UNIVERSIDAD DEL BÍO-BÍO

Facultad de ciencias empresariales

Departamento de sistemas de información

Navegación asistida mediante detección y evasión de obstáculos para vehículos aéreos no tripulados.

Memoria para optar al título de Ingeniero Civil en Informática.

Jorge Lucas Maximiliano Torres Faúndez

Profesor guía:
Roberto Mercado Cuevas

Jorge Lucas Maximiliano Torres Faúndez
Concepción - Chile
jlmax.torres@gmail.com

A los que confiaron,
y esperaron...

Abstract

The convergence in use of information systems and personal electronic devices has led to the fact that it is present in virtually all aspects of daily life, the concept of computing in the ubiquitous or internet of things, where the interconnection of these results in drastically improve our lives.This work is intended to be inserted in this context with the aim to develop and implement a software solution that uses python, embedded in a Raspberry pi↔Ardupilot Mega system, to equip an UAV with ultrasonic sensors to detect and evade obstacles automatically, assisting the operator on the flight.

The python libraries GPIOZERO and RPI.GPIO are used for the handling of five sensors on board of UAV, a Raspberry is used to process information and the action commands are sent to the Ardupilot Mega through MAVLink protocol, so that the drone can react in front of the data received by the sensors. This project also presents a series of tests and results of distance's diagnosis algorithms with ultrasound sensors, evasion algorithms and communication algorithms in RPi↔APM via MAVLink Protocol.

Resumen

La convergencia en el uso de sistemas de información y dispositivos electrónicos digitales, ha llevado a que estén presentes en prácticamente todos los aspectos de la vida cotidiana, el concepto de computación ubicua o internet de las cosas, donde la interconexión de estos proporciona mejoras sustanciales a nuestras actividades es el contexto en el que este trabajo es desarrollado. Con el objetivo de desarrollar e implementar una solución de software utilizando python, embebida en un sistema Raspberry pi↔Ardupilot Mega, para dotar a un drone de un conjunto sensores de ultrasonido para detectar y evadir obstáculos de forma autónoma, asistiendo al operador durante el vuelo.

Se utilizan las librerías python GPIOZERO y RPI.GPIO para el manejo de cinco sensores a bordo del drone, se utiliza una Raspberry pi para procesar la información y enviar comandos de acción al Ardupilot mega mediante el uso del protocolo MAVLink, para que el drone pueda reaccionar frente a los datos recibidos por los sensores. Este proyecto también presenta una serie de pruebas y resultados de los algoritmos de medición de distancia para los sensores de ultrasonido, algoritmos de evasión y comunicación RPi↔APM vía serial con protocolo MAVLink.

Índice

Índice de algoritmos

Índice de tablas

Índice de figuras

Nomenclatura

Siglas Generales

APM	—	ArduPilot Mega.
BEC	—	Circuito eliminador de batería.
BLM	—	Motores sin escobillas, BrushLess Motor.
ESC	—	Controlador electrónico de velocidad, del ingles, Electronic Speed Controller.
GCS	—	Ground Control Station. Estación de Control en Tierra.
GPS	—	Sistema de posicionamiento global. Global Positioning System.
$GPIO$	—	Entrada/Salida de Proposito General. General Purpose Input/Out-Put.
IMU	—	Unidad de Medición Inercial. Inertial Measurement Unit.
$LiPo$	—	Lithium Polymer batteries. Baterias de Polímeros de Lítio.
MAV	—	Micro vehículo aéreo, del inglés, Micro Air Vehicle.
MCU	—	Abreviación de Micro-controlador.
PDB	—	Placa de distribución de energía, Power Distribution Board.
RC	—	Emisora de Radio Control, el control remoto del drone que utiliza el operario.
RPA	—	Remotely Piloted Aircraft o aeronave pilotada remotamente.
ROS	—	Robot Operating System. Sistema Operativo para Robots.
RPi	—	Raspberry Pi 3 modelo B.
UAV	—	Vehículos Aéreos No tripulados. Unmanned Aerial Vehicle.

Modos de Vuelo

$ALTHOLD$	—	El drone mantiene su altura guiado por estimación de altura relativa, permitiendo movilidad horizontal (roll y pitch).
$AUTO$	—	El drone es capaz de seguir una ruta predefinida cargada en el APM, sin intervención de un operador.
$GUIDED$	—	Modo para guiar dinámicamente el drone a una ubicación de destino utilizando telemetría y una aplicación de estación terrestre.
$LAND$	—	Modo de vuelo que indica al UAV aterrizar directamente en la posición actual (latitud, longitud).
$LOITER$	—	El modo Loiter automáticamente intenta mantener la posición, grados y altura. Requiere GPS.
RTL	—	Retorno a Zona de Lanzamiento, modo de vuelo guiado por GPS, del inglés; Return To Launch.

1. Introducción

1.1. Presentación del tema

Debido al rápido avance de las tecnologías UAV (Unmanned aerial vehicle), conocidos popularmente como "drones", y al incremento de su uso comercial, aparecen nuevas propuestas en el mercado para el uso de estas tecnologías específicas en diversas áreas, entre las que destacan fotografía profesional, cine, carreras de velocidad, estudio de suelos agroindustriales y forestales, trabajos de inspección, entre otros.
Para el manejo de vehículos aéreos no tripulados se requiere de una alta experiencia y habilidad, es por esto, que para disminuir la dificultad de operación y mejorar la experiencia de vuelo son requeridos sistemas que asistan al piloto durante el vuelo, obteniendo mejores resultados en las tareas que se realizarán.

En el área de fotografía profesional y amateur, probablemente el tipo de uso más común que se le da a los drones en la actualidad, los vehículos aéreos no tripulados presentan una ventaja frente a otras tecnologías, ya que, estos permiten un acercamiento a lugares de difícil acceso y proveen diversos tipos de tomas aéreas donde, por medios convencionales, se requiere de una importante inversión de recursos y personal especializado para lograrlo. Por otro lado, la inspección de suelos agroindustriales mediante drones que tienen la posibilidad de planificación de rutas de vuelo autónomas, el uso de sistemas FPV para pilotaje de drones de carreras, entre otras áreas, han aumentado el interés por desarrollar soluciones innovadoras utilizando este tipo de tecnología.

Según Martínez-Carranza *et al.*[1], el vuelo autónomo es una capacidad deseada en los drones por diversas razones. Discutiblemente, quizás la más importante es que el dron pueda tomar control de su propio vuelo en caso de que el piloto pierda comunicación con el vehículo, en cuyo caso sería deseable que el dron tuviera la capacidad de identificar dicho evento y por lo tanto volar de manera autónoma hacia algún punto de aterrizaje fuera de riesgo.

Dado esto, un UAV que permita no solo facilidad para acceder a lugares específicos, sino también una reacción semi-autónoma frente al entorno, detectando y evitando posibles obstáculos, enviando información pertinente al operador es altamente eficiente y recomendado. Actualmente la solución para lograr esto es mediante el uso de sensores para cálculo de posición basado en datos de GPS y barómetro, entre otros sensores logrando autonomía en modos de vuelo como RTL, loiter o alt hold. Esta solución responde bien en exteriores donde se tiene una buena recepción de señales satelitales y espacios libres de obstáculos.

Sin embargo esta tecnología es propensa a fallar en ambientes con obstáculos o en interiores donde no existe señal suficiente para que el sistema de GPS funcione, siendo esto el punto de partida para esta investigación.

En Chile, existen normativas que regulan la operación y registro de drones. La Dirección General de Aeronáutica Civil (DGAC) es el organismo que regula la actividad aérea en el país y, por ende, la operación de drones o RPAS, para cual desde abril del año 2015 cuenta con dos normativas aeronáuticas la DAN 151 y DAN 91[2]. Según datos oficiales de la DGAC, hasta abril de 2017 hay registrados 367 drones, una cifra no menor, considerando que la normativa esta vigente desde el año 2015.

1.2. Descripción del problema

Las amplias posibilidades de desarrollo que presentan los UAVs y el uso que se puede aplicar en diversos campos son en si un ventaja importante frente a otras tecnologías, pero el alto costo de drones aplicados a funcionalidades especificas provocan un disuasivo en el corto y mediano plazo para organizaciones pequeñas o individuos que necesitan una mejora en sus procesos comerciales e incluso solo para entretenimiento.

Por otro lado, el desarrollo privativo de soluciones de software/hardware por parte de las empresas pioneras en el campo de diseño y construcción de drones, provocan una creciente brecha de desarrollo competitivo. Es por esta razón que cada vez mas desarrolladores, makers entusiastas y usuarios se unen en comunidades como Dronecode Foundation[3] que fomenta el desarrollo de software de código abierto para la construcción de UAVs, apoyando a desarrolladores, proporcionándoles los recursos y herramientas para ayudarlos a innovar.

Precisamente esta es la idea fuerza que enmarca este proyecto, **plantear una solución de software/hardware libre de bajo costo para detección y evasión de obstáculos durante el vuelo de un UAV**. Permitiendo entre otras cosas la posibilidad de que la comunidad en general pueda hacer mejoras continuas y agregar nuevas funcionalidades al proyecto.

1.3. Estructura del documento

El desarrollo de esta memoria de título presenta una estructura de capítulos y anexos, los cuales son descritos brevemente a continuación.

1. **Capítulo 1. Introducción:** Contiene una breve presentación general del tema que se desarrolla, incluyendo una descripción de la problemática que se busca resolver.

2. **Capítulo 2. Definición del Proyecto:** Contiene una descripción clara y directa de los objetivos, tanto general como específicos, que se planifican alcanzar durante el desarrollo del proyecto. También se plantea la metodología de desarrollo utilizada, restricciones y alcance determinados.

3. **Capítulo 3. Estado del Arte:** Capítulo destinado a contextualizar el área de desarrollo del proyecto, presenta estudios similares y trabajo relacionado a los objetivos del proyecto.

4. **Capítulo 4. Tecnologías utilizadas:** Describe tanto las herramientas de software como hardware y diversas opciones disponibles para ser utilizadas en el proyecto. Por último se presentan conclusiones y observaciones sobre las distintas tecnologías implementadas.

5. **Capítulo 5. Marco Teórico:** Definición de conceptos teóricos necesarios para el correcto desarrollo de la memoria de titulo, a fin de comprender y estudiar la estructura y funcionamiento de un UAV y los procesos de detección y evasión de obstáculos.

6. **Capítulo 6. Desarrollo.**En este capitulo se desarrolla toda la experiencia de construcción y montaje de los componentes que conforman el **"Módulo detección/evasión de Obstáculos"**. Considerando algoritmos de detección/evasion de obstaculos y comunicación bi-direccional RPi↔APM.

7. **Capítulo 7. Implementación pruebas de funcionamiento** Capítulo dedicado a pruebas generales de los algoritmos para detección/evasión de obstáculos y comunicación RPi↔APM.

8. **Capítulo 8. Observaciones y Conclusiones** Capítulo que presenta las conclusiones, comentarios, observaciones y proyección de trabajo futuro de la memoria de título desarrollada.

2. Definición del proyecto

2.1. Objetivos

En este Capítulo, se exponen las metas a cumplir durante el desarrollo de la presente memoria de título, identificando así un objetivo claramente diferenciado como principal y cinco objetivos específicos necesarios para alcanzar el desarrollo de este.

2.1.1. Objetivo general

Desarrollar una solución de software/hardware que permita implementar algoritmos de detección de obstáculos mediante el uso de sensores a bordo y establecer métodos de evasión mediante instrucciones de control integrándolos en las plataformas de hardware de control de vuelo de un vehículo aéreo no tripulado.

2.1.2. Objetivos específicos

1. Establecer comunicación entre controlador de vuelo y dispositivo de procesamiento a bordo, mediante protocolo MAVLink.
2. Capturar datos de entorno mediante uso de sensores a bordo.
3. Implementar algoritmos de control de vuelo en lenguajes de alto nivel.
4. Desarrollar algoritmos de detección de obstáculos/entorno en lenguajes de alto nivel.
5. Ensamblar y Conectar componentes de hardware a bordo de vehículo aéreo no tripulado.

2.2. Alcance de la Investigación

El proyecto se enmarca de la siguiente forma:

- No se contempla el uso de algoritmos para replaneamiento de rutas predefinidas.
- Para el manejo y operación de un drone será necesario contar con un piloto con licencia valida emitida por la DGAC.
- Uso exclusivo de hardware y herramientas se software libre (open source) para el funcionamiento del proyecto.

El uso de tecnologías de hardware y software libre, definido como alcance para este proyecto estimula el desarrollo continuo y abierto a la comunidad y permite reducir costos de implementación y desarrollo para el cumplimiento de los objetivos.

Para el cumplimiento de cada uno los objetivos planteados es necesario el estudio previo de las distintas tecnologías disponibles a utilizar en el proyecto, pruebas de compatibilidad y uso en ambientes controlados.
Ante las pruebas de vuelo y control sobre el drone, que implican el armado de motores del drone, se considera necesario desmontar previamente las hélices para evitar cualquier tipo de accidentes durante el desarrollo y así reducir también el posible daño a los materias y equipos utilizados, también reduciendo al mínimo los vuelos de prueba en exteriores.

2.3. Metodología y plan de trabajo

Este proyecto presenta una serie de características que lo hacen difícil de definir dentro de una metodología de desarrollo de software tradicional, dado su naturaleza y la información disponible. Por tanto, considerando los objetivos definidos del proyecto, se propone una metodología de desarrollo de software iterativo incremental para diseño de sistemas embebidos.

Somerville expone: "Puesto que los sistemas embebidos son sistemas reactivos que reaccionan a los eventos en su entorno, el enfoque más general al diseño de software embebido de tiempo real se basa en un modelo estímulo-respuesta. [...] Los estímulos provienen de sensores en el entorno del sistema y las respuestas se envían a actuadores."[4, pag.540]

La figura 1 muestra el modelo general para procesos entre un sensor y un actuador frente a la dualidad estimulo/respuesta.

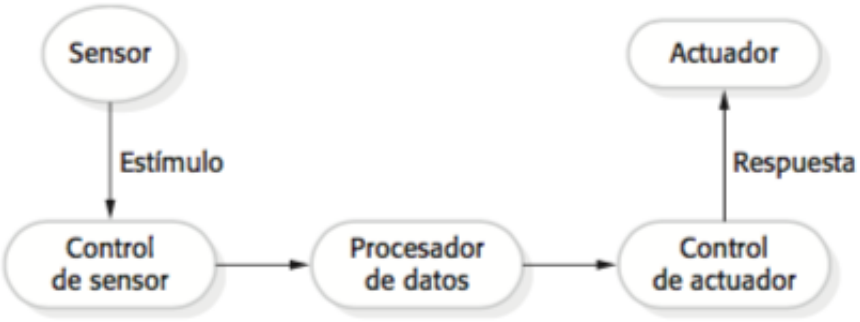


Figura 1: Procesos de sensor y actuador.[4, pag. 542]

La metodología de desarrollo incremental se basa en entregas segmentadas de funcionalidad permitiendo la prueba y evaluación de cada etapa de desarrollo. Considerando las características que propone este modelo sumado al diseño de sistemas embebidos, se considera que es el adecuado para desarrollar este proyecto.

Como Somerville afirma, no existe un proceso de diseño del sistema embebido estándar. En vez de ello, se usan diferentes procesos que dependen del tipo de sistema (es decir, el hardware y el sistema operativo de tiempo real a utilizar).[4, pag.542]
Considerando esto, el plan de trabajo que se propone esta dividido en 5 etapas, cada una centrada en cumplir cada uno de los objetivos específicos planteados anteriormente.

Finalmente, en cada etapa se estima cumplir con una entrega de hitos, basado en las siguientes tareas:

- Primera etapa: Conexión funcional de componentes del UAV (hardware).
- Segunda etapa: Comunicación mediante protocolo MAVLink entre dispositivos hardware.
- Tercera etapa: Desarrollo de algoritmos de control de vuelo.
- Cuarta etapa: Desarrollo de algoritmos de detección de obstáculos.
- Quinta etapa: Prueba de captura de datos y control de vuelo.

3. Estado del Arte

Para situar el contexto de este proyecto, se han consultado variadas fuentes de información, considerando temas de hardware para construcción de drones basado en Hardware/Software Open source, artículos de investigación enfocados en visión artificial monocular y estéreo, replanteamiento de rutas aplicadas en UAVs, protocolos de comunicación para vehículos aéreos no tripulados y documentación relacionada a productos comerciales que coincidan en alguna medida con el objetivo general de este proyecto.

3.1. Vehículo Aéreo no tripulado

Los vehículos aéreos no tripulados, son conocidos por variados nombres y siglas, siendo "drone" uno de los términos más utilizados en el ámbito civil actualmente. Pero para poder hablar de drones, es necesario, recurrir a una definición concisa y clara sobre que características debe poseer un drone para ser considerado como tal.

En el Diccionario de Términos Militares y Asociados del Departamento de Defensa de Estados Unidos[1] se define UAV como un vehículo aéreo motorizado que no transporta a un operador humano, utiliza fuerzas aerodinámicas para proporcionar elevación del vehículo, puede volar de forma autónoma o ser piloteado a distancia, puede ser prescindible o recuperable, y puede transportar una carga útil letal o no letal. Vehículos balísticos o semi balísticos, misiles crucero y proyectiles de artillería no son considerados vehículos aéreos no tripulados. También llamados UAV.
"Drone" y "UAV" son solo dos entre mas de una decena de términos que se han utilizado para describir este tipo de aeronaves desde su concepción. En Chile, la DGAC describe a los drones utilizando el término "RPAs" (Aeronave Pilotada Remotamente) y su uso esta normado según DAN 151 y DAN 91 [2].

Considerando la definición descrita anteriormente, es necesario clasificar los drones de acuerdo a diversos parámetros como tipo de ala y aplicación, entre otros. para agruparlos e identificarlos de forma clara y sencilla.

[1]Fuente Original, Inglés: https://fas.org/irp/doddir/dod/jp1_02-april2010.pdf

Dentro de los diversos tipos de clasificación general de un UAV, se puede considerar como los mas relevantes, los siguientes:

- **Funcionalidad:** De acuerdo a que área de funcionamiento están destinados; militar, fotografía, carreras, transporte, entre otros.
- **Tamaño:** Naturalmente una clasificación útil es según el tamaño del drone, pudiendo variar desde unos centímetros (como drones de juguete) hasta UAVs del tamaño de naves tripuladas (uso exclusivamente militar).
- **Tipo de ala:** de acuerdo a como es su estructura, tipo y funcionamiento de sus alas.

Considerando la clasificación por tipo de alas, se tiene:

- **Drones de Ala fija:** alas unidas/encastradas con el resto de elementos de la aeronave y no poseen movimiento propio.
- **Drones de Ala rotatoria:** alas denominadas "palas", giran alrededor de un eje, de esta forma consiguen sustentación.
- **Drones Híbridos:** Drones capaces de despegar/aterrizar de forma vertical (como aeronaves de ala rotatoria) y realizar vuelos de alta velocidad (como aeronaves de ala fija).

Dentro de la categoría de drones de ala rotatoria tenemos a los multirrotores, aeronaves que poseen tres o más motores y que se pueden subclasificar de acuerdo a su estructura. Dependiendo del número, y de la configuración espacial de sus motores, se pueden dividir en diferentes tipos, ver figura 2:

- **Tricopter:** drone con tres motores., configuración "Y".
- **QuadCopter:** drone con cuatro motores, configuración "X", "I"/"+" y "H".
- **Hexacopter:** drone con seis motores, configuración "Y", "IY", "V" e "I".
- **OctaCopter:** drone con ocho motores, mismas configuraciones espaciales que los Hexacopteros.

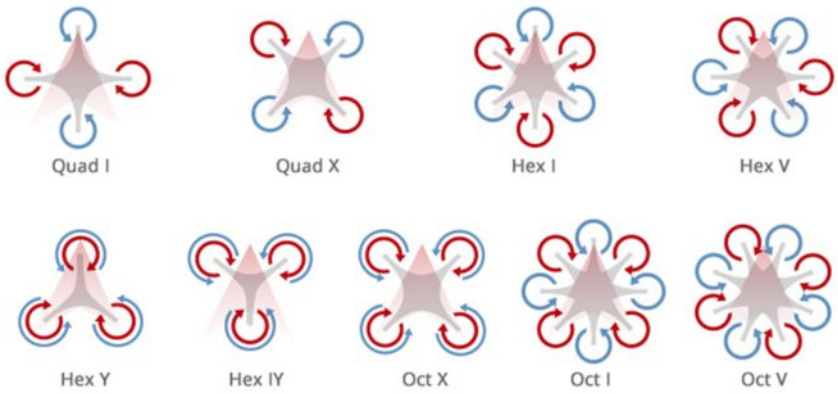


Figura 2: Tipos de UAV Multirrotor. [50].

3.2. Trabajo relacionado

Una de las tareas principales cuando se navega usando características autónomas/semi--autónomas con un dron es la detección y evasión de obstáculos. En la actualidad empresas como DJI[2] están a la vanguardia en temas de navegación asistida para sus drones comerciales, ejemplo de esto es el drone DJI Phantom 4 Pro que posee sensores de ultrasonido e infrarrojos que le permiten analizar el entorno y evitar obstáculos dinámicamente mientras cumple su misión de vuelo.

Investigaciones que hacen uso de sensores inerciales (giroscopios) y de flujo óptico a bordo de los UAVs han permitido determinar estimación de distancias y cálculo de altura, entre otros estados del drone con respecto a su entorno, como se puede encontrar en [5] y [6].

Por otro lado, Martínez-Carranza *et al.* [1] proponen una metodología para realizar navegación autónoma estimando la posición del UAV y obteniendo simultáneamente un mapeo de puntos en $\mathbb{R}^3$ que representan su entorno. Utilizando, como único sensor para capturar datos, una cámara a bordo del drone. De manera similar, Heng *et al.* [7], plantean un sistema de evasión de obstáculos para un MAV (micro air vehicle) capaz de realizar maniobras totalmente autónomas en entornos desconocidos y dinámicos, utilizando algoritmos que se ejecutan exclusivamente en la computadora de a bordo del MAV.

[2]www.dji.com

También, la detección de obstáculos mediante algoritmos de procesamiento de imagen utilizando librerías para procesamiento de video de OpenCV sobre datos de video en tiempo real, que permiten la detección y seguimiento objetos basados en su color aplicado en un quadcoptero ha sido ampliamente investigado y aplicado por Kadouf y Mustafah [8].

Meier *et al.*[9] plantean una investigación en el campo de la robótica MAV que se enfoca en permitir que esta clase de sistemas opere a altitudes más bajas cerca de obstáculos y humanos, con un controlador PIXHAWK sobre un cuadrotor[3] y visión artificial mediante cámaras estéreo y computador abordo.

3.3. Observaciones

Como resultado de la investigación y lectura de información relacionada, se puede observar que la mejora de sistemas UAV para vuelo en lugares con obstáculos ha sido ampliamente discutido, logrando soluciones comerciales de muy buen nivel, aunque a costos que pueden considerarse altos como en el caso de los drones de la Empresa DJI con el modelo Phantom 4 Pro (CLP$1.630.000, según web oficial [10]), equipados con sistemas capaces de reconocer el entorno, hacer seguimiento activo y múltiples modos de vuelo para diversos tipos de tomas audiovisuales aéreas. Por otro lado, el desarrollo para sistemas open source como PIXHAWK y APM, entre otros, no se queda atrás aunque el desarrollo se enfoca en el área experimental, permitiendo a la comunidad proponer diversas formas de obtener resultados con el objetivo de detectar obstáculos, esto observado en la notable cantidad de artículos de investigación relacionados al tema y la variedad de enfoques que se presentan.
Por tanto, el planteamiento de una solución para detección de obstáculos basada en hardware/software libre presente en este proyecto es una base para un desarrollo futuro de aplicaciones modulares para UAVs y otro tipo de vehículos no tripulados, ofreciendo un enfoque simplista al problema de la detección de obstáculos, con el uso de componentes y sensores de bajo costo.

[3]quadcoptero, drone de 4 motores.

4. Tecnologías utilizadas

4.1. Drones Multirrotor

Los multicopteros o multirrotores son un tipo de drone aerodinámicamente inestable, por lo tanto para volar requieren, aparte de la fuerza bruta de sus múltiples motores, un controlador a bordo para lograr estabilidad de vuelo y sustentación en el aire. Un controlador a bordo o controlador de vuelo es, en esencia, un microcontrolador cargado con un firmware especifico para un drone, permite el funcionamiento en conjunto de los diversos componentes que lo conforman (ESCs, radio Rx, sensores externos, servos y telemetría, entre otros.).

El controlador de vuelo combina datos de GPS, giroscopios y acelerómetros para mantener una estimación precisa de su posición y orientación. Gracias a los controladores de vuelo, algunos drones pueden ser capaces de tener rutinas de vuelo autónomo, estas rutinas se pueden cargar desde una estación en tierra, conectada mediante telemetría al drone, o ser cargadas con anterioridad en el controlador de vuelo mediante conexión cableada.

Para este trabajo se utiliza un hexacoptero personalizado (ver figura 3). Como su nombre lo indica, posee seis motores. Sus principales ventajas frente a un cuadcoptero (con características similares) son; su estabilidad en vuelo, precisión de manejo y capacidad de levantar mayor peso/carga. A continuación se exponen los componentes utilizados de forma detallada.

Figura 3: Drone - Hexacopter Tarot 680 Pro

4.1.1. Marco - Frame

Es el esqueleto del multirotor, la estructura que le da la forma y en donde todas las otras partes se instalan y aseguran. Existen diferentes diseños y materiales. Para este proyecto se utiliza el Frame TAROT Iron Man 680, Hexacopter.

Características:

- Diametro base circular: 680mm.
- Soporta hélices de 10" hasta 13".
- peso: $600gr$.
- 6 x $\ominus$ $16mm.$, tubos fibra de carbono 3K (333MM), diseño mate.

4.1.2. Controlador de Vuelo

El controlador de vuelo que se utiliza en el proyecto es un APM 2.6 (ArduPilot Mega, ver figura 4). El APM 2.6 es un sistema autopiloto completamente Open Source basado en el micro-controlador (MCU) Arduino Mega (ATMEGA A2560[11]), con las siguientes características:

- Compatible con Arduino y protocolo MAVLink.
- IMU[4] Acelerómetro/Giroscopio 6 DOF[5] MPU-6000 de 3 ejes.
- Sensor de presión barométrica modelo MS5611-01BA03.
- MCU Atmel ATMEGA2560 y ATMEGA32U-2.
- Interfaces para Telemetria 3DR Radio, GPS, I2C, UART[6], SPI y APM Power Module.

El controlador APM2.6 actualmente a sido actualizado a la unidad PIXHAWK, desarrollado por 3D Robotics, ambos utilizan el mismo firmware para operar y, por tanto, son compatibles con MAVLink, pero se opta por utilizar APM 2.6 dado la disponibilidad logística de este controlador a la fecha de realización de este proyecto.

[4]Inertial Measurement Unit.
[5]Degrees of freedom.
[6]Universal Asynchronous Receiver/Transmitter

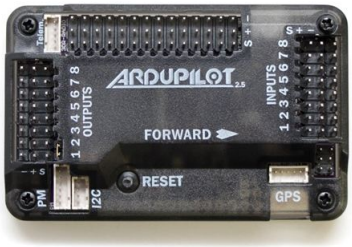


Figura 4: APM 2.6 - ArduPilot Mega [52].

4.1.3. Módulo GPS y brújula

El módulo GPS con compás que se utiliza en conjunto con el APM es el u-blox 6Neo GPS Modulo.
Este posee sistema de posicionamiento de 50 canales con Time-To-First-Fix[7] de menos de 1 segundo [12]. Compatible con Arduino, utiliza los puertos ??I2C" y "GPS port" del Controlador de vuelo APM 2.6. Normalmente el GPS y Brújula vienen integrados y se deben conectar lo mas lejos posible de los ESC y Motores para evitar interferencias electromagnéticas producidas por esos componentes.

4.1.4. Controladores Electrónicos de Velocidad - ESC

Un ESC es un dispositivo electrónico que sirve para controlar la velocidad de los motores tipo brushless (sin escobillas) del drone. Un ESC recibirá la señal PWM[8] del APM y dependiendo de la longitud del ancho de pulso, entregará mayor o menor potencia al motor.
Se utiliza un total de 6 ESCs (uno para cada motor) marca HobbyWing, Platinum-30A-OPTO-PRO [13].

Principales Características:

- Voltaje de entrada: 2-6S Baterias LiPo.
- Velocidad Máxima: 210,000rpm - 2 Polos BLM, 70,000rpm - 6 polos BLM, 35,000rpm - 12 polos BLM.
- Tasa de refresco de la señal de throttle: 50Hz to 432Hz.

[7](TTFF) Medida del tiempo requerido para que un receptor de GPS adquiera señales de satélite y datos de navegación, y calcule una solución de posición (llamada fix).
[8]Modulación por ancho de pulso

4.1.5. Motores

En el Hexacoptero se montan 6 motores brushless marca LOK LM4010SM-450KV. Se puede estimar las revoluciones por minuto (RPM) con carga (hélices) para cada motor según:

$$KV * S * \%NLS = RPM$$

Donde KV es el numero de RPM del motor por cada $Volt$ que recibe, S es la tensión nominal de la batería LiPo (en volts) y %NLS[9] es el porcentaje de velocidad sin carga para un motor.

Características:

- 6Kg torque total.
- soporta 2-6s baterías de LiPo.
- Motor diameter : 46mm.
- Stator diameter: 40mm.
- Stator length: 10mm.
- KV value: 450.

4.1.6. Bateria LiPo

Para energizar todos los componentes del drone, se utiliza una batería LiPo (Polímeros de Litio) recargable marca Wild Scorpion $30C$ $3S$ de 5500 mAh ($miliAmperios/hora$). Al ser una batería $3S$, posee 3 celdas con una tensión nominal total de $11.1V$.

4.1.7. Módulo de Poder APM

Todos los componentes del dron no funcionan con la misma tensión, y es por esto que se necesita un regulador de voltaje para proveer a cada componente de la tensión necesaria para que funcione. El APM power module proporciona de forma estable $5.37V$ - $2.25A$ y soporta baterías de hasta $4S$ ($18V$) y corriente máxima de $90A$ [14].

[9]Se considera que no debe ser menor a 70 %.

4.1.8. Circuito Eliminador de Bateria - BEC

Por otro lado también existe un componente que se puede usar en paralelo al APM power module, y que en este proyecto se usa para obtener $5V$ de forma estable, para alimentar componentes extras en el drone. Básicamente, un BEC es un regulador de electricidad, en este caso, un UBEC Henge 6A 5-6V [15].

Características:

- Salida: $5/6V$, selección con jumper.
- Entrada: baterias LiPo de 2-6S
- Corriente continua de salida: $6A$

4.1.9. Emisora y Receptor de radio control

Para controlar al drone es necesario tener una emisora de radio control y su par receptor, en este caso, una Walkera Devention Tx Devo-7 con un receptor Devention Rx701 ambos de 7 canales [16].

4.1.10. Hélices - propellers

Por último, es necesario tener las hélices adecuadas para el tipo de vuelo, considerando tamaño, material, forma, pitch[10], entre otros. Para este proyecto las helices seleccionadas son: hélices ABS bipalas 13x4.5 3CW/3CCW[11] genéricas.

[10] distancia que recorre la hélice al dar una vuelta completa.
[11] CW: ClockWise. - CCW: Counter-ClockWise.

4.2. Computador a bordo

Para volar y realizar tareas simples como estimar posición, altura, inclinación, entre otros parametros, un UAV necesita de un controlador de vuelo, pero en términos de capacidad de computo, estos no son capaces de realizar tareas que requieran mayor poder de procesamiento, como por ejemplo realizar tareas de reconocimiento de imagen, donde es requerido gran capacidad de almacenamiento y computo.
El APM 2.6 posee un microcontrolador ATMEGA2560 y 4MB para datalogging[11], por tanto, es necesario la utilización de un computador a bordo que, en permanente comunicación con el APM, permita enviar y recibir información de este, procesar datos y ejecutar algoritmos para detección de obstáculos mediante uso de sensores. Para este fin se utiliza la placa de desarrollo Open Source Raspberry Pi 3 Modelo B.

Características:

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU.
- 1GB RAM.
- BCM43438 WLAN y Bluetooth Low Energy (BLE) integrados.
- 40-pin extended GPIO[12]
- 4 Puertos USB.
- Puerto tarjeta SD para almacenamiento y carga de sistema operativo.

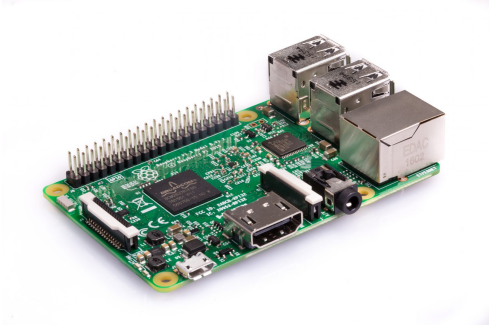


Figura 5: Raspberry Pi 3 Modelo B [54].

[12]General Purpose Input/Output. Su numeración, distribución de pines BROADCOM y tipo de uso se puede ver en anexo A, figura 53.

4.3. Sensores a bordo

Para lograr que un drone pueda cumplir con distintos modos de vuelo (Acro, Loiter, RTL, entre otros) es necesario que el controlador de vuelo obtenga diversos datos de estado, como altitud relativa, georreferencia, inclinación, estado de batería, entre otros. Para ello, el controlador cuenta con sensores internos, como IMU y barómetro, y sensores externos, como GPS y brújula. Para obtener datos del entorno es requerido otro tipo de sensores a bordo del drone, para este proyecto se consideran principalmente tres tipos:

- Sensores de proximidad por ultrasonido.
- Sensores de proximidad por ráfaga de pulsos infrarrojos.
- Sensores de captura de imagen.

4.3.1. Sensores de proximidad por Ultrasonido

Para obtener medidas de distancia se utiliza el sensor de proximidad por ultrasonido HC-SR04 (ver imagen 6), este provee una medición sin contacto con rango de $2cm$ - $400cm$, con una precisión de $3mm$.[17]. Mayormente es usado en proyectos de robótica móvil y etapas de prototipado de proyectos comerciales, dado su buen comportamiento en ambientes controlados. Sus pines ordenados de izquierda a derecha, vista frontal son: V_{cc}, Trigger, Echo y GND. Su funcionamiento se basa en la emisión/recepción de señales de sonido en alta frecuencia.

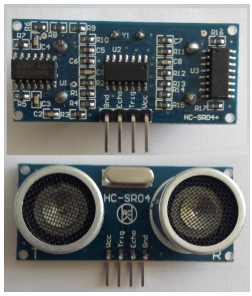


Figura 6: Sensor proximidad por ultrasonido HC-SR04.

4.3.2. Sensor de proximidad por infrarrojos

Otro tipo de sensor que permite medir proximidad a una superficie (obstáculos) son los sensores de radiación infrarroja. El sensor para robótica IR Sharp GP2Y0A02YK0F (ver imagen 7), tiene una taza de refresco de unos 40ms, una salida analógica que varía entre $2.8V$ a $15cms$ y $0.4V$ a $150cms$ cuando se alimenta con voltajes entre 4.5 y $5.5VDC$. El principio de funcionamiento de los sensores de proximidad por infrarrojos se basa en un sistema de emisión/recepción de radiación lumínica en el espectro de los infrarrojos.

Figura 7: Sensor IR Sharp.[56]

4.3.3. Sensores de Imagen

Un sensor que presenta variados usos es una cámara (para fotografías y/o video), en este caso puede ser una cámara con conexión USB o una RPi Camera (ver imagen 8), una cámara creada específicamente para funcionar con una RPi. El uso de una cámara web USB o una RPi Camera sirve para obtener imágenes que pueden ser procesadas por RPi, obteniendo datos del entorno mediante diversos métodos de procesamiento de imágenes en tiempo real.

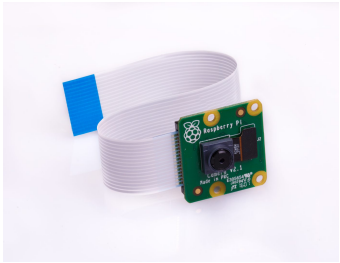


Figura 8: Sensor Imagen RPi Camera.[57]

4.4. Libreria GPIOzero para Raspberry Pi

GPIOzero es una librería open source desarrollada en python por integrantes de Raspberry Pi Foundation, que permite utilizar los GPIO de una RPi de forma sencilla con una multiple gama de dispositivos de entrada/salida como sensores, convertidores analógico a digital, LEDs, kits de robótica y más.[18]

4.5. Protocolo de comunicación MAVLink

MavLink es un protocolo de comunicación para MAVs, de ahí su nombre. Permite comunicación bidireccional entre un controlador de vuelo (APM2.6) y un GCS como el software Mission Planner, también para comunicación entre un AMP2.6 y cualquier otro dispositivo que pueda implementar el protocolo MAVLink a través de puertos serial, en nuestro caso, un Raspberry Pi.
Se puede extraer de la documentación oficial en [19], que cada archivo XML define mensajes que son soportados por un sistema MAVLink en particular, referido como "dialecto". Un conjunto de mensajes de referencia común para la mayoría de los GCS, sistemas Ardupilot y a información completa del protocolo esta disponible a través de la plataforma GitHub[13].
Se pueden empaquetar estructuras en lenguaje C (Implementación de referencia de MAVLink) y otros lenguajes como Java o Python sobre canales seriales con alta eficiencia y enviar estos paquetes al GCS. MavLink ha probado exhaustivamente en las plataformas PX4, PIXHAWK, APM y Parrot AR.Drone. Y sirve allí como red troncal de comunicación para la comunicación MCU/IMU[14], así como para comunicación interprocesos de Linux y el enlace de tierra.

4.6. PyMAVLink Tools

La implementación en Python para trabajar con el protocolo MAVLink es la libreria PyMAVLink, provee una serie de herramientas gráficas y de línea de comandos para la medición y análisis de datos en tiempo real y fuera de linea. Admite stream logs de MAVLink, así como los formatos de registro autodescriptivo de PX4 y APM [20].

El protocolo MAVLink se distribuye con algunos paquetes de mensajes generales, pero es posible generar mensajes personalizados e incluirlos al proyecto. Para el proceso de creación de un mensaje MAVLink se necesita tener un archivo XML con el mensaje estructurado (ver figura 9) y gracias a una de las herramientas en PyMavLink Tools, "mavgen_c.py"[15], es posible obtener una estructura en C (ver imagen 10) listo para agregarlo a las cabeceras de un proyecto.

Los mensajes del protocolo MAVLink estructurado en un archivo XML, contienen:

[13]https://github.com/mavlink
[14]Microcontroller/Inertial Measurement Unit.
[15]Implementación en python para generar cabeceras C de paquetes MAVLink.

```xml
<message id="0" name="HEARTBEAT">
    <description>The heartbeat message shows that a system is present and responding. The type of the MAV and Autopilot hardware allow the receiving system to treat further messages from this system appropriate (e.g. by laying out the user interface based on the autopilot).</description>
    <field type="uint8_t" name="type" enum="MAV_TYPE">Type of the MAV (quadrotor, helicopter, etc., up to 15 types, defined in MAV_TYPE ENUM)</field>
    <field type="uint8_t" name="autopilot" enum="MAV_AUTOPILOT">Autopilot type / class. defined in MAV_AUTOPILOT ENUM</field>
    <field type="uint8_t" name="base_mode" enum="MAV_MODE_FLAG" display="bitmask">System mode bitfield, as defined by MAV_MODE_FLAG enum</field>
    <field type="uint32_t" name="custom_mode">A bitfield for use for autopilot-specific flags</field>
    <field type="uint8_t" name="system_status" enum="MAV_STATE">System status flag, as defined by MAV_STATE enum</field>
    <field type="uint8_t_mavlink_version" name="mavlink_version">MAVLink version, not writable by user, gets added by protocol because of magic data type: uint8_t_mavlink_version</field>
</message>
```

Figura 9: Ejemplo definición mensaje MAVLink - XML

- Identificación del mensaje, *"id"* único.
- Encapsulamiento del mensaje con etiquetas $< message >< /message >$
- Un nombre que identifica el mensaje dentro del protocolo, *"name"*
- Funcionalidad del mensaje, o descripción, dentro de las etiquetas $< description >< /description >$
- Las etiquetas $< field >< /field >$ encapsula los parámetros, tipo de dato y descripción de cada campo del mensaje.
- Los parametros *"type"* y *"name"* indica el tipo de dato, descritos anteriormente, y su nombre.

```c
#define MAVLINK_MSG_ID_HEARBEAT 0

tyederf struct_Mmavlink_hearbeat_t
{
    unit32_t custom_mode; ///< Navigation mode bitfield, This field is autopilot-specific
    unit8_t type; ///< Type of the MAV (quadrotor, helicopter, etc., up to 15 types, defined in MAV_TYPE ENUM)
    unit8_t autopilot; ///< Autopilot type / class. defined in MAV_AUTOPILOT ENUM
    unit8_t base_mode;///< System mode bitfield, as defined by MAV_MODE_FLAG enum
    unit8_t system_status;///< A bitfield for use for autopilot-specific flags
    unit8_t mavlink_version;///< System status flag, as defined by MAV_STATE enum
    unit8_t mavlink_heartbeat_t;
}
```

Figura 10: Ejemplo resultado estructura MAVLink; mavgenerate.py - C

4.7. MAVProxy

MAVProxy es una estación de control de tierra con funcionalidad completa para UAVs. La intención es tener una Estación de Control de Tierra (GCS) minimalista, portable y extensible para cualquier UAV que soporte el protocolo MAVLink.[21, MAVProxy]
Es una aplicación de linea de comandos, por tanto, es útil para ejecutarla en el entorno Raspbian Stretch Lite que se utiliza en la RPi de este proyecto.

4.8. DroneKit – 3DR

DroneKit es un API open source desarrollado por 3DR (3D Robotics) que permite a los desarrolladores crear aplicaciones que se ejecutan en una computadora de a bordo y comunicarse con el controlador de vuelo ArduPilot utilizando un enlace de baja laténcia [22], mediante protocolo MAVLink. El desarrollo de aplicaciones mediante esta API facilita agregar mayor inteligencia al comportamiento del vehículo y realizar tareas que son intensivas o sensibles al tiempo, como visión artificial, modelado 3D o planificación de rutas, entre otros.

DroneKIt es compatible con equipos desarrollados por 3DR y por los miembros de Dronecode Foundation.

Características del API DroneKit:

- conexión a vehículos (UAV, Rover, Plane, entre otros) mediante scripts.
- recibir y configurar estados/telemetría e información de parámetros del vehículo.
- sobrescribir canales de radio control.

4.9. Herramientas para Desarrollo

4.9.1. Herramientas de Hardware

Para hacer posible el desarrollo de este proyecto, se utiliza el computador personal del alumno tesista, tanto para generar documentación como para la implementación de la lógica del proyecto, control y configuración por ssh de RPi y configuración de firmware APM 2.6. Las características técnicas se pueden observar en la tabla 2.

Características	
Sistema Operativo	MacOS High Sierra v.10.13.5 / GNU Linux CentOS 8
Procesador	2,5 GHz Intel Core i5
Memoria RAM	16 GB 1600 MHz DDR3
Gráficos	Intel HD Graphics 4000
Almacenamiento	250GB SSD

Tabla 2: Computador de Desarrollo - Características técnicas.

4.9.2. Herramientas de software

Dentro de las herramientas de software utilizadas para el desarrollo de la presente memoria de título destacan:

- **SublimeText / Atom:** Dos poderosos editores de texto, con diversos grados de personalización.
- **Nano / Vi / Vim:** Editores de texto para consola/terminal, utilizados principalmente en entornos sin interfaz gráfica.
- **Distribución MacTex:** Paquete que contiene editor, compilador y vizualizador PDF para trabajar con LaTeX sobre un entorno macOS.
- **LaTeX:** sistema de composición tipográfica de alta calidad; incluye características diseñadas para la producción de documentación técnica y científica[23].
- **APM Planner:** Software para estación en tierra compatible con APM2.6.
- **Python 2.7/3.6:** Lenguaje de programación poderoso, de alto nivel, interpretado, con tipado dinámico y que presenta gran cantidad de librerías útiles al proyecto.
- **Otros Lenguajes de programación**: En menor medida se hace uso de otros lenguajes/metalenguajes que incluyen C/C++ (adaptaciones de avr-libc, para micros Atmel) y XML.

4.10. Observaciones generales

Para este proyecto se utilizan tecnologías de hardware/software libre, como el proyecto ArduPilot para el controlador de vuelo, el protocolo y las herramientas desarrolladas por 3DR, contenido de la plataforma Dronecode en [3], entre otros, gracias a la importante comunidad detrás de estos proyectos permitiendo constantemente añadir mejoras y nuevos desarrollos a diversos productos.
Por otro lado, el uso de sensores planteados principalmente para robótica móvil permite una construcción modular y prototipado de bajo costo para desarrollo experimental en el drone y que luego se puede extrapolar a desarrollos en comerciales de mayor presupuesto.

Los sensores a bordo aportan una mejora importante a las capacidades de un drone, cada uno dota al drone de distintas capacidades. En este trabajo se presentan tres tipos enfocados en un mismo objetivo; detección de obstáculos. Es importante encontrar métodos de comparación para definir que tipos de sensores se utilizan. La figura 3 muestra una comparación de los tres tipos de sensores mencionados en este proyecto.

Tipo	Sensor IR	Sensor de Ultrasonido	Sensor de imagen (Pi Camera)
Objetos Detectables	Detección afectada por materiales/colores del objeto	Detección no afectada por materiales/colores del objeto	Definido por software
Distancia de Detección	20cm - 150cm	2cm - 400cm	No aplica
Precisión	Alta	Alta - 3mm	Alta - resolución 8mpx imagen / 1080p30fps video
Conexión Directa a RPi	NO - Requiere Convertidor Analógico-Digital	NO - Requiere Logic-Level Shifter 5v to 3.3v	SI
Precio CLP	$15.900.- (www.mcielectronics.cl/)	$3.490.- (www.mcielectronics.cl/)	$29.990.- (www.mcielectronics.cl/)

Tabla 3: Tabla comparativa sensores a bordo.

4.11. Conclusiones

En conclusión, la amplia gama de posibilidades que permiten estas tecnologías para desarrollo experimental y prototipado de productos, en este caso un UAV con sistema de sensores de entorno, la compatibilidad para trabajar sobre el hardware/software utilizando lenguajes de alto nivel e implementación de tecnologías de la información son las principales razones para avalar su uso en este proyecto.

Por tanto, en adición al drone utilizado, se opta por usar una RPi modelo 3B, como computador a bordo dado su sistema operativo Linux Raspbian con soporte oficial de la fundación Raspberry Pi Foundation, tamaño reducido ($850mm\ x\ 530mm$), chipset Wifi y Bluetooth integrados, capacidad de ejecutar paquetes ROS[24] y compatibilidad con el protocolo MAVlink mediante sus puertos USB y GPIO. Siendo esta versión de RPi la más potente disponible en el marcado (Chile) durante el desarrollo de este proyecto, a pesar de que la versión RPi 3B+ ya se presentó por parte de la fundación RaspBerry, ver figura 56 en anexo B .

Finalmente basado en las comparaciones de los tres tipos de sensores de entorno descritos, se selecciona como sensor a bordo para detección de obstáculos el sensor de proximidad por ultrasonido, debido a su bajo costo, alta precisión y mayor rango de detección en comparación a los otros sensores.

5. Marco Teórico

En esta sección se definen los conceptos teóricos relevantes con el área de estudio para este proyecto. Tales como, comunicación serial y protocolos de comunicación, funcionamiento de sensores, esquemas generales del drone utilizado en el proyecto, entre otros.

Al igual que la distinción sobre componentes de hardware básicos necesarios para que un multirrotor funcione, los principios básicos de sustentación en el aire y los fenómenos físicos que lo explican también deben tomarse en consideración.

5.1. Aerodinámica de un Drone

Como se ha mencionado antes, un drone es aerodinámicamente inestable y para comprender como se sustenta en el aire, se deben considerar algunos principios de la aerodinámica, aplicables a cualquier objeto moviéndose a través del aire, que explican como objetos pesados puedan sustentarse en el aire. Para el estudio del vuelo de drones, aviones, helicópteros y otros, es lo mismo considerar que es el objeto el que se mueve a través del aire, como que este objeto esté inmóvil y es el aire (fluido) el que se mueve.

5.1.1. Principio de Bernoulli

Considerando velocidades subsónicas, Bernoulli comprobó experimentalmente que cuando en un fluido ideal se produce una corriente estacionaria horizontal, a lo largo de la línea de corriente, la velocidad del fluido se relaciona con su presión, de acuerdo a la siguiente ecuación:

$$p + dgh + 1/2dv^2 = k;$$

$$1/2dv^2 = pd;$$

Donde p es la presión en un punto dado, d es la densidad del fluido, v es la velocidad en dicho punto, g aceleración de gravedad, h la altura respecto a un nivel de referencia común a todos los puntos del fluido y pd es lo que se denomina presión dinámica.[25]

Enfocando este teorema desde otro punto de vista, se puede afirmar que en un fluido en movimiento la suma de la presión estática (pe) más la presión dinámica (pt) denominada presión total , es constante:

$$pd = pe + v;$$

de donde se infiere que si la velocidad del fluido se incrementa, la presión estática disminuye. Se considera el efecto a bajas velocidades (subsónicas), dado que un UAV, como el utilizado en este proyecto, no supera la velocidad del sónido.

5.1.2. Efecto Venturi

Otro efecto aerodinámico importante a considerar lo comprobó experimentalmente Venturi como un caso particular del principio de Bernoulli, indicando que al pasar por un estrechamiento, las partículas de un fluido (en este caso, el aire) aumentan su velocidad y disminuye su presión de forma proporcional [26].

5.1.3. Tercera Ley de Newton

La tercera Ley de Newton establece: Siempre que un objeto ejerce una fuerza sobre un segundo objeto, el segundo objeto ejerce una fuerza de igual magnitud y dirección opuesta sobre el primero. Con frecuencia se enuncia como "A cada acción siempre se opone una reacción igual" [27].

$$F_{12} = -F_{21}$$

5.1.4. Perfil aerodinámico

El concepto de perfil aerodinámico implica un cuerpo que tiene un diseño determinado para aprovechar al máximo las fuerzas que se originan por la variación de velocidad y presión, cuando este perfil se sitúe en una corriente de aire. Un Drone no posee alas con un perfil aerodinámico, pero si se puede ver este tipo de estructura en las hélices montadas sobre sus múltiples motores, ver Figura 11.

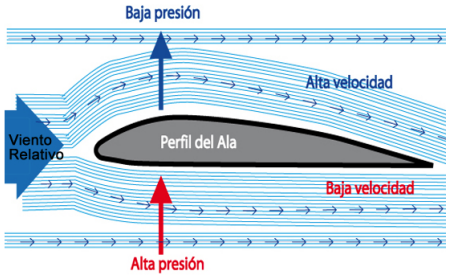


Figura 11: Presión vs Velocidad - Perfil aerodinámico.

En términos de perfil aerodinámico, el aire que fluye por arriba del perfil tendrá una velocidad mayor (efecto Venturi), dado el ángulo o curvatura del perfil (a mayor ángulo del perfil, mayor estrechamiento en la parte superior de este) y el aire que fluye por abajo del perfil (helices de un UAV) tendrá una velocidad menor, implicando que a mayor velocidad del flujo de aire en la parte superior, menor presión (teorema de Bernoulli). Todo este flujo de aire, considerando sus variaciones de velocidad y presión forman en su totalidad la acción, por lo que se genera una reacción idéntica y en sentido opuesto (tercera ley de Newton) que se denomina sustentación.

5.1.5. Observaciones

Si la posición de los motores en un UAV es simétrica y los motores generan la misma fuerza, entonces la sumatoria de fuerzas será únicamente vertical. De esto se puede entender la importancia de controlar el centro de gravedad, la velocidad de los motores, el tipo de helices que utiliza, entre otros componentes que permiten al dron sustentarse en el aire.
Los multirotores tienen la particularidad que sus motores no giran todos en el mismo sentido, sino que giran en sentido opuesto alternadamente, evitando que el UAV gire de forma descontrolada, esto se aprecia en la imagen 12, donde los motores en verde (1, 3, 6) giran en sentido horario y los motores en azul (2, 4, 5) giran en sentido antihorario.

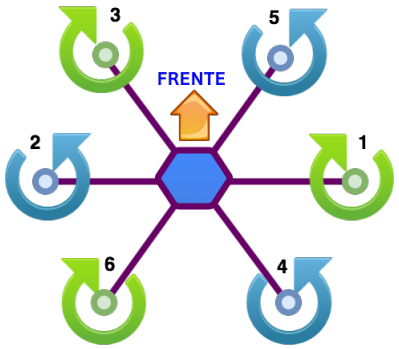


Figura 12: Sentido de giro motores hexacoptero - Hexa tipo X.

5.2. Grados de Libertad

En [28] se describe que el número de variables de coordenadas independientes necesarias para determinar simultáneamente la posición de cada partícula en un sistema dinámico se denomina número de grados de libertad de ese sistema.
En un conjunto N de partículas interrelacionadas moviéndose en un espacio d-dimensional con un número k de ligaduras entre las partículas, el número de grados de libertad del sistema será:

$$GL = 2d * N - k \leq 2d * N$$

Para este proyecto se tiene un solido rígido, UAV hexacoptero. De esto, se puede decir decir que un UAV como partícula libre sin ligaduras, tiene 6 grados de libertad:

$$GL = 6N - k = 6, N = 1, k = 0$$

Los seis grados de libertad hacen referencia a 3 ejes de traslación combinados con 3 ángulos de rotación, el movimiento de traslación es creado por el cambio de dirección en ejes perpendiculares (x, y, z). Para el movimiento de rotación es necesario inclinar el vector de empuje, esto se lleva a cabo cambiando la velocidad de rotación de los motores individualmente.

5.2.1. Ángulos de Navegación

Una forma de ángulos Eulerianos que describen traslación y rotación de objetos en espacios tridimensionales, son los ángulos de navegación. Utilizados frecuentemente en aeronáutica para determinar la posición de una aeronave en un momento dado respecto a un sistema fijo de coordenadas, ver figura 13.

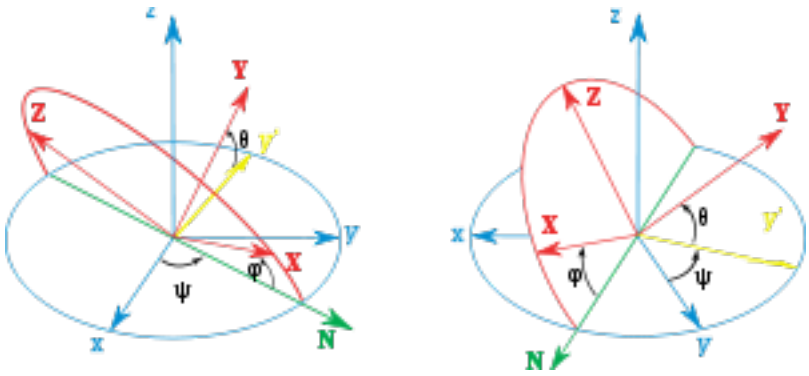


Figura 13: Ángulos de navegación, convención ZXY [58].

Los tres ángulos de navegación, con mismo nombre que el eje sobre el que realiza la rotación son; dirección o alabeo (yaw), elevación o cabeceo (pitch) y ángulo de guiñada (roll). Ademas, hay que considerar una cuarta variable que permite al drone sustentarse en el aire, al proveer de la misma potencia a todos los motores; throttle o acelerador de potencia, variando la altitud y velocidad del UAV, ver figura 14.

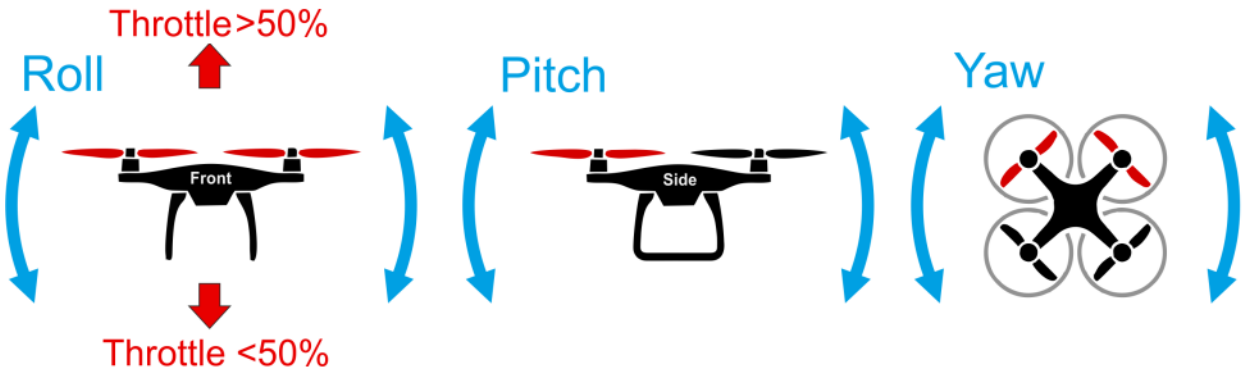


Figura 14: Roll, Pitch, Yaw en UAV [59].

Descripción de los ángulos de navegación:

- **yaw** - (guiñada), permite al drone rotar sobre el eje z, perpendicular al controlador de vuelo en el centro del frame, mediante la diferencia de velocidad de los motores diametralmente opuestos (estos motores tienen el mismo sentido de giro).
- **pitch**.- (cabeceo), permite al drone inclinarse hacia adelante o atrás para avanzar o retroceder mediante la diferencia de velocidad entre los motores delanteros y traseros.
- **roll** - (alabeo), permite al drone inclinarse haca izquierda o derecha para movimiento lateral, mediante diferencia de velocidad entre motores laterales.

5.3. Estructura General del Drone

Los componentes de un UAV pueden variar de acuerdo a las necesidades de uso, aunque se mantiene una estructura básica general para todos los multirrotores, y a partir de ahí, agregar componentes mas específicos como cámaras, gimbal, sistemas FPV, entre otros. Para este proyecto se denominó **"Módulo Control de Obstáculos"** al conjunto de elementos que en conforman una solución a la problemática presentada.

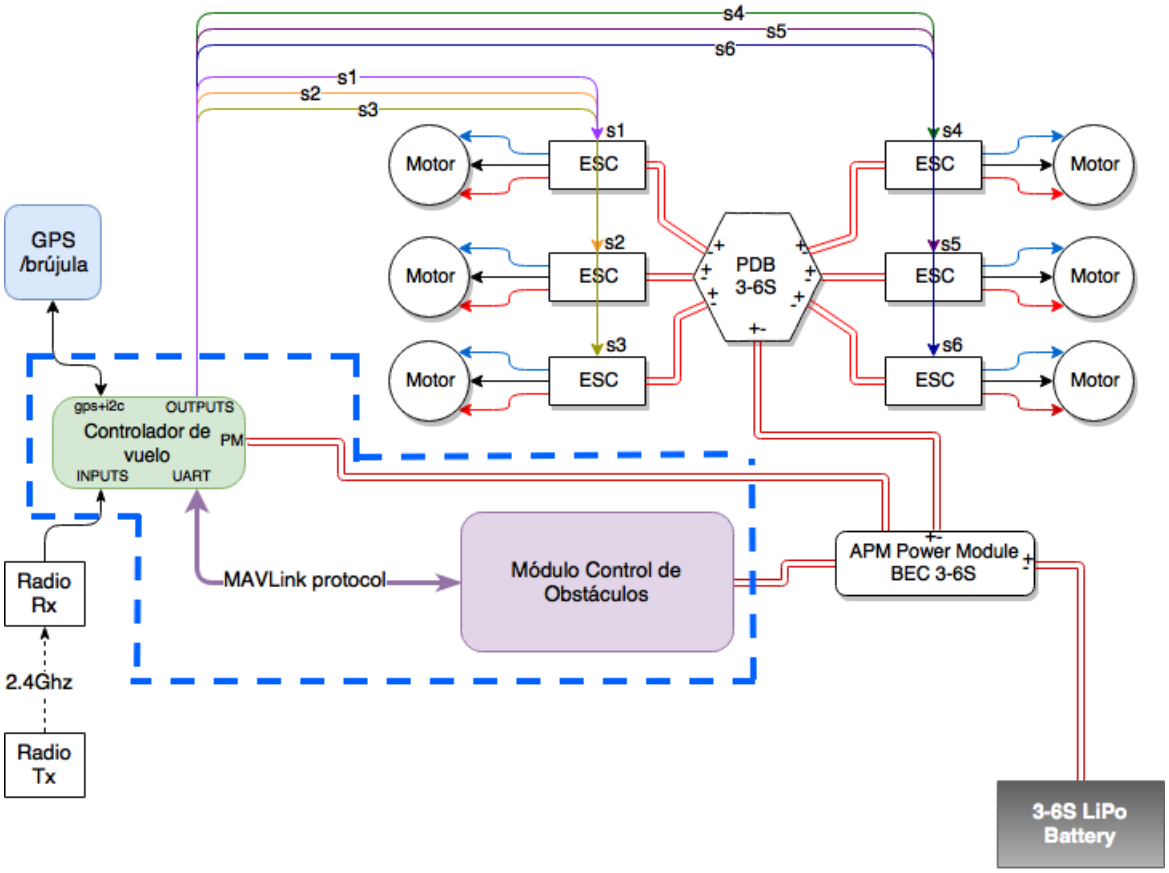


Figura 15: Diagrama General Drone - Hexacoptero.

A partir de este punto se propuso el montaje de los siguientes componentes; RPi como computador a bordo y sensores necesarios para detección de obstáculos (sensor de ultrasónido), encapsulando el desarrollo dentro de la zona delimitada por cuadro segmentado, como enfoque principal para desarrollo, ver figura 15.

5.4. Comunicación Serial

La comunicación serial es un protocolo ampliamente utilizado para establecer comunicación entre dispositivos electrónicos y que se incluye de forma estándar en prácticamente todos los equipos computacionales. El concepto de este tipo de comunicación es que el puerto serial envía y recibe bytes de información un bit a la vez.

De este tipo de comunicación, las características mas importantes que se requieren sean iguales en ambos extremos del canal de comunicación son:

- **Velocidad de transmisión:** Es número de veces por segundo que una señal (bits) cambia de estado, se mide en baudios.
- **Bits de Datos:** Se refiere a la cantidad de bits en la transmisión. El tamaño de un paquete no necesariamente será de 8 bits. Para un paquete ASCII extendido, el rango es de 0 a 255, lo que utiliza 8 bits. Un paquete se refiere a una transferencia de byte, incluyendo los bits de inicio/parada, bits de datos, y paridad. Debido a que el número actual de bits depende en el protocolo que se seleccione, el término paquete se usa para referirse a todos los casos como se ve en [30].
- **Bits de parada:** Usado para indicar el fin de la comunicación de un solo paquete.
- **Paridad:** Es una forma sencilla de verificar si hay errores en la transmisión serial.

En [30] se expone que típicamente, la comunicación serial se utiliza para transmitir datos en formato ASCII. Para realizar la comunicación se utilizan 3 líneas de transmisión: (1) Tierra (o referencia), (2) Transmitir, (3) Recibir. Debido a que la transmisión es asincrónica, es posible enviar datos por un línea mientras se reciben datos por otra. Existen otras líneas disponibles para realizar handshaking, o intercambio de pulsos de sincronización, pero no son requeridas.

5.5. UARTs en Raspberry Pi

Divya *et al.*[31] describen un UART como un dispositivo de hardware que se utiliza para la comunicación de datos en serie, este traduce entre bits de datos paralelos y bits de serie. UART suele ser un SoC[16] embebido en otros sistemas, controlando los puertos serie. Para reducir gastos de enlaces se comunicación que transportan varios bits en paralelo, un UART lo reduce a envíos en secuencia. En la figura 16 se ve un esquema de funcionamiento interno de un UART.

Los últimos modelos Raspberry Pi tienen dos UART incorporados, un UART PL011; SoC desarrollado por ARM, datasheet en [32], y un mini UART.

[16]System-on-Chip

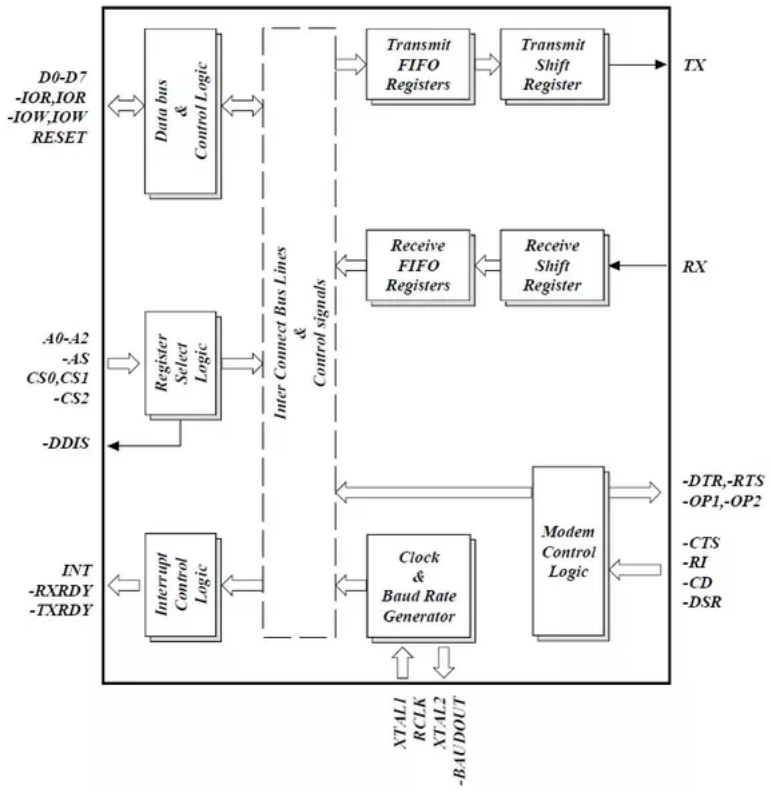


Figura 16: Estructura interna UART [31].

Los Raspberry Pi 3 (b/b+) y Raspberry Pi ZeroW están equipados con un módulo Wifi/-Bluetooth, por tanto el UART PL011 está conectado a este, mientras que el mini UART se utiliza para la salida de la consola Linux. En todos los demás modelos de RPi, el UART PL011 se usa para la salida de la consola Linux. El baudrate del mini UART está vinculado directamente a la frecuencia de funcionamiento del VPU en la GPU VC4 de la RPi, esto produce que el baudrate del mini UART no sea lo suficientemente estable debido a su dependencia de la carga sobre la GPU, limitando su uso. Además, por software se puede activar o desactivar el uso del mini UART y como consecuencia de que el mini UART esté desactivado, la consola también se desactiva.

En términos de Linux, por defecto */dev/ttyS0* hace referencia al mini Uart y */dev/tty/AMA0* hace referencia al UART PL011. La consola de Linux puede ser utilizada via serial a través de */dev/serial0*.

5.5.1. Principales Diferencias entre PL011 y mini UART

Se deben considerar las diferencias entre los UARTs que posee el RPi para determinar el uso que se puede dar, las principales diferencias entre el mini UART PL011 y el UART PL011 es que el primero tiene una cola FIFO mas pequeña y menor control de flujo provocando una mayor perdida de datos a altas tasas de velocidad (baudrate), en general, es menor potente que el UART PL011 principalmente producto de su dependencia de la carga sobre la GPU, variando su baudrate.

5.6. Protocolo MAVLink

Para comprender el funcionamiento del protocolo MAVLink es necesario manejar el concepto de paquetes MAVLink, fundamental para el concepto de encapsulamiento de un mensaje para comunicación.

5.6.1. Paquetes MAVLink

MAVLink package es básicamente una secuencia de bytes codificados y enviados a través de un transductor (a través de USB, radio frecuencia, WiFi, GPRS, etc.). Mediante la codificación se ordena la información en un estructura de datos de manera inteligente añadiendo checksums (suma de control) y número de secuencia que luego se envía a través del canal.[21, MAVLink]

En [33] se expone que los paquetes MAVLink deben seguir una estructura especifica. Existen dos lineas de desarrollo del protocolo; v1.0 y v2.0, que es retrocompatible (la implementación de la versión v.2.0 puede parsear[17] y enviar paquetes v1.0).
Por tanto, referente a las versiones de MAVLink; la versión v2.0 (ver en la figura 18) contiene las cabeceras de la versión v1.0 (ver figura 17).

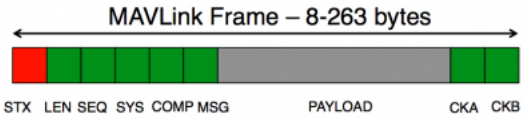


Figura 17: Estructura paquete MAVLink v1.0 [55].

[17] Analizar gramaticalmente.

Gracias a esta característica, se puede implementar la comunicación bidireccional entre una RPi enviando paquetes MAVLink v2.0 al APM compatible con MAVLink v1.0, y proporciona la libertad de trabajar con ambas versiones del protocolo. Se debe considerar que al enviar los paquetes al APM mediante USB, este tiene prioridad sobre el UART para telemetría, por tanto, ambas conexiones no se pueden usar a la vez de forma predeterminada.

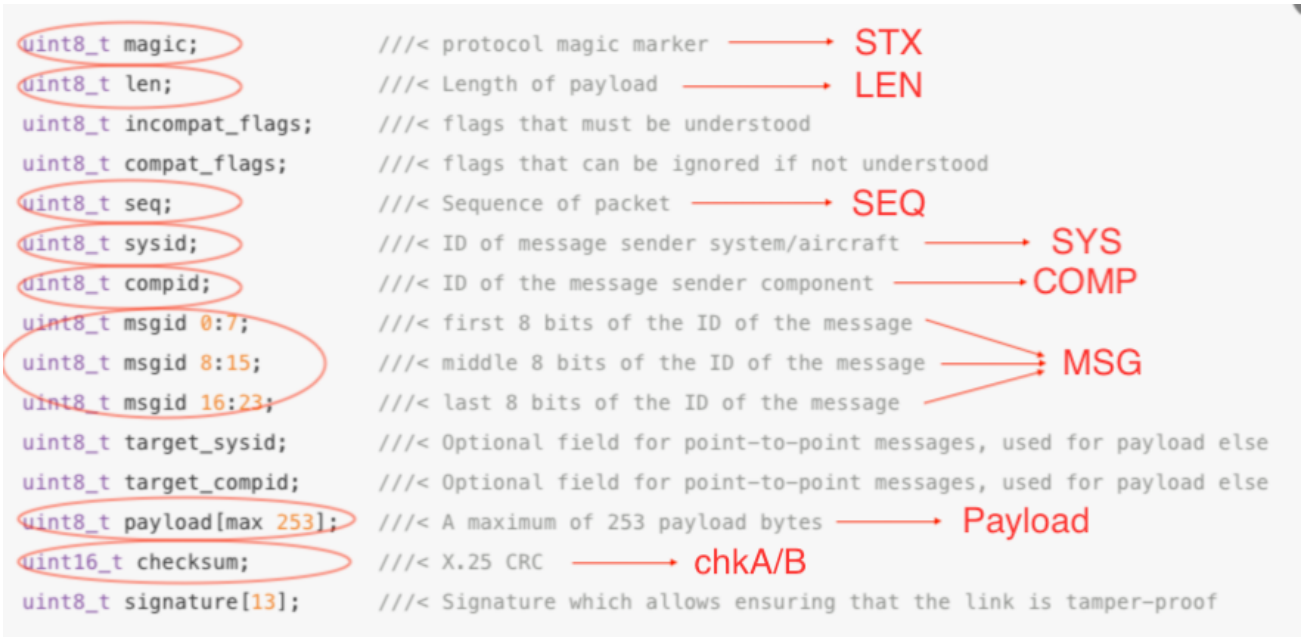


Figura 18: Estructura paquete MAVLink v2.0

En términos generales, el 'mensaje' es un paquete de datos que contiene un número especifico de bytes. El APM obtiene bytes de transmisión y decodifíca el mensaje enviando una respuesta, en caso de solicitud de información o ejecutando acciones, en caso de envío de instrucciones.

Se puede observar en la tabla 4 una especificación completa de cada parte de un paquete del protocolo MAVLink v1.0

Byte Index	Contenido	Valor	Explicación
0	inicio del mensaje	v1.0: 0xFE (v0.9: 0x55)	Indica el inicio de un nuevo paquete.
1	Largo del Payload	0 - 255	Indica longitud del contenido.
2	nº. de secuencia	0 - 255	Cada paquete cuenta con número de secuencia, permite determinar perdida de paquetes.
3	ID del sistema	1 - 255	ID del sistema EMISOR mensaje. ID del sistema de envío. Permite diferenciar diferentes MAVs en la misma red..
4	ID del Componente	0 - 255	ID del componente EMISOR. Permite diferenciar diferentes componentes en el mismo sistema, i.e. IMU, GPS, entre otros.
5	ID del Mensaje	0 - 255	ID del Mensaje - el ID define que "significa.[el] contenido del payload y como debe ser correctamente decodificado.
6 to (n+6)	Datos	(0 - 255)	bytes y datos del mensaje, depende del ID del mensajes.
(n+7) to (n+8)	Checksum (low byte, high byte)		ITU X.25/SAE AS-4 hash, excluido byte de inicio, desde el byte 1..(n+6) Nota: El checksum también incluye MAVLINK_CRC_EXTRA (Número calculado a partir de campos del mensaje. Protege el paquete de ser decodificado de una versión diferente del mismo paquete pero con diferentes variables).

Tabla 4: Especificación Estructura Paquete MAVLink.

- El largo mínimo de un paquete sin payload es 8 bytes.
- La longitud máxima es de 263 bytes con el máximo contenido.
- La suma de control (checksum) es la misma que utilizan los estandares ITU[18] X.25 (ver [34]) y SAE[19] AS-4 (CRC-16-CCITT), documentados en SAE AS5669A, ver [35].

[18]Unión Internacional de Telecomunicaciones.
[19]Aerospace Standard.

MAVLink soporta distintos tipos de datos; *enteros de tamaño fijo*, *IEEE 754 flotantes de precisión simple*, *arrays*, el campo especial *mavlink_version* que es añadido automáticamente al protocolo, entre otros.

Tipos de dato que soporta el protocolo:

- **char** - Characters / Strings.
- **uint8_t** - Unsigned 8 bit.
- **int8_t** - Signed 8 bit.
- **uint16_t** - Unsigned 16 bit.
- **int16_t** - Signed 16 bit.
- **uint32_t** - Unsigned 32 bit.
- **int32_t** - Signed 32 bit.
- **uint64_t** - Unsigned 64 bit.
- **int64_t** - Signed 64 bit.
- **float** - IEEE 754 single precision floating point number.
- **double** - IEEE 754 double precision floating point number.
- **uint8_t_mavlink_version** - Unsigned 8 bit. El campo se rellena automáticamente con la versión actual de MAVLink - no puede ser escrito, solo se puede leer del paquete como un tipo de dato **uint8_t**.

La velocidad de transmisión de datos puede ser ajustada de acuerdo al medio de comunicación entre dispositivos. Por defecto, si la conexión es mediante serial USB, la velocidad de transmisión es de 115.200 baudios[20]. Por otro lado, si la conexión es inalámbrica (telemetría 3DR, por ejemplo), la velocidad de transmisión de datos debe ser ajustada a 57.600 baudios.

[20]número de veces por segundo que una señal de comunicaciones serie cambia de estado - un baudio equivale a un bit por segundo (bps).

5.7. DroneKit-Python

El componente DroneKit-Python y su soporte de cómputo a bordo a través de plataformas como RPi es útil para comunicarse directamente con el controlador de vuelo, proporcionar planificación de rutas, vuelo autónomo e interfaces de telemetría en tiempo real. DroneKit-Python utiliza Pymavlink, una implementación en Python de MAVLink.
DroneKit-Python corresponde a la capa API, formando un bloque con la capa de interfaz de usuario en una estructura funcional de 6 capas para comunicación y control de dispositivos Ardupilot (Rover, Drone, Plane, entre otros), ver imagen 19.

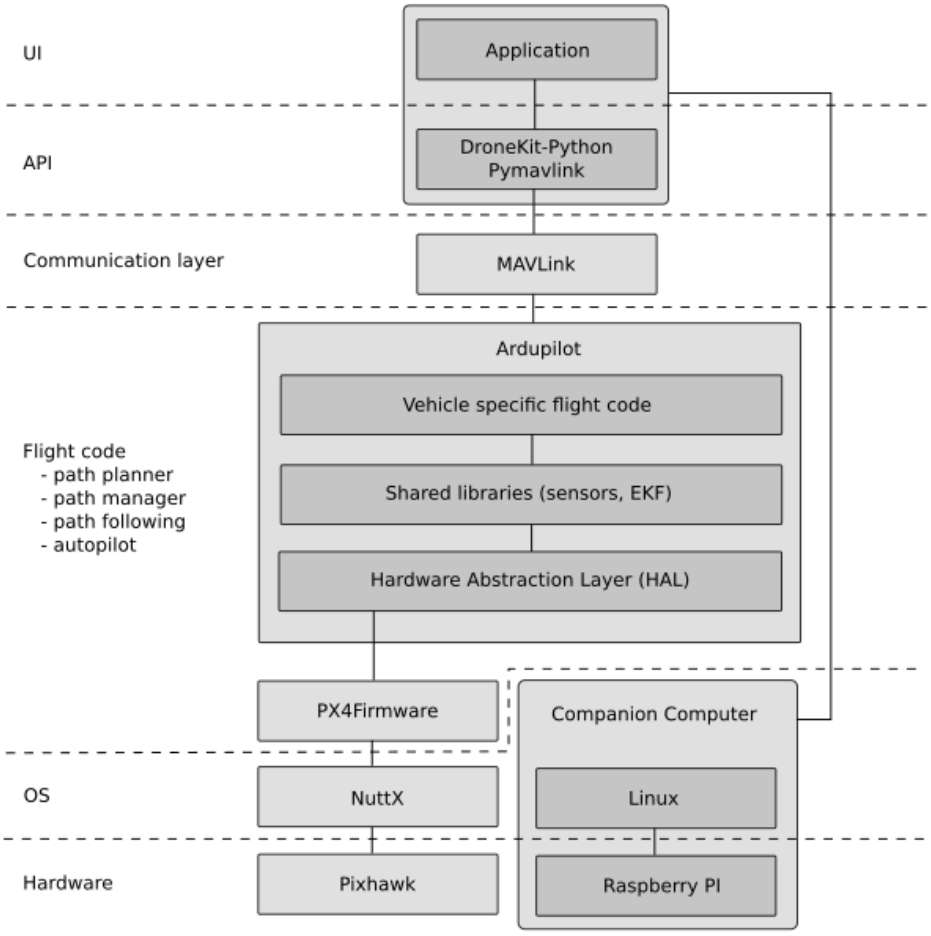


Figura 19: Estructura 6 Capas control Ardupilot, [36].

5.7.1. DroneKit-Python API

Según [36], el funcionamiento del API es muy simple y directo. Su principal responsabilidad es manejar los mensajes de MAVLink y obtener los cambios de estado en el vehículo.

El nivel principal de abstracción es la clase **Vehicle**, contiene atributos generales a todos los vehículos Ardupilot, proporcionando la mayoría de la información de estado del vehículo. Las configuraciones del vehículo están disponibles a través de *Parametros*. Todos los atributos pueden leerse, y además algunos son editables, tales como, *"Vehicle.mode"*, *"Vehicle.armed"* y *"Vehicle.home_location"*.
Para reaccionar o controlar los cambios de estado del vehículo se utilizan elementos denominados "getters y setters", esto permite observar cualquiera de los atributos del objeto vehículo, ya que lo parámetros están en caché, por lo que las funciones "callback" o retrollamadas solo se invocan cuando los valores de los parámetros varían, como se puede encontrar en la documentación [22].

También se plantea que la forma de controlar y reaccionar a los cambios de estado es a través del denominado **"patrón de observadores"** en el que un objeto, llamado sujeto (el vehículo en este caso), mantiene una lista de sus dependientes, llamados observadores, y los notifica automáticamente sobre cualquier cambio de estado, generalmente llamando a uno de sus métodos, ver imagen 20.

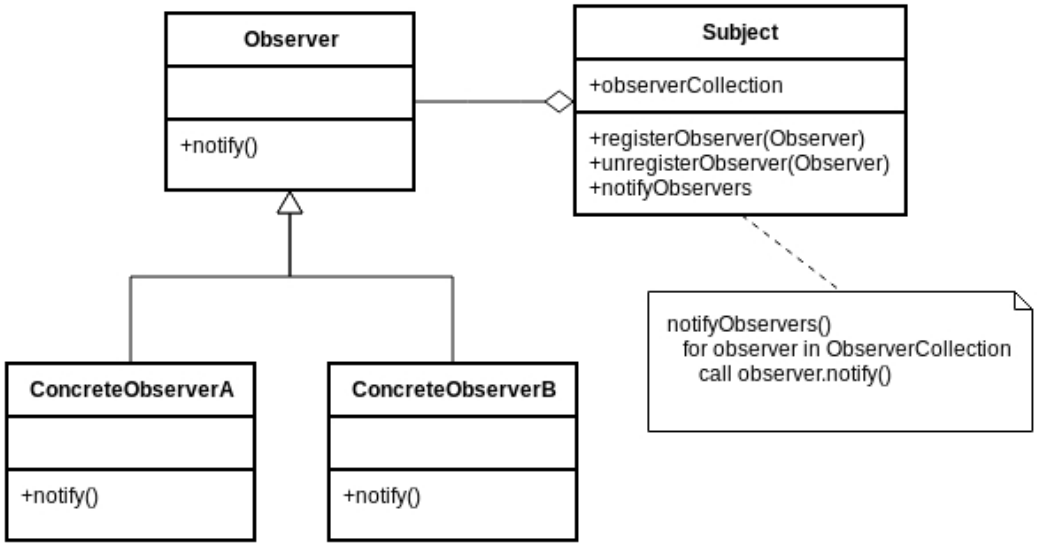


Figura 20: Class Diagram of Observer Pattern, [36].

5.8. Detección de obstáculos

La detección de obstáculos es primordial, dado que si se logra detectar un obstáculo, es posible reaccionar frente a este.
La información capturada por los sensores es procesada en el computador a bordo y se ejecutan las respuestas correspondientes. Se pueden considerar dos sistemas activos para medir proximidad o distancia sin contacto físico, los ópticos y los ultrasónicos. Los principios físicos de funcionamiento pueden ser fotoeléctricos, electromagnéticos, acústicos, capacitívos, entre otros.
La longitud de onda utilizada por sensores ópticos es del orden de la radiación infrarroja[21] y presentan menos Los sistemas ópticos ofrecen mejor precisión y como temperatura y presión afectan a que la longitud de onda involucrada es más corta y a que presentan menor sensibilidad a condiciones ambiéntales como por ejemplo la presión y la temperatura.

Los sensores de proximidad pueden basarse en diferentes principios físicos como es el caso de los sensores fotoeléctricos, electromagnéticos, acústicos y capacitivos, entre otros.

5.8.1. Medición de proximidad por ultrasonidos

El sensor de proximidad por ultrasonido utilizado en robótica móvil, HC-SR04, requiere para funcionar un pulso TTL[22] en su pin "TRIGGER" durante $10\mu S$ permitiendo que el módulo envíe un tren de pulsos de ultrasonido (8 ciclos, señal de disparo) a $40kHz$ y espera por su señal de eco (al rebotar con una superficie), como se aprecia la figura 21. De esto se puede calcular la distancia a través del intervalo de tiempo que hay entre el envío de la señal de disparo (trigger signal) y la recepción de la señal de eco (como salida en su pin "ECHO"), según la ecuación:

$$distancia = (t/2) * v$$

Donde t es el tiempo entre la señal de envío y su señal de eco (en segundos), y v es la velocidad del sonido ($343{,}2m/s$ a 20^oC). A este método de medición de distancia se le llama "Medición por Tiempo de Vuelo", ver figura 22.

Los sensores pueden verse afectados por el fenómeno cross-talking: la onda de un sensor es recibida por otro sensor o un mismo sensor puede recibir su propia onda de un disparo previo si los tiempos de espera entre disparo y disparo no son adecuados, y la disposición física de los sensores producen superposición de sus ángulos de medición (30^o en el sensor HC-SR04).

[21] La radiación infrarroja (IR) tiene longitudes de ondas entre 1 milímetro y 750 nanometros.
[22] "transistor-transistor logic"- tensión de alimentación característica: $5V$.

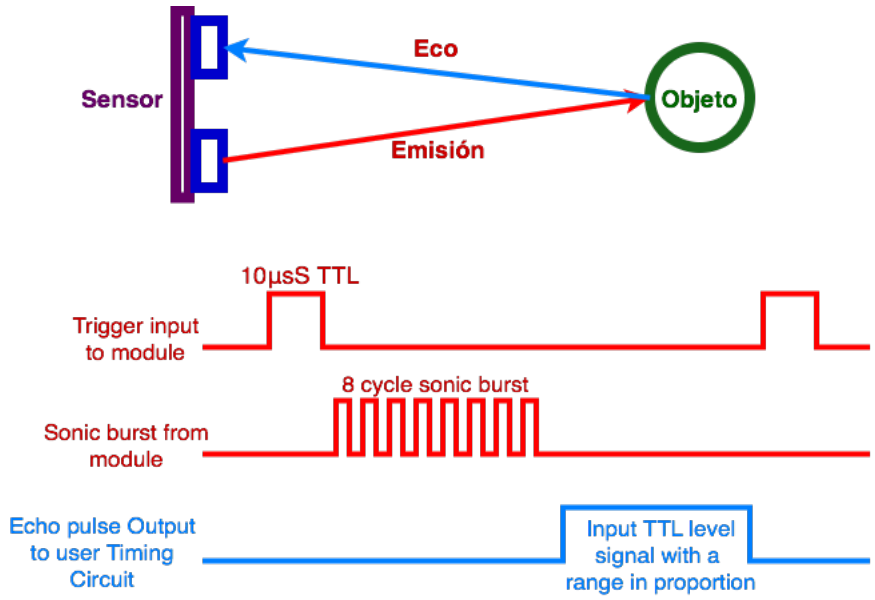


Figura 21: Esquema funcionamiento sensor HC-SR04.

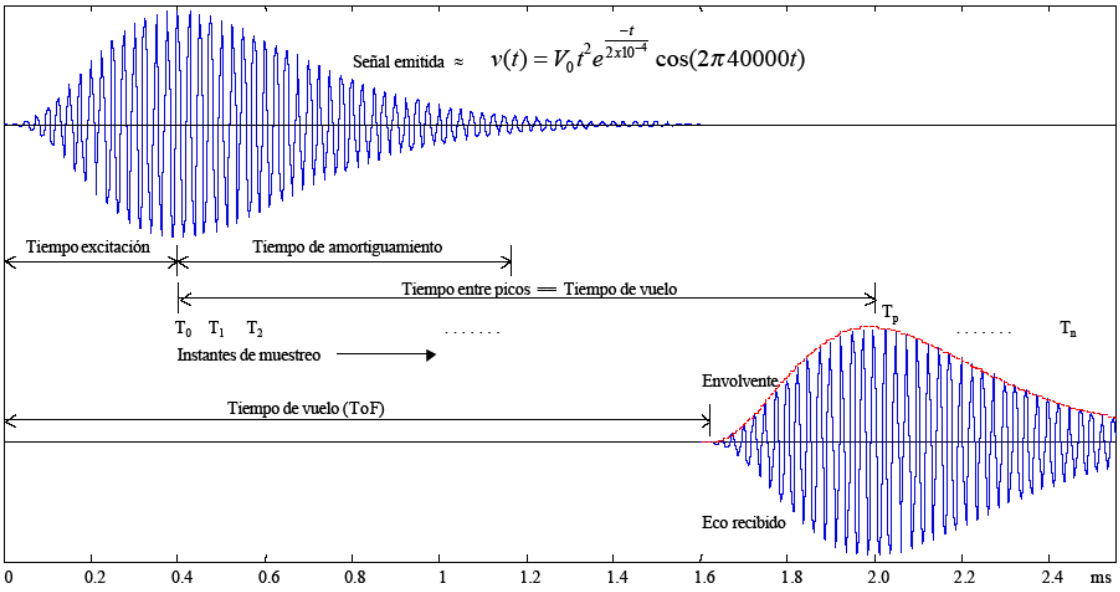


Figura 22: Medición de distancia por Tiempo de Vuelo, [60].

5.8.2. Medición de proximidad por infrarrojos

La forma más usada para medir proximidad utilizando un sensor de infrarrojos es mediante la triangulación de luz que rebota sobre una superficie. Un haz de luz emitido desde el módulo incide sobre la superficie y es reflejado en un ángulo distinto en función de la distancia a la que esta del sensor (ver figura 23).

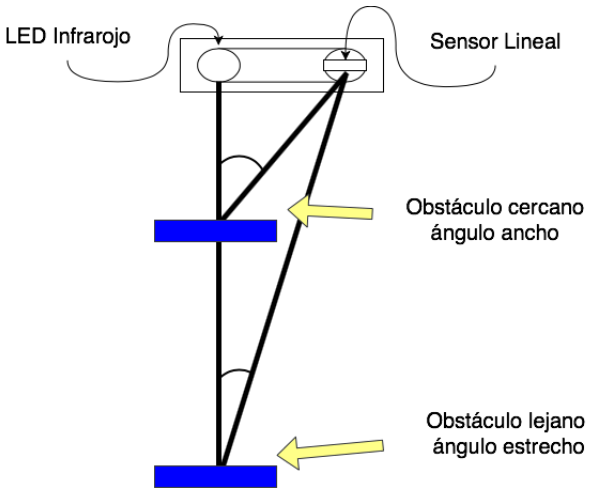


Figura 23: Esquema funcionamiento sensor infrarrojo.

Los sensores IR dedicados a robótica móvil, se dividen principalmente en dos tipos; los que integran circuitos integrados para comparación de tension y respuesta digital, y los que solo entregan valores de tension de forma análoga. El primer tipo enfocado en detección de obstáculos bajo un umbral de distancia, el segundo enfocado en medición de distancia a un objeto.
En el algoritmo 1, se puede observar la lógica para medir distancia con un sensor del tipo infrarrojo/óptico con salida analógica.

Algoritmo 1 Pseudocogido medición de distancia con sensor Infrarrojo

```
1: function DISTANCE(nSamples)
2:     sensorPin ← analog Input
3:     x ← analogRead(sensorPin, nSamples)      ▷ Invoke analogous input reading
4:     l ←AnalogToDistance(x)                   ▷ Invoke conversion Algorithm
5:     Return l
```

5.8.3. Visión artificial

Un sistema de visión artificial aplicado a robótica tiene como objetivo determinar la posición y orientación de un robot móvil (aplicable al caso de un UAV), así como la ubicación y dimensión de los obstáculos.
Para lograr este objetivo es necesario un sensor que permita capturar imágenes a tazas de refresco altas (cámaras y sensores de flujo óptico) y la aplicación de variadas técnicas aplicadas a visión aritficial, tales como redes neuronales profundas, reconocimiento y seguimiento de patrones, entre otros. Siendo estas técnicas ampliamente desarrolladas en el proyecto *Deep Drone* en [37], donde se plantea un marco de desarrollo embebido para proveer a drones de vision artificial.

En el algoritmo 2 se muestra el proceso del ciclo principal de detección y seguimiento planteado en el proyecto *Deep Drone* aplicando técnicas de visión artificial.

Algoritmo 2 Pseudocogido detección y seguimiento proyecto Deep Drone, [37].

```
1:  boxFound ← false
2:  while true do
3:      f ← new frame
4:      while boxFound == false do
5:          detection(f)                          ▷ Invoke detection algorithm
6:          if Box is detected then
7:              boxFound ← true
8:      tracking(f)                               ▷ Invoke tracking algorithm
9:      if Tracking is lost then
10:         boxFound ← false
```

6. Desarrollo

En este capitulo se expone toda la experiencia de construcción y montaje de los componentes que conforman el **"Módulo Control de Obstáculos"**, sus diagramas de conexionado y desarrollo e implementación de los algoritmos de detección y evasión de obstáculos que conforman el sistema embebido para asistencia de navegación integrada al UAV.

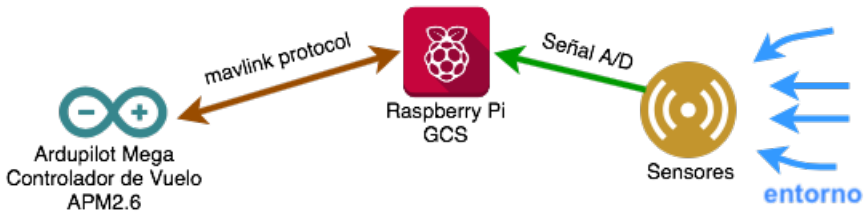


Figura 24: Esquema detalle módulo control de obstáculos integrado.

6.1. Conexión y montaje RPi ↔ APM2.6

El computador a bordo, Rapsberry Pi 3b, es el encargado de recibir la señal obtenida por los sensores, procesarla, obtener resultados y mediante comunicación bidireccional, enviar instrucciones de navegación al controlador de vuelo y obteniendo de este los parámetros de estado interno del drone. Por tanto, por un lado la RPi tomará el rol de GCS para el APM y por otro lado, como centro de procesamiento para señales de sensores externos al drone, ver figura 24.

6.1.1. Montaje de hardware

Para lograr comunicación entre el computador a bordo (RPi) y el controlador de vuelo (APM2.6) se utiliza una conexión serial cableada entre ambos equipos con MAVLink como protocolo de comunicación.

Se utiliza el puerto para telemetría del APM2.6; pines $+5V$, Tx, Rx, vacío y GND. Y los GPIO del RPi; pines VCC, GND, UART0-GPIO-14(T_x) y UART0-GPIO15(R_x) (ver figura 54 en anexo A).
En la figura 25, se indica el circuito esquemático del conexionado cruzado para $T_x \leftrightarrow R_x$[23] entre los dispositivos, logrando así una comunicación bidireccional que permite enviar y recibir mensajes bajo el protocolo MAVlink.

[23]Transmisión/Recepción de información

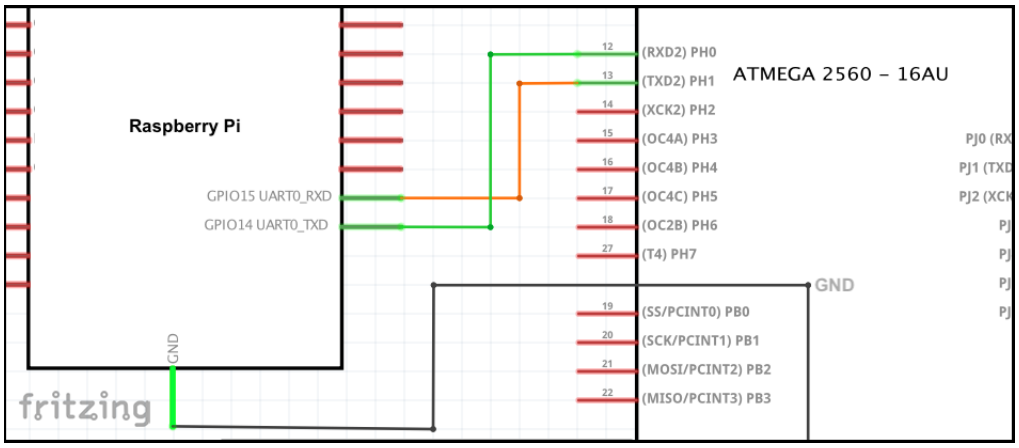


Figura 25: Circuito esquemático conexión serial RPi↔APM.

6.1.2. Configuración de software

Para que efectivamente exista comunicación entre RPi y el APM, es necesario configurar una serie de pre requisitos de software, tanto en el Raspberry Pi como en el Ardupilot:

- **Raspberry Pi**
  - UART - Habilitar uso de UART0/UART1, desactivar Bluetooth.
  - Serial Port - Deshabilitar inicio de sesión por puerto serial.
  - X11 tunneling - Activar la opción túnel X11.
  - Python Packages - Instalar paquetes/librerías necesarias en python para uso de DroneKit y MAVProxy.
- **Ardupilot Mega**
  - Baudrate - Ajustar velocidad de transmisión de datos para comunicación serial con RPi.

El único puerto serie disponible en un RPi esta en los GPIO $14T_x$ y GPIO $15R_x$ pero por defecto está configurado para ser usado por la consola de linux, por tanto es necesario desactivarlo para esta función, también se desactiva el Bluetooth para ajustar el UART0 como UART para comunicación serial en pines GPIO $14T_x$ y GPIO $15R_x$ dado que usa el SoC PL011.

En la imagen 26 se puede observar como a través de la aplicación *raspi-config* disponible en Raspbian, se desactiva el uso de serial para consola Linux y se habilita el uso por hardware. Se edita el archivo *config.txt* en Raspbian para deshabilitar Bluetooth y comprobar que UART0 esta habilitado, ver imagen 27.

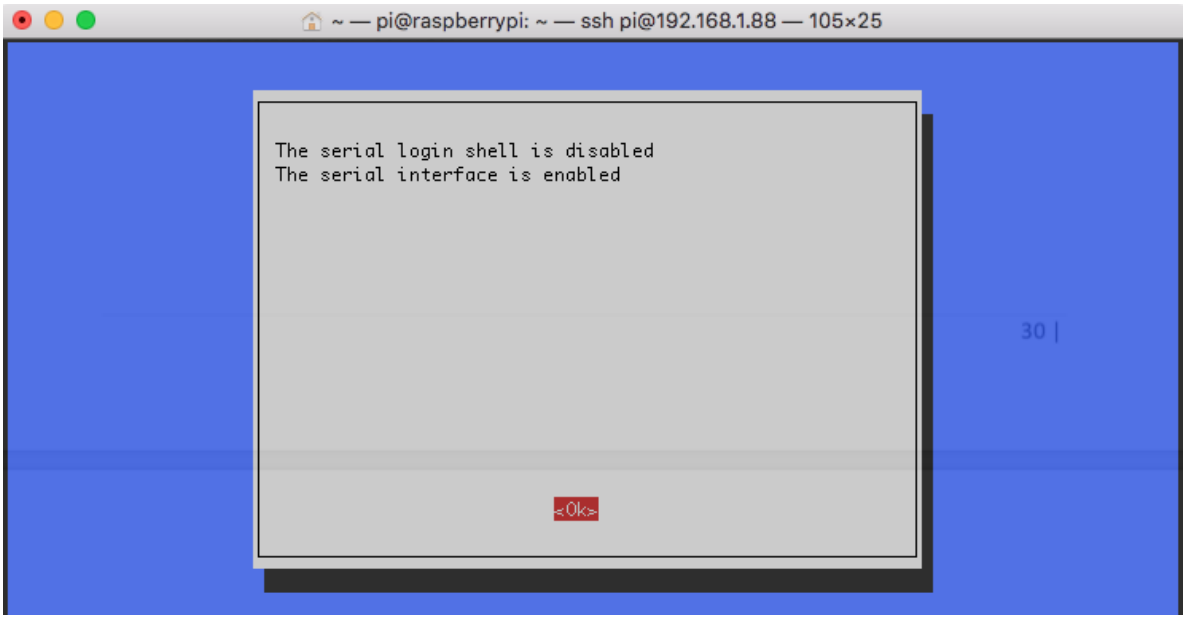


Figura 26: Captura terminal, uso de raspi-config configuración puerto serial RPi.

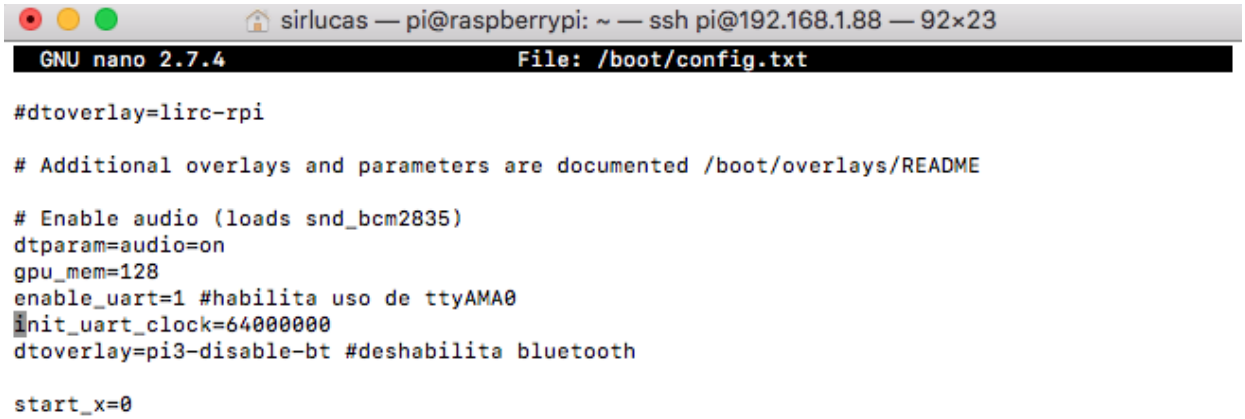


Figura 27: Captura terminal, edición */boot/config.txt* en RPi.

Según la documentación, DroneKit se puede instalar mediante *pip* utilizando el comando:

```
$ pip install dronekit dronekit-sitl
```

Otra opción es clonar el repositorio Git de la librería, compilar e instalar directamente en el Raspberry Pi, ver imagen 28. DroneKit solo esta disponible para Python 2.7.

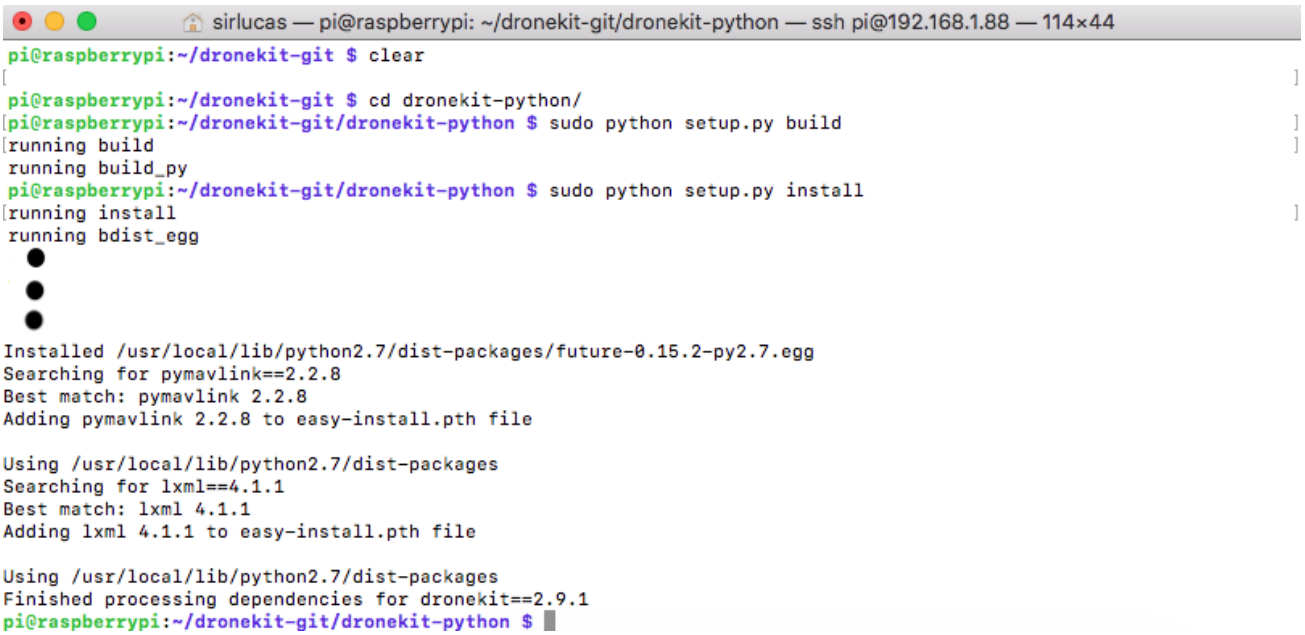


Figura 28: Captura terminal, compilación e instalación librería DroneKit Python en RPi.

Una vez lista la configuración previa del Raspberry Pi, se configura el APM, estableciendo velocidad de transmisión en 115200*Baudios* y protocolo MAVLink a través del puerto serial para telemetría, utilizando la aplicación para macOS; *APM Planner 2.0*, mediante conexión USB con el APM, ver imagen 29.

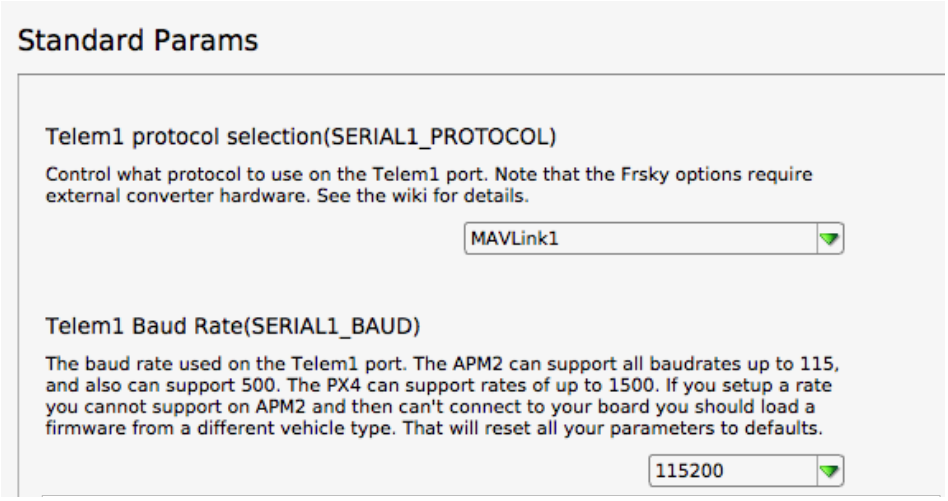


Figura 29: Captura APM Planner 2.0, Configuración serial APM.

Al completar la configuración de hardware y software del RPI y APM, se comienza a utilizar su interconexión para enviar/recibir mensajes entre los componentes. La primera acción que se realiza es un "hola Drone" ejecutando un algoritmo escrito en Python que utiliza la librería DroneKit para crear un objeto de la clase **Vehicle** (ver algoritmo 5, Cap. 7).

El algoritmo 3, muestra como se utiliza *connect()* para obtener una instancia de un vehículo conectado, en este caso, APM↔RPi con los siguientes parámetros:

- **connection_string:** Este parámetro especifica la dirección del objetivo de conexión hacia el vehículo, en este caso usando puerto serial */dev/ttyAMA0* desde RPi.
- **baud:** El valor de *baud_rate* depende del tipo de conexión, en este caso 115200*baudios*.
- **wait_ready:** Atributo booleano, que determina si la conexión se retorna inmediatamente, o se espera a que los parámetros del vehículo estén completamente cargados antes de retornar (por defecto es *False*).

Algoritmo 3 Método connect, libreria DroneKit-Python.

```
from dronekit import connect
apm = connect(connection_string, baud=baud_rate, wait_ready=True)
```

La información de estado del APM, se obtiene consultado distintos atributos del objeto *apm*, entre los mas relevantes para este proyecto estan:

- **apm.gps_0:** Retorna el número de satélites disponibles y cuantos esta utilizando para triangular su posición actual.
- **apm.attitude:** Retorna los parámetros: Yaw, Pitch y Roll que presenta el vehiculo.
- **apm.mode.name:** Retorna el atributo "name" correspondiente al modo de vuelo actual del vehículo.
- **apm.last_heartbeat:** Retorna el tiempo a la ultima conexión via MAVLink con el vehículo, en segundos.
- **apm.is_armable:** Devuelve un booleano (True/False) según el objeto pueda armar motores, luego de ejecutar la rutina "pre-arm check"[24] del vehículo.
- **apm.system_status.state:** Retorna el atributo "state", indicando el estado actual del drone (stand-by, armed, disarmed).

[24]Rutina previa al armado de motores, comprueba estado de triangulación GPS, calibrado inicial de barómetro, brújula y comprueba estado de batería

6.2. Conexión y montaje de sensores

6.2.1. Sensores de Ultrasonido

Los sensores de ultrasonido utilizados en el proyecto corresponden al modelo HC-SR04, este modelo de sensor necesita una tensión de alimentación TTL para funcionar ($5V$), por tanto, su salida "ECHO" también es TTL[17]. Sin embargo, un pin GPIO designado como pin de entrada en la RPi puede leerse como alto ($3,3V$) o bajo ($0V$)[38], a partir de esto se hace necesario utilizar un circuito de control de tensión denominado "Divisor de tensión", ver figura 30.

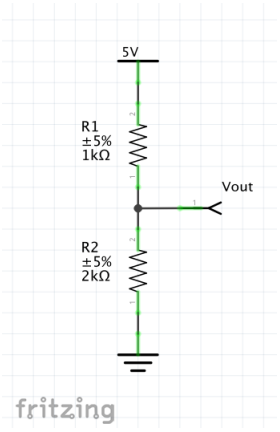


Figura 30: Circuito esquemático: Divisor de Tensión.

La configuración del circuito divisor de tensión, tiene dos resistencias, R_1 y R_2 permitiendo una salida V_{out} proporcional a V_{in} de acuerdo a la siguiente ecuación:

$$V_{out} = \frac{R_2}{R_1 + R_2} * V_{in}$$

Donde, a partir de una tensión de entrada (V_{in}) TTL desde el pin "ECHO" del sensor HC-SR04, con $R_1 = 1k\Omega$ y $R_2 = 2k\Omega$ se obtiene una salida V_{out} de $3V3$ que puede ser conectada a un pin GPIO de entrada de la RPi sin problemas.

Se utilizan 5 sensores, dispuestos de la siguiente forma; uno hacia adelante del drone (Sensor1), uno hacia la derecha (Sensor3), uno hacia atrás (Sensor2), uno hacia la izquierda (Sensor5) y un sensor apuntando hacia abajo (Sensor2) para calcular la altura inicial del drone.
El diagrama de conexión (ver figura 55, anexo A) muestra como se conectan los 5 sensores a una RPi-3b utilizando el descrito divisor de tensión desde el pin ECHO de cada sensor. La imagen 31 muestra la configuración espacial de sensores ya montada en el drone del proyecto.

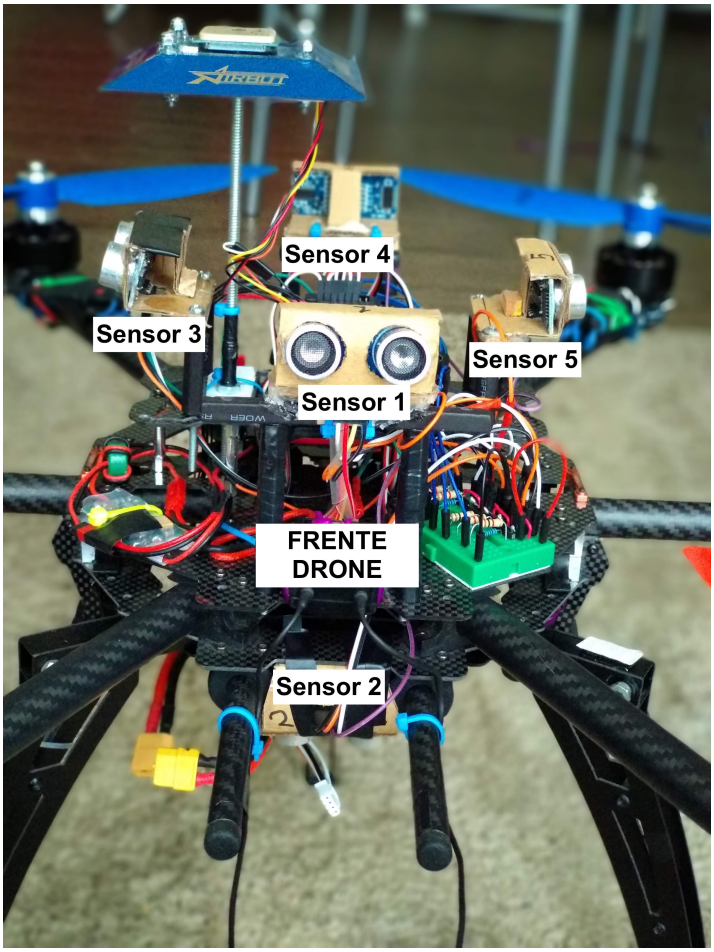


Figura 31: Drone con sensores montados.

6.3. Proceso de detección de obstáculos

El proceso de detección de obstáculos consiste en una configuración de 5 sensores de ultrasonido a bordo (descritos anteriormente) que permiten realizar mediciones periódicas de distancia dentro de un rango de visión ($4m$ y $\pm 30^{o}$, ver imagen 32) y de esta forma detectar obstáculos que estén dentro del area de "visión" del UAV. Para lograr esto, se realizan pruebas con dos librerías Python que permiten el uso de dispositivos GPIO-Raspberry. Las pruebas consisten en medir distancias entre un sensor de ultrasonido HC-SR04 y un objeto. Las librerías Python utilizadas son; GPIOZERO para manejo de dispositivos GPIO desarrollado por Ben Nuttal, miembro de Raspberry Pi Foundation [18] y la librería RPi.GPIO, instalada por defecto en Raspbian Linux.

6.3.1. Descripción de pruebas de medición

Para comprobar el correcto funcionamiento de los sensores que se utilizan para el proyecto se realiza el conjunto de pruebas descrito a continuación:

- Pruebas de distancia, objeto fijo a menos de 0.5m.
- Pruebas de distancia, objeto fijo a más de 3m.
- Pruebas de distancia, objeto fijo entre 0.5m y 3m.
- Tiempo de medición para 15 muestras, cálculo de promedio.
- Prueba lectura de todos los sensores, utilizando la *Clase Sensor*.

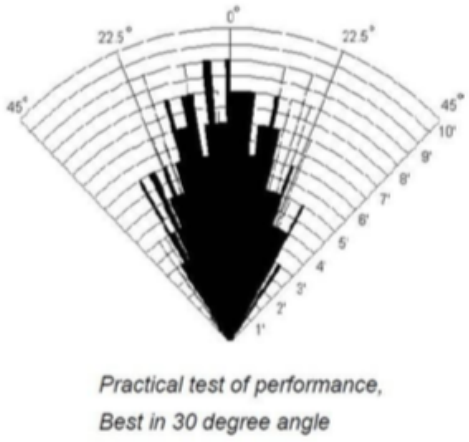


Figura 32: Angulo de detección de objetos para sensor HC-SR04, [17].

6.4. Proceso de evasión de obstáculos

El proceso de evasión de obstáculos consiste en una evaluación de distancia entre el drone y algún obstáculo dentro del rango de visión de los sensores para luego enviar comandos de control por sobrescritura de canales RC T_x o variación de "attitude" (por ángulos de navegación) al APM.

En primer lugar se definen limites de distancia dentro de los cuales los algoritmos de detección y evasión pueden funcionar, esto es determinado por el rango máximo de medición de los sensores, también se considera a que distancia mínima es seguro para el UAV funcionar, evitando daño por colisión.

Los tres limites que conforman la burbuja de funcionamiento (ver figura 33) son:

- **Limite lejano (3m):** Consiste en la distancia máxima a la que el drone considera que existe un obstáculo, cualquier cosa mas allá de este limite es invisible al drone.
- **Limite cercano (1m):** Consiste en la distancia mínima entre el drone y un obstáculo para que este pueda reaccionar.
- **Limite de peligro (0.5m):** Consiste en la distancia de peligro para el drone, si el drone esta en ese limite, debe iniciar protocolo de detención de movimiento y aterrizaje de emergencia, es posible que el drone sufra daños por colisión.

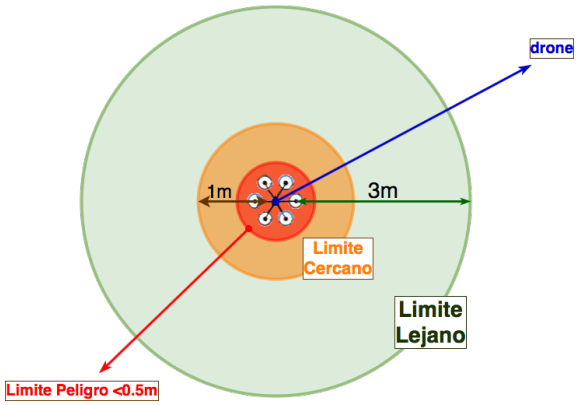


Figura 33: Representación limites de distancia.

Para detectar y evadir obstáculos, también se deben considerar los modos de vuelo disponibles para el drone, considerando que para el APM2.6 se utiliza el firmware *Ardupilot AC3.2.1*.
Se pueden definir dos grupos generales para los modos de vuelo; modos de vuelo **autónomo** que requieren GPS, y modos de vuelo **manual** que pueden no requerir de GPS, pero necesitan un operador mediante RC T_x, considerando también en este segundo grupo los modos semi-asistidos, todos los modos de vuelo se describen en profundidad en [40]. En el primer grupo se tienen, por ejemplo, los modos *"GUIDED"*, *"AUTO"* y *"RTL"* y en el segundo, los modos *"STABILIZED"*, *"LAND"* y *"ALT-HOLD"*.

Esta agrupación permite implementar diferentes soluciones para los algoritmos de detección y evasión de obstáculos dependiendo del tipo de vuelo, a continuación se describen dos acercamientos a la solución de detección y evasión, denominados; **evasión por retroceso** para el grupo de modos de vuelo manual y **evasión por navegación**, para los modos de vuelo autónomo, el funcionamiento del proceso de detección y evasión general se puede observar en la figura 34.

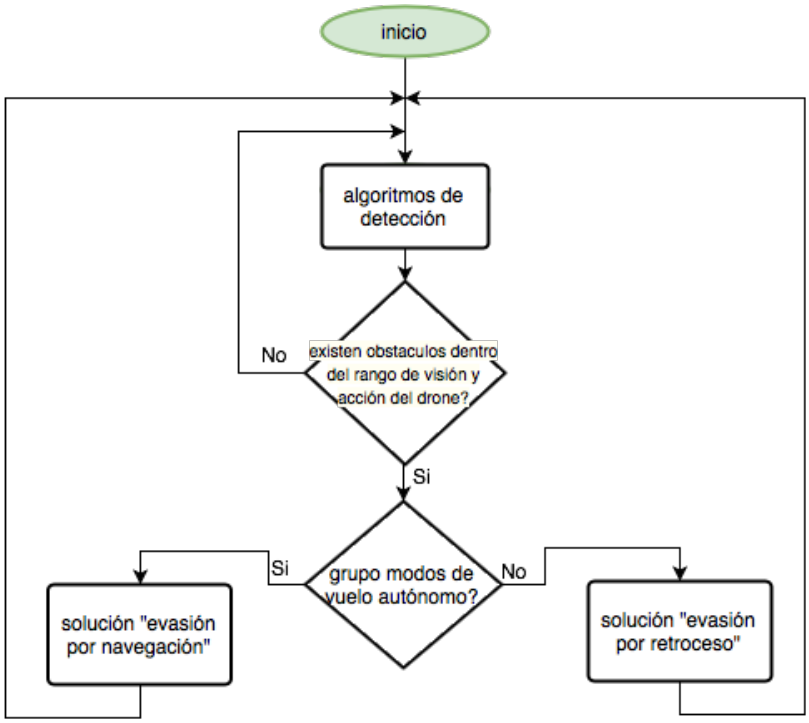


Figura 34: DFD funcionamiento general proceso de detección y evasión de obstaculos.

6.4.1. Evasión por retroceso

El acercamiento inicial para implementar esta solución es evaluar distancia con todos los sensores y comprobar si existen obstáculos que sean necesarios evadir. Entonces se comprueba si el drone tiene espacio en la dirección contraria al obstáculo, es decir evaluación por "pitch" y "roll" (evaluación "adelante-atrás" e "izquierda-derecha", respectivamente) para enviarlo en esa dirección hasta que la distancia al obstáculo este dentro del área de seguridad, mayor al "Limite Cercano", ver representación en imagen 35. Este tipo de acercamiento a una solución de detección y evación de obstáculos es muy útil para drones operados por radio control, es decir, el grupo de modos de vuelo manual.

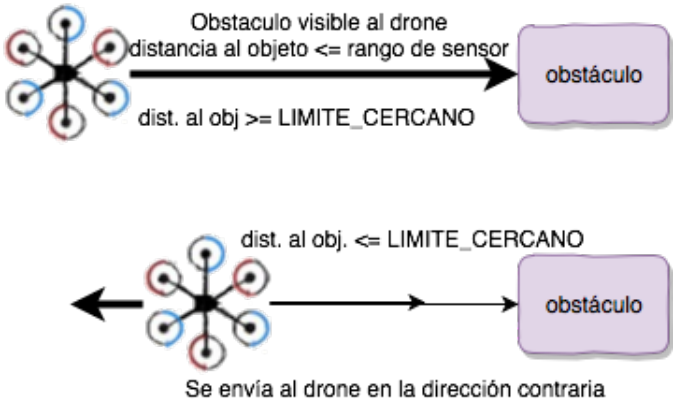


Figura 35: Representación funcionamiento "evasión por retroceso".

Al entrar en modo de evasión por retroceso, el operador no interfiere. Este método funciona por sobrescritura de canales del radio control, es decir, el drone es autopilotado durante un breve tiempo por GCS a bordo (RPi), por tanto es este quien envía instrucciones de movimiento al APM ignorando las instrucciones de entrada por el radio control. Si el operador se mueve hacia un obstáculo intencionada o casualmente, el sistema lo asiste de forma autónoma tomando el control del drone, moviéndolo en sentido contrario al obstáculo hasta alcanzar una distancia segura para devolver el control al operador y este pueda nuevamente enviar comandos de vuelo mediante radio control. En el diagrama 36 se puede observar el proceso de sobrescritura de canales RC para el método de evasión por retroceso al realizar evaluación por "pitch" y "roll" para encontrar obstáculos.

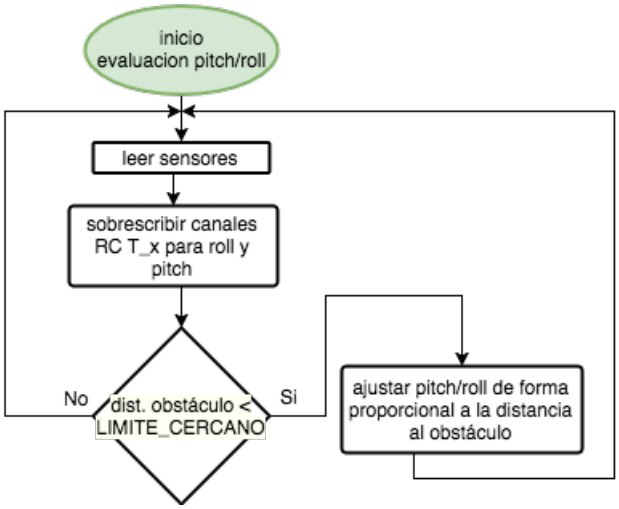


Figura 36: DFD funcionamiento "evasión por retroceso".

6.4.2. Evasión por navegación

Cuando el UAV esta en alguno de los modos autónomos, entonces este debe realizar un vuelo guiado por GPS y navegar hacia el o los puntos definidos según su misión de vuelo. Por ejemplo, para el modo *RTL*, el APM esta configurado para que el drone ajuste su altura a 15m y luego viajar en linea recta hacia la posición de despegue (home location). En este escenario, si hay algún obstáculo, el drone debe replanear la ruta manteniendo el listado de waypoints[25] que tiene como misión. Para evadir un obstáculo de frente al drone, se debe realizar una búsqueda lateral, dentro de un rango de movimiento definido y manteniendo la altura (evaluación por "roll"), si se encuentra espacio adelante dentro del rango de búsqueda, entonces se puede seguir avanzando y recuperar ruta original hacia el destino. Por el contrario, si no hay espacio lateral para evadir, entonces se puede probar ascendiendo para pasar el obstáculo por arriba (dentro de un limite de búsqueda vertical) y luego bajar a la altura original. Finalmente si no encuentra espacio para evadir dentro de los limites de búsqueda, entonces se debe aterrizar porque no es posible evadir el obstáculo.
Los diagramas 37, 38 y 39 muestran acercamiento inicial a algoritmo de evasión por navegación.

[25]coordenadas para referencia geográfica utilizadas en vuelos guiados por GPS

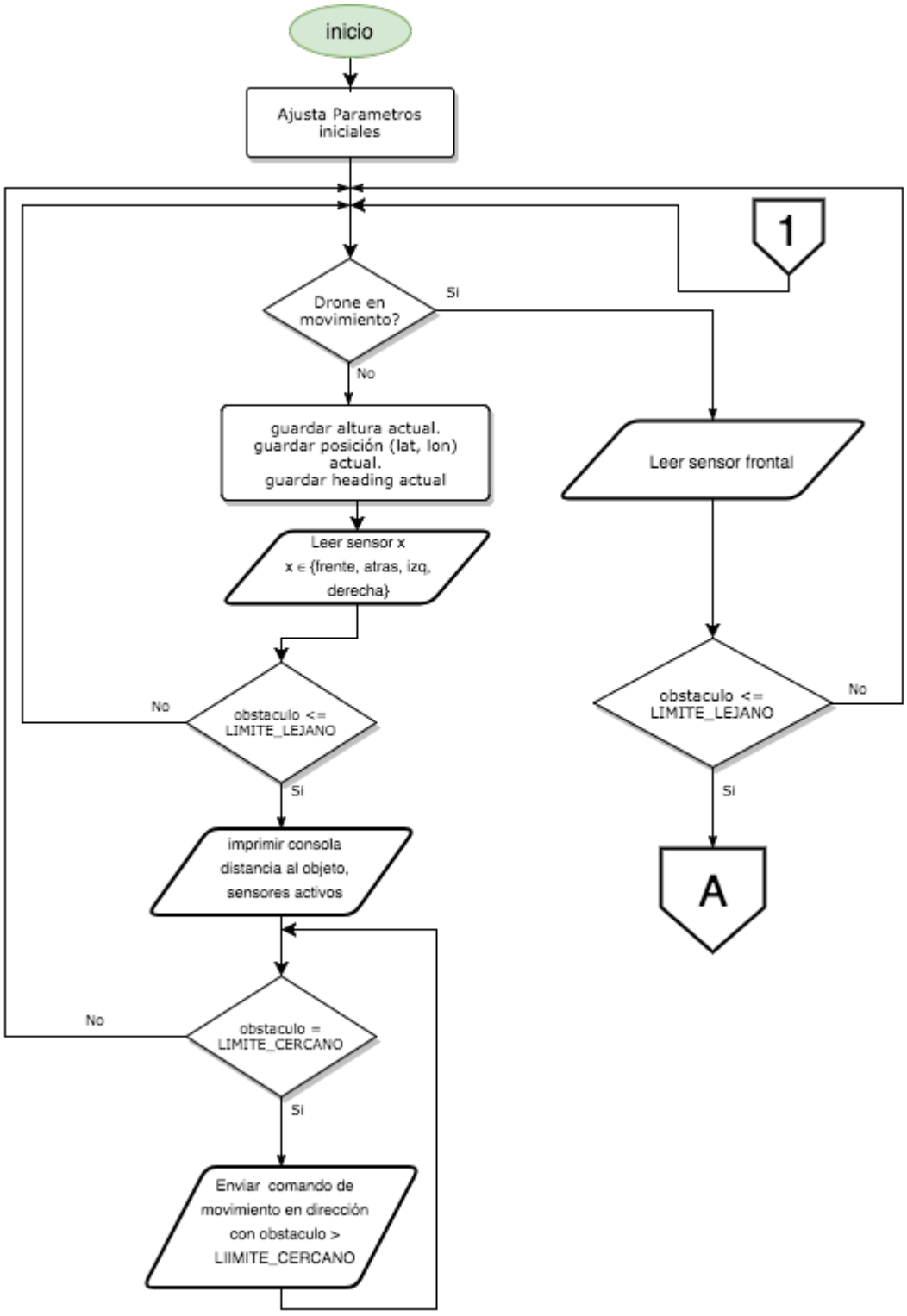


Figura 37: DFD - parte 1. Detección y evasión de obstáculos Clase EvasionObstaculos.

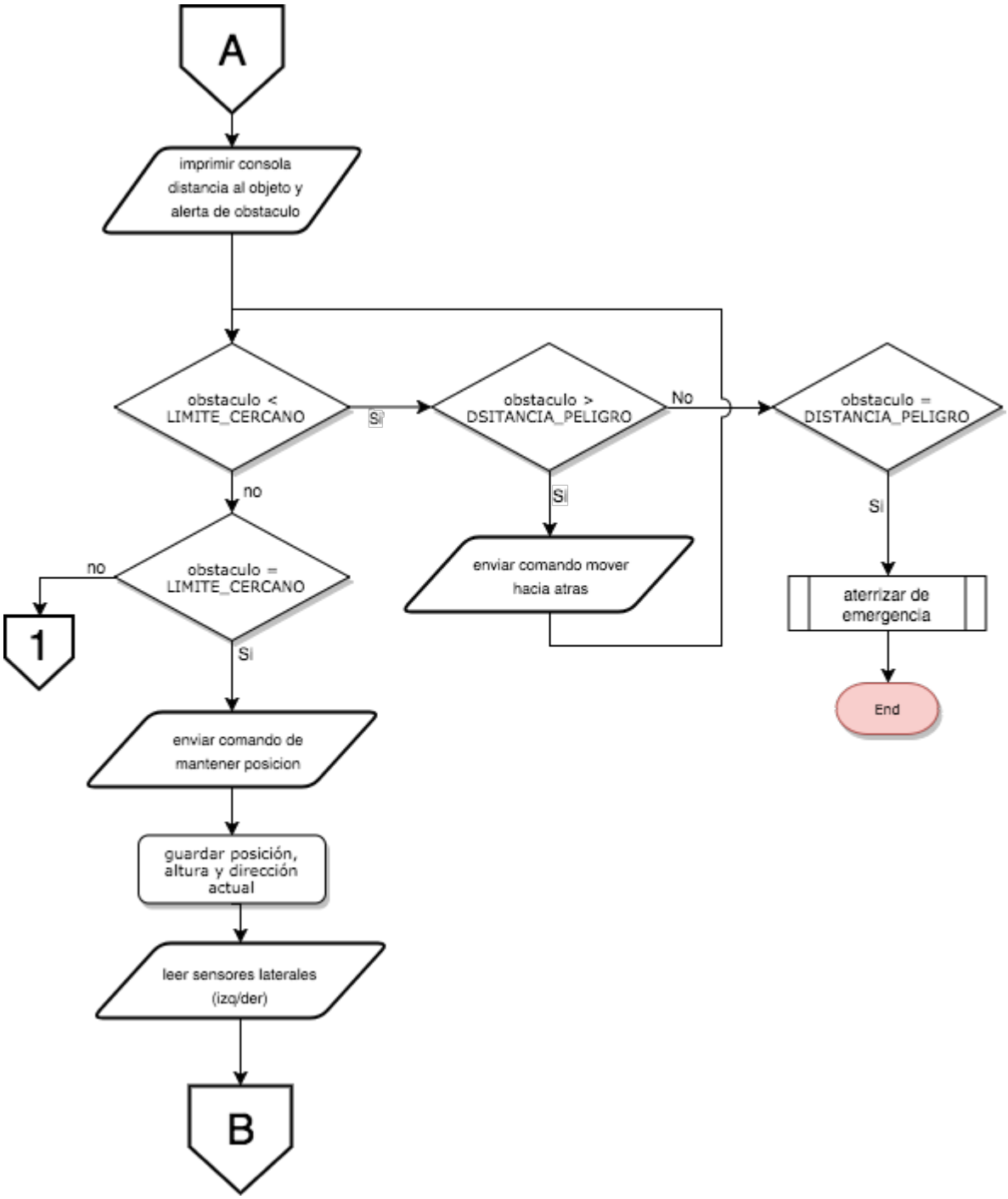


Figura 38: DFD - parte 2. Detección y evasión de obstáculos Clase EvasionObstaculos.

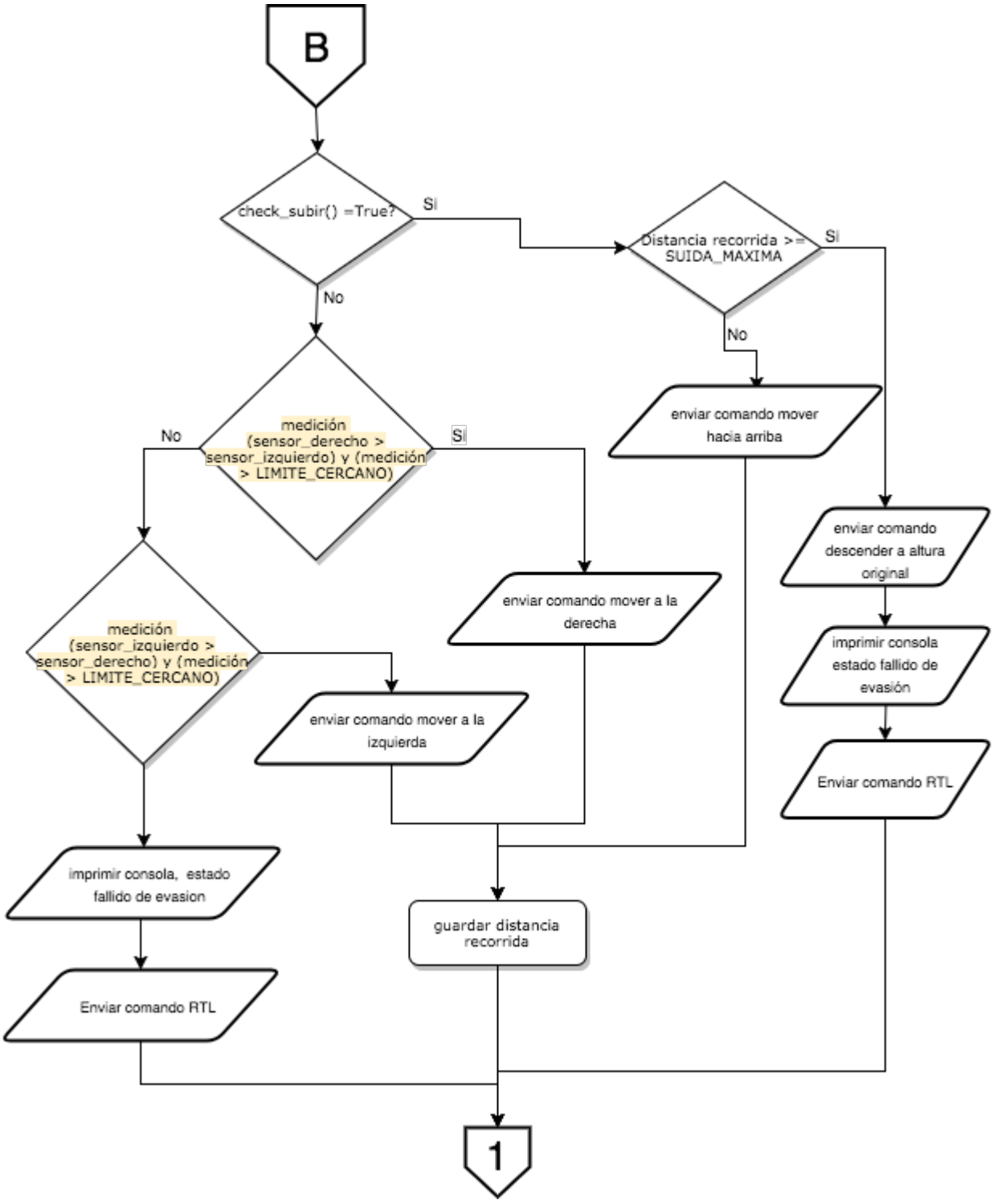


Figura 39: DFD - parte 3. Detección y evasión de obstáculos Clase EvasionObstaculos.

6.5. Algoritmos Implementados

Para cumplir con los objetivos planteados, se implementan cuatro clases; **Sensor**, **Drone**, **ControlVuelo** y **EvadirObstaculo**, ver figura 40. En conjunto conforman el sistema de detección y evasión de obstáculos que se plantea en este proyecto. Estas clases se implementan en Python con librerias que permiten el uso de GPIO en el RPi y comunicación APM2.6↔RPi con MAVLink.

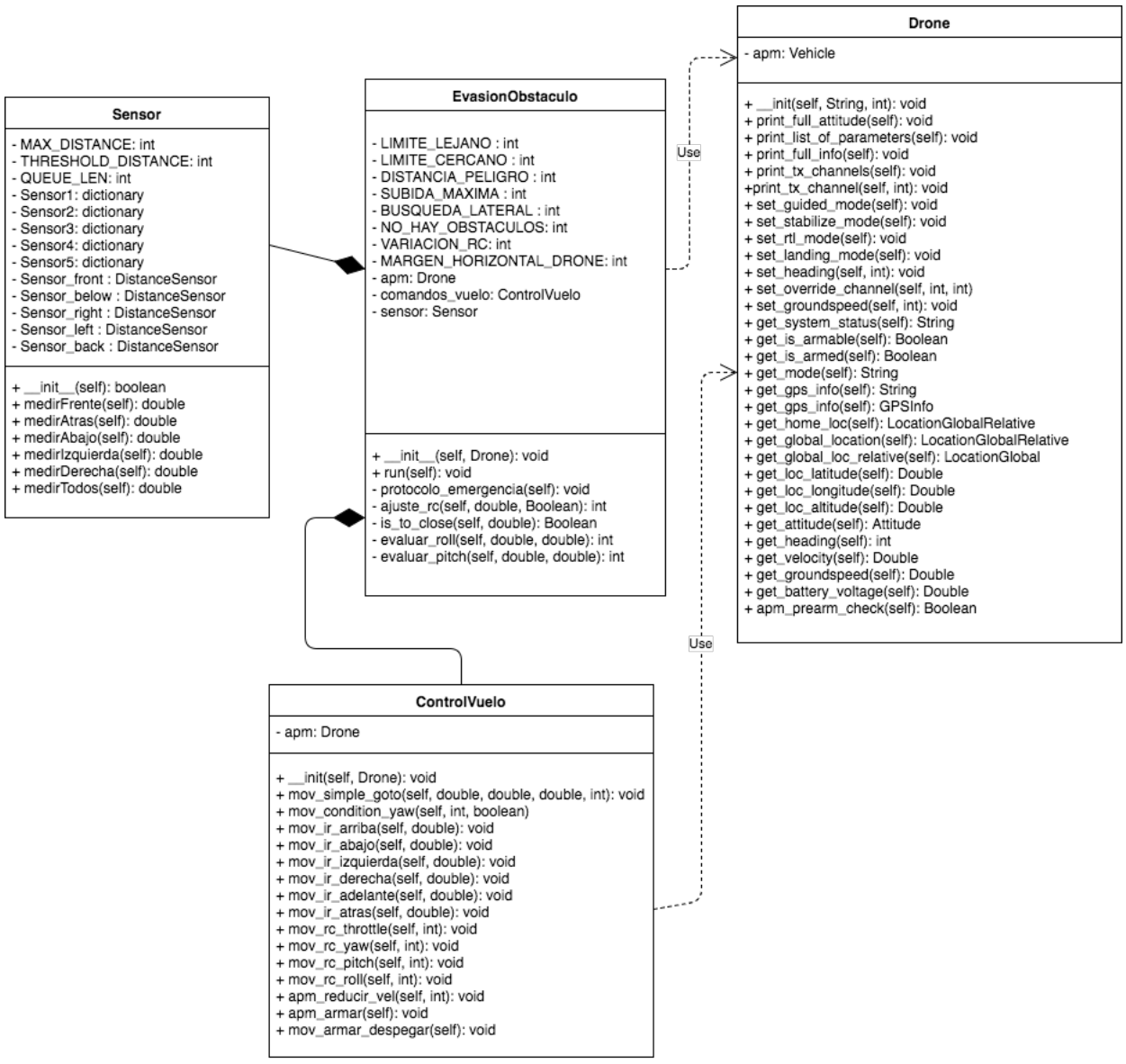


Figura 40: Diagrama de clases desarrollado, sistema de detección y evasión de obstaculos.

6.5.1. Clase Sensor

Para dotar de "vision" del entorno al drone se crea la Clase Sensor (ver algoritmo completo en anexo D). Esta clase implementa la librería GPIOZero e instancia 5 (cinco) sensores de la clase *DistanceSensor* para obtener datos desde los sensores HC-SR04 montados en el drone. Posee métodos públicos para leer cada uno de los sensores por separado y un método para obtener una lista de lectura de los 5 sensores a la vez, ver algoritmo 4.

Atributos de clase (privados):

- **__MAX_DISTANCE:** Indica la distancia máxima para funcionamiento del sensor, coincide con el *LIMITE_LEJANO* propuesto. Tipo: *int*.
- **__THRESHOLD_DISTANCE:** Indica la distancia mínima de seguridad para medición, coincide con *LIMITE_PELIGRO* propuesto. Tipo: *int*.
- **__QUEUE_LEN:** Número de muestras que mide el sensor para calcular una distancia media. Tipo: *int*.
- **__sensor_x**: con $x \in [1, 5]$, Indica los puertos GPIO de cada sensor a utilizar. Tipo: *dict*.

Sus métodos públicos son:

- **is_created(self)**: método que imprime un mensaje de acuerdo a variable *created: boolean* y retorna un valor booleano (True/False).
- **medirFrente(self)**: método que hace lectura del sensor correspondiente y retorna ese valor (*lectura_frente: double*).
- **medirAbajo(slef)**: método que hace lectura del sensor correspondiente y retorna ese valor (*lectura_abajo: double*).
- **medirDerecha(self)**: método que hace lectura del sensor correspondiente y retorna ese valor (*lectura_derecha: double*).
- **medirAtras(self)**: método que hace lectura del sensor correspondiente y retorna ese valor (*lectura_atras: double*).
- **medirIzquierda(self)**: método que hace lectura del sensor correspondiente y retorna ese valor (*lectura_izquierda: double*).
- **medirTodos(self)**: método que hace lectura de todos los sensores y retorna una lista de diccionarios con los valores para cada sensor (*mediciones: list*).

Algoritmo 4 Fragmento de implementación Python, Clase Sensor.py - Métodos.

```
1         #retorna medida sensor frontal en cm.
2         def medirFrente(self):
3                 lectura_frente = round(self.__sensor_front.distance *
                      100)
4                 return lectura_frente
5
6         # retorna medida sensor trasero en cm
7         def medirAtras(self):
8                 lectura_atras = round(self.__sensor_back.distance *
                      100)
9                 return lectura_atras
10
11        # retorna medida sensor lateral izquierdo en cm
12        def medirIzquierda(self):
13                lectura_izquierda = round(self.__sensor_left.distance *
                       100)
14                return lectura_izquierda
15
16        # retorna medida sensor lateral derecho en cm
17        def medirDerecha(self):
18                lectura_derecha = round(self.__sensor_right.distance *
                      100)
19                return lectura_derecha
20
21        # retorna medida sensor adelante-abajo en cm
22        def medirAbajo(self):
23                lectura_abajo = round(self.__sensor_below.distance *
                      100)
24                return lectura_abajo
25
26        def medirTodos(self):
27                mediciones = []
28                mediciones.append({'ID' : 'sensor_front' , 'distancia'
                      : self.medirFrente()})
```

6.5.2. Clase Drone

Para establecer conexión entre el RPi y el APM se utiliza la clase Drone, ver algoritmo completo en anexo E. Se crea un objeto de la clase **Vehicle** del API DroneKit al usar el método *connect()* y se crean métodos públicos que permiten obtener datos de estado y ajustar diversos parámetros del APM.

Su constructor recibe dos atributos; *connection_string* y *baud_rate*, para establecer la dirección de conexión con el drone (serial UART0: */dev/tty/AMA0*) y velocidad del enlace (115200 baudios).

Entre los métodos más relevantes de la clase, destacan:

- **Ajustes (setters):**
  - **set_guided_mode(self):** Este método, ajusta el modo de vuelo actual del drone a "GUIDED" para que el RPI en función de GCS pueda enviar instrucciones de vuelo utilizando coordenadas.
  - **set_stabilized_mode(self):** Este método, ajusta el modo de vuelo actual del drone a "STABILIZE" permitiendo el control manual, mediante radio control, y del RPI por sobrescritura de canales.
  - **set_rtl_mode(self):** Método que ajusta el modo de vuelo actual del drone a "RTL", esto provoca que el drone vuelva a la posición de despegue y aterrice.
  - **set_override_channel(self, num_canal, variacion):** Método que permite sobrescribir los canales de entrada del APM, que normalmente recibe desde el radio control, indicando el numero del canal a sobrescribir y su nivel de variación.
  - **set_attitude(roll_angle, pitch_angle, yaw_rate, thrust, duration):** Método que permite definir movimiento relativo del drone, basándose en ángulos de navegación (pitch, roll, yaw) y throttle o acelerador, indicado como thrust (empuje).
- **Obtención de estado (getters):**
  - **get_location_global_relative(self):** Método que retorna la ubicación del drone en términos de latitud/longitud y altura relativa del drone.
  - **get_channel_value(self, channel):** Método que retorna el valor (*valor_canal:int*) de un canal especifico (*channel: int*) desde la emisora de radio control.
  - **get_home_location(self):** Método que retorna la posición inicial de despegue del drone (latitud ,longitud y altura relativa). Esta información es utilizada por el drone al estar en modo de vuelo RTL.

- **get_attitude(self):** Método que retorna la "actitud" del drone en términos de ángulos de navegación (pitch, roll, yaw).
- **get_heading(self):** Método que retorna hacia donde "apunta" el drone con respecto al norte (Norte = 0), en grados 0º-360º. Type: int.
- **get_gps_info(self):** Método que retorna un objeto del tipo *GPSInfo*, indica HDOP y VDOP[26], nivel de estado fijo (*fix_type*) y número de satélites disponibles para calculo de posición.

▪ **Información por consola (prints):**

- **print_full_info(self):** Imprime por consola, los atributos más relevantes de estado del drone: versión de firmware, attitude, información de gps, entre otros.)
- **print_tx_channel(self, channel):** Imprime por consola, el valor de un canal especifico del RC T_x.
- **print_full_attitude(self):** Imprime por consola el comportamiento completo del drone en términos de ángulos de navegación y nivel de throttle.

[26]Horizontal and Vertical Dilution Of Precision [39].

6.5.3. Clase ControlVuelo

Esta clase (ver algoritmo completo en anexo F), en su constructor, recibe como parámetro un objeto de la clase **Drone**. Permite ajustar movimientos del drone, mediante sobrescritura de canales RC T_x, por ajuste de "attitude" e instrucciones predefinidas como *"mov_simple_goto()"* y *"mov_armar_despegar()"* , los métodos de esta clase que poseen el prefijo *"mov"* indican que corresponden a movimientos.

Los métodos públicos más destacados son:

- **Movimientos predefinidos:**
  - **mov_simple_goto(self, lat, lon, height, vel):** Método que envía instrucción al drone para moverse a un destino especifico (en modo de vuelo "GUIDED"), recibe como parámetros el destino (lat,long y height) y la velocidad de desplazamiento (en m/s).
  - **mov_armar_despegar(self, alturaDespegue):** Método que envía instrucción al drone para armar motores y despegar a una altura especificada por parámetro en la función (*alturaDespegue*), manteniendo posición.
  - **mov_aterrizar(self):** Método que configura el modo "LAND" en la clase Drone, permitiendo al drone aterrizar.
- **Movimiento por sobrescritura de RC-T_x:**
  - **mov_rc_throttle(self, variacion)**: Método que sobrescribe el canal "throttle" que envía el RC, permitiendo al drone elevarse o descender, recibe por parámetro (*"variacion"*) el grado de variación del canal (entre 1100-1900, punto medio en 1500).
  - **mov_rc_yaw(self, variacion)**: Método que sobrescribe el canal "yaw" que envia el RC, permitiendo un giro horizontal sobre el "eje y". Recibe por parámetro (*"variacion"*) el grado de variación del canal (entre 1100-1900, punto medio en 1500).
  - **mov_rc_pitch(self, variacion)**: Método que sobrescribe el canal "pitch" que envía el RC, permitiendo al drone avanzar o retroceder (con referencia al frente del drone). Recibe por parámetro (*"variacion"*) el grado de variación del canal (entre 1100-1900, punto medio en 1500).
  - **mov_rc_roll(self, variacion)**: Método que sobrescribe el canal "roll' que envía el RC, permitiendo al drone desplazarse lateralmente a derecha o izquierda. Recibe por parámetro (*"variacion"*) el grado de variación del canal (entre 1100-1900, punto medio en 1500).

- **Movimiento por ajuste de "Attitude":**
  - **mov_ir_arriba(self, altura):** Método que ajusta la altura del drone, utilizando cálculo de altura según datos de GPS y Barómetro, ascendiendo. Recibe por parámetro la altura (en metros) que se necesita alcanzar.
  - **mov_ir_abajo(self, altura):** Método que ajusta la altura del drone, utilizando cálculo de altura según datos de GPS y Barómetro, descendiendo. Recibe por parámetro la altura (en metros) que se necesita alcanzar.
  - **mov_ir_adelante(self, pitch_angle , duration):** Método que ajusta el ángulo de navegación "pitch" de forma positiva, para mover el drone hacia adelante (movimiento relativo al drone)
  - **mov_ir_atras(self, pitch_angle , duration):** Método que ajusta el ángulo de navegación "pitch" de forma negativa, para mover el drone hacia atrás, retroceder (movimiento relativo al drone)..
  - **mov_ir_derecha(self, roll_angle, duration):** Método que ajusta el ángulo de navegación "roll" de forma positiva, para mover el drone hacia la derecha (movimiento relativo al drone).
  - **mov_ir_izquierda(self, roll_angle, duration):** Método que ajusta el ángulo de navegación "roll" de forma negativa, para mover el drone hacia la izquierda (movimiento relativo al drone).
  - **mantener_altitud(self, duration):** Método que nivela pitch, roll y mantiene altitud.

6.5.4. Clase EvasionObstaculos

En esta clase se encuentran lo algoritmos de evaluación y evasión de obstáculos, acercamiento a solución evasión por retroceso, extiende de la clase **Threading** e instancia un objeto de la clase **Sensor**, permitiendo la medición de distancias de los 5 sensores de ultrasonido montados en el drone a través de sus métodos públicos. Hace uso de las clases **Drone** y **ControlVuelo** para consultar datos de estado como *altitud*, *posición global relativa* y utilizar métodos de control de movimiento como *mov_ir_adelante()*, *mantener_altitud()*, entre otros. El constructor de esta clase recibe por parámetro un objeto de la clase **Drone**; *__init__(self,Drone)*.

Los métodos mas relevantes de esta clase para acercamiento de evasión por retroceso, son:

- **run(self):** Método que se ejecuta al iniciar el hilo, es el método principal de la clase.
- **__protocolo_emergencia(self):** Método que utiliza los métodos públicos de la clase **ControlVuelo** y **Drone** para realizar un aterrizaje de emergencia y desarmado de motores, producto de que existe un obstáculo esta dentro de la *"DISTANCIA_PELIGRO"* del drone. Type: Void.
- **__ajuste_rc(self, Double, Boolean):** Método que ajusta de forma proporcional la variación del canal RC con respecto a la distancia al obstáculo. Retorna el valor para el canal RC que se utiliza en sobrescritura. Type: int (valor_rc $\epsilon[1100, 1800]$)
- **__is_to_close(self, Double):** Método que devuelve un Boolean que indica si un obstáculo esta dentro del limite cercano para reaccionar. Type: Boolean.
- **__evaluar_roll(self, Double, Double):** Método que evalúa distancias medidas por sensores laterales (izquierda y derecha) y determina si es necesario evadir algún obstáculo de forma lateral, mediante sobrescritura de canales RC "roll". Retorna el valor de roll correspondiente para ser sobrescrito y enviado al APM. Type: int.
- **__evaluar_pitch(self, Double, Double):** Método que evalúa distancias medidas por sensores delantero y trasero y determina si es necesario evadir obstáculos al frente o atrás del drone, mediante sobrescritura de canales RC para variar canal "pitch". Retorna el valor de pitch correspondiente para ser sobrescrito y enviado al APM. Type: int.

7. Implementación de pruebas

En este capítulo se describen las pruebas implementadas para comprobar correcto funcionamiento de los componentes de hardware, algoritmos de detección de obstáculos y algoritmos de evasión de obstáculos.

7.1. Prueba de comunicación RPi↔APM

Para comprobar correcta comunicación entre el Raspberry Pi y el Ardupilot, se ejecuta un "hola Drone", ver algoritmo 5. Según se describe en el capítulo 6.1.

Algoritmo 5 Fragmento de implementación Python, test conexión RPi↔APM.

```
from dronekit import connect, VehicleMode
import time

connection_string = "/dev/ttyAMA0" #se utiliza UART0 pines GPIO14/15 Serial
baud_rate = 115200 # establece velocidad maxima para APM2.6 = 115200 baudios

print(">>>> Conectando con APM en: %s <<<<" %(connection_string))
#Conecta con el vehiculo, en este caso el Drone con un APM
apm = connect(connection_string, baud=baud_rate, wait_ready=True)
# Leer informacion desde APM:
print("GPS: %s" %apm.gps_0)
print("Attitude: %s" %apm.attitude) #attitude: roll, pitch, yaw
print("Modo de vuelo actual: %s" %apm.mode.name) # modo actual de vuelo, editable.
print("Estado Bateria: %s" %apm.battery)
print("Ultimo Heartbeat: %s" %apm.last_heartbeat)
print("Se puede armar?: %s" %apm.is_armable) # devuelve True/False
print("Estado: %s" %vehicle.system_status.state)
#Cerrar conexion con vehiculo
apm.close()
print("Prueba Conexion RPi<->APM: Exitosa!")
```

El resultado de esta prueba se observa en la imagen 41, donde se comprueba el uso de UART0 (/ttyAMA0) para comunicación serial, version de firmware Ardupilot (ArduCopter v3.2.1) y tipo de Drone (Frame: Hexa). Por tanto, se concluye que la conexión serial entre un Raspberry Pi 3b y un Ardupilot APM2.6 mediante implementación de DroneKit-Python para manejo de protocolo MAVLink es exitosa y se obtienen datos de estado del drone correspondientes a atributos y parámetros del objeto *"apm"* clase **"Vehicle"**: *"gps_0"*, *"attitude"*, *"mode.name"*, *"last_heartbeat"*, *"is_armable"* y *"system_status.state"*.

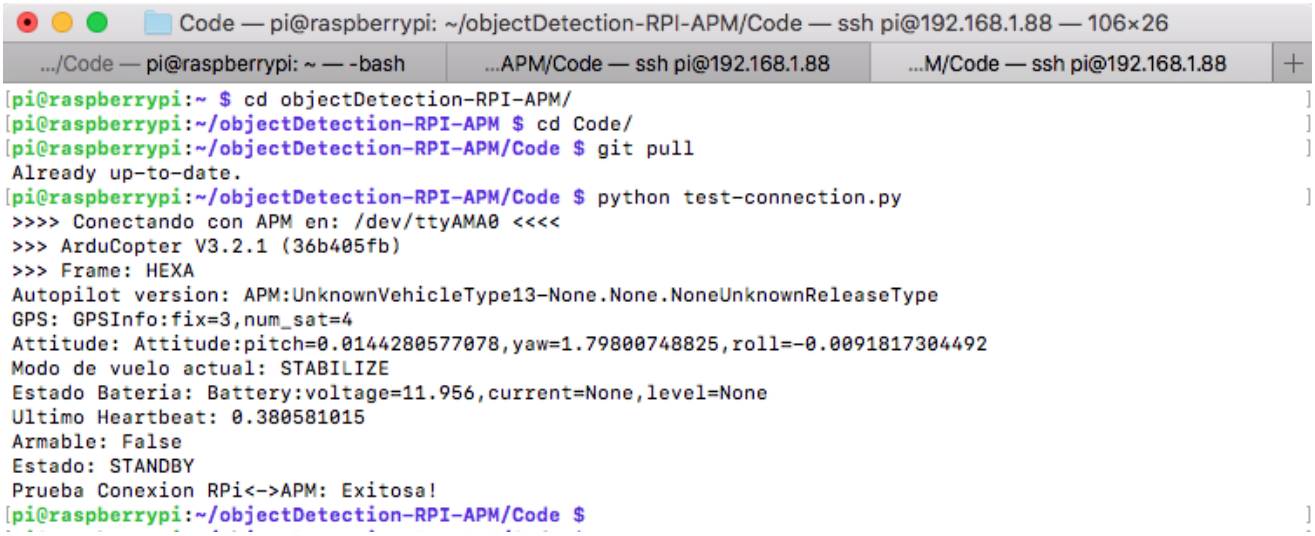


Figura 41: Gráfica medición sensor ultrasonido - Objeto Fijo a 0.4m

7.2. Pruebas de medición de distancias

Para las pruebas propuestas, se calcula el promedio entre 200 muestra de distancia entre los sensores y un objeto, para esto se implementa el algoritmo 6, utilizando la librería RPi.GPIO.

Para la prueba de medición de tiempo en sensar distancia promedio de 15 muestras, se implementan dos algoritmos en python, uno utilizando la librería GPIOZERO (ver algoritmo 7) y otro utilizando la librería RPi.GPIO (ver algoritmo 8).

Por último se ejecuta la prueba de la Clase Sensor, donde se utiliza el método *medirTodos()* para obtener un listado de todas las mediciones, considerando el calculo de distancia sobre 15 muestras por sensor. Para esta prueba se implementa el algoritmo 9, que crea una instancia de la Clase Sensor y realiza el llamado al método mencionado.

Algoritmo 6 Fragmento de implementación Python, test medición de distancia 200 muestras.

```
start = time.time()
for x in range (200):
        for i in range(len(sensor_list)): #listado de 5 sensores
                GPIO.output(sensor_list[i][0],GPIO.LOW) #TRIG pin
                time.sleep(0.0001)
                #Envia puslo trig y corta
                GPIO.output(sensor_list[i][0],GPIO.HIGH) #TRIG pin
                time.sleep(0.00001)
                GPIO.output(sensor_list[i][0],GPIO.LOW) #TRIG pin
                while GPIO.input(sensor_list[i][1]) == 0: #ECHO pin
                        pulse_start[i] = time.time()

                while GPIO.input(sensor_list[i][1]) == 1:# ECHO pin
                        pulse_end[i] = time.time()
                # diferencia de tiempo entre envio y echo
                pulse_duration[i] = pulse_end[i] - pulse_start[i]
                # velocidad del sonido (34300 cm/s) / 2 (ida y vuelta)
                distance[i] =  (pulse_duration[i] * 34300) / 2
                # redondea al entero mas proximo, con 2 decimales
                distance[i] = round( distance[i] , 2)
        print("medidas de distancia: ", distance)
end = time.time()
time = end - start
print ("tiempo lectura 200 valores: ",time,"s")
```

Algoritmo 7 Fragmento de implementación Python, test 15 muestras - librería GPIOZERO.

```
from gpiozero import DistanceSensor
import time
sensor = DistanceSensor(27, 17, queue_len=15, max_distance=5,
    threshold_distance=0.5)
print("objecto creado: ", sensor)
print("iniciando medicion de tiempo para 15 muestras")
start = time.time()
dis = round((sensor.distance*100),2) #distancia en cm
end = time.time()
time = end-start
print ("tiempo de la prueba promedio 15 mediciones: ", time,"seg. --
    Distancia : ", dis,"cm.")
```

Algoritmo 8 Fragmento de implementación Python, test 15 muestras. librería RPi.GPIO

```
        measurement = 0
        start = time.time()
        for x in range(15): #15 muestras
                GPIO.output(pinTrigger, True)
                time.sleep(0.00001)
                GPIO.output(pinTrigger, False)
                pulse_start = time.time()
                pulse_end = time.time()
                while 0 == GPIO.input(pinEcho):
                        pulse_start = time.time()
                while 1 == GPIO.input(pinEcho):
                        pulse_end = time.time()
                pulse_duration = pulse_stop - pulse_start
                measurement = measurement + ( (pulse_duration * 34300)
                    / 2)
        average = round(measurement/15, 2)
        end = time.time()
        time = end - start
        print ("tiempo de la prueba, promedio 15 mediciones: ", time,"
            seg. -- Distancia : ", distanciamedia,"cm.")
```

Algoritmo 9 Implementación Python para prueba de Clase Sensor - test_Sensor.py.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
Main_test, aqui se hacen los test para las clases implementadas Sensor,
    EvasionObstaculos y Drone.
*Intencionalmente no se utilizan tildes*
        main_test-py
        Posicionamiento de los sensores en el drone:
                ["frente-arriba",       "frente-abajo", "derecha",
                        "atras",        'izquierda']
        Meta: Probar Distintas clases implementadas
        Utiliza:  Raspberry pi 3B - raspbian stretch - python2.7/3
"""

from sensor import Sensor
import time

sensores = Sensor()

flag = True

while flag:
    time.sleep(1)
    mediciones = sensores.medirTodos()
    for sensor in mediciones:
        print("medicion sensor: ", sensor['ID']," -> ", sensor['
           distancia'],"cm")
    print("<<-->>")
```

7.2.1. Prueba objeto fijo distancia menor a 0.5m

En esta prueba se busca medir la distancia promedio a un objeto fijo ubicado a menos de 0.4m del sensor HC-SR04, ver figura 42, tabla completa de mediciones en anexo C.
La distancia 0.5m se considera distancia de seguridad mínima para el drone, ya que entre el sensor y helices del drone hay aproximadamente 0.35m. Se utiliza como objeto de prueba un cuadrado de carton de $0{,}09m^2$.

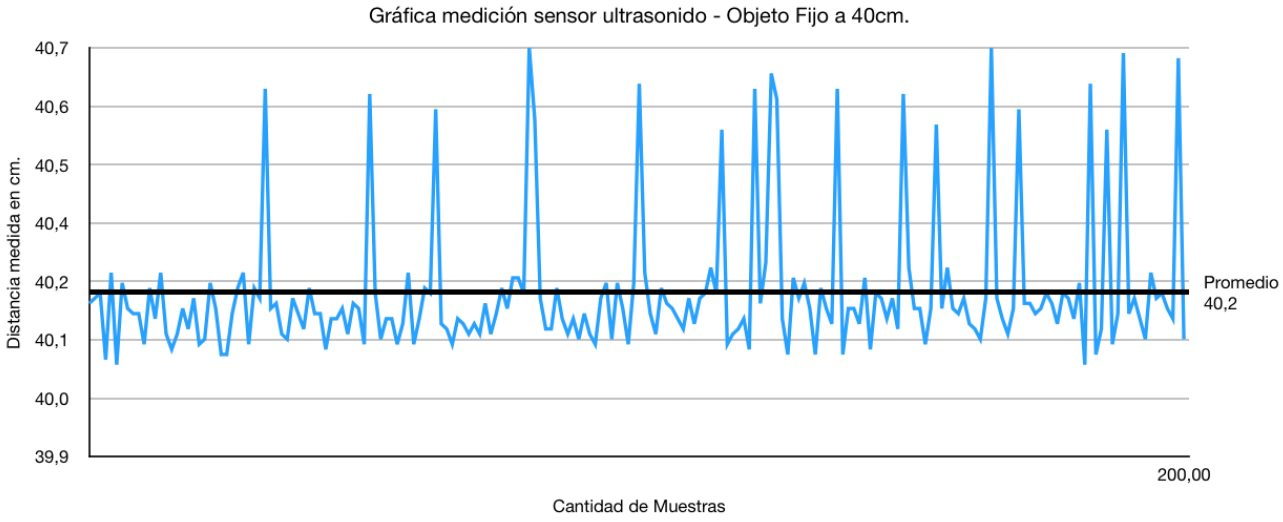


Figura 42: Gráfica medición sensor ultrasonido - Objeto Fijo a 0.4m

7.2.2. Prueba objeto fijo distancia entre 0.5m y 3m

En esta prueba se busca medir la distancia promedio a un objeto fijo ubicado a 2.14m del sensor HC-SR04, ver figura 43, tabla completa de mediciones en anexo C.
La distancias entre 0.5m y 3m se consideran distancias de reacción para el drone, ya que si existe un objeto dentro de ese rango de distancias el drone debe ser capaz de detectarlo e iniciar el protocolo de evasión. Para realizar la prueba se utiliza como objeto un cuadrado de carton de $0{,}09m^2$.

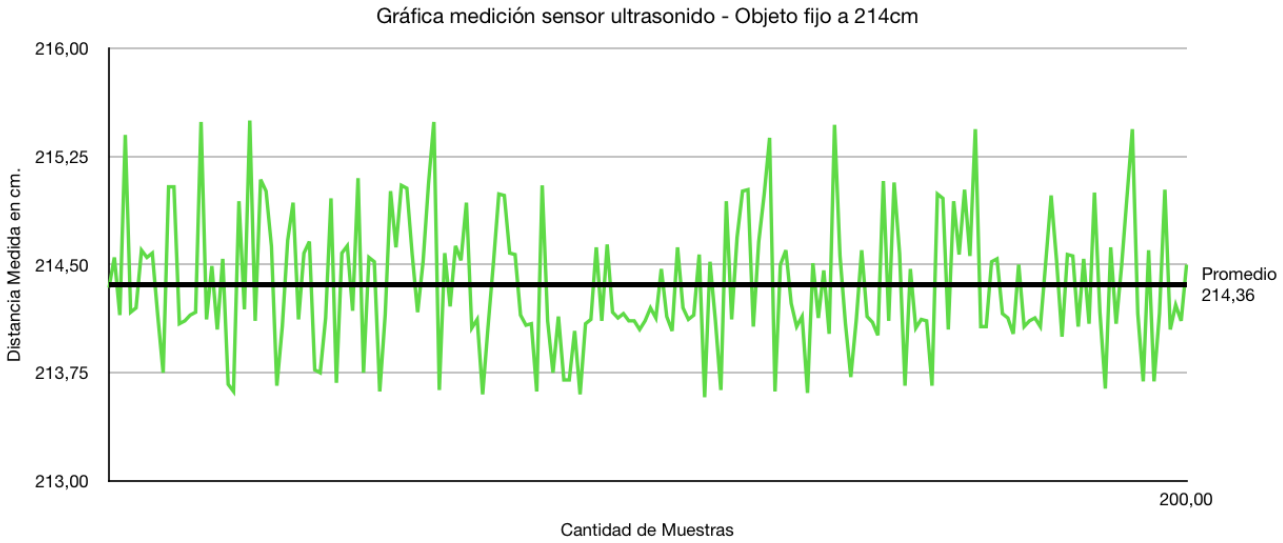


Figura 43: Gráfica medición sensor ultrasonido - Objeto Fijo a 2.14m

7.2.3. Prueba objeto fijo distancia mayor a 3m

En esta prueba se busca medir la distancia promedio a un objeto fijo ubicado a 5m del sensor HC-SR04, ver figura 44, tabla completa de mediciones en anexo C.
Una distancia mayor a 3m se considera distancia de seguridad máxima para el drone, ya que si existe un objeto mayor a ese limite, el sensor no es capaz de obtener una lectura precisa y por tanto se considera que no hay objetos dentro del horizonte de visión del drone. Para realizar la prueba se utiliza como objeto un cuadrado de carton de $0{,}09m^2$.

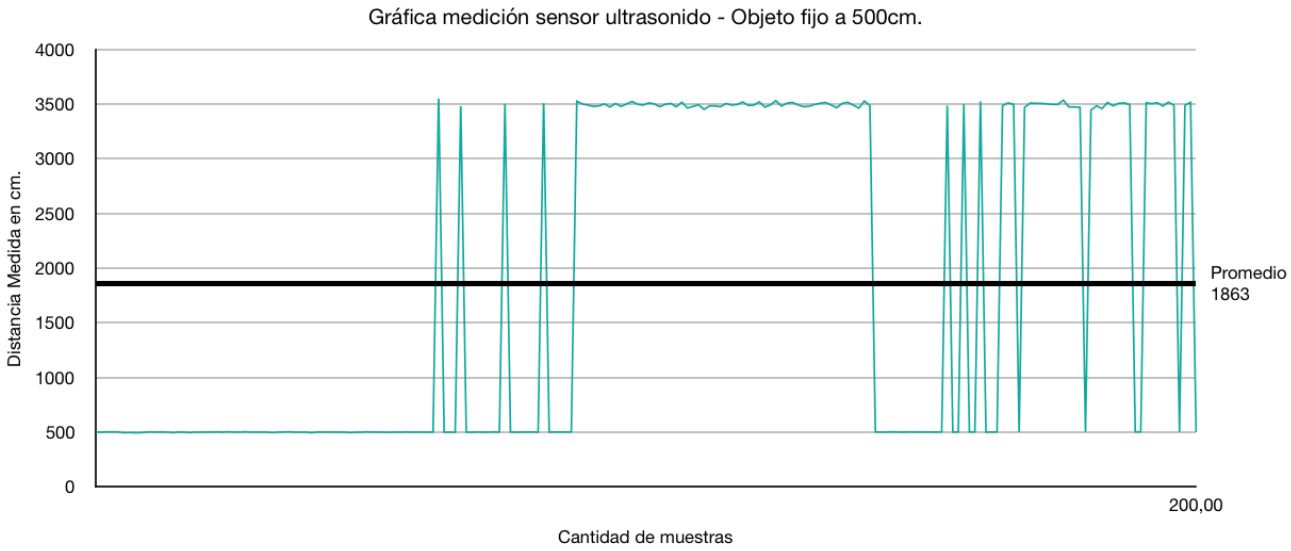


Figura 44: Gráfica medición sensor ultrasonido - Objeto Fijo a 5m

7.2.4. Prueba tiempo de medición para 15 muestras

En esta prueba se busca medir el tiempo que tardan las librerías descritas anteriormente en realizar 15 lecturas de distancia y calcular el promedio. Se consideran 15 muestras para calcular un promedio de distancia para reducir el ruido en las mediciones y obtener cifras con mayor precisión. Para realizar la prueba se utiliza como objeto un cuadrado de carton de 0,09m^2 ubicado a 0.4m del sensor.

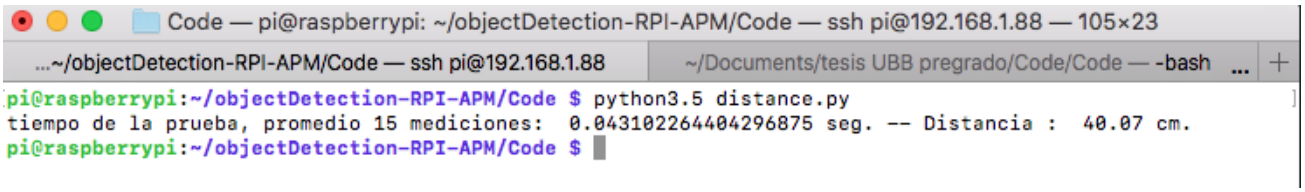


Figura 45: Captura terminal con ejecución de algoritmo 7

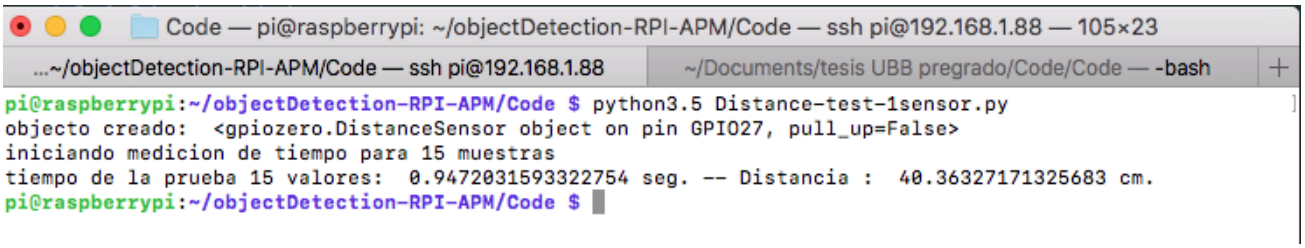


Figura 46: Captura terminal con ejecución de algoritmo 8

7.2.5. Prueba lectura todos los sensores, Clase Sensor

Esta prueba busca comprobar correcto funcionamiento de la Clase Sensor, y probar el método *medirTodos()*, obteniendo una lectura de los cinco sensores montados en el drone. De la prueba se obtiene una medición exitosa de los 5 sensores. Se puede observar que el sensor denominado *sensor_rigth*, correspondiente al sensor ubicado a la derecha del drone tiene una medición de $0cm$, esto indica que en este caso a la derecha del drone no hay obstáculos a menos de $3m$. En la figura 48 se puede ver una representación de la medición obtenida por los sensores.

```
Code — pi@raspberrypi: ~/objectDetection-RPI-APM/Code — ssh pi@192.168.1.85 — 100×37
...aspberrypi: ~/objectDetection-RPI-APM/Code — -bash ...  ...tDetection-RPI-APM/Code — ssh pi@192.168.1.85
pi@raspberrypi:~/objectDetection-RPI-APM/Code $ python3.5 main_test.py
objectos sensor creados
medicion sensor:  sensor_front  ->  28 cm
medicion sensor:  sensor_below  ->  29 cm
medicion sensor:  sensor_right  ->  0 cm
medicion sensor:  sensor_back  ->  227 cm
medicion sensor:  sensor_left  ->  188 cm
<<--->>
```

Figura 47: Captura terminal con ejecución de algoritmo 9.

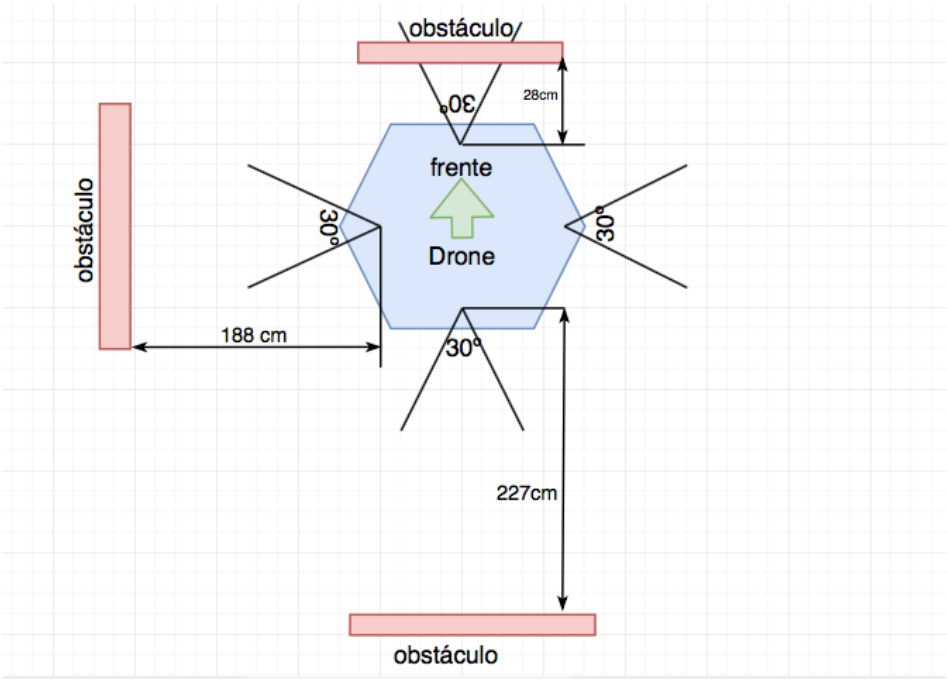


Figura 48: Figura representativa medición obtenida en ejecución de algoritmo 9.

7.3. Pruebas de Control

Se prueba correcto funcionamiento conjunto de las clases **ControlVuelo** y **Drone**. Esto se logra creando un objeto *drone* para luego verificar entradas de control enviadas desde el radio control, permitiendo activar el modo de vuelo asistido (inicia el hilo de la clase **EvadirOstaculos**) de forma manual cada vez que el operador lo requiera. También se arman los motores del drone de forma autónoma y se cambia de modo de vuelo verificando la capacidad del Raspberry Pi para enviar comandos de control vía MAVLink al APM y consultar datos de estado, tales como el nivel del canal *Throttle (3)*, ver algoritmo 10.

El resultado de esta prueba se observa en la imagen 49, se inicia la conexión RPi↔APM y se leen los valores de los canales *Throttle (canal 3)* y *Aux2 (canal 6)* que son enviados desde la emisora radio control. Al variar el valor del canal *Aux2* sobre 1500unidades (>= 50 %), se activa el modo de evasión de obstáculos, donde se inicia el hilo de la clase **EvasionObstaculo** y se procede al armado de motores por control de RPi.

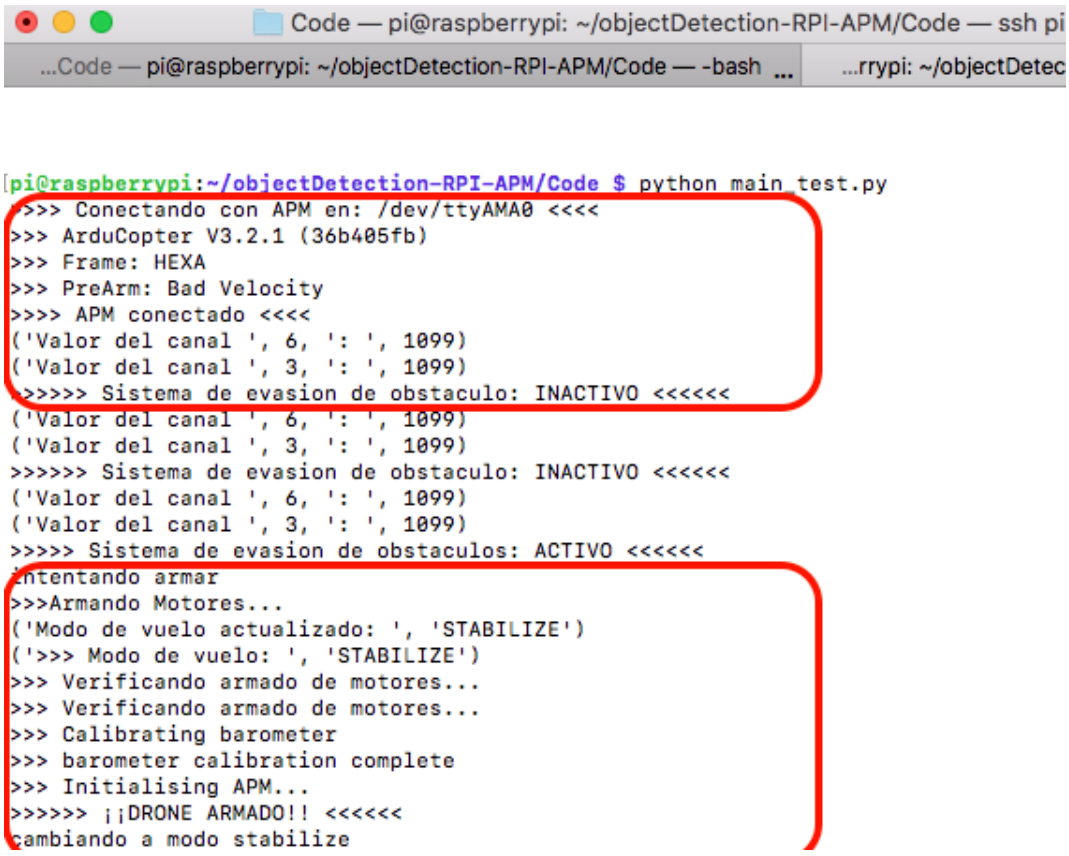


Figura 49: Captura terminal con ejecución de algoritmo 10. Parte 1 de 3.

En la imagen 50, el valor de *Throttle (canal 3)* aumenta, esto indica que desde la emisora se esta acelerando para ascender. Esta prueba se realiza sin hélices, por tanto el drone, a pesar acelerar no despega. Se observa que el canal *Aux2 (canal 6)* mantiene un valor fijo, y que si este varia nuevamente a un valor menor a 1500, el modo de evasión se desactiva, ver imagen 51

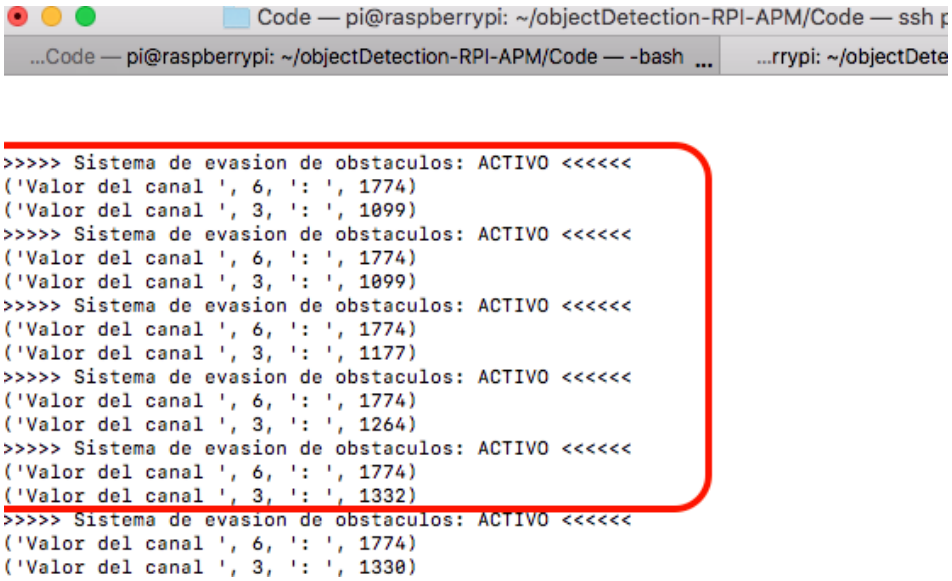


Figura 50: Captura terminal con ejecución de algoritmo 10. Parte 2 de 3.

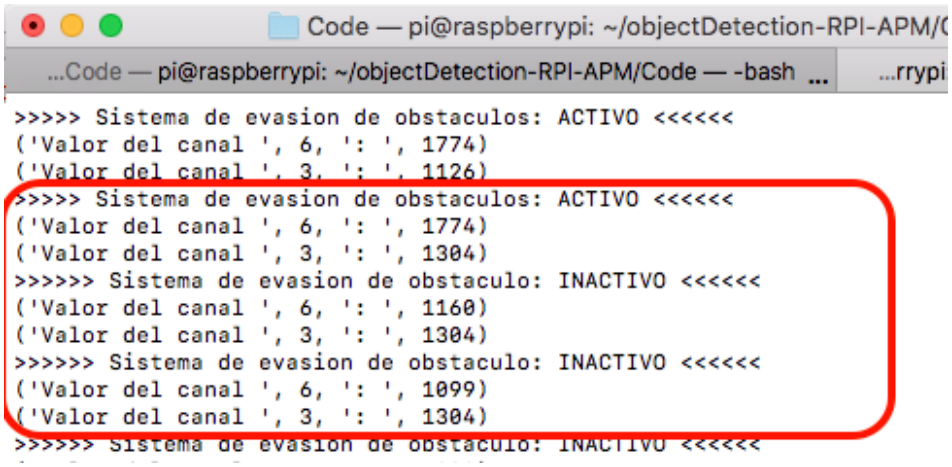


Figura 51: Captura terminal con ejecución de algoritmo 10. Parte 3 de 3.

Algoritmo 10 Implementación Python para prueba de Control - test_control.py.

```
from sensor import Sensor
from evasion_obstaculos import EvasionObstaculos
from drone import Drone
import time

connection_string = "/dev/ttyAMA0" #se utiliza UART0 pines GPIO14/15 Serial /dev/ttyAMA0
baud_rate = 115200 # establece velocidad maxima para APM2.6 = 115200 baudios

Drone = Drone(connection_string,baud_rate)
# EvasionObstaculos = EvasionObstaculos(drone)

while True:
        time.sleep(0.5)
        Drone.print_tx_channel(6) # Canal Aux2 (activa/desactiva Todo)
        Drone.print_tx_channel(3) #Canal Throttle!
        if Drone.get_channel_value(6) >= 1500:
                print(">>>>> Sistema de evasion de obstaculos: ACTIVO <<<<<<")
                if not Drone.get_is_armed():
                        print(">>>>> intentando armar")
                        Drone.apm_armar()
                        print(">>>>> cambiando a modo stabilize")
                        Drone.set_stabilize_mode()
                #obstacle_avoidance_system.start()
        elif Drone.get_channel_value(6) < 1400:
                print(">>>>>> Sistema de evasion de obstaculo: INACTIVO <<<<<<")
```

8. Observaciones y conclusiones

En el capitulo final de este trabajo de título se presentan las observaciones generales, conclusiones y proyección futura que se puede plantear a partir de esta memoria, haciendo énfasis en los objetivos específicos alcanzados durante todo el proceso de desarrollo.

8.1. Conclusiones

Frente a los resultados obtenidos de las distintas pruebas realizadas en función de cumplir con cada objetivo especifico de este proyecto, se puede concluir que un sistema de navegación asistida mediante detección y evasión de obstáculos para vehículos aéreos no tripulados no solo es aplicable y funcional, si no que necesario como característica permanente en este tipo de tecnologías. Como se ha presentado anteriormente, en el mercado ya existen algunos drones que poseen métodos de detección y evasión de obstáculos, por tanto, el desarrollo de mejoras a este tipo de sistemas como proyección de trabajo futuro es muy recomendable, mejorando las técnicas existentes para detección y desarrollando nuevos métodos para evaluación y replanteamiento autónomo de rutas.

La comunicación entre un dispositivo de control de drones basado en Arduino como ArduPilot Mega, y un Raspberry Pi sobre un canal serial con implementación del protocolo de comunicación MAVLink es altamente efectivo para dotar al drone de una estación de control a bordo de alto rendimiento, como se puede ver a lo largo de este proyecto, donde se establece un canal de comunicación bidireccional estable con una velocidad de comunicación de $115200 Baudios$.
Este tipo de comunicación sumado a intercomunicación inalámbrica entre los dispositivos de control a bordo de otros drones plantea opciones de desarrollo mucho mas complejas, permitiendo procesamiento de imágenes, aplicación de técnicas de Machine Learning, vuelo sincronizado basado en comportamientos de enjambre o técnicas similares, entre otras aplicaciones.

El proceso de captura de datos de entorno a través de los sensores de ultrasonido utilizados en este proyecto (HC-SR04) presentan un buen rendimiento en interiores y ambientes controlados, obteniendo mediciones muy precisas como se ve en las pruebas realizadas, pero en exteriores frente a obstáculos rugosos o dinamicos (como árboles) pierde precisión, por otro, lado este tipo de sensores es susceptible a las variaciones de temperatura, teniendo que considerarse estas caracteristicas dentro de los parámetros de uso. Es por esto que para desarrollo preliminar y prototipado es recomendable su uso, pero para aplicaciones que requieran fiabilidad en mayor variedad de situaciones, se recomienda la utilización de otro tipos de sensores como cámaras y sensores infrarrojos o un conjunto de estos como los sensores LIDAR, entre otros. Considerando también el mayor costo de implementación en comparación con los sensores utilizados.

Por tanto, para este proyecto resulta altamente ventajoso el uso de este tipo de tecnología de bajo costo y buen nivel de rendimiento en ambientes controlados.
También se concluye que el uso de la librería GPIOZero para medir distancias con los sensores presenta ventajas frente al uso de la librería instalada por defecto para Python en Raspbian RPI.GPIO, principalmente control de efecto cross-talking, ajuste de cola de lecturas para cálculo de media de distancia y ajuste de distancia máxima de medición como parámetros para constructor de objeto clase DistanceSensor y simplicidad de uso de la librería.

En conclusión, la convergencia de las etapas de establecimiento de comunicación serial RPi↔APM utilizando protocolo MAVLink, captura y medición de datos de entorno (sensores externos e internos del drone) y la implementación de algoritmos de evasión de obstáculos en lenguajes de alto nivel conforman el desarrollo completo de este proyecto, el cual, permitió una implementación de bajo costo de un sistema de evasión de obstáculos para un UAV. Esto permite incluso la aplicación a otro tipo de dispositivos que utilicen el mismo protocolo de comunicación con pequeñas variaciones como en el caso de *Rovers (vehículos terrestres)*, *Planes (aviones o alas)* o *Subs (submarinos)* basados en ArduPilot. De esta forma se cumplen los objetivos planteados considerando las mencionadas limitantes y se plantean las posibilidades de desarrollo que se pueden realizar sobre esta base, considerando el uso de plataformas de control a bordo de código libre como PixHawk, APM, Raspberry, entre otros.

8.2. Observaciones

Las diversas dificultades que acompañaron al proyecto en términos de hardware provocaron un retraso en el desarrollo de los algoritmos, principalmente debido a fallas en la batería del drone y el proceso de prueba y error en el montaje de sensores, para encontrar la mejor posición de lectura sin que sean afectados por las partes móviles del drone, también cuidando el orden en el conexionado y la distribución de energía y consumo del Raspberry Pi.

Por otro lado, se adquieren nuevos conocimientos en tecnologías embebidas, uso de computadores reducidos como RPi, la intercomunicación con otros dispositivos mediante protocolos de comunicación serial y uso de sensores de entorno.
Este proyecto presentó un gran desafío producto de la diversidad de tecnologías utilizadas, tanto a nivel hardware como software.
Elementos como las hélices, considerando su proceso de balanceo, son especialmente importantes a la hora de controlar las vibraciones del drone sumado a elementos como la base de absorción de vibraciones para el APM, esto para que los sensores internos del drone como el acelerómetro pueda tener lecturas con el menor ruido posible, también influye la distribución equitativa del peso de los componentes a bordo del drone para mejorar la estabilidad en vuelo.

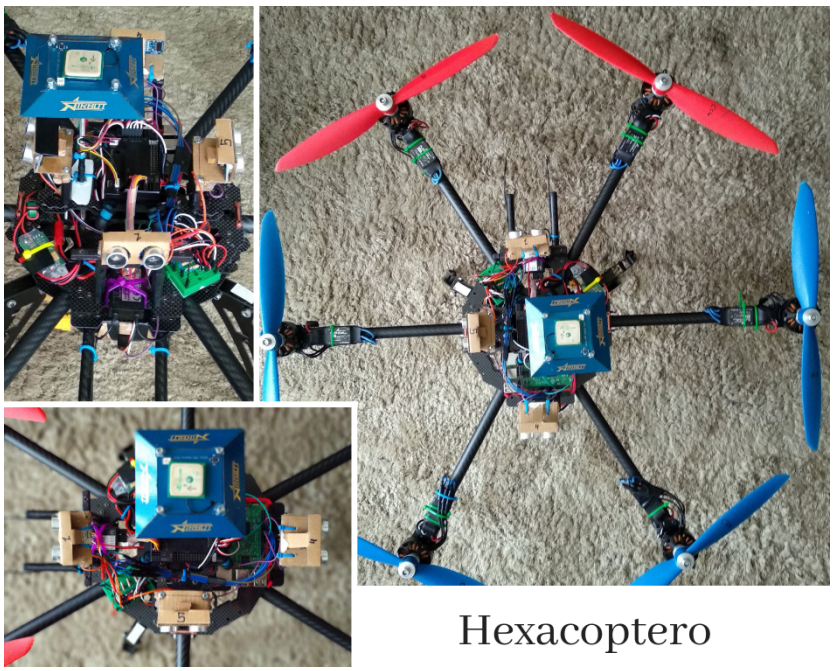


Figura 52: Drone vista general - desarrollo completo.

A. Diagramas de conexionado físico

Anexo para diagramas de conexión físico de los diversos componentes que se utilizan en el proyecto y distribución de GPIO RPi.

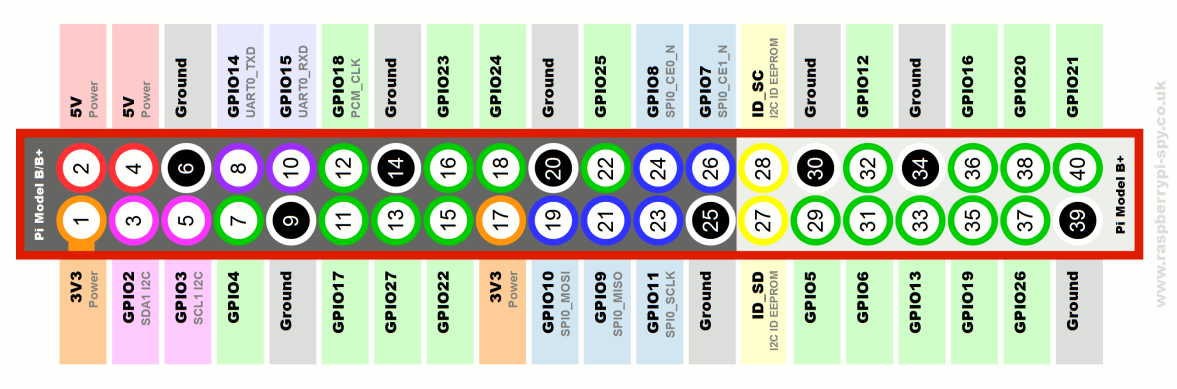


Figura 53: Numeración GPIO - RPi-3b [61].

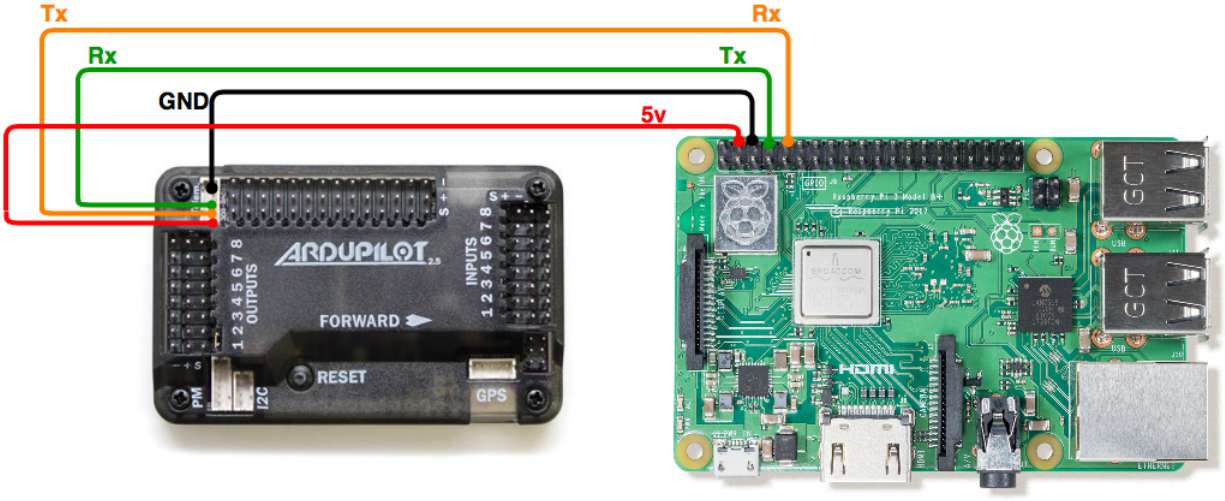


Figura 54: Esquema de conexionado fisico RPi↔APM .

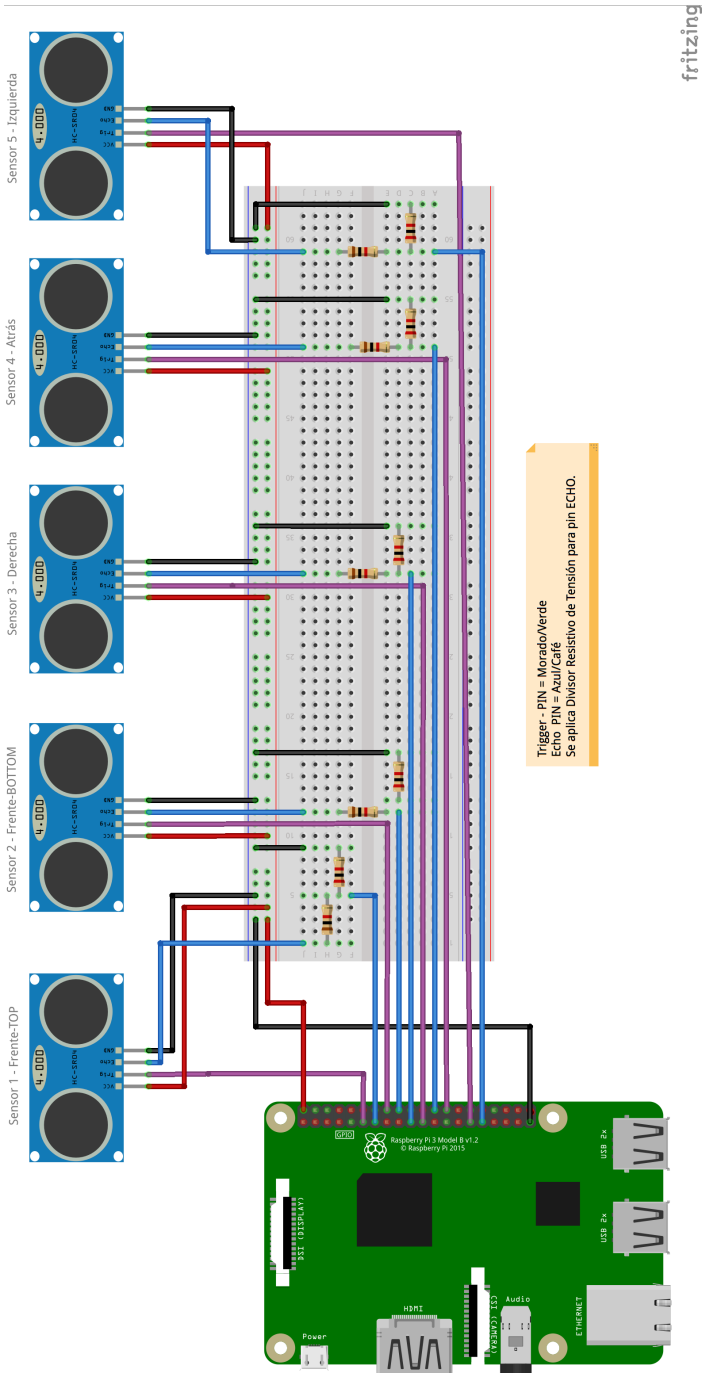
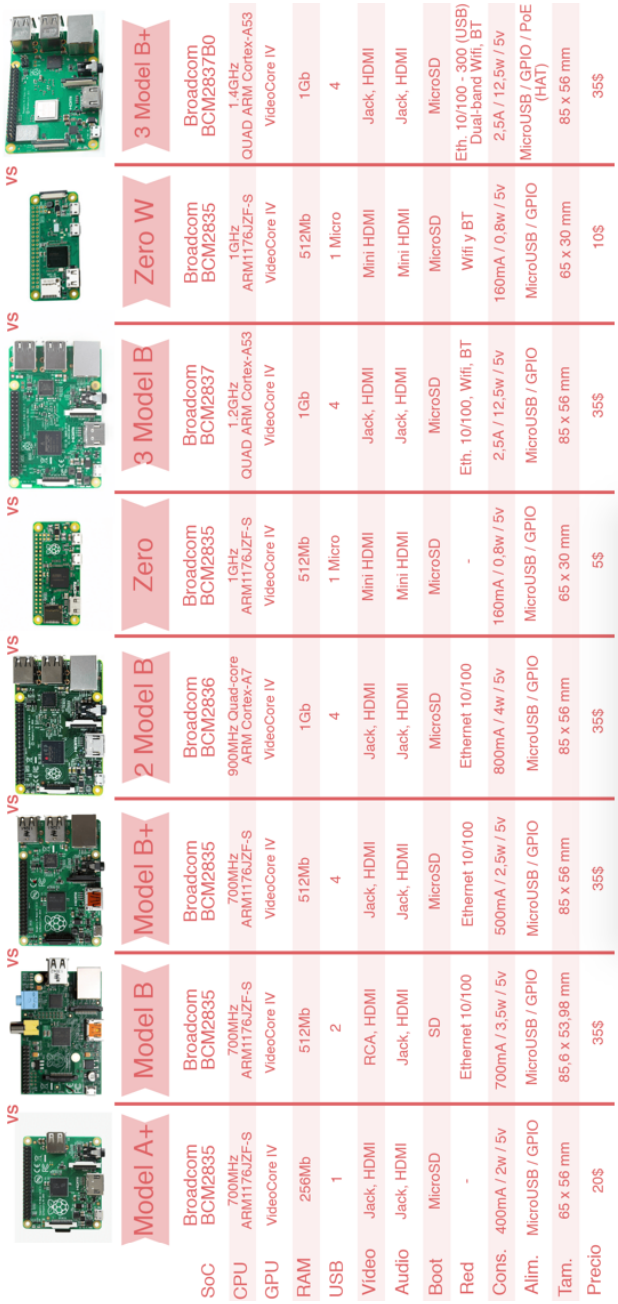


Figura 55: Esquema de conexionado físico RPi↔HC-SR04: Divisor de Tensión.

B. Comparativa - Raspberry Pi



	Model A+	Model B	Model B+	2 Model B	Zero	3 Model B	Zero W	3 Model B+
SoC	Broadcom BCM2835	Broadcom BCM2835	Broadcom BCM2835	Broadcom BCM2836	Broadcom BCM2835	Broadcom BCM2837	Broadcom BCM2835	Broadcom BCM2837B0
CPU	700MHz ARM1176JZF-S	700MHz ARM1176JZF-S	700MHz ARM1176JZF-S	900MHz Quad-core ARM Cortex-A7	1GHz ARM1176JZF-S	1.2GHz QUAD ARM Cortex-A53	1GHz ARM1176JZF-S	1.4GHz QUAD ARM Cortex-A53
GPU	VideoCore IV	VideoCore IV	VideoCore IV	VideoCore IV	VideoCore IV	VideoCore IV	VideoCore IV	VideoCore IV
RAM	256Mb	512Mb	512Mb	1Gb	512Mb	1Gb	512Mb	1Gb
USB	1	2	4	4	1 Micro	4	1 Micro	4
Video	Jack, HDMI	RCA, HDMI	Jack, HDMI	Jack, HDMI	Mini HDMI	Jack, HDMI	Mini HDMI	Jack, HDMI
Audio	Jack, HDMI	Jack, HDMI	Jack, HDMI	Jack, HDMI	Mini HDMI	Jack, HDMI	Mini HDMI	Jack, HDMI
Boot	MicroSD	SD	MicroSD	MicroSD	MicroSD	MicroSD	MicroSD	MicroSD
Red	-	Ethernet 10/100	Ethernet 10/100	Ethernet 10/100	-	Eth. 10/100, Wifi, BT	Wifi y BT	Eth. 10/100 - 300 (USB) Dual-band Wifi, BT
Cons.	400mA / 2w / 5v	700mA / 3,5w / 5v	500mA / 2,5w / 5v	800mA / 4w / 5v	160mA / 0,8w / 5v	2,5A / 12,5w / 5v	160mA / 0,8w / 5v	2,5A / 12,5w / 5v
Alim.	MicroUSB / GPIO	MicroUSB / GPIO	MicroUSB / GPIO	MicroUSB / GPIO	MicroUSB / GPIO	MicroUSB / GPIO	MicroUSB / GPIO	MicroUSB / GPIO / PoE (HAT)
Tam.	65 x 56 mm	85,6 x 53,98 mm	85 x 56 mm	85 x 56 mm	65 x 30 mm	85 x 56 mm	65 x 30 mm	85 x 56 mm
Precio	20$	35$	35$	35$	5$	35$	10$	35$

Figura 56: Comparativa histórica modelos RaspBerry Pi

C. Tabla 200 Muestras, Sensores HC-SR04 vs objeto 50x50cm

Muestras	Obj.40cm	Obj.25cm	Obj.300cm	obj.214cm	Obj.+-190cm	obj.500cm
1	40.2	25.65	3556.4	214.34	188.49	500
2	40.21	25.24	3553.33	214.55	188.71	500
3	40.22	25.59	3573.31	214.15	188.75	502
4	40.09	25.56	3526.84	215.4	189.16	502
5	40.26	25.7	3570.43	214.17	189.25	502
6	40.08	25.53	3575.0	214.2	189.18	494
7	40.24	25.69	3539.03	214.6	189.18	496
8	40.19	25.61	3554.72	214.55	189.16	494
9	40.18	25.69	3571.95	214.58	188.69	494
10	40.18	25.65	3560.27	214.15	189.64	500
11	40.12	25.53	3540.22	213.75	188.63	502
12	40.23	25.62	3533.42	215.04	188.65	500
13	40.17	25.63	3528.4	215.04	189.72	502
14	40.26	25.57	3560.26	214.09	188.71	500
15	40.14	25.59	3558.36	214.11	189.62	494
16	40.11	25.56	3521.2	214.15	188.64	501
17	40.14	25.67	3556.1	214.17	188.77	500
18	40.19	25.67	3584.49	215.49	188.7	495
19	40.15	25.52	329.07	214.12	188.69	500
20	40.21	25.64	3559.58	214.49	189.62	500
21	40.12	25.56	326.63	214.05	189.09	500
22	40.13	25.57	3564.24	214.54	189.6	501
23	40.24	25.63	300.34	213.67	190.08	502
24	40.19	25.67	326.2	213.62	189.17	500
25	40.1	25.6	3568.38	214.94	189.61	503
26	40.1	25.67	3560.96	214.19	189.58	501
27	40.18	25.57	3564.72	215.5	188.76	500
28	40.23	25.64	326.96	214.11	189.16	504
29	40.26	25.71	3569.62	215.09	189.11	500
30	40.12	25.65	3558.51	215.01	189.21	500
31	40.23	25.6	324.49	214.63	188.78	500
32	40.21	25.62	325.83	213.66	189.17	500
33	40.62	25.67	3591.07	214.07	189.64	495

Muestras	Obj.40cm	Obj.25cm	Obj.300cm	obj.214cm	Obj.+-190cm	obj.500cm
34	40.19	25.69	3563.61	214.67	188.65	500
35	40.2	25.65	326.68	214.93	189.25	501
36	40.14	25.67	3535.0	214.12	188.82	503
37	40.13	25.6	3550.19	214.58	188.7	500
38	40.21	25.64	3538.1	214.66	189.61	499
39	40.18	25.56	3597.51	213.77	188.78	500
40	40.15	25.63	327.84	213.75	189.18	494
41	40.23	25.64	3566.12	214.13	189.2	501
42	40.18	25.67	3589.16	214.96	189.66	500
43	40.18	25.63	325.77	213.68	188.81	501
44	40.11	25.58	3550.13	214.58	189.65	500
45	40.17	25.57	3558.59	214.63	189.69	500
46	40.17	25.65	326.59	214.18	188.66	500
47	40.19	25.76	3536.49	215.1	188.71	495
48	40.14	25.63	329.47	213.75	188.76	499
49	40.2	25.65	3544.51	214.55	189.65	499
50	40.19	25.61	3568.99	214.52	188.78	503
51	40.12	25.57	325.23	213.62	188.69	500
52	40.61	25.58	3555.22	214.15	188.71	501
53	40.22	25.61	3546.9	215.01	189.23	499
54	40.13	25.59	3567.05	214.62	189.27	499
55	40.17	25.68	298.99	215.05	189.16	500
56	40.17	25.73	3560.26	215.03	189.61	499
57	40.12	25.69	3535.0	214.57	189.63	502
58	40.16	25.67	3557.83	214.17	189.61	500
59	40.26	25.63	3546.37	214.52	189.63	499
60	40.12	25.59	3559.67	215.06	188.74	501
61	40.17	25.66	3533.78	215.49	189.24	499
62	40.23	25.59	3533.51	213.63	188.67	500
63	40.22	25.67	327.5	214.58	189.23	3549
64	40.58	25.55	301.58	214.21	190.14	500
65	40.16	25.63	3595.24	214.63	188.72	499
66	40.15	25.69	3583.16	214.53	188.72	499
67	40.12	25.7	325.75	214.93	189.68	3480

Muestras	Obj.40cm	Obj.25cm	Obj.300cm	obj.214cm	Obj.+-190cm	obj.500cm
68	40.17	25.61	3529.36	214.06	189.63	501
69	40.16	25.58	3563.25	214.12	188.73	499
70	40.14	25.64	3567.64	213.6	188.79	501
71	40.16	25.69	327.83	214.07	189.68	500
72	40.14	25.59	3560.49	214.51	188.74	500
73	40.2	25.55	326.16	214.99	189.63	502
74	40.14	25.65	326.21	214.98	189.18	501
75	40.18	25.61	328.3	214.58	189.65	3501
76	40.23	25.62	3549.57	214.57	189.11	501
77	40.19	25.67	327.5	214.15	188.72	499
78	40.25	25.61	3533.96	214.08	188.83	501
79	40.25	25.65	325.79	214.09	188.82	500
80	40.22	25.66	326.98	213.62	190.43	501
81	40.7	25.55	325.77	215.05	188.69	500
82	40.56	25.65	324.91	214.11	189.16	3506
83	40.21	25.67	328.3	213.75	190.46	500
84	40.15	25.65	3528.09	214.14	189.63	501
85	40.15	25.67	325.25	213.7	189.19	501
86	40.23	25.68	3560.44	213.7	188.64	501
87	40.17	25.68	3525.76	214.04	188.7	501
88	40.14	25.55	3567.54	213.6	189.61	3525
89	40.17	25.74	3532.37	214.09	189.56	3500
90	40.13	25.54	3567.33	214.12	189.16	3490
91	40.18	25.66	3538.7	214.62	188.78	3478
92	40.14	25.68	3569.47	214.11	188.74	3482
93	40.12	25.6	3568.67	214.64	188.8	3501
94	40.21	25.63	3555.09	214.17	188.66	3472
95	40.24	25.67	3549.31	214.13	189.67	3504
96	40.13	25.6	3597.11	214.16	189.62	3478
97	40.24	25.59	3563.13	214.11	188.67	3500
98	40.19	25.56	3566.76	214.11	188.71	3523
99	40.12	25.65	327.05	214.05	189.11	3497
100	40.24	25.63	327.05	214.11	188.79	3489
101	40.63	25.62	3566.1	214.2	188.7	3510

Muestras	Obj.40cm	Obj.25cm	Obj.300cm	obj.214cm	Obj.+-190cm	obj.500cm
102	40.26	25.67	3536.97	214.13	189.23	3501
103	40.18	25.6	3553.35	214.47	189.19	3475
104	40.14	25.6	3550.51	214.14	188.76	3497
105	40.23	25.64	3553.73	214.04	188.68	3505
106	40.2	25.55	3560.73	214.62	188.8	3474
107	40.19	25.67	3583.57	214.2	188.74	3516
108	40.17	25.63	3526.0	214.12	188.74	3463
109	40.15	25.67	3535.13	214.15	189.19	3477
110	40.21	25.61	3569.64	214.57	188.73	3493
111	40.16	25.75	3552.69	213.58	188.8	3450
112	40.21	25.59	3575.38	214.52	188.69	3482
113	40.22	25.58	3566.95	214.11	188.83	3482
114	40.27	25.57	3531.81	213.63	188.78	3475
115	40.22	25.55	329.13	214.94	189.23	3504
116	40.54	25.65	3558.27	214.12	188.71	3490
117	40.12	25.58	3524.46	214.69	189.2	3495
118	40.14	25.54	3549.33	215.01	189.21	3518
119	40.15	25.57	3555.91	215.02	189.72	3487
120	40.17	25.57	3556.83	214.07	189.22	3489
121	40.11	25.62	326.99	214.66	188.62	3519
122	40.62	25.62	3571.23	214.99	189.63	3471
123	40.2	25.63	3531.5	215.38	188.75	3491
124	40.28	25.67	327.05	213.62	189.22	3530
125	40.65	25.7	326.31	214.5	189.25	3481
126	40.6	25.61	3601.55	214.6	189.14	3507
127	40.17	25.59	3570.84	214.23	189.25	3514
128	40.1	25.64	3545.25	214.07	190.07	3492
129	40.25	25.56	3524.66	214.14	188.73	3475
130	40.21	25.6	3576.49	213.61	189.64	3479
131	40.24	25.64	3514.53	214.51	188.7	3495
132	40.19	25.55	3491.45	214.13	189.26	3506
133	40.1	25.61	3521.33	214.46	188.33	3514
134	40.23	25.66	3555.84	214.02	188.72	3490
135	40.19	25.6	3562.76	215.47	189.61	3466
136	40.16	25.65	327.5	214.56	189.14	3505
137	40.62	25.62	3558.83	214.09	188.68	3515

Muestras	Obj.40cm	Obj.25cm	Obj.300cm	obj.214cm	Obj.+-190cm	obj.500cm
138	40.1	25.55	3586.31	213.72	188.78	3490
139	40.19	25.63	3556.29	214.11	189.29	3462
140	40.19	25.54	3566.75	214.6	188.78	3527
141	40.16	25.63	3554.28	214.14	189.21	3486
142	40.25	25.59	3576.88	214.1	189.2	502
143	40.11	25.53	3551.52	214.01	188.74	500
144	40.22	25.66	3581.24	215.08	189.21	500
145	40.21	25.56	3570.39	214.11	189.14	503
146	40.17	25.57	3569.44	215.07	188.69	501
147	40.21	25.56	3582.41	214.58	189.19	500
148	40.15	25.61	327.85	213.66	190.08	502
149	40.61	25.65	325.77	214.47	189.27	501
150	40.27	25.62	324.86	214.06	188.64	501
151	40.19	25.6	3565.73	214.12	188.74	501
152	40.19	25.55	327.94	214.11	189.21	501
153	40.12	25.59	3538.73	213.66	189.19	500
154	40.19	25.6	326.16	214.99	189.1	502
155	40.55	25.61	3542.31	214.96	190.06	3485
156	40.19	25.66	3544.58	214.05	188.69	500
157	40.27	25.62	301.52	214.94	189.2	501
158	40.19	25.68	324.92	214.57	189.23	3498
159	40.18	25.67	325.86	215.02	189.2	501
160	40.21	25.6	3566.72	214.56	188.64	502
161	40.16	25.61	3550.34	215.44	188.31	3524
162	40.15	25.54	3555.46	214.07	188.71	500
163	40.13	25.56	3559.97	214.07	189.18	501
164	40.21	25.62	325.85	214.52	188.68	501
165	40.7	25.63	3580.45	214.54	188.76	3485
166	40.21	25.69	3552.2	214.16	188.78	3510
167	40.17	25.56	3558.51	214.13	189.24	3496
168	40.14	25.65	3542.79	214.02	188.77	501
169	40.19	25.53	3541.42	214.5	189.67	3469
170	40.58	25.21	3549.89	214.07	189.14	3508
171	40.2	25.64	3584.0	214.11	188.76	3506
172	40.2	25.64	3548.41	214.13	189.24	3505
173	40.18	25.62	3562.66	214.07	188.7	3501

Muestras	Obj.40cm	Obj.25cm	Obj.300cm	obj.214cm	Obj.+-190cm	obj.500cm
174	40.19	25.96	3569.83	214.52	188.79	3496
175	40.22	25.63	326.27	214.98	189.21	3496
176	40.2	25.64	3558.4	214.53	189.57	3535
177	40.16	25.66	3560.24	214.0	190.15	3473
178	40.22	25.64	3545.25	214.57	188.67	3472
179	40.21	25.65	3565.31	214.56	189.22	3470
180	40.17	25.59	3572.43	214.07	189.24	501
181	40.24	25.53	3552.78	214.54	188.7	3445
182	40.08	25.29	3557.8	214.09	188.7	3483
183	40.63	25.6	3568.79	215.0	189.17	3458
184	40.1	25.65	329.07	214.17	188.71	3514
185	40.15	25.58	327.86	213.64	189.13	3483
186	40.54	25.68	3561.5	214.62	188.75	3506
187	40.12	25.63	3547.95	214.09	189.1	3510
188	40.18	25.6	297.38	214.49	189.2	3494
189	40.69	25.61	3576.01	214.97	189.24	502
190	40.18	25.63	3572.23	215.44	189.23	502
191	40.21	25.56	300.24	214.16	188.8	3511
192	40.17	25.67	3565.1	213.69	189.25	3503
193	40.13	25.6	297.27	214.6	189.72	3512
194	40.26	25.61	3563.2	213.69	188.73	3481
195	40.21	25.67	327.09	214.16	188.75	3516
196	40.22	25.68	325.77	215.02	188.76	3490
197	40.19	25.64	3516.74	214.05	189.12	501
198	40.17	25.57	300.74	214.22	189.66	3488
199	40.68	25.67	325.79	214.11	189.65	3515
200	40.13	25.66	298.23	214.5	189.6	500
Promedio	40.22	25.60	2667.10	214.4	189.1	1865

D. Implementación Python Clase Sensor: sensor.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
Clase sensor, encargada de instanciar los 5 sensores HC-SR04 que se utilizan, y proveer los metodos de lectura para ser utilizados en la clase ObstacleAvoidance.py

*Intencionalmente no se utilizan tildes*
        Clase: Sensor.py
        Posicionamiento de los sensores en el drone:
                ["frente-arriba",       "frente-abajo", "derecha",
                        "atras",        'izquierda']
        Meta: Crear metodos de lectura de los distintos sensores
        Utiliza:  Raspberry pi 3B - raspbian stretch - python2.7/3
"""

from gpiozero import DistanceSensor
import time

class Sensor:

        # Constantes de Clase
        __MAX_DISTANCE = 3 # distancia maxima de lectura en metros (m)
        __THRESHOLD_DISTANCE = 0.5 # distancia de seguridad minima en metros (m)
        __QUEUE_LEN = 15 # cantidad de muestras para media de distancia
        # sensor 1 - Frente Arriba Adelante - Configuracion de pines BCM
        __sensor1 = {'ID':'frontSensor','TRIG': 17, 'ECHO': 27}
        # Sensor 2 - Frente abajo Abajo - configuracion de pines BCM
        __sensor2 = {'ID':'belowSensor','TRIG': 23, 'ECHO': 24}
        # sensor 3 - Derecha - Configuracion de pines BCM
        __sensor3 = {'ID':'rightSensor','TRIG': 9, 'ECHO': 10}
        # sensor 4 - Atras -  Configuracion de pines BCM
        __sensor4 = {'ID':'backSensor','TRIG': 7, 'ECHO': 8}
        # sensor 5 - Izquierda - Configuracion de pines BCM
        __sensor5 = {'ID':'leftSensor','TRIG': 5, 'ECHO': 6}

        __created = False

```

```
36          #-------------// instancia de objetos DistanceSensor
                //-----------------#
37          # formato object = DistanceSensor(echo, trigger, queue_len,
                max_distance (en metros), threshold_distance (en metros),
                partial, pin_factory)
38          def __init__(self):
39                  self.__sensor_front = DistanceSensor(echo=self.
                        __sensor1['ECHO'], trigger=self.__sensor1['TRIG'],
                        queue_len=self.__QUEUE_LEN, max_distance=self.
                        __MAX_DISTANCE, threshold_distance=self.
                        __THRESHOLD_DISTANCE)
40                  self.__sensor_below = DistanceSensor(echo=self.
                        __sensor2['ECHO'], trigger=self.__sensor2['TRIG'],
                        queue_len=self.__QUEUE_LEN, max_distance=4,
                        threshold_distance=self.__THRESHOLD_DISTANCE)
41                  self.__sensor_right = DistanceSensor(echo=self.
                        __sensor3['ECHO'], trigger=self.__sensor3['TRIG'],
                        queue_len=self.__QUEUE_LEN, max_distance=self.
                        __MAX_DISTANCE, threshold_distance=self.
                        __THRESHOLD_DISTANCE)
42                  self.__sensor_back = DistanceSensor(echo=self.__sensor4
                        ['ECHO'], trigger=self.__sensor4['TRIG'], queue_len=
                        self.__QUEUE_LEN, max_distance=self.__MAX_DISTANCE,
                        threshold_distance=self.__THRESHOLD_DISTANCE)
43                  self.__sensor_left = DistanceSensor(echo=self.__sensor5
                        ['ECHO'], trigger=self.__sensor5['TRIG'], queue_len=
                        self.__QUEUE_LEN, max_distance=self.__MAX_DISTANCE,
                        threshold_distance=self.__THRESHOLD_DISTANCE)
44                  self.__created = True
45                  print("objectos sensor creados")
46
47          def is_created(self):
48                  if self.__created is False:
49                          print("algo paso con la Clase Sensor.py")
50                          return False
51                  print("todo en orden, clase Sensor.py")
52                  return True
53
54          #-------------// Metodos de Clase //-----------------#
55          # se realizan las lecturas de cada sensor por separado y se
                devuelve la distancia medida
56
57          #retorna medida sensor frontal en cm.
```

```
58          def medirFrente(self):
59                  lectura_frente = round(self.__sensor_front.distance *
                        100)
60                  return lectura_frente
61
62          # retorna medida sensor trasero en cm
63          def medirAtras(self):
64                  lectura_atras = round(self.__sensor_back.distance *
                        100)
65                  return lectura_atras
66
67          # retorna medida sensor lateral izquierdo en cm
68          def medirIzquierda(self):
69                  lectura_izquierda = round(self.__sensor_left.distance *
                         100)
70                  return lectura_izquierda
71
72          # retorna medida sensor lateral derecho en cm
73          def medirDerecha(self):
74                  lectura_derecha = round(self.__sensor_right.distance *
                        100)
75                  return lectura_derecha
76
77          # retorna medida sensor adelante-abajo en cm
78          def medirAbajo(self):
79                  lectura_abajo = round(self.__sensor_below.distance *
                        100)
80                  return lectura_abajo
81
82          def medirTodos(self):
83                  mediciones = []
84                  mediciones.append({'ID' : 'sensor_front' , 'distancia'
                        : self.medirFrente()})
85                  mediciones.append({'ID' : 'sensor_below', 'distancia' :
                         self.medirAbajo()})
86                  mediciones.append({'ID' : 'sensor_right', 'distancia' :
                         self.medirDerecha()})
87                  mediciones.append({'ID' : 'sensor_back' , 'distancia' :
                         self.medirAtras()})
88                  mediciones.append({'ID' : 'sensor_left' , 'distancia' :
                         self.medirIzquierda()})
89                  return mediciones
```

E. Implementación Python Clase Drone: drone.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
"""
*Intencionalmente no se utilizan tildes*
    Clase Drone.py, para construcion de objeto Vehicle de Dronekit
    Posicionamiento de los sensores en el drone:
        ["frente-arriba", "frente-abajo", "derecha", "atras", "
           izquierda"]
    Meta: Establecer Conexion, leer y setear datos del APM desde RPi,
       control de movimientos del drone
    Utiliza: Raspberry pi 3B - raspbian stretch - python2.7/3
    Desarrollador: Jorge Lucas Torres Faundez - github.com/sirlucast -
       instagram.com/sirlucast
    Documentacion dronekit: https://github.com/dronekit/dronekit-python
"""


import math
import time

from dronekit import connect, VehicleMode, LocationGlobal,
    LocationGlobalRelative
from pymavlink import mavutil  # para definir mensajes protocolo
    MAVLink


class Drone:

    # Constantes de Clase

    __VEL_ASC_DEFECTO    = 0.7   #  VELOCIDAD de ascenso por defecto (
       >0.5 sube)
    __VEL_ASC_SUAVE      = 0.6   #  VELOCIDAD de ascencso suave
    __VEL_DESC_DEFECTO   = 0.3   #Â  VELOCIDAD de descenso por defecto
       (<0.5 baja)
    __VEL_DESC_SUAVE     = 0.4   #  VELOCIDAD de descencso suave

    __CANAL_THROTTLE     = '3'   #  stick throttle (Stick Left -
       vertical)
    __CANAL_YAW          = '4'   #  stick yaw (Stick Left - horizontal)
```

```
33      _CANAL_ROLL          = '1'   #   stick roll (Stick Right -
            horizontal)
34      _CANAL_PITCH         = '2'   #   stick pitch/roll (Stick Rigth -
            vertical)
35      _CANAL_FM            = '5'   #   switch modos de vuelo (FMD)
36      _CANAL_KNOV          = '6'   #   aux 2 (knov)
37
38      def __init__(self, connection_string, baud_rate):
39
40          print(">>>> Conectando con APM en: %s <<<<" %connection_string
                )
41          try:
42              self.apm = connect(connection_string, baud=baud_rate,
                    wait_ready=True, heartbeat_timeout=15)
43              print(">>>> APM conectado <<<<")
44
45          # Conexion Serial TTY error
46          except exceptions.OSError as e:
47              print("No se detecta conexion serial")
48
49          # Error del API
50          except dronekit.APIException:
51              print("Excepcion API, timeout")
52
53          # Otro Error
54          except:
55              print("Algun error desconocido u.u")
56
57
58
59      #-------------- Imprmir Estados del Drone! ---------------------#
60      #---------------------------------------------------------------#
61      # basado en ejemplo de dronekit: vehicle_state.py
62      # imprime "actitud" del drone, estados de sus angulos de navegacion
            y hacia donde esta orientado en grados con referencia al Norte
            = 0
63      def print_full_attitude(self):
64          print(">>>>> Estado de Pitch, Yaw, Roll y Heading (grados con
                referencia Norte = 0) <<<<<")
65          print("Pitch : ", round(self.apm.attitude.pitch,2))
66          print("Yaw : ", round(self.apm.attitude.yaw,2))
67          print("Roll : ", round(self.apm.attitude.roll,2))
68          print("Heading : ", round(self.apm.heading,2))
```

```
69          print(">>>>>> Fin angulos de navegacion <<<<<<")
70
71      # imprime la lista completa de parametros que se pueden consultar
            al APM
72      def print_list_of_parameters(self):
73          print(">>>>>> Lista Completa de Parametros en 'apm.parameters:
                <<<<<<")
74          for key, value in self.apm.parameters.iteritems():
75              print(" Key:%s \tValue:%s" %(key,value))
76          print(">>>>>> Fin listado de parametros <<<<<<")
77
78      # imprime la lista completa de los valores que se envian desde el
            RC Tx
79      def print_tx_all_channels(self):
80          print(">>>>>> Valores de todos los canales desde RC Tx (%s)
                <<<<<<" %len(self.apm.channels))
81          print("Valores de Canales RC Tx: ", self.apm.channels)
82
83      #imprime un el valor de un canal en especifico del RC Tx
84      def print_tx_channel(self, channel):
85          if type(channel) != int:
86              raise TypeError('Error de tipo de variable, debe ser un
                    entero (int) y se tiene un: ',type(channel))
87          elif channel < 1 or channel >= len(self.apm.channels):
88              raise ValueError("Los canales pueden pertenecer a [1,7], se
                     tiene: ", str(channel))
89          print("Valor del canal ", channel,": ",self.apm.channels[str(
                channel)] )
90
91
92      def print_full_info(self):
93          print(">>>>>> Obteniendo todos los datos (utiles a este trabajo
                ) de estado del Drone (Lista completa en documentacion
                DroneKit): <<<<<<")
94          print(">> Autopilot Firmware version: %s" %apm.version)
95          print(">> Release version: %s" %apm.version.release_version())
96          print(">> Supports onboard compass calibration: %s" %apm.
                capabilities.compass_calibration)
97          print(">> Supports set position + velocity targets in global
                scaled integers: %s" %apm.capabilities.
                set_altitude_target_global_int)
98          print(">> Last Heartbeat (seg.):%s" %self.apm.last_heartbeat)
99          print(">> Battery:  %s" %self.apm.battery)
```

```
100         print(">> GPS:  %s" % self.apm.gps_0)
101         print(">> Global Location:  %s" % self.apm.location.
                global_frame)
102         print(">> Global Location (relative altitude):  %s" % self.apm.
                location.global_relative_frame)
103         print(">> Local Location:  %s" % self.apm.location.local_frame)
104         print(">> Attitude (not rounded, in radians): %s" % self.apm.
                attitude)
105         print(">> Heading:  %s" % self.apm.heading)
106         print(">> Velocity:  %s" % self.apm.velocity)
107         print(">> Is Armable?:  %s" % self.apm.get_is_armable)
108         print(">> System state:  %s" % self.apm.system_status.state)
109         print(">> Groundspeed:  %s" % self.apm.groundspeed)  # settable
110         print(">> Airspeed: %s" % self.apm.airspeed)  # settable
111         print(">> Mode: %s" % self.apm.mode.name)  # settable
112         print(">> Armed:  %s" % self.apm.armed)  # settable
113         print(">>>>>> Fin Listado Completo de Parametros <<<<<<")
114
115     #-------------------- Configuraciones ! ---------------------#
116     #------------------------------------------------------------#
117     # basado en ejemplos de dronekit github. (vehicle_state.py y otros)
118
119     # Setear Modos de Vuelo
120
121     def set_guided_mode(self):
122         self.apm.mode = VehicleMode("GUIDED")
123         print("Modo de vuelo actualizado: ", self.get_mode())
124
125     def set_landing_mode(self):
126         self.apm.mode = VehicleMode("LAND")
127         print("Modo de vuelo actualizado: ", self.get_mode())
128
129     def set_stabilize_mode(self):
130         self.apm.mode = VehicleMode("STABILIZE")
131         print("Modo de vuelo actualizado: ", self.get_mode())
132
133     def set_rtl_mode(self):
134         self.apm.mode = VehicleMode("RTL")
135         print("Modo de vuelo actualizado: ", self.get_mode())
136
137     # hacia donde apunta el drone (con respecto al Norte, angulo
            absoluto 0-360).
138     def set_heading(self, heading):
```

```
139         self.mov_condition_yaw(heading)
140 
141     def set_groundspeed(self, velocidad):
142         self.apm.groundspeed= velocidad
143 
144     #funcion para sobreescribir canales de entrada desde RC Tx, ingresa
            numero de canal y variacion (positiva o negativa)
145     def set_override_channel(self, num_canal, variacion):
146         self.set_stabilize_mode() #cambiamos a modo de vuelo: "
                STABILIZE"
147         valor_actual = self.apm.channels[str(num_canal)]
148         # el nuevo valor no puede anular el valor del canal
149         print("Se sobre escribira el canal: ", num_canal)
150         if(valor_actual + variacion) < 0:
151             print("Error: ", valor, "es demasiado bajo" )
152             return
153         self.apm.channels.overrides = {str(num_canal):(valor_actual +
                variacion)}
154         self.apm.flush()
155         print("Sobre escritura del canal: ", num_canal,"Exitosa. Nuevo
                valor: ", (valor_actual+variacion))
156 
157     # se puede definir movimiento relativo al drone basandose en
            angulos de navegacion pitch, roll y yaw, durante "duration"
            segundos.
158     def set_attitude(roll_angle = 0.0, pitch_angle = 0.0, yaw_rate =
            0.0, thrust = 0.5, duration = 0):
159         # funcion en documentacion dronekit: https://github.com/
                dronekit/dronekit-python/blob/master/examples/
                set_attitude_target/set_attitude_target.py
160         """
161         Nota: desde el Firmware ardupilot AC3.3 los mensajes deben ser
                enviados cada segundo (despues de 3 segundos sin mensaje, la
                 velocidad relativa cae a 0). Para las versiones anteriores
                a AC3.2.1 (La que usa el APM2.6) la velocidad es persistente
                 hasta que es cancelada. (el codigo deberia funcionar para
                ambas versiones, reenviando el codigo
162         """
163         """
164         The roll and pitch rate cannot be controllbed with rate in
                radian in AC3.4.4 or earlier,
165         so you must use quaternion to control the pitch and roll for
                those vehicles.
```

```
166             """
167             # Thrust >  0.5: Ascender
168             # Thrust == 0.5: Mantener Altura
169             # Thrust <  0.5: Descender
170             msg = self.apm.message_factory.set_attitude_target_encode(
171                 0, # time_boot_ms
172                 1, # Target system
173                 1, # Target component
174                 0b00000000, # Type mask: bit 1 is LSB
175                 self.to_quaternion(roll_angle, pitch_angle), # Quaternion
176                 0, # Body roll rate in radian
177                 0, # Body pitch rate in radian
178                 math.radians(yaw_rate), # Body yaw rate in radian
179                 thrust  # Thrust
180             )
181             self.apm.send_mavlink(msg)
182             start = time.time()
183             while time.time() - start < duration:
184                 self.apm.send_mavlink(msg)
185                 time.sleep(0.1)
186 
187         # convierte angulos de navegacion en grados a un cuaternio y lo
                retorna
188         def to_quaternion(roll = 0.0, pitch = 0.0, yaw = 0.0):
189 
190             t0 = math.cos(math.radians(yaw * 0.5))
191             t1 = math.sin(math.radians(yaw * 0.5))
192             t2 = math.cos(math.radians(roll * 0.5))
193             t3 = math.sin(math.radians(roll * 0.5))
194             t4 = math.cos(math.radians(pitch * 0.5))
195             t5 = math.sin(math.radians(pitch * 0.5))
196 
197             w = t0 * t2 * t4 + t1 * t3 * t5
198             x = t0 * t3 * t4 - t1 * t2 * t5
199             y = t0 * t2 * t5 + t1 * t3 * t4
200             z = t1 * t2 * t4 - t0 * t3 * t5
201 
202             return [w, x, y, z]
203 
204 
205         #-------------- Obtener parametros/atributos ------------------#
206         #--------------------------------------------------------------#
207 
```

```
208
209     def get_system_status(self):
210         return self.apm.system_status.state
211
212     # si el prearmcheck esta ok, devuelve True
213     def get_is_armable(self):
214         return self.apm.is_armable
215
216     # retorna True si esta armado y False si no
217     def get_is_armed(self):
218         return self.apm.armed
219
220     #retorna el modo de vuelo actual
221     def get_mode(self):
222         return self.apm.mode.name
223
224     """
225     get GpsInfo:
226         parametros:
227         eph (Int) - GPS horizontal dilution of position (HDOP).
228         epv (Int) - GPS vertical dilution of position (VDOP).
229         fix_type (Int) - 0-1- no fix, 2: 2D fix, 3: 3D fix
230         satellites_visible (Int) - Number of satellites visible.
231
232     """
233     def get_gps_info(self):
234         return self.apm.gps_0
235
236     def get_channel_value(self, channel):
237         valor_canal = self.apm.channels[str(channel)]
238         return valor_canal
239
240     """
241     infor de location:
242
243     An object of this type is owned by Vehicle.location. See that class
        for information on reading and observing location in the global
       -relative frame.
244
245     Parameters:
246         lat - Latitude.
247         lon - Longitude.
248         alt - Altitude in meters (relative to the home location).
```

```
249
250     """
251     # posicion/ubicacion de despegue
252     def get_home_location(self):
253         return self.apm.home_location
254
255     # ubicacion del drone altitud nivel del mar
256     def get_global_location(self):
257         return self.apm.location.global_frame
258
259     # ubicacion del dorne, con altitud relativa al drone
260     def get_location_global_relative(self):
261         return self.apm.location.global_relative_frame
262
263     # ubicacion, solo latitud
264     def get_loc_latitude(self):
265         return self.apm.location.lat
266
267     # ubicacion, solo longitud
268     def get_loc_longitude(self):
269         return self.apm.location.lon
270
271     # altitud en metros, relativa a la altura en la posicion de
            despegue
272     def get_loc_altitude(self):
273         return self.apm.location.alt
274
275     # comportamiento del drone, en angulos de navegacion
276     def get_attitude(self):
277         return self.apm.attitude
278
279     # hacia donde apunta el drone, relativo al norte = 0. en grados
            0-360
280     def get_heading(self):
281         return self.apm.heading
282
283     # velocidad relativa del drone
284     def get_velocity(self):
285         return self.apm.velocity
286
287     # velocidad "en tierra" del drone
288     def get_groundspeed(self):
289         return self.apm.groundspeed
```

```
290
291     # nivel de tension de la bateria
292     def get_battery_voltage(self):
293         return self.apm.battery.voltage
294
295     #-------------- Proceso de Armado y Despegue! ------------------#
296     #---------------------------------------------------------------#
297     # basado en ejemplos de dronekit github. (simple_goto.py y otros)
298
299     # realiza un pre-arm check para verificar que el drone este listo
            para armar motores.
300     def apm_prearm_check(self):
301         while not self.get_is_armable():
302             print("Esperando que APM inicie...")
303             time.sleep(1)
304         print(">>>Â¿Es armable? ", self.get_is_armable())
305
306     def apm_armar(self):
307         # No se recomienda intentar armar sin hacer un pre-arm check
308         #self.apm_prearm_check()
309         print(">>>Armando Motores...")
310         # Por Defecto se Setea el modo de vuelo GUIDED para armado por
                Control de SW
311         self.set_guided_mode()
312         print(">>> Modo de vuelo: ", self.get_mode())
313         # verifica que el drone se haya armado.
314         while not self.apm.armed:
315             print(">>> Verificando armado de motores...")
316             time.sleep(1)
317             self.apm.armed = True
318         print(">>>>>> Â¡Â¡DRONE ARMADO!! <<<<<<")
```

F. Implementación Python Clase ControlVuelo: control_vuelo.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
*Intencionalmente no se utilizan tildes*
    Clase ControlVuelo.py, para manejo de APM con instrucciones de control
    Posicionamiento de los sensores en el drone:
        ["frente-arriba", "frente-abajo", "derecha", "atras", "izquierda"]
    Meta: enviar comandos de control al APM
    Utiliza: Raspberry pi 3B - raspbian stretch - python2.7/3
    Desarrollador: Jorge Lucas Torres Faundez - github.com/sirlucast - instagram.com/sirlucast
    Documentacion dronekit: https://github.com/dronekit/dronekit-python
"""

import math
import time
# Clase Drone.py
from drone import Drone

from dronekit import connect, VehicleMode, LocationGlobal, LocationGlobalRelative
from pymavlink import mavutil  # para definir mensajes protocolo MAVLink


class ControlVuelo:

    # Constantes de Clase

    __VEL_ASC_DEFECTO   = 0.7   #  VELOCIDAD de ascenso por defecto (>0.5 sube)
    __VEL_ASC_SUAVE     = 0.6   #  VELOCIDAD de ascencso suave
    __VEL_DESC_DEFECTO  = 0.3   #Â  VELOCIDAD de descenso por defecto (<0.5 baja)
    __VEL_DESC_SUAVE    = 0.4   #  VELOCIDAD de descencso suave

```

```
32      _CANAL_THROTTLE     = '3'   #   stick throttle (Stick Left -
            vertical)
33      _CANAL_YAW          = '4'   #   stick yaw (Stick Left - horizontal)
34      __CANAL_ROLL        = '1'   #   stick roll (Stick Right -
            horizontal)
35      __CANAL_PITCH       = '2'   #   stick pitch/roll (Stick Rigth -
            vertical)
36      _CANAL_FM           = '5'   #   switch modos de vuelo (FMD)
37      _CANAL_KNOV         = '6'   #   aux 2 (knov) / cambiar a switch
38
39      def __init(self, apm):
40          # Se comprueba instancia previa del objecto Drone: apm
41          if isinstance(apm, Drone) is False:
42              raise TypeError('Se espera objeto de tipo Drone, se tiene:
                    '+type(apm).__name__)
43          self.__apm = Drone(apm)
44
45      #------------ Proceso de Armado y Despegue! ------------------#
46      #-------------------------------------------------------------#
47      # basado en ejemplos de dronekit github. (simple_goto.py y otros)
48
49      # realiza un pre-arm check para verificar que el drone este listo
            para armar motores.
50      def apm_prearm_check(self):
51          while not self.__apm.is_armable:
52              print("Esperando que APM inicie...")
53              time.sleep(1)
54          print(">>>Â¿Es armable? ", self.__apm.is_armable)
55
56      def apm_armar(self):
57          # No se recomienda intentar armar sin hacer un pre-arm check
58          #self.__apm_prearm_check()
59          print(">>>Armando Motores...")
60          # Por Defecto se Setea el modo de vuelo GUIDED para armado por
                Control de SW
61          #self.__apm.mode = set_guided_mode()
62          printf(">>> Modo de vuelo: ", self.__apm.mode)
63          # verifica que el drone se haya armado.
64          while not self.__apm.armed:
65              print(">>> Verificando armado de motores...")
66              time.sleep(1)
67              self.__apm.armed = True
```

```
68          print(">>>>>> Â¡Â¡DRONE ARMADO!! <<<<<<")
69
70      """
71      # Funcion que permite armar motores y despegar hasta alcanzar una
            altura "alturaDespegue"
72      Esta funcion implementa control de altura con sensor ultrasonico
            que apunta hacia abajo hasta alcanzar los 4m, luego controla la
            altura con:
73      vehicle.location.global_relative_frame.alt
74      alturaDespegue es INT en metros, menor a 15 metros.
75      """
76      def mov_armar_despegar(self, alturaDespegue=1.8):
77          # comprobar tipo de dato y valor.
78          if type(alturaDespegue) != "float":
79              raise TypeError('Error de tipo de variable. Debe ser "float
                    " y se tiene un: ' + type(alturaDespegue))
80          elif alturaDespegue >= 15.0 or alturaDespegue <= 0.0:
81              raise ValueError("La altura maxima permitida para despegar
                    es 15 metros, altura : " + str(alturaDespegue))
82          # self.__apm_prearm_check()
83          printf(">>>>>>> Preparando para ARMAR y DESPEGAR, altura
                prevista: ", alturaDespegue)
84
85          self.__apm_armar() # armar
86          time.sleep(2)
87          self.__apm.simple_takeoff(alturaDespegue)  # Despegar...
88          """
89          #  Despues de despegar, se comprueba inmediatamente su altura
                hasta alcanzar la seteada.
90          Para altura menor a 4m se comprueba con el sensor frontal hacia
                abajo (belowSensor), luego con altura relativa por get del
                apm.
91          """
92          while True:
93              print (">>> Altitud : ", self.__apm.location.
                    global_relative_frame.alt)
94              # se comprueba llegar a la altura deseada con un 95% de
                    margen de error
95              if self.__apm.location.global_relative_frame.alt >= (
                    alturaDespegue*0.95):
96                  print (">>> Altitud alcanzada: ", self.__apm.location.
                        global_relative_frame.alt )
97                  print (">>> Altitud deseada: ", alturaDespegue )
```

```
98                  break
99              time.sleep(1)
100
101     #---------------- Movimientos del Drone ----------------------#
102     #-------------------------------------------------------------#
103     # basado en ejemplos de dronekit github. (simple_goto.py y otros)
104
105     # Enviar al Drone a posicion (de altitud relativa al Drone)
            determinada ej: LocationGlobalRelative(lat=-36.8206005,lon=
            -73.00901,alt=16.13) velocidad en m/s.
106     def mov_simple_goto(self, lat, lon, height, vel=-1):
107         # velocidad de vuelo en m/s.
108         if vel == -1:
109             self.__apm.airspeed = 1
110         else:
111             self.__apm.airspeed = vel
112
113         destino = LocationGlobalRelative(lat, lon, height)
114         self.__apm.simple_goto(destino)
115
116     """
117     !!!!! MOVIMIENTO UTILIZADO EN  CLASE DRONE.py, SETEA POR METODO
            PUBLICO (set_heading(slef, heading)) !!!!!
118     """
119     # basicamente es hacia donde "apunta" el drone y en angulo absoluto
            es con referencia al Norte. Relativo es con referencia al drone
120     def mov_condition_yaw(self, heading, relative=False):
121         """
122         Documentacion en https://github.com/dronekit/dronekit-python/
                blob/master/examples/guided_set_speed_yaw/
                guided_set_speed_yaw.py :
123
124         Convenience functions for sending immediate/guided mode
                commands to control the Copter.
125         The set of commands demonstrated here include:
126         * MAV_CMD_CONDITION_YAW - set direction of the front of the
                Copter (latitude, longitude)
127         * MAV_CMD_DO_SET_ROI - set direction where the camera gimbal is
                aimed (latitude, longitude, altitude)
128         * MAV_CMD_DO_CHANGE_SPEED - set target speed in metres/second.
129         The full set of available commands are listed here:
130         http://dev.ardupilot.com/wiki/copter-commands-in-guided-mode/
131         """
```

```
132         if relative:
133             is_relative = 1  # yaw relative to direction of travel
134         else:
135             is_relative = 0  # yaw is an absolute angle
136         # create the CONDITION_YAW command using command_long_encode()
137         msg = self.__apm.message_factory.command_long_encode(
138         0, 0,  # target system, target component
139         mavutil.mavlink.MAV_CMD_CONDITION_YAW,  # command
140         0,  # confirmation
141         heading,  # param 1, yaw in degrees
142         0,  # param 2, yaw speed deg/s
143         1,  # param 3, direction -1 ccw, 1 cw
144         is_relative,  # param 4, relative offset 1, absolute angle 0
145         0, 0, 0)  # param 5 ~ 7 not used
146 
147         # send command to vehicle
148         self.__apm.send_mavlink(msg)
149         self.__apm.flush()
150 
151     # reducir velocidad en funcion de la distancia que existe a un
        obstaculo
152     def apm_redudcir_vel(self, distancia):
153         # se reduce la velocidad considerando unos 10seg para llegar a
            esa distancia, asi da tiempo al drone de reaccionar
154         self.__apm.set_groundspeed(distancia/10)
155 
156     # MOVIMIENTOS POR SETEO DE ATTITUDE #
157     def mov_ir_arriba(self, altura):
158         # se comienza a subir con velocidad por defecto y cuando esta a
                punto de llegar, reduce la velocidad de ascenso
159         print(">>>>>> ASCENDIENDO <<<<<<")
160         thrust = self.__VEL_ASC_DEFECTO
161         while True:
162             # guardo altura actual (relativa al drone)
163             altitud_actual = self.__apm.location.global_relative_frame.
                alt
164             print(" Altura actual: %.2f || Altura esperada: %.2f" %(
                altitud_actual, altura))
165             if altitud_actual >= (altura * 0.95):
166                 # se ha alcanzado la altura margen +-5%
167                 print("se ha alcanzado la altura deseada")
168                 break
169             elif altitud_actual >= (altura * 0.6):
```

```
170                 # cuando se pasa el 60% de la altura deseada alcanzada,
                        se baja la velocidad.
171                 print("Alcanzando '60%' de latura deseada, se reduce
                        velocidad de ascenso")
172                 thrust = self.__VEL_ASC_SUAVE
173             set_attitude(thrust = thrust)
174             time.sleep(0.2)
175
176
177     def mov_ir_abajo(self, altura):
178         # se comienza a BAJAR con velocidad por defecto y cuando esta a
                punto de llegar, reduce la velocidad de ascenso
179         print(">>>>>> DESCENDIENDO <<<<<<")
180         thrust = self.__VEL_DESC_DEFECTO
181         altura_inicial = self.__apm.location.global_relative_frame.alt
182         if round(altura_inicial,2) == altura:
183             return
184         while True:
185             # guardo altura actual (relativa al drone)
186             altitud_actual = self.__apm.location.global_relative_frame.
                    alt
187             print(" Altura actual: %.2f || Altura esperada: %.2f" %(
                    altitud_actual, altura))
188             if 1 - (altitud_actual-altura)/(altitud_inicial-altura) >=
                    0.95:
189                 # se ha alcanzado la altura margen +-5%
190                 print("Se ha alcanzado la altura deseada")
191                 break
192             elif 1 - (altitud_actual-altura)/(altitud_inicial-altura) >
                    = 0.6:
193                 # cuando se recorre el 60% de la altura deseada
                        alcanzada, se baja la velocidad.
194                 print("Alcanzando '60%' de latura deseada, se reduce
                        velocidad de descenso")
195                 thrust = self.__VEL_DESC_SUAVE
196             set_attitude(thrust = thrust)
197             time.sleep(0.2)
198
199
200     def mov_ir_adelante(self, pitch_angle=-0.1 , duration=0):
201         print("Moviendo hacia el frente")
202         self.set_attitude(pitch_angle = pitch_angle, thrust =0.5,
                duration= duration)
```

```
203
204     def mov_ir_atras(self, pitch_angle=0.1, duration=0):
205         print("moviendo hacia atras")
206         self.set_attitude(pitch_angle = pitch_angle, thrust =0.5,
                duration= duration)
207
208
209     def mov_ir_derecha(self, roll_angle=0.1, duration=0):
210         print("moviendo a la derecha")
211         self.set_attitude(roll_angle = roll_angle, thrust = 0.5,
                duration = duration)
212
213     def mov_ir_izquierda(self, roll_angle=-0.1, duration=0):
214         print("moviendo a la izquierda")
215         self.set_attitude(roll_angle = roll_angle, thrust = 0.5,
                duration=0)
216
217
218     def mantener_altitud(self, duration=0):
219         # funcion que sirve tanto para mantener altura, como para
                detener movimiento
220         print("Detenido, Mantener Altura")
221         self.set_attitude(duration = duration)
222
223     def mov_aterrizar(self):
224         print("Aterrizando")
225         self.__apm.set_landing_mode()
226
227
228         # MOVIMIENTOS DE ACUERDO A VARIACION DE CANALES #
229     """
230     movimientos de roll, pitch, yaw y throttle por sobre escritura de
            entradas del RC Tx
231     """
232
233     #  ascender (Variacion positiva) -- descender (variacion negativa)
234     def mov_rc_throttle(self, variacion):
235         valor_actual = self.apm.channels[self._CANAL_THROTTLE]
236         # el nuevo valor no puede  hacer menor a 0 el valor del canal
237         print("Se sobre escribira el canal: ", self._CANAL_THROTTLE)
238
239         self.apm.channels.overrides = {self._CANAL_THROTTLE :
                variacion}
```

```
240         self.apm.flush()
241         print("Sobreescritura Exitosa!")
242         print("Canal Throttle: ", self.__CANAL_THROTTLE," : ", self.apm
                .channels[self.__CANAL_THROTTLE])
243
244     # Girar en CW (variacion positiva) -- CCW (variacion negatica).
            cambiando direccion del drone
245     def mov_rc_yaw(self, variacion):
246         valor_actual = self.apm.channels[self.__CANAL_YAW]
247         # el nuevo valor no puede  hacer menor a 0 el valor del canal
248         print("Se sobre escribira el canal: ", self.__CANAL_YAW)
249         self.apm.channels.overrides = {self.__CANAL_YAW : variacion}
250         self.apm.flush()
251         print("Sobreescritura Exitosa!")
252         print("Canal yaw: ", self.__CANAL_YAW," : ", self.apm.channels[
                self.__CANAL_YAW])
253
254     # Avanzar (variacion positiva) --Retroceder (variacion negativa)
255     def mov_rc_pitch(self, variacion):
256         valor_actual = self.apm.channels[self.__CANAL_PITCH]
257         # el nuevo valor no puede  hacer menor a 0 el valor del canal
258         print("Se sobre escribira el canal: ", self.__CANAL_PITCH)
259         self.apm.channels.overrides = {self.__CANAL_PITCH : variacion}
260         self.apm.flush()
261         print("Sobreescritura Exitosa!")
262         print("Canal yaw: ", self.__CANAL_PITCH," : ", self.apm.
                channels[self.__CANAL_PITCH])
263
264     # desplazar derecha (variacion positiva) -- desplazar izquierda (
            variacion negativa)
265     def mov_rc_roll(self, variacion):
266         valor_actual = self.apm.channels[self.__CANAL_ROLL]
267         # el nuevo valor no puede  hacer menor a 0 el valor del canal
268         print("Se sobre escribira el canal: ", self.__CANAL_ROLL)
269         self.apm.channels.overrides = {self.__CANAL_ROLL : variacion}
270         self.apm.flush()
271         print("Sobreescritura Exitosa!")
272         print("Canal yaw: ", self.__CANAL_ROLL," : ", self.apm.channels
                [self.__CANAL_ROLL])
```

G. Implementación Python Clase EvasionObstaculos: evasion_obstaculos.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
Clase EvasionObstaculos, aqui estan los algoritmos de evacion de obstaculos, acercamiento para evasion por retroceso
*Intencionalmente no se utilizan tildes*
        Clase: EvasionObstaculos.py
        Posicionamiento de los sensores en el drone:
                ["frente-arriba", "frente-abajo", "derecha", "atras", 'izquierda']
        Meta: Evadir obstaculos detectados
        Utiliza: Raspberry pi 3B - raspbian stretch - python2.7/3
"""
import time
import math
import threading

# Clase Sensor.py
from sensor import Sensor
# Clase Drone.py
from drone import Drone
# Clase ControlVuelo.py
from control_vuelo import ControlVuelo

class EvasionObstaculos(threading.Thread):

        # Constantes de Clase

        __SISTEMA_ACTIVO = False
        # Direccion de movimiento
        __IR_ARRIBA = 'up'
        __IR_IZQ = 'left'
        __IR_DER = 'right'
        __IR_ATRAS = 'back'
        __IR_ABAJO = 'down'

        # Distancias de Control
        __LIMITE_LEJANO = 300 # Distancia maxima de control, debajo de esa distancia el obstaculo es detectado. Distancia en cm.
```

```
37          __LIMITE_CERCANO = 150 # Distancia minima de control, el drone
                deberia evadir el obstaculo sin acercase mas. Distancia en
                cm.
38          __DISTANCIA_PELIGRO = 50 #Distancia de peligro, el drone esta a
                menos de 15cm de CHOCAR con el obstaculo. Medida en cm.
39          __SUBIDA_MAXIMA = 200 # distancia maxima que el drone sube para
                intentar evadir un objeto por arriba
40          __BUSQUEDA_LATERAL = 200 # Distancia maxima en la que el drone
                se mueve a izq/der para evadir obstaculo frontal, antes de
                intentar pasarlo por arriba. Medida en cm.
41          __ALTURA_TEST = 200 # altura a la que el drone se mantiene
                constante para vuelo a baja altura, eliminar esta
                restriccion mas adelante.
42          MARGEN_HORIZONTAL_DRONE = 100 # en cm. el espacio que necesita
                el drone para moverse lateralmente (es el "ancho" del drone)
43
44          __NO_HAY_OBSTACULOS = 0 # cuando la lectura de los sensores es
                0, se considera que no hay obstaculos frente al sensor.
45          __VARIACION_RC = 300 #Los canales a sobrescribir (pitch y roll)
                solo se pueden variar en +-300 desde la posicion central,
                es decir, 1200<->1500<->1800
46
47          def __init__(self, drone):
48
49                  threading.Thread.__init__(self)
50                  self.stoprequest = threading.Event()
51                  # is_active indica que la evasion de obstaculos esta "
                        corriendo" por tanto la clase puede enviar comandos
                        que sobreescriban los canales del RC.
52                  self.__is_active = False
53
54                  self.drone = drone
55                   # se instancian los sensores de medicion de distancia
                        HC_SR04 y se comprueba que esten creados, se
                        realiza una lectura general de los 5 sensores.
56                  self.__sensor = Sensor()
57                  if self.__sensor.is_created() is False:
58                          raise ConnectionError('NO se crearon los
                                sensores')
59                  else:
60                          mediciones = self.__sensor.medirTodos()
61                          for medicion in mediciones:
```

```
62                                  print("Distancia sensor %s: %d" %(
                                        medicion['ID'], medicion['distancia'
                                        ]))
63                          print("Sensores... OK")
64
65                  ## verificar que el objeto de la clase Drone haya sido
                        instanciado
66                  if isinstance(drone, Drone) is False:
67                          raise TypeError('Objeto esperado de tipo Drone,
                                se tiene: '+type(drone).__name__)
68
69                  self.__comandos_vuelo = ControlVuelo() #verificar en el
                        UML (esto tiene pinta de agregacion debil...)
70
71                  # Si existen obstaculos demasiado cerca u ocurre algÃºn
                        tipo de error, el drone debe activar aterrizaje de
                        emergencia y este flag debe ser True.
72                  self.__emergencia = False
73
74                  # distancias recorridas por el drone, sirve para
                        comparar con el limite _BUSQUEDA_LATERAL y
                        __SUBIDA_MAXIMA
75                  self.__distancia_der = 0.0
76                  self.__distancia_izq = 0.0
77                  self.__distancia_subida = 0.0
78
79                  # datos de ubicacion del drone
80                  self.__last_latitude = drone.get_loc_latitude()
81                  self.__last_longitude = drone.get_loc_longitude()
82                  self.__last_height = drone.get_loc_altitude()
83
84                  self.__last_pitch_override = 0
85                  self.__last_roll_overrride = 0
86
87          def __evaluar_pitch(self, d_frontal, d_atras):
88                  diferencia = d_frontal - d_atras
89                  if abs(diferencia) > self.MARGEN_HORIZONTAL_DRONE:
90                          if self.__is_to_close(d_frontal):
91                                  if self.__is_to_close(d_atras):
92                                          if d_frontal < d_atras: #
                                                distancia frontal menor,
                                                deberia mover hacia atras (
```

```
                                        aumentar pitch >1500,
                                        inverso)
93                                          pitch = self.
                                                __ajuste_rc(
                                                d_frontal, True)
94                                          return pitch
95                                      else: # por el contrario se
                                            mueve hacia adelante (pitch
                                            disminuye <1500, inverso)
96                                          pitch = self.
                                                __ajuste_rc(d_atras,
                                                 False)
97                                          return pitch
98                                  else: # no hay obstaculos dentro del
                                        limite cercano hacia atras, mover
                                        hacia atras
99                                      pitch = self.__ajuste_rc(
                                            d_frontal, True)
100                                     return pitch
101                             else: # no hay obstaculos hacia adelante, se
                                    evalua hacia atras.
102                                 if self.__is_to_close(d_atras): #hay
                                        obstaculos atras
103                                     pitch = self.__ajuste_rc(
                                            d_atras, False)
104                                     return pitch
105                                 else: # hay espacio para moverse
106                                     pitch = None
107                                     return pitch
108                         elif(d_frontal == self.__NO_HAY_OBSTACULOS and self.
                                __is_to_close(d_atras)):#se comprueba si alguno de
                                los dos sensores esta con valor 0.
109                             pitch = self.__ajuste_rc(d_atras, False)
110                             return pitch
111                         elif(self.__is_active(d_frontal) and d_atras == self.
                                __NO_HAY_OBSTACULOS):
112                             pitch = self.__ajuste_rc(d_frontal, True)
113                             return pitch
114                         else: # ambos son 0, por tanto hay espacio para moverse
115                             pitch = None
116                             return pitch
117
118
```

```
119         def __evaluar_roll(self, d_der, d_izq):
120                 diferencia = d_der - d_izq
121                 if abs(diferencia) > self.MARGEN_HORIZONTAL_DRONE:
122                         if self.__is_to_close(d_der):
123                                 if self.__is_to_close(d_izq):
124                                         if d_der < d_izq: #distancia a
                                            la derecha menor, deberia
                                            mover hacia izq (disminuir
                                            roll <1500, directo)
125                                                 roll = self.__ajuste_rc
                                                    (d_der, False)
126                                                 return roll
127                                         else: # por el contrario se
                                            mueve hacia la derecha (roll
                                             aumenta  > 1500, directo)
128                                                 roll = self.__ajuste_rc
                                                    (d_izq, True)
129                                                 return roll
130                                 else: # no hay obstaculos dentro del
                                    limite cercano hacia izquierda,
                                    mover hacia izquierda
131                                         roll = self.__ajuste_rc(d_der,
                                            False)
132                                         return roll
133                         else: # no hay obstaculos hacia derecha, se
                            evalua hacia izquierda.
134                                 if self.__is_to_close(d_izq): #hay
                                    obstaculos a la izquierda?, mover a
                                    la derecha
135                                         roll = self.__ajuste_rc(d_izq,
                                            True)
136                                         return roll
137                                 else: # hay espacio para moverse
138                                         roll = None
139                                         return roll # limpiamos el
                                            overrride
140                 elif(d_der == self.__NO_HAY_OBSTACULOS and self.
                    __is_to_close(d_izq)):#se comprueba si alguno de los
                     dos sensores esta con valor 0.
141                         roll = self.__ajuste_rc(d_izq, True)
142                         return roll
143                 elif(self.__is_active(d_der) and d_izq == self.
                    __NO_HAY_OBSTACULOS):
```

```
144                     roll = self.__ajuste_rc(d_der, False)
145                     return roll
146             else: # ambos son 0, por tanto hay espacio para moverse
147                     roll = None
148                     return roll
149
150
151         #Â ajusta de forma proporcional la variacion del canal RC con
               respecto a la distancia del obstaculo, mientras mas cerca,
               mas brusco es el movimiento en sentido contrario
152         def __ajuste_rc(self, distancia, sentido):
153                 if sentido:
154                         valor_rc = 1500 + (self.__LIMITE_CERCANO -
                               distancia) * 3 # (self.__VARIACION_RC/(self.
                               __LIMITE_CERCNO - self.__DISTANCIA_PELIGRO))
                                = 300/100
155                 else:
156                         valor_rc = 1500 - (self.__LIMITE_CERCANO -
                               distancia) * 3 # (self.__VARIACION_RC/(self.
                               __LIMITE_CERCNO - self.__DISTANCIA_PELIGRO))
                                = 300/100
157                 print("valor de ajuste_rc: ", valor_rc)
158                 valor_rc_rounded = round(valor_rc,0)
159                 return valor_rc_rounded
160
161         # evalua si un obstaculo esta dentro del limite cercano para
               reaccionar
162         def __is_to_close(self, distancia):
163                 if distancia != self.__NO_HAY_OBSTACULOS and distancia
                       <= self.__LIMITE_CERCANO:
164                         return True
165                 else:
166                         return False
167
168
169         def __protocolo_emergencia(self):
170                 # esta funcion se llama, cuando no se pueden evadir
                       obstaculos (despues de cubrir todas las opciones) o
                       cuando hay obstaculos demasiado cerca del drone (<50
                       cm) y se debe aterrizar de emergencia.
171                 self.__comandos_vuelo.mov_aterrizar()
172                 self.is_active = False
```

Referencias

[1] Martínez-Carranza, J., Valentin, L., Márquez-Aquino, F., González-Islas, J. C., & Loewen, N., "Detección de obstáculos durante vuelo autónomo de drones utilizando SLAM monocular". *Research in Computing Science*, vol. 114, pp. 111-124, 2016.

[2] Dirección General de Aeronáutica Civil, DGAC. Como operar un drone en Chile [Online]. Disponible en: https://www.dgac.gob.cl/como-operar-un-dron-en-chile/

[3] Dronecode Project, Inc. (2018), a Linux Foundation Collaborative Project [Online] https://www.dronecode.org

[4] Somerville, I., *Ingeniería de software*. Madrid: Pearson Educación, 2005.

[5] Rodríguez Martín, E. (2015). Sistema de posicionamiento para un drone. [Online] Disponible en : https://riull.ull.es/xmlui/bitstream/handle/915/1137/Sistema%20de%20posicionamiento%20para%20un%20drone.pdf?sequence=1&isAllowed=y

[6] McLain, T., Saunders, J., Barber, B., Beard, R. W., & Griffiths, S. R. "Obstacle and Terrain Avoidance for Miniature Aerial Vehicles". En *Advances in Unmanned Aerial Vehicles.* Springer, Dordrecht, 2007. p. 213-244. [Online] Disponible en: https://scholarsarchive.byu.edu/cgi/viewcontent.cgi?article=2927&context=facpub

[7] Heng, L., Meier, L., Tanskanen, P., Fraundorfer, F., & Pollefeys, M. "Autonomous obstacle avoidance and maneuvering on a vision-guided MAV using on-board processing". In *Robotics and automation* (ICRA), May 2011, IEEE international conference on (pp. 2472-2477). IEEE.

[8] Kadouf, H. H. A., & Mustafah, Y. M. "Colour-based object detection and tracking for autonomous quadrotor UAV". In *IOP Conference Series: Materials Science and Engineering* (Vol. 53, No. 1, p. 012086). IOP Publishing, 2013.

[9] Meier, L., Tanskanen, P., Fraundorfer, F., & Pollefeys, M. "The PIXHAWK open-source computer vision framework for MAVs". *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 38(1), C22, 2011.

[10] DJI Store (2018). Phantom 4 Pro. [Tienda Online] Disponible en: https://www.djistore.cl/shop/phantom-4-pro/200449-drone-dji-phantom-4-pro-50-reserva.html

[11] ATMEL Corporation, ATMEGA2560 Datasheet (PDF). [Data Sheet] Disponible en: http://www.alldatasheet.com/datasheet-pdf/pdf/107092/ATMEL/ATMEGA2560.html

[12] Ublox AG, (2011), NEO-6, u-blox 6 GPS Modules. [Data Sheet] Disponible en: https://www.u-blox.com/sites/default/files/products/documents/NEO-6_DataSheet_%28GPS.G6-HW-09005%29.pdf?utm_source=en%2Fimages%2Fdownloads%2FProduct_Docs%2FNEO-6_DataSheet_%28GPS.G6-HW-09005%29.pdf

[13] Hobbywing, User Manual Of PLATINUM PRO Series Brushless Speed Controller. [User Manual] Disponible en: http://cdn.shopify.com/s/files/1/0109/9702/files/Users_Manual_Platinum.pdf?1713

[14] Ardupilot.org. (2017), Common power module. [Online] Disponible en: http://ardupilot.org/copter/docs/common-3dr-power-module.html

[15] Henge, User Manual of Henge 6A UBEC. [Users Manual] Disponible en: http://airtekhobbies.com/downloads/hengeubec.pdf

[16] Walkera Inc., Users Manual of DEVO-7 transmitter. [Users Manual] Disponible en: https://cdn.instructables.com/ORIG/FXB/3MTH/IRXSYQO4/FXB3MTHIRXSYQO4.pdf

[17] ELEC Freaks. Ultrasonic Ranging Module HC - SR04. [Online] Disponible en: https://www.mouser.com/ds/2/813/HCSR04-1022824.pdf

[18] BEN Nuttal (2015-2018). Revision 5085b9cc. GPIOZERO. [Online] Disponible en: https://gpiozero.readthedocs.io/en/stable/

[19] Dronecode Project, Inc., a Linux Foundation Collaborative Project (2018). MAVLink Micro Air Vehicle Developer guide. [Online] Disponible en: https://mavlink.io/en/

[20] pixhawk.org. PyMAVLink Tools. [Online] Disponible en: https://pixhawk.org/dev/pymavlink

[21] Erle Robotics S.L.. (2018), Erle Robotics GitBook. [Online] Disponible en: https://erlerobotics.gitbooks.io/erlerobot/content/es/

[22] 3D Robotics. (2016), DroneKit-Python Documentation. [Online] Disponible en: http://python.dronekit.io

[23] LaTex Project. [Online] Disponible en: https://www.latex-project.org

[24] Open Source Robotics Foundation, ROS Documentation. [Online] Disponible en: http://wiki.ros.org

[25] Miguel A.Muñoz, 1.2 PRINCIPIOS AERODINAMICOS. [Online] Disponible en: http://www.manualvuelo.com/PBV/PBV12.html

[26] CIDEAD. Para saber más, Efecto Venturi. [Online] Disponible en: http://recursostic.educacion.es/secundaria/edad/4esotecnologia/quincena9/4q9_sabermas_1d.htm

[27] Sepúlveda, E. (2016, julio). Tercera ley de Newton. [Online] Disponible en: Física en Línea: http://www.fisicaenlinea.com/06fuerzas/fuerzas23-terceraleynewton.html

[28] Bernal, J., Flowers-Cano, R., & Carbajal-Dominguez, A. "Exact calculation of the number of degrees of freedom of a rigid body composed of n particles". En *Revista Mexicana de Física*, 55(2), 191-195, 2009.

[29] Academic, (2000-2017). Grados de Libertad (Física). [Online] Disponible en: http://www.esacademic.com/dic.nsf/eswiki/539396#sel=

[30] National Instruments Corporation. (2014), Comunicación Serial: Conceptos Generales. [Online] Disponible en: http://digital.ni.com/public.nsf/allkb/039001258CEF8FB686256E0F005888D1

[31] Divya c, Grace John, Jerrin Yomas. "Serial Communication Interface with Error Detection". *IOSR Journal of Electronics and Communication Engineering* (IOSR-JECE) e-ISSN: 2278-2834,p- ISSN: 2278-8735.Volume 10, Issue 6, Ver. I, pp 73-76, Nov - Dec 2015.

[32] ARM Limited. (2005). PrimeCell® UART (PL011) Revision: r1p4 Technical Reference Manual. [Online] Disponible en: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0183f/DDI0183.pdf.

[33] Dronecode Project, Inc., a Linux Foundation Collaborative Project. (2017), MAVLink Developer Guide. [GitBook] Disponible en: https://mavlink.io/en/

[34] International Telecommunication Union,"Interface between Data Terminal Equipment (DTE) and Data Circuit-terminating Equipment (DTE)," ITU-T Recommendation X.25, October 1996. [Online] Disponible en: https://www.itu.int/rec/T-REC-X.25-199610-I/es

[35] SAE International, (2009), SAE AS5669A. SAE web: http://www.sae.org [Online] Disponible en: http://avcs-au.com/library/files/jaus/as5669a.pdf

[36] Javier M. Mellid. (2017). Open Source UAV API, DroneKit-Python and Geopy. [Online] Disponible en: https://javiermunhoz.com/blog/2017/10/06/open-source-uav-api-dronekit-python-and-geopy.html

[37] Song Han, William Shen, Zuozhen Liu. (2016). Deep Drone: Object Detection and Tracking for Smart Drones on Embedded System [Online] Disponible en: https://web.stanford.edu/class/cs231a/prev_projects_2016/deep-drone-object__2_.pdf

[38] Raspberry Pi Foundation (2018). GPIO. [Online] Disponible en: https://www.raspberrypi.org/documentation/usage/gpio/README.md

[39] Richard B. Langley (1999). "Dilution of Precision". [Online] Disponible en: http://gauss.gge.unb.ca/papers.pdf/gpsworld.may99.pdf

[40] Ardupilot.org. (2017), Community: - ArduPilot documentation. [Online] Disponible en: http://ardupilot.org/ardupilot/index.html

[41] Susnea, Ioan; Minzu, Viorel; Vasiliu, Grigore. "Simple, real-time obstacle avoidance algorithm for mobile robots',' In 8th WSEAS International Conference on Computational Intelligence, Man-Machine Systems and Cybernetics (CIMMACS?09), 2009, pp. 24-29.

[42] MAVLink, MAVLink Micro Air Vehicle Protocol. [Online] Disponible en: https://raw.githubusercontent.com/mavlink/mavlink/master/message_definitions/v1.0/common.xml

[43] Dirección General de Aeronáutica Civil, DGAC. RPAs registrados en Chile [Online]. Disponible en: http://www.dgac.gob.cl/wp-content/uploads/2017/08/RPAS_Registrados.pdf

[44] Brandisky, G. & Kaehlet, A., *Learning OpenCV*, California: O'Reilly Media, 2008

[45] Toloza J., "Algoritmos y técnicas de tiempo real para el incremento de la precisión posicional relativa usando receptores GPS estándar", tesis doctoral, Universidad de La Plata, 2010.

[46] Scherer, S., Singh, S., Chamberlain, L., & Saripalli, S. (2007, April). "Flying fast and low among obstacles. In Robotics and Automation", *2007 IEEE International Conference* on (pp. 2023-2029), IEEE.

[47] Balasubramanian, Shyam. MavLink Tutorial for Absolute Dummies (Part I). [Online] Disponible en: http://api.ning.com/files/i*tFWQTF2R*7Mmw7hksAU-u9IABKNDO9apguOiSOCfvi2znk1tXhur0Bt00jTOldFvob-Sczg3*lDcgChG26QaHZpzEcISM5/MAVLINK_FOR_DUMMIESPart1_v.1.1.pdf

[48] Meier, Lorenz. MAVLink Micro Air Vehicle Protocol. [Online] Disponible en: https://github.com/mavlink

[49] Puig, Rodolfo. (2014), IMU and MCU Selection. [Online] Disponible en: https://github.com/rogueminds/ardutendo-meta/wiki/IMU-and-MCU-Selection

[50] Drone Spain (2014-2018). Ilustración multirrotores-700x335. [Figura] Recuperado de: http://dronespain.pro/tipos-de-drones-aereos/

[51] DJI (2017). Ilustración modo Narrow Sensing. [Figura] Recuperado de: https://www.dji.com/phantom-4-pro

[52] Ardupilot.org. (2017), Imagen APM 2.5/2.6. [Figura] Disponible en: http://ardupilot.org/copter/docs/common-apm25-and-26-overview.html#common-apm25-and-26-overview

[53] Hobbywing, Imagen de Platinum 30A OPTO ESC. [Figura] Disponible en: https://www.hobbywingdirect.com/products/platinum-30a-opto-esc

[54] Raspberry Pi Foundation. Imagen Raspberry-Pi-3-1-1619x1080. [Figura] Disponible en: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[55] qgroundcontrol.org, MAVLink-packet. [Figura] Disponible en: http://qgroundcontrol.org/mavlink/start

[56] wiki de Robótica de la Universidad Politécnica de Valencia (UPV). Imagen GP2D12. [Figura] Disponible en: http://wiki.robotica.webs.upv.es/wiki-de-robotica/sensores/sensores-proximidad/sensor-infrarrojos/

[57] Raspberry Pi Foundation. Imagen Pi-Camera-hero-1-1394x1080. [Figura] Disponible en: https://www.raspberrypi.org/products/camera-module-v2/

[58] Academic, (2000-2017). Taitbriangles. [Figura] Disponible en: http://www.esacademic.com/pictures/eswiki/84/Taitbrianangles.svg

[59] PX4 Dev Team. License: CC BY 4.0. basic_movements_multicopter.png [Figura] Disponible en: https://docs.px4.io/en/flying/basic_flying.html

[60] Wiki de Robótica. tof_onda. [Figura] Disponible en: http://wiki.robotica.webs.upv.es/wp-content/uploads/2015/08/tof_onda.png

[61] Matt Hawkins, (2018). Raspberry-Pi-GPIO-Layout-Model-B-Plus-rotated-2700x900-2 [Figura] Disponible en: https://www.raspberrypi-spy.co.uk/2012/06/simple-guide-to-the-rpi-gpio-header-and-pins/

[62] Biblioteca de la Universidad Pública de Navarra. Oficina de Referencia. ?Guía para citar y referenciar. IEEE Style?, 2016. [En línea]. Disponible en: https://goo.gl/LaUj46.