Recibido: 23/9/2020 / Aceptado: 7/11/2020
doi: https://doi.org/10.26439/interfases2020.n013.4857

Software in the Loop para la implementación de un sistema de piloto automático para aeronaves de ala fija

Lennin Paul Quiroz Villalobos
Universidad de Lima, Lima, Perú
(lquirozv@ulima.edu.pe)
https://orcid.org/0000-0003-1517-8963

Resumen

Cuando se desea desarrollar y probar *softwares* para sistemas complejos, como aviones, transbordadores espaciales, satélites, automóviles o plantas nucleares, uno de los mayores inconvenientes es la gran complejidad, riesgos y costos que implica realizar pruebas sobre el sistema real. Software in the Loop (SITL) permite testear algoritmos, códigos fuente o estrategias de control para sistemas complejos dentro de una simulación que contiene el modelo matemático del sistema físico real. El presente trabajo propone el uso de la plataforma Software in the Loop con el fin de desarrollar un sistema de piloto automático para una aeronave de ala fija. Se muestra la arquitectura de la aplicación implementada, el proceso de diseño de *software*, los protocolos utilizados, las estrategias de control, técnicas de programación, los resultados obtenidos y las conclusiones.

PALABRAS CLAVE: SITL / piloto automático / aeronaves / ingeniería de *software* / Simulink / PID

Abstract

SOFTWARE IN THE LOOP FOR THE IMPLEMENTATION OF AN AUTOPILOT SYSTEM FOR FIXED-WING AIRCRAFTS

When you want to develop and test software products for complex systems such as airplanes, space shuttles, satellites, automobiles or nuclear plants, one of the biggest drawbacks is the availability of the physical system, due to the great complexity, risks and costs involved in performing tests on the real system. Software in the Loop (SITL) allows us to test algorithms, source code or control strategies for complex systems within a simulation that contains the mathematical model of the real physical system. This research work proposes the use of a Software in the Loop platform for the development of an autopilot system for a fixed-wing aircraft. The architecture of the implemented application, the software design process, the protocols used, the control strategies, the programming techniques, the results obtained and the conclusions are shown herein.

KEYWORDS: SITL / autopilot / aircrafts / software engineering / Simulink / PID

1. INTRODUCCIÓN

La ingeniería de *software* es una disciplina que tiene como objetivo proporcionar teorías, métodos y herramientas para el desarrollo de *software* de calidad. El ciclo de desarrollo de un producto *software* comprende requisitos, diseño, implementación, pruebas y mantenimiento. Cuando se trata de sistemas complejos el desarrollo debe ser iterativo e incremental, agregando nuevas funcionalidades en cada ciclo. Antes de aceptar o dar por válido un nuevo módulo, este debe pasar por un proceso de verificación y validación. En la etapa de verificación se comprueba si se está cumpliendo con los requisitos establecidos; luego, en la etapa de validación se comprueba si el *software* hace realmente lo que se espera. La mejor manera de validar un *software* es sobre el sistema físico sobre el cual operará; sin embargo, cuando se trata de sistemas complejos, no es posible realizar el proceso de validación parcial o total sobre el sistema real, debido al costo, riesgos o tiempo empleado. En este tipo de proyectos se utiliza una plataforma Software in the Loop, la cual permite interactuar con el modelo matemático del sistema físico en un entorno virtual para probar algoritmos, código fuente, estrategias de control, observar sus efectos sobre el modelo, así como detectar y corregir errores a un costo mínimo, antes de que se propaguen al resto del sistema.

El presente trabajo de investigación, propone el desarrollo de un sistema de piloto automático para una aeronave de ala fija, usando una plataforma Software in the Loop. Inicialmente, se muestra el ambiente de desarrollo implementado y sus componentes, luego, los conceptos correspondientes a la dinámica de vuelo de la aeronave, los parámetros críticos de operación, las estrategias de control propuestas, las pruebas realizadas y las conclusiones.

2. ARQUITECTURA DE LA APLICACIÓN

A continuación, se muestra la arquitectura de la plataforma SITL implementada:

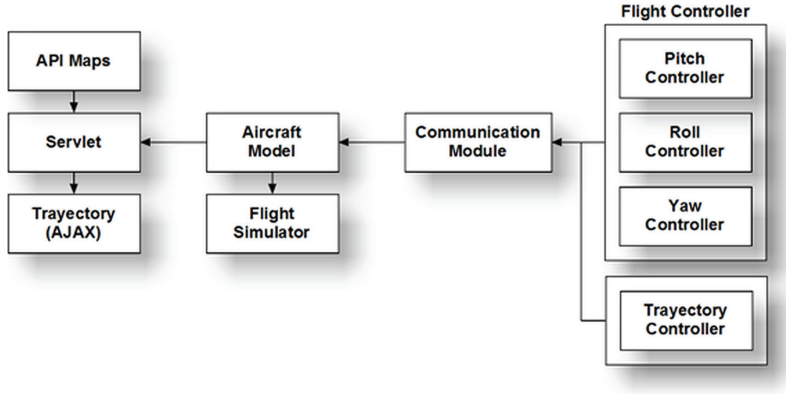

Figura 1. Arquitectura SITL
Elaboración propia

Tal como se muestra en la figura 1, los módulos de control de vuelo (Pitch, Roll y Yaw), así como el módulo de ajuste de trayectoria se comunican con los módulos centrales, conteniendo el modelo de la aeronave de ala fija y la salida del simulador, a través del módulo de comunicaciones, usando el protocolo UDP (User Datagram Protocol). Así mismo, el bloque *servlet*, a partir de las coordenadas latitud y longitud generadas por el simulador, utiliza la API de Microsoft Maps, para mostrar en tiempo real la posición actual y la trayectoria de la aeronave utilizando la tecnología AJAX (Asynchronous JavaScript and XML).

3. DISEÑO DE LA PLATAFORMA SITL

El sistema SITL implementado es un conjunto de componentes conectados entre sí a través de interfaces y protocolos. Un componente es una parte modular de la arquitectura física, cuya principal característica es que oculta su implementación tras un conjunto de interfaces externas, esto permite que las interfaces requeridas y ofertadas puedan ser reemplazables por nuevas versiones, implementadas en diferentes lenguajes o accedidas directamente para propósitos de *testing*.

Durante el proceso de implementación, tener una vista arquitectónica de alto nivel de los componentes del sistema fue fundamental para poder programar y priorizar las tareas de implementación, así como para poder determinar en qué componentes del sistema sería necesario realizar pruebas unitarias y pruebas de integración. Para lograr este objetivo el Diagrama de Componentes del sistema fue necesario.

La figura 2 muestra el diagrama de componentes del sistema implementado. Los módulos del componente Controller fueron desarrollados en Simulink (Matlab R2013b), los módulos del componente Simulator fueron acondicionados usando el simulador FlightGear (Ver 2.10) y los módulos del componente Maps fueron desarrollados en Java (Ver. 8.0.65). Se utilizó el sistema operativo Windows 7.

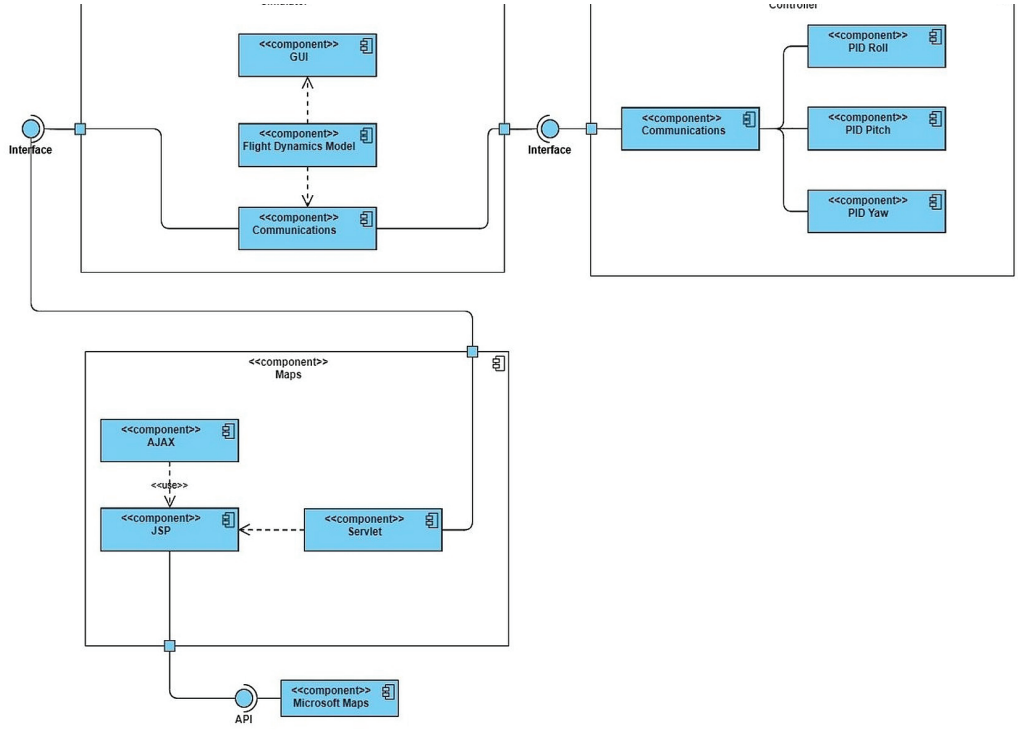

Figura 2. Diagrama de componentes de la plataforma SITL

Elaboración propia

Así mismo, para poder modelar la manera en que los componentes interactúan entre sí a través del tiempo para completar la funcionalidad de la plataforma SITL, el Diagrama de Secuencia del sistema también fue necesario. La principal ventaja del diagrama de secuencia es que nos permite representar la lógica de operación entre los diferentes módulos que componen el sistema en orden cronológico.

Tal como se muestra en la figura 3, el diagrama de secuencia de la plataforma SITL nos permite describir cómo los elementos que componen el sistema intercambian mensajes para llevar a cabo las acciones de control y el trazado de la trayectoria sobre el mapa.

Las líneas de vida (líneas verticales) para los elementos del simulador y los elementos del sistema de control, se van intercalando en un bucle infinito para detectar las posiciones relativas de las superficies de control y aplicar las acciones correctivas necesarias para mantener la aeronave en vuelo.

Así mismo, los parámetros latitud y longitud generados por el simulador son informados periódicamente a los elementos encargados de describir la trayectoria de la aeronave. Utilizando el modelo MVC (Modelo-Vista-Controlador) el *servlet* Controller actualiza permanentemente los mapas desde la API de Microsoft Maps de acuerdo con el desplazamiento de la aeronave; luego, la posición exacta dentro del mapa debe ser especificada mediante un marcador, sin perder el histórico de posiciones previas, para lo cual se utiliza la tecnología AJAX.

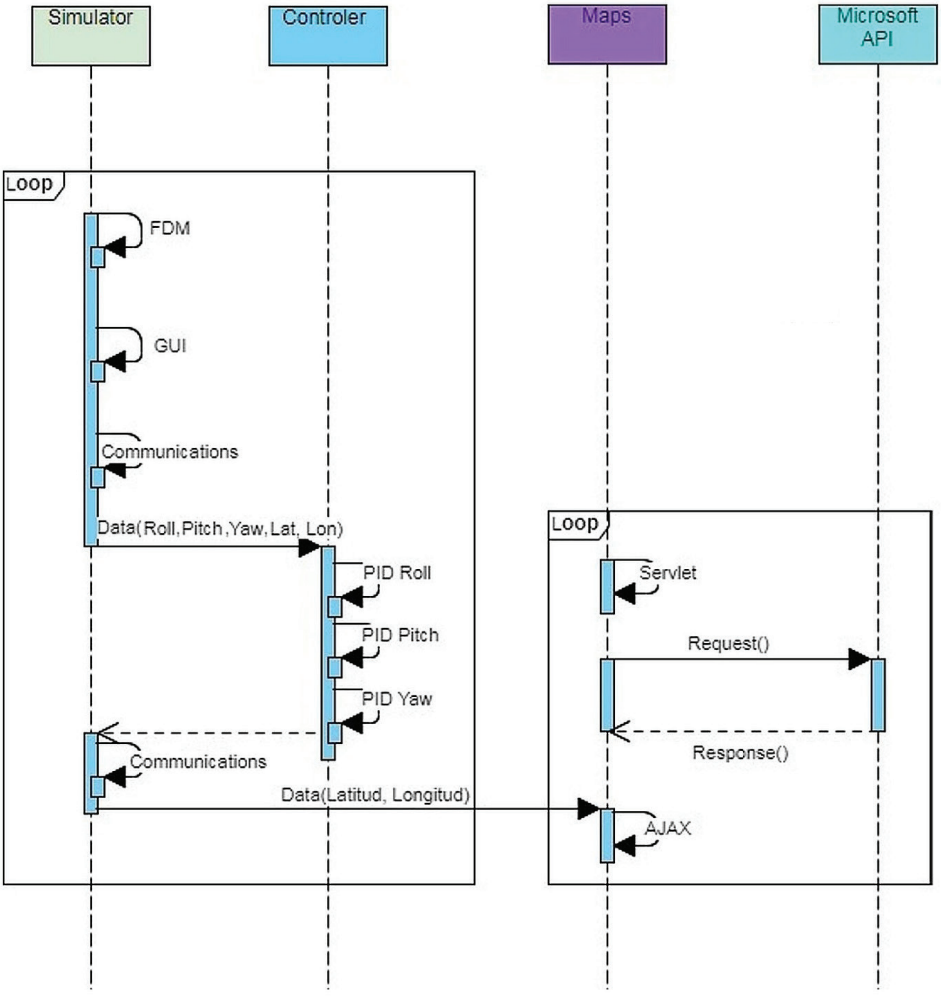

Figura 3. Diagrama de secuencia para la plataforma SITL

Elaboración propia

Finalmente, el ciclo de vida de desarrollo del sistema ha sido iterativo e incremental.

En un ciclo de vida iterativo el desarrollo de un componente empieza desde la definición de requisitos, planificación de tareas, ejecución y evaluación; luego, se pasa a otro componente donde se repite el mismo proceso. El ciclo de vida iterativo se aplicó para el desarrollo de los módulos controladores de Pitch, Roll y Yaw.

En un ciclo de vida incremental se busca el crecimiento progresivo de la funcionalidad del sistema. El procedimiento consiste en dividir el proyecto en módulos bien diferenciados, cada uno con una función específica. Cada módulo se va construyendo de acuerdo a su prioridad dentro del proyecto. Un aspecto sumamente importante en el ciclo de vida incremental son las pruebas de integración entre módulos para verificar que dos módulos que funcionan correctamente de forma separada lo sigan haciendo al integrarse. Así mismo, podría suceder que un módulo que antes funcionaba correctamente ahora se vea afectado por un nuevo módulo, por lo cual las pruebas de regresión también son sumamente importantes. El orden de prioridad en que fueron construidos los módulos de la plataforma SITL son: Simulator, Controller, Maps.

4. CONSTRUCCIÓN DE LA PLATAFORMA SITL

4.1 Dinámica de vuelo

Para un mejor entendimiento del sistema implementado, a continuación, se presenta una versión simplificada de la dinámica de vuelo de un avión. El comportamiento (*attitud*) de un avión en vuelo está determinado por sus ángulos Roll, Pitch y Yaw (Koks, 2008), tal como se muestra en la figura 4:

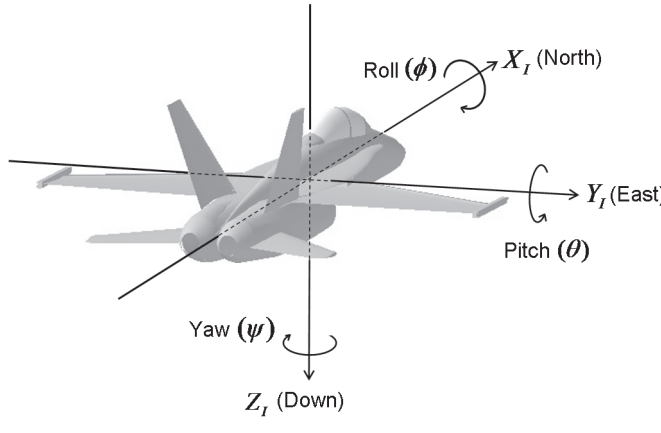

Figura 4. Ángulos de vuelo

Fuente: http://www.chrobotics.com/library/understanding-euler-angles

El Roll (alabeo), se considera como una rotación angular respecto al eje x. El Pitch (cabeceo), se considera como una rotación angular respecto al eje y. El Yaw (guiñada), se considera como una rotación angular respecto al eje z. Para una aeronave en vuelo, al cumplirse la segunda Ley de Newton, F=m.a, será un sistema de referencia no inercial y su orientación estará determinada por los valores de los ángulos Roll, Pitch y Yaw en cada instante de vuelo (Peet, 2010).

Sin embargo, para poder determinar las ecuaciones que gobiernan las leyes de vuelo del sistema, es necesario expresar las fuerzas que actúan en coordenadas de un sistema de referencia inercial. Para cumplir este objetivo, es necesario definir las matrices de rotación respecto a cada uno de los ángulos Roll (ecuación 1), Pitch (ecuación 2) y Yaw (ecuación 3), (Slabaugh, 2017), tal como se muestra:

$$R(x,\phi)=\begin{bmatrix}x_1\\y_1\\z_1\end{bmatrix}=\begin{bmatrix}1&0&0\\0&\cos\phi&-sen\,\phi\\0&sen\,\phi&\cos\phi\end{bmatrix}\begin{bmatrix}x_B\\y_B\\z_B\end{bmatrix} \quad (1)$$

$$R(y,\theta)=\begin{bmatrix}x_2\\y_2\\z_2\end{bmatrix}=\begin{bmatrix}\cos\theta&0&sen\,\theta\\0&1&0\\-sen\,\theta&0&\cos\theta\end{bmatrix}\begin{bmatrix}x_1\\y_1\\z_1\end{bmatrix} \quad (2)$$

$$R(z,\psi)=\begin{bmatrix}x_B\\y_B\\z_B\end{bmatrix}=\begin{bmatrix}\cos\psi&-sen\psi&0\\sen\psi&\cos\psi&0\\0&0&1\end{bmatrix}\begin{bmatrix}x_2\\y_2\\z_2\end{bmatrix} \quad (3)$$

Luego, la matriz de rotación completa se obtiene mediante tres rotaciones sucesivas de los ángulos (Jia, 2020), tal como se muestra:

$$R(\phi,\theta,\psi)=\begin{bmatrix}\cos\theta\cos\psi&\cos\psi\,sen\theta sen\phi-sen\psi\cos\phi&\cos\psi\,sen\theta\cos\phi+sen\psi\,sen\,\phi\\sen\psi\cos\theta&sen\psi\,sen\theta sen\phi+\cos\psi\cos\phi&sen\psi\,sen\theta\cos\phi-sen\phi\cos\psi\\-sen\,\theta&\cos\theta\,sen\,\phi&\cos\theta\cos\phi\end{bmatrix} \quad (4)$$

La ecuación 4, obtenida previamente, servirá para expresar la orientación, posición y movimiento de la aeronave respecto al sistema de referencia inercial.

4.2 Estrategia de control

Un PID (Proporcional Integral Derivativo) es un tipo de control cuya principal característica es generar una señal de corrección para el error que se obtiene al comparar el valor actual de un parámetro y el valor deseado o valor objetivo de dicho parámetro. Esta señal

de corrección se aplicará a los mecanismos actuadores del sistema; el nuevo valor será informado por los sensores y comparado nuevamente, funcionando así de forma realimentada. En un controlador PID la acción proporcional, determina la reacción o magnitud de la respuesta al error, la acción integral genera una señal de corrección proporcional a la integral del error y la acción derivativa determina la reacción con relación al tiempo en que se produce. En algunos casos, en función de la aplicación específica del controlador, alguno de los parámetros del PID podría ser cero (Astrom, 2002; Cova, 2005).

La figura 5 muestra cómo interactúan cada una de las tres acciones de control en un controlador PID.

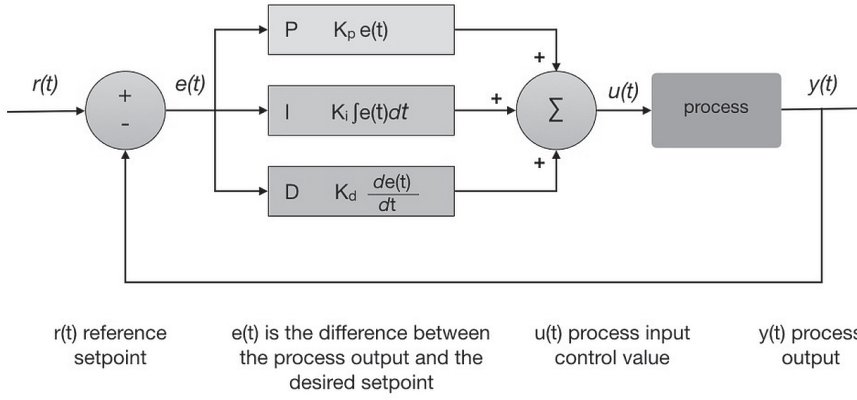

Figura 5. Control PID

Fuente: https://mjwhite8119.github.io/Robots/pid-control

Para la implementación del sistema de control de piloto automático, se han utilizado tres controladores PID, tal como se muestra en la figura 5, uno para cada grado de libertad de la aeronave. Los valores óptimos para los parámetros P, I, D de cada controlador se obtuvieron experimentalmente.

4.3 Flight Dinamics Models (FDM)

Un FDM contiene las ecuaciones matemáticas que son usadas para calcular las fuerzas físicas que actúan sobre la simulación de vuelo de una aeronave. Para la implementación del presente trabajo se usó el simulador FlightGear. Los FDM que incorpora Flight Gear son YASim, JSBSim y UIUC.

FlightGear (https://www.flightgear.org/) es un simulador *open source* y multiplataforma. En la instalación por defecto incluye muchos modelos de aeronaves; sin embargo, modelos adicionales pueden ser agregados desde la web de soporte o creados por el

mismo usuario en aplicaciones 3D externas y cargadas mediante un archivo XML que especifica las características de la aeronave. FlightGear está basado en OpenGL.

4.4 Sistema de control

El sistema de control implementado se muestra en la figura 6. Este está desarrollado utilizando la herramienta Simulink de Matlab, la cual utiliza los bloques de la librería *net_ctrl* packet para realizar la comunicación UDP con el simulador, y los bloques PID del *Control System Toolbox* para la implementación de los controladores. Los valores de latitud y longitud del destino deseado se setean manualmente antes de iniciar cada misión.

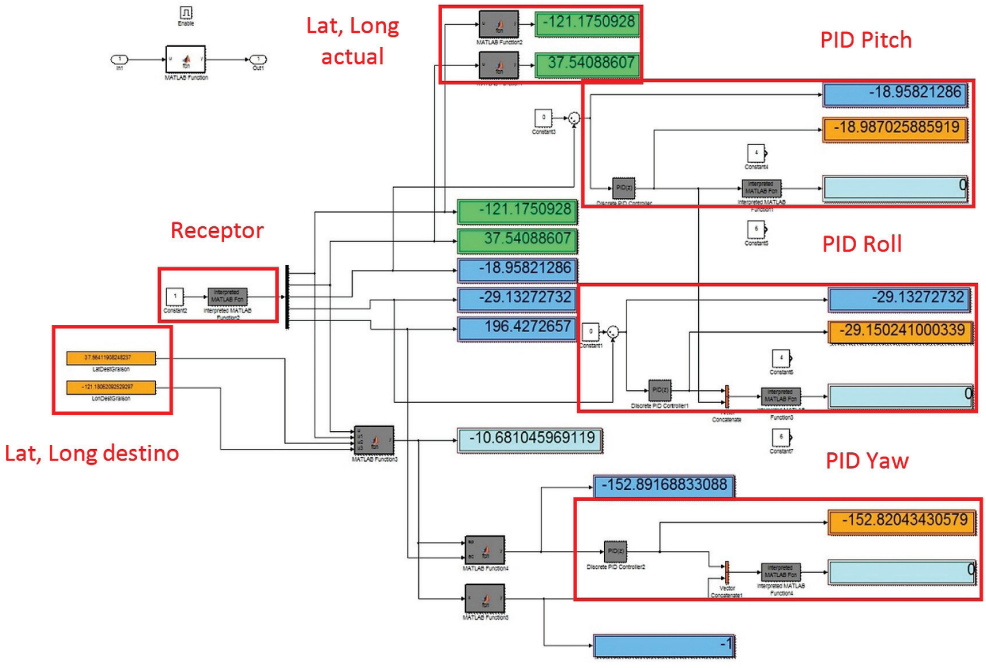

Figura 6. Bloques del sistema de control

Elaboración propia

4.5 Cálculo de distancia

Para que la aeronave pueda ir desde un punto origen a un punto destino, es necesario proporcionar las coordenadas Latitud y Longitud de la ciudad hacia donde queremos ir (destino). Las coordenadas Latitud y Longitud del aeropuerto donde se encuentra actualmente la aeronave se tomarán como coordenadas de origen. Para hacer el cálculo de la distancia entre origen y destino, se debe tener en cuenta que, debido a la curvatura de la Tierra, la distancia más corta entre dos puntos es un arco, tal como se muestra en la

figura 7. Por lo tanto, para realizar este cálculo, el uso de la fórmula Haversine fue necesario (Hartanto, Furqan, Putera, Siahaan y Fitriani, 2017).

Figura 7. Distancia más corta entre dos puntos

Elaboración propia

La longitud del sector circular será: d=θ.r, donde θ es el ángulo central formado por los puntos de origen, destino y r será el radio de la Tierra. La fórmula Haversine hav θ viene dada por: (Surowski, 2011).

$$\text{hav}(\theta) = \text{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1)\cos(\varphi_2)\,\text{hav}(\lambda_2 - \lambda_1) \quad (5)$$

Donde

$$\varphi_1, \varphi_2 : \text{latitud del punto 1 y latitud del punto 2}$$
$$\lambda_1, \lambda_2 : \text{longitud del punto 1 y longitud del punto 2}$$

Luego, la función Haversine de un ángulo θ es:

$$\text{hav}(\theta) = \sin^2\left(\frac{\theta}{2}\right) = \frac{1-\cos(\theta)}{2} \quad (6)$$

Para obtener la distancia d, aplicamos Haversine inverso al ángulo central θ

$$d = r\,hav^{-1}(h) = 2r\arcsin(\sqrt{h}) \tag{7}$$

Donde h=hav *(θ).* Reemplazando, tenemos la distancia d a partir de las coordenadas de Latitud y Longitud de los puntos de origen y destino:

$$\begin{aligned} d &= 2r\arcsin\left(\sqrt{hav(\varphi_2-\varphi_1)+\cos(\varphi_1)\cos(\varphi_2)hav(\lambda_2-\lambda_1)}\right) \\ &= 2r\arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2-\varphi_1}{2}\right)+\cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2-\lambda_1}{2}\right)}\right) \end{aligned} \tag{8}$$

La ecuación 8 muestra la expresión matemática necesaria para calcular la distancia entre dos puntos sobre la Tierra a partir de sus coordenadas de latitud y longitud.

4.6 Cálculo de orientación

Para calcular la orientación correcta, el sistema de control se encargará de ir calculado el vector direccional necesario para poder ajustar el rumbo de la aeronave hacia el destino. Sin embargo, el sistema de control requiere convertir las coordenadas Latitud y Longitud a señales de corrección que se puedan aplicar a los ángulos de la aeronave (Yaw). Para lograr este objetivo, el cálculo del ángulo Bearing también fue necesario.

El Bearing es el ángulo que se forma entre un meridiano y la línea que conecta hacia la posición actual del objeto. Así mismo, el ángulo Heading (Yaw) es el ángulo actual de la aeronave respecto al sistema de referencia no inercial; por lo tanto, para tener la certeza de que la aeronave está en el rumbo correcto, el ángulo Bearing y el Heading deben coincidir.

El cálculo del ángulo Bearing se muestra en la ecuación 9:

$$B = \text{atan}\left((\text{Cos}\,\theta_b * \text{Sin}(\Delta L)), (\text{Cos}\,\theta_a * \text{Sin}\,\theta_b - \text{Sin}\,\theta_a * \text{Cos}\,\theta_b * \text{Cos}\,\Delta L)\right) \tag{9}$$

Donde:

θ_a, θ_b : latitud del punto A y latitud del punto B

ΔL : diferencia de longitud entre el punto A y el punto B

El ángulo Bearing se calcula para cada punto donde se encuentre el avión; luego, el sistema de control se encarga de ir ajustando el Yaw para que los ángulos Bearing y Heading coincidan.

4.7 Mapas

Para la implementación de los mapas en tiempo real, se usó la API de Microsoft Maps mediante una función Java Script. Sin embargo, debido a que la información de Latitud y Longitud actual de la aeronave debe ser extraída desde el simulador, la implementación de un *servlet* en Java fue necesario. La aplicación web implementada está basada en el patrón MVC (Modelo-Vista-Controlador) donde el controlador está implementado en Java, la vista es un JSP (Java Server Pages) y el modelo es generado a través de un archivo CSV.

Así mismo, para la implementación de la gráfica de la trayectoria de la aeronave, manteniendo los registros previos de ubicación, el uso de la tecnología AJAX sobre JSP fue necesario. A continuación, en la figura 8, se muestra la función Ajax que se utilizó para graficar la trayectoria de la aeronave.

```
function Ajax()
{

        if( window.XMLHttpRequest )
                ajax = new XMLHttpRequest();
        else
                ajax = new ActiveXObject("Microsoft.XMLHTTP");

        ajax.open( "GET", "Servidor?"+0, true );
        ajax.send( "" );

        ajax.onreadystatechange = funcionCallback;

        setTimeout("parametros()",500);
}
```

Figura 8. Función AJAX implementada

Elaboración propia

5. PRUEBAS REALIZADAS Y RESULTADOS

Para las pruebas se utilizó la aeronave Piper J3 Cub, la cual se caracteriza por ser simple y ligera. Por esta razón, muchas veces es utilizada como aeronave de entrenamiento. La figura 9 muestra vistas desde diferentes ángulos de esta aeronave.

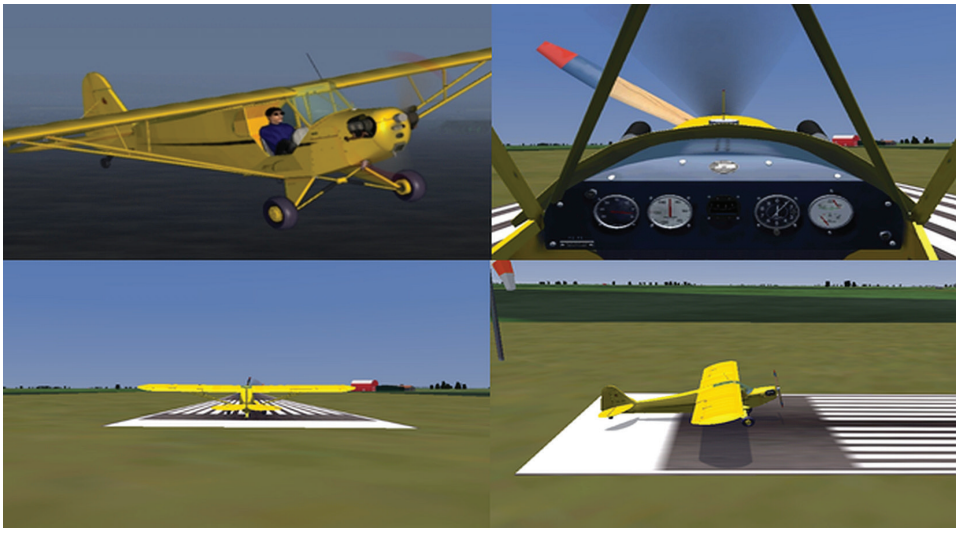

Figura 9. Vistas de la aeronave Piper J3 Cub

Elaboración propia

La figura 10 muestra el mapa de la trayectoria seguida por la aeronave Piper J3 Cub para ir desde el aeropuerto de Westley hacia la ciudad de Grayson (USA).

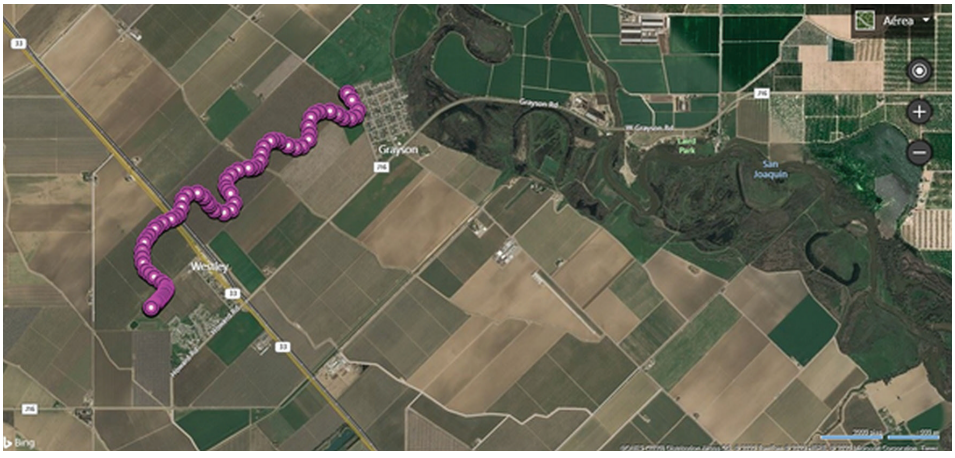

Figura 10. Trayectoria de la aeronave

Elaboración propia

Así mismo, utilizando la herramienta Scope, se graficaron los valores obtenidos para los ángulos Roll, Pitch, Yaw para esta trayectoria, tal como se muestra en la figura 11:

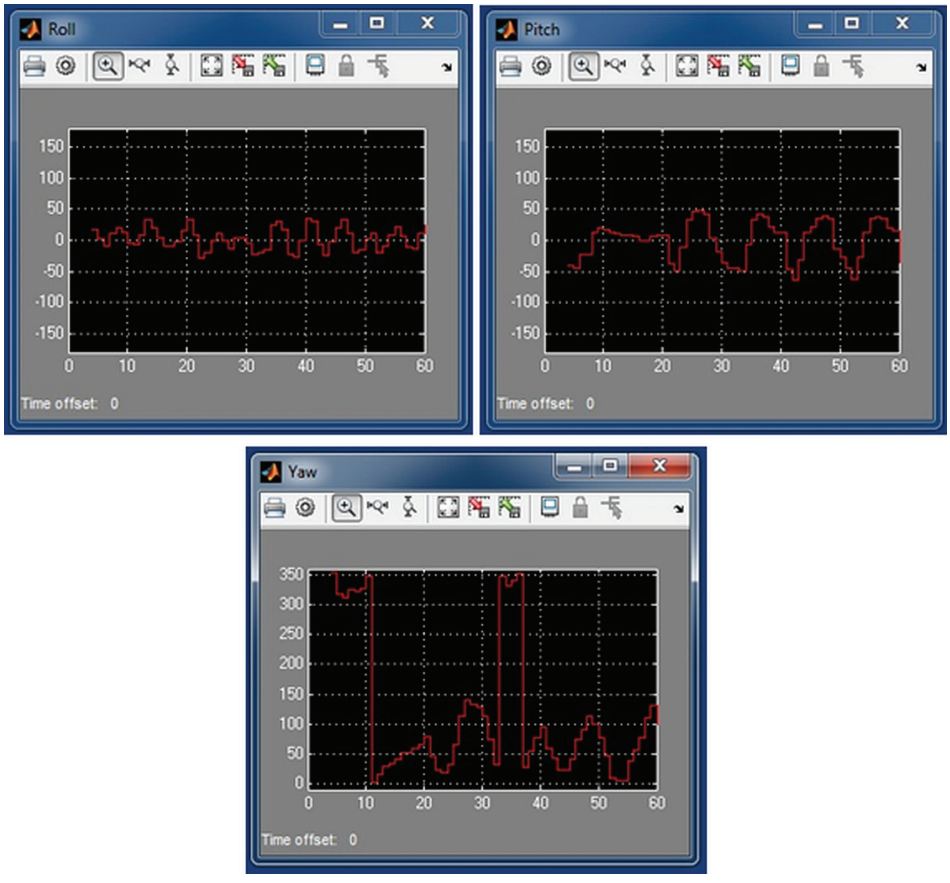

Figura 11. Gráficas de Roll, Pitch, Yaw

Elaboración propia

Tal como se observa en la figura 11, los ángulos Roll y Pitch han tenido variaciones menores, durante toda la trayectoria; sin embargo, el ángulo Yaw tiene variaciones drásticas cada cierto tiempo, esto se debe a la acción de control que se ejerce sobre este ángulo para orientar la aeronave hacia la dirección correcta.

Las pruebas realizadas, también están disponibles en video en el siguiente link: https://www.youtube.com/watch?v=TgjSFvtcUFE

6. CONCLUSIONES

- Software in the Loop ha demostrado ser una técnica válida para el diseño de sistemas complejos como *software* de aviónica, donde por limitaciones de costo, tiempo y seguridad no es posible realizar pruebas directamente sobre el sistema físico real.
- Los diagramas de componentes y secuencia son fundamentales en el proceso de diseño de *software* de sistemas complejos ya que proporcionan una vista de alto nivel de los módulos y componentes del sistema. Así mismo, nos permiten conocer cómo interactúan entre sí los diversos componentes a través del tiempo para completar la funcionalidad deseada.
- El ciclo de vida iterativo e incremental, se ajusta adecuadamente a sistemas con muchos módulos interactuando entre sí, ya que nos permite aplicar el proceso de desarrollo a cada componente individual y de esta manera ir agregando funcionalidades de forma gradual. Sin embargo, para que cada nuevo módulo sea validado será necesario aplicar pruebas de integración y pruebas de regresión con el resto del sistema, generando al final de este proceso una nueva versión.
- Para especificar la dinámica de vuelo de una aeronave de ala fija se necesita un sistema de referencia relativo a la misma aeronave, donde normalmente se miden los ángulos Roll, Pitch y Yaw, así como un sistema de referencia relativo a tierra de tal manera que se pueda determinar la posición de la aeronave en cualquier punto en términos de latitud y longitud.
- A pesar de que una aeronave de ala fija se considera un sistema no lineal, la estrategia de control PID brinda una aproximación aceptable para controlar la dinámica de vuelo de la aeronave durante las etapas de despegue, vuelo crucero y aterrizaje.
- El Modelo Dinámico de Vuelo (FDM) de una aeronave es el componente fundamental en un simulador, ya que al controlar la exactitud con que las fuerzas involucradas son representadas, determina la fiabilidad que tendrá el sistema cuando sea aplicado al sistema físico real.
- Simulink ofrece una plataforma confiable para comunicaciones en tiempo real con aplicaciones externas basadas en el Protocolo de Datagramas de Usuario.
- La fórmula Haversiana es una técnica precisa para calcular el arco de distancia entre dos puntos de la tierra. Así mismo, para determinar la correcta orientación de una aeronave hacia un destino, el cálculo del ángulo Bearing será necesario.

REFERENCIAS

Aboelela, M. A. S., Ahmed, M. F., y Dorrah, H. T. (2012). Design of aerospace control systems using fractional PID controller. *Journal of Advanced Research*, *3*(3), 225–232. http://doi.org/10.1016/j.jare.2011.07.003

Akyürek, Ş., Kürkçü, B., Kaynak, Ü., y Kasnakoğlu, C. (2016). Control Loss Recovery Autopilot Design for Fixed-Wing Aircraft. *IFAC-PapersOnLine*, *49*(9), 117-123. http://doi.org/10.1016/j.ifacol.2016.07.509

Astrom, K. J. (2002). PID Control. *Control System Design*, 216–251. Recuperado de https://www.cds.caltech.edu/~murray/courses/cds101/fa02/caltech/astrom-ch6.pdf

Attya, S. M., y Abdulla, A. I. (2018). PID Controller Design and Simulation for Aircraft Roll Control Based on Evolutionary Technique Using MATLAB, JET, *8*(3), 5-9. Recuperado de http://doi.org/10.17605/OSF.IO/UT8FB

Bansal, H. O. (2009). Tuning of PID Controllers using Simulink. *International Journal of Mathematical Modeling, Simulations and Applications*, *2*(3), 337–344. Recuperado de https://www.researchgate.net/publication/268802558_Tuning_of_PID_Controllers_using_Simulink

Coopmans, C., Podhradský, M., y Hoffer, N. V. (2016). Software- and hardware-in-the-loop verification of flight dynamics model and flight control simulation of a fixed-wing unmanned aerial vehicle. *2015 Workshop on Research, Education and Development of Unmanned Aerial Systems, RED-UAS 2015*, 115-122. http://doi.org/10.1109/RED-UAS.2015.7440998

Cova, W. J. D. (2005). Control PID, un Enfoque Descriptivo. *Universidad Tecnológica Nacional, Facultad Regional La Rioja, Departamento de Electrónica*. Recuperado de http://www.frlr.utn.edu.ar/archivos/alumnos/electronica/catedras/38-sistemas-de-control-aplicado/Publicaciones/Control_PID_Enfoque_Descriptivo.pdf

De Castro, D. F., y dos Santos, D. A. (2016). A software-in-the-loop simulation scheme for position formation flight of multicopters. *Journal of Aerospace Technology and Management*, *8*(4), 431–440. http://doi.org/10.5028/jatm.v8i4.612

Ellingsen, G., y McLain, T. (2017). ROSplane: Fixed-wing autopilot for education and research. *2017 International Conference on Unmanned Aircraft Systems, ICUAS 2017*, 1503–1507. http://doi.org/10.1109/ICUAS.2017.7991397

FlightGear. (2020). FlightGEar Flight Simulator. Recuperado de https://www.flightgear.org/

Gouthami, E., y Rani, M. A. (2016). Modeling of an Adaptive Controller for an Aircraft Roll Control System using PID, Fuzzy-PID and Genetic Algorithm, *11*(1), 15-24. http://doi.org/10.9790/2834-11121524

Hartanto, S., Furqan, M., Putera, A., Siahaan, U., y Fitriani, W. (2017). Haversine Method in Looking for the Nearest Masjid. *International Journal of Engineering Research*, (agosto). http://doi.org/10.23883/IJRTER.2017.3402.PD61H

Islam, M. T., Alam, M. S., Laskar, M. A. R., y Garg, A. (2016). Modeling and simulation of longitudinal autopilot for general aviation aircraft. *2016 5th International Conference on Informatics, Electronics and Vision, ICIEV 2016*, (diciembre del 2017), 490–495. http://doi.org/10.1109/ICIEV.2016.7760051

Jia, Y.-B. (2020). Rotation in the Space. *Iowa State University*, 1-14. Recuperado de http://web.cs.iastate.edu/~cs577/handouts/rotation.pdf

Khalid, A., Zeb, K., y Haider, A. (2019). Conventional PID, adaptive PID, and sliding mode controllers design for aircraft pitch control. *2019 International Conference on Engineering and Emerging Technologies, ICEET 2019*, (julio), 1-6. http://doi.org/10.1109/CEET1.2019.8711871

Koks, D. (2008). Using Rotations to Build Aerospace Coordinate Systems. *Electronic Warfare and Radar Division Systems Sciences Laboratory*. Recuperado de https://apps.dtic.mil/dtic/tr/fulltext/u2/a484864.pdf

Korkmaz, H., Ertin, O. B., Kasnakoğlu, C., y Kaynak, Ü. (2013). Design of a Flight Stabilizer System for a Small Fixed Wing Unmanned Aerial Vehicle using System Identification. *IFAC Proceedings Volumes (IFAC-PapersOnline)*, *1*(PART 1), 145-149. http://doi.org/10.3182/20130916-2-tr-4042.00012

Peet, M. M. (2010). Spacecraft and Aircraft Dynamics. *Illinois Institute of Technology*. Recuperado de http://control.asu.edu/Classes/MMAE441/Aircraft/441Lecture1.pdf

Qays, H. M., Jumaa, B. A., y Salman, A. D. (2019). Design and Implementation of Autonomous Quadcopter using SITL Simulator. *Iraqi Journals of Computers, Communications, Control & Systems Engineering*, *20*(1), 1-16. http://doi.org/10.33103/uot.ijccce.20.1.1

R., R., M., C., S., C., Kumar, P., y N., P. (2020). PID Controller Design for Dynamic Motion of an Aircraft. *SSRN Electronic Journal*, (mayo), 859-862. http://doi.org/10.2139/ssrn.3511417

Redshift Labs. (2020). Understanding Euler Angles. *Chrobotics*. Recuperado de http://www.chrobotics.com/library/understanding-euler-angles

Slabaugh, G. G. (2017). Computing Euler Angles from a Rotation Matrix. *Digital Environment Research Institute (DERI) Queen Mary University of London*, 1-7. Recuperado de https://www.gregslabaugh.net/publications/euler.pdf

Sudha, G., y Deepa, S. N. (2016). Optimization for PID Control Parameters on Pitch Control of Aircraft Dynamics Based on Tuning Methods. *Applied Mathematics and Information Sciences*, *10*(1), 343-350. http://doi.org/10.18576/amis/100136

Surowski, D. (2011). Distance between Points on the Earth's Surface. *Kansas State University, Department of Mathematics*. Recuperado de https://www.math.ksu.edu/~dbski/writings/haversine.pdf

Wahid, N., Hassan, N., Rahmat, M. F., & Mansor, S. (2011). Application of Intelligent Controller in Feedback Control Loop for Aircraft Pitch Control. *Australian Journal of Basic and Applied Sciences*, *5*(12), 1065–1074. Recuperado de http://www.ajbasweb.com/old/ajbas/2011/December-2011/1065-1074.pdf

White, M. (2019). PID Control for Robotics. *Programming.* Recuperado de https://mjwhite8119.github.io/Robots/pid-control