
DESARROLLO E IMPLEMENTACIÓN DE UNA PLATAFORMA PARA EL GUIADO Y NAVEGACIÓN DE UN VEHÍCULO AÉREO NO TRIPULADO: Instrumentación y vuelo de un cuadricóptero



PROYECTO FINAL DE CARRERA

Bastida, Eric

**Facultad de Ingeniería y Ciencias Hídricas
Universidad Nacional del Litoral**

Noviembre 2018

Documento maquetado con TEXIS v.1.0+.

DESARROLLO E
IMPLEMENTACIÓN DE UNA
PLATAFORMA PARA EL
GUIADO Y NAVEGACIÓN DE
UN VEHÍCULO AÉREO NO
TRIPULADO:
Instrumentación y vuelo de un
cuadricóptero

*Memoria que presenta para optar al título de Ingeniero en
Informática*
Bastida, Eric

Dirigida por el Doctor
Lucas Genzeliz

Codirigida por el Ingeniero
Nahuel Deniz

Facultad de Ingeniería y Ciencias Hídricas
Universidad Nacional del Litoral

Noviembre 2018

Recuerda mirar a las estrellas y no tus pies. Intenta dar sentido a lo que ves y pregúntate por lo que hace al universo existir. Sé curioso. Aunque la vida puede parecer difícil, siempre hay algo que puedes hacer y tener éxito.

-Stephen Hawking-

Agradecimientos

Frase celebre

AUTOR

TODO

Resumen

*La única forma de hacer un gran trabajo
es amar lo que haces*

-Steve Jobs-

Últimamente se ha visto que el mercado de los vehículos aéreos no tripulados (UAVs) o mejor conocidos como *drones* se está expandiendo a usos cotidianos en la sociedad, a tal punto que ha llegado a utilizarse en tareas tales como misiones de búsqueda y rescate, inspección de líneas eléctricas, actividades agrícolas, seguridad vial, lucha contra incendios forestales, y hasta en aspectos recreativos, como captura y transmisión de imágenes aéreas. Sin embargo, en la actualidad hay pocas aplicaciones que permitan implementar algoritmos e ideas sobre vehículos de estas características. Por tal motivo y para solventar este problema, se propone implementar un software donde sea posible poner en marcha estas ideas o emprendimientos.

De esta manera, el presente proyecto involucrará el desarrollo de una plataforma que provea de funcionalidades a más alto nivel con el propósito de poder implementar acciones de una manera mucho más fácil para el usuario. Además, como caso particular se incluirá la instrumentación de un cuadricóptero, considerando cada etapa del armado del mismo: desde el ensamblado hasta la prueba de vuelo. Finalizando así con una aplicación sumamente personalizable y *Open-Source*.

Palabras claves: UAVs, drone, cuadricóptero, plataforma, Open Source.

Índice

Agradecimientos	VII
Resumen	VIII
1. Justificación, Objetivos y Alcance	2
1.1. Justificación	3
1.2. Objetivos	6
1.3. Alcance	6
1.3.1. Módulo de operación manual	7
1.3.2. Módulo de comunicación inalámbrica	7
1.3.3. Módulo de gráfico de datos	7
2. Obtención y Análisis de requerimientos	8
2.1. Introducción	9
2.2. Diseño global	11
3. Gestión de componentes y armado del cuadricóptero	13
3.1. Hardware	13
3.1.1. Estructura de vehículo	14
3.1.2. Controlador Electrónico de Velocidad	15
3.1.3. Motores sin escobillas	15
3.1.4. Raspberry Pi 3	15
3.1.5. Navio2	16
3.1.6. Arduino UNO	18
3.1.7. Radio Control	18
3.1.8. Joystick	19
3.1.9. Router	19
3.1.10. Módulos XBee	19
3.1.11. Batería LiPo	19
3.2. Procedimiento	20
3.2.1. Armado del cuerpo	20
3.2.2. Armado y configuración del piloto	21
3.2.3. ConFiguración para el acceso a la red	24
3.2.4. Instalación del firmware	25
3.2.5. Primeras pruebas	25
3.2.6. Conexión de componentes	28

3.2.7. Prueba de funcionamiento inicial	31
4. Desarrollo de la plataforma	33
4.1. Introducción	33
4.2. Diseño detallado de la plataforma	34
4.2.1. Diseño de interfaz	35
4.2.2. Diseño arquitectónico	36
4.3. Codificación y Testing	37
4.3.1. Clase Vehículo	37
4.3.2. Clase Joystick	38
4.3.3. Clase Gráfico Datos	39
4.3.4. Clase HUD	39
4.3.5. Clase Gestor Parámetro	44
4.3.6. Clase Parámetro	45
4.3.7. Clase GUI becopter	46
4.3.8. Clase BEcopter	46
I Apéndices	59
A. Documentación	60
A.1. Documento de Especificación de Requerimientos	60
A.2. Manual de usuario	60
A.3. Mockups de la interfaz gráfica del proyecto	60
A.4. Diagrama de clases inicial del Proyecto	60
Índice alfabético	61
II Anexos	61

Índice de figuras

1.1.	Aplicaciones agrícolas con drones	3
1.2.	Inspección de tendido eléctrico con drones	3
1.3.	Aplicaciones de emergencia	4
1.4.	Aplicaciones de búsqueda y rescate	4
3.1.	Estructura cuadricóptero de fibra de carbono	14
3.2.	Controlador Electrónico de Velocidad	15
3.3.	Motor sin escobillas	15
3.4.	Raspberry Pi 3	16
3.5.	Navio2	16
3.6.	Placa Arduino UNO	18
3.7.	Comando Radio Control	18
3.8.	Módulo de comunicación XBee	20
3.9.	Batería LiPo	20
3.10.	Cuerpo de cuadricóptero armado	21
3.11.	Esquema de instalación de hélices.	21
3.12.	Conexión entre Raspberry Pi 3 y placa Navio2.	22
3.13.	Pantalla de bienvenida luego de instalar el SO	23
3.14.	Captura de pantalla del Software MavProxy	26
3.15.	Radio Control LANYU model y su correspondiente receptor.	27
3.16.	Ubicación de conexiones en el Rx	28
3.17.	Receptor o Rx de 4 canales	29
3.18.	Codificador PWM a SBUS implementado sobre Arduino.	29
3.19.	Esquema de conexiones	30
3.20.	Conexión de datos a los motores.	30
3.21.	Fuente variable regulable.	31
3.22.	Cuadricóptero armado para verificar funcionamiento.	31
3.23.	Ejecución del modulo de calibración de MavProxy	32
4.1.	Head Up Display	35
4.9.	Horizonte artificial indicando un giro a la derecha en descenso.	40
4.2.	Diagrama de clases v2, perteneciente al proyecto BEcoper	49
4.3.	Clase Vehículo	50
4.4.	Marco de referencia global.	50
4.5.	Clase Joystick	50

4.6.	Clase gráficos de datos o de atributos del vehículo.	50
4.7.	Prueba animada en la inserción de dato utilizando la Clase Gráfico Datos.	51
4.8.	Clase Head Up Display	52
4.10.	Indicador de rumbos	52
4.11.	Coordinador de giro.	52
4.12.	HUD v0.1	53
4.13.	Error gráfico al momento de rotar	53
4.14.	Problema gráfico en las esquinas	53
4.15.	Circunferencia con radio que satisface la longitud requerida.	53
4.16.	Error al momento de simular el Pitch	54
4.17.	Insertando un rectángulo perteneciente al piso.	54
4.18.	Mapeo del horizonte artificial.	55
4.19.	Ángulo de banco.	55
4.20.	HUD final con sus correspondientes atributos e instrumentos de vuelo.	56
4.21.	56
4.22.	Clase parámetro	57
4.23.	Clase Graphical User Interface de BEcopter	57
4.24.	Aplicación QT designer	57
4.25.	Ejemplo del diseño de estilo con CSS.	58
4.26.	Clase principal BEcopter	58

Índice de Tablas

|

Capítulo 1

Justificación, Objetivos y Alcance

Frase celebre

Autor

RESUMEN: En este capítulo se detallan los elementos que dan origen a este proyecto y componen la especificación del mismo. Se hace una reseña de los antecedentes identificados y cómo se justifica el trabajo realizado, aclarando los objetivos planteados para ellos y las restricciones tenidas en cuenta para el alcance y las tecnologías utilizadas. Por último, se mencionan los posibles pasos que puede tener en futuro el software desarrollado en este trabajo.

1.1. Justificación

Una de las áreas de la aviación que ha experimentado un rápido crecimiento es aquella relacionada con los vehículos aéreos no tripulados (UAVs). Esta clase de vehículos pueden ser operados de forma remota o totalmente autónoma , presentan reducidos costos de operación y además no ponen en riesgo al operador. En los últimos años han ganado gran interés debido a su utilización en misiones de búsqueda y rescate , inspección de líneas eléctricas , actividades agrícolas , recolección de imágenes , seguridad , lucha contra incendios forestales , entre otros.



Figura 1.1: Aplicaciones agrícolas con drones



Figura 1.2: Inspección de tendido eléctrico con drones

Los UAVs pueden implementarse a través de los dos tipos de vehículos aéreos disponibles: aviones o multirotores (dentro de este último grupo se encuentran los cuadricópteros). Esto hace que los UAVs resultantes presenten características dinámicas y prestaciones (autonomía, maniobrabilidad, capacidad de carga, etc.) muy definidas y diferentes. Los aviones son capaces de recorrer grandes distancias empleando poca energía, gracias a la sustentación provista por las alas, de modo que el UAV resultante tiene una gran autonomía. Sin embargo, las maniobras necesarias para relevar la información son muy complicadas o requieren de sistemas de control avanzados. Por otro lado, los cuadricópteros se caracterizan por su maniobrabilidad y su capacidad de vuelo estacionario (hovering), lo que los hace particularmente aptos para la recolección de información en un punto determinado. Actualmente en el mercado nacional no se



Figura 1.3: Aplicaciones de emergencia



Figura 1.4: Aplicaciones de búsqueda y rescate

han reportado empresas que se dediquen al desarrollo de UAVs, siendo los mismos importados, tanto su hardware como su software. Por otro lado, los UAVs disponibles no son *open source*, lo cual implica que la adaptación de los mismos a las necesidades de clientes y/o empresas se ve dificultada. Desde el punto de vista académico-científico, la posibilidad de implementar algoritmos propios en este tipo de vehículos comerciales es prácticamente nula. Si bien existen empresas que se dedican al desarrollo de UAVs *open source*, como lo es la empresa española Erle Robotics¹, pero los costos de adquisición de dichos vehículos son bastante elevados.

De lo expuesto anteriormente se desprende la necesidad de contar con una plataforma de desarrollo para UAVs. Una plataforma para este tipo de vehículos, es un sistema que sirve como base para hacer funcionar los módulos de hardware o de software. En lo que concierne al hardware hacemos referencia a los componentes electrónicos del UAVs como pueden ser:

- Motores.
- Sensores como acelerómetros, giroscopios, magnetómetros, etc.
- Periféricos de entrada y/o salida tales como tarjeta SD, puertos USB, entre otros.

Para poder obtener información de cada componente y gestionar sus interacciones es necesario que un software administre en un segundo plano estas

¹Página web de Erle Robotics <http://erlerobotics.com/>

tareas, y que de forma sencilla proporcione al usuario funcionalidades para manipular estos componentes electrónicos, con el propósito de realizar acciones de navegación sobre el vehículo

Con el objetivo de solventar las deficiencias presentes en proyectos del mercado, tales como código privativo y escasa visualización de datos en forma evolutiva, se propone realizar el proyecto *open source* e implementar un módulo de visualización de datos a través del tiempo, este tiene el propósito de monitorear y controlar las variables de estado y controles del vehículo. De esta forma, la realización del PFC propuesto será de gran utilidad para el desarrollo de cualquier otro tipo de proyecto que involucre vehículos autónomos, ya que facilitará la tarea del operador para que el mismo enfoque su atención en sus propios objetivos.

Para la realización del PFC, se tendrá a disposición el equipamiento e instalaciones del Instituto de Investigación en Señales, Sistemas e Inteligencia Computacional (*sinc(i)*), donde desempeñan sus actividades de investigación los directores propuestos. El *sinc(i)* fue creado en 2004 por la Facultad de Ingeniería y Ciencias Hídricas (FICH) de la Universidad Nacional del Litoral (UNL), siendo reconocido como unidad ejecutora de doble dependencia UNL-CONICET en 2014. Las tareas de investigación en el *sinc(i)* tienen como objetivo desarrollar nuevos algoritmos para el aprendizaje automático, procesamiento de señales, modelado y análisis de sistemas complejos, proporcionando tecnologías innovadoras para el avance de la salud, la agricultura de precisión, la bioinformática, la energía, los sistemas autónomos e interfaces hombre-máquina. El *sinc(i)* posee la infraestructura necesaria para el correcto desempeño del PFC propuesto, a saber: espacio físico, equipamiento informático, laboratorio de electrónica, software, códigos y algoritmos propios. Recientemente ha adquirido computadoras modelo Raspberry Pi², que operan en conjunto con placas Navio2³, las cuales contienen GPS, doble IMU (unidad de medición inercial), giróscopo y barómetro, siendo de aplicación específica para navegación de vehículos autónomos. Además, cuenta con un kit de GPS con soporte de RTK (los cuales logran precisión centimétrica) y con modelos a escala de un automóvil tipo crawler y de un cuadricóptero. Asimismo, posee un centro de prototipado rápido que permite el desarrollo e implementación de sistemas embebidos a medida.

El desarrollo de este Proyecto Final de Carrera (PFC) impactará positivamente tanto en la industria nacional como en el ámbito académico, científico y tecnológico. En primer lugar, el alumno adquirirá el know-how (experiencia, conocimiento y habilidad) en el manejo de sistemas embebidos para el desarrollo de plataformas *open source* para UAVs. Además, impulsará el salto tecnológico no sólo por el desarrollo de UAVs a nivel local con fines de entretenimiento, sino que también permitirá trasladar el conocimiento adquirido a la obtención de UAVs con fines comerciales, económicos y/o sociales, entre otros, contribuyendo fuertemente al desarrollo de la industria argentina. Desde el punto de vista personal, la realización del PFC propuesto aumentará el activo de conocimientos complementarios obtenidos en la carrera Ingeniería en Informática, ya que el mismo involucra temas relacionados como redes y comunicación de datos, sistemas embebidos y electrónica digital. Asimismo, contribuirá en la for-

²Computadora reducida en tamaño y de bajo coste <https://www.raspberrypi.org/>

³Placa electrónica con un conjunto de sensores destinados para el control de navegación. <https://emlid.com/introducing-navio2/>

mación práctica y experimental, lo cual, en definitiva, es fundamental para el crecimiento de un profesional en este ámbito.

1.2. Objetivos

■ Objetivo General

Instrumentar el cuadricóptero con la Navio2 y la Raspberry Pi.

Diseñar e implementar una plataforma de desarrollo para UAVs.

■ Objetivos Específicos

Desarrollar un módulo que permita la comunicación inalámbrica entre el cuadricóptero y una PC para procesar la información adquirida.

Desarrollar un módulo que permita la adquisición de datos relevantes como ser posición, velocidad, aceleración y actitud.

Desarrollar un módulo de visualización de datos en tiempo real, el cual sea capaz de mostrar los datos provenientes de sensores que poseen alta frecuencia de muestreo.

Evaluar el funcionamiento de la plataforma desarrollada mediante la utilización de un modelo a escala de un cuadricóptero.

1.3. Alcance

El PFC propuesto consiste en desarrollar una plataforma para UAVs, que permita el control manual del mismo, además que permita la visualización de forma gráfica de los datos que fueron adquiridos de sus correspondientes sensores. Una vez desarrollada dicha plataforma será evaluada con un UAV tipo cuadricóptero. El desarrollo del proyecto estará restringido por las siguientes características:

- La plataforma dispondrá del libre acceso al código fuente, es decir, de tipo Open Source.
- Estará orientado al uso de computadoras de escritorio.
- El lenguaje de codificación utilizado será Python.
- Dispondrá de una interfaz gráfica.
- Las opciones de navegación que tendrá el cuadricóptero momentáneamente serán:
 - Punto objetivo.
 - Aterrizar.
 - Despegar.
 - Estabilizarse.
 - Volver al inicio.

En lo que concierne a las funcionalidades de la plataforma serán limitadas por los siguientes módulos:

1.3.1. Módulo de operación manual

El módulo perteneciente al control en modo manual será el encargado de gestionar los comandos que serán enviados desde un joystick conectado a la pc a nuestro cuadricóptero. Estos comandos determinarán los movimientos del vehículo según la restricción 5.

1.3.2. Módulo de comunicación inalámbrica

Corresponde a la comunicación entre el ordenador y el cuadricóptero. Se desarrollará con tecnología inalámbrica, considerando un radio de cobertura según los dispositivos disponibles a nuestro alcance. Además, gestionará el tipo de comunicación entre el receptor y emisor, con eso hacemos referencia al protocolo de comunicación, velocidad de transferencia, entre otros. Por el mismo medio se transmitirán los datos obtenidos por los sensores como también las maniobras establecidas por el usuario de forma manual.

1.3.3. Módulo de gráfico de datos

Este módulo permitirá la selección del sensor de nuestro interés, y mostrará de forma gráfica cómo evolucionan los datos que se están obteniendo de los mismos en tiempo real. Debido a que la mayoría de los sensores utilizados emplean una frecuencia de muestreo alta, es necesario utilizar librerías gráficas que sean capaces de cumplir con los requerimientos de tiempo real. Para asegurar este objetivo, se propone utilizar VisPy o similar, la cual es una librería escrita en lenguaje Python que está diseñada específicamente para la visualización interactiva de gran cantidad de datos de forma rápida, escalable y fácil. Con respecto a la adquisición de datos de los sensores, en el mercado existe un sinfín de éstos que son de utilidad para algún problema que se quiera resolver, como podrían ser: detección de temperatura, presión, humo, obtención de imágenes mediante una cámara o la captura de un video, entre otros.

En este PFC se utilizarán los sensores que nos proporciona la placa Navio2, a saber:

- 2 unidades de medición inercial MPU 9250 9DOF y LSM9DS1 9DOF. Son dispositivos electrónicos que miden e informan acerca de la velocidad, orientación y fuerzas magnéticas utilizando una combinación de acelerómetros y giróscopos.
- 1 barómetro MS5611 para medir la presión atmosférica.
- 1 sistema de navegación por satélite U-blox M8N, el cual utilizan una combinación de las tecnologías Glonass/GPS/Beidou.

Los cuales serán detallados en los próximos capítulos.

Capítulo 2

Obtención y Análisis de requerimientos

Frase celebre

Autor

RESUMEN: El presente capítulo contendrá una descripción del desarrollo de las etapas iniciales al proyecto. Como es sabido todo proyecto de cualquier tamaño es recomendable poder llevar una estructura que pueda favorecer el seguimiento y comprobar que este se está llevando a cabo según lo planificado. Pero antes de iniciar las tareas de desarrollo del proyecto es necesario principalmente definir que es lo que queremos hacer, por tal motivo debemos obtener información de las personas involucradas en el proyecto para saber qué es lo que realmente necesitan y poder obtener las necesidades reales, siempre y cuando sean realizable. Ya obtenido la información de los llamados *stakeholders*, se comienza a realizar modelos iniciales del futuro proyecto con el objetivo de presentarlos a los interesados y corroborar que el modelo cumple con todos los requerimientos obtenidos, y por lo tanto, empezar a iniciar las etapas de desarrollo. De esta manera este capítulo contendrá la descripción del desarrollo de las etapas de obtención y análisis de requerimientos y del Diseño global del sistema que están incluidas en nuestro proyecto y además los inconvenientes en caso de que se hayan presentado

2.1. Introducción

La primera etapa de este proyecto consiste en obtener los requerimientos del personal interesado en el desarrollo del mismo (*Stakeholders*), que en nuestro caso son personal del *El Centro de Investigación en Señales, Sistemas e Inteligencia Computacional sinc(i)*) la Dra. Marina Murillo, Ing. Lucas Genzeliz, Ing. Nahuel Denis y el Ing. Guido Sánchez.

El proceso de obtención de requerimientos ha consistido en una entrevista con el personal dónde se ha tenido una charla especificando las necesidades y naturaleza del proyecto y además, se ha presentando las tecnologías disponibles para el desarrollo del mismo. Una vez obtenido los requerimientos, se inicia la actividad de estudiar el campo involucrado del proyecto, con el fin de entender el problema, buscar alternativas, validar y modificar o eliminar requerimientos.

Como la naturaleza del proyecto se origina en la necesidad de un software que pueda comunicarse con un vehículo aéreo no tripulado y además, poder enviarle acciones que puedan ser ejecutadas desde larga distancia tenemos que analizar el campo del *aeromodelismo*, por lo que el ejecutante del proyecto deberá tener los conocimientos necesarios para el mismo.

Para iniciar con el desarrollo del proyecto es necesario de herramientas; estas ayudan al ejecutante realizar las tareas integradas en cada etapa del proyecto, por lo tanto, es de suma importancia elegir de las indicadas para que el desarrollo se facilite. De esta manera, comenzaremos explicando las herramientas elegidas para el desarrollo del software.

- **Lenguaje de programación Python:** Se trata de un lenguaje de programación multiparadigma, ya que soporta programación orientación a objetos, imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma. Ademas corre con la ventaja de tener una gran cantidad de librerías que son de gran utilidad para el desarrollo de nuestro proyecto.
- **PyCharm:** es un entorno de desarrollo integrado (IDE) utilizado en la programación de aplicaciones , específicamente en lenguaje Python . Es desarrollado por la compañía JetBrains. Proporciona análisis de código, un depurador gráfico, probador de unidades integrado, integración con sistemas de control de versiones (VCS) y admite el desarrollo web con Django. Un aspecto importante es que Pycharm es multi-plataforma, con versiones para Windows, Mac OS y Linux; teniendo disponibles dos versiones, *Professional* y *Community*, siendo esta última la elegida por su utilización gratuita.
- **Qt Designer:** Es un programa (módulo del Framework Qt) para desarrollar interfaces gráficas de usuario y multilenguajes debido a que genera un archivo XML cuyo contenido es el formato del respectivo GUI. Existiendo la posibilidad de convertirlo a Python, que es el lenguaje elegido para el desarrollo del presente proyecto.
- **PyQtGraph** es una librería gráfica y de creaciones de GUI enteramente construida en Python, basadas en las famosas librerías **PyQt4 / PySide** y **numpy**. Está destinado para uso en aplicaciones matemáticas / científicas y de ingeniería. A pesar de estar completamente escrito en Python, la

biblioteca es muy rápida debido a su gran apalancamiento con numpy para el procesamiento de números y al marco de GraphicsView de Qt para una visualización rápida. Por último, PyQtGraph se distribuye bajo la licencia de código abierto MIT.

- **PyQt4:** es una librería gráfica de Qt y realizada bajo Python. Esta librería nos da la opción de poder realizar una interfaz gráfica a nuestra plataforma, para que sea de fácil uso y entendimiento. Además, el software Qt Designer genera código en python utilizando esta librería automáticamente.
- **Pygame:** es un conjunto multi-plataforma de módulos de Python diseñados para la escritura de videojuegos. Incluye gráficos por ordenador y librerías de sonido diseñado para ser utilizado con el lenguaje de programación Python. Se distribuye bajo la GNU Lesser General Public License. Cuenta con muchos tutoriales, muchos de ellos online.
- **Socket:** Al utilizar Python, todos los programas utilizan sockets para lograr comunicarse con otros programas que pueden estar en diferentes computadoras. Por lo que podemos definirlo como un objeto que define una conexión entre dos entidades de una red. De esta manera, un socket posee la dirección IP de la máquina, el puerto en el que escucha, y el protocolo que utiliza. Los tipos y funciones necesarios para trabajar con estos elementos están en el módulo socket de Python.
- **Dronekit:** Es una API desarrollada por 3D Robotics en Python, flexible y abierta para el desarrollo de aplicaciones que se ejecutan desde el ordenador de abordo y comunican con el controlador de vuelo ArduPilot con un enlace de baja latencia.

La API se conecta con los drones a través del protocolo de comunicación MAVLink, un sistema utilizado habitualmente para conectar una estación de control de tierra y un vehículo no tripulado para transmitir, por ejemplo, la ubicación y la velocidad. Gracias a este protocolo, la API tiene acceso a los datos de la telemetría a través de Bluetooth, Wi-Fi o el sistema de radio 3DR o a los parámetros del dron y también permite controlar el movimiento y las operaciones del propio aparato no tripulado.

Entre las características destacadas tenemos:

- Desarrollar aplicaciones que mejoran el vuelo en piloto automático.
- Proporcionar características inteligentes a un dron como la visión por ordenador, la planificación de la ruta o el modelado en 3D.
- Permitir a un vehículo aéreo seguir objetivos mediante GPS.
- Facilitar el seguimiento de una ruta marcada con receptores GPS.

Además es posible instalar en sistemas operativos Linux, Windows y Mac OS X , y la plataforma funciona tanto en ordenadores de escritorio como en móviles. También se puede trabajar con el SDK y acceder a la API en la nube.

- **Login:** Este módulo define funciones y clases que implementan un sistema flexible de registro de eventos para aplicaciones y bibliotecas. Ya que

por naturaleza del proyecto estamos utilizando hardware para el desarrollo del mismo, es de suma importancia llevar un registro de los acontecimientos de cada uno, y en caso de generarse algún tipo de inconveniente poder detectarlo con gran facilidad.

- **Sphinx:** Sphinx es una herramienta que facilita la creación de documentación de una manera simple e inteligente, escrita por Georg Brandl y bajo la licencia BSD. Originalmente fue creada para la documentación de código escrita en Python, y cuenta con excelentes funcionalidades para la documentación de proyectos de software en varios idiomas. Sphinx se destaca por las siguientes características:

- **Formatos de salida:** HTML (incluida la Ayuda HTML de Windows), LaTeX (para versiones PDF imprimibles), ePub, Texinfo, páginas de manual y texto sin formato.
- **Amplias referencias cruzadas:** marcado semántico y enlaces automáticos para funciones, clases, citas, términos de glosario e información similar.
- **Estructura jerárquica:** fácil definición de un árbol de documentos, con enlaces automáticos a documentos relacionados.
- **Índices automáticos:** índice general así como índices de módulos específicos del idioma.
- **Manejo de código:** resaltado automático utilizando el resaltador de Pygments.
- **Extensiones:** prueba automática de fragmentos de código, inclusión de cadenas de documentación de módulos de Python (documentos API) y más de 50 extensiones contribuidas por usuarios en un segundo repositorio; La mayoría de ellos instalables desde PyPI.

En lo que concierne a los requerimientos y restricciones del proyecto han sido debidamente escritos en el Documento de Especificaciones de Requerimiento adjuntado en el Apéndice A.1.

2.2. Diseño global

En esta etapa del proyecto consiste en desarrollar un modelo que mejor represente las necesidades de los interesados y además, que sirva de guía para su respectivo desarrollo. Por tal motivo es necesario seleccionar un modelo que sea de fácil comprensión, con poco lenguaje técnico con el propósito de involucrar en mayor medida a cualquier tipo de *stakeholder* ya que en definitiva ellos determinarán si el producto final satisface sus necesidades y es utilizable. Es por esto, que se decide analizar modelos que representen el proyecto de una forma muy superficial, es decir, una técnica que describa los comportamientos generales del software y sin entrar en mucho tecnicismo. A raíz de esto y con el conocimiento adquirido en lo que concierne a ingeniería de software se seleccionan dos técnicas, que son:

- **Mockup.**

- **Casos de usos (CU).**

La primer propuesta consiste en realizar un bosquejo de la apariencia del software (GUI) y además, incluirle ciertos comportamientos mediante animaciones que den la sensación al usuario de que está utilizando el producto terminado. Por otro lado la segunda propuesta es realizar el modelo comportamental mediante casos de usos, esta técnica consiste en bosquejar un entorno que representa el ámbito donde están las opciones que brindará el sistema con sus respectivos actores o responsables de cada funcionalidad. Analizando las ventajas y desventajas de cada uno se decide utilizar la técnica de **Mockup** ya que es más representativa al producto real y no utiliza lenguaje técnico; como por ejemplo el utilizado en los CU (*include, extends, entorno y actores*). Para el desarrollo del mockup se utiliza la herramienta *Balsamiq Mockup*¹ y el resultado es anexado en el apéndice A.3.

Sin embargo, utilizar una técnica que detalle un aspecto superficial y un comportamiento general no ayuda en el inicio de implementación del sistema, es por eso, que en conjunto con el *mockup* y el *ERS* se decide implementar un modelo que ayude y/o simplifique más la tarea de decodificación del software como lo es el *diagrama de clases*. En ingeniería de software, un diagrama de clases dentro de lo que es el Lenguaje Unificado de Modelado (UML) es un tipo de diagrama de estructura estática que describe la estructura de un sistema mostrando las clases del sistema, sus atributos, operaciones, y las relaciones entre los objetos. En el apéndice A.4 se puede ver el respectivo diagrama inicial del software, cabe aclarar que el mismo durante todo el proceso del desarrollo del proyecto ha ido sufriendo cambios que eran necesarios, ya sea por nuevas funcionalidades, complementar las existentes o simplemente desecharlas ya que no cumplían su respectivo fin.

¹ Utilizando su versión gratuita de 30 días, página web <https://balsamiq.com/>

Capítulo 3

Gestión de componentes y armado del cuadricóptero

La mejor estructura no garantizará los resultados ni el rendimiento. Pero la estructura equivocada es una garantía de fracaso.

Peter Drucker

RESUMEN: Este capítulo contiene información sobre el procedimiento que se ha llevado a cabo en el desarrollo de la tercera etapa del proyecto. Esta etapa consiste principalmente en describir los pasos realizados para el armado del vehículo aéreo no tripulado (VANT) de tipo cuadricóptero. Además, se incluirán los inconvenientes que se han presentado en el transcurso del desarrollo del mismo con sus respectivas alternativas y/o soluciones.

Cabe mencionar que el hardware aquí descripto como también las herramientas utilizadas han sido proporcionadas por el *Centro de Investigación en Señales, Sistemas e Inteligencia sinc(i)* con sede en la Facultad de Ingeniería y Ciencias Hídricas de la Universidad Nacional del Litoral y todo procedimiento riesgoso para el alumno ha sido siempre supervisado bajo personal idóneo en el tema.

3.1. Hardware

Antes de empezar con la descripción del armado de un VANT, es necesario tener en conocimiento el hardware que está involucrado en el vehículo y en consecuencia su funcionamiento, como por ejemplo, los sensores que forman parte del mismo y hacen que el cuadricóptero pueda realizar un vuelo de manera segura y controlada, el tipo de estructura usado para soportar los motores, la tecnología responsable de la comunicación y el ordenador que procesará toda esa información. Es por eso que iniciamos esta sección con una lista del hardware utilizado y su respectiva descripción:

1. Estructura de vehículo.
2. Controlador Electrónico de Velocidad.
3. Motor sin escobillas.
4. Raspberry Pi 3.
5. Navio2.
6. Arduino UNO ¹.
7. Radio Control ¹.
8. Joystick.
9. Router ¹.
10. Módulos XBee
11. Batería LiPo.

3.1.1. Estructura de vehículo

La estructura del cuadricóptero como se puede ver en la Figura 3.1 consta en su parte inferior de un patín de aterrizaje que cumplen la función (como en los helicópteros) de soportar el vehículo cuando se apoye sobre tierra y mantener las hélices lo más alejado posible de piso cuando se encuentra en reposo. En su parte superior proporciona una base donde es posible instalar los dispositivos electrónicos que controlan al vehículo. Por último, desde la base se extienden 4 brazos con la función de ser soporte a los motores con sus respectivas hélices y además, en cada motor se le incluye su controlador electrónico de velocidad o ESC (*Electronic Speed Controller*).



Figura 3.1: Estructura cuadricóptero de fibra de carbono

¹Estos dispositivos no han sido previstos y tampoco forman parte definitiva del proyecto, ya que han surgido para solventar problemas que surgieron en el transcurso de esta etapa.

El material del cuerpo de vehículo esta hecho con fibra de carbono ², este consta con la propiedad de tener una elevada resistencia mecánica que será de suma importancia ya que el vehículo presenta altas probabilidades de sufrir algún choque o aterrizaje forzoso, además de ser un material sumamente liviano por lo que disminuirá la fuerza de los motores para mantener un vuelo y por tanto, menos consumo de batería.

3.1.2. Controlador Electrónico de Velocidad

Un controlador electrónico de velocidad o por sus siglas en inglés *Electronic Speed Control* como su nombre lo dice, es un circuito electrónico con el propósito de variar la velocidad de un motor eléctrico. Además, por sus características puede funcionar como un freno dinámico.

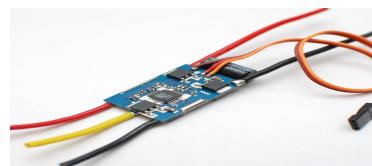


Figura 3.2: Controlador Electrónico de Velocidad

3.1.3. Motores sin escobillas

Son los encargados de transformar la energía eléctrica en mecánica generando movimiento en las hélices, y de esta manera dando propulsión al vehículo para poder volar.



Figura 3.3: Motor sin escobillas

3.1.4. Raspberry Pi 3

Es un ordenador de tamaño reducido que trata de ofrecer las mismas funcionalidades y componentes que el de un ordenador común pero con un rendimiento menor, entre sus componentes, esta placa reducida tiene memoria RAM, GPU, puertos USB, HDMI, ranura para tarjeta SD, conector para una cámara,

²Aunque el mismo puede ser reemplazado por otro tipo de estructura, por ejemplo, un modelo impreso por una impresora 3D

conectividad WiFi, Ethernet, Bluetooth y 40 pines GPIO(*General Purpose Input/Output*) donde es posible extender sus características instalando hardware extra como sensores, led, motores, interfaces, etc.

Especificaciones:

- 1.2GHz 64-bit quad-core ARMv8 CPU
- 1GB RAM (Compartido con la GPU)
- GPU Broadcom VideoCore IV



Figura 3.4: Raspberry Pi 3

3.1.5. Navio2

La placa Navio2 es un conjunto de sensores que se conectan sobre pines GPIO de una Raspberry Pi, transformándolo en un completo controlador de drones. Esta placa controladora suministra a la Raspberry Pi de sensores y entradas que son de suma importancia para cualquier tipo de vehículo, con el objetivo de monitorear y obtener información para algún tipo de maniobra aérea, como también terrestre.



Figura 3.5: Navio2

Dentro del conjunto de sensores que contiene esta placa tenemos:

3.1.5.1. IMU

Unidad de Medición Inercial (o por sus siglas en inglés *Inertial Measurement Unit*) es el componente principal de los sistemas de navegación. Son un conjunto de dispositivos electrónicos, generalmente, una combinación de acelerómetros, magnetómetros y giroscopios que miden e informan características del movimiento del vehículo como pueden ser velocidad, fuerzas magnéticas y orientación. Como es un aspecto importante en este tipo de sistemas la precisión, la placa controladora Navio2 contiene dos de estas unidades con el fin de proporcionar y corroborar su información mediante dos fuentes de referencia distintas. Estas dos unidades son:

1. MPU9250 9DOF.
2. LSM9DS1 9DOF.

3.1.5.2. GNSS

Obtener la posición del vehículo mientras este se encuentra en movimiento es un aspecto sumamente necesario, más cuando la visibilidad del ambiente dificulta hacerlo, es por eso que el GNSS (*Global Navigation Satellite System*) proporciona un posicionamiento geoespacial con cobertura global mediante un conjunto de tecnologías de sistemas de navegación por satélite. Las tecnologías utilizadas en el módulo U-blox M8N de esta placa son:

1. Glonass.
2. GPS.
3. Beidou.

3.1.5.3. Entrada/Salida RC (Radio Control)

La placa Navio2 tiene una entrada habilitada para recibir información proveniente del emisor mediante un radio control, aceptando señales por los protocolos de comunicación PPM (Modulación por Posición de Pulso) y SBUS únicamente.

3.1.5.4. Barómetro

El barómetro tiene sus usos provenientes de la meteorología para poder predecir el tiempo entre otras cosas, pero en este caso la placa controladora utiliza un barómetro MS5611 de alta precisión (con 10cm de resolución) para poder estimar la altura en el cual se encuentra el dispositivo.

3.1.5.5. Interfaces

- Entrada UART (*Universal Asynchronous Receiver-Transmitter*)
- Bus de serie de datos I^2C de sus siglas en inglés (Inter-Integrated Circuit)
- Conversor Analógico/Digital ADC
- Y por último. 14 salidas para servomotores mediante el protocolo PWM (Modulación por Ancho de Pulso)

3.1.6. Arduino UNO

El Arduino UNO es una plataforma computacional física open-source basada en una simple tarjeta de I/O y un entorno de desarrollo que implementa el lenguaje Processing/Wiring. A diferencia de la Raspberry Pi 3, este está fabricado para realizar tareas más sencillas, ya que su poder de procesamiento y memoria son limitados.

- Microcontrolador ATmega328.
- 14 pines digitales de I/O (6 salidas PWM).
- 6 entradas analógicas.
- 32KB de memoria Flash.
- Reloj de 16MHz de velocidad.



Figura 3.6: Placa Arduino UNO

3.1.7. Radio Control

Este control permite gobernar al vehículo a distancia de manera inalámbrica mediante señales de radio.



Figura 3.7: Comando Radio Control

3.1.8. Joystick

Una palanca de mando o comúnmente conocido como *Joystick* es un dispositivo que por lo general contiene 2 palancas con dos ejes cada uno, donde es posible representar la posición de un punto según la posición física de este y un conjunto de botones que al ser presionados y según el software intermedio puede codificarse ciertas acciones. Este puede estar conectado mediante un cable USB o de manera inalámbrica. En nuestro caso, utilizaremos un joystick con conexión USB ya que necesitamos que el envío de información sea confiable.

3.1.9. Router

Un router es un dispositivo de hardware que permite la interconexión de ordenadores en red. El router o enrutador es un dispositivo que opera en capa de nivel de 3. Así, permite que varias redes u ordenadores se conecten entre sí y, por ejemplo, comparten una misma conexión de Internet.

Este dispositivo es usado para el desarrollo del proyecto ya que en el momento de las pruebas no se tienen a disposición los módulos de comunicación verdaderos, estos módulos permiten una mayor distancia de cobertura y funciona exclusivamente para una conexión, lo que logra una mayor confiabilidad a la misma que comparando con el router que debe gestionar varias conexiones a la vez y la información enviada no está asegurada. Por tal motivo de ausencia de estos módulos se recurre a utilizar como medio de comunicación provisoria la tecnología wifi del router, ya que únicamente se pretende realizar pruebas en un ámbito controlado y de poca distancia.

3.1.10. Módulos XBee

Los módulos XBee³ son soluciones integradas que brindan un medio inalámbrico para la interconexión y comunicación entre dispositivos. Estos módulos utilizan el protocolo de red llamado IEEE 802.15.4 para crear redes FAST POINT-TO-MULTIPOINT (punto a multipunto); o para redes PEER-TO-PEER (punto a punto). Están diseñados para aplicaciones que requieren de un alto tráfico de datos, baja latencia y una sincronización de comunicación predecible.

3.1.11. Batería LiPo

La batería LiPo (Litio y Polímero) son baterías recargables, compuestas generalmente de varias células conectadas en paralelo para aumentar la capacidad de la corriente de descarga. Esta contiene un voltaje de 12v con 2200 mA, suministrando dicha energía a los motores y además a la Raspberry Pi con 5V.

³Página web de XBee <https://www.digi.com/xbee>

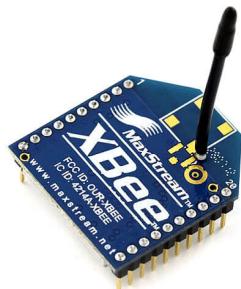


Figura 3.8: Módulo de comunicación XBee



Figura 3.9: Batería LiPo

3.2. Procedimiento

3.2.1. Armado del cuerpo

Una vez ya descripto el hardware adquirido para el desarrollo de esta etapa, estamos habilitados para iniciar el proceso del armado del vehículo. En primera instancia se comienza con el cuerpo del cuadricóptero, esta actividad es realizada en base a las instrucciones que nos proporciona el vendedor *ValueHobby*⁴ obteniendo el cuerpo armado como se puede observar en la Figura 3.10. Como medida de seguridad antes de realizar las pruebas se han extraído las hélices. Pero al momento se ser instaladas con el propósito de que el cuadricóptero no se tumbe con respecto a su eje de orientación cuando este se encuentre en el aire se deben colocar las hélices pares de tal manera que su propulsión al momento de girar sea en sentido contra-horario y las hélices impares deben estar colocadas en sentido horario tal como se ven en la Figura 3.11 .

⁴<http://www.valuehobby.com/media/wysiwyg/upload/Manual/bumblebee-manual.pdf>



Figura 3.10: Cuerpo de cuadricóptero armado

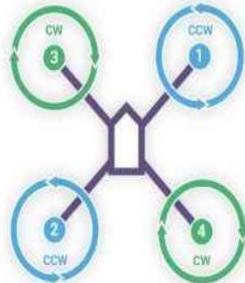


Figura 3.11: Esquema de instalación de hélices.

3.2.2. Armado y configuración del piloto

En esta sección se describe el armado del piloto del cuadricóptero, este va a ser el encargado en gestionar todos los sensores, enviar señales a los motores y administrar la información generada por estos sensores con el fin de ser enviadas a una maquina cliente y que este las pueda interpretar, entre otras cosas.

Para comenzar con el armado debemos instalar la placa Navio2 sobre la Raspberry Pi 3, proporcionando así los sensores indispensables para el control, monitoreo y vuelo del vehículo ya que el ordenador de placa reducida Raspberry Pi no cuenta con estos sensores de forma nativa. Por lo tanto, se procede a conectar los 40 pinos GPIO sobre la placa Navio2, como se muestra en la Figura 3.12

Luego se procede a conectar la antena GNSS sobre la placa Navio2. Una vez ya conectado se debe verificar su correcto funcionamiento de todos los sensores presentes, para eso debemos tener un software que gestione eficazmente todo el hardware y sea el intermediario entre estos dispositivos electrónicos y el usuario, además de proporcionar servicios de utilidad para el mismo, en otras palabras, un sistema operativo (SO). Antes de instalar cualquier sistema opera-

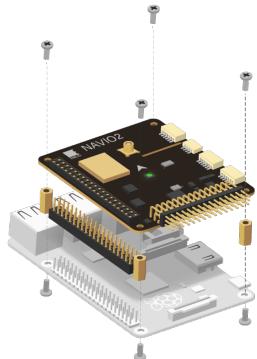


Figura 3.12: Conexión entre Raspberry Pi 3 y placa Navio2.

tivo hay que tener en cuenta que estamos en presencia de un ordenador de placa reducida, es decir, el hardware de los componentes del mismo como memoria, procesador, puertos de entrada y salida, etcétera no son los mismos a los ordenadores comerciales que vemos comúnmente; por poner un ejemplo, el procesador que contiene la Raspberry es un ARM1176JZF-S, con esto queremos decir que el CPU en si mismo está fabricado bajo una arquitectura con un conjunto reducido de instrucciones (o por su siglas en inglés *RISC - Reduced instruction set computing*) en comparación a un CPU que son utilizados normalmente en un ordenador hogareño. Por tal motivo, es necesario de un SO que se adapte a estas características; afortunadamente existen varias organizaciones o empresas que desarrollan y distribuyen este tipo de producto, por ejemplo, *Canonical Ltd* y *Microsoft* han adaptado sus sistemas operativos para este tipo de ordenadores. En nuestro caso utilizaremos un SO proporcionado por la empresa *Emlid* basado en una distribución de Linux llamada *Raspbian* (acrónimo entre Raspberry Pi y Debian) donde ya viene preinstalado las herramientas necesarias.

Una vez seleccionado el SO a instalar, se procede a descargar la imagen y grabarla en una tarjeta SD, una vez realizado este paso se carga la tarjeta SD en la ranura disponible en la Raspberry Pi, se enchufa un teclado, mouse y un conector HDMI que se encuentra conectado a un monitor, y por último se procede a encender el dispositivo mostrándonos la siguiente imagen:

Como se puede observar en la imagen 3.13 la pantalla de bienvenida nos muestra los siguientes pasos que hay que seguir para configurar nuestro piloto.

- **Paso 1:** Se selecciona el tipo de vehículo que se va a manejar, utilizando el comando *update-alternatives*. Esta función se encarga de seleccionar como predeterminada la opción ingresada dentro de otras alternativas, es decir, Es posible que tenga en el sistema varios programas instalados a la vez que realizan la misma función. Por ejemplo, muchos sistemas tienen varios editores de texto instalados al mismo tiempo, lo que deja la elección de qué editor de texto utilizar en manos del usuario, si éste lo desea, pero hace difícil que un programa elija la opción correcta si el usuario no ha definido ninguna preferencia. En nuestro caso es el tipo de autopiloto como puede ser arducopter, arduplaner y ardurover. Como estamos por utilizar un cuadricóptero ingresamos:

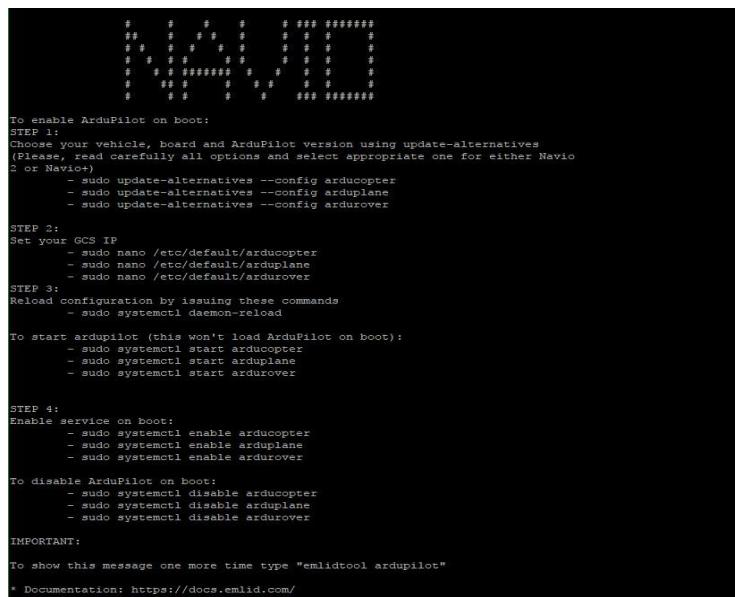


Figura 3.13: Pantalla de bienvenida luego de instalar el SO

```
> sudo update-alternatives --config arducopter
```

- **Paso 2:** Se debe ingresar dentro del archivo `/etc/default/arducopter` el IP perteneciente al GCS (*Ground Control Station*), es decir, este archivo contiene la dirección IP del ordenador que enviará y recibirá información del vehículo. Además, brinda la opción de seleccionar la interfaz por el que se realizará la comunicación. En nuestro caso lo haremos realizaremos mediante tecnología Wifi y utilizando el protocolo UDP⁵.
- **Paso 3:** Una vez ya seleccionado el tipo de vehículo a utilizar e ingresado el IP de nuestro GCS se debe iniciar los servicios y procesos proporcionados por arducopter (ya que este se encargará de administrar las funcionalidades pertinentes), para eso se utiliza el gestor del sistema y servicios **systemd** de Linux con el objetivo de iniciar un conjunto nuevo de procesos y servicios encapsulados en Arducopter, para eso utilizamos los siguientes comandos :

```
> sudo systemctl daemon-reload
> sudo systemctl start arducopter
```

- **Paso 4:** Por último es deseable que los servicios proporcionados por Arducopter se inicien cada vez que el vehículo se encienda, por tal motivo se ingresa el último comando :

⁵Es posible también utilizar el protocolo TCP, pero la razón por la cual se usa este método es porque es un protocolo no orientado a la conexión, por lo tanto como estaremos realizando maniobras que son ejecutadas por el vehículo en tiempo real necesitamos la menor latencia posible y no estar esperando paquetes de confirmación todo el tiempo

```
> sudo systemctl enable arducopter
```

3.2.3. ConFiguración para el acceso a la red

Entre las opciones de conexión a la red que dispone la Raspberry Pi 3, podemos identificar que el mismo tiene una entrada Ethernet como también un módulo interno Wi-Fi, por lo tanto es necesario configurar dichas interfaces para poder conectarnos mediante un cable UTP con un conector RJ45, como también por tecnología *Wireles*. Por lo que dentro del sistema operativo del vehículo y sobre la consola se ingresa el siguiente comando

```
> sudo ifconfig
```

Con este comando podemos verificar nombres de interfaces presentes y si se encuentran en actual funcionamiento. En caso de observar ningún tipo de tráfico por estas interfaces se procede a configurarlas, para esto se utiliza el mismo comando pero con la opción de activar la interfaz deseada, en nuestro caso queremos activar la interfaz del módulo interno de Wi-Fi, para eso ingresamos el siguiente comando

```
> sudo ifconfig intwifio up
```

Una vez ya activo podemos buscar las redes disponibles y que son captadas por el módulo, utilizando el siguiente comando

```
> iwlist wlan0 scan
```

Con este comando nos muestra mediante una lista las redes captadas por el módulo, una vez que hayamos encontrado la red de nuestro interés podemos conectarnos con el siguiente comando

```
> iwconfig wlan0 essid "Nombre de Red" key "Contraseña"
```

y finalmente procederemos a obtener nuestra IP y conectarnos con la siguiente función :

```
> sudo dhclient wlan0
```

En caso de querer almacenar esta red e indicarle al sistema operativo que es nuestra red por defecto podemos modificar el archivo `wpa_supplicant.conf` ingresando el nombre de la red y su correspondiente clave. Por último, se reinicia y una vez iniciado el sistema se comprueba su conectividad realizando chequeo de conexión con algún tipo de página en internet, por ejemplo, Google utilizando el siguiente comando

```
> ping 8.8.8.8
```

3.2.4. Instalación del firmware

Una vez ya comprobado la conectividad a la red y configurado el sistema operativo, es necesario instalar el *firmware* encargado de gestionar todos los sensores que se han incluido a la Raspberry mediante la placa Navio2, con el objetivo de poder administrar esta información y poder enviarla al *GCS*. Para instalar el firmware se procede a descargarlo desde el mismo SO del vehículo mediante el siguiente comando

```
pi@navio: ~\$ wget http://firmware.eu.ardupilot.org/Copter/stable/
      navio2-quad/arducopter-quad
pi@navio: ~\$ chmod +x arducopter-quad}
```

Una vez descargado el firmware se desea que los servicios ya instalados de Arducopter utilicen dicho firmware actualizado, por lo tanto se procede a modificar el archivo /etc/systemd/system/ardupilot.service. Modificando la línea

```
#ExecStart=/bin/sh -c "/home/pi/path/to/your/binary \$ARDUPILOT_OPTS
\}"
```

por

```
#ExecStart=/bin/sh -c "/home/pi/arducopter-quad \$ARDUPILOT_OPTS\}"
```

3.2.5. Primeras pruebas

Ya instalado el firmware correspondiente dentro de la Raspberry Pi estamos en condiciones de poder hacer nuestras primeras pruebas, para eso, dentro del mismo vehículo se ejecutan programas proporcionados por ArduPilot para poder observar los datos generados por los sensores. En la misma carpeta de instalación se encuentran programas de ejemplo para observar los datos provenientes de los sensores y al ejecutar dichos programas evidentemente la información mostrada es errónea ya que estos sensores no se encuentran calibrados, por lo tanto, para realizar estas correcciones utilizamos las herramientas recomendadas y proporcionadas por la empresa ArduPilot como es el software **MavProxy** (Figura 3.14), este software muestra a través de una consola todos los datos obtenidos de los sensores, estado del vehículo entre otras cosas, siendo lo más importante poder calibrar los sensores. Así que una vez iniciado el software se prueba la opción de calibrarlos. Al momento de iniciar el proceso de calibración surge el inconveniente de no poder iniciarla, ya que para realizar dicha acción es de suma importancia tener conectado el radio control al vehículo. Esto genera un inconveniente al momento del desarrollo de esta fase ya que la compra de un radio control conllevaría un tiempo extra en el cronograma del proyecto. Buscando alternativas en conjunto se descubre que es posible realizar la calibración mediante un script que debíamos codificar. Analizando esta propuesta se descubre que para generar dicho script es necesario de estudiar las librerías involucradas lo que conlleva un tiempo considerado y únicamente el objetivo de esta fase es realizar la comprobación funcional de estos dispositivos. De esta manera se obtiene provisoriamente un radio control como se ilustra en la imagen 3.15

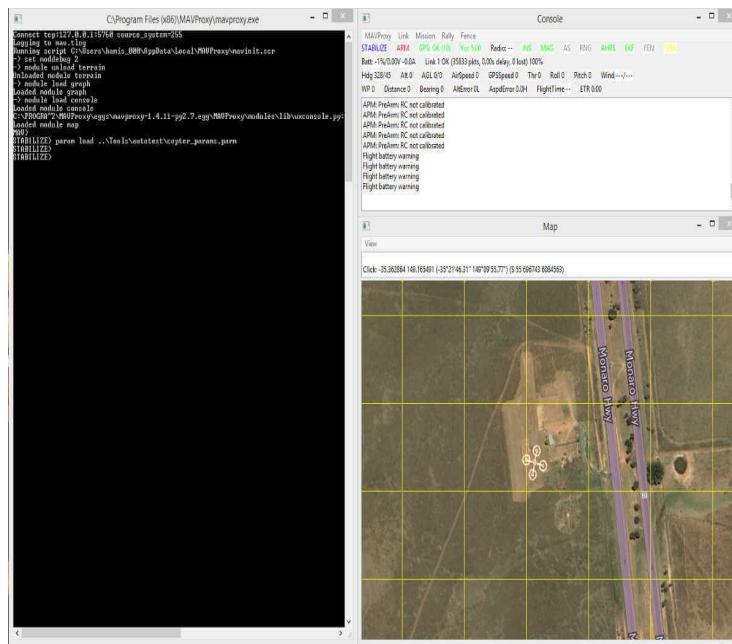


Figura 3.14: Captura de pantalla del Software MavProxy

Este radio control simplemente envía las señales por 4 canales independientes, que son generados por sus dos palancas donde cada una contiene 2 ejes (*eje x* y *eje y*), por lo tanto tendríamos que la palanca 1 enviará información del *eje_{p1x}* y *eje_{p1y}* y de la misma manera, la palanca 2 enviará información de el *eje_{p2x}* y *eje_{p2y}* mediante la técnica de modulación por ancho de pulso. Simplificando un poco, el receptor o (*RX*) tendrá 4 cables con información de cada canal y que deberá conectarse al vehículo, pero recordando la sección 3.1.5.3 el vehículo solo dispone de una entrada compatible con PPM o SBUS. Por lo que aquí surge otro problema, debemos encontrar la forma de poder convertir las señales del radio control que se encontraban en PWM y pasarlas a PPM o SBUS para que el vehículo pueda interpretar esta información. Para solucionar este problema se plantean 3 alternativas:

1. Comprar un radio control con su respectivo receptor compatible con el vehículo.
2. Realizar manualmente un conversor de PWM a PPM o SBUS con componentes electrónicos desde cero.
3. Implementar el conversor sobre una placa Arduino UNO.

Para estas propuestas se realiza el siguiente análisis:

Propuesta 1: La adquisición de un nuevo radio control más allá de sus futuros usos conllevaría un elevado costo adicional para el proyecto ya que el mismo tiene un valor de \$8000⁶, además atrasaría el cronograma los días que se deban esperar el producto desde su ciudad de origen.

⁶Fecha consultada, septiembre del 2017



Figura 3.15: Radio Control LANYU model y su correspondiente receptor.

Propuesta 2: Consiste en realizar la compra de los componentes electrónicos de tipo CMOS CD4001⁷ y CD40106⁸ con un precio estimado de \$100. A pesar de su bajo costo de implementación y siendo la mano de obra realizada por el profesor de la cátedra Electrónica Digital, es una tarea ajena al ejecutante del proyecto, por lo que no complementaría los conocimientos del alumno.

Propuesta 3: Esta propuesta consiste en implementar la función de conversión de PWM a PPM o SBUS mediante un script que se implementará sobre la placa Arduino, donde se conectarán los cables provenientes del Rx a sus entradas y retornará en una de sus salidas un único cable con la información codificada en PPM o SBUS como estaba especificado en el sitio web del fabricante. El precio estimado de la placa Arduino UNO es de \$200, pero existen versiones reducidas como el Arduino Nano que cuentan con un precio a la mitad del Arduino UNO. De manera conveniente se contaba con la adquisición de un Arduino UNO, por lo que no hacia falta realizar una compra. Además, la implementación de este algoritmo contribuye al crecimiento de las aptitudes del ejecutante del proyecto como estudiante de ingeniería en informática. Por lo que finalmente se decide realizar la propuesta 3. Una vez realizada se implementa sobre la placa Arduino UNO y se comprueban los resultados con un osciloscopio, dándonos un resultado favorable.

⁷<http://www.alldatasheet.es/datasheet-pdf/pdf/26834/TI/CD4001.html>

⁸<http://www.alldatasheet.es/datasheet-pdf/pdf/26839/TI/CD40106.html>

3.2.6. Conexión de componentes

Antes de comenzar a describir el proceso de cableado del vehículo se comienza conectando el receptor del comando radio control con el Arduino UNO que va a ser el encargado de recibir las señales PWM y convertirlas en SBUS momentáneamente para poder enviarlas a la Navio2. Hay que tener en cuenta que la ubicación de los conectores y los pines del receptor llevan un orden específico. Por tal motivo, se investiga dicho orden, pero sin encontrar información relevante, hasta ubicar con un transmisor de la misma empresa y deducir las siguientes conexiones



Figura 3.16: Ubicación de conexiones en el Rx

Como se puede observar en la Figura 3.16 la columna izquierda de pines le pertenece a los negativos, la columna central a los positivos y la última columna perteneciente a datos; esta es la de interés ya que por cada *conector_i* estaríamos recibiendo señal PWM del emisor correspondiente al *canal_i*, por lo que cada una son las entradas a nuestro Arduino UNO para poder realizar la codificación. Antes de iniciar la conexión con el codificador se comprueban las señales mediante un osciloscopio.

Ya identificados los cables correspondientes a cada canal, se conectan los mismos a la placa Arduino UNO. Por último, se conecta la salida proporcionada por el codificador a la entrada del vehículo, asegurando así una señal de tipo SBUS como lo indica su fabricante.

Con respecto a las conexiones eléctricas que lleva el cuadricóptero, recurrimos en basarnos en la Figura 3.19, donde muestra gráficamente el orden que llevan las conexiones con sus respectivos cables de colores.

Un aspecto a tener en cuenta es que no todos los componentes que se visualizan en la Figura 3.19 están presentes al momento de la instrumentación ya que los dispositivos marcados con los números (1,2 y 3) son de tipo comunicación y se supone que con el módulo interno de *Wifi* podríamos sustituir estos elementos.

Por lo tanto se procede a realizar las conexiones mostradas. En primera medida se inicia conectando los motores del cuerpo del cuadricóptero con los

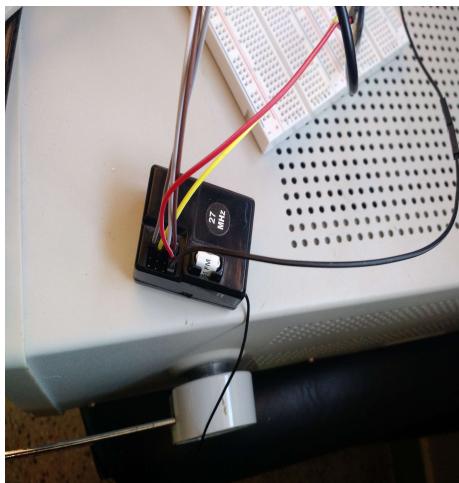


Figura 3.17: Receptor o Rx de 4 canales

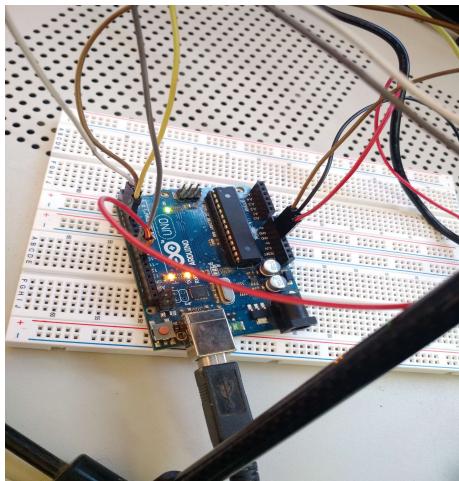


Figura 3.18: Codificador PWM a SBUS implementado sobre Arduino.

pines de la Navio2 como se puede observar en la imagen 3.20

Después de haber hecho las conexiones correspondientes a los motores, es necesario suministrarle energía de algún tipo de fuente para realizar las pruebas iniciales, es por eso que antes de conectar la batería LiPo se considera la siguiente cuestión: ya que esta tiene una carga útil limitada se decide utilizar una fuente variable regulable disponible en el laboratorio de electrónica del *sinc(i)* con el objetivo de realizar las pruebas necesarias sin estar dependiendo de la carga de la batería. Antes de realizar la conexión se ajusta el voltaje según la batería Lipo existente (12V), dejando libre la cantidad de Amperes para que consuma lo que necesite. La fuente utilizada se puede observar en la Figura 3.21

En la Figura 3.19 se puede ver que la batería suministra energía a los cuatro motores y al vehículo, por lo tanto se decide alimentar al vehículo con el transformador de fábrica y suministrar energía a los motores con la fuente. Esta

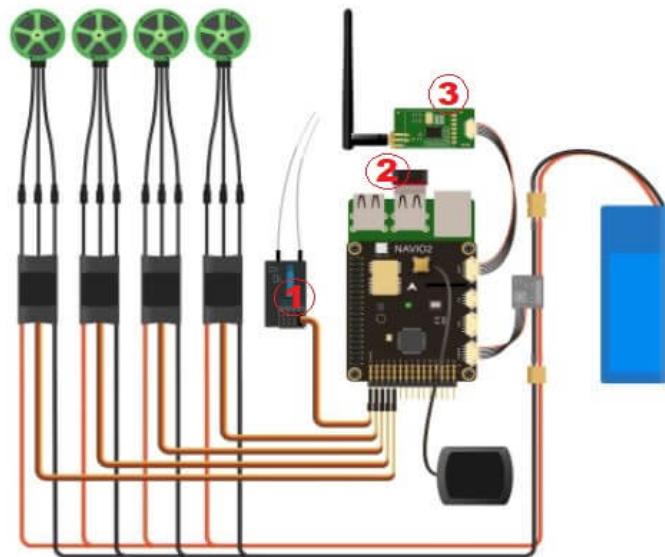


Figura 3.19: Esquema de conexiones

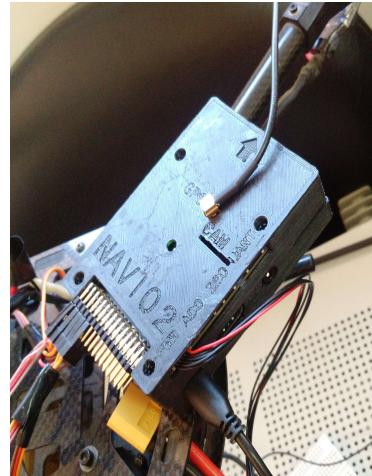


Figura 3.20: Conexión de datos a los motores.

fuente como se puede observar en la Figura 3.21 contiene un único par de salida (positivo y negativo) por lo que es necesario multiplexar esta salida a 4 par de salidas más. Para eso se decide armar y soldar un conjunto de cables con sus respectivos conectores quedando de la siguiente manera.

Por último, se verifican las conexiones pertinentes a los planos y se buscan imperfecciones en las uniones, en caso de encontrar algún tipo de estos se procede a repararlos.



Figura 3.21: Fuente variable regulable.

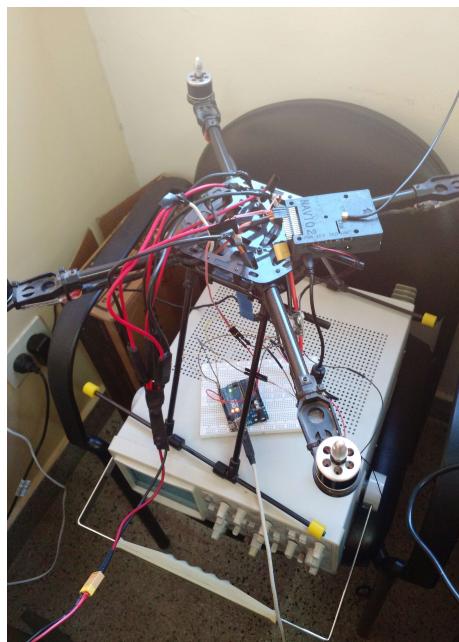


Figura 3.22: Cuadricóptero armado para verificar funcionamiento.

3.2.7. Prueba de funcionamiento inicial

Como etapa final y con el objetivo de verificar el correcto funcionamiento de nuestro vehículo, se inicia el proceso de calibración de sensores, para eso se decide utilizar **Mavproxy** seleccionando el módulo de calibración y consecuentemente realizando los pasos que se nos indican de manera sencilla como se puede observar en la Figura 3.23.

Una vez concluida la etapa de calibración se procede a enviar misiones de prueba, como por ejemplo elevarse del suelo 1 metro desde la posición inicial sin

tener instalado las hélices por motivos de seguridad, dándonos como resultados que el vehículo interpreta los comandos. Aunque estas pruebas fueron realizadas de modo superficial en el capítulo ?? se profundizará que el funcionamiento integral de los componentes responda de manera correcta.

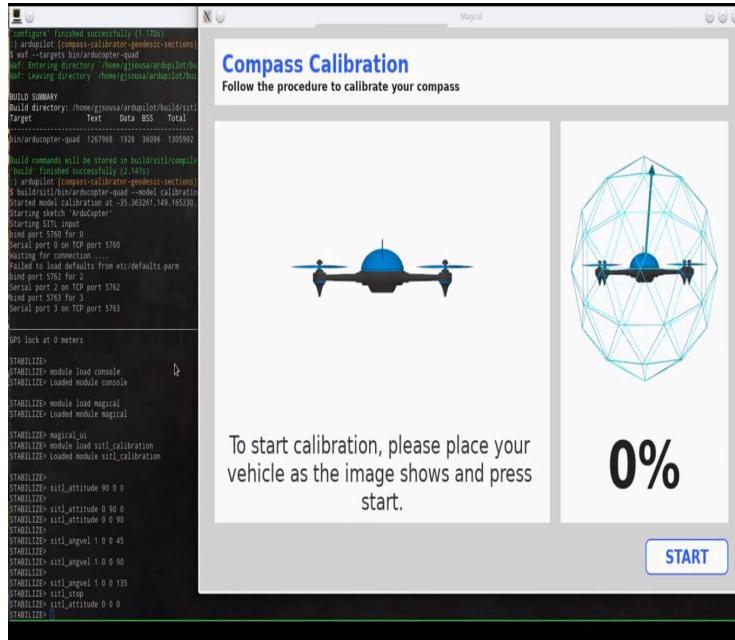


Figura 3.23: Ejecución del modulo de calibración de MavProxy

Capítulo 4

Desarrollo de la plataforma

Frase celebre

Autor

RESUMEN: En este capítulo se describirá todo el proceso realizado de la fase denominada **Desarrollo de la plataforma**, que forma parte del presente proyecto . En esta fase se contempla las tareas del *Diseño detallado de la plataforma* y *Codificación* del mismo. En aspectos generales se define de manera un poco más específica y en base al desarrollo global ya realizado en etapas anteriores, todo el comportamiento que incluirá la plataforma, como pueden ser, las funcionalidades de cada módulo, la interfaz gráfica de usuario o por sus siglas en inglés (GUI), la lógica interna correspondiente a cada uno y sus respectivas interacciones.

4.1. Introducción

En el ámbito informático, como es sabido, la etapa de codificación del software generalmente es la que más tiempo abarca con respecto a la duración total del proyecto. Por esta razón, no es de sorprender que a medida que vayamos avanzando objetivos, herramientas, planificación, diseño y lógica del software estén bajo un constante cambio ya que se van presentando obstáculos, alternativas e incompatibilidades que no han sido previstos en etapas iniciales. De esta manera, se irán ilustrando durante este capítulo capturas del estado del proyecto en distintas etapas del mismo, con el objetivo de poder ver el crecimiento y las técnicas utilizadas a medida que el proyecto va evolucionando y de esta manera poder aprender, para próximos proyectos, las estructuras que uno debe implementar en etapas iniciales con el fin de no cometer los mismos errores.

4.2. Diseño detallado de la plataforma

Un diseño de software es una descripción de la estructura del software que se va a implementar, los datos que son parte del sistema, las interfaces entre los componentes del sistema y, algunas veces, los algoritmos utilizados.

(Ian Sommerville, 2005, p71)

Antes de iniciar directamente con la codificación, ya que por el mismo instinto de cumplir con las restricciones de tiempo, uno necesita finalizarlo lo antes posible; pero antes que nada es necesario, realizar un diseño de todo lo que queremos hacer, esto genera una linea de ejecución a seguir que ayuda al ejecutor del proyecto tener planificado las tareas generales que debe realizar en el transcurso del proyecto y no estar pendiente de como proseguir una vez finalizada cada tarea y re-planificar constantemente el hilo del proyecto. Es por tal motivo que el presente proyecto se ha dividido la fase de diseño en dos, la primera es un diseño global, meramente con la intención de contar con un panorama general del comportamiento del software/hardware y además, poder mitigar la incertidumbre del ejecutor del proyecto, ya que se está iniciando en el desarrollo de proyectos de esta envergadura. En base a la documentación que se ha hecho en etapas anteriores podemos empezar a entablar de forma más precisa los procedimientos o tareas a realizar, lo que conlleva encarar la segunda parte del diseño.

Según Pressman esta etapa produce los siguientes diseños:

- Item

Dentro de lo que es el conjunto de diseños obtenidos en esta etapa es importante saber en que orden realizar cada una. Ya que cada diseño esta relacionado directamente con los demás, y el desarrollo de uno terminará condicionando el siguiente. Es por esto, que se analiza la opción de comenzar primero con el diseño de arquitectura. Suponiendo dicha elección se desarrolla el siguiente flujo:

En el diseño orientado a objetos o más específicamente, en la confección del diagrama de clases se detallan los objetos mediante bloques gráficos denominados clases. Estas clases contienen todos los atributos correspondientes al objeto, como por ejemplo, la clase vehículo puede contener: cantidad de hélices, altura de vuelo, velocidad, posición, etc. Y además, según este diseño contiene todas las acciones que puede realizar a través de funciones; si continuamos con el ejemplo de vehículo aéreo pueden ser: volar, aterrizar, despegar, etc. Esto nos brinda como resultado final una estructura al momento de implementar el código, por lo que facilita la tarea del programador. Además, y un punto importante a tener en cuenta es que muestra como se comunicará cada clase con las demás, esta característica ayuda a descubrir que funciones extras son necesarias implementar para una correcta cooperación en conjunto. Sin embargo, existe una desventaja, dentro del punto de vista de la interacción del usuario, es que no toma en cuenta las funcionalidades que puede desear el usuario para que el uso del software sea más placentero (que es el objetivo de este diseño), sino que el resultado del diseño se centra en que cada clase pueda responder a las necesidades (mensajes) de las demás clases.

En cambio, si primero realizamos el diseño de la interfaz gráfica del usuario, nos determinará que es necesario entre otros aspectos que no han sido incluido en el diseño arquitectónico ya que no se ha tomado como parte importante en el desarrollo, la interacción del usuario. Como por ejemplo, funciones de validación de datos, ventanas de advertencias, gráficos, como es el HUD (figura 4.1).



Figura 4.1: Head Up Display

Como podemos ver y según las restricciones presentes, es conveniente realizar primero un diseño de interfaz, ya que el mismo nos proporcionará más detalles de implementación. Por tal motivo iniciamos primero con el diseño de interfaz.

4.2.1. Diseño de interfaz

En primera instancia se inicia con la interfaz gráfica del usuario o su equivalente en inglés ***Graphical User Interface***; por definición es el proceso que define como será la interacción entre el software y el usuario final. Para esto, se utiliza la herramienta Qt Designer, que proporciona mediante un panel de **widgets** todos los elementos necesarios para el diseño del mismo.

Para iniciar este proceso, nos basamos en los *mockups* desarrollados en la fase de diseño global de la plataforma, basándonos pantalla por pantalla en la confección de cada uno. Cabe mencionar que las pantallas presentes del diseño han sido obtenidas luego de un arduo proceso de modificaciones y actualizaciones debido a las necesidades e inconvenientes que se iban presentando en el transcurso de la fase de codificación. Las imágenes de la GUI se pueden ver en el anexo ??

4.2.1.1. Modificaciones generales

- **Inserción del HUD en lugar de un mapa** Se ha considerado diseñar el HUD, en lugar de un mapa. Ya que el mismo no se considera de suma importancia para la navegación en un marco de referencia relativo. La implementación del mapa, se ha documentado como un requisito para las versiones futuras de la plataforma.
- **Reemplazo de ubicación de iconos de estados** Los iconos posicionados en la barra superior del software se han reubicados dentro de lo que es el HUD, ya que estos iconos sobrecargaban la aplicación con una constante actualización siendo que el HUD es el encargado de mostrar dicha información de manera actualizada.

- **Sustracción de pestaña *Sensores*** Según el requisito **Implementación de calibración de sensores** no se considera para esta versión la posibilidad de calibrar los sensores del vehículo. Si, se incluyen los parámetros del vehículo y sus respectivos valores en la pestaña de configuración, con el fin de ser configurados.

4.2.2. Diseño arquitectónico

Luego de finalizar con el diseño de interfaz y en base al diagrama de clases realizado en etapas anteriores del proyecto, se inicia el modelado más detallado del diagrama de clases de nuestro proyecto. Para esto se utiliza de la herramienta ArgoUML, obteniendo el siguiente estructura.

Para el desarrollo del mismo se ha considerado el diseño de la interfaz gráfica y de manera complementaria ,la comunicación que tiene con las demás clases conectadas, esto nos facilita al momento de determinar que tipo de funciones necesita cada una y la forma en que se los debe enviar dicha información.

4.3. Codificación y Testing

En esta etapa consiste en traducir el diseño en una forma legible por la máquina. Por tanto, iniciaremos esta tarea según los diseños de diagrama de clases generados y siempre teniendo en mente los requerimientos estipulados por los *stakeholders*.

Según, la Figura 4.2 tenemos varias clases que se encargarán de un grupo de funcionalidades en específico y a su vez cada una brinda funcionalidades a sus clases vecinas, para entender un poco el funcionamiento del software comenzaremos explicando cada una, exceptuando las clases que se han utilizado de las librerías pertinentes.

4.3.1. Clase Vehículo

La clase Vehículo es la responsable de representar en su mayor medida el objeto real, en nuestro caso es el cuadricóptero. Para esto se ha decidido asignarle las siguientes funcionalidades

- **Funciones de vuelo:** Según las restricciones establecidas en el ERS que se han estipulado en primera medida, ya sea por falta de conocimiento de librerías y/o complejidad de los mismos, únicamente las siguientes funcionalidades de vuelo **Ascenso, Descenso y Hovering**; en cambio, gracias a las librerías utilizadas se han codificado más funcionalidades correspondientes a las maniobras que puede realizar el vehículo. Dentro de este conjunto se desarrollaron las siguientes funciones (utilizando el protocolo MAVLink *Micro Air Vehicle Link* y la librería Dronekit):

Cabe mencionar que estas funcionalidades han sido diseñadas con la posibilidad de realizar acciones de vuelo en un marco de referencia global, utilizando las coordenadas globales proporcionadas por el GPS en base a la latitud, longitud y altura (figura 4.4). Además, es posible establecer sobre un marco de referencia relativo, es decir, este marco de referencia toma como origen el punto inicial de partida del vehículo, por lo que las coordenadas que ingresemos tendrán como referencia el punto de despegue. Es importante recordar que el marco de referencia global, conlleva la implementación de un mapa y según el requerimiento R025 este es opcional, e insistiendo con lo mencionado, esta funcionalidad se implementará en futuras versiones.

1. **Despegar:** Esta funcionalidad envía un comando al vehículo especificándole que debe propulsarse gracias a sus motores a una altura en particular.
2. **Aterrizar:** Esta función ordena al vehículo que aterrice en un punto establecido, dependiendo del sistema de referencia, ya sea global o relativo.
3. **Suspenderse:** Según un punto en particular el vehículo estará inmóvil y suspendido en el aire por la cantidad de tiempo que se le asigne. Además, esta función entra en el requerimiento R017, donde se establece una función de salvataje encargada de ejecutarse en caso de algún inconveniente o percance con el vehículo y/o conexión.

4. **Ir a un punto en específico:** Como bien su nombre lo especifica, esta orden y según un sistema de referencia le indica al vehículo que debe desplazarse en un punto en particular.
5. **Volver a inicio:** El vehículo almacena en primeras instancias su posición inicial, asignándola en una variable interna llamada *home*, una vez que esta orden ha sido enviada, el vehículo tratará de dirigirse a su punto de partida.
6. **Moverse:** Según el requerimiento R024, que establece distintos tipos de vuelo se han codificado las funciones de *velocidad* y *yaw* que según las ordenes enviadas de un joystick el vehículo modificará su dirección en tiempo real, ya que se han restringido estos comandos a que tengan una duración de ejecución mínima, dando así lugar a un nuevo comando y captar las nuevas indicaciones que han sido enviadas por el usuario.

■ **Modo de vuelo:** Dentro del conjunto de las funcionalidades de vuelo se pueden observar que existen funciones con el prefijo "mis", estos a diferencia de los demás, son utilizados cuando el vehículo se encuentra en modo AUTO, es decir, estos comandos serán enviados al vehículo con el fin de que sean almacenados y ejecutados de manera secuencial misión por misión cuando se envié la orden *ejecutar misiones*. Para cumplir de manera complementaria con el requisito R024, se han codificado funciones que gestionen las misiones pertinentes.

■ **Conexión:** En base a los requerimientos R015,R016 se han incluido en la clase vehículo las funciones de *conectado*, *desconectar* y *el constructor de la clase* estas son encargadas de conectarse de manera inalámbrica a una red LAN, gracias a su módulo WiFi instalado en la placa Raspberry Pi o de manera ad-hoc mediante módulos de comunicación inalámbrica conectados en un puerto serial.

Por último una función que asigna *Callbacks* a los atributos del vehículo, ya que estar consultando constantemente los cambios que se están produciendo dentro del vehículo conlleva un consumo extra de recursos,por lo que se decide implementar una función "setCallback2Param"que cada vez que el atributo correspondiente sufra algún tipo de cambio, este llame a otra función para informarle de dicha transición.

4.3.2. Clase Joystick

La clase Joystick, es la encargada de gestionar dentro del los comandos conectados en el equipo, sus respectivas acciones que han sido ejecutadas por el usuario.

Con la ayuda de la librería Pygame podemos obtener información de los joystick conectados como nombre, cantidad de botones, número de ejes, si se ha presionado o saltado un botón, entre otras acciones, con el objetivo de poder interpretar, según un mapeo de funciones que se han pre-configurado las acciones que desee realizar el usuario en modo manual sobre el vehículo. Cabe mencionar que dicha clase se ha codificado satisfaciendo los requerimiento R012 y R022.

4.3.3. Clase Gráfico Datos

La clase Gráfico de Datos fue creada con el fin de cumplir con el requerimiento R002 que solicita una representación gráfica en tiempo real y evolutiva de la información capturada. Para realizar dicha tarea en primer momento se ha utilizado la librería Vispy como la definición del requerimiento lo especifica; pero a medida que se iba avanzando en el desarrollo y prueba del mismo se ha encontrado el inconveniente de que dicha librería no contiene la documentación actualizada, lo que genera contratiempos al momento de cumplir con el objetivo, por lo que se decide utilizar una librería con una comunidad más activa, como lo es PyQtGraph. Esta librería cuenta con una gran comunidad activa y una documentación detallada de cada funcionalidad disponible, esto facilita mucho al ejecutor del proyecto a dirigir la atención en otros aspectos del proyecto. Cabe destacar que esta librería, más allá de contener las funcionalidades necesarias para cumplir con el requerimiento estipulado. Entre el conjunto de funcionalidades disponibles, se describen las más importantes:

- **Cupo máximo de líneas:** Con fines estéticos se ha implementado la opción de disponer de un número máximo de gráficos en pantalla, ya que el acumulamiento de gráficos sobrecargaría a la aplicación de recursos a consumir y además dejaría de apreciarse cada gráfica en la ilustración por motivos de espacio. En caso de asignar una nueva linea al gráfico correspondiente, el mismo eliminará el primer gráfico ingresada, utilizando una metodología tipo FIFO.
- **Identificación de líneas:** Para poder gestionar las líneas presentes en el gráfico es necesario asignarle a cada una un identificador, este identificador además servirá para las clases antecesoras para la inserción y eliminación de cada una.
- **Pausado de animación:** Es de esperar que el gráfico no este disponible todo el tiempo completo, por lo que se decide incorporar la opción de pausado de la animación con el propósito de optimizar el uso de recursos del equipo y que la aplicación BEcopter sea lo más liviana posible.

4.3.4. Clase HUD

La clase HUD o por su respectiva traducción, *visualización cabeza-arriba*, es la encargada de visualizar al piloto, en este caso el usuario, los datos más relevantes del estado del vehículo, con el objetivo de tener a simple vista el comportamiento del mismo en tiempo real. Por su parte, en el ámbito aeronáutico se utilizan ciertos instrumentos de navegación o símbolos que informan al piloto el estado de la aeronave, ya sea, velocidad, altitud, inclinación, etc; que para el desarrollo de esta clase es conveniente saber su significado. Por tal razón, explicaremos cada uno de estos instrumentos con su respectiva representación gráfica.

4.3.4.1. Instrumentos de navegación

Son los instrumentos esenciales para poder orientarse y seguir la ruta deseada por parte del piloto.

Horizonte artificial

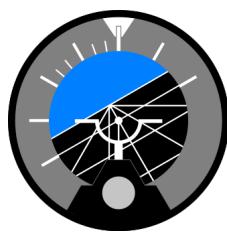


Figura 4.9: Horizonte artificial indicando un giro a la derecha en descenso.

El horizonte artificial muestra la orientación longitudinal de la aeronave (la relación del eje longitudinal del avión con respecto al plano del suelo), es decir: si está girado, inclinado, con la frente del vehículo levantado, bajado o todo a la vez. Sirve de gran ayuda en condiciones en que la visibilidad es poca o nula. El horizonte artificial tiene dos partes: el horizonte propiamente dicho, y el indicador de rumbo. El primero está compuesto por una región azul que representa el cielo, otra normalmente marrón que representa la superficie terrestre, una mira que representa la dirección en que apunta la frente de la aeronave, y varias marcas a su alrededor. Las marcas horizontales a ambos lados representan las alas, el plano de la aeronave, y su ángulo con el límite entre las regiones de cielo y superficie (el horizonte artificial), el cual con dichos planos se produce el ángulo de alabeo. Dispuestas verticalmente a intervalos regulares, hay marcas horizontales más pequeñas que representan ángulos concretos en el plano vertical, a intervalos de 5° , 10° , etc. Muestran el ángulo actual del eje longitudinal con el plano del suelo. Su principio mecánico es basado sobre un giroscopio.

Indicador de rumbos

El indicador de rumbo, o giroscopio direccional, proporciona al piloto la dirección de la aeronave en grados magnéticos.

Coordinador de Giro

En el coordinador de giro vemos en lugar del bastón una Figura de un avión que nos indica el grado de inclinación de las alas.

Una vez concientizado en los instrumentos de vuelo se procede a integrarlos de manera simulada en un *widget*, con el fin de que cumpla las mismas funcionalidades. Para empezar necesitaremos de una librería gráfica en el cual nos proporcione de funcionalidades para poder graficar estos instrumentos, para eso se selecciona la conocida librería OpenGL. Dicha librería está equipada de funcionalidades para dibujar escenas tridimensionales complejas a partir de primitivas geométricas simples, tales como puntos, líneas y triángulos. Además nos dá a disposición la personalización de todos los parámetros que estén dentro de una escena como posición de la luz, cámara, tipo de materiales y luces. Pero más allá de todo este conjunto de herramientas que nos deja a disposición OpenGL, la tarea del diseño del HUD es realizar en un dibujo en 2D, el cual se mostrarán un conjunto de datos y líneas básicas, por lo que se decide no utilizar directamente esta librería para el diseño de la escena, sino que se busca de una API en que podamos graficar nuestros instrumentos de navegación de forma más sencilla, es por eso, que sea realizada una investigación antes de utilizar dicha librería y se encuentra que la librería PyQt, en sus conjuntos de clases contiene una de nombre QPainter, más allá que internamente usa OpenGL, esta librería

facilita un conjunto de funciones tales como dibujar elipse, texto, imágenes, líneas, entre otras funcionalidades y de una manera más comprensible. De esta forma, abordamos la tarea de graficar la escena de la siguiente manera:

- 1. Dibujo del Horizonte Artificial** El horizonte artificial como se puede observar en la Figura 4.9 está compuesto por dos rectángulos, uno representando el cielo y el segundo la tierra, de esta manera, con la funcionalidad de QPainter que nos proporciona dibujar un rectángulo proseguimos a dibujar dos rectángulos simulando el horizonte artificial; quedándonos como se puede ver en la Figura 4.12.

Hasta aquí, podemos observar que la Figura 4.12 satisface las necesidades de representar el horizonte artificial según nuestras especificaciones, pero al momento de proyectar al uso del mismo podemos encontrar los siguientes problemas:

- a) Problema gráfico al momento del Roll:** Como se puede ver en la Figura 4.13 , cuando la escena sufre una rotación sobre su eje perpendicular, es decir, simulando un giro sobre el Roll o alabeo del vehículo, la imagen muestra imperfecciones en las puntas del recuadro.

Este problema fácilmente se puede solucionar ampliando los tamaños de los rectángulos en ambas direcciones, considerando siempre el tamaño más largo entre el alto y ancho de la pantalla, ya que si no tenemos en cuenta este aspecto cuando tengamos una imagen sumamente apaisada y de poca altura, obtendríamos el mismo problema. Por tal motivo, se debe elegir un tamaño adecuado para la imagen; ahora la pregunta es ¿Qué tamaño debo asignarle a la imagen para que no se presenten dichos defectos?

Obtención de la longitud requerida

Como bien muestra la Figura 4.14, cuando la imagen de fondo sufre una rotación no llega a cubrir lo suficiente los límites de la pantalla, por tanto, se produce dicho problema. De tal manera se debe buscar la longitud máxima que debe tener la imagen para que cubra dichos límites, en la Figura 4.14 se puede apreciar una linea roja que representa el largo necesario a calcular. Este deberá ser la mínima longitud que deben tener el alto y ancho de la imagen, es decir :

$$L > Ancho_{Imagenes} \wedge L > Alto_{Imagenes}$$

Para calcular dicho valor, se propone implementar una circunferencia con centro en el origen de la pantalla y que tenga un radio por el cual cubra por completo los límites del mismo, como se puede observar en la Figura 4.15, de esta manera se contemplan las posibilidades de que la imagen de fondo cubra por completo el dominio de la pantalla en cualquier angulo de rotación que sufra.

Por último, obtener esta longitud se traduce en un problema trigonométrico donde se debe obtener la hipotenusa de un triángulo rectángulo con base $w/2$ y alto $h/2$. Utilizando el teorema de pitágoras obtendremos que

$$R = \sqrt{\left(\frac{w}{2}\right)^2 + \left(\frac{h}{2}\right)^2} \quad (4.1a)$$

$$R = \frac{1}{2}(w + h) \quad (4.1b)$$

Como podemos apreciar, la longitud mínima requerida se obtiene de la ecuación 4.1b. Pero como es de saber, este valor es la longitud total mínima que debe tener tanto el alto como el ancho de la imagen, por lo que se debe conocer cual es la diferencia al que debemos agregar al ancho, como también al alto de la imagen. Para esto, simplemente realizamos la diferencia para cada atributo y obtendremos el siguiente valor a agregar

$$Ancho_{aux} = \frac{1}{2}(h + w) - \frac{w}{2} = \frac{h}{2} \quad (4.2a)$$

$$Alto_{aux} = \frac{1}{2}(h + w) - \frac{h}{2} = \frac{w}{2} \quad (4.2b)$$

- b) **Problema al simular el Pitch:** Otro problema que surge, es al momento de simular el cabeceo o pitch, es que dentro de la escena cuando ocurre este comportamiento el script debe realizar un transformación de traslado, siendo proporcional a la inclinación ocurrida en el vehículo. Pero en cambio, el defecto que ocasiona es que únicamente se trasladarán los dos cuadrados ya dibujados, dejando al descubierto el *background*, como se puede ver en la Figura 4.16. Además, hay que tener en cuenta que la escena debe permitir un pitch de 0 a 360 grados, por lo que se tendrá que expandir la escena con el fin de simular un entorno virtual 360 en 3D.

Para solucionar este inconveniente surgen las siguientes propuestas

- 1) **Mapear una textura al rededor de la cámara, mediante coordenadas esféricas:** En esta propuesta se debe generar una textura de ambiente exterior por el cual será mapeada de tal manera que simulará un ambiente 3D. La desventaja de esta propuesta es que la clase QPainter no proporciona funcionalidades para el desarrollo de dibujos en 3D, ni tampoco sus clases antecesoras para el mapeo de textura. Por lo cual, se debería codificar la textura manualmente y utilizando las funcionalidades primitivas de OpenGL.
 - 2) **Inserción de rectángulos dependientes del Pitch:** Aquí lo que se pretende es que a medida que el pitch aumente o disminuya, es decir, que el vehículo cabecee para arriba o abajo, el script interprete estos datos para dibujar un nuevo rectángulo contrario al que se ve en pantalla (cielo/tierra), simulando un entorno infinito, como se puede ver en la Figura 4.17.
- Siendo parecer la propuesta más efectiva, al momento de realizar las pruebas con el vehículo se descubre que presenta inconsistencias, ya que el movimiento del pitch no es continuo, por tanto,

no se puede ir llevando un seguimiento del cuadro que se está viendo actualmente para la inserción del siguiente. Más allá de poder emparchar esta propuesta anexando al algoritmo estructuras extras para el seguimiento de los cuadros, el propósito del mismo no es acomplejar el problema presente, de esta manera, se recurre al desarrollo de la siguiente propuesta.

- 3) **Mapeo del Pitch sobre una imagen:** Finalmente, para solucionar el presente problema, y siempre reduciendo la complejidad a una solución, se propone dibujar una imagen que contenga la escena del horizonte artificial en forma repetida, con el propósito de mapear el rango completo del pitch [0 - 360] dentro de una sola imagen, sin estar utilizando imágenes extras para cubrir lo restante.

Como se puede notar en la Figura 4.18 el mapeo del pitch corresponderá únicamente al dominio de la pantalla marcada en color rojo. Con esto hacemos referencia, que el mapeo no será de forma lineal ya que la entrada del pitch corresponde dentro del dominio de [0 - 360], por lo que se procede a convertirlo en el dominio [-180 - 180] para que sea más representativo a la dirección que apunta el vehículo y luego mapearlo al dominio del tamaño de la imagen. La razón por la cual se ha tomado el dominio de pantalla especificado, es porque cuando el pitch supere el límite de los 180 (grados), inmediatamente el mapeador deberá llevar "la vista" al rango inferior, es decir, a los -180 (grados), esto dará al usuario una sensación continua de la imagen ya que al sobrepasar, ya sean los límites inferior o superior del dominio de la pantalla, las escenas serán prácticamente las mismas.

2. **Dibujo del ángulo de banco** El ángulo de banco representa directamente al instrumento de coordinador de giro. Este instrumento indica al usuario la inclinación en el eje x o Roll de la misma forma en que se detalló en su definición, con la diferencia que este muestra el rango de inclinación [-(-60), 60] y en caso de sobrepasar este límite se indica con una flecha en color rojo a modo de precaución.
3. **Dibujo de líneas de referencia** Estas líneas de referencia tienen el fin de mostrar al usuario en la pantalla el pitch que está teniendo el vehículo en tiempo real, de la misma forma que el ángulo de banco, este muestra un rango limitado de [-40,40].
4. **Dibujo de atributos del vehículo** Por último, se ha tomado la decisión de ilustrar dentro del HUD ciertos atributos del vehículo y conexión, que se han considerado importante antes y durante del inicio del despegue. Dentro de estos atributos se han insertado los siguientes elementos como se puede ver en la imagen 4.20
 - **Indicador de rumbos**
 - **Icono de estado de GPS, Señal y batería**
 - **Un atributo a elección del usuario** En este caso se ha insertado por defecto el pitch del vehículo, pero este espacio puede ser ocupado por cualquier atributo del vehículo.

4.3.5. Clase Gestor Parámetro

Dentro del vehículo además de encontrarse con sus atributos tales como: velocidad, posición, batería, etc. existen parámetros, como bien su nombre lo dice parametrizables, donde es posible establecer o informarse de todo tipo de configuración del vehículo. Existe una gran variedad de estos, ya que es importante poder configurar cada aspecto de los elementos del vehículo. Dentro de este conjunto de parámetros podemos clasificarlos los siguientes grupos:

- **Armado**
- **Batería**
- **Motores**
- **Servo Output**
- **Telemetría**
- **IMU**
- **Compas**
- **Barometro**
- **GPS**
- **RC**
- **WayPoint**
- **Misiones**

Donde en cada grupo se encuentran reunidos un conjunto de parámetros individuales que configuran cierto aspecto de cada grupo. Por tal razón se decide crear una clase Gestión de Parámetro, de tal manera que sea el encargado de agrupar estos parámetros. Para la carga de información de cada parámetros se ha utilizado una planilla de cálculo para almacenar la información, esto es posible gracias a la librería *openpyxl* que nos permite ir guardando cada grupo de parámetros en una hoja independiente dentro del documento.

4.3.6. Clase Parámetro

Cada parámetro tiene la particularidad dentro del software un conjunto de propiedades que ayudan al usuario a entender su significado y poder modificarlos. Para la creación de dicha clase se han creado las siguientes propiedades:

- **Permiso:** Este propiedad determina si el parámetro es de solo lectura o es modificable por el usuario.
- **Código:** Dentro del software se generan códigos a cada parámetro con el fin de poder enviarlos mediante comandos al vehículo.
- **Nombre:** Es una pequeña descripción del código, ya que los mismos están escritos de forma reducida.
- **Descripción:** Describe de manera extendida el significado de cada parámetro y en determinados casos con ejemplos para ayudar al usuario a configurarlos.
- **Rango:** Los valores máximos y mínimos que puede tomar el parámetro.
- **Incremento:** Tiene como fin ser una propiedad para configurar un *slider* dentro de cada parámetro en la GUI.
- **Unidades:** Unidades correspondiente al parámetro.

Como es de notar la clase parámetro no almacena su correspondiente valor, esto es debido a que los valores de cada parámetro son tomados del vehículo cuando se establece la primera conexión. Ya que estar manipulando valores cambiantes en dos lugares a la vez pueden producir accidentes al momento de sincronizarlos.

4.3.7. Clase GUI becopter

En lo que respecta al proyecto es la clase de mayor tamaño, ya que contiene absolutamente todos los elementos gráficos que se pueden ver en BEcopter. Estos están desarrollados en base a la creación de un widget padre, con su respectiva estética y comportamiento. Una vez definido este widget padre se procede a crear sus correspondientes hijos y una vez finalizados se los inserta en su antecesor. Estos widgets cuentan con una gran variedad de propiedades que pueden ser modificadas, y los mismos pueden ser administrados mediante la herramienta Qt Designer (figura 4.24).

Como es de notar BEcopter comprende un gran número de widget, lo que causa que la programación de cada elemento se vuelva engorrosa y consuma tiempo, es por eso que el mismo programa brinda una herramienta que traduce los archivos .ui generados por la aplicación a archivos .py, que es nuestro lenguaje seleccionado, con el fin de agregarlo directamente al proyecto. Cabe mencionar que dicha herramienta nos da la facilidad de modificar los aspectos estéticos de los widget mediante CSS, por lo tanto se ha tenido que estudiar dicha sintaxis y utilizarla para el personalizado de los mismos.

4.3.8. Clase BEcopter

Por último tenemos la clase principal BEcopter, esta es la encargada de gestionar las demás clases presentes en este proyecto. Como podemos observar en el diagrama de clases de la Figura 4.2 la misma está relacionada con la gran mayoría por lo que conlleva varias tareas; si detallamos un poco su interacción, esta se encarga de:

- **Gestionar la interfaz gráfica:** La interfaz proporcionada por la clase GUI_becopter únicamente representa la GUI, por tanto, este no contiene ningún tipo de lógica ni tampoco aspectos dinámicos que son de suma importancia para este proyecto. Por lo que es tarea de la clase principal brindarle de dichos comportamientos para poder interactuar con las acciones del usuario; dentro de sus amplias tareas tales como inicialización de estilos, gestión de pestañas, validación de datos, etc. Tenemos las más destacadas que son :

1. **Ocultamiento de opciones:** A modo de seguridad se ha establecido por defecto que al iniciar la aplicación ciertas opciones estén deshabilitadas, como son las pestañas de (Inicio y Configuración) ya que estos dependen de que el vehículo este conectado y en caso contrario, si proporcionamos a que se puedan modificar misiones o parámetros del mismo pueden ocasionar errores.
 2. **Sincronización de datos:** Poder observar los atributos del vehículo en tiempo real es de suma importancia, por lo que se definen *timer's* que sincronizan cada cierto periodo de tiempo los valores ubicados en la pestaña inicio que contiene un conjunto de botones que muestran datos sobre el vehículo y en la pestaña conexión que presenta información de su respectiva conexión y aspectos generales del vehículo.
- **Gestionar los Joysticks:** Como el fin de BEcopter es poder manejar un único vehículo, se ha restringido a la utilización de un solo Joystick, pero

es importante tener en cuenta que pueden haber más de un dispositivo tipo comando conectado al equipo, por lo que se provee de opciones de selección de comandos para contemplar ese aspecto. Además de gestionar la selección del comando a utilizar y realizar el correspondiente mapeo en relación de acción-botón que ha seleccionado el usuario, este es encargado de captar todos los eventos que se están siendo generados en el joystick, con el propósito de enviarlos al vehículo para su posterior interpretación. Esta función es la responsable de realizar el modo manual en caso de ser solicitado.

- **Gestionar los gráficos:** Dentro de lo que es el proyecto BEcopter están incluido un par de gráficos que ayudan al usuario a tener presente información útil del vehículo y sus sensores. Estos gráficos son :

Gráfico Head UP Display: Más allá de su propio funcionamiento interno de la clase HUD, la clase BEcopter es la responsable de poder inicializar la conexión entre el vehículo y dicha clase. Por eso es importante que este proporcione los datos necesarios a mostrar cada vez que el atributo sufra un cambio, por tal motivo, es que se generan *callback's* que administran dichos eventos.

Gráfico de atributos: La clase vehículo nos brinda un gran conjunto de atributos que informan sobre el estado del vehículo, conexión, como también provenientes de los sensores, de tal manera que la clase BEcopter se debe encargar de mostrar dichos atributos en una lista con la opción poder agregarlos a la clase gráfico de datos para su procedente graficación. Además, BEcopter debe proporcionar en dicha lista la opción agregar/eliminar atributos tanto en el gráfico de datos como también en la botonera.

- **Gestionar el vehículo:** La clase vehículo corresponde al corazón del proyecto, ya que este representa de manera directa el UAVs que estamos por manipular, por tal razón es importante tener actualizado todos los valores que proporciona la clase vehículo hacia el usuario; además hay que tener en cuenta que el usuario deseará enviarle comandos al mismo como, despegar, moverse, aterrizar, desplazarse, etc. Ya sea en modo manual mediante el joystick ya seleccionado o en forma de misiones, es decir, en modo automático. De esta manera la clase BEcopter para contemplar estos aspectos tendrá que realizar las siguientes tareas:

Sincronización: La clase BEcopter tiene la responsabilidad informarle a las clases restantes cuando los atributos que están siendo utilizado sufren algún tipo de cambio, de esta manera, se han creado un conjunto de conexiones mediante *callbacks* y *timer's* que envían de manera sincrona y asíncrona la información requerido a cada entidad.

Misiones: Las misiones son un aspecto importante en BEcopter, ya que proporcionan al usuario la posibilidad de crear un conjunto de comandos parametrizables y que pueden ser enviados al vehículo para que sean ejecutadas. En lo que concierne a estas opciones tenemos las siguientes utilidades:

Descargar misiones: Esta opción permite descargar y mostrar en pantalla las misiones que ya tiene almacenado el vehículo. De manera

complementaria, es una herramienta útil para comprobar que las misiones que han sido enviadas de manera correcta.

Agregar misión: Agrega una nueva linea en blanco para el ingreso de siguiente misión.

Enviar misión: Esta opción tiene la particularidad de únicamente cargar las misiones en el vehículo sin la intención de ejecutarlas, ya que puede suceder de enviar misiones incorrectas, por tal motivo BEcopter se encarga de validar dichas misiones y en caso de encontrar un error se le informará al usuario.

Iniciar: Por último y una vez validada las misiones, BEcopter le envía la confirmación al vehículo para que inicie las misiones enviadas.

Estado de conexión: Cumpliendo con el requerimiento R016, que establece la codificación de un script de control del estado de conexión, la clase BEcopter se encarga de este aspecto, consultando periódicamente dicho estado y en caso de perder la conexión (siempre y cuando no se encuentre en vuelo) se bloquearán las opciones pertinentes al vehículo, como si estuviera iniciando la aplicación de nuevo. En caso de existir una perturbación en las variables tales como conexión, GPS y batería, el sistema informará dicho suceso al usuario para que realice las acciones pertinentes.

- **Logging:** Para satisfacer los requerimientos R010 y R011 del documento de especificación de requerimientos la clase BEcopter como también la clase vehículo utilizan la librería **logging** de Python para la generación de un logging ya sea de la plataforma o del vehículo, que se van mostrando en la pestaña de conexión de BEcopter.

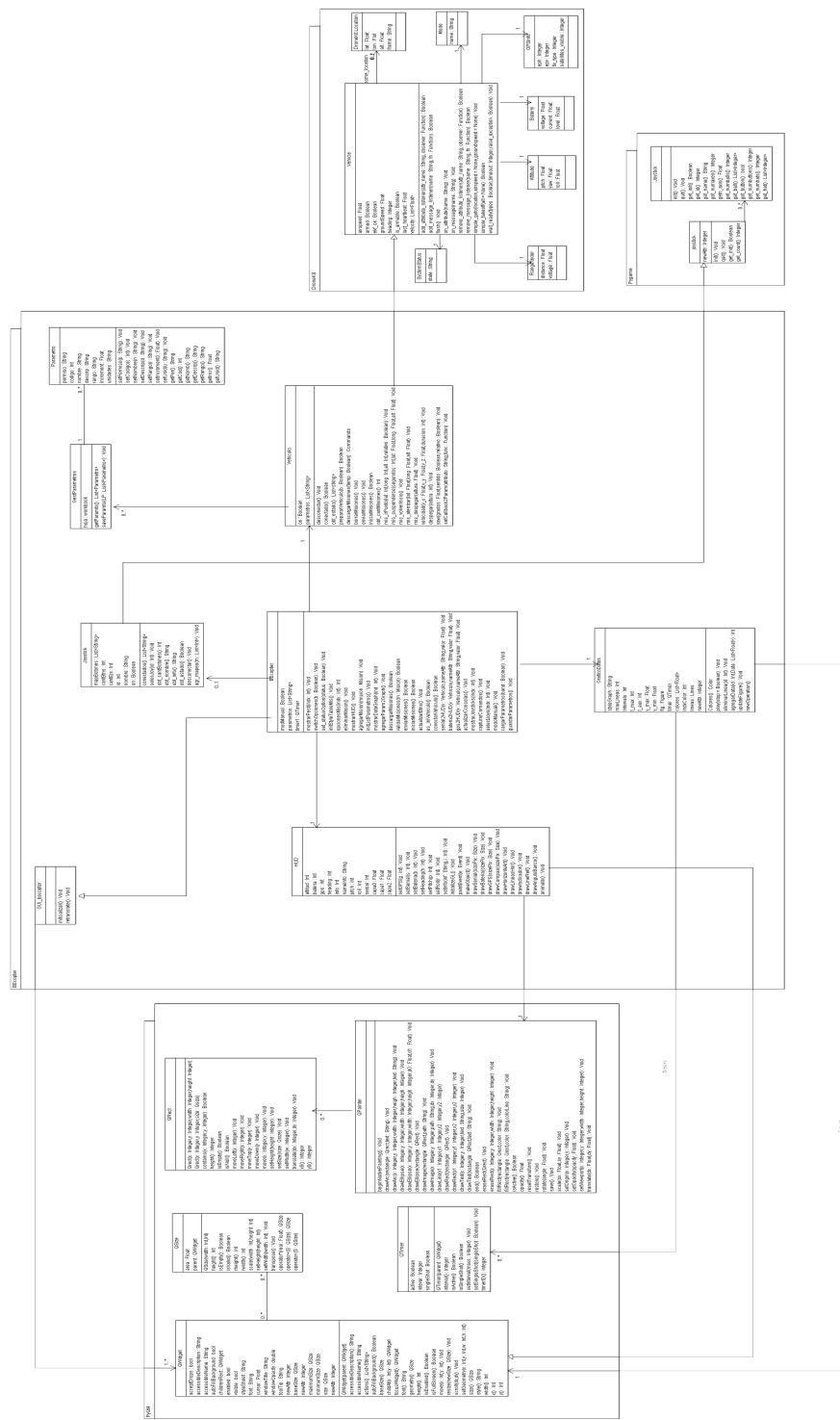


Figura 4.2: Diagrama de clases v2, perteneciente al proyecto BEcoper

Vehículo
on : Boolean
parametros : List<String>
desconectar() : Void
conectado() : Boolean
obt_estado() : List<String>
preparaVehiculo() : Boolean
descargaMisiles(temp : Boolean) : Commands
bombardises() : Void
enviaMisiles() : Void
iniciaMisiles() : Boolean
obt_caracteristicas() : Map<String, Object>
mis_irPunto(x : Int, long : Int, alt : Int, relativo : Boolean) : Void
mis_suspender(segundos : Int) : Float
mis_volverme() : Void
mis_aterrizar(lat : Float, long : Float) : Void
mis_despegarAltura(lat : Float) : Void
velocidadx_y_Float, velocidady_z_Float, duracion : Int) : Void
despegarAltura(alt : Int) : Void
yaw(grados : Float, sentido : Boolean, relativo : Boolean) : Void
setCallBackParam(atributo : String, func : Function) : Void

Figura 4.3: Clase Vehículo

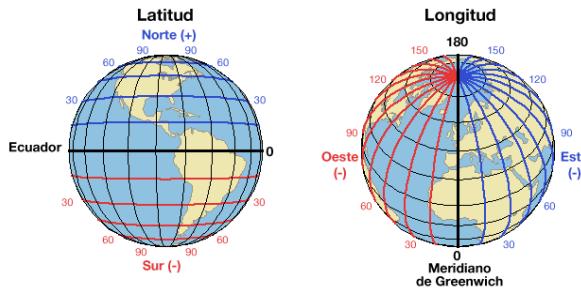


Figura 4.4: Marco de referencia global.

Joystick
mapBotones : List<String>
cantBttns : Int
cantEjs : Int
id : Int
nombre : String
on : Boolean
conectados() : List<String>
selecJoy(d : Int) : Void
obt_cantBotones() : Int
obt_nombre() : String
obt_info() : String
obt_estado() : Boolean
desconectar() : Void
agr_mapeo(m : List<Int>) : Void

Figura 4.5: Clase Joystick

GraficoDatos
styleGraph : String
maxLineas : Int
intervalo : Int
t_max : Int
t_aux : Int
y_max : Float
y_min : Float
fig : Figure
timer : QTimer
colores : List<Float>
indxColor : Int
lineas : Lines
newAtr : Integer
Colores() : Color
play(stop = Boolean) : Void
eliminarLinea(id : Int) : Void
agregarData(id : Int, Data : List<Float>) : Int
updateFigure() : Void

Figura 4.6: Clase gráficos de datos o de atributos del vehículo.

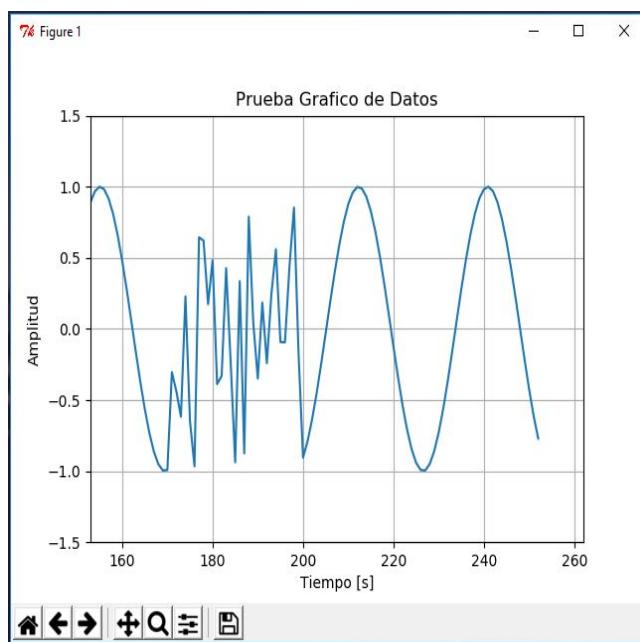


Figura 4.7: Prueba animada en la inserción de dato utilizando la Clase Gráfico Datos.

HUD
<pre> altitud : Int bateria : Int gps : Int heading : Int info : Int nameInfo : String pitch : Int roll : Int senial : Int capa0 : Float capa1 : Float capa2 : Float setGPS(g : Int) : Void setSenial(s : Int) : Void setBateria(b : Int) : Void setHeading(h : Int) : Void setPitch(p : Int) : Void setRoll(r : Int) : Void setInfo(inf : String,i : Int) : Void initializeGL() : Void paintEvent(e : Event) : Void makeObject() : Void drawSenial(sizePix : Size) : Void drawBateria(sizePix : Size) : Void drawGPS(sizePix : Size) : Void drawCompas(sizePix : Size) : Void drawHorizonteArt() : Void drawLineasHor() : Void drawIndicador() : Void drawLineRef() : Void drawAnguloBanco() : Void animate() : Void </pre>

Figura 4.8: Clase Head Up Display



Figura 4.10: Indicador de rumbos

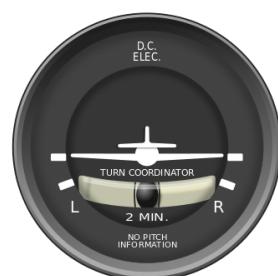


Figura 4.11: Coordinador de giro.

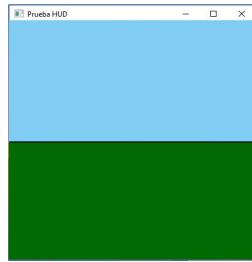


Figura 4.12: HUD v0.1

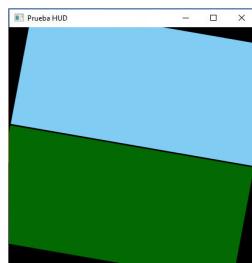


Figura 4.13: Error gráfico al momento de rotar

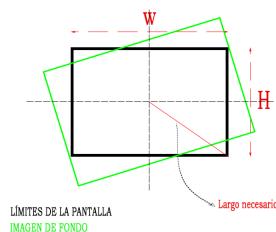


Figura 4.14: Problema gráfico en las esquinas

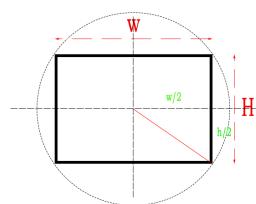


Figura 4.15: Circunferencia con radio que satisface la longitud requerida.

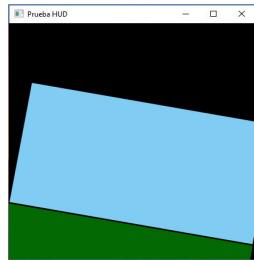


Figura 4.16: Error al momento de simular el Pitch

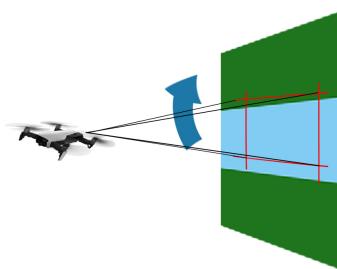
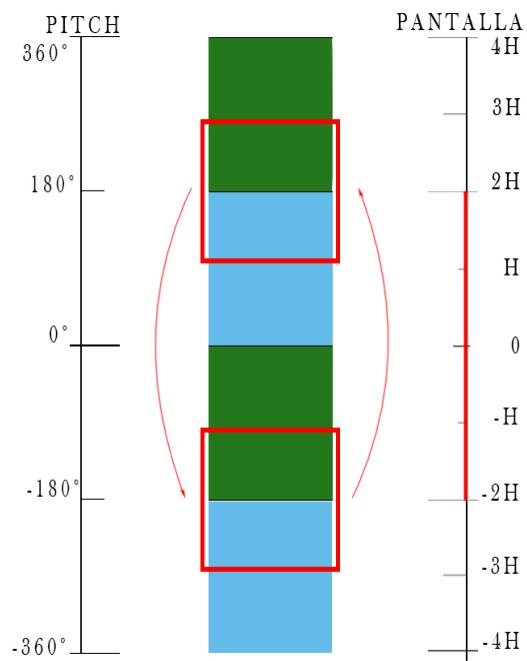


Figura 4.17: Insertando un rectángulo perteneciente al piso.



H = Altura pantalla

Figura 4.18: Mapeo del horizonte artificial.

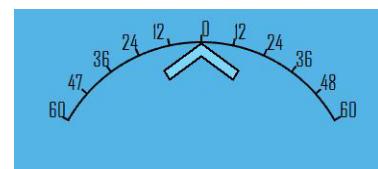


Figura 4.19: Ángulo de banco.

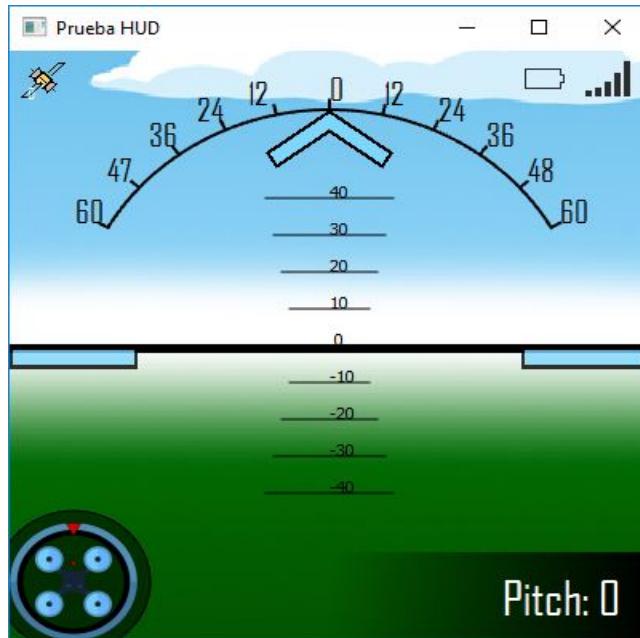


Figura 4.20: HUD final con sus correspondientes atributos e instrumentos de vuelo.

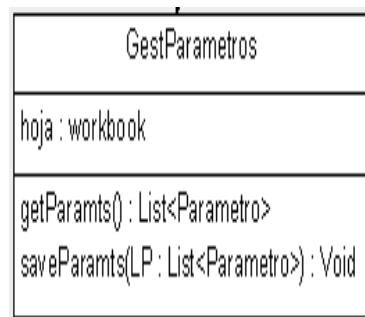


Figura 4.21

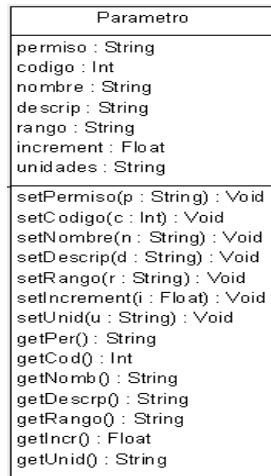


Figura 4.22: Clase parámetro

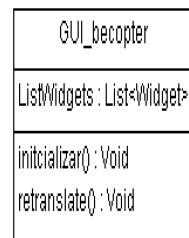


Figura 4.23: Clase Graphical User Interface de BEcopter

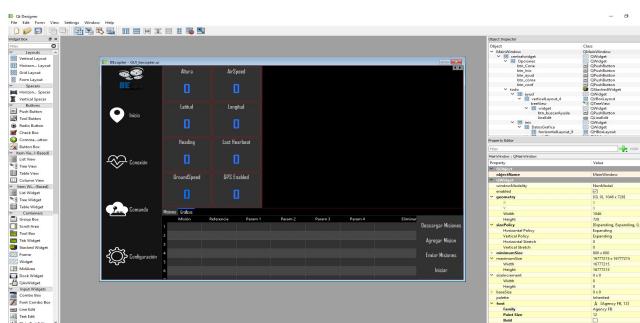


Figura 4.24: Aplicación QT designer

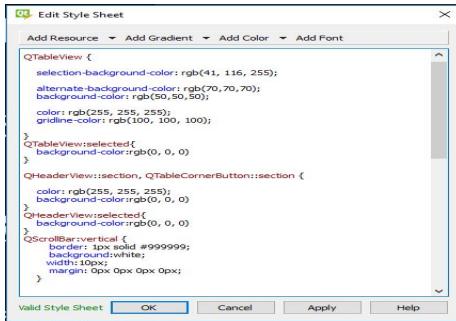


Figura 4.25: Ejemplo del diseño de estilo con CSS.

<pre>BECopter</pre> <pre> modManual : Boolean parametros : List<String> timer1 : QTimer mostrarPest(indx : Int) : Void switchOpciones(b : Boolean) : Void set_statusOption(status : Boolean) : Void initStyleTableMis() : Void opcionesMis(indx : Int) : Int eliminarMision() : Void mostrarHUD() : Void agregarMision(mision : Mision) : Void initListParametros() : Void mostrarDataGraph(ind : Int) : Void agregarParam2Graph() : Void descargarMisiones() : Boolean validarMisiones(m : Mision) : Boolean enviarMisiones() : Boolean iniciarMisiones() : Boolean actualizarBtns() : Void es_iniVehiculo() : Boolean conectarVehiculo() : Boolean bateria2HUD(v : Vehiculo,nameAttr : String,valor : Float) : Void gps2HUD(v : Vehiculo,nameAttr : String,valor : Float) : Void actualizarConsolas() : Void mostrarJoysticks(indx : Int) : Void capturarComandos() : Void selectJoys(indx : Int) : Void modoManual() : Void cargarParametros(band : Boolean) : Void guardarParametros() : Void </pre>

Figura 4.26: Clase principal BEcopter

Parte I

Apéndices

Apéndice A

Documentación

- A.1. Documento de Especificación de Requerimientos
- A.2. Manual de usuario
- A.3. Mockups de la interfaz gráfica del proyecto
- A.4. Diagrama de clases inicial del Proyecto

Parte II

Anexos

