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

26 de octubre de 2023

Índice general

Capítulo 1

Definición del problema

1.1 Contexto

discusión sobre la utilidad de RPAs autónomos y luego hablar sobre los tipos de controladores

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 [1], 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 [2], 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 [3] los software autopiloto no ofrecen una arquitectura accesible que permita el prototipado rápido y la validación de estos sistemas.

Los PID no son optimos citar, implementar un nuevo algoritmo es peligroso en vuelo no citar

1.3 Objetivo

Implementación de una arquitectura para validación de nuevos algoritmos de control en microcontroladores para aeronaves de ala fija por medio de tecnica 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 [4], 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” [5]. 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 [6], entre otras cosas. El diseñar la arquitectura alrededor de software como ArduPilot o Pixhawk 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 Pixhawk o ArduPilot.

1.4.2 Diseño de algoritmos de control

[ARREGLAR] matlab y simulink

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

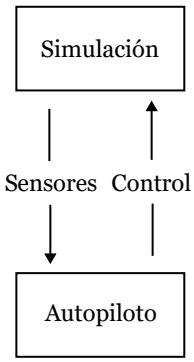


Figura 1.1: Técnica “Processor-in-Loop”

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 emularan. 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 estudiaran 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 [7] 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 [8], 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

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”.

2.1 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 2.1: 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 [9]. Revisando a un poco más detalle cada al-

ternativa 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 [10], pero no es compatible con el software de control en tierra utilizado por ArduPilot y PX4, siendo Mission Planner y QGroundControl los principales.

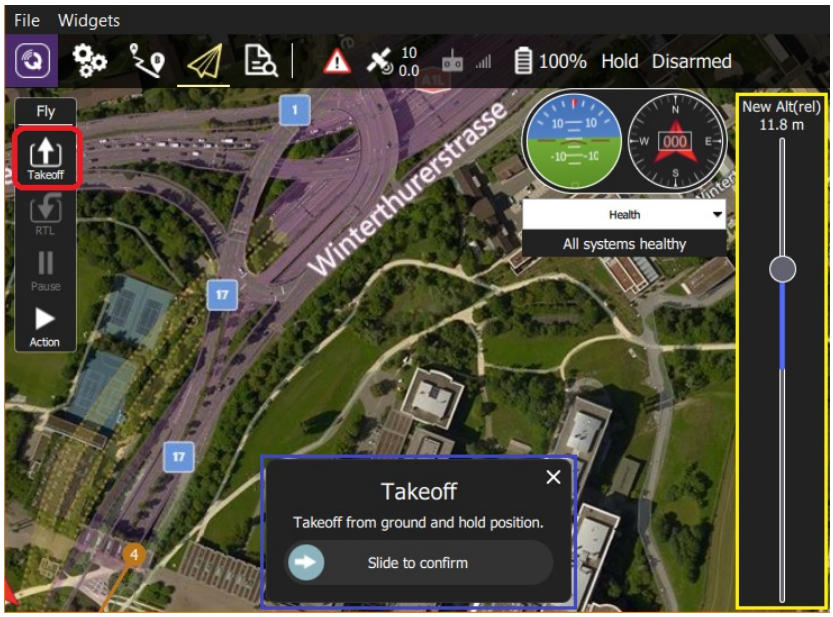


Figura 2.2: Software de estación en tierra QGroundControl

PX4 es una opción atractiva ya que ofrece soporte para trabajar con técnicas "Hardware-in-the-Loop" con simuladores incluyendo X-Plane [11]. Sin embargo, aunque ArduPilot haya deprecado esa opción [12] existe más documentación en comparación con PX4 sobre como modificar existentes e implementar nuevos algoritmos de control [13]. Por esto último se decide en ArduPilot como plataforma de desarrollo.

2.2 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 2.3 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 2.4.

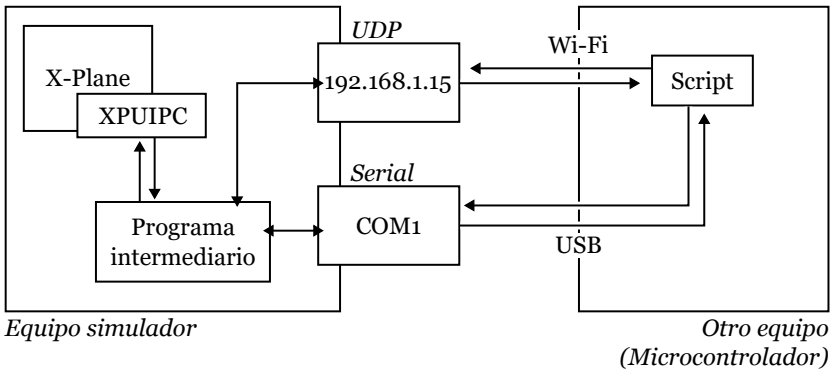


Figura 2.3: Funcionamiento inoFS

Figura 2.4: Programa intermediario de inoFS

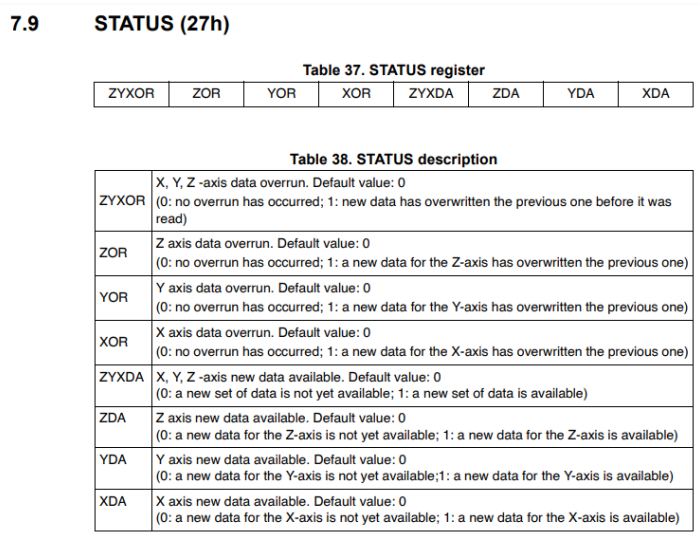
Para poder extender el protocolo de comunicación se debe ser capaz de realizar dos funciones, ingresar a ArduPilot el estado de vuelo de X-Plane como respuesta obtenida de sensores, y el ejecutar las acciones de control producidas por ArduPilot en X-Plane. Con estas funciones se puede considerar lista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

2.2.1 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 2.6. 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 [14].

- 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 [15]. Una de las ubicaciones de memoria en el giroscopio contiene información de sí las lecturas se han visto saturadas o si hay un valor nuevo como se ve en la Figura 2.5. 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 2.5: 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 [16] o para alguna aplicación personalizada que el usuario quisiera programar por medio de scripting en el lenguaje Lua [17].

La siguiente alternativa evaluada es enviar lecturas de sensores generadas con algún 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 de alguna forma programar ArduPilot.

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 del autopiloto, ya que los scripts se cargan de manera dinámica

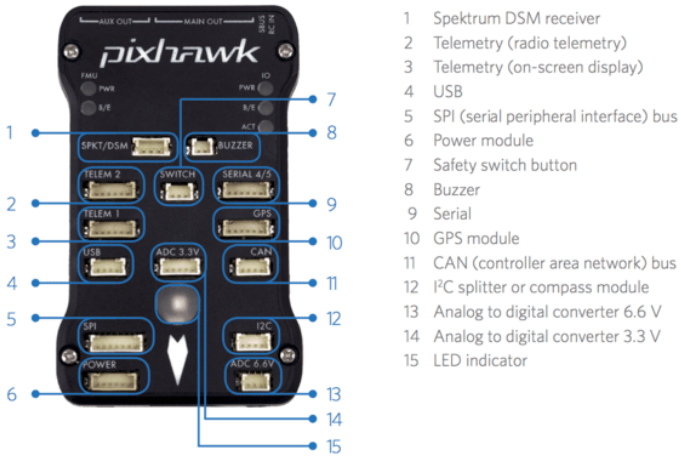


Figura 2.6: Puertos en Pixhawk

desde la tarjeta SD. 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 un tipo de sensor llamado "External Attitude Heading Reference System", que está compuesto por un IMU, giroscopio, magnetómetro y GPS, los sensores principales necesarios para el funcionamiento del autopiloto. ArduPilot da soporte para tres sistemas de estimación de actitud externos disponibles en el mercado [18].

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

Estos sistemas incluyen múltiples sensores, realizan su propia fusión y exportan los datos por puerto serial, ArduPilot tiene drivers escritos para interpretar la información entregada como lecturas para cada uno de los sensores descritos, en reemplazo de los sensores que estén incluidos en el autopiloto. Se decidió por escribir un driver para un nuevo sistema de estimación de actitud externo ficticio que actúe de la misma manera que los existentes y este reciba los datos con un protocolo propio. Esto implica que se debe modificar el código fuente de ArduPilot, en la Figura 2.7 se puede observar la clase principal `AP_ExternalAHRS` y la implementación de los sistemas "MicroStrain" y "VectorNav", en los archivos `AP_ExternalAHRS_MicroStrain5` y `AP_ExternalAHRS_VectorNav`.

Se escribió el nuevo driver para sistema externo ficticio en los archivos `AP_ExternalAHRS_HITL`, hasta este punto con la implementación más básica que lograse compilar y ser cargada en un microcontrolador de pruebas. Para el desarrollo no se utilizó el hardware disponible en el laboratorio sino que un microcontrolador personal, al que ArduPilot también ofrece soporte. Se espera probar el proyecto en uno de los controladores del laboratorio cuando sea el momento de implementar el vuelo manual con radiocontrol. Se lleva registro del desarrollo del nuevo driver en un "fork" de ArduPilot [19].

Para probar que el autopiloto esté ejecutando el código del nuevo driver, ArduPilot se debe compilar

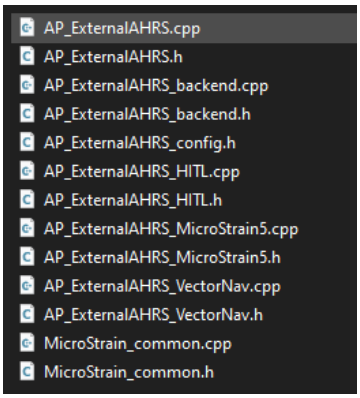


Figura 2.7: Archivos de código fuente para sistemas de estimación de actitud externos

y cargar al autopiloto según las instrucciones en su documentación [20]. Luego se conecta por serial USB a un computador y por medio de alguno de los programas de estación en tierra como Mission Planner se ajusta el parámetro que indica que se utilizara un sistema inercial externo como se puede observar en la Figura 2.8. 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 en el registro como se ve en la Figura 2.9.

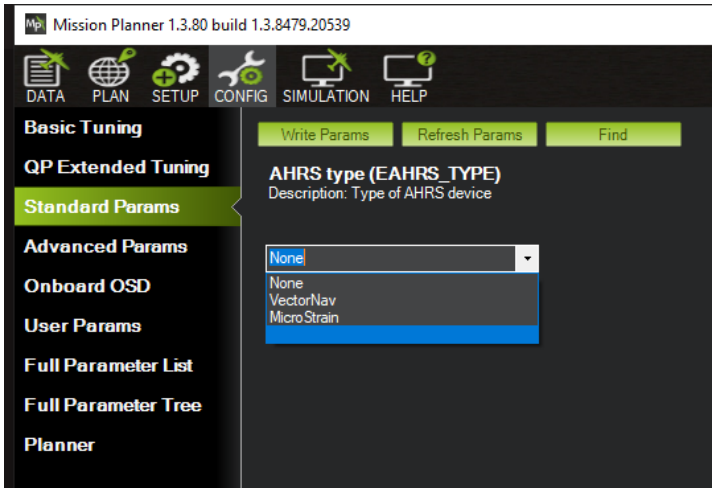


Figura 2.8: Selección de sistema inercial externo en Mission Planner

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 importante en uno de los componentes principales 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 puede leer información de la simulación no está definida, y los sensores encontrados en los autopilotos pueden entregar lecturas a una frecuencia hasta 1000 Hz para algunos como el MPU-

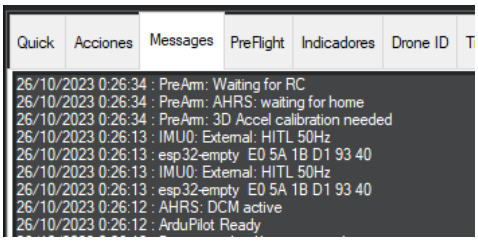


Figura 2.9: Registro de inicialización de Mission Planner

6000 mencionado anteriormente. Además de la incerteza de la velocidad de FSUIPC está la latencia producto de tener que enviar la información a través del programa intermediario. Para evitar el posible problema que esto pueda producir 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 [21].

Figura 2.10: Primera versión de plug-in “PIL” para X-Plane

El puerto serial al que el plug-in de X-Plane envía los datos no es el del autopiloto, en un computador se accede a puerto serial por USB y los sistemas inerciales externos no son USB, sino que utilizan dos pines “RX” y “TX” de entrada y salida. Por esto para completar la implementación del sistema inercial externo ficticio se utiliza otro microcontrolador que reenviara los mensajes desde X-Plane hasta el autopiloto y viceversa, como se visualiza en la Figura 2.11. En este caso se utiliza una Raspberry Pi Pico donde se configuran dos interfaces serial, una por USB y otra por los pines GPIO 4 y 5 ubicados como se ve en la Figura 2.12. El puerto serial que utilizara el autopiloto se configura en los parámetros accesibles en Mission Planner como se ve en la Figura 2.13, donde “Serial 2” hace referencia al puerto serial 2 que está en cada autopiloto asignado a ciertos pines indicados por alguna etiqueta en la carcasa, como se puede observar en el Pixhawk en la Figura 2.6 donde seria “Telem 2”.

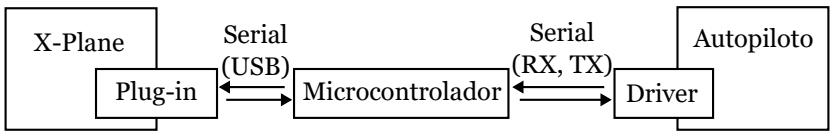


Figura 2.11: Comunicación entre X-Plane y autopiloto

Con todo configurado y conectado como en la Figura 2.14 se logra comprobar que la información de

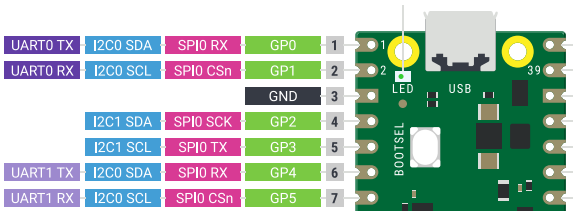


Figura 2.12: Diagrama de pines en Raspberry Pi Pico

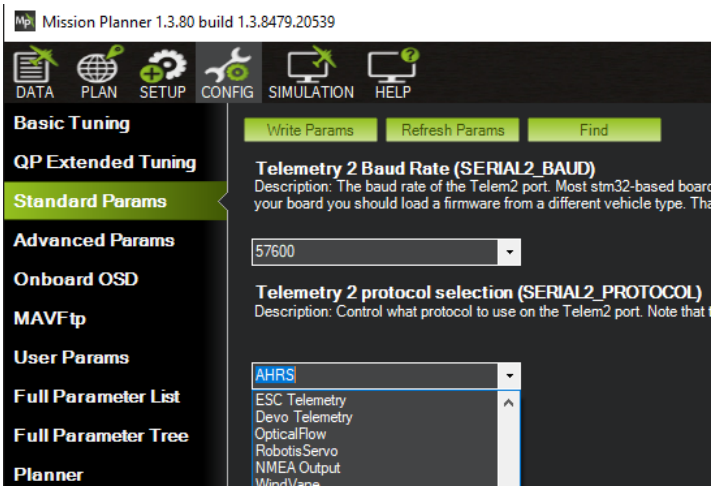


Figura 2.13: Especificación de función de puerto serial en Mission Planner

X-Plane está siendo procesada por el autopiloto, en la Figura 2.15 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 de la forma que espera ArduPilot y no utilizar valores directos desde X-Plane.

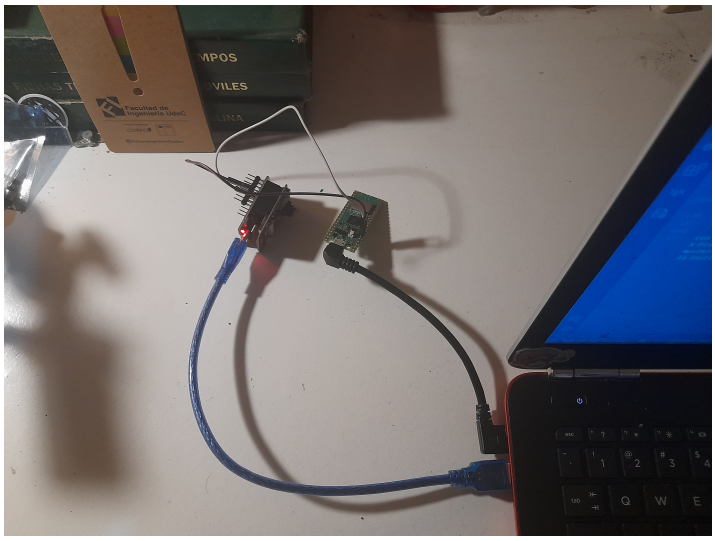


Figura 2.14: Autopiloto y microcontrolador comunicados por dos cables serial

Figura 2.15: Lecturas de inclinación y orientación en QGroundControl

Capítulo 3

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 [22], o el ajuste de parámetros internos [23] 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 [24].

3.1 Modificación de algoritmo existente

3.2 Implementación de nuevo algoritmo

Capítulo 4

Documentación

a

Capítulo 5

Conclusión

aaaa

Bibliografía

[1] Pixhawk. *Controller Diagrams*. URL: https://docs.px4.io/main/en/flight_stack/controller_diagrams.html (visitado 15-09-2023).

[2] Betaflight. *PID Tuning Guide*. URL: https://betaflight.com/docs/wiki/archive/PID-Tuning-Guide (visitado 15-09-2023).

[3] UZH Robotics y Perception Group. *Onboard State Dependent LQR for Agile Quadrotors (ICRA 2018)*. URL: https://www.youtube.com/watch?v=8OVsJNgNfa0 (visitado 15-09-2023).

[4] 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).

[5] 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).

[6] ArduPilot. *Choosing a Ground Station*. URL: https://ardupilot.org/copter/docs/common-choosing-a-ground-station.html (visitado 15-09-2023).

[7] Germán Quijada. *Repositorio inoFS*. URL: https://github.com/qgerman2/inoFS (visitado 15-09-2023).

[8] Pixhawk. *Autopilot Bus & Carriers*. URL: https://docs.px4.io/main/en/flight_controller/pixhawk_autopilot_bus.html (visitado 15-09-2023).

[9] 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).

[10] Paparazzi wiki. *Flight Plans*. URL: https://wiki.paparazziuav.org/wiki/Flight_Plans (visitado 22-10-2023).

[11] Pixhawk. *Hardware in the Loop Simulation (HITL)*. URL: https://docs.px4.io/main/en/simulation/hitl.html (visitado 22-10-2023).

[12] ArduPilot. *HITL Simulators*. URL: https://ardupilot.org/dev/docs/hitl-simulators.html (visitado 22-10-2023).

[13] ArduPilot. *Adding Custom Attitude Controller to Copter*. URL: https://ardupilot.org/dev/docs/copter-adding-custom-controller.html (visitado 23-10-2023).

[14] Pixhawk. *3DR Pixhawk 1 Flight Controller (Discontinued)*. URL: https://docs.px4.io/main/en/flight_controller/pixhawk.html (visitado 24-10-2023).

[15] STMicroelectronics. *MEMS motion sensor: three-axis digital output gyroscope*. URL: https://www.pololu.com/file/0J731/L3GD20H.pdf (visitado 24-10-2023).

[16] ArduPilot. *Telemetry / Serial Port Setup*. URL: https://ardupilot.org/copter/docs/common-telemetry-port-setup.html (visitado 24-10-2023).

[17] ArduPilot. *Lua Scripts*. URL: https://ardupilot.org/copter/docs/common-lua-scripts.html (visitado 24-10-2023).

[18] ArduPilot. *External AHRS Systems*. URL: https://ardupilot.org/copter/docs/common-external-ahrs.html (visitado 24-10-2023).

[19] Germán Quijada. *ArduPilot HITL Fork*. URL: https://github.com/qgerman2/ardupilot-HITL.git (visitado 25-10-2023).

[20] ArduPilot. *Building the code*. URL: https://ardupilot.org/dev/docs/building-the-code.html (visitado 26-10-2023).

[21] Germán Quijada. *X-Plane HITL plug-in*. URL: https://github.com/qgerman2/xplane-HITL.git (visitado 25-10-2023).

[22] ArduPilot. *Plane Flight Modes*. URL: https://ardupilot.org/plane/docs/flight-modes.html (visitado 15-09-2023).

[23] ArduPilot. *Roll, Pitch and Yaw Controller Tuning*. URL: https://ardupilot.org/plane/docs/new-roll-and-pitch-tuning.html (visitado 15-09-2023).

[24] 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