



# Universidad Nacional del Sur

## **Informe Final:**

Desarrollo de un piloto automático para un dron acuático.

Hipperdinger Federico

Julio 2024

## **Índice**

<b>1. Introducción</b>	<b>3</b>
<b>2. Desarrollo</b>	<b>4</b>
<b>3. Conclusiones</b>	<b>9</b>

## 1. Introducción

En este informe se presentan los resultados del proyecto final de la materia Programación de Dispositivos Móviles y Embebidos (PDMyE). El objetivo de este proyecto era la implementación de un piloto automático para un dron acuático tipo catamarán (Figura 1), utilizando una Raspberry Pico, que reemplace el ArduPilot Flight Controller que se encontraba en uso (Figura 2). Este piloto, no solo permitiría implementar las mismas funciones, sino que también brindaría un acceso total al código fuente y hardware, dando la posibilidad de personalizarlo según las necesidades, integrándolo con diversos sensores, implementando lógica interna, etc.



Figura 1: Dron acuático.

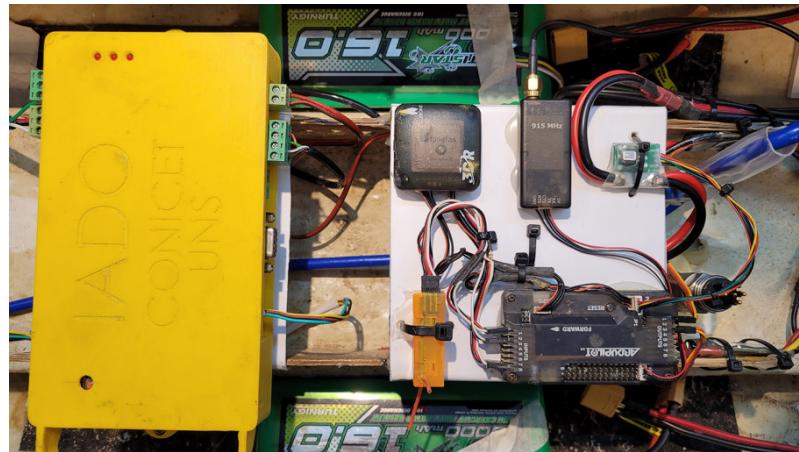


Figura 2: Sistema de control del dron.

## 2. Desarrollo

El proyecto se dividió en dos partes: el desarrollo del piloto per se y la implementación de una Ground Station, lo cual recibió mayor prioridad.

Como ground station se decidió utilizar 'Mission Planner'. Esta aplicación multiplataforma, de código abierto, provee todo lo requerido para una ground station, desde un mapa interactivo hasta una consola para monitorear sensores, quitando la necesidad de desarrollar una aplicación propia. La contraparte de esto, es que se necesita implementar una comunicación entre el dron y la Ground Station, que respete el protocolo MavLink utilizado por Mission Planner.

MAVLink es un protocolo de mensajería ligero para comunicarse con drones y sus componentes. Sigue un patrón de diseño moderno híbrido de publicación-suscripción y punto a punto: los flujos de datos se envían/publican como temas, mientras que los subprotocolos de configuración como el protocolo de misión o el protocolo de parámetros son punto a punto con retransmisión [1].

Los mensajes se definen dentro de archivos XML. Cada archivo XML define el conjunto de mensajes soportados por un sistema MAVLink particular, también conocido como un "dialecto". Existen generadores de código que crean bibliotecas de software para lenguajes de programación específicos a partir de estas definiciones de mensajes XML, para luego ser utilizadas por drones, estaciones de control terrestre y otros sistemas MAVLink para comunicarse [2]. En nuestro caso, mavgen o Mavgenerate (GUI de mavgen) pueden ser utilizados para python.

El problema con estos archivos XML, como el 'common.xml', es que incluyen una cantidad de mensajes muy grande e innecesaria para nuestro propósito, al punto que no son soportados por una raspberry pico. En principio uno podría modificar estos archivos XML, pero esto ya ha sido hecho para la raspberry pico en el proyecto de GitHub 'mav\_rp2040' del usuario stephendade [3], por lo que se procedió a utilizar su implementación, la cual incluye los siguientes mensajes:

- AHRS (163)
- DATA16 (169)
- DATA32 (170)
- DATA64 (171)
- DATA96 (172)
- HEARTBEAT (0)
- SYS\_STATUS (1)
- SYSTEM\_TIME (2)
- GPS\_RAW\_INT (24)
- ATTITUDE (30)
- VFR\_HUD (74)
- COMMAND\_INT (75)
- COMMAND\_LONG (76)
- COMMAND\_ACK (77)

- NAMED\_VALUE\_FLOAT (251)
- NAMED\_VALUE\_INT (252)
- STATUSTEXT (253)
- SCALED\_IMU (26)

A partir de esta librería se generó un módulo propio, 'Mav\_communication.py', con la clase 'Mavlink\_communication'. Esta clase implementa toda la comunicación requerida para que Mission Planner plotee la posición y trayectoria del dron e incluya datos de telemetría, como la velocidad y valores de 3 pines ADC (Figura 3). El funcionamiento se corroboró utilizando un conversor TTL a USB conectado, por un lado, a la PC y, por otro, a la UART1 de la Pico. Cualquier otro método de comunicación soportado por Mission Planner debería poder utilizarse sin ningún tipo de modificación por fuera del código requerido para reemplazar la UART.

Dentro de 'Mavlink\_communication', el método 'sendFloats' se encarga de enviar los datos relacionados al GPS junto a 3 mediciones de ADC y un heartbeat, requerido para mantener la comunicación con Mission Planner activa. Aquí pueden agregarse más datos de telemetría, si es necesario, con el método 'named\_value\_float\_encode'. Por otro lado, el método 'checkForMessages' tiene 2 funcionalidades: establecer y mantener la comunicación vigente con Mission Planner a partir de los heartbeats y recibir mensajes del tipo 'STATUSTEXT', strings a partir de los cuales se pueden establecer comandos para el piloto. A modo de ejemplo, el código interpreta 3 comandos ('command1', 'command2' y 'command3') que generan distintas secuencias de blinks en el led integrado de la Raspberry Pico. Estos mensajes pueden ser enviados desde la ventana 'actions' del Mission Planner (Figura 4).

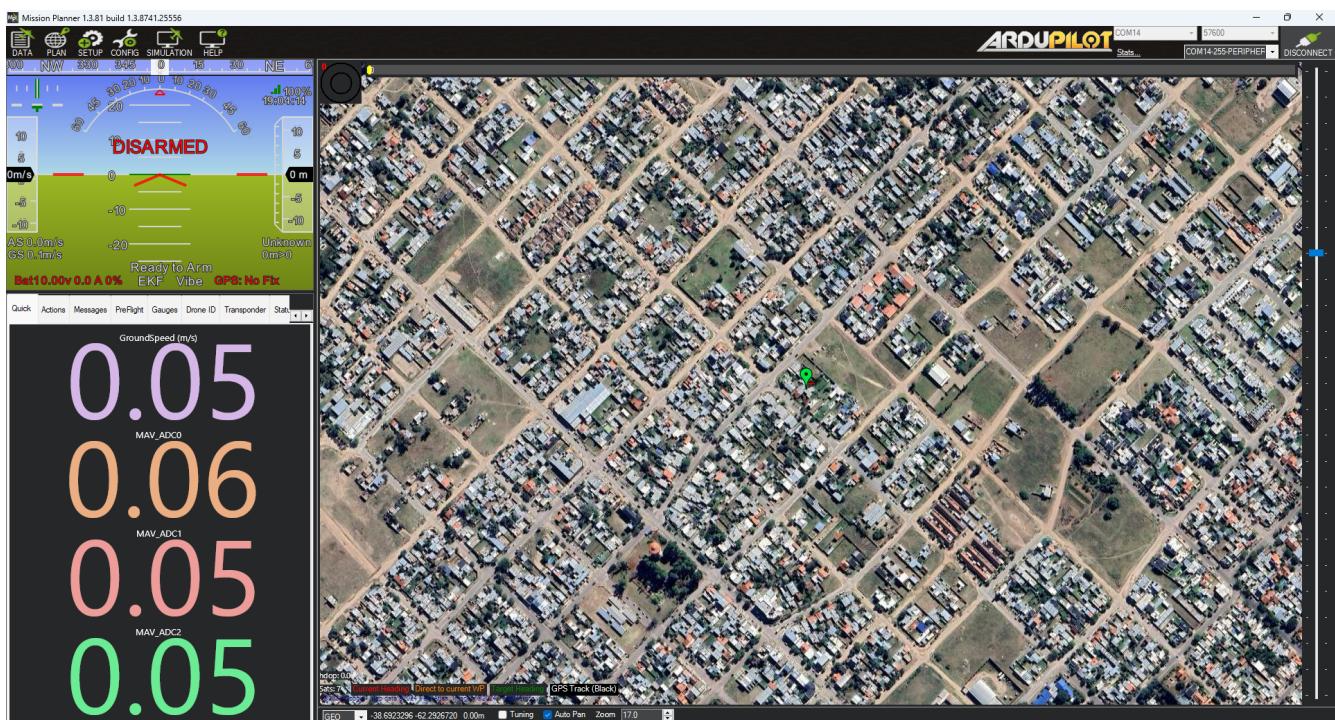


Figura 3: Ground Station - Mission Planner.

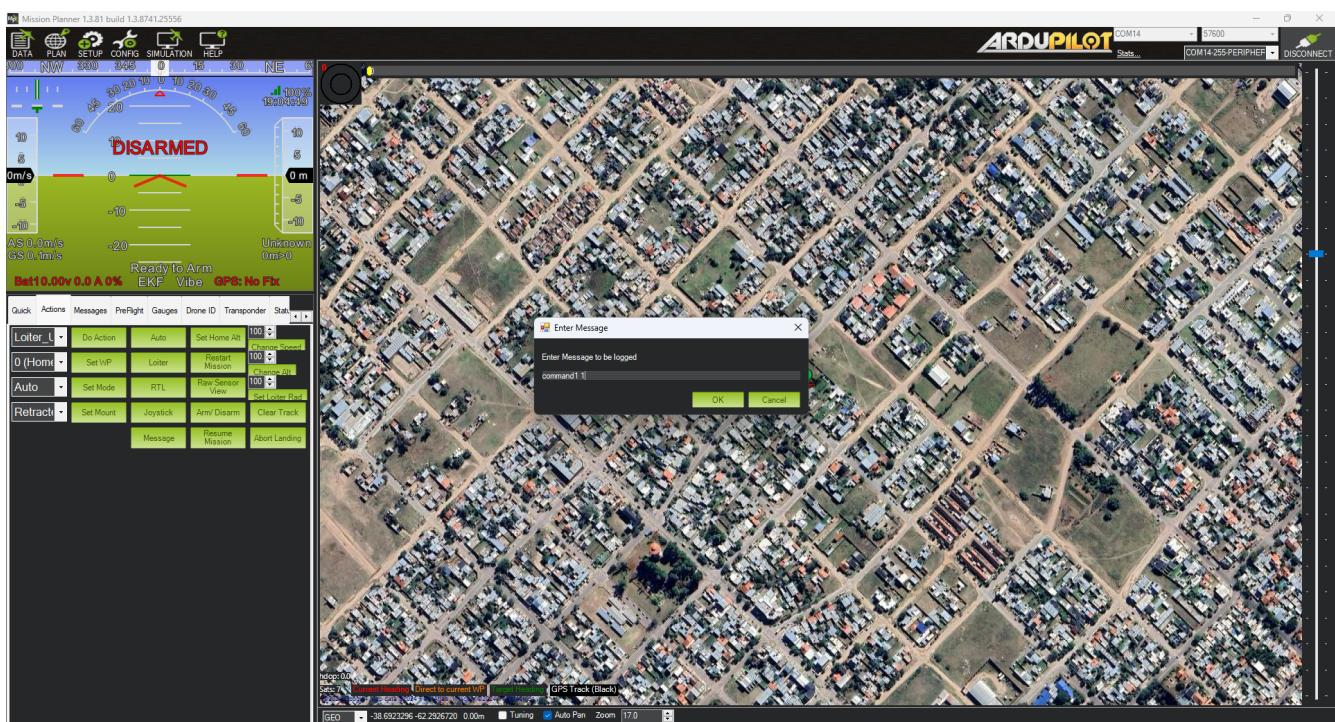


Figura 4: Envío de comandos desde Mission Planner.

La telemetría GPS es obtenida a partir de un módulo GPS GY-NEO6MV2 (Figura 5)[4][5]. Este módulo utiliza sentencias del estándar NMEA-0183, las cuales solo nos interesan las sentencias 'GPGGA' y 'GPRMC' [6]. De todos los datos que proveen, se utilizan:

- Hora actual (UTC)
- Latitud (en formato DDMM.MMM)
- Dirección de la brújula de la latitud
- Longitud (en formato DDDMM.MMM)
- Dirección de la brújula de la longitud
- Velocidad (en nudos por hora)
- Fix (Tipo de fijación; 0 para sin fijación, 1 para GPS, 2 para DGPS)
- Número de satélites usados para la fijación
- Altitud sobre el nivel medio del mar
- Unidades de altitud (M para metros)

Todo lo que contempla la obtención y manipulación de estos datos es manejado en la clase GPS implementada en el módulo 'GPS\_module.py'.

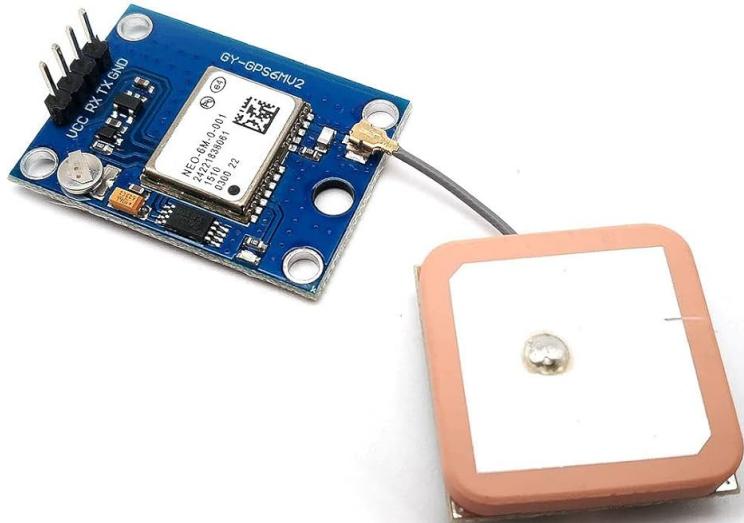


Figura 5: Módulo GPS GY-NEO6MV2.

Finalmente, el piloto automático fue implementado, dentro del módulo 'pilot\_module.py', en la clase 'Pilot'. Esta incorpora un objeto PID y un objeto GPS para encargarse de recolectar la telemetría gps y, a partir de ella, controlar los motores del dron para dirigirlo a la coordenada objetivo. En la creación del

objeto pilot, se toma como parámetro el nombre del archivo de la misión, del tipo json o txt, que debe contener una lista de listas de la forma:

$$\text{waypoint\_list} = [[lat_0, long_0], [lat_1, long_1], \dots, [lat_N, long_N]].$$

El piloto, una vez inicializado, toma su posición inicial como coordenada 'home', a la que debe retornar una vez terminada la misión. Luego determina su primera coordenada objetivo y avanza durante unos segundos para tomar su nueva posición y así orientarse (ya que no se posee un magnetómetro para determinar desde un inicio la orientación del dron). Una vez inicializado, debe llamarse periódicamente al método 'update' para que el dron tome su coordenada actual y corrija su rumbo. Una vez alcanzado el objetivo, cambia automáticamente al siguiente y, si ya se han recorrido todas las coordenadas de la lista, se da por finalizada la misión y se define la coordenada 'home' como destino final. Este método retorna tres flags que indican si el objetivo actual fue alcanzado, si se está retornando a 'home' y si se ha terminado la misión, las cuales pueden ser útiles en la toma de decisiones desde el main.

### 3. Conclusiones

Como pudo observarse en la sección 2, la implementación del ground station fue exitosa. Tanto las coordenadas como los demás datos de telemetría fueron correctamente transferidos a Mission Planner. Además, gracias a un proyecto en paralelo llevado a cabo por otro alumno, se pudo conectar una cámara wifi a la misma aplicación a partir de 'R-Click ->Video ->Set MJPEG Source'.

Respecto al piloto, se realizaron ensayos que parecieran indicar un correcto funcionamiento del mismo. Sin embargo, estos ensayos no se llevaron a cabo en el mismo dron, por lo que no se pudo validar con certeza el algoritmo. Además, se implementaron pocas medidas preventivas para aumentar la robustez del sistema, como la respuesta ante la pérdida de señal de GPS. Por lo tanto, avanzar en estos aspectos permitiría completar el piloto, algo que no fue posible debido a limitaciones de tiempo.

## Referencias

- [1] “Mavlink - página principal.”  
<https://mavlink.io/en/>.
- [2] “Mavlink - library generation.”  
[https://mavlink.io/en/getting\\_started/generate\\_libraries.html#mavgen](https://mavlink.io/en/getting_started/generate_libraries.html#mavgen).
- [3] “Github mav\_rp2040 library.”  
[https://github.com/stephendade/mav\\_rp2040](https://github.com/stephendade/mav_rp2040).
- [4] “Gps gy-neo6mv2 specs.”  
<https://www.datasheethub.com/gy-neo6mv2-flight-control-gps-module/>.
- [5] “Gps gy-neo6mv2 tutorial.”  
<https://microcontrollerslab.com/neo-6m-gps-module-raspberry-pi-pico-micropython/>.
- [6] “Nmea standar.”  
<https://w3.cs.jmu.edu/bernstdh/web/common/help/nmea-sentences.php>.