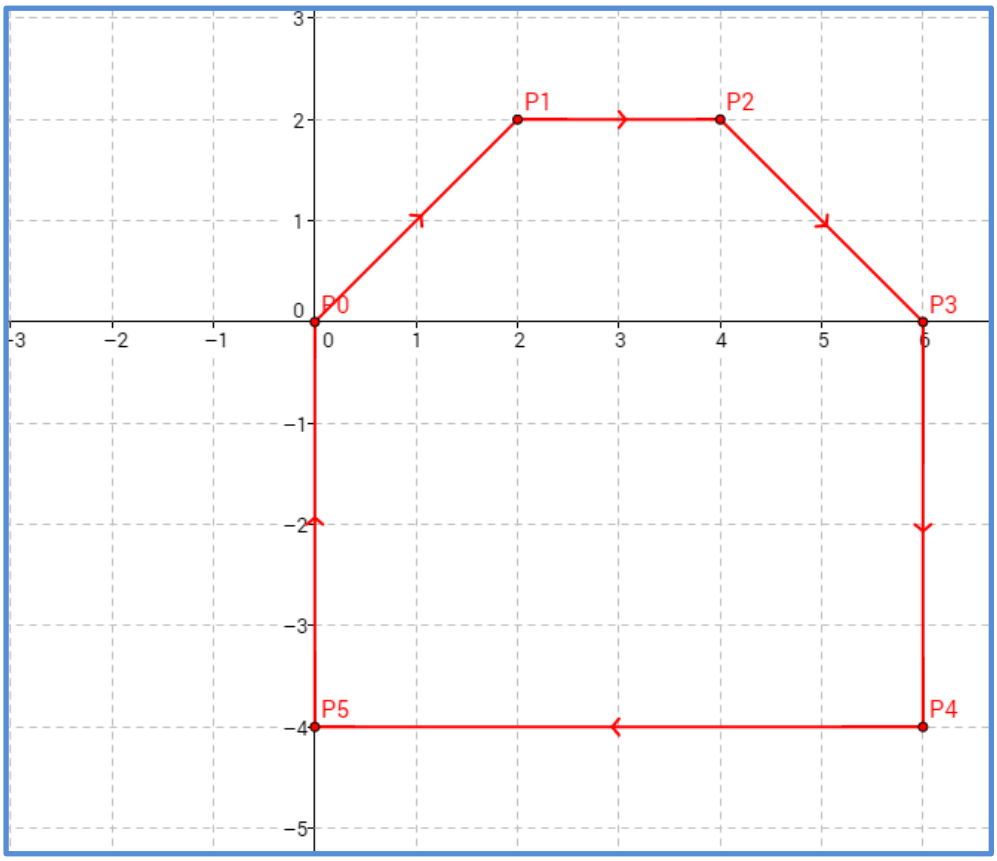


Figura 50. Trayectoria 2 en exterior

5.3.2.1 *Algoritmo de control 1: tmrtestControl2*

Desplazamiento i→i+1	CASO (pág.52)	Punto deseado(x,y)	Navigate()				Punto destino medido(x,y)
			roll	pitch	yaw	gaz	
1→2	**2**	(0,0)→(2,2)	0	-0.05	0.2499	0	-
2→3	**9**	(2,2)→(2,4)	0	-0.05	0.2499	0	-
3→4	**1**	(2,4)→(0,6)	0	-0.05	0.2499	0	-
4→5	**6**	(0,6)→(-4,6)	0	-0.05	0.2499	0	-
5→6	**8**	(-4,6)→(-4,0)	0	-0.05	0.4999	0	-
6→7	**7**	(-4,0)→(0,0)	0	-0.05	0.4999	0	**(-3.3,-2.8)**

Tabla 16. Medidas trayectoria 2, algoritmo 1 en exterior

Error calculado en X= 3.3 metros; Error calculado en Y= 2.8 metros;

http://youtu.be/2R8EDmMYEHs

5.3.3 Trayectoria 3: (0,0), (2,2), (4,2), (4,0), (0,0) en exterior

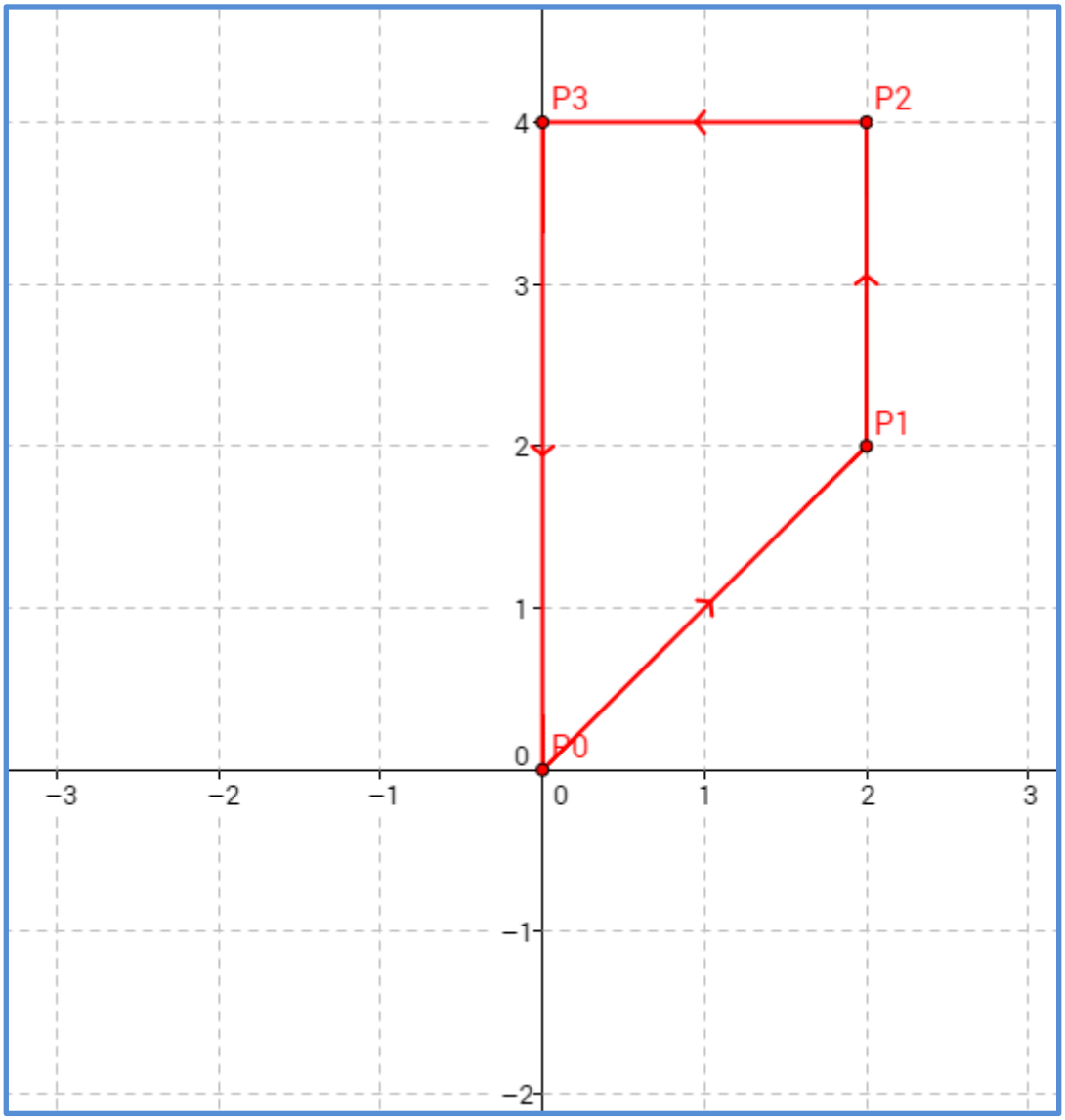


Figura 51. Trayectoria 3 en exterior

5.3.3.1 *Algoritmo de control 1: tmrtestControl2*

Desplazamiento i→i+1	CASO (pág.52)	Punto deseado(x,y)	Navigate()				Punto destino medido(x,y)
			roll	pitch	yaw	gaz	
1→2	**2**	(0,0)→(2,2)	0	-0.05	0.2499	0	-
2→3	**7**	(2,2)→(4,2)	0	-0.05	-0.2499	0	-
3→4	**8**	(4,2)→(4,0)	0	-0.05	-0.4999	0	-
4→5	**6**	(4,0)→(0,0)	0	-0.05	-0.4999	0	**(2.2,0.9)**

Tabla 17. Medidas trayectoria 3, algoritmo 1 en exterior

Error calculado en X= 2.2 metros; Error calculado en Y= 0.9 metros;

http://youtu.be/Vk3WZdFvs2A

5.3.4 Trayectoria 4: (0,0), (-3,-3), (0,-3), (0,0) en exterior

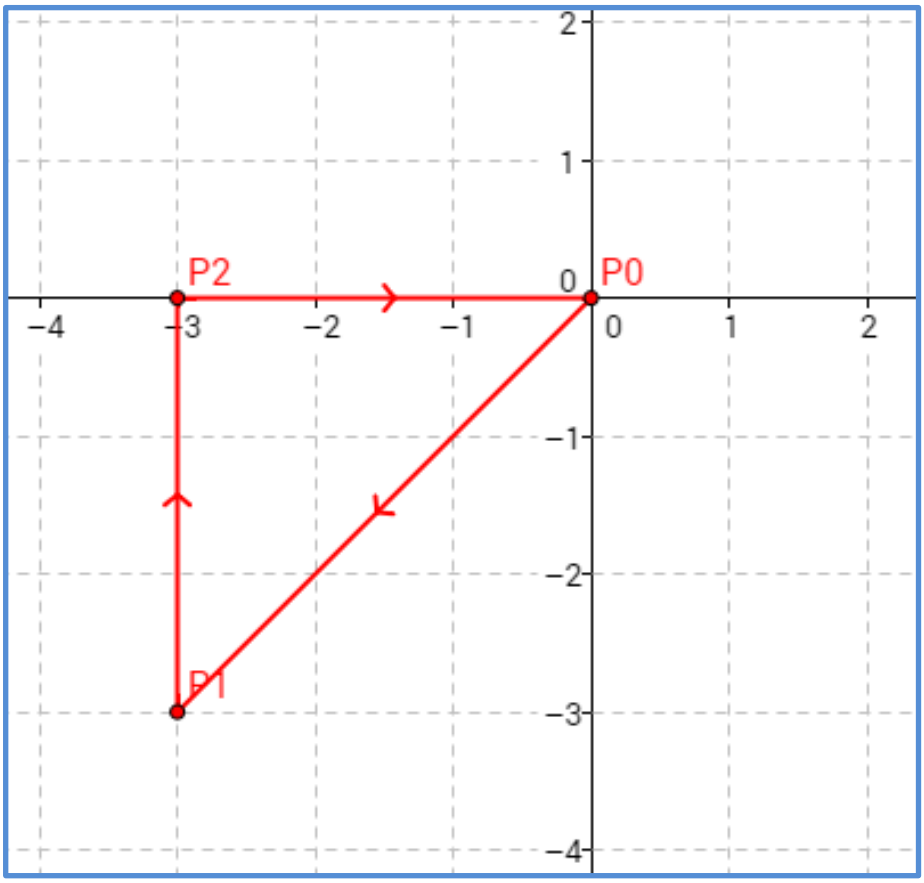


Figura 52. Trayectoria 4 en exterior

5.3.4.1 *Algoritmo de control 1: tmrtestControl2*

Desplazamiento i→i+1	CASO (pág.52)	Punto deseado(x,y)	Navigate()				Punto destino medido(x,y)
			roll	pitch	yaw	gaz	
1→2	**3**	(0,0)→(-3,-3)	0	-0.05	-0.7499	0	-
2→3	**7**	(-3,-3)→(0,-3)	0	-0.05	0.7499	0	-
3→4	**9**	(0,-3)→(0,0)	0	-0.05	0.4999	0	**(-1.2,-1.7)**

Tabla 18. Medidas trayectoria 4, algoritmo 1 en exterior

Error calculado en X= 1.2 metros; Error calculado en Y= 1.7 metros;

http://youtu.be/9eHCNmnAUjA

5.3.5 Trayectoria 5: (0,0), (0,-4), (-3,-1), (-3,-4), (-6,-4), (-6,0), (0,0) en exterior

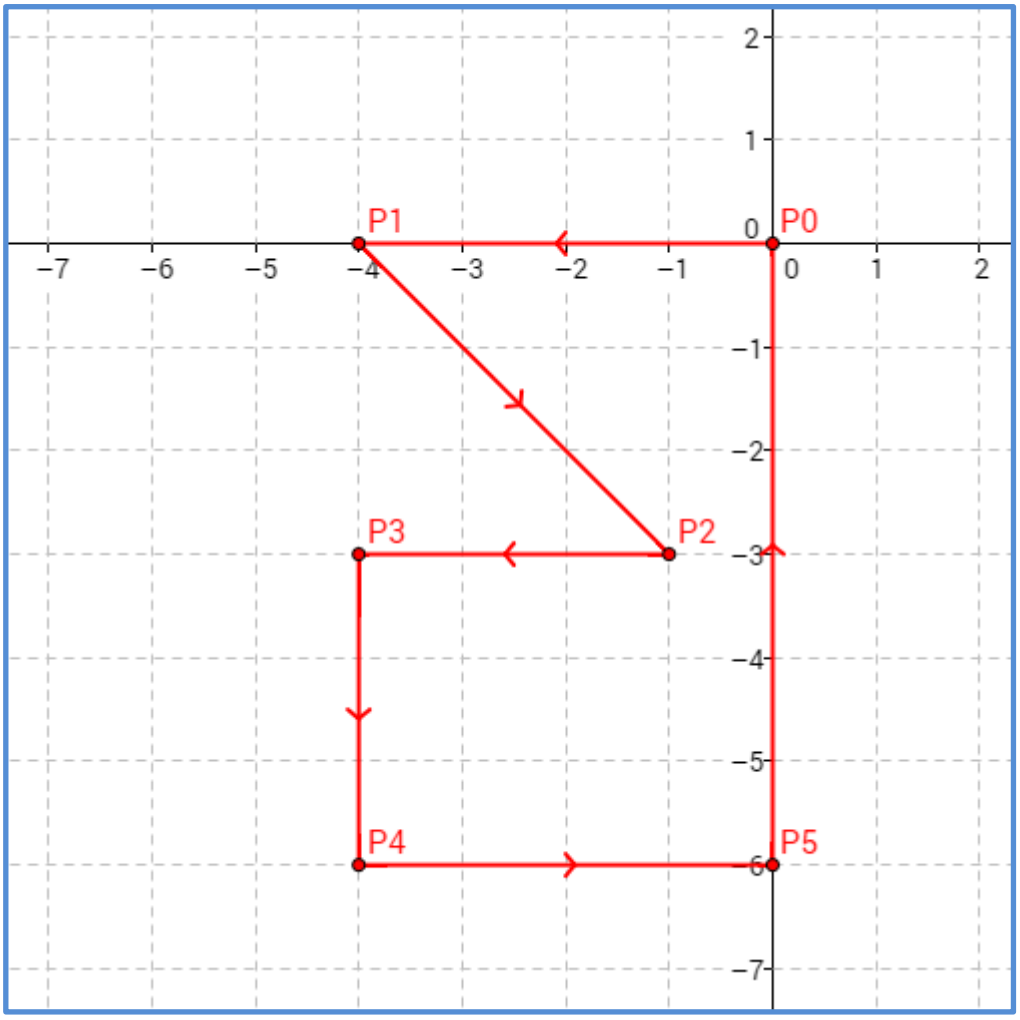


Figura 53. Trayectoria 5 en exterior

5.3.5.1 *Algoritmo de control 1: tmrtestControl2*

Desplazamiento i→i+1	CASO (pág.52)	Punto deseado(x,y)	Navigate()				Punto destino medido(x,y)
			roll	pitch	yaw	gaz	
1→2	**8**	(0,0)→(0,-4)	0	-0.05	-0.4999	0	-
2→3	**1**	(0,-4)→(-3,-1)	0	-0.05	-0.7499	0	-
3→4	**8**	(-3,-1)→(-3,-4)	0	-0.05	-0.7499	0	-
4→5	**6**	(-3,-4)→(-6,-4)	0	-0.05	-0.4999	0	-
5→6	**9**	(-6,-4)→(-6,0)	0	-0.05	-0.4999	0	-
6→7	**7**	(-6,0)→(0,0)	0	-0.05	-0.4999	0	**(-3.5,-2)**

Tabla 19. Medidas trayectoria 5, algoritmo 1 en exterior

Error calculado en X= 3.5 metros; Error calculado en Y= 2 metros;

http://youtu.be/wASqlwsYPd0

5.4 Comentarios

En este apartado vamos a hacer un comentario acerca de la experiencia obtenida al realizar las trayectorias finales.

Como podemos observar en las tablas de resultados, afirmamos como ya sabíamos que las trayectorias en exterior tienen un error mayor que las realizadas en interior. Esto es debido a las corrientes de aire, ya que un aire mínimo hace a nuestro cuadricóptero perder el control y no dejarle avanzar correctamente a su destino.

Respecto a los dos algoritmos calculados podemos decir que el algoritmo 1 es más efectivo para condiciones generales, aunque el algoritmo 2 para trayectorias en que los ángulos son rectos o la trayectoria no tiene que girar éste también responde de manera eficiente.

A nuestras trayectorias finales también le afectan otros parámetros como el deterioro del AR Drone 2.0 debido al uso dedicado al proyecto. Este deterioro es pronunciado en las hélices delanteras, ya que el algoritmo principal (tmrTestControl2) calculado siempre ha utilizado el morro de nuestro avión para desplazarse de punto a punto, y los impactos de esta parte con las paredes han sido numerosos. La propulsión del drone depende totalmente del estado de las hélices, por lo que una de ellas con una mínima deformación afecta a nuestro vuelo. A continuación mostramos una imagen con el estado final y las marcas que ha sufrido nuestro drone, y un vídeo en el que se desmonta un rotor durante las últimas pruebas realizadas:

http://youtu.be/B1eDi3yfCd8

Figura 54. Estado final AR Drone 2.0

También tenemos que comentar que nuestro sistema al ser en lazo abierto y no comparar la salida con la entrada va a tener un cierto error, aunque podemos asegurar que es mínimo en condiciones normales. El algoritmo calculado paso a paso cumple con las expectativas planteadas al principio de este proyecto y responde ante cualquier condición o punto que le introduzcamos.

Capítulo 6: Conclusiones y trabajos futuros

6.1 Conclusiones

Una vez analizado el problema propuesto, se ha estudiado y desarrollado un modelo final que cumple con los requisitos pedidos en este proyecto. Se han realizado diferentes pruebas finales que demuestran un buen control de la trayectoria, que es el objetivo principal de este trabajo, aunque siempre hay pequeños errores debido a que los sensores que utiliza el AR Drone 2.0 son de "low cost" por lo que no es preciso un vuelo al 100%, aun así las pruebas son satisfactorias, principalmente en interior. El capítulo 5 muestra todas las pruebas realizadas con diferentes datos para comprender e interpretar los resultados obtenidos.

En un primer momento se intentó trabajar con el gps de Parrot y programar la trayectoria accediendo a él, pero tras buscar varios días información se llegó a la conclusión de que no se podía hackear por lo que se rechazó y cambiamos a trabajar con el inercial que sí se podía modificar gracias a la plataforma de desarrollo suministrada por Parrot, aunque mi proyecto ha empezado a partir de una versión mejorada de dicha plataforma suministrada por la Universidad Carlos III de Madrid. Se espera y prevé que en un futuro no muy lejano el gps pueda ser accesible por parte de Parrot y sea compartido con sus usuarios. Actualmente el gps simplemente graba la trayectoria realizada y almacena el vídeo del vuelo. Probablemente con gps nuestra trayectoria tendría menos error.

Durante el desarrollo del proyecto han ido surgiendo diferentes problemas que parecían imposibles de resolver, pero que con ayuda de los profesores y trabajo individual se han ido resolviendo hasta llegar al final. Las soluciones adoptadas probablemente no siempre hayan sido las mejores ni las más óptimas, sino que se han escogido las más seguras y viables. El desarrollo del proyecto que se ha seguido a grande escala ha sido en primer lugar estabilizar el vuelo al despegar, lo siguiente fue realizar movimientos simples con la botonera, después se enlazaron dos movimientos de forma autónoma y por último la programación de un sistema que tuviera la capacidad de seguir una trayectoria de puntos de forma automática.

Hay que comentar que al vuelo de cualquier cuadricóptero vía Wi-Fi le afectan diferentes parámetros y que dependiendo del lugar el resultado puede ser mejor o peor, ya que la estación de control no tiene demasiada potencia. Si realizamos un vuelo en un área libre de contaminación radiante, como puede ser un garaje, tiene menos error que por ejemplo en los pasillos de la universidad donde la conectividad Wi-Fi es inmensa ya que pueden interferir con alta probabilidad. Al igual que si el vuelo lo realizamos en la calle, cualquier oleada de aire desestabiliza el drone como se ha comprobado a lo largo de las distintas pruebas realizadas en nuestro proyecto.

También a nuestro sistema cuadricóptero le puede afectar el estado de la batería, ya que hemos tenido la experiencia con una batería de 1000 mAh. Con esta batería, al final del proyecto, era incapaz de avanzar cuando le mandábamos una orden por lo que la desechamos en ese momento.

Como hemos comentado anteriormente los componentes de este drone son de baja precisión, por lo que el sistema de Parrot podría ser útil para probar múltiples proyectos de investigación, como el presente, y después incorporarlo a un sistema de más envergadura con sensores más precisos.

Por último se quiere comentar que la experiencia con el AR Drone 2.0 ha sido muy buena en su totalidad, ya que he aprendido diferentes técnicas de aeronavegación y he conocido un sistema que está en plena expansión. También he reforzado mis conocimientos de programación con un sistema real en un lenguaje como C#.

6.2 Trabajos futuros

Dado que el tema de los drones está en pleno auge se pueden proponer varios trabajos para un futuro dependiendo de la línea en que se quiera trabajar como puede ser el control automático, trayectorias etc. A continuación se proponen diferentes proyectos que pueden realizar futuros compañeros:

A partir del presente proyecto, siempre y cuando el gps de Parrot Flight Recorder sea vulnerable y deje acceder a sus datos, se podría realizar un proyecto que siga una trayectoria mediante el gps de forma similar. También podría hacerse un estudio completo comparando ambos sistemas y sacar conclusiones.

También se ha pensado como continuación de este proyecto que se guarden los datos del sensor de medida inercial IMU en tablas de Excel, para poder calcular y comparar el error entre el vuelo real y los datos que nos proporciona la unidad de medida inercial como son la velocidad, orientación y los ángulos de navegación.

Otro posible proyecto relacionado con este cuadricóptero podría ser realizar un vuelo autónomo pero trabajando con las cámaras, relacionado con sistemas de percepción, que fuera capaz de seguir una determinada imagen que le pongamos a nuestro drone.

Por último otro posible proyecto que se ha pensado es modificar algunos de los sensores internos de nuestro cuadricóptero por otros de más resolución y fiables para tener menor error en los vuelos, y además añadir la coordenada z a nuestro sistema de control para así modificar la altura al seguir una trayectoria.

Capítulo 7: Gestión del proyecto

En este capítulo vamos a exponer la planificación por días y horas que se le han dedicado al proyecto, y un estudio económico del coste total que ha supuesto.

7.1 Planificación

La planificación del trabajo dedicado se va a exponer mediante un diagrama de Gantt con ayuda del programa GanttProject.

Un diagrama de Gantt es una herramienta gráfica que nos sirve para comprender y observar el trabajo dedicado a diferentes tareas a lo largo de un tiempo total determinado. Estos diagramas son una herramienta básica en gestión de todo tipo de proyectos, con la finalidad de representar las diferentes tareas y fases que han formado parte de un proyecto. Está formado por un eje vertical que indica las tareas, y un eje horizontal que indica el tiempo dedicado a esa determinado parte del trabajo.

Para representar nuestro trabajo en el diagrama hemos dividido dicho trabajo en diferentes partes que son las siguientes:

- Instalación de la interfaz LSIDrone.
- Estudio del sistema cuadricóptero, comportamiento y sus componentes.
- Desarrollo completo del software.
- Post Mortem.

7.1.1 Diagramas de Gantt

A continuación mostramos en el diagrama de Gantt todas las tareas llevadas a cabo a lo largo del proyecto, el cual es un archivo de tipo .gan que se adjunta en el DVD del trabajo junto con el resto de archivos.

Nombre	Fecha de inicio	Fecha de fin
0. Instalación interfaz	21/02/14	28/02/14
1. Estudio del sistema cuadricóptero	16/06/14	31/07/14
1.1 Análisis del sistema GPS	16/06/14	21/06/14
1.2 Estudio del inercial cuadricóptero teórico	23/06/14	3/07/14
1.3 Estudio de la interfaz LSIDrone	4/07/14	9/07/14
1.4 Comunicación PC-Drone, primeros vuelos	11/07/14	22/07/14
1.5 Estabilizacion del cuadricóptero	22/07/14	31/07/14
2. Desarrollo del software	16/09/14	14/01/15
2.1 Desarrollo botonera,vuelo manual	16/09/14	30/09/14
2.2 Control vuelo manual y estabilización, con el contador	1/10/14	15/10/14
2.3 Cálculo teórico algoritmo de la trayectoria	16/10/14	22/10/14
2.4 Realizar dos acciones de forma autónoma	23/10/14	30/10/14
2.5 Modificación de la interfaz	3/11/14	7/11/14
2.6 Medida de errores y ajuste del intervalo de control	10/11/14	19/11/14
2.7 Programacion del algoritmo en C#	20/11/14	2/12/14
2.8 Solución a los problemas en el modelo final	3/12/14	14/01/15
3. Post Mortem	26/12/14	27/02/15
3.1 Desarrollo del documento final	26/12/14	21/02/15
3.2 Preparacion y elaboración de la defensa del proyecto	23/02/15	27/02/15

Figura 55. Tareas diagrama de Gantt

Para facilitar la comprensión expondremos un diagrama de cada tarea en las siguientes páginas, dejando las tareas 3 y 4 de forma simultánea ya que se hicieron partes a la vez.

Tras realizar el diagrama se ha exportado el proyecto y se ha generado un informe en HTML, donde podemos comprobar la documentación generada y si es acorde a lo esperado. También se expondrá a continuación un diagrama de los recursos, que en este caso lo ha realizado una sola persona que es la encargada del presente proyecto, y por último mostraremos una descripción generalizada del trabajo que se ha realizado en cada tarea y subtarea del proyecto.

7.1.1.1 *Diagrama instalación de la interfaz*

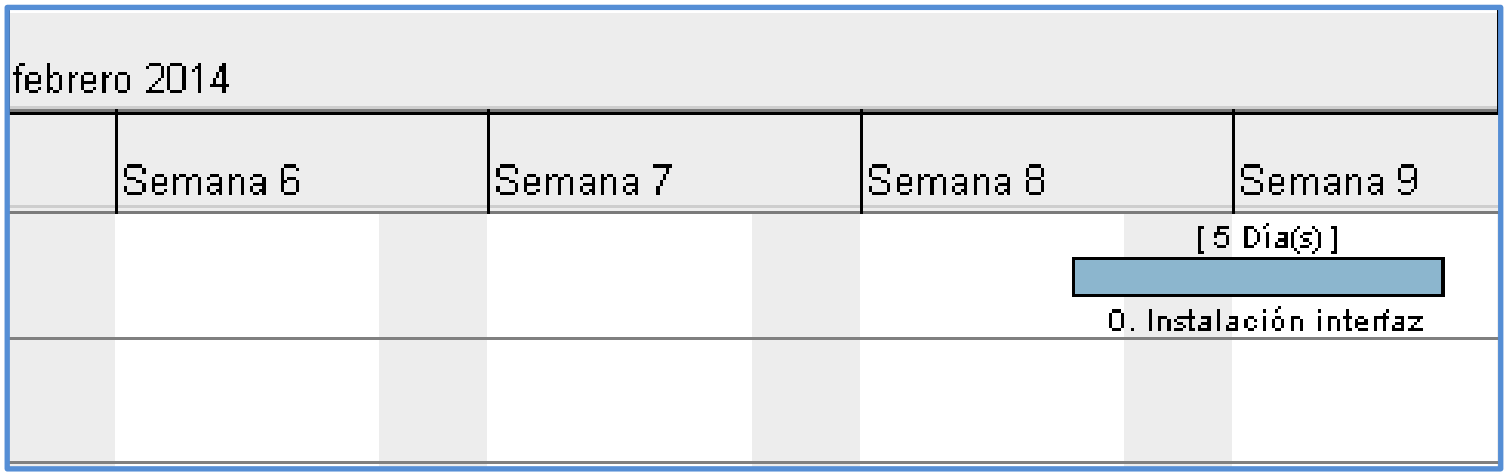


Figura 56. Diagrama de Gantt instalación interfaz

7.1.1.2 *Diagrama estudio del sistema cuadricóptero*

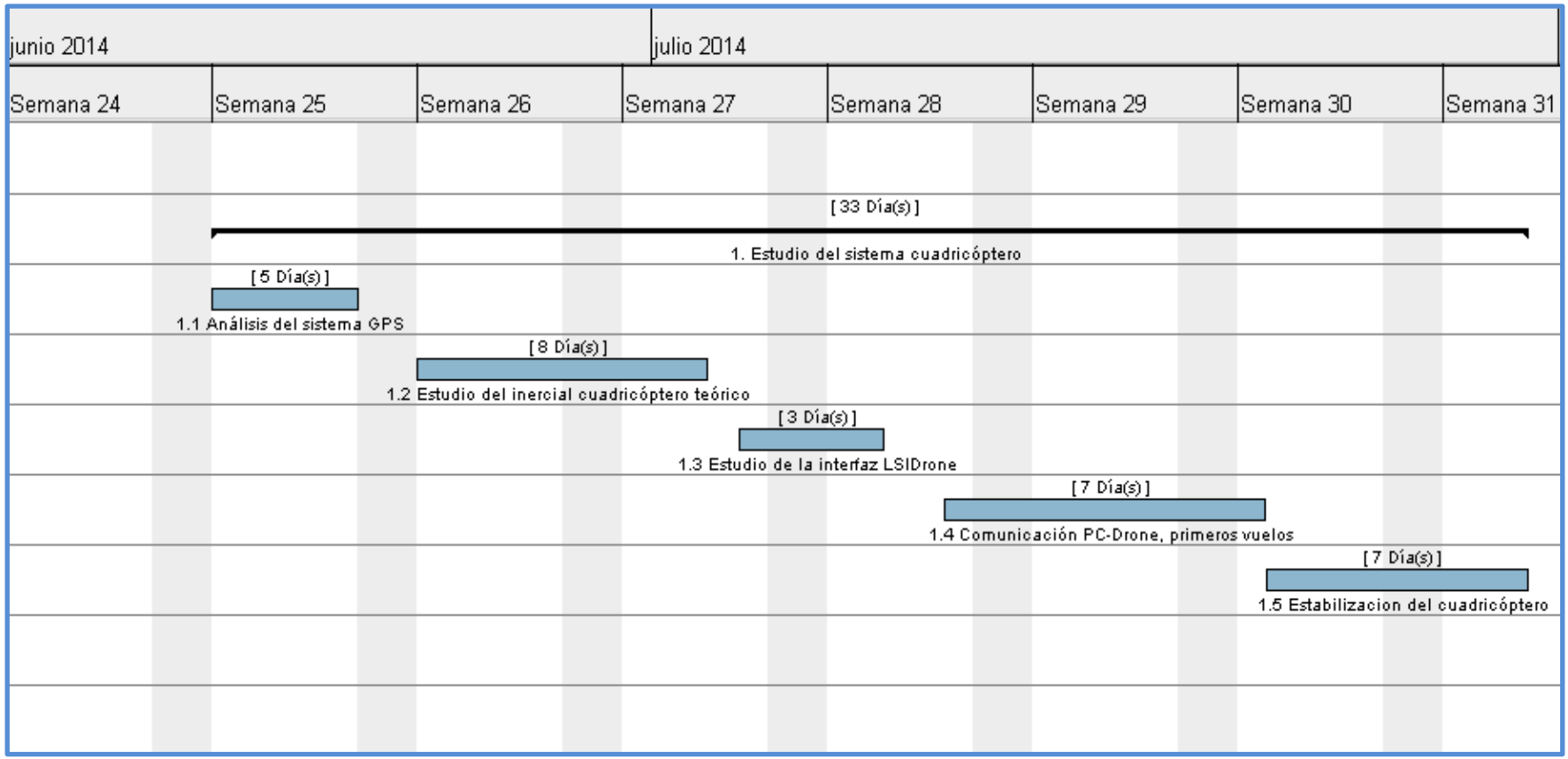


Figura 57. Diagrama de Gantt estudio sistema cuadricóptero

7.1.1.3 *Diagrama desarrollo del software y Post Mortem*

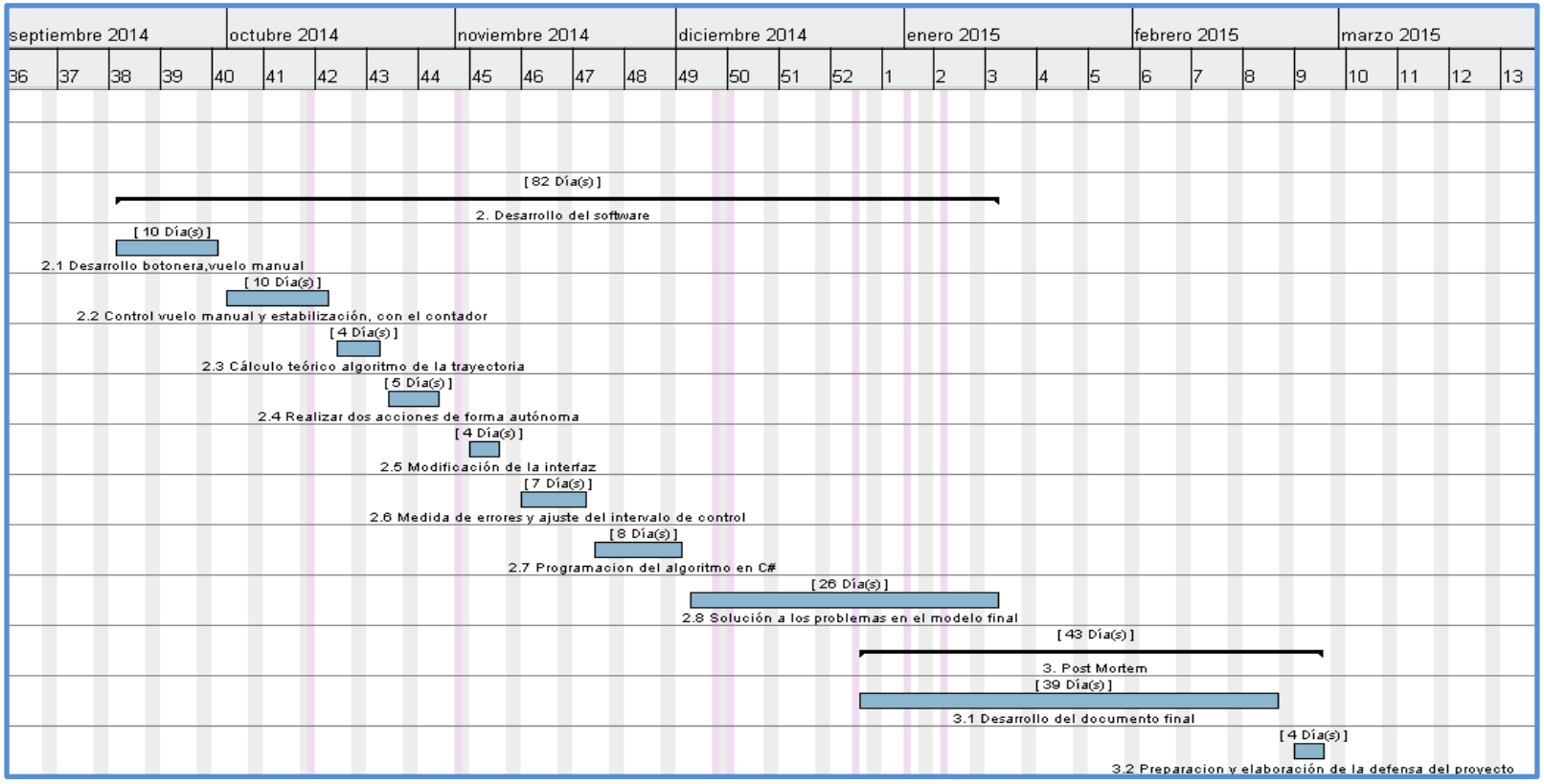


Figura 58. Diagrama de Gantt desarrollo software y Post Mortem

7.1.2 Diagrama de recursos

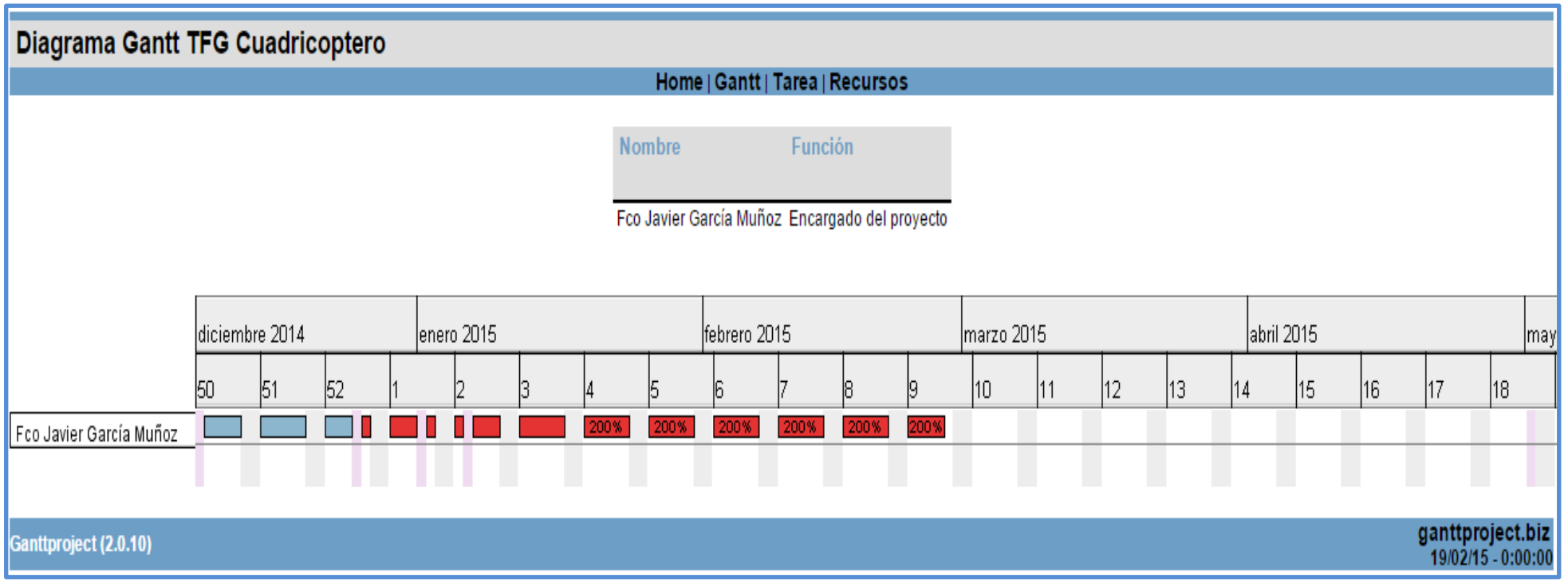


Figura 59. Diagrama de Gantt recursos

7.1.3 Descripción de las tareas

0. Instalación interfaz Instalación de todos los archivos necesarios para poder realizar el proyecto como Microsoft Visual Studio 2013, bibliotecas Emgu, SDK e interfaz LSIDrone.	21/02/14	28/02/14	Fco Javier García Muñoz
1. Estudio del sistema cuadricóptero Estudio de todo el sistema cuadricóptero como los ángulos inerciales, ejes, y velocidades.	16/06/14	31/07/14	Fco Javier García Muñoz
1.1 Análisis del sistema GPS Búsqueda de información relacionada con el GPS de Parrot Flight Recorder, viendo si era hackeable y si se podía acceder a él. Finalmente se determinó que no y se orientó el proyecto en otra dirección para trabajar con el inercial.	16/06/14	21/06/14	Fco Javier García Muñoz
1.2 Estudio del inercial cuadricóptero teórico Análisis de todo el inercial de forma teórica que íbamos a encontrarnos en los cuadricópteros.	23/06/14	3/07/14	Fco Javier García Muñoz
1.3 Estudio de la interfaz LSIDrone Análisis de todo el código en C# proporcionado por Abdulla Al-Kaff en la Universidad Carlos III de Madrid.	4/07/14	9/07/14	Fco Javier García Muñoz
1.4 Comunicación PC-Drone, primeros vuelos Conexión vía Wi-Fi entre nuestro cuadricóptero y nuestro ordenador. Primeros vuelos para empezar a comprender el comportamiento de nuestro drone.	11/07/14	22/07/14	Fco Javier García Muñoz
1.5 Estabilización del cuadricóptero En este paso se realizó la estabilización del cuadricóptero al despegar. También se comprendieron los distintos modos que tenía nuestra interfaz como el HoverMode, Emergency, Flattrim entre otros.	22/07/14	31/07/14	Fco Javier García Muñoz
2. Desarrollo del software Desarrollo software del proyecto en su totalidad.	16/09/14	14/01/15	Fco Javier García Muñoz
2.1 Desarrollo botonera, vuelo manual Modificación de la botonera para que fuera estable al pulsar un determinado botón. También se han añadido otros como ascender y descender el drone.	16/09/14	30/09/14	Fco Javier García Muñoz
2.2 Control vuelo manual y estabilización, con el contador En este paso se puso un contador para que al pulsar un botón el drone se detuviera pasados unos segundos, que por defecto fueron 4, si no se pulsaba otro botón antes. Este fue uno de los pasos críticos para poder avanzar notablemente en el funcionamiento.	1/10/14	15/10/14	Fco Javier García Muñoz
2.3 Cálculo teórico algoritmo de la trayectoria Aquí se calculó el algoritmo para realizar una trayectoria generalizada para cualquier punto que le introdujéramos al sistema. Se determinó que la trayectoria iba a ser ir de punto a punto avanzando hacia delante nuestro cuadricóptero, de forma que antes de comenzar a avanzar tenemos que calcular un ángulo para girar dYaw.	16/10/14	22/10/14	Fco Javier García Muñoz
2.4 Realizar dos acciones de forma autónoma Este fue otro de los pasos críticos antes de comenzar a programar nuestro vuelo. Se consiguió que el drone avanzara hacia delante, se detuviera (HoverMode), realizara un giro y volviera a avanzar de forma autónoma.	23/10/14	30/10/14	Fco Javier García Muñoz
2.5 Modificación de la interfaz Última modificación de la interfaz en la que se ha añadido el botón AUTO que realizara nuestro vuelo de forma autónoma al ser presionado y seguirá una determinada trayectoria. También se han añadido etiquetas (label) para que nos muestren datos importantes de las variables necesarias para nuestro vuelo.	3/11/14	7/11/14	Fco Javier García Muñoz
2.6 Medida de errores y ajuste del intervalo de control En este punto se han realizado medidas para ver la distancia real que avanza nuestro cuadricóptero y se ha establecido un intervalo de control que mejor responde a nuestras instrucciones. Se ha calculado el error en x e y al avanzar una determinada distancia.	10/11/14	19/11/14	Fco Javier García Muñoz
2.7 Programación del algoritmo en C# Se ha introducido en lenguaje C# el algoritmo calculado teóricamente.	20/11/14	2/12/14	Fco Javier García Muñoz

2.8 Solución a los problemas en el modelo final Paso final para la finalización del proyecto. Resolución de todas las incidencias encontradas para el control autónomo, detallado en la memoria final.	3/12/14	14/01/15	Fco Javier García Muñoz
3. Post Mortem Desarrollo del documento final a entregar para la obtención del título Grado Ingeniería Electrónica Industrial y Automática.	26/12/14	27/02/15	Fco Javier García Muñoz
3.1 Desarrollo del documento final Desarrollo de la memoria en Microsoft Office 2010 en Word, y después convertida a .PDF	26/12/14	21/02/15	Fco Javier García Muñoz
3.2 Preparación y elaboración de la defensa del proyecto Desarrollo de la exposición y preparación para presentar delante del tribunal de la Universidad Carlos III de Madrid.	23/02/15	27/02/15	Fco Javier García Muñoz

7.2 Presupuesto

Según la planificación expuesta anteriormente en la que se han dedicado un total de 152 días a la elaboración del trabajo, con una dedicación media de 5 horas al día se ha calculado un total de 760 horas de trabajo. No se ha trabajado en días festivos ni fines de semana a la hora de realizar el diagrama de Gantt.

Incluiremos los costes del apartado de recursos humanos estableciendo un sueldo de 15 € por hora como ingeniero junior. Además en la tabla siguiente calcularemos los precios del hardware y el software utilizado para el desarrollo del proyecto.

Instalación interfaz: 5 días

Desarrollo del software: 82 días*

Estudio del sistema cuadricóptero: 33 días

Post Mortem: 43 días*

A lo largo del proyecto hemos utilizado dos cuadricópteros AR Drone 2.0, uno proporcionado por la Universidad Carlos III de Madrid la versión normal, y otro adquirido por mí la versión Elite Edition, diferenciándose estos únicamente en los colores de las hélices y las carcasas de interior y exterior.

Recursos	Concepto	Precio unitario	Cantidad	Total
Humanos				**11400,00 €**
	Ingeniero	15 €/hora	760 horas	11400,00 €
Hardware				**1062,99 €**
	Ordenador HP Pavilion dv5 **	448,99 €	1	448,99 €
	AR Drone 2.0	289,00 €	1	289,00 €
	AR Drone 2.0 Elite Edition	289,00 €	1	289,00 €
Software				**506,00 €**
	Sistema operativo Windows 8	0,00 €	1	0,00 €
	AR Drone SDK	0,00 €	1	0,00 €
	Microsoft Office 2010	119,00 €	1	119,00 €
	Microsoft Visual Studio 2013	387,00 €	1	387,00 €
	Librerías Emgu universal gpu 2.4.9	0,00 €	1	0,00 €
Total				**12968,99 €**

Tabla 20. Presupuesto del proyecto

* El desarrollo del software y documentación Post Mortem tiene días que se ha trabajado en ambas partes de manera conjunta, por lo que en el presupuesto de estas partes hay un total de 114 días en lugar de 125.

** Respecto al hardware se recomienda usar para futuros trabajos un ordenador con mínimo 4 GB de memoria RAM como el utilizado para este proyecto. En el caso de trabajar con las cámaras u otros elementos que puedan saturar el sistema como Matlab se aconseja usar un ordenador más potente para no sobrecargar el sistema y así evitar errores.

A.-Bibliografía

[1] ROBINSON, S., NAGEL, C., GLYNN J., SKINNER, M., WATSON, K. y EVJEN, B. *Professional C#,* ed 3. Indianapolis: Wrox, 2004. ISBN: 0-7645-5759-9.

[2] SHARP, John. *Microsoft Visual C# 2010 Step by step*. Washington: Microsoft, 2010. Library of Congress Control Number: 2009939912.

[3] MORENO, L, GARRIDO, S. y BALAGUER, C. *Ingeniería de control: modelado y control de sistemas dinámico.* Barcelona: Ariel, 2003. ISBN: 9788434480551

[4] Guía de programación de C#, 2014. [consulta: 18-10-2014]. Disponible en: http://msdn.microsoft.com/es-es/library/67ef8sbd.aspx

[5] GARCÍA ALONSO, I. *Software de control en sistemas simulados y con dispositivos reales* [en línea]. Colmenarejo: Universidad Carlos III de Madrid, 2012. [consulta: 01-07-2014]. Disponible en: http://hdl.handle.net/10016/14677

[6] PEREDA GARCIMARTÍN, Fernando J. *Sistema de telemetría y control de un barco autónomo* [en línea]. Colmenarejo: Universidad Carlos III de Madrid, 2010. [consulta: 05-07-2014]. Disponible en: http://hdl.handle.net/10016/9819

[7] MONZÓN CATALÁN, I. *Desarrollo de un cuadricóptero operado por ROS* [en línea]. Zaragoza: Universidad de Zaragoza, 2013. [consulta: 01-07-2014]. Disponible en: http://zaguan.unizar.es/record/11998?ln=es

[8] KRAJNÍK, T., VONÁSEK, V., FIŠER, D. y FAIGL, J. AR-Drone as a Platform for Robotic Research and Education. En: *Research and Education in Robotics-EUROBOT 2011* [en línea]. David Obdržálek (ed.). Berlin: Springer Heidelberg, 2011, pp. 172-186. [consulta: 08-02-2015]. Communications in Computer and Information Science, vol. 161. Disponible en: http://link.springer.com/chapter/10.1007%2F978-3-642-21975-7_16

[9] CHAMORRO HERNÁNDEZ, W.O. y MEDINA MORA, J. L. *Ensamblaje y control de un cuadricóptero* [en línea]. Quito: Escuela Politécnica Superior, 2013. [consulta: 10-07-2014]. Disponible en: bibdigital.epn.edu.ec/bitstream/15000/6168/1/CD-4822.pdf

[10] AR Drone España, 2014.[foro] [consulta: 25-06-2014]. Disponible en: http://www.ardronespain.com/foros/

[11] Web oficial Parrot AR Drone 2.0, 2015. [consulta: 25-06-2014]. Disponible en: http://ardrone2.parrot.com/

[12] Wikipedia:

Vehículo aéreo no tripulado. [consulta: 15-07-2014]. Disponible en: https://es.wikipedia.org/wiki/Vehículo_aéreo_no_tripulado

Ángulos de navegación. [consulta: 15-07-2014]. Disponible en: http://es.wikipedia.org/wiki/Ángulos_de_navegación

[13] ARDrone open API platform, 2009. [foro] [consulta: 15-09-2014]. Disponible en: https://projects.ardrone.org/projects

[14] AutonomyLab AR Drone. GitHub, 2014. [foro] [consulta: 15-09-2014]. Disponible en: https://github.com/AutonomyLab/ardrone_autonomy

[15] Guía para desarrolladores SDK 2.0. [consulta: 19-09-2914]. Disponible en: http://www.msh-tools.com/ardrone/ARDrone_Developer_Guide.pdf

[16] España. Ministerio de Fomento. [revisado: 17-10-2014]. *Marco regulatorio temporal para operaciones con drones,* 2014. BOE, viernes 17 de octubre de 2014, Sección 6ª Aeronaves civiles pilotadas por control remoto, artículos 50 y 51. [consulta: 04-02-2015]. Disponible en: http://www.seguridadaerea.gob.es/lang_castellano/cias_empresas/trabajos/marco_drones/default.aspx

[17] Geogebra. Science Technology Engineering & Mathematics, 2015. [programa informático online]. Disponible en: http://www.geogebra.org/

[18] Cómo citar bibliografía: UNE-ISO 690, 2015. [consulta: 16-02-2015]. Disponible en: http://portal.uc3m.es/portal/page/portal/biblioteca/aprende_usar/como_citar_bibliografia

B.-Anexos

❖ Anexo I: Manual de instalación en Windows

En este manual se explica cómo instalar la interfaz LSIDrone y el software creado en nuestro proyecto para poder compilar el entorno de manera correcta. En el Dvd adjunto a este proyecto, entregado a los profesores, se encuentran los archivos necesarios para ello.

1) En primer lugar deberemos instalar el programa Microsoft Visual Studio 2013 en este caso, o si se desea la última versión que haya disponible en el momento.

Figura 60. Logotipo Microsoft Visual Studio

2) Lo siguiente será descomprimir el archivo LSIDroneInterface, y una vez realizado esto nos encontraremos con las siguientes carpetas:

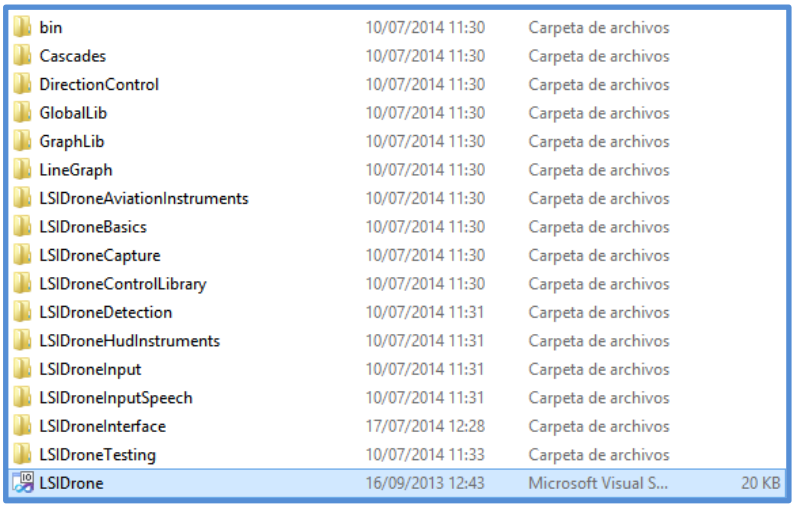


Figura 61. Archivos del proyecto

3) Instalamos el archivo ejecutable DXSDK_Jun10:

Figura 62.
Ejecutable SDK

4) Por último tendremos que copiar las librerías Emgu a nuestro disco duro(:C) para poder compilar nuestro programa:

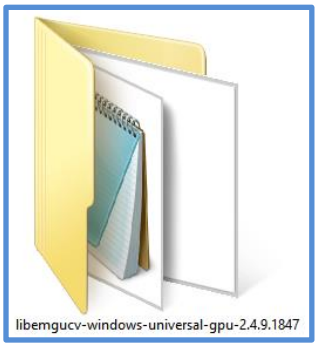


Figura 63. Librerías Emgu

En este caso se ha utilizado la versión gpu-2.4.9.1847, aunque se pueden descargar versiones más nuevas si están disponibles, de la siguiente página web donde se encuentran todas las existentes:

http://www.emgu.com/wiki/index.php/Version_History

❖ Anexo II: Encender y conectar el cuadricóptero

Para encender nuestro cuadricóptero tendremos que conectar la batería al drone a través de los conectores de carga/descarga, como podemos observar a continuación:

Figura 64. Encendido de nuestro cuadricóptero

Una vez conectado se encenderán los 4 leds del cuadricóptero en color rojo lo que indica que se está inicializando la activación de los circuitos. Procederemos a colocar la carcasa, preferiblemente la de interior para evitar daños en el cuadricóptero. Tras unos segundos los leds se pondrán de color verde, y se generará una señal Wi-Fi a la cual nos conectaremos a través de nuestro ordenador, antes de compilar el programa.

Figura 65. Conexión Wi-Fi

❖ Anexo III: Compilar programa

Para compilar nuestro programa y poder utilizar nuestra interfaz, tanto el modo manual como automático deberemos realizar los siguientes pasos, una vez instalado todo nuestros archivos como se indica en el anexo I:

i. Abriremos el proyecto, que se encuentra en la carpeta descomprimida LSIDroneInterface, haciendo doble click sobre el archivo LSIDrone como en la imagen:

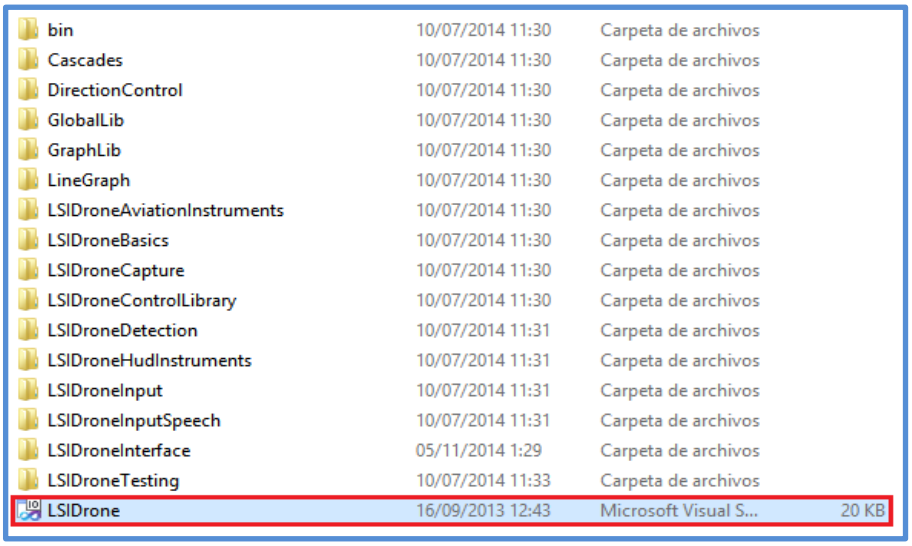


Figura 66. Apertura del proyecto

ii. Se mostrará nuestro proyecto en Visual Studio, y en la ventana del explorador haremos click en START:

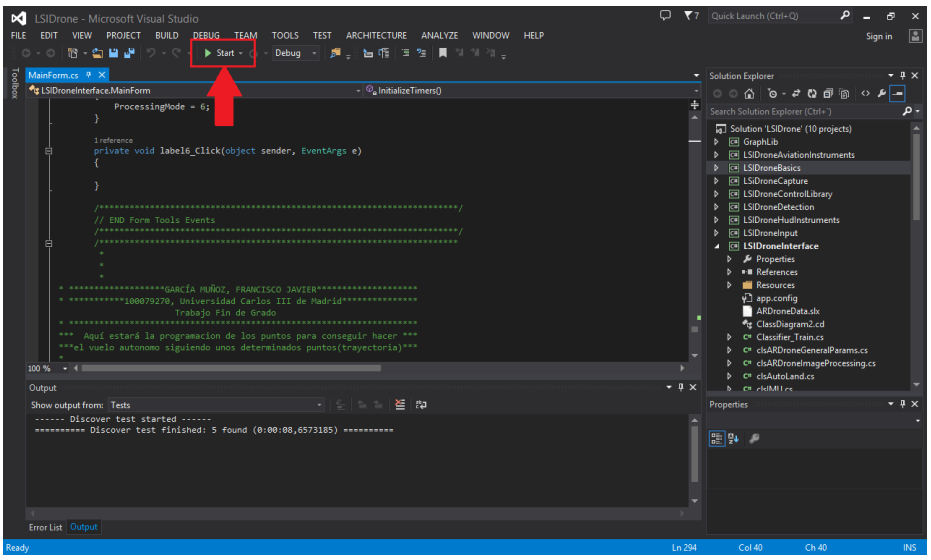


Figura 67. Compilación del proyecto

Si nuestro ordenador se encuentra conectado a la red Wi-Fi del cuadricóptero compilará nuestra interfaz de manera correcta, y en caso de no estarlo nos dará un error, que es el siguiente:

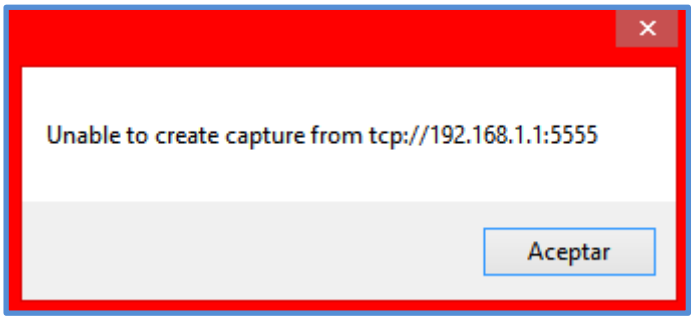


Figura 68. Error Wi-Fi

iii. Tras compilar nuestra interfaz, deberemos conectarla al cuadricóptero haciendo click en el botón conectar:

Figura 69. Botón conectar PC-Drone

Una vez realizados estos pasos correctamente, aparecerá el video de la cámara del cuadricóptero en nuestra interfaz, lo que indica que está todo listo para comenzar a volar. Debemos tener en cuenta que si la batería tiene un porcentaje igual o inferior al 20% de su capacidad, valor que podemos observar en nuestra interfaz, el cuadricóptero no podrá despegar y deberemos cargarla o reemplazarla por otra.

iv. Una vez realizados los pasos anteriores ya podremos comenzar a realizar vuelos manuales utilizando la botonera manual o seguir trayectorias de manera autónoma directamente pulsando cualquiera de los dos botones automáticos dependiendo del movimiento que deseemos.

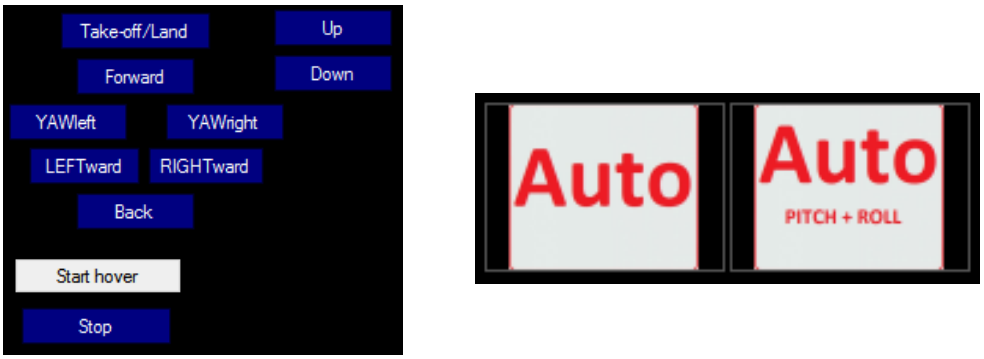


Figura 70. Botonera manual y automática

❖ Anexo IV: Manual Parrot AR Drone 2.0 en Android

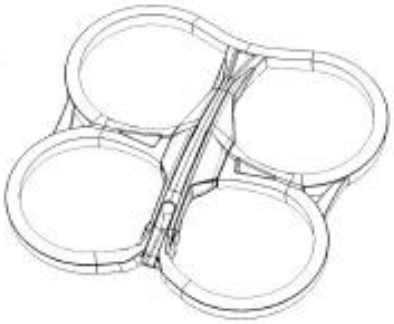
Primera utilización

Antes de comenzar

Nota : Para facilitar la comprensión, los teléfonos o tablets compatibles se denominarán con el término de "smartphone" en este manual.

Contenido del embalaje

AR.Drone 2.0



Carcasa para interior

Carcasa para exterior

Cargador

Batería

Adhesivos

Nota : Quite los adhesivos de protección del AR.Drone 2.0, la cámara y las 2 carcasas.

Dispositivo de detección

Conserve el dispositivo de detección incluido en el embalaje. Deberá utilizarlo para jugar a los juegos AR.Race 2 et AR.Rescue 2.

Descargando la aplicación

Conéctese a Google Play™ y descargue la aplicación gratuita *AR.FreeFlight 2.0*.

Batería

Recargar la batería

1. Seleccione el adaptador correspondiente a su país e introdúzcalo en el cargador.
2. Conecte la batería al cargador y enchufe el cargador a la red eléctrica.

El tiempo de carga de la batería es de 1 hora 30 minutos.

Nota: Las baterías del AR.Drone 2.0 funcionan con el AR.Drone y las baterías del AR.Drone funcionan con el AR.Drone 2.0. Sin embargo, el cargador del AR.Drone 2.0 no permite recargar las baterías del AR.Drone y el cargador del AR.Drone no permite recargar las baterías del AR.Drone 2.0.

Colocar la batería

1. Coloque la batería en el alojamiento previsto para ella.
2. Asegúrese de que quede correctamente sujeta utilizando el sistema de agarre.
3. Conéctela al AR.Drone.

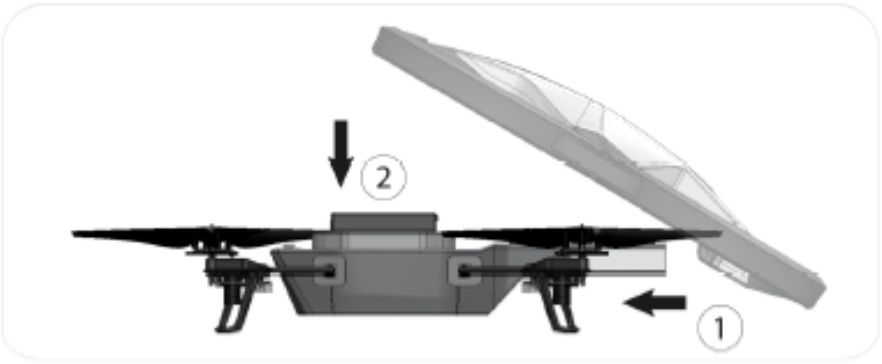
Comprobar el nivel de batería del smartphone

Para utilizar el AR.Drone en condiciones óptimas, asegúrese de que la batería del smartphone está bien cargada. Un nivel de batería bajo o medio puede hacer que el rendimiento del AR.Drone sea menor (en la reactividad de los mandos, la calidad del flujo de vídeo...).

Utilización en el interior

Advertencia: Evite que el AR.Drone 2.0 vuele cerca de niños, animales domésticos u objetos frágiles.

- Instale la carcasa interior para proteger el AR.Drone 2.0 en caso de que choque con algún objeto.



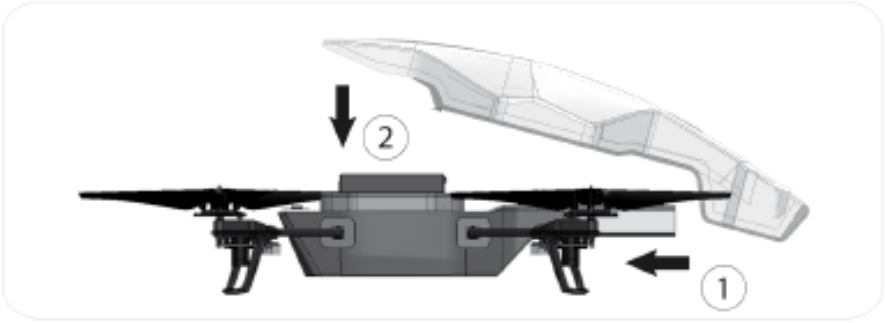
- Coloque el AR Drone en el centro de una habitación (de 4m x 4m como mínimo), sin obstáculos. Manténgase 1 metro detrás del aparato (la parte delantera del aparato es fácilmente

identificable gracias a la cámara).

- En los ajustes de la aplicación *AR.FreeFlight 2.0*, desactive las opciones Outdoor Hull y Outdoor Flight.

Utilización en el exterior

- Instale la carcasa exterior.



- Coloque el AR.Drone 2.0 sobre una superficie plana y seca, en una zona sin obstáculos. Manténgase 1 metro detrás del aparato (la parte delantera del aparato es fácilmente identificable gracias a la cámara).
- En los ajustes de la aplicación *AR.FreeFlight 2.0*, active las opciones Outdoor Hull y Outdoor Flight.

- No utilice el Parrot AR.Drone 2.0 con malas condiciones meteorológicas (lluvia, fuerte viento o

nieve) o cuando las condiciones de visibilidad sean insuficientes (noche). Aunque el piloto automático compensa las turbulencias que se puedan producir por el viento, evite que el AR.Drone vuele si la velocidad de viento supera los 15Km/h.

Note: Recuerde siempre que la fuerza del viento que nota en el lugar en el que está pilotando el AR.Drone puede ser muy diferente a la fuerza del viento que hay en el lugar en el que está el AR.Drone. Puede ser la razón de los cambios de trayectoria no deseados.

Conexión

Conectar el smartphone al AR.Drone 2.0

1. Retire la carcasa e introduzca la batería en el AR.Drone 2.0. Asegúrese de que la batería esté sujeta y conéctela al AR.Drone 2.0.
2. Espere a la inicialización de los motores.
3. Lance en el smartphone una búsqueda de redes Wi-Fi disponibles. Si utiliza un teléfono Android, seleccione Parámetros > Conexiones inalámbricas y redes > Wi-Fi.
4. Seleccione "ardrone2_parrot".
5. Espere hasta que el smartphone se conecte a la Wi-Fi del AR.Drone 2.0. Esta conexión está representada generalmente con la aparición del logotipo Wi-Fi en la pantalla del smartphone.
6. Inicie la aplicación *AR.FreeFlight 2.0*.

> Aparecerá la pantalla de inicio. Está conectado.

Nota : Una vez establecida la conexión entre el smartphone y el AR.Drone, ambos aparatos quedan automáticamente emparejados. El AR.Drone sólo se podrá utilizar con el smartphone que acaba de conectar. Para utilizar el AR.Drone con otro smartphone, consulte la sección Utilizar el AR.Drone 2.0 con varios smartphone.

Utilizar el AR.Drone 2.0 con varios smartphones

Si desea utilizar el AR.Drone 2.0 con un smartphone distinto al primer smartphone conectado al AR.Drone 2.0, la opción Emparejamiento debe estar desactivada en los ajustes de la aplicación *AR.FreeFlight 2.0*. Si la opción Emparejamiento está activada, no será posible conectar ningún otro smartphone a su AR.Drone 2.0.

Para desactivar la opción Emparejamiento :

1. Abra *AR.FreeFlight 2.0* y pulse DESPEGAR.
2. Pulse  .
3. Desactive la opción Emparejamiento.

LEDs

Los LEDs (Diodos Electroluminiscentes) corresponden a las pequeñas luces rojas o verdes. Podemos distinguir 2 tipos de LEDs en el AR.Drone:

- los 4 LEDs de motores que se encuentran en las hélices;

- el LED del sistema, que se encuentra debajo del AR.Drone.

LEDs de motores

Comportamiento de los LEDs	Significado
Los 4 LEDs están en rojo	Puesta en funcionamiento Ha ocurrido un problema
Cada LED parpadea de color rojo, uno tras otro	Los motores se están inicializando.
Los 4 LEDs parpadean en verde	El AR.Drone 2.0 está despegando o aterrizando.
Los 2 LEDs delanteros están en verde	El AR.Drone 2.0 está volando. Estos colores le permitirán distinguir fácilmente la parte trasera y delantera del AR.Drone 2.0 (cuando esté lejos del aparato).
Los 2 LEDs traseros están en rojo	Ha ocurrido un problema

LED del sistema

Espere 20 segundos después de conectar la batería al AR.Drone 2.0 y compruebe después el color del LED del sistema.

Nota: Evite girar el AR.Drone para comprobar el color del LED del sistema. Es preferible que levante el aparato.

Si el LED está verde, ya puede despegar.

Si transcurridos 20 segundos, el LED está rojo o naranja, desconecte y vuelva a conectar la batería.

Si tiene algún problema

Comprobar la dirección IP del AR.Drone 2.0

Compruebe que la dirección IP aparece en la red del AR.Drone 2.0. Para ello, seleccione Parámetros > Conexiones inalámbricas y redes > Parámetros Wi-Fi y seleccione la red "ardrone2".

En el campo IP debe haber una dirección IP que comience por 192.168.1.X.

Si la dirección IP comienza por 169, no se podrá establecer la conexión entre el smartphone y el AR.Drone.

Esta dirección IP incorrecta puede deberse a tres problemas. Consulte nuestra sección de preguntas frecuentes (FAQ) para más información.

Desactivar la opción Emparejamiento

Si desea utilizar el AR.Drone 2.0 con un smartphone distinto al primer smartphone conectado al AR.Drone 2.0, la opción Emparejamiento debe estar desactivada en los ajustes de la aplicación *AR.FreeFlight 2.0*. Si la opción Emparejamiento está activada, no será posible conectar ningún otro smartphone a su AR.Drone 2.0.

Para desactivar la opción Emparejamiento :

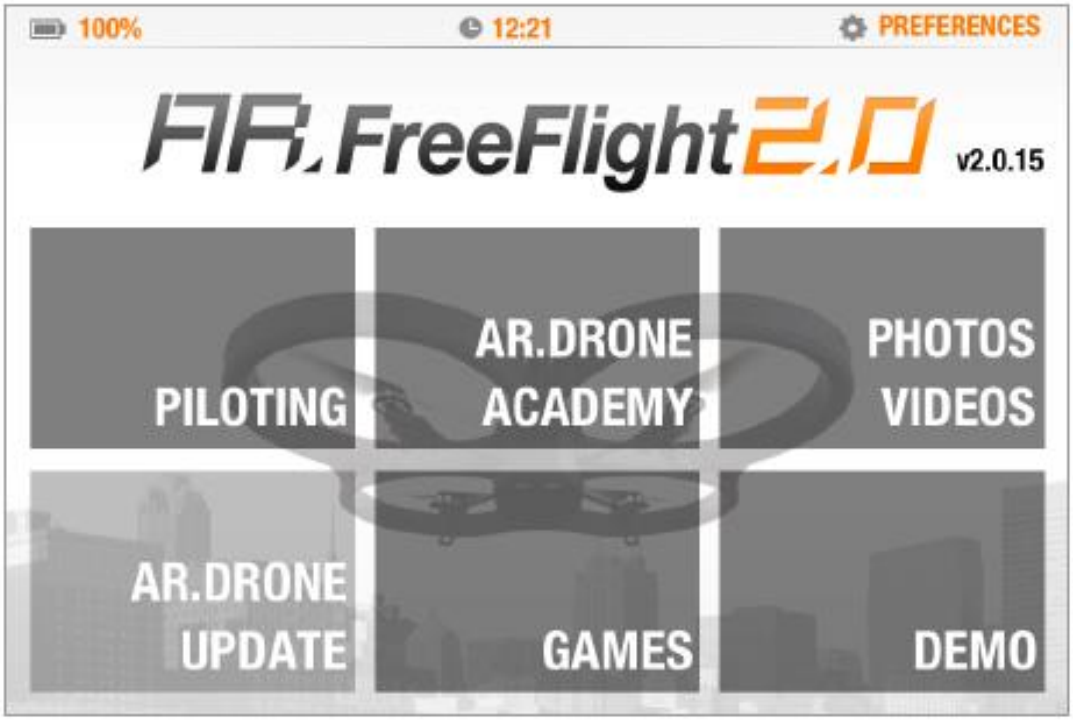
1. Abra *AR.FreeFlight 2.0* y pulse DESPEGAR.
2. Pulse  .
3. Desactive la opción Emparejamiento.

Volver a arrancar el AR.Drone 2.0

Para volver a arrancar el AR.Drone, desconecte y vuelva a conectar la batería.

Vuelo libre

AR.FreeFlight 2.0 es la aplicación básica: permite pilotar el AR.Drone 2.0 desde su smartphone. Le permitirá aprender a realizar los movimientos básicos (subir, bajar, girar, retroceder, avanzar, etc.).



Nota: Asegúrese de que ha conectado la batería del AR.Drone 2.0 y de que los LEDs de los motores están en verde antes de conectar el smartphone a la red Wi-Fi del AR.Drone 2.0 y de abrir una aplicación.

Antes de comenzar

Posicionamiento del smartphone

Por encima de un cierto umbral (~90° con respecto a la horizontal), el acelerómetro del smartphone no tiene en cuenta los movimientos.

Le recomendamos que mantenga el smartphone en una posición cercana a la horizontal para evitar tener que adoptar posiciones incómodas y poco eficaces cuando quiera que el AR.Drone 2.0 retroceda rápidamente.

Se toma como referencia la posición del smartphone en el momento en el que pulse el botón izquierdo.

Posición de las manos en el smartphone

Como coloque las manos en el smartphone puede influir en la calidad de la transmisión Wi-Fi.

Colores de los LEDs

Compruebe el color del LED del sistema que se encuentra debajo del AR.Drone.

Ajustes

1. Abra la aplicación *AR.FreeFlight 2.0* y pulse PILOTAR.
2. Pulse el botón .
3. Asegúrese de que el AR.Drone 2.0 esté colocado sobre una superficie plana y pulse .
4. Seleccione la carcasa que tiene el AR.Drone 2.0 (interior o exterior) y después el tipo de vuelo (interior o exterior) que desea efectuar.

Pilotaje

Abra la aplicación *AR.FreeFlight 2.0* y pulse PILOTAR para utilizar el AR.Drone 2.0 en modo de vuelo libre.

> La señal de vídeo de la cámara frontal del AR.Drone 2.0 aparecerá en la pantalla de su smartphone.

Interfaz

Modo mando Analógico

1 - Despegue

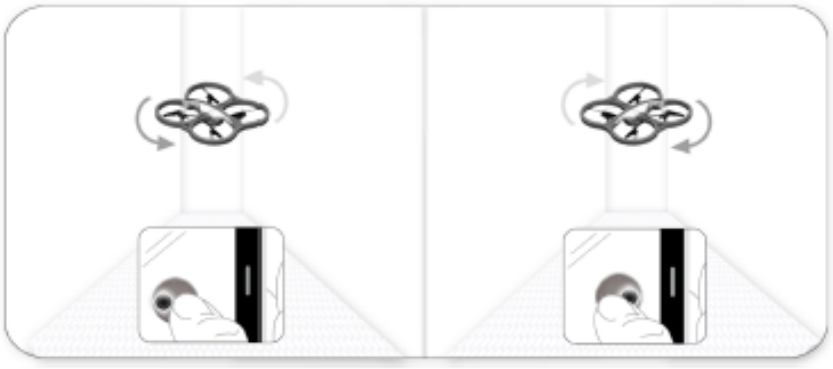
Pulse el botón 

.

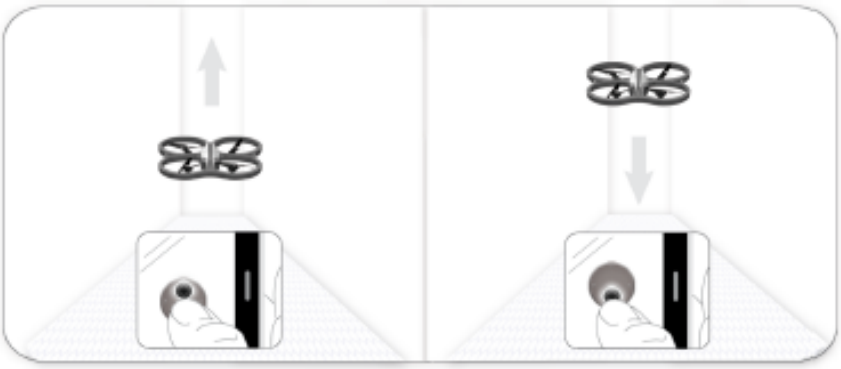
Los motores arrancan y el AR.Drone se coloca automáticamente a una altitud de entre 50 cm y 1 metro.

2 - Desplazamientos

Mueva el joystick hacia la derecha para dirigir el AR.Drone 2.0 hacia la derecha. Mueva el joystick hacia la izquierda para dirigir el AR.Drone 2.0 hacia la izquierda.



Mueva el joystick hacia arriba para hacer que el AR.Drone 2.0 suba. Mueva el joystick hacia abajo para hacer que el AR.Drone 2.0 baje.



Los demás desplazamientos dependen del estado de las opciones Modo Joypad y Control absoluto.

3 - Batería

El indicador de carga de batería en la pantalla del smartphone indica el nivel de batería del AR.Drone.

La autonomía es de unos 12min. El indicador pasa al rojo cuando la batería está baja.

Si el nivel de batería es demasiado bajo, es mejor que haga aterrizar el AR.Drone; de lo contrario el AR.Drone aterrizará automáticamente.

Advertencia : Para evitar cualquier riesgo de contacto entre el AR.Drone y una persona, un animal doméstico o un objeto, le recomendamos que haga aterrizar el AR.Drone manualmente si ve que se

enciende el indicador de batería baja.

4 - Estado de la red

El icono indica el estado de la conexión Wi-Fi entre su smartphone y el AR.Drone 2.0. El número de barras será proporcional a la calidad de la señal.

5 - Ajustes

Pulse el botón para acceder a los ajustes del AR.Drone 2.0. Consulte la sección Ajustes para más información.

6 - Cambio de cámara

Pulse el mando para cambiar la vista del AR.Drone 2.0.

7 - Foto

Pulse el botón para hacer una foto.

Más tarde podrá recuperar las fotos seleccionando FOTOS VÍDEOS en el menú principal. Consulte la sección Gestionar fotos y vídeos para más información.

8 - Vídeo

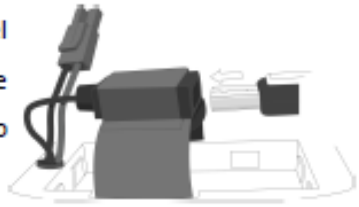
Pulse el botón grabar un vídeo. Los vídeos se pueden grabar en la memoria de su smartphone o en un dispositivo USB conectado al AR.Drone 2.0.

Nota: para conservar una conexión Wi-Fi de buena calidad, es aconsejable grabar los vídeos en un dispositivo USB.

Para grabar sus vídeos en un dispositivo USB:

1. Conecte un dispositivo USB al AR.Drone 2.0. Asegúrese de que el dispositivo USB conectado al AR.Drone 2.0 dispone de más de 100 Mb de espacio libre: no se podrá grabar el vídeo si el espacio disponible es inferior o igual a 100 Mb.

   Nota: un dispositivo USB de 128 Mb permite guardar una grabación de unos 30 segundos.



2. En los ajustes de la aplicación *AR.FreeFlight 2.0*, active la opción Grabación USB.

El contador situado al lado del logo USB indica el tiempo de grabación restante. Este contador cambia a color rojo cuando quedan sólo 30 segundos de grabación. Cuando el espacio libre del dispositivo USB conectado al AR.Drone 2.0 sea insuficiente para grabar un vídeo, aparecerá un mensaje de error.

Para grabar vídeos en su smartphone: desactive la opción Grabación USB en los ajustes de la aplicación *AR.FreeFlight 2.0*.

Más tarde podrá encontrar los vídeos grabados en la memoria del smartphone seleccionando FOTOS VÍDEOS en el menú principal. Consulte la sección Gestionar fotos y vídeos para más información.

Nota: también puede recuperar estos vídeos en formato .mov conectando su smartphone a un ordenador.

9 - Inicio

Pulse el botón  para volver al menú principal. El AR.Drone 2.0 se estabilizará automáticamente.

Pulse PILOTAR para recuperar el control del AR.Drone 2.0.

10 - Botón Emergencia

Si tiene algún problema al pilotar el AR.Drone, pulse el botón 

.

Advertencia: Pulse el botón 

únicamente en caso de emergencia. Se apagarán los motores y el AR.Drone 2.0 caerá sin importar la altura a la que se encuentre. En la mayoría de los

casos, basta con aterrizar el AR.Drone 2.0.

11 - Altitud

El icono  indica la altitud a la que vuela el AR.Drone 2.0.

12 - Velocidad

El icono  indica la velocidad a la que vuela el AR.Drone 2.0.

Aterrizaje

Asegúrese de que el AR.Drone 2.0 esté encima de una superficie plana, seca y sin obstáculos, y pulse el mando 

.

Piloto automático

- Si retira el dedo del smartphone, el piloto automático estabiliza el AR.Drone 2.0 y lo pone en vuelo estacionario.
- Si recibe una llamada o un SMS mientras está utilizando el AR.Drone 2.0, éste aterrizará automáticamente. Si acepta la llamada o si abre el SMS, se cerrará la aplicación. Si rechaza la llamada o no abre el SMS, puede continuar utilizando el AR.Drone 2.0.
- Si la distancia entre el AR.Drone 2.0 y el smartphone se hace excesivamente elevada (más de unos 50 metros, según el entorno Wi-Fi), los dos aparatos se pueden desconectar. Si esto ocurre, no cierre la aplicación en el smartphone y acérquese al AR.Drone 2.0.

Mensajes de error

Consulte nuestra sección de preguntas frecuentes (FAQ) para más información sobre los mensajes de error del AR.Drone 2.0.

❖ Anexo V: Código algoritmo 1 desarrollado

1. Puntos trayectoria, variables y contadores necesarios

```
///Número de puntos que se cargarán

   int numPuntos = 5;

///Arrays de puntos de nuestra trayectoria

        float[] x = { 0.0f, 5.0f, 5.0f, 0.0f, 0.0f };
        float[] y = { 0.0f, 0.0f, -5.0f, -5.0f, 0.0f };
///Variables para medir el tiempo transcurrido en cada acción y cambiar a otras

        int tmrcountr = 0;
        int timerACC1 = 0;
        int timerACC2 = 0;
        int timerACC3 = 0;
        int timerACC4 = 0;
        int timerACC5 = 0;
        int timerACC6 = 0;
        int timerACC7 = 0;
        int timerACC8 = 0;
        int timerACC9 = 0;
///La variable diferenciagiro recoge el valor del giro anterior y hace la diferencia para el
punto siguiente.
        float diferenciagiro = 0;
        float dYaw_comprobacion = 0;
        int i = 1;
```

2. Función principal del control. Algoritmo 1

```
private void tmrTestControl2_Tick(object sender, EventArgs e)
{
```

2.1 Análisis controles de la función Navigate

```
    ///Parámetros generales necesarios para calcular nuestro algoritmo y poder seguir la trayectoria

        float[] distancia;
        distancia = new float[numPuntos];

        float[] tiempoflying;
        tiempoflying = new float[numPuntos];

        float[] beta;
        beta = new float[numPuntos];

        float[] giro;
        giro = new float[numPuntos];

        Controlroll = (float)Math.Round((KPRoll * (dVy -
            (float)(clsARDroneGeneralParams.clsVy))), 5);
        Controlpitch = (float)Math.Round((KPPitch * (dVx -
            (float)(clsARDroneGeneralParams.clsVx))), 5);
```

```
///Programación del control para realizar los giros: Controlyaw

    if (dYaw < 0)
    {
        dYaw = dYaw + 360;
    }
    Yawdiff = (dYaw - (float)(clsARDroneGeneralParams.clsYaw));

    if (dYaw <= 180)
    {
        Controlyaw = (float)Math.Round((KPYaw * dYaw), 5);
    }

    ///para que vaya hacia la izquierda
    if (dYaw > 180)
    {
        dYaw = -(360 - dYaw);
        Controlyaw = (float)Math.Round((KPYaw * dYaw), 5);
    }

    Controlgaz = (float)Math.Round((KPH * (dH -
        clsARDroneGeneralParams.clsintAltitude)), 3);

    label8.Text = dYaw.ToString();
    label6.Text = Yawdiff.ToString();

    if (Controlgaz > 0.9f) { Controlgaz = 0.9f; }
    if (Controlgaz < -0.9f) { Controlgaz = -0.9f; }

    if (Controlyaw > 1.0f) { Controlyaw = 1.0f; }
    if (Controlyaw < -1.0f) { Controlyaw = -1.0f; }


    if (boolpitch == true)
    {
        Controlyaw = 0;
        if (Controlpitch > 0.05f) { Controlpitch = 0.05f; }
        if (Controlpitch < -0.05f) { Controlpitch = -0.05f; }
    }
    if (boolpitch == false)
    {
        if (Controlpitch > 0.02f) { Controlpitch = 0.02f; }
        if (Controlpitch < -0.02f) { Controlpitch = -0.02f; }
    }

///En este control no variaremos la altitud del drone, ni habrá desplazamientos laterales (roll)

    Controlroll = 0.0f;
    Controlgaz = 0.0f;

///Función encargada del movimiento de nuestro cuadricóptero

    Navigate(Controlroll, Controlpitch, Controlyaw, Controlgaz);

///Muestra los valores de la función Navigate() en la interfaz
    SetTextRollInput("Roll: " + Controlroll);
    SetTextPitchInput("Pitch: " + Controlpitch);
    SetTextYawInput("Yaw: " + Controlyaw);
    SetTextGazInput("Gaz: " + Controlgaz);

    tmrcountr = tmrcountr + 1;
    labeltmrcountr.Text = (tmrcountr).ToString();

///Para el primer punto (i=0) estos términos son 0.
    giro[0] = 0.0f;
    distancia[0] = 0.0f;
    tiempoflying[0] = 0.0f;
    beta[0] = 0.0f;
    labelControlyaw.Text = Controlyaw.ToString();
```

```
labeldiferenciagiro.Text = diferenciagiro.ToString();
```

2.2 Loop de control y análisis de todos los casos

```
///Programación de las trayectorias: algoritmo 1-->tmrTestControl2
    while (i < numPuntos)
    {
        distancia[i] = (float)(Math.Sqrt(Math.Pow((x[i] - x[i - 1]), 2) + Math.Pow((y[i]
                - y[i - 1]), 2)));
        tiempoflying[i] = (2 * (1000 * distancia[i] / 500.0f));
        beta[i] = (float)((Math.Atan((x[i] - x[i - 1]) / (y[i] - y[i - 1])) * (180 /
                Math.PI)));
        labeldistancia.Text = distancia[i].ToString();
        labeltiempoflying.Text = tiempoflying[i].ToString();
        label_i_j.Text = i.ToString();
        ///9 COMBINACIONES POSIBLES: todo detallado en la memoria de este TFG
        //CASO 1
        if (x[i] < x[i - 1] && y[i] > y[i - 1])
        {
            timerACC1++;
            giro[i] = 90.0f - beta[i];
            labelgiro.Text = giro[i].ToString();
            ///3.14 segundos para realizar todos los giros; detallado el cálculo en la
                memoria.
            if (timerACC1 >= 8 && timerACC1 < 11.14)
            {
                LeaveHoverMode();
                dH = 1000;
                dVx = 0.0f; dVy = 0.0f;
                dYaw = giro[i] - diferenciagiro;
                UpdateUIAsync("Girando caso 1...");
                boolpitch = false;
                boolroll = false;
                boolYaw = true;
                tmrTestControl2.Start();
            }
            else if (timerACC1 >= 11.14 && timerACC1 < 16)
            {
                EnterHoverMode();
            }
            else if (timerACC1 >= 16 && timerACC1 < (16 + tiempoflying[i]))
            {
                btnForwardAuto.PerformClick();
            }
            else if (timerACC1 >= (16 + tiempoflying[i]))
            {
                EnterHoverMode();
                goto sumamospunto;
            }
            break;
        }
         ///CASO 2
        else if (x[i] > x[i - 1] && y[i] > y[i - 1])
        {
            timerACC2++;
            giro[i] = 90 - beta[i];
            labelgiro.Text = giro[i].ToString();

            if (timerACC2 >= 8 && timerACC2 < 11.14)
            {
                LeaveHoverMode();
                dH = 1000;
                dVx = 0.0f; dVy = 0.0f;
                dYaw = giro[i] - diferenciagiro;
```

```
            UpdateUIAsync("Girando caso 2...");
            boolpitch = false;
            boolroll = false;
            boolYaw = true;
            tmrTestControl2.Start();
        }
        else if (timerACC2 >= 11.14 && timerACC2 < 16)
        {
            EnterHoverMode();
        }
        else if (timerACC2 >= 16 && timerACC2 < (16 + tiempoflying[i]))
        {
            btnForwardAuto.PerformClick();
        }
        else if (timerACC2 >= (16 + tiempoflying[i]))
        {
            EnterHoverMode();
            goto sumamospunto;
        }
        break;
    }
    ///CASO 3
    else if (x[i] < x[i - 1] && y[i] < y[i - 1])
    {
        timerACC3++;
        giro[i] = -90 - beta[i];
        labelgiro.Text = giro[i].ToString();

        if (timerACC3 >= 8 && timerACC3 < 12)
        {
            LeaveHoverMode();
            dH = 1000;
            dVx = 0.0f; dVy = 0.0f;
            dYaw = giro[i] - diferenciagiro;
            UpdateUIAsync("Girando caso 3...");
            boolpitch = false;
            boolroll = false;
            boolYaw = true;
            tmrTestControl2.Start();
        }
        else if (timerACC3 >= 12 && timerACC3 < 16)
        {
            EnterHoverMode();
        }
        else if (timerACC3 >= 16 && timerACC3 < (16 + tiempoflying[i]))
        {
            btnForwardAuto.PerformClick();
        }
        else if (timerACC3 >= (16 + tiempoflying[i]))
        {
            EnterHoverMode();
            goto sumamospunto;
        }
        break;
    }
    ///CASO 4
    else if (x[i] > x[i - 1] && y[i] < y[i - 1])
    {
        timerACC4++;
        giro[i] = -90 - beta[i];
        labelgiro.Text = giro[i].ToString();

        if (timerACC4 >= 8 && timerACC4 < 12)
        {
            LeaveHoverMode();
            dH = 1000;
            dVx = 0.0f; dVy = 0.0f;
            dYaw = giro[i] - diferenciagiro;
            UpdateUIAsync("Girando caso 4...");
```

```
            boolpitch = false;
            boolroll = false;
            boolYaw = true;
            tmrTestControl2.Start();
        }
        else if (timerACC4 >= 12 && timerACC4 < 16)
        {
            EnterHoverMode();
        }
        else if (timerACC4 >= 16 && timerACC4 < (16 + tiempoflying[i]))
        {
            btnForwardAuto.PerformClick();
        }
        else if (timerACC4 >= (16 + tiempoflying[i]))
        {
            EnterHoverMode();
            goto sumamospunto;
        }
        break;
    }
    ///CASO 5
    else if (x[i] == x[i - 1] && y[i] == y[i - 1])
    {
        timerACC5++;
        giro[i] = 0.0f;
        ///Se ha asignado a 0, porque en la interfaz daba valor NAN que puede perder al compilador
        labelgiro.Text = giro[i].ToString();

        if (timerACC5 >= 8 && timerACC5 < 12)
        {
            LeaveHoverMode();
            dH = 1000;
            dVx = 0.0f; dVy = 0.0f;
            dYaw = giro[i] - diferenciagiro;
            UpdateUIAsync("Orientando caso 5,mismo punto...");
            boolpitch = false;
            boolroll = false;
            boolYaw = true;
            tmrTestControl2.Start();
        }
        else if (timerACC5 >= 12)
        {
            EnterHoverMode();
            goto sumamospunto;
        }
        break;
    }
    ///CASO 6
    else if (x[i] < x[i - 1] && y[i] == y[i - 1])
    {
        timerACC6++;
       ///En este caso el Arcotangente da error al dividir por 0, y sabemos que el giro será 180.
        giro[i] = 180.0f;
        labelgiro.Text = giro[i].ToString();
        if (timerACC6 >= 8 && timerACC6 < 11.14)
        {
            LeaveHoverMode();
            dH = 1000;
            dVx = 0.0f; dVy = 0.0f;
            dYaw = giro[i] - diferenciagiro;
            UpdateUIAsync("Girando caso 6...");
            boolpitch = false;
            boolroll = false;
            boolYaw = true;
            tmrTestControl2.Start();
        }
        else if (timerACC6 >= 11.14 && timerACC6 < 16)
        {
            EnterHoverMode();
```

```
    }
    else if (timerACC6 >= 16 && timerACC6 < (16 + tiempoflying[i]))
    {
        btnForwardAuto.PerformClick();
    }
    else if (timerACC6 >= (16 + tiempoflying[i]))
    {
        EnterHoverMode();
        goto sumamospunto;
    }
    break;
}
///CASO 7
else if (x[i] > x[i - 1] && y[i] == y[i - 1])
{
    timerACC7++;
    ///El giro en este caso va a ser cero, hacia el NORTE
    giro[i] = 0.0f;
    labelgiro.Text = giro[i].ToString();

    if (timerACC7 >= 8 && timerACC7 < 12)
    {
        ///Variable declarada para comprobar si tiene que girar o no dependiendo de
       ///las condiciones del punto para el caso 7
        dYaw_comprobacion = giro[i] - diferenciagiro;

        if (dYaw_comprobacion == 0)
        {
            EnterHoverMode();
            UpdateUIAsync("HOVER caso 7...dYaw=0");
        }
        else
        {
            LeaveHoverMode();
            dH = 1000;
            dVx = 0.0f; dVy = 0.0f;
            dYaw = giro[i] - diferenciagiro;
            UpdateUIAsync("Girando caso 7...");
            boolpitch = false;
            boolroll = false;
            boolYaw = true;
            tmrTestControl2.Start();
        }
    }
    else if (timerACC7 >= 12 && timerACC7 < 16)
    {
        EnterHoverMode();
    }
    else if (timerACC7 >= 16 && timerACC7 < (16 + tiempoflying[i]))
    {
        btnForwardAuto.PerformClick();
    }
    else if (timerACC7 >= (16 + tiempoflying[i]))
    {
        EnterHoverMode();
        goto sumamospunto;
    }

    break;
}
 ///CASO 8
else if (x[i] == x[i - 1] && y[i] < y[i - 1])
{
    timerACC8++;
    giro[i] = -90.0f;
    labelgiro.Text = giro[i].ToString();

    if (timerACC8 >= 8 && timerACC8 < 11.14)
    {
```

```
                LeaveHoverMode();
                dH = 1000;
                dVx = 0.0f; dVy = 0.0f;
                dYaw = giro[i] - diferenciagiro;
                UpdateUIAsync("Girando caso 8...");
                boolpitch = false;
                boolroll = false;
                boolYaw = true;
                tmrTestControl2.Start();
            }
            else if (timerACC8 >= 11.14 && timerACC8 < 16)
            {
                EnterHoverMode();
            }
            else if (timerACC8 >= 16 && timerACC8 < (16 + tiempoflying[i]))
            {
                btnForwardAuto.PerformClick();
            }
            else if (timerACC8 >= (16 + tiempoflying[i]))
            {
                EnterHoverMode();
                goto sumamospunto;
            }
            break;
        }
        ///CASO 9
        else if (x[i] == x[i - 1] && y[i] > y[i - 1])
        {
            timerACC9++;
            giro[i] = 90.0f;
            labelgiro.Text = giro[i].ToString();

            if (timerACC9 > 8 && timerACC9 <= 11.14)
            {
                LeaveHoverMode();
                dH = 1000;
                dVx = 0.0f; dVy = 0.0f;
                dYaw = giro[i] - diferenciagiro;
                UpdateUIAsync("Girando caso 9...");
                boolpitch = false;
                boolroll = false;
                boolYaw = true;
                tmrTestControl2.Start();
            }
            else if (timerACC9 >= 11.14 && timerACC9 < 16)
            {
                EnterHoverMode();
            }
            else if (timerACC9 >= 16 && timerACC9 < (16 + tiempoflying[i]))
            {
                btnForwardAuto.PerformClick();
            }
            else if (timerACC9 >= (16 + tiempoflying[i]))
            {
                EnterHoverMode();
                goto sumamospunto;
            }
            break;
        }
    sumamospunto:
        ///Avanzamos en nuestro array de puntos con i++
        diferenciagiro = giro[i];
        i++;
        timerACC1 = 0;
        timerACC2 = 0;
        timerACC3 = 0;
        timerACC4 = 0;
        timerACC5 = 0;
        timerACC6 = 0;
```

```
            timerACC7 = 0;
            timerACC8 = 0;
            timerACC9 = 0;
            labelgiro0.Text = giro[i - 1].ToString();
            ///Si el control ha llegado al último punto, nuestro vuelo finaliza
            if (i >= (numPuntos))
            {
                EnterHoverMode();
                Land();
                UpdateUIAsync("¡¡FIN trayectoria control 2!!");
                tmrTestControl2.Stop();
                tmrcountr = 0;
            }
            ///Fin del bucle while
        }
```

3. Fin de la función de control

```
    }
```

❖ Anexo VI: Código algoritmo 2 desarrollado

1. Puntos trayectoria, variables y contadores necesarios

```
///Número de puntos que se cargarán

   int numPuntos = 5;

///Arrays de puntos de nuestra trayectoria

        float[] x = { 0.0f, 5.0f, 5.0f, 0.0f, 0.0f };
        float[] y = { 0.0f, 0.0f, -5.0f, -5.0f, 0.0f };
///Variables para medir el tiempo transcurrido en cada acción y cambiar a otras

        int tmrcountr3 = 0;
        int timerACC1 = 0;
        int timerACC2 = 0;
        int timerACC3 = 0;
        int timerACC4 = 0;
        int timerACC5 = 0;
        int timerACC6 = 0;
        int timerACC7 = 0;
        int timerACC8 = 0;
        int timerACC9 = 0;
        int j = 1;
```

2. Función principal del control. Algoritmo 2

```
        private void tmrTestControl3_Tick(object sender, EventArgs e)
        {
```

2.1 Análisis controles de la función Navigate

```
            float[] distancia;
            distancia = new float[numPuntos];

            float[] tiempoflying;
            tiempoflying = new float[numPuntos];

            Controlroll = (float)Math.Round((KPRoll * (dVy -
                (float)(clsARDroneGeneralParams.clsVy))), 5);
            Controlpitch = (float)Math.Round((KPPitch * (dVx -
                (float)(clsARDroneGeneralParams.clsVx))), 5);

            if (dYaw < 0)
            {
                dYaw = dYaw + 360;
            }
            Yawdiff = (dYaw - (float)(clsARDroneGeneralParams.clsYaw));
            if (Yawdiff < -180)
            {
                Yawdiff = Yawdiff + 360;
            }
            if (Yawdiff <= 180)
            {
                Controlyaw = (float)Math.Round((KPYaw * Yawdiff), 5);
```

```
}

if (Yawdiff > 180)
{
    Yawdiff = -(360 - Yawdiff);
    Controlyaw = (float)Math.Round((KPYaw * Yawdiff), 5);
}

Controlgaz = (float)Math.Round((KPH * (dH -
    clsARDroneGeneralParams.clsintAltitude)), 3);

label8.Text = dYaw.ToString();
label6.Text = Yawdiff.ToString();

if (Controlgaz > 0.9f) { Controlgaz = 0.9f; }
if (Controlgaz < -0.9f) { Controlgaz = -0.9f; }

if (Controlyaw > 1.0f) { Controlyaw = 1.0f; }
if (Controlyaw < -1.0f) { Controlyaw = -1.0f; }

if (boolpitch == true)
{
    if (Controlpitch > 0.05f) { Controlpitch = 0.05f; }
    if (Controlpitch < -0.05f) { Controlpitch = -0.05f; }
}
if (boolpitch == false)
{
    if (Controlpitch > 0.02f) { Controlpitch = 0.02f; }
    if (Controlpitch < -0.02f) { Controlpitch = -0.02f; }
}

if (boolroll == true)
{
    if (Controlroll > 0.05f) { Controlroll = 0.05f; }
    if (Controlroll < -0.05f) { Controlroll = -0.05f; }
}
if (boolroll == false)
{
    if (Controlroll > 0.02f) { Controlroll = 0.02f; }
    if (Controlroll < -0.02f) { Controlroll = -0.02f; }
}

///En este control no habrá giros, ni variaremos la altitud del drone
Controlyaw = 0;
Controlgaz = 0;

///Función encargada del movimiento de nuetsro cuadricóptero
Navigate(Controlroll, Controlpitch, Controlyaw, Controlgaz);

///Muestra los valores de la función Navigate() en la interfaz
SetTextRollInput("Roll: " + Controlroll);
SetTextPitchInput("Pitch: " + Controlpitch);
SetTextYawInput("Yaw: " + Controlyaw);
SetTextGazInput("Gaz: " + Controlgaz);
```

2.2 Loop de control y análisis de todos los casos

```
while (j < numPuntos)
{
    distancia[j] = (float)(Math.Sqrt(Math.Pow((x[j] - x[j - 1]), 2) +
            Math.Pow((y[j] - y[j - 1]), 2)));
    tiempoflying[j] = (2 * (1000 * distancia[j] / 500.0f));
    labeldistancia.Text = distancia[j].ToString();
    labeltiempoflying.Text = tiempoflying[j].ToString();

    label_i_j.Text = j.ToString();
```

```
///9 COMBINACIONES POSIBLES: todo detallado en la memoria de este TFG
///CASO 1
if (x[j] < x[j - 1] && y[j] > y[j - 1])
{
    timerACC1++;
    if (timerACC1 >= 8 && timerACC1 < (8 + tiempoflying[j]))
    {
        LeaveHoverMode();
        dH = 1000;
        ///multiplicamos las velocidades por 1000 para obtenerlo en
                milimetros/segundo, en todos los casos
        dVx = -((x[j] - x[j - 1]) * 1000 / tiempoflying[j]);
        dVy = ((y[j] - y[j - 1]) * 1000 / tiempoflying[j]);
        dYaw = 0.0f;
        UpdateUIAsync("Movimiento caso 1...");
        boolpitch = true;
        boolroll = true;
        boolYaw = false;
        tmrTestControl3.Start();
    }
    else if (timerACC1 >= (8 + tiempoflying[j]) && timerACC1 < (12 +
        tiempoflying[j]))
    {
        EnterHoverMode();
        goto sumamospunto3;
    }
    break;
}
///CASO 2
else if (x[j] > x[j - 1] && y[j] > y[j - 1])
{
    timerACC2++;
    if (timerACC2 >= 8 && timerACC2 < (8 + tiempoflying[j]))
    {
        LeaveHoverMode();
        dH = 1000;
        dVx = -((x[j] - x[j - 1]) * 1000 / tiempoflying[j]);
        dVy = ((y[j] - y[j - 1]) * 1000 / tiempoflying[j]);
        dYaw = 0.0f;
        UpdateUIAsync("Movimiento caso 2...");
        boolpitch = true;
        boolroll = true;
        boolYaw = false;
        tmrTestControl3.Start();
    }
    else if (timerACC2 >= (8 + tiempoflying[j]) && timerACC2 < (12 +
        tiempoflying[j]))
    {
        EnterHoverMode();
        goto sumamospunto3;
    }
    break;
}
///CASO 3
else if (x[j] < x[j - 1] && y[j] < y[j - 1])
{
    timerACC3++;
    if (timerACC3 >= 8 && timerACC3 < (8 + tiempoflying[j]))
    {
        LeaveHoverMode();
        dH = 1000;
        dVx = -((x[j] - x[j - 1]) * 1000 / tiempoflying[j]);
        dVy = ((y[j] - y[j - 1]) * 1000 / tiempoflying[j]);
        dYaw = 0.0f;
        UpdateUIAsync("Movimiento caso 3...");
        boolpitch = true;
        boolroll = true;
        boolYaw = false;
```

```
            tmrTestControl3.Start();
        }
        else if (timerACC3 >= (8 + tiempoflying[j]) && timerACC3 < (12 +
            tiempoflying[j]))
        {
            EnterHoverMode();
            goto sumamospunto3;
        }
        break;
    }
    ///CASO 4
    else if (x[j] > x[j - 1] && y[j] < y[j - 1])
    {
        timerACC4++;
        if (timerACC4 >= 8 && timerACC4 < (8 + tiempoflying[j]))
        {
            LeaveHoverMode();
            dH = 1000;
            dVx = -((x[j] - x[j - 1]) * 1000 / tiempoflying[j]);
            dVy = ((y[j] - y[j - 1]) * 1000 / tiempoflying[j]);
            dYaw = 0.0f;
            UpdateUIAsync("Movimiento caso 4...");
            boolpitch = true;
            boolroll = true;
            boolYaw = false;
            tmrTestControl3.Start();
        }
        else if (timerACC4 >= (8 + tiempoflying[j]) && timerACC4 < (12 +
            tiempoflying[j]))
        {
            EnterHoverMode();
            goto sumamospunto3;
        }
        break;
    }
    ///CASO 5
    else if (x[j] == x[j - 1] && y[j] == y[j - 1])
    {
        timerACC5++;
        if (timerACC5 >= 8 && timerACC5 < 12)
        {
            LeaveHoverMode();
            dH = 1000;
            dVx = 0.0f;
            dVy = 0.0f;
            dYaw = 0.0f;
            UpdateUIAsync("Caso 5 STOP...");
            boolpitch = false;
            boolroll = false;
            boolYaw = false;
            tmrTestControl3.Start();
        }
        else if (timerACC5 >= (12 + tiempoflying[j]))
        {
            EnterHoverMode();
            goto sumamospunto3;
        }
        break;
    }
    ///CASO 6
    else if (x[j] < x[j - 1] && y[j] == y[j - 1])
    {
        timerACC6++;

        if (timerACC6 >= 8 && timerACC6 < (8 + tiempoflying[j]))
        {
            LeaveHoverMode();
            dH = 1000;
            dVx = -((x[j] - x[j - 1]) * 1000 / tiempoflying[j]);
```

```
            dVy = 0.0f;
            dYaw = 0.0f;
            UpdateUIAsync("Movimiento caso 6...");
            boolpitch = true;
            boolroll = false;
            boolYaw = false;
            tmrTestControl3.Start();
        }
        else if (timerACC6 >= (8 + tiempoflying[j]) && timerACC6 < (12 +
            tiempoflying[j]))
        {
            EnterHoverMode();
            goto sumamospunto3;
        }
        break;
    }
    ///CASO 7
    else if (x[j] > x[j - 1] && y[j] == y[j - 1])
    {
        timerACC7++;

        if (timerACC7 >= 8 && timerACC7 < (8 + tiempoflying[j]))
        {
            LeaveHoverMode();
            dH = 1000;
            dVx = -((x[j] - x[j - 1]) * 1000 / tiempoflying[j]);
            dVy = 0.0f;
            dYaw = 0.0f;
            UpdateUIAsync("Movimiento caso 7...");
            boolpitch = true;
            boolroll = false;
            boolYaw = false;
            tmrTestControl3.Start();
        }
        else if (timerACC7 >= (8 + tiempoflying[j]) && timerACC7 < (12 +
            tiempoflying[j]))
        {
            EnterHoverMode();
            goto sumamospunto3;
        }
        break;
    }
    ///CASO 8
    else if (x[j] == x[j - 1] && y[j] < y[j - 1])
    {
        timerACC8++;

        if (timerACC8 >= 8 && timerACC8 < (8 + tiempoflying[j]))
        {
            LeaveHoverMode();
            dH = 1000;
            dVx = 0.0f;
            dVy = ((y[j] - y[j - 1]) * 1000 / tiempoflying[j]);
            dYaw = 0.0f;
            UpdateUIAsync("Movimiento caso 8...");
            boolpitch = false;
            boolroll = true;
            boolYaw = false;
            tmrTestControl3.Start();
        }
        else if (timerACC8 >= (8 + tiempoflying[j]) && timerACC8 < (12 +
            tiempoflying[j]))
        {
            EnterHoverMode();
            goto sumamospunto3;
        }
        break;
    }
    ///CASO 9
```

```
        else if (x[j] == x[j - 1] && y[j] > y[j - 1])
        {
            timerACC9++;
            if (timerACC9 >= 8 && timerACC9 < (8 + tiempoflying[j]))
            {
                LeaveHoverMode();
                dH = 1000;
                dVx = 0.0f;
                dVy = ((y[j] - y[j - 1]) * 1000 / tiempoflying[j]);
                dYaw = 0.0f;
                UpdateUIAsync("Movimiento caso 9...");
                boolpitch = false;
                boolroll = true;
                boolYaw = false;
                tmrTestControl3.Start();
            }
            else if (timerACC9 >= (8 + tiempoflying[j]) && timerACC9 < (12 +
                tiempoflying[j]))
            {
                EnterHoverMode();
                goto sumamospunto3;
            }
            break;
        }
     ///Avanzamos en nuestro array de puntos
     sumamospunto3: j++;
        timerACC1 = 0;
        timerACC2 = 0;
        timerACC3 = 0;
        timerACC4 = 0;
        timerACC5 = 0;
        timerACC6 = 0;
        timerACC7 = 0;
        timerACC8 = 0;
        timerACC9 = 0;
    ///Si el control ha llegado al último punto, nuestro vuelo finaliza
        if (j >= (numPuntos))
        {
            EnterHoverMode();
            Land();
            UpdateUIAsync("¡¡FIN trayectoria control 3!!");
            tmrTestControl3.Stop();
            tmrcountr3 = 0;
        }
    ///Fin del bucle while
     }
```

3. Fin de la función de control

```
}
```

C.-Acrónimos y definiciones

API Application Programming Interface (Interfaz de programación de aplicaciones)

CW Clock Wise (agujas del reloj, relacionado con los giros)

CCW Counter Clock Wise (contrario a las agujas del reloj). En Inglés, ACW (Anti Clock Wise)

DLL Dynamic-Link Library (Biblioteca de vínculos dinámicos)

GPS Global Positioning System (Sistema de Posicionamiento Global)

IMU Inertial Measurement Unit (Unidad de Medida Inercial)

IP Internet Protocol (Protocolo de Internet)

PID Proportional Integral Derivative Controller (Control Proporcional-Integral-Derivativo)

RC Remote Control (Control Remoto)

UAV Unmanned Aerial Vehicle (Vehículo Aéreo no Tripulado)

UCAV Unmanned Combat Aerial Vehicle (Vehículos no tripulados de combate aéreo)

USB Universal Serial Bus

Δ Incremento

β Ángulo de giro para orientar el drone hacia el siguiente punto

K_p Constante de proporcionalidad usada en el control para la función Navigate