Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Automática y Sistemas Computacionales

TRABAJO DE DIPLOMA

Implementación del Algoritmo de guiado *I-LOS* sobre el autopiloto *ArduPlane*

Autor: Alejandro José Ortiz Larquin

Tutor: Dr. C. Luis Hernández Santana

Ing. Hector Daniel Alvarez Pérez

Santa Clara

2016

"Año 58 de la Revolución"

Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Automática y Sistemas Computacionales

TRABAJO DE DIPLOMA

Implementación del Algoritmo de guiado *I-LOS* sobre el autopiloto *ArduPlane*

Autor: Alejandro José Ortiz Larquin

aolarquin@uclv.cu

Tutor: Dr. C. Luis Hernández Santana

Dpto. de Automática. Facultad de Ing. Eléctrica. UCLV.
luishs@uclv.edu.cu

Ing. Hector Daniel Alvarez Pérez

Dpto. de Automática. Facultad de Ing. Eléctrica. UCLV.
hectorap@uclv.cu

Santa Clara

2016

"Año 58 de la Revolución"

Hago constar que el presente trabajo de diploma fue realizado en la Universidad Central "Marta Abreu" de Las Villas como parte de la culminación de estudios de la especialidad de Ingeniería en Automática, autorizando a que el mismo sea utilizado por la Institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos, ni publicados sin autorización de la Universidad.

Firma del Autor

Los abajo firmantes certificamos que el presente trabajo ha sido realizado según acuerdo de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

Firma del Autor

Firma del Jefe de Departamento donde se defiende el trabajo

Firma del Responsable de Información Científico-Técnica

PENSAMIENTO

"*El verdadero signo de la inteligencia no es el conocimiento, sino la imaginación.*"

Albert Einstein.

DEDICATORIA

A mis padres,
por ser los primeros en creer en mí y enseñarme que puedo llegar tan lejos como quiera.

A mi hermano,
por estar siempre presente sin importar la distancia.

A mis abuelos,
porque siempre he tenido su apoyo, tanto en este mundo como en el siguiente.

AGRADECIMIENTOS

Antes que todo agradezco a Dios, mi amigo fiel en batallas y momentos de gloria, en quien he puesto mi confianza y nunca me ha abandonado.

A mamá y papá, quienes en toda mi vida me han apoyado en todo lo que me he propuesto, me han animado a creer en mí mismo, superarme y crecer.

A mi hermano, el flaco, que a pesar de nuestras diferencias y boberías de hermanos, siempre que necesité una mano de ayuda la encontré al final de su brazo.

A mis abues, que con gran genialidad me enseñaron con su ejemplo el valor del esfuerzo y el trabajo, a la vez que me malcriaban y me mimaban mucho.

A mis amigos Martica, el Millo, Dizi, José Daniel, la “gente del cuarto”: Hector, Nelson, el Charly y muchos más, gracias por llevar tan bien la dificilísima tarea de soportarme todos estos años.

A Erick, por todos esos momentos súper estresantes que me enseñaron a aprender de verdad y que siempre hay una forma, solo hay que hallarla.

A mis tutores y profesores que ofrecieron su tiempo y su intelecto para brindarme los conocimientos que me acompañarán toda la vida.

A todos aquellos que rezaron por mí, me apoyaron, me sonrieron y de una forma u otra contribuyeron a hacer de mí la persona que soy hoy.

Santa Clara, Cuba, 2016.

RESUMEN

Los algoritmos de guiado, así como las plataformas de hardware y software donde se implementan, son elementos determinantes en el desempeño del funcionamiento de los vehículos autónomos no tripulados. En esta investigación se modifica la programación del autopiloto *ArduPlane* de la plataforma de hardware y software *ArduPilot,* incluyendo una implementación del algoritmo de guiado *I-LOS* para el desarrollo de vehículos aéreos no tripulados (*UAV*), en aras de brindar una aplicación real que permita a la comunidad científica desarrollar aplicaciones y profundizar en el estudio del desempeño de dicho algoritmo. Los resultados de la investigación se validan mediante simulación *Hardware-In-The-Loop* (*HIL*), utilizando una placa de hardware *APM2* con la cual se corrobora el correcto funcionamiento del sistema desarrollado en la investigación.

TABLA DE CONTENIDOS

INTRODUCCIÓN

El campo de los vehículos autónomos no tripulados se ha convertido en un tema de interés y de profundos avances científicos en el ámbito de la automática, donde el papel de los vehículos aéreos no tripulados, conocidos comúnmente por *UAV* (*Unmanned Aircraft Vehicles*), se ha convertido en el objeto de estudio de importantes centros de investigación y prestigiosos autores (Dalamagkidis, 2014) (Hagen, 2014) (Spearman, 1983) (Woo & Son, 2007).

Como resultado del auge de estos estudios han surgido en los últimos años varias plataformas de desarrollo de hardware y software para *UAV*, con el propósito de brindarle a los desarrolladores una base sólida sobre la cual implementar sus investigaciones a un precio razonable y con un tiempo de desarrollo reducido. Un exponente importante en estas plataformas lo constituye el proyecto *APM/ArduPilot* (ArduPilot, 2016).

Este proyecto se enfoca en el desarrollo de un sistema autopiloto de código abierto al cual se le incluyen diversas funcionalidades de propósito general, que resultan muy útiles en el desarrollo de nuevas aplicaciones. Cuenta con una importante comunidad de desarrolladores que continuamente revisan y mejoran su implementación y un arsenal de bibliotecas de código, que permiten el continuo desarrollo de nuevas funcionalidades y modificaciones a las ya existentes. Como parte de esta plataforma se encuentra el proyecto *ArduPlane* en el cual se desarrolla un sistema autopiloto para aviones. El mismo goza de una gran popularidad entre la comunidad de desarrolladores y es compatible con varias placas de hardware fuera del proyecto *ArduPilot*.

Las aplicaciones que pueden implementarse con un *UAV* son muy variadas. Entre ellas pueden encontrarse la inspección de cultivos y líneas eléctricas, la toma de filmaciones cinematográficas, el lanzamiento de misiles en maniobras de combate de manera autónoma, entre otras. Un punto a destacar en estas aplicaciones es que normalmente la efectividad de las mismas depende de que el *UAV* sea capaz de moverse siguiendo una trayectoria determinada.

En la implementación del autopiloto de un *UAV*, el control sobre el movimiento siguiendo una trayectoria predefinida se logra mediante la utilización de un algoritmo de guiado. Por

consiguiente, estos algoritmos juegan un papel determinante en el éxito de las misiones llevadas a cabo. Muchas son las investigaciones referentes a los algoritmos de guiado (Breivik & Fossen, 2008) (Caharija, 2014) (Fossen & Breivik, 2007) (Hagen, 2014) (Lekkas & Fossen, 2012) donde se han alcanzado excelentes resultados y se ha plasmado la importancia de estos algoritmos para el desarrollo de aplicaciones de *UAV*.

En el Grupo de Automatización, Robótica y Percepción (GARP) se han llevado a cabo diversas investigaciones sobre los algoritmos de guiado (Hernández Julián, 2014) (Martínez, 2014) (de Ávila, 2008) (Valeriano Medina, 2013) (Rivero Rodríguez, 2015) mediante el desarrollo de diversas aplicaciones de *UAV* y vehículos autónomos no tripulados en general (Ortega Escudero, 2015) (Álvarez, 2006). En muchas de estas investigaciones se ha trabajado sobre la base del algoritmo de guiado *I-LOS*. Con este algoritmo se han logrado excelentes resultados y además, es abordado por importantes autores en la bibliografía (Caharija, 2014) (Fossen & Breivik, 2007). Independientemente de estas investigaciones, el GARP no cuenta actualmente con una implementación en tiempo real de este algoritmo que permita analizar y estudiar su desempeño sobre una aplicación real.

Teniendo en cuenta lo expresado anteriormente, resulta de interés para el GARP contar con una implementación en tiempo real de este algoritmo de guiado. En aras de resolver este problema se plantea la siguiente **hipótesis**:

Realizando modificaciones en la programación del *firmware* de *ArduPilot*, se puede implementar el algoritmo de guiado *I-LOS* sobre el autopiloto *ArduPlane* desarrollado en esta plataforma.

De manera similar se establecen los objetivos del trabajo de la siguiente manera:

Objetivo general: Implementar el algoritmo de guiado *I-LOS* en el proyecto autopiloto *ArduPlane*, realizando modificaciones en la programación de la distribución oficial de *ArduPilot*.

Objetivos específicos:

- Estudiar los temas relacionados con los algoritmos de guiado, los *UAV* y las plataformas de desarrollo de *UAV* que aparecen en la literatura especializada.

- Analizar las características de la programación de la plataforma *ArduPilot* para el desarrollo de *UAV*.
- Programar el algoritmo de guiado *I-LOS* en base a la programación desarrollada en la distribución oficial de la plataforma *ArduPilot*.
- Simular los resultados obtenidos utilizando el método *Hardware-In-The-Loop* (*HIL*).

Para dar cumplimiento a estos objetivos se plantean las siguientes **tareas de investigación**:

- Estudio de los temas relacionados con la temática de los algoritmos de guiado, los *UAV* y las plataformas de desarrollo de *UAV* que aparecen en la literatura especializada.
- Estudio de la plataforma para el desarrollo de *UAV ArduPilot*.
- Implementación del algoritmo de guiado *I-LOS* sobre *ArduPlane*
- Estudio de los temas relacionados con la simulación *HIL*.
- Análisis del desempeño de la aplicación mediante simulación *HIL*.
- Elaboración del informe científico de la investigación.

El principal aporte que se persigue con el desarrollo de esta investigación, radica en obtener una implementación viable en tiempo real del algoritmo de guiado *I-LOS* sobre el autopiloto *ArduPlane*, que permita desarrollar aplicaciones para *UAV* empleando dicho algoritmo y sirva como herramienta para realizar investigaciones sobre el algoritmo de guiado *I-LOS*.

Estructura y contenido de la tesis:

El presente trabajo se constituye por cuatro capítulos, conclusiones, recomendaciones, referencias bibliográficas y anexos. El contenido de cada capítulo se manifiesta de la siguiente manera:

Capítulo 1: Se abordan los temas fundamentales relacionados con el desarrollo, la estructura y los componentes fundamentales que integran un *UAV*. Se estudian algunos ejemplos de aplicaciones reales desarrolladas sobre la base de estos vehículos. Finalmente, se hace una descripción de la estructura y los principales recursos de algunas plataformas para el desarrollo de *UAV*, abordando principalmente las características de la plataforma *ArduPilot* seleccionada en el desarrollo de esta investigación.

Capítulo 2: Trata sobre los conceptos fundamentales referentes al tema del guiado y el seguimiento de trayectorias en vehículos autónomos no tripulados, partiendo de un estudio

de la bibliografía especializada. Se presentan algunas leyes de seguimiento de caminos empleadas actualmente y se hace un estudio de las características, principios físicos y el desarrollo matemático que intervienen en el proceso de concepción y funcionamiento del algoritmo de guiado *I-LOS.*

Capítulo 3: Se abordan las características principales de la programación que constituye el *firmware* de los proyectos de *ArduPilot.* Se analizan los principales módulos de software empleados en la programación del algoritmo de guiado *I-LOS,* así como la metodología seguida en la implementación del mismo. De igual forma, se analizan los elementos que integran el autopiloto del proyecto *ArduPlane* y la metodología seguida en la implementación del controlador de alabeo. Finalmente se aborda el procedimiento seguido para la compilación del *firmware* de dicho proyecto.

Capítulo 4: Se describe el funcionamiento de la simulación *HIL* y su empleo en el análisis de elementos de control en dispositivos empotrados. Se abordan, además, las principales características de las herramientas de hardware y software empleadas para materializar dicho proceso de simulación. Finalmente, se analiza el procedimiento desarrollado para estudiar el desempeño del controlador *I-LOS* implementado, haciendo uso de la simulación *HIL* y los resultados obtenidos a partir de la misma.

CAPÍTULO 1. TENDENCIAS Y DESARROLLO DE LOS VEHÍCULOS AÉREOS NO TRIPULADOS

1.1 Introducción

En este capítulo se exponen los aspectos fundamentales relacionados con el desarrollo de la estructura y el funcionamiento de los vehículos aéreos no tripulados, así como su empleo e importancia en diversas aplicaciones. Se analizan además algunas plataformas de desarrollo para aplicaciones de este tipo. Finalmente, se hace una descripción técnica de la plataforma *ArduPilot* empleada en el desarrollo de esta investigación.

1.2 Desarrollo de los vehículos aéreos no tripulados

Varios autores expresan criterios sobre qué son los vehículos aéreos no tripulados, pero todos coinciden en que son aeronaves que vuelan sin la presencia de un piloto humano a bordo de las mismas, por lo que requieren de cierto grado de automatización controlada, ya sea remotamente o por medio de un microprocesador, con el objetivo de llevar a cabo las misiones que se les asignan (Hagen, 2014) (Hernández, 2014) (Martínez, 2014) (Álvarez, 2006). Comúnmente son referidos por su nombre en inglés *Unmanned Aircraft Vehicles* (*UAV*) o simplemente dron.

El concepto de *UAV* tiene varias décadas. El caso de uso más antiguo data de los tiempos posteriores a la Primera Guerra Mundial y ya eran empleados en la Segunda Guerra Mundial para el entrenamiento con cañones antiaéreos. Sin embargo, no es hasta el siglo XX que aparecen los primeros *UAV* operados mediante radiocontrol con características de autonomía. En la actualidad, con el creciente desarrollo de los microprocesadores y las tecnologías de sensores inerciales de estado sólido, es posible la fabricación de estas máquinas a precios asequibles para su desarrollo industrial. Las ventajas de no utilizar un piloto humano a bordo de la aeronave propician que los *UAV* gocen de una gran flexibilidad en cuanto al tamaño y la anatomía del vehículo, la disminución de los costos de fabricación y el entrenamiento de personal calificado, así como su utilización en misiones donde se arriesgue la vida de los pilotos a bordo de la aeronave. Por consiguiente, estos vehículos se han convertido en una

rama importante de investigación en el campo de la aeronáutica y la industria aeroespacial (Grankvist, 2006) (Adiprawita & Sembiring, 2007) (Woo & Son, 2007) (Hagen, 2014) (de Ávila, 2008).

Actualmente los *UAV* existen en diferentes formas y tamaños, diferenciándose por sus características y desempeño ante varias tareas. Su desarrollo se extiende desde aviones de pequeño porte hasta aviones de mediano y gran tamaño. Un número elevado de estos vehículos se utilizan en aplicaciones civiles (Dalamagkidis, 2014). Ejemplo de estas aeronaves son el *DJI Phantom* ,que se muestra en la Figura 1-1, utilizado principalmente para la toma de fotografías en zonas de difícil acceso y el *md4-1000*, mostrado en la Figura 1-2, que es utilizado para la entrega de medicinas y provisiones en lugares remotos.

Figura 1-1 DJI Phantom.

Figura 1-2 md4-1000.

Por otra parte, existen también los llamados sistemas aéreos de combate no tripulados o mayormente conocidos por *UCAS* (*Unmanned Combat Aircraft Systems*), cuyo desarrollo se enfoca principalmente en aplicaciones militares. Algunos ejemplos de *UCAS* son el *MQ-9 Reaper* utilizado en combates aéreos, mostrado en la Figura 1-3, y el *Schiebel S-100* que se emplea para el combate con misiles de peso ligero, mostrado en la Figura 1-4.

Figura 1-3 MQ-9 Reaper.

Figura 1-4 Schiebel S-100.

Debido a la acelerada difusión de los *UAV* y su elevada popularidad tanto en el ámbito civil como el militar, la Administración Federal de Aviación *FAA* (*Federal Aviation Administration*), entidad gubernamental responsable de la regulación de todos los aspectos de la aviación civil en los Estados Unidos, permite su uso en territorio de ese país mediante la sección 336 de la ley pública 112-95, como medios de entretenimiento o hobbies siempre que la aeronave cumpla con los lineamientos de seguridad establecidos por dicha ley (FAA, 2012). Esto ha permitido un desarrollo aún mayor por parte de individuos aficionados a la aviación y pequeñas compañías para fines no lucrativos.

En Cuba algunas empresas como GEOCUBA, CITMA, MINAGRI, AZCUBA, CEDAI y CEMPALAB, entre otras han comenzado investigaciones sobre los *UAV* con vistas a utilizarlos en sus respectivas áreas de trabajo, debido a las ventajas que estos ofrecen y con el objetivo de desarrollar alternativas nacionales que permitan reducir costos por importaciones.

1.3 Componentes y Estructura de los *UAV*

Son muchos los componentes que conforman la estructura básica que permite el funcionamiento de un *UAV*. Para una mejor ilustración se dividen en tres categorías: estructura física, hardware y software. La estructura física se centra en los elementos relacionados con la construcción del *UAV* tales como fuselaje, hélices, herramientas de carga, entre otras. El hardware se enfoca en los elementos de procesamiento, comunicación, sensores, actuadores de pequeña envergadura, entre otros. El software incluye la

programación de los algoritmos que controlan el funcionamiento de la aeronave, la atención a los sensores y los protocolos de comunicación.

Existe una gran variedad de materiales para la fabricación del fuselaje de un *UAV*. Los más utilizados son el aluminio, el poliestireno expandido, el policloruro de vinilo *(PVC*) y la fibra de vidrio. La evolución de las impresoras *3D* permite el diseño de estructuras de fuselaje más complejas, minimizando el peso de la aeronave y contribuyendo al desarrollo de estructuras más aerodinámicas (LiveScience, 2016).

Los *UAV* más pequeños dependen generalmente de baterías de polímero de litio (*Li-Po*), mientras que vehículos más grandes emplean combustibles convencionales, como la gasolina o el petróleo y algunas fuentes de energía alternativa como la solar. Las aplicaciones de los *UAV* se caracterizan por tener largos períodos de duración de batería, por lo que se requiere de componentes que minimicen el consumo de energía o que puedan obtener energía de forma natural, como es el caso de los *UAV* satélites atmosféricos conocidos como *atmosats*, que funcionan con energía solar y pueden permanecer a una altura de 20km sobre la superficie terrestre por un tiempo aproximado de 5 años (Lara, 2015). Se ha avanzado mucho en el desarrollo de fuentes híbridas, así como en potencia de hidrógeno, extendiendo el funcionamiento de los *UAV* por varias horas (GTRI, 2016) (Gizmag, 2016).

Los sistemas empotrados de los *UAV* más pequeños han migrado en los últimos años del uso de microcontroladores, al empleo de sistemas en un chip conocidos como *SOC* (*System on a chip*) o computadoras de una sola placa denominadas *SBC* (*Single Board Computers*). El hardware de un *UAV* tiende a especializarse para mejorar en la velocidad de los cálculos con un bajo poder de procesamiento (aplicaciones de tiempo crítico) y una alta capacidad computacional capaz de soportar sistemas operativos completos.

La mayoría de los *UAV* emplean en sus módulos de comunicación un sistema de radiofrecuencia, que permite a la aeronave alejarse distancias de varios cientos de metros de la estación de control en tierra sin perder la comunicación. Otros que se utilizan en aplicaciones de distancias más reducidas emplean módulos de *Wi-Fi* o *Bluetooth*. En los *UAV* de aplicaciones militares son muy comunes los módulos de comunicación satelital, que le permiten a la aeronave recibir instrucciones en zonas remotas, así como transmitir información referente a su estado actual e información sobre la misión en desarrollo.

Para todo *UAV* son fundamentales los sensores a bordo, dado que a través de estos la aeronave puede cumplir correctamente con la misión asignada. Los tipos de sensores utilizados por los *UAV* pueden dividirse en tres categorías (Arduino, 2016):

- **Interoceptivos**. Se refiere a los sensores que captan información del estado y los movimientos inherentes al cuerpo en sí. Entre ellos se encuentran los giróscopos, acelerómetros, brújulas, altímetros, módulos *GPS*, entre otros.
- **Exteroceptivos**. Se refiere a los sensores que captan la información proveniente del medio exterior del cuerpo. Se destacan las cámaras, los medidores infrarrojos, radares, sonares, barómetros y otros.
- **Exterinteroceptivos**. Se refieren a los sensores que captan información común tanto para el medio interno del cuerpo como para el externo. Algunos ejemplos son los termómetros, micrófonos, fotosensores entre otros ejemplos.

Los actuadores encontrados en un *UAV* dependen del tipo de vehículo y las aplicaciones para las cuales está destinado. Los más comunes son los controladores electrónicos digitales. También pueden encontrarse pequeños servomotores y motores paso a paso para el control de la posición de los alerones.

El software de un *UAV* puede dividirse en una pila de capas de software o capas de abstracción, de acuerdo a los diferentes requerimientos en la aplicación. Al conjunto de todas las capas se le denomina pila de vuelo, del inglés *flight stack* o autopiloto. En la Figura 1-5 se observa un esquema de capas de software de un *UAV*, el cual se corresponde además con la pila de capas de abstracción empleado en las plataformas para el desarrollo de *UAV* analizadas en esta investigación. En dichos esquemas las capas inferiores brindan servicios a las capas superiores y una capa inferior no puede utilizar un servicio provisto por una capa superior. Este esquema de abstracción permite alcanzar un elevado nivel de organización, con una estructuración eficiente del trabajo de los desarrolladores y a la vez realizar ajustes y correcciones en partes del código perteneciente a una capa, sin que esto afecte al código perteneciente a otra capa.

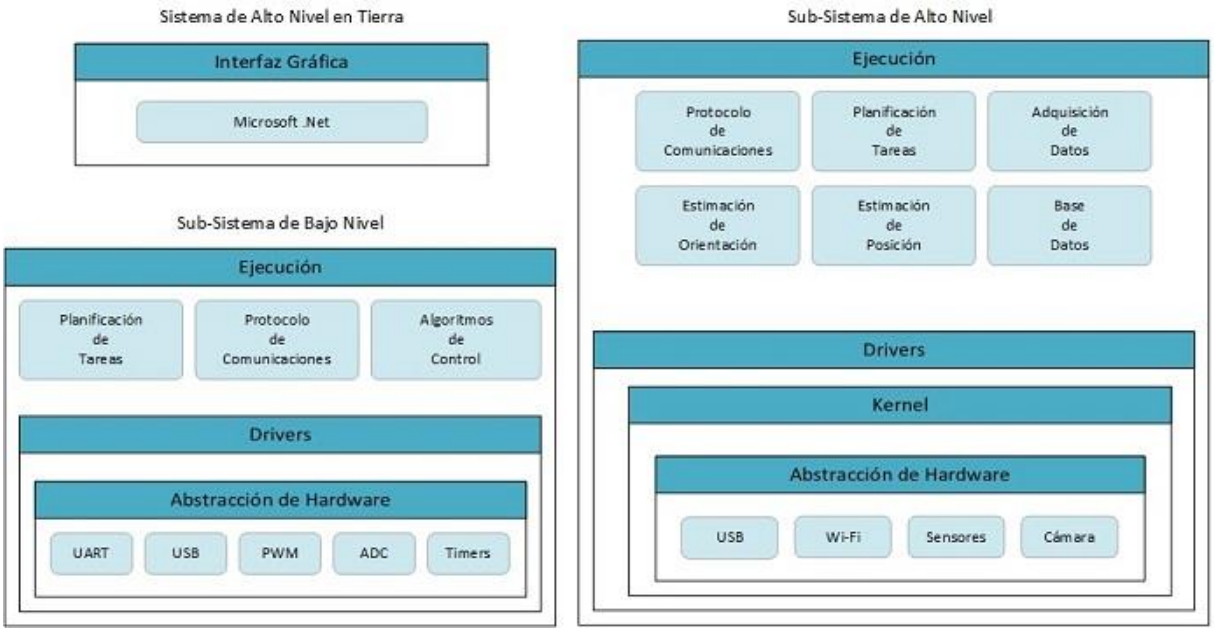

Figura 1-5 Capas de software de un UAV.

En el esquema de la Figura 1-5 puede observarse la capa de Sistema de Alto Nivel en Tierra. Esta capa encapsula el código perteneciente a la interfaz gráfica con la que interactúa el operador desde la estación de control en tierra, para recibir la información proveniente de la aeronave y modificar los parámetros de operación de la misma. Seguidamente está la capa de Sub-Sistema de Bajo Nivel, la cual encapsula el código que se encarga de ejecutar las tareas referentes a la estación de control en tierra. En esta capa se llevan a cabo funcionalidades vitales, tales como la ejecución de los algoritmos de planificación de tareas y la ejecución de los protocolos de comunicación. Para realizar las operaciones pertinentes se basa en la capa *Drivers,* la cual tiene su base en la Capa de Abstracción de Hardware. Esta última encapsula el código de máquina que interactúa con los componentes de hardware y expone sus servicios a través de la capa *Drivers*.

La capa de Sub-Sistema de Alto Nivel encapsula el código que define el funcionamiento de la aeronave. En la misma se ejecutan tareas necesarias, como son la adquisición de datos a través de los sensores a bordo, la ejecución de los algoritmos de estimación de orientación y posición, el manejo de las bases de datos, la ejecución de los protocolos de comunicación, entre otros. Posee capas internas como *Drivers*, *Kernel* y Abstracción de Hardware, la cual al igual que en la capa de Sub-Sistema de Bajo Nivel, encapsula el código de máquina que

interactúa con los elementos de hardware para dar el tratamiento adecuado a los sensores, elementos de cómputo y demás recursos de hardware a bordo de la aeronave. Es importante resaltar que este esquema no es más que una abstracción del software que implementa un *UAV*, por lo tanto, el mismo software puede ser descrito en un esquema con un número diferente de capas o con capas diferentes a las enunciadas anteriormente.

Comúnmente las aplicaciones de *UAV* se programan con la capacidad de realizar operaciones de forma autónoma sin el consentimiento del operador. Los mismos reciben desde la estación en tierra las instrucciones para la misión. De esta forma, el procesamiento de la información y la toma de decisiones referentes a los algoritmos de navegación, ocurren a bordo de la aeronave sin la intervención de los operadores. De igual manera, a muchos *UAV* se les programa una misión por conexión directa en tierra y es llevada a cabo totalmente de manera autónoma hasta su culminación, sin comunicación con una estación de control terrestre.

Algunos sistemas operativos convencionales tales como *Windows*, *Linux* o *MacOS* generalmente no proveen un funcionamiento adecuado en el autopiloto a bordo de un *UAV*, por lo que se requiere de la adición de componentes de software conocidos como *middlewares*, que mejoren la respuesta en tiempo real de estos sistemas. Algunos ejemplos de plataformas que utilizan este método son *Raspberry Pi* y *Beagleboards*, que se acoplan con *NAVIO* o *PXFMini*. También existen plataformas de software para *UAV* que se diseñan desde sus inicios para funcionar en aplicaciones que requieren de tiempo real duro. Ejemplos de estas plataformas son *Nuttx*, *preemp-RT Linux*, *Xenomai*, *DDS-ROS 2.0* entre otras.

1.4 Aplicaciones y Ejemplos de los *UAV*

En la actualidad existe un elevado número de aplicaciones de *UAV*, tanto civiles como militares. En el ámbito de las aplicaciones civiles se encuentran el reconocimiento aéreo de cultivos, tomas acrobáticas en filmaciones cinematográficas, seguimiento de huracanes y control del clima, operaciones de búsqueda y rescate, inspección de tuberías y líneas de corriente, estudios científicos de vida salvaje, entrega de medicamentos y otros tipos de accesorios en lugares remotos, detección de incendios forestales, distribución de señales de internet, seguimiento de planificación urbanística, estudios de hidrología, geología y topografía, además de otras aplicaciones variadas (Fung, 2013) (Peterson, 2013).

En la industria militar los *UAV* son considerados una estrella en ascenso, sus aplicaciones incluyen su uso en misiones de reconocimiento, vigilancia de fronteras, protección de convoyes, misiones de búsqueda y rescate, combate aéreo, detección de campos minados, desarme de bombas, entrenamiento en maniobras antiaéreas, la protección antimisiles, entre otras (McFarland, 2014) (Pasztor & Emshwiller, 2012) (Campoy, 2011).

Ejemplo del uso de drones en el área de la industria militar es el caso de los *UAV MQ-1 Predator,* como el mostrado en la Figura 1-6. Estos *UAV* han sido utilizados por el ejército de los Estados Unidos como plataformas para atacar blancos terrestres. Fueron utilizados por vez primera a finales del año 2001 desde bases ubicadas en Pakistán y Uzbekistán, principalmente para eliminar objetivos de alto perfil (líderes terroristas, entre otros) ubicados en territorio afgano. (DID, 2005) (NBC, 2005) (Carroll, 2012).

Muchos departamentos de policía en varios países han hecho uso de los *UAV* para un mejor cumplimiento de la ley y el orden. En el 2010, la policía de Mereyside capturó a un ladrón de autos gracias al apoyo de un *UAV* (Lundin, 2013).

Luego del impacto de varios tornados en las zonas de Luisiana y Tejas en Estados Unidos en el año 2008, se utilizaron *UAV* de tipos *Aeryon Scout,* como el mostrado en la Figura 1-7, para labores de búsqueda y rescate así como la evaluación de daños. Estos drones llevaban a bordo sensores ópticos y un radar de apertura sintética el cual puede capturar imágenes a través de nubes, lluvia, niebla y en condiciones de poca luminosidad, todas en tiempo real. Tomando como base las imágenes tomadas por los *UAV* antes y después de la catástrofe un software puede compararlas para detectar las zonas afectadas (WM, 2008).

Figura 1-6 MQ1-Predator lanzando un misil.

Figura 1-7 Aeryon Scout en pleno vuelo.

1.5 Plataformas de desarrollo para *UAV*

Debido al acelerado desarrollo de la electrónica, las comunicaciones y los sistemas automáticos para vehículos autónomos no tripulados, el proceso de diseño e implementación de una aplicación para *UAV* requiere de una alta gama de herramientas y conocimientos que influyen de forma negativa en el tiempo de desarrollo y gestión de recursos de una aplicación de este tipo. Para dar solución a esta problemática han surgido en los últimos tiempos diversas plataformas de desarrollo para hardware y software con el propósito de viabilizar el proceso de implementación de aplicaciones utilizando *UAV*, sobre la base de una estructura estándar.

Estas plataformas abarcan un amplio espectro de escenarios de diseño para los cuales brindan herramientas y componentes modificables según los requisitos de la aplicación. De esta forma, los elementos básicos que intervienen en el desarrollo de un *UAV* ya se encuentran implementados y es tarea del desarrollador modificarlos para que cumplan con dichos requisitos y desarrollar aquellos que son específicos para la aplicación.

Entre las funcionalidades más comunes que se encuentran en estas plataformas están: el manejo de los protocolos de comunicación entre el vehículo y la estación de control, manejo de los sensores y actuadores de la aeronave, implementación del autopiloto del vehículo con sus respectivos algoritmos de guiado y lazos de estabilidad, desarrollo del software referente a la estación de control en tierra, entre otras.

Entre las plataformas para el desarrollo de *UAV* más populares se encuentra la plataforma *NAVIO* (Emlid, 2016). Esta plataforma se distribuye como una versión libre para la comunidad y una versión comercial con módulos y herramientas extras. En esencia se

compone de un sistema autopiloto basado en *Linux,* configurado para funcionar en tiempo real sobre una computadora *Raspberry Pi.* Posee soporte para los proyectos de autopilotos *ArduPlane, ArduCopter* y *Rover*. Es compatible además con diferentes estaciones de control en tierra diseñadas para los sistemas operativos *Windows, Linux, MacOS, Android* e *iOS.*

Entre las características principales de la placa de hardware implementada en esta plataforma se encuentra un receptor *GNSS* integrado, con soporte para los sistemas satelitales *GPS, GLONNAS* y *Beidou.* Un coprocesador *RC* de entrada/salida que acepta *PPM y SBUS* y 14 canales de *PWM* con frecuencia variable. Además, posee un barómetro de altas prestaciones con mediciones de altitud de 10cm de resolución.

Otra de estas plataformas es *Erle-Brain 2* (ErleRobotics, 2016). Esta plataforma de hardware abierto es definida por sus autores como un cerebro robótico artificial basado en *Linux* con soporte oficial para el sistema operativo *ROS* (*Robot Operating System*). En esencia es una computadora diseñada para *Linux Raspberry Pi 2* conectada a una placa hija, la cual contiene varios sensores, módulos de entrada/salida y la electrónica de potencia que sostiene la aplicación. Es compatible con sistemas autopilotos como *ArduPlane* y *DroneKit-Python.*

Entre sus principales características se encuentra que provee acceso a la tienda de aplicaciones de *Linux, Snappy Ubuntu* y permite instalar aplicaciones desde la misma. La placa de hardware contiene módulos de comunicación *Wi-Fi* con un proxy *MAVProxy* que permite crear un puente entre paquetes del protocolo *MAVLink* y la comunicación establecida a través de la conexión *Wi-Fi.* Cuenta con sensores de gravedad, giróscopos, brújula digital, sensores de presión y temperatura, entre otros. Posee además una interfaz de conexión para una cámara opcional y un módulo *Bluetooth.*

Entre las plataformas para el desarrollo de *UAV* más populares se encuentra también *Pixhawk* (Pixhawk, 2016). Esta plataforma es un proyecto independiente de hardware abierto cuyo objetivo es proveer un sistema autopiloto a la comunidad de desarrolladores. Consiste en una placa de hardware que contiene sensores, dispositivos de cómputo, medios de comunicación y demás componentes de hardware necesarios en una aplicación para *UAV.* El autopiloto implementado funciona sobre la base de un sistema operativo para tiempo real *RTOS,* el cual provee un ambiente de trabajo al estilo de *POSIX.*

Entre sus principales características se observa que es compatible con los sistemas para autopilotos *ArduPlane* y el sistema *PX4* desarrollado específicamente para las aplicaciones basadas en *Pixhawk.* La placa de hardware funciona con un procesador a 168 MHz *Cortex M4F*. Posee acelerómetros, giróscopos, magnetómetros, sensores de presión barométrica en conjunto con otros sensores necesarios para las aplicaciones de *UAV* más comunes.

Otra de estas plataformas para el desarrollo de *UAV* es la plataforma *ArduPilot* (ArduPilot, 2016), también conocida como *APM.* Una de sus principales características es que en esta plataforma se desarrolla el sistema autopiloto de código abierto *ArduPlane.* Se estructura en varios proyectos de acuerdo al tipo de aplicación que se desarrolle. Implementa la placa de hardware abierto *APM* la cual actualmente se encuentra en su versión 2.6 y posee recursos de hardware tales como sensores, interfaces de comunicación, elementos de cómputo, entre otros que son indispensables en las aplicaciones de *UAV* más comunes.

El autopiloto *ArduPlane* desarrollado en esta plataforma es empleado por otras plataformas de desarrollo ajenas al proyecto *ArduPilot*/*APM* tales como *Pixhawk*, *NAVIO* y *Erle-Brain 2* mencionadas anteriormente. El mismo goza de gran popularidad en la comunidad de desarrolladores y es soportado por la mayoría de las plataformas de desarrollo de *UAV*.

Actualmente el GARP cuenta con varios ejemplares de placas *APM* en la versión *2.5*, la cual cuenta con los recursos de hardware necesarios para realizar una aplicación de *UAV* en tiempo real y es compatible con el sistema autopiloto *ArduPlane*. Tomando en consideración estos elementos se selecciona a *ArduPilot/APM*, como la plataforma de desarrollo de *UAV* para la implementación de una aplicación en tiempo real con el algoritmo de guiado *I-LOS* propuesto en esta investigación. Una ventaja a destacar en la utilización del autopiloto *ArduPlane* es que, dado que es compatible con varias plataformas de desarrollo de *UAV*, los resultados de la investigación pueden ser fácilmente reutilizados en otras plataformas además de *ArduPilot.*

1.6 Estructura y recursos de *ArduPilot*

Como se menciona en el epígrafe anterior, en esta investigación se trabaja sobre la base de la plataforma de hardware y software *ArduPilot*. La misma es un sistema autopiloto de código abierto para el desarrollo íntegro de aplicaciones de vehículos autónomos no tripulados. El

prefijo *'Ardu'* utilizado en el nombre se deriva de *Arduino,* debido a que la primera placa original de *ArduPilot* estaba desarrollada con base en este ambiente de desarrollo. Actualmente la plataforma ha experimentado muchas transformaciones y un elevado crecimiento que sobrepasa la capacidad del ambiente *Arduino* y por tanto, no hace uso exclusivo de sus librerías de ejecución, aunque si soporta la compilación para las placas *APM1* y *APM2* basadas en *AVR,* con una versión del ambiente de desarrollo *Arduino* ligeramente modificado (ArduPilot, 2016).

Dado que la plataforma soporta otros modelos de placas además de las que son basadas en *Arduino,* tal como *Pixhawk,* los desarrolladores han decidido reemplazar el prefijo *'Ardu'* con el prefijo *'APM'* (*Ardu Pilot Mega*) en sus distribuciones de *firmware* más recientes por lo que es común encontrar en la bibliografía más actualizada alusiones a proyectos de *ArduPilot* utilizando dicho prefijo.

ArduPilot se compone de un tronco común donde se implementa la estructura central de las funcionalidades de control y navegación de los vehículos, en conjunto con varios proyectos que facilitan los servicios requeridos para la implementación de una aplicación de este tipo. Cada uno de los proyectos que conforman *ArduPilot* cuenta con una amplia comunidad de desarrolladores que continuamente expanden su alcance y efectividad. Actualmente los proyectos asociados a *ArduPilot* son (ArduPilot, 2016):

- ***DroneKit.*** *SDK* de *APM* para aplicaciones corriendo en vehículos, dispositivos móviles y la nube.
- ***ArduPlane (APM Plane).*** Autopiloto para aviones.
- ***ArduCopter (APM Copter).*** Autopiloto para multirrotores y helicópteros tradicionales.
- ***Rover (APM Rover).*** Autopiloto para vehículos terrestres.
- ***Mission Planner.*** Estación en tierra para control de misiones.
- ***APM Planner 2.0*** Estación en tierra específica para *APM*.
- ***MAVProxy.*** Estación en tierra orientada a la línea de comandos.
- ***MinimOSD.*** Visor de datos de vuelo *on-screen*.
- ***AndroPilot.*** Estación en tierra para el sistema operativo *Android*.

- ***DroneAPI.*** *API* de desarrolladores para aplicaciones *WEB* y coprocesadores de drones.
- ***DroidPlanner2.*** Estación en tierra para el sistema operativo *Android*.
- ***QGroundControl.*** Estación en tierra alternativa escrita en *C++* utilizando bibliotecas del *framework Qt*.
- ***MAVLink.*** Protocolo de comunicaciones entre la estación en tierra, controlador de vuelo y algunos periféricos incluyendo el *OSD*.

El código fuente de *ArduPilot*/*APM* está escrito sobre la capa de abstracción de hardware *AP-HAL*, lo que hace posible portar el código a un amplio rango de placas de autopiloto. Las placas controladoras de autopiloto soportadas actualmente son *Pixhwak*, *PX4 FMU*, *APM2*, *Arsov AUAV-X2*, *Erle-Brain 2* y *NAVIO+* (ArduPilot, 2016).

El código principal de vuelo de *ArduPilot* está escrito en *C++* y las herramientas de soporte en una amplia variedad de lenguajes, más comúnmente en *Pyhton*. Actualmente el código principal de los vehículos se escribe en archivos de extensión '*.pde'*, la cual se deriva del sistema de compilación de *Arduino*. Los archivos *'.pde'* son preprocesados en archivos *'.cpp'* de *C++* como parte de la compilación (ArduPilot, 2016).

Los sistemas autopiloto implementados en base a la plataforma de *ArduPilot* implementan diversas funcionalidades tales como la comunicación con la estación en tierra y dispositivos de control remoto por radiofrecuencia. Incluyen además las implementaciones de las operaciones de seguridad ante fallos, los protocolos de vuelos autónomos, la generación de bases de datos, entre otras.

El código de *ArduPilot* en conjunto con sus proyectos asociados está liberado como parte de la comunidad de software libre, bajo la Licencia Pública General de *GNU* versión 3 y superiores (*GPLv3*) (GNU, 2016). Esto facilita su desarrollo y extensión por parte de terceros y desarrolladores independientes.

Como parte de las funcionalidades que implementa *ArduPilot* es de interés en esta investigación el algoritmo de guiado presente en el sistema autopiloto *ArduPlane*. Como se observa en el epígrafe 1.4, las aplicaciones más generales para un *UAV* requieren que el mismo sea capaz de trasladarse siguiendo una trayectoria predefinida con un mínimo de error en la posición, lo que destaca la importancia de estos algoritmos. En la implementación actual

de *ArduPlane* se desarrolla únicamente el algoritmo de guiado *L1*. Este algoritmo ha sido empleado en numerosas aplicaciones de *UAV*, desarrolladas utilizando la plataforma *ArduPilot y* con excelentes resultados en su desempeño (ArduPilot, 2016). En el GARP se han realizado además diversas investigaciones alrededor de este algoritmo obteniéndose resultados satisfactorios en las mismas (Rivero Rodríguez, 2015). La forma en que se estructura la programación de *ArduPilot* permite que se modifique el código del algoritmo de guiado sin comprometer la integridad de los restantes elementos del autopiloto *ArduPlane*.

De manera similar en el GARP se han llevado a cabo extensas investigaciones sobre el algoritmo de guiado *I-LOS* (Ortega Escudero, 2015) (Hernández Julián, 2014) logrando excelentes resultados en dichas investigaciones. Este algoritmo ha sido referenciado además por importantes autores de la bibliografía sobre el campo de los algoritmos de guiado (Caharija, 2014) (Fossen, 2002) (Elfes, 2003) destacando la efectividad del mismo en sus aplicaciones. Teniendo en cuenta estos elementos se hace importante para el GARP contar con una implementación del algoritmo de guiado *I-LOS* sobre *ArduPlane* en aras de profundizar en el estudio científico de este algoritmo y analizar su desempeño sobre la plataforma *ArduPilot* para el desarrollo de nuevas aplicaciones de *UAV*.

1.7 Consideraciones finales del capítulo

Los *UAV* tienen la capacidad de ser empleados en diversas aplicaciones civiles y militares, alcanzando resultados superiores a los obtenidos mediante el empleo de una aeronave controlada por un piloto humano. La plataforma *ArduPilot* resulta en una alternativa viable y eficaz en el desarrollo de nuevas aplicaciones para *UAV*, dado que provee al desarrollador con el sistema autopiloto *ArduPlane*, el cual está ampliamente soportado por numerosas plataformas de desarrollo de *UAV*. Mediante el empleo de una plataforma como *ArduPilot*, se puede alcanzar un tiempo de desarrollo menor con una reducción de costos en la implementación de una aplicación, ante la alternativa de realizarla sin el empleo de una de estas plataformas. Una investigación de este tipo goza de gran importancia y utilidad en las actuales investigaciones llevadas a cabo en el GARP sobre los *UAV* y los algoritmos de guiado.

CAPÍTULO 2. ESTUDIO SOBRE LOS ALGORITMOS DE GUIADO

2.1 Introducción

En este capítulo se exponen los aspectos fundamentales relacionados con el desarrollo de los algoritmos de guiado y en particular el controlador *I-LOS* implementado en esta investigación, a partir de un estudio de la bibliografía especializada. Se analizan algunos ejemplos de estos algoritmos, enfatizando en los principales conceptos relacionados con el guiado y los sistemas de autopiloto, así como el desarrollo matemático requerido para su implementación.

2.2 Estudio sobre los algoritmos de guiado

La trayectoria determina el movimiento de un objeto en el espacio, la misma puede ser descrita a partir de la geometría del camino o a través de la posición del vehículo en el tiempo. El guiado de un vehículo se define como el proceso mediante el cual un objetivo que generalmente se encuentra en movimiento, sigue una trayectoria determinada a partir de un conjunto de puntos (Breivik & Fossen, 2008).

Los ejemplares más antiguos en la literatura sobre este tema son los relacionados con el guiado de misiles, este se definía como el desplazamiento de vehículos autónomos que contienen en sí los medios para controlar su trayectoria (Fossen & Breivik, 2007). El elevado desarrollo en el guiado y el control de los *UAV* en gran parte se basa en la teoría sobre el guiado de misiles desarrollada a partir de la Segunda Guerra Mundial (Spearman, 1983).

El funcionamiento de los sistemas de guiado está determinado a partir de dos elementos principales: el controlador de rumbo del vehículo y un algoritmo o ley de guiado para controlar el seguimiento de la trayectoria, que puede incluir o no la velocidad. En la *Figura 2-1* se muestra un esquema general del sistema de control de movimiento de un *UAV*, donde se observan estos elementos.

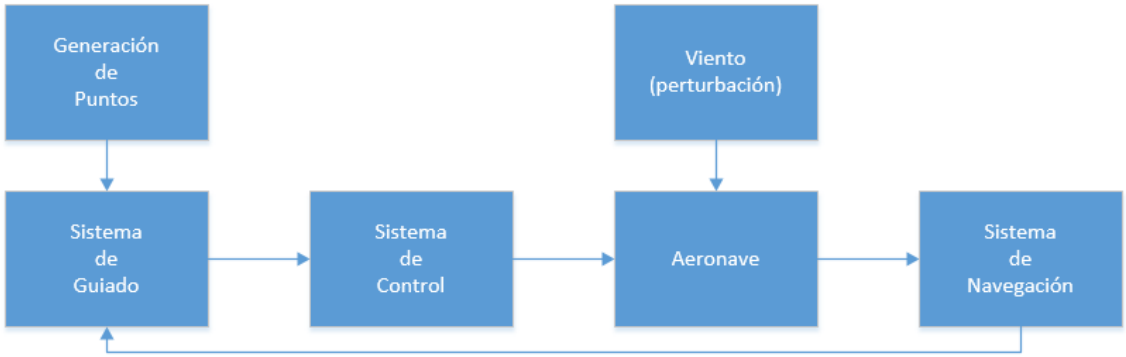

Figura 2-1 Esquema general del control de movimiento de un UAV.

Los escenarios existentes para el control de movimiento del vehículo y sus respectivos objetivos son (Breivik & Fossen, 2008):

Seguimiento del objetivo: Se basa en el seguimiento de un objetivo el cual puede estar en movimiento o estacionario (estabilización de puntos). Su movimiento instantáneo es conocido y la información sobre el objetivo futuro es desconocida. En este caso es imposible separar las restricciones espaciales de las temporales.

Seguimiento de Camino: Conocido por sus siglas en ingles *PF* (*Path Following*), basa su funcionamiento en seguir un camino predefinido, el cual solo involucra restricciones de tipo espaciales.

Seguimiento de objetivo: Conocido por sus siglas en inglés *TT* (*Target Tracking*) el mismo sigue un objetivo que se mueve a lo largo de un camino predefinido.

Maniobrabilidad: Optimiza el camino a seguir poniendo las restricciones espaciales sobre las temporales.

Como se menciona anteriormente, el estudio de los algoritmos de guiado comienza principalmente con el estudio del guiado de misiles en la industria armamentista, por lo que se hace muy común encontrar términos de uso militar en la bibliografía más avanzada sobre los algoritmos de guiado, aún cuando se traten temas ajenos a las aplicaciones militares. De esta forma son utilizados dos conceptos fundamentales: interceptor y objetivo. El interceptor se refiere al misil que busca dar en el blanco y generalmente hace alusión al vehículo que se

encuentra en movimiento, mientras que el objetivo se define como el blanco y por lo general supone el punto final al que se dirige el vehículo (Fossen & Breivik, 2007).

Es importante aclarar que en el idioma español los términos seguimiento de objetivo y seguimiento de camino pueden ser utilizados como sinónimos. La principal diferencia entre ambos se encuentra en el tipo de restricciones que involucran. En el caso del término seguimiento de objetivo se involucran tanto las restricciones temporales como las espaciales mientras que en seguimiento de camino solo importan las restricciones espaciales (Cruz, Aranda, & Girón, 2012).

El recorrido del interceptor se divide en tres fases: lanzamiento, mitad del recorrido y final. La fase final es la que requiere de mayor precisión, puesto que el interceptor debe arribar al blanco con el mínimo error posible. Las principales estrategias de guiado en esta fase son (Fossen & Breivik, 2007):

Línea de Visión: Conocido por sus siglas en inglés *LOS* (*Line of Sight*), es la estrategia del controlador *I-LOS* utilizado en esta investigación. La misma es una estrategia conformada por tres puntos: referencia, interceptor y objetivo. Se basa en la existencia de un vector en línea recta que intercepta con la línea de visión existente entre la referencia y el objetivo. Se utiliza frecuentemente en misiles tierra-aire y en vehículos marinos y aéreos (Rysdyk, 2003) (Osborne & Rysdyk, 2005) (Nelson, 2007) (Rysdyk, 2007) (Velasco, 2008) (Hagen, 2014) (Lekkas, 2014) (Caharija, 2014) (Kothari, Postlethwaite, & Gu, 2014).

Persecución: Conocido por sus siglas en inglés *PP* (*Pure Pursuit*) consiste en una estrategia de guiado conformada por dos puntos: interceptor y objetivo. El mismo supone alinear la velocidad del interceptor con la del objetivo a través de una línea de visión entre ambos. Existe una variante de este algoritmo conocida como navegación en dirección fija, la misma funciona a partir de alinear la velocidad del interceptor en dirección a una línea de visión con un ángulo constante para interceptar el movimiento del objetivo. Es utilizado en misiles tierra-aire y en vehículos marinos (Breivik & Fossen, 2008).

Dirección Constante: Conocido por sus siglas en inglés *CB* (*Constant Bearing*) es una estrategia semejante a *PP*. La misma supone alinear el interceptor a una velocidad relativa (interceptor-objetivo) a lo largo de la línea de visión entre ambos. Cuando el objetivo se

encuentra estacionario las estrategias *PP* y *CB* se asumen como exactamente iguales. Es utilizado en misiles aire-aire (Breivik & Fossen, 2008).

En la *Figura 2-2* se observan las particularidades que caracterizan a cada una de estas estrategias.

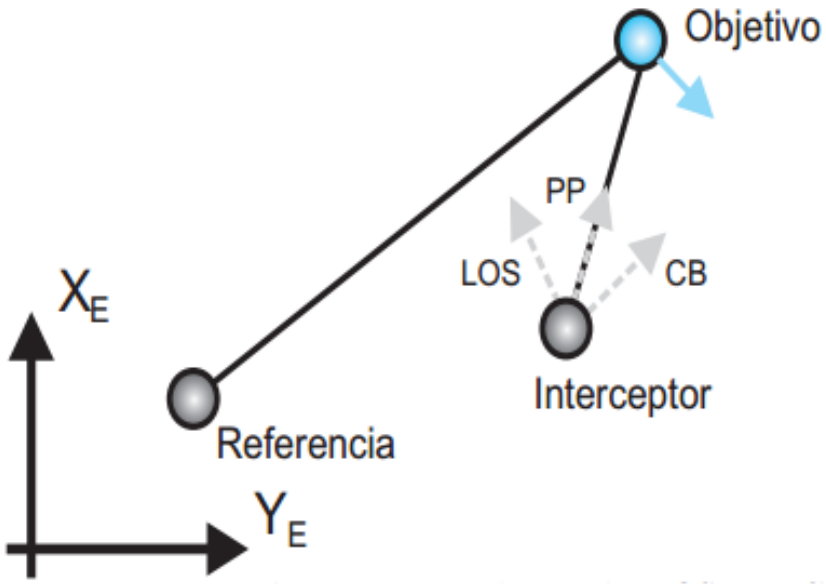

Figura 2-2 Estrategias de guiado LOS, PP y CB.

Si se considera un camino en línea recta definido al menos por dos puntos, uno de referencia y uno como punto siguiente, donde el eje *x* del primero se rota un ángulo positivo respecto al origen de coordenadas inercial, entonces el error de guiado depende de este ángulo. Este error se compone por el error de seguimiento a lo largo del camino y el error de seguimiento perpendicular al camino. En aquellos escenarios donde solo son consideradas las restricciones espaciales es de principal interés hacer nulo el error de seguimiento perpendicular al camino, asegurando la convergencia del vehículo a la trayectoria. Para lograr esto resulta necesario el empleo de una de las dos leyes de dirección (Breivik & Fossen, 2008):

- Ley de dirección basada en el encierro circular del camino.
- Ley de dirección basada en la distancia de la proyección de la posición del vehículo sobre el camino, conocida por su nombre en inglés *lookahead.*

Cuando los caminos no describen una línea recta pueden utilizarse varias técnicas de parametrización con el propósito de utilizar estas leyes.

A la hora de abordar el problema del seguimiento de trayectorias pueden considerarse dos alternativas. La primera consiste en separar el guiado del vehículo y su control de dirección en un lazo de control externo y otro interno respectivamente. La segunda alternativa consiste en utilizar una aproximación integral para diseñar un único elemento de control que desempeñe ambas funciones de guiado y control de dirección en la aeronave. En muchas de las aplicaciones de vuelo más actuales se utiliza la alternativa de lazos interno y externo, dado que existen numerosos métodos simples de implementar y eficientes para el desarrollo de este tipo de controladores. En este tipo de estrategia comúnmente se ajustan controladores de tipo *P* y *PD* para el control del error perpendicular al camino. Si la trayectoria es parecida a una línea recta se puede alcanzar un rendimiento muy elevado. No obstante, cuando la tarea requiere de un seguimiento de camino a través de trayectorias complejas, este tipo de estrategia puede no ser suficiente para alcanzar el rendimiento deseado y conviene entonces implementar una estrategia de control que implemente un controlador de posición y un controlador de dirección de la aeronave simultáneamente (Park, Deyst, & P. How, 2010).

Este es el caso de algoritmos de guiado tales como *I-LOS* y *L1*. Como se menciona en el epígrafe 1.6, el algoritmo de guiado *L1* se encuentra implementado en las distribuciones oficiales de los *firmwares* de *ArduPlane*, mostrando un buen desempeño en las aplicaciones de *UAV* desarrolladas a partir de esta plataforma (ArduPilot, 2016).

Este algoritmo se utiliza a partir de una trayectoria descrita por puntos ordenados según el movimiento deseado del vehículo. De esta forma, seleccionando un punto de referencia en la trayectoria deseada, se puede generar un mando de aceleración lateral α utilizando dicho punto de referencia. Para esto se hace necesario conocer la distancia *L1* existente desde el vehículo hasta el punto de referencia y el ángulo η conformado entre el vector *V* de la velocidad del vehículo y la distancia *L1*, como se observa en la *Figura 2-3*.

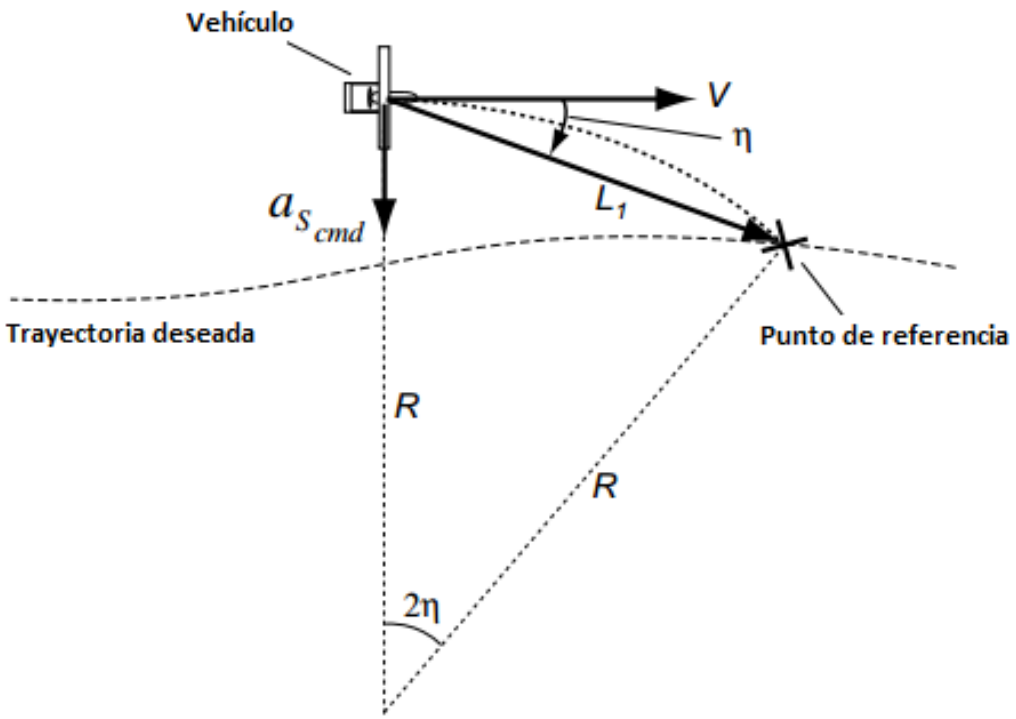

Figura 2-3 Diagrama de parámetros de L1.

A partir de estos parámetros se puede calcular el valor de la aceleración lateral mediante el empleo de cálculos matemáticos trigonométricos, si se conocen las coordenadas de los puntos de referencia anterior y siguiente de la trayectoria.

Linealizando esta lógica no lineal de guiado se obtiene un controlador *PD* para el error perpendicular al camino. La proporción entre la velocidad del vehículo *V* y la distancia al punto de referencia *L1* es un factor determinante en las ganancias de los controladores proporcional y derivativo. La distancia de separación puede ser seleccionada llevando a cabo un análisis de estabilidad con el modelo lineal de la planta y el controlador lineal sintetizado. El modelo de la planta debe incluir la dinámica del vehículo con el controlador del ángulo de banqueo de la aeronave y cualquier dinámica de los sensores en el lazo de medición.

Es importante resaltar que los resultados satisfactorios de *L1* se deben principalmente a que el ángulo η, conformado entre el vector *V* de la velocidad del vehículo y la distancia *L1*, provee una corrección de cabecera. Para pequeñas variaciones en la trayectoria provee un

controlador *PD* para el control del error perpendicular al camino y una señal de mando anticipatoria para el seguimiento de trayectorias curvas. Este esquema de control a través de una trayectoria descrita por puntos, donde el controlador minimice el error de seguimiento perpendicular al camino, es similar al empleado en el algoritmo de guiado *I-LOS* descrito a continuación.

2.3 Estudio sobre el algoritmo de guiado *I-LOS*

En esta investigación se supone la trayectoria a recorrer por el *UAV* como el conjunto de segmentos de líneas rectas establecidos entre dos puntos: referencia y siguiente, donde el eje *x* del primer punto ha sido rotado un ángulo positivo respecto al origen de coordenadas inercial, de esta forma el error de guiado depende únicamente de este ángulo. Además, se dan las condiciones para la utilización de una estrategia basada en la distancia *lookahead.*

Muchas son las investigaciones realizadas en torno a la temática del guiado. En los inicios era común que este se utilizara bajo la suposición de la no existencia de perturbaciones externas que afecten el vehículo (Lekkas, 2014). En la realidad esto rara vez ocurre, dado que en la práctica los *UAV* siempre se encuentran afectados por perturbaciones tales como la presencia del viento. Precisamente para dar solución a problemas de este tipo surge el algoritmo de guiado *I-LOS*.

En este algoritmo se incorpora una acción integral en el controlador proporcional *LOS* para reducir el error de seguimiento provocado por la acción de perturbaciones que tienden a llevar al vehículo lejos de la trayectoria deseada. En la bibliografía especializada se aprecia que esta técnica tiene una gran actualidad. Existen varias investigaciones recientes que reportan su uso en distintos tipos de vehículos autónomos (Elfes, 2003) (Caharija, 2014). Algunos autores (Breivik & Fossen, 2008) plantean la idea de utilizar un controlador de seguimiento tipo *PI*, como parte de la ley de guiado, realizando el ajuste del mismo a partir de la distancia *lookahead.* Esta estrategia ha sido probada con éxito por el GARP en el vehículo subacuático *HRC–AUV* (Hernández Julián, 2014) y en el *UAV* avión *N606LS* (Ortega Escudero, 2015).

Si se considera el punto de referencia como $\boldsymbol{P_k} = [y_k, x_k]^T \epsilon \mathbb{R}^2$ y el punto siguiente como $\boldsymbol{P_{k+1}} = [y_{k+1}, x_{k+1}]^T \epsilon \mathbb{R}^2$ y se traza un segmento de línea recta entre ellos, el objetivo sería

lograr que el vehículo siga esta línea recta con un error de seguimiento mínimo (Hagen, 2014). Considerando que el eje *x* del punto de referencia ha sido rotado respecto al sistema de referencia inercial del vehículo un ángulo positivo α_k de la forma:

$$\alpha_k = atan\left(\frac{y_{k+1} - y_k}{x_{k+1} - x_k}\right) \tag{2-1}$$

entonces las coordenadas cinemáticas del vehículo respecto al camino trazado pueden ser calculadas mediante la ecuación:

$$\boldsymbol{\varepsilon}(t) = \boldsymbol{R}(\alpha_k)^T(\boldsymbol{P}(t) - \boldsymbol{P}_k) \tag{2-2}$$

Siendo $\boldsymbol{R}(\alpha_k)$ la matriz que permite rotar la posición del vehículo con respecto al sistema de referencia ubicado en el camino:

$$\boldsymbol{R}(\alpha_k) = \begin{bmatrix} cos\alpha_k & -sen\alpha_k \\ sen\alpha_k & cos\alpha_k \end{bmatrix} \tag{2-3}$$

y $\boldsymbol{\varepsilon}(t) = [s(t),\ e(t)]^T \epsilon \mathbb{R}^2$ de forma tal que *s(t)* sea el error de seguimiento a lo largo del camino y *e(t)* el error de seguimiento perpendicular al camino, como se observa en la Figura 2-4.

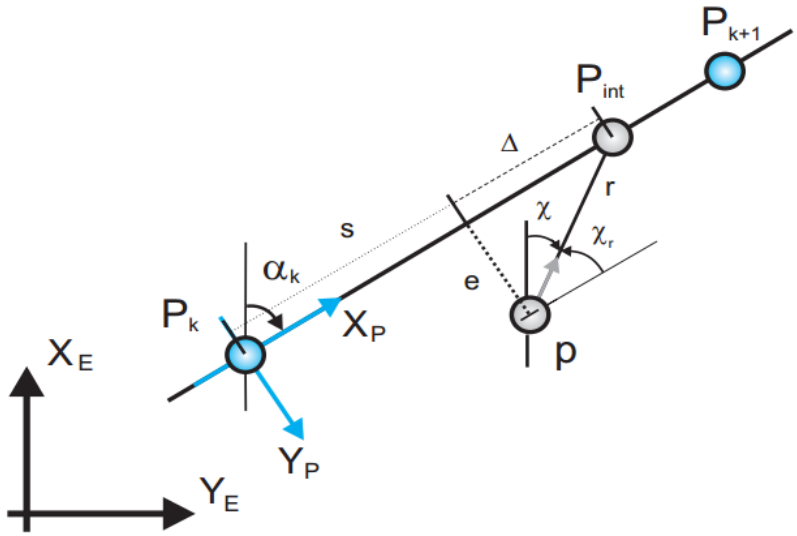

Figura 2-4 Esquema de variables del algoritmo I-LOS.

Dado que solo interesan las restricciones espaciales, el objetivo se centra en reducir el valor de *e(t)*, puesto que logrando que *e(t)* = 0 se asegura la convergencia del vehículo hacia la

trayectoria. Combinando entonces las ecuaciones 2-1, 2-2 y 2-3 se arriba a la expresión de *e(t)*:

$$e(t) = (y_t - y_k)cos\alpha_k - (x_t - x_k)sen\alpha_k \tag{2-4}$$

Por tanto el objetivo de control para el seguimiento de caminos en línea recta se transforma en $\lim_{t\to\infty} e(t) = 0$.

La estrategia basada en la distancia *lookahead* se emplea con el propósito de conducir al vehículo hacia el vector *LOS* a la vez que se disminuye el error perpendicular al camino. El vector *LOS* está orientado desde la aeronave hasta el punto donde intercepta con la trayectoria, este punto es situado en una línea tangencial al camino a una distancia *lookahead* Δ de la proyección de la aeronave sobre el camino (Lekkas, 2014). La distancia Δ es un parámetro que puede considerarse como variable o constante. Es conveniente considerar dicho parámetro como constante dado que esto simplifica en gran manera el proceso de síntesis del controlador. De igual forma se hace importante resaltar que el valor de Δ varía según la aeronave en que se implementa el algoritmo y debe ser proporcional al largo de la misma. Algunos autores plantean utilizar el doble de dicha longitud (Pettersen & Lefeber, 2001) (Fossen, 2002) (Lekkas & Fossen, 2012) mientras que otros proponen que sea *n* veces su valor (Healey, 2006).

Para eliminar entonces el efecto de las perturbaciones durante el seguimiento de trayectoria se emplea un controlador de tipo *PI* considerando el ángulo de deslizamiento lateral $\beta \approx 0$. De esta forma el ángulo de guiado $x_r(e)$ queda determinado por la ecuación:

$$x_r(e) = arctan\left(-k_p * e(t) - \int_0^t k_i e(T)dT\right) \tag{2-5}$$

Donde $k_p = \frac{1}{\Delta}$ y $k_i > 0$

Con la utilización de la acción integral es conveniente el uso de un método anti *wind-up* (Lekkas, 2014) que permita mantener los límites de salida del controlador en el rango concebido para $x_r(e)$. Para ello se propone un controlador *PI* cuya estructura matemática se obtiene a partir de redefinir la ecuación 2-5 de la forma:

$$x_r(e) = -arctan\left(k_p e(t) + k_i y_{int}\right) \tag{2-6}$$

donde $k_i = k * k_p$, siendo k un parámetro de diseño y $\dot{y}_{int}$ se define como:

$$\dot{y}_{int} = \frac{e\Delta}{\Delta^2 + (e + k * y_{int})^2} \tag{2-7}$$

Con la idea de eliminar el efecto *wind-up* se utiliza este controlador *PI* como el controlador *I-LOS* en el desarrollo de esta investigación.

Es importante destacar que actualmente los estudios comparativos sobre el rendimiento de los algoritmos *L1* e *I-LOS* son escasos en la bibliografía especializada. Es por ello que entre otras razones resulta de interés para el GARP una implementación de *I-LOS* sobre *ArduPlane*, el cual es un exponente significativo en el uso de *L1*, dado que esto facilitaría la realización de pruebas y simulaciones sobre la base de ambos algoritmos en una misma plataforma, lo que permite determinar sus diferencias y establecer cuál sería el más adecuado para determinados proyectos e investigaciones desarrolladas en el GARP.

2.4 Consideraciones finales del capítulo

En este capítulo se ha hecho una breve descripción de los principales conceptos, escenarios y estrategias que intervienen en el proceso de concepción y desarrollo de un algoritmo de guiado. Partiendo sobre este análisis se concluye que el algoritmo de guiado *I-LOS* logra la convergencia del vehículo sobre la trayectoria deseada a partir de reducir el error perpendicular al camino. Puede ser implementado en un sistema autopiloto a partir del conocimiento de las coordenadas de posición del vehículo, el punto de referencia siguiente y el punto de referencia anterior de la trayectoria.

CAPÍTULO 3. IMPLEMENTACIÓN DEL ALGORITMO DE GUIADO *I-LOS*

3.1 Introducción

En este capítulo se hace un breve estudio sobre la estructura de clases de la programación del *firmware* de *ArduPilot* y sus principales funciones. Posteriormente se aborda la implementación del algoritmo de guiado *I-LOS* en el autopiloto *ArduPlane* sobre la base de la misma, se explican las estructuras de los controladores, los principales bloques de funciones y artificios matemáticos utilizados en el proceso de implementación del algoritmo, así como el empleo de funciones pertenecientes a las bibliotecas de *ArduPilot* y el proceso requerido para la compilación de un *firmware* sobre el proyecto *ArduPlane*.

3.2 Estructura del *firmware* de *ArduPilot*

El código de *ArduPilot* se encuentra implementado en el lenguaje de programación *C++*, el mismo está constituido por una estructura modular donde las diversas funcionalidades del código se separan en clases orientadas a una tarea en específico. Estas clases se agrupan en directorios que conforman módulos o bibliotecas individuales nombradas según la función que desempeñen dentro del sistema en general.

Dichos módulos se constituyen de clases con funciones de entrada o inicialización de parámetros, funciones de salida o de operación y declaraciones de funciones, estructuras y variables globales. Esta estructura permite que se modifique la implementación de las funciones que integran un módulo determinado sin que esto implique un funcionamiento incorrecto de este u otros módulos, siempre y cuando se respeten las declaraciones y los propósitos de dichas funciones. Por lo anterior, el funcionamiento de un módulo es independiente de la implementación interna de otro, solo se requiere que se respeten las funcionalidades con las cuales se estructura el sistema y el propósito para el cual se utiliza el módulo.

Actualmente la programación del *firmware* de *ArduPilot* cuenta con 69 módulos. Los mismos se ubican dentro del directorio *libraries* en el directorio raíz de *ArduPilot* y, dentro de *libraries*, en directorios nombrados de forma similar al módulo que contienen. La forma en la que se estructuran y utilizan dichos módulos está en dependencia del proyecto para el que se compile la aplicación de *ArduPilot* (ArduPilot, 2016). En esta investigación se compila el *firmware* para el proyecto *ArduPlane* que, como se menciona en el epígrafe 1.6, es el proyecto de *ArduPilot* enfocado en el desarrollo de autopilotos para aviones. A continuación, se enuncian algunos de los principales módulos que intervienen en el desarrollo de esta investigación:

AP_Common. En este módulo se implementan definiciones y rutinas comunes para los restantes módulos de *ArduPilot*. Basa su funcionamiento en las bibliotecas estándar del lenguaje *C++*: stdint, *stdbool* y *stdlib.* En el mismo se declaran varias estructuras y funciones de propósito general.

AP_Math. Este módulo implementa una serie de funciones y variables declaradas globales las cuales están relacionadas con las operaciones matemáticas más comunes en el campo de los autopilotos y la navegación. Basa su funcionamiento en las bibliotecas *math* y *stdint* del estándar *C++* y los módulos *AP_Common* y *AP_Param.* Incluye además definiciones útiles referentes a los valores de las constantes matemáticas empleadas comúnmente en los cálculos relacionados con los sistemas de posición y algoritmos de guiado.

AP_AHRS. Este módulo implementa el Sistema de Referencia de Rumbo de Posición, conocido por sus siglas en ingles *AHRS* (*Attitude Heading Reference System).* El mismo se encarga de procesar las mediciones relacionadas con la posición y el rumbo provenientes de los sensores a bordo de la aeronave a través de diversas funciones. Basa su funcionamiento en los módulos *AP_Compass, AP_Airspeed, AP_GPS, AP_InertialSensor, AP_Baro* y *AP_Param,* que se encargan de lo referente al manejo y configuración de los sensores.

AP_Scheduler. Este módulo se encarga de implementar el planificador de tareas del pequeño sistema operativo que controla el funcionamiento interno de la placa donde se escribe el *firmware* de *APM Plane*. Sus funciones se ejecutan a una frecuencia aproximada de 50*Hz*.

AP_Mission. Este módulo se encarga de manejar todo lo referente al control de la misión en la aeronave. Es el responsable de manejar la lista de comandos que se ejecutan durante el

transcurso de la misión, leer y escribir los comandos de la misión hacia el almacenamiento interno de la placa, proveer un acceso a los puntos de navegación anterior, actual y siguiente, invocar los comandos principales del programa y verificar su función. Basa su funcionamiento en los módulos *AP_Math, AP_Common, AP_Param, AP_AHRS* y *AP_HAL*. Dentro de su implementación incluye una serie de definiciones para la configuración de la misión.

AP_Navigation. Este módulo implementa una serie de funciones virtuales con el objetivo de brindar una interfaz para el desarrollo de algoritmos de guiado y de navegación en general, compatibles con la estructura de los autopilotos de *ArduPilot.*

AP_L1_Control. Este es el módulo que se encarga de la implementación del algoritmo de guiado, en este caso *L1*, junto con un conjunto de controles auxiliares para apoyar el funcionamiento del algoritmo y el desempeño en tiempo de vuelo. El mismo utiliza las funciones provistas por los módulos *AP_Math, AP_AHRS, AP_Param y AP_NAvigation.* En este módulo se implementa la clase ***AP_L1_Control,*** la cual encapsula el algoritmo de guiado junto con otras funcionalidades auxiliares.

El autopiloto implementado en el proyecto *ArduPlane* posee varios modos de vuelo, entre ellos los más utilizados son:

- **Manual.** En este modo el usuario tiene control total sobre la aeronave y no interviene ninguna lógica de guiado o de estabilización del autopiloto. La aeronave modifica su posición únicamente a través de los comandos de la estación en tierra o la señal de radiofrecuencia del control remoto.
- **FBWA.** En este modo el autopiloto a bordo de la aeronave ejecuta algoritmos de control sobre el cabeceo y el alabeo, en aras de mantener dichos movimientos en un rango estable. El operador puede modificar el rumbo de la aeronave mediante los comandos de la estación en tierra y el control remoto, al mismo tiempo que se ejecutan los controladores de estabilidad de la aeronave.
- **Automático.** En este modo el sistema autopiloto del *UAV* ejecuta un control total sobre el movimiento de la aeronave, en respuesta al algoritmo de guiado implementado y la trayectoria trazada. El usuario puede utilizar los comandos de la estación en tierra o el control remoto para ejecutar acciones sobreponiéndose a la

lógica de guiado y, una vez finalizada su operación, el *UAV* continúa su operación de forma normal atendiendo a dicha lógica.

En la Figura 3-1 se observa el diagrama general de casos de uso que describen los modos de funcionamiento del autopiloto mencionado anteriormente.

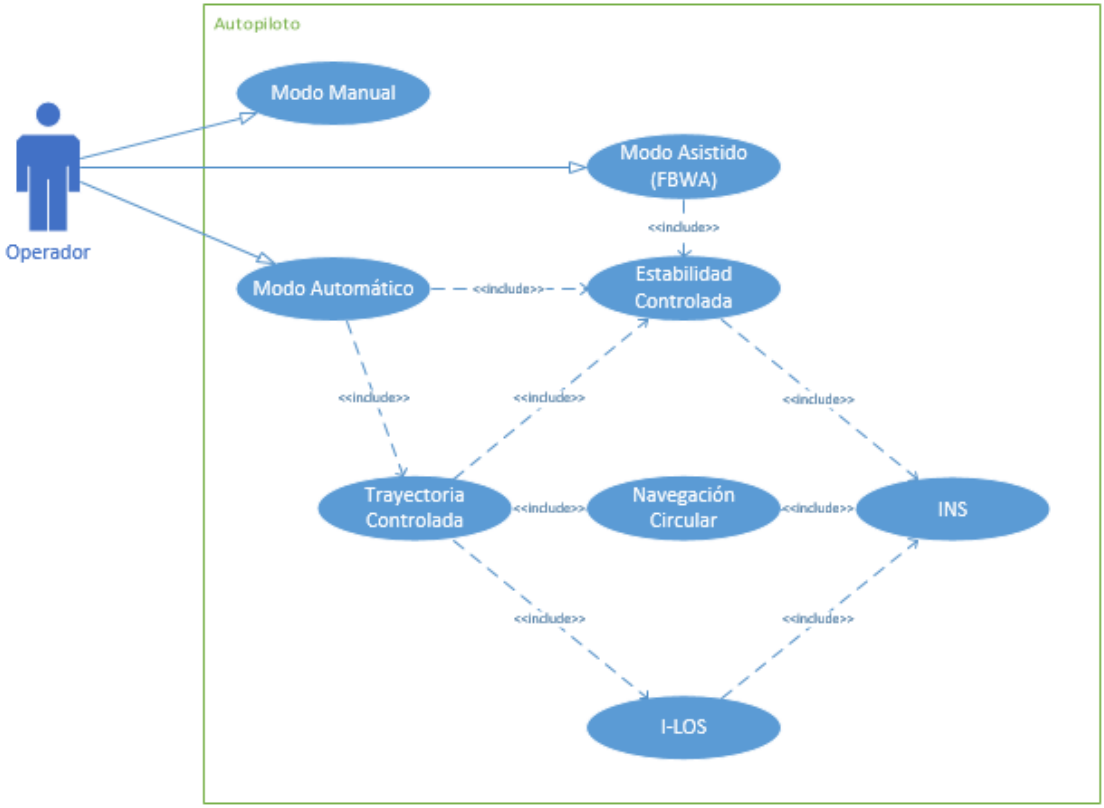

Figura 3-1 Casos de uso de los modos de vuelo de ArduPlane.

3.3 Normalización de la medición de posición

Como se observa en el epígrafe 2.3 el controlador *I-LOS* depende principalmente de la variación del error de seguimiento perpendicular al camino *e(t)*. El mismo está determinado por la variación en las coordenadas *(x, y)* partiendo del punto donde se encuentra el vehículo al denominado punto siguiente de referencia, lo que se entiende también como las distancias en los ejes *(x, y)* desde el vehículo al punto siguiente. En las aplicaciones sobre *ArduPilot* se utiliza un sensor *GPS* para la medición de posición a bordo de la aeronave, cuyos parámetros

de salida son las coordenadas de latitud y longitud de la posición instantánea del vehículo. (ArduPilot, 2016).

En algunas aplicaciones es posible ajustar las ganancias de los controladores para utilizar las variaciones de longitud y latitud pertenecientes al vehículo y el punto siguiente como las variaciones en las coordenadas de posición (*x, y)* respectivamente. Esta aproximación sirve únicamente cuando no se requiere de mucha exactitud, dado que la Tierra no describe una esfera perfecta, sino que es achatada por los polos y abultada por el ecuador. Esto provoca que las variaciones de las coordenadas de latitud y longitud no se comporten de forma lineal.

Para dar solución a este problema se utilizan las fórmulas de *Haversine* para el cálculo de la distancia exacta en unidades de metros entre dos puntos a partir de los valores de latitud y longitud de ambos. Las mismas se definen como:

$$a = sen\left(\frac{\varphi_2 - \varphi_1}{2}\right)^2 + cos\varphi_1 * cos\varphi_2 * sen\left(\frac{\lambda_2 - \lambda_1}{2}\right)^2 \tag{3-1}$$

$$c = 2 * atan\left(\frac{\sqrt{a}}{\sqrt{1-a}}\right) \tag{3-2}$$

$$D = R * c \tag{3-3}$$

donde φ_1 y φ_2 se definen como las latitudes y λ_1 y λ_2 como las longitudes de los puntos actual y siguiente respectivamente, *R* es el radio medio de la Tierra dado en metros y *D* la distancia entre ambos puntos.

Para el cálculo de la distancia en el eje *x* se asume que ambos puntos están en la misma coordenada de latitud φ de la posición actual del vehículo ($\varphi = \varphi_1 = \varphi_2$). Utilizando la ecuación 3-1 el cálculo del parámetro *a* se reduce entonces a la forma:

$$a = cos\varphi^2 * sen\left(\frac{\lambda_2 - \lambda_1}{2}\right)^2 \tag{3-4}$$

Para el caso del cálculo de la distancia en el eje *y* se asume que ambos puntos están en la misma coordenada de longitud ($\lambda_1 = \lambda_2$) y el parámetro *a* queda de la forma:

$$a = sen\left(\frac{\varphi_2 - \varphi_1}{2}\right)^2 \tag{3-5}$$

Tomando las fórmulas anteriores se implementan dos funciones para el cálculo de las distancias en los ejes (*x, y)* dentro de la clase *AP_L1_Control*. La primera se nombra como c*omputeX* y se encarga de calcular la distancia desde la aeronave hasta el punto siguiente sobre el eje *x* basándose en la ecuación 3-4, mientras que la segunda se nombra como *computeY* y se encarga de calcular la distancia desde la aeronave hasta el punto siguiente en el eje *y* utilizando la ecuación 3-5.

3.4 Implementación del controlador *I-LOS*

Como se menciona en el epígrafe 3.2 la implementación del algoritmo de guiado se desarrolla dentro la clase *AP_L1_Control*. Esta clase hereda de la clase *AP_Navigation* la cual sirve como una interfaz que provee métodos virtuales que le dan una estructura normada al algoritmo de guiado que se implementa. De esta forma, el programador no tiene que implementar el cuándo y dónde se ejecuten las funciones pertenecientes al algoritmo de guiado, sino implementar su algoritmo en la estructura ya existente dentro de *ArduPlane* para hacerlo compatible con el resto de las funcionalidades del código.

En esta implementación *AP_L1_Control* se mantiene con el código original presente en el *firmware* oficial de *ArduPilot* y se modifican únicamente las funciones *update_waypoint* y *nav_roll_cd*, implementadas en el archivo *AP_L1_Control.cpp*, que son las encargadas de lo referente al desarrollo del algoritmo de guiado *I-LOS*, de esa forma, las restantes funcionalidades del autopiloto, tales como el vuelo en círculo y la detección de puntos vencidos, se mantienen con su implementación original. Además, se le agregan una serie de variables y funciones privadas en su definición en el archivo de cabecera *AP_L1_Control.h* que son utilizadas en el nuevo desarrollo de las funciones modificadas tales como las funciones *computeX* y *computeY*, mencionadas en el epígrafe anterior. En la Figura 3-2 se puede observar el diagrama *UML* de clases de la misma.

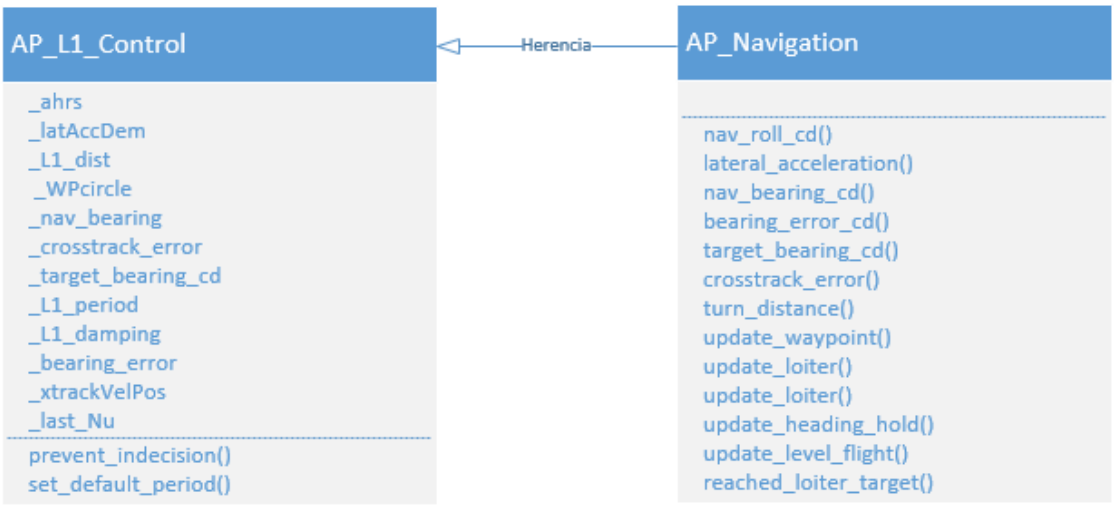

Figura 3-2Diagrama UML de la clase AP_L1_Control.

La función *update_waypoint* es invocada a través del planificador de tareas de *ArduPilot* a una frecuencia de 50*Hz*. La misma implementa el algoritmo de guiado presente en el sistema y calcula la señal de mando que determina el comportamiento de los lazos de control que manipulan los actuadores de la aeronave. En esta investigación se elimina totalmente la implementación original dada en *ArduPilot* y se sustituye por una implementación que contiene la lógica del algoritmo de guiado *I-LOS*.

Al inicio de la función se leen los datos de la posición de la aeronave y las coordenadas de los puntos anterior y siguiente mediante las funciones del módulo *AP_AHRS*. Posteriormente se analizan dichos valores para determinar si la aeronave se encuentra en el inicio de un nuevo segmento de la trayectoria o recorriendo el mismo de la llamada anterior de la función. En caso de que se encuentre recorriendo un nuevo segmento de la trayectoria se calculan algunos valores que son constantes para cada segmento, como el ángulo de corrección α_k , la longitud del segmento, o sea la distancia entre el punto anterior y el punto siguiente, los valores del seno y el coseno del ángulo de corrección entro otros. Dado que estos valores son constantes para cada segmento de la trayectoria y su cálculo requiere del empleo de funciones trigonométricas que suponen un alto costo computacional para el autopiloto, es más eficiente calcularlos una vez y almacenarlos en la memoria ante la alternativa de calcularlos en cada iteración de la función.

En caso de que no se inicie un nuevo segmento de trayectoria o se haya finalizado los cálculos correspondientes al inicio de uno, se procede entonces a condicionar las mediciones de la posición utilizando las funciones *computeX* y *computeY*, enunciadas anteriormente, y calcular los valores del controlador *I-LOS* mediante las ecuaciones 2-7, 2-6 y 2-4. En la Figura 3-3 se observa el diagrama de flujo de esta función.

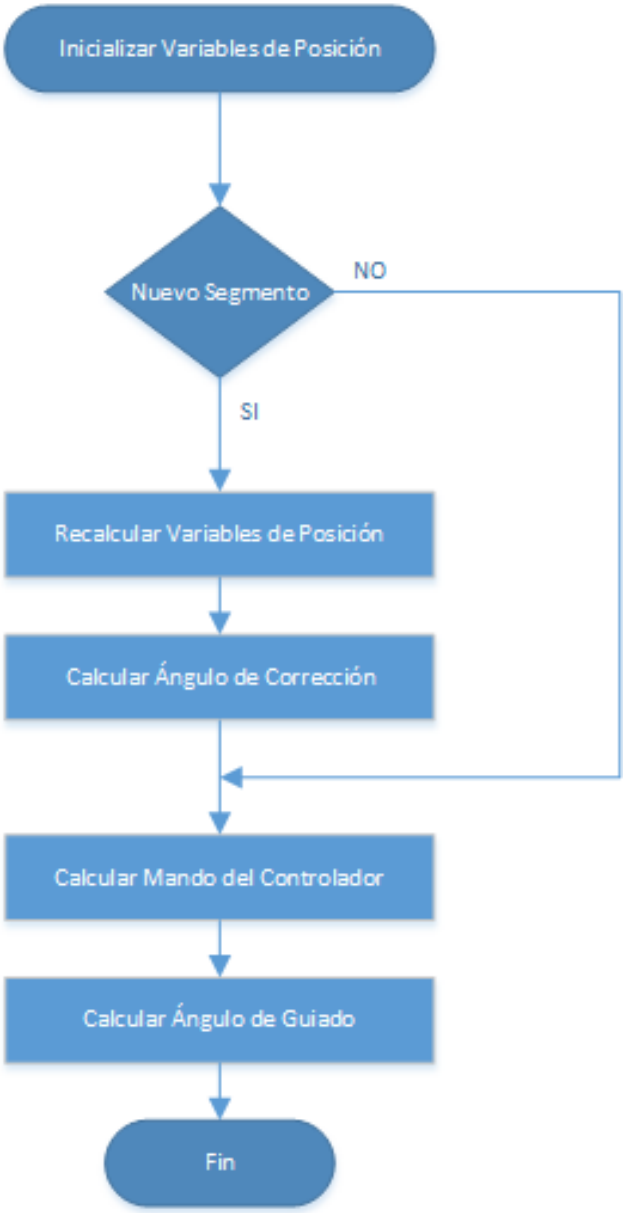

Figura 3-3 Diagrama de flujo de la función update_waypoint.

3.5 Implementación del controlador de guiñada

Las aeronaves son capaces de rotar alrededor de los tres ejes imaginarios perpendiculares entre sí cuyo punto de intersección se localiza sobre el centro de gravedad de la misma. El eje que va desde una punta a la otra de las alas del avión se le conoce como el eje lateral o transversal y el movimiento que realiza el avión alrededor de este eje se denomina cabeceo. El eje que se extiende desde el morro hasta la cola del avión se denomina eje longitudinal y el movimiento que realiza el avión sobre el mismo se conoce como alabeo o balanceo. Finalmente, el eje que, pasando por el centro de gravedad del avión, es perpendicular a los ejes transversal y longitudinal, contenido en un plano que pasa por el centro de gravedad desde arriba hacia abajo, se denomina eje vertical y el movimiento que realiza el avión sobre el mismo se define como guiñada. En la Figura 3-4 se observan los tres posibles movimientos de un avión sobre dichos ejes.

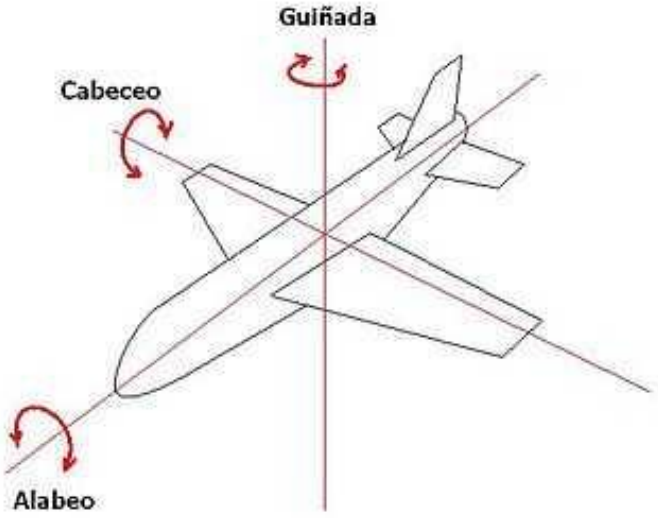

Figura 3-4Movimiento de un avión.

El desplazamiento de un avión en tiempo de vuelo se determina principalmente por el alabeo y el cabeceo, siendo el cabeceo el principal factor en el movimiento vertical del avión y el alabeo en el movimiento horizontal.

El controlador *L1* implementado en *ArduPlane* expresa su salida en términos del ángulo de alabeo requerido para que el avión converja hacia la trayectoria trazada, posteriormente en el

código de *ArduPlane* se implementa un controlador de alabeo que utiliza este ángulo para generar una señal de mando a los actuadores del *UAV*, siguiendo la dirección del mismo. De esta forma, controlando el alabeo se logra que la aeronave se desplace siguiendo la trayectoria deseada. En la Figura 3-5 se observa el esquema de control de la implementación del autopiloto *ArduPlane* con el controlador *L1*.

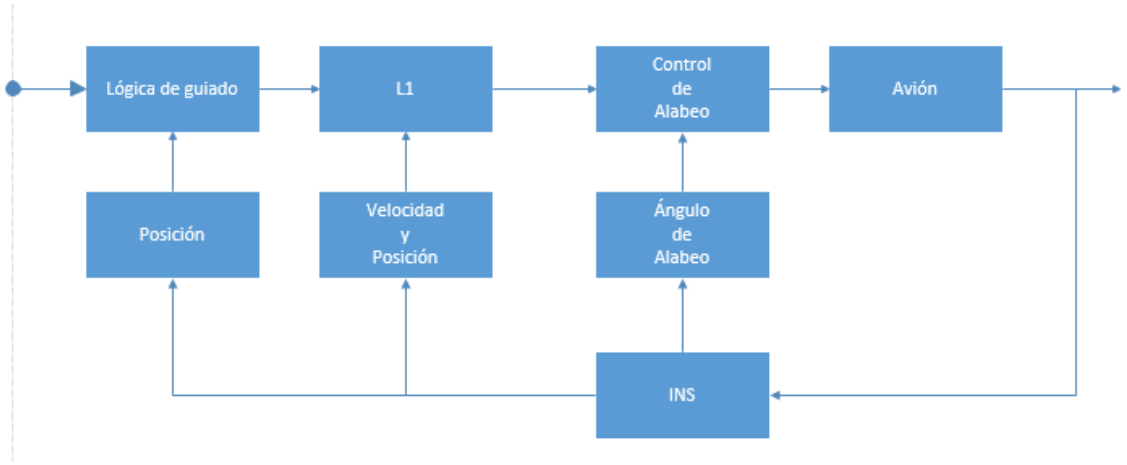

Figura 3-5 Esquema de Control del autopiloto de ArduPlane con L1.

El controlador *I-LOS* expresa su salida en el ángulo de guiado que debe adoptar el avión para converger hacia la trayectoria, este ángulo se corresponde con el ángulo de guiñada en algunos vehículos terrestres y marinos, así como en aeronaves. Dado que el movimiento en tiempo de vuelo de un *UAV* no puede determinarse por la guiñada, sino que se determina por el ángulo de alabeo, se requiere entonces de un controlador adicional que convierta los valores del ángulo de guiado del controlador *I-LOS* a los valores del ángulo de alabeo necesario para llevar la aeronave hacia la dirección requerida, el cual a su vez sirve como parámetro de entrada al controlador de alabeo, que genera la señal de mando a los actuadores del *UAV*. En la Figura 3-6 se muestra el esquema de control de la implementación del nuevo autopiloto con el algoritmo de guiado *I-LOS* implementado.

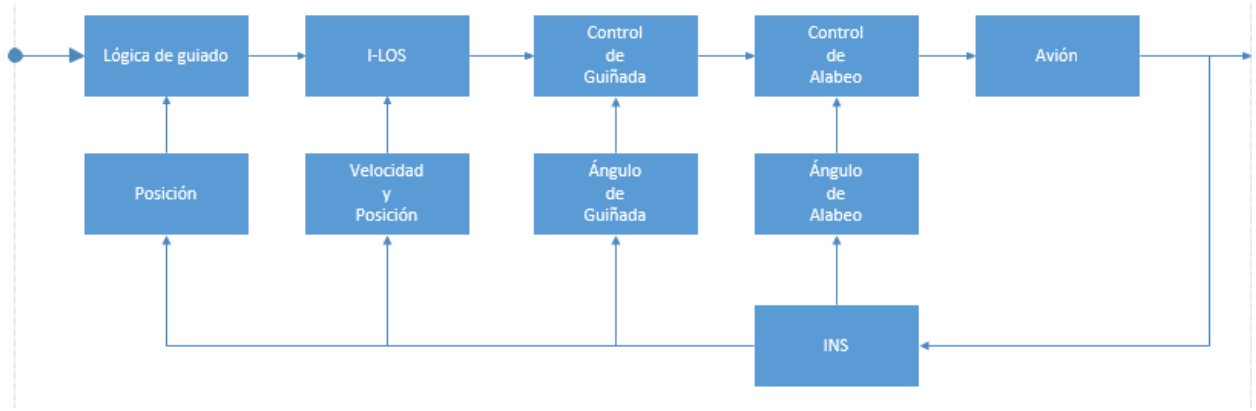

Figura 3-6 Esquema de Control de APM Plane con I-LOS.

El controlador de guiñada se implementa en la función *nav_roll_cd* perteneciente a la clase *AP_L1_Control*. Esta función se encarga de calcular y devolver el ángulo de alabeo requerido para hacer a la aeronave seguir la trayectoria trazada. En la implementación interna del algoritmo de guiado *L1*, este controlador se implementa a través de un controlador proporcional *(P)*. Por tanto, por analogía con el algoritmo *L1*, en esta función se incluye un controlador proporcional para el control del ángulo de guiñada, con una salida en ángulo de alabeo en cascada con el controlador *I-LOS* dentro de la función *nav_roll_cd*. Este ángulo de alabeo calculado se utiliza entonces en el controlador de alabeo implementado en el resto del código de *ArduPlane*.

3.6 Compilación del *firmware* de *ArduPlane*

El código fuente de *ArduPlane* se encuentra en el directorio "*ArduPlane*", en el directorio raíz del código oficial de *ArduPilot*. Dentro de este directorio pueden encontrarse los archivos de funcionamiento y configuración que intervienen en el proceso de compilación de este proyecto. Entre los archivos principales se encuentran:

- ***APM_Config.h*.** Este archivo de cabecera contiene las definiciones de configuración para el *firmware* compilado. En el mismo se establecen parámetros tales como el modo de funcionamiento del autopiloto, los tipos de sensores a bordo de la aeronave, los protocolos de comunicación con la estación en tierra, las maniobras desarrolladas ante la pérdida de comunicación con la estación en tierra entre otros.

- ***Parameters.h.*** Este archivo de cabecera se utiliza para establecer los parámetros del *firmware* relacionados con los recursos de hardware y los modos de operación del autopiloto. Los desarrolladores de *ArduPilot* recomiendan nunca modificar los valores de este archivo (ArduPilot, 2016).
- ***ArduPlane.pde.*** Este es el archivo principal de *ArduPlane*. En él se definen los parámetros principales de funcionamiento del *firmware,* tales como los módulos de *ArduPilot* que se van a utilizar y la forma de utilizarlos, atendiendo a los archivos de configuración del proyecto, y establece las reglas principales para la generación del *firmware* de acuerdo a los recursos requeridos en dichos archivos.

Para generar el *firmware* se hace uso del software *ArduPilot-Arduino.* Este no es más que una distribución del *framework* oficial de *Arduino,* a la cual se le hace una serie de modificaciones en su código fuente para hacerlo compatible con los proyectos de *ArduPilot* y las placas que son compatibles con el mismo. El ambiente de trabajo del software es el mismo que en la distribución oficial de *Arduino* y mantiene la compatibilidad con las instrucciones, los archivos de configuración (*Makefile*) y las extensiones (*.pde*), que pueden encontrarse en un proyecto nativo de *Arduino* (ArduPilot, 2016).

Una vez terminada toda la implementación del software se accede al código de *ArduPilot* a través de *ArduPilot-Arduino,* y se selecciona el proyecto *ArduPlane* como objetivo de compilación. Esto desencadena un proceso de verificación de errores por parte del software *ArduPilot-Arduino* donde se comprueba la existencia e integridad de los archivos requeridos en el proceso de compilación y la integridad del código implementado en dichos archivos. Una vez finalizado este proceso el software genera un archivo con la extensión *'.hex',* nombrado según la fecha de compilación, conteniendo el código fuente que conforma el *firmware* compilado en formato hexadecimal listo para ser escrito en la memoria de una placa compatible con *ArduPlane*. En el desarrollo de esta investigación se emplea la versión 3.2.0 de *ArduPlane* y la versión 1.0.3 de *ArduPilot-Arduino.*

3.7 Consideraciones finales del capítulo

La estructura modular de *ArduPilot* permite la modificación de partes del código referentes al algoritmo de guiado implementado, sin que esto perjudique el correcto funcionamiento del resto del sistema y el proceso de compilación de un *firmware* con base en *ArduPlane*. Un controlador de tipo proporcional *(P)* constituye una alternativa suficiente para el desarrollo del controlador de guiñada del sistema autopiloto, como se observa en la implementación oficial del algoritmo de guiado *L1*. En *ArduPlane*, se encuentran implementadas varias funcionalidades relacionadas con el guiado, tales como la lógica de generación de trayectorias y la lógica de detección de punto vencido. Dichas funcionalidades facilitan el proceso de implementación de nuevas estrategias de guiado sobre la estructura de este proyecto de autopiloto.

CAPÍTULO 4. VALIDACIÓN DE LOS RESULTADOS MEDIANTE SIMULACIÓN

4.1 Introducción

En este capítulo se tratan los elementos referentes a la validación de los resultados de la investigación mediante el empleo de la simulación *HIL,* haciendo uso del *firmware* de *ArduPilot* compilado para el proyecto *ArduPlane* y las distintas herramientas de software y hardware utilizadas para el desarrollo de la simulación *HIL* y el ajuste de los controladores que intervienen en el sistema desarrollado.

4.2 Simulación *Hardware-In-The-Loop*

La simulación *Hardware-In-The-Loop,* comúnmente referida en la bibliografía por las siglas *HIL* o *HWIL,* es una técnica empleada en el análisis y desarrollo de controladores sobre sistemas empotrados complejos que funcionan en tiempo real. La misma se vale de los modelos matemáticos de los sistemas dinámicos que se relacionan con la planta bajo control, conformando lo que se conoce como simulación de la planta o planta virtual. Esto propicia una plataforma efectiva de simulación, dado que incluye toda la complejidad de la planta que controla el sistema empotrado. El controlador bajo análisis interactúa con esta "simulación de la planta" (Hwang, 2006).

La simulación *HIL* incluye la simulación electrónica de los sensores y actuadores que intervienen en el proceso desarrollado por la planta. Estas simulaciones sirven de interfaz entre el modelo de la planta virtual y el sistema empotrado real que se analiza. Los valores de los sensores se generan por la planta virtual de acuerdo a los valores típicos que pueden observarse en un proceso como el que se simula y el estado de las variables dentro del modelo de la planta virtual, estos a su vez son leídos por el controlador implementado en el dispositivo empotrado. Posteriormente el controlador ejecuta su lógica y transmite su señal de mando hacia los actuadores simulados dentro de la planta virtual. De esta forma, los cambios en las señales de control provocan cambios en los valores de las variables contenidas

en el modelo de la planta virtual. En la Figura 4-1 se observa un esquema de control típico de una simulación *HIL*.

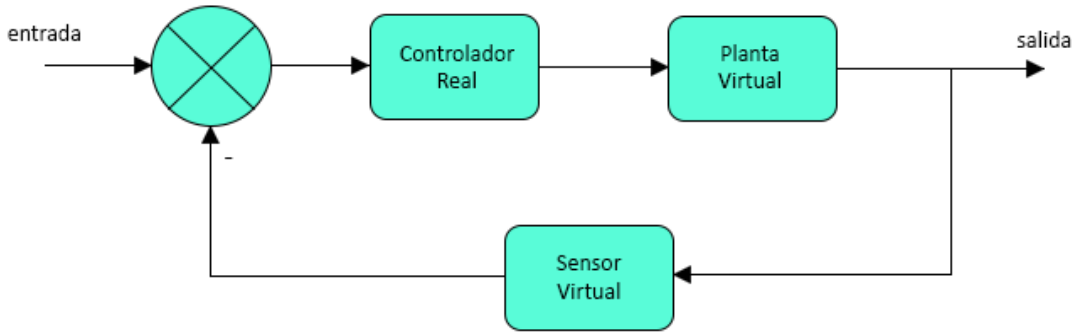

Figura 4-1 Esquema de control de simulación HIL.

En muchos casos la forma más efectiva de desarrollar y analizar el desempeño de un sistema de control empotrado es conectándolo a la planta real para la cual fue diseñado, pero en algunos casos la simulación *HIL* resulta en una alternativa más viable. Para seleccionar el método más adecuado se debe tener en cuenta cuestiones tales como el costo del proyecto en desarrollo, la fecha límite de entrega, la seguridad del proceso y la viabilidad del método empleado, entre otros.

La simulación *HIL* normalmente aumenta la calidad del proceso de análisis y ajuste, incrementando el campo de prueba del sistema analizado. Muchas de las plantas reales que se analizan imponen limitaciones en cuanto a las pruebas que pueden ser realizadas sobre las mismas. Por ejemplo, en el proceso de experimentación de la unidad de control de un motor, la planta real no puede utilizarse en un rango mayor a los parámetros nominales de operación del motor o en condiciones de fallas sin arriesgar la integridad del sistema. No obstante, la simulación *HIL* permite analizar el sistema en cualquier rango y condición de funcionamiento.

Las apretadas fechas de entrega que caracterizan a los proyectos aeroespaciales, militares y de automática, por lo general no permiten posponer el proceso de verificación y análisis de los controladores del sistema hasta la creación de un prototipo viable para tal fin. En muchos

de estos proyectos la simulación *HIL* se emplea en paralelo con el desarrollo de la planta para la cual se diseña el controlador que se simula, lo que permite a su vez reunir información provechosa no solo para el desarrollo del controlador sino también para el desarrollo de la planta misma.

Dado que en el GARP no se cuenta con un arsenal de aeronaves disponibles para el proceso de desarrollo y simulación, las tareas relacionadas con la escritura de un *firmware* sobre una placa de autopiloto y su ensamble sobre la aeronave misma, normalmente son lentas y engorrosas, en esta investigación se emplea la simulación *HIL* como herramienta de simulación, lo que permite agilizar el proceso de revisión y validación de nuevos cambios en la programación del autopiloto y realizar experimentos sin peligro de dañar los medios técnicos empleados en la investigación.

4.3 Herramientas de software para la simulación

En el desarrollo de esta investigación se emplean diversas herramientas de software en aras de llevar a cabo la simulación del controlador *I-LOS* implementado mediante la técnica *HIL*.

El software *Mission Planner* es parte de estas herramientas. El mismo está incluido dentro de los proyectos oficiales de *ArduPilot*, como se menciona en el epígrafe 1.6. Se trata de una estación de control en tierra para el planeamiento y supervisión de misiones de vehículos no tripulados. Actualmente el mismo es compatible con los proyectos de *ArduPilot ArduPlane*, *Rover* y *ArduCopter*, para los que ofrece una interfaz de comunicación y control que permite acceder a las distintas opciones implementadas en estos proyectos (MissionPlanner, 2016). Desarrollado por Michael Oborne como parte de la suite de *ArduPilot* se distribuye bajo una licencia de software libre *GPLv3* (GNU, 2016). Basado en las bibliotecas *.Net* de *Microsoft*, es compatible con el sistema operativo *Windows* y puede instalarse en los sistemas *Linux* y *MacOS* si se instalan las correspondientes bibliotecas para los mismos.

Entre las principales funcionalidades que brinda este software se encuentran (MissionPlanner, 2016):

- Posee una interfaz para cargar *firmwares* de los proyectos *ArduPlane*, *Rover* y *ArduCopter* en las placas soportadas por *ArduPilot*. Estos *firmwares* pueden ser

descargados por el software directamente desde el servicio oficial de *ArduPilot* para la distribución de *firmwares* en internet, o seleccionados desde el almacenamiento interno de la PC donde se ejecute el programa.

- Posee una interfaz de configuración, ajuste y auto-ajuste de los controladores internos que rigen el desempeño de los vehículos.
- Permite planificar, salvar y recuperar misiones autónomas en los autopilotos soportados valiéndose de la información provista por el servicio global de mapas *Google Maps*.
- Proporciona una interfaz para conectarse con simuladores de vuelo tales como *Flight Gear* o *X-Plane* y llevar a cabo simulaciones *HIL* sobre las placas de *ArduPilot*.
- Permite monitorear el estado del vehículo durante la ejecución de la misión y cambiar parámetros de la misma, tales como los puntos que conforman la ruta establecida.
- Brinda la posibilidad de operar el vehículo en modo vista en primera persona *FPV (First Person View)*.

En la Figura 4-2 se observa la ventana principal de *Mission Planner*.

Figura 4-2 Ventana principal de Mission Planner.

En la zona izquierda de la ventana se observan las lecturas de las variables principales que intervienen en el vuelo de una aeronave, tales como la altitud, la velocidad relativa a tierra, el ángulo de dirección, la velocidad vertical, entre otras y un indicador de vuelo similar al que poseen muchas aeronaves reales. En la zona derecha se observa un mapa de la Tierra, en este caso el mapa provisto por los servicios de *Google Maps*, donde se observa la ubicación actual del vehículo y la trayectoria de la misión programada.

En la Figura 4-3 se observa la ventana de planificación de misión de *Mission Planner*.

Figura 4-3 Planificación de Misión en Mission Planner.

En esta figura se observa la ruta planificada descrita por los distintos puntos que la conforman, los cuales están numerados según el orden en que deben recorrerse. El punto designado como *Home* corresponde al lugar desde donde la aeronave despega y al cual debe regresar en caso de pérdidas de la comunicación y otras fallas, según la configuración establecida.

En la parte inferior de la ventana puede observarse la configuración de cada punto de trayectoria, donde se establecen valores esenciales tales como la latitud, la longitud y la altitud de cada uno, respectivamente.

Otra de las herramientas de software utilizadas es el simulador de vuelo *Flight Gear*. El mismo no forma parte de los proyectos asociados a *ArduPilot*. Distribuido bajo una licencia

de software libre *GPLv3* (GNU, 2016), goza de una gran comunidad de desarrolladores y de una alta popularidad entre los diversos software para la simulación de aeronaves. Puede utilizarse sobre los sistemas operativos *Windows, Linux* y *MacOS* (FlightGear, 2016).

Este software posee una interfaz que permite acceder y modificar los valores de las variables del modelo que rige el comportamiento de la aeronave simulada en tiempo de simulación. Esto permite llevar a cabo simulaciones *HIL* para controladores de *UAV*, utilizando este software como la planta simulada. En la Figura 4-4 se observa la ventana de simulación en ejecución de *Flight Gear*.

Figura 4-4 Simulación con Flight Gear.

4.4 Herramientas de hardware para la simulación

En aras de llevar a cabo la simulación *HIL* sobre el controlador *I-LOS* desarrollado en esta investigación, se necesita un dispositivo real sobre el cual se escriba el autopiloto implementado en el proyecto *ArduPlane.* Para ello se emplea un ejemplar de la placa *APM2* en su versión 2.5. Como se menciona en el epígrafe 1.6, esta placa es totalmente compatible con los autopilotos de *ArduPilot*.

Diseñada por el equipo de desarrolladores de *ArduPilot* para los proyectos *ArduPlane, ArduCopter* y *Rover*, la placa *APM2* puede obtenerse en un precio de 49.99 *USD*. La misma se distribuye totalmente ensamblada dentro de un pequeño chasis de plástico (MyRCMart, 2016). Entre sus principales características se encuentra (ArduPilot, 2016):

- Compatible con proyectos de *Arduino.*
- Posee un giróscopo y acelerómetro de 3 ejes.
- Posee una brújula digital de 3 ejes *HMC5883LTR* de *Honeywell.*
- Incluye un módulo *GPS uBlox NEO6M* a 5Hz.
- Posee un altímetro de alta resolución.
- Incluye un sensor de presión barométrica *MS561101BA03* de *Measurement Specialties*.
- Utiliza los microcontroladores *ATMEGA2560* y *ATMEGA32U2* de *Atmel* para el procesamiento de las tareas del autopiloto y la atención al puerto *USB* respectivamente.
- Sus dimensiones son de 70.5x45x13.5mm y tiene un peso de 31g, lo que la hace ideal para aplicaciones ligeras de dimensiones pequeñas.

En la Figura 4-5 puede observarse la estructura de la misma junto con sus puertos de entrada y salida, *GPS* y comunicación.

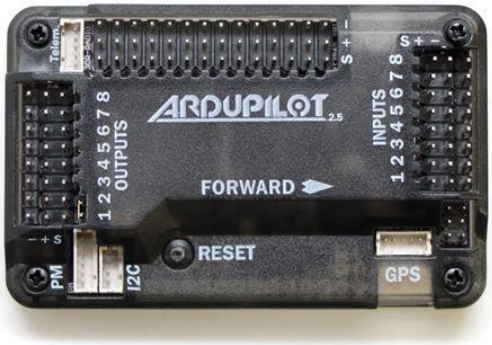

Figura 4-5Placa APM 2.5.

Además de la placa *APM2* se emplea también en esta investigación el controlador de aeronaves por radiofrecuencia *Aurora 9* de la compañía *Hitech*, Este controlador es compatible con el proyecto *ArduPilot* y puede ser empleado para controlar aeronaves desarrolladas sobre la base del proyecto *ArduPlane.* Entre sus principales características se encuentran:

- Comunicación a 2.4GHz compatible con *PPM, PCM* y *AFHSS*.
- Una interfaz personalizable y ajustable a diferentes aplicaciones de *UAV*.

- Limitador de escala ajustable desde 0.025° hasta 5° por paso.
- 9 canales de control asignables.
- Soporta los modelos de programación *ACRO*, *GLID* y *HELI*.
- Posee modos a prueba de fallos y de entrenamiento.

En la Figura 4-6 se observa la estructura de este controlador y los distintos controles e interruptores de los que dispone para el control de aeronaves.

Figura 4-6 Controlador Aurora 9.

4.5 Simulación del controlador *I-LOS* mediante simulación *Hardware-In-The-Loop*

En aras de desarrollar una simulación *HIL* con los elementos de software y hardware descritos en los epígrafes 4.3 y 4.4 respectivamente, se hace necesario compilar el *firmware* de *ArduPlane* en una configuración para funcionar en modo *HIL*. Esto se logra incluyendo en el archivo *APM_Config.h*, descrito en el epígrafe 3.6, las definiciones de configuración requeridas para una simulación *HIL*. Posteriormente se compila el *firmware* con esta nueva configuración y se escribe a través de una conexión *USB* sobre la placa *APM2* empleando la interfaz de escritura de *firmwares* del software *Mission Planner*.

La simulación *HIL* se lleva a cabo empleando la placa *APM2* cargada con el *firmware* de *APM Plane* como elemento principal de control, el software *Mission Planner* como el elemento generador de trayectoria o lógica de guiado, el software *Flight Gear* como la planta virtual sobre la cual se efectúa la acción de control y la interfaz de simulación de *Mission*

Planner como el elemento sensor de las variables de salida de la planta. En la Figura 4-7 se muestra el esquema general de la simulación *HIL* desarrollada en esta investigación.

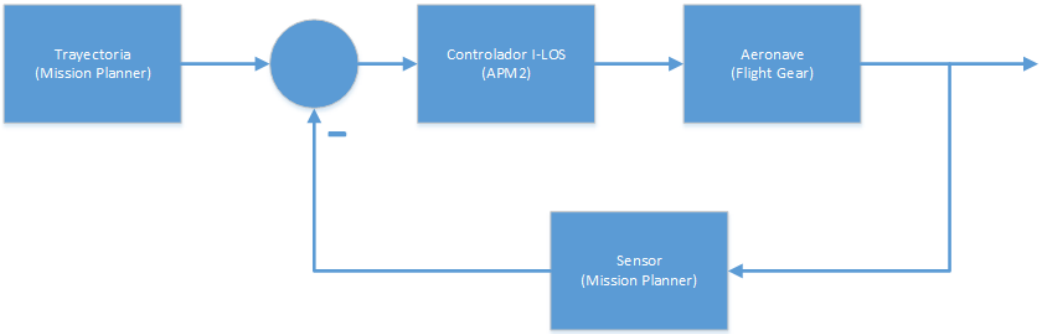

Figura 4-7 Esquema general de simulación HIL.

Para desarrollar esta simulación se conecta la placa *APM2* mediante conexión *USB* con el software *Mission Planner*, seguidamente se inicia una simulación *HIL* a través de la interfaz de simulación de este software, seleccionando el software *Flight Gear* como la planta virtual de la simulación. En este punto el software *Mission Planner* inicia una comunicación con el software *Flight Gear* mediante el protocolo de comunicaciones *MavLink*, intercambiando datos de la operación, tales como las variables de posición de la aeronave virtual, los puntos que componen la trayectoria, la distancia entre dichos puntos y la aeronave, la acción de control entre otros. Seguidamente transmite a través de la conexión *USB* los valores de las mediciones provenientes de *Flight Gear* hacia la interfaz de lectura de la placa *APM2* y se leen las acciones provenientes del controlador implementado en la misma, las cuales se envían a los actuadores virtuales presentes en el software *Flight Gear*. En la Figura 4-8 se observa el flujo de información entre la placa *APM2* y la aeronave virtual simulada en el software *Flight Gear*, mediante las interfaces de comunicación del software *Mission Planner*.

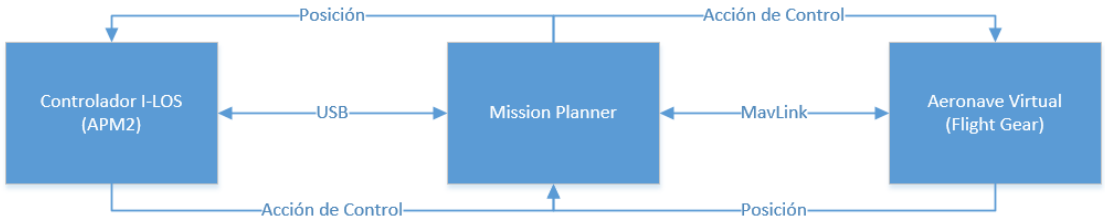

Figura 4-8 Flujo de Información en la simulación.

Con el objetivo de evaluar satisfactoriamente el funcionamiento de la estrategia de guiado *I-LOS* implementada, resulta necesario ajustar los parámetros del controlador *I-LOS* y el controlador de guiñada respectivamente. Dado que no se cuenta con el modelo matemático de la aeronave implementada en el simulador de vuelos *Flight Gear*, se hace muy complejo realizar el ajuste de estos controladores mediante métodos analíticos de ajuste y estabilidad, por esta razón en esta investigación el ajuste de los controladores se desarrolla a través de métodos experimentales.

En un primer momento se ajusta el controlador de guiñada. Con este objetivo se modifica la programación del controlador *I-LOS* en el autopiloto para dar una salida constante y predeterminada hacia el controlador de guiñada, ignorando toda la lógica del mismo. De esta forma, se establece un ángulo deseado y puede analizarse y ajustarse la respuesta del controlador de guiñada ante esta entrada. Para desarrollar estas operaciones se despega la aeronave virtual utilizando el modo manual del autopiloto y el controlador de aeronaves *Aurora 9*. Una vez que la aeronave se encuentra volando de forma estable se cambia al modo automático del autopiloto. Siguiendo esta estrategia se realizaron de forma iterativa varios experimentos con diferentes valores de la ganancia proporcional del controlador de guiñada y ángulo de guiado deseado, hasta alcanzar un desempeño adecuado en la salida del controlador. A continuación, se enuncian dos de estos experimentos.

Experimento 1 (Figura 4-9):

- Ángulo de guiñada: 180°
- Ganancia proporcional de guiñada: 0.4
- Altura: 100m.
- Sin perturbaciones.

Figura 4-9 Experimento 1 del ajuste del controlador de guiñada.

En la Figura 4-9 la línea púrpura sobre el mapa representa la trayectoria seguida por la aeronave durante el experimento. En la misma puede observarse como el controlador logra seguir de forma suficientemente precisa un ángulo de guiñada de 180°.

Experimento 2 (Figura 4-10**):**

- Ángulo de guiñada: 90°
- Ganancia proporcional de guiñada: 1
- Altura: 100m.
- Sin perturbaciones.

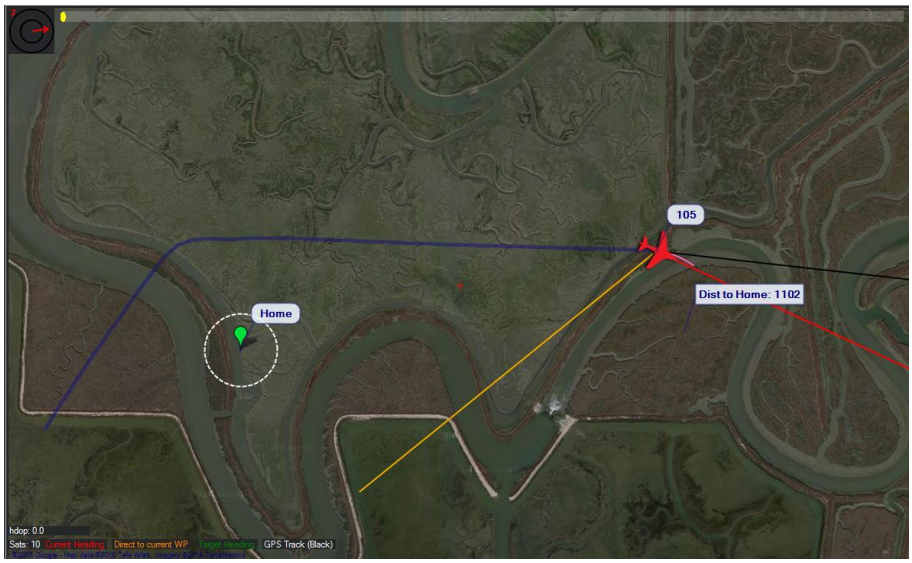

Figura 4-10 Experimento 2 del ajuste del controlador de guiñada.

Al igual que en el experimento anterior, en la Figura 4-10, la línea púrpura sobre el mapa representa la trayectoria seguida por la aeronave durante el experimento. En la misma puede observarse como el controlador con los parámetros de ajuste anteriores logra seguir de forma suficientemente precisa un ángulo de guiñada de 90°.

Tomando en cuenta los resultados de los experimentos realizados, se concluye que el controlador de guiñada implementado funciona de manera estable y acertada con un valor de ganancia proporcional de aproximadamente el valor unidad, para el modelo de aeronave implementado en el simulador de vuelos *Flight Gear*.

Partiendo sobre la base del ajuste realizado sobre el controlador de guiñada, se procede al ajuste del controlador *I-LOS*. Primeramente, se restaura su programación al modo de trabajo normal del autopiloto. Este ajuste se lleva a cabo siguiendo una estrategia similar a la empleada en el ajuste del controlador de guiñada. Para esto se definen en la interfaz de planeamiento del software *Mission Planner* un conjunto de trayectorias diferentes, las cuales se emplean en simulaciones *HIL* sobre este controlador. Mediante un proceso iterativo se varían los parámetros del controlador para cada trayectoria de acuerdo a los resultados obtenidos en cada experimento y se analiza su comportamiento. A continuación, se enuncian dos de estos experimentos.

Experimento 1 (Figura 4-11**):**

- Ganancia proporcional del controlador de guiñada: 1
- *Kp:* 15
- *Ki:* 0.4
- Altura: 100m
- Sin perturbaciones

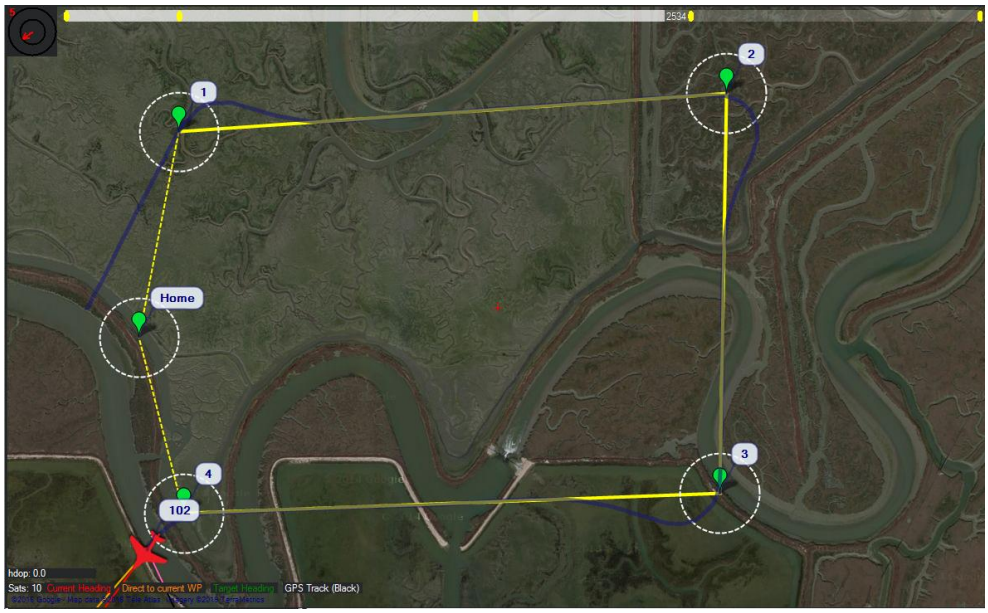

Figura 4-11 Experimento 1 del ajuste del controlador I-LOS.

En la Figura 4-11 se observa el desarrollo de este experimento. Las líneas amarillas representan la trayectoria planificada y la línea púrpura, la trayectoria seguida por la aeronave. En esta figura se observa como el controlador *I-LOS* con los ajustes anteriores logra seguir satisfactoriamente la trayectoria deseada con un error de seguimiento perpendicular al camino *e(t)* suficientemente pequeño para el modelo de la aeronave implementada en el simulador *Flight Gear*.

Experimento 2 (Figura 4-12**):**

- Ganancia proporcional del controlador de guiñada: 1
- *Kp:* 15
- *Ki:* 0.4
- Altura: 100m
- Con perturbaciones

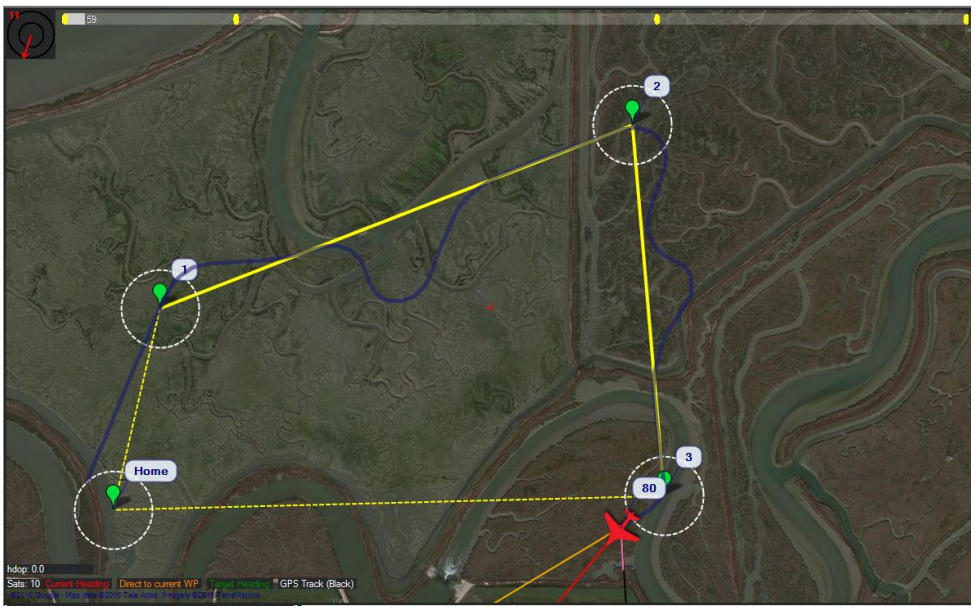

Figura 4-12 Experimento 2 del ajuste del controlador I-LOS.

En la Figura 4-12 se observa el desarrollo de este experimento. Las líneas amarillas representan la trayectoria planificada y la línea púrpura la trayectoria seguida por la aeronave. En esta figura se observa como el controlador *I-LOS* con los ajustes anteriores, logra recuperarse satisfactoriamente ante perturbaciones que lo desplacen fuera de la trayectoria deseada, con un error de seguimiento perpendicular al camino *e(t).*

Tomando en cuenta los resultados de los experimentos realizados se concluye que el controlador *I-LOS* implementado funciona de manera estable y suficientemente precisa con un valor de *Kp* aproximado de 15 y un valor de *Ki* aproximado de 0.4 para el modelo de aeronave implementado en el simulador de vuelos *Flight Gear*.

Con todos los controladores ajustados se llevaron a cabo simulaciones con diferentes trayectorias y condiciones de operación. Entre las mismas se emplearon trayectorias cuadradas, triangulares, en zigzag y describiendo figuras complejas. En cada una de estas pruebas el controlador *I-LOS* implementado con los ajustes anteriores, logró seguir las trayectorias trazadas con un mínimo de error en la posición de la aeronave. En la Figura 4-13 se muestra una de estas trayectorias trazadas y el correspondiente movimiento de la aeronave siguiendo la misma. Las líneas amarillas describen la trayectoria deseada y la línea purpura la trayectoria seguida por la aeronave.

Esto demuestra que el controlador *I-LOS* desarrollado funciona correctamente, puesto que logra seguir las trayectorias planificadas a través de la estación de control terrestre, con un error mínimo de seguimiento perpendicular al camino *e(t)* y se recupera ante el efecto de las perturbaciones, su rendimiento se condiciona de acuerdo a los ajustes dados al controlador. Además, se integra de manera adecuada con el resto de la programación del *firmware* de *ArduPlane* sin interrumpir el flujo de operación del autopiloto.

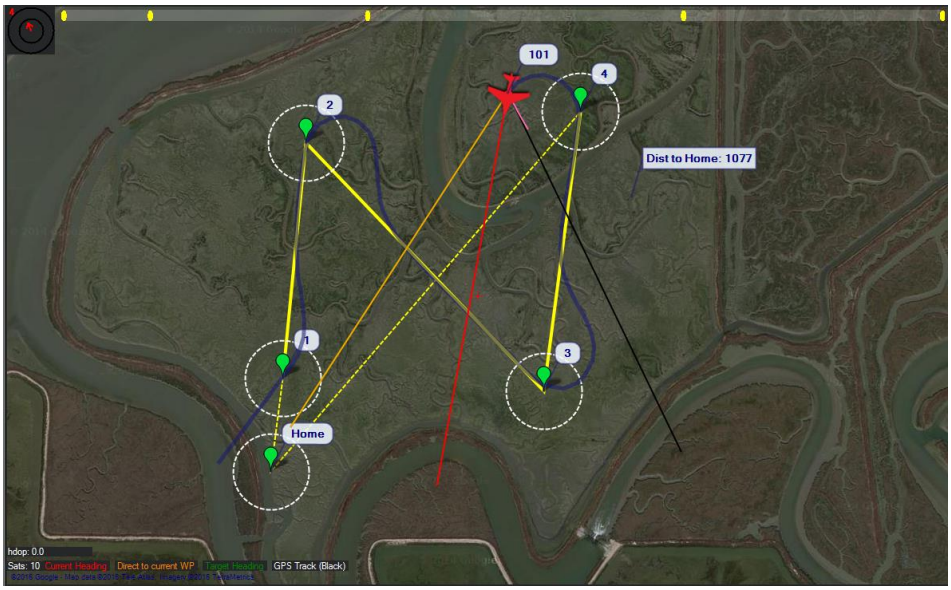

Figura 4-13 Movimiento de la aeronave bajo el controlador I-LOS.

4.6 Valoración Económica

En esta investigación se obtiene una implementación del algoritmo de guiado *I-LOS* sobre un sistema autopiloto perteneciente a la plataforma de desarrollo de *UAV ArduPilot.* Puesto que la misma, junto con todos los software y proyectos asociados a la investigación (*ArduPlane*, *Mission Planner*, *Flight Gear*) se distribuye como parte de la comunidad del software libre, el costo para la obtención de los recursos de software es inexistente. Esto representa una gran ventaja económica ante la alternativa de la utilización de plataformas comerciales de código privado como *NAVIO*, con la cual se obtienen resultados similares a los alcanzados en esta investigación. De igual forma, la utilización de una plataforma como *ArduPilot* permite reducir de forma significativa el tiempo y los recursos necesarios en el desarrollo de la investigación, lo que resulta en una alternativa más económica en

comparación con un proceso de desarrollo que no incluya el uso de dicha plataforma. Los resultados obtenidos pueden comercializarse en mercados internacionales y emplearse en la implementación de investigaciones y aplicaciones de *UAV*, que apoyen en la producción y el desarrollo tecnológico y comercial del país, influyendo de manera positiva en su economía.

4.7 Consideraciones finales del capítulo

El controlador *I-LOS* implementado logra seguir satisfactoriamente las trayectorias planificadas en el proceso de simulación, dándole cumplimiento al objetivo general de esta investigación. El método de simulación *HIL* constituye una variante muy útil, puesto que ofrece la posibilidad de variar las condiciones de la simulación sin poner en riesgo los medios técnicos de la investigación, producto de errores en el diseño y permite comprobar el funcionamiento del controlador implementado sin la utilización de la planta real para la cual fue diseñado. Las herramientas para la simulación tratadas a lo largo del capítulo constituyen una base sólida y fiable para el desarrollo de las simulaciones *HIL*, para *UAV* desarrollados sobre la plataforma *ArduPilot* y el autopiloto *ArduPlane*.

CONCLUSIONES

Como resultado final de esta investigación se implementa el algoritmo de guiado *I-LOS* sobre el autopiloto *ArduPlane*, en la plataforma para el desarrollo de vehículos autónomos no tripulados *ArduPilot*, partiendo sobre la base de realizar modificaciones a la programación oficial de la misma. Estos resultados se comprueban mediante simulación. Atendiendo a estos resultados se arriban a las siguientes conclusiones generales:

- El controlador *I-LOS* implementado sobre el autopiloto *ArduPlane* logra seguir las trayectorias definidas en su lógica de guiado de forma satisfactoria, lo que le da cumplimiento al objetivo general de la investigación.
- La estructura implementada en la programación del código que conforma el *firmware* de la plataforma *ArduPilot* permite que se eliminen, agreguen y modifiquen funcionalidades sobre la base del mismo, sin que esto implique un funcionamiento erróneo del autopiloto desarrollado en el proyecto *ArduPlane.*
- El empleo de una plataforma para el desarrollo de *UAV* viabiliza el proceso de investigación e implementación de nuevas aplicaciones a la vez que reduce los costos del proyecto y permite acortar las fechas de entrega.
- El software *Mission Planner* ofrece diversas opciones para la manipulación de los *firmwares* compilados para los proyectos de *ArduPilot* junto con variadas interfaces y opciones para su simulación. Teniendo todo esto en cuenta este software constituye una poderosa herramienta, no solo para la explotación de un vehículo desarrollado a través de *ArduPilot*, sino también en el proceso de desarrollo e implementación de aplicaciones e investigaciones sobre la base de esta plataforma.
- La simulación *HIL* permite realizar simulaciones sobre el elemento de control desarrollado, independientemente del estado de la planta para la cual se diseña, sin arriesgar los medios técnicos del proyecto y en un campo de análisis superior al que puede ser alcanzado con una simulación sobre el dispositivo real.

RECOMENDACIONES

Para establecer la necesaria continuidad que debe tener este trabajo se presentan las siguientes recomendaciones:

- Profundizar en el estudio de la programación del *firmware* de los autopilotos de *ArduPilot* en aras de desarrollar una solución que permita la implementación de varios algoritmos de guiado dentro de un mismo autopiloto, brindando al usuario la posibilidad de seleccionar uno de ellos en tiempo de ejecución del programa.
- Analizar el funcionamiento del controlador *I-LOS* implementado en la investigación sobre los restantes proyectos de *ArduPilot* que implementan autopilotos (*APM Copter* y *APM Rover),* lo que daría paso al estudio y desarrollo de nuevas aplicaciones sobre la base de estos.
- Realizar un estudio comparativo entre el rendimiento y efectividad del algoritmo de guiado *I-LOS* implementado en la investigación con el algoritmo de guiado *L1* implementado en la programación original de *ArduPlane*, valiéndose del controlador implementado en esta investigación.

BIBLIOGRAFÍA

Adiprawita, W., & Sembiring, J. (2007). *Automated flight test and system identification for rotary wing small aerial platform using frequency responses analysis*. *Journal of Bionic Engineering, IV*, 237-244.

Álvarez, L. (2006). Control visual de un vehículo aéreo autónomo usando detección y seguimiento de características en espacios exteriores. Tesis Doctoral, Universidad Politécnica de Madrid, Madrid, España.

Arduino. (2016). *Arduino Playground*. Recuperado en febrero 2016, 4, de playground.arduino.cc: http://playground.arduino.cc/Main/

ArduPilot. (2016). *Developer | APM open source autopilot*. Recuperado en enero 30, 2016, de dev.ardupilot.com: http://dev.ardupilot.com/

Breivik, M., & Fossen, T. I. (2008). *Underwater vehicles. InTech*, 51-76.

Caharija, W. (2014). *Integral Line-of-Sight Guidance and Control of Underactuated Marine Vehicles.* Tesis Doctoral, NTNU, Noruega.

Campoy, A. (2011, diciembre 13). *The Law's New Eye in the Sky*. *The Wall Street Journal*, 10-11.

Carroll, R. (2012, agosto 2). *The philosopher making the moral case for US drones*. *The Guardian*, 12-14.

Cruz, J. M., Aranda, J., & Girón, J. M. (2012). Tutorial automática marina: una revisión desde el punto de vista del control. *Revista Iberoamericana de Automática e Informática industrial*, 205-218.

Dalamagkidis, K. (2014). *Aviation history and unmanned flight*. *Handbook of Unmanned Aerial Vehicles*, 3-4.

de Ávila, D. (2008). *Estrategias de control basado en modelo para minihelicópteros no tripulados.* Trabajo de Diploma, UCLV, Departamento de Automática y Sistemas Computacionales, Santa Clara, Cuba.

DID. (2005, mayo 19). *Predator Kills Important Al-Qaeda Leader in Pakistan*. *Defense Industry Daily*, 10-11.

Elfes, A. (2003). *Robotic airships for exploration of planetary bodies with an atmosphere: autonomy challenges*. *Autonomous Robots*, 147-164.

Emlid. (2016). *Emlid*. Recuperado en febrero 16, 2016, de emlid.com: https://emlid.com/

ErleRobotics. (2016). *Erle-Brain 2 | Erle Robotics*. Recuperado en febrero 18, 2016, de erlerobotics.com: https://erlerobotics.com/blog/product/erle-brain-v2/

FAA. (2012). *FAA MODERNIZATION AND REFORM ACT OF 2012.* Estados Unidos: FAA.

Feltman, R. (2014, febrero 4). *The Future of Sports Photography*: *Drones*. *The Atlantic*, 16-17.

FlightGear. (2016). *Introduction | FlightGear Flight Simulator*. Recuperado en marzo 8, 2016, de www.flightgear.org: http://www.flightgear.org/about/

Fossen, T. I. (2002). *Guidance, Navigation, and Control of Ships, Rigs and Underwater Vehicles.* Noruega: Marine Cybernetics.

Fossen, T. I., & Breivik, M. (2007). *Applying missile guidance concepts to motion of marine craft*. *Control Applications in Marine Systems, VII*, 349-354.

Fung, B. (2013, agosto 17). *Why drone makers have declared war on the word 'drone'*. *The Washington Post*, 3-4.

Gizmag. (2016). *Hydrogen-powered Hycopter quadcopter could fly for 4 hours at a time*. Recuperado en febrero 4, 2016, de www.gizmag.com: http://www.gizmag.com/horizon-energy-systems-hycopter-fuel-cell-drone/37585/

GNU. (2016). *The GNU General Public License v3.0 - GNU Project - Free Software Foundation*. Recuperado en enero 30, 2016, de gnu.org: http://www.gnu.org/licenses/gpl.html

Grankvist, H. (2006). *Autopilot design and path planning for a UAV.* Reporte Científico FOI-R-2224–SE, FOI -Swedish Defence Research Agency, Estocolmo, Suecia.

GTRI. (2016). *Flying on Hydrogen: Georgia Tech Researchers Use Fuel Cells to Power Unmanned Aerial Vehicle | Georgia Tech Research Institute*. Recuperado en febrero 4, 2016, de www.gtri.gatech.edu: http://www.gtri.gatech.edu/casestudy/flying-hydrogen

Hagen, I. (2014). *Autopilot Design for Unmanned Aerial Vehicles.* Tesis de Maestría, NTNU, Noruega.

Healey, A. J. (2006). *Advances in unmanned marine vehicles.* Gran Bretaña.

Hernández Julián, A. (2014). Estrategia de control para el seguimiento de camino de un vehículo autónomo subacuático. Trabajo de Diploma, UCLV, Departamento de Automática y Sistemas Computacionales, Santa Clara, Cuba.

Hernández, D. (2014). Modelado dinámico de aviones subsónicos a partir de herramientas de software. Tesis de Grado, UCLV, Departamento de Automática y Sistemas Computacionales, Santa Clara, Cuba.

Hwang, T. (2006). *Development of HIL Systems for active Brake Control Systems*. *SICE-ICASE International Joint Conference.* Estados Unidos.

Kothari, M., Postlethwaite, I., & Gu, D. (2014). *Uav path following in windy urban environments*. *Journal of Intelligent & Robotic Systems*, 1013–1028.

Lara, J. (2015, marzo 18). *Drones Might Save Lives in Chilean Beaches*. *Drone's Republic*, 15-16.

Lekkas, A. M. (2014). *Guidance and Path-Planning Systems for Autonomous Vehicles.* Tesis Doctoral, Noruega.

Lekkas, A. M., & Fossen, T. I. (2012). A *time-varying lookahead distance guidance law for path following*. *9th IFAC Conference on Manoeuvring and Control of Marine Craft*, (págs. 398-403). Arezano, Italia.

LiveScience. (2016). *Jet-Propelled 3D-Printed Drone Claims Speed Record*. Recuperado en febrero 4, 2016, de LiveScience.com: http://www.livescience.com/52853-jet-powered-3d-printed-drone.html

Lundin, L. (2013, febrero 3). Eye in the Sky. *UAV Drones*, 7-8.

Martínez, M. E. (2014). Estrategia de control para seguimiento de camino en avión no tripulado. Tesis de Maestría, UCLV, Departamento de Automática y Sistemas Computacionales, Santa Clara, Cuba.

McFarland, M. (2014, septiembre 17). *In Switzerland, police find a use for drones*. *The Washington Post*, 20-21.

McNeil, M. (2012). *Understanding "the loop": Regulating the next generation of war machines.* Harvard, Estados Unidos.

MissionPlanner. (2016). *Mission Planner Overview | Mission Planner*. Recuperado en marzo 8, 2016, de planner.ardupilot.com: http://planner.ardupilot.com/wiki/mission-planner-overview/

MyRCMart. (2016). *RCX08-015 : APM 2.5.2 ArduPilot Mega 2.5 Flight Control Board*. Recuperado en abril 16, 2016, de myrcmart.com: http://myrcmart.com/apm-252-ardupilot-mega-25-flight-control-board-multicopter-plane-helicopter-p-6816.html

NBC. (2005). *CIA drone said to kill A-Qaeda operative - US news - Security - NBC News*. Recuperado en enero 8, 2016, de msnbc.com: http://www.msnbc.msn.com/id/7847008/

Nelson, D. (2007). *Vector field path following for miniature air vehicles*. 519–529. IEEE Transactions on Robotics.

Nichols, M. (2013, agosto 1). *Italian firm to provide surveillance drone for U.N. in Congo*. *Reuters*.

Ortega Escudero, F. R. (2015). Controlador I-LOS para seguimiento de caminos en líınea recta en el avión Hobbico. Trabajo de Diploma, UCLV, Departamento de Automática y Sistemas Computacionales, Sta. Clara, Cuba.

Osborne, J., & Rysdyk, R. (2005). *Waypoint guidance for small uavs in wind*. *AIAA Infotech Aerospace*, 1-12.

Park, S., Deyst, J., & P. How, J. (2010). *A New Nonlinear Guidance Logic for Trajectory Tracking.* Massachusetts Institute of Technology, Massachusetts, Estados Unidos.

Pasztor, A., & Emshwiller, J. (2012, abril 21). *Drone Use Takes Off on the Home Front*. *The Wall Street Journal*, 3-4.

Peterson, A. (2013, agosto 22). States are competing to be the Silicon Valley of drones. *The Washington Post*, 4-5.

Pettersen, K. Y., & Lefeber, E. (2001). *Way-point tracking control of ships*. *Proceedings of the 40th IEEE Conference on Decision and Control*, (págs. 940-945). Florida, Estados Unidos.

Pixhawk. (2016). *Home - Pixhawk Flight Controller Hardware Project*. Recuperado en marzo 20, 2016, de pixhawk.org: https://pixhawk.org/start

Rivero Rodríguez, W. (2015). Estudio del algoritmo de control de rumbo L1 implementado en el autopiloto Ardupilot. Trabajo de Diploma, UCLV, Departamento de Automática y Sistemas Computacionales, Santa Clara, Cuba.

Rogers, E. (2014, enero 30). *One day a drone might throw you a life preserver*. *The Washington Post*, 6-7.

Rysdyk, R. (2003). *Uav path following for constant line-of-sight. 2th AIAA Unmanned Unlimited. Conf. and Workshop and Exhibit.* California, Estados Unidos: Aerospace Research Central.

Rysdyk, R. (2007). *Course and heading changes in significant wind. Journal of guidance control, and dynamics*, 1168-1171.

Spearman, M. L. (1983). *Historical development of worldwide guided missiles.* Reporte Técnico , NASA Langley Research Center, Hampton, Estados Unidos.

Valeriano Medina, Y. (2013). Modelado dinámico de un vehículo autónomo subacuático. Tesis de Maestría, UCLV, Departamento de Automática y Sistemas Computacionales, Santa Clara, Cuba.

Velasco, F. J. (2008). *Techniques for tuning track-keeping controllers of an autonomous in-scale fast-ferry model*. *International Journal Of Systems Applications, Engineering & Development*, 227–236.

Wang, D., & Du, J. (2007). *Hardware-in-the-loop simulation approach to testing controller of sequential turbocharging system. Proceedings of the IEEE International Conference on Automation and Logistics.* Estados Unidos.

WM. (2008, septiembre 15). *New strategies for damage assessment*. *Wayback Machine*, 10-12.

Woo, J., & Son, K. (2007). *Vision-based uav navigation in mountain area*. *IAPR Conference on Machine Vision Applications*, (págs. 239-239). Tokio, Japón.

Wood, J. (2015, mayo 27). *Science, technology and the future of small autonomous drones*. *Nature*, 12-13.