ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)
INGENIERO ELECTROMECÁNICO

SISTEMA DE NAVEGACIÓN AUTÓNOMA PARA UN CUADRICÓPTERO EN EXTERIORES

Autor: Jaime Loring Castillo
Director: Juan Luis Zamora Macho

Madrid

Agosto de 2018

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. JAIME LORING CASTILLO

DECLARA ser el titular de los derechos de propiedad intelectual de la obra: *Navegación autónoma de un cuadricóptero en exteriores*, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente).*

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.

d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 16 de agosto de 2018

ACEPTA

Fdo……

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

NAVEGACIÓN AUTÓNOMA DE UN CUADRICÓPERO EN

EXTERIORES en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas

en el curso académico 2017-2018 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada

de otros documentos está debidamente referenciada.

Fdo.: Jaime Loring Castillo Fecha: 16/Agosto/2018

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Juan Luis Zamora Macho Fecha: 30/Agosto/ 2018

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)
INGENIERO ELECTROMECÁNICO

SISTEMA DE NAVEGACIÓN AUTÓNOMA PARA UN CUADRICÓPTERO EN EXTERIORES

Autor: Jaime Loring Castillo
Director: Juan Luis Zamora Macho

Madrid

Agosto de 2018

Agradecimientos

A Juan Luis, por su dedicación, orientación y paciencia.

A Elsa, parte activa e imprescindible en este proyecto a todos los niveles.

NAVEGACIÓN AUTÓNOMA DE UN CUADRICÓPTERO EN EXTERIORES

Autor: Loring Castillo, Jaime.

Director: Zamora Macho, Juan Luis.

Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

I. INTRODUCCIÓN

Con el desarrollo continuo de la electrónica, el ser humano está encontrando multitud de soluciones tecnológicas innovadoras cuyo impacto en la industria y en la vida cotidiana es absolutamente diferencial y está llamado a ser cada vez mayor. En los últimos años destaca el nacimiento de productos autónomos capaces de realizar tareas concretas de forma eficiente y segura. Entre estos productos aparecen los vehículos conocidos como drones o UAV.

Un UAV es un vehículo aéreo no tripulado, que puede funcionar tripulado por control remoto o de forma autónoma. Actualmente este tipo de vehículos se emplean en diversos ámbitos, entre los que destacan vigilancia, seguridad y defensa. En el ámbito civil destacan por su capacidad de tomar imágenes aéreas de alta calidad y precisión, ya que su versatilidad les permite grabar en todo tipo de situaciones. En el ámbito industrial se utilizan para realizar tareas de vigilancia de procesos en entornos no seguros para seres humanos. Su presencia en el sector va aumentando paulatinamente. Existen proyectos para la utilización de UAVs en inventariado y transporte de mercancías.
En definitiva, los UAVs no solo están ya ampliamente extendidos, sino que cada día ganan importancia en el ámbito industrial.

El objetivo de este proyecto es dotar a un vehículo aéreo, un cuadricóptero, de la capacidad de realizar una misión en exteriores de forma autónoma, sin el control de un ser humano. Ésta es una de las capacidades diferenciales de un UAV, ya que le permite realizar tareas de forma independiente, aumentando la eficacia de todo tipo de procesos e incidiendo en la automatización imperante en la industria hoy día.

II. METODOLOGÍA

El concepto final de este proyecto es el de desarrollar la capacidad autónoma de un UAV. En este sentido, se utilizarán simultáneamente una base de tierra y el propio vehículo. La base de tierra diseñará y enviará al UAV una misión determinada por waypoints, es decir, distintos puntos geográficos que deberá recorrer el UAV para completar la ruta, y será el propio vehículo el que gestione las trayectorias.

Una vez desarrollada la autonomía del UAV, se realizarán simulaciones lo más fieles a la realidad posible para comprobar su funcionamiento antes de implantar el sistema en el UAV.

Con el objetivo de probar el sistema en un entorno real, se programarán y

comprobarán las comunicaciones con los distintos sensores necesarios para la navegación.

Los objetivos que abarca el proyecto son los siguientes:

1. Correcta comunicación entre la base terrestre y el vehículo.
2. Capacidad de vuelo autónomo.
3. Simulaciones que determinen el funcionamiento del sistema
4. Comunicaciones con los sensores necesarios.

Para satisfacer dichos objetivos se detallan las siguientes tareas:

- Creación de la estructura de comunicación entre la base de tierra y el UAV dentro del protocolo MAVLink.
- Diseño de rutas en Mission Planner.
- Implantación del protocolo de recepción de misiones en el UAV.
- Implantación del protocolo de envío de misiones en la base terrestre.
- Gestión de referencias para el vuelo autónomo del UAV.
- Limitación de referencias para cumplir restricciones del control.
- Simulación del sistema de vuelo autónomo.
- Estudio de sensores y creación y adaptación de comunicaciones con los mismos.

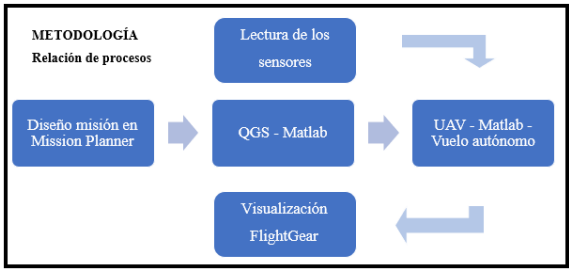


Figura 1: Relación de procesos

III. RESULTADOS

Los resultados del proyecto se pueden expresar según los distintos desarrollos.

A. TRANSMISIÓN DE MISIÓN

En este desarrollo el objetivo era diseñar la misión y transmitirla al drone.

Para diseñar la misión se utiliza Mission Planner. Sobre un mapa físico proporcionado por Google Maps se traza la ruta que debe describir el UAV. Para trazar la ruta se van marcando los waypoints que debe atravesar el UAV. Esta misión se guarda en el ordenador para su posterior uso.

Figura 2: Entorno gráfico Mission Planner

A continuación, la base terrestre programada en Matlab lee este archivo de misión y lo envía por puerto serie al UAV. El envío se hace utilizando el protocolo MAVLink, y su procedimiento específico de envío de misiones. Este procedimiento se programa exitosamente en Matlab con una máquina de estados.

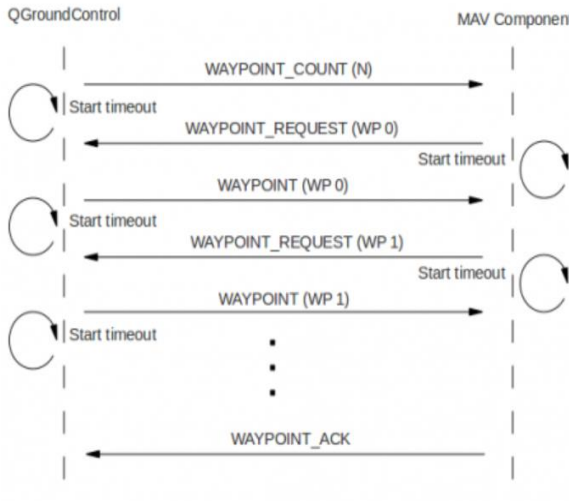


Figura 3: Procedimiento de misiones

En el UAV se programa la recepción de la misión. Se añaden los mensajes necesarios al decodificador del protocolo MAVLink y se añade el procedimiento de recepción. El UAV guardará en su memoria una matriz formada por los waypoints, en formato LLA (Latitud Longitud Altura). Esta transmisión puede hacerse por Bluetooth o por radio.

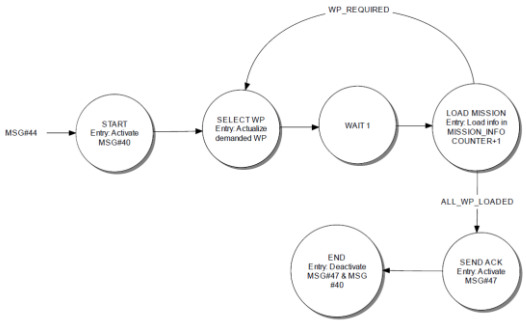


Figura 4: Máquina de estados de recepción

De esta forma, programamos en el UAV la capacidad de recibir una misión programada en su base de tierra a través de módulos de telemetría de acuerdo al funcionamiento usual de los vehículos autónomos.

B. GESTIÓN DE TRAYECTORIAS

A continuación, dotamos al UAV de la lógica para generar referencias utilizando la matriz de waypoints. Para ello debe ser capaz de traducir las coordenadas geográficas a XYZ según su propio sistema de referencia. Así, se acuerda un primer punto como origen de referencia, que suele ser el punto de puesta en marcha del UAV, y las coordenadas en XYZ se dan sobre este punto, con el eje Z perpendicular al suelo y positivo hacia abajo, el eje X positivo hacia el norte geográfico y el eje Y positivo hacia el este geográfico, de forma análoga a las coordenadas NED.

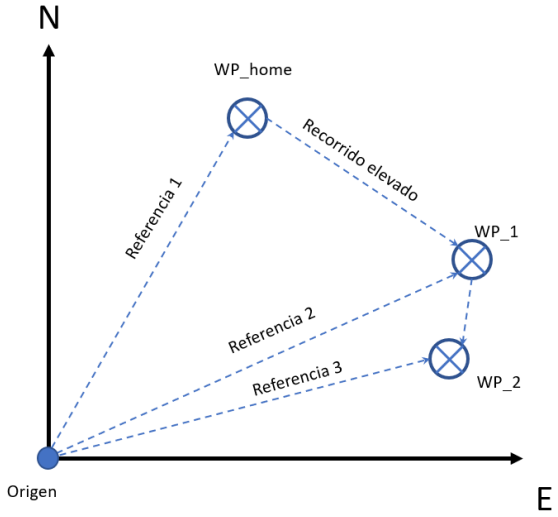


Figura 5: Esquema de referencias sin límite

Sobre este origen se calculan las coordenadas en XYZ del primer punto de la misión o punto HOME, y a continuación se dan las referencias de cada waypoint sucesivo a medida que el UAV los va alcanzando. Esto puede suponer recorridos muy elevados para el UAV, capaces de saturar el mando e incluso generar inestabilidades en el control.

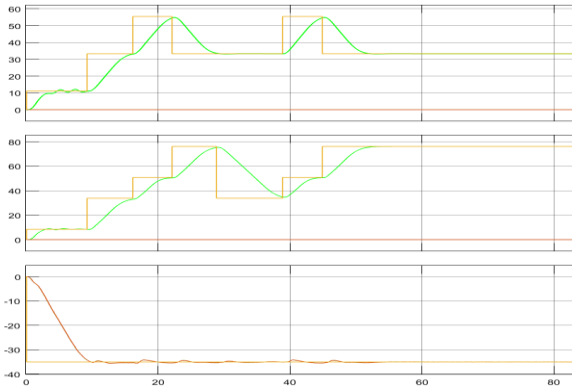


Figura 6: Vuelo autónomo sin limitación

Para solucionar este problema hemos de tener en cuenta la situación efectiva del UAV, de forma que crearemos waypoints virtuales entre su posición y la del waypoint destino para dar referencias que no saturen el mando.

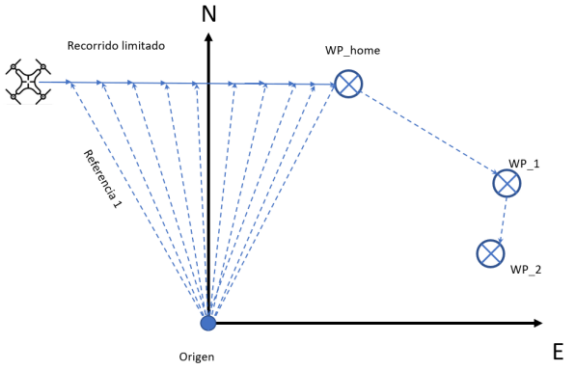


Figura 7: Esquema de referencias limitadas

De esta forma, se escalonan las referencias logrando mayor precisión y estabilidad.

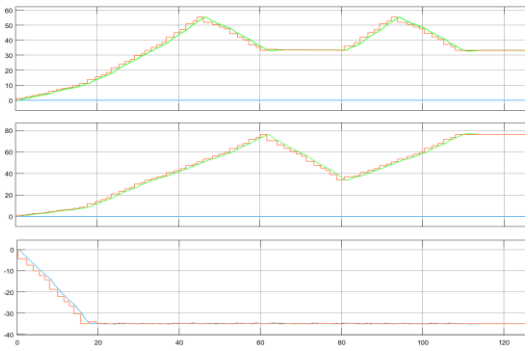


Figura 8: Vuelo autónomo con limitación

C. SIMULACIÓN

Las simulaciones se efectúan utilizando el entorno de simulación gráfica FlightGear. Se simulan las físicas y funcionamiento del UAV en Matlab y se envía la información a FlightGear para su visualización a través de una conexión UDP.

Figura 9: Entorno de simulación

Se simula toda la operación, desde el envío de la misión a través de puerto serie desde una estación base que se simula con un ordenador, la activación de los rotores, el despegue con controladora y el paso a modo autónomo.

D. SENSORES

El objetivo final de este proyecto sería implantar y probar la solución en un UAV real. Este trabajo se realizará en futuros desarrollos. Como último paso se programan las comunicaciones con los sensores necesarios para la navegación en exteriores. Estos son el barómetro (MS5611), el magnetómetro (AK8963, integrado en la IMU MPU9250) y el GPS (M8N de Ublox).

IV. CONCLUSIONES

Como conclusión se puede afirmar que los objetivos del proyecto se han superado satisfactoriamente.

Hemos logrado crear, leer y transmitir la misión desde la base terrestre y

almacenarla correctamente en el UAV a través de un puerto serie. Para ello hemos tenido que probar diversas configuraciones en los puertos, optimizar el decodificador, añadir mensajes al protocolo y crear una máquina de estados específica para el envío y la recepción de la misión.

A su vez, se ha diseñado y probado el sistema de navegación autónoma. Por un lado, añadimos un algoritmo de traducción de coordenadas geográficas a coordenadas locales, y por otro un algoritmo de gestión de waypoints. Sobre un primer desarrollo añadimos una función que tiene en cuenta la posición del UAV para dar mejores referencias, que no saturen el mando y que den mayor precisión y seguridad.

Con el objetivo de comprobar el funcionamiento del proyecto, se ha creado una estructura de simulación válida para todo tipo de proyectos relacionados con el UAV. Se ha conseguido establecer la conexión entre el simulador de Matlab y el entorno gráfico FlightGear, se han descargado los escenarios deseados y se ha realizado la simulación satisfactoriamente.

Por último, se han programado y comprobado las comunicaciones con los sensores de navegación. Se ha comprobado que la información enviada al UAV después de la configuración es correcta, de forma que se puede emplear para la navegación. Gracias a esto, las últimas pruebas sobre el UAV real se pueden realizar directamente.

Este proyecto se integra en otros cuyo objetivo general es desarrollar un UAV funcional con capacidad de vuelo autónomo enteramente en ICAI. En este aspecto, una parte fundamental del desarrollo del proyecto ha sido trabajar coordinadamente con otros proyectos relacionados con funcionalidades del UAV. Así, finalmente hemos obtenido un único desarrollo de UAV con múltiples capacidades. En un marco general, este proyecto ha contribuido a robustecer las comunicaciones del UAV, optimizando el funcionamiento del módulo de comunicaciones de MAVLink gracias a diversos errores que hemos localizado. Además, este proyecto ha incrementado la capacidad de simulación y de debuggeado, al añadir ciertas funciones y elementos de visualización en el programa.

V. REFERENCIAS

[1] [NIMA00] National lmagery and mapping Agency. “Department of defense World Geodetic System 1984” Año 2000.

[2] [GLOSA17] Glosarios Alicante [Online] http://glosarios.servidor-alicante.com/topografiageodesia-gps/coordenadas-tridimensionales---centradasen-la-tierra-ecef

[3] [GARC17] José Antonio E. García. Así funciona. [Online] http://www.asifunciona.com/electronica/af_gps/af_gps_10.htm

[4] [ARDU] Ardupilot.org. [Online] http://ardupilot.org/ardupilot/index.html

[5] [BONE08] Xavier Bonet. Creación de librerías para la asignatura Navegación Aérea, Año 2008

AUTONOMOUS NAVIGATION OF A QUADCOPTER AT OUTSIDE SCENARIOS

Author: Loring Castillo, Jaime.

Director: Zamora Macho, Juan Luis.

Collaborating Entity: ICAI – Universidad Pontificia Comillas

PROJECT SUMMARY

I. INTRODUCTION

Due to the constant advances in electronics, humans are finding a lot of innovative technological solutions whose impact in industry and common life is remarkable and constantly rising. During the last years, the creation of autonomous devices is making the difference. They can execute different tasks in a more efficient and safe way than human workers. Among these devices que can find the unmanned aerial vehicles i.e. UAV.

An UAV is a vehicle that can perform both controlled by a human team and in an autonomous way. Nowadays this kind of vehicles are used in many areas, mainly in vigilance, security and defense. For civil purposes it is mainly used for filming precise and versatile aerial videos. At the industrial area, they are used for vigilance tasks at dangerous environments. However, their importance is constantly increasing. There are projects to develop UAVs to be used to transport goods and to make inventory. To sum up, UAVs are not just very important, but they are gaining importance in every area.

The main objective of this project is to give an aerial vehicle, a quadcopter, the capacity to complete a mission outside autonomously. This is one of the most important capacities of an UAV, because it allows it to perform tasks independently, increasing the efficiency of them and improving the automatization of the process.

II. METHODOLOGY

The final concept of this project is to develop the autonomous capacity of an UAV. Looking forward this objective a ground control station and an UAV will work simultaneously. Ground control station will design and send the mission to the UAV. The mission will be determined by waypoints, i.e. points which will have to be reached by the UAV to complete the route.

Once developed the autonomous capacity of the vehicle, several simulations will be performed. They will fit the reality as best as possible, so after their execution the system can be implanted in a real UAV.

Looking forward to facilitating the implantation, the communications with the demanded sensors will be programmed and probed.

The objectives of this project are:

1. Robust communications between our ground control station and the vehicle.
2. Autonomous flight capacity.
3. Simulations to determine the performance of the solution.

4. Communications with the demanded sensors.

To satisfy the objectives, the following tasks must be achieved:

- Creation of the structure of communication between the ground control station and the UAV using the protocol MAVLink.
- Mission design using Mission Planner.
- Implementation of the protocol of reception of the missions at the UAV.
- Implementation of the protocol of mission sending at the ground control station.
- Management o references by the UAV.
- Limitation of the references to fit UAV´s control specification.
- Simulation of the autonomous navigation system.
- Communications with demanded sensors.

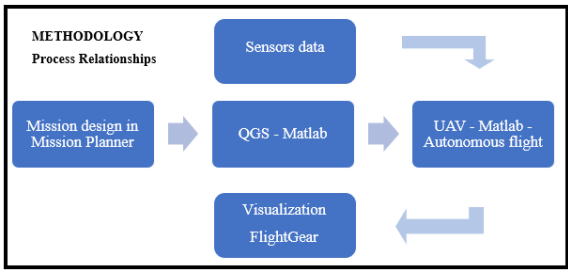


Figure 1: Methodology

III. RESULTS

Results of the project can be considered in different areas.

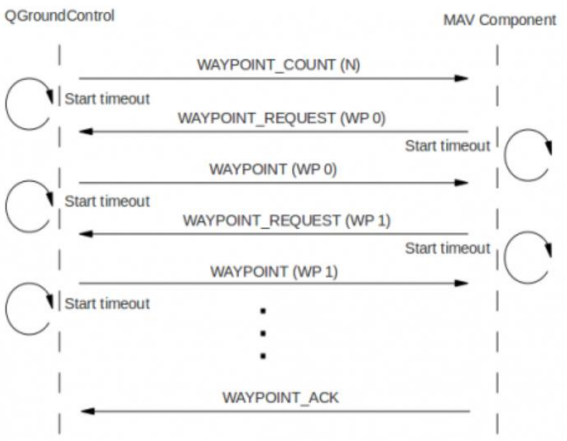
A. MISSION TRANSMISSION

The objective is to design and send the mission to the UAV.

We use Mission Planner to design the mission. We mark points on a map given by Google Maps to determine the route. The UAV will have to achieve these points. The mission is saved into the computer for the ground control station to use it.

Figure 2: Mission design

Right next, the ground control station programmed in Matlab reads this document and sends it via serial port to the UAV. For sending it we use MAVLink and a specific procedure to send missions. This procedure is programmed in Matlab with a state machine.



At the UAV we program the reception of the mission. We add the necessary messages to the MAVLink receiver and we program the specific procedure of

Figure 3: Mission transmission procedure

mission receiving. The UAV will save into his memory a matrix formed by the waypoints in LLA format (Latitude Longitude and Altitude). This transmission will be done by radio or Bluetooth

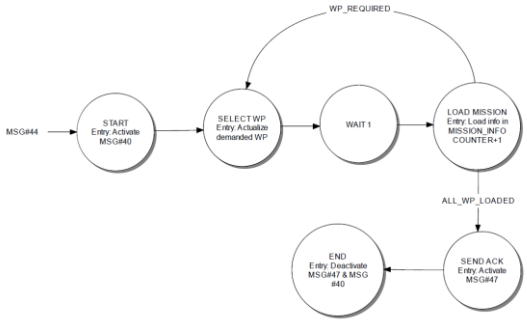


Figure 4: Receiver state machine

Hence, we give the UAV the capacity of receiving a mission from a ground control station by radio as it usually done in this kind of vehicles.

B. TRAJECTORY MANAGEMENT

After that, we give the UAV the capacity to generate references using the waypoints matrix. To do this, it must be able of convert the geographical coordinates to local references. Thus, we use a specific point, usually the point where the UAV is turned on, as origin of references. Axis Z looks down towards earth, axis X looks towards geographical north and axis Y looks towards geographical east.

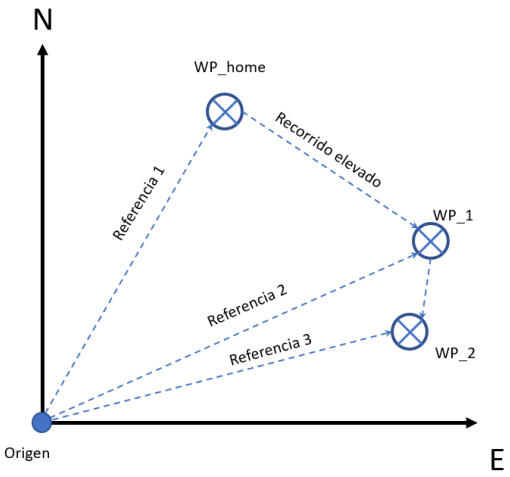


Figure 5: Unlimited references

Around this origin coordinates XYZ of the first waypoint (HOME) are calculated, and afterwards every waypoint will be calculated relative to the previous one.

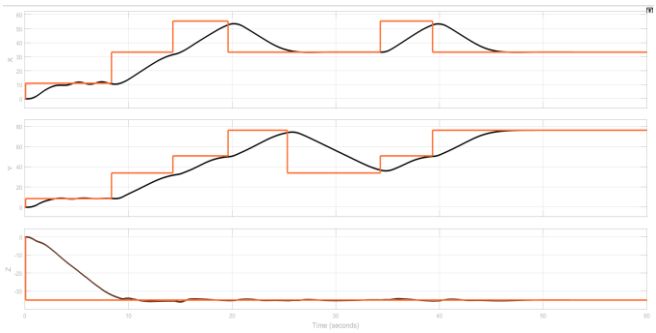


Figure 6: Autonomous unlimited flight

This kind of working generates some saturation problems. To solve these problems, we must consider the position of the UAV, creating virtual waypoints between its position and waypoints.

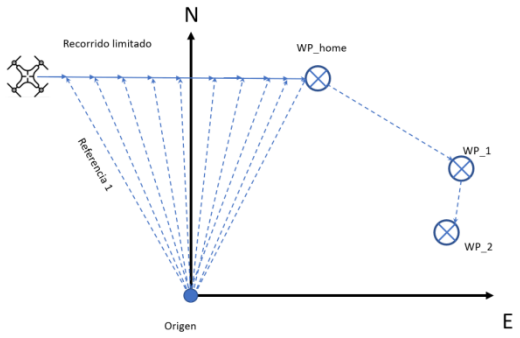


Figure 7: Limited references

Thanks to this we reduce the module of the references, achieving a softer and more precise solution.

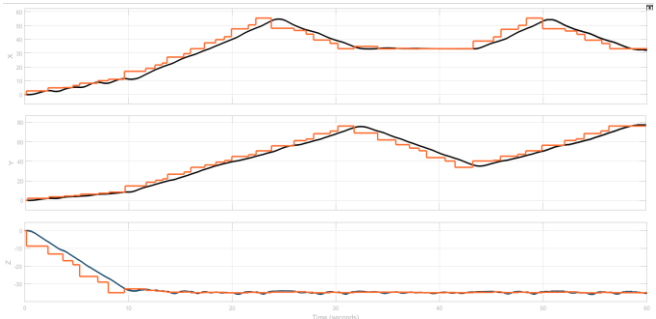


Figure 8: Autonomous limited flight

C. SIMULATION

Simulations are carried out using the graphic simulator FlightGear. The physics and the UAV are simulated in Matlab and this information is sent to FlightGear through a UDP port to its visualization.

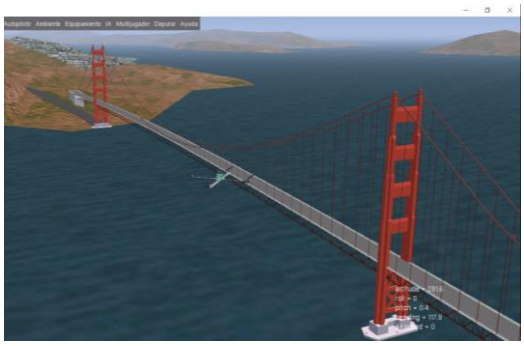


Figure 9: FlightGear

During the simulation, we try to simulate the operation as loyal to reality as possible. We simulate the mission transmission, the initialization of the motors, the takeoff and the autonomous mode.

D. SENSORS

The final objective of this project could be to implement and probe the solution in a real UAV. This work will be done in future projects. Looking forward to it, we program the communications with the needed sensors for the navigation. These are the barometer (MS5611), the magnetometer (AK8963, implemented in the IMU MPU9250) and the GPS (M8N of Ublox).

IV. CONCLUSIONS

To sum up, we can conclude that we have successfully accomplished the projects objectives.

We have managed to create, read and send a mission from the ground control station and save it in the UAV without errors through a serial port. To do this, we have probed several communication configurations of the ports, optimize the MAVLink receiver, add certain messages and create a specific state machine for sending and receiving the mission.

At the same time, the autonomous navigation system has been designed and probed. We add an algorithm to transform geographical to local coordinates and another one to manage the waypoints. Over the first solution, we develop a function to find better references to get better precision and security.

Looking forward to test the solution, we have developed a structure of simulation which is valid for other projects about UAVs We have managed to stablish the connection between the Matlab simulator and the viewer FlightGear. The simulations have been carried out successfully.

To end, we have programmed and probed the communications with the navigation sensors. We have checked that the information sent to the UAV after the configuration is correct, hence it can be used for navigation. Thanks to this, the implementation on the UAV can be done directly.

This project is developed simultaneously with others to design a UAV with autonomous capacity entirely at ICAI. Hence, a critical part of this project has been to work coordinately with other projects related to functionalities of the UAV. Thus, at the end we have developed a unique UAV with different capacities. This project has contributed to strengthen the communications of the UAV, optimizing the MAVLink coder thanks to different errors that we could debug. This project has also incremented

the capacity of simulation and debugging adding several functions and visualization elements.

V. REFERENCES

[1] [NIMA00] National lmagery and mapping Agency. "Department of defense World Geodetic System 1984" Año 2000.

[2] [GLOSA17] Glosarios Alicante [Online] http://glosarios.servidor-alicante.com/topografiageodesia-gps/coordenadas-tridimensionales---centradasen-la-tierra-ecef

[3] [GARC17] José Antonio E. García. Así funciona. [Online] http://www.asifunciona.com/electronica/af_gps/af_gps_10.htm

[4] [ARDU] Ardupilot.org. [Online] http://ardupilot.org/ardupilot/index.html

[5] [BONE08] Xavier Bonet. Creación de librerías para la asignatura Navegación Aérea, Año 2008

ÍNDICE DE PROYECTO

PARTE 1 Memoria .....

Parte 1
Memoria

ÍNDICE DE LA MEMORIA

ÍNDICE DE FIGURAS

Capítulo 1. INTRODUCCIÓN

En este capítulo se realizará una breve introducción sobre el proyecto. Se hablará sobre el estado actual de los UAVs en diferentes disciplinas y su situación en los mercados. Se hará una presentación sobre las características del modelo del UAV desarrollado en ICAI antes de realizar este proyecto que servirá para contextualizar el resto de la memoria, explicar los objetivos de este proyecto y para cuantificar el impacto del mismo sobre el modelo general.

1.1. UAVs

Un UAV es un vehículo aéreo no tripulado, que puede funcionar manejado por control remoto o de forma autónoma. Este tipo de vehículos trabaja siempre con una base de tierra que gestiona su funcionamiento. A todo el sistema de funcionamiento se le denomina UAS (Unmanned Aerial System). Dentro de estos sistemas, dependiendo de si hay o no un piloto remoto, encontramos los sistemas RPAS (Remotely Piloted Aircraft System) y los AAS (Autonomous Aerial Systems). En este proyecto buscamos convertir un RPAS en un AAS, de forma que esta última nomenclatura sería la idónea para referirnos a nuestro sistema tras la ejecución del proyecto. Los UAV son vehículos que destacan por su maniobrabilidad, precisión y limpieza. Son menos contaminantes y más baratos que otras soluciones. Reducen los riesgos del ser humano, ya que pueden actuar en entornos peligrosos y realizar tareas de riesgo que antes ejecutaban personas. Reduce los tiempos de ejecución en todo tipo de tareas. Exigen una formación breve y sencilla. Son operados a distancia, con lo que un operario puede abarcar un terreno muy amplio desde un único centro de control. Son capaces de realizar tareas de forma autónoma, reduciendo la mano de obra y optimizándolas. Sin embargo, su mayor virtud es su versatilidad. Son capaces de contribuir y ofrecer soluciones optimizadas o innovadoras en casi todos los ámbitos, tanto profesionales como civiles. Así, encontramos un mercado muy fragmentado, en el que se producen UAVs con características concretas,

profundamente especializados, y UAVs de carácter más general aptos para distintos tipos de uso. [1]

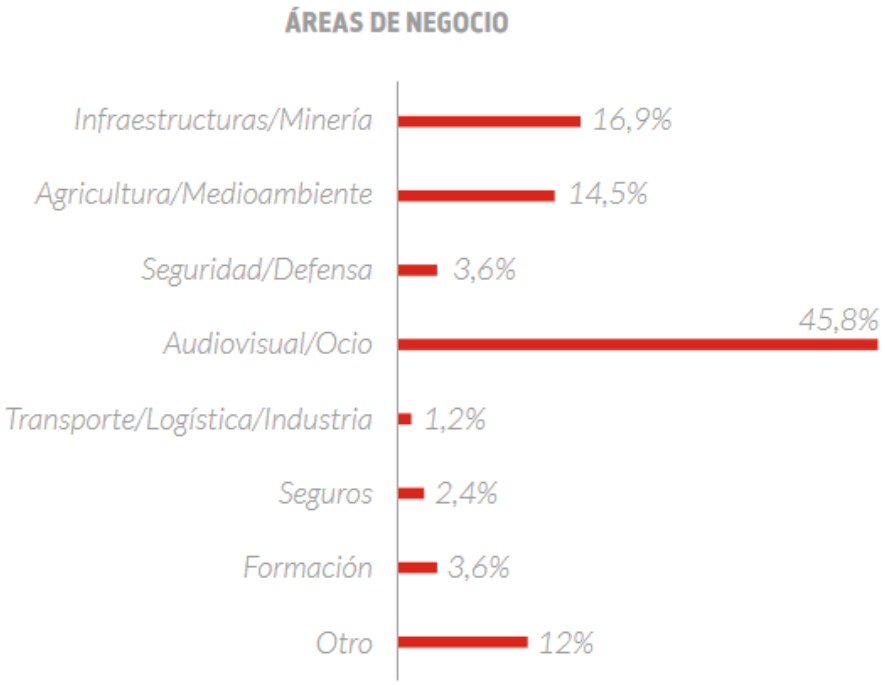


Figura 1. Áreas de negocio del mercado del UAV del barómetro ToDrone

En la *Figura 1* podemos observar las principales áreas de negocio según una encuesta realizada en 2016 entre las empresas dedicadas al desarrollo y producción de drones en España. Destacan los ámbitos audiovisual y ocio, infraestructura y minería y agricultura y medio ambiente. Pese a su bajo porcentaje de mercado, el sector de seguridad y defensa es el que adquiere un mayor peso económico, puesto que, aunque es un sector especializado en el que participan pocas empresas, es uno de los ámbitos con más aplicaciones y más desarrollados. La participación de los UAVs en el sector industrial es también de interés debido a la vocación de este proyecto a su incesante crecimiento. Se hablará a continuación sobre las condiciones de los UAVs que se utilizan en estos ámbitos.

1.1.1. Sector audiovisual y ocio

El sector audiovisual es sin duda uno de los más beneficiados por la irrupción del UAV. La polivalencia y bajo coste de estos productos, así como su fácil manejo, ha permitido que los drones entren este sector y realicen cada vez más tareas. De los operadores registrados en España en 2017, más de la mitad están dedicados a este sector. Así, cumplen funciones de grabación en deportes, películas, documentales y conciertos. [2]
Destacan especialmente por su capacidad para tomar planos cenitales. Son capaces de sustituir equipos de grabación como helicópteros, avionetas y dirigibles. Toman imágenes de mayor calidad que estos debido a su maniobrabilidad y precisión. Además, eliminan el riesgo humano, ya que los equipos dejan de estar en el aire y controlan la grabación desde una base de tierra. Reducen los tiempos, ya que pueden alcanzarse los puntos de grabación más rápidamente y eliminan los protocolos de seguridad en el arranque de aeronaves con pasajeros. Así, se han convertido en métodos de grabación esenciales para la producción de películas y documentales de todo presupuesto.

Figura 2. Imagen concurso DJI

Dentro de este sector son especialmente útiles para la retransmisión y grabación de deportes. Se utilizan en deportes en entornos adversos y diversos como el surf profesional,

el alpinismo y la bajada de aguas rápidas en kayak. Pueden alcanzar altas velocidades, de forma que pueden situarse en todo momento en posiciones idóneas para la grabación. Se utilizan a su vez en deportes mayoritarios como el fútbol o el baloncesto para obtener tomas de alta calidad, impensables hace pocos años. Son especialmente útiles en el seguimiento de carreras y rallys. En estos seguimientos destacan los UAVs con capacidad de reconocimiento visual, que sin la necesidad de un operario son capaces de seguir a un corredor o grupo concreto. Varios de estos drones autónomos son capaces de monitorizar una carrera completa, ofreciendo imágenes alta calidad e información sobre el estado de los participantes con un mínimo o nulo control humano. Un ejemplo es el UAV HEXO+, que se comercializa en Estados Unidos. Es un drone de seguimiento completamente autónomo, y cuyo precio oscila en los 1.000$.

Los drones dedicados a este sector son bastante similares entre ellos. Suelen ser UAVs pequeños y polivalentes. Incluyen elementos de grabación de gran calidad, lo que incrementa susceptiblemente su precio. Sin embargo, este se distribuye en una gama muy amplia, por lo que se pueden encontrar drones de grabación para proyectos pequeños de bajo presupuesto o drones de alta calidad y especificaciones concretas y exigentes que suponen un desembolso mayor. En general están controlados por equipos humanos, aunque algunas funciones son ideales para drones autónomos.

1.1.2. Sector infraestructuras y minería

En este sector, los drones cumplen funciones de inspección de infraestructuras, mapeo 3D de infraestructura minera subterránea, reconocimiento y mapeo de terrenos de construcción, servicios de topografía, prospección y control de operaciones. Ofrecen a su vez servicios para la prevención de riesgos laborales y respuesta a accidentes.

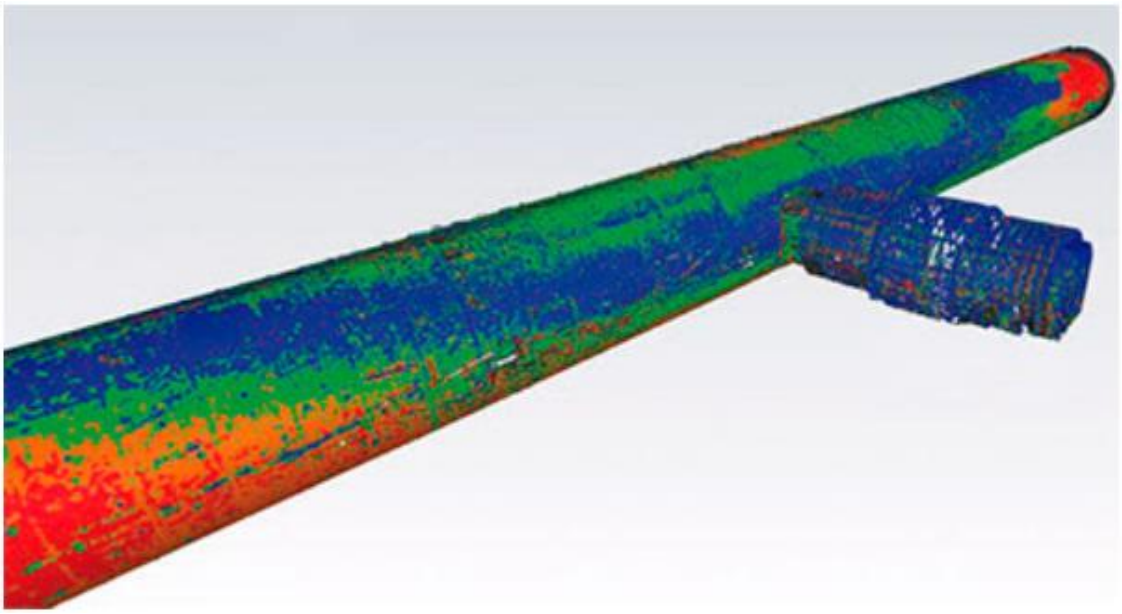


Figura 3. Mapeo 3D de instalaciones subterráneas

Este tipo de UAVs requieren sensores y características específicas. Así, el control de la altura y posicionamiento (ocasionalmente en la oscuridad) es esencial para su funcionamiento. Deben ser robustos porque pueden trabajar en condiciones desfavorables. La herramienta más importante de este tipo de drones es el Lidar, un sensor que permite tomar una gran cantidad de datos geográficos georreferenciados que permiten realizar mapeados 3D. [1]

En este tipo de funciones, la autonomía del UAV es vital, ya que suelen realizar tareas repetitivas como inspecciones o de larga ejecución como mapeados y revisiones. La mayoría de estas funciones se realizan de forma autónoma a través de rutas planificadas por operarios en su base de operaciones.

1.1.3. Sector agricultura y medioambiente

En sector de agricultura realizan tareas muy diversas. Se utilizan en el control de plagas, tanto para la monitorización de las mismas como para el rociado de insecticidas. Permiten controlar el crecimiento del cultivo, creando mapas de estado de los terrenos para optimizar el uso y la aplicación de los recursos agrarios. Son necesarios para la agricultura de precisión, que precisa de datos geográfico, espectrales y temporales para la optimización del cultivo en los terrenos disponibles.

Figura 4. Mapeo segmentado de crecimiento de cultivo

En el sector ambiental son especialmente útiles para trabajar con animales porque son menos invasivos que un cámara humano e infinitamente más versátiles que una cámara fija. Así, cumplen funciones de monitorización de animales en reservas, búsqueda de especímenes en áreas extensas para su control o grabación de costumbres de animales para su estudio. Pueden trabajar con especies peligrosas eliminando el riesgo humano, pueden trabajar con especies profundamente recelosas y territoriales, y pueden cubrir áreas muy extensas en poco tiempo. Además, son esenciales para alcanzar terrenos especialmente adversos como islas de difícil acceso o zonas escarpadas.

En este sector destacan también por sus funciones de vigilancia de entornos en peligro, como ríos con vertidos o bosques deforestados. Son útiles para monitorizar el impacto ambiental de la actividad humana en el medio y para obtener distintos indicadores, como la salinidad o la presencia de O2 en el aire, que determinan el estado de un medio concreto.

Casi todas estas tareas se realizan de forma autónoma, sin el manejo remoto de un ser humano. Pueden funcionar con rutas repetitivas o con procesado de visión. [1]

1.1.4. Sector seguridad y defensa

Este sector aporta el mayor movimiento económico del mercado de UAVs. Se prevé que en 2020 el valor de la producción de drones militares superará los seis billones americanos. Es el sector de nacimiento de estas tecnologías. Actualmente, todos los ejércitos cuentan con una flota de UAVs que realizan todo tipo de misiones.
En el sector de la seguridad se encuentran multitud de funciones que pueden ejecutar este tipo de vehículos. Así, su uso está muy extendido en la vigilancia autónoma de perímetros, en urbanizaciones, zonas forestales, almacenes y polígonos. Es habitual que cumplan funciones de salvamento, o de detección de accidentes o problemas estructurales. [2]

1.1.5. Sector industrial

La presencia de los UAVs en el sector industrial está creciendo en los últimos años y se espera que continúe haciéndolo hasta que cumplan un gran número de funciones. Actualmente aparecen sobre todo en inspecciones de plantas, que permiten análisis detallados para localizar errores antes imperceptibles. Permiten intervenciones rápidas y múltiples en planta, reducen el riesgo humano y pueden trabajar en entornos nocivos.
Se espera que su presencia esté muy ligada a la industria 4.0. Actualmente grandes multinacionales como Amazon o Apple desarrollan proyectos relacionados con los drones. En todo caso se busca utilizar sistemas autónomos, que permitan mejorar servicios, aumentando su eficacia y precisión, y realizando gran cantidad de tarea con poca presencia humana. En este sentido aparecen proyectos como la realización de inventariado de almacén con UAVs o el reparto de paquetes y mercancías a través de estos vehículos. En ningún caso serían pilotados por personas, aunque siempre estarían monitorizados y controlados por operarios. [2]

1.2. Antecedentes

En este apartado se explicará el modelo de UAV previo a este proyecto. Se explicarán sus partes y en qué punto de desarrollo se encuentra.

A lo largo del proyecto se varían sustancialmente algunas partes del modelo. Así, si inicialmente se planea utilizar un microcontrolador STM32, finalmente se decide utilizar una Raspberry Pi Zero. Esta ofrece una capacidad computacional mucho más alta, necesaria para el correcto funcionamiento del UAV. Esto supone un cambio en las comunicaciones. Para el primer controlador se plantean comunicaciones utilizando la librearía Waijung, pero finalmente utilizaremos el bloque específico de comunicaciones de la Raspberry. Se cambian también diversas estructuras de buses y la distribución general de funcionamiento. En este documento se hablará únicamente del modelo final, excluyendo de la memoria los primeros desarrollos y las diversas adaptaciones.

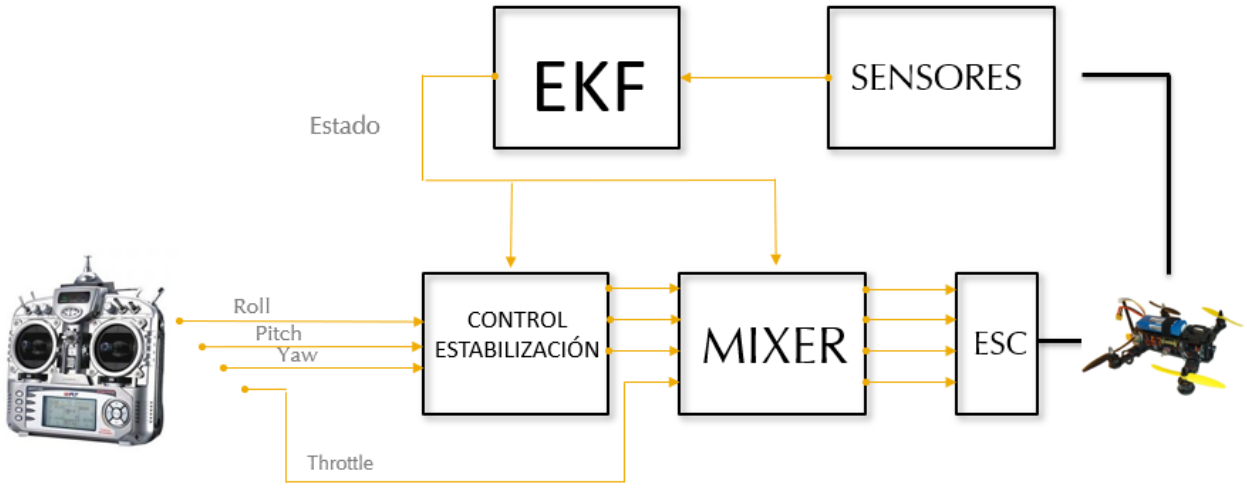


Figura 5. Estructura software

En la *Figura 5* se expone la estructura del software al comienzo del proyecto. Las referencias vienen dadas por una emisora. Estas pasan por el control de estabilización y por el mixer para dar referencias PWM a los ESC que controlan los motores que impulsan las hélices.

El sistema de control del UAV se programa en Matlab y Simulink. En el archivo completo encontramos además una base de tierra y todos los ficheros externos de configuración, así como diversa documentación.

El modelo final está estructurado en tres boques principales y dos buses de comunicaciones.

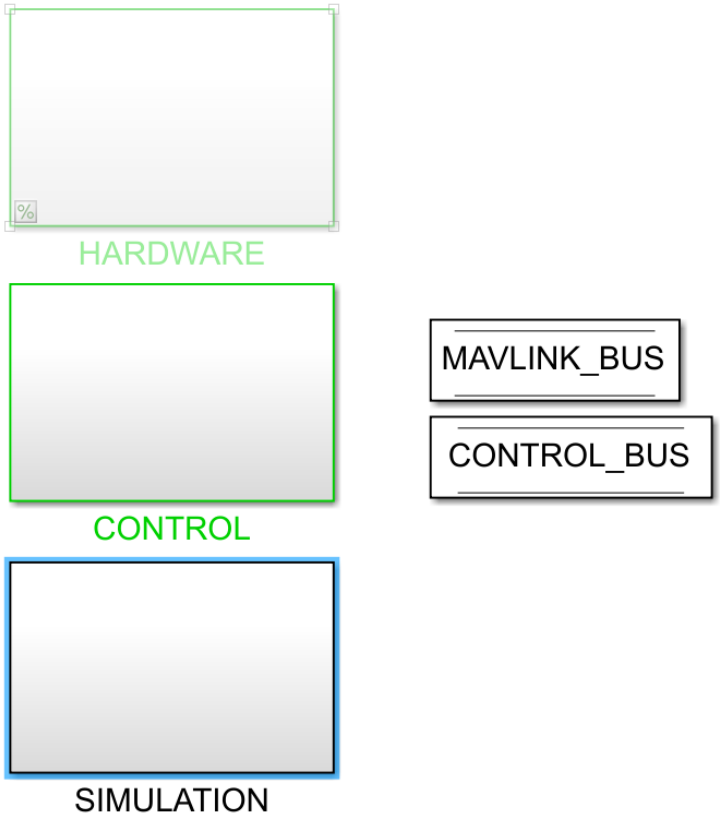


Figura 6. Estructura principal del UAV

El bloque "HARDWARE" es el menos relacionado con el proyecto, y únicamente tomamos contacto con él en los últimos desarrollos de comunicaciones y configuración de sensores. En él encontramos todo lo relacionado con los actuadores, la gestión de la carga computacional, las comunicaciones y los sensores. Todos los sensores están programados para el controlador STM32, de modo que solo algunos que están actualizados son válidos para el modelo final.

Prácticamente todo el desarrollo del proyecto se hace en el bloque "CONTROL". En él encontramos dos bloques, el "CONTROL_UPDATE" y el "COMMUNICATIONS". En el primero están programados el control del UAV y la máquina de estados que gestiona sus modos de funcionamiento. Por tanto, este primer bloque se encarga de todo lo relacionado con el estado y el control del UAV, y en él tendrán que añadirse el estado de vuelo autónomo y la gestión de algunos aspectos del mismo. En el segundo bloque encontramos los bloques de codificación y decodificación del protocolo MAVLink. Por tanto, son estos bloques los que habilitan al UAV para comunicarse con la base terrestre, y enviar y recibir información importante para el funcionamiento. Estos bloques

contienen la información de los mensajes que se han estado utilizando hasta ahora. Habrá que añadir los referentes al modo de funcionamiento autónomo.

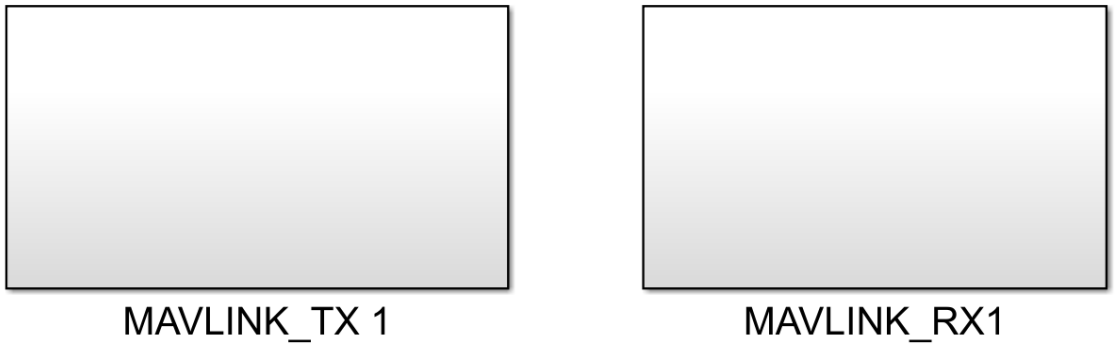


Figura 7. Codificador y decodificador MAVLink

Además, se añadirá un tercer bloque, "MISSION MANAGEMENT", en el que se programará buena parte de la gestión del modo autónomo, como se expondrá en capítulos posteriores.

En el bloque "SIMULATION" encontramos todo lo necesario para llevar a cabo las simulaciones. Encontramos dos bloques principales, el modelo del UAV y el modelo de los sensores, que incluye ruidos y fallos aleatorios. Encontramos además un nuevo bus, el "MODEL_BUS", que contiene todas las señales relativas a la simulación. Estos dos bloques se encargan de simular toda la información que nos daría el bloque "HARDWARE" si estuviese funcionando realmente. Además, encontramos bloques para las comunicaciones con la controladora, para la visualización de señales internas y para controlar el ritmo de la simulación. Habrá que añadir bloques para la inicialización de puertos, para el visionado de MAVLink y para las comunicaciones con FlightGear. Además, añadiremos un visualizador para indicar si la misión está cargada.

Como puede observarse, el modelo funciona con bloques independientes. Las señales se propagan de unos a otros a través de los bus. El bus "MAVLINK" contiene únicamente información relativa al protocolo. El bus "COMMAND" contiene algunas señales y otros buses de comunicaciones. Entre estos buses encontramos el bus "INPUTS", y dentro de él el bus "MISSION". Todas las señales que utilizaremos se incluirán en este bus, por lo que para modificar aspectos relacionados con el modo de funcionamiento autónomo habrá que acudir a dicho bus.

Existen además otros ficheros de configuración de hardware y de software. Todos se manejan desde el fichero de configuración principal, el fichero "config". Este fichero determina el modo de ejecución del programa y por tanto el tipo de inicialización de buses y de componentes. Además, activa y desactiva algunos bloques principales del proyecto en función de las necesidades el usuario.

Se puede concluir con que el desarrollo del UAV se encontraba en el punto de un vehículo aéreo pilotado que funcionaba correctamente en simulación. El modelo incluía capacidades de simulación, y las comunicaciones con los sensores estaban probadas. Todo el hardware estaba implantado, y el control y el estimador funcionaban correctamente. Era en definitiva un RPAS funcionando en simulaciones.

1.3. Motivación del proyecto

En apartados anteriores puede apreciarse el potencial del desarrollo de drones autónomos. Muchas de las de las actividades que realizan actualmente los UAVs y casi todas las que están por implementarse requieren de funcionamiento autónomo. Este desarrollo constituye en sí mismo una motivación. Profundizar en aspectos como el control, las comunicaciones y la construcción tiene un interés académico considerable, así como una aplicación al mundo empresarial y la vida laboral directa.

Este proyecto constituye una continuación de otros realizados en la escuela. A través de la realización de estos proyectos consecutivos, se va formando el cuerpo de un UAV capaz de navegar en exteriores construido íntegramente por ICAI. Es además un proyecto que da pie a otros, pues una vez cumplidos los objetivos, todavía quedarán numerosas funciones por implementar y aspectos que optimizar.

El objetivo de este proyecto es crear un cuadricóptero versátil, que permita funcionamiento pilotado y autónomo y al que se le puedan añadir funcionalidades específicas y equipos dedicados para adaptarlo a diversos escenarios. Este cuadricóptero puede convertirse en una herramienta que con pocas modificaciones sea capaz de cubrir las necesidades de cualquier usuario.

1.4. Técnicas de navegación

En este capítulo se considerarán diferentes técnicas empleadas en los sistemas de navegación autónomos actualmente. Se considerarán los sensores implicados, la carga computacional del sistema, necesidades del UAV y capacidades específicas que obtiene con cada sistema. A partir de esto se identificará cuál es más provechoso para nuestras necesidades.

En este aspecto es muy importante distinguir entre navegación en interiores y en exteriores. Dado que el UAV que ocupa este proyecto está diseñado para navegar en exteriores, nos centraremos en estos últimos.

1.4.1. Reconocimiento visual

Se trata de una de las técnicas que está ganando importancia actualmente. Se puede utilizar en exteriores y en interiores. Se caracteriza porque su sensorización principal es una cámara o sistema de visión externa.

En exteriores estos sistemas tienen múltiples aplicaciones. De cara a la navegación estos sistemas utilizan diferentes estrategias, detección de características del entorno observado para determinar la ruta (líneas de producción, líneas de cultivos), clasificación automatizada de objetos (clasificación de infraestructuras) o seguimiento y detección de elementos móviles (seguimiento de personas o vehículos). [7]

La principal ventaja de estos sistemas de navegación es su capacidad para crear rutas dinámicas, y su versatilidad. Son capaces de actuar en muchos ambientes y desarrollar sus funciones autónomas sin una ruta fija, sino generándola ellos mismos tras el procesado de la visión. [8] [9]

Implican una sensorización muy exigente, puesto que la visión debe ser fiable y de calidad. Además, la carga computacional es alta. Pese a ser un campo que va ganando importancia, decidimos que no será el que utilicemos. Para su desarrollo son necesarios equipos dedicados y suele ser una función a añadir tras desarrollar un sistema de navegación más tradicional. [10]

1.4.2. Sistemas de navegación inerciales

Se trata de sistemas de bajo coste compuestos por giróscopos, acelerómetros y magnetómetros. Son especialmente robustos en cuanto a que no requieren de información exterior para funcionar. Sin embargo, son los menos precisos a la hora de realizar rutas porque no pueden corregir los errores acumulados de cada uno de los sensores.
Estos sistemas necesitan utilizar un filtro extendido de Kalman para aunar las medidas de los sensores y obtener una medida precisa de los ángulos de Euler en cada momento. Basándose en estos ángulos, tras realizar una integración, el UAV es capaz de determinar su posición. Estos sistemas tienen la gran desventaja de la no corrección de errores. El tipo de sensores que utiliza suele tener una desviación continua que se va acumulando hasta generar valores altos de error en largos recorridos. Es por esto por lo que no utilizaremos estos sistemas.
[2]

1.4.3. Misión determinada por waypoints

Sistema de navegación más extendido. Son sistemas que poseen sensorización inercial, correcciones exteriores vía GPS y otros sensores para ejecutar su ruta.
Están enfocados hacia navegación en exteriores. Se da al UAV una misión generada en una base terrestre de control conformada por una matriz de waypoints, es decir, puntos que tiene que atravesar el UAV para completar su misión. Estos puntos se dan como referencias geográficas, de forma que el GPS es absolutamente necesario. En todo momento el UAV considerará dónde está el siguiente waypoint, y determinará la mejor trayectoria para alcanzarlo.
Es el sistema de navegación autónoma más extendido. Permite realizar rutas de forma precisa y con apoyo exterior. Necesita un módulo GPS, una IMU, barómetro y magnetómetro. Decidimos utilizar este sistema por su estandarización y prestaciones.

1.5. Objetivos

En este proyecto se han fijado los siguientes objetivos:

1. Capacidad de transmisión de la misión y correcta comunicación entre la base de tierra y el UAV.
2. Control autónomo completo: seguimiento de trayectoria definida por waypoints.
3. Simulación de todo el proceso de misión.
4. Comunicaciones y configuración de los sensores.

1.6. Cronograma y tareas

Tarea	Enero				Febrero				Marzo				Abril				Mayo			
	Sm 1	Sm 2	Sm 3	Sm 4	Sm 1	Sm 2	Sm 3	Sm 4	Sm 1	Sm 2	Sm 3	Sm 4	Sm 1	Sm 2	Sm 3	Sm 4	Sm 1	Sm 2	Sm 3	Sm 4
Anexo B																				
Instalación de Waijung																				
Familiarización con simulador																				
Creación rutas MissionPlanner																				
Protocolo envío de MissionPlanner																				
Implantación en el micro																				
Programación vuelo autónomo																				
Vuelo estacionario																				
Completar simulador																				
Simulaciones																				
Comunicaciones sensores																				
Ajustes finales																				
Redacción de la memoria																				

1.7. Recursos del proyecto

En este proyecto se emplearán los siguientes recursos:

- Cuadricóptero con microcontrolador Raspberry Pi Zero.
- Ordenador del laboratorio de proyectos con Matlab 2018a.
- Módulo GPS M8N de Ublox.
- Barómetro MS5611.

- Magnetómetro AK88963 incluido en la IMU MPU9250.
- Software gratuito MissionPlanner y FlightGear.
- Software gratuito Realterm

1.8. Estructura de la memoria

En el capítulo dos se hablará sobre el hardware del proyecto, el UAV. Se explicarán las distintas partes que lo componen y se expondrán los sensores necesarios para la navegación. Se detallará su configuración y sus comunicaciones con el microcontrolador.
En el capítulo tres se expone todo lo relativo a la generación y transmisión de la señal. Ocupa desde el diseño de la misión en MISSION PLANNER hasta que el UAV la adquiere y graba en su memoria. Incluye información sobre el formato de la misión, el canal de transmisión, el protocolo de comunicaciones, la base de tierra programada en Matlab y los procedimientos específicos de envío y recepción.
En el capítulo cuatro se explica el sistema de navegación implantado en el UAV para la ejecución de la misión. Se explica cómo se gestiona la matriz de waypoints, qué problemas de saturación de mando aparecen y que solución se les da a través de la gestión de referencia y finalmente se detalla el proceso de implantación en el modelo del UAV y el funcionamiento concreto de las distintas partes del código a través de diagramas de flujo.
En el capítulo cinco se justifica y explica el sistema de simulación. Se exponen los distintos elementos de la simulación, su función y su conexionado. Además, se detalla todo el proceso a simular, y se muestra la ejecución de las distintas simulaciones.
En el capítulo seis se exponen los resultados. Este capítulo está dividido en resultados de la simulación y resultados de las comunicaciones hardware. Se explicará exactamente qué se ha conseguido, qué aporta el proyecto y cómo se ha comprobado su funcionalidad.
En el capítulo siete se expondrán posibles futuros desarrollos aplicables a este proyecto en relación al sistema de navegación.
En el capítulo ocho se añade la bibliografía utilizada en este proyecto.
Para terminar, encontramos los anexos con la totalidad del código y un presupuesto final del proyecto.

Capítulo 2. HARDWARE

En este capítulo se habla sobre la estructura del UAV. En el primer apartado se expondrán los elementos que lo conforman, como el microcontrolador o los sensores. A continuación, se tratarán las comunicaciones y configuración de cada sensor propio de la navegación en exteriores.

2.1. Estructura del UAV

En esta sección se detallará el hardware del UAV.

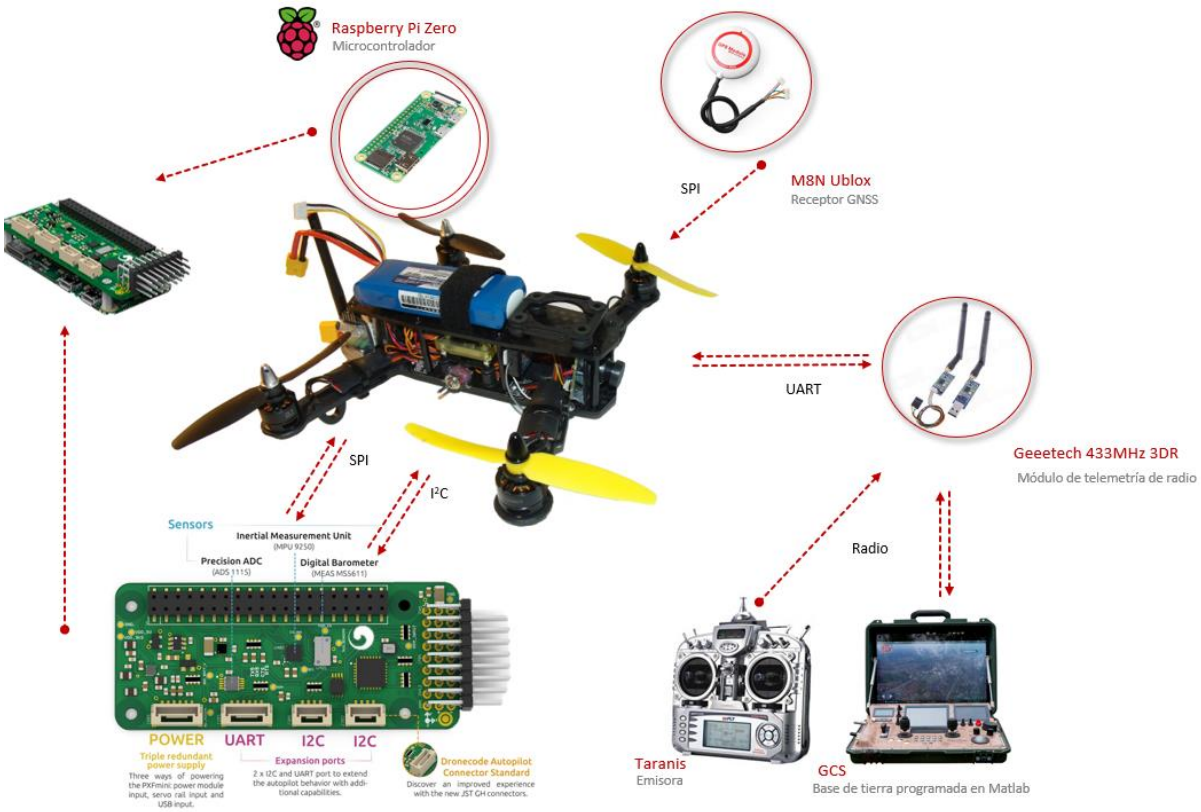


Figura 8. Esquema hardware

En la *Figura 9* puede observarse un esquema de los distintos elementos hardware del UAV y las distintas comunicaciones que emplean.

Puede observarse que el microcontrolador es una Raspberry Pi Zero. Se ha decidido utilizar este microcontrolador en vez del STM32 debido a que tiene una mayor capacidad computacional.
Se utiliza un módulo de telemetría para las comunicaciones con la base terrestre y la emisora.
Las comunicaciones con los sensores, su funcionamiento y programación se expondrán en los siguientes capítulos. Tanto el barómetro como el magnetómetro están integrados en la placa PFXmini. Es una placa diseñada para trabajar como esclava de la Raspberry Pi Zero. Tiene todos los puertos de comunicación para conectar sensores y sensores propios necesarios para la navegación.

2.2. Barómetro MS5611

En este capítulo se explican las características de este sensor y cómo entabla las comunicaciones con el microcontrolador. El sensor está integrado en la placa PFXmini.
Se trata de un sensor barométrico de pequeñas dimensiones capaz de ofrecer precisiones de altitud de hasta 10 cm. Su consumo de energía es muy reducido. Es capaz de ofrecer un valor de temperatura muy preciso con una precisión digital de 24 bits. Tiene diversas configuraciones para responder de la mejor manera posible a las necesidades del usuario. Permite comunicaciones por SPI y por I^2C de hasta 20 MHz. Decidimos utilizar una comunicación I^2C. El proceso de obtención de datos sigue el siguiente flujo. [4]

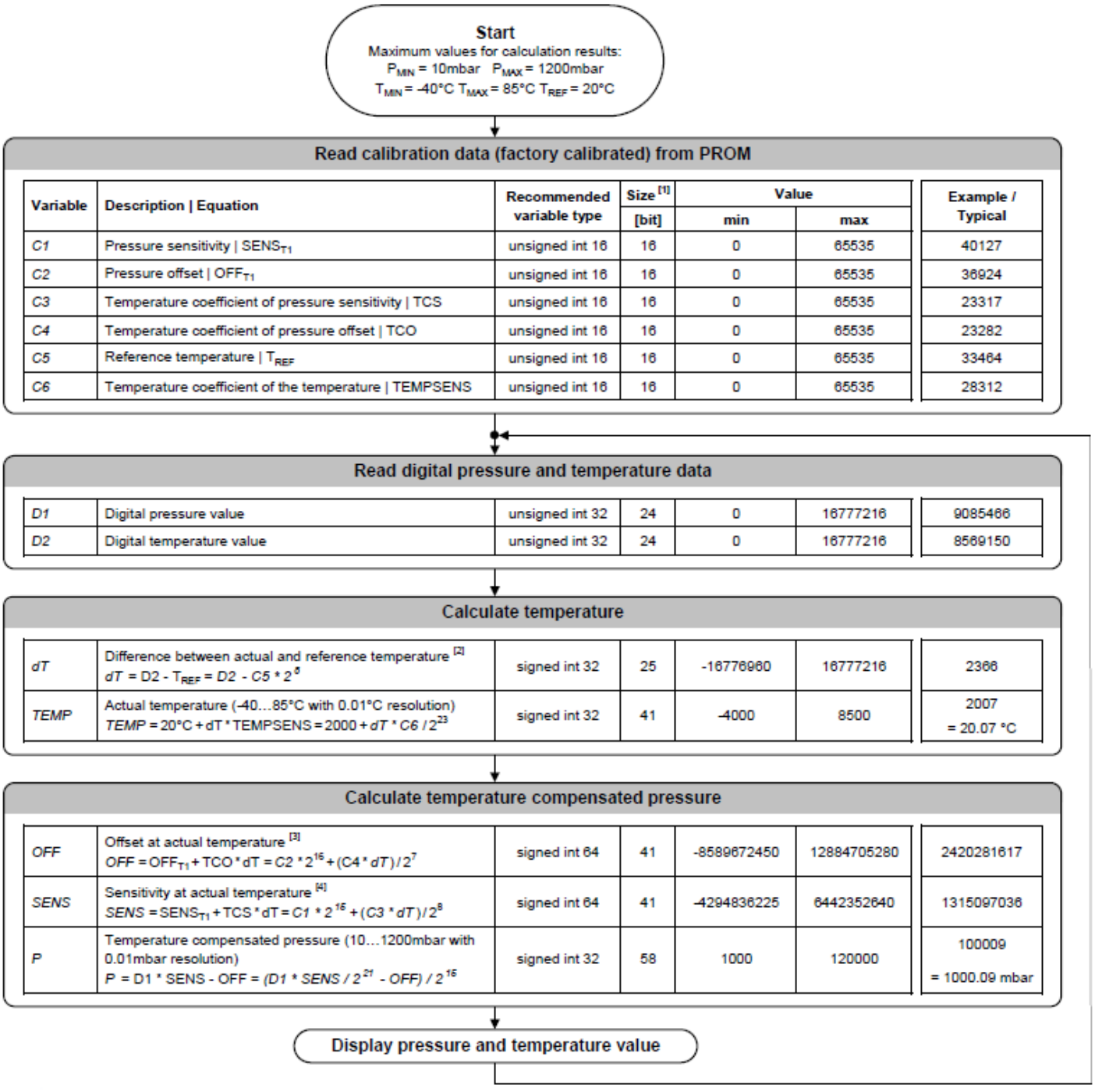


Figura 9. Diagrama de flujo para la obtención de medidas

En primer lugar, se realiza una lectura de la PROM para conocer valores de fábrica necesarios para la obtención de las medidas correctas. Esta primera lectura se almacena en variables internas y se realiza una única vez, considerándose la inicialización del sensor.

A continuación, se realizan las medidas de los registros "RAW_TEMP" y "RAW_PRESSURE". Estas medidas se realizan de forma consecutiva, es decir, en un período de muestreo (10 ms) se lee el primer valor y en el siguiente el segundo. Tras leer el primer valor es necesario preparar al sensor para la próxima lectura especificándole que realice la conversión digital-analógica, y así sucesivamente.
Una vez obtenidas estas medidas se procesan según las indicaciones de la *Figura 8* para obtener medidas fiables de la temperatura (TEMP) y la presión (P) en las que se encuentra el barómetro.

2.3. Magnetómetro AK8963

Este magnetómetro se integra en la IMU MPU9250, integrada en la placa PFXmini. Es un sensor accesorio a la misma. Funciona como elemento esclavo de la IMU, que a su vez es esclava del microcontrolador. Existen diversas formas de acceder al magnetómetro, pero finalmente se decide comunicarse únicamente con la IMU por SPI y que sea la IMU la que se comunique internamente con el sensor para configurarlo y obtener sus medidas por un bus auxiliar con comunicación I^2C.
Para hacer esto, debemos darle a la IMU la dirección del sensor esclavo, indicando si queremos que escriba y lea, dar las direcciones de los registros que queremos leer o escribir y facilitarle el contenido a escribir o el espacio a rellenar por la lectura. Será la IMU, a continuación, la que estructure esta información y obtenga así los datos del magnetómetro. [5]
Una vez hecho esto, se pedirá a la IMU la información de los registros correspondientes a la información del magnetómetro, y se almacenará en variables internas.

2.4. GPS M8N Ublox

El M8N es un receptor GNSS de la marca Ublox. El termino GNSS hace referencia a un receptor capaz de utilizar más de una constelación de satélites. En nuestro caso, se utilizarán las constelaciones GPS y GLONASS, americana y rusa.

Figura 10. Módulo M8N de Ublox

Tiene una precisión de hasta 5 metros, es ligero y se puede embarcar en el UAV. Dispone de un arranque rápido para comenzar a dar posición lo antes posible. Se comunicará con el microprocesador por I^2C y enviará las coordenadas geográficas en todo momento.
Para configurarlo es necesario utilizar una herramienta de Ublox, el Ublox Center. Desde ahí se determina el modo de funcionamiento, las constelaciones a utilizar, el formato de salida de datos y se guarda en su memoria no volátil. [6]

Capítulo 3. MISIÓN

En este proyecto se define el término misión como el conjunto de puntos en el espacio o waypoints que definen la trayectoria que ejecuta un UAV cuando funciona de forma autónoma.

En este capítulo se describirá todo lo referente a la misión. Por un lado, se hablará del protocolo que se empleará para su transmisión, de su diseño en Mission Planner, de su forma y de su lectura por la base de tierra programada en Matlab. Por otro lado, se hablará de los procedimientos específicos de envío y recepción.

3.1. Formato y diseño de la misión

La misión se genera en la base de tierra. En este proyecto se utilizará MISSION PLANNER. Es un software gratuito y cuyo uso está muy extendido en este tipo de aplicaciones. Permite monitorizar el estado del UAV y definir el plan de vuelo. Para ello, se marcan puntos sobre un mapa físico proporcionado por Google Maps.

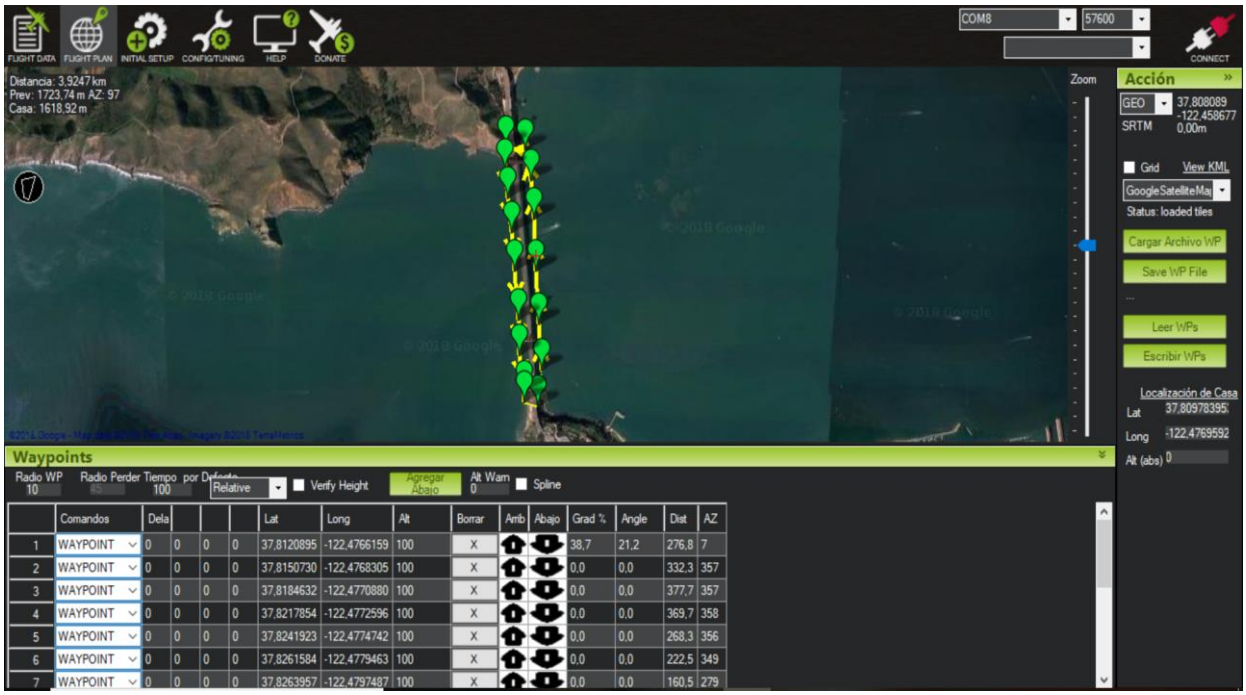


Figura 11. Interfaz MISSION PLANNER

Pese a que se utilizará MISSION PLANNER para generar la ruta y controlar el estado del cuadricóptero, para tener un mayor control sobre el protocolo de envío y los mensajes concretos que se utilizan diseñaremos nuestra propia base de tierra en Matlab para cargar la misión. Así, se creará la misión utilizando la interfaz de MISSION PLANNER y se grabará en el ordenador utilizando su función "Save WP File" en un archivo de texto. En la *Figura 12* se puede ver el formato de este tipo de archivos. [7]

MISSION PLANER agrupa los waypoints que conforman la misión matricialmente, siendo las filas el número de waypoints y habiendo un total de doce columnas, que incluirán los puntos y los parámetros asociados a ellos y a la misión.

```
QGC WPL 110
0	1	0	0	0	0	0	0	0	0	0	1
1	0	0	16	0.000000 0.000000 0.000000 0.000000 40.451066	-3.693960	100.000000	1
2	0	0	16	0.000000 0.000000 0.000000 0.000000 40.450993	-3.693992	100.000000	1
3	0	0	16	0.000000 0.000000 0.000000 0.000000 40.451001	-3.694164	100.000000	1
4	0	0	16	0.000000 0.000000 0.000000 0.000000 40.451262	-3.694228	100.000000	1
5	0	0	16	0.000000 0.000000 0.000000 0.000000 40.451336	-3.694051	100.000000	1
```

Figura 12. Formato de archivo de waypoints

A continuación, la cargaremos en nuestra base de tierra programada en Matlab y Simulink para realizar el envío al UAV. De esta forma, se puede controlar el proceso de envío completamente y monitorizar los formatos de envío. Sin embargo, se utilizará un protocolo de envío normalizado en MAVLink, por lo que su funcionamiento está asegurado con todas las bases de tierra que utilicen este protocolo.

A la hora de definir la misión hay que describir los parámetros que intervienen en su configuración. En el protocolo MAVLink, cuando se envía una misión, se definen hasta doce parámetros que determinan cada punto y describen cómo se envía y cómo se interpreta. Dado que el funcionamiento del modo autónomo viene determinado por los objetivos, algunos de estos parámetros no los determinará la base de tierra, sino que estarán 'hardcoded' para que el drone funcione de acuerdo a nuestros requerimientos. Así, de los doce campos, solo determinarán la misión cuatro, la secuencia o número de waypoint y los tres términos que lo sitúan en el espacio. La definición de cada uno de los doce campos aparece en la siguiente tabla:

Parámetro	Significado	Hardcoded
Secuencia	Indica a qué waypoint hace referencia la fila	-
Frame	Sistema de referencia en el que se define el waypoint	0 Sistema LLA
Current	Waypoint actualmente en uso	0 - False
Command	Orden asociada al waypoint	16 Go to Waypoint
Param 1	-	0
Param 2	-	0
Param 3	-	0
Param 4	-	0
X	Longitud	-
Y	Latitud	-
Z	Altitud	-
Autocontinue	Pasar automáticamente de un waypoint a otro	1 Activa la secuencia automática

Para definir los puntos en el espacio se decida utilizar el sistema LLA, Latitud Longitud y Altitud. El sistema LLA o de coordenadas geográficas es un sistema de posicionamiento. Referencia cualquier punto del globo terrestre utilizando tres valores, dos para el posicionamiento horizontal y uno para determinar la altura.

Coordenadas geográficas:

En este modelo se determinan dos planos de referencia. Por un lado, el Ecuador, un plano perpendicular al eje de rotación de la Tierra que contiene el círculo máximo imaginario del planeta. Todos los planos paralelos a este a lo largo del eje de rotación se denominan paralelos, y su radio va decreciendo hasta alcanzar los polos. Por otro lado, se definen los meridianos como el haz de planos perpendiculares al Ecuador que contienen al eje de rotación. Todos tienen el mismo perímetro, y han existido diversos meridianos utilizados como referencia. Desde 1984 el más aceptado como referencia es el Meridiano de Greenwich, aquel que pasa por el London´s Greenwich Observatory. Sobre estos dos planos de referencia se dan la Latitud y la Longitud. La latitud hace referencia al ángulo entre cualquier paralelo y el Ecuador, en el rango de +-90. La longitud hace lo propio entre cualquier meridiano y el meridiano referencia, en el rango de +-180.

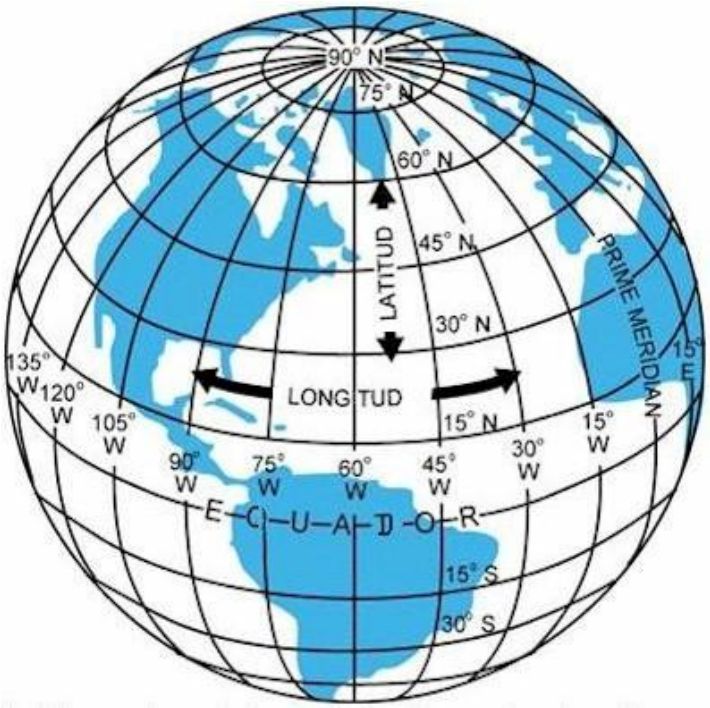


Figura 13. Mapa con meridianos y paralelos

El valor de la altura depende del modelo terrestre escogido. En nuestro caso será el WGS84.

3.2. MAVLink

MAVLink (**M**icro **A**ir **V**ehicules Link) es un protocolo de comunicaciones para pequeños UAVs. Fue lanzado por primera vez en 2009 por Lorenz Meier. Se utiliza para establecer comunicaciones entre estaciones de tierra y vehículos autónomos, principalmente micro UAVs, aunque también puede emplearse en barcos y vehículos de tierra. Soporta hasta 30 tipos de vehículos como ala fija, multirotores, helicópteros, submarinos o globos y permite controlar determinados periféricos de los vehículos, como cámaras o sensores.
MAVLink nace con la idea de convertirse en el primer protocolo de comunicaciones extendido para pequeños UAVs. Su objetivo es convertirse en el estándar de las bases de tierra de código abierto disponibles en Internet, de forma que cualquier vehículo que tuviese incorporado un codificador y decodificador del protocolo MAVLink y que respetase los procedimientos específicos del protocolo para intercambios de información podría integrarse naturalmente con cualquier base de tierra gratuita. Además, generalizar

este protocolo serviría para crear una comunidad más activa y unida en torno a este tipo de proyectos. De acuerdo a esta idea de generalización, MAVLink permite añadir mensajes propios al protocolo, de forma que sea un protocolo válido para cualquier tipo de aplicación.

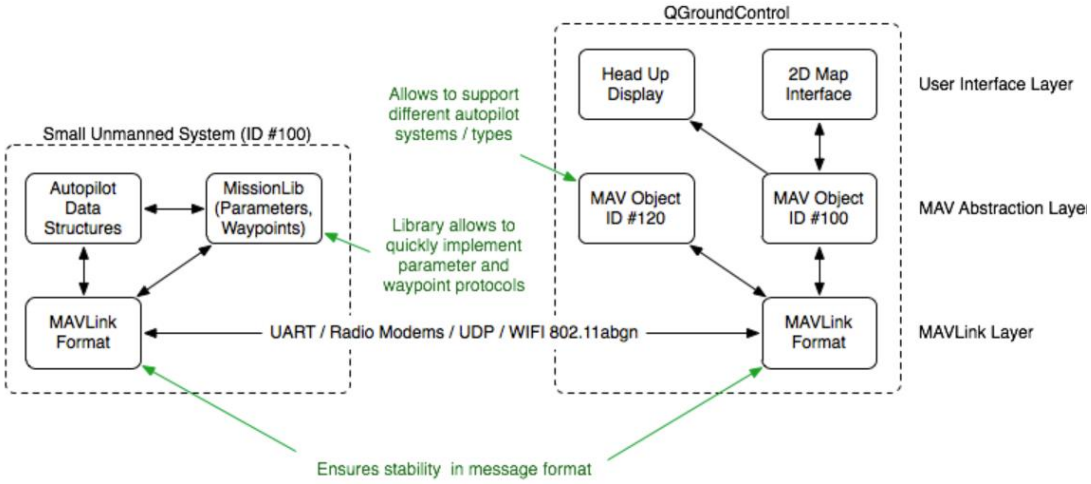


Figura 14. Explicación de la función de MAVLink en un sistema

Es un software LGPL (Licencia Pública General para Bibliotecas), por lo que se puede compartir y modificar libremente por cualquier usuario. Esta licencia permite que MAVLink se use en proyectos open-source y close-source, ya que pueden crearse licencias sobre el trabajo derivado del programa original. Su condición de software libre y su relativa sencillez han sido determinantes para su expansión desde su lanzamiento hace casi 10 años. Actualmente, los principales autopilotos y bases de tierra que se utilizan en este tipo de proyectos utilizan el protocolo MAVLink, por lo que se decide que el UAV utilice este protocolo para poder integrarse naturalmente con otros componentes del entorno de los UAV.

Composición de los mensajes:

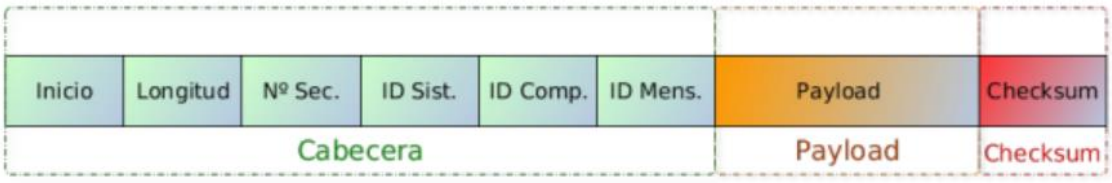


Figura 15. Estructura de un mensaje MAVLink

Los mensajes de MAVLink tienen una cabecera de 6 bytes, dos bytes finales de detección de errores y un payload (la información relevante que se transmite en el mensaje) variable de 0 a 255 bytes. La cabecera, que siempre tiene la misma forma, está formada por los siguientes campos:

Byte Index	Content	Value	Explanation
0	Packet start sign	v1.0: 0xFE (v0.9: 0x55)	Indicates the start of a new packet.
1	Payload length	0 - 255	Indicates length of the following payload.
2	Packet sequence	0 - 255	Each component counts up his send sequence. Allows to detect packet loss
3	System ID	1 - 255	ID of the SENDING system. Allows to differentiate different MAVs on the same network.
4	Component ID	0 - 255	ID of the SENDING component. Allows to differentiate different components of the same system, e.g. the IMU and the autopilot.
5	Message ID	0 - 255	ID of the message - the id defines what the payload "means" and how it should be correctly decoded.

El byte de inicio, cuyo valor hexadecimal es 0xFE, permite detectar la llegada de un nuevo mensaje. Va seguido por el byte de longitud, que indica el tamaño del payload del mensaje. A continuación, encontramos el byte de secuencia. Es un identificador del mensaje. Los dos siguientes bytes hacen referencia al sistema y al componente que envía el mensaje. Son elementales cuándo se controlan dos UAV al mismo tiempo, pero no son necesarios en este proyecto. El sexto y último byte de cabecera indica el ID del mensaje que se está enviando, y por tanto la forma exacta de su contenido.

Los dos bytes finales constituyen un sistema de detección de errores conocido como Checksum o suma de verificación. La idea de este sistema es que el emisor del mensaje envíe el valor calculado de Checksum sobre el mensaje original y que el receptor del mensaje calcule por el mismo método el Checksum del mensaje recibido, comprobando la integridad de éste al comparar ambos valores. Cuando el Checksum calculado no coincide con el enviado, se califica el mensaje como un mensaje corrupto o como ruido.

Por tanto, el encapsulamiento del payload es siempre de 8 bytes, de forma que el mensaje más largo que puede transmitirse utilizando el protocolo será de 263 bytes.
A la hora de recibir e identificar mensajes, MAVLink trabaja con el byte de inicio, la longitud del mensaje y el Checksum. En todo momento se encuentra a la espera de recibir el byte de inicio de mensaje. En cuanto recibe un byte con este valor considera que puede

estar recibiendo un mensaje, de forma que comprueba que efectivamente sea un mensaje y no ruido. Para ello, lee el byte siguiente, que teóricamente determina la longitud del payload. A continuación, calcula el Checksum de este teórico payload, y lo compara con los dos bytes siguientes. Si coinciden estos dos bytes teóricos de Checksum con el que ha calculado, interpreta que realmente ha recibido un mensaje, y decodifica su contenido. En caso contrario, descarta ese teórico inicio de mensaje y considera que es ruido o que ha habido errores en el envío del mensaje. Este funcionamiento es el que da sentido al byte de secuencia que aparece en la cabecera de cada mensaje. En todo momento se calculará el ratio de transmisión correcta de mensajes a través de la información sobre mensajes perdidos o descartados que da el byte de secuencia. En caso de que se produzcan errores sistemáticos en la transmisión de mensajes por una mala configuración, MAVLink enviará un mensaje de alerta informando sobre la inestabilidad de la comunicación.
Cuando MAVLink detecta un mensaje y lo acepta según este procedimiento, pasa a decodificarlo. Para ello, en primer lugar, determina qué mensaje está recibiendo fijándose en la información que le otorga el sexto byte de cabecera, el ID del mensaje, para conocer la estructura del payload. El payload de todos los mensajes de MAVLink está definido, de forma que todos los campos que lo componen son conocidos. Así, se separan todos los campos y se asignan a variables para su posterior uso o almacenamiento.

Procedimiento de transmisión de misión:

Para transmitir una misión, MAVLink tiene un procedimiento concreto de actuación. Esta normalización es útil para que cualquier UAV con este procedimiento programado pueda comunicarse con cualquier base terrestre que utilice MAVLink. Se puede encontrarlo en la página web de ardupilot en el apartado de misiones y waypoints. Este procedimiento

define el orden de envío de los mensajes y el final de la comunicación. Aparece en la *Figura 18.*

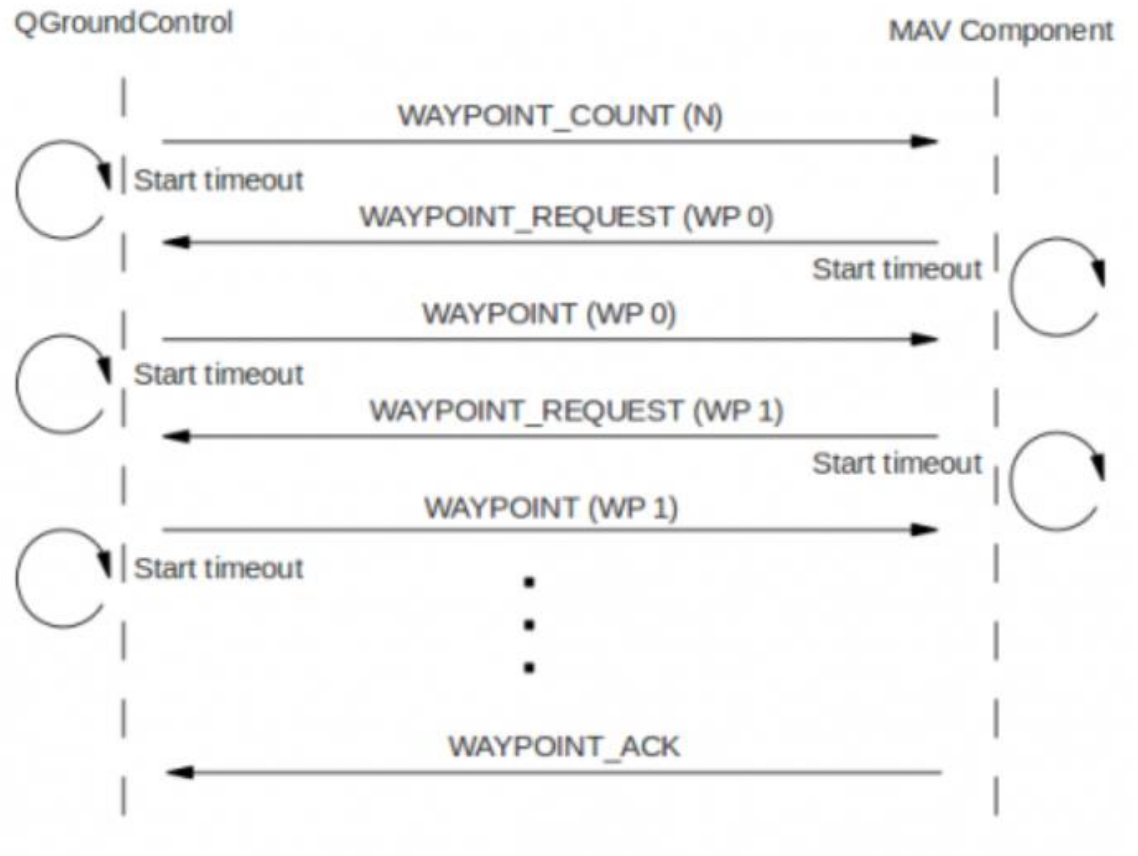


Figura 16. Procedimiento de transmisión de misión

Para implantar este procedimiento en primer lugar hace falta describir el método de decodificación de los mensajes recibidos y el método de construcción de los mensajes a enviar. En una primera instancia todo el desarrollo se hace desde el punto de vista del UAV, puesto que teóricamente vamos a utilizar como base de tierra MISSION PLANNER, que utiliza MAVLink y tiene ya incorporados todos los mensajes y procedimientos. [4]

Se pueden encontrar estos mensajes en el catálogo de mensajes del protocolo. Una vez conocidos, definimos primero la decodificación de los mensajes recibidos. Para hacer esto, en primer lugar, se determina el mensaje que se está recibiendo. Se extrae el

payload, desechando los bytes de cabecera, longitud, número de mensaje, sistema, componente, secuencia y CRC. Una vez hecho esto se distribuye este payload en variables tal y como indica la documentación del mensaje, para poder utilizar la información que nos está llegando. Cabe destacar que la documentación no describe exactamente el formato del mensaje, puesto que las variables que define no tienen por qué aparecer en ese orden. Por defecto MAVLink ordena el contenido de los mensajes de acuerdo a su longitud, de forma que los campos de mayor tamaño irán primero, aunque en la documentación puedan aparecer los campos en otro orden.

Esto es vital a la hora de decodificar los mensajes correctamente. Una vez añadidos los mensajes al decodificador, definimos el método de construcción de los mensajes que vamos a enviar. En este caso, tenemos que definir el contenido del mensaje, puesto que ya existen funciones que lo encapsulan de acuerdo a los requerimientos del protocolo. Por tanto, definimos las variables que constituirán el payload del mensaje y las colocamos en el orden adecuado. Estas variables las definirá el procedimiento de la imagen.

Una vez establecida la comunicación entre la base y el UAV, se procede a enviar la misión. Como puede observarse en la *Figura 16*, la base comenzará el procedimiento de carga enviando el mensaje #44, WAYPOINT_COUNT, que contiene el número de waypoints que forman la misión.

El drone, si la primera transmisión se produce correctamente, responderá solicitando el primer waypoint utilizando el mensaje #40 de MAVLink, el MISSION_REQUEST. A medida que se le vayan solicitando, la base enviará cada waypoint utilizando el mensaje #39, MISSION_ITEM.

Cuando el UAV haya solicitado y recibido la matriz completa de waypoints, enviará un mensaje confirmando la recepción de la misión. Será el mensaje #47, ACKNOWLEDGE.

3.3. Base terrestre

En una etapa intermedia del proyecto se decide que lo mejor es programar y utilizar nuestra propia base de tierra. Esto nos permite controlar el procedimiento de envío, la definición de los mensajes y ver cómo están efectuándose las comunicaciones.

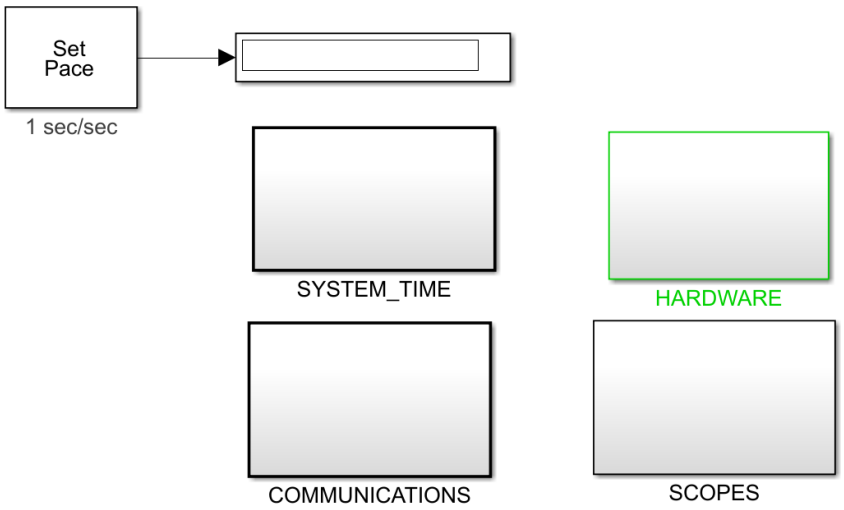


Figura 17. Estructura de la base de tierra

Como se puede observar en la *Figura 17* la base de tierra tendrá cuatro bloques principales. En el bloque de "HARDWARE" se inicializan los puertos que se utilizarán en las comunicaciones de la base terrestre. Será un puerto serie funcionando a 57.600 baudios.

En el bloque "SYSTEM_TIME" controla el tiempo de la base terrestre, es decir, su cuenta de ejecuciones y su reloj interno.

Los bloques más interesantes son "COMMUNICATIONS" y "SCOPES". En el bloque "COMMUNICATIONS" encontramos el codificador MAVLink (para generar los mensajes que queremos enviar) y el decodificador MAVLink (para comprender los mensajes que recibimos). En estos dos bloques se añaden los mensajes necesarios para el procedimiento de transmisión de misión. Encontramos a su vez el bloque "MISSION MANAGEMENT", en el que programaremos la máquina de estados que gestionará el envío de la misión. Este bloque es la base para desarrollar el modo autónomo. Por último, encontramos los botones de lectura de misión (READ) y de envío (SEND). Recordemos que la base terrestre debe leer la misión de un archivo de texto generado en MISSION PLANNER y enviarla al UAV.

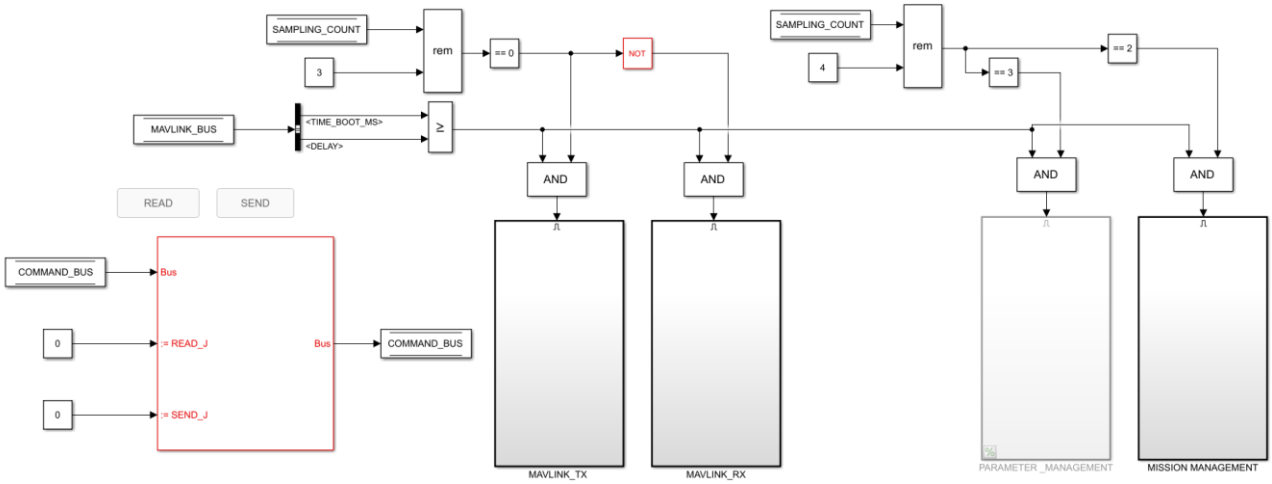


Figura 18. Bloque "COMMUNICATIONS" de la base

El bloque "SCOPES" es especialmente útil y una de las justificaciones de esta base de tierra propia. Nos permite ver en todo momento qué mensajes se están enviando y recibiendo, de forma que podremos localizar errores y solventarlos. Está formado únicamente por visualizadores del buffer.

3.4. Implementación de mensajes

Para poder ejecutar el procedimiento es necesario desarrollar e implantar los mecanismos necesarios para codificar y decodificar cada mensaje según nuestra necesidad tanto en el UAV como en la base terrestre que vamos a programar. El vehículo en el que se cargará la misión necesitará decodificar los mensajes #44 y #39 y codificar los mensajes #40 y #47. La base terrestre por el contrario decodificará los mensajes #40 y #47 y codificará los mensajes #44 y #39.

Para implantar los mensajes necesitamos conocer de qué campos está compuesto su payload. Los buscamos en el documento que contiene los mensajes de MAVLink y definimos su estructura y los campos que nos interesan:

MISSION_ITEM (#39)

Message encoding a mission item. This message is emitted to announce the presence of a mission item and to set a mission item on the system. The mission item can be either in x, y, z meters (type: LOCAL) or x:lat, y:lon, z:altitude. Local frame is Z-down, right handed (NED), global frame is Z-up, right handed (ENU). See also http://qgroundcontrol.org/mavlink/waypoint_protocol.

Field Name	Type	Description
target_system	uint8_t	System ID
target_component	uint8_t	Component ID
seq	uint16_t	Sequence
frame	uint8_t	The coordinate system of the MISSION. see MAV_FRAME in mavlink_types.h
command	uint16_t	The scheduled action for the MISSION. see MAV_CMD in common.xml MAVLink specs
current	uint8_t	false:0, true:1
autocontinue	uint8_t	autocontinue to next wp
param1	float	PARAM1, see MAV_CMD enum
param2	float	PARAM2, see MAV_CMD enum
param3	float	PARAM3, see MAV_CMD enum
param4	float	PARAM4, see MAV_CMD enum
x	float	PARAM5 / local: x position, global: latitude
y	float	PARAM6 / y position: global: longitude
z	float	PARAM7 / z position: global: altitude (relative or absolute, depending on frame.

Figura 19. Mensaje #39

Este mensaje lo envía la base terrestre al UAV con la información de los waypoints.

La variable “seq” indica el waypoint que se está enviando. La variable frame indica el sistema en el que se envían las referencias, en nuestro caso el sistema de posicionamiento geográfico LLA (Longitud Latitud Altitud). La variable “command” indicará la orden asociada a ese waypoint. Este campo es indiferente en nuestro UAV, pues el modo de funcionamiento será único, y no se tendrán en cuenta los distintos tipos de órdenes. Sin embargo, en futuros desarrollos puede ser interesante tener en cuenta este indicador, que definirá el tipo de trayectoria que debe describir nuestro UAV y cómo debe hacerla. Tanto la variable “current” como “autocontinue” definen parámetros de funcionamiento. En nuestro caso serán también indiferentes, pues solo esperamos un tipo de comportamiento del UAV.

EL waypoints queda definido con los campos “param” y XYZ. En este proyecto, los campos “param” son indiferentes, por lo que la información del waypoints la darán los campos XYZ.

MISSION_REQUEST (#40)

Request the information of the mission item with the sequence number seq. The response of the system to this message should be a MISSION_ITEM message. http://qgroundcontrol.org/mavlink/waypoint_protocol

Field Name	Type	Description
target_system	uint8_t	System ID
target_component	uint8_t	Component ID
seq	uint16_t	Sequence

Figura 20. Mensaje #40

Este mensaje lo envía el UAV a la base solicitando cada waypoint.

En nuestro caso, dado que la comunicación se establece punto a punto, no es necesario definir un target_system ni target_component, por lo que se fijarán por defecto como 0x00. Será la variable seq del mensaje la que defina el waypoint que se está solicitando.

MISSION_COUNT (#44)

This message is emitted as response to MISSION_REQUEST_LIST by the MAV and to initiate a write transaction. The GCS can then request the individual mission item based on the knowledge of the total number of MISSIONs.

Field Name	Type	Description
target_system	uint8_t	System ID
target_component	uint8_t	Component ID
count	uint16_t	Number of mission items in the sequence

Figura 21. Mensaje #44

Mensaje de inicio del envío de la misión. Lo manda la base al UAV con el número de waypoints que componen la misión.

De nuevo, los dos primeros campos estarán fijados a 0x00. El campo count indicará el número de waypoints que componen la misión.

MISSION_ACK (#47)

Ack message during MISSION handling. The type field states if this message is a positive ack (type=0) or if an error happened (type=non-zero).

Field Name	Type	Description
target_system	uint8_t	System ID
target_component	uint8_t	Component ID
type	uint8_t	See MAV_MISSION_RESULT enum

Figura 22. Mensaje #47

Mensaje de fin de envío de misión. Lo envía el UAV a la base indicando si se ha recibido correctamente la información.

En este mensaje únicamente hay que definir el campo “type”, que será ‘1’ si la misión se ha recibido correctamente y ‘0’ en caso contrario.

Para implementar los mensajes en el UAV acudimos al bloque de comunicaciones del autopiloto. En este encontramos dos bloques que sirven para codificar los mensajes que enviará el UAV y para decodificar los que recibe. El MAVLINK_TX y el MAVLINK_RX respectivamente.

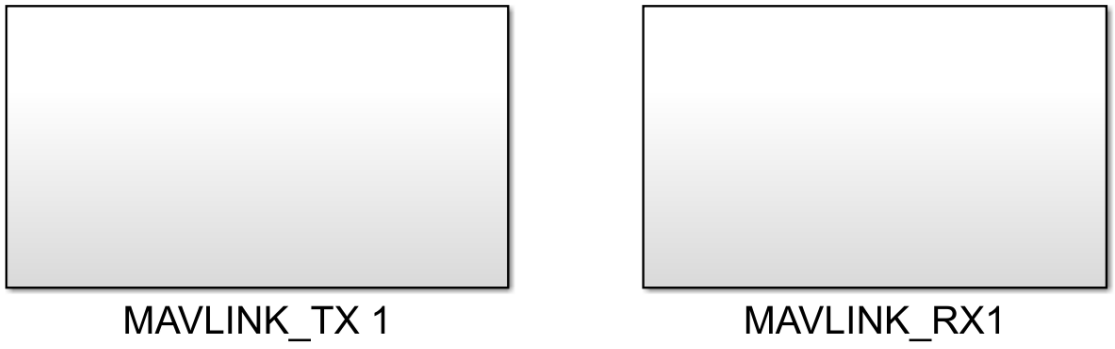


Figura 23. Imagen del bloque “COMMUNICATIONS”

Es en estos bloques donde debemos añadir los mensajes que vamos a utilizar. Abrimos el bloque MAVLINK_TX para empezar a añadir los mensajes. En la función escribimos e implementamos los nuevos mensajes que vamos a usar. Dado que es el envío del UAV, únicamente serán los mensajes #40 y #47.

```
if MESSAGE_ID(41) % MISSION_REQUEST (  #40 ) - 12 bytes
    MESSAGE_INDEX = uint8(41);
    % Request the information of the mission item with the sequence number
    % seq. The response of the system to this message should be a
    % MISSION_ITEM message. http://qgroundcontrol.org/mavlink/waypoint_protocol
    %
    % Field Name        Type        Description
    % target_system     uint8_t     System ID
    % target_component  uint8_t     Component ID
    % seq               uint16_t    Sequence
    % Payload definition
    PAYLOAD = [ typecast(uint16(COMMAND.MISSION_SEQ),'uint8')'
                uint8(MAVLINK_OUT.TARGET_SYSTEM)
                uint8(MAVLINK_OUT.TARGET_COMPONENT)            ];
    % MAVLINK message codification
    [MESSAGE,MESSAGE_LEN] = MAVLINK_CODER(MESSAGE_INDEX-1,SYSTEM_ID,...
                            COMPONENT_ID,PAYLOAD,SEQUENCE_NUMBER,...
                            MAVLINK_MESSAGE_CRCS(MESSAGE_INDEX));
    % Buffer update
    BUFFER(BUFFER_LEN+1:BUFFER_LEN+MESSAGE_LEN) = MESSAGE;
    BUFFER_LEN = BUFFER_LEN+MESSAGE_LEN;
    % SEQUENCE_NUMBER update
    if SEQUENCE_NUMBER == uint8(255)
        SEQUENCE_NUMBER = uint8(0);
    else
        SEQUENCE_NUMBER = SEQUENCE_NUMBER + uint8(1);
    end
end
```

Figura 24. Integración mensaje #40

Como puede observarse, pese a ser el mensaje #40, en el bloque de envío sumamos uno a su MESSAGE_ID. Sin embargo, al codificar el mensaje, le restamos uno de forma que el sistema que reciba el mensaje sepa que es el #40. El único contenido variable del payload es el COMMAND.MISSION_SEQ, ya que el resto está fijado a 0x00. El valor de esta variable determinará el waypoint que estamos solicitando a la base terrestre. Su valor se fijará en la máquina de estados de recepción del mensaje que programaremos a continuación. Además, puede apreciarse que al construir el payload situamos en primer lugar las variables mayores, como se comentó anteriormente.

El resto de mensajes se implementarán de forma similar, tanto en la base terrestre como en el UAV, respetando el formato normalizado de MAVLink y de acuerdo al datasheet.

3.5. Procedimiento de envío

En la base de tierra programamos el procedimiento de envío de misión a través de una máquina de estados. Para llevar a cabo este procedimiento hay que activar y desactivar los mensajes en unos plazos concretos. Además, hay que determinar su contenido en cada caso. El código completo referido a este apartado se encuentra en el Anexo A.

Para comenzar el procedimiento de envío se pulsa el botón "SEND" en el bloque de simulación. En este momento se activa el mensaje #44, que se enviará cada x milisegundos durante x tiempo. Este mensaje contendrá la información relativa al número de waypoints. Cuando acabe de enviar este mensaje, estará recibiendo una solicitud de waypoint del UAV. Utilizando el mensaje #39 enviará el waypoint solicitado. A medida que el UAV vaya recibiendo los waypoints, irá solicitando el siguiente. Una vez que haya enviado el mismo número de waypoints que indicó en el mensaje #44, recibirá un mensaje de acknowledge del UAV indicando si ha recibido correctamente la misión.

Es importante señalar que, en este procedimiento, el mensaje #39 está activo desde la última emisión del mensaje #44 y hasta la recepción del mensaje de acknowledge. El contenido del mensaje, es decir, la fila de la matriz de waypoints que enviará la base de tierra, lo determina una variable que se actualiza según la información que solicita el UAV mediante el mensaje #40. De esta forma, la única manera de interrumpir la comunicación es habiendo enviado toda la misión y habiendo recibido el mensaje de terminación. Es posible que el UAV solicite repetidamente el mismo waypoint hasta que lo reciba correctamente.

Una vez recibido el mensaje #47, el de acknowledge, se desactivarán todos los mensajes de transmisión de misión y se saldrá de la máquina de estados.

3.6. Procedimiento de recepción

Este procedimiento se lleva a cabo en el UAV, y sirve para obtener y guardar la misión en la memoria del vehículo. A la hora de la transmisión de la misión, el UAV está a la

espera, ya que el encargado de iniciar la transmisión es la base de tierra. El código completo de este apartado se encuentra en el Anexo B.

Para llevar a cabo este procedimiento diseñamos e implantamos una máquina de estados en un nuevo bloque que se creará en el bloque de comunicaciones. Este bloque se llamará MISSION MANAGEMENT y contendrá toda la lógica necesaria para el funcionamiento autónomo.

Esta máquina se activará exclusivamente cuando se detecte que se ha recibido el mensaje #44. De este se extraerá el número de waypoints que componen la misión. Una vez conocido, se activará el mensaje de solicitud de ítem de misión, el #40. Para enviar este mensaje tenemos que definir el sistema, el componente y el número de ítem. Los dos primeros serán por defecto '0', puesto que establecemos la comunicación de punto a punto y no es necesario definir ningún sistema o componente concreto. La variable de secuencia será la que determinará el waypoint que vamos a recibir, por lo que iremos incrementando su valor de uno en uno hasta recibir todos los waypoints. Este incremento se realizará en función del tiempo, es decir, se calcula el tiempo necesario para realizar varias solicitudes del mismo waypoint, de forma que se incrementen las posibilidades de obtener correctamente el waypoint, y a continuación comienza a solicitarse el siguiente waypoint. Cuando el número de secuencia haya alcanzado el valor de cuenta extraído del mensaje #44, activamos el mensaje de acknowledge, terminando el envío de misión. Una vez enviado este mensaje, desactivamos el envío de los mensajes #40 y #47, y salimos de la máquina de estados. De esta forma, habremos cargado la misión en una matriz denominada MISSION_INFO, que usaremos para generar las referencias que daremos al control para realizar la ruta.

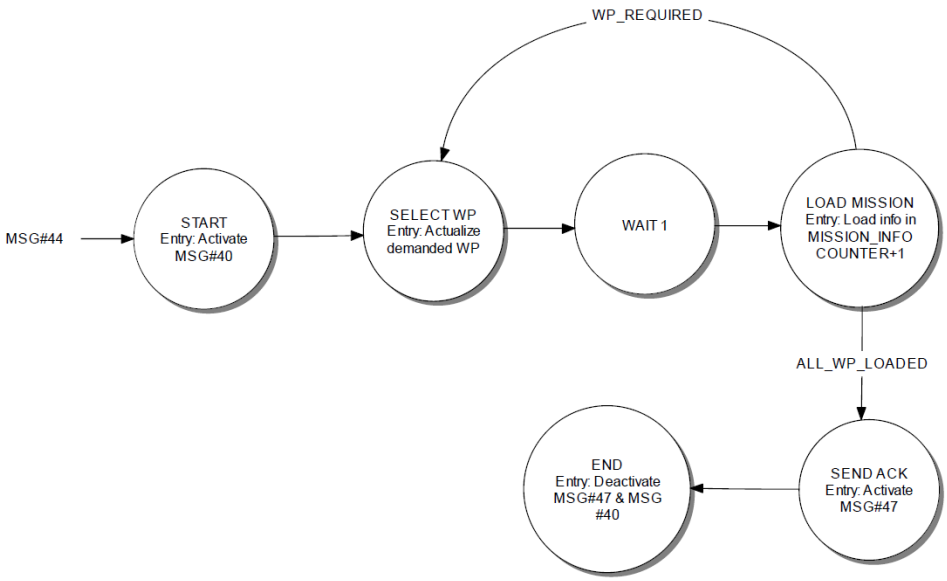


Figura 25. Máquina de estados de recepción de misión

Capítulo 4. NAVEGACIÓN

En este capítulo se explicará cómo funciona el modo autónomo. Toda la gestión se hace en el bloque MISSION MANAGEMENT, aunque algunas decisiones se toman en la máquina de estados de vuelo, del bloque STATE_MACHINE. En esta máquina habrá un estado dedicado al vuelo autónomo. Para acceder a este estado se utilizará uno de los interruptores de la controladora.

4.1. Gestión de waypoints

Para generar la ruta el modo autónomo debe gestionar los waypoints que componen la misión que ha recibido anteriormente. Esta gestión consiste en ir seleccionando el nuevo waypoint destino e ir transformándolo a coordenadas locales.

En modo autónomo se generan referencias en coordenadas locales. Las coordenadas locales funcionan en un sistema de referencia propio del UAV. El origen de este sistema suele ser el punto en el que despega el drone. En el período de inicialización, antes de activar los motores, el UAV toma la posición que indica el GPS como origen de coordenadas. Para ello transforma las coordenadas geográficas que le proporciona el GPS (que será el M8N de Ublox) a coordenadas de *flat earth* o tierra plana. Este tipo de coordenadas toma como origen el punto de corte entre el Ecuador (origen de la Latitud) y el Meridiano de Greenwich (origen de la Longitud). Una vez conocido el punto de inicialización del vehículo se realiza una traslación al mismo desde el origen de *flat earth*, de forma que todas las referencias tomarán como origen este punto. Los ejes coordenados, XYZ, son análogos a las coordenadas NED. El eje X apuntará al norte geográfico, el eje Y apuntará al este geográfico y el eje Z apuntará a la superficie terrestre. Así, se calculará la posición de cada waypoint en estas coordenadas locales. El cálculo se realizará cada vez que se seleccione un nuevo waypoint.

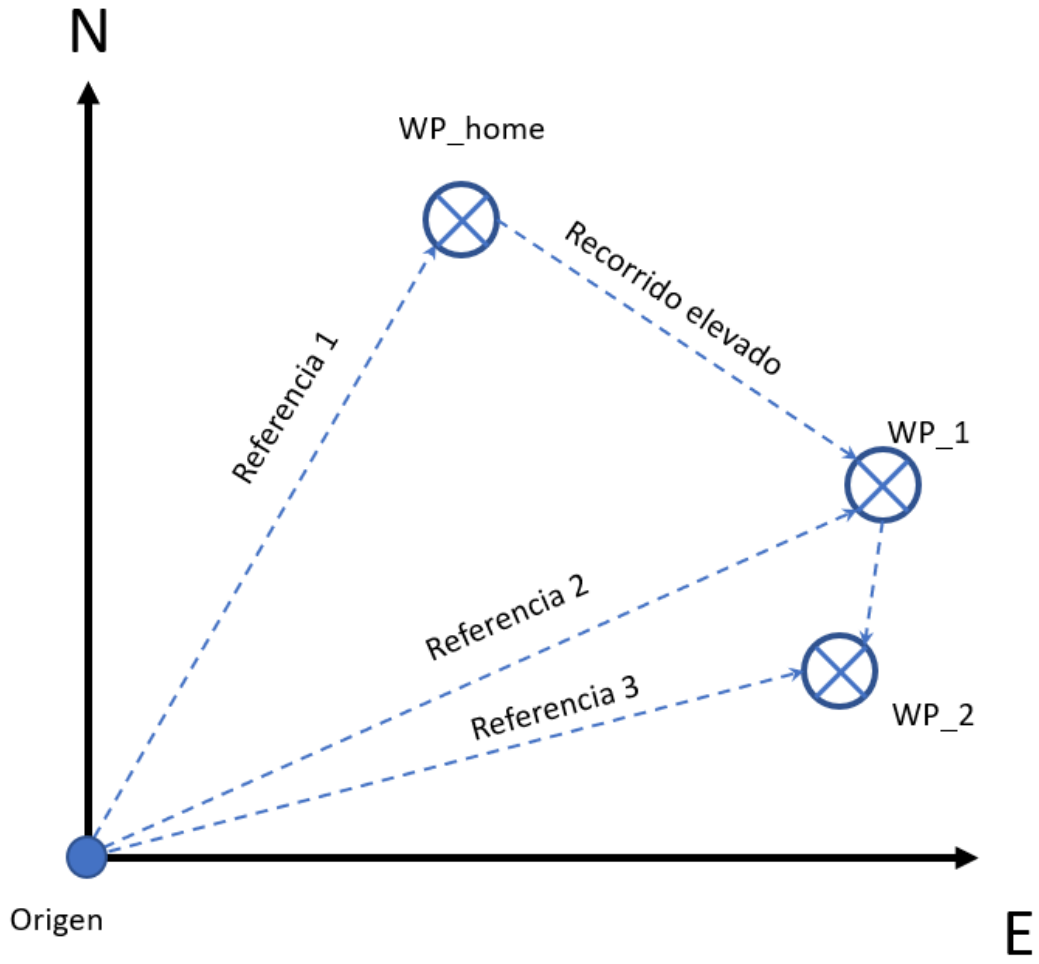


Figura 26. Esquema de funcionamiento de navegación autónoma sin límite

En la gestión de los waypoints es importante notar que el primero se considera como HOME de la misión. Así, en cualquier circunstancia que implique retorno a HOME, el UAV irá a este punto. La posición del resto de waypoints se calculará respecto a HOME. De esta forma, se realiza una nueva traslación para poder dar las referencias respecto al origen de coordenadas, que implica sumar la posición relativa de cada waypoint y el anterior a la posición de HOME.

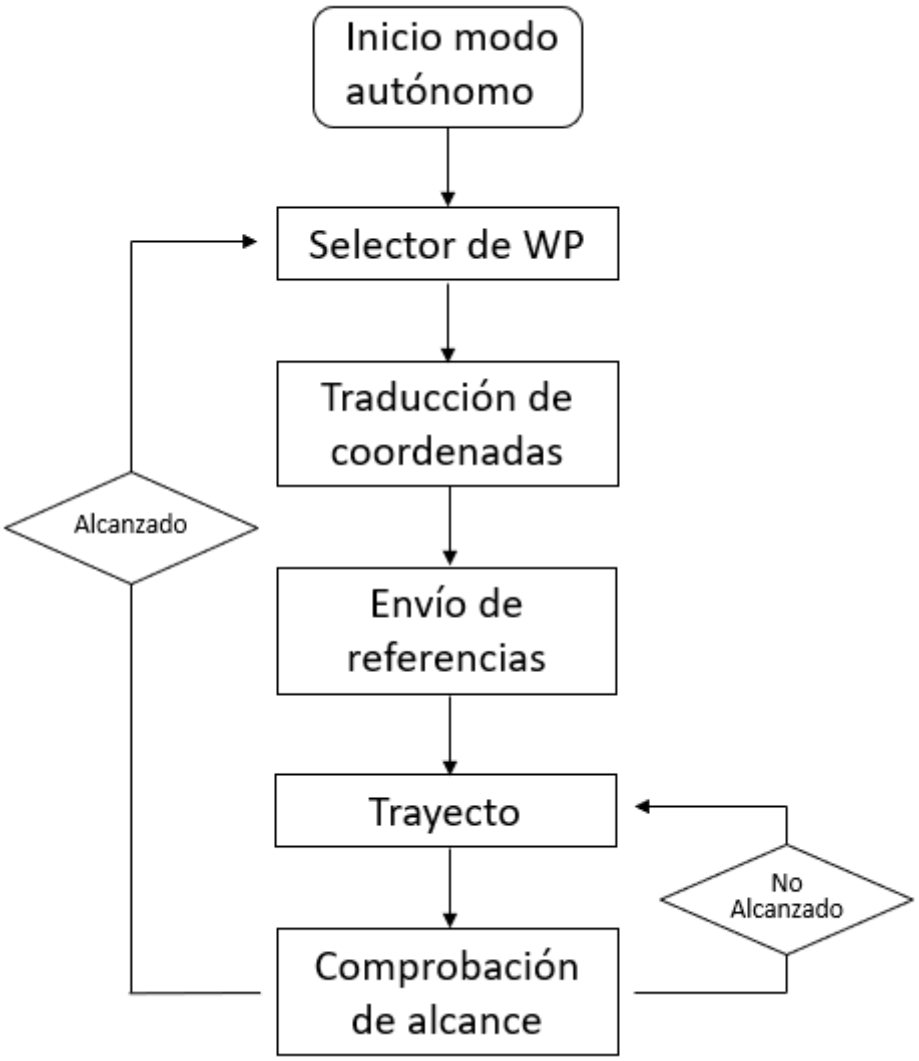


Figura 27. Diagrama de flujo del programa sin limitación

En la *Figura 29* se observa el diagrama de flujo de la programación de la gestión de waypoints. Todos los detalles relativos al código se encuentran en el apartado 4.3.

4.2. Creación de waypoints virtuales y envío de referencias

Como se ha explicado en el apartado anterior, se calculan las posiciones de los waypoints en el sistema de referencia local a medida que se van alcanzando. Inicialmente se prueba a dar como referencia directa la posición de cada waypoint, es decir, se introducen las coordenadas del waypoint destino directamente como referencia para el control.

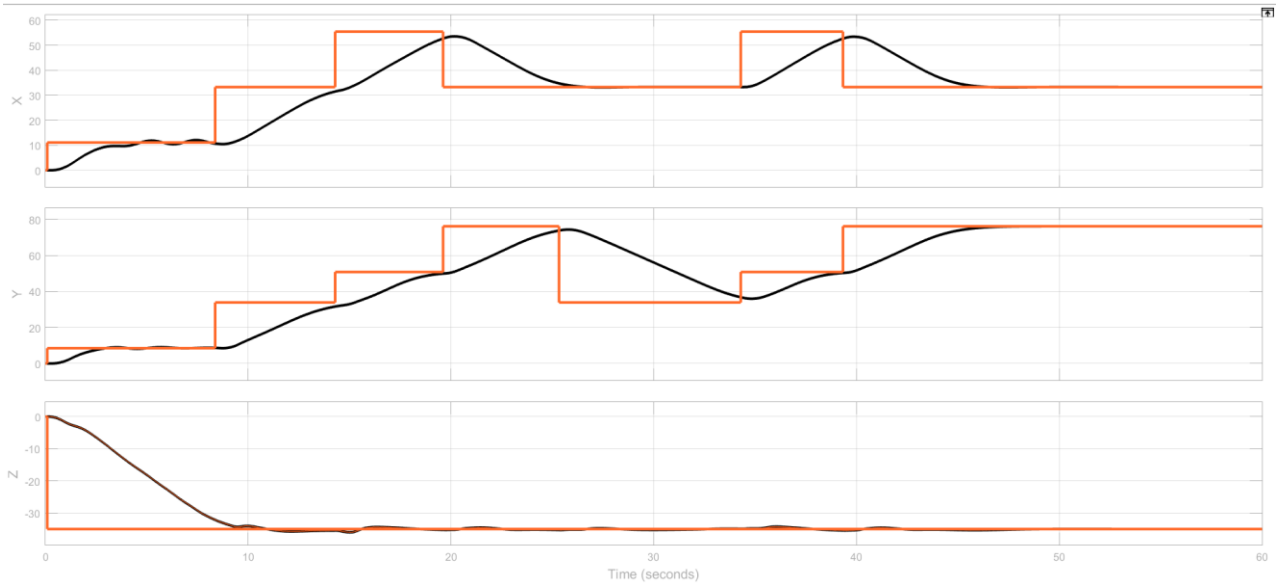


Figura 28. Misión de siete puntos según el primer desarrollo

En la *Figura 27* se observa una misión de siete waypoints. El primer gráfico hace referencia a la posición en X, el segundo a la posición en Y y el tercero a la posición en Z. En el eje Z la posición aparece en negativo porque determinamos que el eje apunta a la superficie terrestre. Podemos observar que en simulación el UAV es capaz de realizar la misión. Sin embargo, en la realidad referencias demasiado elevadas desembocarán en saturación del mando e inestabilidades.

Nos marcamos el objetivo de generar referencias menos exigentes, que acompañen al UAV en su recorrido, haciendo que el vuelo sea más estable y seguro. Para ello, se busca reducir la distancia a recorrer por el UAV en cada actualización de referencia. Así, pasa a ser necesario conocer la situación del UAV en cada momento. Conocemos su posición en coordenadas geográficas gracias al GPS incorporado, de forma que las transformamos a coordenadas locales. A continuación, calculamos la distancia entre el UAV y el

waypoint destino a través del vector que los une. Para acompañar al UAV, lo que haremos será dar las referencias sobre este vector en función de una distancia máxima.

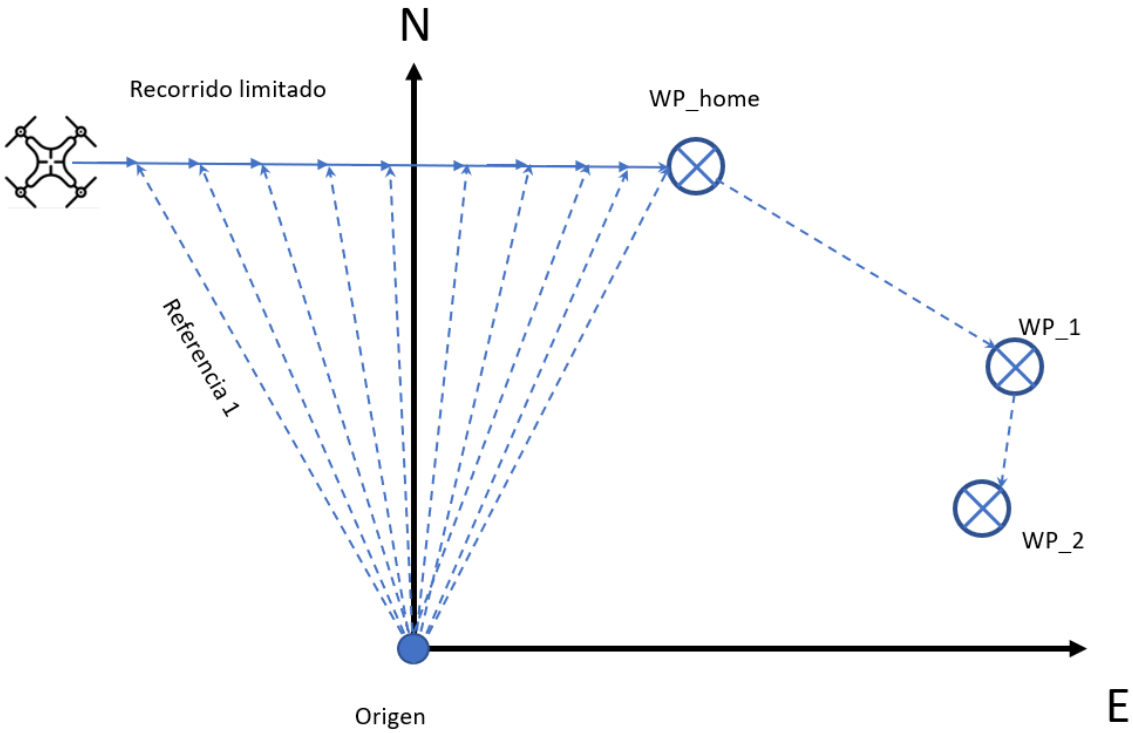


Figura 29. Esquema de funcionamiento con distancia limitada

Esta distancia se determinará con un parámetro de navegación editable en función de las necesidades del usuario y de las capacidades del UAV. A menor distancia máxima, más tiempo tardará el UAV en realizar el recorrido y más waypoints virtuales se crearán.

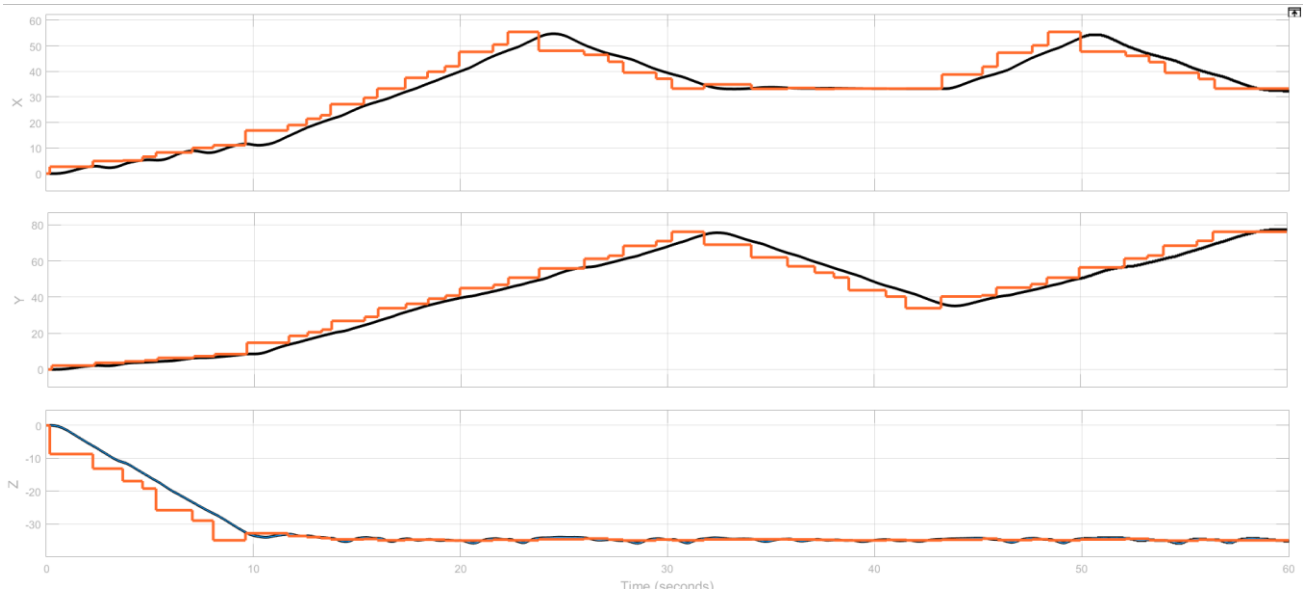


Figura 30. Misión de siete puntos con limitación de mando

En la *Figura 33* encontramos el nuevo diagrama de flujo de la programación.

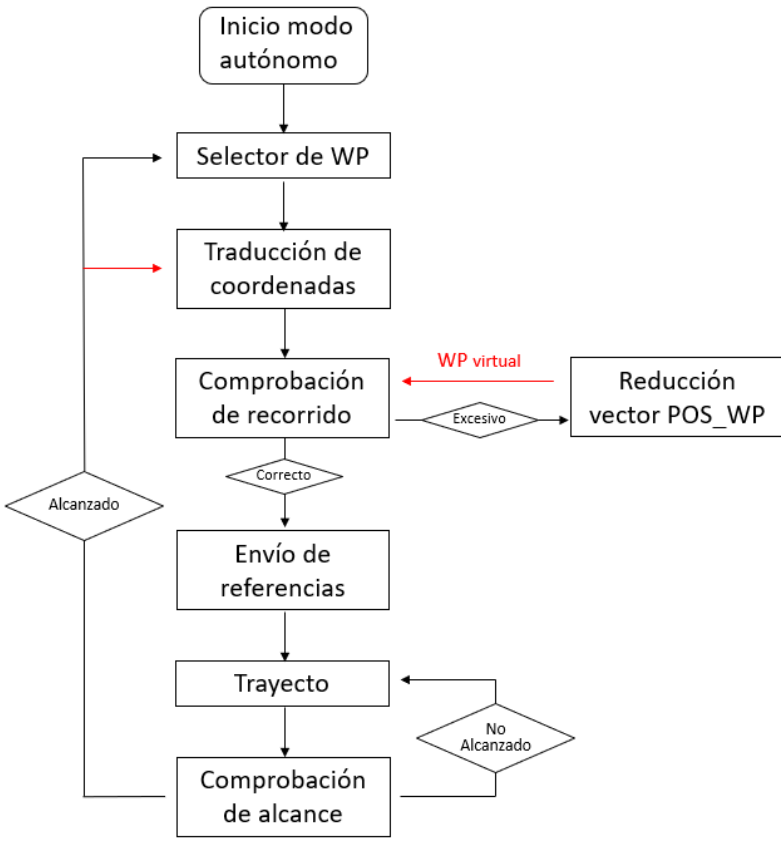


Figura 31. Diagrama de flujo del programa con limitación

4.3. Implementación y funcionamiento del modo autónomo

En este capítulo se explicará cómo y dónde se ha implantado el código de control autónomo. Además, se explicará detalladamente la programación del mismo.

La programación se realiza en dos bloques. Por un lado, dentro del bloque CONTROL, se acude al bloque COMMUNICATIONS, donde encontraremos los bloques que aparecen en la *Figura 32*.

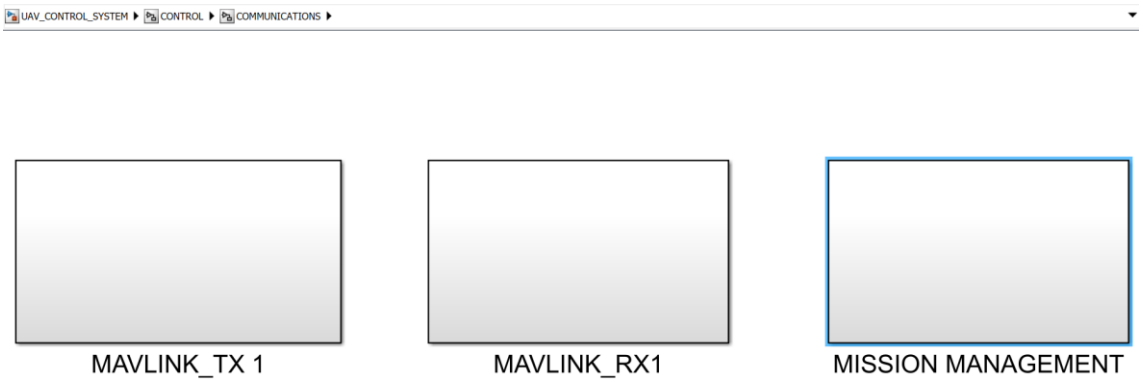


Figura 32. Bloque y ruta "MISSION MANAGEMENT"

Entre estos, en este proyecto se ha añadido el bloque MISSION MANAGEMENT. En este, se programa todo lo necesario para la funcionalidad de vuelo autónomo.

Es un bloque al que llegan ambos buses, tanto el MAVLINK como el CONTROL.

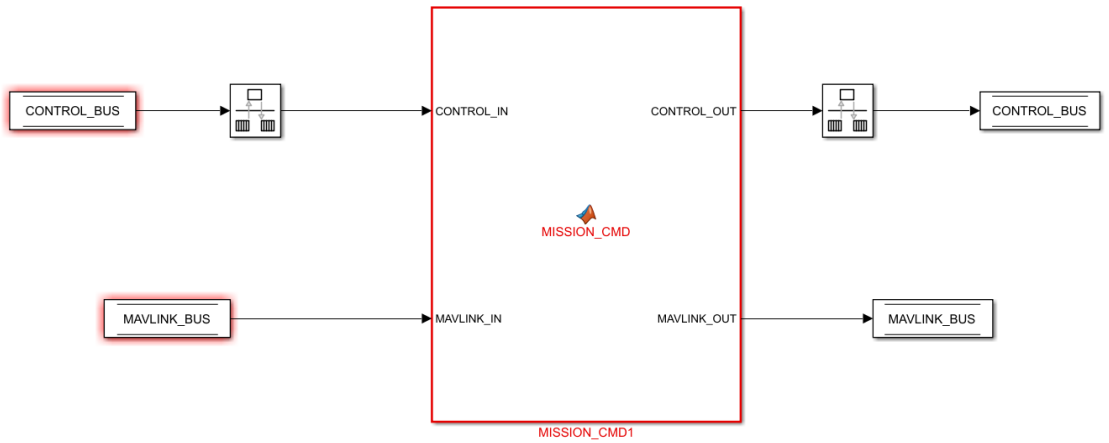


Figura 33. Diferentes tiempos de muestreo de buses

El primer segmento de código se encarga de la recepción de la misión, y está explicado en la sección 2.6.

Seguidamente se encuentra la gestión de la posición de waypoints en la matriz de la misión, cuyo nombre en el bus es MISSION_INFO.

```
% WP selector
if CONTROL_IN.INPUT.MISSION.ITEM_REACHED
    CONTROL_OUT.INPUT.MISSION.ITEM_REACHED = uint8(0);
    CONTROL_OUT.INPUT.MISSION.CURRENT_COUNT_J = uint16(WP);
    WP = WP+1;
end
if CONTROL_OUT.INPUT.MISSION.CURRENT_COUNT_J > CONTROL_IN.INPUT.MISSION.COUNT_J-1
    CONTROL_OUT.INPUT.MISSION.CURRENT_COUNT_J = CONTROL_IN.INPUT.MISSION.COUNT_J-1;
end
```

Figura 34. WP selector

En esta sección de código, titulada como WP selector, aparece la variable ITEM_REACHED, que se gestiona desde el bloque STATE_MACHINE, y cuyo valor se actualiza a '1' cuando el UAV entra en el rango de aceptación del waypoint destino. En cuanto se actualiza a '1', se actualiza la variable CURRENT_COUNT_J a través de la variable persistente WP, que después de la actualización incrementa su valor. Además, se actualiza la variable ITEM_REACHED a '0' para indicar que hay un nuevo destino. Las siguientes líneas limitan el valor de CURRENT_COUT_J al valor máximo de waypoint, de forma que cuando se alcance el último waypoint las coordenadas dadas pasen a ser constantes de ese waypoint, haciendo que el UAV espere en esa posición. De no realizarse esta operación, el sistema tomaría la siguiente línea de la matriz, que está constituida por ceros, y daría como coordenadas destino el punto de origen del sistema de referencia local.

A continuación, encontramos el algoritmo de traducción de coordenadas geográficas a coordenadas locales.

```
% LLA TO FLAT EARTH
% Relative WP to HOME_UAV
if CONTROL_IN.INPUT.MISSION.GO_HOME
    LATITUDE = HOME_LLA(1)-HOME_UAV(1); % deg
    LONGITUDE = HOME_LLA(2)-HOME_UAV(2); % deg
    ALTITUDE = HOME_LLA(3)-HOME_UAV(3); % m
    LATITUDE_H = LATITUDE;
    LONGITUDE_H = LONGITUDE;
    ALTITUDE_H = ALTITUDE;
else
    % Relative WP to HOME_WP
    LATITUDE_R = CONTROL_IN.INPUT.MISSION.MISSION_INFO(CONTROL_OUT.INPUT.MISSION.CURRENT_COUNT_J+1 ,9)-HOME_LLA(1); % deg
    LONGITUDE_R = CONTROL_IN.INPUT.MISSION.MISSION_INFO(CONTROL_OUT.INPUT.MISSION.CURRENT_COUNT_J+1 ,10)-HOME_LLA(2); % deg
    ALTITUDE_R = CONTROL_IN.INPUT.MISSION.MISSION_INFO(CONTROL_OUT.INPUT.MISSION.CURRENT_COUNT_J+1 ,11)-HOME_LLA(3); % m
    % Translation from relative WP to HOME_WP to HOME_UAV
    LATITUDE = LATITUDE_R + LATITUDE_H;
    LONGITUDE = LONGITUDE_R + LONGITUDE_H;
    ALTITUDE = ALTITUDE_R + ALTITUDE_H;
end
```

Figura 35. Creación de vectores relativos a coordenadas locales

En su primera ejecución, la variable GO_HOME está activada. Tras esta ejecución se actualiza a ‘0’. Sirve para guardar en variables persistentes el vector de posición de el waypoint cero o HOME de la misión en coordenadas locales. El resto de coordenadas se calcularán con respecto a la posición HOME, por lo que para realizar la traslación a coordenadas locales utilizaremos estas variables persistentes. En estas líneas puede observarse la traslación al origen de coordenadas locales que se comentó anteriormente. En la primera ejecución, en las variables LATITUDE, LONGITUDE y ALTITUDE, se resta la posición HOME_UAV, que es la posición de arranque del vehículo, de forma que el origen de coordenadas pasa a ser este. En el resto de ejecuciones, se calculan las variables respecto al waypoint cero, HOME de la misión, y a continuación se realiza la traslación al origen de coordenadas utilizando las variables persistentes.

Una vez realizada esta operación obtenemos coordenadas geográficas en un sistema de referencia local. Es necesario traducirlas a distancias en metros para convertirlas en referencias para el control del UAV.

```
% Latitude and longitude wrapping
CONTROL_OUT.INPUT.MISSION.EARTH_POS_TARGET_LLA = [1e7 1e7 1e3]'.*[...
    CONTROL_IN.INPUT.MISSION.MISSION_INFO(CONTROL_OUT.INPUT.MISSION.CURRENT_COUNT_J+1 ,9) ...
    CONTROL_IN.INPUT.MISSION.MISSION_INFO(CONTROL_OUT.INPUT.MISSION.CURRENT_COUNT_J+1 ,10) ...
    CONTROL_IN.INPUT.MISSION.MISSION_INFO(CONTROL_OUT.INPUT.MISSION.CURRENT_COUNT_J+1 ,11)]';
LAT_WRAPPED = (abs(LATITUDE)>180)*(mod(LATITUDE+180,360)-180) + ...
                    (abs(LATITUDE)<=180)*LATITUDE;
WRAP_FLAG = (abs(LAT_WRAPPED)>90);
LAT_WRAPPED = sign(LAT_WRAPPED)*(-abs(LAT_WRAPPED)+180)*(WRAP_FLAG==1) + ...
                    LAT_WRAPPED*(WRAP_FLAG==0);
LON_WRAPPED = LONGITUDE + 180*(WRAP_FLAG==1);
LON_WRAPPED = (abs(LON_WRAPPED)>180)*(mod(LON_WRAPPED+180,360)-180) + ...
                    (abs(LON_WRAPPED)<=180)*LON_WRAPPED;
% Conversion to rad
LAT_RAD = pi/180*LAT_WRAPPED;
LON_RAD = pi/180*LON_WRAPPED;
% Initial latitude in rad
LAT_HOME = pi/180*HOME_UAV(1);
% Find Radian/Distance
ECUATORIAN_RADIUS = (6378137 + 6356752)/2;
FLATNESS = 1/298.257223563;
EPS = sqrt(1-(1-FLATNESS)^2);
DENOM = 1-EPS^2*sin(LAT_HOME)^2;
dNorth = atan2(1,(1-EPS^2)*ECUATORIAN_RADIUS/DENOM^1.5);
dEast = atan2(1,ECUATORIAN_RADIUS*cos(LAT_HOME)/sqrt(DENOM));
% Psi: angle clockwise between the x-axis and north
PSi = 0;
s_Psi = sin(PSi);
c_Psi = cos(PSi);
POS_X =  LAT_RAD/dNorth*c_Psi + LON_RAD/dEast*s_Psi;
POS_Y = -LAT_RAD/dNorth*s_Psi + LON_RAD/dEast*c_Psi;
POS_Z = -ALTITUDE;
WP_XYZ = single([POS_X POS_Y POS_Z]'); % m (WP XYZ)
```

Figura 36. Traducción a coordenadas locales

Esto se hace a través de las operaciones matemáticas que se realizan en la *Figura 34*. Cabe comentar que las variables de la sección '%Find Radian/Distance' se obtienen del modelo terrestre WGS84. A la variable Psi se le da el valor '0' porque el sistema de referencia apunta al norte. A través de estas operaciones obtenemos el vector WP_XYZ, que recoge las coordenadas locales del waypoint destino. Se realizará la misma operación para obtener los vectores en coordenadas locales POS_WP y POS_HOME. El primero recoge la distancia entre la posición efectiva del UAV y el waypoint destino, y el segundo recoge la distancia entre la posición del UAV y el origen de coordenadas (HOME_UAV). Estos vectores pueden apreciarse en la *Figura 35*.

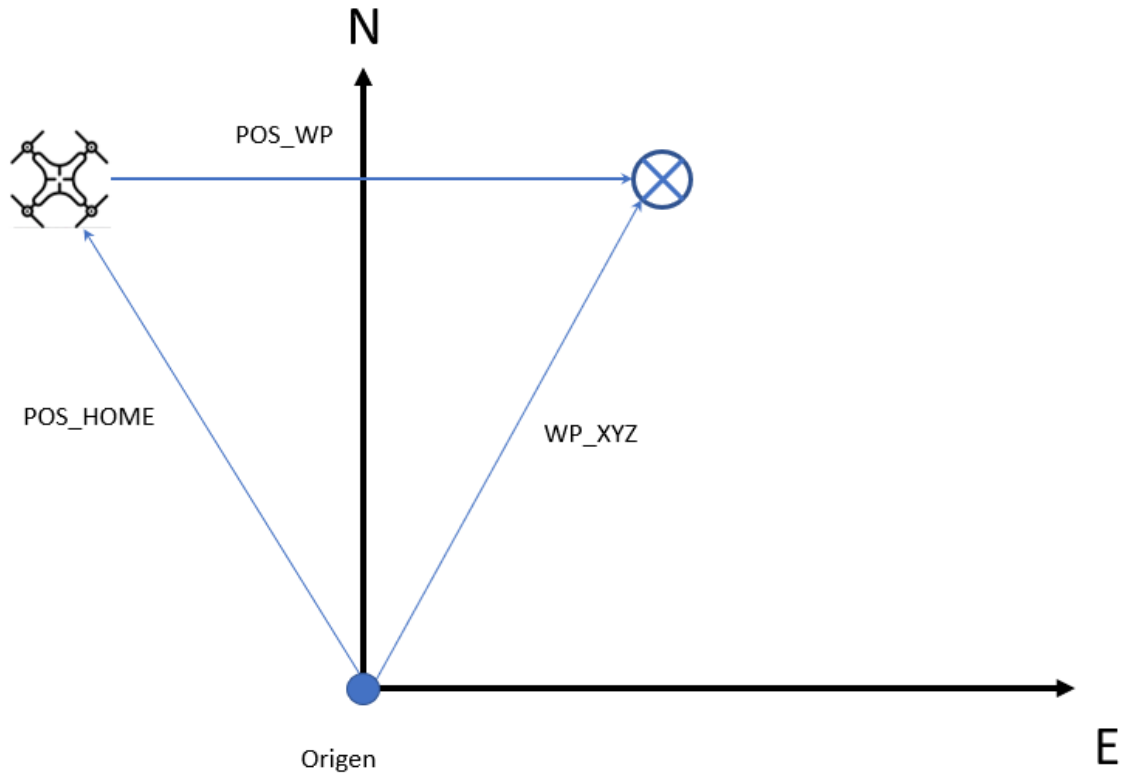


Figura 37. Esquema de vectores relevantes

Por último, se actualizan los buses y esta sección no vuelve a ejecutarse hasta que se actualiza la variable ITEM_REACHED a '1', de forma que los vectores calculados se guardan hasta que se alcanza el waypoint y se realizan os cálculos para el siguiente.

```
CONTROL_OUT.INPUT.MISSION.POS_HOME = single(POS_HOME);
CONTROL_OUT.INPUT.MISSION.POS_WP = single(POS_WP);
CONTROL_OUT.INPUT.MISSION.WP_XYZ = single(WP_XYZ);
CONTROL_OUT.INPUT.MISSION.GO_HOME = uint8(0);
```

Figura 38. Escritura de buses

En un primer desarrollo, las componentes del vector WP_XYZ se utilizaban como referencias para el control directamente. Sin embargo, se decide limitar las referencias para recudir el mando y aumentar la seguridad y estabilidad del vehículo. Así, se darán las referencias sobre el vector POS_WP, creando tantos waypoints virtuales como sea necesario, para reducir la distancia a recorrer en cada actualización por el UAV.

```
POS_WP = CONTROL_IN.INPUT.MISSION.POS_WP; % m
POS_HOME = CONTROL_IN.INPUT.MISSION.POS_HOME;
WP_XYZ = CONTROL_IN.INPUT.MISSION.WP_XYZ;
DIS = (POS_WP(1)^2+POS_WP(2)^2+POS_WP(3)^2)^0.5;
```

Figura 39. Variable DIS

En primer lugar, se cargan los vectores y se calcula el módulo del vector POS_WP. De esta forma, podremos conocer la distancia entre el UAV y su waypoint destino.

```
if DIS > DIS_LIMIT
  POS_WP = POS_WP.*0.5;
  CONTROL_OUT.INPUT.MISSION.POS_WP = POS_WP;
else
  if DIS > 0
  EARTH_POS_TARGET = POS_WP + POS_HOME;
  DIS = ((WP_XYZ(1)-EARTH_POS_TARGET(1))^2+(WP_XYZ(2)-EARTH_POS_TARGET(2))^2+...
      (WP_XYZ(3)-EARTH_POS_TARGET(3))^2)^0.5;
  if DIS > 3 && ~CONTROL_IN.INPUT.MISSION.PARTIAL_ITEM
      WP = WP-1;
      CONTROL_OUT.INPUT.MISSION.PARTIAL_ITEM = uint8(1);
      if WP < 0
          WP = 0;
      end
  end
```

Figura 40. Proceso de limitación de referencias

A continuación, se comprueba si esta distancia supera la máxima fijada por nosotros. El factor DIS_LIMIT depende de las necesidades del usuario y de las capacidades del UAV concreto, y se puede editar para obtener distintos márgenes de seguridad y distintos tiempos de ejecución de la misión.

En caso de superar la distancia límite marcada, se procederá a dividir el vector entre dos. La idea es que este algoritmo se ejecute el número de veces necesario para alcanzar la distancia aceptada, de forma que en cada ejecución el módulo del vector se divida entre dos. Así, por ejemplo, si la distancia límite fuese de 10 metros, y la distancia real fuese de 1200 metros, serían necesarias tan solo siete ejecuciones (reduciendo el vector POS_WP a un módulo de 9.375 metros), es decir, tardará 70 ms, un tiempo asumible incluso en distancias tan elevadas como 1200 metros.

Cuando el módulo del vector POS_WP cumple la distancia límite se hace una comprobación de correcto funcionamiento en el que el módulo sea positivo. En las

primeras ejecuciones y en algunos transitorios puede darse el caso de que esta condición no se cumpla, y de asumir como correctas estas referencias el control se volvería inestable. Una vez realizada la comprobación, se asigna a la variable de referencias del control la suma del POS_WP y POS_HOME, dando de esta forma referencias sobre el vector POS_WP.

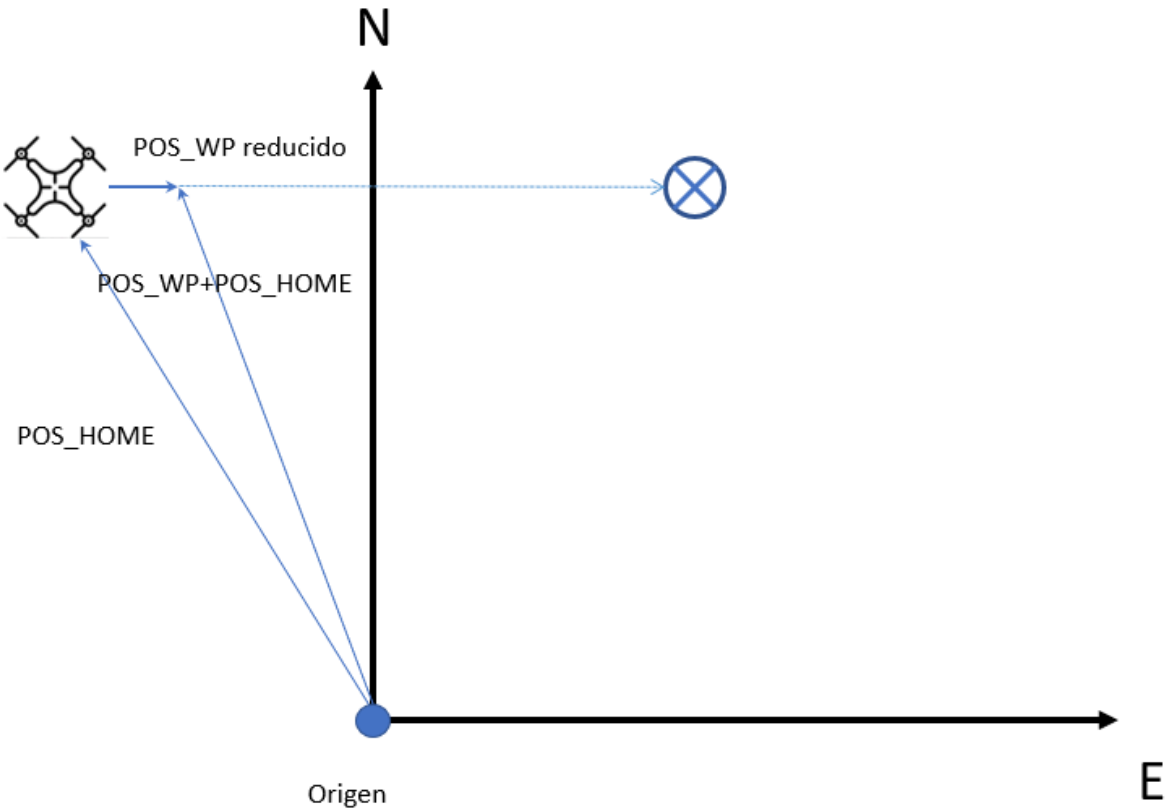


Figura 41. Esquema de referencias limitadas

En las siguientes líneas se comprueba si se trata de un waypoint virtual o si está alcanzando el waypoint real destino. Para ello, se calcula la distancia entre el waypoint real y las coordenadas referencia. Si esta distancia es menor a 3 metros, asumimos que el waypoint destino, sea o no virtual, se aproxima lo suficiente al real como para despreciarlo. Así, si es menor no se considerará un waypoint virtual sino el destino real. En caso de ser mayor, se activa la variable PARTIAL_ITEM, y se resta a la variable persistente WP uno. Así, al ejecutarse la primera parte del código, se seleccionará el mismo waypoint real de la matriz de misión pese a haberse activado la señal ITEM_REACHED.

Para finalizar el bloque MISSION MANAGEMENT, se actualiza el bus de referencias y se activa la variable NEW_TARGET, de forma que la gestión de la máquina de estados sepa que ha habido una actualización de referencias.

```
264 -        CONTROL_OUT.INPUT.EARTH_POS_TARGET = single(EARTH_POS_TARGET);
265 -        CONTROL_OUT.INPUT.MISSION.NEW_TARGET = uint8(1);
```

Figura 42. Envío de referencias

El resto de la gestión del modo autónomo se realiza en el bloque STATE MACHINE UPDATE, dentro del bloque CONTROL_UPDATE de CONTROL. En este bloque se actualizarán algunas variables de funcionamiento del UAV y se realizarán las gestiones para comprobar que se ha alcanzado el waypoint destino.

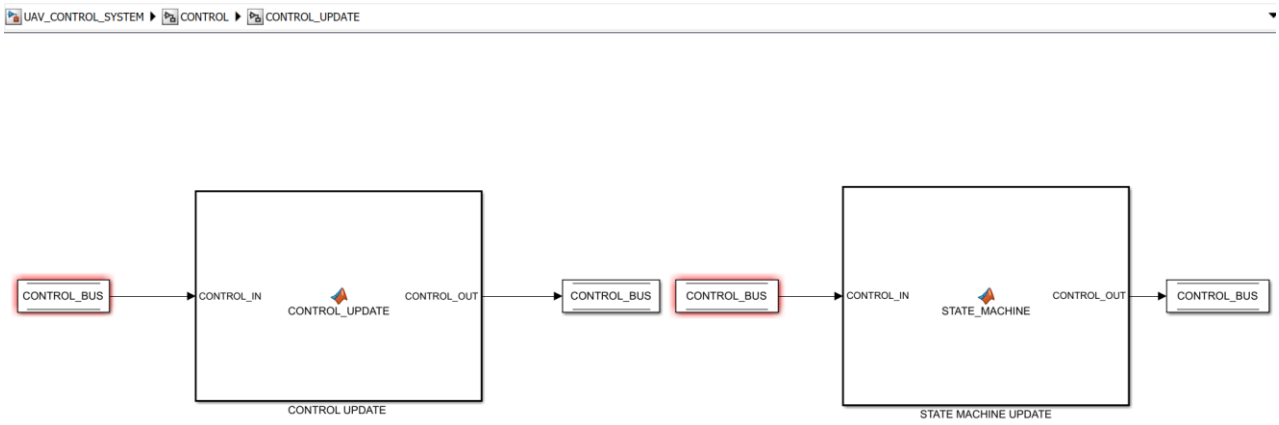


Figura 43. Imagen y ruta del bloque "CONTROL"

En esta máquina de estados creamos uno nuevo dedicado al vuelo autónomo. Solo se puede acceder a él una vez inicializados los sensores y habiendo despegado el UAV.

En este estado actualizamos las formas de funcionamiento del control (3D NAVIGATION), del estimador de Kalman (control en XYZ), de los motores (armados) y del modo de vuelo (AUTONOMOUS).

```
case 11 % AUTONOMOUS FLIGHT
    NEXT_STATUS = uint8(11);
    % FLIGHT MODE: AUTO
    FLIGHT_MODE = uint8(3);
    % CONTROL MODE: 3D
    CONTROL_MODE = uint8(4);
    % EKF MODE: POS_XYZ
    EKF_MODE = uint8(7);
    % MOTOR MODE: ARMED
    MOTOR_MODE = uint8(2);
```

Figura 44. Estado de vuelo autónomo

En la primera ejecución actualizará el valor GO_HOME a '1', y este valor pasará a ser '0' en cuanto se alcance la posición de HOME de la misión. A continuación, se extrae la información de las referencias y posición del UAV de los buses correspondientes y se calcula la variable ERROR.

```
if ~HOME_REACHED
    CONTROL_OUT.INPUT.MISSION.GO_HOME = uint8(1);
end
POS_XYZ = CONTROL_IN.INPUT.EARTH_POS;
EARTH_POS_TARGET = CONTROL_IN.INPUT.EARTH_POS_TARGET;
ERROR = sqrt((EARTH_POS_TARGET(1)-POS_XYZ(1))^2+(EARTH_POS_TARGET(2)-POS_XYZ(2))^2 ...
+(EARTH_POS_TARGET(3)-POS_XYZ(3))^2);
if SKIPER > 100
SKIPER = 6;
end
HOME_REACHED = 1;
NEW_TARGET = CONTROL_IN.INPUT.MISSION.NEW_TARGET;
if ERROR < PRECISION && SKIPER > 5 && NEW_TARGET && ~CONTROL_OUT.INPUT.MISSION.ITEM_REACHED
    CONTROL_OUT.INPUT.MISSION.ITEM_REACHED = uint8(1);
    HOME_REACHED = 1;
    SKIPER = 0;
    CONTROL_OUT.INPUT.MISSION.NEW_TARGET = uint8(0);
    CONTROL_OUT.INPUT.MISSION.PARTIAL_ITEM = uint8(0);
else
    SKIPER = SKIPER + 1;
end
```

Figura 45. Gestión en la máquina de estados

El ERROR calculado considera la distancia entre la posición del UAV y el waypoint destino. Una vez que este error sea menor a un rango de aceptación que establecemos con el parámetro persistente PRECISION, que depende de las necesidades del usuario, se actualiza la variable ITEM_REACHED a '1'. Para activar esta variable es necesario que

su valor sea '0' y que la variable NED_TARGET esté activa. Después de activar la variable ITEM_REACHED, desactiva NEW_TARGET y PARTIAL_ITEM.

El acceso y la salida de este estado depende de la controladora. Así, para acceder debe estar activo un interruptor (el correspondiente al canal 6), y para salir debe estar desactivado.

```
CH6 = CONTROL_IN.INPUT.RCRX.SCALED_CH(6);
if CH6 == -10000
   NEXT_STATUS = uint8(7);
end
```

Figura 46. Selección de estado

El canal puede editarse en el bloque de la controladora en función de las necesidades del usuario.

Cabe destacar que todas las variables que se emplean tanto en la transmisión de la misión como en la gestión del modo autónomo se alojan en el bus MISSION, perteneciente al bus INPUTS contenido en el bus CONTROL. Por lo tanto, para editar cualquier aspecto del vuelo autónomo debe accederse a dicho bus.

Capítulo 5. SIMULACIÓN

En este capítulo se describirá el proceso a simular y se expondrá el método de simulación. Es necesario para explicar las funcionalidades de simulación del sistema, qué posibilidades otorga y cómo se ha hecho. Se expondrán a su vez los distintos elementos que intervienen en la simulación.

5.1. Propósito y sentido de la simulación

Antes de realizar las pruebas sobre el UAV, una vez implantado el sistema de navegación autónoma, es necesario comprobar su funcionamiento en simulación. Para que sea una simulación descriptiva del proceso real, se intentan generar las condiciones necesarias para aproximarse lo máximo posible a la realidad.

Para ello se simulan condiciones meteorológicas como viento racheado. Además, se simula ruido típico en los sensores. Se colocan visualizadores en señales internas de interés, como el EULER_RATE y el EULER_ANG, la velocidad de giro y el valor de los ángulos de Euler respectivamente. Además, habrá visualizadores de la velocidad y la posición terrestres en coordenadas locales.

Se simulará el proceso completo, la creación de la ruta, la lectura por la base terrestre, el envío por puerto serie, la puesta en marcha del UAV, el paso a modo autónomo y la ejecución de la ruta. Se contará con un entorno gráfico de simulación, FlightGear, para comprobar el funcionamiento de una forma intuitiva. Además, se comprobarán las señales internas ya mencionadas y el correcto envío de la misión.

Estas capacidades de simulación son claves para comprobar los distintos desarrollos que se van realizando y también desarrollos futuros. Es útil invertir tiempo en este sistema para poder realizar simulaciones fiables que faciliten los procesos de desarrollos, dando un primer veredicto de validez sobre los mismos y permitiendo que la implantación de los modelos en el drone sea más seguro y controlada. Probar los desarrollos directamente en el UAV presenta muchas complicaciones, tanto por dificultad y tiempos de implantación como por posibles daños en el hardware. Además, estas pruebas pueden no ser concluyentes, mientras que las simulaciones nos permiten probar los sistemas en multitud de escenarios.

5.2. Preparación de la simulación

En primer lugar, hay que tener en cuenta los elementos que participan de la simulación. Utilizaremos un ordenador para las bases terrestres, otro para la simulación del UAV y FlightGear y una controladora para el arranque y el despegue del UAV.

El primer UAV tendrá instalado MISSION PLANNER para generar la ruta. Una vez generada, se leerá en la base terrestre programada en Matlab y se procederá al envío de la misma. Los ordenadores estarán conectados por puerta UART. Para acceder a esta puerta se utilizarán puertos serie y traductores de USB a UART.

El segundo ordenador tendrá que recibir la misión y almacenarla en su memoria. A continuación, simulará el arranque del UAV, el despegue y sus dinámicas. Enviará esta información al entorno gráfico de FlightGear para realizar una simulación intuitiva.

Todos los elementos referentes a la programación de la simulación se encuentran en el bloque "SIMULATION". Dentro del mismo encontramos dos bloques principales para la simulación, el modelo del UAV y el modelo de los sensores. Encontramos a su vez bloques específicos para nuestra simulación. Uno de ellos es el bloque "FLIGHTGEAR". Es el encargado de enviar la información al programa.

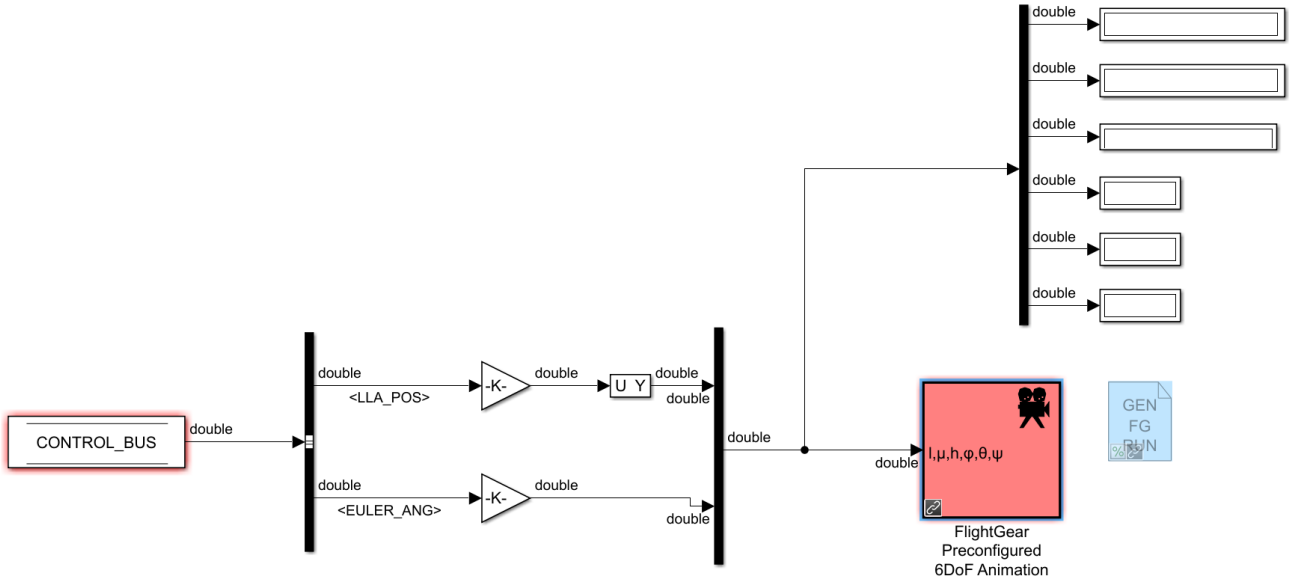


Figura 47. Bloque FlightGear

En la *Figura 45* puede observarse que este bloque envía únicamente las señales "LLA_POS" y "EULER_ANG". La primera da las coordenadas geográficas del UAV y la segunda los ángulos de Euler del mismo. Para la simulación, decidimos que el drone ejecutará una ruta en torno al puente de San Francisco. Su punto de inicio será muy cercano al primer waypoint (HOME). Este punto de inicio se determina en el bus de configuración de elementos de hardware para simulación. Para poder realizar esta trayectoria, nos tenemos que descargar los escenarios correspondientes para FlightGear desde una base de datos que contiene un alto porcentaje de escenarios terrestres para el simulador.

Otro bloque, "RC_TRANSMITTER", contiene la programación necesaria para poder utilizar la controladora. Este bloque incluye el tratamiento de los canales y definición. Así, en los distintos visualizadores puede verse el valor de cada canal. Los canales se asignan a los interruptores y botones desde la propia controladora. En el bloque "RC_CHANNEL_PROCESSING" se procesan las señales y se modifican según la función del canal. Para nuestra simulación, asignamos al canal seis la función de activar o desactivar el vuelo autónomo.

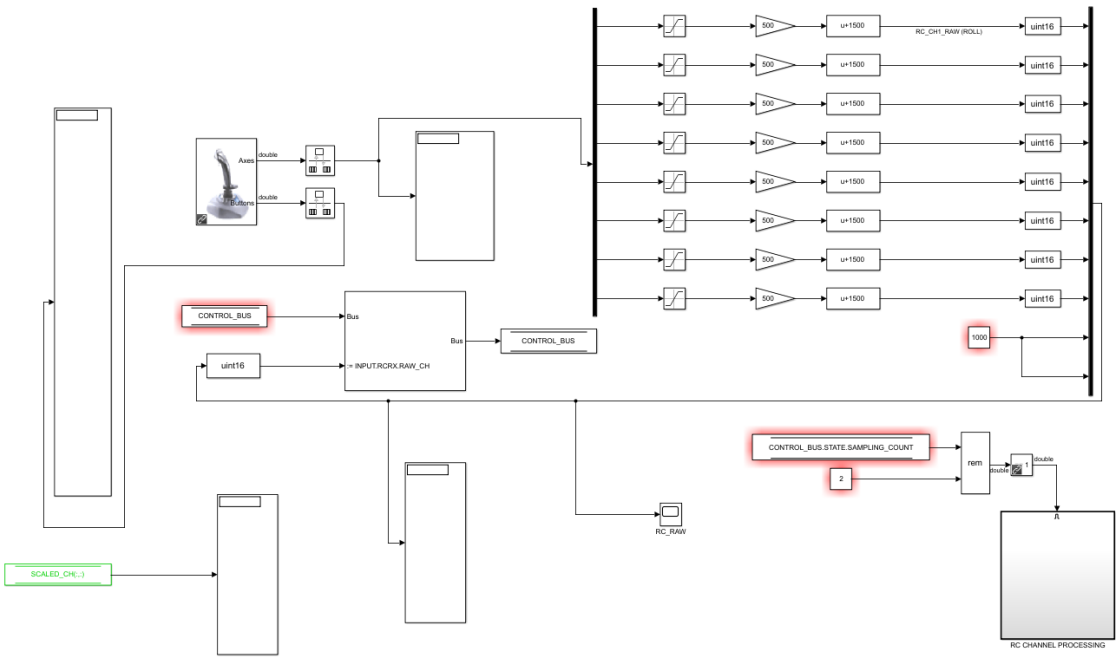


Figura 48. Bloque de la controladora

Referentes a la transmisión de la misión encontramos dos bloques, "HARDWARE" y "MAVLINK". El primero contiene la inicialización del puerto serie para comunicación

UART. Hay además bloques para puertos UDP y TCP/IP. El segundo contiene visualizadores de los buffers de envío y recepción para controlar en todo momento la información que se está transmitiendo.

Para controlar diversas variables internas encontramos el bloque “SCOPES”. Contiene todos los visualizadores necesarios para monitorizar el estado del UAV.

Capítulo 6. RESULTADOS Y CONCLUSIONES

En este capítulo se expondrán los resultados y conclusiones de los desarrollos

6.1. Resultados de las simulaciones

Durante la simulación, nos percatamos de diversos problemas del modelo. En primer lugar, nos damos cuenta de que es incapaz de seguir el ritmo real, es decir, por alguna razón la simulación va ralentizándose, lo que indica falta de capacidad de computación o algún error en la inicialización de los puertos. Vamos comprobando cada bloque por separado hasta que vemos que el error está relacionado con la transmisión MAVLink. Gracias a esto se decide modificar el funcionamiento del buffer, pasando de un buffer estático a uno dinámico y cambiando su período de muestreo, que será el doble que el del resto del modelo, para evitar la pérdida de información.

Una vez solucionado este problema, se realiza el proceso completo de simulación con resultados satisfactorios. Se consigue enviar correctamente la misión, recibirla correctamente, ejecutar el arranque y el despegue con la controladora, pasar a modo autónomo y que el UAV ejecute la ruta indicada. Se monitoriza primeramente el envío de la misión y a continuación la representación gráfica en FlightGear y los visualizadores de señales internas.

Una vez completado todo el proceso, se realizan algunas simulaciones rápidas para comprobar la conveniencia de algunas variables de programación y las capacidades del UAV.

Como se comenta en el capítulo 3.3. existen algunas variables de programación definibles por el usuario y que dependen de factores como las necesidades del usuario o las capacidades del UAV. Así, en la siguiente tabla encontramos una comparativa de ejecuciones de una misma misión estándar de siete puntos en función de estos parámetros:

DIS_LIMIT [m]	PRECISION [m]	Tiempo ejecución [s]	Sobrepaso [%]	Máx. ángulo Euler [º]
6	1	110	4	23
12	1	75	5	41
6	3	100	5	32
12	3	68	6	44

En la tabla se puede observar cómo influye la modificación de los parámetros en la ejecución de la misión. Los factores más relevantes son el tiempo de ejecución y el máximo ángulo de Euler. A mayor permisividad tanto en la máxima distancia entre referencias como en la aceptación de waypoint alcanzado, menores tiempos de ejecución, pero mayor inestabilidad. El máximo de esta situación se alcanza cuando no se marcan límites en la distancia máxima a recorrer entre referencia. El sobrepaso se mantiene limitado y no constituye un límite de funcionamiento. En las siguientes figuras se pueden observar las oscilaciones en los ángulos de Euler.

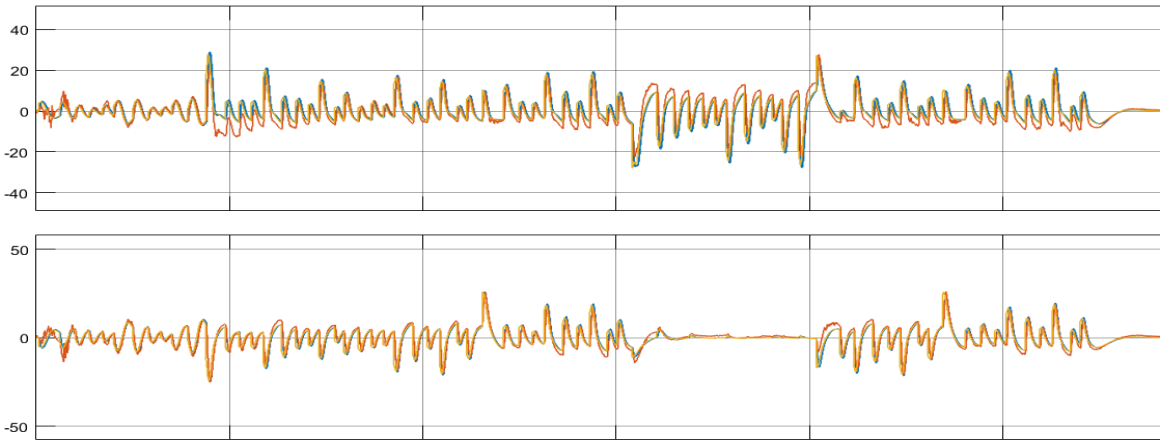


Figura 49. Ángulos de Euler en máxima limitación

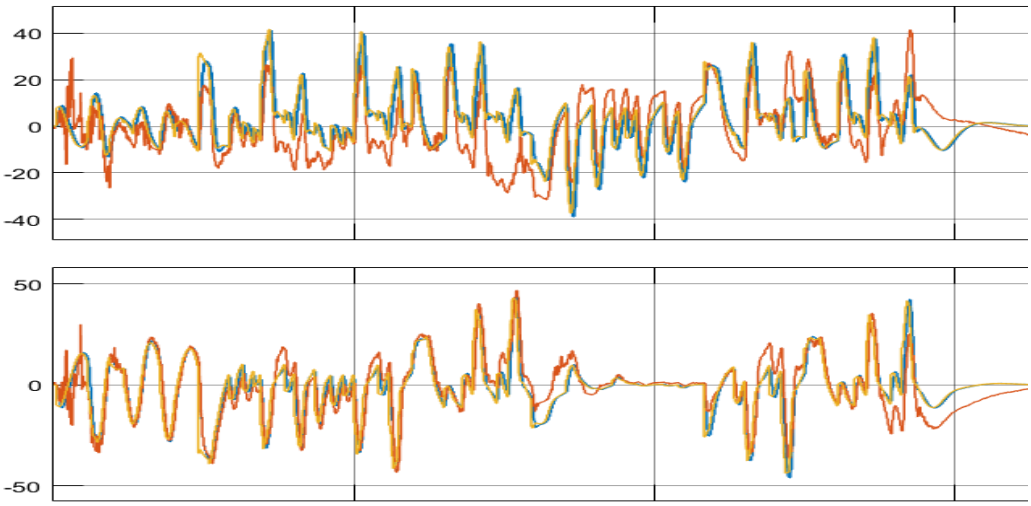


Figura 50. Ángulos de Euler en mínima limitación

Cabe destacar la diferencia apreciable a simple vista de los tiempos de ejecución. Será el usuario quien tenga que determinar el valor de estas variables de programación. Por defecto tienden a las situaciones más seguras.

6.2. Resultados de hardware

Se consiguen programar correctamente las comunicaciones con los del PFXmini. Se comprueba que se obtienen medidas razonables.
El barómetro es el que ofrece mejores resultados. Las medidas coinciden con las indicadas por distintas fuentes.
El magnetómetro ofrece resultados razonables en dos ejes, pero el tercero se mantiene constante. En futuros desarrollos habrá que analizar a qué se debe este problema y realizar las calibraciones propias de este tipo de sensores.
No se consigue comprobar el GPS debido a que faltan conexiones de hardware, pero se dejan preparadas las comunicaciones para su implementación y test.

Capítulo 7. FUTUROS DESARROLLOS

En este capítulo se expondrán futuros desarrollos referidos al sistema de navegación observados durante este proyecto.

Funciones en la generación de trayectorias. En el apartado 3.1. se explica el formato de la misión y qué factores influyen en ella. Uno de ellos es el comando de misión, que en este proyecto está fijado en el valor "16", es decir, ir al waypoint directamente. El protocolo MAVLink permite cambiar este valor para añadir matices en la trayectoria. Así, podemos dar órdenes de trazar trayectorias suaves que integran toda la matriz de waypoints, de forma que el UAV no realice giros bruscos, sino que genere curvas para realizar la ruta.

Comprobación de compatibilidad con bases terrestres. Hay una gran oferta de bases terrestres en Internet, por lo que sería útil comprobar la compatibilidad del sistema con las más importantes para que sea lo más robusto posible. Además, se pueden optimizar las comunicaciones y el sistema de transmisión de misión.

Generación dinámica de rutas. En ocasiones puede ser útil ir generando la ruta desde la base terrestre a medida que el UAV la va ejecutando, de forma que esta no se envía completa en el período de carga, sino que va añadiendo waypoints durante la navegación.

Posibilidades de sensorización visual. Actualmente un campo que está desarrollándose y que parece que va a ser clave para la presencia de los UAV en la industria es el de los sistemas de navegación visuales. Añadir estas capacidades al UAV sería muy interesante y lo haría mucho más polivalente.

Implantación y pruebas de hardware. Siendo el futuro desarrollo más importante, incluye implantar el sistema en el UAV y probar cada una de sus partes. Esto nos permitirá conocer la fiabilidad de las simulaciones, el desempeño de los sensores y el funcionamiento del sistema de navegación.

Capítulo 8. BIBLIOGRAFÍA

[1] «Qué es un UAV,» RCTecnic, 2 Diciembre 2017. [En línea]. Available: https://www.rctecnic.com/blog/86_que-es-uav. [Último acceso: 8 Julio 2018].

[2] Á. Celorio, «El uso de drones aumenta en el sector audiovisual,» Cadena SER - Economía, 5 Agosto 2017. [En línea]. Available: http://cadenaser.com/ser/2017/08/04/economia/1501859957_363739.html. [Último acceso: 21 Julio 2018].

[3] A. Insights, «Drones para minería: Usos y aplicaciones,» Aerial Insights, 4 Marzo 2018. [En línea]. Available: http://www.aerial-insights.co/blog/drones-en-canteras-y-mineria/. [Último acceso: 28 Mayo 2018].

[4] C. C. Russo, H. D. Ramón y S. Serafino, «SEDICI Repositorio Institucional de la UNLP,» 2018. [En línea]. Available: http://sedici.unlp.edu.ar/handle/10915/68308. [Último acceso: 3 marzo 2018].

[5] J. M. Torrego, «Principales agentes del sector de los drones,» El referente, 7 Junio 2018. [En línea]. Available: http://www.elreferente.es/Rob%C3%B3tica/drones-espana-fabricantes-eventos-formacion-29521. [Último acceso: 2 Julio 2018].

[6] A. Vilches, «Uso de drones en el sector industrial,» Web Pilotando, Junio 2017. [En línea]. Available: https://www.pilotando.es/drones-para-sector-industrial/. [Último acceso: 3 Marzo 2018].

[7] E. Miras, «Drones: la gran tecnología hacia el horizonte,» El mundo, 17 Mayo 2017. [En línea]. Available: https://www.abc.es/summum/living/tecnologia/abci-drones-gran-tecnologia-hacia-horizonte-201705171327_noticia.html. [Último acceso: 12 Abril 2018].

[8] J. E. Ortega Pacheco y A. Simancas Mateus, «Sistema autónomo de navegación de vehículos RC con procesamiento digital de imágenes,» Universidad del Norte, 18 Noviembre 2011. [En línea]. Available: http://manglar.uninorte.edu.co/handle/10584/7798. [Último acceso: 23 Marzo 2018].

[9] S. De La Parra y J. Ángel, «Sistema de navegación para aeronaves no tripuladas de fácil realización,» Low cost navigatiom system for UAVs, 6 Septiembre 2005. [En línea]. Available: https://www.sciencedirect.com/science/article/pii/S1270963805000829. [Último acceso: 1 Abril 2018].

[10] M. H. Murillo, «Diseño de sistemas de control y navegación para vehículos aéreos no tripulados mediante simulación virtua,» Universidad Nacional del Litoral, 26 Febrero 2012. [En línea]. Available: http://web10.unl.edu.ar:8080/tesis/handle/11185/679. [Último acceso: 18 Abril 2018].

[11] J. D. Guallichico Loachamín, «Diseño e implementación de un sistema de navegación inercial tipo strapdown para estimar la posición de un robot móvil, aplicable a un prototipo de autopiloto de un UAV,» Respositorio Digital EPN, 5 Marzo 2013. [En línea]. Available: http://bibdigital.epn.edu.ec/handle/15000/5912. [Último acceso: 22 Abril 2018].

[12] «Precision Micro Barometer Module MS5611,» Amsys Info, 2013. [En línea]. Available: https://www.amsys.info/products/ms5611.htm. [Último acceso: 2 Julio 2018].

[13] L. Llamas, «Usar arduino con los IMU de 9DOF MPU 9150 MPU 9250,» 26 Septiembre 2016. [En línea]. Available: https://www.luisllamas.es/usar-arduino-con-los-imu-de-9dof-mpu-9150-y-mpu-9250/. [Último acceso: 13 Junio 2018].

[14] «Módulo gps ublox M8N,» RC-Innovations, Junio 2017. [En línea]. Available: https://rc-innovations.es/modulo-gps-ublox-m8n-CC3D. [Último acceso: 20 Junio 2018].

[15] T. Berteska, «Photogrammetric manning based on UAV,» Taylor and Francis Online, 12 Diciembre 2013. [En línea]. Available: https://www.tandfonline.com/doi/abs/10.3846/20296991.2013.859781. [Último acceso: 25 Mayo 2018].

[16] «MAVLINK Mission Command Messages,» ArduPilot Dev Team, 2017. [En línea]. Available: http://ardupilot.org/planner/docs/common-mavlink-mission-command-messages-mav_cmd.html. [Último acceso: 5 Junio 2018].

ANEXO A – ENVÍO DE MISIÓN

Código referente al envío de la misión desde la base de tierra. PC_CONTROL_STATION --> COMMUNICATIONS --> MISSION MANAGEMENT

```
function [CONTROL_OUT,MAVLINK_OUT] =
MISSION_CMD(CONTROL_IN,MAVLINK_IN)
CONTROL_OUT = CONTROL_IN;
MAVLINK_OUT = MAVLINK_IN;

persistent INICIALIZATION  COUNTER RACTUAL RLAST WACTUAL
WLAST
if isempty(INICIALIZATION)
    RACTUAL=0;
    WACTUAL=0;
    INICIALIZATION=1;
    COUNTER=0;
end

RLAST=RACTUAL;
if(CONTROL_IN.INPUT.MISSION.READ_J == true)
    RACTUAL=1;
else
    RACTUAL=0;
end

WLAST=WACTUAL;
if(CONTROL_IN.INPUT.MISSION.SEND_J == true)
    WACTUAL=1;
else
    WACTUAL=0;
end

if(RACTUAL > RLAST)
%     DATA=get_WP;
%     WP=length(DATA{1});
%     LONGITUDE=str2num(cell2mat(DATA{9}));
%     LATITUDE=str2num(cell2mat(DATA{10}));
%     ALTITUDE=str2num(cell2mat(DATA{11}));
        CONTROL_OUT.INPUT.MISSION.MISSION_INFO(2,9) =
37.808784;
        CONTROL_OUT.INPUT.MISSION.MISSION_INFO(2,10) = -
122.476959;
        CONTROL_OUT.INPUT.MISSION.MISSION_INFO(2,11) = 85;
```

```
        CONTROL_OUT.INPUT.MISSION.MISSION_INFO(3,9) =
37.813259;
        CONTROL_OUT.INPUT.MISSION.MISSION_INFO(3,10) = -
122.476788;
        CONTROL_OUT.INPUT.MISSION.MISSION_INFO(3,11) = 85;
        CONTROL_OUT.INPUT.MISSION.MISSION_INFO(4,9) =
37.818548;
        CONTROL_OUT.INPUT.MISSION.MISSION_INFO(4,10) = -
122.477474;
        CONTROL_OUT.INPUT.MISSION.MISSION_INFO(4,11) = 85;
        CONTROL_OUT.INPUT.MISSION.MISSION_INFO(5,9) =
37.823836;
        CONTROL_OUT.INPUT.MISSION.MISSION_INFO(5,10) = -
122.478161;
        CONTROL_OUT.INPUT.MISSION.MISSION_INFO(5,11) = 85;
        CONTROL_OUT.INPUT.MISSION.MISSION_INFO(6,9) =
37.828107;
        CONTROL_OUT.INPUT.MISSION.MISSION_INFO(6,10) = -
122.478161;
        CONTROL_OUT.INPUT.MISSION.MISSION_INFO(6,11) = 100;
        CONTROL_OUT.INPUT.MISSION.MISSION_INFO(7,9) =
37.828243;
        CONTROL_OUT.INPUT.MISSION.MISSION_INFO(7,10) = -
122.480478;
        CONTROL_OUT.INPUT.MISSION.MISSION_INFO(7,11) = 100;
        CONTROL_OUT.INPUT.MISSION.MISSION_INFO(8,9) =
37.826667;
        CONTROL_OUT.INPUT.MISSION.MISSION_INFO(8,10) = -
122.480478;
        CONTROL_OUT.INPUT.MISSION.MISSION_INFO(8,11) = 100;
        CONTROL_OUT.INPUT.MISSION.COUNT_J = uint16(7);
        CONTROL_OUT.INPUT.MISSION.MISSION_REQUEST_SEQ_J =
uint16(1);
end

if(WACTUAL > WLAST)
    CONTROL_OUT.INPUT.MISSION.STATE=uint8(1);
end

switch CONTROL_IN.INPUT.MISSION.STATE
    case 1 % START
        MAVLINK_OUT.DIVISOR(45)=2;
        MAVLINK_OUT.OFFSET(45)=1;
        CONTROL_OUT.INPUT.MISSION.STATE=uint8(2);
    case 2 %WAIT 160ms
        COUNTER=COUNTER+1;
        CONTROL_OUT.INPUT.MISSION.STATE=uint8(2);
        if(COUNTER>=2)
```

```
            CONTROL_OUT.INPUT.MISSION.STATE=uint8(3);
            COUNTER=0;
        end
    case 3 % START SENDING
        MAVLINK_OUT.DIVISOR(45)=uint8(100);
        MAVLINK_OUT.OFFSET(45)=0;
        MAVLINK_OUT.DIVISOR(40)=1;
        MAVLINK_OUT.OFFSET(40)=0;
        CONTROL_OUT.INPUT.MISSION.STATE=uint8(4);
    case 4 %WAIT TO SEND EVERY WP
        CONTROL_OUT.INPUT.MISSION.STATE=uint8(4);
       %WP=WP+1;
       %MAVLINK_OUT.MISSION_REQUEST_SEQ_J = uint16(WP);
        if (CONTROL_IN.INPUT.MISSION.MISSION_ACK_J)
            CONTROL_OUT.INPUT.MISSION.STATE=uint8(5);
        end
    case 5 %ACK
        MAVLINK_OUT.DIVISOR(40)=uint8(100);
        MAVLINK_OUT.OFFSET(40)=0;

end
```

ANEXO B - RECEPCIÓN DE MISIÓN

Código referente a la recepción de la misión por el UAV.

UAV_CONTROL_SYSTEM-->COMMUNICATIONS--> MISSION MANAGEMENT

```
function [CONTROL_OUT,MAVLINK_OUT] =
MISSION_CMD(CONTROL_IN,MAVLINK_IN)

CONTROL_OUT = CONTROL_IN;
MAVLINK_OUT = MAVLINK_IN;

persistent INITIALIZATION SEQUENCE COUNTER WP SKIPER
LONGITUDE_H...
            LATITUDE_H ALTITUDE_H
if isempty(INITIALIZATION)
    SEQUENCE = 0;
    INITIALIZATION = 1;
    COUNTER = 0;
    SKIPER = 0;
    WP = 1;
    LONGITUDE_H = 0;
    LATITUDE_H = 0;
    ALTITUDE_H = 0;
end

%% LOAD MISSION
% This function allows the mission itinerary to be loaded into
the UAV
if(CONTROL_IN.INPUT.MISSION.MSG_J == 44) % Receive message #44
to initiate transmission
    CONTROL_OUT.INPUT.MISSION.LOAD_STATE = uint8(1);
end

switch CONTROL_IN.INPUT.MISSION.LOAD_STATE % Start loading
    case 1 % START: Sending message #40 (request mission item)
        MAVLINK_OUT.DIVISOR(41) = 8;
        MAVLINK_OUT.OFFSET(41) = 1;
        CONTROL_OUT.INPUT.MISSION.LOAD_STATE = uint8(2);
    case 2 % REQUEST: Determining WP to be demanded
        COUNTER=0;
        CONTROL_OUT.INPUT.MISSION.MISSION_SEQ =
uint8(SEQUENCE+1);
        CONTROL_OUT.INPUT.MISSION.LOAD_STATE = uint8(3);
    case 3 %WAIT A PERIOD: To make sure the base receives the
message
        CONTROL_OUT.INPUT.MISSION.LOAD_STATE = uint8(4);
    case 4 % Wait 400ms while loading info
        CONTROL_OUT.INPUT.MISSION.MISSION_INFO(SEQUENCE+1,9) =
CONTROL_IN.INPUT.MISSION.LATITUDE_J;
```

```
        CONTROL_OUT.INPUT.MISSION.MISSION_INFO(SEQUENCE+1,10) =
CONTROL_IN.INPUT.MISSION.LONGITUDE_J;
        CONTROL_OUT.INPUT.MISSION.MISSION_INFO(SEQUENCE+1,11) =
CONTROL_IN.INPUT.MISSION.ALTITUDE_J;
        COUNTER = COUNTER+1;
        CONTROL_OUT.INPUT.MISSION.LOAD_STATE = uint8(3);
        if (COUNTER >= 7) && (SEQUENCE >
CONTROL_IN.INPUT.MISSION.COUNT_J-2)
            CONTROL_OUT.INPUT.MISSION.LOAD_STATE = uint8(5);
        elseif (COUNTER >= 7)
            CONTROL_OUT.INPUT.MISSION.LOAD_STATE = uint8(2);
            SEQUENCE = SEQUENCE+1;
        end
    case 5 % Send ACK
        MAVLINK_OUT.DIVISOR(41) = uint8(100);
        MAVLINK_OUT.OFFSET(41) = 0;
        MAVLINK_OUT.DIVISOR(48) = 1;
        MAVLINK_OUT.OFFSET(48) = 0;
        CONTROL_OUT.INPUT.MISSION.LOAD_STATE = uint8(6);
        CONTROL_OUT.INPUT.MISSION.MISSION_TYPE = uint8(0);
    case 6 % Turn off ACK and exit
        MAVLINK_OUT.DIVISOR(48) = uint8(100);
        MAVLINK_OUT.OFFSET(48) = 0;
        CONTROL_OUT.INPUT.MISSION.LOAD_STATE = uint8(0);
        CONTROL_OUT.INPUT.MISSION.MSG_J = uint8(0);
        SEQUENCE = 0;
        CONTROL_OUT.INPUT.MISSION.LOADED = uint8(1);
end
```

ANEXO C – GESTIÓN DE TRAYECTORIAS

Código completo de la gestión de trayectorias.

UAV_CONTROL_SYSTEM-->COMMUNICATIONS--> MISSION MANAGEMENT

```
function [CONTROL_OUT,MAVLINK_OUT] =
MISSION_CMD(CONTROL_IN,MAVLINK_IN)

CONTROL_OUT = CONTROL_IN;
MAVLINK_OUT = MAVLINK_IN;

persistent INITIALIZATION SEQUENCE COUNTER WP SKIPER
LONGITUDE_H...
            LATITUDE_H ALTITUDE_H
if isempty(INITIALIZATION)
    SEQUENCE = 0;
    INITIALIZATION = 1;
    COUNTER = 0;
    SKIPER = 0;
    WP = 1;
    LONGITUDE_H = 0;
    LATITUDE_H = 0;
    ALTITUDE_H = 0;
end

%% AUTONOMOUS MODE MANAGEMENT
% This function defines the autonomous mode. It requires
the mission info
% to have been loaded and processed to give references. If
there is no
% mission info, the UAV will wait stabilized at it´s
position.

if CONTROL_IN.STATE.FLIGHT_MODE == 3 &&
(CONTROL_IN.INPUT.MISSION.GO_HOME ||
CONTROL_IN.INPUT.MISSION.ITEM_REACHED)
    % POSITION
    HOME_UAV = double(CONTROL_IN.INPUT.GPS.LLA_HOME); %
[deg deg m]
    % MISSION HOME
    HOME_LLA = [CONTROL_IN.INPUT.MISSION.MISSION_INFO(1,9)
CONTROL_IN.INPUT.MISSION.MISSION_INFO(1,10) ...
        CONTROL_IN.INPUT.MISSION.MISSION_INFO(1,11)];

    % WP selector
```

```
    % CONTROL_OUT.INPUT.MISSION.CURRENT_COUNT_J =
uint16(WP);
    if CONTROL_IN.INPUT.MISSION.ITEM_REACHED
        CONTROL_OUT.INPUT.MISSION.ITEM_REACHED = uint8(0);
        CONTROL_OUT.INPUT.MISSION.CURRENT_COUNT_J =
uint16(WP);
        WP = WP+1;
    end
    if CONTROL_OUT.INPUT.MISSION.CURRENT_COUNT_J >
CONTROL_IN.INPUT.MISSION.COUNT_J-1
        CONTROL_OUT.INPUT.MISSION.CURRENT_COUNT_J =
CONTROL_IN.INPUT.MISSION.COUNT_J-1;
    end

    % LLA TO FLAT EARTH
    % Relative WP to HOME_UAV
    if CONTROL_IN.INPUT.MISSION.GO_HOME
        LATITUDE = HOME_LLA(1)-HOME_UAV(1); % deg
        LONGITUDE = HOME_LLA(2)-HOME_UAV(2); % deg
        ALTITUDE = HOME_LLA(3)-HOME_UAV(3); % m
        LATITUDE_H = LATITUDE;
        LONGITUDE_H = LONGITUDE;
        ALTITUDE_H = ALTITUDE;
    else
        % Relative WP to HOME_WP
        LATITUDE_R =
CONTROL_IN.INPUT.MISSION.MISSION_INFO(CONTROL_OUT.INPUT.MIS
SION.CURRENT_COUNT_J+1 ,9)-HOME_LLA(1); % deg
        LONGITUDE_R =
CONTROL_IN.INPUT.MISSION.MISSION_INFO(CONTROL_OUT.INPUT.MIS
SION.CURRENT_COUNT_J+1 ,10)-HOME_LLA(2); % deg
        ALTITUDE_R =
CONTROL_IN.INPUT.MISSION.MISSION_INFO(CONTROL_OUT.INPUT.MIS
SION.CURRENT_COUNT_J+1 ,11)-HOME_LLA(3); % m
        % Translation from relative WP to HOME_WP to
HOME_UAV
        LATITUDE = LATITUDE_R + LATITUDE_H;
        LONGITUDE = LONGITUDE_R + LONGITUDE_H;
        ALTITUDE = ALTITUDE_R + ALTITUDE_H;
    end
    % Latitude and longitude wrapping
    CONTROL_OUT.INPUT.MISSION.EARTH_POS_TARGET_LLA = [1e7
1e7 1e3]'.*[...

CONTROL_IN.INPUT.MISSION.MISSION_INFO(CONTROL_OUT.INPUT.MIS
SION.CURRENT_COUNT_J+1 ,9) ...
```

```
CONTROL_IN.INPUT.MISSION.MISSION_INFO(CONTROL_OUT.INPUT.MIS
SION.CURRENT_COUNT_J+1 ,10) ...

CONTROL_IN.INPUT.MISSION.MISSION_INFO(CONTROL_OUT.INPUT.MIS
SION.CURRENT_COUNT_J+1 ,11)]';
    LAT_WRAPPED =
(abs(LATITUDE)>180)*(mod(LATITUDE+180,360)-180) + ...
                        (abs(LATITUDE)<=180)*LATITUDE;
    WRAP_FLAG = (abs(LAT_WRAPPED)>90);
    LAT_WRAPPED = sign(LAT_WRAPPED)*(-
abs(LAT_WRAPPED)+180)*(WRAP_FLAG==1) + ...
                        LAT_WRAPPED*(WRAP_FLAG==0);
    LON_WRAPPED = LONGITUDE + 180*(WRAP_FLAG==1);
    LON_WRAPPED =
(abs(LON_WRAPPED)>180)*(mod(LON_WRAPPED+180,360)-180) + ...
                        (abs(LON_WRAPPED)<=180)*LON_WRAPPED;
    % Conversion to rad
    LAT_RAD = pi/180*LAT_WRAPPED;
    LON_RAD = pi/180*LON_WRAPPED;
    % Initial latitude in rad
    LAT_HOME = pi/180*HOME_UAV(1);
    % Find Radian/Distance
    ECUATORIAN_RADIUS = (6378137 + 6356752)/2;
    FLATNESS = 1/298.257223563;
    EPS = sqrt(1-(1-FLATNESS)^2);
    DENOM = 1-EPS^2*sin(LAT_HOME)^2;
    dNorth = atan2(1,(1-
EPS^2)*ECUATORIAN_RADIUS/DENOM^1.5);
    dEast =
atan2(1,ECUATORIAN_RADIUS*cos(LAT_HOME)/sqrt(DENOM));
    % Psi: angle clockwise between the x-axis and north
    PSi = 0;
    s_Psi = sin(PSi);
    c_Psi = cos(PSi);
    POS_X =  LAT_RAD/dNorth*c_Psi + LON_RAD/dEast*s_Psi;
    POS_Y = -LAT_RAD/dNorth*s_Psi + LON_RAD/dEast*c_Psi;
    POS_Z = -ALTITUDE;
    WP_XYZ = single([POS_X POS_Y POS_Z]'); % m (WP XYZ)

    % LLA TO FLAT EARTH
    % Relative WP to POS_UAV
    POS_LLA = double(CONTROL_IN.INPUT.GPS.LLA_POS);
    WP_LLA = [1e-7 1e-7 1e-
3]'.*CONTROL_OUT.INPUT.MISSION.EARTH_POS_TARGET_LLA;
    LATITUDE = WP_LLA(1)-POS_LLA(1); % deg
    LONGITUDE = WP_LLA(2)-POS_LLA(2); % deg
    ALTITUDE = WP_LLA(3)-POS_LLA(3); % m
```

```
    LAT_WRAPPED =
(abs(LATITUDE)>180)*(mod(LATITUDE+180,360)-180) + ...
                        (abs(LATITUDE)<=180)*LATITUDE;
    WRAP_FLAG = (abs(LAT_WRAPPED)>90);
    LAT_WRAPPED = sign(LAT_WRAPPED)*(-
abs(LAT_WRAPPED)+180)*(WRAP_FLAG==1) + ...
                        LAT_WRAPPED*(WRAP_FLAG==0);
    LON_WRAPPED = LONGITUDE + 180*(WRAP_FLAG==1);
    LON_WRAPPED =
(abs(LON_WRAPPED)>180)*(mod(LON_WRAPPED+180,360)-180) + ...
                        (abs(LON_WRAPPED)<=180)*LON_WRAPPED;
    % Conversion to rad
    LAT_RAD = pi/180*LAT_WRAPPED;
    LON_RAD = pi/180*LON_WRAPPED;
    % Initial latitude in rad
    LAT_HOME = pi/180*HOME_UAV(1);
    % Find Radian/Distance
    ECUATORIAN_RADIUS = (6378137 + 6356752)/2;
    FLATNESS = 1/298.257223563;
    EPS = sqrt(1-(1-FLATNESS)^2);
    DENOM = 1-EPS^2*sin(LAT_HOME)^2;
    dNorth = atan2(1,(1-
EPS^2)*ECUATORIAN_RADIUS/DENOM^1.5);
    dEast =
atan2(1,ECUATORIAN_RADIUS*cos(LAT_HOME)/sqrt(DENOM));
    % Psi: angle clockwise between the x-axis and north
    PSi = 0;
    s_Psi = sin(PSi);
    c_Psi = cos(PSi);
    POS_X =  LAT_RAD/dNorth*c_Psi + LON_RAD/dEast*s_Psi;
    POS_Y = -LAT_RAD/dNorth*s_Psi + LON_RAD/dEast*c_Psi;
    POS_Z = -ALTITUDE;
    POS_WP = single([POS_X POS_Y POS_Z]'); % m (XYZ from
POS to WP)

    % LLA to FLAT EARTH
    % POS to HOME
    LATITUDE = POS_LLA(1)-HOME_UAV(1); % deg
    LONGITUDE = POS_LLA(2)-HOME_UAV(2); % deg
    ALTITUDE = POS_LLA(3)-HOME_UAV(3); % m
    LAT_WRAPPED =
(abs(LATITUDE)>180)*(mod(LATITUDE+180,360)-180) + ...
                        (abs(LATITUDE)<=180)*LATITUDE;
    WRAP_FLAG = (abs(LAT_WRAPPED)>90);
    LAT_WRAPPED = sign(LAT_WRAPPED)*(-
abs(LAT_WRAPPED)+180)*(WRAP_FLAG==1) + ...
                        LAT_WRAPPED*(WRAP_FLAG==0);
    LON_WRAPPED = LONGITUDE + 180*(WRAP_FLAG==1);
```

```
    LON_WRAPPED =
(abs(LON_WRAPPED)>180)*(mod(LON_WRAPPED+180,360)-180) + ...
                       (abs(LON_WRAPPED)<=180)*LON_WRAPPED;
    % Conversion to rad
    LAT_RAD = pi/180*LAT_WRAPPED;
    LON_RAD = pi/180*LON_WRAPPED;
    % Initial latitude in rad
    LAT_HOME = pi/180*HOME_UAV(1);
    % Find Radian/Distance
    ECUATORIAN_RADIUS = (6378137 + 6356752)/2;
    FLATNESS = 1/298.257223563;
    EPS = sqrt(1-(1-FLATNESS)^2);
    DENOM = 1-EPS^2*sin(LAT_HOME)^2;
    dNorth = atan2(1,(1-
EPS^2)*ECUATORIAN_RADIUS/DENOM^1.5);
    dEast =
atan2(1,ECUATORIAN_RADIUS*cos(LAT_HOME)/sqrt(DENOM));
    % Psi: angle clockwise between the x-axis and north
    PSi = 0;
    s_Psi = sin(PSi);
    c_Psi = cos(PSi);
    POS_X =  LAT_RAD/dNorth*c_Psi + LON_RAD/dEast*s_Psi;
    POS_Y = -LAT_RAD/dNorth*s_Psi + LON_RAD/dEast*c_Psi;
    POS_Z = -ALTITUDE;
    POS_HOME = single([POS_X POS_Y POS_Z]'); % m (XYZ from
POS to HOME)

    % IMU BUS UPDATE
    if SKIPER == 1
    CONTROL_OUT.INPUT.MISSION.POS_HOME = single(POS_HOME);
    CONTROL_OUT.INPUT.MISSION.POS_WP = single(POS_WP);
    CONTROL_OUT.INPUT.MISSION.WP_XYZ = single(WP_XYZ);
    CONTROL_OUT.INPUT.MISSION.GO_HOME = uint8(0);
    end
end

%% TRAYECTORY MANAGEMENT
% :)
if SKIPER == 1 && CONTROL_IN.STATE.FLIGHT_MODE == 3 &&
~CONTROL_IN.INPUT.MISSION.ITEM_REACHED
    POS_WP = CONTROL_IN.INPUT.MISSION.POS_WP; % m
    POS_HOME = CONTROL_IN.INPUT.MISSION.POS_HOME;
    WP_XYZ = CONTROL_IN.INPUT.MISSION.WP_XYZ;
    DIS = (POS_WP(1)^2+POS_WP(2)^2+POS_WP(3)^2)^0.5;
    if DIS > 6
      POS_WP = POS_WP.*0.5;
      CONTROL_OUT.INPUT.MISSION.POS_WP = POS_WP;
    else
```

```
      if DIS > 0
      EARTH_POS_TARGET = POS_WP + POS_HOME;
      DIS = ((WP_XYZ(1)-EARTH_POS_TARGET(1))^2+(WP_XYZ(2)-
EARTH_POS_TARGET(2))^2+...
          (WP_XYZ(3)-EARTH_POS_TARGET(3))^2)^0.5;
      if DIS > 3 && ~CONTROL_IN.INPUT.MISSION.PARTIAL_ITEM
          WP = WP-1;
          CONTROL_OUT.INPUT.MISSION.PARTIAL_ITEM =
uint8(1);
          if WP < 0
              WP = 0;
          end
      end
      CONTROL_OUT.INPUT.EARTH_POS_TARGET =
single(EARTH_POS_TARGET);
      CONTROL_OUT.INPUT.MISSION.NEW_TARGET = uint8(1);
      end
    end
end

SKIPER = 1;
```

ANEXO D - DATASHEETS

En este anexo se incluyen los enlaces a los componentes que se han utilizado en este proyecto.

Raspberry Pi Zero

https://cdn.sparkfun.com/assets/learn_tutorials/6/7/6/PiZero_1.pdf
https://cdn-learn.adafruit.com/downloads/pdf/introducing-the-raspberry-pi-zero.pdf

Magnetómetro AK8963 integrado en IMU MPU6250

https://www.invensense.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf
https://www.akm.com/akm/en/file/datasheet/AK8963C.pdf

Barómetro MS5611

http://www.te.com/commerce/DocumentDelivery/DDEController?Action=showdoc&DocId=Data+Sheet%7FMS5611-01BA03%7FB3%7Fpdf%7FEnglish%7FENG_DS_MS5611-01BA03_B3.pdf%7FCAT-BLPS0036

GPS M8N Ublox

https://www.u-blox.com/sites/default/files/NEO-M8_DataSheet_(UBX-13003366)

Parte 2
Presupuesto

ÍNDICE DE PRESUPUESTO

1. Mediciones

1.1 Componentes

COMPONENTE	UNIDADES [unid]
Microcontrolador Raspberry Pi Zero	1
Hélices 30mm	8
Batería LIPO 6V/210mAh	2
Magnetómetro AK8963	1
Barómetro MS5611	1
Módulo GPS M8P	2

Tabla 1: Componentes

1.2 Equipos y herramientas

COMPONENTE	UNIDADES [unid]
Ordenador	1
Conector USB	2
Cargador de baterías	1

Tabla 2: Equipos

1.3 Software

COMPONENTE	UNIDADES [unid]
Matlab	1
Mission Planner	1
u-Center	1
Flightgear	1
Waijung Blockset	1

Tabla 3: Software

1.4 Mano de obra

COMPONENTE	HORAS DE TRABAJO [horas]
Estado del arte	20
Calibración de sensores	20
Programación	140
Documentación	130
Simulación	70
Diseño	130

Tabla 4: Mano de obra

2. Precios unitarios

2.1 Componentes

COMPONENTE	PRECIO UNITARIO [€/unid]
Microcontrolador Raspberry Pi Zero	5.61
Hélices 30mm	0.31
Batería LIPO 6V/210mAh	16.00
Magnetómetro AK8963	2.35
Barómetro MS5611	4.10
Módulo GPS M8P	11.10

Tabla 5: Precio unitario componentes

2.2 Equipos y herramientas

COMPONENTE	PRECIO UNITARIO [€/unid]
Ordenador	1400
Conector USB	4.50
Cargador de baterías	100

Tabla 6: Precio unitario Equipos

2.3 Software

COMPONENTE	PRECIO UNITARIO [€/unid]
Matlab	700.00
Mission Planner	0.00
u-Center	0.00
Flightgear	0.00
Waijung Blockset	0.00

Tabla 7: Precio unitario Software

2.4 Mano de obra

COMPONENTE	PRECIO HORA [€/h]
Estado del arte	10
Calibración de sensores	50
Programación	50
Documentación	30
Simulación	30
Diseño	60

Tabla 8: Precio hora Mano de obra

3. Sumas parciales

3.1 Componentes

COMPONENTE	UNIDADES [unid]	PRECIO UNITARIO [€/unid]	TOTAL [€]
Microcontrolador RPI	1	5.61	5.61
Hélices 30mm	8	0.31	2.48
Batería LIPO 6V/210mAh	2	16.00	32.00
Magnetómetro AK8963	1	2.35	2.35
Barómetro MS5611	1	4.10	4.10
Módulo GPS M8P	2	11.10	22.20
		COSTES DIRECTOS	118.36
		COSTES INDIRECTOS (2.5%)	2.95
		TOTAL	121.26

Tabla 9: Precio total componentes

3.2 Equipos y herramientas

COMPONENTE	UNIDADES [unid]	PRECIO UNITARIO [€/unid]	TOTAL [€]
Ordenador	1	1400.00	1400.00
Conector USB	2	4.50	9.00
Cargador de baterías	1	100.00	100.00
		COSTES DIRECTOS	1518.00
		COSTES INDIRECTOS (2.5%)	37.95
		TOTAL	1555.95

3.3 Software

COMPONENTE	UNIDADES	PRECIO UNITARIO [€/unid]	TOTAL [€]
Matlab	1	700.00	700.00
Mission Planner	1	0.00	0.00
u-Center	1	0.00	0.00
Flightgear	1	0.00	0.00
Waijung Blockset	1	0.00	0.00
		COSTES DIRECTOS	700
		COSTES INDIRECTOS (2.5%)	17.5
		TOTAL	717.5

Tabla 10: Precio total equipos y herramientas

Tabla 11: Precio total Software

3.4 Mano de obra

COMPONENTE	HORAS DE TRABAJO [horas]	PRECIO HORA [€/h]	TOTAL [€]
Estado del arte	20	10	200.00
Calibración de sensores	20	50	1000.00
Programación	140	50	7000.00
Documentación	130	30	4200.00
Simulación	70	30	2100.00
Diseño	130	60	8400.00
		COSTES DIRECTOS	22900
		COSTES INDIRECTOS (2.5%)	572.5
			23472.5

Tabla 12: Precio total mano de obra

4. Presupuesto general

CONCEPTO	COSTE PARCIAL [€]	
Componentes	121.26	
Equipos y herramientas	1555.95	
Software	717.5	
Mano de obra	23472.5	
P.E.M.		25867,21
Gastos Generales (13%)		3362.73
Beneficio Industrial (6%)		1681.4
Suma		30911.34
IVA		6491.4
Total		37402.74

Tabla 13: Presupuesto general