
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 Ingeniero
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

Si caminas solo, irás más rápido; si caminas acompañado, llegarás más lejos.

Proverbio chino

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	1
1.1. Justificación	2
1.2. Objetivos	5
1.3. Alcance	5
1.3.1. Módulo de operación manual	6
1.3.2. Módulo de comunicación inalámbrica	6
1.3.3. Módulo de gráfico de datos	6
2. Obtención y Análisis de requerimientos	7
2.1. Introducción	8
2.2. Diseño global	10
3. Gestión de componentes y armado del cuadricóptero	12
3.1. Hardware	12
3.1.1. Estructura de vehículo	13
3.1.2. Controlador Electrónico de Velocidad	14
3.1.3. Motores sin escobillas	14
3.1.4. Raspberry Pi 3	14
3.1.5. Navio2	15
3.1.6. Arduino UNO	17
3.1.7. Radio Control	17
3.1.8. Joystick	18
3.1.9. Router	18
3.1.10. Módulos XBee	18
3.1.11. Batería LiPo	18
3.2. Procedimiento	19
3.2.1. Armado del cuerpo	19
3.2.2. Armado y configuración del piloto	20
3.2.3. Configuración para el acceso a la red	23
3.2.4. Instalación del firmware	24
3.2.5. Primeras pruebas	24
3.2.6. Conexión de componentes	27

3.2.7. Prueba de funcionamiento inicial	30
4. Desarrollo de la plataforma	32
4.1. Introducción	32
4.2. Diseño detallado de la plataforma	33
4.2.1. Diseño de interfaz	35
4.2.2. Diseño arquitectónico	36
4.3. Codificación y Testing	36
4.3.1. Clase Vehículo	36
4.3.2. Clase Joystick	39
4.3.3. Clase Gráfico Datos	39
4.3.4. Clase HUD	41
4.3.5. Clase Gestor Parámetro	49
4.3.6. Clase Parámetro	51
4.3.7. Clase GUI becopter	52
4.3.8. Clase BEcopter	52
5. Prueba integral de la plataforma y del cuadricóptero	56
5.1. Introducción	56
5.2. Conceptos preliminares	57
5.3. Evaluación funcional del Cuadricóptero y la Plataforma	58
5.3.1. Verificación y validación de la plataforma	58
5.3.2. Verificación y validación del cuadricóptero	60
5.3.3. Prueba integral	68
5.4. Resultados	75
5.5. Conclusión	77
I Apéndices	78
A. Documento de Especificación de Requerimientos	79
B. Manual de usuario	95
C. Mockups de la interfaz gráfica del proyecto	116
D. Interfaz Gráfica del Usuario (GUI)	125
E. Diagrama de clases inicial del Proyecto	126
F. Diagrama de clases final del Proyecto	127
Índice alfabético	129
II Anexos	129

Índice de figuras

1.1.	Aplicaciones agrícolas con drones	2
1.2.	Inspección de tendido eléctrico con drones	2
1.3.	Aplicaciones de emergencia	3
1.4.	Aplicaciones de búsqueda y rescate	3
3.1.	Estructura cuadricóptero de fibra de carbono	13
3.2.	Controlador Electrónico de Velocidad	14
3.3.	Motor sin escobillas	14
3.4.	Raspberry Pi 3	15
3.5.	Navio2	15
3.6.	Placa Arduino UNO	17
3.7.	Comando Radio Control	17
3.8.	Módulo de comunicación XBee	19
3.9.	Batería LiPo	19
3.10.	Cuerpo de cuadricóptero armado	20
3.11.	Esquema de instalación de hélices.	20
3.12.	Conexión entre Raspberry Pi 3 y placa Navio2.	21
3.13.	Pantalla de bienvenida luego de instalar el SO	22
3.14.	Captura de pantalla del Software MavProxy	25
3.15.	Radio Control LANYU model y su correspondiente receptor.	26
3.16.	Ubicación de conexiones en el Rx	27
3.17.	Receptor o Rx de 4 canales	28
3.18.	Codificador PWM a SBUS implementado sobre Arduino.	28
3.19.	Esquema de conexiones	29
3.20.	Conexión de datos a los motores.	29
3.21.	Fuente variable regulable.	30
3.22.	Cuadricóptero armado para verificar funcionamiento.	30
3.23.	Ejecución del modulo de calibración de MavProxy	31
4.1.	Head Up Display	35
4.2.	Clase Vehículo	37
4.3.	Marco de referencia global.	37
4.4.	Clase Joystick	39
4.5.	Clase gráficos de datos o de atributos del vehículo.	40
4.6.	Prueba animada en la inserción de dato utilizando la Clase Gráfico Datos.	40

4.7. Clase Head Up Display	41
4.8. Horizonte artificial indicando un giro a la derecha en descenso.	42
4.9. Indicador de rumbos	42
4.10. Coordinador de giro.	43
4.11. HUD v0.1	43
4.12. Error gráfico al momento de rotar	44
4.13. Problema gráfico en las esquinas	44
4.14. Circunferencia con radio que satisface la longitud requerida.	45
4.15. Error al momento de simular el Pitch	46
4.16. Insertando un rectángulo perteneciente al piso.	46
4.17. Mapeo del horizonte artificial.	47
4.18. Ángulo de banco.	48
4.19. HUD final con sus correspondientes atributos e instrumentos de vuelo.	49
4.20.	49
4.21. Clase parámetro	51
4.22. Clase Graphical User Interface de BEcopter	52
4.23. Aplicación QT designer	52
4.24. Ejemplo del diseño de estilo con CSS.	53
4.25. Clase principal BEcopter	53
 5.1. Plan de QA - Nivel 0	59
5.2. Plan de QA - Nivel 1	59
5.3. Plan de QA - Nivel 2	60
5.4. Esquema de conexiones eléctricas del cuadricóptero	61
5.5. Multímetro digital	62
5.6. Fuente de alimentación para Arduino UNO 7 [Volts] - 800[mA]	63
5.7. Fuente de alimentación Nitto de 12v - 5A	63
5.8. Fuente de alimentación para la Navio2 con 5V y 2500mA	64
5.9. Esquema de conexiones para realizar prueba del motor	65
5.10. Gráfico tiempo-RPM de los motores probados con el script	66
5.11. Conexiones terminadas	67
5.12. Módulo de conexión del proyecto	68
5.13. Módulo de representación de datos	72
5.14. Head Up Display	73
5.15. Configuración del módulo de control manual	74
5.16. Módulo de control automático del vehículo	75
5.17. Pestaña de parametrización del Autopiloto	76
 E.1. Diagrama de clases inicial de BEcopter.	126
F.1. Diagrama de clases v2, perteneciente al proyecto BEcopter	127

Índice de Tablas

5.1.	Suministro energía Arduino UNO	62
5.2.	Suministro energía de motores	63
5.3.	Suministro energía de Navio2	64

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 [8, 9] 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 [2], inspección de líneas eléctricas [4], actividades agrícolas [10], recolección de imágenes [1], seguridad [5], lucha contra incendios forestales [5, 6], 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.

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 D.

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 [3]. En el apéndice E 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.

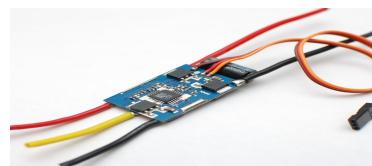


Figura 3.2: Controlador Electrónico de Velocidad

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>

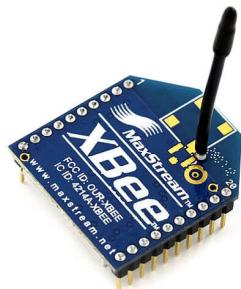


Figura 3.8: Módulo de comunicación XBee



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

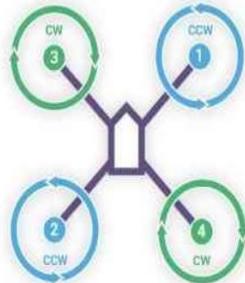


Figura 3.11: Esquema de instalación de hélices.

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-

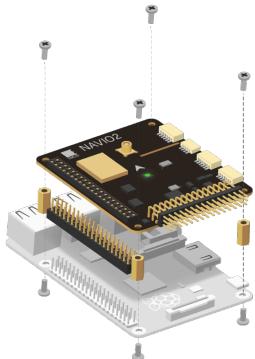


Figura 3.12: Conexión entre Raspberry Pi 3 y placa Navio2.

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:

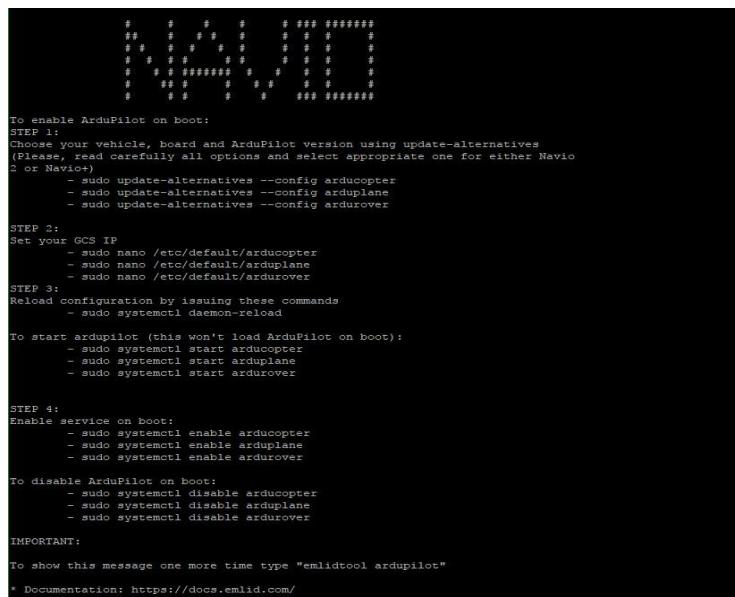


Figura 3.13: Pantalla de bienvenida luego de instalar el SO

```
> 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 *Wireless*. 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

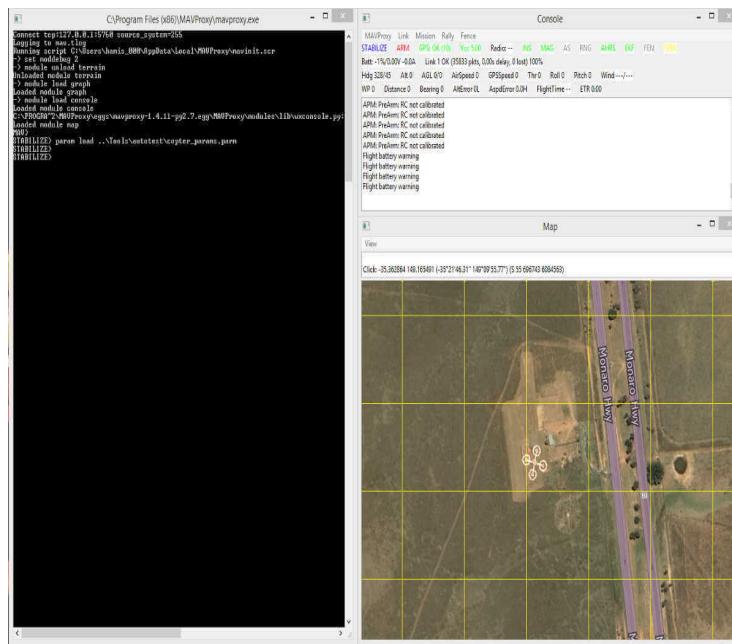


Figura 3.14: Captura de pantalla del Software MavProxy

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

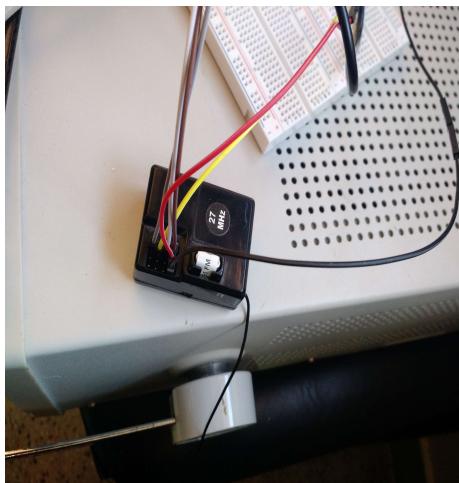


Figura 3.17: Receptor o Rx de 4 canales

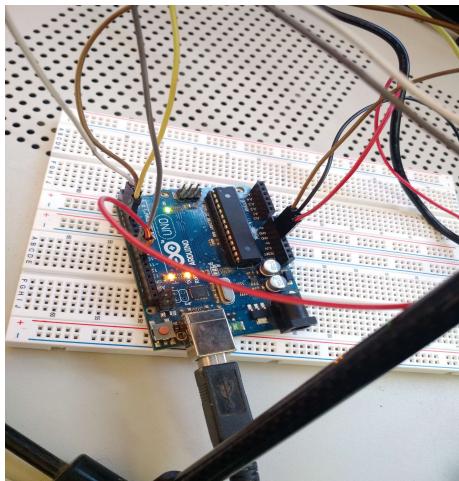


Figura 3.18: Codificador PWM a SBUS implementado sobre Arduino.

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

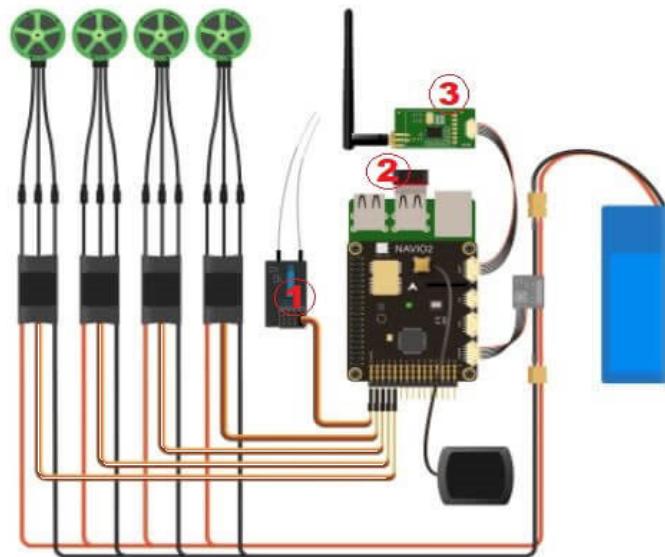


Figura 3.19: Esquema de conexiones

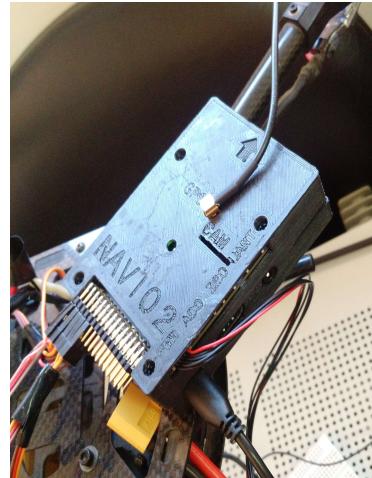


Figura 3.20: Conexión de datos a los motores.

fuentecomo 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 imperfectos en las uniones, en caso de encontrar algún tipo de estos se procede a repararlos.



Figura 3.21: Fuente variable regulable.

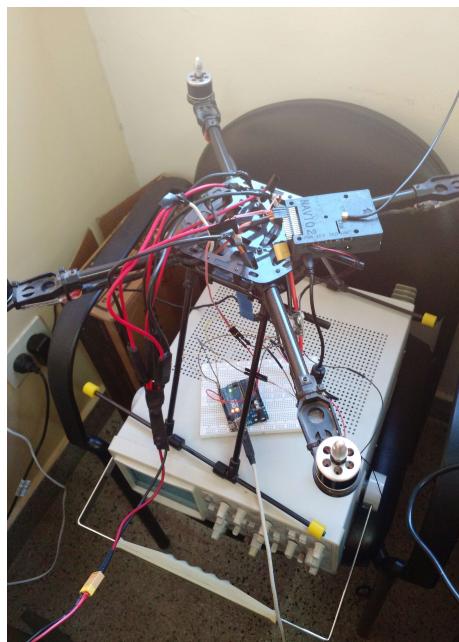


Figura 3.22: Cuadricóptero armado para verificar funcionamiento.

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 5 se profundizará que el funcionamiento integral de los componentes responda de manera correcta.

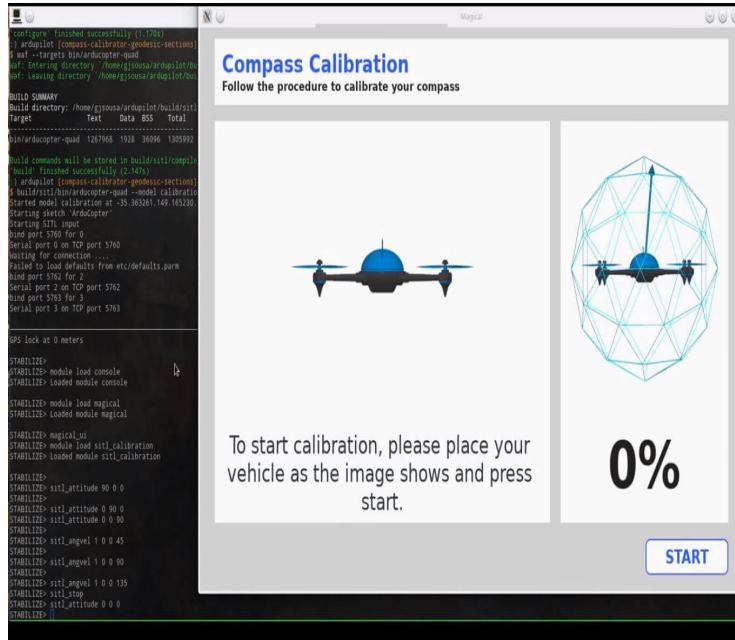


Figura 3.23: Ejecución del modulo de calibración de MavProxy

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:

- **Diseño de datos:** El diseño de datos esencialmente se encarga de transformar el modelo de dominio de la información creado durante el análisis [7]. En el caso particular de este proyecto el diseño de datos no juega un papel determinante dado que la herramienta de software propuesta, de la manera en que sea físicamente desarrollada e implementada, no requiere momentáneamente estructuras de datos complejas, ni de un esquema de bases de datos como ejemplo.
- **Diseño arquitectónico:** En el diseño arquitectónico se definen las relaciones entre los principales elementos estructurales del programa [7]. Cuando se habla de elementos estructurales, se hace referencia entidades lógicas que componen la arquitectura de nuestro diseño, dentro del abanico de posibilidades de diseños, podemos seleccionar, diagramas de flujo de datos, modelos de entidad-relación, diagramas de clases, casos de usos entre otros esquemas gráficos. Viendo todas estas posibilidades, podemos encarar a diseñar nuestra estructura con el modelo que más se ajuste a nuestras necesidades y sea lo más útil posible para las etapas posteriores. Por tal razón, es necesario de un diseño que sea lo más representativo posible en el momento de codificación, que utilice un paradigma orientado a objetos , ya que queremos representar objetos como vehículos, posición geográfica, Joystick, etc; y que además tengan cada uno sus correspondientes atributos, es por eso que seleccionamos un diseño orientado a objetos, utilizando lo que anteriormente llamamos *Diagramas de clases*. Estos diagramas son de suma utilidad ya que muestran de manera fácil como será

la estructura de nuestro código y sus correspondientes relaciones entre cada uno, con el propósito de facilitar la tarea en las etapas posteriores al mismo.

- **Diseño de interfaz:** El diseño de interfaz describe cómo se comunica el software consigo mismo, con los sistemas que operan con él, y con los operadores que lo emplean [7]. Dentro del aspecto de comunicación de software consigo mismo y con otros sistemas es contemplado en el ítem anterior con el tipo de diseño elegido. Aquí lo que nos incumbe es la interacción con el operador del software. De esta manera, se selecciona una herramienta que nos permita realizar nuestra interfaz gráfica del usuario, como por ejemplo, botones, *label*, gráficos, *checkboxes*, etc. ayudando al usuario a realizar sus respectivas tareas sin entrar en tanto detalle en comandos específicos que se debe memorizar con el fin de ser lo más amigable posible.
- **Diseño procedimental:** El diseño procedural transforma elementos estructurales de la arquitectura del programa en una descripción procedural de los componentes del software [7]. Para no ser extensos en lo que es el desarrollo de este proyecto y teniendo en mente que el tamaño del mismo no corresponde a uno de grandes dimensiones y considerando que únicamente existe un único programador para la etapa de desarrollo, se descarta este tipo diseño.

Dentro de lo que es el conjunto de diseños obtenidos en esta etapa es importante saber en qué orden realizar cada una. Ya que cada diseño está 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 cómo se comunicará cada clase con las demás, esta característica ayuda a descubrir qué 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 qué 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 5.14).



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 D

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 ArgoUML¹, obteniendo como resultado el diagrama de clases adjuntado en el apéndice F

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, el diagrama de clases adjuntado en el apéndice F 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 4.3). 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

¹Página web de ArgoUML <http://argouml.tigris.org/>

Vehículo
on : Boolean
parametros : List<String>
desconectar(): Void
conectado() : Boolean
cdt_estado() : List<string>
prepararVehiculo(b : Boolean) : Boolean
descargarMisiones(temp : Boolean) : Commands
borrarMisiones() : Void
enviarMisiones() : Void
iniciarMisiones() : Boolean
cdt_cantMisiones() : Int
mis_irPunto(lat : Int,long : Int,alt : Int,relativo : Boolean) : Void
mis_suspendeder(segundos : Int,lat : Float,long : Float,alt : Float) : Void
mis_volverInicio() : Void
mis_aterrizar(lat : Float,long : Float,alt : Float) : Void
mis_despegar(altura : Float) : Void
velocidad(x : Float,y : Float,z : Float,duracion : Int) : Void
despegar(altura : Int) : Void
yaw(grados : Float,sentido : Boolean,relativo : Boolean) : Void
setCallback2Param(atributo : String,func : Function) : Void

Figura 4.2: Clase Vehículo

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.

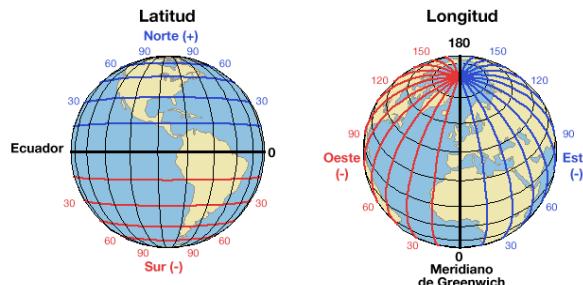


Figura 4.3: Marco de referencia global.

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. **Suspendeder:** 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 envíe 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.

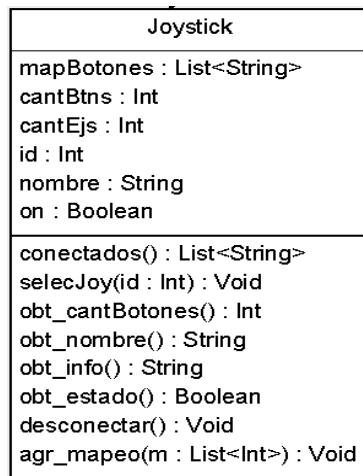


Figura 4.4: Clase Joystick

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-configureado 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

GraficoDatos	
<td></td>	
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	
newAttr : Integer	
Colores() : Color	
play(stop = Boolean) : Void	
eliminarLinea(id : Int) : Void	
agregarData(id : Int, Data : List<Float>) : Int	
updateFigure() : Void	

Figura 4.5: Clase gráficos de datos o de atributos del vehículo.

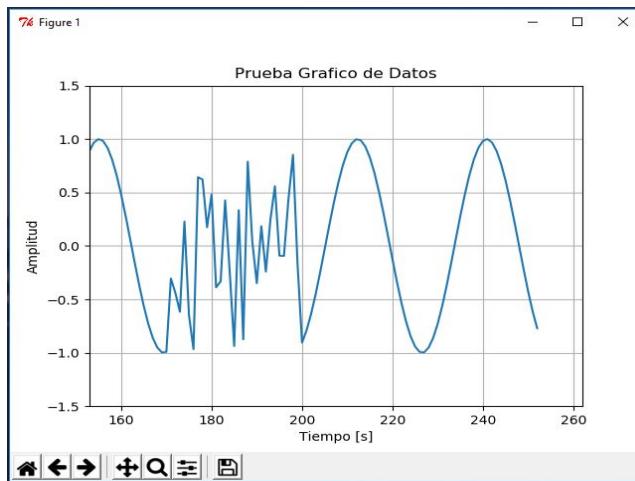


Figura 4.6: Prueba animada en la inserción de dato utilizando la Clase Gráfico Datos.

el acumulamiento de gráficos sobrecargará a la aplicación de recursos a consumir y además dejará 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 lineas:** Para poder gestionar las lineas 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.

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 4.7: Clase Head Up Display

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

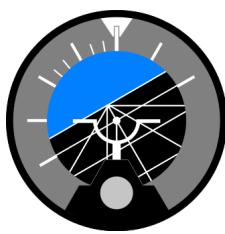


Figura 4.8: Horizonte artificial indicando un giro a la derecha en descenso.

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.



Figura 4.9: Indicador de rumbos

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 den-

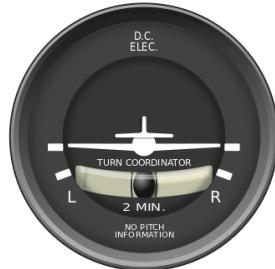


Figura 4.10: Coordinador de giro.

tro 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 mostraran 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 realiza 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 4.8 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 4.11.

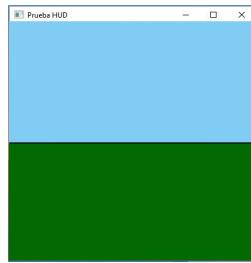


Figura 4.11: HUD v0.1

Hasta aquí, podemos observar que la Figura 4.11 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 4.12 , 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.

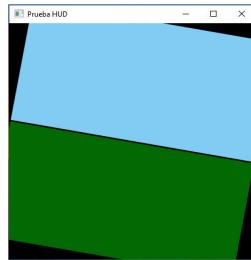


Figura 4.12: Error gráfico al momento de rotar

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

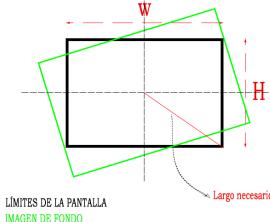


Figura 4.13: Problema gráfico en las esquinas

Como bien muestra la Figura 4.13, 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 4.13 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 :

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

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 4.14, 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.

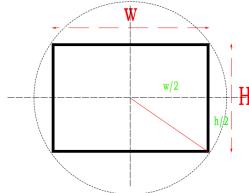


Figura 4.14: Circunferencia con radio que satisface la longitud requerida.

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 4.15. 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

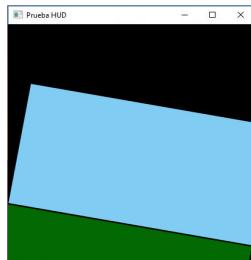


Figura 4.15: Error al momento de simular el Pitch

- 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 cabeceee 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 4.16.

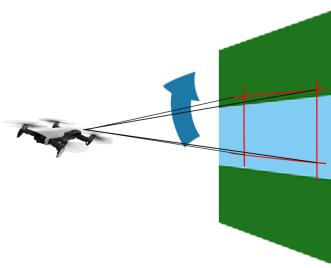


Figura 4.16: Insertando un rectángulo perteneciente al piso.

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.

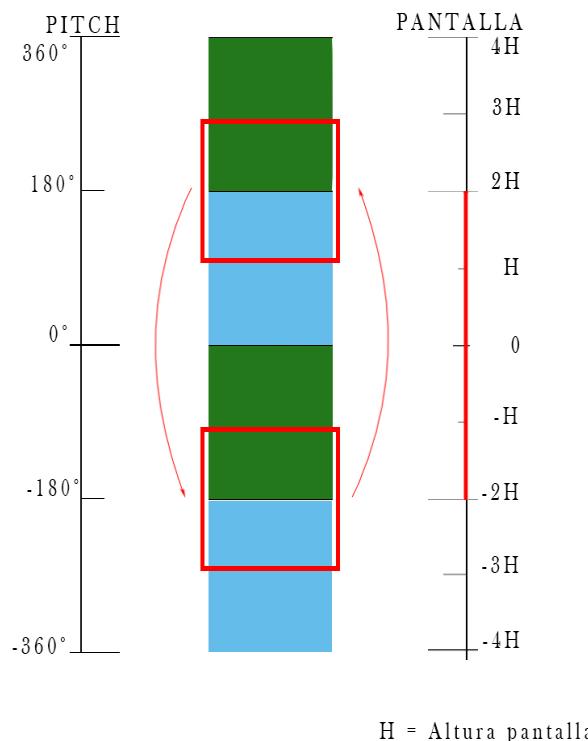


Figura 4.17: Mapeo del horizonte artificial.

Como se puede notar en la Figura 4.17 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 continuidad 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.

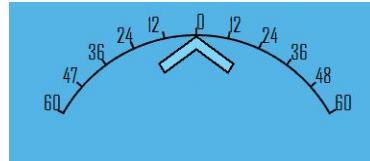


Figura 4.18: Ángulo de banco.

3. **Dibujo de lineas 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 4.19

- **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.

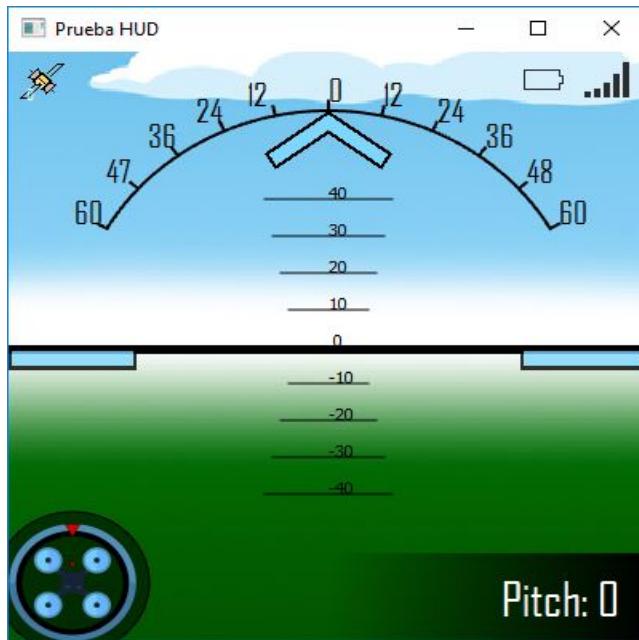


Figura 4.19: HUD final con sus correspondientes atributos e instrumentos de vuelo.

4.3.5. Clase Gestor Parámetro

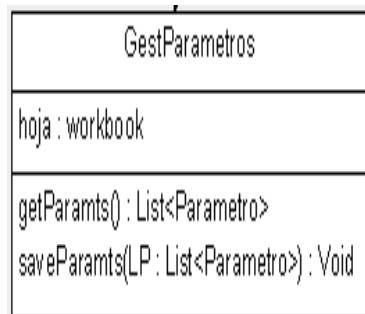


Figura 4.20

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

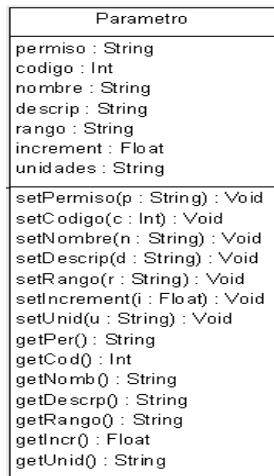


Figura 4.21: Clase parámetro

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

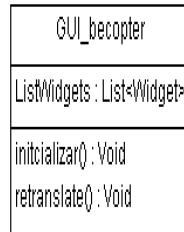


Figura 4.22: Clase Graphical User Interface de BEcopter

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 4.23).

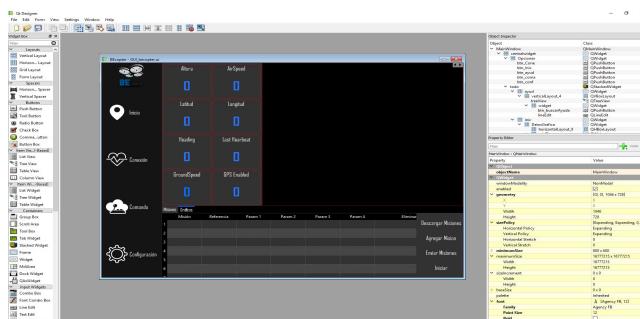


Figura 4.23: Aplicación QT designer

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 4.25 la misma está relacionada con la gran mayoría por lo que conlleva varias tareas; si detallamos un poco su interacción, esta se encarga de:

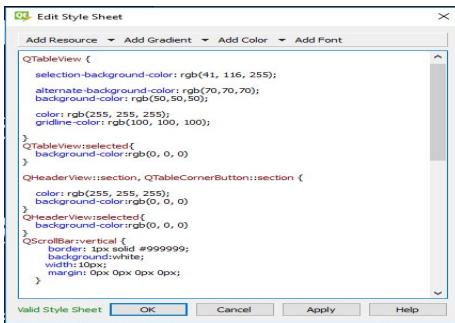


Figura 4.24: Ejemplo de diseño de estilo con CSS.

BEcopter
<pre> modManual : Boolean parametros : List<String> timer1 : QTimer mostrarPest(index : Int) : Void switchOpciones(b : Boolean) : Void set_statusOption(status : Boolean) : Void initStyleTableMis() : Void opcionesMis(index : Int) : Int eliminarMision() : Void mostrarHUD() : Void agregarMision(mision : Mision) : Void initListParametros() : Void mostrarDataGraph(index : Int) : Void agregarParam2Graph() : Void descargarMisiones() : Boolean validarMisiones(m : Mision) : Boolean enviarMisiones() : Boolean iniciarMisiones() : Boolean actualizarBtms() : Void es_inVehicle() : Boolean conectarVehicle() : Boolean serial2HUD(v : Vehiculo, nameAttr : String, valor : Float) : Void bateria2HUD(v : Vehiculo, nameAttr : String, valor : Float) : Void gps2HUD(v : Vehiculo, nameAttr : String, valor : Float) : Void actualizarConsolas() : Void mostrarJoysticks(index : Int) : Void capturarComandos() : Void selectJoys(index : Int) : Void modoManuel() : Void cargarParametros(band : Boolean) : Void guardarParametros() : Void </pre>

Figura 4.25: Clase principal BEcopter

■ **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 asincrona la información requerida 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.

Capítulo 5

Prueba integral de la plataforma y del cuadricóptero

Divide et Impera

Julio Cesar

RESUMEN: En el presente capítulo se describirá todo el proceso realizado de la fase denominada **Prueba integral de la plataforma y del cuadricóptero** que forma parte del actual proyecto. En esta fase se contempla la tarea de *Evaluar el funcionamiento del cuadricóptero y de la plataforma* con el fin de satisfacer los requerimientos estipulados y de manera complementaria que el sistema se comporte de forma estable.

5.1. Introducción

Al momento de encargar el desarrollo de cualquier tipo de proyecto y estemos avanzando en sus respectivas fases es necesario comprobar que el funcionamiento de lo que se va desarrollando esté cumpliendo con los resultados esperados, ya que de no ser así, la continuidad del mismo se verá afectada. Por tal motivo, en las siguientes secciones se detallarán los distintos tipos de pruebas que han sido ejecutadas luego de codificar alguna funcionalidad del mismo. Cabe mencionar que por la naturaleza del proyecto que estamos desarrollando es importante que el mismo se comporte exactamente a los requerimientos estipulados y que además, el software proporcione mecanismos de seguridad en caso de que ocurriese alguna inconsistencia, ya que podría provocar accidentes que dañen el equipo como también a las personas que se encuentren en la intersección de su trayectoria. Dicho esto, introduciremos el concepto de **Aseguramiento de la calidad** en las próximas secciones.

5.2. Conceptos preliminares

El **plan de QA** atraviesa el proceso de desarrollo desde el nacimiento de la idea hasta la implementación del software. En las primeras etapas, verifica que los objetivos estén bien planteados y los requerimientos sean precisos. En las fases de diseño y codificación, vigila el cumplimiento de los estándares fijados. Finalmente, revisa que el software en funcionamiento respete los requerimientos pedidos y que la entrega al cliente se haga en las condiciones adecuadas.

El responsable del QA se tendrá que basar en conjunto de pruebas de calidad, entre las que se incluyen:

- **Testeo unitario:** se prueba que cada módulo funcione bien por separado.
- **Prueba de estrés :** se prueba la resistencia de la aplicación enviándole una cantidad de peticiones excesiva, buscando que colapse.
- **Test de integración:** los módulos probados independientemente durante el testeo unitario se acoplan y se prueban en conjunto.
- **Test funcional:** se prueba que el software ofrezca las funciones solicitadas.
- **Test de aceptación:** el usuario verifica que el producto satisfaga sus expectativas. En este caso, como se trata de una necesidad que inicialmente ha surgido de integrantes del *sinc(i)* el software será validado por dichos responsables y una vez que el producto esté finalizado, y con el objetivo de que éste cumpla de manera eficaz con todas sus funcionalidades, el mismo será publicado en una plataforma de desarrollo colaborativo con el fin de que éste realice un *feedback* de las inconsistencias y/o errores del mismo.

Las pruebas de QA no sólo son beneficiosas para el usuario final que recibirá un producto de calidad, sino también para el ejecutor del proyecto, ya que al establecer un control permanente sobre el proceso evitará en buena medida los costos de tener que corregir errores en etapas avanzadas del proyecto.

5.3. Evaluación funcional del Cuadricóptero y la Plataforma

5.3.1. Verificación y validación de la plataforma

Para el desarrollo del proyecto, se ha utilizado un paradigma de programación *orientado a objetos*. Este paradigma nos ha facilitado la división lógica de las funcionalidades y por lo tanto, nos ha sido de gran facilidad probar cada componente lógico una vez codificado. Para validar y verificar que las funcionalidades cumplan con las especificaciones del cliente; se realiza el siguiente **plan de QA**.

En primera instancia se valida con el documento de requerimientos y con el diagrama de clases generado en etapas anteriores. Se examina de manera general que cumpla con las necesidades de los usuarios y, en caso de encontrar inconsistencias que no cumplen completamente dicha necesidad se procede a modificar el diagrama de clases. Una vez codificadas, se procede a realizar los siguientes niveles de prueba:

- **[Descripción] Nivel 0** Por funcionalidad se implementan las pruebas unitarias, realizando el siguiente procedimiento:

Se ingresan datos reales o verdaderos en las entradas de las funciones, y luego de ejecutarla, observamos si obtuvimos los resultados deseados. A partir de aquí podemos obtener 2 casos, que la función se comporte según lo esperado o que la misma contenga errores. En caso de contenerlos se procede a solucionarlo siempre y cuando al momento de obtener dicha solución la misma no conlleve demasiado tiempo y no sea dependiente de otras tareas; se registra dicho inconveniente con su respectiva fecha de incidencia y sus posibles riesgos para ser resuelta en etapas posteriores. Por otro lado y una vez obtenido los resultados con los datos verdaderos, se procede a cambiar el tipo de entrada por datos erróneos o posibles alternativas, como por ejemplo , datos numéricos en vez de texto, caracteres especiales, datos fuera de rango, etc. con el fin de “proporcionar robustez a nuestro código” (prueba de estrés) y adaptar el código para mantener el control en caso de ocurrir algún tipo de inconsistencia. Cabe mencionar, que sólo se suspenden temporalmente funcionalidades, si se consideran de tipo *no funcionales* o no resuelve completamente un requerimiento del usuario. En la Figura 5.1 se esquematiza con un diagrama de flujo este procedimiento.

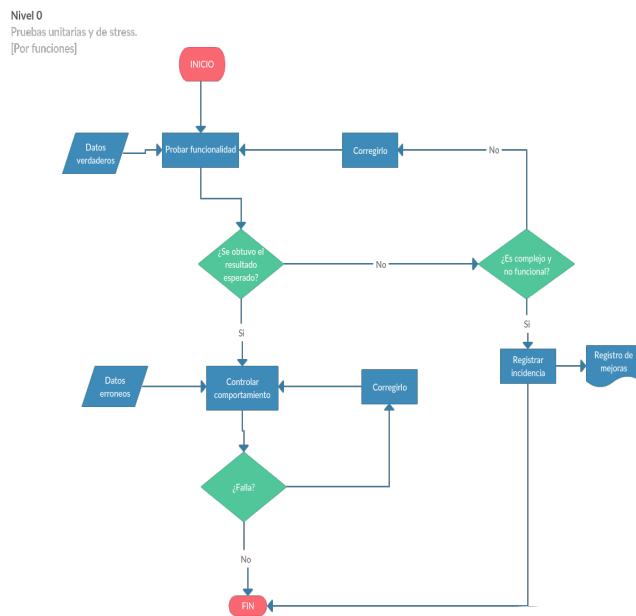


Figura 5.1: Plan de QA - Nivel 0

- **[Descripción] Nivel 1** En este nivel lo que se busca es realizar los test de integración y funcionales (una vez ya finalizados los test del nivel 0), para esto se procede de la siguiente manera:

En el mismo módulo, se instancia cada clase perteneciente al mismo y se prueban las funcionalidades que dependen de otras clases (ya que las funcionalidades que no dependen de otras ya han sido probadas) y se toma el mismo procedimiento por funcionalidad que en el Nivel 0.

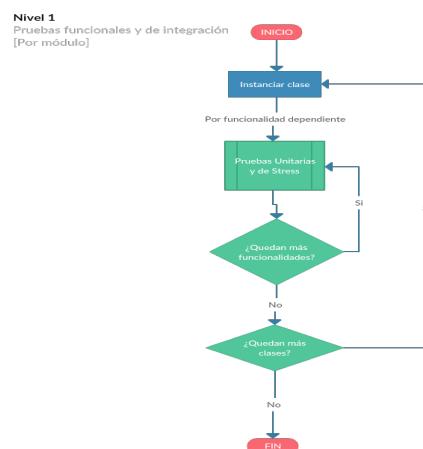


Figura 5.2: Plan de QA - Nivel 1

- **[Descripción] Nivel 2** Este nivel es el último que se realiza, tiene como principal objetivo cumplir con los test de aceptación (considerando temporalmente que el programador es el usuario final):

para realizar esto se tiene como prioridad que la interacción entre el vehículo y el software se comporte correctamente, con esto hacemos referencia a que se muestren los resultados correspondientes, como señal de GPS, batería desenchufada, vehículo desconectado de la red, que el recibimiento de los comandos se puedan comprobar con las revoluciones de los motores, etc. En caso de encontrar algún tipo de falla o desperfecto, nos dirigimos directamente hasta el nivel 0 y desde ahí se procede a repetir el procedimiento.

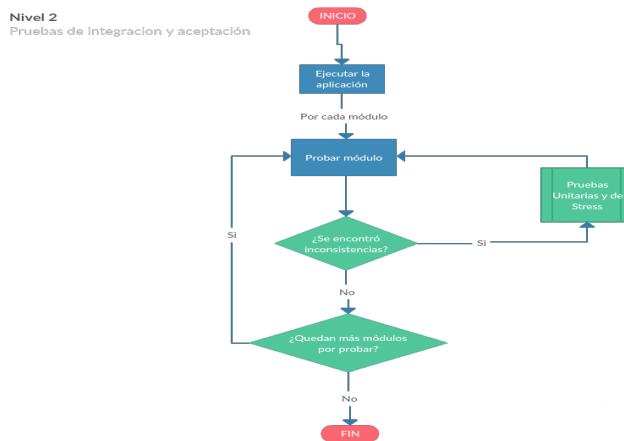


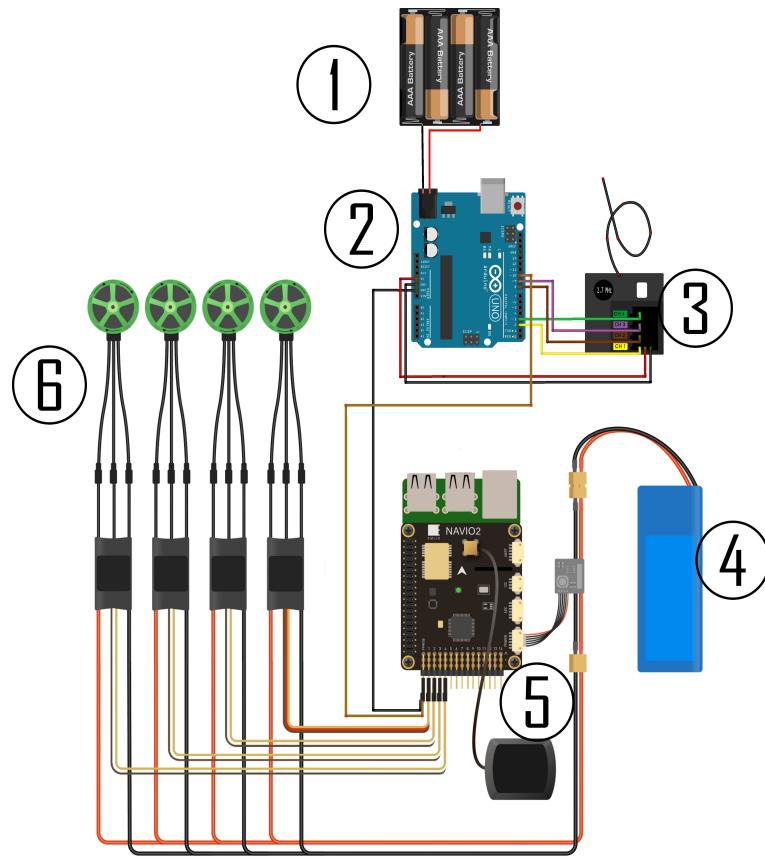
Figura 5.3: Plan de QA - Nivel 2

5.3.2. Verificación y validación del cuadricóptero

Más allá que el concepto del plan de QA se aplique comúnmente en el desarrollo de software, es posible crear un punto de intersección y adoptar la misma estructura lógica en donde se realizan los distintos tipos de pruebas que en el plan creado; pero en este caso sobre elementos electrónicos. Con esto hacemos referencia, que podemos ver a cada componente electrónico como un módulo, por el cual nosotros como QA lo conceptualizamos como una **caja negra**, en donde ingresan datos de entrada y deberíamos obtener determinados datos de salida. Estos datos pueden ser obtenidos por sus correspondientes fabricantes que proporcionan en su gran mayoría el *data-sheet* de cada componente. Por lo tanto en lo que concierne a esta sección se explicará el procedimiento utilizado para probar el correcto funcionamiento de cada componente antes de ponerlos en marcha de manera conjunta.

5.3.2.1. Conectores

Los conectores o cables, son los encargados de comunicar o transportar las señales a cada entrada/salida entre componentes electrónicos, por tal razón



fritzing

Figura 5.4: Esquema de conexiones eléctricas del cuadricóptero

es de suma importancia evaluar su estado, ya que cumplen un rol importante al transportar la información. Para determinar el estado de cada conector se utilizó el tester en modo “comprobación de continuidad”, como se puede ver en la Figura 5.5, el cual conectando las puntas del cable con las puntas del tester, nos indicará con un sonido cuando el cable se encuentre en buenas condiciones, en caso contrario se intenta repararlo o desecharlo en el peor de los casos.

5.3.2.2. Fuentes de alimentación

En lo que respecta a las pruebas, hay que hacer foco en que es necesario disponer de un uso considerable de los componentes, ya que probar cada funcionalidad conlleva realizar pruebas continuas y por lo tanto al ser componentes electrónicos el suministro de energía en esta etapa deben ser , “ilimitado”. Es por tal razón que la portabilidad que disponemos al tener baterías es innecesaria (más allá que serán útiles al momento de realizar vuelos a distancia), por lo tanto estas baterías son reemplazadas por fuentes transformadoras de energía,



Figura 5.5: Multímetro digital

las cuales dependen del suministro de energía de tipo alterna proveniente de las instalaciones del domicilio del ejecutante, y de esta manera obtendremos independencia en en tiempo de uso. En la Figura 5.4 los componentes etiquetadas con los números 1 y 4 corresponden a las baterías que deben tener las siguientes especificaciones según los dispositivos electrónicos que alimentan:

- **Batería 1:** La misma provee de energía a la placa Arduino UNO. Por lo tanto, según las especificaciones de Arduino debe recibir los siguientes suministros¹ :

Especificaciones Arduino UNO	
Voltaje operativo	5 [Volts]
Entrada de voltaje (recomendado)	7 - 12 [Volts]
Entrada de voltaje (límites)	6 - 20 [Volts]

Tabla 5.1: Suministro energía Arduino UNO

Para reemplazar esta batería se ha utilizado una fuente transformador de 7 Volts y 800mA, como se puede observar en la Figura 5.6

- **Batería 4:** La segunda batería como se puede ver en la Figura 5.4 provee de energía a los 4 motores y además alimenta de manera paralela a la placa Navio2/Raspberry Pi. Y por el otro lado tenemos los motores HobbyKing² que reciben los suministros mostrados en la Tabla 5.2

Para reemplazar la alimentación proveniente de dicha batería se ha adquirido una fuente de alimentación como se puede observar en la Figura 5.7, la cual será responsable de alimentar a los 4 motores y como se explicará más adelante al Radio Control.

¹Especificaciones de Arduino UNO <https://store.arduino.cc/usa/arduino-uno-rev3>

²Página web del fabricante hobbyking.com/es_es/hobbyking-30a-blueseries-brushless-speed-controller.html



Figura 5.6: Fuente de alimentación para Arduino UNO 7 [Volts] - 800[mA]

Especificaciones motor	
Voltaje de entrada	7.4 - 14.8 [Volts]
Baterías	2-4 célula Lipo / 5-12 de la célula de Ni-XX

Tabla 5.2: Suministro energía de motores

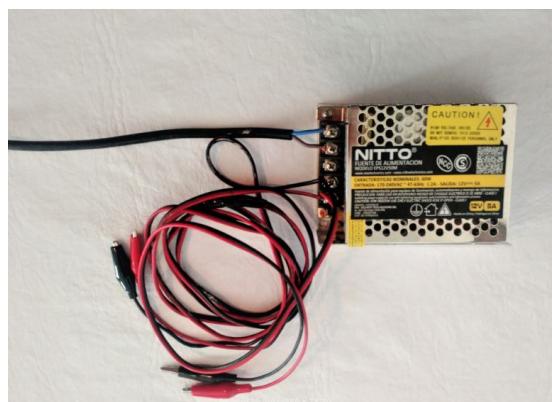


Figura 5.7: Fuente de alimentación Nitto de 12v - 5A

Por último debemos alimentar a nuestra Raspberry Pi / Navio2 ³ que recibe los valores mostrados en la Tabla 5.3

En este caso para alimentar a las placas se nos ha suministrado un transformador incluido en la compra de nuestra Raspberry Pi 3 con las siguientes especificaciones: 5V y 2500mA, como se puede observar en la Figura 5.8.

Para comprobar que los respectivos valores de salida de cada fuente sean correctos y no provengan con algún tipo de falla técnica, se procede a medir los voltajes correspondientes con un tester, dándonos como resultado valores dentro del rango de especificaciones de fábrica.

³<https://emlid.com/navio/>

Especificaciones Navio2	
Entrada de voltaje	4.75 - 5.25 [Volts]
Consumo promedio	150 [miliAmperes]

Tabla 5.3: Suministro energía de Navio2



Figura 5.8: Fuente de alimentación para la Navio2 con 5V y 2500mA

5.3.2.3. Radio Control

La comprobación del radio control en primera instancia se reemplazan las 8 baterías de 1.5v, con lo cual tenemos una cantidad suministrada de 12V necesarios para alimentarlo, siendo justamente la misma cantidad de Voltaje proveniente de nuestra fuente de alimentación de la Figura 5.7. Luego de conectar la fuente se procede a controlar que las señales enviadas por los movimientos del Joystick de control lleguen de manera correcta al receptor de señal. Para realizar dicho procedimiento se debería medirlo mediante un osciloscopio y ver de manera precisa la forma de la señal. Pero al no contar con dicho instrumento se analizan los pines con el tester midiendo el voltaje de la corriente continua, y se observa que al producir movimientos en los joystick cada pin en promedio muestra un voltaje cercano a los 5 Volts. Exceptuando el primer pin (el que corresponde al Eje x del primer joystick) donde no se observan valores de voltaje significativos. De esta manera, se tendrá en cuenta dicha falencia para las próximas pruebas.

5.3.2.4. Motores

En la prueba de motores se tuvo que codificar un script ?? para que sea ejecutado sobre la placa Arduino. Una vez codificado dicho script se procedió a realizar sus respectivas conexiones como se puede observar en la Figura 5.9.

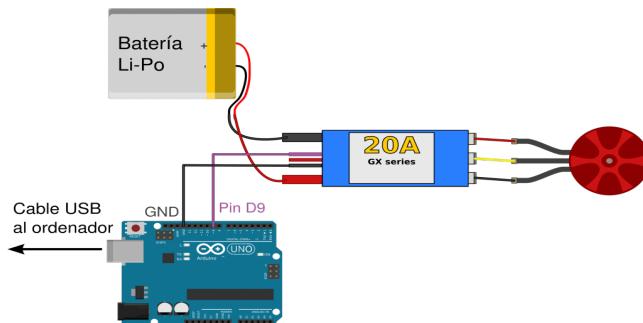


Figura 5.9: Esquema de conexiones para realizar prueba del motor

```

1  /* Programacion de un ESC con Arduino
2  *
3  * La velocidad del motor puede cambiarse enviando
4  * un entero entre 1000 (vel. minima) y 2000 (vel. max.)
5  * por Serial.
6  */
7  */
8  #include<Servo.h>
9
10 //Crear un objeto de clase servo
11 Servo ESC;
12
13 //Amplitud del pulso
14 int vel = 2000;
15
16
17 void setup()
18 {
19     //Asignar un pin al ESC
20     ESC.attach(9);
21
22     //Activar el ESC -> 1000 = 1ms
23     ESC.writeMicroseconds(1000);
24
25     //Esperar 5 segundos para hacer la activacion
26     delay(5000);
27
28     //Iniciar puerto serial
29     Serial.begin(9600);
30     Serial.setTimeout(10);
31 }
32
33
34 void loop()
35 {
36     if(Serial.available() >= 1)
37     {

```

```

38     vel = Serial.parseInt(); //Leer un entero por serial
39     if(vel != 0)
40     {
41         //Generar un pulso con el numero recibido
42         ESC.writeMicroseconds(vel);
43     }
44 }
45 }
```

En la Figura 5.10 (que fue obtenida mediante el módulo de representación de datos en etapas posteriores) se puede observar una gráfica de tiempo vs rpm

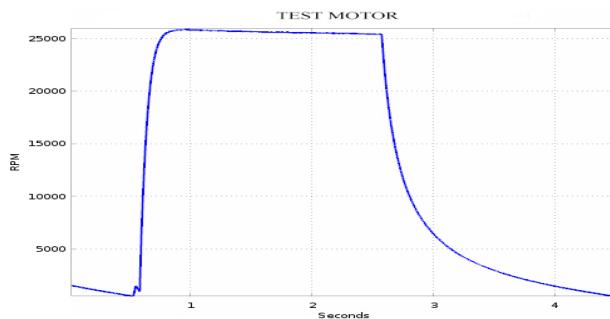


Figura 5.10: Gráfico tiempo-RPM de los motores probados con el script

5.3.2.5. Raspberry Pi - Navio2

La prueba de funcionamiento de dichas placas se comprueba con la interacción con el software desarrollado.

5.3.2.6. Arduino

En lo que compete a la placa Arduino, en el esquema presentado en la Figura 5.4 la misma cumple la función de un conversor PWM a PPM, el cual es el formato de señal necesario para que la placa Navio2 interprete las señales enviadas por el radio-control. En nuestro caso, ya se han probado los dispositivos conectados al mismo, por lo tanto sólo nos queda corroborar que el tipo de señal enviada sea el correcto. Para realizar esta prueba es necesario contar con un osciloscopio, que muestre el tipo de señal enviada por el pin configurado en la placa Arduino. Como no disponemos momentáneamente de un osciloscopio, se recurre a la simulación; de manera que se utiliza el software de la siguiente página web www.library.io, el cual proporciona un simulador de código y además nos da la posibilidad de mostrar la señal generada por un osciloscopio digital. Luego de corroborar que la señal es correcta mediante el programa de simulación, se mide mediante un tester que la corriente promedio de salida se mantenga en un valor aceptable de voltaje para comprobar que la placa este funcionando correctamente. Por último, en la Figura 5.11 se puede apreciar todo el sistema conectado según el diagrama de la Figura 5.4. Exceptuando los motores, que han sido debidamente desconectados al momento de realizar las pruebas; y únicamente se ha conectado un motor sobre una base de madera,



Figura 5.11: Conexiones terminadas

con el fin de mantener el mismo inmóvil al momento de enviar las órdenes de arrancar.

5.3.3. Prueba integral

Por último y ya probado que tanto la plataforma como el vehículo estén en condiciones, se procede a realizar las pruebas integrales entre ambas partes (Proyecto-Vehículo). Para esto se descomponen dichas pruebas en categorías o "módulos" que se han definido en el comienzo del proyecto por los *stakeholders*.

5.3.3.1. Conexión

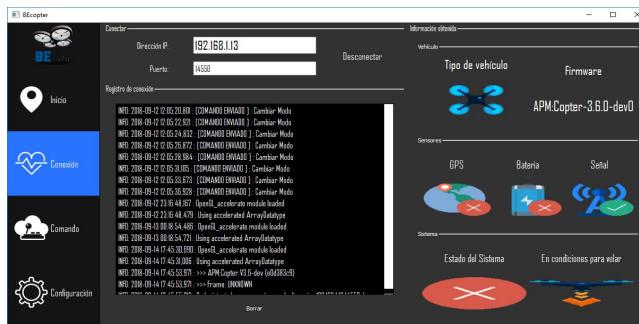


Figura 5.12: Módulo de conexión del proyecto

El módulo de conexión es el encargado de habilitar y mantener toda la comunicación entre el vehículo y nuestro software, por tanto éste debe ser lo bastante robusto para interpretar los principales errores producidos por la red de comunicación elegida. El siguiente código muestra la funcionalidad principal que se encarga de conectarse con nuestro vehículo y mediante los try-except gestiona el control de errores proporcionando al usuario (en caso de generarse alguno) su respectivo mensaje informativo.

```
1 try:
2     if port_host != None:
3         # Concatenamos la direccion IP con el puerto
4         source = ip_host + ":" + port_host
5     else:
6         # En caso de entrar aqui es porque se ha ingresado un puerto
7         # serial
8         source = ip_host
9     # Utilizamos la funcion connect de DroneKit
10    self.v = connect(source,
11                      wait_ready=True,
12                      heartbeat_timeout=10,
13                      status_printer=logging.info)
14    # El parametro wait_ready = True garantiza que connect () no
15    # volvera
16    # hasta que Vehicle.parameters y la mayoria de los otros
17    # atributos
18    # predeterminados se hayan rellenado con valores del
19    # vehiculo.
```

```

17     logging.info("Se ha conectado con la direccion {} de manera
18         exitosa".format(source))
19
20     self._on = True
21     self.descargarMisiones(False)
22
23     # Error TCP connection
24     except socket.error:
25         logging.info('No se ha encontrado el vehiculo, por favor
26             compruebe la conexion')
27         raise WindowsError('No se ha encontrado el vehiculo, por
28             favor compruebe la conexion')
29
30     # Error TTY connection
31     except exceptions.OSError as e:
32         logging.info('No se pudo establecer conexion con el puerto
33             serial')
34         raise WindowsError('No se pudo establecer conexion con el
35             puerto serial')
36     return
37     except APIException:
38         logging.info('Se ha exedido el tiempo de espera para
39             establecer la conexion!')
40         raise WindowsError('Se ha exedido el tiempo de espera para
41             establecer la conexion!')
42
43     except :
44         logging.info('Ha ocurrido un error inesperado. ')
45         raise WindowsError('Ha ocurrido un error inesperado. ')

```

Según el código mostrado, podemos realizar una conexión con nuestro vehículo mediante un dispositivo que se conecte por puerto serial o a través de la red, ya sea mediante Ethernet o Wireless (utilizando UDP).

- **Conexión mediante el puerto serial:** Para comprobar el funcionamiento, se han realizado anteriormente pruebas de conexión utilizando el protocolo SSH, el cual nos permite interactuar mediante una consola de comandos con nuestro vehículo, dándonos los datos solicitados sin ningún tipo de problema. Cabe mencionar que el objetivo de este tipo de conexión es utilizar los módulos inalámbricos XBee/LoRa.
- **Conexión mediante la red local:** Por comodidad al momento de transportar el vehículo en los lugares en que la fuente de alimentación necesitaba conectarse, este tipo de conexión nos ha resultado la más beneficiosa. El mismo utiliza un Router TP LINK TL-WR841N con una cantidad de 10 clientes conectados de manera inalámbrica en el domicilio utilizado. Además, se han realizado las mismas pruebas pero dentro de las instalaciones del *sinc(i)* con una cantidad de 50 clientes conectados aproximadamente al mismo router de pruebas. Y para este caso, la conexión sufrió de varias interrupciones continuas. Por lo que obtenemos como conclusión para estos casos que el router utilizado para el envío de comandos no debe ser utilizados por muchos usuarios, que de ser así la conexión no será la óptima.

Pruebas de interrupción de conexión

Un aspecto importante a tener en cuenta, es el momento en que la conexión se corta. Por tal inconsistencia que puede ocurrir fácilmente se han realizado pruebas simulando dicho fenómeno y comprobando cómo se comporta el software y el vehículo al ocurrir este hecho. Cuando este inconveniente se presente, tanto el vehículo como el software comparten una variable en común llamada *last heartbeat*, la cual contiene información sobre el tiempo (en segundos) en que ha recibido su último mensaje proveniente del vehículo.

```

1 # Testing on SITL indicates that last_heartbeat averages
2     about .5 seconds,
3 # but will rarely exceed 1.5 seconds when connected.
4 # Whether heartbeat monitoring can be useful will very much
5     depend on the application.

6 if self.v.last_heartbeat > 1 and self.v.last_heartbeat <=
7     1.5 :
8     estados['signal'] = None
9 elif self.v.last_heartbeat <= 1 and self.v.last_heartbeat :
10    estados['signal'] = True
11 else:
12     estados['signal'] = False

```

En el código anterior se puede observar que existe una variable correspondiente al estado de la señal, el mismo puede contener 3 valores: [False, None, True]. En caso de tener el valor de False, esto quiere decir que la conexión se ha perdido, ya que se ha superado el rango establecido; cuando el valor de *last heartbeat* está comprendido entre 1 y 1.5 segundos, tendrá un estado None, el mismo significa que está dentro de valores de precaución ya que está recibiendo mensajes con un poco de demora. Y por último en caso de estar en valores normales de tiempo tendrá un valor de True, indicando que estamos recibiendo información a tiempo del vehículo, esta estructura simula el comportamiento de un semáforo.

- **Comportamiento del vehículo:** Como mencionamos, la variable *last heartbeat* es compartida entre el software y el vehículo. Por lo tanto cuando el vehículo detecte que se ha perdido la conexión tenemos 3 posibilidades:

1. **Aterrizar**
2. **Volver al inicio**
3. **Quedarse suspendido en el aire**

Por defecto el vehículo estará en suspensión en el aire (siempre y cuando el nivel de batería lo permita) y en caso de agotar la batería, se procederá a aterrizar el vehículo. Se procede de esta forma ya que se considera que el vehículo puede encontrarse en una zona no apta para aterrizar, como por ejemplo sobre una edificación, un río, árbol, etc. El mismo puede llegar a estropearse, por lo tanto seguirá pendiente hasta que la conexión se restablezca y en el último de los casos se procede a aterrizar (ya que se

supone el problema de que el equipo de comunicación se estropee o el mismo software cierre inesperadamente su ejecución por algún motivo). Es importante tener en cuenta que estas acciones pueden ser cambiadas en la respectiva configuración de la aplicación.

- **Comportamiento del software:** Cuando este inconveniente se presente, el software, en primera instancia, tratará de reconnectarse por un tiempo estipulado (5 segundos) y en caso de no poder restablecerá el estado del programa al mismo que cuando se inicia la aplicación en primera instancia, con el objetivo de deshabilitar las funciones dependientes del vehículo y dejar a disposición del usuario volver a conectarse en caso de ser necesario.

5.3.3.2. Representación de datos

La visualización de los datos es un aspecto importante cuando estamos controlando algún objeto a distancia, ya que es necesario conocer el estado del mismo, y con mayor importancia cuando el objeto no está en nuestro campo visible. Por tal motivo se han realizado varias pruebas sobre los módulos de representación de datos, con el propósito de que éstos muestren fielmente los datos capturados por el vehículo en tiempo real.

- **Gráfico temporal de datos:** El objetivo de este módulo de software en nuestro proyecto, como su nombre lo dice, es poder representar la evolución temporal de los datos provenientes de algún sensor o estado del vehículo mediante una gráfica.

Según lo especificado en el ERS, se ha propuesto utilizar la librería Vispy⁴, esta librería escrita en Python está diseñada específicamente para la visualización interactiva de gran cantidad de datos de forma rápida, escalable y fácil. Pero más allá de sus prestaciones, al momento de realizar las **pruebas de integración** se ha observado que el rendimiento de nuestro software se ha reducido considerablemente; esto es debido a que Vispy utiliza distintas tecnologías al renderizar las figuras, por lo que se debió incrustar dicho gráfico de manera brusca para poder ver su funcionamiento. Al momento de querer optimizar dicho rendimiento, al ser Vispy una librería nueva, su documentación y por lo tanto su comunidad era escasa, y como es de esperar no se encontraba la ayuda necesaria para poder solventar los problemas presentes. Por tales motivos se recurre a la búsqueda de alguna alternativa que se adapte a nuestras necesidades; luego de encontrar y analizar las librerías disponibles se llega a la decisión de utilizar PyQtGraph⁵. Esta librería además de todas su prestaciones en el ámbito científico / matemático / aplicaciones ingenieriles, está construida con las mismas librerías que utilizamos en nuestro proyecto, que es PyQt⁶. Por lo tanto, esto generó una excelente adaptabilidad y rendimiento al momento de realizar las pruebas de integración. Para las **pruebas unitarias/estrés/funcionales** se ha creado el módulo *GraphDatav2*, el cual ha sido puesto en prueba hasta su máximo potencial. Utilizando una señal de tipo aleatoria como datos de prueba (ya que el mismo contiene una gran cantidad de componentes frecuenciales en su dominio) y se ha generado su

⁴Sitio web de Vispy <http://vispy.org/>

⁵Sitio web de PyQtGraph <http://pyqtgraph.org/>

⁶Sitio Web de PyQt <https://riverbankcomputing.com/software/pyqt/intro>

respectiva transformada de Fourier (utilizando la Transformada Rápida de Fourier), con el fin de corroborar que los cálculos de los mismo no ralenticen el funcionamiento del mismo, y luego de realizar las pruebas nos suministró resultados dentro de las expectativa de rendimiento. Siendo estas expectativas:

- Fluidez en la animación de la figuras.
- Que no interrumpa el flujo de trabajo de la aplicación en general.
- La interfaz ofrece de herramientas como zoom y paneo, por lo tanto al momento que el usuario las utilice es deseable que el mismo no retarde el renderizado.
- Fluidez al momento de graficar el límite de figuras permitidas (5 - 10).

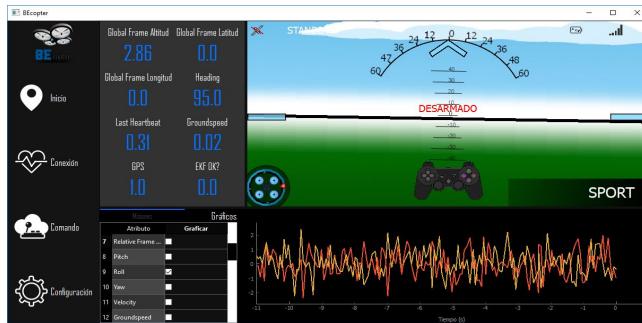


Figura 5.13: Módulo de representación de datos

- **HUD:** A diferencia del módulo anterior, su responsabilidad es mostrar al usuario de manera instantánea los estados sobre los componentes integrados al vehículo, como también algunos de sus respectivos valores. Por lo tanto, más allá de sus pruebas unitarias/funcionales/estrés/integración aquí entra en juego las pruebas de aceptación proporcionadas por “los pilotos de los vehículos”, ya que el mismo debe ser indispensable al momento de mostrar los valores que ayuden al piloto del vehículo en tener un conocimiento general del mismo en plena ejecución. Así que de manera preventiva se ha desarrollado el mismo imitando en lo mejor posible a los HUD’s reales, con el fin de tener una base de diseño que sea válida en la vida real. Con respecto a las pruebas de integración hemos detectado que no cumple con nuestras necesidades de rendimiento, esto causa que el software se ralentice y no pueda interpretar “en tiempo real” la información recibida del vehículo como también los comandos enviados. Por lo que se decidió temporalmente deshabilitar algunas características del HUD hasta que se encuentre su respectiva solución.

5.3.3.3. Control del vehículo

El control del vehículo, ya sea de forma manual o automática, es una de las funcionalidades principales de este software. En lo que es el control manual, consiste en estar consultando constantemente los eventos producidos por un



Figura 5.14: Head Up Display

Joystick y según una respectiva pre-asignación se ejecuta un comando sobre el vehículo. Y en el caso del control automático, se requiere que se envíen un conjunto de misiones pre-configuradas y al ser recibidas por el vehículo, que el mismo las ejecute.

- **Control Manual:** Para poder ejecutar las pruebas del funcionamiento del control manual, es indispensable que el módulo de comunicación funcione correctamente, ya que el control manual depende del mismo. Al momento de ejecutar las pruebas unitarias, entran en juego varias clases, como lo son la clase Vehículo y Joystick. Por lo tanto se ha realizado en primera instancia el control manual con envío de acciones con una duración definida de tiempo, simulando la interacción con el joystick, y para determinar si los comandos han sido interpretados por el vehículo se ha analizado la evolución temporal del giro de los motores (utilizando el módulo de representación de datos) medido en [rpm], y mostrándonos resultados aceptables y razonables como se puede ver en la Figura 5.9

Luego de comprobar el correcto funcionamiento de los motores según comandos limitados en tiempo, se inicia el proceso de pruebas con el joystick. En este caso se utiliza una clase auxiliar *Capturador* que ha sido creada con el fin de ejecutarse en un hilo de ejecución independiente para no congestionar el procesamiento de manera secuencial o interrumpir el hilo de ejecución padre. En el siguiente código podemos apreciar su funcionamiento:

```

1  class Capturador(QThread):
2      "Clase encargada de capturar en background los
3          comandos que se envian del Joystick "
4      condicion_para_capturar = False
5      def __init__(self, parent):
6          super(Capturador, self).__init__(parent)
7          # QThread.__init__(parent)
8
8      def capturing(self, run):
9          self.condicion_para_capturar = run
10
11     def __del__(self):

```

```

12     self.exit(0)
13
14
15     def capturarComandos(self):
16         try:
17             while True :
18
19                 if self.condicion_para_capturar:
20                     eventos = pygame.event.get()
21                     for e in eventos:
22                         # e.type == 7, es un codigo de tipo joymotion
23                         if e.type == 7:
24                             self.emit(SIGNAL('mostrarEje(int,float)'), e.axis, e.
25                                         value)
26                             self.emit(SIGNAL('sendManualCmd(int,float,bool)'), \
27                                         e.axis+12, e.value, True)
28                         if e.type == 10: #Es de tipo JoyButtonDown
29                             self.emit(SIGNAL('sendManualCmd(int,float,bool)'), e.
29                                         button, 1.0, False)
30                     except:
31                         WindowsError("Error de comando", "Ah ocurrido un error
32                                         al enviar los comandos" )

```

La función *sendManualCmd* posee comandos preestablecidos por el usuario, por lo tanto, cuando se presione algún botón o se mueva algún joystick (palancas) identificado con un id (primer parámetro entero) y su respectivo valor, la clase *Capturador* emitirá una señal a su respectivo objeto instanciador y procederá a enviar los comandos pertinentes al vehículo. Para comprobar dichas funcionalidades únicamente se ha observado el comportamiento de los motores al girar, ya que al enviar un comando de detención, los mismos respondían correctamente a dicha acción.

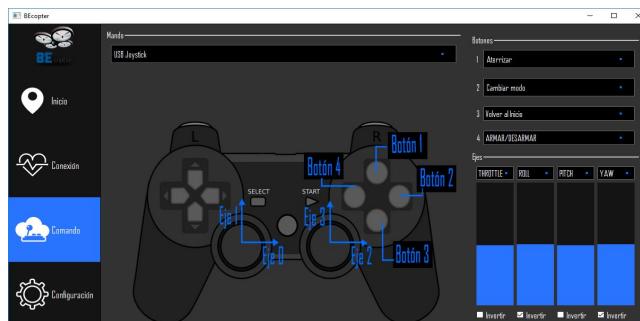


Figura 5.15: Configuración del módulo de control manual

- **Control automático:** En este apartado se comprueba el gestionamiento de las misiones, como por ejemplo: que las mismas puedan ser enviadas y recibidas correctamente por el vehículo, y que sean debidamente validadas mediante la interfaz gráfica. Por otro lado, tenemos que verificar que dichas misiones sean ejecutadas por el vehículo. Para comprobar dicha funcionalidad se han cargado y enviado, mediante la aplicación, misiones

unitarias con el fin de corroborar el comportamiento del vehículo según la misión asignada. Pero antes de enviar cualquier tipo de misión, se tuvo que atar al vehículo a una base fija con el fin de que al momento de realizar algún tipo de movimiento inadecuado, el mismo no pierda el control y se estrelle con algún objeto. Las pruebas por cada misión llevaron el siguiente procedimiento:

- **Despegar:** En este tipo de misión el vehículo procede a elevarse, de esta manera, los motores generaron la suficiente propulsión para elevar la estructura del vehículo.
- **Aterrizar:** Posicionando el vehículo a una altura no mayor a 30cm, dejamos que descienda el mismo suavemente, procurando que no se desestabilice y se apoye suavemente sobre la base a la cual está atada.
- **Suspenderse en el aire:** Al momento de ejecutar dicha misión, se ha suspendido con una cuerda al vehículo con el fin de observar que el mismo pueda mantenerse en el aire el tiempo indicado en la misión (3 segundos) de manera estable.
- **Punto Objetivo:** En este caso, se ha ordenado al vehículo simplemente moverse hacia la izquierda. Por lo tanto el mismo realizó un pequeño giro sobre su eje *Roll*, propulsándose lentamente hacia a su izquierda, como se le ordenó.



Figura 5.16: Módulo de control automático del vehículo

5.3.3.4. Parametrización

El módulo encargado de la parametrización, es simplemente una interfaz gráfica que muestra las opciones disponibles y configurables del firmware del autopiloto, como se puede ver en la Figura 5.17. Por lo tanto, se valida que esta información sea mostrada de manera correcta. Además, esta pestaña permite modificar los valores de cada parámetro, por lo que es necesario validar esta modificación y que sea guardada de forma correcta. Por último probando que el vehículo con sus respectivas parámetros modificados se comporte según lo configurado.

5.4. Resultados

Como se pudo ver en las secciones anteriores las pruebas realizadas hasta el momento de los módulos pertinentes se han realizado de manera satisfactoria.

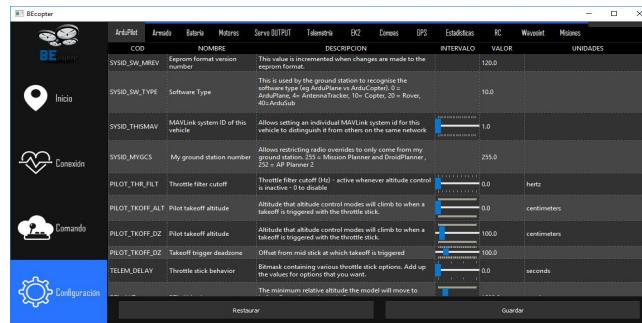


Figura 5.17: Pestaña de parametrización del Autopiloto

Como paso final, se realizará una prueba funcional del proyecto ejecutando un conjunto de pruebas de vuelo que involucren la mayor cantidad de funcionalidades integradas en el software. Por tal razón y a modo de evitar roturas en la estructura del vehículo, se ha optado por enviar a imprimir un modelo 3D de la estructura del vehículo, con el fin de ganar autonomía en la reposición de partes y facilitar a que el proyecto sea de fácil implementación gracias a esta tecnología. Por lo tanto se adjuntará un video de las pruebas realizadas una vez obtenido dicho modelo.

5.5. Conclusión

Finalmente, luego de haber terminado con la realización de los distintos tipos de pruebas y comprobar que cada funcionalidad de nuestra aplicación, como también de nuestro vehículo estén funcionando correctamente, llegamos a la conclusión de que el *Aseguramiento de la calidad* es un aspecto muy importante en el desarrollo de cualquier tipo de proyecto.

De tal manera, como mencionamos en la introducción de este informe, este proceso se aplica de manera integral desde los inicios del proyecto hasta en las últimas etapas; pero cabe destacar que este proceso del *Aseguramiento de la calidad* no ha sido aplicado correctamente en las etapas iniciales al proyecto ya que se consideraba como una tarea irrelevante, por lo cual , y como es de esperarse en etapas posteriores nos encontrábamos que muchos errores y con la consecuencia de tener módulos deficientes e inseguros. Lo que impedía proseguir con el proyecto ya que se dificultaba la tarea del desarrollo debido a código no robusto y metodologías poco probadas. Así que nos vimos obligados de reformular la estrategia de las etapas siguientes con respecto a las pruebas y además, aplicar dicha técnica por primera vez a las etapas iniciales para asegurar que en un futuro la aparición de los errores disminuya. De esta manera se ha aprendido, y por lo tanto, se incorporará como “buenas prácticas”, integrar el proceso de QA en proyectos futuros para el cual participe el alumno.

Parte I

Apéndices

Apéndice A

Documento de Especificación de Requerimientos



BEcopter

Especificación de Requerimientos de Software

Bastida Eric

septiembre, 2017

Fecha	Versión	Descripción	Autor
24/09	1.0	Especificación de Requerimientos	Bastida Eric
25/09	1.1	Se realizan cambios, comentarios y sugerencias	Marina Murillo
25/09	1.2	Se resuelven las sugerencias	Bastida Eric

Índice

1. Introducción	2
1.1. Propósito	2
1.2. Alcance	2
1.3. Definiciones, Acrónimos y Abreviaturas	3
1.4. Referencias	5
1.5. Visión General del Documento	5
2. Descripción General	6
2.1. Perspectiva del Producto	6
2.2. Funciones del Producto	6
2.3. Características de los Usuarios	7
2.4. Restricciones	7
2.5. Suposiciones y Dependencias	8
2.6. Requisitos Futuros	8
3. Requisitos Específicos	9
3.1. Interfaces Externas	9
3.1.1. Interfaces de usuario:	9
3.1.2. Interfaces de hardware:	11
3.1.3. Interfaces de software:	11
3.1.4. Interfaces de comunicación:	12
3.2. Requisitos funcionales	12
3.3. Requisitos no funcionales	14

1. Introducción

En este documento destinado a la Especificación de Requerimientos de Software (ERS) se describirán los aspectos que están involucrados en el desarrollo del proyecto BEcopter . Dicho proyecto consiste en una plataforma para el guiado y navegación de un vehículo aéreo no tripulado (VANT), es decir, este software permitirá al usuario realizar maniobras aéreas de forma manual con un joystick o bien mediante un conjunto de instrucciones cargadas en modo *batch* dentro del vehículo que le permitirán desplazarse en un determinado ambiente. Además esta plataforma le brindará al usuario toda la información necesaria del estado del vehículo y sensores.

Cabe mencionar que el formato en que se desarrolló este documento fue basado en el estándar 830-1998, proporcionada por la IEEE.

1.1. Propósito

El presente documento tiene como propósito definir los diferentes tipos de requerimientos que han sido obtenidos de los usuarios que utilizarán de manera directa o indirecta el sistema (*stakeholders*) mediante técnicas de ingeniería en requerimientos [1], incluyendo además, los juicios que se han considerado para la planificación y diseño del software. Toda esta información es de utilidad para el desarrollador del presente proyecto para avanzar en el desarrollo del mismo y, sobre todo, informar de manera clara y precisa a los agentes evaluadores del proyecto: directores e integrantes de la cátedra *proyecto final de carrera* (PFC).

1.2. Alcance

El nombre BEcopter es un acrónimo originado por las iniciales del ejecutor del proyecto y el tipo de vehículo que se utiliza para realizar las pruebas. En nuestro caso hacemos uso de un VANT tipo cuadricóptero (o quadcopter en su traducción al inglés) , por lo tanto, para el desarrollo del nombre se descarta el prefijo “cuadri” ya que este determina la cantidad de hélices que contiene el vehículo y además esta descripción limita al software al uso de VANTs de cuatro hélices; y como se explicará en la sección 2.6 no únicamente se pretende utilizar cuadricópteros. El sistema proporcionará las acciones necesarias para que el vehículo pueda realizar maniobras en el aire mediante el control de un joystick que será comandado por el usuario o a través de un conjunto de instrucciones que serán cargadas en la plataforma y luego enviadas al vehículo para su ejecución. Estas acciones serán:

- Ascenso,
- Descenso,
- *Hovering*.

Este software proporcionará a la comunidad una plataforma con la cual el usuario final podrá implementar sus propias ideas sin entrar en detalles internos del hardware pertinente al vehículo.

1.3. Definiciones, Acrónimos y Abreviaturas

- **Stackholders:** Este término se utiliza comúnmente para referirse a cualquier persona o grupo que se verá afectado por el sistema de manera directa o indirectamente.
- **Plataforma:** Para este tipo de vehículos, es un sistema que sirve como base para hacer funcionar los módulos de hardware y/o de software en conjunto. En lo que concierne al hardware hacemos referencia a los componentes electrónicos del VANT, como pueden ser:
 - Motores,
 - Sensores como por ejemplo, acelerómetros, giroscopios, magnetómetros, etc.,
 - Periféricos de entrada y/o salida tales como tarjetas 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 tareas, y que de forma sencilla proporcione al usuario funcionalidades para manipular estos componentes electrónicos, con el propósito de realizar acciones de vuelo sobre el vehículo.

- **VANT o UAV:** Un vehículo aéreo no tripulado (VANT), conocido también por su nombre en inglés como *Unmanned Aerial Vehicle* (UAV) o dron, es una aeronave que vuela sin tripulación. El mismo es capaz de mantener un nivel de vuelo controlado, sostenido y propulsado por uno o varios motores de explosión, electrónicos, o de reacción.
- **Cuadricóptero:** Un cuadricóptero, cuadrirrotor o quadrotor es un helicóptero con cuatro rotores para su sostén y su propulsión. Los cuatro rotores están generalmente colocados en las extremidades de una cruz. A fin de evitar que el aparato se tumbe respecto a su eje de orientación es necesario que dos hélices giren en un sentido y las otras dos en el otro sentido.
- **Roll, Pitch, Yaw:** [2] Los ángulos de navegación (ángulos de Euler) se utilizan para describir la orientación de un objeto en tres dimensiones como se puede observar en la figura 1.
- **Firmware:** Es un programa informático que establece la lógica de más bajo nivel que controla los circuitos electrónicos de un dispositivo de cualquier tipo. Está fuertemente integrado con la electrónica del dispositivo, es el software que tiene directa interacción con el hardware, siendo así el encargado de controlarlo para ejecutar correctamente las instrucciones externas. Por lo tanto un firmware es un software que maneja físicamente al hardware.
- **Modo batch:** (o procesamiento por lotes) es la ejecución de un programa sin el control o supervisión directa del usuario (que se denomina procesamiento interactivo). Este tipo de programas se caracterizan porque su ejecución no precisa ningún tipo de interacción con el usuario.

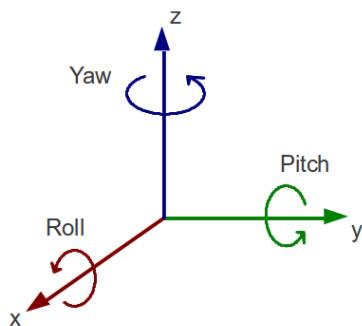


Figura 1: Sistema de referencia.

- **Acelerómetro:** Instrumento que sirve para medir la aceleración de movimiento de un vehículo.
- **Magnetómetro:** Instrumento para medir la fuerza y la dirección de un campo magnético.
- **ESC:** Un control electrónico de velocidad o ESC es un circuito electrónico con el propósito de variar la velocidad de un motor eléctrico, su dirección y posiblemente también actuar como un freno dinámico.
- **GPS:** Sistema americano de navegación y localización mediante satélites.
- **Giroscopio:** Dispositivo electrónico que sirve para medir la orientación en el espacio de algún aparato o vehículo.
- **Compás:** Instrumento que proporciona una dirección de referencia (respecto al norte) en el plano horizontal y permite la medición de ángulos horizontales con respecto a esta dirección.
- **Barómetro:** Instrumento para medir la presión atmosférica.
- **Horizonte artificial:** El horizonte artificial muestra la orientación longitudinal de la aeronave (la relación del eje longitudinal del vehículo con respecto al plano del suelo), es decir: si está girado, inclinado, con el frente levantado, bajado o todo a la vez. Sirve de gran ayuda en condiciones en que la visibilidad es poca o nula. Su principio mecánico de funcionamiento es giroscópico

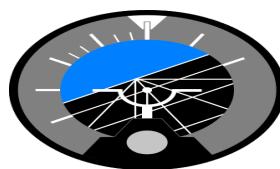


Figura 2: Horizonte artificial indicando un giro a la derecha en descenso.

- **Instrumentos de vuelo:** Conjunto de dispositivos tales como giroscopio, acelerómetro, magnetómetro, ESC, GPS, barómetro, indicador de rumbos, horizonte artificial, brújula que equipan una aeronave y que permiten al piloto llevar a cabo una operación de vuelo en condiciones seguras. Dependiendo de su tamaño o grado de sofisticación, una aeronave puede contar con un número variable de instrumentos[3].
- **Aeromodelismo:** es una afición y un deporte derivado de la técnica de construcción y vuelo de aeroplanos de pequeño, mediano y gran tamaño, denominados aeromodelos, que han sido preparados para volar sin tripulación.

1.4. Referencias

- [1] S. Ian, *Ingeniería del software*. Pearson Educación, 2005.
- [2] J. G. de la Cuesta, *Aviation Terminology = Terminología Aeronáutica*. Diaz de los Santos, 2003.
- [3] Wikipedia, “Instrumentos de vuelo - wikipedia, la enciclopedia libre,” 2017. [Internet; descargado 27-agosto-2017].

1.5. Visión General del Documento

De manera informativa y para guiar al lector en la lectura del presente documento, esta sección se presenta un breve resumen sobre la estructura del mismo. En forma general, este documento se compone de cuatro partes importantes:

1. **Introducción:** En esta sección se proporcionará un panorama general a todo el documento de Especificación de Requisitos Software (ERS). Consta de varias subsecciones: 1.1 Propósito, 1.2 Alcance del sistema, 1.3 Definiciones y 1.4 Referencias.
2. **Descripción General:** En esta sección se describen todos aquellos factores que afectan al producto y a sus requisitos. No se describen los requisitos, sino su contexto. Además, consta de las siguientes subsecciones: 2.1 Perspectiva del producto, 2.2 funciones del producto, 2.3 características de los usuarios, 2.4 restricciones, factores que se asumen y 2.6 futuros requisitos.
3. **Requisitos Específicos** Esta sección contiene los requisitos a un nivel de detalle suficiente como para permitir a los programadores diseñar un sistema que satisfaga estos requisitos, y que permita al equipo de pruebas planificar y realizar las pruebas que demuestren si el sistema satisface, o no, los requisitos. Todo requisito especificado describirá comportamientos externos del sistema, perceptibles por parte de los usuarios, operadores y otros sistemas.

2. Descripción General

2.1. Perspectiva del Producto

El producto BEcopter tiene como perspectiva ser una plataforma para el guiado y navegación de VANTs, esto incluye, aeroplanos, helicópteros o multirotores como pueden ser tricópteros, cuadricópteros, hexacópteros, etc. El producto depende de un firmware que es distribuido bajo la licencia GNU por la empresa Ardupilot¹, este es el encargado de gestionar todo el hardware perteneciente al vehículo mediante un software a bajo nivel, y que a su vez, brinda funcionalidades básicas de control e información sobre el estado del vehículo.

2.2. Funciones del Producto

En esta subsección del documento se explicarán, a grandes rasgos, las funcionalidades que proporcionará la plataforma BEcopter :

- **Control del vehículo:** Mediante esta funcionalidad se pretende que el sistema pueda proporcionar de manera clara y sencilla al usuario acciones que puedan ser ejecutadas por el vehículo, como pueden ser: despegar, moverse y aterrizar. En lo que respecta a las funciones se podrán hacer de dos maneras distintas.
 1. **Modo manual:** El usuario podrá controlar el vehículo mediante un joystick que enviará las señales correspondientes al sistema y según una configuración predeterminada serán interpretados por el vehículo para realizar las acciones que han sido fijadas.
 2. **Modo batch:** En este modo el sistema brindará distintos tipos de acciones (denominadas misiones) que el usuario podrá seleccionar de manera secuencial, es decir, siguiendo un orden correspondiente como por ejemplo: primero despegar luego elevarse e ir a una ubicación específica, etc. Además, se le podrán asignar atributos a cada acción como pueden ser: velocidad, posición, altura, etc. Una vez que el usuario selecciona este conjunto o lote de acciones el sistema las enviará al vehículo para que sean ejecutadas.
- **Comunicación inalámbrica:** Para que el sistema pueda comunicarse con el vehículo se debe seleccionar algún tipo de medio por el cual sea posible transmitir los datos; ya que el vehículo se desplazará en grandes espacios despejados, es recomendable que los datos se envíen de manera inalámbrica. Por tanto, el sistema tendrá la responsabilidad de proporcionar las funcionalidades correspondiente a la gestión de comunicación inalámbrica entre la plataforma y el vehículo.
- **Visualización de datos:** Poder representar datos como ubicación, velocidad, altura, estado de motores, entre otros aspectos, es de suma importancia ya que brinda información relevante al usuario para poder realizar, en tiempo real, maniobras de salvaguarda sobre el vehículo o capturar información para realizar algún tipo de análisis para su posterior estudio. Es por eso que el sistema representará de manera numérica todos los datos

¹ArduPilot - www.ardupilot.org

que sean necesarios para la navegación, control y sensores del vehículo. Además, dará la opción de poder visualizar estos datos, que van cambiando continuamente a través del tiempo en forma gráfica. Esto es un aspecto importante ya que permite al usuario ver cómo es el comportamiento temporal de variables críticas del vehículo y/o datos a analizar.

2.3. Características de los Usuarios

En lo que respecta al tipo de usuario final que utilizará el sistema son dos:

1. Usuario normal:

Tipo de usuario	Normal.
Formación	Conocimientos básicos en informática y aeromodelismo.
Habilidades	Navegación en vehículos aéreos.
Actividades	Uso y configuración de la plataforma.

2. Usuario desarrollador:

Tipo de usuario	Desarrollador
Formación	Ciencias de la computación
Habilidades	Programación en Python y navegación en vehículos aéreos.
Actividades	Uso y configuración de la plataforma. Implementación de rutinas.

2.4. Restricciones

Esta subsección corresponde a aquellas limitaciones que se imponen al desarrollador del producto.

- Sistema Operativo: Linux.
- Tipo de software: Desarrollado para ordenadores de tipo escritorio.
- Funciones de vuelo: Momentaneamente se desarrollarán 3 funciones
 1. Ascender,
 2. Descender,
 3. *Hovering*.
- Interfaz gráfica de usuario: Desarrollada mediante la librería PyQt ².
- Limitaciones de hardware: Por pertenecer a un software de tipo de control de navegación de vehículos aéreos es necesario tener los siguientes elementos:
 1. Raspberry pi 3 ³.
 2. Placa Navio2 ⁴.

²PyQt - www.riverbankcomputing.com/software/pyqt/intro

³Raspberry - www.raspberrypi.org

⁴Navio2 - www.emlid.com/navio/

3. Joystick.
 4. Módulos de comunicación inalámbrica XBee⁵.
 5. Estructura de un VANT.
- Lenguaje de programación: Python 2.7.
 - Consideraciones acerca de la seguridad: El producto no implementa un modo de seguridad o *safety mode* por tanto es responsabilidad del usuario cualquier tipo de accidente que se produzca utilizando el producto.
 - Desarrollado bajo la Licencia Pública General de GNU.

2.5. Suposiciones y Dependencias

Con lo establecido hasta el momento no se suponen factores que puedan intervenir en modificaciones a un nivel profundo sobre el desarrollo del producto, en caso fortuito sobre cualquier tipo de inconveniente, se incorporarán de manera adicional al documento para solventar el problema sin afectar en su totalidad a los demás requisitos. El producto tiene una gran dependencia tecnológica con el hardware ya que de manera conjunta establecen un canal de comunicación y además, es el paso principal que condiciona el inicio de las funcionalidades del sistema, por esta razón es de suma importancia tener de antemano el vehículo armado y correctamente configurado.

Por distintos factores que intervienen en la gestación del proyecto, el desarrollo de las funcionalidades tendrá foco en la navegación de un VANT tipo cuadricóptero, descartando en primera medida opciones de vuelo orientadas a aeroplanos.

2.6. Requisitos Futuros

Se prevén los siguientes requisitos que pueden ser implementados en el producto.

- **Más acciones de vuelo:** se pretende desarrollar funcionalidades de navegación que equipen al vehículo con menos restricciones al momento de desplazarse en el aire, ya sean:
 - *Desplazamiento en forma circular con respecto a un punto (objetivo) como centro:* esta funcionalidad por lo general es útil al momento de hacer tomas cinematográficas o de algún tipo de video casero, proporcionando así una buena perspectiva del objetivo a analizar.
 - *Follow me:* en esta opción se designara un objetivo (por ejemplo una persona) y una vez ingresado al software el vehículo procederá a seguir este objetivo según parámetros preestablecidos con anterioridad como: altura, distancia al objeto, ubicación, etc.
- **Integración a más tipos de vehículos:** Elección a más tipos de vehículos que puedan ser comandados y guiados por la plataforma, como por ejemplo, aeroplanos, helicópteros, hexacópteros, tricópteros, etc.

⁵XBEE - www.xbee.cl/que-es-xbee/

- **Adquisición de imágenes:** En el caso que vehículo tenga incorporada una cámara la plataforma brindará opciones para poder visualizar las imágenes que se están capturando en tiempo real.

3. Requisitos Específicos

En esta sección se realiza una descripción detallada de cada uno de los requerimientos. Además de describirlos, en esta sección también se clasifican los mismos de acuerdo al módulo al cual pertenecen para facilitar su trazabilidad a través del desarrollo de la aplicación.

3.1. Interfaces Externas

3.1.1. Interfaces de usuario:

Número de requisito	R001
Nombre del requisito	Visualización de datos de vuelo y estado del vehículo
Tipo	Requisito Restricción
Descripción	El sistema debe proporcionar la representación en tiempo real y numérica de los datos de vuelo, esto incluye, información proveniente de todos los sensores del vehículo y niveles de intensidad generados por los periféricos de entrada/salida, tales como joystick y señal de cobertura.
Prioridad del requisito	Alta/Essencial Media/Deseado Baja/Opcional

Número de requisito	R002
Nombre del requisito	Representación gráfica en tiempo real y evolutiva de la información capturada.
Tipo	Requisito Restricción
Descripción	El sistema debe dar la opción de representar gráficamente y en forma evolutiva la información capturada por algún determinado periférico de entrada. Esto se debe representar mediante una ventana y utilizando la librería Vispy ⁶ .
Prioridad del requisito	Alta/Essencial Media/Deseado Baja/Opcional

Número de requisito	R003
Nombre del requisito	Representación gráfica de las variables características para el guiado, navegación y control del vehículo.
Tipo	Requisito Restricción
Descripción	El sistema deberá ilustrar de manera gráfica y representativa los datos provenientes de los instrumentos de vuelo. Como el horizonte de artificial, indicador de rumbos, etc.
Prioridad del requisito	Alta/Essencial Media/Deseado Baja/Opcional

Número de requisito	R004
Nombre del requisito	Manual de usuario.
Tipo	Requisito Restricción
Descripción	Se deberá realizar un manual que especifique y ejemplifique las principales funcionalidades del sistema. Este manual será un medio por el cual los usuarios podrán aprender a utilizar el sistema, por lo que deberá ser realizado con este propósito.
Prioridad del requisito	Alta/Esencial Media/Deseado Baja/Opcional

Número de requisito	R005
Nombre del requisito	Inserción de comandos de vuelo
Tipo	Requisito Restricción
Descripción	El sistema ofrecerá una lista de comandos que el usuario podrá elegir de manera secuencial y luego serán enviadas al vehículo para que sean procesadas. Estos comandos serán: ascender, descender y <i>hovering</i> . Además de incluirse sus respectivas características como velocidad, posición y tiempo de ejecución.
Prioridad del requisito	Alta/Esencial Media/Deseado Baja/Opcional

Número de requisito	R006
Nombre del requisito	Modos de vuelo
Tipo	Requisito Restricción
Descripción	A modo de seguridad el sistema debe permitir el guiado de vehículo en modo manual
Prioridad del requisito	Alta/Esencial Media/Deseado Baja/Opcional

Número de requisito	R007
Nombre del requisito	Representación gráfica del vehículo
Tipo	Requisito Restricción
Descripción	Como medida de referencia el sistema ilustrará un VANT representativo mediante un gráfico que simule las acciones que está realizando el vehículo en tiempo real.
Prioridad del requisito	Alta/Esencial Media/Deseado Baja/Opcional

Número de requisito	R008
Nombre del requisito	Calibración
Tipo	Requisito Restricción
Descripción	El sistema debe informar al usuario si los sensores principales del vehículo se encuentran correctamente calibrados, en caso contrario se dará la opción de calibrarlos.
Prioridad del requisito	Alta/Esencial Media/Deseado Baja/Opcional

Número de requisito	R009
Nombre del requisito	Posicionamiento geográfico.
Tipo	Requisito Restricción
Descripción	Mediante un mapa el sistema debe ubicar geográficamente utilizando el GPS del VANT y representarlo con algún tipo de gráfico en dicha ubicación.
Prioridad del requisito	Alta/Esencial Media/Deseado Baja/Opcional

Número de requisito	R010
Nombre del requisito	Implementación de un Log perteneciente a la plataforma
Tipo	Requisito Restricción
Descripción	Brindar un log de todo tipo de inconvenientes, errores o precauciones que se presentan sobre el uso de las funcionalidades de la plataforma.
Prioridad del requisito	Alta/Esencial Media/Deseado Baja/Opcional

Número de requisito	R011
Nombre del requisito	Implementación de un Log perteneciente a los sensores.
Tipo	Requisito Restricción
Descripción	Brindar un log de todo tipo de inconvenientes, errores o precauciones que se presentan sobre el uso de los sensores instalados en el vehículo.
Prioridad del requisito	Alta/Esencial Media/Deseado Baja/Opcional

3.1.2. Interfaces de hardware:

Número de requisito	R012
Nombre del requisito	Identificación y control del comando.
Tipo	Requisito Restricción
Descripción	El sistema deberá identificar el comando de tipo joystick conectado a la computadora y dar opciones de mapeo de las funciones de navegación a elección del usuario, siendo por defecto uno ya estipulado, utilizando la librería Pygame ⁷ .
Prioridad del requisito	Alta/Esencial Media/Deseado Baja/Opcional

Número de requisito	R013
Nombre del requisito	Gestión con el hardware de comunicación inalámbrica.
Tipo	Requisito Restricción
Descripción	El sistema deberá identificar el hardware pertinente a la comunicación inalámbrica y brindar las opciones para su correspondiente configuración de red.
Prioridad del requisito	Alta/Esencial Media/Deseado Baja/Opcional

3.1.3. Interfaces de software:

Número de requisito	R014
Nombre del requisito	Integración de Drone-kit ⁸
Tipo	Requisito Restricción
Descripción	Uso de la librería Drone-kit en Python para poder gestionar todo tipo de interacción con el VANT.
Prioridad del requisito	Alta/Essencial Media/Deseado Baja/Opcional

3.1.4. Interfaces de comunicación:

Número de requisito	R015
Nombre del requisito	Comunicación inalámbrica
Tipo	Requisito Restricción
Descripción	El sistema debe permitir el establecimiento de una comunicación inalámbrica con el vehículo según los módulos de comunicación existentes en ambos extremos, además de poder buscar, establecer o interrumpir la comunicación con el vehículo presente.
Prioridad del requisito	Alta/Essencial Media/Deseado Baja/Opcional

3.2. Requisitos funcionales

Número de requisito	R016
Nombre del requisito	Estado de conexión
Tipo	Requisito Restricción
Descripción	Se deberá proveer una sentencia en el script principal encargada de consultar periódicamente el estado de conexión del vehículo o de acercarse fuera del rango de cobertura. En caso de perder la conexión se deberá invocar una función de salvataje.
Prioridad del requisito	Alta/Essencial Media/Deseado Baja/Opcional

Número de requisito	R017
Nombre del requisito	Función de salvataje
Tipo	Requisito Restricción
Descripción	Implementar acciones de resguardo del vehículo en caso de perder la conexión o tener un bajo nivel de recepción de señal. Este script deberá consultar los niveles de energía restantes y en caso de obtener niveles moderados, se le enviarán al vehículo las acciones necesarias para que el mismo se estabilice y/o quede suspendido en el aire. En caso de no tener la energía necesaria se procederá a realizar un aterrizaje.
Prioridad del requisito	Alta/Essencial Media/Deseado Baja/Opcional

Número de requisito	R018
Nombre del requisito	Script base para el chequeo del vehículo.
Tipo	Requisito Restricción
Descripción	El sistema deberá comprobar los estados de cada sensor instalado en el vehículo, con esto el script debe corroborar la existencia de cada sensor y su correcta calibración, en caso contrario se deberá deshabilitar la opción de despegue del vehículo hasta que se solucione el problema. De manera complementaria se deberá informar el origen del mismo.
Prioridad del requisito	Alta/Esencial Media/Deseado Baja/Opcional

Número de requisito	R019
Nombre del requisito	Estructura de clases para VANT.
Tipo	Requisito Restricción
Descripción	Programar mediante el paradigma orientado a objetos las entidades necesarias para la gestión del vehículo como también conexión y control.
Prioridad del requisito	Alta/Esencial Media/Deseado Baja/Opcional

Número de requisito	R020
Nombre del requisito	Clase sensor.
Tipo	Requisito Restricción
Descripción	Cada sensor perteneciente al vehículo corresponde a una entidad lógica para la gestión del vehículo, por tanto, este deberá proveer información sobre el estado, dar la opción de calibrar y obtención de información del mismo mediante funcionalidades implementadas en una clase.
Prioridad del requisito	Alta/Esencial Media/Deseado Baja/Opcional

Número de requisito	R021
Nombre del requisito	Clase conexión.
Tipo	Requisito Restricción
Descripción	Debido a que la conexión entre la plataforma y el vehículo es un aspecto importante se deberá implementar una clase que gestione toda esta interacción.
Prioridad del requisito	Alta/Esencial Media/Deseado Baja/Opcional

Número de requisito	R022
Nombre del requisito	Clase Comando.
Tipo	Requisito Restricción
Descripción	El joystick encargado de comandar la nave debe ser capaz de contener todas las funcionalidades que serán ejecutadas por el vehículo y además de corresponder dichas funciones sobre un mapeo personalizado. La clase comando deberá gestionar toda esta información.
Prioridad del requisito	Alta/Esencial Media/Deseado Baja/Opcional

Número de requisito	R023
Nombre del requisito	Clase vehículo.
Tipo	Requisito Restricción
Descripción	Esta clase contendrá todo tipo de información proveniente del vehículo físico, además de sensores que este contiene y se encargará de interpretar las acciones de navegación que han sido enviadas por la plataforma.
Prioridad del requisito	Alta/Esencial Media/Deseado Baja/Opcional

Número de requisito	R024
Nombre del requisito	Modos de vuelo.
Tipo	Requisito Restricción
Descripción	Proveer una funcionalidad dentro de la clase vehículo que permita guiar el vehículo de manera manual con el comando o mediante una lista de acciones. En caso de utilizar la segunda opción se permitirá además poder manejar el vehículo de manera manual para evitar algún tipo de accidente.
Prioridad del requisito	Alta/Esencial Media/Deseado Baja/Opcional

3.3. Requisitos no funcionales

Número de requisito	R025
Nombre del requisito	Seguimiento del vehículo a través de un mapa.
Tipo	Requisito Restricción
Descripción	Mediante un mapa ubicar la posición geográfica del vehículo con respecto a los datos obtenidos del sistema de posicionamiento global.
Prioridad del requisito	Alta/Esencial Media/Deseado Baja/Opcional

Número de requisito	R026
Nombre del requisito	Conversión de unidades.
Tipo	Requisito Restricción
Descripción	En aeromodelismo se utilizan una variada cantidad de unidades métricas para magnitudes físicas y la utilización de estos depende de cada usuario, es por esto, que se debe suministrar la opción de un conversor de unidades.
Prioridad del requisito	Alta/Esencial Media/Deseado Baja/Opcional

Número de requisito	R027
Nombre del requisito	GitHub
Tipo	Requisito Restricción
Descripción	Con fines de portabilidad el versionado del proyecto o producto final será gestionado mediante el software GitHub.
Prioridad del requisito	Alta/Esencial Media/Deseado Baja/Opcional

Apéndice B

Manual de usuario

Manual de Usuario - BEcopter

Release v0.9

Eric Bastida

octubre de 2018

Contenido:

Ventajas	1
Introducción a BEcopter	2
Restricciones	2
Instalación	2
Primeros pasos	3
Pestaña de Inicio	4
Botonera	4
HUD (<i>Head Up Display</i>)	5
Misiones y Gráficos	8
Pestaña de Conexión	11
Pestaña de Comando	13
Mando	13
Botones	13
Ejes	14
Pestaña de Configuración	15

BEcopter es un software que ha sido desarrollado como proyecto final de la carrera Ingeniería en Informática de la Universidad Nacional del litoral *UNL* provincia de Santa Fe, Argentina. El origen de este proyecto surge de la necesidad de integrantes del Centro de Investigación en Señales, Sistemas e Inteligencia *sinc(i)* en tener una plataforma que pueda ser el intermediario para controlar y navegar un vehículo aéreo no tripulado. En este manual explicaremos como utilizar *BEcopter* con el fin de poder controlar su vehículo y ejecutar maniobras aéreas que serán enviadas desde una PC a nuestro vehículo.

Ventajas

BEcopter proporciona las siguientes facilidades

1. No es necesario un radio control, por lo general estos artefactos son caros y a la hora de poder controlar vehículos aéreos son obligatorios. Con *BEcopter* esto ya no es necesario y puede ser reemplazado por un simple Joystick de computadora.
2. Es *Open Source* por lo tanto puede ser modificado a necesidad del usuario.
3. Su desarrollo esta orientado a no simplemente el control de un tipo de vehículo (aéreo), sino a varios, tales como vehículos terrestres y/o acuáticos.¹

¹ Dicha característica será implementada en las próximas versiones.

Introducción a BEcopter

BEcopter es una plataforma para el guiado y navegación de vehículos aéreos no tripulados, en la cual en versiones posteriores se tendrá como fin abarcar más tipos de vehículos, ya sean aéreos o no. Este proyecto ha sido desarrollado con la ventaja de poder ser ejecutado en varias plataformas, por lo cual, puede correr en distintos tipos de sistemas operativos como Windows, Linux y MacOS. Se distribuye bajo la licencia GLP, por lo tanto, es de tipo open source y puedes realizar tus respectivas modificaciones si lo crees necesario; para realizar modificaciones debes ingresar al siguiente link en [GitHub](#)

Restricciones

Es importante mencionar, que al encontrarnos en etapas tempranas del *RELEASE*, el proyecto es compatible *inicialmente* con el siguiente hardware

Hardware

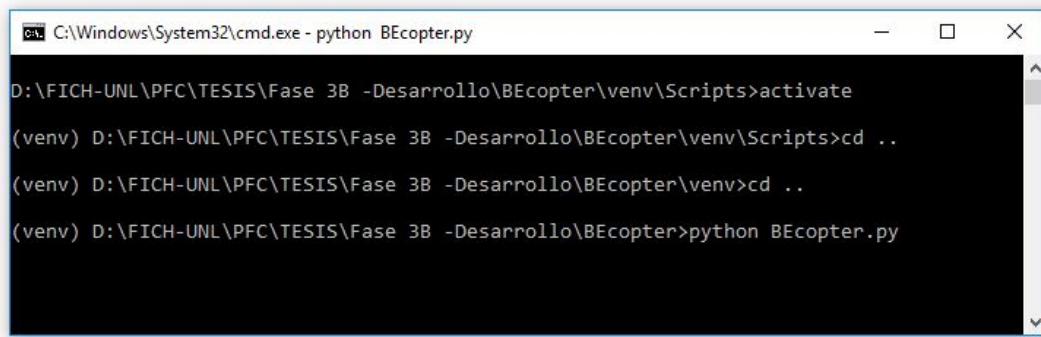
- Placa Raspberry Pi 3
- Placa Navio2. [Navio2](#)
- Estructura del vehículo con sus respectivos motores y cables de datos/alimentación.
- Un Joystick de PC.
- Batería tipo LiPo
- Router Wifi / Módulos XBee.

Software

- Tener instalado [Python 2.7.x](#)
- El vehículo debe tener instalado el firmware proporcionado por [ArduPilot](#) (Además de tener calibrados sus respectivos sensores)

Instalación

Para su respectiva instalación simplemente descargamos el proyecto desde [GitHub](#) y mediante los siguientes comandos podemos empezar a utilizar BEcopter



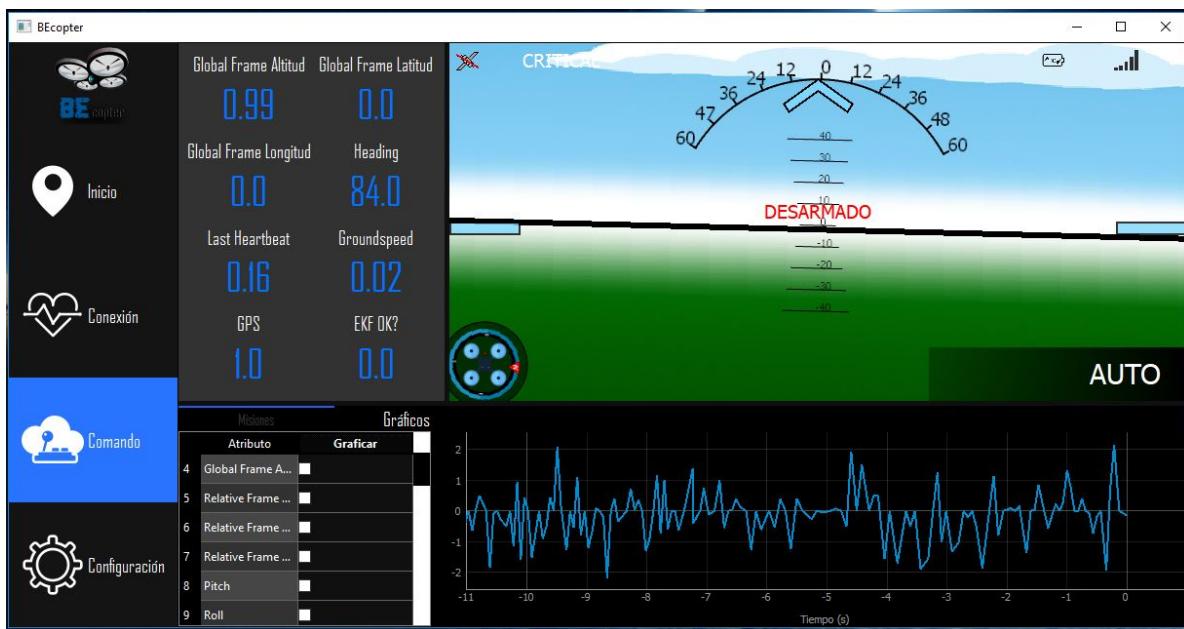
```
C:\Windows\System32\cmd.exe - python BEcopter.py
D:\FICH-UNL\PFC\TESIS\Fase 3B -Desarrollo\BEcopter\venv\Scripts>activate
(venv) D:\FICH-UNL\PFC\TESIS\Fase 3B -Desarrollo\BEcopter\venv\Scripts>cd ..
(venv) D:\FICH-UNL\PFC\TESIS\Fase 3B -Desarrollo\BEcopter\venv>cd ..
(venv) D:\FICH-UNL\PFC\TESIS\Fase 3B -Desarrollo\BEcopter>python BEcopter.py
```

Primeros pasos

Cuando iniciemos *BEcopter* por defecto nos mostrará la pestaña de ayuda, donde nos proporcionará información sobre el uso de *BEcopter*, en caso de necesitarla. En primera instancia vamos a tener habilitado dos pestañas que son la pestaña de *CONEXION* y la pestaña de *COMANDO*. Estas pestañas son de suma importancia ya que de estas dos depende el control del vehículo.

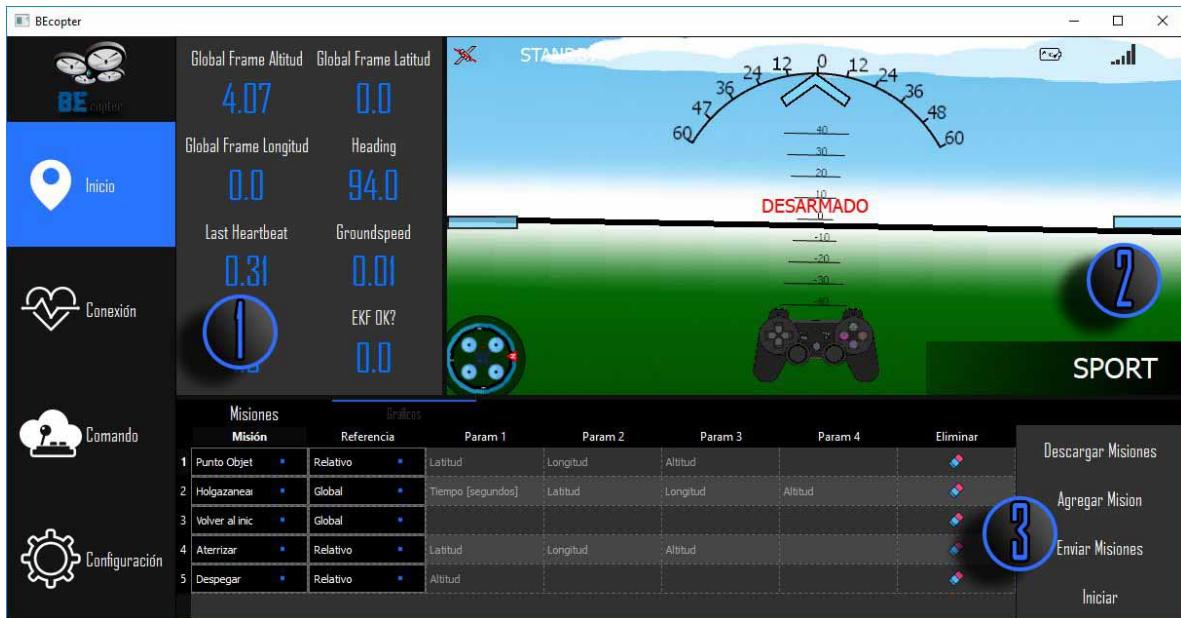
En la pestaña de conexión vamos a tener disponible dos campos en el cual debemos escribir el IP de nuestro equipo conectado a la red (debe ser la misma al que está conectado el vehículo) y el puerto por defecto es 14450. Una vez ingresados pulsamos conectar y si todo está correcto *BEcopter* establecerá una conexión con el vehículo mostrando la información pertinente. En la pestaña *COMANDO* se deberá seleccionar el Joystick que será el encargado de controlar el vehículo y además este habilitará el envío de misiones.

Por último, tendremos a *BEcopter* corriendo y en condiciones de volar nuestro vehículo.



Pestaña de Inicio

La pestaña de Inicio de *BEcopter* es la responsable de mostrar la información proveniente del vehículo de varias maneras y además, es la encargada de enviar los comandos al vehículo para realizar las respectivas maniobras. En esta pestaña podemos subdividirla en regiones, de la siguiente manera



Botonera

La región n° 1 contiene un conjunto de botones, los cuales pueden ser pulsados y tener la opción de elegir el parámetro que se desea mostrar según la necesidad del piloto. Estos son una alternativa a la gráfica mostrada en la región 3.b *Gráficos* pero con la diferencia que se pueden observar de manera instantánea los cambios de valores.



HUD (Head Up Display)



El HUD o según su respectiva traducción al español «Pantalla de cabeza arriba» es la encargada de mostrar gráficamente el estado actual del vehículo, con esto hacemos referencia que simula una vista en primera persona desde el vehículo con el fin de que el piloto vea el estado del mismo. Dentro del *HUD* podremos observar varios elementos gráficos que dará información en tiempo real de lo que esta sucediendo con nuestro vehículo. Estos elementos son:

GPS

Este elemento gráfico mostrará el estado del GPS, en caso de no ser aceptable la señal percibida mostrará un «satélite» con una línea cruzada, lo que indica que dicha señal no es aceptable. Para solucionar el problema, es aconsejable que

ubique el vehículo en un espacio libre de obstáculos, como edificios, árboles, fuentes de electromagnetismo, etc.

Estado del vehículo

El estado del sistema de vehículo es un aspecto importante a tener en cuenta, ya que si no se encuentra en condiciones no va a hacer posible despegar o recibir misiones. Esta es una lista de los estados que puede tener el vehículo:

- UNINIT: El estado es desconocido o todavía no se ha iniciado.
- BOOT: El sistema se encuentra arrancando.
- CALIBRATING: Un proceso de calibración se encuentra ejecutando, por lo tanto, no es posible arrancar.
- STANDBY: Se encuentra en la espera de comandos.
- ACTIVE: El sistema está activo y podría estar en el aire ejecutando maniobras.
- CRITICAL: El sistema está en un modo de vuelo anormal. Sin embargo, todavía puede maniobrar.
- EMERGENCY: Aparece cuando se ha perdido el control sobre el vehículo. De esta manera, este podría estar realizando un aterrizaje forzoso.
- POWEROFF: Se está iniciando el proceso de apagado del sistema.

Batería

Mostrará información del nivel de carga de la batería, en caso de encontrarse conectado a una fuente de alimentación mostrará en su interior las iniciales de DC *Corriente Directa*.

Señal

El nivel de señal estará basado según el último *heartbeat* o pulso recibido en BEcopter, entre mayor sea duración del último pulso menor será la calidad de señal recibida.

Ángulo de banco

Este instrumento indica al usuario la inclinación en el *eje x* o *Roll* del vehículo, con la particularidad de que se encuentra limitado en el rango $[-(-60^\circ), 60^\circ]$ y en caso de sobrepasar este límite se indicará con una flecha en color rojo a modo de precaución y estabilización.

Líneas de referencia

Estas líneas de referencia tienen el fin de mostrar en pantalla el pitch que está teniendo el vehículo en tiempo real, de la misma forma que el ángulo de banco, esta muestra un rango limitado de $[-40^\circ, 40^\circ]$.

Indicador de rumbos

O también conocido como giróscopos direccionales, es un instrumento que indica el rumbo que lleva la aeronave. Este instrumento se alinea en base a los polos magnéticos de la tierra según los datos obtenidos del magnetómetro dentro del vehículo. Por lo cual nos estaría indicando según *«la rosa de los vientos»* si nos estamos dirigiendo en dirección al Sur, Norte, Noreste, etc.

Icono de Josystick

Este icono representando un joystick, tiene la finalidad informar al piloto del vehículo que el modo de vuelo que tiene seleccionado el vehículo acepta comandos desde el Joystick, por lo tanto es posible controlar el vehículo en modo manual.

Modos

Dependiendo del objetivo del vehículo existen distintos tipos de vuelo se pueden aceptar, como, por ejemplo

Modo	Alt Ctr	Pos Ctrl	GPS	Resumen
Acro	-	-		Mantiene la posición, sin auto nivelarse
Alt Hold	s	+		Mantiene la posición y auto-controla el Roll & Pitch
Auto	A	A	Y	Se ejecutan misiones pre-cargadas
AutoTune	s	A	Y	Procedimiento automatizado de inclinación y banco para mejorar el control en los circuitos.
Brake	s	A	Y	Hace que el vehículo se detenga inmediatamente
Circle	s	A	Y	Empieza a girar sobre la posición actual del vehículo
Drift	-	+	Y	Igual a Stabilize, pero controla el Yaw & Roll tal como si fuera un aeronave
Loiter	s	s	Y	Mantiene la altitud y posición, usa el GPS para moverse
PosHold	s	+	Y	Igual a Loiter, pero el control del Roll & Pitch son manuales si se percibe comando del joystick.
Stabilize	-	-		Autonivela el Roll & Pitch.
Sport	s	s		Alt-hold, but holds pitch & roll when sticks centered
Throw	A	A	Y	Mantiene la posición luego de realizar un despegue.
Land	A	A	Y	Reduce su altitud en línea recta hasta conseguir aterrizar.
RTL	A	A	Y	Retorna al punto de despegue inicial.
Guided No GPS	A	A		Es similar a Guided, exceptuando que no requiere el GPS y únicamente acepta misiones de posición.
Guided	A	A		Solo acepta misiones de posición.

Símbolo	Definición
-	Control Manual
+	Control manual con limitación de altura
s	Estabilización automática controlada
A	Control Automático

Misiones y Gráficos

En la parte inferior de la pestaña *Inicio* nos encontramos con dos pestañas más, la primera (denotada como sección 3a) contiene las opciones de gestión de misiones sobre el vehículo. En la sección 3b vamos a poder observar y seleccionar los datos que deseamos observar su respectiva evolución en el tiempo.

3.a Misiones

Misiones		Trámites					
Misión	Referencia	Param 1	Param 2	Param 3	Param 4	Eliminar	
1 Punto Objetivo	Relativo	Latitud	Longitud	Altitud			
2 Holgazaneal	Global	Tiempo [Segundos]	Latitud	Longitud	Altitud		
3 Volver al inic	Global						
4 Aterrizar	Relativo	Latitud	Longitud	Altitud			
5 Despegar	Relativo	Altitud					

3a

- Descargar Misiones
- Agregar Mision
- Enviar Misiones
- Iniciar

BEcopter proporciona un conjunto de misiones que pueden ser enviados al vehículo con el propósito de que sean ejecutadas de manera secuencial. Pero antes de describir las distintas opciones que tenemos, hay que tener en mente cierto conceptos tales como los son las *Coordenadas Globales* y *Coordenadas Relativas*: Al momento de asignar misiones al vehículo podemos observar que la mayoría depende de coordenadas para ser enviadas, por lo tanto hay que tener en cuenta que estas coordenadas pueden estar asociadas a un sistema de referencia

- **Global:** Sistema de georreferenciación [latitud, longitud, altura] que pueden ser proporcionadas por el GPS.
- **Relativo:** El sistema de referencia tiene su origen en la ubicación de despegue del vehículo.

Dentro de las alternativas tenemos las siguientes opciones que podemos enviarles, que son:

- **Punto Objetivo:** Esta misión recibe las coordenadas de la nueva posición a la que debe ubicarse el vehículo (según el sistema de referencia establecido).
- **Suspenderse:** Dado un cierto punto de ubicación esta misión tiene como finalidad mantenerse en el aire por una cantidad de tiempo (en segundos) estipulada por el usuario.
- **Volver al inicio:** Antes de iniciar con las misiones el vehículo tomará como punto de partida la posición de despegue, por lo tanto cuando se ejecute esta misión el vehículo tratará de volver a dicha ubicación almacenada.
- **Aterrizar:** Para realizar esta acción es necesario establecer un punto de aterrizaje, el vehículo se dirigirá a dicha ubicación (más allá de la altura establecida) procederá a descender de manera suave hasta detectar que su respectiva altura no está teniendo cambios.
- **Despegar:** Cuando se ejecute esta misión, el vehículo ascenderá desde el suelo lentamente hasta llegar a la altura establecida por el usuario.

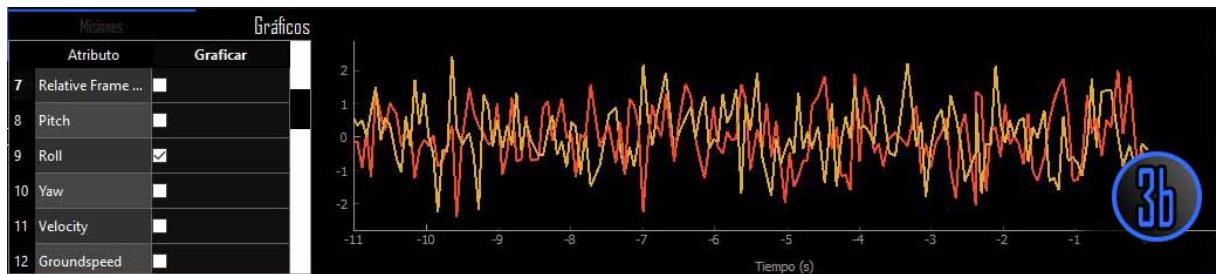
Atención: Las características de cada misión pueden ser modificadas en la pestaña Configuración, como por ejemplo, cantidad de misiones, velocidad de ascenso y descenso, etc.

Luego de indicar los tipos de acciones es momento de enviarlas al vehículo, para esto BEcopter brinda un conjunto de acciones que gestionan estas misiones. En la parte derecha de la lista de misiones podemos observar que contamos con un conjunto de botones de la siguiente manera:

- **Descargar Misiones:** Esta opción descarga las misiones que están almacenadas y no ejecutadas hasta el momento por el vehículo.
- **Agregar Misión:** Agrega una nueva fila en la lista de misiones de BEcopter.

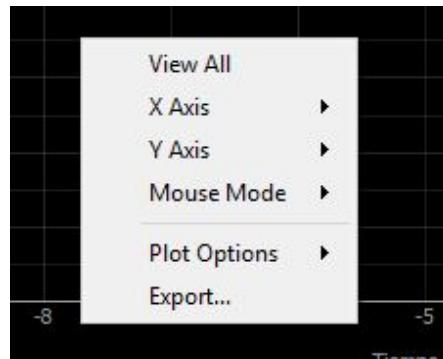
- **Enviar Misiones:** Valida y en caso de estar todo correcto, envía las misiones al vehículo para que sean almacenadas.
- **Iniciar:** Realiza un chequeo de que el vehículo y joystick estén en condiciones para realizar las misiones, y en caso afirmativo se procederá a informar el inicio de las mismas o no.

3.b Gráficos

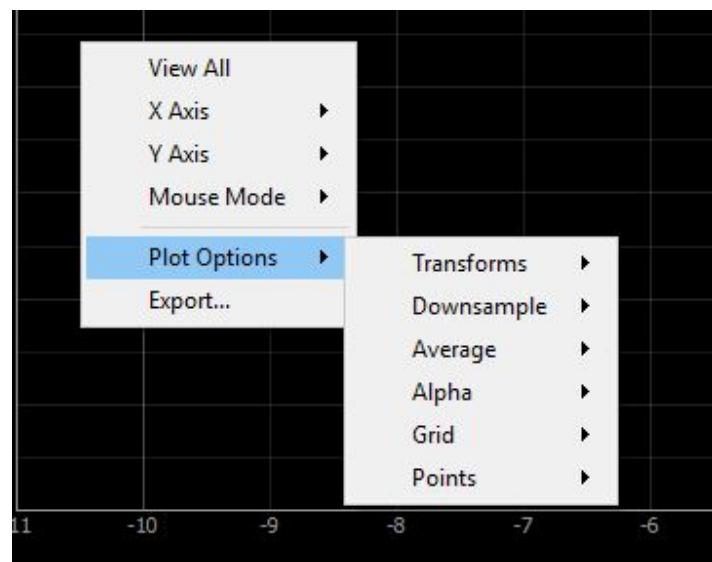


En este apartado podremos observar los valores de los sensores del vehículo como pueden ser Pitch, Roll, Velocidad, Altitud , etc. En la parte izquierda existe una lista con los atributos disponibles para graficar en la parte derecha; a continuación del nombre del atributo podemos ver un *checkbox*, el cual tiene como finalidad ser un valor de condición para poder ver los valores pertinentes en la gráfica. Es importante mencionar que por razones de apreciación la gráfica solamente podrá contener 5 atributos graficados de manera simultánea, por lo tanto, una vez que hayamos seleccionado el 6to atributo automáticamente se procederá a eliminar el primer atributo graficado.

De manera complementaria *BEcopter* ofrece sobre los gráficos opciones que personalizan el comportamiento del mismo. Dentro de estas herramientas tenemos las siguientes opciones

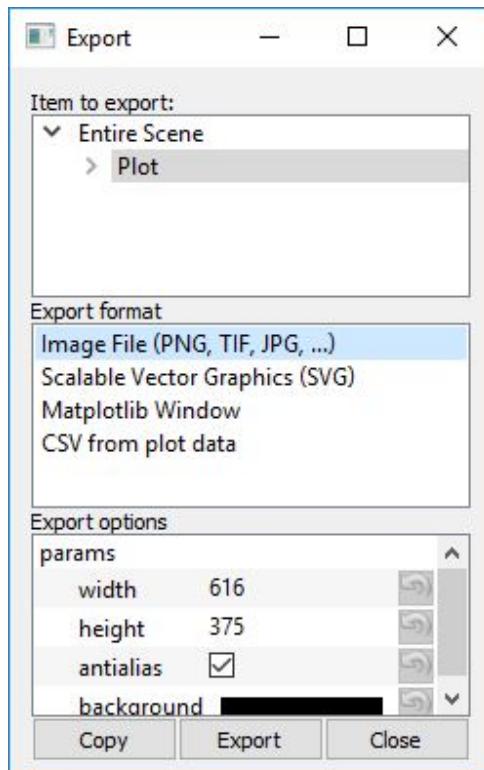


- **Ver todo:** Reestablece la vista para que se puedan observar todas las gráficas presentes.
- **Eje X:** Opciones sobre el eje x, como definir el intervalo dinámico o invertido.
- **Eje Y:** Opciones sobre el eje y, como definir el intervalo dinámico o invertido.
- **Opciones de ploteo:**



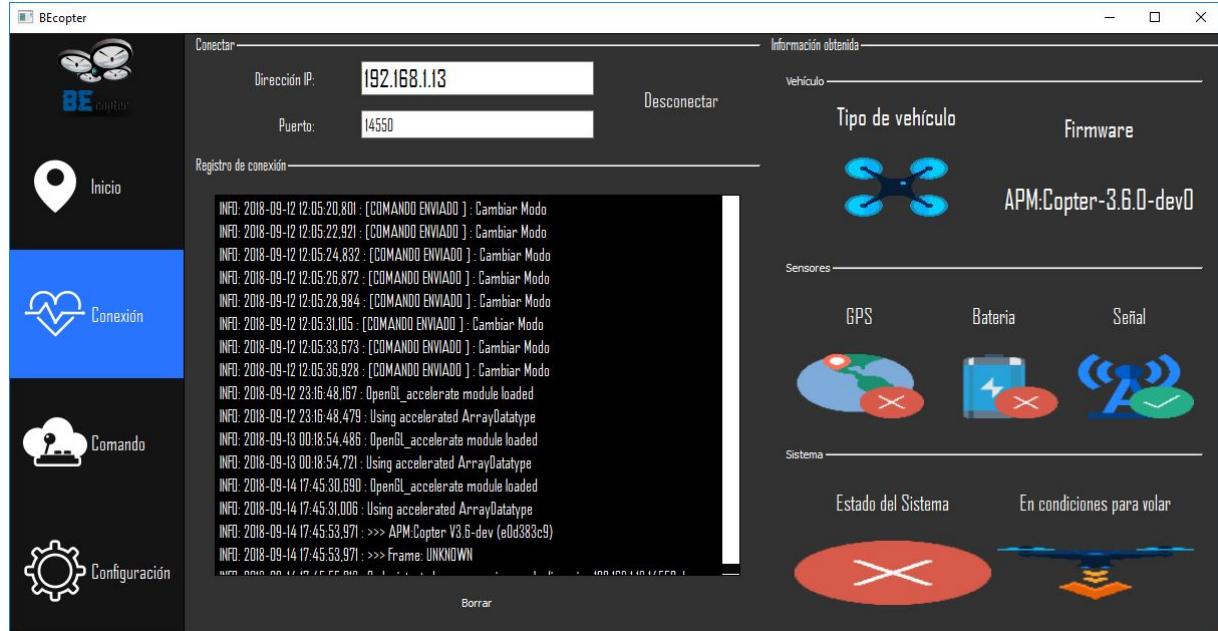
Esta opción contendrá varias funcionalidades que personalizan la estética de la gráfica como pueden ser: Alpha (Transparencia) , Grid (Grilla) , Points (Puntos) Average (Promediado), Downsample (Muestreado). Por último y de manera complementaria podemos aplicar transformaciones a los datos mostrados, en caso de necesitar ver otro punto de vista de los datos como puede ser la *Transformada rápida de Fourier*, *Transformación Logarítmica de X* y la *Transformación Logarítmica de Y*.

- **Exportar datos:** Por último *BEcopter* ofrece la opción de poder exportar los datos graficados actualmente, dando las distintas alternativas de tipos de formato de guardado, como puede ser imágenes, formato SVG (*Gráfico vectorial escalable*), exportarlo a una ventana utilizando la interfaz gráfica de Matplotlib y por último, simplemente como CSV (Valores separados por coma).



Pestaña de Conexión

En esta pestaña como su nombre si indica se brindan las opciones referentes a la conexión con el vehículo. Además, muestra información (una vez conectado) del estado del vehículo, como también los mensajes que se intercambian.



En la región de conectar, tenemos disponible de dos campos. El primero es para ingresar el IP de red asignado a la maquina en el cual está corriendo *BEcopter*. Debemos prestar atención en que debemos estar conectados a la misma red al que está conectado el vehículo. Una vez ingresado ese valor y dejando por defecto el puerto ya asignado (siempre y cuando estos valores sean los mismos que estén configurados en el vehículo) para más información se puede consultar el siguiente link de [Navio2](#)

En el apartado de *Registro de conexión* obtendremos toda información referente a los mensajes que hemos enviado al vehículo y en contraparte con respecto al vehículo.

Advertencia: Por lo tanto, es de buena práctica utilizar dicha información en caso de que ocurriera algún tipo de error o inconsistencia.

Una vez conectados al vehículo en la región de *Información obtenida* se mostrarán datos tales como :

- **Tipo de vehículo:** Ya sea tricóptero, hexacóptero, cudricóptero, etc.
- **Firmware:** La información mostrada aquí, corresponderá a la versión del firmware instalado en el vehículo.

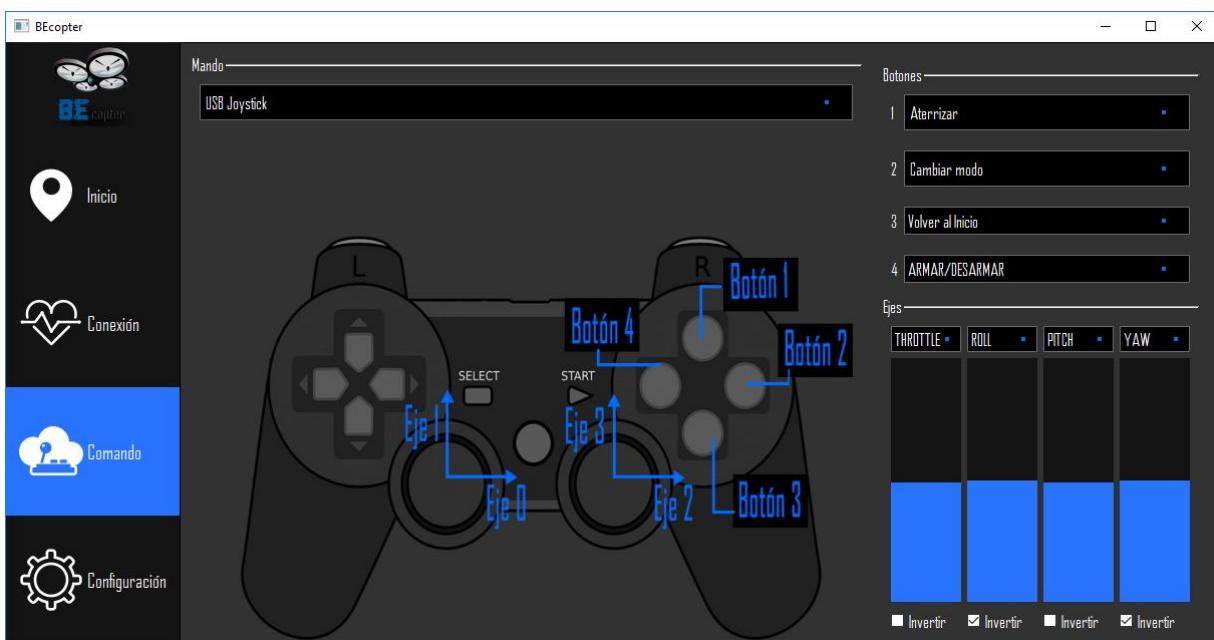
Atención: Corroborar que su vehículo se encuentre actualizado.

Antes de poder realizar un despegue es de suma importancia que el vehículo se encuentre en condiciones para realizarlo. Es por tal motivo, que antes de comenzar a iniciar con las misiones o un despegue se deberá controlar que todos los sensores como GPS, Batería, Señal , posicionamiento estén correctamente en condiciones. En caso contrario el inicio de un despegue será simplemente rechazado.

En el apartado de *Sistema* tenemos dos iconos, el primero representa el resultado si el vehículo está bien posicionado para iniciar el vuelo, para obtener este valor se comprueba que el vehículo no se encuentre muy inclinado (ya sea verti-

cal o horizontalmente), que no este en movimiento y los sensores calibrados, como compas, giróscopo y acelerómetro. Luego, el siguiente icono «es una traba de seguridad» con el fin de no incidir en un vuelo accidental, por lo tanto, una vez que el vehículo se encuentre en condiciones, es decir, todos los iconos muestren resultados favorables se tendrá que «ARMAR» el vehículo, para realizar esto se deberá enviar un comando mediante un Joystick ordenando a vehículo que este preparado para iniciar su vuelo. Este comando puede ser asignado al Joystick en la pestaña *COMANDO*.

Pestaña de Comando



La pestaña *COMANDO* es una de las primeras en estar habilitadas, ya que la selección de un joystick para el control del vehículo es de suma importancia para la navegación del mismo. Es por tal motivo que esta pestaña proporciona la opción de seleccionar un joystick que este conectado al equipo en donde este corriendo *BEcopter* y poder asignarle a una cierta cantidad de acciones predefinidas.

Mando

En esta sección podemos encontrar una lista con Joysticks reconocidos por *BEcopter*. En donde se podrá ver su respectiva configuración, en cada botón y *sticks* del joystick.

Advertencia: El gráfico del comando mostrado en pantalla puede no representar estrictamente la numeración de los botones en el joystick real.

Atención: Una manera de corroborar la correcta asignación de acciones es mediante la consola dentro de la pestaña *CONEXION*

Botones

En este apartado se pueden asignar accesos rápidos a los botones, que pueden ser ejecutadas en tiempo real por el vehículo. Las opciones disponibles son:

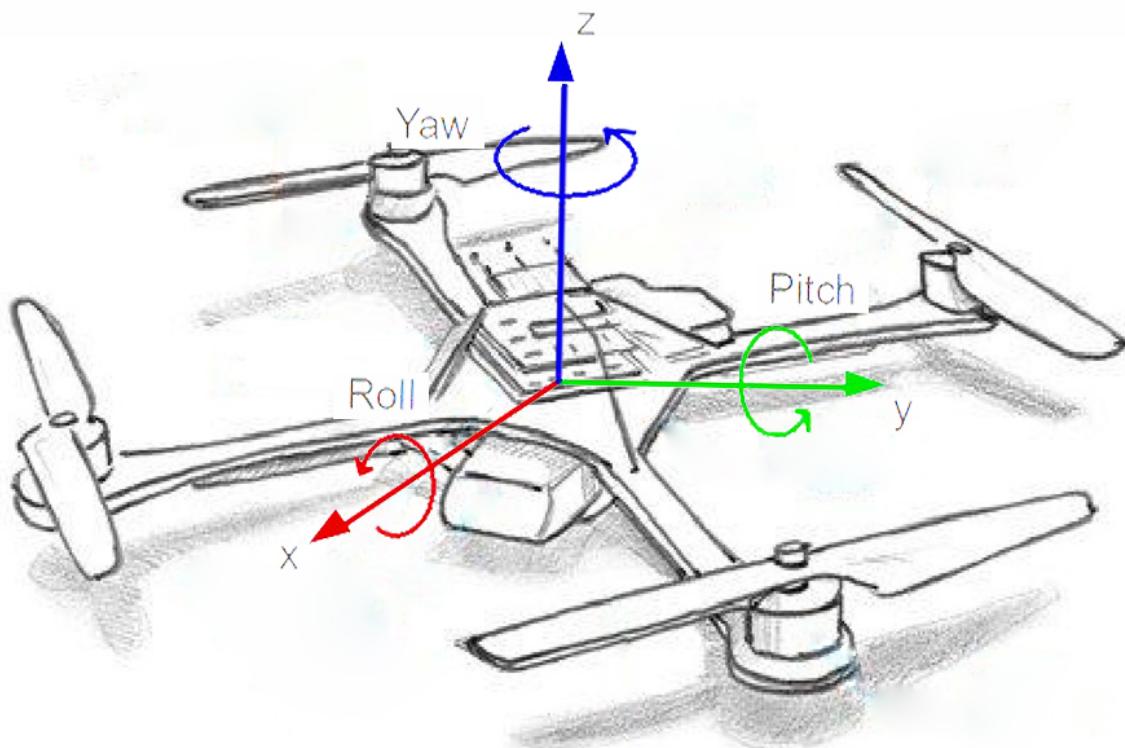
- **Aterrizar:** Como se explicó en la pestaña de *Inicio* este comando le ordena al vehículo desender suavemente hasta no encontrar cambios en la altura del mismo.
- **Cambiar Modo:** En caso de necesitar cambiar el modo de un control manual o automático (entre otras opciones), esta opción es la que se debe seleccionar.

- **Volver al inicio:** En caso de necesitar que el vehículo regrese a la posición inicial de partida, por cierta circunstancia como por ejemplo : poca señal, poca batería, el vehículo no se encuentre visible al piloto, etc. Esta opción es la indicada.
- **Armar/Desarmar:** Como medida de seguridad el vehículo cuenta con la opción de «trabar o desatrabar» la habilitación para comenzar a volar (el objetivo de este comando es evitar despegues/vuelos accidentales) por lo tanto esta opción alterna entre el desarmado y armado del vehículo.

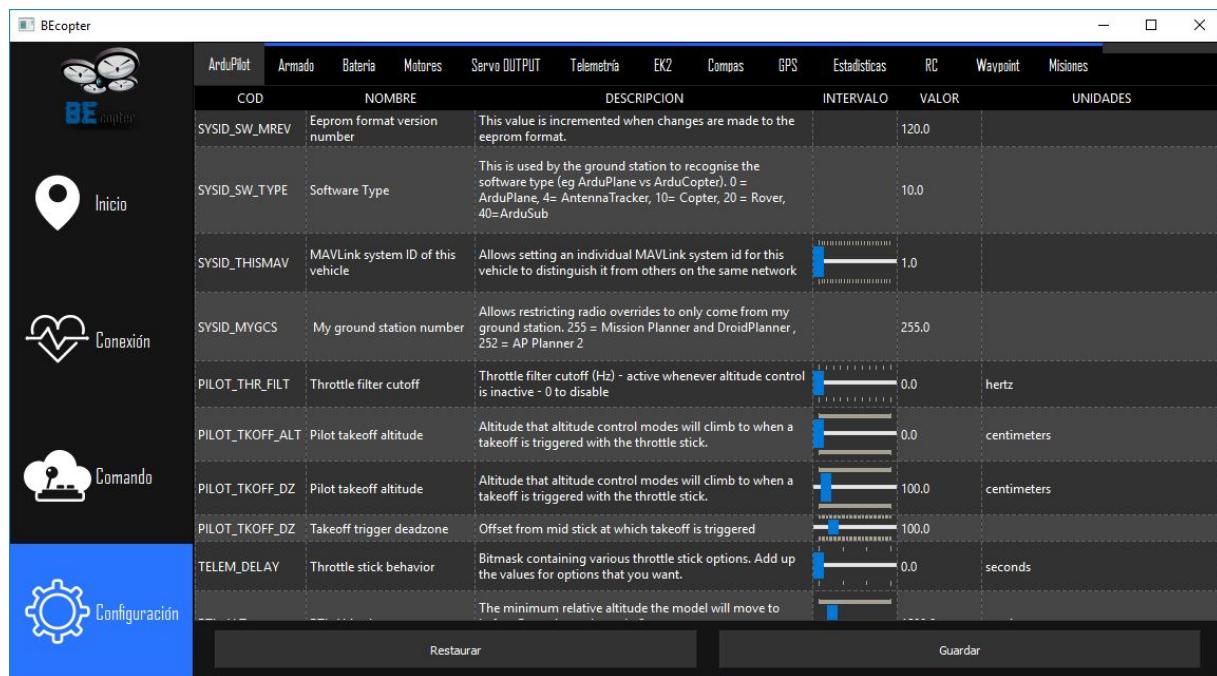
Ejes

Los ejes estan asociados a los valores proporcionados por las palancas «analógicas» comunmente llamadas. Estos valores serán reflejados en las barras ubicadas en la parte inferior izquierda de la pestaña de *COMANDO* con la opción de poder asignarles los siguientes movimientos

- **THROTRLE:** Potencia brindada a los motores.
- **ROLL:** Giro sobre el eje X del sistema de referencia.
- **PITCH:** Giro sobre el eje Y del sistema de referencia.
- **YAW:** Giro sobre el eje Z del sistema de referencia.
- **Invertir eje:** Por último y en la parte inferior de cada barra esta la opción de poder invertir el orden de los valores. Ya que existen casos en que los valores originales provenientes del periférico provienen en sentido contrario.



Pestaña de Configuración



Por último *BEcopter* proporciona la posibilidad de modificar los parámetros de configuración del Autopiloto los cuales determina el comportamiento del vehículo, como puede ser velocidad de despegue, voltaje máximo enviado a los motores, condiciones de pre-armado como también información del uso del vehículo, entre otras cosas.

BEcopter ha organizado los parámetros del vehículo en secciones, para una mejor organización. Por lo tanto, tenemos las siguientes pestañas

- **Parámetros generales:** Se configuran aspectos generales del Autopiloto, como restricciones en velocidad, altitud , posición en cada modo.
- **Armado** En este apartado encontraras los parámetros para el pre-armado.
- **Batería** Aquí se tendrá todo en cuenta los aspectos generales de la batería, como definir el tipo de batería instalada y sus propiedades.
- **Motores** El comportamiento de los motores al momento de realizar algún tipo de acción se puede definir en este apartado, como también sus respectivas restricciones.
- **Servo OUTPUT:** Además de poder configurar los motores, es posible administrar los pines extras en el cual pueden estar conectados servo motores. Como por ejemplo la frecuencia de la señal emitida en PWM.
- **Telemetría:** Se definen todas las características del módulo de telemetría instalado en su vehículo.
- **EK2:** *Extended Kalman Filter v2* o Filtro extendido de Kalman ([+info](#)) , es un algoritmo de estimación de posición, velocidad y angulación del vehículo según los datos provenientes de los sensores como el giroscopio, acelerómetro, compás, GPS, velocidad del viento y barómetro. Esto ayuda al Autopiloto tener un conocimiento del estado del vehículo, por lo tanto, al ser un algoritmo parametrizable el mismo puede configurarse en esta pestaña.
- **Compas:** Se muestran todas las características del compás instalado en su vehículo.
- **GPS** Se muestran todas las características configurables del GPS instalado en su vehículo.

- **Estadísticas:** Se muestra información de las horas de vuelo, cantidad de inicios del sistema y de reinicios del Autopiloto.
- **Radio Control:** Se muestran todas las características configurables del RC instalado en su vehículo. Como la restricción de sus correspondientes canales (Yaw, Pitch, Roll y Throttle).
- **WayPoints:** Cuando se realicen misiones, el vehículo se debe mover de un punto a otro a una cierta velocidad, como también puede acelerar, entre otras cosas. Aquí se podrá configurar dicho comportamiento.
- **Misiones:** En esta pestaña dedicada a las misiones se tendrá información sobre el total de misiones que se pueden cargar y el comportamiento del vehículo al momento de cambiar entre control manual y automático.

Advertencia: La manipulación de estos valores queda sumamente bajo la responsabilidad del usuario. En caso de necesitar información extra puede consultar la siguiente enlace de [MAVLink](#) que es el protocolo que se utiliza para la comunicación entre el vehículo y BEcopter.

Apéndice C

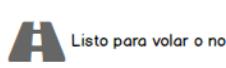
Mockups de la interfaz gráfica del proyecto

Main

Icono | Info



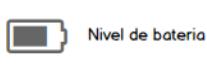
Conectado o no



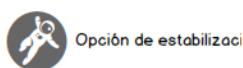
Listo para volar o no



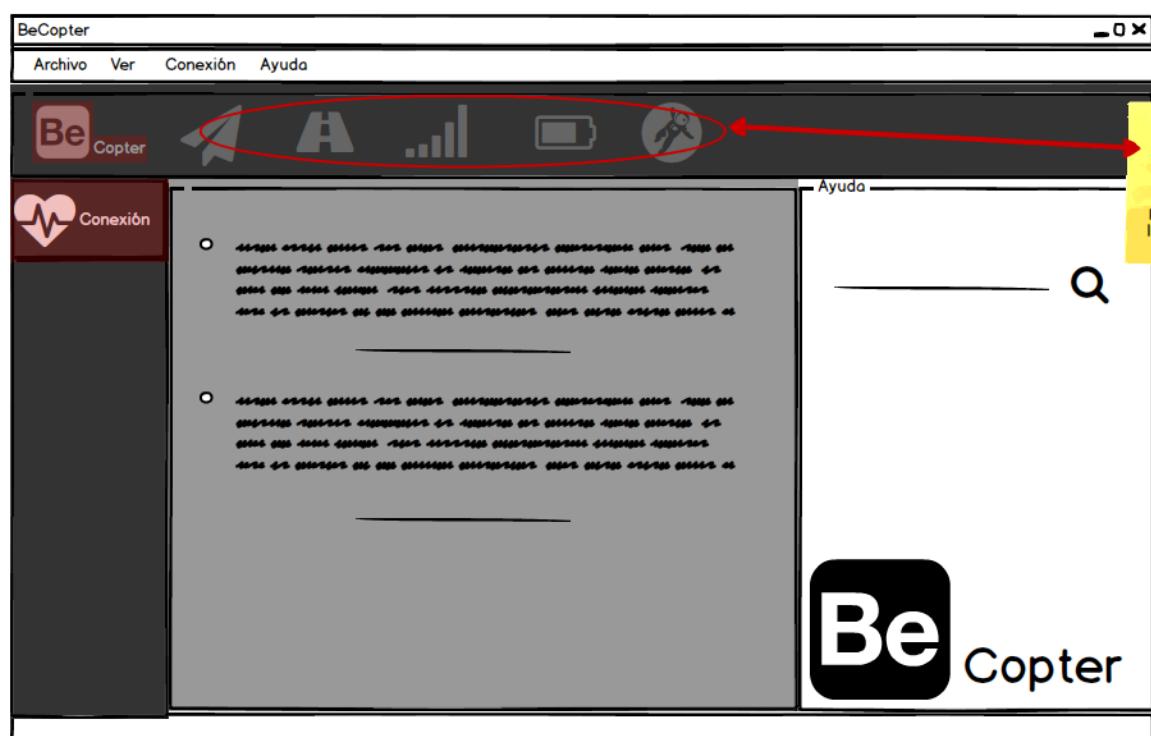
Nivel de señal



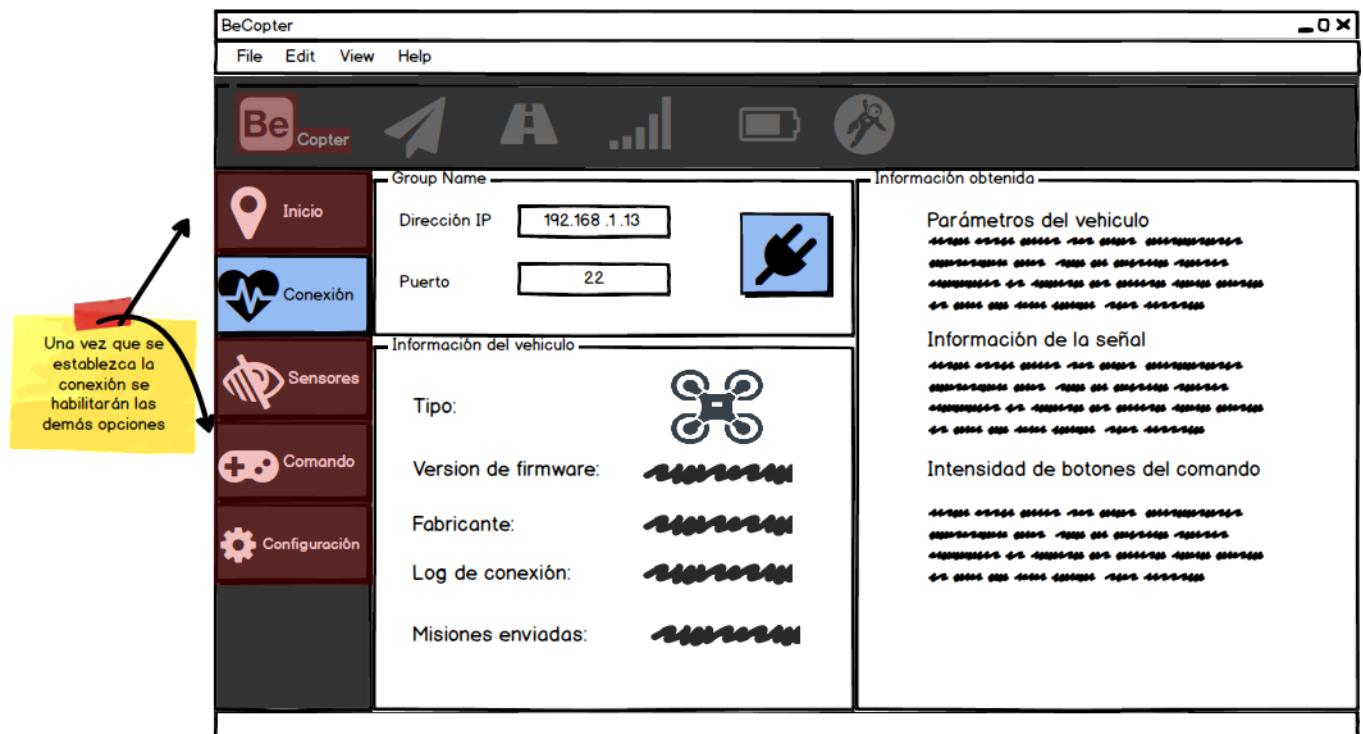
Nivel de batería



Opción de estabilizaci

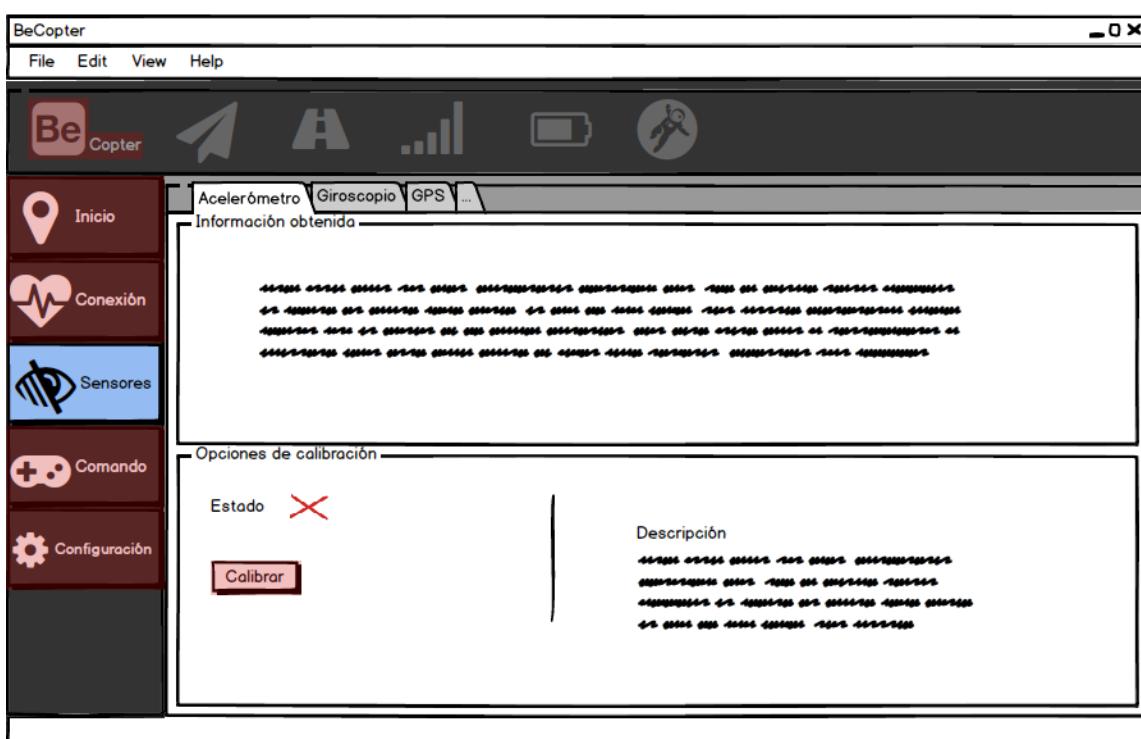






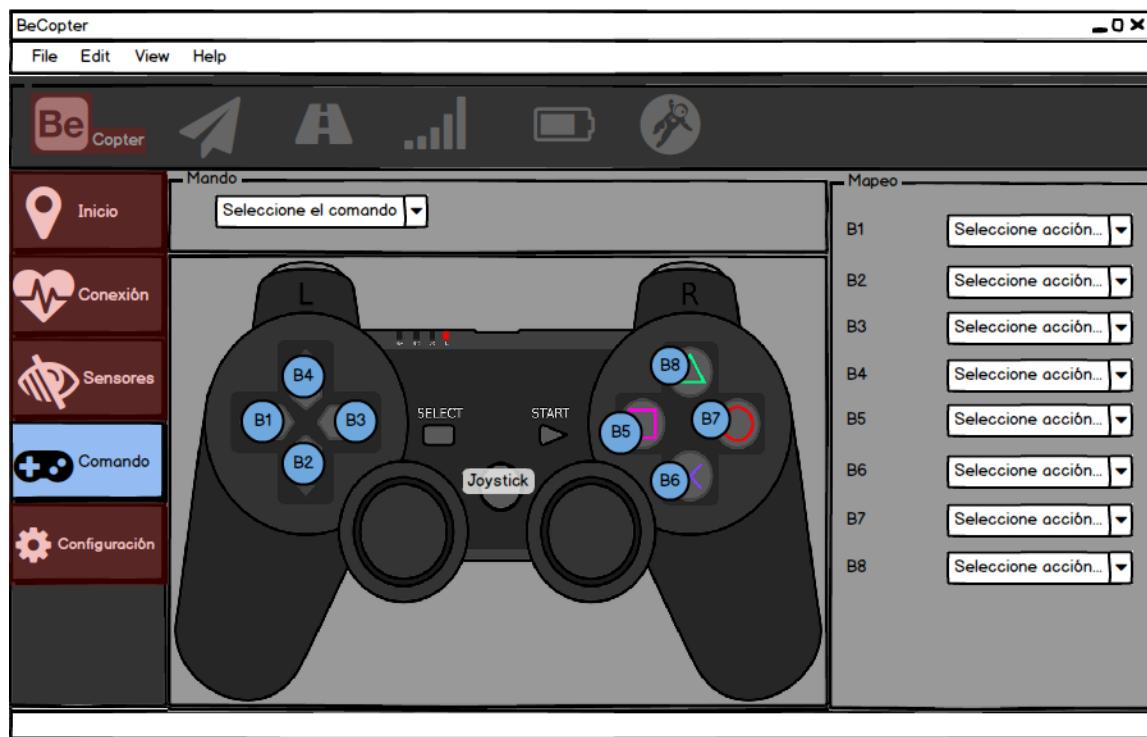
Sensores

4 / 8



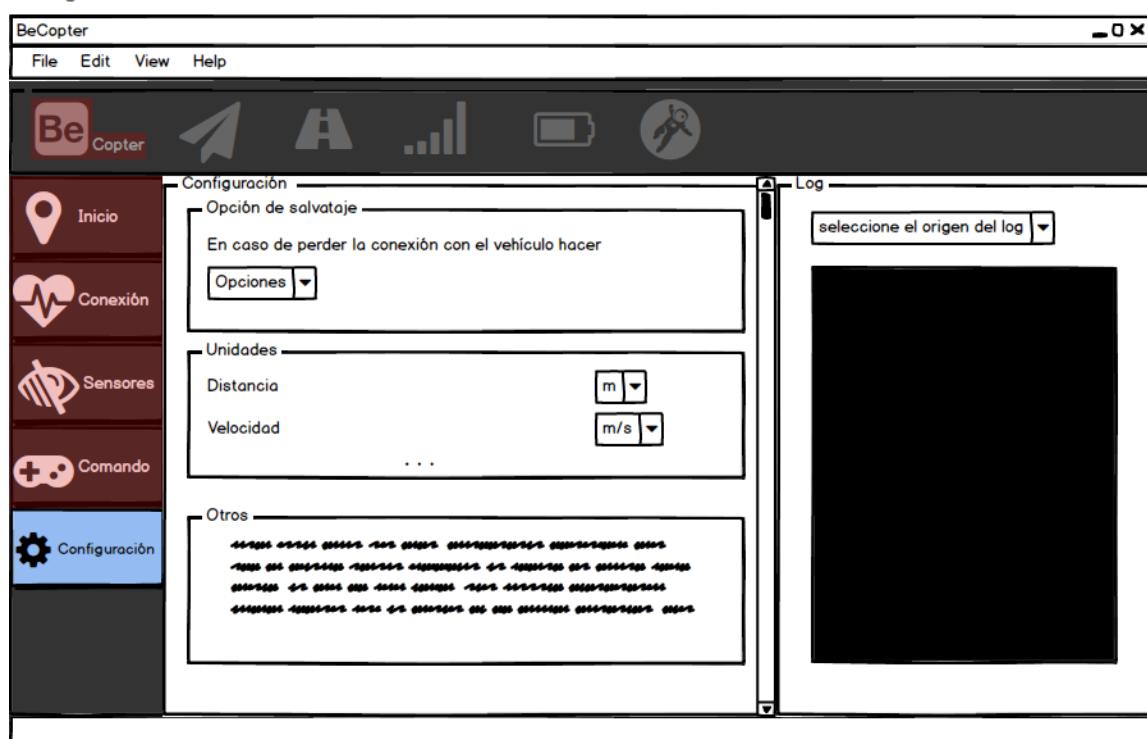
Comando

5 / 8



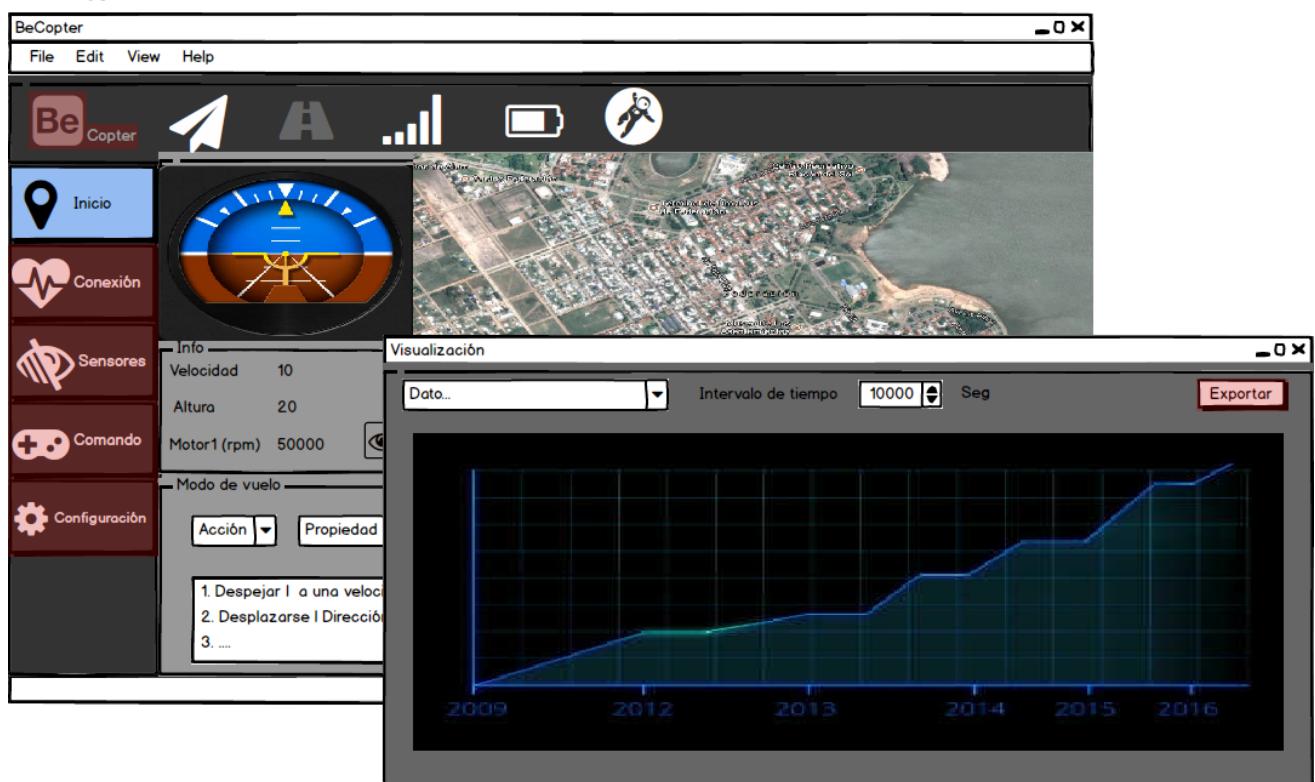
Configuración

6 / 8



Inicio copy

7 / 8



Calibrar

8 / 8



Apéndice D

Interfaz Gráfica del Usuario (GUI)

Apéndice E

Diagrama de clases inicial del Proyecto

Diagrama de clases BEcopter

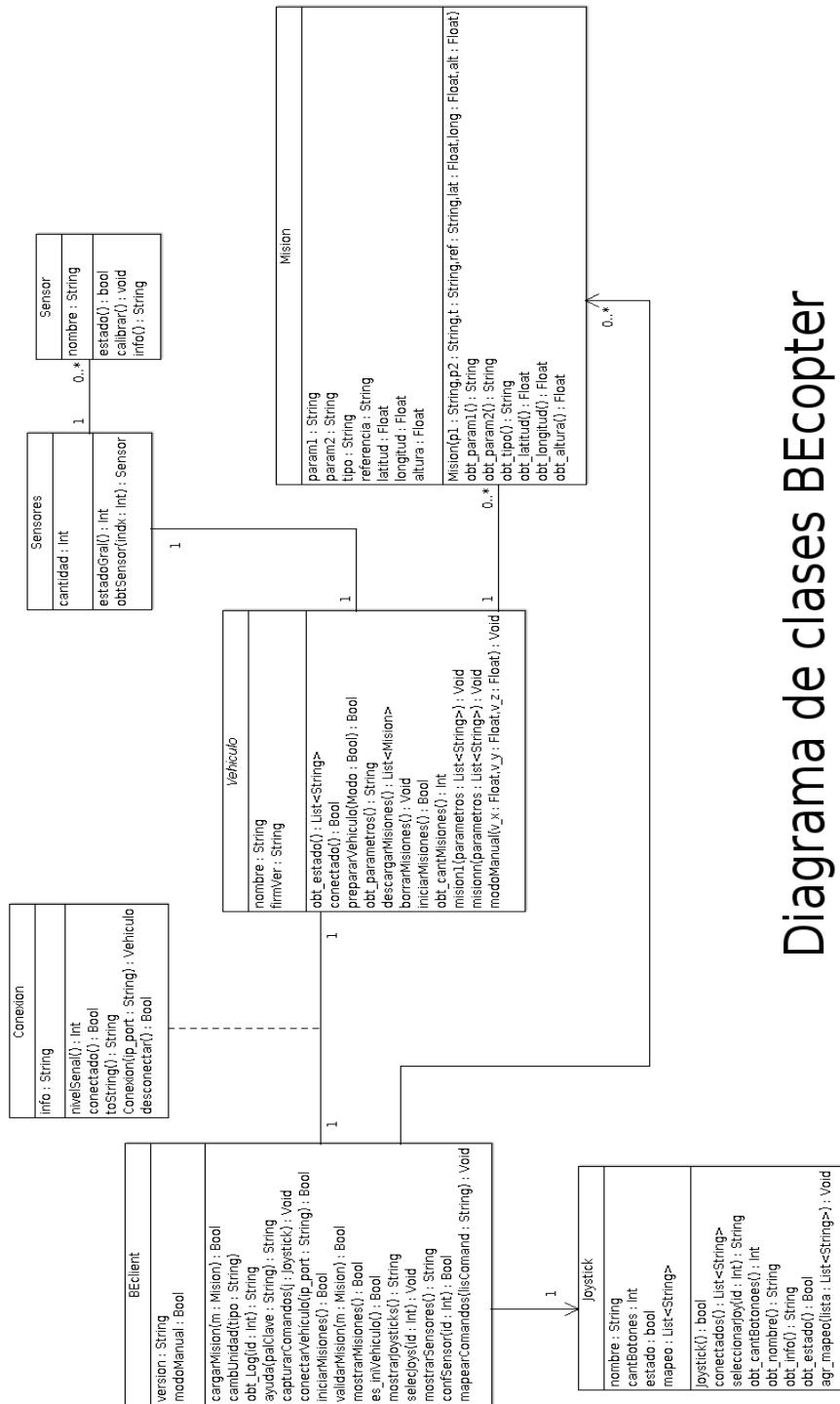


Figura E.1: Diagrama de clases inicial de BEcopter.

Apéndice F

Diagrama de clases final del Proyecto

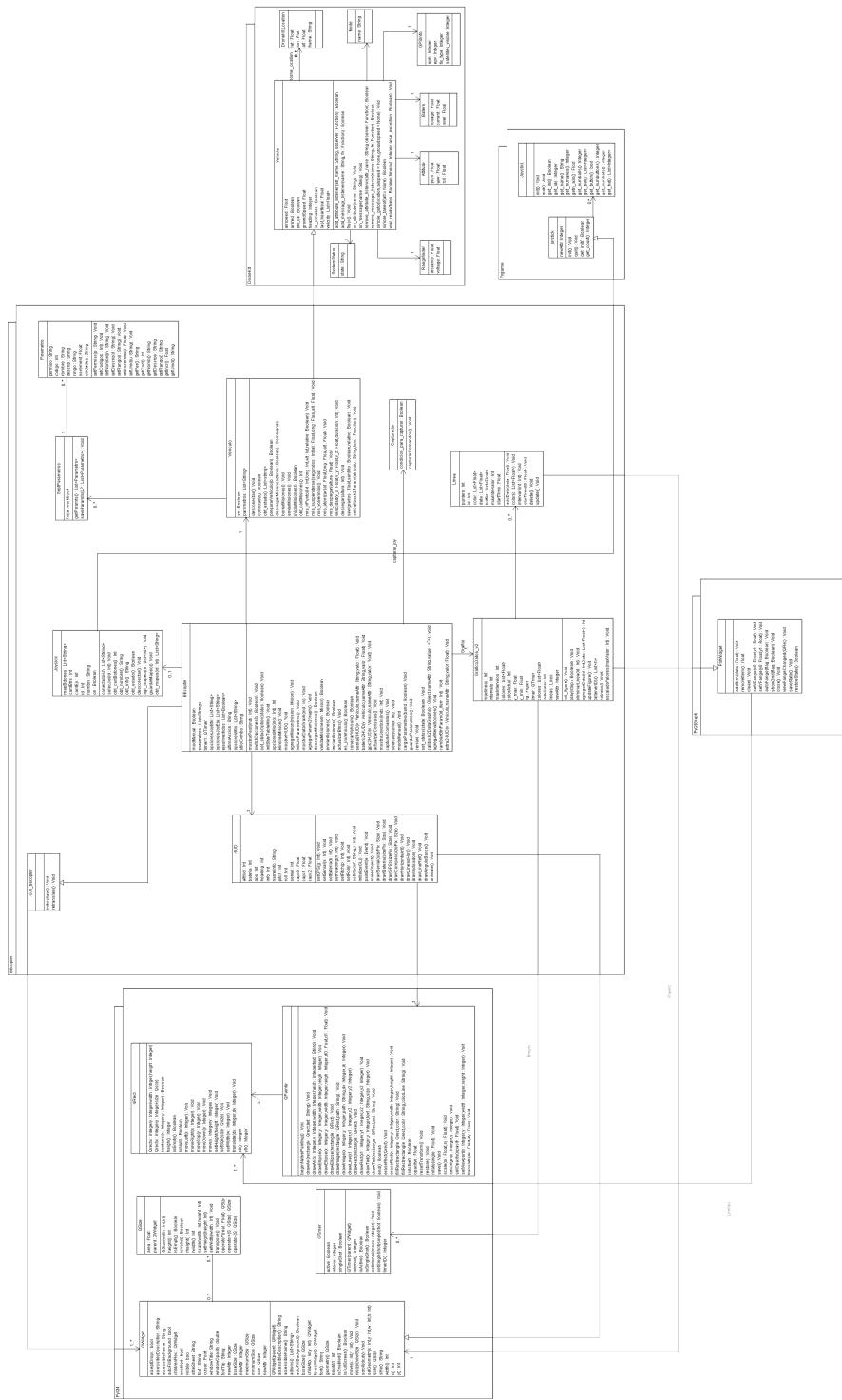


Figura F.1: Diagrama de clases v2, perteneciente al proyecto BEcoper

Bibliografía

- [1] Stuart M Adams, Marc L Levitan, and Carol J Friedland. High resolution imagery collection utilizing unmanned aerial vehicles (uavs) for post-disaster studies. In *Advances in Hurricane Engineering: Learning from Our Past*, pages 777–793. 2013.
- [2] Patrick Doherty and Piotr Rudol. A uav search and rescue scenario with human body detection and geolocation. In *Australasian Joint Conference on Artificial Intelligence*, pages 1–13. Springer, 2007.
- [3] Sommerville Ian. *Ingeniería del software*. Pearson Educacion, 2005.
- [4] Dewi Jones. Power line inspection-a uav concept. In *Autonomous Systems, 2005. The IEE Forum on (Ref. No. 2005/11271)*, pages 8–pp. IET, 2005.
- [5] Iván Maza, Fernando Caballero, Jesús Capitán, José Ramiro Martínez-de Dios, and Aníbal Ollero. Experimental results in multi-uav coordination for disaster management and civil security applications. *Journal of intelligent & robotic systems*, 61(1-4):563–585, 2011.
- [6] Anibal Ollero and L Merino. Unmanned aerial vehicles as tools for forest-fire fighting. *Forest Ecology and Management*, 234(1):S263, 2006.
- [7] Roger S Pressman and Jose Maria Troya. Ingeniería del software. 1988.
- [8] Kimon P Valavanis. *Advances in unmanned aerial vehicles: state of the art and the road to autonomy*, volume 33. Springer Science & Business Media, 2008.
- [9] Kimon P Valavanis, Paul Oh, and Les A Piegl. *Unmanned Aircraft Systems: International Symposium on Unmanned Aerial Vehicles, UAV 08*. Springer Science & Business Media, 2008.
- [10] Chunhua Zhang and John M Kovacs. The application of small unmanned aerial systems for precision agriculture: a review. *Precision agriculture*, 13(6):693–712, 2012.

Parte II

Anexos

