Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE GRADO

TÍTULO DEL TFG: Diseño e implementación del sistema de aterrizaje autónomo para un UAV en una embarcación

TITULACIÓN: Grado en ingeniería de aeronavegación

AUTOR: Raúl Pedrosa Cabello

DIRECTOR: SERGI TRES MARTÍNEZ

CO-DIRECTOR: ÓSCAR CASAS PIEDRAFITA

FECHA: JULIO 2020

Título: Diseño e implementación del sistema de aterrizaje autónomo para un UAV en una embarcación

Autor: Raúl Pedrosa Cabello

Director: Sergi Tres Martínez

Codirector: Óscar Casas Piedrafita

Fecha: Julio 2020

Resumen

Freeda es un proyecto social con el objetivo de dar soporte mediante tecnología dron a la ONG Proactiva Open Arms.

La intención con el soporte es ayudar en las tareas de búsqueda de rescate a personas que quedan a la deriva en el Mediterráneo tras el intento fallido de alcanzar las costas europeas.

Freeda es totalmente autónomo, de tal forma que despeja, realiza las misiones que tenga programadas y aterriza solo. El desarrollo se divide en varias líneas y este estudio se focaliza en el aterrizaje.

En este documento se da un primer paso hacia el desarrollo del aterrizaje, en el que se expone las diferentes vías para desarrollarlo y se centra especialmente en la idea principal que tenían en el proyecto general, que el dron aterrice mediante visión artificial en una red del barco, en el que no necesite ayuda externa y tuviera que estar recibiendo información.

La meta a resolver y que se da respuesta durante el documento es cumplir dos objetivos de los requisitos necesarios para el aterrizaje mediante visión artificial. Por una parte, desarrollar un algoritmo capaz de obtener un vector que se dirija hacia la red. El segundo es tener el control total del mecanismo que mueve a la cámara.

Como alternativa también se hace un estudio y se exponen los requisitos para realizar el aterrizaje con ayuda externa.

Title: Design and implementation of the autonomous landing system for a UAV on a boat

Author: Raúl Pedrosa Cabello

Director: Óscar Casas Piedrafita

Date: July 2020

Overview

Freeda is a social project with the aim of supporting the ONG Proactiva Open Arms with drone technology.

The intention with the support is to help rescue people who are adrift in the Mediterranean after the failed attempt to reach the European coasts.

Freeda is completely autonomous, in such a way that it clears, carries out the missions it has scheduled and lands alone. The development is divided into several lines and this study focuses on landing.

In this document a first step towards the development of the landing is given, in which the different ways to develop it are exposed and it focuses especially on the main idea they had in the general project, that the drone lands using artificial vision in a net of the ship, where you do not need outside help and have to be receiving information.

The goal to be solved and that is answered during the document is to resolve two objectives of the requirements necessary for landing using artificial vision. On the one hand, to develop an algorithm capable of obtaining a vector that is directed towards the network. On the other hand, the second is to have full control of the mechanism that moves the camera.

As an alternative, a study is also made and the requirements for landing with external help are presented.

AGRADECIMIENTOS

A mis padres y a mi hermano,
por ser siempre un apoyo incondicional.

A Sergi Tres, por darme conocimientos
y guiarme en esta gran experiencia.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABLAS

ABREVIATURAS

Abreviatura	*Definición*
UAV	Unmanned Aerial Vehicle
VTOL	Vertical Take-Off and Landing
RPAS	Remotely Piloted Aircraft System
ILS	Instrument Landing System
GPS	Global Positioning System
BEC	Battery Eliminator Circuit
GPIO	General Purpose Input / Output
DC	Duty Cycle
DGPS	Differential Global Positioning System
RTK	Real Time Kinematic

INTRODUCCIÓN

La finalidad de este documento es hacer un análisis y comparación de sistemas de aterrizaje autónomo de un UAV en una embarcación, para posteriormente poder desarrollar e implementar la mejor opción en el proyecto Freeda, el cual consiste en la detección de embarcaciones en el mar Mediterráneo a partir de tecnología dron, para poder rescatar a los refugiados. El proyecto ha sido diseñado para despegar y aterrizar en la embarcación Open Arms.

Debido a la problemática de la crisis de los refugiados, ha surgido la necesidad de buscar soluciones al respecto. Tanto este problema como la tecnología de la que se dispone en el proyecto, se explica en el capítulo 1.

La idea principal para Freeda, es que el UAV pueda aterrizar en la embarcación por si solo. Es decir, aparte de que sea totalmente autónomo, que el dron no necesite obtener información externa.

Como se explica en el capítulo 2, existen dos tipos bien diferenciados de sistemas de aterrizaje autónomos. Sistemas cooperativos, en el que se recibe ayuda externa y sistemas no cooperativos. Al ser el segundo, la meta principal del proyecto Freeda, este marcará los objetivos principales de la memoria.

Uno de los dos objetivos principales del documento es obtener un vector con origen en el UAV y dirección hacia la red, que es donde el dron aterrizará. En la segunda parte del capítulo 2 se detalla un algoritmo capaz de cumplir este objetivo.

El segundo objetivo principal es tener el control total del gimbal, el cual es el mecanismo donde está colocada la cámara. Este punto es primordial, ya que será necesario para poder estar apuntando la cámara hacia la red, y poder dar partida al objetivo anterior. Este objetivo se desarrolla en el capítulo 3.

Como posible sistema redundante o alternativa, al final de la memoria se hace mención y se explica los posibles pasos a seguir para futuros desarrollos con el sistema cooperativo.

CAPÍTULO 1. SITUACIÓN ACTUAL

1.1. Problemática

1.1.1. La crisis de los refugiados

Desde hace años, miles de personas deben abandonar sus hogares en el norte de África debido a la guerra y a la pobreza. La mayoría de los refugiados son de origen siriano, componiendo casi un 30% del total. Seguidos por los inmigrantes de Afganistán e Irak.[1]

Tabla 1.1. Nacionalidades de procedencia de los refugiados

Nacionalidad	TOTAL	Porcentaje
Syria	643848	28,76%
Afghanistan	328022	14,65%
Iraq	134368	6,00%
Eritrea	118284	5,28%
Nigeria	98597	4,40%
Syria	63434	2,83%
Tunisia	51474	2,30%
Otros	800280	35,75%
TOTAL	2238307	100,00%

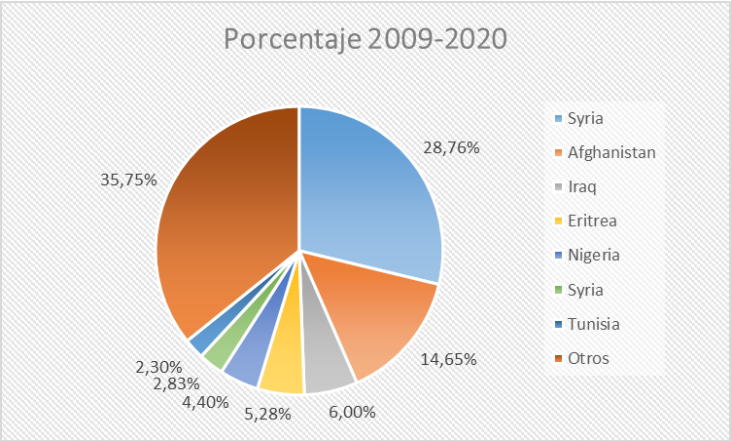

Fig. 1.1. Nacionalidades de procedencia de los refugiados

En 2016, se hizo cierre de la llamada “ruta de los Balcanes”, por lo que se limitó bruscamente el flujo de personas sin la documentación adecuada en países como Albania, Macedonia o Bulgaria, dejando a cientos de refugiados atrapados en Grecia, por lo que se ha convertido en una opción menos atractiva para refugiados y migrantes que se adentran en el Mediterráneo hacia las fronteras de Europa. Esto supone optar por rutas más peligrosas y

mortíferas. Miles de refugiados recurren a traficantes de personas a falta de vías legales para alcanzar un lugar seguro.[2]

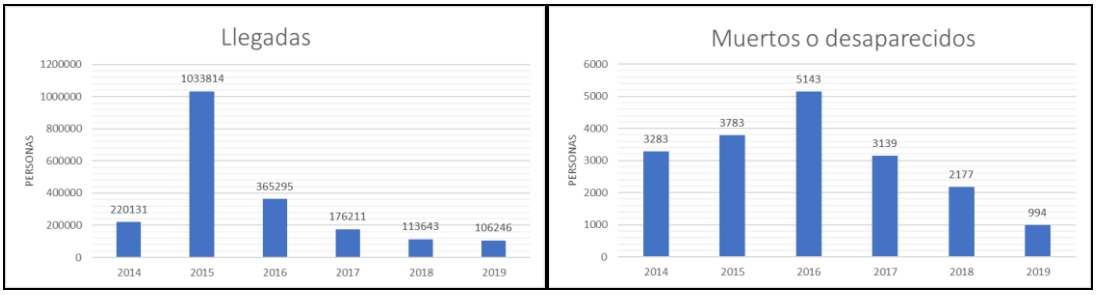

Fig. 1.2. Llegadas y muertes 2014-2019

Debido al cierre de fronteras terrestres, tanto en Grecia como en Turquía, empezó a crecer el flujo de rutas por el mar mediterráneo. Voluntariamente se adentran en embarcaciones poco seguras, con la esperanza de llegar o ser interceptados por guardacostas.

1.1.2. Proactiva Open Arms

Proactiva Open Arms es una organización no gubernamental y sin ánimo de lucro, formada en 2015 y cuya principal misión es proteger con su presencia en el mar a aquellas personas que intentan llegar a Europa huyendo de conflictos bélicos, persecución o pobreza. [3]

Se originó a partir de Pro-Activa Serveis Aquàtics, una empresa de servicios de rescate y salvavidas, original de Barcelona y fundada en el año 2000. Fue a partir de septiembre de 2015 cuando decidieron formar Open Arms, y adentrarse en la isla de Lesbos para ayudar a los refugiados que llegaban a la costa griega.

Open Arms durante los años que ha estado activo, se ha enfrentado a diversos problemas, tales como encontrarse en la situación de no poder desembarcar en ningún puerto europeo. Esto se debe a que la acogida de migrantes ha sido, desde siempre, un punto de enfrentamiento en la Unión Europea, ya que no consiguen solventar el reparto de inmigrantes rescatados.

Otro punto débil con el que se encuentran actualmente son las averías del barco. El Open Arms tiene casi 50 años, y su motor no funciona del todo bien y deben cambiarlo. Para financiarse, realizan campañas de crowdfunding.[4]

Fig. 1.3. Open Arms

1.1.3. Hemav Foundation

Freeda está llevado a cabo dentro de Hemav Foundation, una organización privada sin ánimo de lucro de la empresa Hemav, para dar cabida a proyectos sociales de innovación, a través de tecnología dron, los cuales sin la fundación no podrían tener cabida dentro de la empresa, ya que no aportan beneficios económicos directos. [5]

Gracias a la fundación, la empresa es capaz de desarrollar proyectos tales como Locust, para el control de la plaga de langostas; Kids, un proyecto para adentrar en el mundo de los drones a los más jóvenes o Freeda, proyecto del cual forma parte este documento.

1.2. Tecnología usada en Freeda

En el siguiente apartado se expondrá la tecnología elegida para el proyecto Freeda. Cabe recordar que el objetivo de esta memoria es estudiar y dar un primer paso en el desarrollo del aterrizaje autónomo del UAV, por lo que el estudio de qué tipo de dron o que componentes utilizar, ya se hizo anteriormente.

Plataforma UAV

Para elegir la plataforma UAV, se tuvo en cuenta varios aspectos, tales como la duración de vuelo, versatilidad, facilidad en su uso y precio. Entre las opciones se encuentran Multirrotor, Helicóptero autónomo, ala fija y VTOL.

Después de valorar las diferentes opciones, se tomó la decisión de que la mejor plataforma sería un UAV de ala fija, ya que pese a complicarse de esta forma el despegue y el aterrizaje en frente de las otras opciones, es la opción con mayor autonomía y alcance, aspectos cruciales en el proyecto.

Fig. 1.4. UAV de ala fija de Hemav

1.2.1. Hardware

Computadora - Nvidia Jetson Nano

La Nvidia Jetson Nano es un ordenador de placa simple que lleva incorporado una CPU ARM Cortex-A57 MPCore de 4 núcleos (capaz de proporcionarnos 472 gigaflops de potencia), una GPU Nvidia Maxwell con 128 núcleos CUDA (capaz de ejecutar la librería de procesado de datos CUDA-X AI), 4 Gb de RAM, 16 GB de almacenamiento y 4 puertos USB 3.0.[6]

En principio se iba a utilizar la Raspberry Pi, computadora más económica, pero ante la necesidad de tener mayor potencia para el procesamiento de los algoritmos, se tomó la decisión de usar la nvidia, ya que cuenta con el

rendimiento y las capacidades necesarias para ejecutar cargas de trabajo de la IA moderna de forma rápida.

Tabla 1.2. Especificaciones Técnicas de Nvidia Jetson Nano

Especificaciones técnicas	
GPU	Maxwell de 128 núcleos
CPU	ARM de cuatro núcleos A57 a 1,43 GHz
Memoria	LPDDR4 de 4 GB y 64 bits a 25,6 GB/s
Almacenamiento	microSD (no incluida)
Codificación de vídeo	4K a 30 | 4x 1080p a 30 | 9x 720p a 30 (H.264/H.265)
Descodificador de vídeo	4K a60 | 2x 4Ka 30 | 8x 1080p a 30 | 18x 720p a 30| (H.264/H.265)
Cámara	2x MIPI CSI-2 DPHY lanes
Conectividad	Gigabit Ethernet, M.2 tipo E
Pantalla	HDMI y DP
USB	5 USB 3.0, USB 2.0 Micro-B
Otros	GPIO, I2C, I2S, SPI, UART
Mecánicas	101 mm x 80 mm x 29 mm

Fig. 1.5. Nvidia Jetson Nano

Controladora – Pixhawk 4

La controladora tiene incorporado tanto el hardware como software necesario para recoger la información de los sensores y procesar una respuesta para proporcionar las instrucciones que accionan los motores y los actuadores del dron.[7]

La controladora seleccionada es la Pixhawk 4, la cual puede actuar como piloto automático, ya que puede procesar la información recibida por la computadora y dirigir el UAV como estuviese programado.

Tabla 1.3. Especificaciones técnicas Pixhawk 4

Especificaciones técnicas	
Procesador principal de FMU	STM32F765 32 Bit Arm ® Cortex®-M7, 216MHz, memoria de 2MB, 512KB RAM
Procesador IO	STM32F100 32 Bit Arm ® Cortex®-M3, 24MHz, 8KB SRAM
Accel / Gyro	ICM-20689
Barómetro	MS5611
Dimensiones	44x84x12mm
Peso	44g

Interfaces

- 8-16 PWM salidas servo (8 de IO, 8 de FMU)
- 3 entradas PWM / Capture dedicadas en FMU
- Entrada de R / C dedicada para CPPM
- Entrada de R / C dedicada para Spektrum / DSM y S.Bus con entrada analógica / PWM RSSI
- Dedicado servo salida S.Bus
- 5 puertos seriales de uso general, 2 con control de flujo de flil, 1 con límite de corriente de 1.5 A por separado
- 3 I2C puertos
- 4 autobuses SPI
- Hasta 2 CANBuses para CAN dual con ESC serial, cada CANBus tiene controles silenciosos individuales o control ESC RX-MUX
- Entradas analógicas para tensión / corriente de 2 baterías
- 2 entradas analógicas adicionales
- Salida del módulo de potencia: 4.9 ~ 5.5V
- Voltaje máximo de entrada: 6V
- Detección de corriente máxima: 120A
- Entrada de alimentación USB: 4.75 ~ 5.25V
- Servo Entrada de riel: 0 ~ 36V
- Temperatura de funcionamiento ~ 40 ~ 85C
- Temperatura de almacenamiento. -40 ~ 85C
- Cumple con RoHS (sin plomo)

Fig. 1.6. Pixhawk 4

Telemetría – kit pMDDL2450

Con la telemetría se permite tener comunicación con el UAV, de esta forma se puede obtener la información de los sensores del dron para saber como está actuando, y a la vez poder enviarle información a la computadora para que actúe de la forma que queramos.[8]

En Freeda se utiliza para la telemetría el kit de desarrollo pMDDL2450, el cual es un enlace de datos digitales inalámbrico 2x2 MIMO de alta potencia diseñado para proporcionar conexiones inalámbricas de alto rendimiento en un módulo OEM compacto y robusto para la integración del sistema.

El kit está formado por 2 módulos pMDDL2450 OEM, 2 placas base Pico Ethernet, 4 cables de antena UFL, 4 antenas ducky de goma, 2 cables Cat5 Ethernet, 2 cables RS232 Serial y 2 adaptadores de potencia 12VDC.

Fig. 1.7. Componentes del Kit de desarrollo pMDDL2450

Características:

- 2X2 MIMO robusto de 2.4 GHz de operación
- Combinación de Relación Máxima (MRC), Decodificación de Probabilidad Máxima (ML)
- Verificación de Paridad de Baja Densidad (LDPC)
- Hasta 25 Mbps Iperf Rendimiento @ 8 MHz canal (-78 dBm)
- Hasta 2 Mbps Rendimiento de Iperf a un canal de 4 MHz (-102 dBm)
- Extremadamente pequeña huella y muy ligero
- Puerto de comunicación de serie: puertos duales Ethernet 10/100 (LAN / WAN)
- Admite redes punto a punto y punto a multipunto
- Modos de operación Master, Remote y Relay
- Potencia de transmisión total ajustable (hasta 1 W)
- Interfaz a través de la consola local, telnet y navegador web.
- Actualización de firmware inalámbrico remoto y local y local a través de FTP.
- Voltajes de entrada: Digital Voltage = 3.3 V / RF Voltage = 5 V
- Temperatura de funcionamiento: -40 ºC – 85 ºC
- Humedad de funcionamiento: 5 – 95%

Cámara – Arducam 18MP AR1820HS

Fig. 1.8. Cámara AR1820HS

Características:[9]

- Tipo de sensor: AR1820
- Tamaño de píxel: 1.25um x 1.25um
- Tamaño óptico: 1 / 2.3 pulgadas (4: 3)
- Píxeles activos: 4912 H x 3684 V
- Matriz de filtro de color: patrón RGB Bayer

- Especificaciones de la lente: Predeterminado: lente M12 de baja distorsión (Número de pieza: M27280M07S)
- Resolución de ADC: 12 bits, en chip
- Velocidad de fotogramas: 15 fps a 18 Mp de resolución completa (MIPI-4L), 20 fps a 14 Mp (MIPI-4L), 30 fps a 8 Mp (MIPI-4L), 60 fps a Full HD + 27.5% EIS, 120 fps a Full HD
- Tipo de interfaz: 2-Lane / 4-Lane MIPI
- Formato de salida: salida RAW8 / RAW10
- Tamaño del tablero: 40x40 mm

1.2.2. Software

Sistema Operativo - Linux

La computadora Nvidia Jetson Nano, funciona con el sistema operativo Linux. Más concretamente, el sistema oficial para la placa se llama Linux4Tegra, que es una versión de Ubuntu 18.04 diseñada para ejecutarse en el hardware de Nvidia.[10]

Lenguaje de Programación - Python

Hasta el momento, todo el desarrollo de software de programación ha sido llevado a cabo mediante el lenguaje Python, ya que no se necesita licencia de pago para su uso, y pose las librerías necesarias para los algoritmos de visión por computador, necesarios para detectar las embarcaciones, y la librería Dronekit.

Librería UAV - Dronekit

Es la librería de python que permite obtener los valores de los sensores de la controladora del UAV, y permite dar órdenes al dron.

Estación Terrestre - Mission planner

Mission planner es una aplicación de estación terrestre para el UAV, que se utiliza para configurar el control dinámico del dron, pudiendo calibrar sus sensores. Además, se puede visualizar en un mapa donde se encuentra, la trayectoria y los waypoints que ha tenido, y poder ordenar al UAV un plan de vuelo. Esta configurado de tal forma, que se puede conectar con el programa creado en python, y poder visualizar desde el Mission Planner lo que va ocurriendo en el código programado.[11]

Fig. 1.9. Captura de pantalla de Mission Planner

CAPÍTULO 2. SISTEMAS DE ATERRIZAJE

2.1. Introducción

2.1.1. Estado del arte

En la aviación, una de las maniobras más complicadas siempre ha sido, junto al despegue, el aterrizaje. Siendo esta maniobra donde han ocurrido más accidentes, por ser un delicado proceso en el que actúa una gran cantidad de energía cinética y potencial. Por ello, se ha querido automatizar esta operación.

En la aviación comercial se creó un instrumento para dar soporte a esta operación, el ILS (Instrument Landing System)[12]. Con este sistema, formado por dos subsistemas independientes, uno guía lateral y otro vertical, a través de radiofrecuencia, permite que el avión pueda aterrizar con precisión.

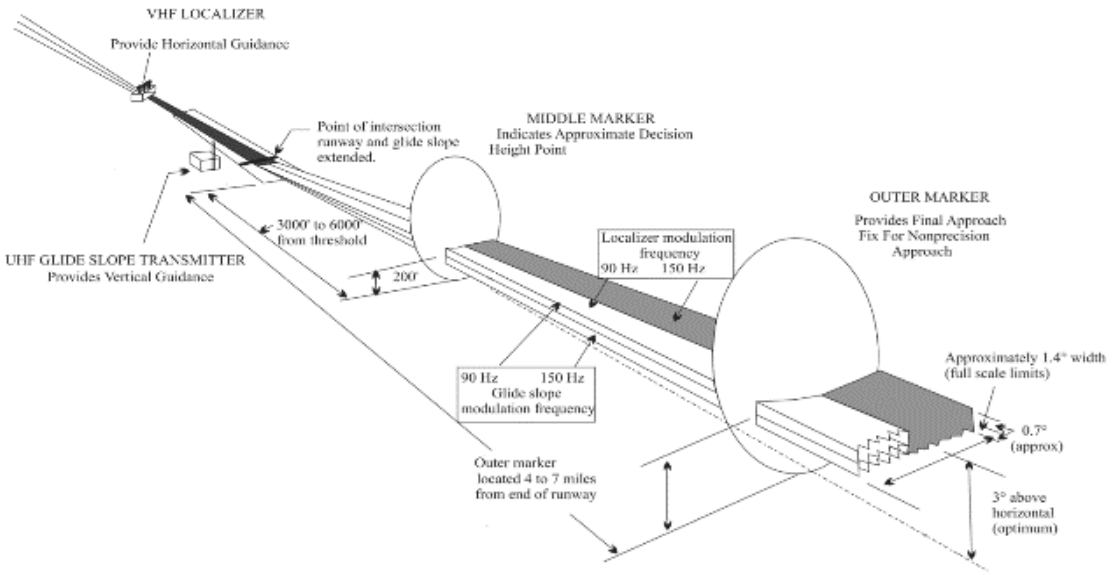

Fig. 2.1. Esquema de ILS

Para un UAV, no existe ningún sistema expandido similar, por lo que normalmente se aterrizan manualmente, en el que el piloto, ya sea visualizándolo directamente o mediante una cámara a bordo, guía al dron hacia la dirección correcta para aterrizar. Siendo un procedimiento bastante complejo en el que el piloto debe tener bastante práctica.

En el ámbito militar, existe desde hace años el UAV de larga duración Global Hawk[13], el cual fue desarrollado por Estados Unidos y es utilizado para vigilancia y reconocimiento. El UAV posee un sistema de aterrizaje autónomo mediante GPS diferencial de alta precisión, en el que obtiene su posición relativa por un radar en tierra. El sistema que utiliza o un posible planteamiento similar queda descartado para nuestro sistema de aterrizaje, ya que éste

necesita un equipamiento y unas radiocomunicaciones que son complejas y el factor más importante, su gran coste que conlleva.

Yéndonos fuera del ámbito militar, existen diversos proyectos que han estudiado la manera para que un UAV pueda aterrizar de forma autónoma. En su mayoría se pueden diferenciar en sistemas cooperativos y no cooperativos, los cuales son explicados en el siguiente apartado[14].

Cabe mencionar que se deben diferenciar los UAV cuadricópteros de los de ala fija, ya que la forma de aterrizar es distinta, y por lo tanto su sistema también. Este estudio está hecho para drones de ala fija, ya que es el tipo de dron que se utiliza en el proyecto Freeda.

2.1.2 Tipos de sistemas

Como se ha comentado en el apartado anterior, existen dos posibles vías para desarrollar el sistema de aterrizaje: cooperativo (ground-based) o no cooperativo (airborne).

Ground based

En un sistema cooperativo, el UAV por si solo no puede obtener toda la información necesaria para realizar la operación, por lo que necesita una ayuda externa. Debe recibir información adicional desde tierra, desde donde debe haber una comunicación con el dron, ya sea unidireccional, en la que el dron solamente recibe la información, o bidireccional, en la que el dron también envía información. En el caso de la comunicación bidireccional, no solamente tendremos que desarrollar el sistema dentro del dron, sino que también tendremos el sistema de la base terrestre y la comunicación que habrá entre ellos.

Resumiendo, el objetivo es que se obtenga la posición de la red donde aterrizará el dron desde una base terrestre y que esta información se envíe de forma correcta al UAV. De esta forma, debería desarrollarse tanto el sistema que se lleva a bordo como el sistema de la base terrestre.

Para ello primero de todo se deberá de partir de una computadora que estará a bordo del barco, junto donde se encuentre la red. A esta computadora se le incluirá un GPS para obtener las coordenadas. Por último, tendrá que tener un sistema de telemetría para comunicarse con el dron para enviarle la posición y con esta información pueda realizar la trayectoria para poder aterrizar.

Airborne

Por otro lado, está el sistema no cooperativo, en el que el dron por si solo es capaz de obtener toda la información necesaria para poder realizar correctamente el aterrizaje, detectando la red y dirigiéndose hacia ella.

En el sistema airborne, no será necesario tener una estación en tierra, por lo que únicamente con los componentes que lleva a bordo y los algoritmos desarrollados obtendrá la posición de aterrizaje.

Con este sistema el objetivo sería obtener la posición haciendo uso de la cámara. El dron, mediante un algoritmo de detección, detectará en cada fotograma que analice el contorno y el centro de la red, devolviendo un vector desde el centro de la imagen hacia el centro de la red. De este modo, sabremos si se encuentra arriba, abajo, derecha o izquierda de donde esté apuntando la cámara.

El objetivo inicial de Freeda para el aterrizaje, es el sistema que se utilizase fuese el airborne, por lo que los objetivos principales de esta memoria, se basaran en los requisitos necesarios para este sistema.

2.1.3 Requisitos y comparación de sistemas

Tanto el sistema Airborne como el Ground Based están pensados para cumplir el mismo objetivo, obtener un mecanismo para dirigirse y aterrizar en la red.

Para elegir qué sistema desarrollar, se ha de tener en cuenta que será para un proyecto con un bajo presupuesto.

Ambos sistemas tienen pros y contras que hace que no exista un sistema ideal, pero a la hora de tomar una decisión hay que estudiar cuál de ellos se adapta mejor al proyecto Freeda, teniendo en cuenta los recursos que posee.

Primero de todo hay que saber qué características y especificaciones técnicas tienen que cumplir cada sistema.

Requisitos sistema airborne

- Algoritmo capaz de, a partir de un fotograma, detectar la red y dar un output de un vector con origen centro del fotograma y dirección hacia el centro de la red.

- Cámara con buena calidad para que pueda detectarse la red.

- Algoritmo para obtener vector con origen en la posición del dron y dirección hacia la red.

- Gimbal capaz de moverse correctamente, saber su orientación respecto al dron y con una velocidad de reacción adecuada al tiempo de duración del aterrizaje.

Requisitos sistema Ground Based

- GPS capaz de obtener la posición de la red.
- Telemetría capaz de comunicarse con el dron a la distancia adecuada antes de iniciar el aterrizaje.
- Algoritmo que a partir de la posición del dron y la de posición de la red dibuje la trayectoria que debe seguir el dron.

Tabla 2.1. Pros y contras de los sistemas de aterrizaje.

	AIRBORNE	GROUND BASED
PROS	- No necesita una estación en tierra.	- Se obtiene la posición de la red (coordenadas).
CONTRAS	- Mayor procesamiento de datos en la computadora a bordo. - No se obtiene la posición de la red, únicamente se obtiene la dirección hacia ella.	- Necesidad de más recursos. - Aumentan costes - Dependencia del funcionamiento externo al dron.

Como la idea es que el aterrizaje fuese utilizando el sistema airborne, debemos asegurar que se cumplen sus requisitos para poder implementarlo.

"Algoritmo capaz de, a partir de un fotograma, detectar la red y dar un output de un vector con origen centro del fotograma y dirección hacia el centro de la red."

El requisito de poder detectar la red con la cámara y saber en qué posición del fotograma se encuentra su centro, se desarrolló previamente a esta memoria. El algoritmo lo que consigue es devolver un vector con origen en el centro del fotograma, hacia el centro de la red. De esta forma podemos saber hacia donde hay que orientar el gimbal, para que el centro del fotograma esté centrado en el centro de la red, y por lo tanto estar apuntando hacia ella.

Fig. 2.2. Fotograma de la prueba del algoritmo para detectar red.

"Cámara con buena calidad para que pueda detectarse la red."

La elección de la cámara que se usa en el proyecto y que se hace mención en el primer capítulo, fue previa a esta memoria, por lo que partiremos con la idea que en el momento de la elección tuvieron en cuenta este requisito y concluyeron que lo cumplía. Debieron de tener en cuenta este requisito, ya que para la detección de embarcaciones es el mismo requisito.

"Algoritmo para obtener vector con origen en la posición del dron y dirección hacia la red."

El algoritmo para obtener el vector director hacia la red es el primero de los dos objetivos a conseguir de la memoria, y se encuentra explicado y desarrollado en la segunda parte de este capítulo.

"Gimbal capaz de moverse correctamente, saber su orientación respecto al dron y con una velocidad de reacción adecuada al tiempo de duración del aterrizaje."

La configuración del gimbal para el proyecto, así como toda la programación a desarrollar para poder tener el control sobre él, es el segundo objetivo de esta memoria y se encuentra explicado en el capítulo 3.

2.2. Desarrollo algoritmo de obtención vector director hacia red

2.2.1. Introducción

El objetivo de este apartado es obtener un vector con origen en el dron y dirección hacia la red, para que se pueda ordenar a la controladora Pixhawk que el dron tenga como vector velocidad, el vector obtenido.

Para obtener este vector, será necesario saber como esta orientado el dron respecto al espacio. Este factor se consigue gracias a que la controladora del dron tiene un sistema inercial y que, mediante calibración previa, se puede obtener durante el vuelo los ángulos pitch, roll y yaw, que son los que definen como esta orientado el UAV respecto al espacio.

El objetivo sería primero que el gimbal estuviese apuntando hacia la red, por lo que este vector fuese idealmente 0, y después tener un algoritmo capaz de obtener el vector director gimbal-red y hacerlo coincidir con el vector velocidad del dron, de este modo el dron se estaría dirigiendo hacia la red.

Como se puede ver en la figura 2.3, en que la línea amarilla es el vector gimbal, que apunta directamente hacia la red, y la línea verde x_b es el vector velocidad del dron. Para que el UAV se dirija correctamente, la línea verde x_b debería igualarse a la amarilla.

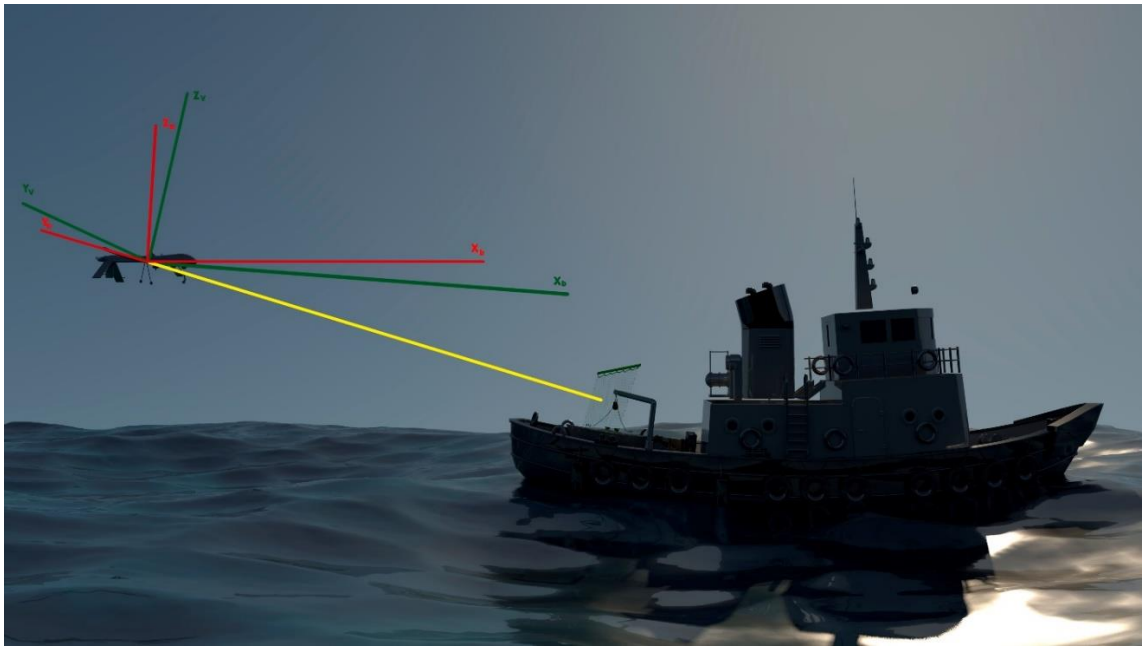

Fig. 2.3. Vista perfil sistema airborne

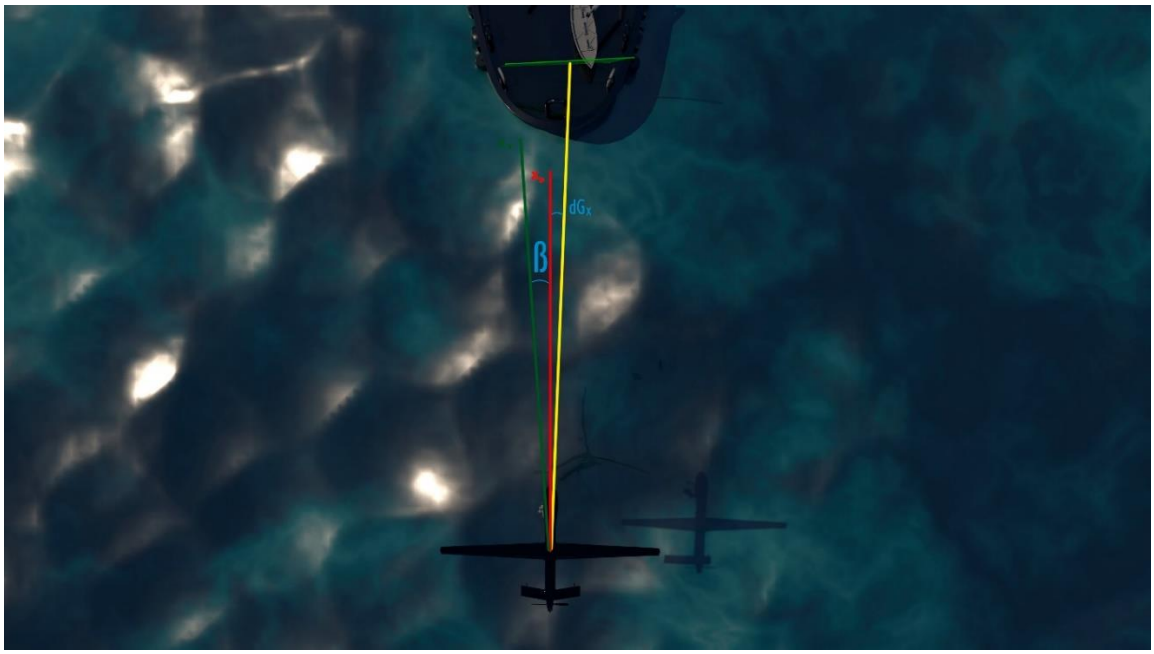

Fig. 2.4. Vista en planta sistema airborne

Para que este sistema funcione correctamente, se debe tener un control total del gimbal, capaz de saber en todo momento como está orientado respecto al dron, poder darle ordenes de movimiento según nos convenga y que tenga una respuesta lo suficientemente rápida como para que actúe a tiempo.

En el capítulo 3 se explica detalladamente como es el gimbal del proyecto Freeda, modelizándolo para tener un control sobre él y estudiando la viabilidad para que pueda ser usado para el aterrizaje.

Suponiendo que tuviésemos un gimbal que cumpliese con los requisitos necesarios, capaz de estar apuntando directamente a la red, llegaríamos al punto de partida del objetivo del apartado. Para ello debemos realizar una rotación hasta hacer coincidir los ejes que tengamos conocidos con el eje dirección hacia la red.

La información que tenemos es:

- Ángulos DG_y y DG_x: son los ángulos que forman los ejes cuerpo del dron con los ejes cuerpo del gimbal. En la figura 2.4 se puede observar de forma visual el ángulo DG_x, es cuál es el ángulo que forma el eje x del cuerpo dron o de los ejes horizonte local (línea roja), con el eje x del cuerpo gimbal (línea amarilla). Estos ángulos se obtienen gracias a la clase gimbal creada en el lenguaje de programación python y explicada en el capítulo 3.

- Velocidad: la velocidad tanto en módulo como en vector respecto ejes horizonte local, se pueden obtener de la controladora del UAV, en el caso del UAV Freeda, de la Pixhawk 4.

- Pitch, roll y yaw: ángulos de Euler que describen la orientación del sistema de ejes cuerpo respecto al sistema de ejes horizonte local. Estos tres ángulos se interpretan físicamente de la siguiente manera: el ángulo pitch es el ángulo existente entre el eje x_b del avión y su proyección sobre el plano horizontal; el ángulo yaw, es el ángulo existente entre la dirección de referencia x_h (por ejemplo, el norte) y la proyección de x_b sobre el plano horizontal; y el ángulo roll, es el ángulo existente entre el eje y_b y la intersección del plano y_b-z_b con el plano horizontal[15]. Es decir, estos ángulos nos muestran cómo está orientado el UAV respecto el espacio terrestre. Estos tres ángulos se obtienen de la controladora del UAV.

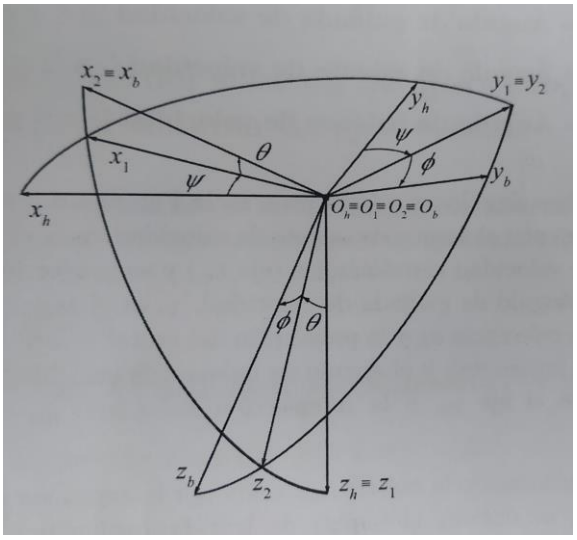

Fig. 2.5. Orientación ejes cuerpo (b) con ejes horizonte (h)

2.2.2. Planteamiento teórico

Nuestro objetivo es obtener un vector, el cual será el vector velocidad en dirección a la red, a partir de los inputs nombrados anteriormente. Para obtener este output, se puede hacer uso de los ángulos de Euler, los cuales son tres rotaciones en un orden específico (primero alrededor del eje z_1, después del eje y_1 y por último del eje x_1).

Componiendo las tres matrices de cada rotación individual, obtenemos la matriz de rotación global L_{ba}, en la que, si a esta matriz se le multiplica un vector en sistemas de ejes A, se obtiene este mismo vector es sistemas de ejes B.[15]

$$L_{ba} = \begin{pmatrix} c\delta_2\, c\delta_3 & c\delta_2\ s\delta_3 & -\sin\delta_2 \\ s\,\delta_1 s\,\delta_2 c\,\delta_3 - c\,\delta_1 s\ \delta_3 & s\,\delta_1 s\,\delta_2 s\,\delta_3 + c\,\delta_1 c\,\delta_3 & s\,\delta_1 c\,\delta_2 \\ c\delta_1 s\,\delta_2 c\,\delta_3 + s\,\delta_1 s\ \delta_3 & c\,\delta_1 s\,\delta_2 s\,\delta_3 - s\,\delta_1 c\,\delta_3 & c\,\delta_1 c\,\delta_2 \end{pmatrix} \quad \textbf{(2.1)}$$

$$\vec{V}_b = L_{ba}\vec{V}_a \quad \textbf{(2.2)}$$

En la matriz L_{ba}, los ángulos δ_3 δ_2 δ_1, son los ángulos de rotación respecto a los ejes z_a, y_a y x_a, respectivamente.

La matriz de transformación entre dos sistemas L_{ba}, es ortogonal, por lo tanto es igual a su traspuesta. Es decir, si el valor que quiero obtener en V_a, y tengo V_b, se puede obtener con la traspuesta de L_{ba}.[15]

$$L_{ab} = L_{ba}^{-1} \rightarrow \vec{V}_a = L_{ab}\vec{V}_b \quad \textbf{(2.3)}$$

En nuestro caso, los sistemas de referencia que tenemos y que se pueden ver en la figura 2.3 son los siguientes:

- Sistema de ejes gimbal (g): el eje x_g está dirigido según el vector donde enfoca la cámara (red), el eje z_g está situado en el plano de simetría perpendicular a x_g y orientado hacia abajo y el eje y_g es ortogonal a los dos anteriores. Nuestro objetivo es que este sistema de ejes sea el sistema de ejes viento (w), en el que el eje x_w está dirigido por el vector velocidad aerodinámica.

- Sistema de ejes cuerpo (b): x_b contenido en el plano de simetría del UAV, y_b perpendicular al plano de simetría y apuntando hacia el ala derecha y x_b es ortogonal a y_b y x_b.

- Sistema de ejes horizonte local (h): los ejes de este sistema son paralelos a los ejes tierra desde el UAV, donde x_t apunta hacia el norte, z_t hacia el centro de la tierra y y_t es ortogonal a estos dos. Con este sistema la controladora del UAV se orienta respecto al espacio, por lo que deberemos obtener el vector velocidad en este sistema.

No tenemos unos ángulos directos que relacionen el sistema de ejes gimbal (sistema donde tenemos situado el vector velocidad que queremos tener) con el sistema de ejes horizonte local (sistema donde queremos obtener el vector velocidad que queremos). Aun así, el sistema de ejes cuerpo relaciona estos dos sistemas, ya que tenemos ángulos que relacionan el sistema de ejes gimbal con el de ejes cuerpo y otros ángulos que relacionan el sistema de ejes cuerpo con el de ejes horizonte local.[15]

Orientación del sistema ejes gimbal respecto al sistema de ejes cuerpo:

- $\delta_3 \equiv -DG_y$
- $\delta_2 \equiv DG_x$
- $\delta_1 \equiv 0$

Orientación del sistema ejes cuerpo respecto al sistema de ejes horizonte local:

- $\delta_3 \equiv$ yaw
- $\delta_2 \equiv$ pitch

- δ_1 ≡ roll

En los cálculos partimos del resultado que queremos obtener, en el que el vector velocidad se encuentre en el eje x del sistema de ejes gimbal. Por lo que:

$$\vec{v}_w = \vec{v}_g = (|v|, \quad 0, \quad 0) \tag{2.4}$$

Este vector velocidad lo queremos representar en el sistema de ejes cuerpo, por lo que debemos calcular la matriz transformación del sistema ejes gimbal al sistema de ejes cuerpo (L_{bg})

$$L_{bg} = \begin{pmatrix} \cos(\mathrm{DGx})\cos(\mathrm{DGy}) & -\cos(\mathrm{DGx})\sin(\mathrm{DGy}) & -\sin(\mathrm{DGx}) \\ \sin(\mathrm{DGy}) & \cos(\mathrm{DGy}) & 0 \\ \sin(\mathrm{DGx})\cos(\mathrm{DGy}) & \sin(\mathrm{DGx}) & \cos(\mathrm{DGx}) \end{pmatrix} \tag{2.5}$$

Una vez teniendo la matriz L_{bg}, podemos obtener el vector velocidad en el sistema de ejes cuerpo de la siguiente forma:

$$\vec{V}_b = L_{bg}\vec{V}_g = \begin{pmatrix} \cos(\mathrm{DGx})\cos(\mathrm{DGy}) & -\cos(\mathrm{DGx})\sin(\mathrm{DGy}) & -\sin(\mathrm{DGx}) \\ \sin(\mathrm{DGy}) & \cos(\mathrm{DGy}) & 0 \\ \sin(\mathrm{DGx})\cos(\mathrm{DGy}) & \sin(\mathrm{DGx}) & \cos(\mathrm{DGx}) \end{pmatrix} \begin{pmatrix} |v| \\ 0 \\ 0 \end{pmatrix} \tag{2.6}$$

Teniendo la velocidad en el sistema de ejes cuerpo y necesitándola en el sistema de ejes horizonte local, calculamos la matriz transformación L_{hb}, la cual es la traspuesta a la matriz L_{bh}:

$$L_{bh} = \begin{pmatrix} \cos(pitch)\cos(yaw) & \cos(\mathrm{pitch})\sin(yaw) & -\sin(pitch) \\ \sin(roll)\sin(pitch)\cos(yaw) - \cos(roll)\sin(yaw) & \sin(roll)\sin(pitch)\sin(yaw) + \cos(roll)\cos(yaw) & \sin(roll)\cos(pitch) \\ \cos(roll)\sin(pitch)\cos(yaw) + \sin(roll)\sin(yaw) & \cos(roll)\sin(pitch)\sin(yaw) - \sin(roll)\cos(yaw) & \cos(roll)\cos(pitch) \end{pmatrix} \tag{2.7}$$

$$L_{hb} = {L_{bh}}^{-1} \tag{2.8}$$

Por último, multiplicamos a la matriz el vector velocidad en ejes cuerpo:

$$\vec{V}_h = L_{hb}\vec{V}_b \tag{2.9}$$

Ya tenemos el vector velocidad apuntando hacia la red, en el sistema de ejes horizonte local, con el cual deberemos imponer al dron que realice.

2.2.3. Implementación del algoritmo

Para que el UAV realice estos cálculos, creamos una función en código Python, la cual recibirá los inputs DGy, DGx, pitch, roll, yaw y velocidad y devolverá el vector velocidad en ejes horizonte local. Los ángulos los recibirá en grados, mientras que los cálculos se realizan en radianes, por lo que previamente se debe hacer unca conversión para poder calcularse de forma correcta. Para hacer cálculo con matrices se hace uso de la librería Numpy, y para obtener el valor de π para hacer la conversión de grados a radianes se hace uso de la librería Math.

```
8   def deg2rad(angulo):
9       a=angulo*(math.pi/180)
10      return a
```

Fig. 2.6. Función para convertir grados a radianes

```
51  def GetCorrectVectorVelocity(DGy,DGx,pitch,roll,yaw,velocity):
52
53      DGy=deg2rad(DGy)
54      DGx=deg2rad(DGx)
55      pitch=deg2rad(pitch)
56      roll=deg2rad(roll)
57      yaw=deg2rad(yaw)
58
59      Vg=[velocity,0,0]
60
61      Lbh = np.array([[math.cos(roll) * math.cos(yaw), math.cos(roll) * math.sin(yaw), -math.sin(roll)],
62                      [math.sin(pitch) * math.sin(roll) * math.cos(yaw) - math.cos(pitch) * math.sin(yaw),
63                       math.sin(pitch) * math.sin(roll) * math.sin(yaw) + math.cos(pitch) * math.cos(yaw),
64                       math.sin(pitch) * math.cos(roll)],
65                      [math.cos(pitch) * math.sin(roll) * math.cos(yaw) + math.sin(pitch) * math.sin(yaw),
66                       math.cos(pitch) * math.sin(roll) * math.sin(yaw) - math.sin(pitch) * math.cos(yaw),
67                       math.cos(pitch) * math.cos(roll)]])
68
69      Lbg=np.array([[math.cos(DGy)*math.cos(DGx),-math.cos(DGy)*math.sin(DGx),-math.sin(DGy)],
70                    [math.sin(DGx),math.cos(DGx),0],
71                    [math.sin(DGy)*math.cos(DGx),-math.sin(DGy)*math.sin(DGx),math.cos(DGy)]])
72
73      Lhb=np.matrix(Lbh).transpose()
74
75      Vb = np.dot(Lbg, Vg)
76
77      Vh= np.dot(Lhb, Vb)
78
79      return Vh
```

Fig. 2.7. Función para obtener vector velocidad en ejes horizonte

Hacemos una prueba del código, en el que le damos valores sencillos en los que deberíamos intuir el resultado. Por ejemplo, dándole valores nulos a los ángulos pitch, yaw y roll, para que el dron esté orientado paralelo a tierra, y un ángulo de 0° a DGx y de -45° a DGy. De esta forma el gimbal está apuntando en la misma dirección que el dron, con una desviación de 45 grados hacia abajo, es decir, el resultado tendrá que ser 0 para el eje Y ya que no hay ninguna desviación en el plano XZ, y el resultado para el eje Z tendría que ser igual que en el eje X, ya que son 45 y está apuntando hacia abajo. El resultado da el esperado.

```
83  VELOCIDAD=GetCorrectVectorVelocity(-45,0,0,0,0,17)
84  print(VELOCIDAD)

Run  PruebaCorrectVelocityVector

C:\Users\raulp\Python3\python.exe C:/Users/raulp/De
[[ 12.02081528   0.         -12.02081528]]

Process finished with exit code 0
```

Fig. 2.8. Test de la función creada para obtener el vector velocidad

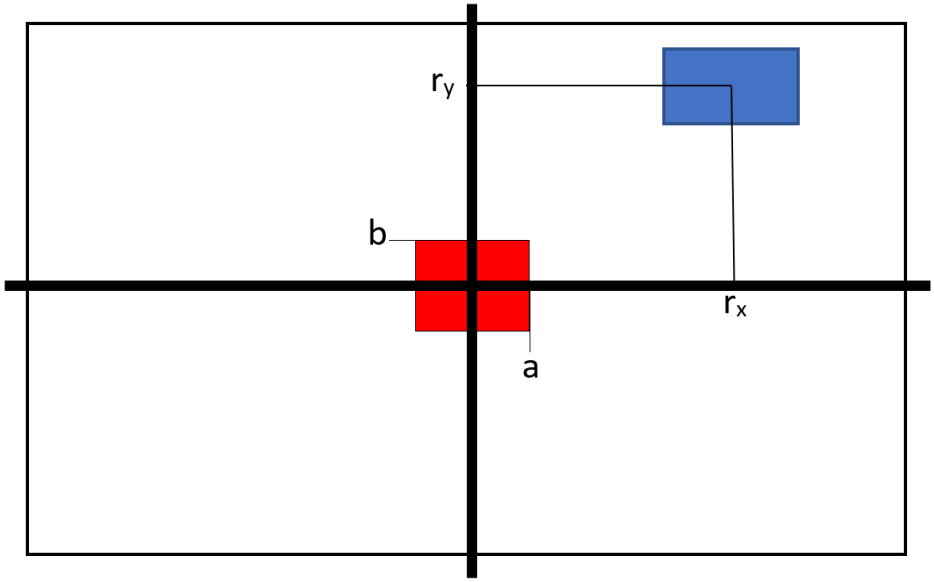

Fig. 2.9. Esquema cámara con detección de red

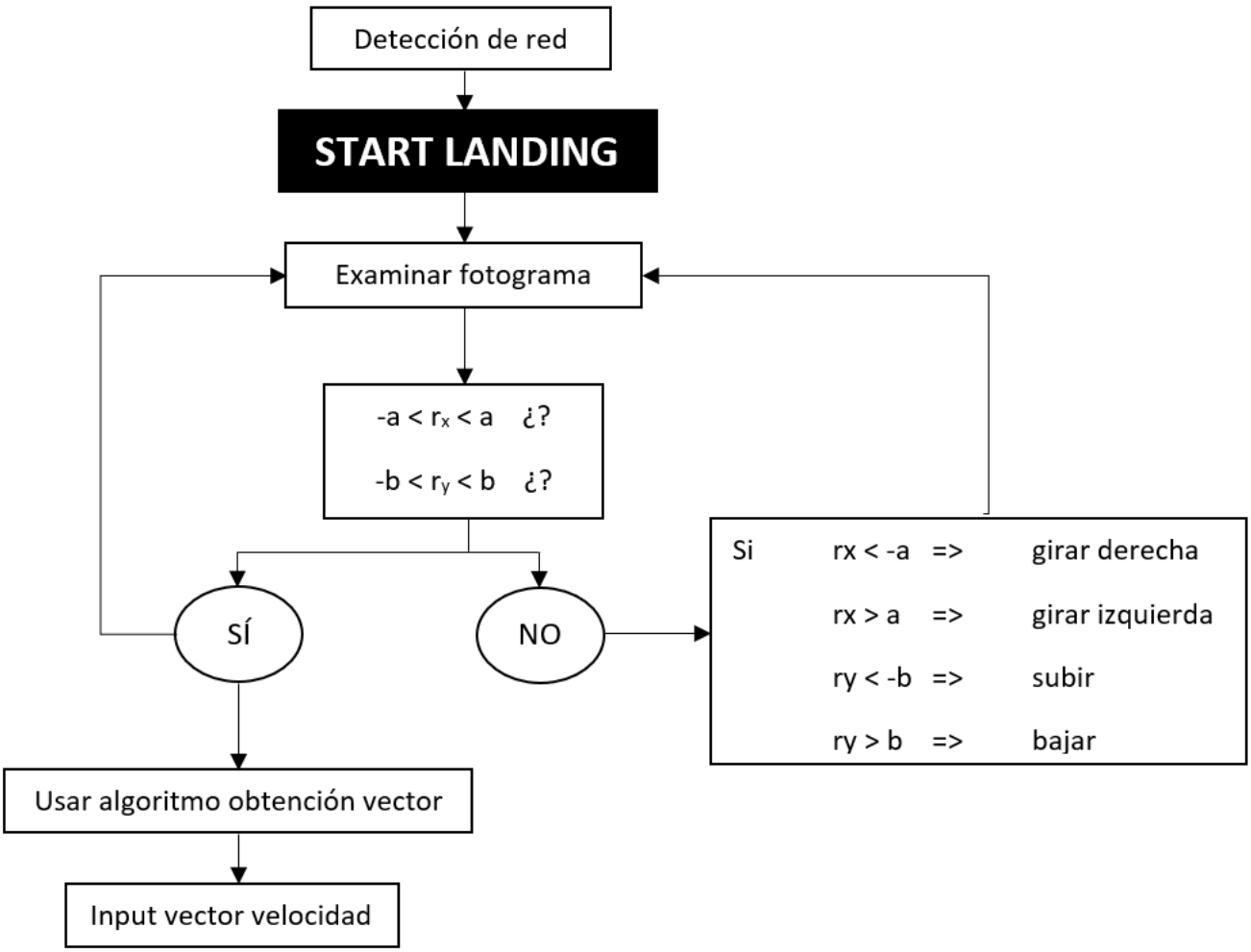

Fig. 2.10. Diagrama aterrizaje tipo onboard

En la figura 2.9 se muestra el esquema de la imagen que obtiene la cámara con la red detectada, en el que hay unos ejes centrados en la imagen, el vector dirección hacia la red, el cual se obtiene con visión por computador coordenadas (rx, ry) y unos márgenes (a en ejes x y b en ejes y) para las cuales se aceptaría seguir con el aterrizaje

CAPÍTULO 3. MODELIZACIÓN DEL GIMBAL

3.1. Introducción

Para el sistema Airborne, el objetivo del Gimbal es que esté constantemente apuntando hacia la red, y saber cómo está orientado respecto al UAV, para poder tener una estimación de dónde se encuentra la red y poder dirigirse hacia ella. Para ello hemos de saber en todo momento que ángulo tiene el gimbal respecto el UAV y ser capaz de aumentar o disminuir este ángulo para poder apuntar hacia la red.

3.2. Diseño del Gimbal

3.2.1. Componentes

El Gimbal consta de 3 ejes de movimiento, por lo tanto, tiene 3 servos digitales, uno para cada eje. Un servo es un dispositivo pequeño que tiene un eje de salida. Este eje se puede colocar en posiciones angulares específicas enviando al servo una señal codificada. Mientras exista la señal codificada en la línea de entrada, el servo mantendrá la posición angular del eje. A medida que cambia la señal codificada, cambia la posición angular del eje. [16]

Los servos utilizados son Corona DS-929MG. Tiene las siguientes características:[17]

- Tamaño: 22.5X11.5X24.6 mm (0.88 "x0.45" x0.96 ")
- Peso: 13.6g
- Voltaje de funcionamiento: 4.8V ~ 6.0V
- Corriente de trabajo: 200mA / 60o
- Tarifa de trabajo: 0.11sec / 60o
- Par de torsión: 2.0kg.cm
- Torque Estático: 1.8kg
- Ancho de banda muerto: ≤3uSec
- plan de trabajo: 40o / pulso unilateral 400us
- Tipo de motor: motor de cepillo
- Tipo de potenciómetro: 2 deslizadores / accionamiento directo
- Rodamientos de alta precisión: MR85
- Material del engranaje: Metal
- Longitud del cable: 215 mm
- Temperatura de funcionamiento: -20 ° C a + 60 ° C

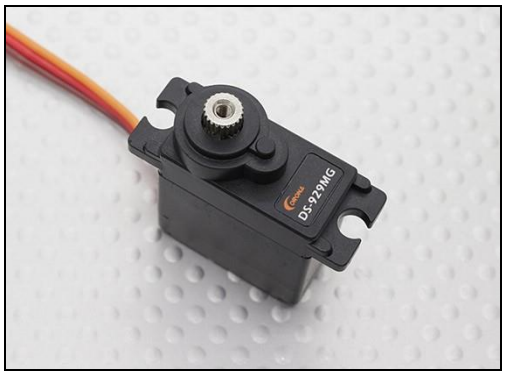

Fig. 3.1. Servo DS-929MG

Estos servos son alimentados por una batería (Tipo LI-PO) de la cual se le limita su voltaje mediante un BEC (*Battery Eliminator Circuit*), para que reciba un voltaje dentro de su rango.

3.2.2. Conexión

Los tres servos del Gimbal se conectan a los conectores de la parte 1 de la figura 3.2. De estos conectores se unen por separado los cables del mismo tipo.

- Los tres cables marrones (de tierra) se unen y de ellos salen dos, uno para tierra de la Raspberry y otro se unirá al cable tierra del BEC. La parte 4 es el cable que se conecta a tierra de la Raspberry.

- Los cables rojos (alimentación) se unen y de ellos sale uno para unirse al del BEC.

- Los cables amarillos (señal) se conectan a sus correspondientes GPIOs de la raspberry (Parte 2).

La parte 3 es el conector del BEC con la batería.

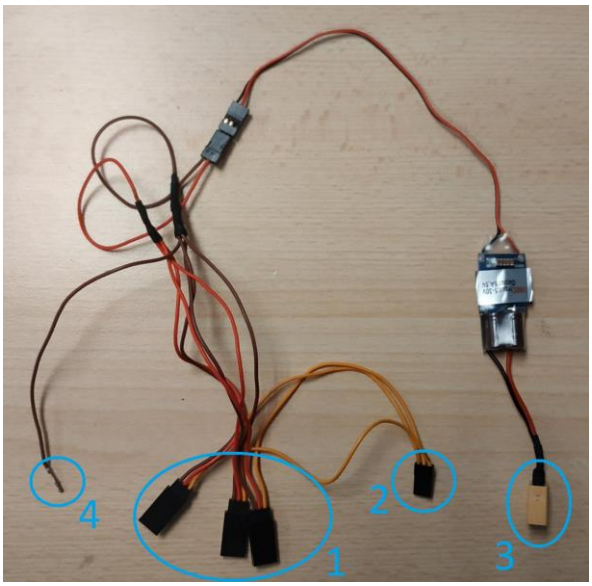

Fig. 3.2. BEC adaptado para Gimbal y Raspberry

3.3. Creación de clase

Para poder saber en todo momento la orientación del Gimbal y poder darle órdenes para su movimiento, se ha creado la clase GIMBAL. Se ha realizado en lenguaje Python, ya que todo el proyecto Fredda consta de este lenguaje.

Una clase se caracteriza tanto de atributos (elementos que definen a la clase en sí) y de métodos (funciones para obtener, asignar y manipular los atributos).

Los atributos se pueden distinguir en tres bloques: conexión Raspberry, estado actual y calibración.

- Conexión raspberry: son cuatro atributos, uno es la función ".pi" de la librería Pigpio y sirve para detectar y conectarse a la Raspberry. Los otros tres son para seleccionar el pin que se va a usar en cada servo.

- Estado actual: son seis y se pueden subdividir en dos tipos. El primer tipo nos indica el pulso del servo y el segundo nos indica el ángulo que forman el UAV y el eje del servo correspondiente.

- Calibración: son doce, y corresponden a los pulsos y ángulos mínimos y máximos. Son valores fijos que se les fueron asignados en el momento de calibrar el Gimbal.

Las funciones creadas para el control del gimbal son las mismas para cada eje y sirven para imponer un ángulo determinado, obtener el ángulo que en el cual está colocado y aumentar o disminuir esté ángulo. En todas las funciones sin incluir a la de obtener los ángulos, se les compara el ángulo que se le quiere modificar para ver si se encuentran dentro de su rango de movimiento. En el caso que sea superior se le asigna el máximo y si es inferior el mínimo.

Cabe mencionar que los servos no entienden de ángulos, sino de ciclos de trabajo ("*duty cycle*"), aunque los ciclos, con las calibraciones y conversiones adecuadas, se pueden convertir en ángulos. Por lo que tanto los inputs como los outputs son ángulos, aunque los procesos intermedios, se deben realizar con ciclos de trabajo, haciendo conversiones antes y después.

```
17      def dc(angle):
18          angle = float(angle)
19          dutyCycleMs = 1000 * angle / 90 + 500
20
21          return round(dutyCycleMs)
```

Fig. 3.3. Función para convertir ángulo en ciclo de trabajo

```
10      class GIMBAL():
11
12      #atributos
13          def __init__(self,servoPan,servoTilt,servoRoll):...
49
50      #funciones para conectar y desconectar el gimbal
51          def initialize_servos(self):...
57          def disconnect_servos(self):...
61
62      #funciones para el pan angle
63          def set_pan_angle(self,angulo):...
78          def get_pan_angle(self):...
80          def increase_pan_angle(self,angulo):...
97          def decrease_pan_angle(self, angulo):...
114
115     #funciones para el tilt angle
116         def set_tilt_angle(angulo,self):...
131         def get_tilt_angle(self):...
133         def increase_tilt_angle(self, angulo):...
150         def decrease_tilt_angle(self, angulo):...
167
168     #funciones para el roll angle
169         def set_roll_angle(self, rollAngle):...
183         def get_roll_angle(self):...
185         def increase_roll_angle(self, angulo):...
201         def decrease_roll_angle(self, angulo):...
217
```

Fig. 3.4. Clase GIMBAL

```
12      #atributos
13          def __init__(self,servoPan,servoTilt,servoRoll):
14
15              self.raspi = pigpio.pi()
16
17              self.servoPan = servoPan #13
18              self.servoTilt = servoTilt #19
19              self.servoRoll = servoRoll #26
20
21              self.raspi.set_servo_pulsewidth(self.servoPan, 1500)
22              self.raspi.set_servo_pulsewidth(self.servoTilt, 1500)
23              self.raspi.set_servo_pulsewidth(self.servoRoll, 1500)
24
25              self.pan_angle=0
26              self.tilt_angle=-90
27              self.roll_angle=0
28
29              self.servos = [self.servoPan, self.servoTilt, self.servoRoll]
30
31              time.sleep(5)
32
33              #CALIBRATION
34              self.pan_DCmin=550
35              self.tilt_DCmin = 550
36              self.roll_DCmin = 650
37
38              self.pan_DCmax=2350
39              self.tilt_DCmax = 1810
40              self.roll_DCmax = 2450
41
42              self.pan_angle_min= -90
43              self.tilt_angle_min = 0
44              self.roll_angle_min = -90
45
46              self.pan_angle_max= 90
47              self.tilt_angle_max = 90
48              self.roll_angle_max = 90
```

Fig. 3.5. Atributos de la clase GIMBAL

```
63      def set_pan_angle(self,angulo):  # <-- angle in degrees
64          self.pan_angle=angulo
65          dcMinPan = self.pan_DCmin
66          dcMaxPan = self.pan_DCmax
67          dcPan = dc(angulo)
68
69          if dcPan > dcMaxPan:
70              dcPan = dcMaxPan
71              self.pan_angle=self.pan_angle_max
72          elif dcPan < dcMinPan:
73              dcPan = dcMinPan
74              self.pan_angle=self.pan_angle_min
75          else:
76              self.pan_angle = angulo
77          self.raspi.set_servo_pulsewidth(self.servoPan, dcPan)
```

Fig. 3.6. Función para asignar ángulo

```
80      def get_pan_angle(self):
81          return self.pan_angle
```

Fig. 3.7. Función para obtener ángulo

```
85      def increase_pan_angle(self,angulo):
86          IncreaseDcPan=dc(angulo)
87          dcPan=self.raspi.get_servo_pulsewidth(self.servoPan)
88          dcPan=dcPan+IncreaseDcPan
89
90          dcMinPan = self.pan_DCmin
91          dcMaxPan = self.pan_DCmax
92
93          if dcPan > dcMaxPan:
94              dcPan = dcMaxPan
95              self.pan_angle=self.pan_angle_max
96          elif dcPan < dcMinPan:
97              dcPan = dcMinPan
98              self.pan_angle=self.pan_angle_min
99          else:
100             self.pan_angle = angulo
101         self.raspi.set_servo_pulsewidth(self.servoPan, dcPan)
```

Fig. 3.8. Función para incrementar ángulo

```
105     def decrease_pan_angle(self, angulo):
106         DecreaseDcPan = dc(angulo)
107         dcPan = self.raspi.get_servo_pulsewidth(self.servoPan)
108         dcPan = dcPan - DecreaseDcPan
109 
110         dcMinPan = self.pan_DCmin
111         dcMaxPan = self.pan_DCmax
112 
113         if dcPan > dcMaxPan:
114             dcPan = dcMaxPan
115             self.pan_angle=self.pan_angle_max
116         elif dcPan < dcMinPan:
117             dcPan = dcMinPan
118             self.pan_angle=self.pan_angle_min
119         else:
120             self.pan_angle = angulo
121         self.raspi.set_servo_pulsewidth(self.servoPan, dcPan)
```

Fig. 3.9. Función para disminuir ángulo

3.4. Validación del código

Una vez tenemos la clase GIMBAL, realizamos un código para comprobar que funcione correctamente el movimiento del Gimbal:

- Que los componentes estén bien conectados.
- Ordenes efectuadas correctamente.
- Poder saber en qué ángulo está colocado cada eje.

Para ello se ha creado un código en el que se utiliza la clase y en el que realizamos para cada eje por separado, un movimiento que englobe todo su rango, desde el ángulo más pequeño al más grande y del más grande al más pequeño, mostrando por pantalla en ángulo que tiene en cada momento.

```
12  MyGimbal=GIMBAL(13,19,26)
13 
14  time.sleep(2)
15 
16  MyGimbal.set_pan_angle(MyGimbal.pan_angle_min)
17  time.sleep(2)
18 
19  while(MyGimbal.pan_angle<=MyGimbal.pan_angle_max):
20      MyGimbal.increase_pan_angle(5)
21      print(str(MyGimbal.get_pan_angle()))
22      time.sleep(2)
23 
24  while(MyGimbal.pan_angle>MyGimbal.pan_angle_min):
25      MyGimbal.decrease_pan_angle(5)
26      print(str(MyGimbal.get_pan_angle()))
27      time.sleep(2)
```

Fig. 3.10. Trozo de código del comportamiento del Gimbal

Después de realizar la prueba, se puede ver que el gimbal responde y actúa acorde le dicta el código.

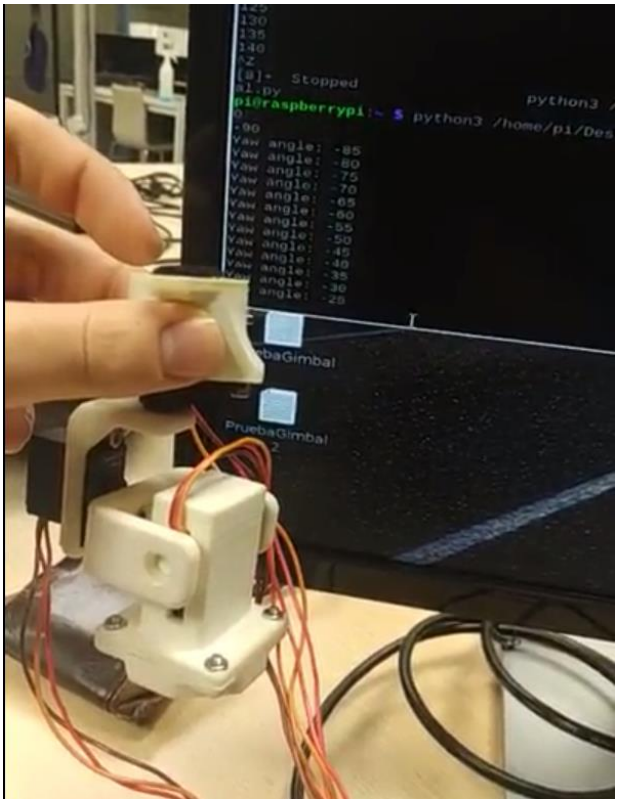

Fig. 3.11. Prueba del funcionamiento del Gimbal

Pese a que el Gimbal reaccionaba correctamente, al conectarse tardaba un poco en reaccionar. Se deberían de hacer pruebas con el UAV, en conjunto con el algoritmo explicado en el capítulo 2, para obtener datos experimentales y llegar a la conclusión de si este sistema es o no viable para Fredda.

Capítulo 4. PRIMER PASO HACIA UN SISTEMA GROUND BASED

4.1. Introducción

Como posible sistema redundante o alternativa, se podría desarrollar en estudios posteriores un sistema cooperativo.

El objetivo de este capítulo es dar un primer paso al desarrollo de un sistema Ground based, en el que se marcará los requisitos nombrados en el capítulo 2 y se detallará los pasos para poder cumplirlos.

La idea principal es que el dron reciba las coordenadas de la red y mediante un algoritmo de control y guiaje, con inputs tanto de las coordenadas de la red y las coordenadas de la posición del UAV, aterrice de forma correcta.

Para ello, hay dos puntos a tener en cuenta que se deberían profundizar más en estudios posteriores, ya que son los que marcan si el sistema cumple los requisitos para poder ser utilizado. Estos dos puntos son:

- Obtener las coordenadas de la red con la suficiente precisión.

- Telemetría con capacidad de alcance al dron y un delay aceptable.

Para la obtención de las coordenadas, se debe buscar e implementar un GPS que cumpla los requisitos.

Para la telemetría, se debe buscar la mejor forma de comunicación entre la red y el UAV, en la que la información que se envíe desde la red tenga alcance hasta el dron y llegue con un delay aceptable para que se pueda hacer uso del algoritmo para el aterrizaje.

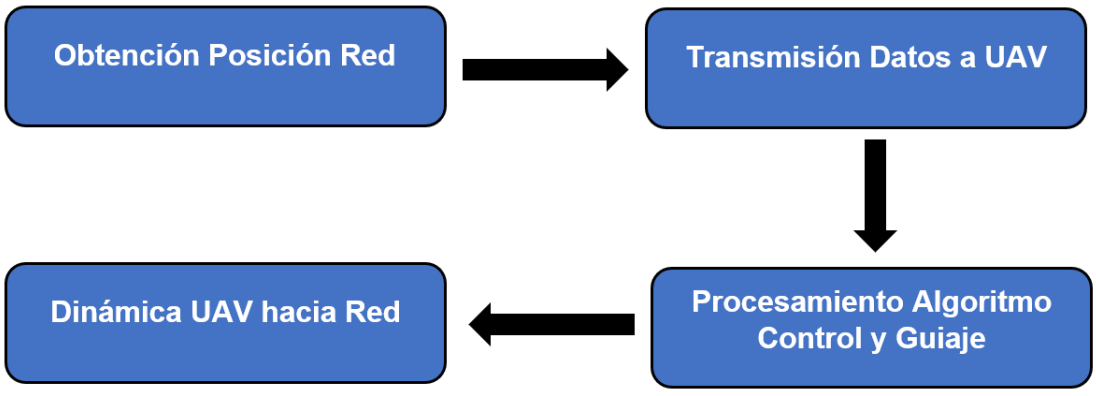

Fig. 4.1. Esquema Procedimiento Sistema Ground Based

4.2. Requisitos

Los siguientes requisitos son los nombrados en el capítulo 2.

"GPS capaz de obtener la posición de la red"

La forma más sencilla y económica para obtener las coordenadas de la red sería colocar en la red una computadora, como podría ser la Raspberry 4 o la Nvidia Nano, con un módulo de GPS, como podría ser el "Adafruit Ultimate GPS HAT", el cual se puede configurar para obtener la posición.[18]

Fig. 4.2. Raspberry y modulo "Adafruit Ultimate GPS HAT"

El problema de este sistema es que la precisión de los GPS no es la suficiente para poder asegurar que el UAV pueda aterrizar. Para ello habría que estudiar alternativas al GPS convencional que cuenten con mayor precisión.

Un caso a estudiar sería el GPS diferencial (DGPS – "Differential Global Positioning System"). El DGPS es una mejora del GPS. Que proporciona una mejor precisión de la ubicación, pasando de una precisión de 15 metros aproximadamente, a 1-3 cm.[19]

Otro caso a estudiar sería el sistema RTK ("Real Time Kinematic"), el cual está basado en el uso de medidas de fase de navegadores con señales GPS, donde una estación de tierra proporciona correcciones en tiempo real.[20][21]

"Telemetría capaz de comunicarse con el dron a la distancia adecuada antes de iniciar el aterrizaje."

El proyecto Freeda, como se explica en el primer capítulo, ya posee un sistema de telemetría, así que para este punto habría que comprobar que se pudiera adaptar al sistema de aterrizaje. Para ello, debería ser capaz de enviar las coordenadas de la red al UAV con la suficiente rapidez y con un delay aceptable.

"Algoritmo que a partir de la posición del dron y la de posición de la red dibuje la trayectoria que debe seguir el dron."

En este punto, se debería desarrollar un algoritmo que a partir de dos inputs (posición del dron y posición de la red) modifique y actualice a tiempo real el path (trayectoria) para que el dron corrija la trayectoria con tal de interceptar la red.[22][23]

La posición del dron es siempre conocida por señal GPS y la de la red le llegaría al dron por los puntos antes nombrados.

El algoritmo de seguimiento debería ser integrado en el sistema para ser capaz de actuar a tiempo real.

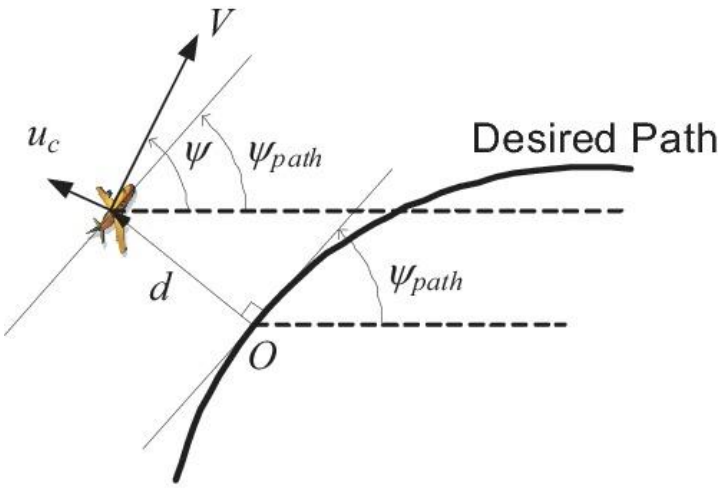

Fig. 4.3. Esquema algoritmo de seguimiento

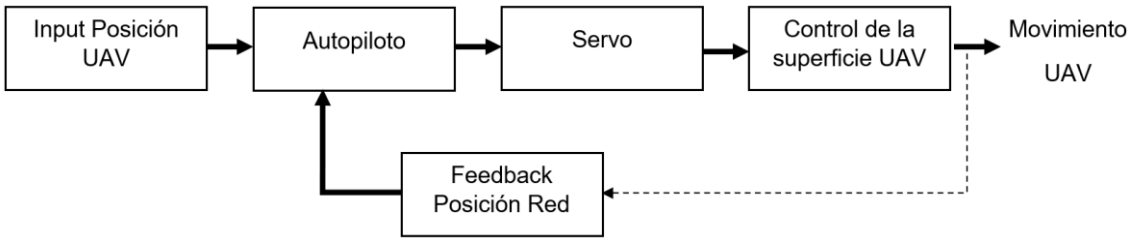

Fig. 4.4. Esquema control automático

CONCLUSIONES

El objetivo principal de este estudio era hacer un análisis y comparación de sistemas de aterrizaje autónomo de un UAV en una embarcación. En él, se ha podido observar que hay dos posibles vías para el desarrollo, y pese a que Fredda tuviera una idea en mente, hay otro posible camino para conseguir el objetivo final de que aterrice el dron en el Open Arms.

Los requisitos a cumplir para el sistema Airborne han sido planteados y se les ha dado, uno a uno, soluciones.

Para el primer objetivo de obtener el vector velocidad hacia la red, se ha desarrollado el algoritmo capaz de conseguirlo.

Por otra parte, el objetivo de tener el total control del mecanismo que mueve la cámara también se ha cumplido. Tenemos el código capaz de obtener el control del gimbal, en el que podemos saber como está orientado respecto al UAV, y a la vez poder moverlo según nos convenga.

Pese haber cumplido los dos objetivos, todavía falta hacer pruebas experimentales para poder concluir si el sistema planteado en el documento cumple o no con lo esperado. Se ha validado en software in the loop, pero queda pendiente para un futuro trabajo de validación realizar pruebas con hardware.

Por no haberse podido llegar a una conclusión definitiva de si el sistema airborne es un sistema viable para Freeda, al final del documento se exponen los requisitos y los pasos a seguir con la segunda posible vía en el desarrollo del aterrizaje.

Los pasos a seguir después de este estudio realizado, sería realizar pruebas con lo desarrollado a lo largo del documento y, por otra parte, desarrollar el sistema Ground based, siguiendo las pautas planteadas en el último capítulo.

BIBLIOGRAFÍA

[1] “Frontex | European Union Agency.” https://frontex.europa.eu/ (accessed Jan. 12, 2020).

[2] “Refugiados en Europa: Italia y España, dos vías de entrada en aumento.” https://eacnur.org/es/actualidad/noticias/emergencias/refugiados-en-europa-italia-y-espana-dos-vias-de-entrada-en-aumento (accessed Jan. 12, 2020).

[3] “Open Arms - Datos.” https://www.openarms.es/es?gclid=Cj0KCQjw6ar4BRDnARIsAITGzlBVz8adaVI7lzjD5f0YmBco4fj1LXoQXGE3s9csV_uyExtEQmgdP9saAvpQEALw_wcB (accessed Jul. 12, 2020).

[4] “Open Arms.” https://www.openarms.es/salvaopenarms/es/ (accessed Feb. 12, 2020).

[5] “Hemav Foundation, proyectos dron para la sociedad.” http://hemavfoundation.com/ (accessed Feb. 18, 2020).

[6] “NVIDIA Jetson Nano.” https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/jetson-nano/ (accessed Mar. 08, 2020).

[7] “PX4 v1.9.0 User Guide.” https://docs.px4.io/v1.9.0/en/flight_controller/pixhawk4.html (accessed Feb. 11, 2020).

[8] “Microhard - pMDDL2450 - Miniature MIMO OEM 2.4 GHz Digital Data Link.” http://www.microhardcorp.com/pMDDL2450.php (accessed Apr. 12, 2020).

[9] “Arducam 18MP AR1820HS Color Camera Module for Jetson Nano - UCTRONICS.” https://www.uctronics.com/arducam-cmos-ar1820hs-1-2-3-inch-18mp-color-camera-module-jetson-nano.html (accessed Jul. 12, 2020).

[10] “Nvidia Jetson Nano: the Raspberry Pi of AI? | Tom’s Hardware.” https://www.tomshardware.com/news/jetson-nano-features-price,38856.html (accessed May 12, 2020).

[11] “Mission Planner Home — Mission Planner documentation.” https://ardupilot.org/planner/ (accessed Mar. 12, 2020).

[12] “Sistema de aterrizaje ILS.”

[13] “Global Hawk.” https://fas.org/irp/program/collect/global_hawk.htm (accessed Apr. 12, 2020).

[14] N. P. Santos, V. Lobo, and A. Bernardino, “Autoland project: Fixed-wing UAV landing on a fast patrol boat using computer vision,” *Ocean. 2019 MTS/IEEE Seattle, Ocean. 2019*, no. Cv, 2019, doi: 10.23919/OCEANS40490.2019.8962869.

[15] Miguel Ángel Gómez Tierno, *Mecánica del Vuelo 2ª edición*, 2ª edición. 2012.

[16] S. Behnke and M. Schreiber, “Digital Position Control for Analog Servos.”

[17] “Corona ds-929mg.” https://www.banggood.com/CORONA-DS-929MG-13_6g-Metal-Gear-Digital-Servo-p-1100755.html?utm_source=google&utm_medium=cpc_ods&utm_campaign=nolan-sds-Broad-fishing-

all&utm_content=nolan&ad_id=437423476960&gclid=Cj0KCQjw6ar4BRDnARIsAITGzlDHrTzLBojUctr2u4N-rEq (accessed May 12, 2020).
[18] “Adafruit Ultimate GPS HAT for Raspberry Pi A+/B+/Pi 2/3/Pi 4 [Mini Kit] ID: 2324 - $44.95 : Adafruit Industries, Unique & fun DIY electronics and kits.” https://www.adafruit.com/product/2324 (accessed Jul. 12, 2020).
[19] “Global Differential GPS.” https://www.gdgps.net/ (accessed Jul. 12, 2020).
[20] Y. Feng and J. Wang, “GPS RTK Performance Characteristics and Analysis,” *J. Glob. Position. Syst.*, vol. 7, no. 1, pp. 1–8, Jun. 2008, doi: 10.5081/jgps.7.1.1.
[21] E. M. Arias, “INTEGRACIÓN DE SISTEMA REAL-TIME KINEMATIC CON UNIDAD DE MEDICIÓN INERCIAL DENTRO DEL SOFTWARE AEROSTACK Pascual Campoy Hriday Bavle.”
[22] M. G. Seo, “Guidance Systems,” no. November, 2019.
[23] D. M.-G. Seo, “Guidance \&{Navigation} for {Autonomous} {Systems} ({GNAS}),” p. 65, 2019.

ANNEXOS

TÍTULO DEL TFG: Diseño e implementación del sistema de aterrizaje autónomo para un UAV

TITULACIÓN: Grado en ingeniería de aeronavegación

AUTOR: Raúl Pedrosa Cabello

DIRECTOR: SERGI TRES MARTÍNEZ

CO-DIRECTOR: ÓSCAR CASAS PIEDRAFITA

FECHA:

ANEXOS

Anexo A. Código obtención vector velocidad

```
import numpy as np
import math

def GetCorrectVectorVelocity(DGy,DGx,pitch,roll,yaw,velocity):

    DGy=deg2rad(DGy)
    DGx=deg2rad(DGx)
    pitch=deg2rad(pitch)
    roll=deg2rad(roll)
    yaw=deg2rad(yaw)

    Vg=[velocity,0,0]

    Lbh = np.array([[math.cos(roll) * math.cos(yaw), math.cos(roll) *
math.sin(yaw), -math.sin(roll)],
                    [math.sin(pitch) * math.sin(roll) * math.cos(yaw)
- math.cos(pitch) * math.sin(yaw),
                     math.sin(pitch) * math.sin(roll) * math.sin(yaw)
+ math.cos(pitch) * math.cos(yaw),
                     math.sin(pitch) * math.cos(roll)],
                    [math.cos(pitch) * math.sin(roll) * math.cos(yaw)
+ math.sin(pitch) * math.sin(yaw),
                     math.cos(pitch) * math.sin(roll) * math.sin(yaw)
- math.sin(pitch) * math.cos(yaw),
                     math.cos(pitch) * math.cos(roll)]])

    Lbg=np.array([[math.cos(DGy)*math.cos(DGx),-
math.cos(DGy)*math.sin(DGx),-math.sin(DGy)],
                  [math.sin(DGx),math.cos(DGx),0],
                  [math.sin(DGy)*math.cos(DGx),-
math.sin(DGy)*math.sin(DGx),math.cos(DGy)]])

    Lhb=np.matrix(Lbh).transpose()

    Vb = np.dot(Lbg, Vg)

    Vh= np.dot(Lhb, Vb)

    return Vh

VELOCIDAD=GetCorrectVectorVelocity(-45,0,0,0,0,17)
print(VELOCIDAD)
```

Anexo B. Código de creación clase Gimbal

```
import RPi.GPIO as GPIO
import time

def dc(angle):
    angle = float(angle)
    dutyCycleMs = 1000 * angle / 90 + 500

    return round(dutyCycleMs)

class GIMBAL():

#atributos
    def __init__(self,servoPan,servoTilt,servoRoll):

        self.raspi = pigpio.pi()

        self.servoPan = servoPan #13
        self.servoTilt = servoTilt #19
        self.servoRoll = servoRoll #26

        self.raspi.set_servo_pulsewidth(self.servoPan, 1500)
        self.raspi.set_servo_pulsewidth(self.servoTilt, 1500)
        self.raspi.set_servo_pulsewidth(self.servoRoll, 1500)

        self.pan_angle=0
        self.tilt_angle=-90
        self.roll_angle=0

        self.servos = [self.servoPan, self.servoTilt, self.servoRoll]

        time.sleep(5)

        #CALIBRATION
        self.pan_DCmin=550
        self.tilt_DCmin = 550
        self.roll_DCmin = 650

        self.pan_DCmax=2350
        self.tilt_DCmax = 1810
        self.roll_DCmax = 2450

        self.pan_angle_min= -90
        self.tilt_angle_min = 0
        self.roll_angle_min = -90

        self.pan_angle_max= 90
        self.tilt_angle_max = 90
        self.roll_angle_max = 90

#funciones para conectar y desconectar el gimbal
    def initialize_servos(self):

        self.raspi.set_servo_pulsewidth(self.servoPan, 1500)
        self.raspi.set_servo_pulsewidth(self.servoTilt, 1500)
        self.raspi.set_servo_pulsewidth(self.servoRoll, 1500)
        time.sleep(5)
    def disconnect_servos(self):
        self.raspi.set_servo_pulsewidth(self.servoPan,0)
```

```
        self.raspi.set_servo_pulsewidth(self.servoTilt,0)
        self.raspi.set_servo_pulsewidth(self.servoRoll,0)

#funciones para el pan angle
    def set_pan_angle(self,angulo):  # <-- angle in degrees
        self.pan_angle=angulo
        dcMinPan = self.pan_DCmin
        dcMaxPan = self.pan_DCmax
        dcPan = dc(angulo)

        if dcPan > dcMaxPan:
            dcPan = dcMaxPan
            self.pan_angle=self.pan_angle_max
        elif dcPan < dcMinPan:
            dcPan = dcMinPan
            self.pan_angle=self.pan_angle_min
        else:
            self.pan_angle = angulo
        self.raspi.set_servo_pulsewidth(self.servoPan, dcPan)


    def get_pan_angle(self):
        return self.pan_angle



    def increase_pan_angle(self,angulo):
        IncreaseDcPan=dc(angulo)
        dcPan=self.raspi.get_servo_pulsewidth(self.servoPan)
        dcPan=dcPan+IncreaseDcPan

        dcMinPan = self.pan_DCmin
        dcMaxPan = self.pan_DCmax

        if dcPan > dcMaxPan:
            dcPan = dcMaxPan
            self.pan_angle=self.pan_angle_max
        elif dcPan < dcMinPan:
            dcPan = dcMinPan
            self.pan_angle=self.pan_angle_min
        else:
            self.pan_angle = angulo
        self.raspi.set_servo_pulsewidth(self.servoPan, dcPan)



    def decrease_pan_angle(self, angulo):
        DecreaseDcPan = dc(angulo)
        dcPan = self.raspi.get_servo_pulsewidth(self.servoPan)
        dcPan = dcPan - DecreaseDcPan

        dcMinPan = self.pan_DCmin
        dcMaxPan = self.pan_DCmax

        if dcPan > dcMaxPan:
            dcPan = dcMaxPan
            self.pan_angle=self.pan_angle_max
        elif dcPan < dcMinPan:
            dcPan = dcMinPan
            self.pan_angle=self.pan_angle_min
        else:
```

```
            self.pan_angle = angulo
        self.raspi.set_servo_pulsewidth(self.servoPan, dcPan)


#funciones para el tilt angle
    def set_tilt_angle(angulo,self):
        dcMinTilt = self.tilt_DCmin
        dcMaxTilt = self.tilt_DCmax
        dcTilt = dc(angulo)

        if dcTilt > dcMaxTilt:
            dcTilt = dcMaxTilt
            self.tilt_angle=self.tilt_angle_max
        elif dcTilt < dcMinTilt:
            dcTilt = dcMinTilt
            self.tilt_angle = self.tilt_angle_min
        else:
            self.tilt_angle = angulo

        self.raspi.set_servo_pulsewidth(self.servoTilt, dcTilt)
    def get_tilt_angle(self):
        return self.tilt_angle
    def increase_tilt_angle(self, angulo):
        IncreaseDcTilt = dc(angulo)
        dcTilt = self.raspi.get_servo_pulsewidth(self.servoTilt)
        dcTilt = dcTilt + IncreaseDcTilt
        dcMinTilt = self.tilt_DCmin
        dcMaxTilt = self.tilt_DCmax

        if dcTilt > dcMaxTilt:
            dcTilt = dcMaxTilt
            self.tilt_angle=self.tilt_angle_max
        elif dcTilt < dcMinTilt:
            dcTilt = dcMinTilt
            self.tilt_angle = self.tilt_angle_min
        else:
            self.tilt_angle = angulo

        self.raspi.set_servo_pulsewidth(self.servoTilt, dcTilt)
    def decrease_tilt_angle(self, angulo):
        DecreaseDcTilt = dc(angulo)
        dcTilt = self.raspi.get_servo_pulsewidth(self.servoTilt)
        dcTilt = dcTilt - DecreaseDcTilt
        dcMinTilt = self.tilt_DCmin
        dcMaxTilt = self.tilt_DCmax

        if dcTilt > dcMaxTilt:
            dcTilt = dcMaxTilt
            self.tilt_angle=self.tilt_angle_max
        elif dcTilt < dcMinTilt:
            dcTilt = dcMinTilt
            self.tilt_angle = self.tilt_angle_min
        else:
            self.tilt_angle = angulo

        self.raspi.set_servo_pulsewidth(self.servoTilt, dcTilt)

#funciones para el roll angle
    def set_roll_angle(self, rollAngle):
        dcMinRoll = self.roll_DCmin
        dcMaxRoll = self.roll_DCmax
```

```
        dcRoll = dc(rollAngle)

        if dcRoll > dcMaxRoll:
            dcRoll = dcMaxRoll
            self.roll_angle=self.roll_angle_max

        elif dcRoll < dcMinRoll:
            dcRoll = dcMinRoll
            self.roll_angle = self.roll_angle_min

        self.raspi.set_servo_pulsewidth(self.servoRoll, dcRoll)
    def get_roll_angle(self):
        return self.roll_angle
    def increase_roll_angle(self, angulo):
        IncreaseDcRoll = dc(angulo)
        dcRoll = self.raspi.get_servo_pulsewidth(self.servoRoll)
        dcRoll = dcRoll + IncreaseDcRoll
        dcMinRoll = self.roll_DCmin
        dcMaxRoll = self.roll_DCmax

        if dcRoll > dcMaxRoll:
            dcRoll = dcMaxRoll
            self.roll_angle=self.roll_angle_max

        elif dcRoll < dcMinRoll:
            dcRoll = dcMinRoll
            self.roll_angle = self.roll_angle_min

        self.raspi.set_servo_pulsewidth(self.servoRoll, dcRoll)
    def decrease_roll_angle(self, angulo):
        DecreaseDcRoll = dc(angulo)
        dcRoll = self.raspi.get_servo_pulsewidth(self.servoRoll)
        dcRoll = dcRoll - DecreaseDcRoll
        dcMinRoll = self.roll_DCmin
        dcMaxRoll = self.roll_DCmax

        if dcRoll > dcMaxRoll:
            dcRoll = dcMaxRoll
            self.roll_angle=self.roll_angle_max

        elif dcRoll < dcMinRoll:
            dcRoll = dcMinRoll
            self.roll_angle = self.roll_angle_min

        self.raspi.set_servo_pulsewidth(self.servoRoll, dcRoll)
```

Anexo C. Código de prueba de la clase Gimbal

```
import RPi.GPIO as GPIO
from GIMBAL import GIMBAL
import time
def dc(angle):
    angle = float(angle)
    dutyCycleMs = 1000 * angle / 90 + 500

    return round(dutyCycleMs)

time.sleep(2)

MyGimbal=GIMBAL(13,19,26)

time.sleep(2)

MyGimbal.set_pan_angle(MyGimbal.pan_angle_min)
time.sleep(2)

while(MyGimbal.pan_angle<=MyGimbal.pan_angle_max):
    MyGimbal.increase_pan_angle(5)
    print(str(MyGimbal.get_pan_angle()))
    time.sleep(2)

while(MyGimbal.pan_angle>MyGimbal.pan_angle_min):
    MyGimbal.decrease_pan_angle(5)
    print(str(MyGimbal.get_pan_angle()))
    time.sleep(2)

MyGimbal.set_tilt_angle(MyGimbal.tilt_angle_min)
time.sleep(2)

while (MyGimbal.tilt_angle <= MyGimbal.tilt_angle_max):
    MyGimbal.increase_tilt_angle(5)
    print(str(MyGimbal.get_tilt_angle()))
    time.sleep(2)

while (MyGimbal.tilt_angle > MyGimbal.tilt_angle_min):
    MyGimbal.decrease_tilt_angle(5)
    print(str(MyGimbal.get_tilt_angle()))
    time.sleep(2)

MyGimbal.set_roll_angle(MyGimbal.roll_angle_min)
time.sleep(2)

while (MyGimbal.roll_angle <= MyGimbal.roll_angle_max):
    MyGimbal.increase_roll_angle(5)
    print(str(MyGimbal.get_roll_angle()))
    time.sleep(2)

while (MyGimbal.roll_angle > MyGimbal.roll_angle_min):
    MyGimbal.decrease_roll_angle(5)
    print(str(MyGimbal.get_roll_angle()))
    time.sleep(2)
```