



UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA MECÁNICA



**IMPLEMENTACIÓN DE UNA ARQUITECTURA PARA VALIDACIÓN DE
CONTROLADORES DE AERONAVES DE ALA FIJA EN X-PLANE**

POR

Germán Adolfo Quijada Arriagada

Profesor guía:

Bernardo Andrés Hernández Vicente

6 de diciembre de 2023

Índice general

1 Definición del problema	4
1.1 Contexto	4
1.2 Planteamiento del problema	5
1.3 Objetivo	6
1.3.1 Objetivos específicos	6
1.4 Condiciones de diseño	6
1.4.1 Entorno de desarrollo	6
1.4.2 Diseño de algoritmos de control	7
1.5 Metodología de trabajo	7
1.5.1 Selección de autopiloto base	7
1.5.2 Extensión de protocolo	7
1.5.3 Prototipo de implementación de nuevos algoritmos	8
1.5.4 Formulación de metodología para implementación de algoritmos	8
2 Marco teórico y estado del arte	9
3 Entorno de desarrollo	10
3.1 Selección de autopiloto base	10
3.2 Extensión de protocolo de comunicación	11
3.2.1 Ingresar información de sensores a ArduPilot	12
3.2.2 Ejecución de acciones de control en X-Plane	22
3.2.3 Calibración de acelerómetro y magnetómetro	24
3.2.4 Pruebas en X-Plane	26
3.2.5 Otros descubrimientos y problemas durante el desarrollo	28
3.2.6 Puesta en marcha	28
4 Diseño de algoritmos de control	30
4.1 Modificación de algoritmo existente	30
4.2 Implementación de nuevo algoritmo	30
5 Documentación	31

6 Conclusión	32
A Carta Gantt	36

Índice de figuras

1.1 Aeronave de ala fija “Parrot Disco”	4
1.2 Categorización de software autopiloto de código abierto.	5
1.3 Técnica “Processor-in-Loop”	7
3.1 Autopiloto Pixhawk	10
3.2 Software de estación en tierra QGroundControl	11
3.3 Funcionamiento inoFS	12
3.4 Programa intermedio de inoFS	12
3.5 Ubicación en memoria del giroscopio con información de salud de las lecturas	13
3.6 Puertos en Pixhawk	14
3.7 Archivos de código fuente para sistemas de estimación de actitud externos	15
3.8 Selección de sistema inercial externo en Mission Planner	15
3.9 Registro de inicialización de Mission Planner	16
3.10 Primera versión de plug-in “HITL” para X-Plane	16
3.11 Raspberry Pi Pico conectado a Pixhawk 4	17
3.12 Comunicación entre X-Plane y autopiloto	17
3.13 Especificación de función de puerto serial en Mission Planner	18
3.14 Lecturas de inclinación y orientación en QGroundControl	18
3.15 Diferencia entre norte magnético y geográfico	20
3.16 Aeronave en Mission Planner	21
3.17 Configuración de estimación de posición y actitud en Mission Planner	21
3.18 Chequeos de prearmado	23
3.19 Instrucciones de calibración de acelerómetro	24
3.20 Aeronave en calibración nivelada	25
3.21 Aeronave en calibración inclinada a la izquierda	26
3.22 Calibración de magnetómetro completa en Mission Planner	26
3.23 Vuelo con radio en X-Plane 9	27
3.24 Aeronave RC en misión autónoma en X-Plane 9	27
3.25 Plug-in funcionando en X-Plane 11	28

Capítulo 1

Definición del problema

1.1 Contexto

Actualmente, las aeronaves no tripuladas son utilizadas para actividades recreativas tanto como comerciales en la industria. Por nombrar algunas aplicaciones estas pueden ser agricultura donde se requiere monitorear cultivos, topografía para creación de mapas y planificación, búsqueda y rescate donde las aeronaves pueden abarcar un gran área mientras transmiten imagen en alta resolución, o para la fotografía en general. En la Figura 1.1 se puede observar una aeronave tipo ala delta llamada “Parrot Disco” caracterizada por su capacidad de grabar video en alta resolución y ofrecer visión en primera persona, su controlador se puede programar con el software ArduPilot donde en su documentación se ofrecen instrucciones para ello [1].



Figura 1.1: Aeronave de ala fija “Parrot Disco”.

Los controladores que gobiernan el vuelo de una aeronave no tripulada son sistemas autopiloto que cuentan con funciones que van desde asegurar la estabilidad en un vuelo manual hasta la ejecución de misiones completamente autónomas. Estos sistemas son programados con un software que tiene como tarea básica extraer lecturas del estado actual desde los sensores y generar acciones de control apropiadas para la actual aplicación. Existe una amplia variedad de software autopiloto que se puede

categorizar en la función para la que está diseñado y de si el código fuente del mismo está disponible o no, en la Figura 1.2 se presenta una muestra de los software autopiloto existentes de código abierto según su aplicación [2].

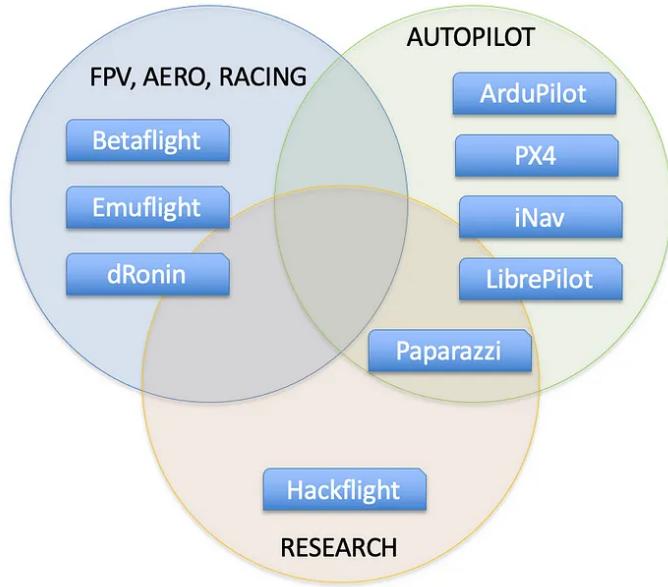


Figura 1.2: Categorización de software autopiloto de código abierto.

1.2 Planteamiento del problema

Los software controlador de RPAS mantenidos actualmente como ArduPilot o Pixhawk incluyen sistemas de autopiloto y vuelo asistido generalmente a base de algoritmos PID. En algunos casos como con Pixhawk los detalles de implementación del controlador están documentados [3], permitiendo que un usuario experimentado pueda **ajustar los parámetros en busca de un mejor desempeño**. Quizás se quiera lograr un vuelo más estable si se está pilotando manualmente [4], o en misiones autónomas reducir el consumo de las baterías, maximizar la distancia a recorrer o reducir el tiempo que una misión pueda tomar. Otra manera de lograr estos objetivos **puede ser reemplazando por completo el algoritmo PID por controladores basados en modelos dinámicos como lo son LGR y LQR**. Si bien estos tipos de controlador han sido probados [5] los software autopiloto no ofrecen una arquitectura accesible que permita el prototipado rápido y la validación de estos sistemas. El prototipado rápido en este caso definido como el abrir acceso al código de los algoritmos para su modificación e iteración, donde se encuentren las entradas de los sensores y se genere la acción de control. La validación del sistema modificado sería el confirmar que en cierto contexto el algoritmo de control se pueda sintonizar a una aeronave y ejecutar planes de vuelo adecuadamente. Debido a lo experimental de una arquitectura así y lo costoso que puede ser una aeronave no tripulada se decide por su implementación en un ambiente simulado. Específicamente la verificación con X-Plane, donde el prototipado se realizará con un autopiloto que maneje una aeronave

en simulación y la validación tendrá valor al lograrse la sintonización de aeronaves simuladas, pero desarrolladas por equipos profesionales donde el vuelo es comparable con la realidad.

1.3 Objetivo

Implementación de una arquitectura para validación de nuevos algoritmos de control en microcontroladores para aeronaves de ala fija por medio de técnica Processor-in-Loop con X-Plane.

1.3.1 Objetivos específicos

1. Estudio de algoritmos de control y opciones de personalización disponibles para estos en los software de autopiloto actuales y selección de software autopiloto base.
2. Desarrollo de extensión de protocolo de comunicación de X-Plane para controladores autopiloto.
3. Implementación de distintos algoritmos de control a los ya implementados en el software y validación en simulador X-Plane como prototipo de arquitectura.
4. Establecer y documentar proceso sistemático para la implementación de algoritmos de control en la nueva arquitectura.

1.4 Condiciones de diseño

Al tratarse de un trabajo para el laboratorio de técnicas aeroespaciales, es de especial interés que la arquitectura sea accesible para alumnos y docentes en la facultad y el actual informe debe poder servir de documentación. Para estructurar la presentación del proyecto este se puede dividir en dos partes, el entorno de desarrollo que servirá como base para la investigación en X-Plane con un autopiloto, y el estudio de diseño de algoritmos de control.

1.4.1 Entorno de desarrollo

El entorno de desarrollo debe ser construido como una extensión del protocolo de comunicación con X-Plane establecido en el proyecto de ingeniería [6], habilitando una interfaz a microcontroladores con software autopiloto donde X-Plane emule las funciones que haría una aeronave real. X-Plane debe proporcionar lecturas de sensores simulados y responder apropiadamente a entradas de control de radio lo cual es una técnica conocida como “Processor-in-Loop” [7]. Esta técnica facilita que el trabajo realizado en simulación pueda ser aplicado en aeronaves reales, puesto que para el controlador autopiloto no existirá diferencia entre estar conectado a X-Plane o funcionando en una aeronave real. Los autopilotos además se conectan a un software de estación en tierra que habilita una interfaz gráfica que permite monitorear el vuelo y crear una ruta para la misión [8], entre otras cosas. El diseñar la arquitectura alrededor de software como ArduPilot o PX4 permite aprovechar todas estas funciones existentes. Los sistemas de autopiloto cuentan con interfaces estandarizadas para conectar sensores y entradas de radiocontrol. Es imperativo seleccionar una base que cuente con estos estándares definidos

para la posterior implementación de los sensores simulados y la entrada de radiocontrol. El autopiloto base debe ser uno de los que ya se tenga experiencia en el laboratorio como PX4 o ArduPilot.

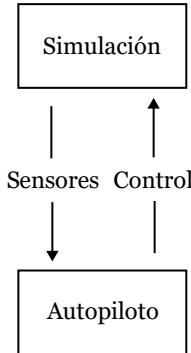


Figura 1.3: Técnica “Processor-in-Loop”

1.4.2 Diseño de algoritmos de control

La funcionalidad de habilitar la modificación de los algoritmos de control existentes o la implementación de uno nuevo debe incorporar el uso de lenguajes diseñados para ello, como Simulink, en el que se tenga acceso a las entradas como lecturas de los sensores y a generar acciones de control. Un algoritmo podrá ser validado en el software de simulación X-Plane como primera iteración, en este se podrá probar la interfaz entre el controlador de vuelo y X-Plane, la capacidad de sintonizar el algoritmo a aeronaves específicas y la realización de misiones autónomas. Esto será parte de un proceso sistemático con instrucciones con el objetivo final de que los algoritmos puedan ser probados en aeronaves reales. El procedimiento de diseño de controladores fomentará la experimentación con base en herramientas ya utilizadas en el laboratorio como lo son Simulink o Python. Con especial enfoque en aeronaves de ala fija.

1.5 Metodología de trabajo

1.5.1 Selección de autopiloto base

A partir de las condiciones de diseño establecidas se realizará una comparación entre las soluciones de autopiloto existentes. Además se investigarán trabajos ya realizados que hayan tenido objetivos similares a los de este proyecto, en especial el de modificar e implementar algoritmos de control puesto que en mi opinión es la parte más difícil.

1.5.2 Extensión de protocolo

Con el autopiloto base seleccionado, se debe proceder a replicar las interfaces de hardware estandarizadas para simular la experiencia de una aeronave real, donde X-Plane junto al protocolo de comunicación la emularán. Los datasheets de los sensores disponibles serán utilizados para replicar las señales que estos entregan al controlador de acuerdo a su capacidad o precisión. Se estudiarán los factores adversos

que pudiesen ocurrir afectando la radiofrecuencia de control debido al clima o distancias para también hacer una estimación de ellos. Todo esto se implementará en el software inoFS [9] en su programación. Para la interfaz de hardware se conectará a la Raspberry Pi Pico un bus de comunicación de acuerdo al estándar de autopiloto [10], sea uno proporcionado por el laboratorio o replicado con los insumos electrónicos que se deban comprar. Para que este objetivo pueda ser logrado no hace falta modificar el código del autopiloto, si los algoritmos PID logran funcionar con simulaciones de X-Plane es suficiente.

1.5.3 Prototipo de implementación de nuevos algoritmos

Ya sea a partir de esfuerzos previos realizados por la comunidad o el estudio del código fuente del autopiloto base se implementara otro sistema de control que se ejecute en el microcontrolador con el software de autopiloto. Al existir la extensión de protocolo probada con PID el funcionamiento del nuevo prototipo de controlador podrá ser verificado en X-Plane de la misma manera. El desarrollo del nuevo algoritmo será a partir de sistemas creados en Simulink y Python, donde el código final será exportado a C o C++ para su integración en el microcontrolador de autopiloto. El objetivo se considerará listo cuando se demuestre que el nuevo algoritmo logra cumplir tareas básicas como estabilización o el ajuste autónomo de las condiciones de vuelo a otras condiciones preestablecidas.

1.5.4 Formulación de metodología para implementación de algoritmos

Durante el desarrollo del prototipo probablemente se habrán intentado varias alternativas para lograr el funcionamiento adecuado, el último objetivo es preparar una metodología junto a documentación que defina el camino adecuado para la implementación de nuevos algoritmos en microcontroladores con software de autopiloto. Se reducirá el procedimiento a los pasos mínimos esenciales en documentación web del repositorio de inoFS, además de lo reportado en el informe de memoria de título. La metodología abarcará desde la creación de un algoritmo sencillo hasta su verificación en X-Plane.

Capítulo 2

Marco teórico y estado del arte

github funcionamiento basico ardupilot usuario funcionamiento basico interno de ardupilot (sensores -> ekf -> rc + control -> servo) mission planner y parametros

Capítulo 3

Entorno de desarrollo

Este capítulo describe el proceso seguido para establecer el entorno de desarrollo, incluyendo los problemas que se encontraron, como se solucionaron y finalmente el resultado obtenido. Este entorno servirá de base para el trabajo posterior en los siguientes capítulos que, como está descrito en la sección de condiciones de diseño, tiene como función principal habilitar la validación del trabajo realizado por medio de la técnica “Processor-in-Loop”.

3.1 Selección de autopiloto base

En el laboratorio de técnicas aeroespaciales se cuenta con hardware Pixhawk programado con el firmware de fábrica PX4. Existe un requisito que inmediatamente acota la selección de autopiloto, el software debe ser de código abierto debido a que posiblemente este se deba modificar en caso de que las prestaciones del software no sean suficientes para cumplir los objetivos.



Figura 3.1: Autopiloto Pixhawk

El hardware autopiloto Pixhawk se puede programar con otro software distinto al del fabricante, entre aquellas soluciones las que son aplicables según las condiciones de diseño y de código abierto son ArduPilot, iNAV y Paparazzi, además del mismo PX4 [2]. Revisando a un poco más detalle cada

alternativa se puede notar que, iNAV no ofrece opciones de vuelo autónomo avanzadas como aterrizaje automático, que serán de interés en el capítulo de diseño de algoritmos de control. Paparazzi es una opción bastante interesante que si tiene un sistema autónomo completo por medio de archivos “XML” con la información del plan de vuelo [11], pero no es compatible con el software de control en tierra utilizado por ArduPilot y PX4, siendo Mission Planner y QGroundControl los principales.



Figura 3.2: Software de estación en tierra QGroundControl

PX4 es una opción atractiva ya que ofrece soporte para trabajar con técnicas “Hardware-in-the-Loop” con simuladores incluyendo X-Plane [12]. Sin embargo, aunque ArduPilot haya deprecado esa opción [13] existe más documentación en comparación con PX4 sobre como modificar existentes e implementar nuevos algoritmos de control [14]. Por esto último se decide en ArduPilot como plataforma de desarrollo.

3.2 Extensión de protocolo de comunicación

El protocolo de comunicación desarrollado durante el proyecto de ingeniería aeroespacial, inoFS, permite a X-Plane comunicarse con un microcontrolador. La interfaz disponible para lograrlo es aquella disponible en prácticamente todos los controladores, UART o mejor conocida como serial sobre USB. Para microcontroladores que ofrezcan acceso a la red por wifi o Ethernet inoFS también puede trabajar con paquetes UDP.

Los detalles de su funcionamiento están en el informe de proyecto de ingeniería, pero cabe destacar el “Programa intermediario” en la Figura 3.3 que es el principal responsable de traducir y reenviar los mensajes entre el simulador y el microcontrolador, se puede observar el programa funcionando junto a X-Plane en la Figura 3.4.

Para poder extender el protocolo de comunicación se debe ser capaz de realizar dos funciones, ingresar a

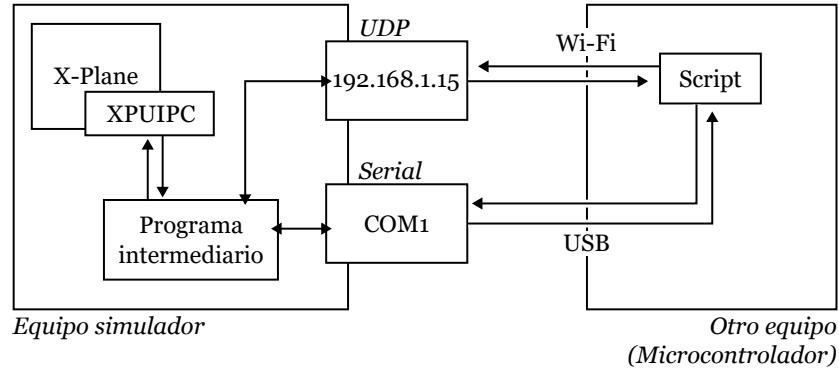


Figura 3.3: Funcionamiento inoFS

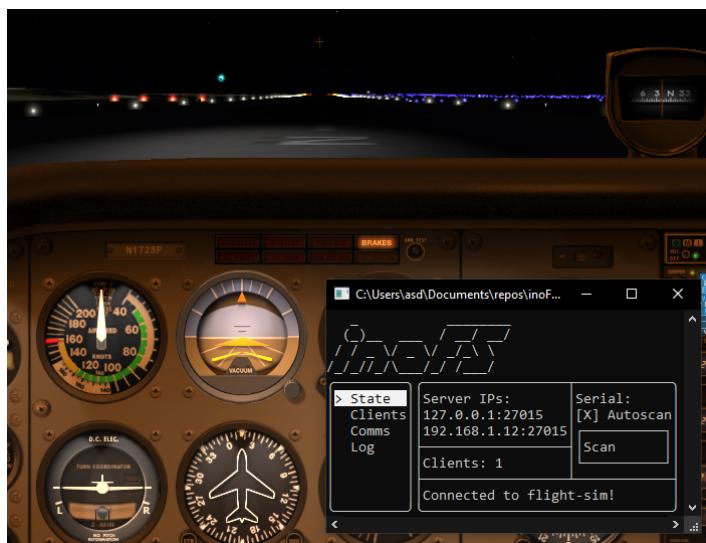


Figura 3.4: Programa intermediario de inoFS

ArduPilot el estado de vuelo de X-Plane como respuesta obtenida de sensores, y el ejecutar las acciones de control producidas por ArduPilot en X-Plane. Con estas funciones se puede considerar lista la implementación “Processor-in-Loop” y a continuación se describe el proceso para lograr cada una.

3.2.1 Ingresar información de sensores a ArduPilot

La primera opción a evaluar para poder entregarle a ArduPilot información de sensores es la descrita en la metodología en el capítulo 1, emular la misma respuesta que producirían sensores de verdad con algún microcontrolador y comunicarla por los puertos apropiados como son los de la Figura 3.6. Para sensores sencillos que producen una respuesta análoga como un potenciómetro esto es posible, pero los sensores utilizados en RPAS en su mayoría son digitales, por ejemplo los sensores en el Pixhawk disponible en el laboratorio son los siguientes [15].

- ST Micro L3GD20H 16 bit gyroscope

- ST Micro LSM303D 14 bit accelerometer / magnetometer
- Invensense MPU 6000 3-axis accelerometer / gyroscope
- MEAS MS5611 barometer

Todos son sensores digitales, esto significa que al iniciar el sistema se les entrega alguna opción de configuración con la ayuda de un driver, y luego estos responden con las lecturas en un formato específico para cada sensor en forma de paquetes discretos. Por ejemplo el giroscopio L3GD20H espera que el microcontrolador (en este caso el que esté programado con ArduPilot) haga lecturas a ubicaciones en memoria del giroscopio [16]. Una de las ubicaciones de memoria en el giroscopio contiene información de si las lecturas se han visto saturadas o si hay un valor nuevo como se ve en la Figura 3.5. Emular cada sensor a una suficiente fidelidad capaz de “engañar” a ArduPilot de que está recibiendo lecturas reales es demasiado trabajo, por lo que se decide buscar alternativas.

7.9 STATUS (27h)

Table 37. STATUS register

ZYXOR	ZOR	YOR	XOR	ZYXDA	ZDA	YDA	XDA
-------	-----	-----	-----	-------	-----	-----	-----

Table 38. STATUS description

ZYXOR	X, Y, Z -axis data overrun. Default value: 0 (0: no overrun has occurred; 1: new data has overwritten the previous one before it was read)
ZOR	Z axis data overrun. Default value: 0 (0: no overrun has occurred; 1: a new data for the Z-axis has overwritten the previous one)
YOR	Y axis data overrun. Default value: 0 (0: no overrun has occurred; 1: a new data for the Y-axis has overwritten the previous one)
XOR	X axis data overrun. Default value: 0 (0: no overrun has occurred; 1: a new data for the X-axis has overwritten the previous one)
ZYXDA	X, Y, Z -axis new data available. Default value: 0 (0: a new set of data is not yet available; 1: a new set of data is available)
ZDA	Z axis new data available. Default value: 0 (0: a new data for the Z-axis is not yet available; 1: a new data for the Z-axis is available)
YDA	Y axis new data available. Default value: 0 (0: a new data for the Y-axis is not yet available; 1: a new data for the Y-axis is available)
XDA	X axis new data available. Default value: 0 (0: a new data for the X-axis is not yet available; 1: a new data for the X-axis is available)

Figura 3.5: Ubicación en memoria del giroscopio con información de salud de las lecturas

El hardware autopiloto donde puede funcionar ArduPilot (como es el Pixhawk disponible en el laboratorio) incluye puertos serial disponibles para muchas funciones como son el recibir lecturas de sensores que trabajen con esa interfaz, enviar y recibir mensajes que utilizan el protocolo de telemetría MavLink [17] o para alguna aplicación personalizada que el usuario quisiera programar por medio de scripting en el lenguaje Lua [18].

La siguiente alternativa evaluada es enviar lecturas de sensores generadas con un protocolo propio por alguno de los puertos serial, y programar ArduPilot para que las interprete como lecturas de un sensor personalizado. Esto último es difícil, porque se debe programar, recompilar y cargar el nuevo firmware al autopiloto.

Primero se consideró aprovechar el motor de scripting en Lua para esta tarea. La ventaja de utilizar scripts es que no necesita modificar el código fuente de ArduPilot y por lo tanto tampoco reprogramar por completo el microcontrolador, ya que los scripts se cargan de manera dinámica desde la tarjeta SD.



Figura 3.6: Puertos en Pixhawk

Lamentablemente los scripts no cuentan con la funcionalidad de poder generar lecturas de sensores críticos como son el IMU o GPS por lo que se descarta esta opción.

Estudiando el código fuente de ArduPilot se identificó que este soporta sistemas de estimación de actitud externos o “External Attitude Heading Reference System”, están compuestos por una combinación de sensores y un procesador que realiza su propia fusión exportando un cuaternión describiendo la actitud junto con posición y velocidad en un marco de referencia inercial. Actualmente ArduPilot soporta tres sistemas de estimación de actitud externos disponibles en el mercado [19].

- MicroStrain 3DM® Series
- VectorNav VN-300 AHRS
- VectorNav VN-100 AHRS

ArduPilot ofrece la opción de utilizar la estimación de actitud producida por el sistema externo directamente, o alternativamente solo recibir las mediciones de los sensores y por medio de un filtro de Kalman generar su propia estimación. En el contexto del proyecto ambas opciones son útiles, el que ArduPilot reciba la información de actitud perfecta de X-Plane permite descartar esta variable de los problemas que se vayan a producir durante las pruebas del controlador personalizado. Y el solo incorporar las lecturas de sensores y que ArduPilot realice su propia estimación con filtro de Kalman es lo que este haría en un vuelo real.

Se decidió escribir un driver para un nuevo sistema de estimación de actitud externo ficticio el cual recibirá lecturas de sensor y actitud por medio de conexión serial, que además entregara por esta misma conexión la respuesta de la acción de control que esté produciendo ArduPilot como salidas de servo. La creación de este nuevo driver en el lado de ArduPilot implica que se debe modificar el código fuente, en la Figura 3.7 se pueden observar los archivos que componen los sistemas de estimación de actitud externos existentes. La interfaz principal AP_ExternalAHRS y la implementación de los sistemas “MicroStrain” y “VectorNav”, en los archivos AP_ExternalAHRS_MicroStrain5 y AP_ExternalAHRS_VectorNav. El

código fuente del nuevo driver se escribió en los archivos AP_ExternalAHRS_HITL, hasta este punto con la implementación más básica que lograse compilar y ser cargada en un autopiloto de pruebas.

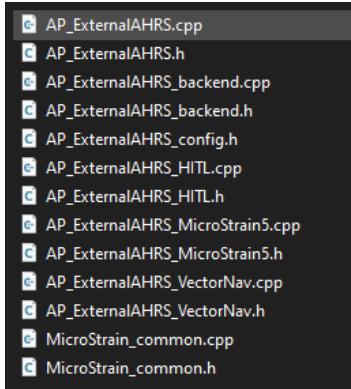


Figura 3.7: Archivos de código fuente para sistemas de estimación de actitud externos

El proceso de compilación de ArduPilot está documentado [20] y los firmware modificados se pueden cargar al autopiloto por USB de la misma forma que el oficial. En el laboratorio de técnicas aeroespaciales se proporcionó de un autopiloto Pixhawk 4 soportado por los desarrolladores de ArduPilot [21]. Para realizar la modificación del firmware de manera ordenada se creó un “fork” de la versión estable de ArduPlane en el momento de escritura del informe 4.4.2, sobre la que se escribe el nuevo driver y se lleva un registro de cada cambio en el repositorio en GitHub [22].

Una vez cargado ArduPilot modificado, con el programa de estación en tierra Mission Planner se ajusta el parámetro que indica que se utilizará un sistema inercial externo como se puede observar en la Figura 3.8. Mission Planner no reconoce el nombre de la nueva opción correctamente mostrando un espacio en blanco, pero al seleccionarla y reiniciar el autopiloto se puede comprobar que se está ejecutando el nuevo driver observando el registro como se ve en la Figura 3.9.

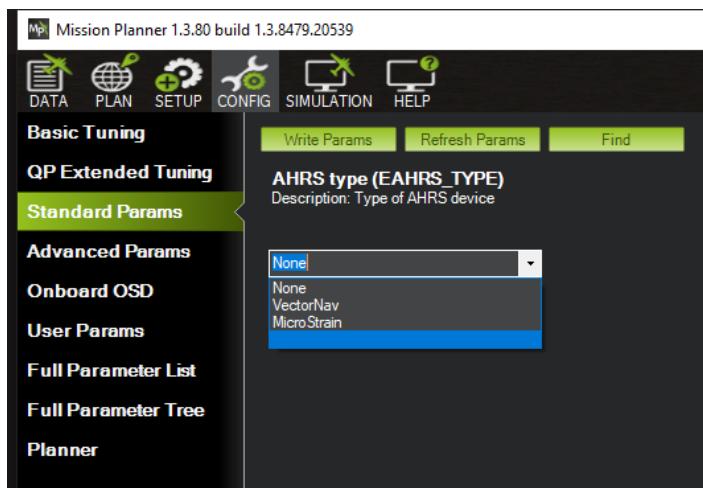


Figura 3.8: Selección de sistema inercial externo en Mission Planner

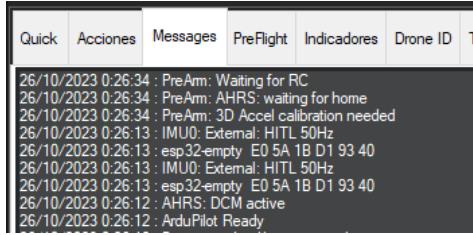


Figura 3.9: Registro de inicialización de Mission Planner

Durante las pruebas se descubrió un nuevo posible problema, como ha sido descrito hasta ahora se busca extender el protocolo de comunicación creado en el proyecto de ingeniería, inoFS. Sin embargo existe una limitación producto de su diseño. inoFS en sí extiende FSUIPC, el cual es un plug-in para Microsoft Flight Simulator que permite lectura y escritura arbitraria de variables internas del simulador, XPUIPC es una reimplementación del plug-in compatible con X-Plane que trabaja de la misma forma que FSUIPC. El problema es que la frecuencia a la que FSUIPC o XPUIPC pueden leer información de la simulación no está definida, y los sensores encontrados en los autopilotos pueden llegar a entregar lecturas a una frecuencia de hasta 1000 Hz como lo hace el MPU-6000. Además de la incertezza de la velocidad de FSUIPC está la latencia producto de tener que enviar la información a través del programa intermedio. Para asegurar la máxima frecuencia de datos posible se decidió no utilizar inoFS, y en su lugar crear un plug-in específico para X-Plane que envíe información por puerto serial directo luego de cada paso de la simulación, eliminando cualquier retraso posible en el canal de comunicación. Se lleva registro del desarrollo del plug-in en un repositorio en GitHub [23] de la misma manera que para el “fork” de ArduPilot.



Figura 3.10: Primera versión de plug-in “HITL” para X-Plane

El desarrollo de plug-ins para X-Plane se realiza programando instrucciones en C++ y enlazando el SDK (Software Development Kit) que da acceso a una selección de librerías útiles para interactuar con el simulador [24]. El entorno de desarrollo en específico para este trabajo es Visual Studio Code con el compilador de Microsoft MSVC [25], los archivos disponibles en el repositorio incluyen la configuración (en el archivo `.vscode/tasks.json`) para poder compilar el plug-in en el editor de texto si se ha instalado MSVC.

El plug-in proporciona lecturas de sensor ficticio y la actitud de la aeronave por medio de la interfaz serial. En primera instancia el dispositivo serial seleccionado en el plug-in es un Raspberry Pi Pico

conectado al computador por USB el cual retransmite la información al autopiloto también por serial por uno de los puertos de telemetría como se observa en las Figuras 3.11 y 3.12. El programa cargado en el microcontrolador (utilizando el entorno de desarrollo Arduino IDE 2) solo realiza la tarea de retransmitir los datos recibidos por serial USB a serial por los pines GPIO 4 y 5 y el código para cumplir la función es el Código 1

```
UART Serial2(4, 5, NC, NC);
void setup() {
    Serial.begin(115200);
    Serial2.begin(115200);
}
void loop() {
    if (Serial.available()) {
        Serial2.write(Serial.read());
    }
    if (Serial2.available()) {
        Serial.write(Serial2.read());
    }
}
```

Código 1: Retransmisor serial

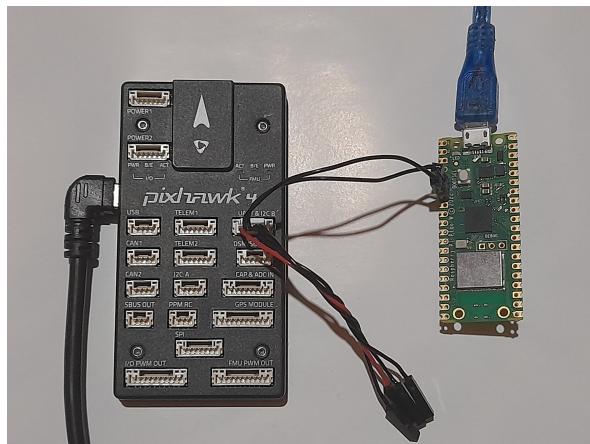


Figura 3.11: Raspberry Pi Pico conectado a Pixhawk 4

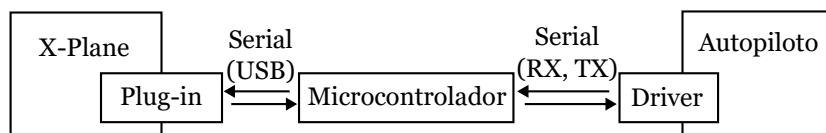


Figura 3.12: Comunicación entre X-Plane y autopiloto

Para terminar de conectar el Raspberry con el autopiloto, se configura el puerto de telemetría donde está conectado el microcontrolador como interfaz para sistema de estimación de actitud externo, como se observa en la Figura 3.13 seleccionando la opción “AHRS”.

Con todo configurado se logra comprobar que la información de X-Plane está siendo procesada por el

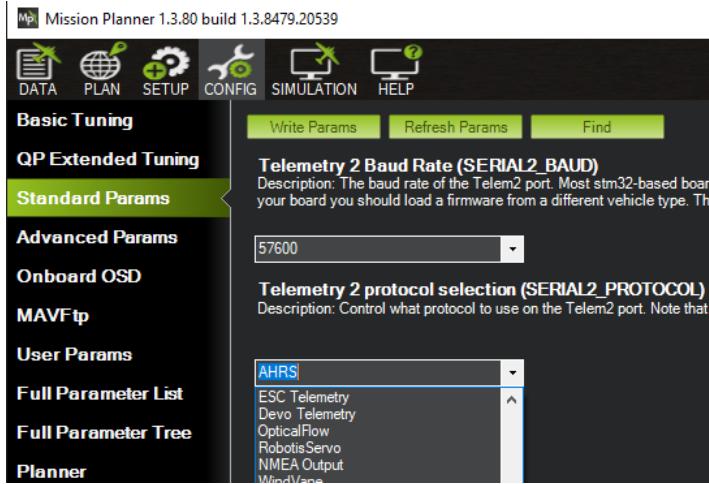


Figura 3.13: Especificación de función de puerto serial en Mission Planner

autopiloto, en la Figura 3.14 se observa que los movimientos del horizonte artificial del software en tierra (ahora QGroundControl) corresponden al de la inclinación de la aeronave. Hasta este punto se nota que el horizonte esta al revés porque aún falta procesar y calibrar un poco los datos para entregarlos en el formato que espera ArduPilot.



Figura 3.14: Lecturas de inclinación y orientación en QGroundControl

Como mencionado anteriormente los sistemas de estimación de actitud externos ofrecen dos tipos de información, lecturas de sensores y estimación de la actitud exportada como un cuaternión. La manera en que ArduPilot espera recibir esta información está descrita en las estructuras de datos que se observan en los Códigos 2 y 5.

Las estructuras de datos de lecturas de sensor corresponden a las de acelerómetro y giroscopio, barómetro, magnetómetro y GPS como aparecen respectivamente. Se debe obtener esta información de X-Plane por medio del plug-in y la forma en que este recopila los datos de la simulación es accediendo a los Dataref continuamente con la librería “XPLMDatAccess” del SDK [26], para la primera implementación del programa los Dataref utilizados se describen en el Código 3. Los Dataref son

```

// Código de ArduPilot
// Libraries/AP_ExternalAHRS/AP_ExternalAHRS.h
typedef struct {
    Vector3f accel;      // m/s2
    Vector3f gyro;       // rad/s
    float temperature;   // C
} ins_data_message_t;

typedef struct {
    uint8_t instance;
    float pressure_pa; // Pa
    float temperature; // C
} baro_data_message_t;

typedef struct {
    Vector3f field;
} mag_data_message_t;

typedef struct {
    uint16_t gps_week;
    uint32_t ms_tow;
    uint8_t fix_type;
    uint8_t satellites_in_view;
    float horizontal_pos_accuracy;
    float vertical_pos_accuracy;
    float horizontal_vel_accuracy;
    float hdop;
    float vdop;
    int32_t longitude;    // deg
    int32_t latitude;     // deg
    int32_t msl_altitude; // cm
    float ned_vel_north; // m/s
    float ned_vel_east;  // m/s
    float ned_vel_down; // m/s
} gps_data_message_t;

```

Código 2: Estructuras de datos para lecturas de sensor

variables internas de X-Plane que permiten acceder y modificar el estado de todos los componentes de la simulación, estos están disponibles de manera que desarrolladores pueden hacer uso fácilmente gracias a la extensa documentación [27]. La característica de enviar lecturas como telemetría por serial en el plug-in funciona en dos pasos, el primero es recopilar la información con los Dataref y el segundo es procesar los datos ya sea convertir unidades o generar la lectura del magnetómetro para luego enviarlos por serial como se observa en el Código 4, este proceso se repite en cada fotograma.

La lectura del magnetómetro no se puede generar de manera realista utilizando solo las librerías en la SDK de X-Plane. Un magnetómetro entrega un vector que apunta al norte magnético de la tierra con magnitud proporcional a la “intensidad” del campo, el cual se puede describir en términos de la “declinación” e “inclinación” que son los ángulos que forma este vector respecto al norte geográfico como se observa en la Figura 3.15. X-Plane solo ofrece acceso a la declinación por medio del Dataref `sim_flightmodel_position_magnetic_variation`, no a la inclinación ni a la intensidad del campo.

```
// Código de plug-in para X-Plane
// telemetry.cpp
XPLMDataRef accel_x = XPLMFindDataRef("sim/flightmodel/forces/g_axil");
XPLMDataRef accel_y = XPLMFindDataRef("sim/flightmodel/forces/g_side");
XPLMDataRef accel_z = XPLMFindDataRef("sim/flightmodel/forces/g_nrml");
XPLMDataRef gyro_x = XPLMFindDataRef("sim/flightmodel/position/P");
XPLMDataRef gyro_y = XPLMFindDataRef("sim/flightmodel/position/Q");
XPLMDataRef gyro_z = XPLMFindDataRef("sim/flightmodel/position/R");
XPLMDataRef quat = XPLMFindDataRef("sim/flightmodel/position/q");
XPLMDataRef baro = XPLMFindDataRef("sim/weather/barometer_current_inhg");
XPLMDataRef temperature = XPLMFindDataRef("sim/weather/temperature_ambient_c");
XPLMDataRef latitude = XPLMFindDataRef("sim/flightmodel/position/latitude");
XPLMDataRef longitude = XPLMFindDataRef("sim/flightmodel/position/longitude");
XPLMDataRef elevation = XPLMFindDataRef("sim/flightmodel/position/elevation");
XPLMDataRef local_vx = XPLMFindDataRef("sim/flightmodel/position/local_vx");
XPLMDataRef local_vy = XPLMFindDataRef("sim/flightmodel/position/local_vy");
XPLMDataRef local_vz = XPLMFindDataRef("sim/flightmodel/position/local_vz");
```

Código 3: Datarefs utilizados para telemetría

```
// Código de plug-in para X-Plane
// telemetry.cpp
void Telemetry::Loop(float dt) {
    UpdateState(); // Recopilar información de Datarefs
    ProcessState(); // Procesar y enviar por serial
}
```

Código 4: Loop de telemetría en plug-in para X-Plane

Sin embargo es posible obtener la inclinación e intensidad con librerías externas [28], puesto que el campo magnético de la tierra cambia muy lentamente. Durante el desarrollo del plug-in se realizaron pruebas entregando como lectura un vector que apunta perfectamente al norte geográfico y ArduPilot funcionó correctamente. Hasta que no se retrabaje la implementación para hacerla más realista esta solución es suficiente.

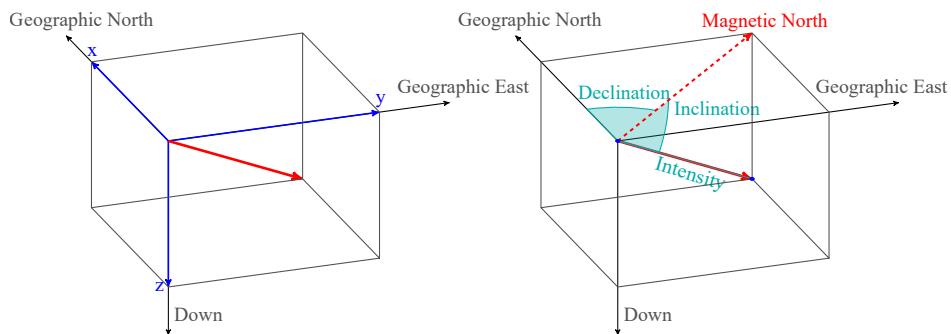


Figura 3.15: Diferencia entre norte magnético y geográfico

Con ArduPilot recibiendo las lecturas de sensores de la manera esperada se logra observar correctamente el estado de la aeronave en Mission Planner, además el filtro de Kalman incluido en el autopiloto logra realizar la estimación de actitud exitosamente como se observa en la Figura 3.16.

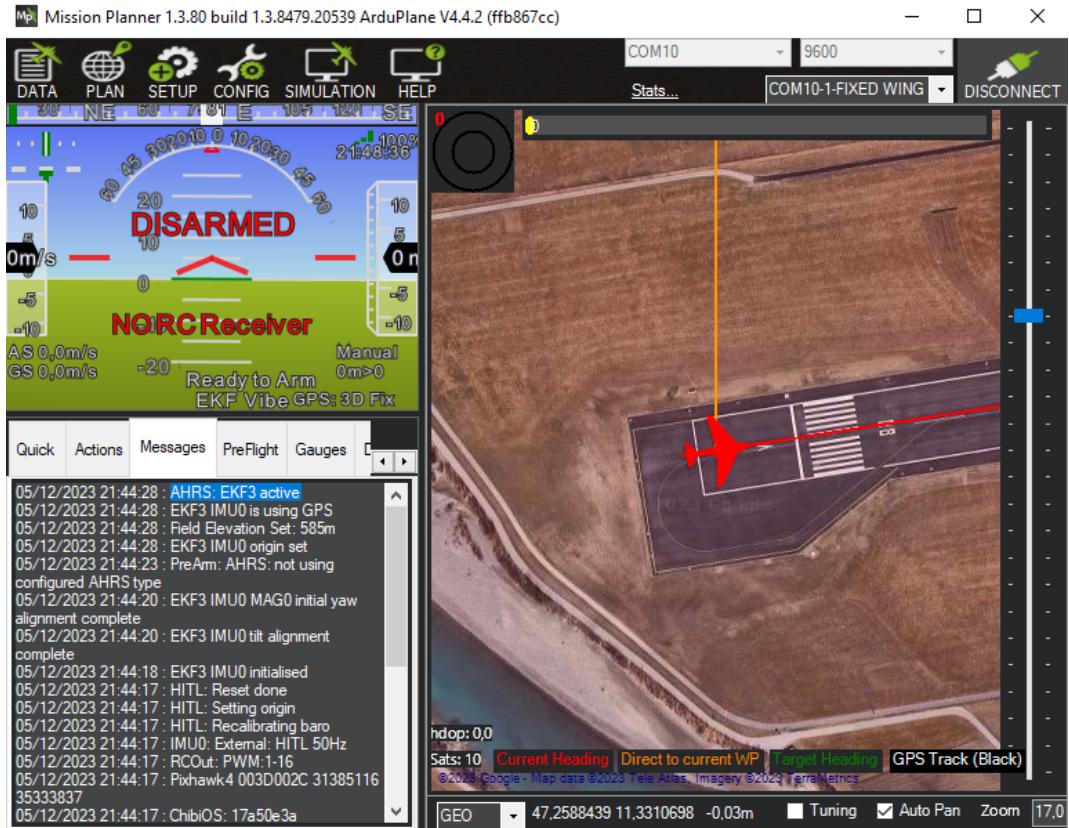


Figura 3.16: Aeronave en Mission Planner

La estructura de datos que describe la posición y actitud de la aeronave incluye varias de las mismas variables que se entregan como lecturas de sensor, las excepciones son el cuaternión `quat` que describe la orientación y la posición `origin` que indica la posición inicial de la aeronave (antes de iniciar un despegue o una misión automatizada). La importante es el cuaternión el cual se obtiene directamente de X-Plane que de utilizarse permite omitir la necesidad de estimar la variable. Esta opción se puede activar modificando el parámetro `AHRS_EKF_TYPE` en Mission Planner como se ve en la Figura 3.17 seleccionando “ExternalAHRS”, a diferencia de “Enable EKF3” que activa el filtro de Kalman del autopiloto recibiendo de X-Plane solo lecturas de sensor.

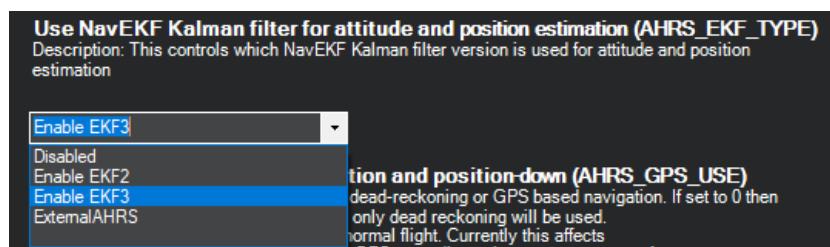


Figura 3.17: Configuración de estimación de posición y actitud en Mission Planner

Si bien las lecturas de X-Plane llegan correctamente al autopiloto, estas no coinciden con las lecturas

```

// Código de ArduPilot
// Libraries/AP_ExternalAHRS/AP_ExternalAHRS.h
struct state_t {
    HAL_Semaphore sem;
    Vector3f accel;      // m/s2
    Vector3f gyro;       // rad/s
    Quaternion quat;
    Location location; // [hor: deg, deg, ver: cm]
    Vector3f velocity; // m/s
    Location origin;   // [hor: deg, deg, ver: cm]
    bool have_quaternion;
    bool have_origin;
    bool have_location;
    bool have_velocity;
} state;

```

Código 5: Estructura de datos para posición y actitud

que producen los sensores internos del autopiloto. El Pixhawk 4 incluye acelerómetro, giroscopio, barómetro y magnetómetro los cuales son necesario desactivar para que no interfieran, el proceso para hacerlo se realiza por medio de modificación de parámetros en Mission Planner y configuración en una de las pestañas para el magnetómetro. A partir de una configuración por defecto de ArduPlane 4.4.2 los cambios a realizar son los siguientes.

- 1. Activar sistema de estimación de actitud externo.**

Configurar los parámetros EAHRS_TYPE=3, SERIALX_BAUD=115200 y SERIALX_PROTOCOL=36 de acuerdo al puerto serial correspondiente y reiniciar el autopiloto, asegurarse que en el registro aparezca habilitado el driver HITL.

- 2. Desactivar acelerómetro y giroscopio interno.**

Configurar el parámetro INS_ENABLE_MASK=1.

- 3. Desactivar magnetómetro interno.**

Configurar el parámetro COMPASS_TYPEMASK=252927, desactivando todos los drivers excepto por el del sistema externo. Reiniciar el autopiloto e ir a la pestaña “SETUP” y en la sección “Mandatory Hardware / Compass” presionar “Remove Missing”, reiniciar.

- 4. Activar GPS externo.**

Configurar el parámetro GPS_TYPE=21.

3.2.2 Ejecución de acciones de control en X-Plane

Con la telemetría funcionando es posible habilitar las opciones de ArduPilot para ejecutar misiones automatizadas o las asistencias de estabilización de vuelo, siempre y cuando se tenga acceso a las superficies de control y potencia del motor. Las acciones de control de ArduPilot son exportadas como señales PWM de servo, para ArduPlane las salidas de servo por defecto son las superficies de control elevador, timón, alerón y el acelerador del motor [29]. Internamente en el código fuente de ArduPilot los valores de salida se pueden obtener desde la clase SRV_Channels, por lo que por medio de esta se

realiza la lectura y posterior transmisión de información desde el autopiloto hacia X-Plane como se observa en el Código 6.

```
// Código de driver personalizado en ArduPilot
// Libraries/AP_ExternalAHRS/AP_ExternalAHRS_HITL.cpp
void AP_ExternalAHRS_HITL::send_rc() {
    if (reset_step > 0) {
        rc_msg.state = reset_step + 1;
    } else {
        if (AP::arming().is_armed_and_safety_off()) {
            rc_msg.state = 1;
        } else {
            rc_msg.state = 0;
        }
    }
    rc_msg.aileron = SRV_Channels::get_output_norm(SRV_Channel::k_aileron);
    rc_msg.elevator = SRV_Channels::get_output_norm(SRV_Channel::k_elevator);
    rc_msg.rudder = SRV_Channels::get_output_norm(SRV_Channel::k_rudder);
    rc_msg.throttle = SRV_Channels::get_output_norm(SRV_Channel::k_throttle);
    rc_msg.ahrs_count = ahrs_count_send;
    uart->write(reinterpret_cast<uint8_t *>(&header), sizeof(header));
    uart->write(reinterpret_cast<uint8_t *>(&rc_msg), sizeof(rc_msg));
}
```

Código 6: Lectura y transmisión de salida de servos por serial

Con los datos recibidos en el plug-in de X-Plane, se desactivan los controles convencionales del simulador que pueden ser el mouse o joysticks para las superficies de control y los motores. En su lugar se activa el control por medio de los Datarefs dedicados para esta aplicación como se observa en el Código 7.

Con estos cambios el autopiloto ya es capaz de controlar las superficies de control y el motor de la aeronave, sin embargo no es capaz de “armar”, ya que aún se reportan problemas, estos son que se encuentran descalibrados el acelerómetro y magnetómetro entre otros como se observa en la Figura 3.18. Para poder continuar es necesario calibrar estos sensores de la misma manera que se haría en una aeronave de verdad.

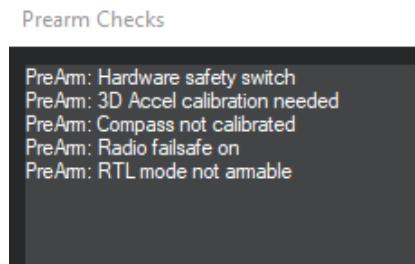


Figura 3.18: Chequeos de preambulo

```

// Código de plug-in para X-Plane
// remote.cpp
// ...
XPLMDataRef or_roll = XPLMFindDataRef("sim/operation/override/override_joystick_roll");
XPLMDataRef or_pitch = XPLMFindDataRef("sim/operation/override/override_joystick_pitch");
XPLMDataRef or_yaw = XPLMFindDataRef("sim/operation/override/override_joystick_heading");
XPLMDataRef or_throttle = XPLMFindDataRef("sim/operation/override/override_throttles");
XPLMDataRef roll = XPLMFindDataRef("sim/joystick/yoke_roll_ratio");
XPLMDataRef pitch = XPLMFindDataRef("sim/joystick/yoke_pitch_ratio");
XPLMDataRef yaw = XPLMFindDataRef("sim/joystick/yoke_heading_ratio");
XPLMDataRef throttle = XPLMFindDataRef("sim/flightmodel/engine/ENGN_thro_use");
// ...
if (override_joy) {
    XPLMSetDataf(DataRef::roll, std::clamp(servo_msg.aileron, -1.0f, 1.0f));
    XPLMSetDataf(DataRef::pitch, std::clamp(servo_msg.elevator, -1.0f, 1.0f));
    XPLMSetDataf(DataRef::yaw, std::clamp(servo_msg.rudder, -1.0f, 1.0f));
    float throttle[16];
    std::fill_n(throttle, 16, std::clamp(servo_msg.throttle, 0.0f, 1.0f));
    XPLMSetDatafv(DataRef::throttle, throttle, 0, 16);
}
// ...

```

Código 7: Datarefs para control de aeronave en plug-in de X-Plane

3.2.3 Calibración de acelerómetro y magnetómetro

Antes de que ArduPilot permita armar la aeronave y así encender el motor es necesario realizar la calibración del acelerómetro y magnetómetro. El proceso consiste en ajustar la orientación de la aeronave paso a paso siguiendo las instrucciones en Mission Planner [30].

Para el acelerómetro en la pestaña “Accel Calibration” se muestran instrucciones en pantalla como se observa en la Figura 3.19. Se agregó la función de calibración en el plug-in para X-Plane que permite orientar la aeronave como se indica en Mission Planner, por medio de botones en la interfaz gráfica como se ve en las Figuras 3.20 y 3.21.

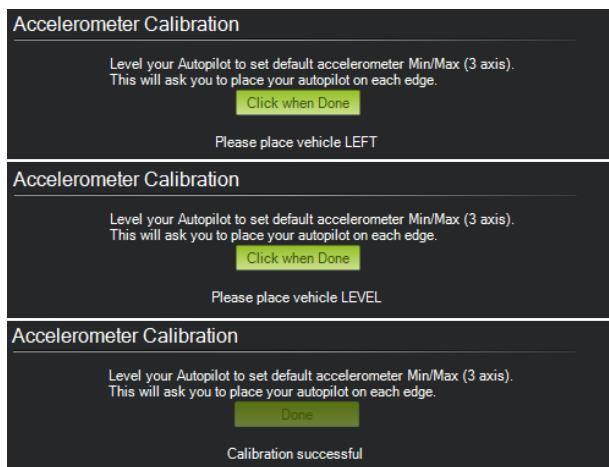


Figura 3.19: Instrucciones de calibración de acelerómetro



Figura 3.20: Aeronave en calibración nivelada

Elevar la aeronave y mantenerla quieta se logra sobreescribiendo todas las velocidades a 0, estableciendo posición y orientación específica y repetir ese proceso cada fotograma. Un problema que surge con este método es que la aceleración reportada por los Dataref es 0, siendo que debería ser un vector indicando la aceleración de gravedad. Se soluciona enviando por X-Plane el vector esperado, puesto que se conoce la orientación de la aeronave y la dirección que es hacia abajo en el marco de referencia inercial como está descrito en el Código 8.

```
// Código de plug-in para X-Plane
// telemetry.cpp
// ...
// Inertial sensor
if (!Calibration::IsEnabled()) {
    msg.ins.accel = -state.accel * GRAVITY_MSS;
} else {
    Eigen::Vector3f down = { 0, 0, -GRAVITY_MSS };
    msg.ins.accel = state.rot.conjugate() * down;
}
// ...
```

Código 8: Vector de acelerómetro enviado por telemetría

La calibración del magnetómetro es similar, se debe orientar la aeronave de la misma manera que durante la calibración del acelerómetro, pero en cada paso se rota el rumbo una vuelta completa [31]. El plug-in de X-Plane incluye un botón para empezar a rotar la aeronave de manera continua con este propósito como se observa en las Figuras 3.20 y 3.21. De esta forma se logra calibrar el magnetómetro como se ve en la Figura 3.22.

Los otros problemas observados en los chequeos de prearmado en la Figura 3.18 reportan un “Hardware safety switch” y “Radio failsafe on”, estos como su nombre indica son funciones de seguridad presentes



Figura 3.21: Aeronave en calibración inclinada a la izquierda

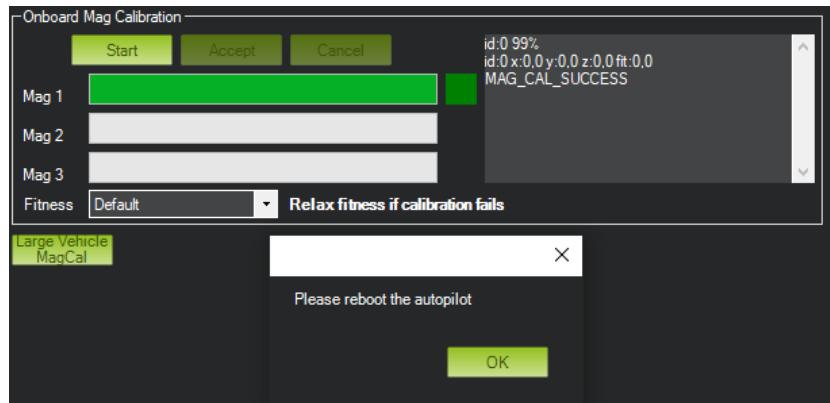


Figura 3.22: Calibración de magnetómetro completa en Mission Planner

en la aeronave y el radiocontrol. Para corregirlo se puede conectar un switch físico al autopiloto en el puerto correspondiente y activarlo, y en la radio desactivar la seguridad de acuerdo a las instrucciones de ArduPilot y la radio específica. Otra forma es establecer los parámetros `BRD_SAFETY_DEFLT=0` para desactivar el safety switch y `THR_FAILSAFE=0` para desactivar la seguridad de la radio. Con todo resuelto es posible armar la aeronave y utilizar Mission Planner tal cual se tratase de una aeronave real.

3.2.4 Pruebas en X-Plane

Conectando un receptor de radio al puerto correspondiente en el Pixhawk 4 se prueban distintas aeronaves en el simulador X-Plane 9 como se observa en la Figura 3.23. Entre las aeronaves disponibles por defecto se encuentran el Cessna 172 y un aeromodelo RC los cuales se logran volar de manera satisfactoria en los modos de vuelo estabilizado y en misión autónoma definida por waypoints en Mission

Planner como se muestra en la Figura 3.24, incluso sin ajustar ningún parámetro del controlador PID.



Figura 3.23: Vuelo con radio en X-Plane 9



Figura 3.24: Aeronave RC en misión autónoma en X-Plane 9

En su estado actual el plug-in junto con el firmware modificado tienen potencial de ser utilizados para el aprendizaje del uso general de ArduPilot y Mission Planner, más en comparación con las soluciones SITL porque esta alternativa permite conectar un receptor RC real y realizar un proceso de calibración que emula el real. La continuación de su desarrollo para este propósito consiste en simular los defectos presentes en los sensores en lugar de transmitir lecturas perfectas.

Otra prueba fue compilar el plug-in para versiones más actualizadas de X-Plane como X-Plane 11. El plug-in funciona correctamente en esta versión del simulador abriendo las mismas posibilidades que con X-Plane 9.



Figura 3.25: Plug-in funcionando en X-Plane 11

3.2.5 Otros descubrimientos y problemas durante el desarrollo

Eliminación del microcontrolador intermedio

Durante el desarrollo se descubrió que el autopiloto Pixhawk 4 abre dos puertos serial por la misma conexión USB, el primer puerto es el que se ha estado utilizando para conectarse por Mission Planner, el segundo se puede configurar para cualquier uso por medio de los parámetros y en Pixhawk 4 corresponde al puerto serial 7, si se configura este puerto como sistema de estimación de actitud externo el plug-in es capaz de conectarse y enviar información. Esto elimina la necesidad del Raspberry retransmisor lo cual es bastante conveniente.

Problemas al teletransportar la aeronave

Al seleccionar un nuevo aeropuerto, cambiar la aeronave o luego de un choque fatal X-Plane regenera la aeronave en la pista de aterrizaje independiente de donde se haya encontrado previamente. Este cambio brusco en la posición y orientación confunde al autopiloto el que comienza a reportar errores en la estimación de actitud entre otros. La solución encontrada fue reiniciar internamente las variables del filtro de Kalman integrado en el autopiloto luego de cualquier teletransporte. Realizar esto no es tan fácil, puesto que el autopiloto no incluye una función para hacerlo ni es una situación que ocurriría en la vida real. Reiniciar de manera arbitraria el filtro de Kalman a veces detenía por completo el autopiloto el cual reportaba errores internos críticos luego de ser encendido la próxima vez. Corregir esto fue un proceso de prueba de error restableciendo las variables internas una por una hasta lograr una secuencia que no detuviera el autopiloto, pero que reinicia el filtro.

3.2.6 Puesta en marcha

La primera versión del entorno de desarrollo compuesto por el plug-in y driver cumple para continuar pruebas de modificaciones posteriores que se hagan en ArduPilot y también promete ser una gran

herramienta de aprendizaje en el funcionamiento de un autopiloto y el uso de Mission Planner. Poner en marcha el entorno en un computador que no sea el de desarrollo consiste en descargar el plug-in para X-Plane desde el repositorio en GitHub [23] y descargar el firmware del autopiloto que se tenga disponible también en el repositorio correspondiente [22] para programarlo con Mission Planner.

Capítulo 4

Diseño de algoritmos de control

Por lo general los software de autopiloto cuentan con un solo tipo de algoritmo de control para cada lazo y aunque ofrezcan opciones para activar o desactivar ciertas asistencias [32], o el ajuste de parámetros internos [33] no tienen forma de cambiar por completo el algoritmo de control. Esto por lo general es una decisión de diseño, ya que los lazos PID han demostrado ser suficientes para los requerimientos de la plataforma y dar soporte para más alternativas es trabajo extra. Los controladores de vuelo realizan varias funciones internas de menor nivel para poder habilitar las asistencias y hacer de autopiloto, por ejemplo la recopilación de información de los sensores y posterior filtrado no es un problema trivial y en ArduPilot ha pasado por varias iteraciones [34].

4.1 Modificación de algoritmo existente

4.2 Implementación de nuevo algoritmo

Capítulo 5

Documentación

[todo]

Capítulo 6

Conclusión

[todo]

Bibliografía

- [1] ArduPilot. *Setup Guide for Parrot Disco*. URL: <https://ardupilot.org/plane/docs/airframe-disco.html> (visitado 26-10-2023).
- [2] David Such. *A Review of Open-Source Flight Control Systems*. 19 de mayo de 2023. URL: <http://archive.today/2023.05.19-181849/https://reefwing.medium.com/a-review-of-open-source-flight-control-systems-2fe37239c9b6> (visitado 22-10-2023).
- [3] Pixhawk. *Controller Diagrams*. URL: https://docs.px4.io/main/en/flight_stack/controller_diagrams.html (visitado 15-09-2023).
- [4] Betaflight. *PID Tuning Guide*. URL: <https://betaflight.com/docs/wiki/archive/PID-Tuning-Guide> (visitado 15-09-2023).
- [5] UZH Robotics y Perception Group. *Onboard State Dependent LQR for Agile Quadrotors (ICRA 2018)*. URL: <https://www.youtube.com/watch?v=80VsJNgNfa0> (visitado 15-09-2023).
- [6] Germán Quijada. *ESTABLECIMIENTO DE PROTOCOLO DE COMUNICACIÓN ENTRE X-PLANE Y MICROCONTROLADORES EXTERNOS EN EL SIMULADOR DE VUELO DEL LABORATORIO DE TÉCNICAS AEROESPACIALES*. URL: <https://github.com/qgerman2/pia/blob/main/pia.pdf> (visitado 26-10-2023).
- [7] Mangesh Kale, Narayani Ghatwai y Suresh Repudi. *Processor-In-Loop Simulation: Embedded Software Verification & Validation In Model Based Development*. URL: <https://www.design-reuse.com/articles/42548/embedded-software-verification-validation-in-model-based-development.html> (visitado 22-10-2023).
- [8] ArduPilot. *Choosing a Ground Station*. URL: <https://ardupilot.org/copter/docs/common-choosing-a-ground-station.html> (visitado 15-09-2023).
- [9] Germán Quijada. *Repositorio inoFS*. URL: <https://github.com/qgerman2/inoFS> (visitado 15-09-2023).
- [10] Pixhawk. *Autopilot Bus & Carriers*. URL: https://docs.px4.io/main/en/flight_controller/pixhawk_autopilot_bus.html (visitado 15-09-2023).
- [11] Paparazzi wiki. *Flight Plans*. URL: https://wiki.paparazziuav.org/wiki/Flight_Plans (visitado 22-10-2023).
- [12] Pixhawk. *Hardware in the Loop Simulation (HITL)*. URL: <https://docs.px4.io/main/en/simulation/hitl.html> (visitado 22-10-2023).
- [13] ArduPilot. *HITL Simulators*. URL: <https://ardupilot.org/dev/docs/hitl-simulators.html> (visitado 22-10-2023).

- [14] ArduPilot. *Adding Custom Attitude Controller to Copter*. URL: <https://ardupilot.org/dev/docs/copter-adding-custom-controller.html> (visitado 23-10-2023).
- [15] Pixhawk. *3DR Pixhawk 1 Flight Controller (Discontinued)*. URL: https://docs.px4.io/main/en/flight_controller/pixhawk.html (visitado 24-10-2023).
- [16] STMicroelectronics. *MEMS motion sensor: three-axis digital output gyroscope*. URL: <https://www.pololu.com/file/0J731/L3GD20H.pdf> (visitado 24-10-2023).
- [17] ArduPilot. *Telemetry / Serial Port Setup*. URL: <https://ardupilot.org/copter/docs/common-telemetry-port-setup.html> (visitado 24-10-2023).
- [18] ArduPilot. *Lua Scripts*. URL: <https://ardupilot.org/copter/docs/common-lua-scripts.html> (visitado 24-10-2023).
- [19] ArduPilot. *External AHRS Systems*. URL: <https://ardupilot.org/copter/docs/common-external-ahrs.html> (visitado 24-10-2023).
- [20] ArduPilot. *Building the code*. URL: <https://ardupilot.org/dev/docs/building-the-code.html> (visitado 26-10-2023).
- [21] ArduPilot. *Pixhawk4 Flight Controller*. URL: https://github.com/ArduPilot/ardupilot/blob/master/libraries/AP_HAL_ChibiOS/hwdef/Pixhawk4/README.md (visitado 05-12-2023).
- [22] Germán Quijada. *ArduPilot HITL Fork*. URL: <https://github.com/qgerman2/ardupilot-HITL.git> (visitado 25-10-2023).
- [23] Germán Quijada. *X-Plane HITL plug-in*. URL: <https://github.com/qgerman2/xplane-HITL.git> (visitado 25-10-2023).
- [24] Laminar Research. *Developing Plugins*. URL: <https://developer.x-plane.com/article/developing-plugins/> (visitado 05-12-2023).
- [25] Microsoft. *C/C++ for Visual Studio Code*. URL: <https://code.visualstudio.com/docs/languages/cpp> (visitado 05-12-2023).
- [26] Laminar Research. *XPLMDDataAccess*. URL: <https://developer.x-plane.com/sdk/XPLMDDataAccess/> (visitado 05-12-2023).
- [27] Laminar Research. *X-Plane Datarefs*. URL: <https://developer.x-plane.com/datarefs/> (visitado 05-12-2023).
- [28] Nathan Zimmerberg. *XYZgeomag*. URL: <https://github.com/nhz2/XYZgeomag> (visitado 05-12-2023).
- [29] ArduPilot. *Choosing Servo Functions*. URL: <https://ardupilot.org/plane/docs/servo-functions.html> (visitado 05-12-2023).
- [30] ArduPilot. *Accelerometer Calibration*. URL: <https://ardupilot.org/plane/docs/common-accelerometer-calibration.html> (visitado 06-12-2023).
- [31] ArduPilot. *Compass Calibration*. URL: <https://ardupilot.org/plane/docs/common-compass-calibration-in-mission-planner.html> (visitado 06-12-2023).
- [32] ArduPilot. *Plane Flight Modes*. URL: <https://ardupilot.org/plane/docs/flight-modes.html> (visitado 15-09-2023).
- [33] ArduPilot. *Roll, Pitch and Yaw Controller Tuning*. URL: <https://ardupilot.org/plane/docs/new-roll-and-pitch-tuning.html> (visitado 15-09-2023).

- [34] ArduPilot. *Extended Kalman Filter (EKF)*. URL: <https://ardupilot.org/copter/docs/common-apm-navigation-extended-kalman-filter-overview.html> (visitado 15-09-2023).

Apéndice A

Carta Gantt

