UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Universidad Politécnica de Valencia

Trabajo de Fin de Grado

Grado en Ingeniería Aeroespacial

Control del simulador X-Plane mediante una cámara RGBD

Autor

Barberá Chuliá, Adrián adbarchu@etsid.upv.es

Tutor

Rodas Jordá, Ángel arodas@disca.upv.es

15 de julio de 2021

Resumen

El presente trabajo versa sobre el desarrollo de una aplicación relacionada con el mundo de la aviación. Actualmente, este sector está en auge y emplea simuladores de vuelo para la prueba de nuevos diseños y formación de futuros pilotos. En este trabajo de fin de grado, se aplica el análisis gestual al guiado de una aeronave localizada en un simulador de vuelo, permitiendo un control más novedoso de esta.

En la realización del proyecto se requiere la intervención de una cámara RGBD (siendo usado el dispositivo LEAP Motion) para ofrecer un método alternativo al de teclado y ratón durante el control de simuladores de vuelo. Con su ayuda, se desarrolla una aplicación que permita el control de los mandos de vuelo básicos de la aeronave, permitiendo no solo la realización del despegue de esta, sino también, la activación del autopiloto.

Para ello, el trabajo requiere ser desarrollado en el lenguaje de programación de Java, que ofrece la posibilidad de interacción entre el simulador de vuelo y la cámara RGBD.

Palabras clave: LEAP Motion; Simulador de vuelo; X-Plane; Java; Netbeans.

Abstract

This work deals with the development of an application related to the world of aviation. Currently, this sector is booming and uses flight simulators for testing new designs and training future pilots. In this final degree project, gestural analysis is applied to the guidance of an aircraft located in a flight simulator, allowing a more innovative control of it.

The project requires the intervention of an RGBD camera (using the LEAP Motion device) to offer an alternative method to the keyboard and mouse during the control of flight simulators. With its help, an application is developed that allows the control of the basic flight controls of the aircraft, allowing not only the realization of the takeoff of this, but also the activation of the autopilot.

For this, the work requires to be developed in the Java programming language, which offers the possibility of interaction between the flight simulator and the RGBD camera.

Key words: LEAP Motion; Flight simulator; X-Plane; Java; Netbeans.

Resum

El present treball versa sobre el desenvolupament d'una aplicació relacionada amb el món de l'aviació. Actualment, aquest sector està en auge i empra simuladors de vol per a la prova de nous dissenys i formació de futurs pilots. En aquest treball de fi de grau, s'aplica l'anàlisi gestual al guiat d'una aeronau localitzada en un simulador de vol, permetent un control més nou d'aquesta.

En la realització del projecte es requereix la intervenció d'una càmera RGBD (sent usat el dispositiu LEAP Motion) per a oferir un mètode alternatiu al de teclat i ratolí durant el control de simuladors de vol. Amb la seua ajuda, es desenvolupa una aplicació que permeta el control dels comandaments de vol bàsics de l'aeronau, permetent no sols la realització de l'enlairament d'aquesta, sinó també, l'activació de l'autopilot.

Per a això, el treball requereix ser desenvolupat en el llenguatge de programació de Java, que ofereix la possibilitat d'interacció entre el simulador de vol i la càmera RGBD.

Paraules clau: LEAP Motion; Simulador de vol; X-Plane; Java; Netbeans.

Índice general

Índice de figuras

Índice de cuadros

Lista de símbolos

$RGBD$	Red, Green, Blue, Depth
$ADS-B$	Automatic Dependent Surveillance Broadcast
LED	Light-Emitting Diode
$VTOL$	Vertical Take-Off and Landing
SDK	Software Development Kit
API	Application Programming Interface
UDP	User Datagram Protocol

1 | Introducción

Según un estudio de Oneair, se producirá un crecimiento significativo de la aviación comercial en las próximas décadas, donde Airbus prevé un crecimiento anual medio del $4,3\,\%$ a nivel mundial, como consecuencia de la evolución de dicho sector en Oriente Medio y Asia-Pacífico, principalmente [1]. Este hecho no solo supondrá un incremento en la flota global de aeronaves, sino también la necesidad de una mayor formación de pilotos capaces para controlarlas.

La evolución tecnológica ha llevado a que la formación de nuevos pilotos pueda ser llevada a cabo mediante la utilización de simuladores de vuelo. Los simuladores de hoy en día, constan de replicas idénticas de aeronaves y un alto realismo durante el pilotaje de las misma, permitiendo que los pilotos adquieran un gran manejo y conocimiento sobre el avión que están volando, a la vez que se limita el gasto económico y se eliminan los posibles accidentes durante las etapas de aprendizaje. Por su parte, los simuladores de vuelo también permiten a pilotos más experimentados aprender a realizar maniobras más arriesgadas para posibles casos de emergencia y a los ingenieros a probar como sería el comportamiento aerodinámico de nuevos modelos de aeronaves que busquen introducir en el mercado, permitiendo la obtención de resultados fiables con poca inversión y en un corto periodo de tiempo.

1.1. Motivación y justificación

Con el deseo de conseguir realizar esta tarea, ya bien sea destinada al aprendizaje de como es pilotar una aeronave o al estudio del comportamiento de la misma, es de donde surge este proyecto. En la actualidad, varios simuladores de vuelo han sido validados para poder ser empleados como método de aprendizaje de pilotos, donde los estudiantes a piloto realizan las pruebas en entornos con un alto nivel tecnológico, en el cual las cabinas del simulador son capaces de reproducir a la perfección las sensaciones que experimentarían durante el vuelo, incluyendo aspectos con las vibraciones del avión.

En este documento, se indagará sobre la tecnología de control por gestos, siendo una de las vías más básicas de expresión del ser humano, que podría permitir el pilotaje de una aeronave de una forma algo menos convencional. Mediante este control, se pretende ofrecer una vía de uso de los simuladores con menor complejidad que la que conlleva una cabina, a la vez que permite un mayor manejo que el que se tendía si se desea emplear un simulador de vuelo mediante el método de control tradicional basado en el teclado y

ratón.

Es por esto, por lo que la motivación principal del actual proyecto es proponer una solución alternativa que dé la posibilidad a ingenieros de poder testear el comportamiento de diferentes piezas sobre aeronaves determinadas o, proporcione a futuros pilotos el conocimiento básico de pilotaje con la ayuda de nuevas tecnologías, que resultan económicamente más asequibles para el público y que pueden agilizar el proceso de estudio de la simulación al no requerir, prácticamente, tiempo de preparación para proceder con el control del simulador.

Dicho todo esto, resulta necesario mencionar que el objetivo del proyecto no es el de obtener un control completo de la aeronave, sino el de poder actuar sobre controles básicos de la misma a fin de poder evaluar el comportamiento durante el vuelo. Por otra parte, la fiabilidad de los dispositivos por control de gestos encontrados en el mercado, a coste asequible para el consumidor, como es el caso del dispositivo LEAP Motion empleado en el presente proyecto, difícilmente podrían resultar suficiente para poder abarcar todos las posibilidades que se encuentran dentro de una cabina.

Finalmente, el presente documento estudiará una opción alternativa a la ejecución de simuladores de vuelo mediante un dispositivo lector de gestos, a partir del cual, podría ampliarse las posibilidades de control que se estipulen para hacerlo todavía más completo.

1.2. Objetivos

El objetivo principal del trabajo presentado en este TFG es el de la realización de una aplicación que permita el control de una aeronave en el simulador de vuelo X-Plane, mediante el uso de un sensor LEAP Motion. Todo ello será llevado a cabo mediante la programación en lenguaje Java.

Este objetivo principal puede ser dividido en diferentes fases o subobjetivos que permitan alcanzar el propósito establecido. Las fases encontradas son las siguientes:

- Adquirir conocimientos profundos sobre la **programación en Java**, para poder llegar a manejar Netbeans durante el desarrollo de la aplicación que permitirá el control. Parte esencial del proyecto, dado que es la base necesaria para la conexión LEAP Motion con X-Plane y se parte sin ninguna experiencia previa.
- Investigación sobre el funcionamiento de la tecnología por control de gestos del dispositivo LEAP Motion, así como la lectura de los datos proporcionados al mismo. Esta fase del proyecto es elemental para poder comprender las posibilidades que esta tecnología nos ofrece, y en base a ellas realizar una aplicación fiable, que permita ser empleada por cualquier usuario.
- Los conocimientos del **simulador de vuelo X-Plane** deberán ser ampliados, con la finalidad de comprender todas las posibilidades que se ofrecen y ser capaz de seleccionar las necesarias. Además es necesario estudiar como se lleva a cabo el

proceso de lectura y transmisión de datos desde la propia aplicación para poder actuar sobre los distintos controles

- Con todos los conocimientos necesarios adquiridos previamente, se deberá llevar a cabo la **combinación** de los mismos para poder desarrollar la aplicación final, mediante la cual se leerán los datos del controlador LEAP Motion, y mediante Java se enviará dicha información que será plasmada en el simulador de X-Plane.

2 | Estado del arte

El primer paso para la realización de un proyecto, es siempre conocer las tecnologías que van a ser empleadas en el mismo con detalle, así como proyectos previos realizados con dichas tecnologías, los cuales pueden resultar de gran utilidad. En este trabajo de fin de grado son relevantes tanto, la tecnología de reconocimiento de gestos, concretamente la relacionada con el uso de dispositivos LEAP Motion, y el uso del simulador de vuelo X-Plane.

2.1. LEAP Motion

Previamente a la aparición del controlador LEAP Motion, los dispositivos más eficaces que podrían ser encontrados en el mercado para el seguimiento de gestos eran dispositivos de detección equipados en la mano, tratándose de guantes equipados con sensores para medir las diferentes posiciones de las manos y de sus articulaciones, pudiendo detectar la dirección en la que se señala o la fuerza ejercida en ese momento. Sin embargo, el elevado coste de estos junto con la poca naturalidad de los movimientos detectados, así como la complejidad de su montaje, evitaron su triunfo en el mercado [2].

El nacimiento de la tecnología del dispositivo LEAP Motion, junto a su precio asequible, ha supuesto un aumento en la cantidad de estudios relacionados con la lectura de gestos. En la actualidad es posible encontrar varios estudios acerca de esta tecnología entre los cuales se encuentran los siguientes proyectos.

Por una parte, se encuentra un trabajo de fin de máster centrado en las pasibilidades de desarrollo que ofrece el LEAP Motion, junto con la implementación de gestos sencillos de realizar pero que permitan un control robusto durante el diseño de un sencillo caso práctico [3].

Por otra parte, aquí en la UPV han sido realizados varios trabajos en los cuales se conectaba la información del LEAP Motion con otras aplicaciones. El trabajo de Álvaro Edo Martínez [4] se centra en realizar una aplicación que permite optimizar las tareas más básicas relacionadas con el control del tráfico aéreo, mediante la interacción del dispositivo LEAP Motion con la antena ADS-B ubicada en la Escuela Técnica Superior de la Ingeniería del Diseño (ETSID), recibiendo datos del tráfico aéreo actual del servidor de dicha antena. En cuanto al trabajo de Kevin Esperanza Martínez [5], se emplea el dispositivo LEAP Motion para la realización de una aplicación en el entorno de desarrollo

de *Processing* que permita controlar un dron físico.

2.2. Simulador de vuelo X-Plane

La evolución tecnológica a permitido que se empleen métodos alternativos que posibiliten el estudio de las diferentes características de las aeronaves antes de la fabricación de estas. En este ámbito aparecen los simuladores de vuelo, que introduciendo los datos de diseño de la aeronave que se desee estudiar.

En la actualidad la mayoría de los estudios realizados con X-Plane tienen como finalidad el estudio de determinadas aeronaves o la comparación entre ellas. En el caso de trabajo de fin de grado de Lorena Calvente Roldán [6], estudia las posibilidades de aplicación de técnicas de aprendizaje supervisado para diseñar un piloto automático que permita aterrizar una aeronave confiando únicamente en los instrumentos de la aeronave, donde se emplea el X-Plane para probar y demostrar el comportamiento del piloto automático diseñado.

2.3. Tecnología de reconocimiento de gestos en simuladores de vuelo

Dado que tanto la tecnología de reconocimiento de gestos, como los simuladores de vuelo se tratan de tecnologías que se encuentran en auge, es lógico pensar que es posible encontrar casos en los que se han realizado interacciones entre estas.

Los propios simuladores de vuelo, ofrecen opciones de control mediante el uso de cascos de realidad virtual. En el caso del propio simulador de X-Plane, se pueden ejecutar los controles mediante el uso de cascos de realidad virtual, con modelos como: *HTC Vive*, *Oculus Rift* o *Windows Mixed Reality (WMR)*.

En este escenario de compatibilizar ambas tecnologías para una misma función, pueden ser encontrados varios desarrollos de aplicaciones, que conectan el dispositivo LEAP Motion con simuladores de vuelo.

Estas aplicaciones pueden consistir en un control muy simple, en el que se varia la dirección de la aeronave mediante la actuación sobre las superficies de control de la aeronave, para lo cual solo sería necesario el uso de un único dedo, como en el caso de la referencia [7], donde se emplea al igual que en el presente documento el dispositivo LEAP Motion para controlar el simulador de X-Plane 10.

Otras aplicaciones realizadas, pueden llegar a ser mucho más complejas, como es el caso del control realizado en el vídeo de la referencia [8]. En este caso, *SimHanger Flight Simulation* (el autor del vídeo), hace uso de la tecnología de LEAP Motion en conjunto con unos casos de realidad virtual para poder ejercer un control bastante completo de la

aeronave, empleando los simuladores de vuelo conocidos como: *PREPAR3D* y *FLYINSIDE flight simulator*.

Por tanto, como ha sido visto, el desarrollo del presente proyecto tiene sentido en el ámbito presentado, donde el auge de la aviación requerirá de nuevos métodos de formación y estudio, mientras que el las tecnologías empleadas para el desarrollo se encuentran en un periodo de estudio por diferentes autores.

3 | Materiales y métodos

En esta sección se llevará a cabo un desglose de los distintos materiales y métodos que han sido necesarios para la realización del presente proyecto, para familiarizar al lector con las herramientas empleadas.

3.1. Equipo informático

El equipo principal para el desarrollo de la aplicación consiste en dos ordenadores. Dado que la función de estos reside en leer los datos proporcionados por el dispositivo LEAP Motion, ejecutar la aplicación desarrollada en Netbeans y mantener abierta el programa de X-Plane, se requerirá que el ordenador tenga un mínimo de potencia para poder ejecutar todo sin problemas (no necesariamente debe ser muy potente, pero cuanto más lo sea, no se sobrecalentará y menos problemas dará). La razón por la cual dos ordenadores son requeridos es porque la sincronización de X-Plane con LEAP Motion puede dar errores de lectura (más información sobre este aspecto será introducida en la Sección 4.3).

Durante la realización del proyecto un Lenovo Z50-70, Intel Core i7-4510U de 15,6 pulgadas ha sido empleado. El otro ordenador empleado se trata de un Portátil ASUS E751JF-T2052H, Intel Core i7-4712MQ de 17,3 pulgadas ha sido empleado. De todas formas cabe destacar que cualquier otro modelo de ordenador ya bien sea de Windows, Mac o Linux podría ser empleado al ser los tres sistemas operativos compatibles con los programas utilizados, siempre que cumplan con los requisitos para la utilización del controlador de LEAP Motion.

3.2. Dispositivo LEAP Motion

La tecnología de reconocimiento de gestos proporciona datos en tiempo real a un ordenador para que cumpla con todos los comandos proporcionados por el usuario. Los sensores de movimiento implementados en los dispositivos pueden detectar e interpretar gestos, usándolos como fuente primaria de datos. Las soluciones de reconocimiento de gestos cuentan con una característica combinación de cámaras de profundidad 3D y cámaras infrarrojas junto con un sistema de aprendizaje automático [9]. Un exponente de esta

tecnología puede ser el kinect de XBOX, que se trata de una cámara RGDB que permite la lectura en tres dimensiones del cuerpo del usuario permitiendo a este interactuar con la consola.

El reconocimiento de gestos podría ser dividido en tres niveles de seguimiento básicos [9]:

- **Detección:** con la ayuda de la cámara, el dispositivo es capaz de detectar movimientos y un algoritmo de aprendizaje automático divide la imagen para poder diferenciar las diferentes posiciones del cuerpo o de la mano.
- **Seguimiento:** el dispositivo monitoriza los movimientos frame a frame para capturar todos los movimientos y proporcionar una entrada precisa.
- **Reconocimiento:** el sistema busca distintos patrones basados en la información recogida. Cuando el sistema encuentra alguna coincidencia en los gestos visualizados, realiza las acciones correspondientes asociadas a dichos gestos.

Esta tecnología de reconocimiento de gestos nos ofrece una nueva manera de interactuar con otros dispositivos. Bajo esta tecnología surgen los dispositivos LEAP Motion, el cual es empleado para la realización del presente TFG.

El dispositivo LEAP Motion de Ultraleap consiste en una pequeña cámara, que utiliza cámaras estéreo de infrarrojos como sensores de seguimiento, que captura el movimiento de las manos y dedos del usuario para que pueda interactuar libremente con el contenido digital. Es un dispositivo pequeño, rápido y preciso, que puede ser empleado para la producción de aplicaciones con ordenadores (función para la que se destinará este dispositivo en el proyecto desarrollado), integrado en pantallas o soluciones de hardware a nivel empresarial, o conectado a cascos de realidad virtual para creación de nuevos prototipos. investigación y desarrollo [10].

Ahora se procederá con una descripción de las especificaciones técnicas del dispositivo LEAP Motion de Ultraleap, para posteriormente explicar su principio de funcionamiento, datos capturados y conectividad.

3.2.1. Características técnicas

El dispositivo LEAP Motion se trata de un instrumento de tamaño bastante reducido en comparación a otros dispositivos similares de reconocimiento de gestos que pueden ser encontrados en el mercado. Las dimensiones son las mostradas en la Figura 3.1 [10], donde puede ser observado su pequeño tamaño.

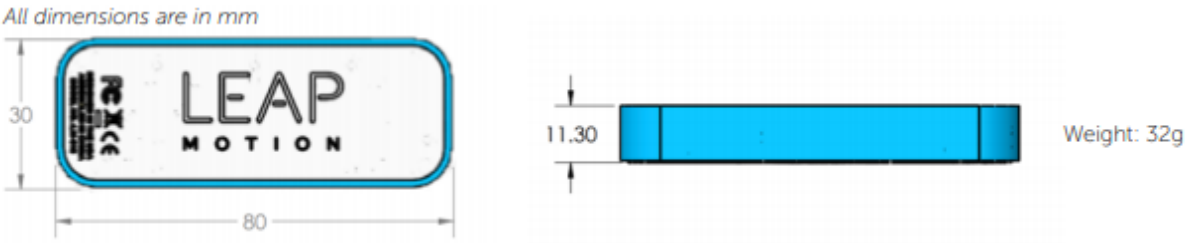

Figura 3.1. Dimensiones del dispositivo LEAP Motion

El LEAP Motion siempre irá conectado al ordenador mediante un puerto USB para poder ser reconocido por el equipo empleado.

Desde una perspectiva del hardware, el reconocimiento de manos se lleva a cabo de una forma relativamente sencilla. Este dispositivo esta compuesto, en primer lugar, por dos cámaras de 640x240 píxeles que cuentan con una ventana transparente infrarroja, operando en un rango de 850 nm $\pm$ 25 nm. Por otra parte, cabe destacar también la presencia de tres LEDs infrarrojos capaces de emitir luz infrarroja con una longitud de onda de 850 nm, estando dentro del rango de funcionamiento de las cámaras previamente mencionadas. Estos sensores LED, se encuentran situados a los laterales y entre las cámaras, cabe destacar que estos sensores se encuentran aislados mediante pequeñas paredes de plástico para evitar superposiciones con las cámaras y una saturación de la imagen [10]. Una vista interna del dispositivo LEAP Motion puede observarse en la Figura 3.2 [10].

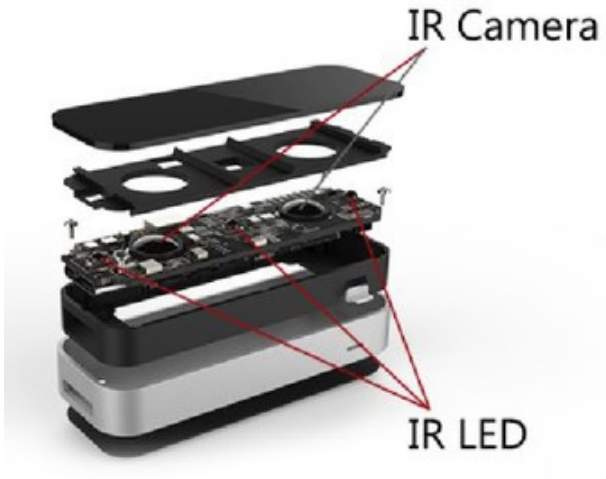

Figura 3.2. Estructura interna del dispositivo LEAP Motion

En lo referente al rango de visión del LEAP Motion, este dispositivo posee la zona de cobertura mostrada en la Figura 3.3 [11]. Analizando en detalle este rango de visión, se puede observar como tiene una forma aproximadamente esférica, teniendo un rango a lo

ancho de 120° y a lo largo de 150°, pudiendo llegar hasta una altura máxima de unos 80 cm desde el dispositivo. El LEAP Motion tiene una desviación típica menor de un milímetro, aunque la precisión disminuye a medida que la distancia al dispositivo aumenta. Puede que en determinadas situaciones una mano se esconda detrás de la otra y la cámara deje de tener contacto directo con esta, pero un algoritmo permite al LEAP Motion estimar la posición de dicha mano, de todas formas, siempre resulta preferible mantener ambas manos visibles a las cámaras [11].

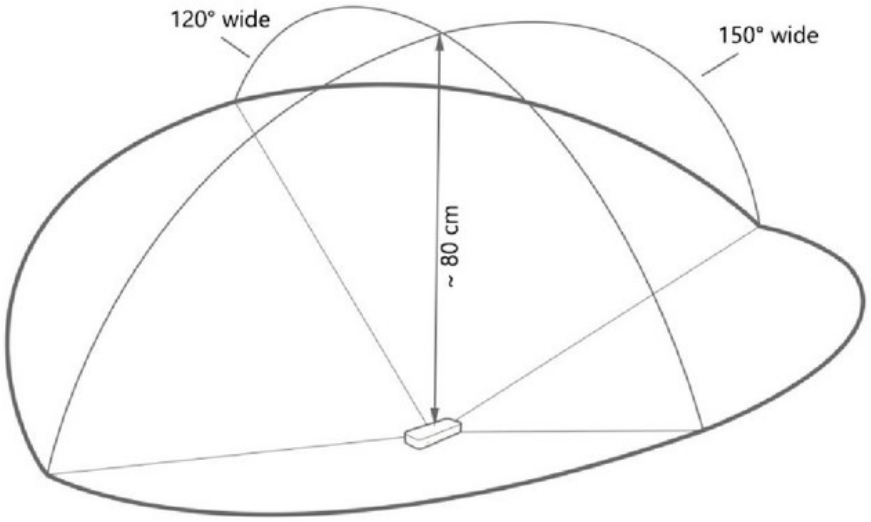

Figura 3.3. Rango de visión del dispositivo LEAP Motion

Antes de proceder con la explicación del principio de funcionamiento del LEAP Motion, se ha considerado imprescindible mencionar los requisitos mínimos con los que ha de cumplir el sistema que quiera trabajar con este dispositivo [10].

- **Versión:** Windows 7+ o Mac OS X 10.7+
- **Procesador:** AMD Phenom II o Intel Core i3 / i5 / i7
- **Memoria:** 2 GB RAM
- **Puerto:** USB 2.0

3.2.2. Principio de funcionamiento

Una vez la estructura interna ha sido definida se procede a analizar el principio de funcionamiento, por el cual se rige el dispositivo LEAP Motion. En este caso, se emplea como referencia el modelo de visión estereoscópica. Se trata de un modelo presente en algunos seres vivos (particularmente en los seres humanos), donde el desplazamiento relativo de los ojos permite obtener la posición de los objetos situados en el rango de visión mediante

un método de triangulación a partir de las imágenes generadas de la misma escena por cada ojo. El sistema visión estereoscópica se encuentra representado en la Figura 3.4 [12].

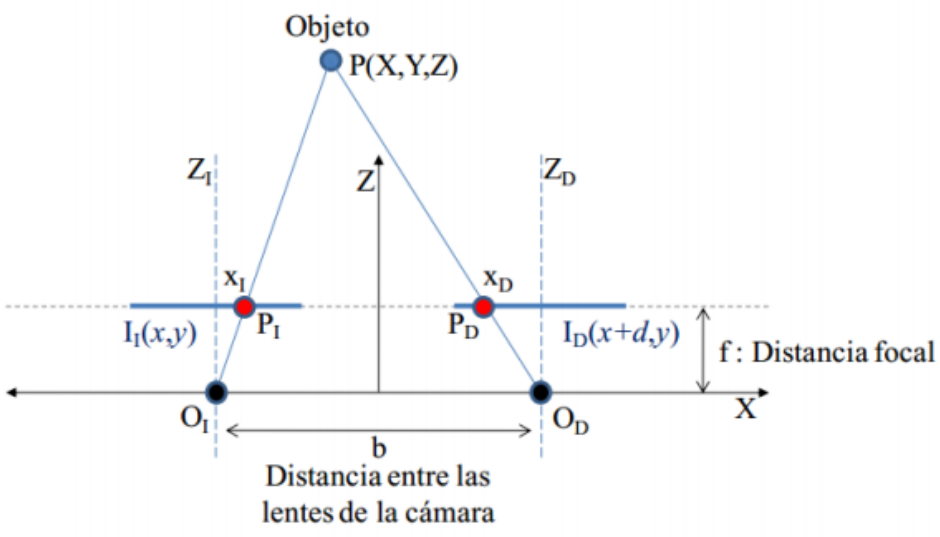

Figura 3.4. Geometría de visión estereoscópica con dos cámaras

En el caso del dispositivo LEAP Motion, los LEDs se emplearán para iluminar la zona de cobertura mediante una luz infrarroja. La luz será recogida por las cámaras que combinando las imágenes de las mismas podrán conseguir las distancias a los puntos detectados.

El proceso del cálculo de la profundidad consta de las dos cámaras con sus ejes ópticos (Z_I y Z_D) separados paralelamente por una distancia denominada línea base. Este escenario de ejes ópticos paralelos implica que las imágenes captadas por ambas cámaras difieren únicamente en su componente horizontal, dando lugar a la restricción epipolar (las rectas epipolares consisten en líneas que proyectan un mismo punto de la escena en la imagen izquierda y derecha), que ayuda a encontrar las correspondencias entre ambas imágenes [12]. La disparidad para cada par de puntos ($P_{(x_D,y_D)}$ y $P_{(x_I,y_I)}$) viene dada por la Ecuación 3.1.

$$d = x_I - x_D \tag{3.1}$$

Conociendo los datos de las distancias focales de las cámaras, a la vez que la distancia relativa entre ellas, se puede obtener mediante semejanza de triángulos la profundidad del punto P en el sistema de coordenadas global. Relacionando los dos triángulos para cada una de las cámaras x_I y x_D pueden ser obtenidos según las ecuaciones 3.2 y 3.3.

$$O_I : \frac{\frac{b}{2} + X}{Z} = \frac{x_I}{f} \rightarrow x_I = \frac{f}{Z}\left(X + \frac{b}{2}\right) \tag{3.2}$$

$$O_D : -\frac{\frac{b}{2} - X}{Z} = \frac{x_D}{f} \rightarrow x_D = \frac{f}{Z}\left(X - \frac{b}{2}\right) \tag{3.3}$$

A partir de estas dos últimas ecuaciones podemos obtener la disparidad previamente definida, mediante la Ecuación 3.4.

$$d = x_I - x_D = \frac{f}{Z}\left(X + \frac{b}{2}\right) - \frac{f}{Z}\left(X - \frac{b}{2}\right) = \frac{f \cdot b}{Z} \tag{3.4}$$

Finalmente, se puede deducir que la profundidad, coordenada Z, es inversamente proporcional a la disparidad, adoptando la forma de la Ecuación 3.5.

$$Z = \frac{f \cdot b}{d} \tag{3.5}$$

Una vez realizado el proceso correspondiente a la detección de los distintos puntos, con sus respectivas coordenadas generales, localizados en las escenas visualizadas por las cámaras, se aplicarán técnicas de reconocimiento de imágenes par obtener las manos localizadas en la escena. Como ejemplo de visualización de las manos, la Figura 3.5 ha sido incluida.

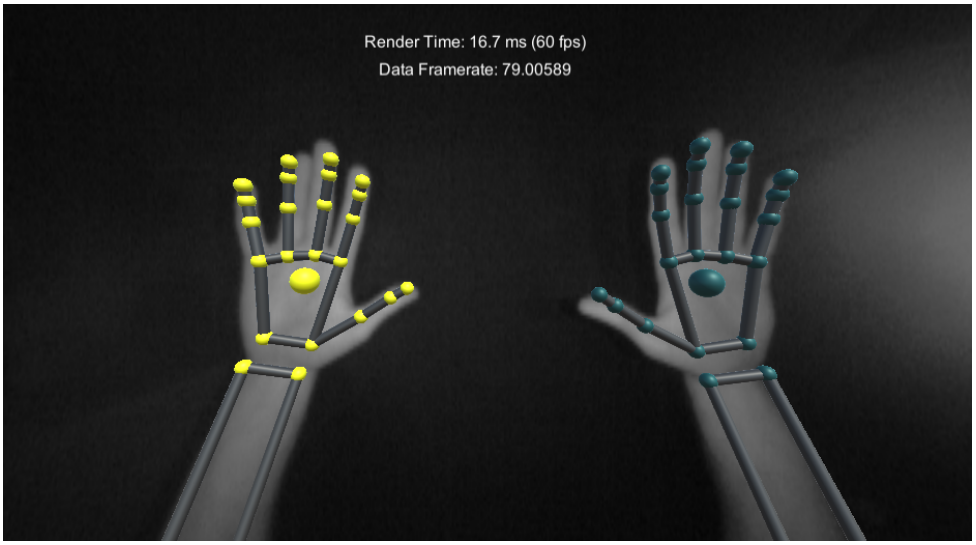

Figura 3.5. Captura obtenida mediante la aplicación LEAP Motion Visualizer

Toda la información recogida por las cámaras, podrá ser finalmente extraída del dispositivo LEAP Motion gracias al API, que nos permite obtener magnitudes físicas en tiempo real de las manos del usuario. Las unidades empleadas por la API son:

- **Distancia:** mm

- **Tiempo:** μs
- **Velocidad:** $\frac{mm}{s}$
- **Ángulos:** radianes

3.2.3. Datos de seguimiento

A medida que el controlador LEAP Motion va rastreando las manos y dedos que se encuentran dentro de su campo de visión, proporciona continuas actualizaciones sobre los datos leídos. Cada *frame* representa un marco que contiene información detallada sobre las propiedades de los objetos visualizados en cada instante. Estos marcos son la base del modelo de datos del LEAP Motion, en el que se pueden encontrar tres objetos principales: manos, dedos y brazos [13].

La clase *Hand*, proporciona información sobre la identidad, posición y otras características de las manos detectadas en cada marco, además de poder detectar a que brazo está unida cada mano y los dedos asociados a ella. Como ha sido mencionado, el dispositivo LEAP Motion emplea un modelo de detección de manos basado en el modelo interno de la mano humana, para predecir el seguimiento de datos cuando esta deja de estar visible. Este modelo también proporciona el seguimiento de los dedos, aunque es óptimo cuando la silueta de la mano con todos sus dedos es visible. Dicho esto, el sensor es capaz de detectar más de dos manos simultáneamente. De todas formas se recomienda limitar la cantidad de manos detectadas a dos para que la calidad de seguimiento sea óptima.

La clase *Arm* consiste en la detección de un objeto con forma de hueso que proporciona información sobre su orientación, longitud, ancho y los puntos finales del brazo. En los casos en los que el codo no se encuentra en visión, se estima su posición basándose en las proporciones típicas del cuerpo humano.

Finalmente, se encuentra la clase *Finger*, que proporciona información sobre cada uno de los dedos de la mano. En caso de que no se encuentren visibles, sus datos se estiman basados en el modelo anatómico de la mano humana. Este objeto es capaz de obtener información sobre la velocidad y dirección de la punta del dedo.

3.2.4. Conectividad

Finalmente, para finalizar con la descripción de las características y funcionalidad del dispositivo LEAP Motion, es importante conocer la manera de comprobar la conectividad del LEAP.

En primer lugar, puede ser observada una luz en uno de los laterales del aparato que indicará las posibles condiciones de operación en las que se encuentra el dispositivo dependiendo del color que esta tenga. En el caso en el que se encuentre de color verde como en la Figura 3.6 [10], significa que el dispositivo está operando con normalidad. En

cualquier otro caso, la luz será de otros colores. Para conocer más información sobre estas otras situaciones dirigirse a la referencia [14].

Figura 3.6. Icono de las condiciones de operación, de color verde

Por otro lado, si desde el computador al cual tenemos conectado el dispositivo LEAP Motion se abre el LEAP Motion Control Panel, se observa el estado en el que se encuentra el dispositivo y una serie de test que nos muestran si se encuentra operando correctamente [15]. Estas pruebas consisten en:

- **Service status:** valida la identidad del firmware del controlador. Si esta prueba falla, comuníquese con el soporte de Leap Motion.
- **Device status:** verifica que el dispositivo envía señales al ordenador.
- **Calibration status:** comprueba si el dispositivo esta calibrado correctamente. Si falla se debe repetir el proceso de calibración.
- **Rastreo status:** comprueba que el dispositivo puede analizar correctamente la escena en su campo de visión, para ello se debe mantener la mano encima del aparato.
- **Bandwith status:** verifica que la conexión USB puede transmitir información con la suficiente velocidad. Si el test falla se debe conectar el LEAP Motion a otro puerto USB.
- **Lighting status:** busca fuentes de luz infrarroja demasiado brillantes en el campo de visión. Si falla intente bajar las persianas o las cortinas de las ventanas.
- **Smudge status:** detecta si se encuentra alguna mancha en el dispositivo. Si el test falla se debe limpiar la ventana.

En caso de que todo funcione correctamente, el LEAP Motion Control Panel deberá tener la forma de la Figura 3.7.

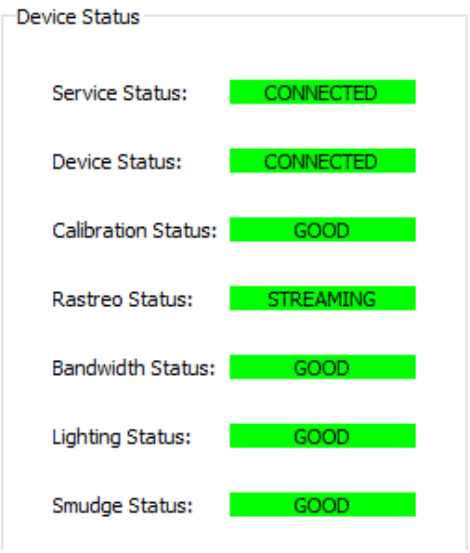

Figura 3.7. LEAP Motion funcionando correctamente

3.3. Aplicación: X-Plane 10

X-Plane es un simulador de vuelo publicado por Laminar Research desde 1995. Este simulador tiene versiones de escritorio para Windows, MacOS y Linux, a la vez que también puede ser obtenido en sus versiones móvil para Android, iOS y webOS desde 2009 [16]. El logotipo del programa puede verse en la Figura 3.8 [16].

Figura 3.8. Logotipo del programa X-Plane

En lo referente a los requisitos mínimos que debe tener el sistema para poder trabajar con el programa X-Plane, debe contar con una CPU Intel Core i3, i5 o i7 con 2 o más núcleos. Además de ser necesario un espacio en la memoria de 4 GB de RAM. Si se deseara descargar X-Plane 10 en Mac o Linux, el usuario debería emplear procesadores de Nvidia o de AMD previamente [17].

La razón por la que se emplea la versión de X-Plane 10 en lugar de la nueva versión disponible del simulador, siendo esta X-Plane 11, es que esta última no esta apoyada por *Flyinside*. Por tanto, no es compatible el uso de X-Plane con LEAP Motion. Sin embargo,

la versión de X-Plane 10 si que se encuentra apoyada por *Flyinside*, permitiendo que la conexión de X-Plane con LEAP Motion pueda ser llevada a cabo.

Durante el desarrollo del presente proyecto, se trabajará principalmente con una aeronave en particular, la CESNA 172 SP, pudiéndose trabajar con otras de las aeronaves disponibles si se desease.

3.3.1. Descripción y usos de la aplicación

X-Plane es el simulador de vuelo de uso comercial para ordenadores personales más potente y completo del mundo, proporcionando los modelos de vuelo más realistas de la actualidad. Esta aplicación, más que como un videojuego podría definirse como una herramienta de ingeniería que puede emplearse para predecir el comportamiento de cualquier aeronave en vuelo. Asimismo, se trata de una herramienta magnífica para formar pilotos, puesto que X-Plane cuenta con las herramientas necesarias que permiten volar el avión como en la realidad. Respecto a esto, la Administración Federal de Aviación (FAA) de los Estados Unidos de América ha autorizado su uso, con un hardware específico, para formar pilotos bajo las condiciones de vuelo instrumental, e incluso ha sido aprobado su uso para la obtención del Certificado de Transporte de Aerolínea [18].

Este simulador contiene dinámicas de vuelo tanto subsónicas como supersónicas, a la vez que incluye modelos de avión ya sean propulsados, a reacción, monomotor o multimotor (también pueden ser controlados veleros, helicópteros y VTOLs) [18].

El paquete de descarga incluye escenarios de toda la tierra, con más de 35.000 aeropuertos y más de 15 aeronaves base a las cuales se pueden incluir más [18]. La meteorología del programa también es muy variada, incluyendo tanto cielos despejados con alta visibilidad, como tormentas, turbulencias, ráfagas de viento, lluvia, nieve...

3.3.2. Principio de funcionamiento

El funcionamiento del simulador de vuelo X-Plane consiste en leer los datos geométricos de la aeronave seleccionada para luego averiguar como volará dicha aeronave. Esto se lleva a cabo mediante la conocida "Teoría del Elemento de Pala (TEP)", que supone la división del avión en partes muy pequeñas para averiguar las fuerzas aplicadas sobre cada elemento muchas veces por segundo. Estas fuerzas serán convertidas en aceleraciones, que permitirán obtener las velocidades y posiciones de la aeronave mediante un proceso de integración [19].

El simulador X-Plane pasa por los siguientes pasos para la reproducción del vuelo [19]:

- **Separación de elementos:** realizado únicamente durante la inicialización, el programa divide las alas, estabilizadores (vertical y horizontal) y hélices en un número finito de elementos.

- **Determinación de la velocidad:** se considera que las velocidades lineales y angulares de la aeronave, unidas a los brazos longitudinales, verticales y laterales de cada uno de los elementos en los que había quedado dividido la aeronave, permiten determinar las velocidades en los diferentes elementos.
- **Cálculo de los coeficientes:** los datos de las superficies aerodinámicas son bidimensionales, por lo que X-Plane aplica una reducción finita de la pendiente de sustentación del ala, la reducción CLmax del ala finita, la resistencia inducida por el ala finita y la reducción del momento del ala finita apropiada para la relación de aspecto, conicidad relación y barrido del ala, estabilizador horizontal, estabilizador vertical o pala de la hélice en cuestión. Los efectos de flujo compresible se consideran usando Prandtl-Glauert.
- **Fortalecimiento de las fuerzas:** mediante la utilización de los coeficientes, las áreas de los elementos y las presiones dinámicas, las fuerzas totales de la aeronave pueden ser obtenidas.
- **Repetición del proceso:** se repite el proceso desde el paso 2 al menos 15 veces por segundo.

Este método de cálculo de fuerzas de una aeronave es mucho más detallado, flexible y avanzado en comparación a otros modelos de vuelo empleados en simuladores de vuelo diferentes. Es común que otros simuladores de vuelo empleen el modelo conocido como "Derivadas de Estabilidad", esta técnica implica forzar el morro de la aeronave para que regrese a una posición centrada a lo largo del vuelo con una determinada aceleración por cada grado de desviación. Esta es una regla empírica, pero es demasiado simple para usarla, al no poder considerar efectos como la asimetría de vuelo debida a fallos de motores, efectos turbulentos, etc. Además, difícilmente puede incluir efectos como el drag transónico o la compresibilidad del flujo [19].

3.4. Lenguaje de programación: Java

El lenguaje de programación de Java se trata de una tecnología empleada durante el desarrollo de una gran cantidad de aplicaciones. Java es la base de prácticamente todos los tipos de aplicaciones de red, que podemos encontrar. Además, puede emplearse para el desarrollo y la distribución de diversas aplicaciones, juegos y demás contenidos basados en red y software de empresa.

Java ha sido probado, ajustado y ampliado por una comunidad de desarrolladores y arquitectos de aplicaciones. Java esta diseñado para permitir el desarrollo de aplicaciones portátiles de elevado rendimiento, pudiendo poner a disposición de todo el mundo aplicaciones en entornos heterogéneos, proporcionando más servicios y aumentando la productividad [20].

Por otra parte, java permite que las aplicaciones desarrolladas bajo su tecnología puedan ser multiplataforma. Además, existen una gran cantidad de bibliografía a la que

acudir en caso de necesidad. Finalmente, el fabricante del LEAP Motion nos proporciona las librerías necesarias en varios idiomas de programación, siendo uno de ellos Java. Es en este contexto en el que Java ha sido elegido como el lenguaje de programación de este proyecto.

3.5. Entorno de desarrollo: NetBeans IDE 8.2

Una vez el lenguaje de programación ha sido seleccionado, el siguiente paso es elegir el entorno de programación. En ese contexto se presenta el NetBeans IDE 8.2, cuyo logo se observa en la Figura 3.9 [21]. NeatBeans es un entorno de desarrollo libre, de código abierto y multiplataforma, basado en el lenguaje de Java.

Figura 3.9. Logotipo del entorno de desarrollo NetBeans

El paquete de NetBeans contiene lo necesario para el desarrollo de plugins y de aplicaciones, sin la necesidad de requerir SDKs adicionales. Esto permite que las aplicaciones descarguen módulos dinámicamente o que permitan a los usuarios la posibilidad de realizar las descargas o actualizaciones, dentro de la misma aplicación durante su ejecución, sin la necesidad de volver a actualizar toda la aplicación de nuevo.

En Junio de 2000, la empresa Sun Microsystems convirtió NetBeans en código abierto, la cual siguió siendo el patrocinador del proyecto hasta que en 2010 se convirtió en patrocinador del proyecto. Finalmente, en 2016 Oracle donó el código fuente de NetBeans a Apache Software Foundation. Este hecho causo que las versiones posteriores al Netbeans IDE 8.2 adoptasen el nombre de Apache Netbeans. En 2019 Apache NetBeans se convirtió en un proyecto de alto nivel dedicado a proporcionar productos de desarrollo de software que abordan las necesidades del usuario permitiéndole la producción de productos rápidos y eficientes. Actualmente, el IDE de NetBeans se encuentra con más de 18 millones de descargas y más de 800.000 desarrolladores [22].

3.6. Archivos de la carpeta: *AutoPilotFrame-sol*

Para poder llevar a cabo el desarrollo de una interfaz de comunicación UDP con el simulador de vuelo X-Plane 10, se ha recurrido a implementaciones ya desarrolladas anteriormente, y que son empleadas en distintas asignaturas de la UPV.

La carpeta «*AutoPilotFrame-sol*», contiene una librería desarrollada por Luiz Cantoni en su tesis doctoral [23], en la que se desarrolla una interfaz que permite conectar con X-Plane desde el lenguaje de programación de Java. Esta librería es acondicionada para el uso de la misma en NetBeans por los profesores Joan Vila y Ángel Rodas, para su empleo en las prácticas de la asignatura de segundo curso de máster «Sistemas de gestión de vuelo por computador» [24], que ellos mismos imparten, en la que se programa un autopiloto desde NetBeans para manejar el simulador X-Plane.

Estas librerías serán imprescindibles en el desarrollo de este proyecto, dado que permiten la lectura y escritura de datos desde NetBeans, para actuar sobre X-Plane.

4 | Proceso de desarrollo

Una vez la finalidad del proyecto ha sido presentada, así como los distintos materiales y métodos que son necesarios para el presente trabajo, se pretende desarrollar el proceso seguido durante la producción de este proyecto. Es necesario comprender varios aspectos principales antes de comenzar. Por un lado, dada la inexperiencia anterior al no haber trabajado con Java previamente, ha sido necesaria una primera etapa de aprendizaje de la plataforma para adaptarse a este lenguaje de programación y poder conocer las posibilidades que ofrece así como las posibles limitaciones que podríamos encontrar, lo cual no resultará un problema gracias a la gran cantidad de bibliografía que puede ser encontrada en internet. Por otra parte, la tecnología de reconocimiento gestual carece de tanta bibliografía sobre su uso, por tanto, el aprendizaje del mismo se llevará a cabo mediante la obtención de información proporcionada por el fabricante del dispositivo LEAP Motion.

A continuación se presentarán las diferentes fases en las que la producción del proyecto ha sido dividido. Dado que este proyecto requiere la fusión de varios programas entre sí, se tratará de comprender el funcionamiento de las diferentes unificaciones poco a poco sin tratar de mezclar los todos de golpe, y así, facilitar el proceso de familiarización y comprensión de los diferentes programas empleados. Por todo ello, se definirán tres fases.

En primer lugar, se intentará indagar en el comportamiento y las posibilidades que ofrece el controlador de LEAP Motion, mediante la producción de una aplicación en NetBeans que permita obtener distinta información sobre el lector de gestos, para poder comprender que nos ofrece el dispositivo y cuales podrán ser utilizadas para el desarrollo de la aplicación posterior.

En segundo lugar, se trabajará en detalle con el programa de X-Plane. En esta segunda fase se indagará sobre las funcionalidades que ofrece del simulador de vuelo seleccionado y sobre como pueden ser los datos del X-Plane leídos y escritos desde el entorno de desarrollo de Apache NetBeans IDE 12.3.

Finalmente, una vez se haya podido trabajar separadamente con LEAP Motion y X-Plane, comprendiendo las funcionalidades de ambos programas, se procederá con la creación de la aplicación definitiva que incluya la unión de todos los programas y materiales descritos previamente.

4.0.1. Interacción entre los materiales empleados

Una vez se hayan completado las distintas fases descritas se habrá conseguido cumplir con el objetivo del proyecto: la creación de una aplicación que permita el control del simulador X-Plane mediante el dispositivo LEAP Motion.

Para ello, el diagrama del equipo empleado se muestra en la Figura 4.1, donde en el ordenador principal, se lleva a cabo el desarrollo de la aplicación en NetBeans, así como la conexión del dispositivo LEAP Motion. Una vez ejecutada la aplicación desarrollada, esta enviará y recibirá los datos al segundo ordenador (mediante una conexión UDP), donde estará abierto el programa de X-Plane 10.

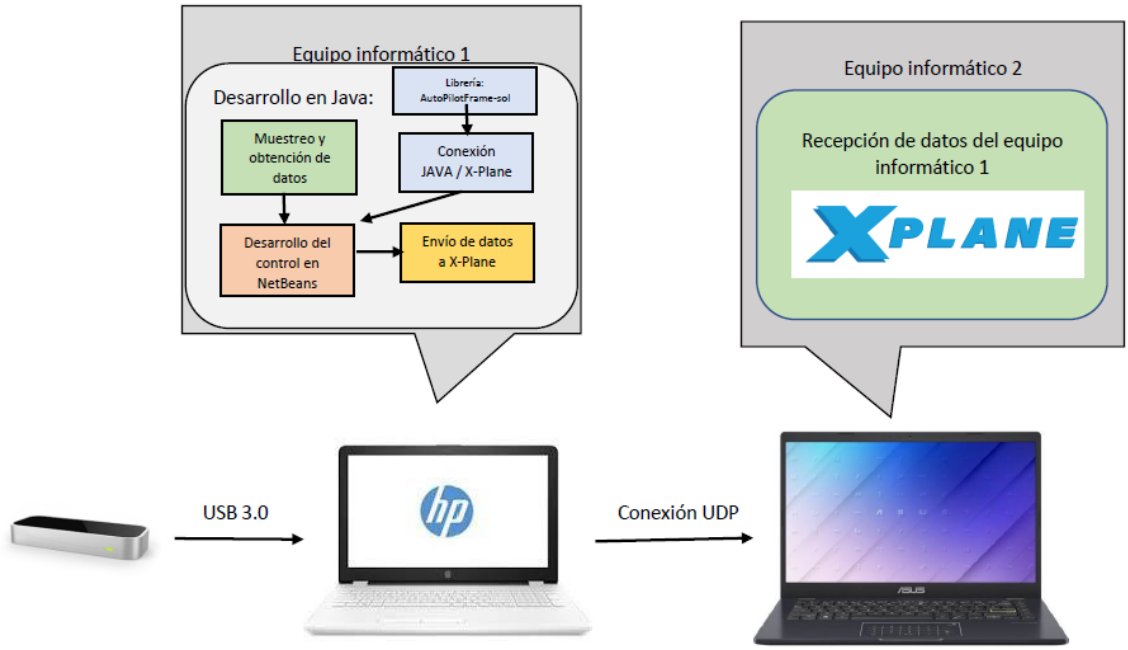

Figura 4.1. Diagrama de funcionamiento de la aplicación

A continuación, se procederá con una descripción más detallada de cada una de las tres fases en las que el proceso de desarrollo ha sido dividido.

4.1. Lectura de datos del controlador LEAP Motion

Como ha sido introducido anteriormente, esta primera fase del desarrollo de la aplicación final consta en el aprendizaje sobre las capacidades de lectura que ofrece el dispositivo LEAP Motion. Con dicha finalidad se desarrolla una aplicación del tipo «Hello World» para aprender a conectar el controlador y poder acceder a sus datos básicos de seguimiento.

Como fue mencionado en la Sección 3.2, el dispositivo LEAP Motion recibe y analiza

datos, específicamente para manos, dedos y brazos. Esto se produce mediante la comparación de esta información con un modelo interno de una mano humana, determinando el mejor ajuste. Estos datos son analizados y enviados a las aplicaciones habilitadas para el controlador LEAP Motion, donde el objeto «Frame» contiene toda la información (velocidad, posición, identidad ...) sobre los elementos detectados.

Por tanto, teniendo en cuenta la información proporcionada por *Ultraleap* sobre el SDK y las APIs del dispositivo LEAP Motion, se comienza el desarrollo de la aplicación.

4.1.1. Clases internas

La aplicación «*LEAPMotion_Sample.java*» consiste a nivel interno de una única clase que extiende la clase *Listener* de Java, que incluye cinco métodos: *OnInit*, *OnConnect*, *OnDisconnect*, *OnExit* y *OnFrame*.

- ***OnInit:*** mostrará en pantalla la palabra «*Initialized*» cuando se inicie la aplicación.
- ***OnConnect:*** indicará que el dispositivo LEAP Motion se encuentra conectado mediante la palabra «Connected».
- ***OnDisconnect:*** al contrario que el método *OnConnect*, mostrara el termino «Disconnected» cuando el dispositivo LEAP Motion no se encuentre conectado.
- ***OnExit:*** al salir de la aplicación aparecerá el mensaje «Exited».
- ***OnFrame:*** se empleará para leer los frames más recientes y mostrar los datos leídos junto al cartel «Processing».

El código de los cuatro primeros métodos se muestra en la Figura 4.2, mientras que el método *OnFrame* es más extenso y se explicará con más detalle a continuación.

4.1.2. Obtención de datos

Ahora se profundizará en el método *OnFrame*, el cual se encarga de mostrar en pantalla la lectura de los datos deseados. Para ello se debe tener muy en cuenta las posibilidades que nos ofrece el fabricante sobre este lector de gestos [25].

Por tanto, en la clase *OnFrame* se intenta obtener una gran variedad de datos sobre las manos, dedos... Con la finalidad de leerlos y observar los resultados, mediante la representación de los mismos con el uso de la función *System.out.println*.

Dado que el SDK proporcionado por el fabricante contiene un par de errores en algunas de sus funciones, como es el caso de: hands.isLeft(), hands.isRight() (que son funciones que sirven para identificar si la mano que se encuentra en visión del dispositivo es la derecha o la izquierda) o finger.type() (que sirve para detectar cuál es cada uno de los

```
public class PruebaLeap extends Listener {
    @Override
    public void onInit(Controller controller) {
        System.out.println("Initialized");
    }

    @Override
    public void onConnect(Controller controller) {
        System.out.println("Connected");
        controller.enableGesture(Gesture.Type.TYPE_SWIPE);
        controller.enableGesture(Gesture.Type.TYPE_CIRCLE);
        controller.enableGesture(Gesture.Type.TYPE_SCREEN_TAP);
        controller.enableGesture(Gesture.Type.TYPE_KEY_TAP);
    }

    @Override
    public void onDisconnect(Controller controller) {
        //Note: not dispatched when running in a debugger.
        System.out.println("Disconnected");
    }

    @Override
    public void onExit(Controller controller) {
        System.out.println("Exited");
    }
```

Figura 4.2. Métodos de inicialización del LEAP Motion

dedos de la mano), entre otros. Se ha indagado en busca de archivos de SDK anteriores para poder emplear dichas funciones en el presente proyecto. En caso de encontrase con el mismo problema, deben revisar el Apéndice B, donde se desarrolla con más detenimiento los fallos de la última versión, así como la solución encontrada.

Una vez presentado el problema actual del SDK y la solución empleada, se procede con el estudio del funcionamiento del controlador LEAP Motion y con las posibilidades de lectura que nos ofrece.

En primer lugar, cuando la ejecución de la aplicación se encuentre dentro del método *OnFrame*, se mostrará la palabra «Processing». Seguido de ello, se leerán los datos generales de cada frame detectado por la cámara, seguido de lecturas de datos más específicas.

Las lecturas generales de cada frame son leídas mediante los comandos de la Figura 4.3. El uso de los códigos empleados es el siguiente:

- ***frame.id():*** se trata de la identificación de cada uno de los marcos observados por la cámara.
- ***frame.timestamp():*** el tiempo que ha pasado en cada uno de los marcos estudiados desde un punto arbitrario del pasado.
- ***frame.hands().count():*** número de manos detectadas en el marco.
- ***frame.fingers().count():*** número de dedos detectados en el marco.

- ***frame.tools().count():*** número de objetos detectados en el marco.
- ***frame.gestures().count()):*** cantidad de gestos reconocidos en cada marco.

```
@Override
public void onFrame(Controller controller) {
    System.out.println("Processing");
    // Get the most recent frame and report some basic information
    Frame frame = controller.frame();

    System.out.println("Frame id: " + frame.id()
            + " timestamp: " + frame.timestamp()
            + ", hands: " + frame.hands().count()
            + ", fingers: " + frame.fingers().count()
            + ", tools: " + frame.tools().count()
            + ", gestures " + frame.gestures().count());
```

Figura 4.3. Código de las lecturas de cada frame

En segundo lugar, se procederá con una lectura más específica sobre la información que puede ser extraída de las manos y de los dedos.

Para la mano, se determinará en primer lugar que mano está posicionada sobre el dispositivo LEAP Motion, seguido de datos específicos de la mano, entre los cuales se incluirá la definición de vectores sobre la posición, velocidad y dirección de esta. Para los dedos, se detectará cuáles se encuentran extendidos y se identificarán los distintos tipos de dedos encontrados en el cuadro, definidos con sus características geométricas. Para la lectura de datos, mostrada en las Figuras 4.4 y 4.5, se emplearán las siguientes funciones principales:

- ***hand.isLeft():*** indica si la mano localizada en el marco de la cámara es la izquierda, se trata de un *boolean* que devuelve *true* si así es. Se empleará para definir una cadena que devuelva si la mano detectada es la derecha o la izquierda.
- ***hand.palmNormal():*** define un vector normal a la palma de la mano, que permitirá detectar el alabeo con la extensión *.roll()*.
- ***hand.direction():*** vector unitario que señala en la misma dirección que el vector normal a la mano. Este vector nos permitirá obtener información sobre el cabeceo y la guiñada con las extensiones *.pitch()* y *.yaw()*.
- ***hand.palmPosition():*** permite detectar la posición de cada una de las tres coordenadas espaciales en milímetros.
- ***hand.palmVelocity():*** calcula la tasa de cambio de la posición de la palma de la mano en milímetros por segundo.
- ***hand.fingers().extended():*** detecta el número de dedos en el cuadro que se encuentran extendidos.

- ***hand.grabStrength():*** devuelve un número entre 0 y 1 dependiendo de lo abierta o cerrada que este la palma de la mano, simulando el acto de agarrar un objeto, 1 siendo completamente cerrada.
- ***finger.type():*** este código indica el nombre de cada uno de los 5 dedos de la mano, lo cuál permitirá obtener información sobre ellos.
- ***finger.length():*** devuelve la longitud de cada uno de los dedos detectados.
- ***finger.width():*** devuelve el grosor de cada uno de los dedos detectados.

```
//Get hands
for (Hand hand : frame.hands()) {
    String handType = hand.isLeft() ? "Left hand" : "Right hand";

    // Get the hand's normal vector and lists needed
    Vector normal = hand.palmNormal();
    Vector direction = hand.direction();
    Vector position = hand.palmPosition();
    Vector velocity = hand.palmVelocity();
    FingerList extendedFingerList = hand.fingers().extended();

    System.out.println(handType + ", id: " + hand.id()
            + ", palm position: " + hand.palmPosition()
            + ", Grab position: " + hand.grabStrength()
            + ", Extended fingers: " + +extendedFingerList.count());
    System.out.println("  pitch: " + Math.toDegrees(direction.pitch()) + " degrees, "
            + "roll: " + Math.toDegrees(normal.roll()) + " degrees, "
            + "yaw: " + Math.toDegrees(direction.yaw()) + " degrees");
    System.out.println("X: " + position.getX() + " mm, "
            + "Y: " + position.getY() + " mm, "
            + "Z: " + position.getZ() + " mm");
    System.out.println("Vx: " + velocity.getX() + " mm/s, "
            + "Vy: " + velocity.getY() + " mm/s, "
            + "Vz: " + velocity.getZ() + " mm/s");
    System.out.println("Normal Vector: " + hand.palmNormal());
```

Figura 4.4. Código de las lecturas relacionadas con las manos

```
// Get fingers
for (Finger finger : hand.fingers()) {
    System.out.println("    " + finger.type() + ", id: " + finger.id()
            + ", length: " + finger.length()
            + "mm, width: " + finger.width() + "mm ");

}
```

Figura 4.5. Código de las lecturas relacionadas con los dedos

Para poder comprender correctamente los resultados obtenidos en esta sección, será necesario definir el sistema de coordenadas que emplea el dispositivo LEAP Motion, el cual emplea un sistema de coordenadas cartesiano para diestros. El origen del mismo se encuentra localizado directamente en la parte superior del controlador LEAP Motion. Los ejes X y Z se encuentran localizados en el plano horizontal, siendo el eje X el que mantiene la longitud longitudinal paralela al lado largo del dispositivo, con valores positivos

en la dirección a derechas. Por su parte, el eje Y es vertical la base del controlador, incrementando en valor positivo hacia arriba. Finalmente, el eje Z es positivo en dirección al usuario. La Figura 4.6 muestra los ejes que emplea el dispositivo LEAP Motion [13].

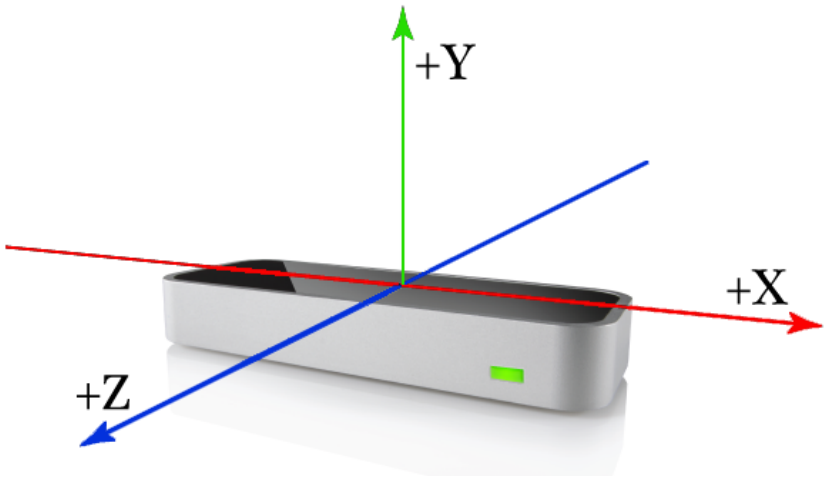

Figura 4.6. Sistema de coordenadas del dispositivo LEAP Motion

A parte de las lecturas de manos y dedos, también se ha tratado de indagar en aspectos sobre la detección de objetos (como un lápiz) o de los gestos básicos del LEAP Motion (*CircleGesture*, *KeyTapGesture*, *ScreenTapGesture*, y *SwipeGesture*). Pero como se explica en el Apéndice B, la nueva versión del SDK del LEAP Motion tiene errores de lectura en estas funciones, por tanto, no se ha podido extraer información sobre estos aspectos.

Finalmente, cabe mencionar que para finalizar con la ejecución de la aplicación tan solo debe ser pulsado cualquier botón. De este modo, se dejarán de recoger los datos del dispositivo LEAP Motion y NetBeans dejará de trabajar.

4.1.3. Lectura de datos

En la Figura 4.7, se muestra un ejemplo de lectura proporcionado mediante la aplicación presentada. En este caso, se observa como en el frame 38648, se encontraban las dos manos en el rango de visión de la cámara.

Sin entrar en demasiado detalle sobre las posiciones y velocidades de las manos, se puede contemplar que la mano izquierda se encuentra completamente cerrada, dado que no hay dedos extendidos y que el valor del término *Grab position* es 1. En cuanto a la mano derecha, esta tiene 4 dedos completamente extendidos, y además, si se observa los datos del vector normal, la mano se encuentra con la palma mirando hacia arriba, dado que el signo del eje y es positivo en esta mano.

```
Processing
Frame id: 38648 timestamp: 106517464736, hands: 2, fingers: 10, tools: 0, gestures 0
Left hand, id: 29, palm position: (-62.4065, 178.458, -10.6519), Grab position: 1.0, Extended fingers: 0
  pitch: -9.78526795309783 degrees, roll: -9.540634482690066 degrees, yaw: 16.01695069803081 degrees
X: -62.406467 mm, Y: 178.45798 mm, Z: -10.651901 mm
Vx: -4.337398 mm/s, Vy: 19.456554 mm/s, Vz: 5.710079 mm/s
Normal Vector: (-0.164517, -0.978851, 0.121591)
    TYPE_THUMB, id: 290, length: 47.24686mm, width: 17.920433mm
    TYPE_INDEX, id: 291, length: 54.026886mm, width: 17.117598mm
    TYPE_MIDDLE, id: 292, length: 61.814407mm, width: 16.811756mm
    TYPE_RING, id: 293, length: 59.373173mm, width: 15.997452mm
    TYPE_PINKY, id: 294, length: 46.596268mm, width: 14.210187mm
Right hand, id: 30, palm position: (97.9681, 152.564, 48.5377), Grab position: 0.0, Extended fingers: 4
  pitch: 30.423021770881626 degrees, roll: -162.6469395356768 degrees, yaw: -3.4967748962998537 degrees
X: 97.968056 mm, Y: 152.56436 mm, Z: 48.537678 mm
Vx: 40.12802 mm/s, Vy: 22.830086 mm/s, Vz: -205.36687 mm/s
Normal Vector: (-0.258145, 0.826113, 0.500898)
    TYPE_THUMB, id: 300, length: 48.26175mm, width: 17.920433mm
    TYPE_INDEX, id: 301, length: 58.14904mm, width: 17.117598mm
    TYPE_MIDDLE, id: 302, length: 63.84293mm, width: 16.811756mm
    TYPE_RING, id: 303, length: 60.79486mm, width: 15.997452mm
    TYPE_PINKY, id: 304, length: 47.074066mm, width: 14.210187mm
```

Figura 4.7. Lectura de los datos del LEAP Motion mediante la aplicación de NetBeans

Con esto, la primera fase del proyecto ha sido finalizada, habiéndose cumplido con el objetivo de familiarizarse con el controlador LEAP Motion y con las diferentes oportunidades que este nos ofrece para la realización de la aplicación final.

4.2. Control del simulador de vuelo X-Plane 10

En esta segunda fase del desarrollo de la aplicación final, se pretende conocer más información acerca del funcionamiento interno del simulador de vuelo y como se puede leer y enviar información a este desde otra aplicación, siendo este caso desde NetBeans. En esta sección, se explicará el proceso que debe seguirse para dicha finalidad paso a paso.

4.2.1. Entrada y salida de datos

En primer lugar, se debe comprender donde pueden ser leída o enviada información dentro de la aplicación de X-Plane. En este contexto introducimos la ventana «Entrada y salida de datos», dentro de los ajustes de la aplicación. Esta ventana está mostrada en la Figura 4.8, donde se encuentran numerados las distintos campos de datos sobre las cuales se puede actuar. Cada dato esta situado junto a 4 cajas con funciones diferentes: Internet a través de UDP, archivo en disco «data.txt» (esta opción enviará los datos de salida al archivo «data.txt» ubicado en la carpeta principal del programa), Visualización ver gráfica en ver datos (se mostrarán en el Gráfico de Salida de Datos, al que se accede en el menú Desarrollador) y cabina durante el vuelo. Para este proyecto, será necesaria la activación de los campos en las casillas de Internet a través de UDP, para permitir el envío de datagramas a través de la red sin que haya conexión entre emisor y receptor. Los datagramas serán enviados especificando un puerto de destino.

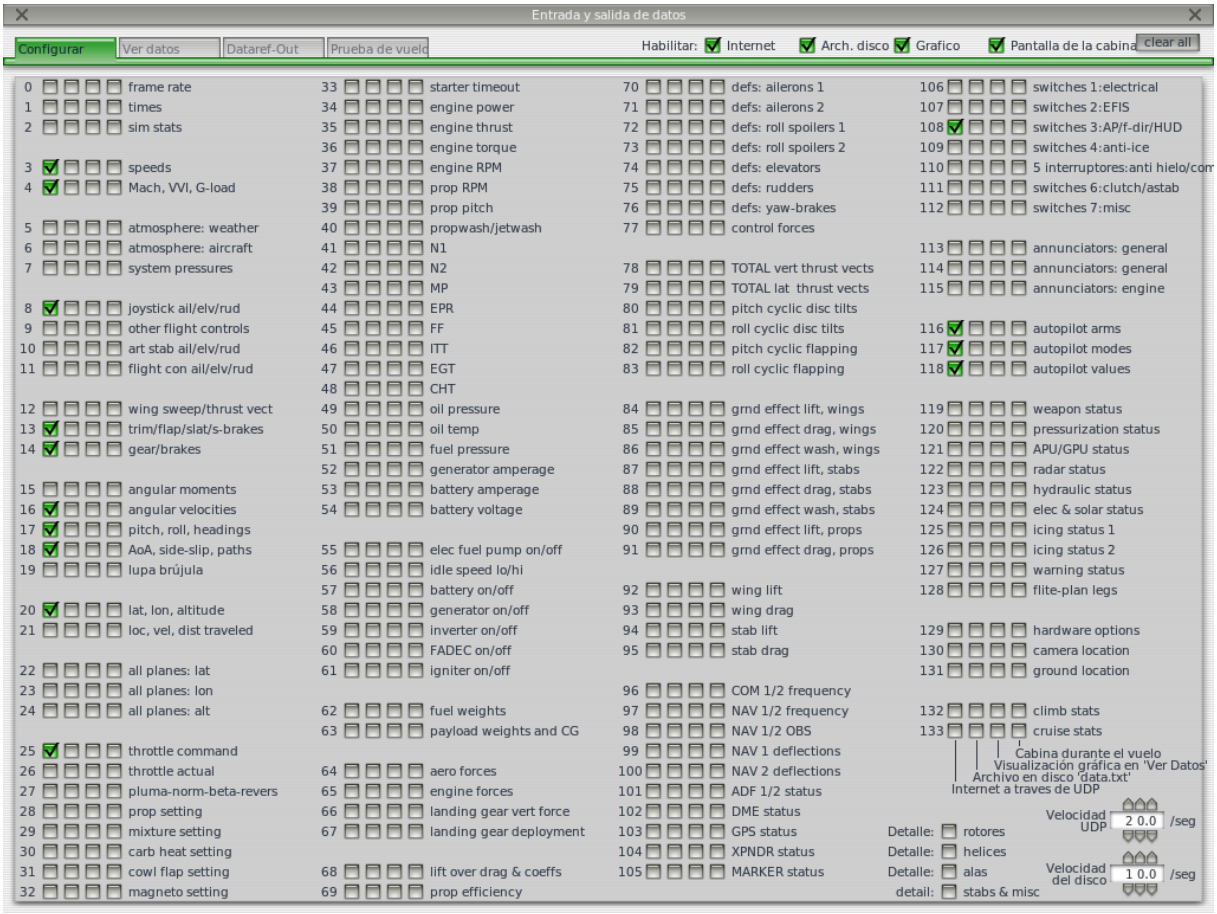

Figura 4.8. Ventana de entrada y salida de datos de X-Plane 10

Antes de proceder con la identificación de cada uno de los campos de datagramas, se mostraran los puertos UDP que usa X-Plane, mostrados en las Figuras 4.9 y 4.10, donde se observa como el puerto de recepción de información de X-Plane es el 49000 y el que X-Plane emplea para la salida de datos es el 9500. Se observa como en la Figura 4.10 se introduce la dirección IP del primer ordenador, es decir en el cual se desarrolla la aplicación en NetBeans.

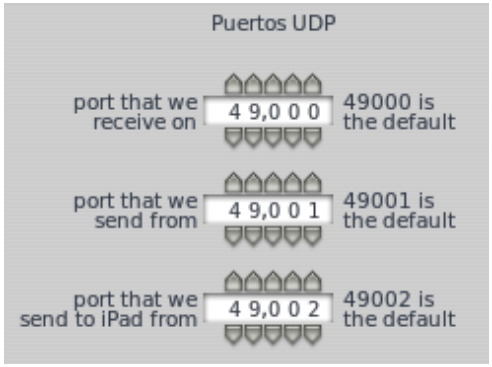

Figura 4.9. Puertos UDP para envío de datos a X-Plane

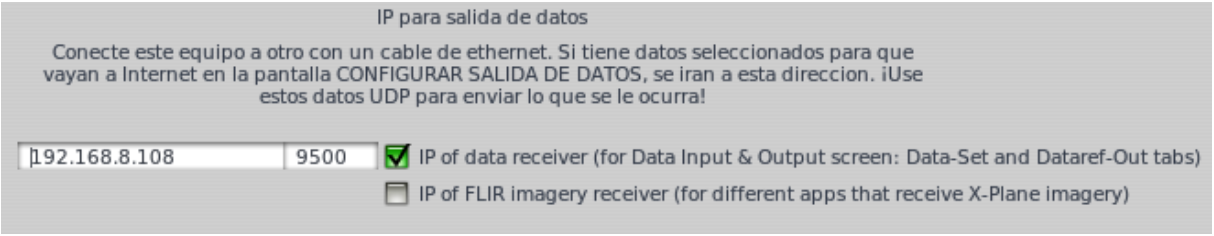

Figura 4.10. Puertos UDP para la recepción de datos a X-Plane

El siguiente paso es comprender que significa o que contiene cada uno de los campos de datos mostrados en la Figura 4.8. Se encuentran más de 100 campos de datos diferentes en el panel de entrada y salida de datos de X-Plane, y cada uno de ellos cuenta con hasta 8 datos distintos. Descubrir que datos hay localizados en cada uno de los campos presentes en la Figura 4.8 puede resultar laborioso, pero hay una forma sencilla de conocerlos.

Para ello, será necesario marcar la segunda casilla del panel de entrada y salida de datos localizado mostrado en la Figura 4.8, de esta manera, como ha sido mencionado previamente, el programa escribirá la información seleccionada en el archivo «data.txt», no solo mostrando los resultados sino también los nombres de las variables escritas. A modo de ejemplo, se han extraído los datos del campo 20 (correspondiente con «lat,lon,altitude») y han sido creada la Tabla 4.1 con los datos escritos en el archivo «data.txt».

lat,deg	lon,deg	alt,ftmsl	alt,ftagl	on,runwy	alt,ind	lat,south	lon,west
47.46363	-122.30775	397.36902	0.06873	1.00000	397.30035	46.00000	-124.00000
47.46363	-122.30775	397.43628	0.13601	1.00000	397.3690	46.00000	-124.00000

Tabla 4.1. Tabla obtenida del archivo data.txt

De esta manera podrán ser descubiertos todos los datos localizados en el interior de los campos, para poder decidir sobre cuales actuaremos, ya bien sea para su lectura o modificación.

Ahora que ya pueden ser conocidos todos los datos que nos ofrece X-Plane, es necesario conocer que información guarda cada uno de ellos y así conocer su utilidad. Para ello, deberemos acudir a los «DataRefs» de *xsquawkbox* [26], que nos permitirá conocer a que se corresponde cada uno de los nombres descubiertos mediante el paso anterior. Estos nombre se encuentran en la página de *xsquawkbox* con información sobre: el tipo de dato que es, si este dato puede ser escrito o solo leído, las unidades del mismo y se encuentra acompañado por una breve descripción, indicando su funcionamiento y como modificarlo.

Para facilitar la comprensión de como es el proceso completo de actuación sobre las variables de X-Plane de una manera sencilla, se continuará con el mismo caso de la Tabla 4.1. En la Tabla 4.2 [26], se muestra un ejemplo de como pude ser encontrada la información sobre las variables que se desean analizar en la página de *Squawkbox*. En este caso, se estudian únicamente la latitud y la longitud. Puede observarse como estas variables son del tipo «*double*», la cual se usa para obtener una mayor precisión que la que se obtiene con las variables del tipo «*float*» al poder contener más dígitos

significativos. Además, es importante notar que esta variable solo puede ser leída y no se podrá actuar directamente sobre ella. Finalmente, se aprecia que como puede esperarse de estas variables, se encuentran expresadas en grados.

Name	Type	Version	Writable	Units	Description
latitude	double	660+	no	degrees	The latitude of the aircraft
longitude	double	660+	no	degrees	The longitude of the aircraft

Tabla 4.2. Tabla obtenida a partir de la página de *Squawkbox*

4.2.2. Conexión NetBeans con X-Plane

Una vez se conoce el funcionamiento directo de X-Plane y cuáles son las variables que este posee, el siguiente paso consiste en aprender como pueden ser leídos estos datos desde otro programa. En esta sección, se procederá con la explicación sobre como debe de ser llevado este proceso para modificar y leer datos de X-Plane desde una aplicación creada en NetBeans.

La conexión de NetBeans con X-Plane se realiza mediante los archivos de la carpeta denominada como «*AutoPilotFrame-sol*», la cual contiene las librerías de Luiz Cantoni [23]. Los archivos más importantes de esta carpeta, enfocados a la correcta interacción con el programa de X-Plane serán comentados.

En primer lugar, es necesario realizar la conexión a través de puertos UDP como ha sido introducido previamente. Para ello se hace uso del archivo «*XPlaneInterface.java*», que se encarga de conectar correctamente con X-Plane, introduciendo la dirección IP de donde esta siendo ejecutado X-Plane y especifica los puertos UDP para recibir y enviar información al simulador de vuelo, mostrados en las Figuras 4.9 y 4.10.

En segundo lugar, siguiendo el procedimiento completo explicado en la Sección 4.2.1, se decidirán todas las variables que puedan llegar a ser empleadas. Todas estas variables deberán ser escritas en el archivo XML [1] «DATAConfigGroup.xml» (para su modificación se recomienda el uso de la aplicación Notepad++). En este archivo se definirán las variables deseadas siguiendo una estructura determinada, para permitir su lectura posterior desde NetBeans.

La Figura 4.11 muestra una captura de dicho archivo para mostrar la estructura que debe tener la definición de variables. A continuación, se explicará el procedimiento que debe ser seguido para la correcta introducción de nuevos datagramas.

Como se observa en la esquina superior izquierda de la Figura 4.11, ese conjunto particular de variables ha sido agrupado bajo el nombre de *position*, lo que solo tiene la finalidad de estructurar los datos y conocer donde encontrarlos de una forma más sencilla al seguir un orden. En cuanto a cada una de las variables se encuentran en este archivo,

[1] XML es el acrónimo de Extensible Markup Language, es decir, es un lenguaje de marcado que define un conjunto de reglas para la codificación de documentos

deben ser definidas mediante la especificación de una serie de campos. Los campos que deben ser rellenados son los siguientes:

- ***name:*** se debe introducir el nombre de la variable que se desea definir, no tiene porque coincidir con el nombre de la variable que se obtenía en el archivo «data.txt».
- ***message:*** hace referencia al grupo de datagramas, de la Figura 4.8, al que pertenece el dato que se desea leer.
- ***parameter:*** para obtener que valor debe ser introducido en este campo, debemos retomar otra vez el proceso de lectura del archivo «data.txt». Dado que cada campo de datagramas incluía hasta 8 datos, el valor que debe ser introducido en *parameter* debe ir entre 0 y 7, correspondiéndose el valor 0 con el primer término que aparecerá en «data.txt».
- ***readOnly:*** se debe introducir *true* si la variable solo puede ser leída, y *false* si también se puede escribir en ella.

```
<DATAGroup name="position">
    <Entries>
        <DATA name="ias" message="3" parameter="0" readOnly="true" xplaneName="" unit="" convertTo="" desc=""></DATA>
        <DATA name="eas" message="3" parameter="1" readOnly="true" xplaneName="" unit="" convertTo="" desc=""></DATA>
        <DATA name="tas" message="3" parameter="2" readOnly="true" xplaneName="" unit="" convertTo="" desc=""></DATA>
        <DATA name="gs" message="3" parameter="3" readOnly="true" xplaneName="" unit="" convertTo="" desc=""></DATA>
        <DATA name="vs" message="4" parameter="2" readOnly="false" xplaneName="" unit="" convertTo="" desc=""></DATA>
        <DATA name="lat" message="20" parameter="0" readOnly="true" xplaneName="" unit="" convertTo="" desc=""></DATA>
        <DATA name="lon" message="20" parameter="1" readOnly="true" xplaneName="" unit="" convertTo="" desc=""></DATA>
        <DATA name="altMsl" message="20" parameter="2" readOnly="true" xplaneName="" unit="" convertTo="" desc=""></DATA>
        <DATA name="altAgl" message="20" parameter="3" readOnly="true" xplaneName="" unit="" convertTo="" desc=""></DATA>
        <DATA name="altInd" message="20" parameter="5" readOnly="true" xplaneName="" unit="" convertTo="" desc=""></DATA>
        <DATA name="X" message="21" parameter="0" readOnly="false" xplaneName="" unit="" convertTo="" desc=""></DATA>
        <DATA name="Y" message="21" parameter="1" readOnly="false" xplaneName="" unit="" convertTo="" desc=""></DATA>
        <DATA name="Z" message="21" parameter="2" readOnly="false" xplaneName="" unit="" convertTo="" desc=""></DATA>
        <DATA name="vX" message="21" parameter="3" readOnly="false" xplaneName="" unit="" convertTo="" desc=""></DATA>
        <DATA name="vY" message="21" parameter="4" readOnly="false" xplaneName="" unit="" convertTo="" desc=""></DATA>
        <DATA name="vZ" message="21" parameter="5" readOnly="false" xplaneName="" unit="" convertTo="" desc=""></DATA>
        <DATA name="distNm" message="21" parameter="7" readOnly="false" xplaneName="" unit="" convertTo="" desc=""></DATA>
    </Entries>
</DATAGroup>
```

Figura 4.11. Definición de datos en el archivo «DATAConfigGroup.xml»

Una vez ha sido asegurada la conexión entre los puertos UDP y las diferentes variables han sido definidas en el archivo «DATAConfigGroup.xml», el siguiente paso es pensar que datos deseamos obtener de X-Plane. Conocidos los datos sobre los que queremos obtener información, los dividiremos en dos grupos (por mantener mantener una estructura ordenada y localizar fácilmente lo que se necesite entre los archivos): datos que solo pueden ser leídos y aquellos sobre los que se puede escribir. En el archivo «*DataXplane.java*» se definirán aquellas variables que solo pueden ser leídas, como es el caso de la altitud y longitud como se ha visto en la Tabla 4.2, mientras que en el archivo «*ActionXplane.java*» se definirán las variables sobre las que sí se puede escribir, como será el caso de los controles de la aeronave. El objetivo de estos archivos no es solo el de definir las variables, sino el de también crear las funciones *getter* y *setter*, que permiten leer o modificar la información almacenada en las mismas, respectivamente.

Finalmente, la conexión de NetBeans con X-Plane se lleva a cabo con la aplicación «*XplaneInputOutput.java*». Esta aplicación tiene dos funciones principales.

Por un lado define los puertos y conecta correctamente con X-Plane, haciendo uso del archivo «*XplaneInputOutput.java*», mediante el código de la Figura 4.12, que como se observa como se introduce el valor del UDP de 49000, a la vez que se introducen los distintos campos de la Figura 4.8 que serán necesarios para la obtención de variables. Aparte, en el caso de la Figura 4.12, el primer término introducido en la función «*new XPlaneInterface*», el correspondiente a 192.168.8.105, hace referencia a que la información será enviada a la dirección IP del servidor del segundo ordenador, es decir, en el que se encontrará abierto el programa de X-Plane 10.

```
public XplaneInputOutput() {

    try {
        //xpi = new XPlaneInterface("127.0.0.1", 9500, 49000, "DATAGroupConfig.xml");
        xpi = new XPlaneInterface("192.168.8.105", 9500, 49000, "DATAGroupConfig.xml");
        xpi.unregisterDATAMessages("*");
        xpi.registerDATAMessages("3,4,8,13,14,16,17,18,20,25,108,116,117,118"); // 20 lat lon alt replaces  22 23 24
        xpi.startReceiving();
        try {
            Thread.sleep(1000); // wait a few before send actions
        } catch (InterruptedException ex) {
            Logger.getLogger(ControlPanel.class.getName()).log(Level.SEVERE, null, ex);
        }
    } catch (SocketException ex) {
        Logger.getLogger(ControlPanel.class.getName()).log(Level.SEVERE, null, ex);
    } catch (UnknownHostException ex) {
        Logger.getLogger(ControlPanel.class.getName()).log(Level.SEVERE, null, ex);
    }
    data = new DataXplane();

}
```

Figura 4.12. Parte 1 del código del archivo «*XplaneInputOutput.java*»

Mientras que las Figuras 4.13 y 4.14, muestran el código que relaciona la creación de las variables, explicada previamente, con los datos obtenidos de X-Plane mediante la definición de variables del archivo «DATAConfigGroup.xml».

```
public DataXplane read() {

    data.setLat(xpi.getValue("position.lat"));
    data.setLon(xpi.getValue("position.lon"));
    data.setAlt(xpi.getValue("position.altMsl"));
    data.setRoll(xpi.getValue("orientation.roll"));
    data.setPitch(xpi.getValue("orientation.pitch"));
    data.setHead(xpi.getValue("orientation.headingTrue"));
    data.setAoa(xpi.getValue("orientation.alfa"));
    data.setBeta(xpi.getValue("orientation.beta"));
    data.setVpath(xpi.getValue("orientation.vpath"));
    data.setHpath(xpi.getValue("orientation.hpath"));
    data.setBrake(xpi.getValue("controls.brake"));
    data.setIas(xpi.getValue("position.ias"));
    data.setTas(xpi.getValue("position.tas"));
    data.setGs(xpi.getValue("position.gs"));
    data.setVs(xpi.getValue("position.vs"));
    return data;

}
```

Figura 4.13. Parte 2 del código del archivo «*XplaneInputOutput.java*»

```
public void write(ActionXplane action) {

    xpi.setValue("controls.aileronsPosition", action.getAilerons());
    xpi.setValue("controls.elevatorsPosition", action.getElevators());
    xpi.setValue("controls.rudderPosition", action.getRudder());
    xpi.setValue("engine.throttleCommandl", action.getThrottle());
    xpi.setValue("controls.brake", action.getBrake());
    xpi.setValue("controls.flapHandl", action.getFlaps());
    xpi.setValue("autopilot.appArm", action.getGlideslope());
    xpi.setValue("autopilot.altArm", action.getAltArm());
    xpi.setValue("autopilot.navArm", action.getLocalizer());
    xpi.setValue("autopilot.fdirMode", action.getFdirmode());
    xpi.setValue("autopilot.heading", action.getHeadingValue());
    xpi.setValue("autopilot.vvi", action.getVviValue());
    xpi.setValue("autopilot.useAlt", action.getUseAlt());
    xpi.setValue("autopilot.modeHeading", action.getModeheading());
    xpi.setValue("autopilot.modeAlt", action.getModealt());
}
```

Figura 4.14. Parte 3 del código del archivo «*XplaneInputOutput.java*»

4.2.3. Desarrollo de aplicación

Una vez ha sido conseguida la sincronización de NetBeans con X-Plane, se procede con una realización de una simple cabina que permita modificar los controles básico de una aeronave, que permita hacer volar la aeronave manualmente, así como, la activación del autopiloto de la aeronave.

Esta aplicación creará un cuadro de control simulando la cabina de la aeronave de forma muy sencilla, conteniendo esta tres tipos de controles distintos. Por un lado, se han incluido botones que permitirán la activación y desactivación de determinadas variables (como es el caso de los frenos o la selección del autopiloto). Por otra parte, se incluyen deslizadores para controlar el desplazamiento de los controles básicos de la aeronave (entre los que se incluyen: flaps, elevadores, alerones, palanca de gases...). Finalmente, se incluye un *spinner* para variar el rumbo seleccionado para el autopiloto. En la Figura 4.15, se muestra la previsualización del diseño de la cabina.

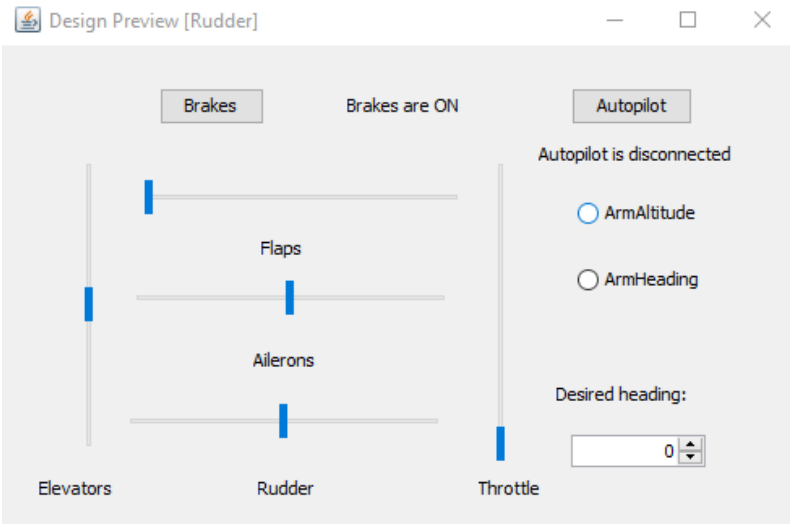

Figura 4.15. Simulador de la cabina de la aeronave

A continuación se explicará el proceso sencillo que debe seguirse para poder controlar la aeronave del simulador de vuelo mediante el uso de esta cabina.

Para la realización del despegue, se comenzará quitando los frenos (*brakes*) y se seguirá con el incremento de la palanca de gases (*throttle*). Una vez arranque la aeronave y vaya acelerando en tierra, se deberá ir jugando con la posición del deslizador del empenaje vertical (*rudder*), hasta que se obtenga una velocidad que permita el despegue con una deflexión en los flaps y en el elevador.

Una vez en el aire, se deben modificar los controles ligeramente para tratar de mantener la dirección deseada. Un ejemplo de como se ve en el simulador X-Plane se encuentra en la Figura 4.16.

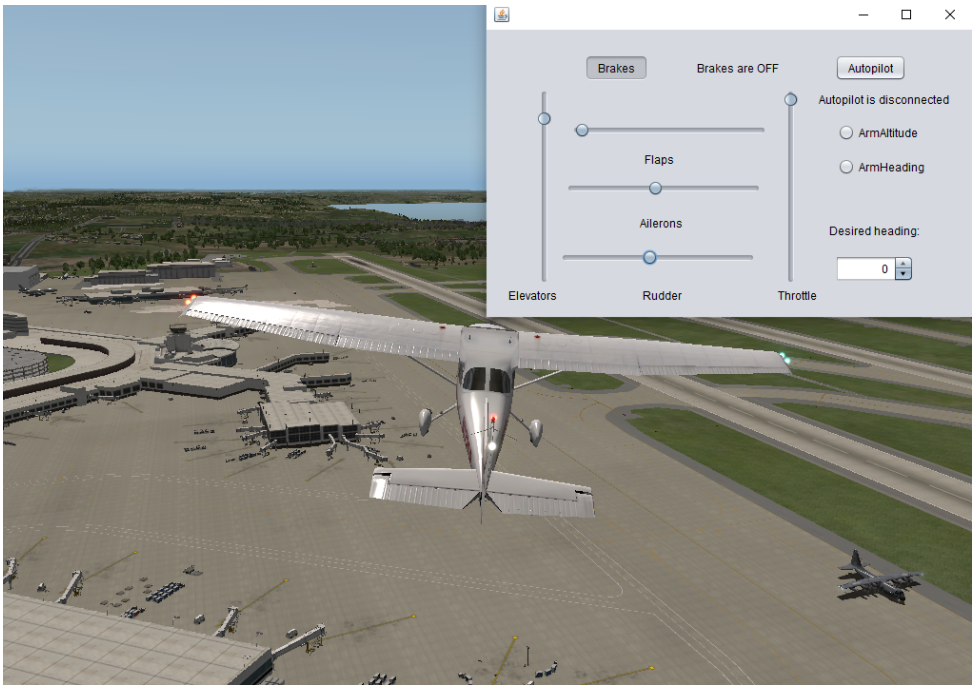

Figura 4.16. Ejemplo de control mediante la aplicación «*Rudder.java*»

Para la activación del autopiloto, se armarán el rumbo en primer lugar, para que luego al activar el autopiloto, este funcione correctamente y siga el rumbo que se establezca en el *spinner*. La activación del autopiloto, permite que la aeronave sea pilotada de forma automática sin necesidad de ejercer ningún otro tipo de control sobre ella, lo único es que en la configuración en la que se encuentra actualmente, la altitud seguirá aumentando. Por lo que una vez se alcance la altitud de crucero deseada el botón de armado de la altitud debe ser activado para que la aeronave mantenga dicha altitud. Un ejemplo de ello se muestra en la Figura 4.17, donde el autopiloto establece que la aeronave siga un rumbo de 140° en el caso mostrado.

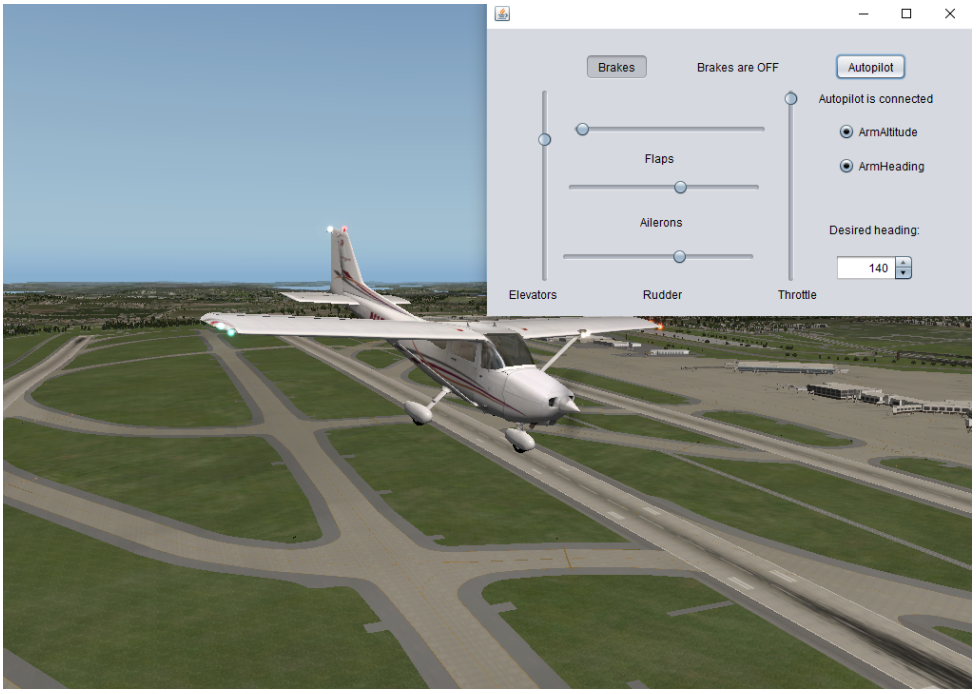

Figura 4.17. Ejemplo de activación del autopiloto mediante la aplicación «*Rudder.java*»

4.3. Aplicación final: Control del simulador X-Plane mediante el uso de LEAP Motion

Finalmente, una vez se comprende el funcionamiento del dispositivo LEAP Motion y del envío de información al simulador de X-Plane mediante NetBeans, es posible comenzar con el desarrollo de la aplicación que controle el simulador X-Plane mediante el dispositivo LEAP Motion.

En esta aplicación se pretende conseguir el desarrollo de una aplicación que no sea únicamente funcional, sino que también sea agradable de usar y ofrezca una una mejoría en los conocimientos y habilidades del usuario.

Para ello se ha desarrollado la aplicación «*Conexión*» que permite el control de la aeronave desde la visión interna de la cabina, al necesitar que el usuario observe los instrumentos principales de la cabina, como son: el indicador de altitud, de rumbo, de giro o de velocidad vertical, junto con el altímetro. Además, a sido creado un pequeño panel de control en la aplicación «*CabinaLEAPXPLANE*» con la finalidad de que el usuario pueda observar de forma rápida diferentes condiciones de vuelo en las que se encuentra la aeronave, los controles de la misma y información sobre las manos detectadas por el dispositivo LEAP Motion. De esta forma uno puede asegurarse del correcto funcionamiento del LEAP Motion y de X-Plane.

4.3.1. Problemas encontrados

El principal problema encontrado durante el desarrollo de la aplicación era el hecho de que pese a funcionar correctamente el LEAP Motion, siendo capaz de leer todos los datos requeridos, la lectura de datos se bloqueaba y, por tanto, el simulador de vuelo no recibía ningún dato.

Inicialmente, esto no sucedía cuando se empleaban las SDKs proporcionadas por *Ultraleap*, pero como ha sido comentado en la Sección 4.1.2, estas no funcionaban correctamente y aparecían diversos fallos, que fueron solucionados con el procedimiento explicado en el Apéndice B. Pero cuando se introducían los archivos mencionados, al abrir el programa de X-Plane 10 y ejecutar cualquier aplicación de Netbeans que incluyese el uso del controlador LEAP Motion, no se obtenía ninguna lectura.

La manera de solucionar dicho problema ha sido la inclusión en el proyecto de un segundo ordenador. De esta forma se deberá tener abierto el programa de X-Plane en el segundo monitor, mientras el dispositivo LEAP Motion quedará conectado con el primero. Es por ello que en la aplicación «*XplaneInputOutput.java*» el valor de la dirección IP se corresponde con el segundo ordenador.

De esta forma, es posible llevar a cabo la conexión entre el dispositivo LEAP Motion con el simulador de vuelo X-Plane.

Por otra parte, en caso de que se produzcan bloqueos en la lectura de los gesto o que la aplicación tarde en inicializarse, la causa se debe principalmente a que el cortafuegos del antivirus descargado en el ordenador cause un bloque, por tanto, la desactivación del mismo conllevará al buen funcionamiento de la aplicación.

Finalmente, si todo funciona correctamente durante un tiempo y al cabo de un rato deja de leer correctamente el dispositivo LEAP Motion, suele deberse a que este se ha recalentado del uso, para evitar que esto suceda se recomienda desconectar el controlador durante un breve periodo entre pruebas, de esta manera se enfriará y volverá a funcionar como debería.

4.3.2. Desarrollo y control de la aplicación

Código de inicialización

En esta sección se llevará a cabo el desglose del código principal de la aplicación.

En primer lugar, se comienza mostrando las librerías empleadas para el desarrollo del programa, las librerías usadas se muestran en la Figura 4.18.

```
//Libraries
import autopilot.ActionXplane;
import java.io.IOException;
import com.leapmotion.leap.*;
import xplane.XplaneInputOutput;
```

Figura 4.18. Librerías empleadas en la aplicación «*Conexión.java*»

- ***xplane.XplaneInputOutput:*** consiste en la librería de la aplicación que permite la conexión entre el Netbeans y el programa de X-Plane.
- ***autopilot.ActionXplane:*** esta librería se encarga de que se pueda actuar sibre los controles de la aeronave seleccionada en X-Plane 10.
- ***java.io.IOException:*** se trata de una librería que detecta si la clase experimenta alguna excepción producida por un fallo u otra interrupción. Se emplea para la salida de la aplicación.
- ***com.leapmotion.leap.*:*** se trata de la librería de LEAP Motion, que permite la lectura de datos sobre las manos detectadas.

El siguiente paso importante es crear las variables del sistema que sirven para conectar con el programa de X-Plane y poder modificar los controles que deseemos, mediante la función de *xplane.write(action)*, donde se envía a X-Plane todas las acciones guardadas en la variable *action*. En la Figura 4.19, se envía a X-Plane información sobre la posición inicial en la que se deben encontrar los controles básicos de la aeronave. Por tanto una vez se llame al constructor desde el *main*, se crea el panel (cuyo diseño se explica en la Sección 4.3.3) y la conexión con el X-Plane.

```
XplaneInputOutput xplane;
ActionXplane action;

private DataXplane data;
private CabinaLEAPXPLANE mypanel;

public SampleListener(CabinaLEAPXPLANE mypanel) {
    this.mypanel = mypanel;
    xplane = new XplaneInputOutput();
    action = new ActionXplane();
    action.setAilerons((float) 0);
    action.setElevators((float) 0);
    action.setRudder((float) 0);
    action.setThrottle((float) 0);
    action.setBrake(1);
    action.setFlaps((float) 0);
    xplane.write(action);
}
```

Figura 4.19. Valor de los controles iniciales introducidos por la aplicación «*Conexión.java*»

A continuación, el resto de pasos coinciden con los procesos de inicialización de las clases internas explicados en la Sección 4.1.1, para la comprobación de en que estado se encuentra el controlador.

Una vez se entra dentro de la clase *OnFrame*, es donde se introducen los comandos necesarios para la lectura de datos de las manos mediante la cámara de LEAP Motion, y se envían los datos a X-Plane para el control de la aeronave. En primer lugar, se procede con la lectura de las variables y el envío de estas a la aplicación «*CabinaLEAPXPLANE.java*» para que sean mostradas, tal como muestra la Figura 4.20. Se observan tres *displays* distintos, el correspondiente a *data* permite leer los datos de «*DataXplane.java*», *action* los de «*ActionXplane.java*» y el de *data*, permite acceder a la información de cada marco del controlador LEAP Motion.

```
// To send xplane data to the developed cabin
data = xplane.read();
mypanel.display(data);

xplane.write(action);
mypanel.display(action);

// Get the most recent frame and report some basic information
Frame frame = controller.frame();

mypanel.displayLEAP(frame);
```

Figura 4.20. Lectura y envío de datos a la aplicación «*CabinaLEAPXPLANE.java*»

```
//Get hands
for (Hand hand : frame.hands()) {

    mypanel.displayLEAPHand(hand);
    // Get the hand's normal vector and direction
    Vector normal = hand.palmNormal();
    Vector direction = hand.direction();
    Vector position = hand.palmPosition();
    Vector velocity = hand.palmVelocity();
    // Get the number of extended fingers
    FingerList extendedFingerList = hand.fingers().extended();
    FingerList IndexFingerList = hand.fingers().fingerType(Finger.Type.TYPE_INDEX);
    FingerList MiddleFingerList = hand.fingers().fingerType(Finger.Type.TYPE_MIDDLE);
    FingerList PinkyFingerList = hand.fingers().fingerType(Finger.Type.TYPE_PINKY);
    FingerList RingFingerList = hand.fingers().fingerType(Finger.Type.TYPE_RING);
    FingerList ThumbFingerList = hand.fingers().fingerType(Finger.Type.TYPE_THUMB);
    // Get a single finger of each type from the previous lists
    Finger IndexFinger = IndexFingerList.get(0);
    Finger MiddleFinger = MiddleFingerList.get(0);
    Finger PinkyFinger = PinkyFingerList.get(0);
    Finger RingFinger = RingFingerList.get(0);
    Finger ThumbFinger = ThumbFingerList.get(0);
    // Create a string with the information about wich hand is being detected
    String handType = hand.isLeft() ? "Left hand" : "Right hand";
```

Figura 4.21. Vectores, listas y cadenas creadas en la aplicación «*Conexión.java*»

En segundo lugar, dentro de esta clase se definen una serie de vectores o listas que

luego serán necesarios para la lectura de parámetros, junto con el envío de información de la variable *hands* a la aplicación «*CabinaLEAPXPLANE.java*», como se muestra en la Figura 4.21.

Finalmente, para abrir la cabina desarrollada en la aplicación «*CabinaLEAPXPLANE.java*», desde el bucle *main* se conecta con la cabina y se muestra visible en la pantalla del monitor, el código empleado para ello se muestra en la Figura 4.22.

```
public static void main(String[] args) {
    // TODO code application logic here
    CabinaLEAPXPLANE micabina = new CabinaLEAPXPLANE();
    SampleListener listener = new SampleListener(micabina);
    micabina.setVisible(true);
    // Create a sample listener and controller
    Controller controller = new Controller();

    // Have the sample listener receive events from the controller
    controller.addListener(listener);

    // Keep this process running until Enter is pressed
    System.out.println("Press Enter to quit...");
    try {
        System.in.read();
    } catch (IOException e) {
    }

    // Remove the sample listener when done
    controller.removeListener(listener);
}
```

Figura 4.22. Bucle main de la aplicación «*Conexión.java*»

Esquema del control planteado

Para continuar con la fase correspondiente al código del control del X-Plane, se considera necesaria la presentación de la idea del control.

Durante la realización de varias pruebas en el desarrollo de la aplicación, ha sido notado que el pilotaje de la aeronave de un destino a otro conlleva un periodo de tiempo elevado, provocando cansancio en los hombros del usuario. Con el objetivo de evitar dicho agotamiento físico y que la aplicación pueda ser usada de forma agradable por cualquier usuario, se requiere que esta ofrezca la posibilidad de tomar un descanso durante el vuelo extrayendo las manos del rango de visión del dispositivo LEAP Motion.

El hecho de extraer las manos del campo de visión del controlador puede resultar en lecturas de datos no deseadas al ser el campo de visión del dispositivo LEAP Motion bastante amplio, y al tratar de quitar la mano esta puede no mantener siempre la posición deseada. Para evitar estas lecturas erróneas se delimitará una zona de control y otra en la que se dejará de leer datos.

Por otro lado, una propuesta para el control de la aeronave eran gestos con la palma de la mano enfocando el techo, pero de esta forma los datos sobre los dedos de la mano leídos no resultan ser lo suficientemente fiable para la realización del control.

Finalmente, la solución propuesta para la realización del control, incluye una limitación de espacios como la mostrada en la Figura 4.23. Donde se muestran tres zonas diferenciadas, localizándose el controlador LEAP Motion en el centro. Las zonas son las siguientes:

- **Zona A:** esta región tiene unas dimensiones de 40 cm de ancho por 30 cm de alto, y el LEAP Motion siempre supondrá el centro de la misma. Es en esta zona del espacio en la que se localizarán las mano para la alteración de los controles básicos de la aeronave, siendo estos los flaps, alerones, estabilizador horizontal, timón de cola, palanca de gas y frenos.
- **Zona B:** estas regiones servirán para pasar del modo pilotaje al modo autopiloto, y como indica la Figura 4.23 tienen unas dimensiones de 40 cm de alto por 15 cm de largo.
- **Zona C:** esta última región se empleará para poder extraer las manos del rango de visión del dispositivo LEAP Motion y evitar lecturas erróneas. Por lo que si las manos se encuentran en está región no se enviará información de estas el simulador X-Plane.

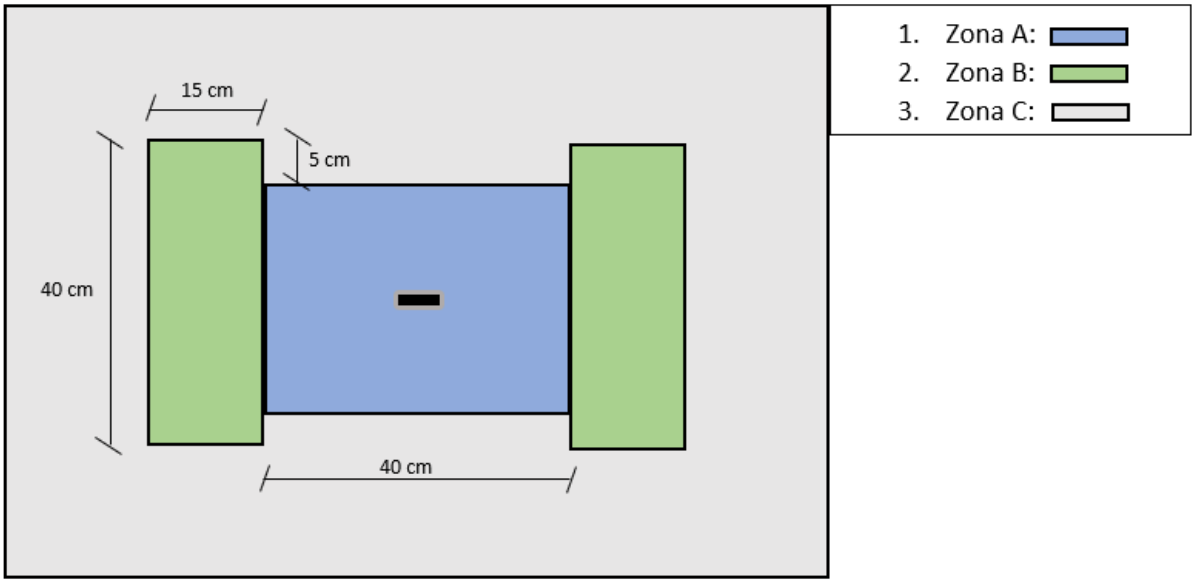

Figura 4.23. Esquema de las zonas de control

La limitación presentada a sido seleccionada después de diversas pruebas y teniendo en cuenta que el ancho de la zona A se corresponde con el ancho de hombros de una persona media y la posición de las manos en ambas zonas resultaría cómodo para la mayoría de los usuarios, al ser posiciones de los brazos que no dañarían los hombros.

Una vez la idea a desarrollar ha sido introducida, se continua con el desarrollo del código de la aplicación. Separándolo, dependiendo de en que zona se encuentran.

Zona A: Control

Como ha sido mencionado, es en esta región donde se realizará la actuación sobre los controles básicos de la aeronave. Estos se dividirán entre ambas manos.

Por una parte, la mano izquierda será la encargada de la modificación del empenaje horizontal, los alerones y el timón de cola, siendo los encargados de las variaciones en el cabeceo, alabeo y guiñada, respectivamente.

Respecto al código empleado, cuando el sensor detecte que la mano izquierda se encuentra en el espacio limitado para la zona de control, se crean unas variables como se muestra en la Figura 4.24, las cuales serán necesarias para enviar la información del movimiento de la mano a X-Plane.

```
// Declaration of variables
float pitch;
pitch = (float) (Math.toDegrees(direction.pitch())) / 35;
float yaw;
yaw = (float) (Math.toDegrees(direction.yaw())) / 50;
float roll;
roll = (float) -(Math.toDegrees(normal.roll())) / 70;
```

Figura 4.24. Declaración de las variables de alabeo, cabeceo y guiñada

Como se observa en la Figura 4.24, han sido creadas tres variables que representan el cabeceo, guiñada y alabeo. Estos movimientos, se corresponden con los movimientos de la mano mostrados en la Figura 4.25 [27]. Con la finalidad de que realizar el control de forma cómoda, una conversión ha sido realizada para que el cabeceo tenga un rango hasta ±35°, la guiñada hasta ±50° y el alabeo hasta ±70°. De esta forma se consigue llegar hasta la posición máxima y mínima de los actuadores sin la necesidad de forzar la muñeca.

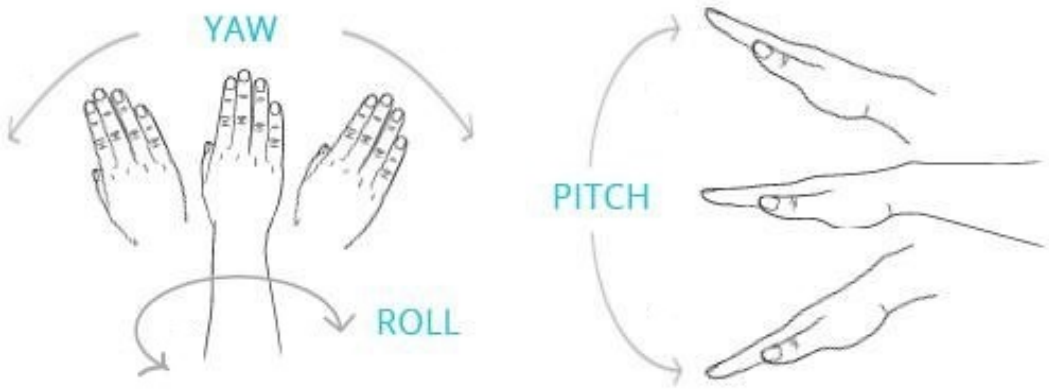

Figura 4.25. Representación de los movimientos de alabeo, cabeceo y guiñada

La forma en la que se realiza la conversión de grados a los valores que adaptan los controles (valores entre -1 y 1), se realiza tal y como se observa en la Figura 4.26, donde está representado el código para el caso del control del timón de cola mediante el movimiento de guiñada con la mano. Cuando este se encuentra entre ±50°, el timón de cola adoptará su valor correspondiente mediante la conversión entre ±1. En el caso en el que se excedan los límites implantados con la mano, como el timón de cola no puede exceder esos valores, se bloqueará en 1 o -1. Para los casos del estabilizador horizontal y los alerones, se procede de la misma manera.

```
//Control rudder
if (yaw > -1 && yaw < 1) {
    action.setRudder(yaw);
    xplane.write(action);
} else if (yaw < -1) {
    action.setRudder(-1);
    xplane.write(action);
} else {
    action.setRudder(1);
    xplane.write(action);
}
```

Figura 4.26. Código para el envío de información al simulador de vuelo, ejemplo: timón de cola

Además la mano izquierda también será la encargada de desactivar el autopiloto cuando se desee. Para ello, bastará con cerrar el puño, pudiéndose posteriormente abrir para actuar como ha sido mencionado sobre el resto de controles. El código que ejecuta está acción se muestra en la Figura 4.27, donde se desactiva el modo mantener rumbo y el de mantener altitud a parte del autopiloto en sí.

```
// For the deactivation of the autopilot
if (extendedFingerList.count() == 0) {
    action.setFdirmode(0);
    action.setModealt(3);
    action.setModeheading(0);
    xplane.write(action);
}
```

Figura 4.27. Código para para la cancelación del autopiloto

Por otra parte, al introducir la mano derecha dentro de esta zona, el usuario será capaz de actuar sobre los flaps durante el despegue y aterrizaje, además de poder manipular la palanca de gas y poner y quitar los frenos de la aeronave.

Al igual que ha sido realizado antes, se deben definir primero las variables que se emplearán para enviar la información a X-Plane. La palanca de gases varía con la posición vertical de la mano, los frenos con la posición de «agarre» de la mano y los flaps, funcionan como el estabilizador vertical explicado previamente.

```
// Declaration of variables
float thrust;
thrust = (float) 0.5 * (position.getY() - 100) / 100;
float brakes;
brakes = (float) hand.grabStrength();
float flaps;
flaps = (float) (Math.toDegrees(direction.pitch())) / 35;
```

Figura 4.28. Declaración de las variables flaps, palanca de gases y frenos

De forma individual, el código de estas variables es muy similar al de la Figura 4.26, salvo que para el caso de los flaps y la palanca de gases, estos solo pueden adoptar valores entre 0 y 1. Asimismo, solo se podrá actuar sobre los flaps (que únicamente serán empleados en las fases de despegue y aterrizaje) cuando haya menos de 5 dedos extendidos. En caso de que se encuentren los cinco dedos extendidos, el valor que adoptarán los flaps será 0.

Zona B: Activación del autopiloto

Esta zona está enfocada manejo del autopiloto, pudiendo armar los modos necesarios y activar el autopiloto, además de poder seleccionar a que altitud se desea establecer y el rumbo deseado. Al igual que en el caso anterior, se realiza una división de las funciones realizadas en base a la mano introducida.

La mano derecha, se encarga de la selección del rumbo, para ello dado que el rango de valores que posibles para selección del rumbo es de 360°, y se desea una gran precisión para poder seleccionar de forma fiable la dirección que vaya a seguir la aeronave. Dicha selección se realizará teniendo en cuenta la posición de la mano (la inicialización de dicha variable se muestra en la Figura 4.29), y para obtener una mayor precisión, como ha sido mencionado, se determinaran una serie de rangos, en base a los dedos que se encuentren extendidos.

```
float heading1 = (float) (position.getY() - 100) / 8;
float heading = 0;
```

Figura 4.29. Declaración del rumbo

La selección del rumbo será realizada en base a la Tabla 4.3, y el código empleado para ello se muestra en la Figura 4.30, con el ejemplo de la detección del dedo índice extendido.

Selección del rumbo	
Dedos extendidos	Rango
0 dedos	$0° - 50°$
Índice	$50° - 100°$
Corazón	$100° - 150°$
Anular	$150° - 200°$
Meñique	$200° - 250°$
Pulgar	$250° - 300°$
5 dedos	$300° - 360°$

Tabla 4.3. Selección del rumbo mediante la posición de la mano

```
} else if (IndexFinger.isExtended() && extendedFingerList.count() < 5) {

    if (heading1 > 0 && heading1 < 50) {
        action.setHeadingValue(heading1 + 50);
        xplane.write(action);
        heading = heading1 + 50;

    } else if (heading1 < 0) {
        heading1 = 50;
        heading = 50;
        action.setHeadingValue(heading1);
        xplane.write(action);

    } else {
        heading1 = 100;
        heading = 100;
        action.setHeadingValue(heading1);
        xplane.write(action);

    }
```

Figura 4.30. Código de la división de rangos del rumbo

En esta zona B, la mano izquierda controlará el armado y activación del autopiloto. La mano tendré tres gestos reconocidos para la realización de la activación del autopiloto y sus funciones, cuyo código se encuentra en la Figura 4.31. En primer lugar, si el puño se encuentra cerrado, se activará el modo seguimiento del rumbo. En el caso en el que se abra la palma de la mano, mirando está hacia el suelo, se activará el autopiloto y la aeronave comenzará a seguir el rumbo deseado. Finalmente, si cambiamos la dirección de la mano, y la posicionamos con su palma orientada al techo, se activará el modo de mantener altitud.

```
// To arm the autopilot
if (hand.grabStrength() == 1) {
    action.setModealt(3);
    action.setModeheading(1);
    action.setFdirmode(1);
    xplane.write(action);

} else if (hand.grabStrength() == 0) {

    if (abs(Math.toDegrees(normal.roll())) < 120) {

        // To activate the autopilot
        action.setFdirmode(2);
        xplane.write(action);

    } else if (abs(Math.toDegrees(normal.roll())) > 120) {

        // To select the ald altitude mode
        action.setModealt(6);
        xplane.write(action);
    }
}
```

Figura 4.31. Código para la activación del autopiloto

4.3.3. Diseño del panel de control

Después del desarrollo del código de la aplicación final, se procede con el análisis de la aplicación «*CabinaLEAPXPLANE.java*» introducida previamente. Esta aplicación trata de simular un pequeño panel de control con los datos básicos de vuelo de una aeronave, además de incluir información sobre las manos detectadas por el controlador LEAP Motion.

El desarrollo de este panel incluye principalmente controles del tipo *Text Field* o **Radio Button** para la lectura de datos o la comprobación de si algún control se encuentra activado o desactivado.

Las variables que vayan a ser mostradas, se agrupan en los *displays* introducidos por las Figuras 4.20 y 4.21. Un ejemplo de como son leídas las variables en la cabina se muestra en la Figura 4.32, en este caso se encuentran las correspondientes a las lecturas de «*DataXplane.java*», pero para la escritura del resto de variables, se actúa de manera simular.

En la Figura 4.33, se muestra el diseño que tiene el panel de control desarrollado, en la cuál se diferencian cuatro secciones distintas:

- **X-Plane Data:** introduce los datos básicos que pueden resultar interesantes durante el vuelo de la aeronave.
- **Autopilot:** introduce información relativa al proceso de selección del autopiloto.
- **X-Plane Control:** se definen los controles básicos sobre los que se actúa.
- **LEAP Motion:** añade información relacionada con las lecturas realizadas por el dispositivo LEAP Motion.

```
public void display(DataXplane dx) {

    latText.setText(String.format("%.2f", dx.getLat()));
    lonText.setText(String.format("%.2f", dx.getLon()));
    altText.setText(String.format("%.2f", dx.getAlt()));
    pitchText.setText(String.format("%.1f", dx.getPitch()));
    rollText.setText(String.format("%.1f", dx.getRoll()));
    yawText.setText(String.format("%.1f", dx.getHead()));
    alphaText.setText(String.format("%.2f", dx.getAoa()));
    betaText.setText(String.format("%.2f", dx.getBeta()));
    vpathText.setText(String.format("%.1f", dx.getVpath()));
    hpathText.setText(String.format("%.1f", dx.getHpath()));
    iasText.setText(String.format("%.1f", dx.getIas()));
    tasText.setText(String.format("%.1f", dx.getTas()));
    vsText.setText(String.format("%.1f", dx.getVs()));
    gsText.setText(String.format("%.1f", dx.getGs()));
    brakesButton.setSelected(dx.getBrake() != 0);
    headingText.setText(String.format("%.1f", dx.getHeadingValue()));
    headingButton.setSelected(dx.getModeheading() != 0);
    altitudeButton.setSelected(dx.getModealt() != 3);
    AutoPilotoText.setText(String.format("%.0f", dx.getFdirmode()));

}
```

Figura 4.32. Código para la escritura de las variables data

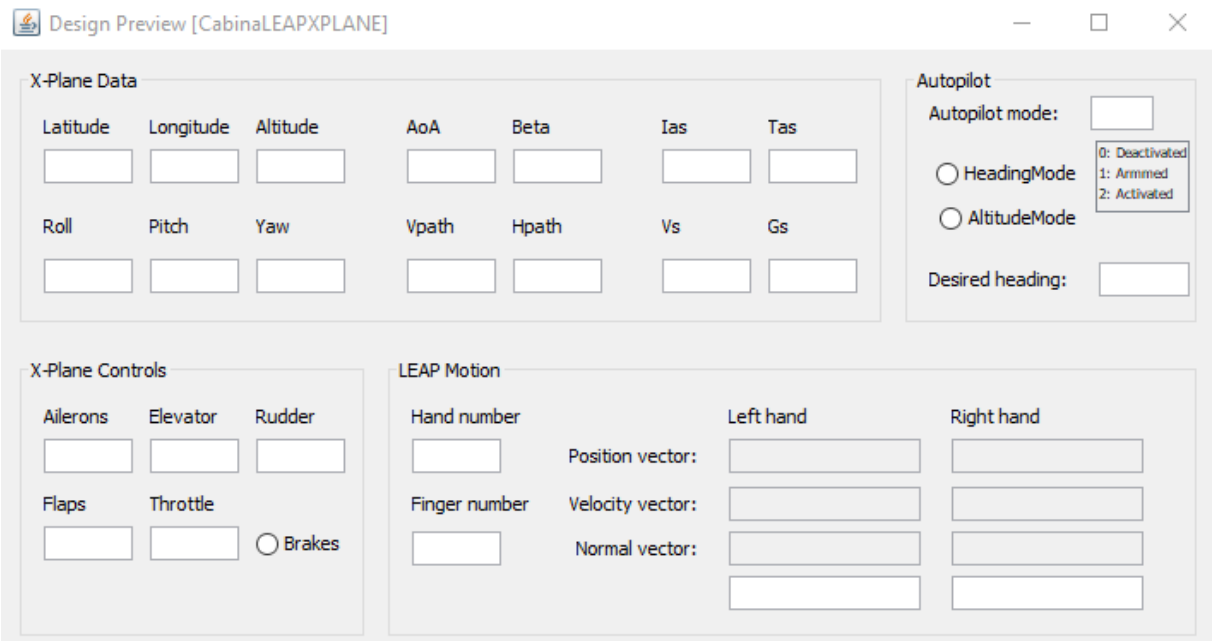

Figura 4.33. Diseño del panel de control desarrollado

4.3.4. Resultado final

Introducción de la cabina

Una vez ha sido desarrollada la aplicación, se puede proceder con el análisis de los resultados obtenidos, pero para ello se debe introducir ligeramente la cabina de la aeronave y sus instrumentos principales. En la Figura 4.34, se muestra un cuadro destacando los instrumentos principales que deben ser conocidos para el uso de la aplicación y comprensión de los resultados.

Figura 4.34. Cabina del Cessna C172

Los instrumentos que se deben conocer, se introducen siguiendo el orden numérico de la Figura 4.34, y son los siguientes:

- **Indicador de la velocidad:** muestra la velocidad de la aeronave relativa al aire.
- **Indicador de la altitud:** muestra la altitud de la aeronave relativa al horizonte. Se utiliza para informar al piloto si la aeronave se encuentra volando recta, girando, ascendiendo o descendiendo.
- **Altímetro:** muestra la altitud de la aeronave sobre el nivel del mar.
- **Indicador de viraje:** indica al piloto el ratio de giro, y si la aeronave se desplaza lateralmente durante el mismo.
- **Indicador giroscópico:** presenta el rumbo magnético del avión. Además, en el se podrá seleccionar el rumbo que la aeronave deba seguir cuando se active el autopiloto.
- **Indicador de la velocidad vertical:** informa al piloto de la velocidad de ascenso, o la velocidad de descenso.

Resultados

Una vez la aplicación final ha sido completamente desarrollada, una grabación de las pruebas finales ha sido realizada, incluyendo grabación de la pantalla y de la cámara para mostrar los gestos realizados, para mostrar la funcionalidad del programa. En esta sección, se analiza un fotograma del vídeo par analizar la funcionalidad de la aplicación. En caso de desear conocer más acerca del uso de la aplicación, diríjase al Apéndice A

En la Figura 4.35 se muestra un fotograma del vídeo realizado para la demostración del funcionamiento de la aeronave, donde los datos del mismo se muestran en la Figura 4.36.

Figura 4.35. Fotograma extraído del vídeo demostrativo realizado durante las pruebas finales

Finalmente, en lo referente a los resultados obtenidos, puede concluirse que han sido satisfactorios, dado que la idea de control inicial ha podido ser implementada en la aplicación de forma operativa, junto con una ventada de datos que proporciona al usuario información sobre las características básicas de actuación de la aeronave y sobre las manos detectadas por el dispositivo LEAP Motion.

En el proceso de desarrollo se ha logrado cumplir paso a paso con los objetivos establecidos en la Sección 1.2, los cuales han permitido la adquisición de un mayor conocimiento sobre las distintas tecnologías empleadas durante el proyecto.

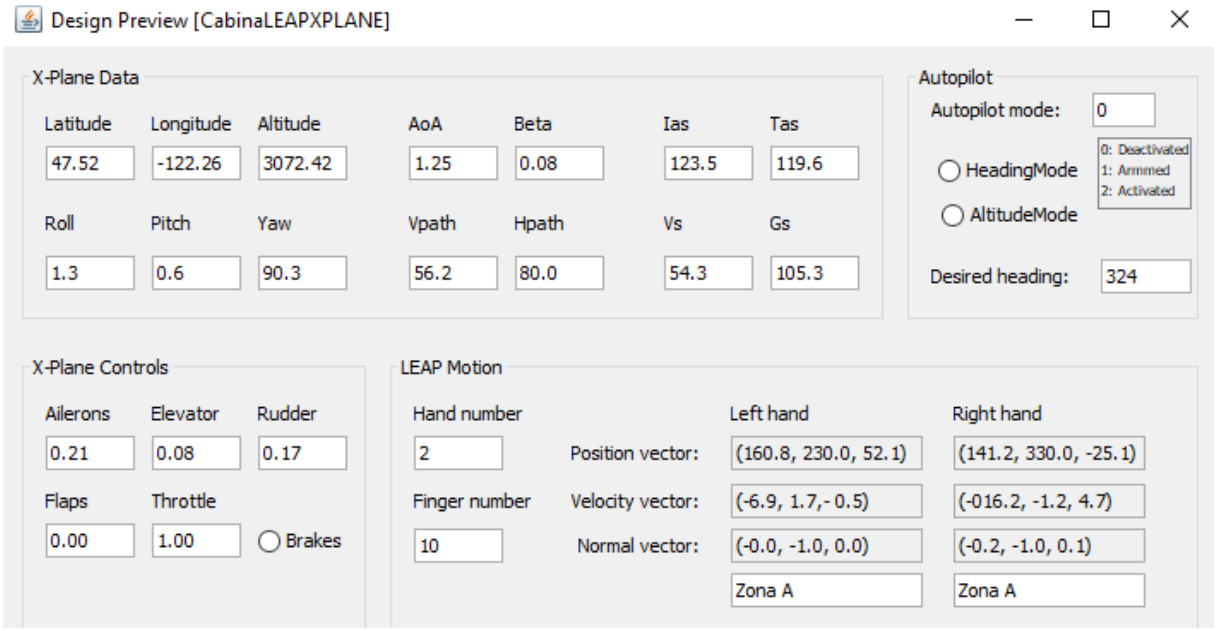

Figura 4.36. Datos obtenidos en el panel de control del fotograma de la Figura 4.35

5 | Conclusión

Con la realización de este proyecto, se ha podido llevar a cabo el aprendizaje de el lenguaje de programación de *JAVA* y el de una tecnología que se encuentra en auge en los últimos tiempos como la tecnología de reconocimiento gestual. Además, la implementación del simulador de vuelo en la aplicación desarrollada ha permitido, por una parte, la revisión de conceptos de aviación y algún problema que pueden llegar a encontrar durante las operaciones de vuelo y, por otra, la recapitulación del proceso de despegue llevado a cabo por una aeronave, como se estudió en asignaturas a lo largo de la carrera.

En cuanto al análisis del proceso de desarrollo realizado, dado que se ha en diferentes pasos, conclusiones de cada uno de ellos pueden ser extraídas. En primer lugar, se ha podido comprobar en la Sección 4.1 el funcionamiento y las posibilidades que nos ofrece la tecnología de reconocimiento de gestos, pero pese a ello, se observa como esta tecnología todavía una presenta algunos errores de funcionamiento por los SDKs proporcionados. En segundo lugar, en la Sección 4.2 se llega a comprender la complejidad y versatilidad que ofrece el simulador de vuelo de X-Plane para el manejo de diferentes aeronaves.

Finalmente, en la aplicación final desarrollada en la Sección 4.3, la creación de una aplicación funcional e intuitiva ha sido lograda. En ella se le permite al usuario no solo la posibilidad de aprender sobre el pilotaje de una aeronave, sino que además ofrece información del vuelo de forma simultánea en la cabina desarrollada, pudiendo ser útil para la realización de un estudio del vuelo.

Con todo ello, se ha conseguido cumplir con el objetivo del proyecto, desarrollando una aplicación final que sin necesidad de obtener datos mediante el método tradicional del teclado y ratón, se encuentra controlada en su totalidad por el uso de gestos, trazando el camino para la facilitación de futuras aplicaciones con la inclusión de la tecnología de reconocimiento de gestos.

6 | Trabajo futuro

Como ha sido explicado a lo largo de todo el documento, se ha conseguido llevar a cabo el desarrollo de una aplicación que permite el control de un simulador de vuelo mediante la utilización de una tecnología de reconocimiento de gestos. Este puede tratarse del comienzo del camino que pueda llegar a permitir la utilización de la tecnología de reconocimiento gestual para la obtención más rápida, eficaz y económica de los datos del vuelo. Por tanto, pueden ser posibles los desarrollos de continuaciones del presente proyecto, entre los cuáles se podrían detectar varios caminos: indagando más en el empleo de gestos más dinámicos y en la optimización del uso del LEAP Motion o aumentando los controles de la aeronave que ofrece X-Plane.

En primer lugar, ha sido comprobado que las posibilidades que ofrece el dispositivo LEAP Motion empleado durante este proyecto tiene infinitas posibilidades, pero también consta con varios fallos. Por lo que cuando se solucionen los errores presentes en la presente SDK proporcionada por el desarrollador, podría tratarse de implementar un control más visual y parecido al que podría encontrarse a los mandos de la cabina de vuelo.

En segundo lugar, han sido observadas la gran cantidad de posibilidades que ofrece el simulador de vuelo de X-Plane. El presente proyecto se ha centrado en la cabina de vuelo de un CESNA 172 SP, realizándose un control básico de las características de vuelo de dicha avioneta. En este aspecto puede profundizarse mucho más para ofrecer al usuario una mayor cantidad de posibilidades. Estas posibilidades podrían ir desde aumentar las funcionalidades controladas en la presente aeronave (como podría ser el activado y cambio de las frecuencias de radio u otros controles secundarios, en cuanto al hecho de volar la aeronave, pero que son imprescindibles durante el pilotaje de cualquier aeronave), hasta trabajar con aeronaves más grandes y complejas que requieren del uso de una mayor cantidad de controles, como sería el caso de un B-747.

7 | Presupuesto

En esta sección se dará a conocer el coste estimado para la realización del proyecto. Para la presentación de los costes se realizará un desglose de los mismos en 3 grupos principales: los costes asociados a la mano de obra, los costes de material y los costes de adquisición de las licencias de software.

7.1. Mano de obra

Para el cálculo de los costes asociados a la mano de obra, se incluirán tanto las oras dedicadas por el alumno como las del profesor tutor del trabajo.

Para esta estimación de las horas totales del proyecto, se ha considerado que este, debe realizarse en en un total de 360 horas al corresponderse con 12 créditos ECTS (30 horas de trabajo por crédito obtenido). Respecto a los costes de supervisión del tutor, se han realizado 12 tutorías de unas 2 horas aproximadamente. Estos precios quedan resumidos en la Tabla 7.1.

	Cantidad [h]	Coste unitario [€/h]	Coste total [€]
Alumno	360	12,00	4320,00
Tutor	24	30,00	720,00
		Subtotal	5040,00

Tabla 7.1. Costes asociados a la mano de obra

7.2. Material

Los costes asociados a la adquisición del material necesaria para la realización del proyecto pueden encontrarse en la Tabla 7.2.

Se recuerda, que en la Sección 3.1 se menciona que no es necesario que los ordenadores seleccionados para la realización del proyecto sea los mismos, pero si deberán cumplir con los requisitos mínimos del controlador LEAP Motion descritos en la Sección 3.2.1 y con los requisitos mínimos del programa X-Plane 10 mostrados en la Sección 3.3.

Material	Cantidad [h]	Coste unitario [€/h]	Coste total [€]
HP 17-ca2000ns	1	549,00	549,00
ASUS E751JF-T2052H	1	836,95	836,95
Dispositivo LEAP Motion	1	91,21	91,21
		Subtotal	1477,16

Tabla 7.2. Costes de los materiales empleados

7.3. Licencias de Software

En lo referente al coste de adquisición de las licencias necesarias para la realización del proyecto, el coste de las mismas puede observarse en la Tabla 7.3. Como se ha trabajado en este proyecto con X-Plane 10 se ha incluido el precio de esta aplicación. Puede observarse que a excepción de las licencias de *X-Plane 10* y de *Microsoft Office 365*, el resto de licencias son gratuitas.

Licencias	Cantidad [h]	Coste unitario [€/h]	Coste total [€]
Microsoft Office 365	1	69,00	69,00
Java SDK Version 8	1	0,00	0,00
LEAP Motion SDK 3.2.0	1	0,00	0,00
X-Plane 10	1	40,96	40,96
NetBeans IDE 8.2	1	0,00	0,00
Notepad ++	1	0,00	0,00
		Subtotal	109,96

Tabla 7.3. Costes de las licencias necesarias

7.4. Coste total

	Coste total [€]
Mano de obra	5040,00
Materiales	1477,16
Licencias	109,96
Total	6627,12

Tabla 7.4. Costes total del proyecto

Finalmente, en la Tabla 7.4, se observa que el precio total conlleva la realización del proyecto es de 6627,12 €.

A | Apéndice: Manual del usuario

En las siguientes páginas se encontrará una guía con los pasos que deben seguirse para poder ejecutar la aplicación «*Conexión.java*» y poder controlar la CESNA 172 SP correctamente, pudiendo realizar el despegue, ascenso, activado del autopiloto y vuelta a la toma de control de la aeronave para poder comenzar el descenso.

Pasos iniciales

Lo primero es asegurarse de cumplir con la configuración necesaria para la correcta ejecución. Para ello, se presenta una descripción rápida y simple que debe llevarse a cabo para su puesta en marcha, la cual llevará apenas 5 minutos.

- Asegurarse de poseer el equipo informático necesario para la ejecución de la aplicación. Esto incluye dos ordenadores. El primero de ellos se empleará para la lectura de datos del dispositivo LEAP Motion, y ejecutará la aplicación creada en NetBeans (recordar que debe tener un procesador Intel Core i3 o superior, además de 2 GB de RAM como mínimo). El segundo de ellos, será empleado para abrir el programa de X-Plane (debiendo tener un procesador Intel Core i3 o superior, además de 4 GB de RAM como mínimo).
- Conectar el dispositivo LEAP Motion al primer ordenador y comprobar desde el programa *LEAP Motion Control Panel*, que este se encuentra operando correctamente según las instrucciones de la Sección 3.2.4.
- Comprobar que se tiene ambos ordenadores conectados a la misma red, para poder enviar información a través de la dirección IP, desde la aplicación «*XplaneInputOutput.java*».

Una vez se cumpla con las condiciones anteriores, asegúrese de tener en una pantalla la aplicación desarrollada de NetBeans abierta, mientras que en la otra será necesario tener el programa de X-Plane 10 abierto. El siguiente paso sería el de conectar el LEAP Motion y localizarlo en frente del segundo ordenador para estar mirando a la aeronave pilotada mientras se realizan los gestos, la luz verde del controlador deberá estar encarada en dirección al usuario. Un ejemplo de la configuración se muestra en la Figura A.1.

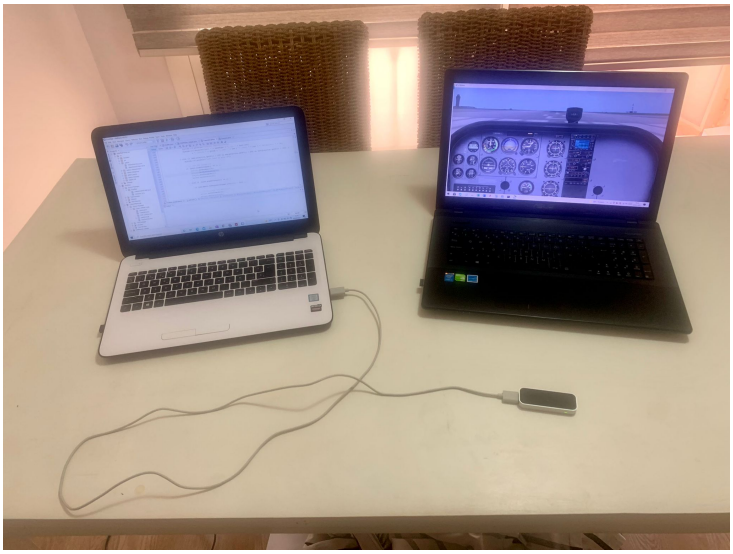

Figura A.1. Configuración de la zona de pruebas

Inicialización de la aplicación

Para asegurarse del correcto funcionamiento del programa, se recomienda situarse en un entorno con baja iluminación y buena conexión a internet. De esta forma, las lecturas del dispositivo LEAP Motion serán más precisas y el envío de datos de un ordenador a otro no llevará retrasos.

A modo de comprobación de que todo funciona correctamente se recomienda que al ejecutar la aplicación «*Conexión.java*», se trate de introducir la mano derecha en la Zona A abriendo y cerrando la mano, de esta forma se debería ver como se bloquean y desbloquean los frenos. Si esto funciona, el programa ya está listo para ser empleado de forma apropiada.

A continuación se desarrollara el proceso que debe seguirse para la realización del despegue, ascenso y vuelo en crucero, junto con consejos para e usuario.

Proceso de despegue

Para iniciar la carrera de despegue, el usuario debe introducir ambas manos en la definida zona A, estas deben estar lo más planas posibles, para que los valores iniciales de los controles no se modifiquen mucho en su introducción. Para la mano derecha, se recomienda que se introduzca a menos de 15 cm de altura respecto al LEAP Motion y con el puño cerrado, de esta forma los frenos estarán activados y la palanca de gases tendrá un valor nulo.

Cuando se desee comenzar con la carrera de despegue, el usuario deberá abrir la palma de la mano derecha y ir subiendo dicha mano poco a poco. Mientras, la mano

izquierda deberá ir controlado el timón de cola para que la CESSNA 172 SP, realizando un movimiento de guiñada con la muñeca. En la Figura A.2 se muestra un fotograma del vídeo realizado, en el que la aeronave se encuentra en la carrera de despegue.

Figura A.2. Fotograma extraído del vídeo demostrativo realizado durante las pruebas finales durante la fase de carrera de despegue

Una vez el indicador de velocidad alcance unos 90 nudos, se recomienda cabecear con la mano derecha para activar los flaps (cabe recordar que para poder actuar sobre los flaps de la aeronave, el usuario debe tener la palma de la mano abierta pero con el pulgar cerrado, como indica la Figura A.3 [28]. En ella, la mano de la izquierda tiene el pulgar extendido y no modificará los flaps, mientras que la de la derecha con el pulgar no extendido permitirá el control de los flaps.)

Figura A.3. Posicionamiento de las manos

De esta forma la aeronave comenzará a ganar altitud. También se puede actuar sobre el empenaje horizontal para ayudar en el ascenso de la aeronave. De esta forma, se entra en la fase de ascenso.

Proceso de ascenso

Durante la fase de ascenso, se puede extender el pulgar de la mano derecha para que los flaps no aumenten considerablemente el drag. Esto supone que el usuario deberá mantener la mano derecha a una altura de 30 cm o superior, para que la palanca de gases se sitúe al máximo.

En cuanto a la mano izquierda, el usuario deberá de prestar más atención. Ella controlará tanto el cabeceo, como la guiñada y el alabeo de la aeronave. Teniendo en cuenta el indicador de la altitud y el de viraje, se deberá corregir con sutileza (sin realizar cambios bruscos con la mano) las aptitudes de la aeronave. En la Figura A.4, se muestra un fotograma de la fase de ascenso extraído de las pruebas finales.

Figura A.4. Fotograma extraído del vídeo demostrativo realizado durante las pruebas finales durante la fase de ascenso

La CESSNA 172 SP deberá subir hasta una altitud de unos 2000 metros para poder actuar sobre el autopiloto. Una vez se alcance dicha altitud, el usuario puede mover lateralmente la mano izquierda hacia la zona B (para ello la aeronave debe estar volando de forma equilibrada) con el puño cerrado, de esta forma se armará el autopiloto y se activará el modo seguimiento del rumbo. Posteriormente, se abre la palma de la mano para activar el autopiloto y que la aeronave comience a seguir el rumbo establecido.

Una vez se active el autopiloto y la aeronave pueda controlarse sola, llevar la mano derecha a la zona B para seleccionar el rumbo deseado según la Tabla 4.3. Una vez seleccionado, la mano derecha puede ser extraída del campo de lectura del dispositivo entrando en la zona C para luego ser eliminada del campo de visión sin causar lecturas indeseadas.

Proceso de vuelo en crucero

Una vez se encuentra el autopiloto activado y la aeronave sigue el rumbo establecido, esta irá ascendiendo en altitud poco a poco. Una vez la altitud de crucero sea alcanzada, se debe seleccionar el modo mantener altitud, esto es realizado girando la mano izquierda, enfocando ahora hacia el techo, con la palma abierta. Así la aeronave oscilara ligeramente un par de segundos hasta mantener altitud constante. Finalmente, ya puede ser extraída la mano izquierda del rango del LEAP Motion. En la Figura A.5, se muestra un fotograma del vídeo realizado, en el que el autopiloto ya ha sido activado y ambas manos se encuentran fuera del rango de visión del dispositivo LEAP Motion.

Figura A.5. Fotograma extraído del vídeo demostrativo realizado durante las pruebas finales con el autopiloto activado

En caso de desear modificar el rumbo puede introducirse la mano derecha en la zona B y seleccionar el nuevo rumbo deseado.

Proceso de retoma del control

Una vez se alcance el destino deseado y se pretenda retomar el control de la aeronave, basta con introducir las manos en la zona A y cerrar el puño de la mano izquierda. Así se podrá controlar sin problema la CESSNA 172 SP de nuevo.

B | Apéndice: Errores última versión 3.2.0

La última versión proporcionada por el fabricante *«Ultraleap»* del dispositivo LEAP Motion, posee una serie de errores que limitan el potencial del lector de gestos.

Desde que se avanzo de la versión 2 a la 3, se han detectado nuevos errores que previamente no existían. Los desarrolladores no han conseguido averiguar a que se deben y, por tanto, no existe a día de hoy una solución a dichos problemas. Además, como el fabricante no ofrece la posibilidad de descargar versiones anteriores del SDK [1], es difícil encontrar alguna de las versiones anteriores para su descarga.

Fallos de la nueva versión

Algunos de los fallos que presenta esta última versión del SDK [30], son mostrados a continuación con la finalidad de entender en que consisten y que pueden suponer durante el uso del dispositivo LEAP Motion.

- La acción de auto actualizado de la versión no funciona correctamente, por lo que hay que actualizarla de forma manual hasta versiones posteriores.
- Algunas aplicaciones ya no son compatibles con la última versión: Digit Duel, NY Times, Google Earth, Out of the Blocks, PWN9, and TVO Kids Caterpillar Count.
- El seguimiento de algunas acciones o funciona como lo hacía en las versiones V2. Esto supone que los gestos que vienen por defecto guardados en las librerías no funcionan correctamente. Asimismo, hay determinadas funciones importantes durante el desarrollo de cualquier aplicación basada en el uso del dispositivo LEAP Motion que dan error al ejecutarse, como es el caso de: hands.isLeft(), hands.isRight() (que son funciones que sirven para identificar si la mano que se encuentra en visión del

[1] Un kit de desarrollo de software, también conocido como SDK (Software Developement Kit) es un conjunto de herramientas que ofrece el fabricante de una plataforma hardware o un sistema informático. Los SDK permiten que unos desarrolladores puedan crear aplicaciones en determinados lenguajes de programación y facilitar al usuario dichas piezas para poder llevar a cabo el trabajo deseado. Los SDK suelen incluir: un compilador, un depuraador, interfaces (API), documentación, bibliotecas, editores, entornos de desarrollo, herramientas de prueba, controladores y protocolos de red [29].

dispositivo es la derecha o la izquierda) o finger.type() (que sirve para detectar cuál es cada uno de los dedos de la mano), entre otros.

- La detección de herramientas sostenidas en las manos del usuario no es compatible con esta última versión.

Recomendaciones durante el uso

Además de estos errores de la versión 3.2.0 del dispositivo LEAP Motion, si se desea un correcto funcionamiento del lector de gestos es necesario seguir las indicaciones del fabricante, evitando, en medida de lo posible, ocultar una mano detrás de la otra dado que puede haber situaciones en las que el dispositivo no llegue a detectar ambas manos y por otra parte no se recomienda el uso de brazaletes, relojes, anillos o mangas, dado que pueden interferir en el seguimiento de los gestos impidiendo su correcto funcionamiento.

Posible solución

Con la intención de poder ejercer un control cómodo con ambas manos y que limite los posibles fallos de detección de las mismas, es necesario el uso de las funciones: *hands.isLeft()* y *hands.isRight()*.

Para ello, en la bibliografía el link [31], permite la descarga de unas librerías que contienen archivos que permitirán ejecutar las funciones sin problemas. Para ello se descargará el archivo *LeapJava.jar* y los archivos que se encuentran dentro de las carpetas paralelas dependiendo del tipo de ordenador para el que se requieran. En este proyecto se han necesitado los archivos mostrados en la Figura B.1, sustituyendo el archivo *LeapJava.jar* por el proporcionado por el fabricante, así como los dos archivos internos de la carpeta señalada, siendo estos: *Leap.dll* y *LeapJava.dll*.

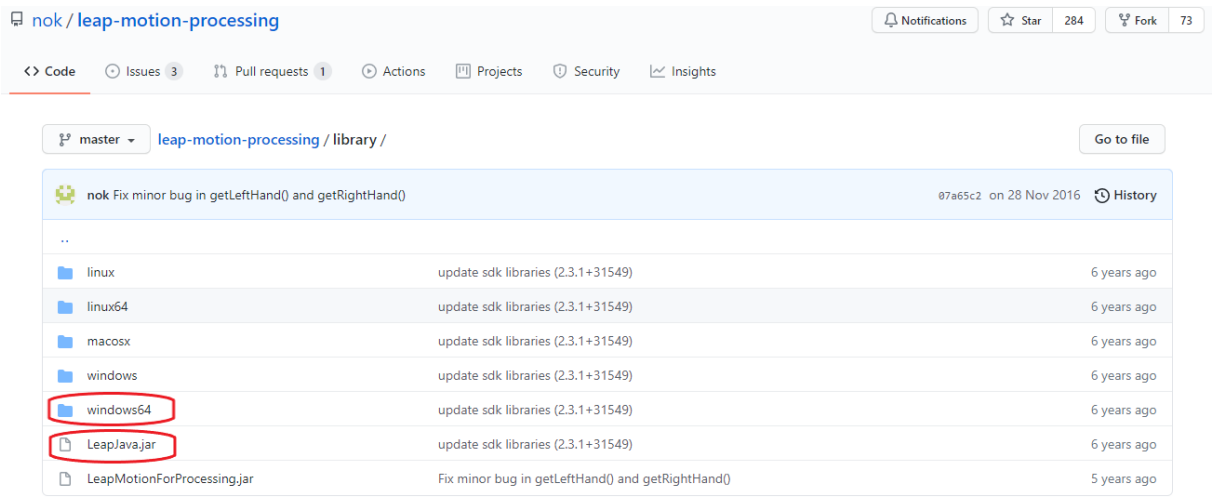

Figura B.1. Archivos descargados para Windows de 64 bits

Una vez se encuentren estos archivos en la carpeta deseada y se ejecute el programa de NetBeans, las funciones que no eran detectadas, anteriormente, funcionarán. Cabe destacar que el resto de fallos como la no detección de los gestos básicos del dispositivo LEAP Motion o los errores de detección debidos al uso de complementos en la mano o el posicionamiento incorrecto de esta, seguirán posibilitando fallos en la detección por parte del dispositivo LEAP Motion.

Bibliografía

[1] Grupo One Air. *GROWTH AND FORECAST 2020 – 2039*. 2020. URL: https://www.grupooneair.com/analysis-global-growth-commercial-aviation/. (Accessed: 20/05/2021).

[2] Robert Y.Wang y Jovan Popovic. *Real-time hand-tracking with a color glove*. 2009. URL: https://dl.acm.org/doi/10.1145/1531326.1531369. (Accessed: 22/06/2021).

[3] Carlos Sagüés Blázquiz. "Reconocimiento e interpretación de gestos con dispositivo Leap". En: *Universidad de Zaragoza* (2013).

[4] Álvaro Edo Martínez. "Diseño de una interfaz de usuario avanzada para software de control aéreo mediante una cámara RGBD". En: *Universidad Politécnica de Valencia* (2019).

[5] Kevin Esperanza Martínez. "Aplicación de análisis gestual al guiado de drones". En: *Universidad Politécnica de Valencia* (2019).

[6] Lorena Calvente Roldán. "Sistema de aterrizaje automatizado de aeronaves basado en técnicas de aprendizaje reforzado". En: *Universidad de Sevilla* (2019).

[7] George Böhnisch. *LEAP Motion - X-Plane Integration*. 14/02/2013. URL: https://www.youtube.com/watch?v=CnvwuhBO6K4. (Accessed: 29/06/2021).

[8] SimHanger Flight Simulation. *Leap Motion for VR Pilots. Time to throw away your Yokes? : VR Guide for Flight Simmers : PART 4*. 22/04/2019. URL: https://www.youtube.com/watch?v=74u87KBCapE. (Accessed: 29/06/2021).

[9] Nataliia Bubniuk. *Hand Tracking and Gesture Recognition Using AI: Applications and Challenges*. 2020. URL: https://www.intellias.com/hand-tracking-and-gesture-recognition-using-ai-applications-and-challenges/#:~:text=Gesture. (Accessed: 21/05/2021).

[10] Ultraleap. *Leap Motion Controller*. 2020. URL: https://www.ultraleap.com/product/leap-motion-controller/. (Accessed: 21/05/2021).

[11] Peter Wozniak y col. *Possible applications of the LEAP motion controller for more interactive simulated experiments in augmented or virtual reality*. 2016. URL: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/9946/99460P/Possible-applications-of-the-LEAP-motion-controller-for-more-interactive/10.1117/12.2237673.full?SSO=1. (Accessed: 22/05/2021).

[12] J. Belda. *Leap Motion (II): principio de funcionamiento*. 2015. URL: https://showleap.com/leap-motion-ii-principio-de-funcionamiento/. (Accessed: 22/05/2021).

[13] Ultraleap. *API Overview*. 2019. URL: https://developer-archive.leapmotion.com/documentation/java/devguide/Leap_Overview.html. (Accessed: 16/06/2021).

[14] Ultraleap. *Using the Leap Motion Control Panel*. URL: https://developer-archive.leapmotion.com/documentation/v2/python/supplements/Leap_Application.html. (Accessed: 23/05/2021).

[15] Ultraleap. *Running the Leap Motion Diagnostics*. 2019. URL: https://support.leapmotion.com/hc/en-us/articles/360004363657-Running-the-Leap-Motion-Diagnostics. (Accessed: 22/05/2021).

[16] X-Plane. *XPlane11*. 2021. URL: https://www.x-plane.com/. (Accessed: 27/05/2021).

[17] Tyler Young. *X-Plane 10 System Requirements*. 2021. URL: https://www.x-plane.com/kb/x-plane-10-system-requirements/. (Accessed: 27/05/2021).

[18] Laminar Research. *Flight Simulator XPLANE11 Manual*. 2017. URL: https://www.x-plane.com/wp-content/uploads/2017/04/Manual_XPlane11_sp_web.pdf. (Accessed: 27/05/2021).

[19] X-Plane. *How X-Plane Works*. 2021. URL: https://www.x-plane.com/desktop/how-x-plane-works/. (Accessed: 27/05/2021).

[20] Java. *Conozca más sobre la tecnología Java*. URL: https://www.java.com/es/about/. (Accessed: 25/05/2021).

[21] Oracle. *JDK 8u111 with NetBeans 8.2*. 2021. URL: https://www.oracle.com/technetwork/java/javase/downloads/jdk-netbeans-jsp-3413139-esa.html. (Accessed: 30/05/2021).

[22] The Apache Software Foundation. *About Apache NetBeans*. 2021. URL: http://netbeans.apache.org/about/index.html. (Accessed: 30/05/2021).

[23] Luiz Fernando Abras Cantoni. "Avaliaçao do uso da linguagem pddl no planejamento de missoes para robôs aéreos". En: *PhD thesis, Universidade General de Minas Gerais* (2010).

[24] Ángel Rodas y Joan Vila. "Materiales de la asignatura: Sistemas de gestión de vuelo por computador". En: *Segundo curso de máster en ingeniería aeroespacial* ().

[25] Ultraleap. *Java SDK Documentation*. 2019. URL: https://developer-archive.leapmotion.com/documentation/v2/java/index.html. (Accessed: 08/06/2021).

[26] Xsquawkbox. *Datarefs for X-Plane*. 2015. URL: http://www.xsquawkbox.net/xpsdk/docs/DataRefs.html. (Accessed: 15/06/2021).

[27] Martin Molina. *Natural user interfaces for human-drone multi-modal interaction*. 1/06/2016. URL: https://www.researchgate.net/publication/304816895_Natural_user_interfaces_for_human-drone_multi-modal_interaction. (Accessed: 29/06/2021).

[28] Microone. *Manos gestos vector iconos conjunto en blanco y negro. Ilustración del símbolo de comunicación*. URL: https://es.123rf.com/photo_69616349_manos-gestos-vector-iconos-conjunto-en-blanco-y-negro-ilustracion.html. (Accessed: 29/06/2021).

[29] Red Hat. *APLICACIONES NATIVAS DE LA NUBE ¿Qué es un SDK?* 2021. URL: https://www.redhat.com/es/topics/cloud-native-apps/what-is-SDK. (Accessed: 31/05/2021).

[30] Ultraleap. *Leap Motion Release Notes*. 2021. URL: https://developer-archive.leapmotion.com/documentation/java/supplements/SDK_Release_Notes.html#version-3-2-0. (Accessed: 30/05/2021).

[31] GitHub. *leap-motion-processing*. 2021. URL: https://github.com/nok/leap-motion-processing. (Accessed: 15/06/2021).