Instituto Tecnológico de Costa Rica

Área académica de Ingeniería Mecatrónica

Diseño y desarrollo de una estructura de software-hardware para el hospedaje de rutinas de comportamiento autónomo en vehículos aéreos no tripulados

Informe de Proyecto de Graduación para optar por el título de Ingeniero en Mecatrónica con el grado académico de Licenciatura

Pablo Andrés Araya Castillo

Carné: 2014098435

Cartago, marzo de 2019

Instituto Tecnológico de Costa Rica

Área Académica de Ingeniería en Mecatrónica

Proyecto de Graduación

Tribunal Evaluador

Acta de Evaluación

Proyecto de Graduación defendido ante el presente Tribunal Evaluador como requisito para optar por el título de Ingeniero en Mecatrónica con el grado académico de Licenciatura del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal

M. Sc. Jaime Mora Meléndez
Profesor Lector

M. Sc. Yeiner Arias Esquivel
Profesor Lector

Dr. Ing. Juan Luis Crespo Mariño
Profesor Asesor

Los Miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por el Área Académica de Ingeniería Mecatrónica.

Cartago, 1 de Marzo de 2019.

Declaro que el presente Proyecto de Graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado bibliografía, he procedido a indicar las fuentes mediante las respectivas citas bibliográficas.

En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

Pablo Andrés Araya Castillo
Cédula: 7-0245-0407
Cartago, 1 de Marzo de 2019.

Resumen

El Grupo Integrado de Ingeniería (GII) es un grupo de I+D de la Universidad de La Coruña que posee una línea de investigación de vehículos aéreos no tripulados (UAV, del inglés *unmanned aerial vehicle*). Han trabajado en múltiples aplicaciones con UAVs usando el autopiloto Pixhawk. Los vehículos reciben instrucciones de un mando remoto a la vez que envían información del estado de vuelo a una computadora de control en tierra (CCT) para visualizar lo que está pasando durante la ejecución de una misión.

Sin embargo, los UAVs eran incapaces de ejecutar tareas que involucraran comportamiento autónomo, además de que la comunicación con la CCT no era suficientemente fluída para visualizar el vuelo correctamente.

Para solucionar el problema se implementó una estructura funcionalmente flexible de hardware y software en un UAV, que incluyó una mini PC a bordo en la cual se pueden programar rutinas de vuelo tanto autónomas como no autónomas, que le dan instrucciones al Pixhawk directamente. Además en una CCT se visualiza el estado de vuelo con una comunicación más fluída y las decisiones que está tomando el vehículo.

Se verificó el correcto funcionamiento de la estructura a través de simulaciones y se validó con un vuelo de prueba autónomo, en el cual el UAV decidió y ejecutó su propia ruta de acuerdo a las condiciones de la misión programada.

Palabras clave: UAV, autónomo, Pixhawk, mini PC, CCT, puerta de enlace

Abstract

The Integrated Group for Engineering Research (GII) is an R & D group from the University of A Coruña that has a research line of unmanned aerial vehicles (UAV). They have worked on multiple applications with UAVs using the Pixhawk autopilot. The vehicles receive instructions from a remote control while sends flight status information to a ground control station (GCS), a computer to visualize what is happening during the execution of a mission.

However, the UAVs were unable to perform tasks that involved autonomous behavior, and the communication with the GCS was not fluid enough to visualize the flight correctly.

To solve the problem, a functionally flexible hardware and software structure was implemented in a UAV, which included a mini PC on board in which autonomous and non-autonomous flight routines can be programmed, which give the Pixhawk instructions directly. In addition, a GCS displays the flight status with more fluid communication and the decisions taken by the vehicle.

The correct performance of the structure was verified through simulations and validated with an autonomous test flight, in which the UAV decided and executed its own route according to the conditions of the programmed mission.

Keywords: UAV, autonomous, Pixhawk, mini PC, GCS, gateway

Dedicatoria

A mi familia, especialmente a mis padres por todo lo que me han enseñado y dado, sin su apoyo no sería quien soy hoy.

Agradecimientos

A todos los que hicieron posible mi estancia en España para el desarrollo de mi proyecto de graduación, empezando por el Dr. Ing. Juan Luis Crespo Mariño y miembros de la Universidad de La Coruña que me permitieron hacer la pasantía en el Grupo Integrado de Ingeniería. A mi familia y al Instituto Tecnológico de Costa Rica por brindarme los recursos necesarios para el viaje.

A los colaboradores del GII por guiarme y ayudarme en todo lo que necesité para desarrollar el proyecto, especialmente a los ingenieros Félix Orjales Saavedra y Javier Losada Pita.

Índice General

Índice de figuras

Índice de tablas

Capítulo 1. Introducción

Los UAV (*Unmanned Aerial Vehicles*) o vehículos aéreos no tripulados han alcanzado una gran popularidad en los últimos años debido a su gran potencial de uso, su flexibilidad y el abaratamiento y mejora de los sistemas de control que usan. Al igual que con otros vehículos, los UAV constituyen una herramienta susceptible de mejorar mediante la inclusión de inteligencia y autonomía.

Con esta idea en mente surge la línea de investigación de UAV en el Grupo Integrado de Ingeniería (GII) de la Universidad de La Coruña, España, como parte de la línea de robótica autónoma; se trabaja tanto en dotar de mayor capacidad de realización de tareas de forma independiente así como en la coordinación de múltiples UAV para la consecución de tareas de forma colaborativa, para lo cual cuenta con múltiples UAV funcionales, tanto de ala fija como multicópteros equipados con autopilotos de código abierto, basados en el hardware Pixhawk 2.

En su sitio web el GII se define de la siguiente manera: “Somos un grupo interdisciplinar de investigación aplicada en ingeniería orientado a la transferencia de conocimiento y a la generación de nuevos productos en el entorno industrial” [1]. Está formado por un conjunto interdisciplinar de profesionales (mayoritariamente de ingeniería), conformando un equipo estable con gran capacidad y experiencia.

1.1 Problema existente

Los UAV poseen una serie de actuadores (motores) que hacen girar hélices que permiten el vuelo. Sin embargo, necesitan de una estructura de hardware-software que permita recibir instrucciones de movimiento y pueda gestionar los actuadores para cumplir la misión: esta es la función de lo que se conoce como un autopiloto, un dispositivo con múltiples entradas, que incluyen instrucciones de un usuario y sensores necesarios para que tome decisiones, así como salidas eléctricas para motores e informar sobre el estado del vuelo. En los UAV del GII el dispositivo que cumple dicha función es el Pixhawk v2.

Pese a que los UAV del GII cuentan desde antes de la ejecución del proyecto con autopilotos sometidos a pruebas satisfactorias y que ofrecen funcionalidades básicas como el mantenimiento de la posición, así como el retorno a la zona de despegue y la realización de rutas preprogramadas de forma automática; los mismos contaban prácticamente con un nulo nivel de inteligencia y autonomía, tanto a nivel de software y hardware.

Esto implicaba que los UAV eran incapaces de realizar una tarea que requiriera un comportamiento de naturaleza autónoma, en el que tuviera que definir sus propias rutas y la manera de emplear sus sensores y actuadores, de acuerdo a los estímulos del entorno y el fin propio de la misión que estuviese llevando a cabo; los UAV que se tienen sólo podían funcionar con rutas preprogramadas o con un mando remoto empleado por un usuario.

Por su parte, los Pixhawk v2 usaban un módem conectado a uno de sus puertos con el que intercambiaban información por radiofrecuencia con otro módem, que conectado a una computadora de control en tierra (CCT), permitía monitorear el estado del vuelo al instante, como se muestra en la Figura 1.1.1 en color rojo. Sin embargo, dicho par de módems no permitían operar los UAV's a más de 200 m, distancia a la que se perdía la conexión, mientras que la legislación española permite hacerlo hasta a 500 m, además que aún con distancias cortas había un retardo de hasta 2 s para visualizar en la CCT, lo que ocurría.

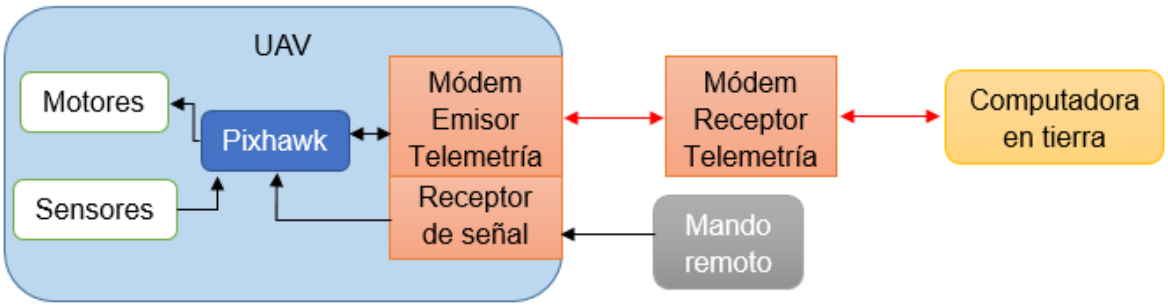


Figura 1.1.1 Esquema general de conexiones UAV-mando remoto-computadora en tierra. Fuente: elaboración propia.

Capítulo 2. Objetivos

2.1 Objetivo General

Diseñar una estructura de hardware-software que permita comportamientos complejos y la generación de rutas dinámicamente por medio de la concesión de autonomía a los UAV's.

2.2 Objetivos Específicos

- Caracterizar formalmente la estructura de hardware/software actual en los UAV's con el fin de entender su comportamiento mediante la creación de la documentación respectiva.
- Establecer un canal de comunicación entre una mini PC y el autopiloto, con ayuda del conocimiento generado en la documentación, para dotar de autonomía a los UAV's .
- Validar los procesos de verificación y comunicación entre la mini PC y el autopiloto con el fin de garantizar un producto de calidad mediante el sometimiento a un conjunto de pruebas.
- Diseñar una biblioteca de software de comunicaciones que permita la ampliación y mejora del mismo para posteriores aplicaciones.

Capítulo 3. Marco Teórico

3.1 Conceptos relacionados con los sistemas y procesos previos a la realización del proyecto

Los vehículos aéreos no tripulados (VANT) o UAV (del inglés *unmanned aerial vehicle*) carecen de un piloto a bordo, por lo que se suelen manejar con un mando remoto, con rutas preprogramadas o con programas autónomos a bordo que definen sus propias rutas de acuerdo a la misión que les sea encomendada.

Si bien en un principio su fin era meramente militar, esto ha venido cambiando hasta llegar incluso a manos de las personas para fines recreativos. Aunque sin duda los fines industriales están tomando fuerza, los cuales son de interés en este proyecto.

Los UAV's se pueden clasificar según el tipo de sustentación en **ala fija** o **ala rotatoria**. Los primeros tienen un perfil alar (alas con una sección transversal como las de los aviones), requieren estar en movimiento para experimentar la sustentación y por lo tanto mantenerse en vuelo; dado esto, son ideales para mapear grandes superficies de terreno, por lo que se emplean ampliamente en labores agrícolas y de fotogrametría, ya que además tienen un consumo energético relativamente bajo y su batería generalmente soporta tiempos considerablemente grandes. Otro detalle importante de estos UAV's es que para despegar requieren ser lanzados ya sea por una persona o catapulta (no pueden levantarse verticalmente), mientras que para aterrizar algunos ya pueden hacerlo de forma autónoma. [2]

Por otro lado, los UAV's de **ala rotatoria** o multirrotores, generan su sustentación a partir de la fuerza que generan las hélices unidas a sus motores. Estos tienen la ventaja de que permiten la adición de diversos artefactos como cámaras, sensores o actuadores y debido a que no necesitan estar en movimiento constante para sostenerse, pueden realizar vuelos estacionarios, lo que los hace especiales para labores de inspección, además de que pueden despegar y aterrizar verticalmente sin mayor cuidado de la superficie desde la que lo hacen. Sin

embargo, presentan una desventaja respecto a los de ala fija y es que su consumo energético es elevado debido a que usan más motores, con lo que sus misiones no suelen durar tanto porque su batería se agota más fácilmente. [2]

En la Figura 3.1.1 y Figura 3.1.2 se pueden observar algunas diferencias anteriormente mencionadas entre los multirrotores y los UAV's de ala fija.

Figura 3.1.1 Despegue de un UAV de ala fija. Fuente: [2]

Figura 3.1.2 Vuelo de un multirrotor. Fuente: [3]

Por su parte, si bien los UAV's requieren de sus actuadores para moverse hasta una posición a una velocidad específica, requieren de algún dispositivo que gestione las señales a los diferentes motores para cumplir con las órdenes dadas, que además garantice al máximo la estabilidad de la aeronave. Estos dispositivos son denominados autopilotos.

El autopiloto es imprescindible en los UAVs, generalmente constituye toda una estructura de firmware que transforma las órdenes en señales eléctricas; sin él estos vehículos no son más que un chasis con motores cuyos rotores están unidos a una hélice. Uno de los más usados es el Pixhawk.

3.1.1 Ardupilot, pixhawk y protocolo MAVLink

Ardupilot es un software de autopilotos de código abierto, lo cual favorece toda una comunidad pendiente de su progreso y mejora, con lo que se obtiene documentación fácilmente, razón por la que es ampliamente usado. Por su parte, el Pixhawk es un autopiloto de hardware abierto, respaldado por el software de Ardupilot [4]. Estos dispositivos pueden controlar diferentes tipos de vehículos,

incluso submarinos; la forma de indicarles cuál nave van manejando es conectándolos a una computadora y cargarles el firmware correspondiente a través de un software de vuelo (*Ground control Station*, GCS en inglés) como Qgroundcontrol o Mission Planner (se hablará de ellos más adelante).

Ardupilot tiene prestablecidos varios modos de vuelo para cada tipo de vehículo, los cuales le permiten volar con una u otra condición particular. Por ejemplo, los modos para multirrotores se presentan en la Tabla 3.1, cuya simbología se explica en la Tabla 3.2.

Tabla 3.1 Modos de vuelo de ardupilot para multirrotores. Fuente: [5]

Mode	Alt Ctrl	Pos Ctrl	GPS	Summary
Acro	-	-		Holds attitude, no self-level
Alt Hold	s	+		Holds altitude and self-levels the roll & pitch
Auto	A	A	Y	Executes pre-defined mission
AutoTune	s	A	Y	Automated pitch and bank procedure to improve control loops
Brake	s	A	Y	Brings copter to an immediate stop
Circle	s	A	Y	Automatically circles a point in front of the vehicle
Drift	-	+	Y	Like stabilize, but coordinates yaw with roll like a plane
Flip	A	A		Rises and completes an automated flip
FlowHold	s	A		Position control using Optical Flow
Follow	s	A	Y	Follows another vehicle
Guided	A	A	Y	Navigates to single points commanded by GCS
Land	A	s	(Y)	Reduces altitude to ground level, attempts to go straight down
Loiter	s	s	Y	Holds altitude and position, uses GPS for movements
PosHold	s	+	Y	Like loiter, but manual roll and pitch when sticks not centered
RTL	A	A	Y	Retruns above takeoff location, may aslo include landing
Simple/Super Simple			Y	An add-on to flight modes to use pilot's view instead of yaw orientation
SmartRTL	A	A	Y	RTL, but traces path to get home
Stabilize	-	+		Self-levels the roll and pitch axis
Sport	s	s		Alt-hold, but holds pitch & roll when sticks centered
Throw	A	A	Y	Holds position after a throwing takeoff

Tabla 3.2 Simbología de la Tabla 3.1. Fuente: [5]

Symbol	Definition
-	Manual control
+	Manual control with limits & self-level
s	Pilot controls climb rate
A	Automatic control

A pesar de ser tantos, son pocos los que se usan regularmente, como por ejemplo, el modo estabilizado (*stabilize*) que permite mantener el UAV horizontal, el modo mantener altura (*alt hold*) que permite hacer desplazamientos en un plano horizontal y ahora en el emergente área de los UAVs autónomos empieza a tomar importancia el modo guiado (*guided*), que hace que el vehículo reciba y ejecute órdenes de una GCS e ignore al mando remoto que tenga conectado.

Por su parte, el protocolo de comunicación que usa el Pixhawk es MAVLink, especialmente diseñado para el intercambio de información entre autopilotos de micro UAVs y softwares de vuelo. La información (toda la necesaria para llevar a cabo la comunicación) se encuentra en ficheros de extensión .xml, permitiéndole estar estructurada y empaquetada [6]. Un paquete MAVLink es una secuencia de bytes con la estructura mostrada en la Figura 3.1.3.

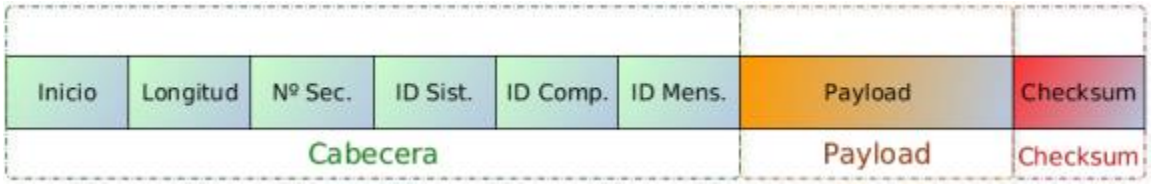


Figura 3.1.3 Estructura de un paquete de información del protocolo MAVLink. Fuente: [6].

El payload es la información como tal que se está transmitiendo. Por su parte, la cabecera da información de cómo interpretar dicho payload, lo cual se explica en la Tabla 3.3. El control de errores se lleva a cabo con la información del checksum [6].

Tabla 3.3 Uso de cada byte de la cabecera de un paquete de información MAVLink. Fuente: [6]

Byte	Explicación
Inicio	Este byte indica el comienzo del paquete MAVLink.
Longitud	Indica el número de bytes que se van a enviar como payload del mensaje. Puede tomar un valor entre 0 y 255.
Número de secuencia	Muestra el número de paquete enviado según su pertenencia a la secuencia de los mismos con el objetivo de reordenar éstos en el destino (por si fuera éste de gran tamaño) o detectar paquetes perdidos. Puede tomar un valor entre 0 y 255.
Identificación del sistema	Identifica el sistema, mediante un número natural, que ha enviado el mensaje permitiendo diferenciar entre varios sistemas de la misma red. Puede tomar un valor entre 0 y 255.
Identificación del componente del sistema	Señala, mediante un número natural, el componente de dicho sistema que ha enviado el mensaje permitiendo de este modo diferenciar varios componentes dentro del mismo sistema. Puede tomar un valor entre 0 y 255.
Identificación del mensaje	Indica el tipo de mensaje enviado, señalando cómo se tiene que leer el payload que viene a continuación. Utilizando un símil, es una leyenda que indica cómo se debe leer y traducir la carga útil del mensaje. Puede tomar un valor entre 0 y 255.

Por otro lado, Pixhawk tiene varias versiones de autopilotos, la que se usa en este proyecto la segunda (Pixhawk v2) o el cubo como también se le llama, debido a que su IMU (del inglés *Inertia Measurement Unit*) o unidad de medición inercial está, en cierta manera, aparte de la placa principal y tiene la forma de un cubo, como se muestra en la Figura 3.1.4.

Figura 3.1.4 Vistas superior e isométrica del Pixhawk v2. Fuente: [7]

En [8] se explica que el Pixhawk v2 consta de dos partes principales: la IMU (anteriormente mencionada) y la unidad de manejo de vuelo FMU (del inglés *Flight Management Unit*). [9] dice que la IMU es un dispositivo electrónico cuyo objetivo

es obtener mediciones de velocidad, rotación y fuerzas gravitacionales en forma autónoma. Se utilizan como componentes fundamentales en los sistemas de navegación de barcos, aviones, helicópteros, misiles o cualquier móvil en que sea necesario estimar estas mediciones; en el caso de los UAV's son de especial importancia para evitar una inclinación excesiva en el vuelo que provoque un vuelco.

[8] afirma que la IMU del Px4 V2, al igual que la FMU, contiene acelerómetros, giroscopios y barómetros que se complementan (la IMU además tiene un magnetómetro). En la FMU está el firmware como tal, además de las entradas (otros sensores como GPS, instrucciones de un mando remoto) y salidas (PWM de motores, envío de información de estado de vuelo); en ella se procesa la información de los sensores propios y de otros externos que se quieran colocar, así como de las instrucciones que se reciben para generar las salidas correspondientes a los motores y a los indicadores del estado del vuelo. En la Figura 5.1.1 se muestra un diagrama que explica lo anterior de forma más clara.

3.1.2 Telemetría y transmisores de radiotelemetría

En [10] se explica que la telemetría es una técnica de comunicaciones que mide y recopila datos de distinta naturaleza en cualquier aplicación, los cuales se encuentran en lugares remotos y son transmitidos para efectos de monitoreo y registro. Un sistema de telemetría normalmente consiste de un transductor como dispositivo de entrada, un medio de transmisión, ya sea líneas de cable u ondas de radio (generalmente de la segunda manera, en lo que se conoce como radiotelemetría), dispositivos de procesamiento de señales y dispositivos de grabación o visualización de datos.

El transductor convierte una magnitud física como la temperatura, presión o vibraciones en una señal eléctrica correspondiente, o viceversa, que es transmitida a distancia [10]. En el caso de la información que genera un autopiloto como el Pixhawk, se tienen varios transductores, constituidos por sensores que lleva incorporados, por ejemplo de GPS o velocidad y el medio de transmisión suele ser un transmisor de telemetría, que es un módem especial para este tipo de

información, el cual envía información a otro que está conectado algún dispositivo que permita visualizar los datos, generalmente una computadora con un software específico.

3.1.3 Software de vuelo: mission planner, Qgroundcontrol

Javier Losada, colaborador del área de UAVs del GII, indicó que los software de vuelo son esenciales para llevar a cabo cualquier misión con un vehículo no tripulado que use un autopiloto como Pixhawk, ya que con el autopiloto conectado a un puerto de la computadora que contiene el software, se le puede indicar qué tipo de nave va a controlar, configurar cuáles entradas se van a usar y para qué, así como prefijar rutas y modos de vuelo. Ya en el aire se puede monitorear el estado de la misión, tanto en localización con GPS, como en características cinemáticas (siempre que se cuente con una señal suficientemente potente). Por su parte, incluso durante el vuelo se le pueden ir dando localizaciones que se quiere que se siga el UAV. Dos software ampliamente usados son Mission Planner y Qgroundcontrol (J. Losada, comunicación personal, 25 de setiembre de 2018).

Este tipo de software también permite simular programas de vuelo de manera que se pueda visualizar lo que teóricamente haría el vehículo, con lo cual se podrían encontrar errores también. En [11] se comenta que existen 2 tipos de simulación: *software in the loop* (SITL) y *hardware in the loop* (HITL).

En el primer tipo, el software contiene un modelo fiel del autopiloto, razón por la que no se necesita conectar el hardware, todo el procesamiento se hace en la PC simulando que es el propio Pixhawk, se ve como el vehículo se desplaza en un entorno también simulado [11]. Por otro lado, el HITL tiene la simulación del chasis del vehículo, así como de su entorno, pero usa un autopiloto real conectado a la PC, el cual ejecuta todo el procesamiento necesario para mover el UAV según se solicite, movimiento que igualmente se observa en el software de vuelo [11].

En la Figura 3.1.5 se muestra el estado de vuelo de la simulación de una misión con el software Mission Planner. A la izquierda se observa una serie de indicadores como la inclinación del UAV, la dirección, altura, rapidez con respecto

a tierra, entre otros. Por su parte, a la derecha se muestra la ruta que el vehículo está haciendo, la dirección instantánea (línea negra) y la localización gps (en la parte inferior). Todo lo anterior se va actualizando conforme avanza el tiempo.

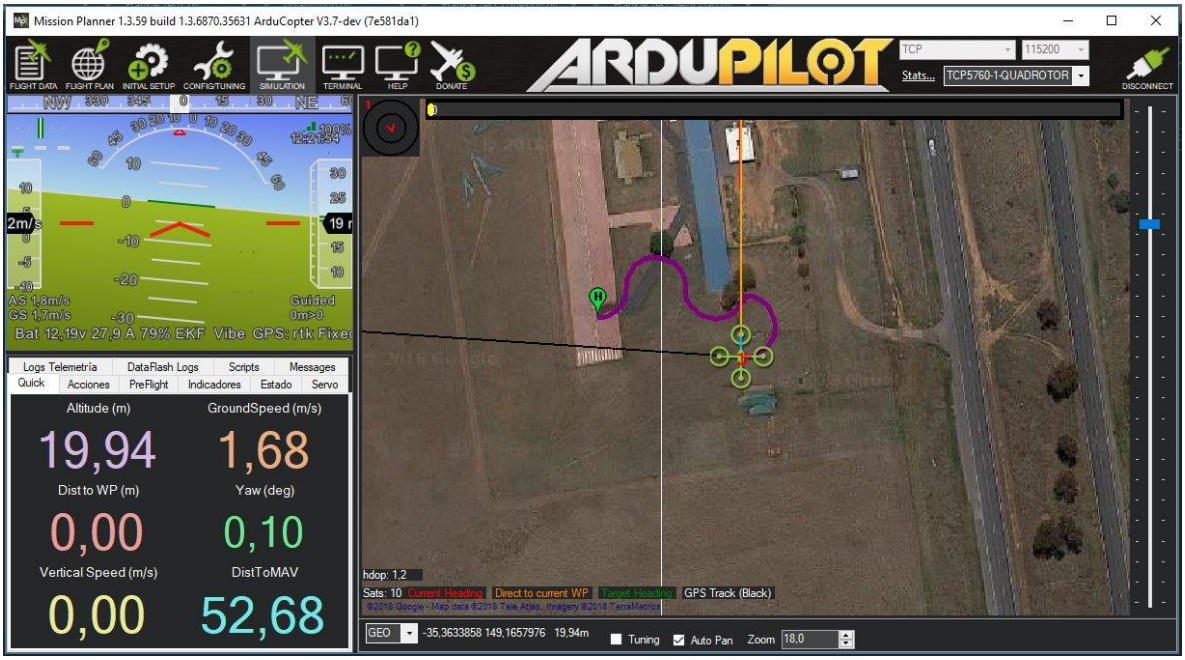


Figura 3.1.5 Simulación de un trozo de trayectoria senoidal, seguido de una trayectoria circular. Fuente: Elaboración propia.[1]

Mission Planner guarda toda la información de cada vuelo (simulado o real) en archivos de extensión .tlog. Desde este mismo software se puede abrir archivos tlog y volver a ver el vuelo, además puede convertirlos a archivos de extensión .kmz, los cuales se pueden abrir en Google Earth para ver la ruta en un espacio tridimensional (J. Losada, comunicación personal, 10 de noviembre de 2018), como se mostrarán en el capítulo 6 de este escrito.

[1] La imagen es propiedad del autor del escrito, no así el software con el que se obtuvo [30].

3.2 Descripción del hardware y software necesarios para la solución

En este apartado se expondrá la teoría necesaria para entender todo lo que se dice a partir del capítulo 4, donde se detallan las condiciones previas en los octocópteros (tipo de multirrotores sobre los que actúa el proyecto) y con especial importancia el capítulo 5; la solución que se dio al problema.

3.2.1 Conceptos de redes informáticas

Una red de computadoras, también llamada red de ordenadores o red informática, es un conjunto de equipos conectados por medio de cables, señales, ondas o cualquier otro método de transporte de datos, que comparten información (archivos), recursos (CD-ROM, impresoras, etc.) y servicios (acceso a internet, e-mail, chat, juegos), etc [12]. En las redes existen elementos que permiten el flujo de información y tienen diferentes funciones, aquí se mencionarán los *switches* (conmutadores), *routers* (enrutadores), *gateways* (puertas de enlace) y puentes.

Un conmutador es un dispositivo que permite la existencia de una red, a él se conectan todos los equipos. De acuerdo a la Figura 3.2.1, el conmutador permite enviar información de una computadora a otra directamente, mientras que otras dos computadoras pueden estarse comunicando al mismo tiempo. Lo que no permite es que varias computadoras quieran enviarle información a otra al mismo tiempo. Los mensajes tienen direcciones MAC, de esta forma identifican la computadora destinataria [13].

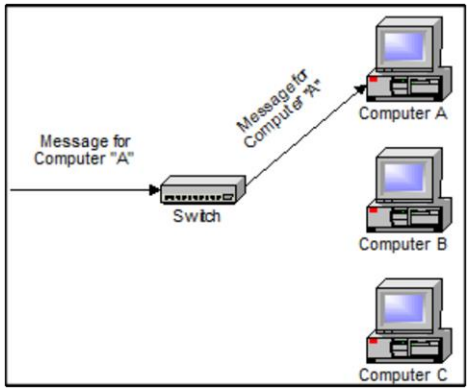


Figura 3.2.1 Un mensaje viajando a través de un switch únicamente a la computadora destino, y no a todas las de la red. Fuente: [13]

Por su parte, los enrutadores pueden hacer lo mismo que los conmutadores y más. Permiten hallar la ruta más cercana para enviar información de un equipo A a un equipo B, con la gran ventaja de que ambos computadores pueden ser de redes diferentes, es decir, permite la transferencia de información entre redes. Por otro lado, la puerta de enlace es el dispositivo que conecta dos redes, por el que sale información de una red hacia otra y viceversa; cada red tiene un dispositivo conectado a la puerta [14].

Cuando las comunicaciones son inalámbricas hay un dispositivo esencial: los puentes (*wireless bridge*). Se conecta uno a cada una de las PCs que quieren intercambiar información y forman un canal de comunicación donde los datos se transmiten por el aire. Elementos como conmutadores, enrutadores o puertas de enlace en algunas ocasiones los traen incorporados.

Dentro de cada red cada elemento tiene un número de identificación único denominado dirección IP, del inglés *Internet Protocol*. Este ha tenido varias versiones; IPv4 (la cuarta) define la forma que tienen las direcciones IP normalmente: de acuerdo a la Figura 3.2.2, son 4 octetos de números binarios, que se suelen representar también por sus correspondientes números decimales.

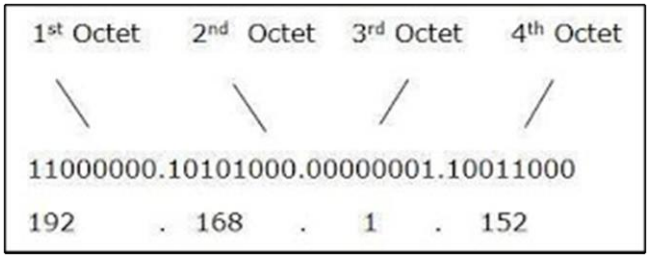


Figura 3.2.2 Estructura de una dirección IP. Fuente: [15]

Según [16], IPv4 define 3 clases de IP para redes locales:

- 10.0.0.0 – 10.255.255.255 (Clase A) – 16.777.216 direcciones.
- 172.16.00 – 172.31.255.255 (Clase B) – 1.048.576 direcciones.
- 192.168.0.0 – 192.168.255.255 (Clase C) – 65.536 direcciones.

Siendo la clase C la que tiene menos direcciones posibles, es la que se usa más habitualmente en los enrutadores de los hogares por disponerse de una

cantidad pequeña de dispositivos, por lo cual es habitual ver direcciones IP del tipo 192.168.XXX.XXX [16]. Por su parte, hay otro parámetro que cada red tiene asociada: la máscara de subred.

En [17] se explica el concepto de máscara de subred: “Partimos de la dirección 192.168.1.24. Si tenemos una máscara de subred 255.255.255.0, quiere decir que todas las direcciones tendrán como parte fija 192.168.1 y el último octeto será que el presentará la variación. Es decir, desde la 192.168.1.0 a 192.168.1.255.”

En Windows, la IP de la computadora se puede configurar fácilmente (realmente es la IP de uno de sus puertos), pero no sólo requiere especificar la dirección deseada, sino la máscara de subred y un parámetro denominado *gateway*, como se muestra en la Figura 3.2.3. Como se dijo antes, el *gateway* es el dispositivo por el que se comunica una red con otra, y dado que en una computadora el enrutador también cumple dicha función para conectarse a internet, en la casilla de *gateway* se coloca la dirección IP del *router*.

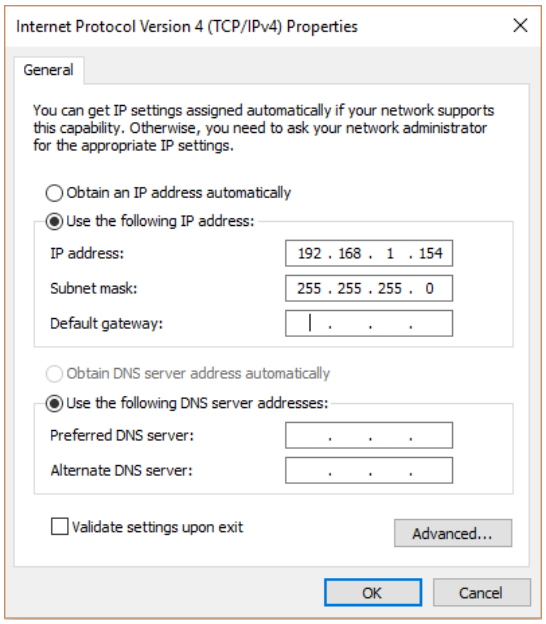


Figura 3.2.3 Configuración de la IP del puerto Ethernet de una PC. Fuente: elaboración propia.

3.2.2 Uso de Python para el planeo de un vuelo: mavproxy y dronekit

Python ofrece una serie de funcionalidades amplia en cuanto a programación, especialmente cuando se trata de ligar y controlar hardware externo a la computadora donde esté instalado dicho lenguaje de programación, gracias a sus múltiples bibliotecas y aplicaciones. El área de los UAVs no es la excepción; en ella se usan principalmente mavproxy y dronekit.

Mavproxy por su parte, es una aplicación de Python que en esencia es otro software de vuelo como Mission Planner, tiene una interfaz gráfica y demás, pero también ofrece otras funcionalidades sumamente útiles, como la posibilidad de tomar un puerto donde está conectado un UAV enviando y recibiendo mensajes MAVLink y crear un canal de comunicación bidireccional a otros puertos (incluso varios a la vez), tanto internos (UDP o TCP), como externos; puede llegar a otras computadoras siempre que haya una red de por medio, lo cual es ideal para enviarle instrucciones al UAV desde una computadora a distancia y visualizar la telemetría.

Por su parte, dronekit es una biblioteca de Python que hace posible no sólo enviar instrucciones a un Pixhawk, sino solicitar información de vuelo, como la velocidad, altitud, ubicación, entre otros. Una de sus funciones principales, “connect”, que recibe como parámetro, entre otras cosas, el puerto de la PC en el que se encuentra conectado el autopiloto, crea un objeto que tiene como atributos una serie de indicadores del estado de vuelo, los cuales pueden observarse y modificarse.

Por otro lado, dronekit tiene dos instrucciones principales: “simple_takeoff” para elevarse a una altura especificada y “simple_goto” para moverse a una posición dada en formato GPS, aunque normalmente resulta conveniente y necesario acompañarlas de otros comandos, ya que estas funciones por sí solas no brindan una señal cuando el UAV alcanzó la posición deseada y si en un programa, por poner un ejemplo común, se da un “take_off” seguido de un “goto”, la segunda orden se sobrescribirá a la primera sin haberse elevado completamente.

Capítulo 4. Procedimiento metodológico

A continuación, se da una descripción general de las etapas en que se divide el trabajo, desde las circunstancias iniciales hasta la finalización del proyecto.

4.1 Identificación de las condiciones previas al proyecto y del problema a solucionar

Al llegar al Grupo de Integrado de Ingeniería se tuvo una reunión con los ingenieros Félix Orjales Saavedra (director del proyecto) y Javier Losada Pita (colaborador directo del área de UAV's), además se tuvo conversaciones varias y se participó en vuelos de prueba (relacionados a otros proyectos) con los multirrotores del GII; actividades que tenían como fin entender cómo estaban funcionando los UAV's, delimitar el proyecto y entender el problema a solucionar así como identificar las variables que lo afectaban. Toda esta información se brinda a continuación.

El Pixhawk de los octocópteros del GII (multirrotores a los que se limitó el proyecto) recibe instrucciones de un mando remoto del tipo elevarse, bajar, mantener su posición o su altura y moverse en un plano horizontal. Dicho mando tiene un interruptor para hacer que el multirrotor cambie de modo, incluyendo al guiado (de especial importancia) y complementando con algún software de vuelo se pueden prefijar puntos o indicarlo durante el vuelo.

Por su parte, el proyecto actuó sobre la línea roja de la Figura 1.1.1, ya que los módems que se usaban (*V2.0 3DR Radio Telemetry 433mhz-915mhz Data Transmission Module For Px4*) tenían un funcionamiento inapropiado. Además, se necesitaba dotar al UAV de una estructura de hardware y software que le permitiera ejecutar rutinas de comportamiento autónomo. Dicha estructura está hecha bajo los siguientes requerimientos:

- Debe permitir a los UAV's en vuelo definir sus rutas y forma de emplear sus sensores y actuadores, así como llevarlas a cabo, de acuerdo a la misión que esté desarrollando.

- Debe permitir visualizar las decisiones que toman las rutinas de comportamiento autónomo desde una computadora en tierra.
- Debe seguir siendo posible visualizar el estado de vuelo desde una computadora en tierra con ayuda de software como Mission Planner o Qgrondcontrol.
- Deben usarse, en la medida posible, equipos de los que ya disponga el GII en vez de comprar nuevos.
- Debe incrementarse la distancia a la que se puede visualizar en una computadora en tierra el estado de vuelo a, por lo menos, 500 m.
- El retardo en la visualización del estado de vuelo debe reducirse lo más que se pueda.

4.2 Pasos seguidos

El proyecto se dividió en las siguientes etapas:

- **Estudio de la estructura de conexiones de los UAV:**

Consistió en una búsqueda bibliográfica de los componentes del Pixhawk, los protocolos de comunicación, así como una serie de visitas al Ing. Javier Losada Pita, a quien se le consultaron los detalles más relevantes de las conexiones que tienen los UAV's del GII, con el fin de comprender y describir su funcionamiento para así cumplir con el primer objetivo de documentar el dispositivo que se tiene y tener el conocimiento del autopiloto para poder desarrollar las demás tareas.

- **Configuración de una mini PC a bordo para el almacenaje y ejecución de rutinas de comportamiento autónomo:**

Se conectó el Pixhawk a una mini PC que va a bordo de los UAVs, en la cual se ejecutan diferentes rutinas de comportamiento autónomo que le dan órdenes al autopiloto y reciben retroalimentación de este. Por su parte, la información de vuelo es enviada a través de una nueva red, a una computadora de control en tierra para su visualización, a la vez que se accede al terminal de la mini PC con el fin de ejecutar las rutinas autónomas en el momento deseado, permitiendo visualizar todas las decisiones que toman los programas.

- **Biblioteca de software**

Se creó una biblioteca con 12 funciones que permiten enviar órdenes y monitorear lo que está pasando durante un vuelo, de una manera más funcional que si sólo se usaran las funciones básicas de dronekit. Por su parte, también se creó un archivo que contiene variables globales al que se llama en la biblioteca y también debe llamarse en los programas que se hagan.

- **Verificación (simulación) y validación (vuelo de prueba)**

Una vez que se tuvo una primera versión de la biblioteca se desarrollaron varias rutinas de vuelo para simulación, en las que la biblioteca fue modificada en múltiples ocasiones para alcanzar los resultados deseados, que incluyeron desplazarse hasta algún vector de desplazamiento, moverse de un punto con coordenadas GPS a otro, dibujar formas en el aire y por supuesto ejecutar una rutina de comportamiento autónomo, la cual se usó para validar el proyecto por medio de un vuelo de prueba.

Capítulo 5. Descripción detallada de la solución

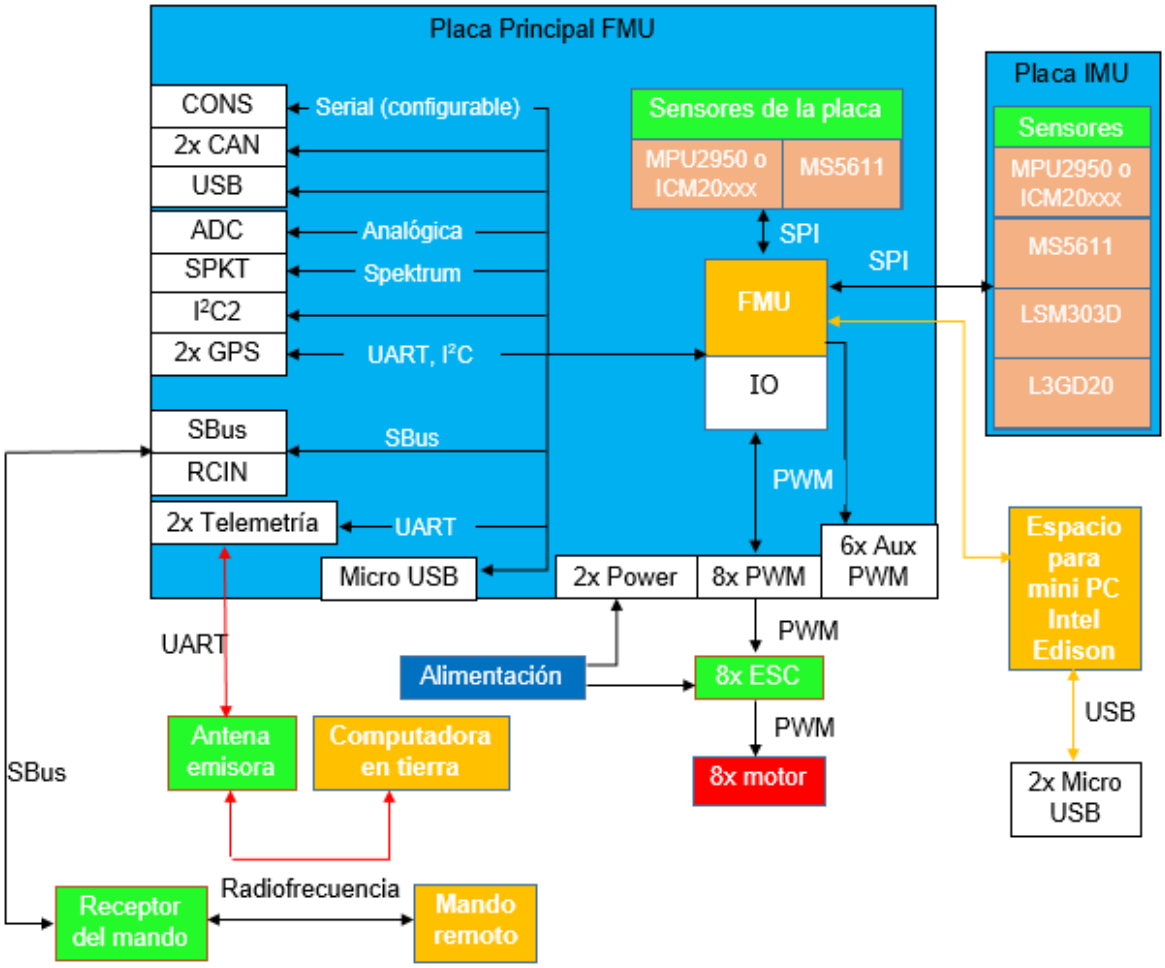
5.1 Descripción de las conexiones previas de los UAV's del GII

Hablar de las conexiones del Pixhawk de un UAV es hablar de las conexiones del UAV como tal; todo está inherentemente relacionado con el autopiloto. En la Figura 5.1.1 se presenta un esquema que explica cómo se relacionan algunos de los elementos que poseen los multirrotores (octocópteros todos) del GII, por su parte la información del uso de cada puerto para cada uno de los UAV's de interés se encuentra en la Tabla 5.1.

Las dos placas FMU e IMU, como ya se dijo, integran el Pixhawk, el cual tiene todos los puertos y sensores mostrados en la Figura 5.1.1, así como los protocolos de comunicación respectivos. Los puertos se nombraron haciendo referencia a los nombres que tienen impresos la placa, como se ve en la Figura 3.1.4. Los puertos denominados GPS tienen un canal UART y uno I^2C, con todos los pines respectivos, sin embargo, el puerto GPS1 tiene dos pines adicionales, con respecto al GPS2, que se usan generalmente para un botón de seguridad [8].

Además, se muestra una etapa de potencia (ESC, de *Electronic Speed Control*) entre el autopiloto y los motores, el receptor de señal del mando remoto, una prevista de conexión para una mini PC Intel Edison que trae el Pixhawk, la cual no se va a usar (flecha naranja) debido a que dicha PC está descontinuada.

Pero sin duda, la parte más importante a identificar es la que contiene flechas rojas, ya que este proyecto actúa directamente sobre ese canal, debido a que entre el Pixhawk y la computadora de control en tierra (CCT) debe colocarse la Mini PC para dar instrucciones emitidas por rutinas de inteligencia artificial y que además comunique a los usuarios en tierra las instrucciones que toma, a través de un nuevo par de transmisores que permitan operar el UAV a mayor distancia a la vez que reduzcan el retardo existente (aproximadamente 2 s).



Código de colores de bloques	
	Placas
	Puertos
	Elementos de procesamiento de datos
	Elementos intermedios
	Actuadores
	Sensores
	Alimentación

Códigos de algunos componentes del Pixhawk	
Componente	Código
Flight managment Unit (FMU)	STM32F427
Entradas y salidas (IO)	STM32F100
Acelerómetro/ giroscopio	MPU2950 ó ICM20xxx
Barómetro	MS5611
Giroscopio	L3GD20
Acelerómetro/ magnetómetro	LSM303D
Controlador electrónico de velocidad (ESC)	Hobbywing 40 A 2-6S lipo No BEC

Figura 5.1.1 Esquema de conexiones de los UAV's del GII. Fuente: elaboración propia.

Por su parte, los UAV's tienen una serie de funciones en los puertos del Pixhawk que no se muestran en el esquema anterior, las cuales se especifican en la Tabla 5.1.

Tabla 5.1 Funciones llevadas a cabo en los puertos de los autopilotos de cada uno de los UAV's de interés. Fuente: elaboración propia.

Funciones llevadas a cabo en los puertos del Pixhawk 2				
Función	Tipo de puerto requerido	Nombre del puerto usado en cada UAV		
		Pequeño	Mediano	Grande
GPS	UART	GPS1	GPS2	GPS1
Botón de seguridad	I2C	GPS1	GPS1	GPS1
Led de estado	I2C	I2C2 (El puerto está ampliado)	–	I2C2
Envío de información de vuelo al OSD (on screen display), para ser superpuesta en el vídeo que es enviado al emisor de vídeo	UART	Telem1	Telem1	Telem1
Emisión de telemetría	UART/USB	GPS2	Telem2	Telem2
Buzzer	USB	USB	USB	USB
Extensión del puerto USB	USB	USB	–	USB

Hay varias funciones relacionadas con la información del estado de vuelo, como el GPS, que como se puede intuir, da información al Pixhawk de su localización. El led de estado y el Buzzer emiten diferentes luces y sonidos respectivamente, cuando algo no anda bien. Cuando se tienen cámaras (no siempre se tienen) desde Telem1 se envía información de vuelo que se superpone en la grabación y se envía así a alguna CCT preparada para recibir el video, el cual se envía por un transmisor distinto al de telemetría (telemetría que se transmite a la computadora de control en tierra).

El botón de seguridad sirve para indicarle al UAV en qué momento puede empezar a ejecutar instrucciones, lo cual permite que sólo hasta que se esté seguro de que se puede empezar el vuelo, el UAV ejecute cualquier instrucción.

5.2 Estructura de software y hardware para hospedaje y ejecución de rutinas de comportamiento autónomo

En este apartado se explica la esencia misma del proyecto, del cual se presenta un diagrama general en la Figura 5.2.1; Se añadió una mini PC Intel NUC 7i5BNK de la cual disponía el GII, a bordo del UAV pequeño (un octocóptero) para albergar y ejecutar software de comportamiento autónomo que define el movimiento de la nave y se lo ordena al Pixhawk. Por su parte, este último le envía a la mini PC (cuyo sistema operativo es Windows 10) toda la información de telemetría (estado de vuelo) necesaria como realimentación para la toma de decisiones.

Dentro de la mini PC se creó un canal adicional para que la telemetría se visualice también en la CCT con ayuda de un software de vuelo. Por su parte, se accede al terminal de la mini PC desde la computadora en tierra, desde la cual se ejecutan las rutinas de interés almacenadas en el UAV y se visualizan las decisiones autónomas que toma. Para dichas rutinas autónomas se creó una biblioteca de software con algunas funciones útiles para enviar órdenes y visualizar el estado de vuelo.

Por su parte, se añadió un nuevo par de dispositivos emisor y receptor que posibilitan la red creada y son el canal de comunicación entre ambas PCs. Si bien ambos comparten información bidireccionalmente, el emisor tiene como principal función enviar la información de vuelo y de comportamiento autónomo a la CCT, mientras que la del receptor es captar la misma.

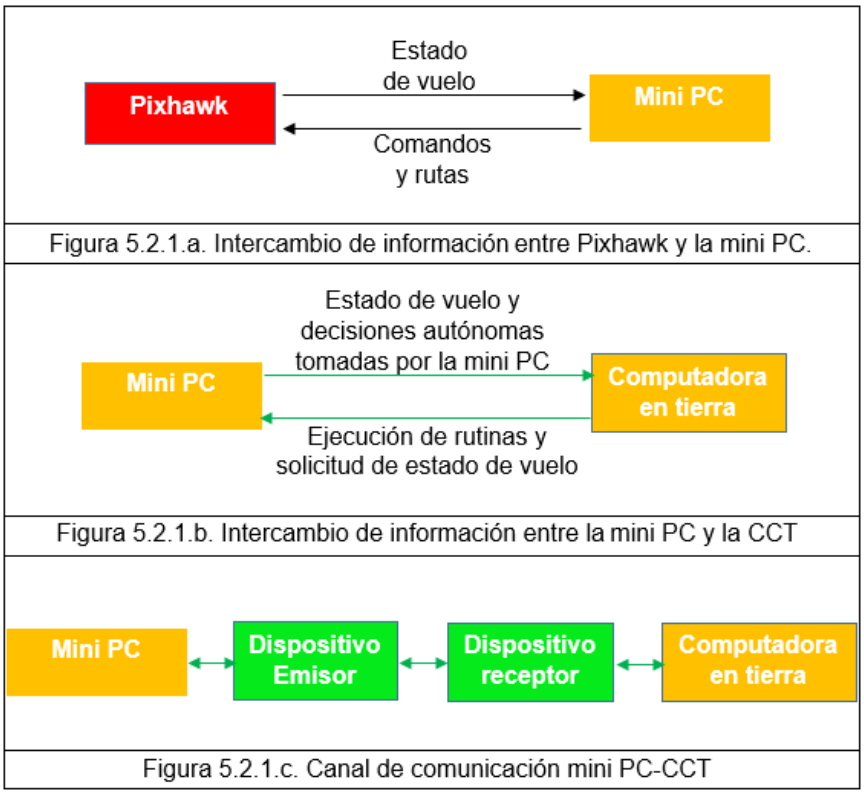


Figura 5.2.1 Estructura de software y hardware para rutinas de comportamiento autónomo. Fuente: elaboración propia.

5.2.1 Nueva red inalámbrica

El primer paso consistió en hacer una red local con la mini PC y una CCT, sustituyendo los módulos transmisores de telemetría de 3DR, por dos puente Ethernet inalámbrico/puerta de enlace (pueden cumplir ambas funciones) de la serie *IPn20 de Microhard Systems INC* de los que disponía el GII y no estaba usando.

Los módulos de 3DR tienen una potencia de transmisión de apenas 100 mW, mientras que las puertas de enlace pueden entregar hasta 1 W. Además el dispositivo que se queda en tierra tiene una antena omnidirecional de largo alcance. Para caracterizar el efecto de la nueva red en la visualización del estado de vuelo se conectó un Pixhawk a la mini PC, el cual transmitía la telemetría a la CCT través de la NUC y la red. En la computadora de control se abrió el Mission Planner para visualizar dicha telemetría; por medio de una observación cualitativa se vio que al efectuar movimientos en el Pixhawk estos se visualizan casi de inmediato, con un retardo estimado de medio segundo.

Por su parte, las puertas de enlace empleadas se configuraron de acuerdo al apéndice 10.1, basado en las instrucciones de [18]. Estos puentes pueden usarse en diferentes topologías de red, una de ellas se denomina “punto a punto” y es para transferir información entre únicamente 2 de estos dispositivos (con otras topologías se pueden crear redes de más grandes, pero en este caso no era necesario). Para esto se define uno como maestro y uno como remoto. El maestro es el que se queda en tierra y se le asignó 192.168.1.254 como dirección IP, mientras que el remoto es el que acompaña a la NUC en el UAV, con dirección IP 192.168.1.253.

Por otro lado, también se designaron las direcciones IP de los puertos Ethernet de la mini PC y la CCT como 192.168.1.154 y 192.168.1.155 respectivamente, necesarios porque en dicho puerto se conectan las puertas de enlace.

La red inalámbrica se muestra funcionando en la Figura 5.2.2; se hizo un ping desde la CCT hasta la mini PC. A la izquierda está la NUC y a la derecha la CCT con el terminal abierto, cada una conectada por un cable Ethernet a las puertas de enlace (negro en la mini PC y blanco en la CCT), las cuales tienen sus respectivas antenas (negra y pequeña en la puerta de enlace aérea, blanca y grande en la de tierra) y baterías (rectangulares y blancas).

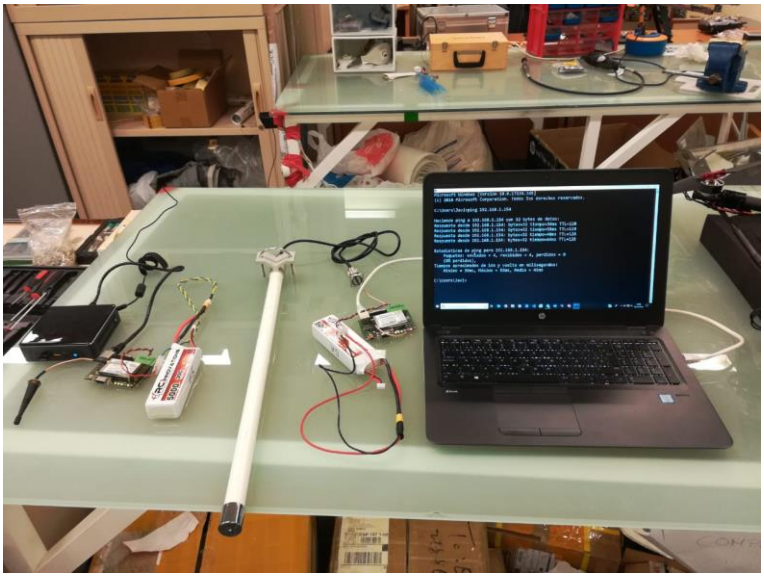


Figura 5.2.2 Nueva red inalámbrica operando. Fuente: elaboración propia.

En la Figura 5.2.3 se muestra la respuesta en el terminal de la computadora de control al hacerle ping a la NUC.

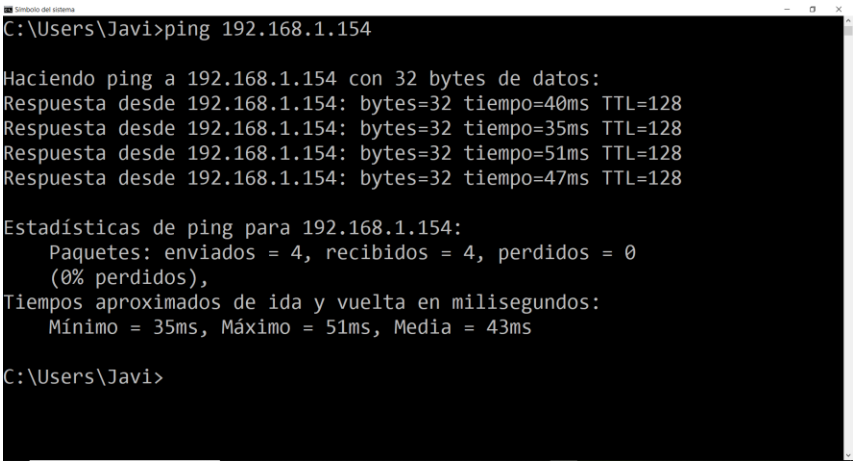


Figura 5.2.3 Ping desde la CCT con IP 192.168.1.155 hacia la mini PC con IP 192.168.1.154. Fuente: elaboración propia.

5.2.2 Flujo de información autopiloto-mini PC-computadora en tierra

Se prescindió del uso de los puertos UART del Pixhawk para enviar la telemetría según la Tabla 5.1 y Figura 5.1.1; se usó el puerto micro USB para enviar la información que genera, ya que es más práctico, dado que basta con un cable USB - micro USB para conectarlo a la mini PC y sin tener que configurar nada previamente. El Mission Planner detecta paquetes mavlink en los puertos de la computadora (en este caso el USB) y permite establecer la conexión para visualizar los datos.

Con la creación de la red ya se tenía el medio para compartir información entre la NUC (a la cual llega la telemetría del Pixhawk) y la CCT, pero aún no se estaba enviando a esta última, que es donde interesa visualizar todo lo relativo a los vuelos. Para resolver esto, se hizo uso del software mavproxy para crear un “switch virtual”; la señal del puerto COM4 (donde está conectado el Pixhawk) es replicada a dos puertos UDP, uno interno en la NUC para el procesamiento de las instrucciones autónomas con dirección IP 127.0.0.1:14551 y otro puerto interno de la CCT con dirección IP 192.168.1.155:14552. Esta acción se lleva a cabo en el terminal con las líneas de comandos que se muestran en la Figura 5.2.4.

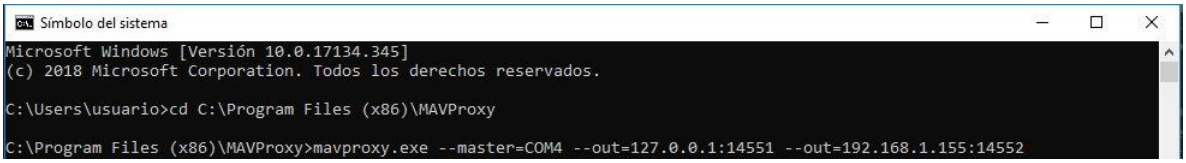


Figura 5.2.4 Instrucciones en el terminal de Windows para replicar la señal del puerto COM4 (maestro) en dos puertos UDP (salidas). Fuente: Elaboración propia

Por su parte, es importante mencionar que la dirección 127.0.0.1 es lo que se conoce como *local host*. Básicamente es la dirección IP con la que apunta la computadora (en este caso la NUC) a ella misma y al añadir el ":14551" hace referencia a su puerto interno 14551.

Entonces dentro de la NUC las rutinas de comportamiento autónomo no "llaman" al COM4 para conectarse con el Pixhawk, sino al puerto UDP 14551, mientras que el Mission Planner en la CCT se conecta al UDP 14552 para visualizar el estado de vuelo.

Por su parte, en un campo de vuelo no se tiene acceso a un monitor, teclado ni ratón, los cuales son necesarios para ejecutar las líneas de código de la Figura 5.2.4 en la mini PC. Dado esto, se colocaron las mismas líneas de código en un archivo batch, el cual se colocó en la dirección C:\Users\SuNombredeUsuario\AppData\Roaming\Microsoft\Windows\StartMenu\Programs\Startup, carpeta donde se almacenan todos los programas que se ejecutan automáticamente al encender la PC [19]; así una vez que se inicie la NUC, empezará a transmitir la telemetría a su puerto interno y a la CCT.

El archivo batch (extensión .bat) editado con el editor de texto Sublime Text se muestra en la Figura 5.2.5 con toda la documentación respectiva. Es importante recalcar algunas indicaciones que ahí se dan, como la necesidad de que las IPs sean estáticas y con las direcciones correctas previo a iniciar cualquier misión, ya que en el caso de la mini PC sería crítico llegar a un campo de vuelo con una IP incorrecta, dado que sin monitor (ni su alimentación), teclado y ratón, se vuelve una tarea tediosa. Por su parte, es necesario que la suspensión automática de la NUC

esté desactivada, ya que si se llega a suspender en vuelo sería catastrófico para el UAV; el Pixhawk dejaría de recibir indicaciones y el vehículo podría caer.

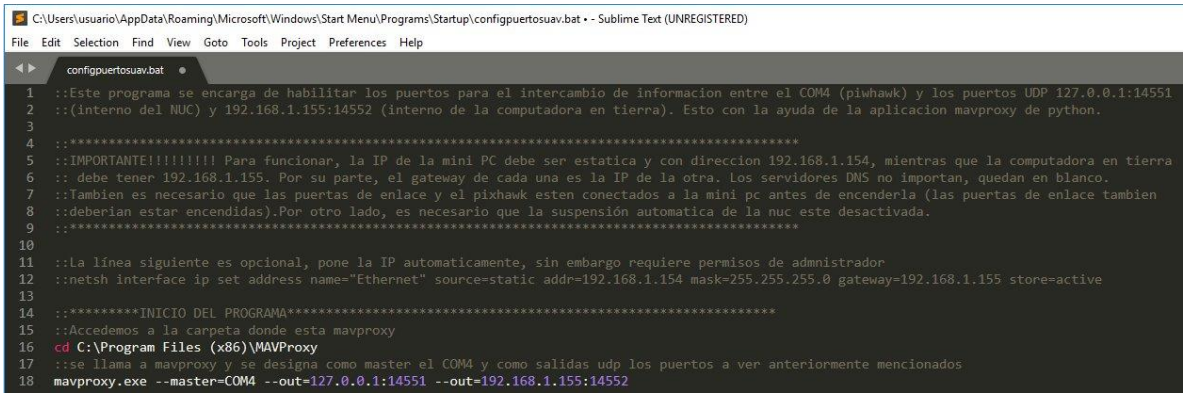


Figura 5.2.5 Archivo batch que permite el intercambio de información entre el Pixhawk, NUC y CCT. Fuente elaboración propia.

Por otro lado, se necesitaba tener control de la NUC desde la CCT; una forma de ejecutar los programas de la mini PC en el momento deseado, para lo cual se dejó un servidor SSH abierto en la mini PC usando el puerto 22, sin necesidad de instalar programas externos, ya que Windows tiene el suyo propio. Se configuró para que iniciara automáticamente al encender la mini PC, de acuerdo a las indicaciones dadas por [20] . Por su parte, se puede acceder desde cualquier CCT con programas como Putty, aunque Windows también tiene un cliente SSH.

En la Figura 5.2.6 se observa el acceso al terminal de la mini PC desde la CCT, teniendo control sobre la primera. En dicha imagen se ejecutó el programa “prueba_conexion.py”, el cual establece una conexión con el UAV conectado en el puerto UDP 14551.

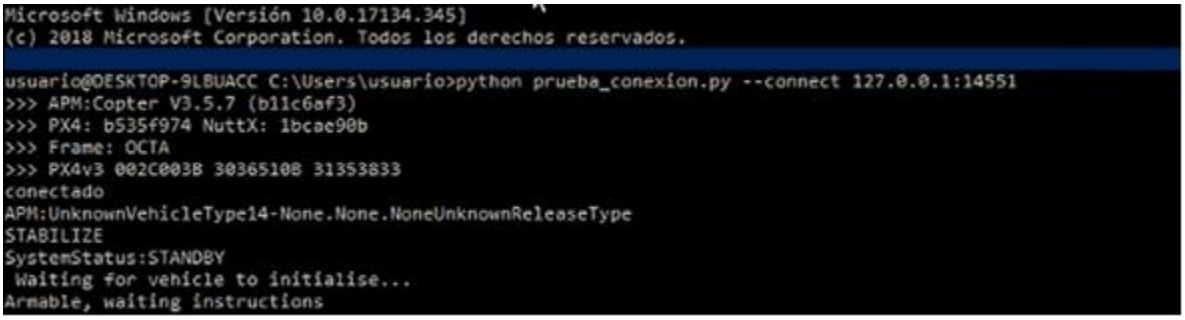


Figura 5.2.6 Acceso al terminal de la mini PC desde la CCT con el servidor y cliente SSH de Windows. Fuente: elaboración propia.

5.2.3 Biblioteca de software de comunicaciones

El lector podrá darse cuenta en la sección 3.2.3 de este escrito, que la forma de escribir rutinas de vuelo es empleando la biblioteca dronekit de Python, la cual, como se dijo antes, tiene dos instrucciones principales: “simple_takeoff” para elevarse a una altura especificada y “simple_goto” para moverse a una posición en formato GPS que se requiera. Sin embargo, el empleo de nada más que estas dos se queda corto en cuando a funcionalidad para definir trayectorias, además no indican cuándo el uav llega al lugar solicitado, y si por ejemplo en una línea de programa se da simple_takeoff y en la siguiente un simple_goto, esta última se sobreescribe a la primera, por lo que no terminará de elevarse. Por razones como estas normalmente hay que crear funciones con usos más amplios, haciendo uso de otros recursos de dronekit.

Debido a esto, se decidió hacer una biblioteca de Python llamada “estructurauav”, con varias funciones que servirán de enlace entre la inteligencia artificial y el vehículo. Para crear una biblioteca en Python basta con hacer un programa en dicho lenguaje con únicamente definiciones de funciones, darle un nombre (sin olvidar incluir la extensión .py) e incluirlo en la carpeta C:\PythonXY\Lib (donde XY son números que indican la versión de Python del computador, 27 en el caso de Python 2.7, por ejemplo).

La primera versión de la biblioteca contaba con 4 funciones únicamente, pero conforme se empezó a usar para simulación de varias rutinas de vuelo se vio que estas eran insuficientes y que además necesitaban depurarse, con lo que se llegó

hasta a 12 funciones y a necesitar una biblioteca extra de 4 variables globales denominada “globalvariables”, incluída también en la carpeta de bibliotecas de la versión de Python usada.

Las funciones de estructurauav se podrían dividir en 2; las que se usan para simulación principalmente y las que se usan tanto en simulación como en vuelos reales. De estas últimas, la mayoría monitorean el estado de vuelo –entre otras cosas- y usan parte de las variables globales anteriormente mencionadas para darse cuenta cuándo hay una desconexión entre la mini PC y el autopiloto (el modo de vuelo dejar de ser guiado) y así dejar de ejecutarse.

La biblioteca globalvariables está en el apéndice 10.3.1 donde se puede observar que tiene 4 variables globales debidamente explicadas: para el monitoreo del modo guiado están gui2other y valid_disc, mientras que para el monitoreo de la batería Battery_Volt y Battery_Level. La forma de usarlas se detalla en el apéndice 10.4 en la página 84, donde se usan en el programa de un vuelo de prueba; están debidamente importadas y llamadas, al igual que en estructurauav (apéndice 10.3.2) y sus funciones. A continuación se comentan las funciones:

- **arm:** se dice que un UAV con autopiloto pixhawk es armable cuando este último ha comprobado que se cumplen una serie de requisitos para poder volar. Una vez armable, debe armarse el vehículo (es enviarle una señal eléctrica), ya que de lo contrario no va a ejecutar ninguna orden. Esta función verifica si está armable y cuando lo esté, lo arma. Se usa en simulación principalmente, porque en un vuelo real, normalmente se hace con el mando remoto.
- **disarm:** Desarma el vehículo si está armado con lo que no seguirá ninguna instrucción. Usado en simulación principalmente.
- **guidedmode:** Como se menciona en la sección 3.1.1 el UAV debe estar en modo guiado para ejecutar instrucciones de una PC, por lo cual se hizo esta función que ordena a un UAV cambiar a modo guiado y espera hasta que esto se haga para salir de la función a ejecutar más instrucciones. También

es usado principalmente en simulación porque desde un mando remoto se puede hacer fácilmente.

- **elev_eval:** Eleva un UAV a una altura especificada usando la función take off, y constantemente avisa cuánto ha subido. Cuando alcanza la altura deseada da un mensaje y termina la función, así ninguna orden se sobreescribe al take off antes de que haga su trabajo. La función puede usarse en vuelos reales, pero normalmente los usuarios prefieren elevar un UAV con un mando remoto, ya que les da más control del proceso de elevación que ya de por sí es riesgoso.

A partir de ahora las funciones se usan tanto en vuelos reales como simulación y en particular las siguientes 4 funciones usan las variables globales gui2other y valid_disc, para que si se da una desconexión el programa sepa qué hacer tanto en el momento de que el modo deja de ser guiado como cuando se da la reconexión. Para esto, en el programa de vuelo se deben usar también estas variables en ciclos con instrucciones de movimiento del UAV como se explica en el primer ciclo “While True” que aparece en el código del apéndice 10.4 (página 84).

- **manual_elev:** Es la alternativa a elev_eval para monitorear la altura del UAV cuando está siendo elevado manualmente con un mando remoto por ejemplo. De igual forma avisa cuando se alcanza la altura deseada y le indica al usuario que puede activar el modo guiado para que la PC sea quien dé las instrucciones.
- **send_ned_velocity:** Permite enviarle a un UAV la orden de moverse con un vector de velocidad en m/s con 3 componentes cartesianas por un tiempo definido, dando como resultado un vector aproximado de desplazamiento.
- **eval_distxy:** Una vez que se ha dado un simple_goto a alguna dirección gps, esta función permite dar un seguimiento en el plano horizontal del movimiento del UAV, indicando periódicamente cuántos metros faltan para llegar al punto deseado; al llegar a una distancia aceptable definida por el usuario (típicamente menos de 1 metro) avisa que se llegó y se termina la función.

- **eval_distxyz:** La diferencia con la función anterior es que esta da un seguimiento por el espacio tridimensional, ya que también considera la altura.
- **get_location_metresxyz:** Teniendo un punto gps como referencia para un vector de desplazamiento en metros al que se desea que llegue el UAV, esta función permite transformar dicho vector, a una posición con coordenadas gps válidas para usar simple_goto. Esta última trabaja con direcciones obtenidas através de las funciones de dronekit LocationGlobal o LocationGlobalRelative.
- **get_location_metresxy:** la diferencia con la función anterior es que esta sólo admite un vector en el plano horizontal, por lo que le deja la misma altura que el punto usado como referencia.
- **bat_info:** siempre que el usuario indique valores mínimos admisibles de la tensión de la batería y porcentaje de carga, esta función hace una revisión del valor de estas variables e indica al usuario si se corre peligro. Es llamada en otras funciones de estructurauav, con lo que mientras se monitorea el vuelo, también se indicaría si queda poca batería.
- **printdelay:** usada en otras funciones de la biblioteca, esta función permite contar un tiempo de espera, principalmente después de un movimiento del UAV para darle la posibilidad de estabilizarse.

5.2.4 Alimentación

Previamente los UAVs ya contaban con dos baterías en paralelo de 5000 mAh de energía cada una a 11.1 V que daban alimentación al autopiloto y los ESC de los motores; el Pixhawk contaba con un convertidor dc-dc que permitía pasar los 11.1 V a los 5 V que requiere, mientras que los ESC se conectan directamente a las baterías. El consumo total de las dos baterías juntas llegaba hasta 20 A máximo durante el vuelo.

Con la estructura autónoma surgieron dos elementos más que alimentar en el vehículo: la puerta de enlace con rango de alimentación de 7 V a 30 V y potencia de 1W, así como la NUC con una tensión de 19 V y corriente de 3.43 A. Por su parte, el convertidor del Pixhawk se eliminó debido a que al conectarse este a la

mini PC vía USB no sólo comparte información, sino que también obtiene su energía.

Para la puerta de enlace fue tan sencillo como conectarla a una terminación extra de una de las baterías, sin embargo, para la mini PC la tensión era insuficiente, por lo que se optó por usar el circuito integrado XL6019; un convertidor dc-dc, reductor-elevador del que ya se disponía en el GII. Además a la salida de este se soldó un conector de PC macho compatible con la NUC.

Todo el circuito de alimentación de la mini PC se puede apreciar en la Figura 5.2.7. Adherido al chasis del UAV se encuentra el dc-dc, el cual está unido por dos cables soldados a un conector amarillo por el que recibe alimentación de la batería y en el otro extremo se observa el conector macho que va a la PC.

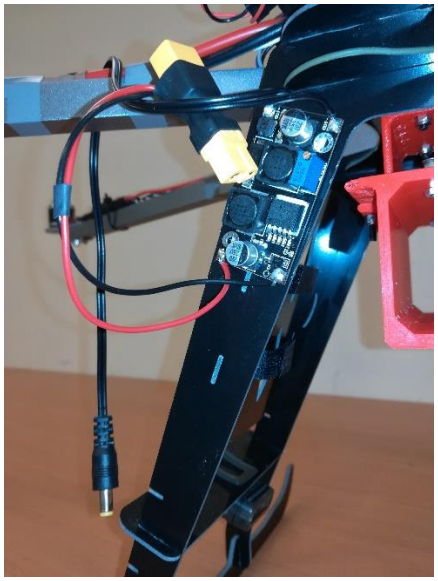


Figura 5.2.7 Circuito adaptador de alimentación de la Mini PC. Fuente: elaboración propia.

Por su parte, en la Figura 5.2.8 se muestra un diagrama del flujo de energía desde las baterías a todos los elementos del UAV.

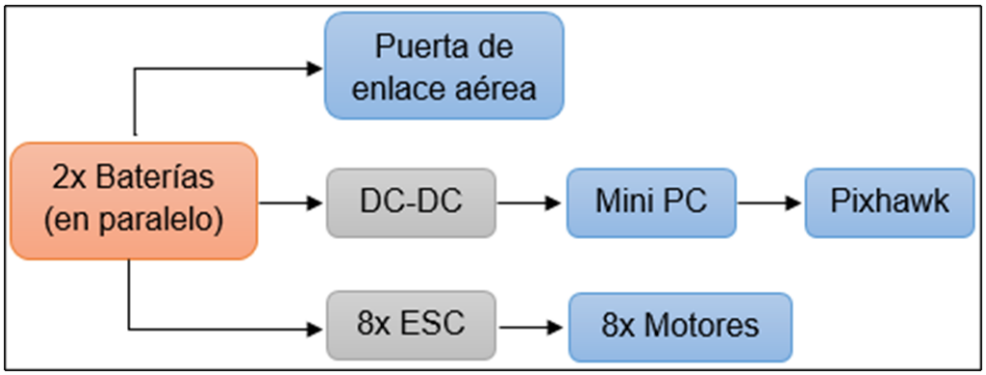


Figura 5.2.8 Flujo de Energía en el UAV. Fuente: Elaboración propia.

5.2.5 Soporte mecánico temporal

Se necesitó un soporte mecánico para que el UAV pudiera llevar tanto a la NUC como a la puerta de enlace. La primera se puede considerar como un prisma rectangular cuyo frente mide 115 mm, su costado 111 mm y una altura de 35 mm, con dos agujeros ubicados en la parte inferior, tal y como se ve en la Figura 5.2.9, los cuales tienen rosca M3 x 5 para ser sujetados a alguna superficie.

Figura 5.2.9 Vista Inferior de la Mini PC. Fuente: elaboración propia.

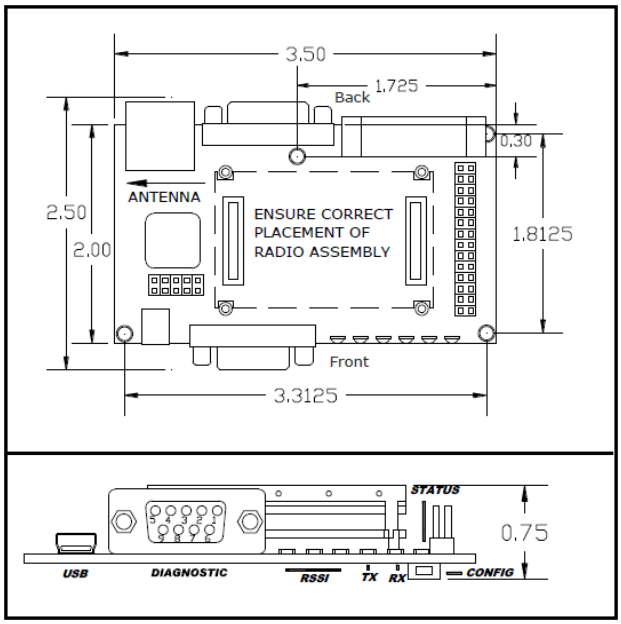


Figura 5.2.10 Dimensiones de la puerta de enlace (cotas en pulgadas). Fuente: [18]

Por su parte, en la Figura 5.2.10 se muestran las dimensiones de la puerta de enlace, así como 3 agujeros que sirven para sujetar el dispositivo con pernos también de 3 mm de diámetro.

Tomando en cuenta estas dimensiones y las del UAV pequeño en el que se probó la estructura autónoma, además de que se estaba contra el tiempo por las malas condiciones climáticas para volar, se optó por una solución rápida; Aprovechando que el GII tenía una impresora 3D, se diseñó e imprimió un soporte mecánico para los dos elementos anteriormente mencionados.

Por la necesidad de una solución rápida, el diseño prácticamente no tomó en cuenta ahorro de material ni mayores facilidades constructivas, pero sí tomó en cuenta que se cumpliera con las dimensiones de los equipos que iba a transportar y que soportara la carga mecánica. El soporte consta del contenedor de la Figura 5.2.11 y 2 sujetadores que lucen como el ensamble de la Figura 5.2.12.

La mini PC se sujeta a la base del contenedor con 2 tornillos M3 x 10, mientras que en la cara superior del contenedor se sujeta la puerta de enlace con 3 pernos M3 x 16, mismos que se usan para unir el contenedor a los dos sujetadores y estos al UAV.

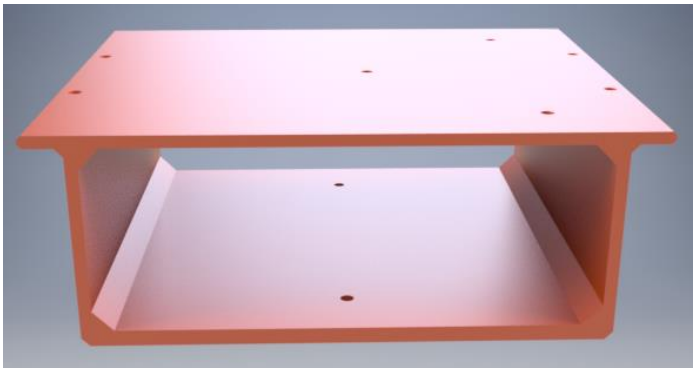


Figura 5.2.11 Contenedor de la mini PC y la puerta de enlace. Fuente: Elaboración propia.

En la Figura 5.2.13 se aprecia el soporte sujeto al UAV sosteniendo la Mini PC y la puerta de enlace ligeramente oculta. Esta última se identifica por la antena que le sobresale y está colgando frente a la NUC. Por otro lado, en la Figura 5.2.14 se aprecian las dos baterías que están a los costados del UAV (desconectadas), así como la Mini PC conectada a su circuito de alimentación (el cable que tiene conectado más a la izquierda) y a la puerta de enlace por medio de un cable Ethernet (el que está enrollado sobre el UAV).

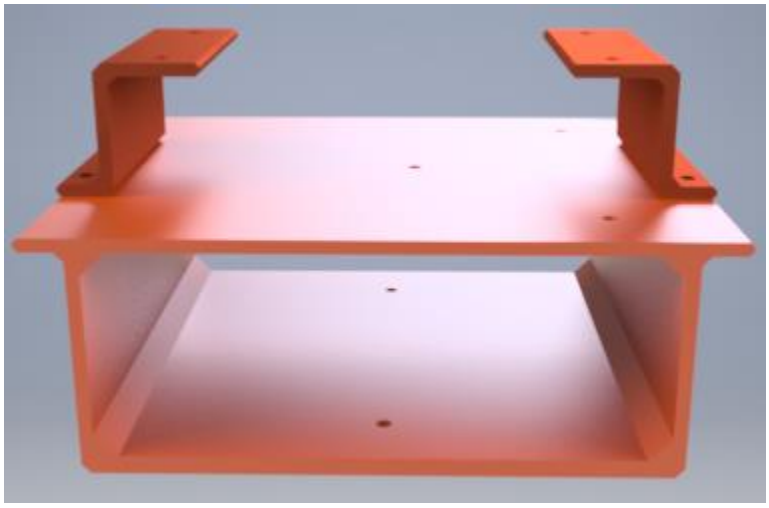


Figura 5.2.12 Soporte mecánico de la Mini PC y la puerta de enlace. Fuente: Elaboración propia.

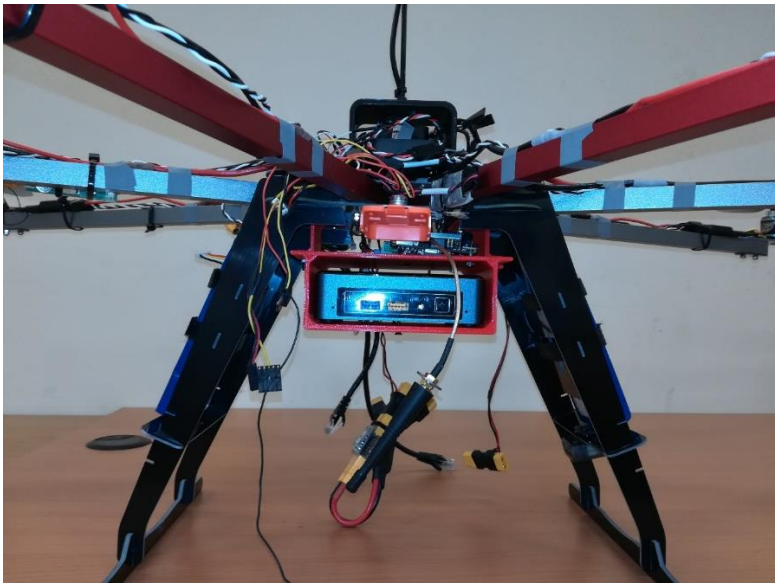


Figura 5.2.13 Vista frontal del UAV autónomo. Fuente: Elaboración propia.

Figura 5.2.14 Parte trasera del UAV autónomo. Fuente: Elaboración propia.

Figura 5.2.15 Vista total del UAV autónomo. Fuente: Elaboración propia.

En la Figura 5.2.15 se tiene una vista general del UAV autónomo donde se ve que para cada hélice hay un motor conectado a un ESC (dispositivos negros sobre los brazos del UAV), mientras que en la parte superior se encuentra el GPS que tiene apariencia de una especie de antena situado sobre una carcasa protectora que contiene al Pixhawk. En el apéndice 10.2 se explica el procedimiento para la colocación de cada componente.

El sujetador se imprimió con la técnica FDM, usando material PLA. Se sabe que no se imprimió con densidad completa, pero se desconoce el valor exacto y cómo afecta a su resistencia mecánica, por lo cual se indagó en varias fuentes bibliográficas para encontrar propiedades mecánicas realistas.

[21] hace un estudio de estructuras impresas con 20%, 50% y 100% de densidad con material ABS. Por su parte, en [22] y [23] se presentan propiedades mecánicas del PLA y ABS con densidad completa, donde el primero tiene una resistencia a la tensión promedio de 54 MPa, frente a 28 MPa del ABS. Por esta

razón se asumirá que se imprimió en ABS con 20% de densidad para verificar que el soporte no falla, ya que si no lo hace con dicho material, no lo hará con PLA.

Se puede intuir que el mayor esfuerzo se da en las paredes verticales de los dos sujetadores, debido a que su sección transversal es inferior a la de las paredes del contenedor. La Figura 5.2.16, obtenida con el software [24], muestra que la sección transversal de la pared es un rectángulo de 56 mm x 3 mm, además de un corte donde se puede notar que hay un esfuerzo máximo compuesto por la suma del esfuerzo axial y el de flexión.

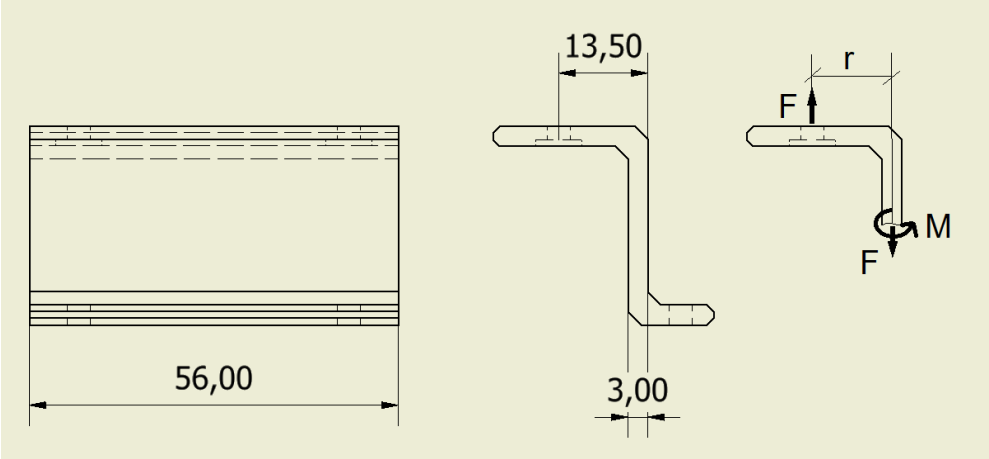


Figura 5.2.16 Dimensiones relevantes (mm) para el cálculo de esfuerzo en el sujetador. Fuente: Elaboración propia.

La carga total (m) del sujetador se compone de la masa de la NUC (1 kg), de la puerta de enlace (70 g), de la estructura impresa, pernos y tuercas (130 g las 3 últimas cosas):

$$m = 1 + 0.07 + 0.13 = 1.21\ kg$$

El peso w está dado por:

$$w = m \cdot g = 1.21 * 9.81 = 11.87\ N$$

. En la Figura 5.2.16, F corresponde a la mitad de w por haber 2 sujetadores, mientras que M es el momento debido a F [25], con una distancia r que va desde el centro del agujero para el tornillo hasta la mitad de la pared:

$$F = \frac{w}{2} = \frac{11.87}{2} = 5.93\ N$$

$$r = 13.5 - 1.5 = 12\ mm$$

$$M = F \cdot r = 5.93 \cdot 12 = 71.2\ N \cdot mm$$

Los esfuerzos axial σ_{ax}, de flexión σ_f y total se calculan según [25]:

$$\sigma_{ax} = \frac{F}{A} = \frac{5.93}{56 \cdot 3} = 0.0353\ MPa = 35.3\ kPa$$

$$\sigma_f = \frac{M \cdot c}{I} = \frac{M \cdot c}{\frac{b \cdot h^3}{12}} = \frac{71.2 \cdot 1.5}{\frac{56 \cdot 3^3}{12}} = 283\ kPa$$

Donde I es el momento de inercia de la sección rectangular (con base b de 56 mm y altura h de 3 mm) con respecto a su fibra neutra y c es la distancia entre esta última y el punto donde se calcula el esfuerzo (superficie de la pared), es decir 1.5 mm (la mitad del espesor de la pared). Por su parte, es esfuerzo total es la suma del axial y el de flexión:

$$\sigma_t = \sigma_{ax} + \sigma_f = 35.3 + 283 = 318\ kPa$$

Por tratarse de un elemento con esfuerzo en una única dirección se puede emplear la teoría del esfuerzo normal máximo de [26] para obtener el factor de seguridad. [21] reporta una resistencia a la tensión de 15.62 MPa para el ABS con 20% de densidad. El factor de seguridad Fs para la ruptura se calcula a continuación:

$$Fs = \frac{15.62}{0.318} = 49.1$$

Por otro lado, en la Figura 5.2.17 se muestra una simulación de carga con [24], donde se indica que la máxima deflexión se da en el contenedor producto de cargar la NUC (la cual tiene 4 puntos de apoyo) y la puerta de enlace (asumido como una carga puntual), pero es de apenas 0.02 mm, mientras que el máximo admisible era 2 mm, de manera que no se apreciara a simple vista.

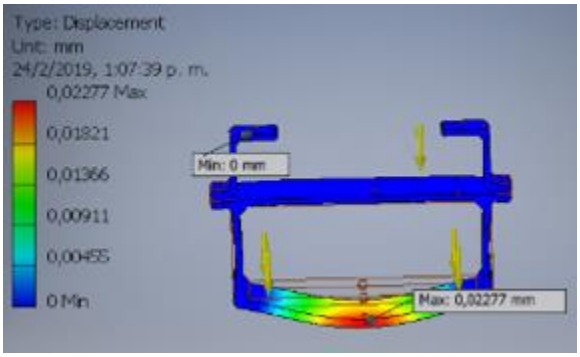


Figura 5.2.17 Deflexión máxima en el contenedor temporal. Fuente: elaboración propia.

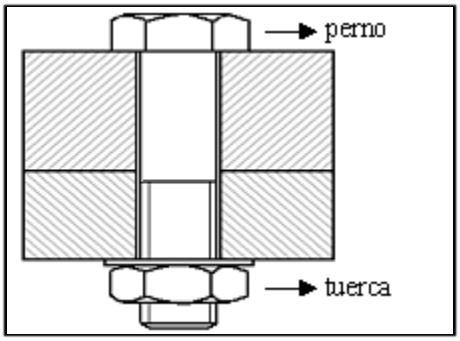


Figura 5.2.18 Unión por perno de dos elementos. Fuente: [27]

En cuanto a las juntas roscadas que se emplearon, estas lucen como la de la Figura 5.2.18. Las 4 que unen el soporte al UAV son las que llevan la mayor carga, ya que sostienen la totalidad del peso de la estructura, mini PC y puerta de enlace. Cada junta soporta una carga P igual a la cuarta parte de w, es decir 2.97 N. Según [26], la carga de la junta se distribuye entre el perno y los elementos sujetados de la siguiente manera:

$$Perno: F_b = C \cdot P + Fi$$

$$Material\ sujetado: F_m = (1 - C) \cdot P - Fi$$

Donde F_b y F_m son la fuerza que soporta cada perno y par de elementos sujetados, respectivamente. C es un factor cuyo valor está entre 0 y 1 e indica cuánto de la carga P es asumido por el perno. Por último, Fi es conocida como la precarga; es la fuerza a la que se ve sometida el perno producto de las vueltas que se le den a la tuerca hasta que se tenga un apriete deseado [26].

Según [26] un perno M3, tiene un área de esfuerzo a tensión A_t de 5.03 mm^2 y una resistencia de prueba S_p (la carga que genera una deformación permanente) de 310 MPa. Haciendo una aproximación conservadora para la fuerza soportada por el perno igual a 6 veces la carga de la junta (C=1) [26] se obtiene el siguiente factor de seguridad para el perno:

$$Fs = \frac{S_p}{F_b/A_t} = \frac{310}{6x2.97/5.03} = \frac{310}{3.54} = 87.6$$

Por su parte, suponiendo que en el material sujetado (asumido completamente como ABS al 20% de densidad) F_m se distribuye en un anillo con diámetro interno igual al del perno y diámetro externo igual a 1.5 veces el interno, se puede calcular el FS haciendo la misma aproximación anterior para la fuerza soportada por el material ABS:

$$Fs = \frac{15.62}{6 * 2.97\ /[(4.5^2 - 3^2) * \pi/4]} = \frac{15.62}{2.02} = 7.73$$

Por último, el circuito de alimentación del que se habló en la sección 5.2.5 también hubo una sujeción mecánica temporal. En este caso el integrado XL6019 se adhirió al chasis del UAV con cinta doble propósito, como se puede observar en la Figura 5.2.7. Sin embargo, tanto este convertidor como otros, tienen agujeros para pernos, con lo que podrían sujetarse al UAV de manera más segura.

5.2.6 Soporte mecánico propuesto

Una vez que se llevó a cabo el vuelo de prueba, se mejoró el diseño para el soporte mecánico de manera que se redujo considerablemente el material empleado, y por lo tanto el peso y tiempo de impresión, mientras que se aseguró de que no fallaría mecánicamente. En la Figura 5.2.19 se presentan varias vistas de dicho soporte (diseñado con [24]) incluyendo una representación simbólica de la mini PC y puerta de enlace, mientras que los planos están en el apéndice 10.5.

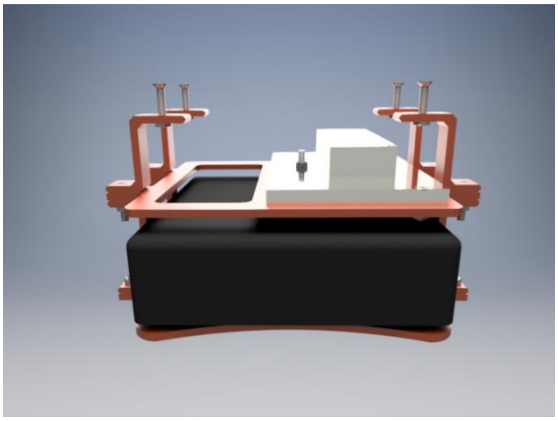


Figura 5.2.19.a Vista Frontal

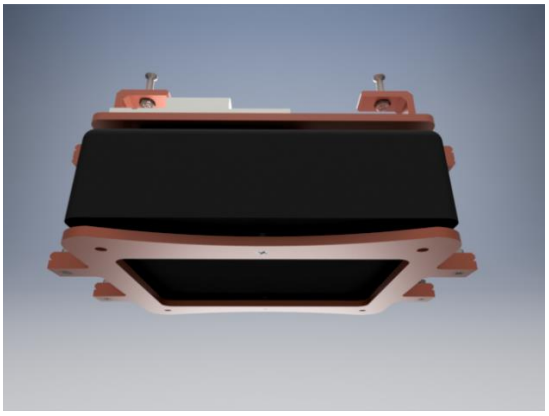


Figura 5.2.19.b Vista Inferior

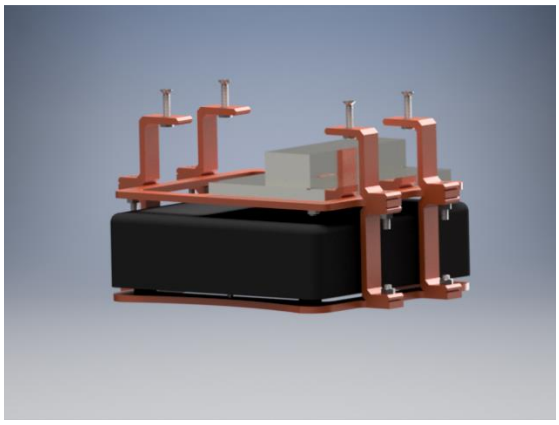


Figura 5.2.19.c Vista lateral

Figura 5.2.19 Diseño mecánico propuesto. Fuente: Elaboración propia.

En cuanto a la fabricación de las piezas no estandarizadas (las que tienen un color rojizo en la Figura 5.2.19) se volvió a elegir la técnica FDM, pero en esta ocasión se considera que el material ABS tiene mejores prestaciones que el PLA, debido a que tiende a ser más duradero, más resistente a golpes (que en caso de que el UAV se estrelle sería una ventaja) y permite pintarse más fácilmente, así como darle acabados superficiales con acetona [28].

Además, [28] recomienda que si las piezas impresas están encajadas con otras, sometidos a esfuerzos mecánicos, vibraciones, temperatura o sol, se emplee material ABS. Este último presenta una desventaja con respecto al PLA y es que no es biodegradable, sin embargo es reciclable, con lo cual se reduciría su impacto ambiental.

Por otro lado, si sólo se dispone de PLA no es un problema, la estructura sería funcional, pero se perderían cualidades del ABS que lo hacen ligeramente más apto para aplicaciones de un UAV, expuestas a sol, vibraciones e incluso golpes (nunca son deseados, pero en ocasiones los hay).

El interior de la Intel NUC (la PC como tal) se sujeta a su carcasa negra por medio de los 4 tornillos que se observan en la Figura 5.2.9. Dicha carcasa representa un peso cercano al 50% del total de la NUC, razón por la cual se puede considerar sustituir dicha carcasa por una más liviana. Previendo esto, se hicieron pequeños agujeros en los puntos de apoyo, de manera que si se retira la carcasa o se cambia, la PC puede sujetarse con los mismos 4 tornillos.

Para determinar los esfuerzos mecánicos en el soporte se usó [24] para simular la carga con el mismo ABS al 20% de densidad con el que se supuso el diseño del primer soporte. En la Figura 5.2.20 se muestran los esfuerzos de Von Mises, donde el máximo es 5.09 MPa.

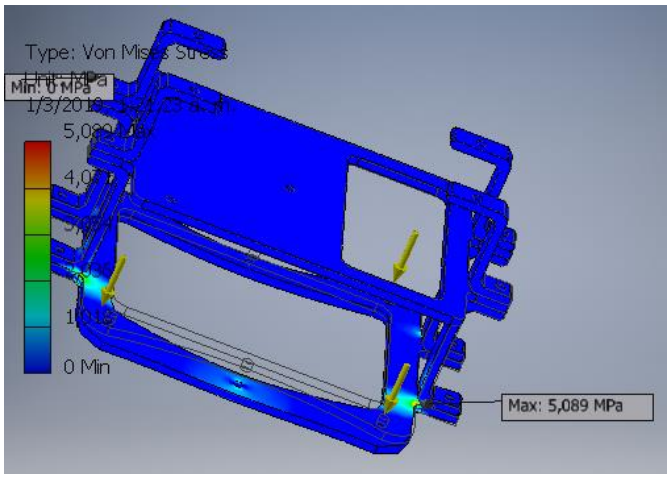


Figura 5.2.20 Esfuerzos de Von mises en el nuevo soporte mecánico. Fuente: elaboración propia.

Según la teoría de la energía de distorsión [26] el FS se calcula como sigue a continuación:

$$FS = \frac{15.62}{5.09} = 3.07$$

Con dicho factor de seguridad se puede asegurar que el soporte no fallará mecánicamente. Por otro lado, en la Figura 5.2.21 se muestra que la máxima deflexión es de 0.75 mm, lo cual está dentro del rango aceptable de 2 mm.

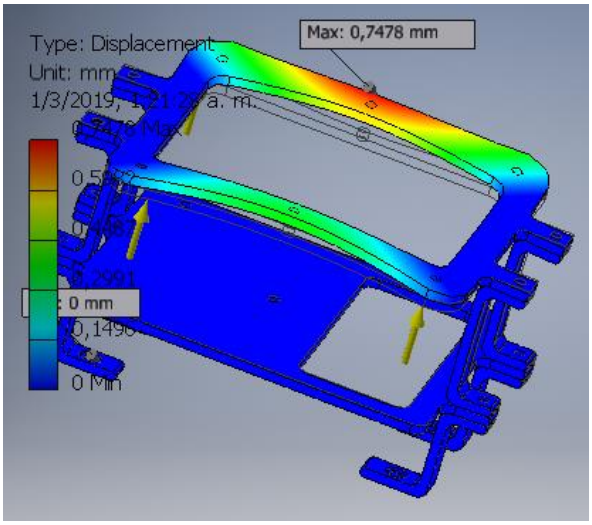


Figura 5.2.21 Deflexión en nuevo soporte mecánico. Fuente: elaboración propia.

Por último, en la sección anterior se vio que los tornillos del primer soporte mecánico resisten la carga. Este diseño tiene un peso menor, razón por la cual los tornillos siguen teniendo un factor de seguridad apropiado.

Capítulo 6. Análisis de Resultados

A continuación, se muestran los resultados de los procesos de verificación y validación de la estructura creada con su respectivo análisis.

La verificación es un conjunto de pruebas que se hacen después de cada etapa crucial de la solución que se está dando al problema planteado, esto para asegurar que se cumpla con todos sus requerimientos previo a su uso real, es decir, tener un fundamento teórico para asegurar que la solución es funcional.

La validación viene a ser un conjunto de pruebas diseñadas de manera tal que ya teniendo la solución implementada se pueda corroborar que la solución desarrollada puede hacer lo que se supone que debe hacer.

6.1 Verificación de funcionalidad de la estructura creada

La verificación de funcionalidad se dio en 2 etapas; una de hardware y una de software. En la primera se verificó que el hardware incorporado permitiera el intercambio de información entre el Pixhawk, mini PC y computadora de control en tierra (CCT), mientras que en la segunda etapa se verificó a través de la simulación SITL que la biblioteca de software fuese útil para programar rutinas de vuelo.

6.1.1 Verificación del hardware de la estructura del UAV autónomo

Conforme se iba añadiendo hardware se iban haciendo pruebas a manera de verificación; lo primero que se verificó es que la red creada con las puertas de enlace sirviera de canal de comunicación entre la CCT y la NUC, para lo cual se hizo un Ping desde la CCT a esta última, cuyo resultado se muestra en la Figura 5.2.3, lo cual indica que la conexión está hecha.

Sin embargo, se dan retardos inevitables en la comunicación debido al tiempo que toman las conversiones de formato de los datos, tanto en la mini PC como en las puertas de enlace (de Ethernet a radio señal y viceversa, por ejemplo). Aún así, la visualización de la telemetría en la CCT se redujo considerablemente; se ve prácticamente al instante.

Por otra parte, durante la ejecución de programas desde la CCT en la NUC vía SSH (Figura 5.2.6) hubo retardos de duración variable (menores a 7 segundos) en la visualización del terminal, sin embargo, los colaboradores del área de UAVs del GII habían usado los anteriores módems para ejecutar instrucciones vía SSH desde un PC en otro y habían tenido retardos mayores, por lo que también hubo mejoría.

6.1.2 Verificación del software de la estructura del UAV autónomo

Se escogió simulación SITL para efectuar la verificación, ya que el Mission Planner (programa con el que lo colaboradores del área de UAVs del GII estaban más familiarizados) permitía hacerlo fácilmente. El HITL ofrece la ventaja de procesar las señales directamente en el autopiloto que se va a usar en un vuelo real, sin embargo se ha volado muchas veces con el Pixhawk y se sabe que es funcional, además de que el modelo del simulador SITL es fiel y confiable.

Por otro lado, si bien el proyecto buscaba crear una estructura de hardware y software para UAVs autónomos, la biblioteca que se creó también funciona para comportamiento no autónomo o mixto, lo cual le da un valor adicional. Se simularon varios programas de vuelo, de los cuales se destacan 2 no autónomos y 1 autónomo, cuyos resultados se comentan a continuación.

El primer programa de vuelo fue una labor común para un UAV; consistió en tomar 4 vectores de desplazamiento en metros, transformarlos a coordenadas GPS con get_location_metresxyz y hacer que el UAV fuera a todos estos puntos en un orden escogido por el programador. En cada punto el UAV debía descender 5 metros, volver a subirlos y seguir hacia el siguiente punto, como si se tratara de la entrega de un paquete. En la Figura 6.1.1 se aprecia el trayecto del UAV desde su despegue hasta que llegó al primer punto y descendió.

Una vez que el UAV recorriera cada uno de los puntos debía volver a su punto de despegue, para lo cual se hizo uso del modo RTL que precisamente lleva el vehículo hasta su *“home position”* independientemente de dónde se encuentre.

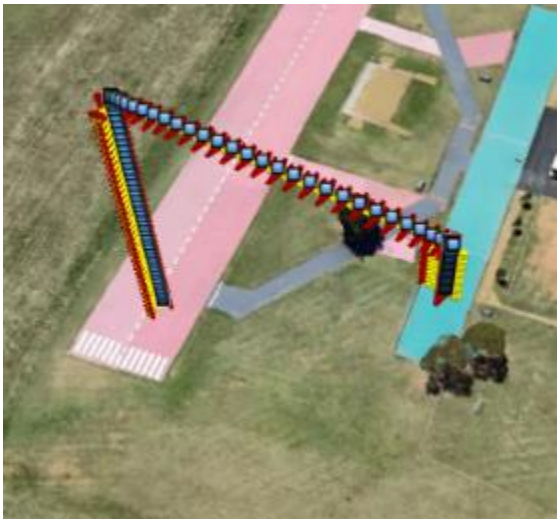


Figura 6.1.1 Primer punto de la trayectoria simulada. Fuente: Elaboración propia.[2]

El trayecto completo está en la Figura 6.1.2. Las líneas verticales permiten que el lector ubique correctamente cada punto de la trayectoria, ya que van desde el suelo hasta la altura de cada punto. La parte en color anaranjado fue en modo guiado, mientras que la amarilla en modo RTL. Se pueden apreciar cada una de las 4 paradas con su respectivo descenso, además la distancia en tierra desde el punto de despegue hasta la primer parada (línea horizontal amarilla) tomada con [29], que se programó para que fuera 50 m.

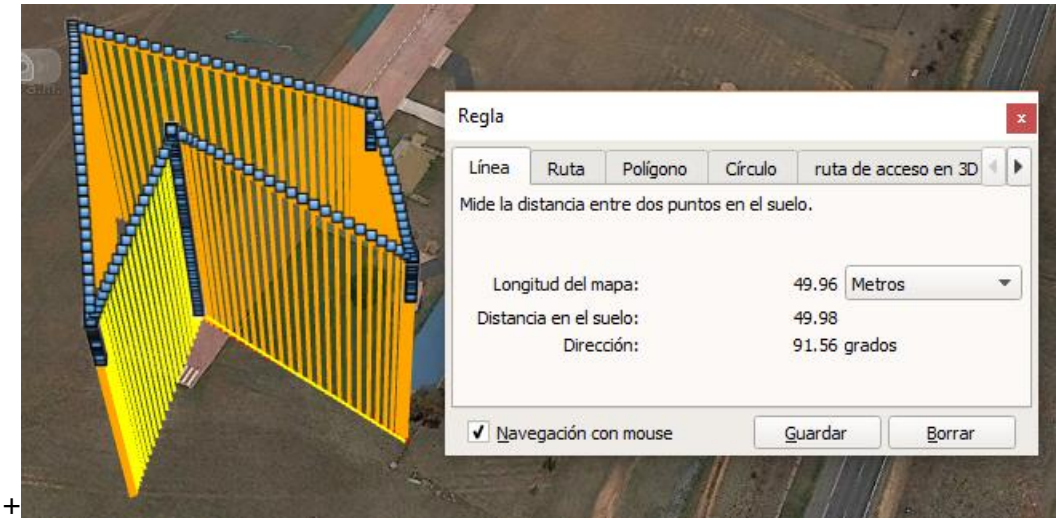


Figura 6.1.2 Trayectoria simulada, con la medida de la distancia en tierra desde el punto de despegue hasta la primer parada. Fuente: elaboración propia.

[2] La imagen es propiedad del autor del escrito, no así el software empleado [29].

El hecho de que la medida no sea 50 m exactos se debe a varios factores. El primero es que la función get_location_metresxyz usa una aproximación de Tierra esférica para hacer el cálculo de la longitud y latitud a la que el UAV debe ir a pesar de que es más que sabido que el planeta no es una esfera perfecta, y aunque no lo es, en muchas fuentes bibliográficas se encuentran diferentes valores del radio de la tierra (valor necesario en el cálculo).

Por otro lado, se usó la función eval_distxy, que también usa una aproximación de tierra esférica y revisa cada cierto tiempo la distancia faltante, y si en el momento que revisa el UAV está dentro de la distancia aceptable (típicamente menos de un metro) da paso a ejecutar más instrucciones, con lo cual, normalmente, el UAV recorrerá una distancia ligeramente menor a la que fue programada, ya que además habría que tener mucha suerte para que en el momento justo en que se revisa se esté exactamente en el punto al que se debía llegar.

Al seguir experimentando con rutinas de vuelo se vio la capacidad que se tiene para que un UAV dibuje figuras en el aire mediante comandos de velocidad con la función send_ned_velocity. Por ejemplo, para dibujar un cuadrado con 10 m de arista, basta con 4 llamados a dicha función, una que haga que se mueva a 1 m/s por 10 s hacia el norte, otra hacia el este, al sur y al oeste hasta llegar al mismo punto, mientras que la forma complicada implica tener un punto de referencia, obtener los vectores de cada uno de los vértices respecto a esa referencia, transformarlos a coordenadas gps y dar un simple_goto a cada vértice seguido de eval_distxy.

Repetir este proceso largo para dibujar un círculo o un senoide es aún más tedioso por la forma de dichas trayectorias y todos los puntos que habría que generar, pero resulta más sencillo con comandos de velocidad. En la Figura 6.1.3 se muestran una imagen de [29] mostrando una simulación en la que el UAV despega, hace una trayectoria senoidal, se mueve hacia otro punto para dibujar un círculo y al volver al mismo punto se traslada un poco más para hacer un cuadrado. El código que lo hizo posible está en la Figura 6.1.4.

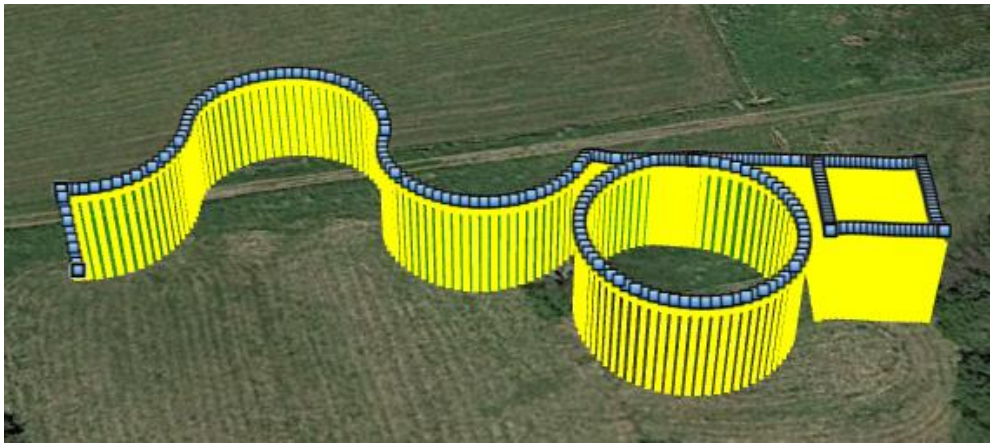


Figura 6.1.3 Figuras obtenidas en simulación. Fuente: Elaboración propia.

```
print "Senoidal path"
for i in range(0,43):
    Vx=10*0.1745*math.sin(0.1745*i)
    Vy=math.fabs(10*0.1745*math.cos(0.1745*i))
    send_ned_velocity(Frankie,Vx,Vy,0, 1,1)
#the derivatives of the senoidal function with respect to both
#axes are calculated and sent each second
send_ned_velocity(Frankie,0,0,0, 1,1)
#the call with velocity=0 is for stabilize the vehicle
send_ned_velocity(Frankie,0,1,0, 10,1)
send_ned_velocity(Frankie,0,0,0,2,1)

print "Circle\n"
for i in range(0,36):
    Vx=-10*0.1745*math.sin(0.1745*i)
    Vy=10*0.1745*math.cos(0.1745*i)
    send_ned_velocity(Frankie,Vx,Vy,0, 1,1)
send_ned_velocity(Frankie,0,0,0, 1,1)

send_ned_velocity(Frankie,0,1,0, 12,1)
send_ned_velocity(Frankie,0,0,0,2,1)

print "Now a Square"
send_ned_velocity(Frankie,-1,0,0,10,1)
send_ned_velocity(Frankie,0,0,0,2,1)
send_ned_velocity(Frankie,0,1,0,10,1)
send_ned_velocity(Frankie,0,0,0,2,1)
send_ned_velocity(Frankie,1,0,0,10,1)
send_ned_velocity(Frankie,0,0,0,2,1)
send_ned_velocity(Frankie,0,-1,0,10,1)
send_ned_velocity(Frankie,0,0,0,2,1)
```

Figura 6.1.4 Código para obtener figuras con send_ned_velocity. Fuente: Elaboración propia.

Tal y como se muestra en la Figura 6.1.4, para el círculo y senoide se usan ciclos “for” que están calculando en el tiempo la velocidad que debe tomar el UAV para dibujar dichas figuras. Según se explica en el apéndice 10.3.2 (biblioteca de software), send_ned_velocity tiene 6 argumentos; el primero es el nombre del UAV, los siguientes 3 son los componentes de velocidad en distintos ejes, posteriormente el tiempo durante el que se quiere que viaje con esa velocidad y el último es un periodo de refresco de la orden.

El movimiento fue en un plano horizontal, pero pudo haber sido en 3 dimensiones. La componente que aparece en cero en los llamados a la función dentro de los ciclos “for”, es la vertical y si por ejemplo, en el ciclo del círculo se le hubiera dado un valor distinto de cero, se habría generado un helicoide. Por su parte las llamadas a la función con las 3 componentes de velocidad en cero hacen que el UAV se estabilice.

En el caso del cuadrado, hay 8 llamados a la función en cuestión, donde 4 son nada más de estabilización (velocidad cero) y los otros generan las aristas. Si se lee la documentación de la función en estructurauav se puede entender que el cuadrado se generó moviendo el vehículo en orden sur, este, norte y oeste.

Una vez que se efectuaron los programas de vuelo anteriores, se hizo la simulación de un vuelo autónomo el cual usó inteligencia artificial para definir su propia ruta, cuyo código se muestra en el apéndice 10.4. El objetivo de dicho programa fue probar que el software permite comportamiento autónomo, que es lo que buscaba el proyecto.

Es importante recordar que estos programas están pensados para estar en una mini PC a bordo del UAV, con lo cual todo el procesamiento se daría también a bordo del vehículo, indicándole al Pixhawk las instrucciones de manera directa y comunicando a una PC en tierra tanto las decisiones que está tomando como el estado de vuelo.

Aprovechando el ejemplo de la Figura 6.1.2, se ideó, entre las tantas aplicaciones que puede tener un UAV autónomo, una en la que simula un UAV

repartidor de paquetes al cual le dan un conjunto de puntos donde debe entregar paquetes, pero sólo puede llevar un máximo de 4 a la vez, así que por medio de un algoritmo genético genera aleatoriamente posibles rutas y las muta, buscando hacer la más corta.

Una vez que el programa escoge su ruta, la ejecuta de manera completamente autónoma. Al llegar a cada punto, desciende 5 metros (como si estuviera “entregando un paquete”) y pregunta al usuario si ya “recibió el paquete”, vuelve a subir y continúa con el vuelo. En la Figura 6.1.5 se brinda una imagen de [29] con la trayectoria obtenida en la simulación.

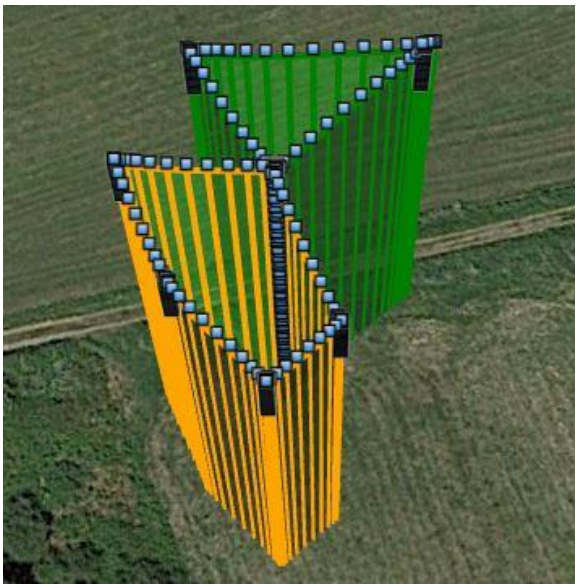


Figura 6.1.5 Simulación de vuelo autónomo. Fuente: Elaboración propia.

En la Figura 6.1.5 se observa un punto central que es el de despegue. Al programa se le dieron 6 puntos a visitar, por lo que requirió de dos vuelos. En el primero escogió los 4 puntos de la ruta en anaranjado (ruta óptima), se observa que en cada uno de ellos descendió y volvió a subir una vez “entregado cada paquete”. Por último, volvió al punto de despegue a “recargar” y entregar los 2 restantes (ruta en verde).

Al finalizar esta simulación se probó que el UAV es capaz de crear su propia ruta y ejecutarla. La biblioteca permite dar algunas instrucciones y monitorear el vuelo, sin embargo, no incluye ninguna funcionalidad especial que dote de

autonomía al vehículo, debido a que las aplicaciones autónomas que puede tener un UAV son de naturalezas muy diferentes. El usuario debe programar la inteligencia artificial que le convenga y una vez que “toma una decisión”, la biblioteca ayuda a ejecutarla.

6.2 Validación de la estructura creada para el UAV autónomo

Para validar el trabajo realizado se llevó a cabo un vuelo real una rutina de comportamiento autónomo como la de la simulación que se comentó previamente (apéndice 10.4). Se llevó toda la estructura del vehículo autónomo; el UAV tal y como está en la Figura 5.2.15, una computadora de control en tierra (CCT) y un mando remoto para tomar control del vehículo en caso de ser necesario.

La rutina de comportamiento autónomo estaba en la misma mini PC del UAV, con lo que la inteligencia artificial se ejecutó a bordo del mismo vehículo. En la CCT se estaba viendo las decisiones que iba tomando el UAV por medio de la red. Por su parte, lo único que no hizo de manera autónoma fue elevarse y descender, ya que son procesos que prefieren hacerse manualmente (en un modo diferente al guiado) según las normas de seguridad del GII.

En la Figura 6.2.1 se presenta un recorte de pantalla de [30] que compara el primer trayecto que hizo el UAV (al ser 6 puntos requería de 2 vuelos) en la realidad y en simulación (imágenes izquierda y derecha respectivamente). El vuelo real empezó en el punto 6 y siguió al 3, 2 y 5. Al “entregar su último paquete” se dirigió hacia el punto H (de despegue); cuando estaba muy cerca se tomó control del UAV con el mando remoto (se cambió el modo) para descender manualmente. Este descenso y elevación mencionados no se muestran en la Figura 6.2.1.

Figura 6.2.1 Comparación entre el vuelo de prueba real (izquierda) y el simulado (derecha). Fuente: elaboración propia.

Respecto a la ruta elegida, si se compara el vuelo real con el simulado, en la Figura 6.2.1 se observa que el UAV no eligió el mismo orden de los puntos; el vuelo real fue a los puntos 6, 3, 2 y 5, mientras que el simulado fue a 6, 3, 5 y 2, respectivamente. Esto se debe en primer lugar a que la población inicial de rutas es generada al azar con el módulo “random” de python y no necesariamente va a ser igual cada vez que se ejecuta el programa, lo que puede estar sesgando las posibles soluciones, ya que la ruta elegida sale de la población inicial o de las mutaciones que sufra conforme se va iterando y eliminando las rutas más largas.

Tomando en cuenta que los puntos que el UAV podía escoger para la ruta simulada y la real están a las mismas distancias, se vio que el vehículo no eligió la ruta óptima en el vuelo de prueba (puntos 6, 3, 5 y 2, respectivamente). Para solucionar esto se hizo que también las rutas con una longitud corta respecto al resto (las mejores) sufrieran mutaciones, ya que antes sólo se mutaban las peores.

Por su parte, en la Figura 6.2.2 obtenida con [29] se ve que el UAV no se movió en líneas estrictamente rectas a diferencia de la simulación de la Figura 6.1.5, lo cual se debió principalmente a que en el lugar y momento en que se podía llevar a cabo el vuelo, era inevitable el viento, máxime que se voló a 40 metros sobre el nivel del suelo y dicha variable no se contempla en la simulación. Véase la parte

inferior izquierda de la Figura 6.2.1, donde se muestran la latitud (43.53073°), longitud (-8.09953°) y altura del UAV (39.94 m), respectivamente.

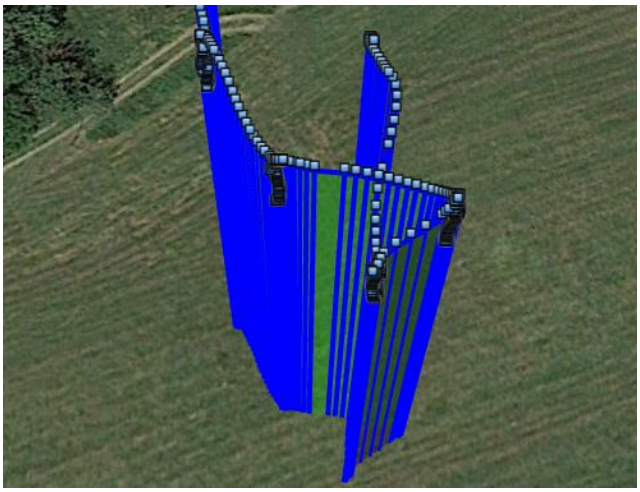


Figura 6.2.2 Vuelo de prueba real. Fuente Elaboración propia.

Si bien el UAV cumplió el objetivo de definir y ejecutar su propia ruta autónomamente, se hicieron otras modificaciones a la aplicación autónoma, además de la que se mencionó respecto a las mutaciones. En ese momento (después de haber volado) fue cuando se vio la oportunidad de incrementar las prestaciones de la estructura desarrollada, añadiendo código a la biblioteca y programas de vuelo que permitiera tomar acciones cuando se desconectaba el modo guiado. Además se vio que se necesitaba monitorizar mejor la batería.

Es así como producto de este vuelo se agregaron las variables globales que permiten que el UAV recupere el progreso que ha alcanzado en la misión asignada aunque se haya desconectado el modo guiado producto de alguna inestabilidad en el vuelo que haga al usuario pensar que es mejor tomar el control con el mando remoto. También se agregaron variables que permiten el monitoreo del estado de la batería y avisan cuando se llega a un valor insuficiente para mantener el UAV en el aire.

Todas las modificaciones anteriores ahora son parte de los códigos que se encuentran en el apéndice 10.3 y 10.4. Dichas modificaciones no se llevaron a un segundo vuelo de prueba, debido a que había llegado el invierno y en el lugar de

las pruebas había vientos cuya magnitud superaba los 15 km/h, condición climática que por norma de seguridad del GII impide volar un UAV. Sin embargo, fueron simuladas y funcionan a la perfección.

En cuanto a la mini PC, poco tiempo antes del vuelo de prueba se le detectó un sonido fuera de lo normal, se midió la corriente que le estaba llegando y era menor a los 3.43 A que requiere. Sin embargo, esta pequeña falta de corriente no fue determinante, la NUC aún funcionaba y debido a que se estaba contra el tiempo por las malas condiciones climáticas para volar, se decidió operar así y no hubo problema. No obstante, para un futuro uso lo ideal es adquirir un convertidor que pueda dar la corriente necesaria; se vio que hay varios modelos que dan hasta 5 A con la tensión y potencia necesarios.

Por último, el retardo obtenido durante el vuelo en la visualización del terminal de la mini PC vía SSH fue un factor negativo, ya que por ejemplo, cuando el vehículo llegaba a cada punto se anunciaba en el terminal hasta varios segundos después y por lo tanto, la confirmación de “entrega de paquete” que pedía también llegaba tarde, cuando había pasado tiempo muerto suspendido en el mismo punto sin hacer nada, exponiéndose al viento y agotando la energía de sus baterías, peligrando caer desde cerca de 40 metros de altura.

Se ejecutaron programas con las mismas 2 PCs vía SSH conectadas por medio de un cable Ethernet en vez de las puertas de enlace y el retardo fue inapreciable, lo que da entender que el problema no está en las PCs, sino en la red. Una variable que podría incidir es la tasa de transferencia de datos (*link rate*) de las puertas de enlace, cuyo manual [18] dice que entre menor sea, mayor es la potencia de salida de la señal de los dispositivos, dando a entender que se comporta como un filtro paso bajo.

El retardo no puede darse por falta de potencia de la señal, porque existe aun cuando los dispositivos están a menos de un metro de distancia, lo que indica que el problema podría deberse a que el *link rate* no fue lo suficientemente grande, por lo que en un futuro podría volverse a llevar a la práctica ampliando dicho valor

(en la Figura 10.1.6 se muestra la opción para hacerlo en la configuración de las puertas de enlace), siempre y cuando la disminución de la potencia de la señal no sea tan grande como para perder la comunicación.

Otra posible causa es que la potencia de las puertas de enlace sea tan grande que genera interferencias que podría afectar la comunicación y es algo que puede ajustarse según la Figura 10.1.6, siempre que dicha disminución de potencia no haga que se pierda la comunicación. Al final de cuentas, puede ser una combinación de ambos factores.

Capítulo 7. Análisis económico

Como se ha venido comentando a lo largo de este escrito, el GII ya tenía todos los materiales necesarios para el proyecto, sin embargo, se estimará el coste total del proyecto incluyendo el precio de los mismos, de manera que el lector pueda saber el valor económico que tendría replicar la estructura creada y se pueda comparar con los UAVs autónomos que se consiguen comercialmente. En la Tabla 7.1 desglosan los costos del trabajo realizado.

Tabla 7.1 Costos aproximados de la implementación del proyecto. Fuente: Elaboración propia.

Costos aproximados de la implementación del proyecto		
Descripción	Cantidad	Costo ($)
Meses de trabajo del desarrollador del proyecto	3,75	4375
UAV con Pixhawk y mando remoto	1	770
Horas de trabajo de compañeros	26,5	708
Intel NUC 7i5BNK	1	459
Puente Ethernet	2	378
Computadora de control	1	330
Baterías 5000 mAh	2	80
Impresión 3D	1	16
Convertidor dc-dc	1	1,6
Otros	-	25
Total	-	7142,6

El costo de comprar un UAV y convertirlo en autónomo alcanza unos $ 7143, donde el mayor porcentaje de dicho monto se debe a horas de investigación para idear la manera de resolver el problema. Sin embargo, este monto se reduciría drásticamente en una segunda implementación, debido a que el conocimiento ya está en este escrito, listo para usarse.

Los UAVs autónomos comerciales de tamaño similar al usado en este proyecto oscilan entre los 1500$ y 3000$, con sensores que les ayudan a tomar sus decisiones, pero presentan la desventaja de que están hechos para una o unas pocas aplicaciones, mientras que aplicar los conocimientos de este proyecto, permite que un UAV tenga infinita cantidad de aplicaciones, tanto autónomas como

no autónomas, con una gran flexibilidad de uso y la posibilidad de añadir los sensores y/o actuadores necesarios, así como de modificar la biblioteca de software a conveniencia.

Si bien comprar 3 vehículos autónomos de 2500$ hubiese tenido un costo muy similar a haber hecho este proyecto comprando todos los materiales, el trabajo desarrollado aquí tiene una mayor flexibilidad, porque se puede programar el UAV para que haga justo lo que se requiere, en vez de tener que idear cómo usar lo que ya está hecho y que no necesariamente resuelve el problema de manera directa, además de que no cualquier UAV comercial tiene un canal de comunicación con una computadora de control en tierra durante el vuelo. Esta flexibilidad es la que hace viable económicamente la solución empleada.

Capítulo 8. Conclusiones y recomendaciones

8.1 Conclusiones

Con base en el análisis de resultados se puede afirmar que se añadió una estructura flexible de hardware y software a un UAV, la cual le permite almacenar y ejecutar rutinas de vuelo, ya sea preprogramadas, manuales, autónomas o mixtas (parte manual y parte autónoma), que además puede comunicar a una computadora de control en tierra las acciones que está tomando al instante para una mejor monitorización del estado de vuelo, ventaja que no ofrece cualquier UAV comercial.

Se mejoró el tiempo de retardo para visualizar la telemetría en una computadora en tierra con el software mission planner; pasó de 2 s a ser casi inapreciable. Por su parte, el retardo para visualizar el terminal no fue lo suficientemente pequeño en la prueba de vuelo real, sin embargo es algo que puede modificarse en la configuración de la red para futuras aplicaciones.

Para el Grupo Integrado de Ingeniería el proyecto fue viable económicamente, porque no sólo se usaron equipos que ya tenía, sino que se le dio un valor funcional a materiales que estaban desaprovechando por desuso. El único gasto que asumieron fue el de las horas que sus colaboradores invirtieron cuando se requirió ayuda de los mismos. En todo caso se analizó que si el proyecto se implementa desde cero, su flexibilidad le da un alto valor que justifica la diferencia de precio con UAVs autónomos que se consiguen comercialmente.

8.2 Recomendaciones

- Adquirir y añadir un convertidor dc-dc cuya potencia sea suficiente para la mini PC.
- Usar la función elev_eval para hacer que un UAV despegue en un vuelo real sin intervención de un operario así como el modo RTL para que aterrice. Si se ve que estas dos acciones se pueden implementar sin que el vehículo sufra daños se pueden usar en otras rutinas de vuelo, lo cual incrementaría la autonomía.

- Buscar disminuir el retardo en la comunicación ssh reduciendo la potencia de salida de las puertas de enlace y aumentando el link rate, siempre y cuando esto no genere una pérdida de la señal.
- Hacer un vuelo con la aplicación autónoma del apéndice 10.4 para validar que la inclusión de las variables globales gui2other y valid_disc (después del vuelo de prueba que se hizo) permiten recuperar el progreso perdido cuando se desconecta el modo guiado (en simulación funciona).
- Aprovechar la estructura al máximo dándole múltiples usos

Capítulo 9. Referencias

[1] «Grupo Integrado de Ingeniería,» 2014. [En línea]. Available: http://www.gii.udc.es/. [Último acceso: 4 marzo 2018].

[2] Aerial Insights, «¿Cuántos tipos de drones existen en el mercado?,» 2017. [En línea]. Available: http://www.aerial-insights.co/blog/tipos-de-drones/.

[3] N. Carrodeguas, «NorfiPC compartiendo en Internet,» [En línea]. Available: https://norfipc.com/utiles/tutorialbatch1.html. [Último acceso: 1 noviembre 2018].

[4] Ardupilot, «Ardupilot,» 2016. [En línea]. Available: http://ardupilot.org/about. [Último acceso: 30 Septiembre 2018].

[5] Ardupilot Dev Team, «Ardupilot,» 2019. [En línea]. Available: http://ardupilot.org/copter/docs/flight-modes.html. [Último acceso: 15 Enero 2019].

[6] E. Santana, «XDrones,» 2018. [En línea]. Available: http://www.xdrones.es/mavlink/. [Último acceso: 2 noviembre 2018].

[7] ArduPilot Dev Team, «The Cube Overview,» 2016. [En línea]. Available: http://ardupilot.org/copter/docs/common-thecube-overview.html.

[8] Proficnc, «Pixhawk v2 Feature Overview,» 25 julio 2016. [En línea]. Available: http://www.hex.aero/wp-content/uploads/2016/07/DRS_Pixhawk-2-17th-march-2016.pdf.

[9] R. Maureira, «Unidad de medicón Inercial (IMU),» 24 diciembre 2010. [En línea]. Available: http://smartdreams.cl/unidad-de-medicion-inercial-imu/.

[10] P. Ruesca, «TELEMETRÍA – APLICACIONES DE MEDIDA A DISTANCIA,» 25 Septiembre 2016. [En línea]. Available: http://www.radiocomunicaciones.net/radio/telemetria/.

[11] Dronecode Project , «Simulation,» 2018. [En línea]. Available: https://dev.px4.io/en/simulation/. [Último acceso: 5 Diciembre 2018].

[12] L. Gorgona, «Teoría de Redes de Computadoras,» 22 agosto 2009. [En línea]. Available:

https://www.oas.org/juridico/spanish/cyber/cyb29_computer_int_sp.pdf. [Último acceso: 2 noviembre 2018].

[13] Xataka, «Cuáles son las diferencias entre Hub, Switch y Router,» 5 setiembre 2018. [En línea]. Available: https://www.xataka.com/basics/cuales-son-las-diferencias-entre-hub-switch-y-router. [Último acceso: 2 noviembre 2018].

[14] Ubiquitour.com, «Diferencia entre gateway y router,» 6 junio 2018. [En línea]. Available: http://www.ubiquitour.com/aOg5XMQW/. [Último acceso: 2 noviembre 2018].

[15] M. Mega, «Las clases de direcciones IPv4, A, B, C, D y E,» 23 octubre 2016. [En línea]. Available: http://thehittoslab.blogspot.com/2016/10/las-clases-de-direcciones-ipv4-b-c-d-y-e.html. [Último acceso: 2 noviembre 2018].

[16] C. González, 10 setiembre 2015. [En línea]. Available: https://www.adslzone.net/2015/09/10/por-que-las-direcciones-de-tu-router-son-192-168-x-x/. [Último acceso: 2 noviembre 2018].

[17] J. Crespo, 9 julio 2017. [En línea]. Available: https://www.redeszone.net/2017/07/09/mascara-subred-descripcion-utilidad/. [Último acceso: 2 noviembre 2018].

[18] Microhard Systems INC., de *Operating Manual, Nano IP series*, Calgary, 2011, pp. 14-15,26-29,32-34,50-52,.

[19] Microsoft, «Microsoft,» 8 noviembre 2015. [En línea]. Available: https://answers.microsoft.com/es-es/windows/forum/windows_10-start/iniciar-un-programa-al-encenderse-windows-10/1704b332-b34d-472d-838f-a621ce4a58b7. [Último acceso: 1 octubre 2018].

[20] Geekland, «Blog de Tecnología,» 3 junio 2018. [En línea]. Available: https://geekland.eu/instalar-cliente-servidor-ssh-en-windows/. [Último acceso: 2 noviembre 2018].

[21] M. Fernández, S. Ferrandiz y A. Conejero, «Effect of Infill Parameters on Tensile Mechanical,» *3D PRINTING AND ADDITIVE MANUFACTURING,* vol. 3, nº 3, p. 187, 2016.

[22] B. Tymrak, M. Kreiger y J. Pearce, «Mechanical properties of components fabricated with open-source 3-D printers under realistic environmental conditions,» *Materials and Design,* vol. 58, pp. 242-246, 2014.

[23] M. G. G. S. M. M. Antonio Lanzotti, «The impact of process parameters on mechanical properties of parts fabricated in PLA with an open-source,» *Rapid Prototyping Journal,* vol. 21, nº 5, pp. 604-617, 2015.

[24] Autodesk, «Inventor (2016) [Software],» [En línea]. Available: https://www.autodesk.com/education/free-software/inventor-professional.

[25] F. Beer, E. Russel, J. DeWolf y D. Mazuerek, Mecánica de Materiales, Ciudad de México: McGraw Hill, 2009.

[26] R. Budynas y J. Nisbett, Diseño en Ingeniería Mecánica de Shigley, Ciudad de México: McGraw-Hill, 2008.

[27] A. Pérez, «Mecapedia,» 12 Abril 2016. [En línea]. Available: http://www.mecapedia.uji.es/union_roscada.htm. [Último acceso: 25 Febrero 2019].

[28] S. Rohringer, «PLA vs ABS: comparación de filamentos para impresión 3D,» 19 Junio 2018. [En línea]. Available: https://all3dp.com/es/filamento-abs-filamento-pla-comparacion-impresion-3d/. [Último acceso: 26 Febrero 2019].

[29] Google, «Google Earth Pro (7.0 beta) [Software],» 2019. [En línea]. Available: https://www.google.es/earth/download/gep/agree.html.

[30] M. Oborne, «Mission Planner (1.3.59) [Software],» 2018. [En línea]. Available: http://ardupilot.org/planner/docs/mission-planner-installation.html.

[31] J. Mora, «Diseño de sistemas mecatrónicos,» 2018.

[32] O. Liang, «Types of multirotor,» 3 noviembre 2016. [En línea]. Available: https://oscarliang.com/types-of-multicopter/.

Capítulo 10. Apéndice

10.1 Elementos de la red y procedimiento para su configuración

Los dispositivos empleados para hacer la red tienen su propio manual en [18], este apéndice es prácticamente una traducción al español de las instrucciones importantes para este proyecto.

Estas puertas de enlace tienen 3 modos de operación posibles: maestro, repetidor y remoto. El primero le permite al dispositivo controlar el flujo de información y sólo puede haber uno por red, mientras que los remotos son simplemente los *"endpoints"* o puntos finales de la red que se comunican con el máster. Por su parte, los repetidores no son necesarios en este proyecto, pero su uso puede consultarse en el capítulo 4 de [18].

De la misma manera, en el capítulo 5 de [18] se habla de todas las posibles topologías de red, las cuales permiten diferentes características de envío de información entre estos dispositivos, sin embargo, en este proyecto sólo interesa la denominada "punto a punto", cuyo procedimiento de configuración es el que se expondrá más adelante.

Las partes importantes de los dispositivos empleados se muestran en la Figura 10.1.1 y Figura 10.1.2:

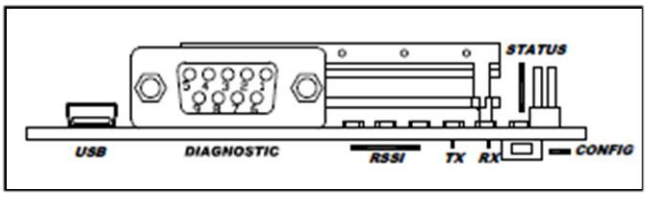


Figura 10.1.1 Perfil frontal de la puerta de enlace. Fuente: [18]

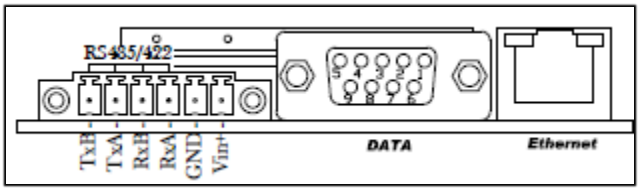


Figura 10.1.2 Perfil Trasero de la puerta de enlace. Fuente: [18]

- La **alimentación** se conecta entre los pines Vin+ y GND del perfil trasero y el rango de tensión de entrada está entre 7 V y 30 V.
- También en la parte trasera se encuentra el **puerto Ethernet** necesario para conectar el dispositivo a una computadora.
- El **RSSI** (*Receive Signal Strength Indicator*) de la cara frontal consiste en 3 leds verdes que indican qué tan fuerte es la señal. Una señal débil se puede notar porque sólo el led de la izquierda está encendido y conforme se incrementa, se encienden los demás leds de izquierda a derecha.
- Los leds **TX** y **RX** (verdes) al encenderse indican que el dispositivo está transmitiendo y recibiendo datos, respectivamente.
- El ***Status Led*** se enciende por aproximadamente 20 s una vez que se conecta la alimentación. Seguidamente, parpadea por otros 25 s indicando que está cargando, para luego volver a ser una luz sólida, lo que significa que está listo para transmitir datos.
- Si ya pasó poco más de 1 minuto dese que se encendió la puerta y se mantiene presionado el ***Config button*** por 8 s, el dispositivo volverá a su configuración de fábrica que implica un modo de operación de remoto con una dirección IP estática de 192.168.1.254, máscara de subred 255.255.255.0 y puerta de enlace 192.168.1.1, útil si no se conocen las configuraciones del equipo.

La configuración de red Punto a Punto se detalla a continuación:

1. Alimente los dispositivos
2. La puerta que tiene la antena más grande será configurada como maestro, para empezar conéctelo con un cable Ethernet a la computadora de control en tierra (CCT). Al puerto Ethernet de la PC se le debe asignar una dirección IP estática de 192.168.1.10, con máscara de subred de 255.255.255.0.
3. Abra un navegador, en la línea de direcciones escriba la dirección IP de la puerta que está conectada y presione la tecla enter.
4. Aparecerá una ventana solicitando un usuario y contraseña (Figura 10.1.3), ambos son: admin.

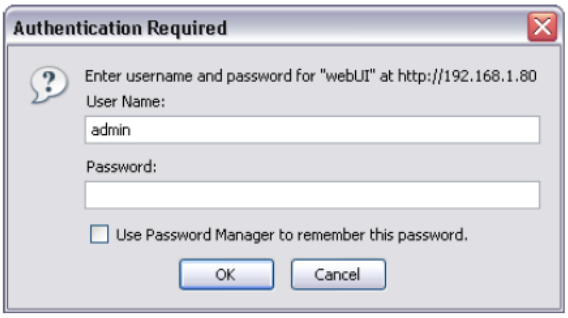


Figura 10.1.3 Ventana de acceso a la puerta de enlace. Fuente: [18]

5. Al entrar aparecerá la siguiente ventana (Figura 10.1.4). Acceda a *Network Configuration*, donde tendrá varias opciones, elija *Local IP Config*, donde se encontrará una ventana como la de la Figura 10.1.5, haga click en “static” y escriba los valores de los parámetros de IP que desee.

Figura 10.1.4 Ventana de Inicio de la puerta de enlace. Fuente: [18]

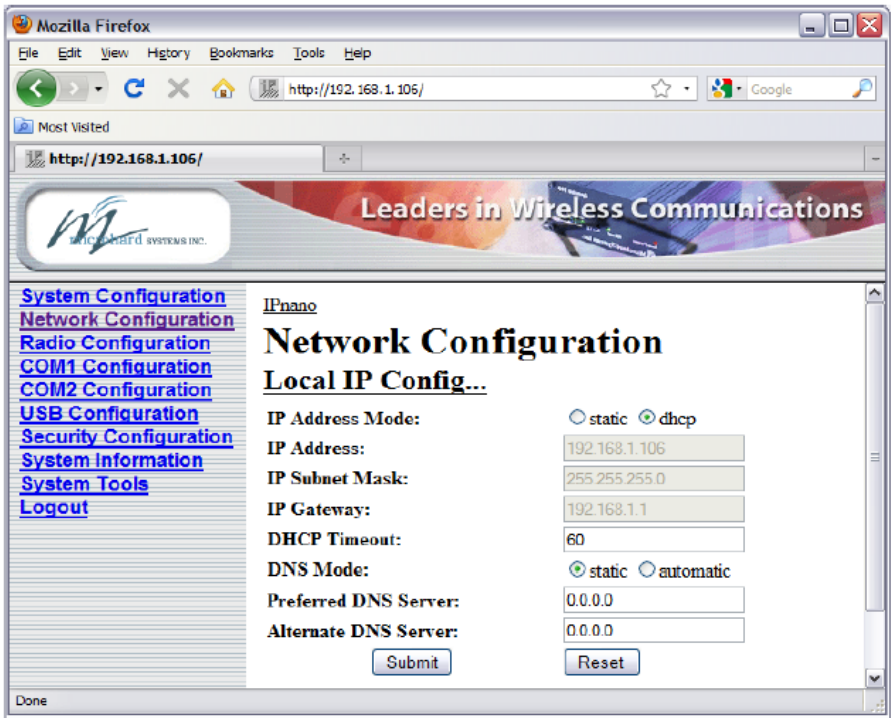


Figura 10.1.5 Configuración de red de la puerta de enlace. Fuente: [18]

6. Ahora acceda a “Radio Configuration”, en la Figura 10.1.6 se muestra la configuración tanto para el maestro como para el remoto (la opción se elige en “Operation Mode”). En “Network type” elija “Point to Point”.

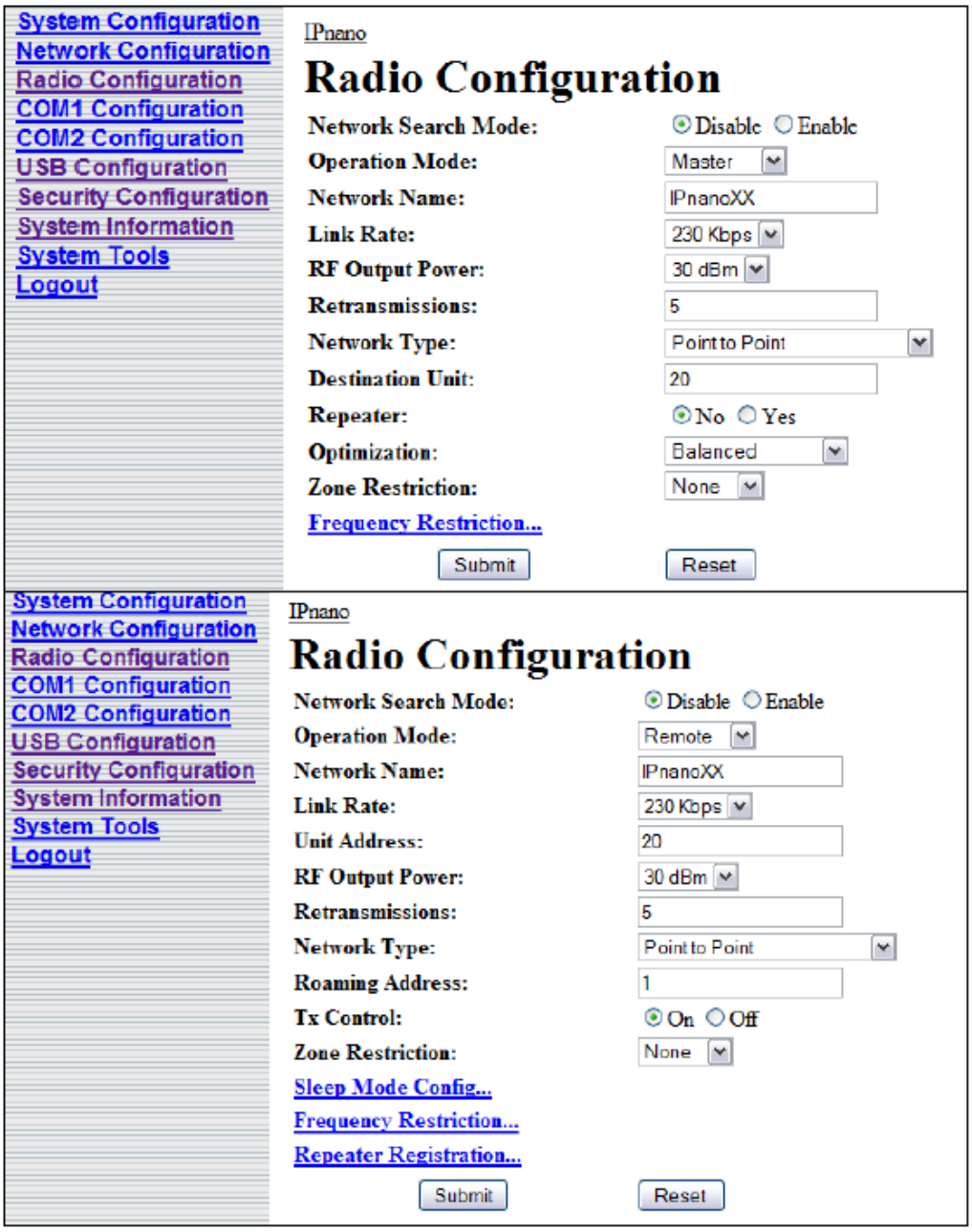


Figura 10.1.6 Radio configuración de las puertas de enlace. Fuente: [18]

A cada dispositivo dentro de la red se le asigna un número (“Unit address"), que en el caso del maestro está fijado por defecto como 1. En el ejemplo se observa que el remoto tiene un *unit address* de 20. Estos números también deben colocarse en la configuración del otro dispositivo para que sepa con “quién” comunicarse, en el caso del maestro se coloca en “Destination Unit” y en el remoto en “Roaming Address”.

Las demás opciones pueden consultarse en la sección 6.1.5 de [18], en este proyecto se dejaron tal cual está en la Figura 10.1.6.

Termine de configurar el maestro y repita los pasos anteriores con el remoto. Cuando esto pase, los RSSI leds deberían estar iluminados siempre que estén cerca los dispositivos.

10.2 Conexión de la estructura autónoma

A continuación se detalla el procedimiento a seguir para conectar la estructura de hardware y software para la ejecución de rutinas autónomas. Lo primero que se lista es una serie de verificaciones en la mini PC previo a llevar el UAV al campo de vuelo, dado que requieren de un monitor, ratón y teclado, recursos de los que no siempre se dispone en dicho lugar. Posterior a esto se detallan los pasos a seguir para conectar todos los recursos de la estructura desarrollada en este proyecto.

a) Previo a la conexión de la estructura autónoma verifique lo siguiente en la mini PC:

1. La dirección IP del puerto Ethernet es 192.168.1.154 y en la CCT es 192.168.1.155, con máscara de subred 255.255.255.0
2. En la carpeta “Inicio” que está en el escritorio se encuentra el archivo configpuertosuav.bat
3. La suspensión automática se encuentra desactivada (se revisa en Configuración>Sistema>Inicio/apagado y suspensión).
4. Se puede acceder al terminal de la mini PC desde la CCT vía ssh. En caso de que no, [20] brinda todos los pasos para hacerlo.

b) Conexión de la estructura autónoma

1. Sujetar las piezas superiores del soporte (sostenedores) al UAV con pernos.
2. Conectar el extremo de un cable Ethernet a la puerta de enlace aérea y sujetarla a su base correspondiente con 3 pernos.

3. Colocar la NUC en el espacio correspondiente del soporte, sujetarla con los tornillos correspondientes y conectarla con el cable Ethernet a la puerta de enlace aérea.
4. Conectar la puerta de enlace de tierra a la computadora de control en tierra (CCT) con un cable Ethernet.
5. Sujetar los sostenedores con las bases que contienen la mini PC y puerta de enlace, usando 4 pernos.
6. Conectar el Pixhawk desde su puerto mini USB a uno USB de la NUC, con un cable USB-mini USB. La NUC debería designar dicho puerto USB como COM4.
7. Alimentar las puertas de enlace y esperar a que ambos tengan las 3 luces de intensidad de señal encendidas.
8. Conectar la alimentación de la mini PC, encenderla y esperar poco menos de un minuto, tiempo suficiente para que inicie el sistema operativo de la NUC y empiece a transmitir telemetría a su puerto interno y a la CCT.
9. En la CCT, abrir el software de vuelo a usar (normalmente mission planner) y conectarse al puerto interno UDP 14552. Para comprobar que la conexión funciona bien, incline varias veces el UAV y confirme que en dicho software se vean dichas inclinaciones.
10. Desde la misma CCT abrir un PowerShell y escribir: ssh -p 22 usuario@192.168.1.154.
11. Después de esto, se solicitará una contraseña: usuario. Se presiona enter y se accede al terminal de la mini PC. Desde este se puede abrir los programas que se deseen ejecutar.

Nota: Recuerde que la telemetría en la NUC se transmite a su puerto UDP 127.0.0.1:14551. Además, es normal que al digitar la contraseña para la comunicación SSH, el programa no despliegue los caracteres que se han escrito, ni siquiera los típicos asteriscos (*), pero nada más debe digitarse completa y presionar enter.

10.3 Bibliotecas de software para programación de rutas

10.3.1 Global variables

```
global gui2other#True when the guided mode has just been disconnected (The onboard PC loses control of the UAV)
gui2other=False
global valid_disc#True when the UAV is supossed to be controled by a remote control, so it's a valid disconnection
valid_disc=True
global BatteryVolt#the variable is made to contain the lowest acceptable value of battery voltage.
BatteryVolt=0#It has to be assigned in the main program.
global BatteryLevel#the variable is made to contain the lowest acceptable value of battery level.
BatteryLevel=0#It has to be assigned in the main program.
```

10.3.2 Estructura uav

```
from dronekit import connect, Command, VehicleMode, LocationGlobalRelative, LocationLocal, LocationGlobal
from pymavlink import mavutil
import time, math
import globalvariables
# This library contains some basic functions to program a mission in a uav (copter), controlled by an ardupilot autopilot.
#It can be expanded and modified at your convenience. It was mainly done to execute a mission from a mini PC on board,
#but certainly other uses can be given, such as simulation or even give instructions to a uav from a computer in ground.

def printdelay (t):
        #This function makes a delay with an integer quantity of seconds (t), while prints the remaining time. If "t" is not an
        #integer, the function takes its integer part.
        t=int(t)
        print "It was defined a ", t, " seconds delay"
        for x in range(0,t+1):
                print t-x, " seconds remaining"
                time.sleep(1)
        print "\n"
```

```
def arm(uav,time_review,t):
        #The function waits for the vehicle to be armable, indicating every 5 seconds why it is not armable yet. Once
        #armable, the uav is ordered to arm. Then, the function periodically checks if the uav is already armed and
        #resend the "arm" instruction each time it checks it is not armed.
        #You have the option to leave a final delay, given by the user, before leaving the function to execute other
        #instructions.
        #Parameters:
        #uav= name given to the uav
        #time_review= the period between every check (seconds)
        #t=final delay (entire quantity of seconds)
        while not uav.is_armable:#Exploring and printing why the drone is not armable yet
                print "Not armable yet"
                print " Waiting for vehicle to initialise..."
                if uav.gps_0.fix_type < 2:
                        print "Waiting for GPS", uav.gps_0.fix_type
                if uav.ekf_ok:
                        print "EKF Ok"
                if not uav.ekf_ok:
                        print "EKF problem"
                print uav.system_status
                print "\n"
                time.sleep(5)
        print"Arming\n"
        con=0
        while uav.is_armable and not uav.armed:#This part was thought for the user to know how much it takes to arm the
        #vehicle
                con=con+1#by decreasing "time_review" to the minimun value that keep the counter "con" with a value of 1
                uav.armed=True
                time.sleep(time_review)
        if uav.armed:
                print "Armed"
                if con==1:
```

```
                print"The 'arm' order was sent only once\n"
            else:
                print "The 'arm' order was sent ", con, " times\n"
        printdelay(t)

def disarm(uav,time_review,t):
        #The uav is ordered to disarm. Then, the function periodically checks if the uav is already disarmed and resend
        #the "disarm" instruction each time it checks it is still armed.
        #The vehicle must be in landed, otherwise, it won't be disarmed.
        #You have the option to leave a final delay, given by the user, before leaving the function to execute other
        #instructions.
        #Parameters:
        #uav= name given to the uav
        #time_review= the period between every check (seconds)
        #t=final delay (entire quantity of seconds)
        print"Disarming\n"
        con=0
        while  uav.armed:#This part was thought for the user to know how much it takes to disarm the vehicle
            con=con+1#by decreasing "time_review" to the minimun value that keep the counter "con" with a value of 1
            uav.armed=False
            time.sleep(1)
        if not uav.armed:
            print "Disarmed"
            if con==1:
                print"The 'disarm' order was sent only once\n"
            else:
                print "The 'disarm' order was sent ", con, " times\n"
        printdelay(t)

def elev_eval(uav,altitude, percent, sampletime,t):
        """
        The uav is raised to a desired altitude and when it reaches a target percentage, defined by the user, the
```

```
        function prints that the target was achieved. Normally it is used only in SITL simulation, because in the
        real life it is preferable to elevate a UAV through a remoto control and not by software.
        While it is raising it will be printing the height it has every "sampletime" seconds.
        You have the option to leave a final delay, given by the user, before leaving the function to execute other
        instructions
         Parameters:
        uav= name given to the uav
        altitude measured since the take off point
        percent=target percentage of the altitude (between 0 and 1)
        sampletime= the period between every altitude notification (seconds)
        t=final delay (entire quantity of seconds)"""
        #global globalvariables.gui2other
        bat_info(uav,)
        guidedmode(uav)
        time.sleep(5)
        arm(uav,1,0)
        uav.simple_takeoff(altitude) # Take off to the desired altitude
        print "Taking off"
        while True:
                print "Altitude: ", uav.location.global_relative_frame.alt, "target: ", altitude

                if uav.location.global_relative_frame.alt>=altitude*percent:
                        print "Reached target altitude \n"
                        printdelay(t)
                        break
                if globalvariables.gui2other:
                        break
                bat_info(uav)
                time.sleep(sampletime)

def manual_elev(uav, altitude):#When an user elevates a uav through a remote control this function monitor the height
        globalvariables.valid_disc=True# it is imposible to use guided mode
```

```
        print "Elevate the UAV"
        while True:
                print "Altitude: ", uav.location.global_relative_frame.alt, "target: ", altitude
                if uav.location.global_relative_frame.alt>=altitude*0.95:
                        print "You already reached the target position. Turn to guided mode"
                if uav.mode=="GUIDED":#once guided mode the loop breaks and valid_disc is false to execute a series of
                        #PC instructions
                        break
                time.sleep(5)
                bat_info(uav)
        globalvariables.valid_disc=False


def send_ned_velocity(uav,velocity_x, velocity_y, velocity_z, duration,re_sent_period):
    """
    Move vehicle in direction based on specified velocity vectors during "duration" seconds, resending the order
    every "re_sent_period" (it's recommended a value between 1 and 3).
    Components x, y and z are positive in the north, east and downward directions, respectively. In the other hand,
    x,y and z are negative in the south, west and upward directions, respectively.

    """
    bat_info(uav)
    msg = uav.message_factory.set_position_target_local_ned_encode(
        0,       # time_boot_ms (not used)
        0, 0,    # target system, target component
        mavutil.mavlink.MAV_FRAME_LOCAL_NED, # frame
        0b0000111111000111, # type_mask (only speeds enabled)
        0, 0, 0, # x, y, z positions (not used)
        velocity_x, velocity_y, velocity_z, # x, y, z velocity in m/s
        0, 0, 0, # x, y, z acceleration (not supported yet, ignored in GCS_Mavlink)
        0, 0)    # yaw, yaw_rate (not supported yet, ignored in GCS_Mavlink)
```

```
    # send command to vehicle every "re_sent_rate" seconds
    coc=duration//re_sent_period
    res=duration%re_sent_period

    for x in range(0,coc+1):
        if globalvariables.gui2other:#Verifies if the mode is not guided and break if there was a disconnection
                print "send_ned_velocity. Turn to guided mode to continue with the route"
                break
        if x==coc:
                re_sent_period=res
        uav.send_mavlink(msg)

        time.sleep(re_sent_period)


def get_location_metresxy(original_location, dNorth, dEast):
        """
        It transforms a position vector in a horizontal plane with dNorth and dEast components into gps positions. This
        vector is relative to a gps location.
        Parameters:
        original_location=reference gps location, obtained with LocationGlobal or LocationGlobalRelative from dronekit.
        dNorth= distance to the North (positive value) or the South (negative value) in metres.
        dEast= distance to the East (positive value) or the West (negative value) in metres.
        """
        earth_radius=6378137.0 #Radius of "spherical" earth
        #Coordinate offsets in radians
        dLat = dNorth/earth_radius
        dLon = dEast/(earth_radius*math.cos(math.pi*original_location.lat/180))

        #New position in decimal degrees
        newlat = original_location.lat + (dLat * 180/math.pi)
```

```
        newlon = original_location.lon + (dLon * 180/math.pi)
        if type(original_location) is LocationGlobal:
                targetlocation=LocationGlobal(newlat, newlon,original_location.alt)
        elif type(original_location) is LocationGlobalRelative:
                targetlocation=LocationGlobalRelative(newlat, newlon,original_location.alt)
        else:
                raise Exception("Invalid Location object passed\n")

        return targetlocation;

def eval_distxy(uav, aLocation, dist_obj, sampletime, t):
        """
        This function was made to be used after executing the simple_goto function and it monitors the GROUND distance
        from the uav to the desired position and when it reaches a target distance prints a notification. It uses an
        "spherical earth" aproximation.
        You have the option to leave a final delay, given by the user, before leaving the function to execute other
        instructions
        Parameters:
        uav= name given to the uav
        aLocation= target gps position (the same used in simple_goto)
        dist_obj= acceptable distance from the uav to aLocation to consider the target was reached
        sampletime= the period between every distance notification (seconds)
        t=final delay (entire quantity of seconds)
        """

        dist=dist_obj+1
        if type(aLocation) is LocationGlobal:
                while dist >= dist_obj:
                        if globalvariables.gui2other:#if the mode is not guided anymore, the function stop been executed
                                "eval_distxy. Turn to guided mode to continue with the route"
                                break
                        dlat = aLocation.lat - uav.location.global_frame.lat# dlat y long in decimal degrees, not radians
```

```
                dlong = aLocation.lon - uav.location.global_frame.lon
                earth_radius=6378137.0
                dNorth=earth_radius*(dlat*math.pi/180)
                dEast=(dlong*math.pi/180)*(earth_radius*math.cos(math.pi*uav.location.global_frame.lat/180))
                dist=math.sqrt(dNorth**2+dEast**2)
                #dist=math.sqrt((dlat*dlat) + (dlong*dlong)) * 1.113195e5 #just another aproximation
                print "I am in : ", uav.location.global_frame, ", I am going to: ", aLocation, ". ", dist, " meters
remaining\n"

                if dist<=dist_obj and not globalvariables.gui2other:#allows to skip the sampletime when the target has
                        #been reached
                        print "I reached the target position"
                        printdelay(t)
                        break
                time.sleep(sampletime)
                bat_info(uav)
        elif type(aLocation) is LocationGlobalRelative:
                while dist >= dist_obj:
                        if globalvariables.gui2other:#if the mode is not guided anymore, the function stop been executed
                                "eval_distxy. Turn to guided mode to continue with the route"
                                break
                        dlat = aLocation.lat - uav.location.global_relative_frame.lat# dlat y long in decimal degrees, not
                        #radians
                        dlong = aLocation.lon - uav.location.global_relative_frame.lon
                        earth_radius=6378137.0
                        dNorth=earth_radius*(dlat*math.pi/180)

        dEast=(dlong*math.pi/180)*(earth_radius*math.cos(math.pi*uav.location.global_relative_frame.lat/180))
                        dist=math.sqrt(dNorth**2+dEast**2)
                        #dist=math.sqrt((dlat*dlat) + (dlong*dlong)) * 1.113195e5 #just another aproximation
                        print "I am in : ", uav.location.global_relative_frame, ", I am going to: ", aLocation, ". ", dist, " meters
remaining\n"
```

```
                    if dist<=dist_obj and not globalvariables.gui2other:#allows to skip the sampletime when the target has
                    # been reached
                        print "I reached the target position"
                        printdelay(t)
                        break
                    time.sleep(sampletime)
                    bat_info(uav)

def get_location_metresxyz(original_location, dNorth, dEast,dUp):
    """
    It transforms a position vector with dNorth, dEast and dUp components into gps positions. This vector is
    relative to a gps location. It uses an "spherical earth" aproximation.
    Parameters:
    original_location=reference gps location, obtained with LocationGlobal or LocationGlobalRelative from dronekit.
    dNorth= distance to the North (positive value) or the South (negative value) in meters.
    dEast= distance to the East (positive value) or the West (negative value) in meters.
    dUp=vertical distance (downward is negative , upward is positive) in meters.
    """
    earth_radius=6378137.0 #Radius of "spherical" earth
    #Coordinate offsets in radians
    dLat = dNorth/earth_radius
    dLon = dEast/(earth_radius*math.cos(math.pi*original_location.lat/180))

    #New position in decimal degrees
    newlat = original_location.lat + (dLat * 180/math.pi)
    newlon = original_location.lon + (dLon * 180/math.pi)
    newalt = original_location.alt + dUp
    if type(original_location) is LocationGlobal:
        targetlocation=LocationGlobal(newlat, newlon,newalt)
    elif type(original_location) is LocationGlobalRelative:
        targetlocation=LocationGlobalRelative(newlat, newlon,newalt)
    else:
```

```
                raise Exception("Invalid Location object passed\n")

        return targetlocation;
def eval_distxyz(uav, aLocation, dist_obj, sampletime, t):
        """
        This function was made to be used after executing the simple_goto function and it monitors the distance from
        the uav to the desired position and when it reaches a target distance prints a notification.
        It uses an "spherical earth" aproximation.
        You have the option to leave a final delay, given by the user, before leaving the function to execute other
        instructions
        Parameters:
        uav= name given to the uav
        aLocation= target gps position (the same used in simple_goto)
        dist_obj= acceptable distance from the uav to aLocation to consider the target was reached
        sampletime= the period between every distance notification (seconds)
        t=final delay (entire quantity of seconds)
        """
        #global globalvariables.gui2other
        #globalvariables.gui2other=globalvariables.gui2other
        dist=dist_obj+1
        if type(aLocation) is LocationGlobal:
                while dist > dist_obj:
                        if globalvariables.gui2other:#if the mode is not guided anymore, the function stop been executed
                                "eval_distxyz. Turn to guided mode to continue with the route"
                                break
                        dlat = aLocation.lat - uav.location.global_frame.lat# dlat y long in decimal degrees, not radians
                        dlong = aLocation.lon - uav.location.global_frame.lon
                        earth_radius=6378137.0
                        dNorth=earth_radius*(dlat*math.pi/180)
                        dEast=(dlong*math.pi/180)*(earth_radius*math.cos(math.pi*uav.location.global_frame.lat/180))
                        dAlt=aLocation.alt - uav.location.global_frame.alt
                        dist=math.sqrt(dNorth**2+dEast**2+dAlt**2)
```

```
                print "I am in : ", uav.location.global_frame, ", I am going to: ", aLocation, ". ", dist, " meters
remaining\n"

                if dist<=dist_obj and not globalvariables.gui2other:#allows to skip the sampletime when the target has
                    #been reached
                    print "I reached the target position"
                    printdelay(t)
                    break
                time.sleep(sampletime)
                bat_info(uav)
        elif type(aLocation) is LocationGlobalRelative:
            while dist > dist_obj:
                if globalvariables.gui2other:#if the mode is not guided anymore, the function stop been executed
                    "eval_distxyz. Turn to guided mode to continue with the route"
                    break
                dlat = aLocation.lat - uav.location.global_relative_frame.lat# dlat y long in decimal degrees, not
                #radians
                dlong = aLocation.lon - uav.location.global_relative_frame.lon
                earth_radius=6378137.0
                dNorth=earth_radius*(dlat*math.pi/180)

        dEast=(dlong*math.pi/180)*(earth_radius*math.cos(math.pi*uav.location.global_relative_frame.lat/180))
                dAlt=aLocation.alt - uav.location.global_relative_frame.alt
                dist=math.sqrt(dNorth**2+dEast**2+dAlt**2)
                print "I am in : ", uav.location.global_relative_frame, ", I am going to: ", aLocation, ". ", dist, " meters
remaining\n"
                if dist<=dist_obj and not globalvariables.gui2other:#allows to skip the sampletime when the target has
                #been reached
                    print "I reached the target position"
                    printdelay(t)
                    break
                time.sleep(sampletime)
```

```
                    bat_info(uav)

def guidedmode(uav):#Used to set the guided mode in the uav. The only parameter is the name of the uav.
        print uav.mode
        con=0
        if uav.mode !="GUIDED":#Turn to guided mode, which is neccesary to follow instructions from a PC
                while  uav.mode!="GUIDED":#Review the mode is guided, otherwise, gives the order again
                        con=con+1
                        uav.mode = VehicleMode("GUIDED")
                        print "changing to guided mode"
                        time.sleep(3)
                print uav.mode
        print "The change mode order was sent ",con, "times"

def bat_info(uav):#If the battery has a dangerous value, advices it, for the user to take control of the uav and
# make it to go to home position
        if globalvariables.BatteryVolt!=None and uav.battery.voltage!=None and
globalvariables.BatteryVolt>=uav.battery.voltage:
                print "Caution: the BATTERY VOLTAGE reached a DANGEROUS VALUE:",uav.battery.voltage,". Go back
to home position as soon as posible"
        if globalvariables.BatteryLevel!=None and uav.battery.level!=None and
globalvariables.BatteryLevel>=uav.battery.level:
                print "Caution: the BATTERY LEVEL reached a DANGEROUS VALUE:",uav.battery.level,". Go back to
home position as soon as posible\n"
```

10.4 Aplicación autónoma

```
from dronekit import connect, Command, VehicleMode, LocationGlobalRelative, LocationLocal, LocationGlobal
from estructurauav import printdelay,guidedmode, arm,disarm,manual_elev,elev_eval,send_ned_velocity,
get_location_metresxy, eval_distxy, get_location_metresxyz, eval_distxyz
#Always import the functions from estructurauav that way and not all the library, the sintax through the program
#becomes easier
import globalvariables# import it in every program and in estructura uav
from pymavlink import mavutil
import time, sys, argparse, math
import itertools
import os
import random

"""*********MAIN DESCRIPTION OF THE PROGRAM********

Here there is IA programn (genetic algorithm) to plan routes for a multicopter UAV with a new software-hardware
structure for autonomous operation. This program should be in a mini PC onboard with comunication with a ground
control station from which a user can access the mini PC's terminal to execute any program.

In the main code of the program there is a list of vectors called positions, each vector has an "east" component
and a "north" component, respectively, both in meters, which are refered to the home position; positive values
are for east and north, while negative values are for west and south. The program emulates a package deliverer UAV,
which can carry only 4 packages at the same time, so through a genetic algorithm chooses 4 points for the UAV to
"deliver a package" in, trying to do the shortest posible route and returning back to home position to continue
delivering packages.
Each iteration mutates some routes and reduces the quantity of them, in order to get a good one. The user controls
whether the program makes another iteration or chooses and executes one of the calculated routes. When a route is
executed the UAV takes off, goes to each of the four points, goes some meter down and ask the user if is ready to
continue the route (like asking whether already "delivered the respective package").
It is posible to add more points or disable and enable manually any of them.
```

```
The  information of each gps cordinate point is in a "Point" object"""
class Point:
        def __init__(self, num):
                self.number=num #identificator of the point
                self.gps="" #gps position, obtained with LocationGlobal or LocationGlobalRelative
                self.state=1 #1 if it a UAV must go there, 0 if not, 2 if a UAV is fliying to the point
                self.dist=[]#this list will contain all the distances to each point, where each index corresponds each
                #point's number
        def askpos(self):#point number 0 will be the home location
                lat=float(input("latitude: "))
                lon=float(input("longitude: "))
                alt=float(input("altitude: "))
                self.gps=LocationGlobal(lat,lon,588)
                print "\n"
        def setpos(self, pos):
                self.gps=pos
        def setsta(self,sta):
                self.state=sta
        def add_dist(self,distance):
                self.dist.append(distance)

def valid_input(l):#This function ask the user for an input and determine if it is in the parameter "l", which has
        #to be a list
        #with all the posible vales the input can have. Once found a valid input, the program returns it
    m=raw_input()
    enter=True
    if m.isdigit():#if the input is a number, the function will take only the integer part
        m=int(m)
    while not m in l:
        print"type a valid value"
        m=raw_input()
        if m.isdigit():
```

```
        m=int(m)
    os.system('cls')
    return m

def managepoints(uav,altitude): #The main function, it manages all the things the program can do
        global points_list#Global variable only in this program, not in the libraries. Contains all the "Point" objects
        ent=True
        while ent:#------------------Main Menu
                os.system('cls')
                print"Points Management Menu"
                print"1.Add points"#for more destinations
                print"2.Show points"#Shows all the points, gps cordinates and distances between each one
                print"3.Change state of points"#For enable or disable manually any point
                print"4.Calculate route"#start the IA route algorithm
                print"5.Leave the menu\n"#Finish the program
                print "Write the corresponding number:"
                menu=valid_input([1,2,3,4,5])

                if menu==1:
                        add=True
                        while add:
                                addpoint()
                                print "Do you want to add more points? (1 for yes or 0 for no)"
                                m=valid_input([0,1])
                                if m==0:
                                        add=False

                elif menu==2:
                        for x in points_list:
                                print "Point ",x.number, ": ", x.gps,". State: ",x.state,"\n"
                        print "Distances matrix"
                        for x in points_list:
```

```
                        print x.dist
                print "\n"
                #time.sleep(15)
                raw_input("Press any button to continue")


        elif menu==3:
                change=True
                while change:
                        print"Enter the number of the point whose state you want to change : \n"
                        index=valid_input(range(0,len(points_list)))
                        print "Point ",points_list[index].number, ": ", points_list[index].gps,". State:
",points_list[index].state,"\n"
                        print "Posible values: "
                        print "0. Inactive: if the point doesn't need to be visited any more"
                        print "1. Active: if a UAV has to take this point into account to plan its route"
                        print "2. In Process: if a UAV is already going to this point\n"
                        print"Enter the state number: "
                        state=valid_input([0,1,2])
                        points_list[index].setsta(state)
                        print"Do you want do change more points? (1 for yes or 0 for no)"
                        m=valid_input([0,1])
                        if m==0:
                                change=False
        elif menu==4:
                routing(Frankie,altitude,4,7,30)
        elif menu==5:
                ent=False
        else:
                print"Insert a valid number\n"

def addpoint():#add a new point, asking the user the gps cordinates, and add it to point_list
```

```
        global points_list
        num=len(points_list)
        p=Point(num)#create the object wit the respective number
        print "Point number ", num
        p.askpos()#ask gps cordinates
        points_list.append(p)
        num=len(points_list)
        for x in range(0,num):#calculates the distance to each point and put it in the distances list of each one
                if x==num-1:
                        points_list[x].add_dist(0)
                else:
                        d=distance(points_list[x].gps, points_list[num-1].gps)
                        points_list[x].add_dist(d)
                        points_list[num-1].add_dist(d)

def setpoint(position):#similar to add point, but don't ask the user the position, it's given as a parameter
        global points_list
        num=len(points_list)
        p=Point(num)
        p.setpos(position)
        points_list.append(p)
        num=len(points_list)
        for x in range(0,num):
                if x==num-1:
                        points_list[x].add_dist(0)
                else:
                        d=distance(points_list[x].gps, points_list[num-1].gps)
                        points_list[x].add_dist(d)
                        points_list[num-1].add_dist(d)

def distance (pointa,pointb):#calculate the distance between two points which have been obtained with the dronekit
        #location functions. It uses a spherical earth aproximation
```

```
        dlat = pointa.lat - pointb.lat# dlat y long in degrees, not radians
        dlong = pointa.lon - pointb.lon
        earth_radius=6378137.0
        dNorth=earth_radius*(dlat*math.pi/180)
        dEast=(dlong*math.pi/180)*(earth_radius*math.cos(math.pi*pointb.lat/180))
        #dAlt=pointa.alt - pointb.alt
        dist=math.sqrt(dNorth**2+dEast**2)# you should add +dAlt**2 if take into account altitude to calculate distance
        return round(dist,1)# the distance is rounded with one decimal

def routing(uav,altitude,ts,factor,td):
        #uav= name given to the uav
        #altitude measured since the take off point
        #ts: travel size, number of points to visit in one travel
        #factor: a multiplicative factor to increase the number of posible COMBINATIONS (not permutations) of points
        #to create the size of the initial population
        #td=target porcentual difference in lenght between
        global points_list
        globalvariables.valid_disc=True#It doesn't matter if an user changes the mode at the time of calculating routes
        home=LocationGlobalRelative(points_list[0].gps.lat,points_list[0].gps.lon, altitude)
        #the first element (index 0) is the home location

        act_points=[]#active points
        for x in points_list[1:]:
                if x.state==1:
                        act_points.append(x.number)
        if act_points==[]:
                max_dif=td
                print "All the points are inactive, there are not routes"
                raw_input("Press any button to continue")
        else:#If there is at least one active point
                n=len(act_points)#number of active points
                if n==1:
```

```
                print "There is only one active point, the number ",act_points[0]
                #manual_elev(uav,altitude)#used in the flight field in case the user does not want to take off the uav
                #by software
                elev_eval(uav, altitude, 0.96, 3,3)#take off the uav by software, mainly used for simulation
                globalvariables.valid_disc=False#once at the right altitude, it is not correct to disconnect the guided
                # mode

                while True:#--------------------VERY IMPORTANT------------------------------
                        """when there is a group of instructions to be completed one after other, where each one
depends on the last one, to ensure the program will do everything is suposed to do, even if there is a
disconnection during or between any of the instructions, the most efficient thing is to USE THIS KIND OF LOOP:
the estructurauav functions finish their execution when a mode change is detected (they review gui2other, which
is automatically turn to True if the mode stop being guided in the mode_callback function, which is just before
the main code) and as you can see, after the flight instructions there is a guided mode verification, which only
let the program to continue until there the mode is guided again (in case it was changed), and then turn gui2other
to false (it has to be manually, unlike when it becomes true).
When the task is done, there must be something that brake the loop: in this particular case, when the point's
state is 0"""
                        uav.simple_goto(points_list[act_points[0]].gps)#go to the point
                        eval_distxy(uav, points_list[act_points[0]].gps , 1, 7, 0)#evaluate how far is the uav from the
                        # point
                        send_ned_velocity(uav,0,0,0,5,1)# move down to "deliver the package"
                        descend(uav,act_points[0])#ask the user is the package is already delivered, and turn the state
                        # to 0
                        if points_list[act_points[0]].state==0:
                                break# no matter if there was a disconnection before deliver the package, the task is
                                # done
                        while uav.mode!="GUIDED":
                                print "Waiting for the guided mode"
                                time.sleep(5)
                        if uav.mode=="GUIDED":
                                globalvariables.gui2other=False
```

```
        while True:
                print "Going to home position"
                uav.simple_goto(home)
                eval_distxy(uav, home , 1, 7, 0)
                if globalvariables.gui2other==False: #If at this point there wasn't any mode change, it's
                        # completed
                        globalvariables.valid_disc=True#Time for the user to take control of the uav
                        break
                while uav.mode!="GUIDED":
                        print "Waiting for the guided mode"
                        time.sleep(5)
                if uav.mode=="GUIDED":
                        globalvariables.gui2other=False

        while uav.mode=="GUIDED":
                print"Take the control, change the mode"
                time.sleep(5)

        max_dif=td#max:dif will be used to contain the porcentual difference between shortest and longest
        # route
else:#more than one active point

        random.shuffle(act_points)#Shake the order of active point, just to give variability
        con=0#con is the number of iteration
        max_dif=td+1#for the loop to start
        if ts>len(act_points):
                ts=len(act_points)
        #Getting the real optimal route to compare with the results of the Genetic algorithm
        all_routes_generator=itertools.permutations(act_points,ts)#all the posible 4-point routes
        all_routes=[]
```

```
                all_lengths=[]
                for x in all_routes_generator:
                        x=list(x)
                        all_routes.append(x)
                        d=0
                        route=[0]+x+[0]#include home position at the beginning and the end to calculate the distance
                        for i in range(0,len(route)-1):
                                d=d+points_list[route[i]].dist[route[i+1]]#calculating distance
                        all_lengths.append(d)
                routes=zip(all_lengths,all_routes)
                routes.sort()#order the routes according their lenghts
                all_lengths,all_routes=zip(*routes)#These two lists contain all the posible routes and lenghts
                nroutes=len(routes)
                routes=[]

while max_dif>td:#------------GENETIC ALGORITHM--

        if con==0:#INITIAL POPULATION. con is the number of iteration
                con=con+1
                if ts>n:
                        ts=n
                sp=int(factor*math.factorial(n)/(math.factorial(ts)*math.factorial(n-ts)))#size of the initial population
                print "First iteration. Size population: ", sp,". Calculating routes...\n"
                routes_list=[]
                routes_length=[]
                if ts>n:
                        ts=n
                for x in range(0,sp):# create the routes
                        new_route=random.sample(act_points,ts)#a route is created as a random sample of ts=4 of all
                        # the available points
                        d=0
                        route=[0]+new_route+[0]
```

```
                    for y in range(0,len(route)-1):
                        d=d+points_list[route[y]].dist[route[y+1]]
                    routes_list.append(new_route)
                    routes_length.append(d)
            routes=zip(routes_length,routes_list)
            routes.sort()
            routes_length,routes_list=zip(*routes)#ordered calculated routes
            routes_length=list(routes_length)
            routes_list=list(routes_list)

            max_dif=100*(routes_length[sp-1]-routes_length[0])/routes_length[0]#calculus of max_dif
        else:#not the first iteration. The population is reduced and mutated
            con=con+1
            sp=len(routes)#number of routes
            ngr=int(math.ceil(sp*0.2)) #number of good routes (20 percent of the total)
            nbr=sp-ngr #number of bad routes
            print "Iteration number",con,". Size population: ", int(ngr+math.ceil(0.5*ngr)+math.ceil(0.4*nbr)),".
Calculating routes...\n"
            routes=routes_list
            routes_list=routes_list[:ngr]#best routes, they are not going to change in the current iteration
            routes_length=routes_length[:ngr]
            #mutation of good routes
            mut_good_routes=random.sample(routes_list,int(math.ceil(0.5*ngr)))
            for x in mut_good_routes:
                random.shuffle(x)#each route keeps the same points but the order is altered
                d=0
                route=[0]+x+[0]
                for y in range(0,len(route)-1):
                    d=d+points_list[route[y]].dist[route[y+1]]#calculus of new distances
                routes_list.append(x)
                routes_length.append(d)
```

```
        #mutation of worse routes
        bad_routes=routes[ngr:]
        bad_routes=random.sample(bad_routes,int(math.ceil(0.4*nbr)))
        for x in bad_routes:
                random.shuffle(x)#each route keeps the same points but the order is altered
                d=0
                route=[0]+x+[0]
                for y in range(0,len(route)-1):
                        d=d+points_list[route[y]].dist[route[y+1]]#calculus of new distances
                routes_list.append(x)
                routes_length.append(d)
        routes=zip(routes_length,routes_list)
        sp=len(routes)
        routes.sort()
        routes_length,routes_list=zip(*routes)
        routes_length=list(routes_length)
        routes_list=list(routes_list)
        max_dif=100*(routes_length[sp-1]-routes_length[0])/routes_length[0]

ent=True
r=[]
travel=False
while ent:#Current iteration menu
        print "The longest route is ", max_dif,"% longer than the shortest"
        print"What do you want to do"
        print"1.Show routes"
        print"2.Execute a route"
        print"3.Continue\n"#continue iterating
        print"Write the corresponding number: \n"
        menu=valid_input([1,2,3])
        if menu==1:
                if nroutes>10:
```

```
                nroutes=10 #maximun number of real good routes displayed
        print nroutes," real best routes"#Just to compare, the user only can choose among the routes
        # obtained with GA
        for x in range(0,nroutes):
                print "Route ",x,all_routes[x],". Longitude: ",all_lengths[x]
        print "\n Calculated routes"
        for x in range(0,sp):
                print "Route ",x,": ", routes_list[x],". Longitude: ",routes_length[x]
        raw_input("Press any button to continue")
        os.system('cls')
elif menu==2:
        print "Enter the route number"
        r=valid_input(range(0,len(routes)))
        #guidedmode(Frankie)
        print r#Once selected the route, the execution starts

        elev_eval(uav, altitude, 0.96, 3,3)
        #manual_elev(uav,altitude)
        globalvariables.valid_disc=False
        print "Executing"
        for x in routes_list[r]:
                points_list[x].setsta(2)
        con=0
        for x in routes_list[r]:
                con=con+1

                ent1=True
                while ent1:#look the code above to see the explanation of how the loop works and has
                        # to be used
                        uav.simple_goto(points_list[x].gps)
                        print "Going to Point number: ",x,"\n"
                        eval_distxy(uav, points_list[x].gps , 1, 7, 0)
```

```
                        send_ned_velocity(uav,0,0,0,5,1)
                        print "Stop number ",con,". Point number: ",x
                        descend(uav,x)
                        if points_list[x].state==0:
                                ent1=False
                        while uav.mode!="GUIDED":
                                print "Waiting for the guided mode"
                                time.sleep(5)
                        if uav.mode=="GUIDED":
                                globalvariables.gui2other=False

                travel=True#travel is true if all the points in the route were visited
                while True:
                        print "Going to home position"
                        uav.simple_goto(home)
                        eval_distxy(uav, home , 1, 7, 0)
                        if globalvariables.gui2other==False: #If at this point there wasn't any mode change, it's
                                # completed
                                globalvariables.valid_disc=True
                                break
                        while uav.mode!="GUIDED":
                                print "Waiting for the guided mode"
                                time.sleep(5)
                        if uav.mode=="GUIDED":
                                globalvariables.gui2other=False

                while uav.mode=="GUIDED":
                        print"Take the control, change the mode"
                        time.sleep(5)
                ent=False
        else:#more iterations
                ent=False
```

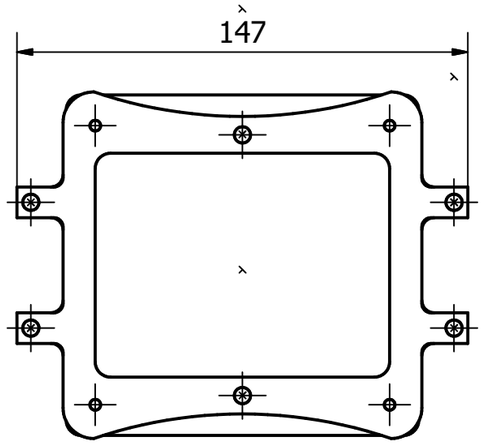
```
                if travel==True:#if the route was made succesfully, the program returns to the main menu
                        break


def descend(uav,x):#the function simulates when the uav is over a delivery point, so moves the vehicle down,
#ask the user wheter the package was already received, set the state in 0 and then moves the uav up to continue.
        global points_list
        if uav.mode=="GUIDED":
                print "Descending..."
        send_ned_velocity(uav,0,0,0.5,10,1)#down
        send_ned_velocity(uav,0,0,0,2,1)#stabilize the uav

        if uav.mode=="GUIDED":
                raw_input("Giving supplies. Press any button to continue ")#ask the user to continue
                points_list[x].setsta(0)
        send_ned_velocity(uav,0,0,-0.5,10,1)#up
        send_ned_velocity(uav,0,0,0,2,1)

def mode_callback(self, attr_name, msg):#this function along with the add_attribute_listener in the main code
#allows to monitor whether there is a mode change, and make some actions if that happens

        print "Mode change detected"
        print Frankie.mode.name
        if Frankie.mode != "GUIDED":#For an autopilot to follow the instructions from a PC is needed the uav mode
                #is guided
                print "Manual disconnection"#so if the mode is not guided, there is a "disconnection" between the uav and
                #the PC
                if globalvariables.valid_disc==False:
#if the UAV is in a moment when is supossed to be controlled by the PC, the indicator of disconnection (gui2other)
# becomes true
                        globalvariables.gui2other=True
```

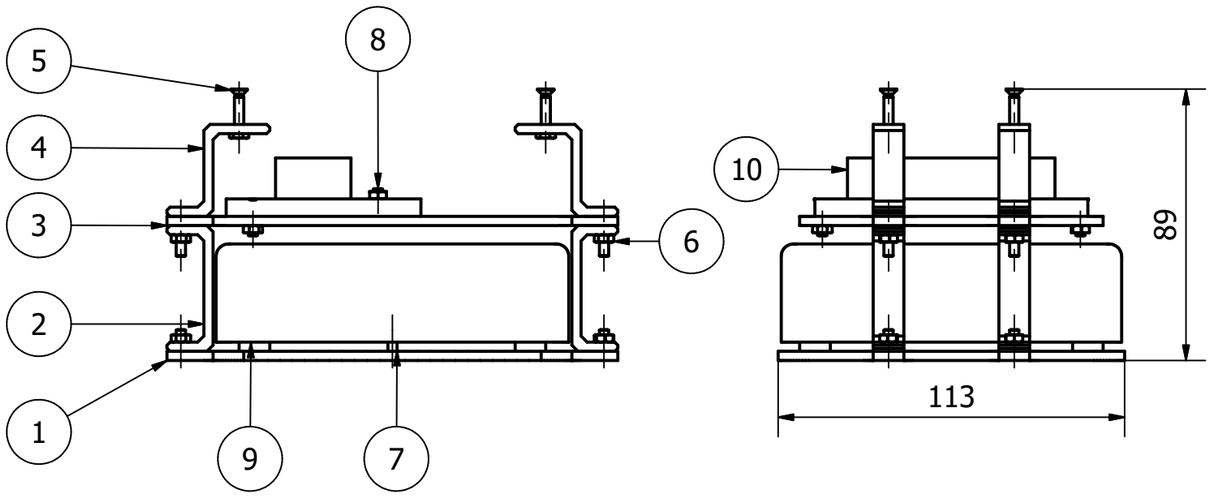
```
                print "Mode callback. Turn to guided mode to continue with the route"
                time.sleep(5)
        print "\n"

#-----------------MAIN CODE----------------------------

parser = argparse.ArgumentParser(description='Autonomous Frankie test mission')
parser.add_argument("--connect", help="Serial port of the connection")
parser.add_argument("--alt", help="Altitude", type=float)
parser.add_argument("--bat_volt", help="Minimal safe battery voltage", type=float)
parser.add_argument("--bat_lev", help="Minimal safe battery level", type=float)
args=parser.parse_args()
port=args.connect
altitude=args.alt
globalvariables.BatteryVolt=args.bat_volt
globalvariables.BatteryLevel=args.bat_lev

Frankie = connect(port, baud=57600, wait_ready=False, heartbeat_timeout=60)#connecting to the UAV
print "Connected"
Frankie.add_attribute_listener('mode', mode_callback)



global points_list#the list that contains all the gps points to be visited
points_list=[]
#guidedmode(Frankie)
#arm(Frankie,2,0)
home=Point(0)#assigning the home position as the point number 0
home.setpos(Frankie.location.global_relative_frame)#assigning location
home.add_dist(0)
points_list.append(home)
```

```
"""positions=[[-35.362577,149.1638929],[-35.363,149.165],[-35.365,149.168],[-35.362,149.167],[-
35.361902,149.165433],[-35.362660,149.166798],[-35.363765,149.165850]]
for x in positions:
        setpoint(LocationGlobal(x[0],x[1],home.gps.alt+10))"""
positions=[[10,10],[-10,4],[-8,-6],[9,-6],[-14,0],[0,-10]]# distances as vectors, each one has 2 components
#first component for north and south and second for east and west. Positive values for east and north, and
#negative for west and south.
for x in positions:
        setpoint(get_location_metresxyz(home.gps,x[0],x[1],altitude))
#turning these vectors into gps cordinates and assigning them as points for the uav to go
managepoints(Frankie,altitude)#call to managepoints to control al the funcionalities of the program
```

10.5 Planos mecánicos del nuevo soporte mecánico

Los planos se muestran a partir de la siguiente página.





Lista de Partes					
Form.	Item	Denominación	Designación	Cant.	Descripción
		Documentos:			
A3		Ensamble General	AU_0		
A3		Dibujo Explosionado	AU_1		
		Piezas:			Procedimiento y acabado: Impresión 3D FDM Material: ABS
A3	1	Base NUC	AU_0_01	1	
A3	2	Separador bases	AU_0_02	4	
A3	3	Base Gateway	AU_0_03	1	
A3	4	Sostenedor	AU_0_04	4	
		Elementos estandarizados:			
	5	Perno M3 x 16	AU_0_05	8	Une las piezas 4 al UAV y las 2, 3 y 4
	6	Tuerca M3	AU_0_06	15	
	7	Perno M3 x 10	AU_0_07	6	Une la pieza 9 a la 1 y la 1 a la 2
	8	Perno M3 x 12	AU_0_10	3	Sujetan la pieza 10
	9	Mini PC Intel Nuc 7i5BNK	AU_0_08	1	Representación simbólica
	10	Puente Ethernet	AU_0_09	1	Representación simbólica

TEC | Tecnológico de Costa Rica

ÁREA ACADÉMICA DE INGENIERÍA MECATRÓNICA

CURSO: Proyecto de Graduación

REVISIÓN: 1

	NOMBRE	FIRMA	FECHA
DIBUJ.	P. Araya		28/2/2019
VERIF.	P. Araya		28/2/2019
APROB.	P. Araya		28/2/2019
FABR.			

ACABADO:

NOTAS DE FABRICACIÓN:

DENOMINACIÓN: Ensamble General

TOLERANCIAS:
LINEAL: X ±0.5
.X ±0.3
.XX ±0.05
ANGULAR: X ±0.5°

MATERIAL:

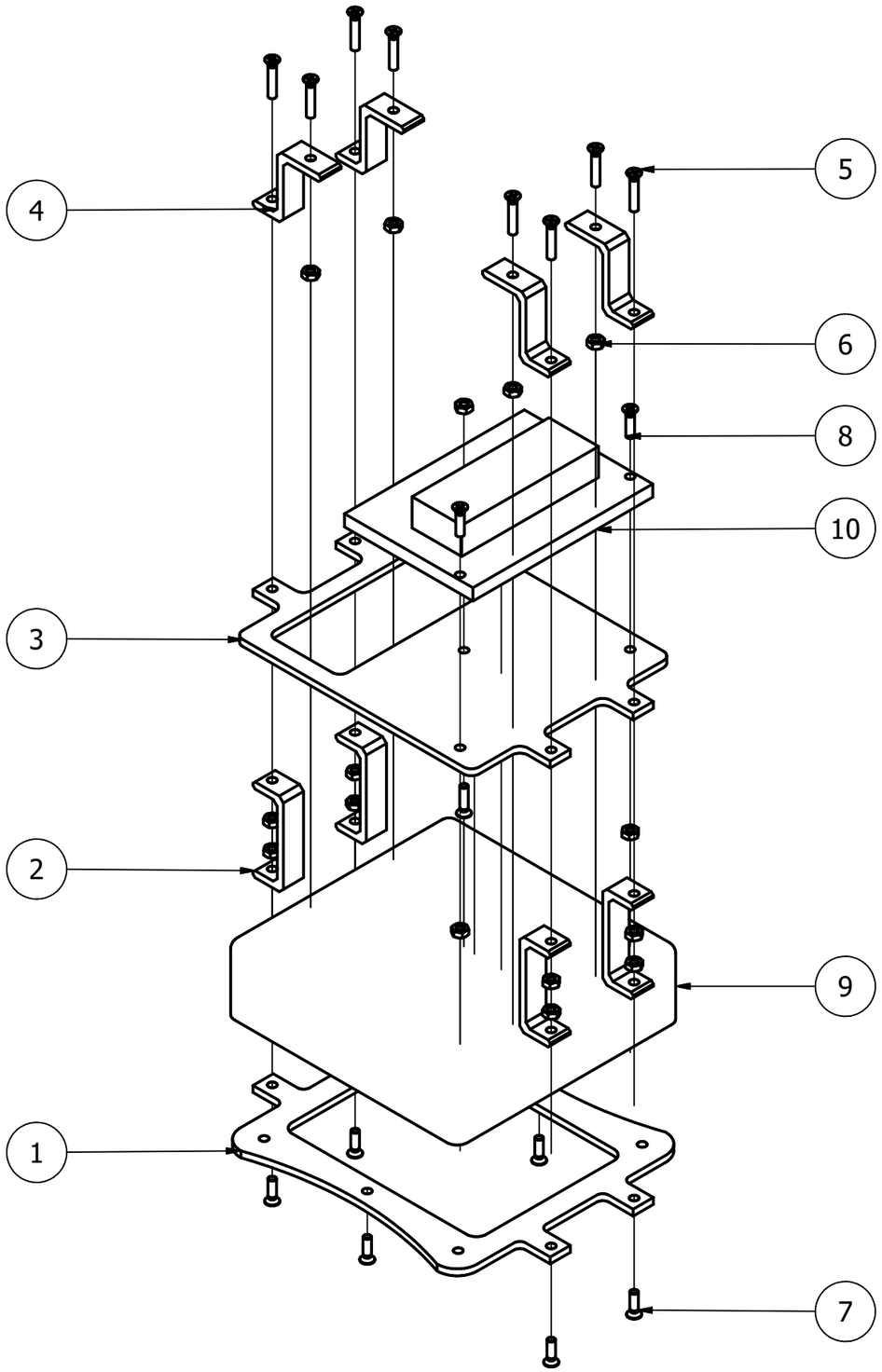
DESIGNACIÓN: AU_0

FORMATO: A3

ACOTACIÓN: mm

ESCALA: 1:2

HOJA 1 DE 4



Lista de Partes					
Form.	Item	Denominación	Designación	Cant.	Descripción
	1	Documentos:			
A3		Ensamble General	AU_0		
A3		Dibujo Explosionado	AU_1		
		Piezas:			Procedimiento y acabado: Impresión 3D FDM Material: ABS
A3	1	Base NUC	AU_0_01	1	
A3	2	Separador bases	AU_0_02	4	
A3	3	Base Gateway	AU_0_03	1	
A3	4	Sostenedor	AU_0_04	4	
		Elementos estandarizados:			
	5	Perno M3 x 16	AU_0_05	8	Une las piezas 4 al UAV y las 2, 3 y 4
	6	Tuerca M3	AU_0_06	15	
	7	Perno M3 x 10	AU_0_07	6	Une la pieza 9 a la 1 y la 1 a la 2
	8	Perno M3 X 12	AU_0_08	3	Sujetan la pieza 10
	9	Mini PC Intel Nuc 7i5BNK	AU_0_09	1	Representación simbólica
	10	Puente Ethernet	AU_0_10	1	Representación simbólica

TEC | Tecnológico de Costa Rica

ÁREA ACADÉMICA DE INGENIERÍA MECATRÓNICA

CURSO: Proyecto de Graduación

REVISIÓN: 1

	NOMBRE	FIRMA	FECHA
DIBUJ.	P. Araya		28/2/2019
VERIF.	P. Araya		28/2/2019
APROB.	P. Araya		28/2/2019
FABR.			

ACABADO:

NOTAS DE FABRICACIÓN:

DENOMINACIÓN: Dibujo Explosionado

TOLERANCIAS:
LINEAL: X ±0.5
.X ±0.3
.X ±0.05
ANGULAR: X ±0.5°

MATERIAL:

DESIGNACIÓN: AU_1

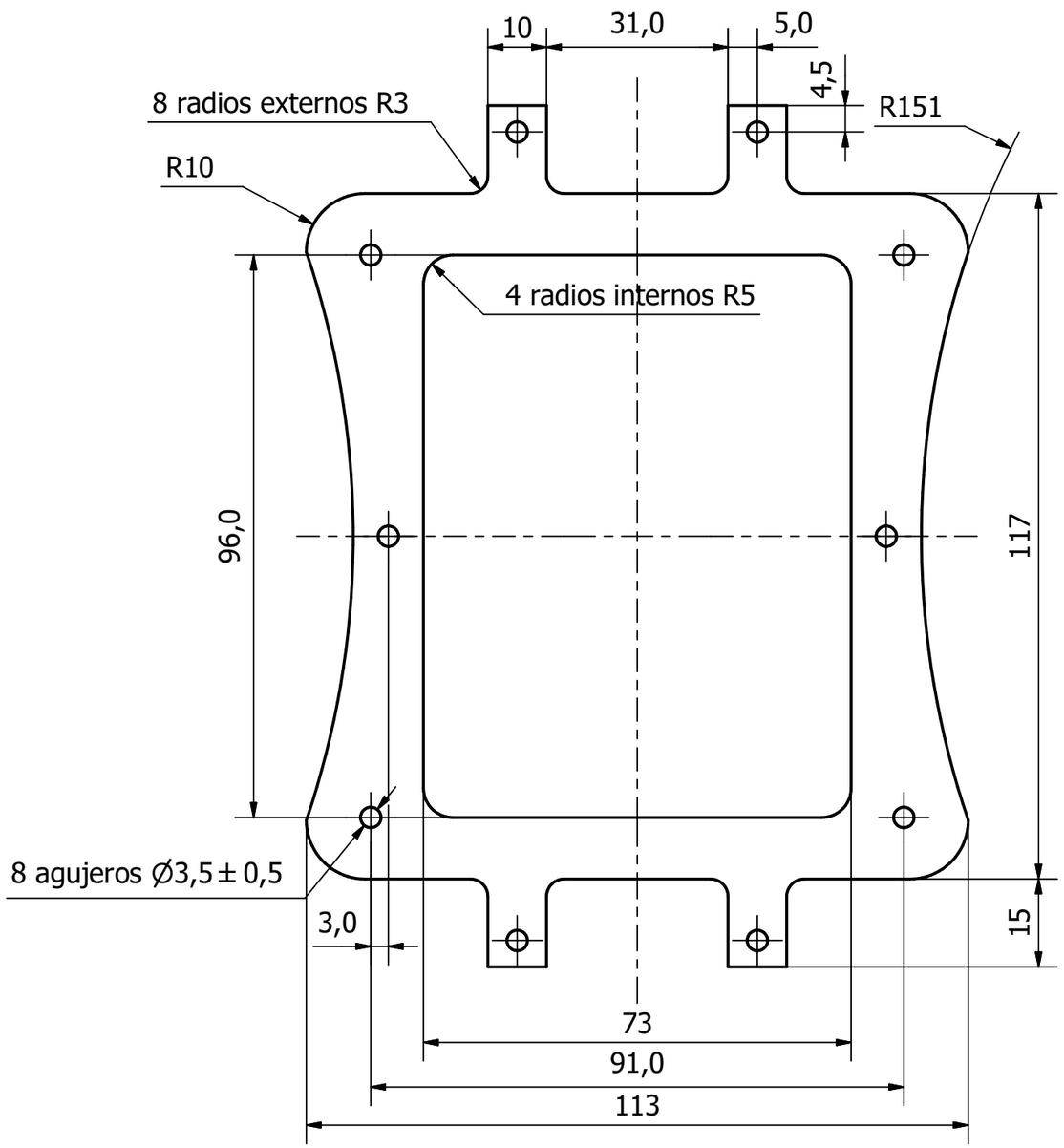
FORMATO: A3

ACOTACIÓN: mm

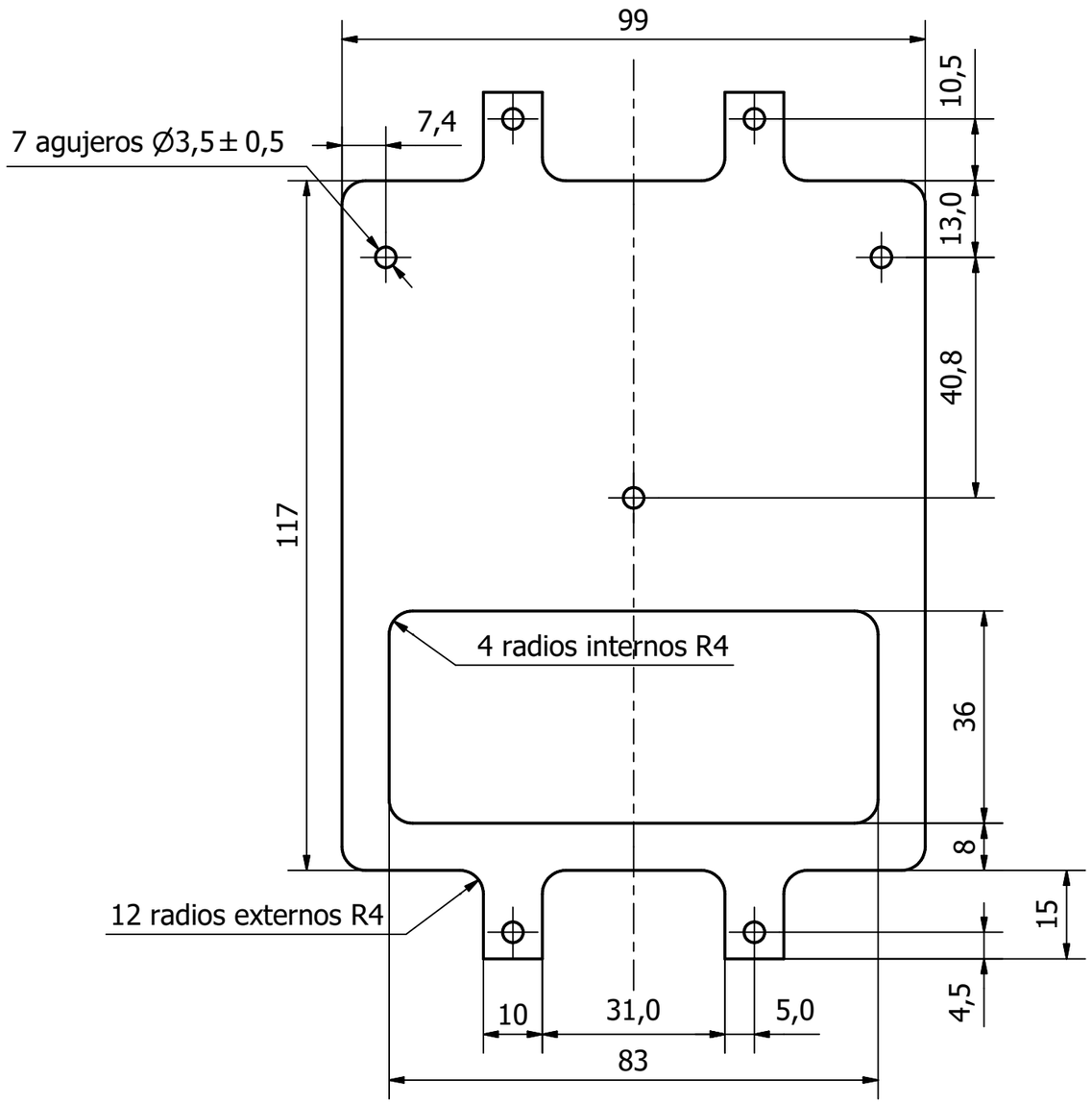
ESCALA: 1:2

HOJA 2 DE 4

AU_0_1



AU_0_3



Tanto la pieza AU_0_1 y AU_0_3 tienen un espesor de 3 mm

Las cotas cuya tolerancia no se encuentren en el plano, se rigen por la información de tolerancias del cajetín.

Para todas las superficies se admite una rugosidad Ra menor o igual 64 µm

TEC | Tecnológico de Costa Rica

ÁREA ACADÉMICA DE INGENIERÍA MECATRÓNICA

CURSO: Proyecto de Graduación

REVISIÓN: 1

	NOMBRE	FIRMA	FECHA
DIBUJ.	P. Araya		28/2/2019
VERIF.	P. Araya		28/2/2019
APROB.	P. Araya		28/2/2019
FABR.			

ACABADO:

NOTAS DE FABRICACIÓN: Modelado por deposición fundida (FDM)

DENOMINACIÓN: Base NUC y Base Gateway

TOLERANCIAS:
LINEAL: X ±0.5
.X ±0.3
.XX ±0.05
ANGULAR: X ±0.5°

MATERIAL: Acrilonitrilo butadieno estireno (ABS)

DESIGNACIÓN: AU_0_1 Y AU_0_3

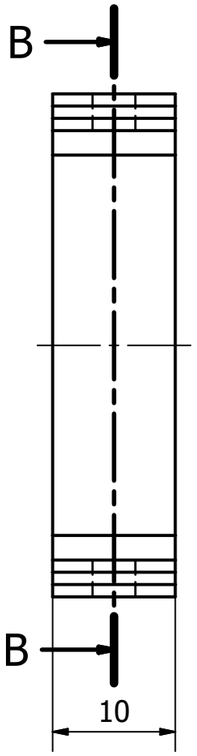
FORMATO: A3

ACOTACIÓN: mm

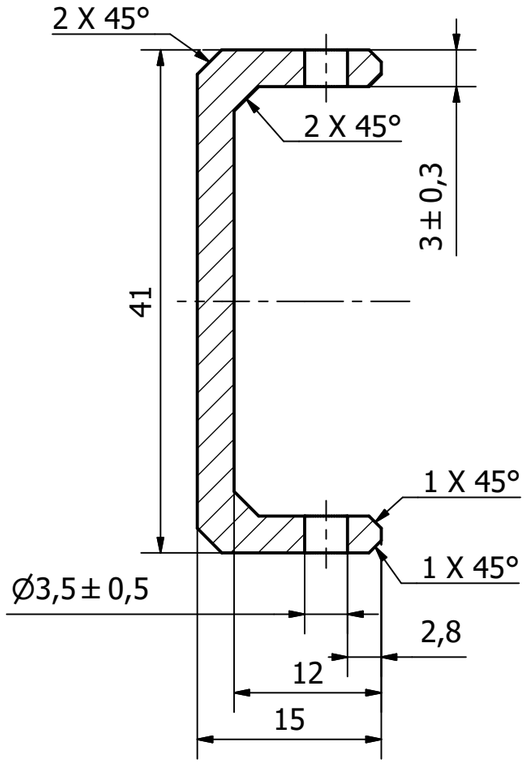
ESCALA: 1:2

HOJA 3 DE 4

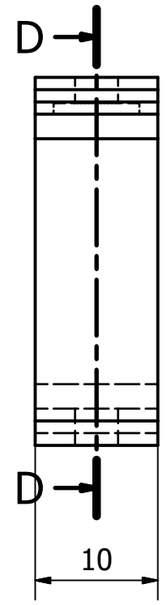
AU_0_2



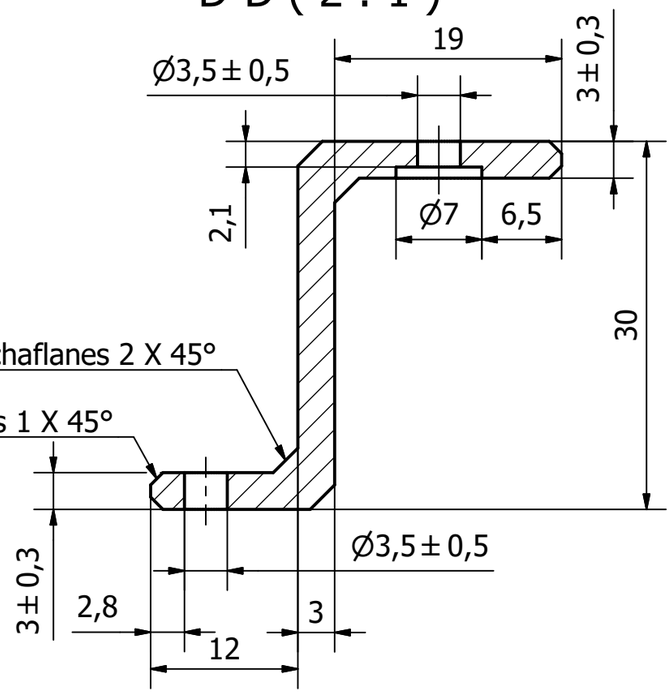
B-B (2 : 1)



AU_0_4



D-D (2 : 1)



Las cotas cuya tolerancia no se encuentren en el plano, se rigen por la información de tolerancias del cajetín.

Para todas las superficies se admite una rugosidad Ra menor o igual 64 µm

TEC | Tecnológico de Costa Rica		ÁREA ACADÉMICA DE INGENIERÍA MECATRÓNICA	REVISIÓN: 1
		CURSO: Proyecto de Graduación	

	NOMBRE	FIRMA	FECHA
DIBUJ.	P. Araya		28/2/2019
VERIF.	P. Araya		28/2/2019
APROB.	P. Araya		28/2/2019
FABR.			

ACABADO: Remover soportes de impresión	DENOMINACIÓN: Separador bases y Sostenedor		
NOTAS DE FABRICACIÓN: Modelado por deposición fundida (FDM)			
TOLERANCIAS: LINEAL: X ±0.5, .X ±0.3, .XX ±0.05; ANGULAR: X ±0.5°	MATERIAL: Acrilonitrilo butadieno estireno (ABS)	DESIGNACIÓN: AU_0_2 Y AU_0_4	FORMATO: A3
	ACOTACIÓN: mm	ESCALA: 2:1	HOJA 4 DE 4