**Logotipo

Descripción generada automáticamente**

**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA**

**INGENIERIA DE COMPUTADORES**

**“INTRUMENTOS DE VUELO Y SISTEMAS OPERATIVOS EN TIEMPO REAL”**

**Realizado por**

**FRANCISCO JAVIER DÍAZ VENTURA**

**Dirigido por**

**ÁNGEL FRANCISCO JIMÉNEZ FERNÁNDEZ**

**Departamento**

**ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES**

**Sevilla, (junio,2024)**

# Resumen

Este trabajo de fin de grado se abordar la creación del panel de instrumentos de vuelo que nos permita ver en tiempo real la información relevante del sobre el estado de la aeronave, mostrarla en pantalla y permitir al usuario interaccionar con ella.

Para ello se hará uso de un simulador de vuelo realista como Xplane11, del cual se extraerán los datos que se quieren mostrar en la pantalla, también se desarrollar un cliente que nos permita extraer los datos del simulador y realizar la transferencia de esos datos hacia un microcontrolador a través de una conexión USB.

Se utilizará un microcontrolador de la familia STM32 junto a un sistema operativo en tiempo real, el cual será el encargado de crear y gestionaran las diferentes tareas que controlaran cada uno de los aspectos de este trabajo como la comunicación con el microcontrolador, la generación de alertas de sonido y el renderizado de las imágenes en la pantalla. Para mostrar los datos se diseñarán tres interfaces graficas de usuario que generarán animaciones complejas, además de permitir al usuario interaccionar con el dispositivo a través de una serie de botones, con los que podremos realizar transiciones entre los diferentes instrumentos o activar diferentes funcionalidades del sistema.

**Palabras claves:** RTOS, Sistemas operativos en tiempo real, Tarea, Cola, Mensaje, Cliente, Simulador, STM32, Microcontrolador, RAM, FLASH, CPU, Sistemas empotrados, C/C++.

# Abstract

This final degree project deals with the creation of a flight instrument panel that allows us to see in real time the relevant information about the state of the aircraft, display it on the screen and allow the user to interact with it.

To do this, we will use a realistic flight simulator such as Xplane11, from which we will extract the data we want to show on the screen. We will also develop a client that will allow us to extract the data from the simulator and transfer it to a microcontroller via a USB connection.

A microcontroller of the STM32 family will be used together with a real-time operating system, which will be in charge of creating and managing the different tasks that will control each of the aspects of this work, such as communication with the microcontroller, the generation of sound alerts and the rendering of the images on the screen. To display the data, three graphic user interfaces will be designed to generate complex animations, as well as allowing the user to interact with the device through a series of buttons, with which let the user make transitions between the different instruments or activate different functionalities of the system.

**Keywords:** RTOS, Sistemas operativos en tiempo real, Tarea, Cola, Mensaje, Cliente, Simulador, STM32, Microcontrolador, RAM, FLASH, CPU, Sistemas empotrados, C/C++.

# Agradecimientos

En primer lugar, me gustaría agradecer a mis padres Francisco Javier Díaz y María José ventura por apoyarme durante todos estos años en la carrera para que no me rindiera.

En segundo lugar, a mi tío David, por ayudarme a seguir siempre adelante y ser una persona honrada.

En tercer lugar, a todos mis amigos tanto de la universidad, del instituto y del Erasmus por compartir durante todos esto años el peso de la carrera y apoyarme en los momentos más duros.

Finalmente me gustaría agradecer a mi tutor del TFG Ángel, el cual me ha servido de gran ayudar dando me consejos e ideas, además de confiar en mi durante el desarrollo del proyecto.

# Tabla de contenido

[Resumen 3](#_Toc170079065)

[Abstract 5](#_Toc170079066)

[Agradecimientos 7](#_Toc170079067)

[Tabla de contenido 9](#_Toc170079068)

[Lista de figuras 12](#_Toc170079069)

[1 Descripción del proyecto 14](#_Toc170079070)

[1.1 Introducción 14](#_Toc170079071)

[1.2 Objetivos del proyecto 16](#_Toc170079072)

[1.2 Estado del Arte 16](#_Toc170079073)

[1.4 Requisitos y análisis de riesgos 18](#_Toc170079074)

[1.4.1 Requisitos Hardware 18](#_Toc170079075)

[1.4.2 Requisitos Software 19](#_Toc170079076)

[1.4.3 Análisis de riesgo 19](#_Toc170079077)

[2 Ejecución del proyecto 21](#_Toc170079078)

[2.1 Introducción 21](#_Toc170079079)

[2.2 Diseño del sistema 21](#_Toc170079080)

[2.3 Tecnologías empleadas 22](#_Toc170079081)

[2.3.1 Introducción 23](#_Toc170079082)

[2.3.2 Xplane 11 23](#_Toc170079083)

[2.3.3 XplaneConnect 24](#_Toc170079084)

[2.3.4 C++ 24](#_Toc170079085)

[2.3.5 Visual Studio 2022 25](#_Toc170079086)

[2.3.5 Win32 25](#_Toc170079087)

[2.3.6 STM32F746G Discovery 26](#_Toc170079088)

[2.3.7 FreeRTOS 27](#_Toc170079089)

[2.3.8 TouchGFX 28](#_Toc170079090)

[2.3.9 TouchGFX Designer 29](#_Toc170079091)

[2.3.10 STM32cubeIDE 29](#_Toc170079092)

[2.4 Desarrollo 29](#_Toc170079093)

[2.4.1 Introducción 29](#_Toc170079094)

[2.4.2 Xplane11 29](#_Toc170079095)

[2.4.3 ClienteXplane 30](#_Toc170079096)

[2.4.4 STM32F746G-Discovery 35](#_Toc170079097)

[2.4.5 Tarea 1: Serial\_RX 40](#_Toc170079098)

[2.4.6 Tarea 2: SoundTask 41](#_Toc170079099)

[2.4.7 Tarea 3: Play\_waw\_file 43](#_Toc170079100)

[2.4.8 Tarea 5: Render\_Time 46](#_Toc170079101)

[2.4.9 Tarea 4: TouchGFXTask 46](#_Toc170079102)

[2.5 Pruebas del sistema 52](#_Toc170079103)

[3 Planificación del proyecto 56](#_Toc170079104)

[3.1 Introducción 56](#_Toc170079105)

[3.2 Fases del proyecto 56](#_Toc170079106)

[3.2.1 Diagrama de Gantt 59](#_Toc170079107)

[3.3 Coste del proyecto 60](#_Toc170079108)

[3.4 Estudio de mercado 62](#_Toc170079109)

[3.4.1 Clientes potenciales 62](#_Toc170079110)

[3.4.2 Plan de comercialización 65](#_Toc170079111)

[Conclusión 67](#_Toc170079112)

[Trabajo futuro 69](#_Toc170079113)

[Bibliografía 70](#_Toc170079114)

# Lista de figuras

[Figura 1.1: Logitech G Pro Flight Instrument Panel 17](#_Toc170079115)

[Figura 1.2: Class Echo Flight Panel 18](#_Toc170079116)

[Figura 2.1 Diagrama de bloques de la arquitectura del proyecto 22](#_Toc170079117)

[Figura 2.2: Esquema de los componentes del microcontrolador 27](#_Toc170079118)

[Figura 2.3: Interfaz de configuración de Xplane11 30](#_Toc170079119)

[Figura 2.4: Diagrama de flujo de la aplicación 34](#_Toc170079120)

[Figura 2.5: Esquema del USB-OTG-FS 35](#_Toc170079121)

[Figura 2.5: Diagrama de bloques de la interfaz SDMMC 36](#_Toc170079122)

[Figura 2.6: configuración de pines del chipset TFBGA216 38](#_Toc170079123)

[Figura 2.7: Configuración global del reloj 39](#_Toc170079124)

[Figura 2.8: Diagrama de flujo de la tarea Serial\_RX 41](#_Toc170079125)

[Figura 2.9: Diagrama de SounTask y del GPWS 43](#_Toc170079126)

[Figura 2.10: Formato de la cabecera de los archivos WAVE 43](#_Toc170079127)

[Figura 2.11: Diagrama de flujo de la tarea play\_waw\_file 45](#_Toc170079128)

[Figura 2.12: Framebuffer 2D 47](#_Toc170079129)

[Figura 2.13: Esquema del procesado de imágenes en microcontroladores 48](#_Toc170079130)

[Figura 2.14: Esquema del backend del sistema 49](#_Toc170079131)

[Figura 2.15: Ángulos de una aeronave 50](#_Toc170079132)

[Figura 2.16: Interfaz gráfica del horizonte virtual 50](#_Toc170079133)

[Figura 2.17: Interfaz gráfica de navegación 51](#_Toc170079134)

[Figura 2.18: Interfaz gráfica del motor 51](#_Toc170079135)

[Figura 2.19: Prueba 1 de rendimiento del horizonte artificial. 52](#_Toc170079136)

[Figura 2.20: Prueba 2 de rendimiento del horizonte artificial. 53](#_Toc170079137)

[Figura 2.21: Prueba 1 de rendimiento de la navegación. 53](#_Toc170079138)

[Figura 2.22: Prueba 2 de rendimiento de la navegación. 54](#_Toc170079139)

[Figura 2.23: Prueba 1 de rendimiento de la interfaz del motor. 55](#_Toc170079140)

[Figura 3.1: Diagrama de Gantt 59](#_Toc170079141)

[Figura 3.2: Resumen generado por cloc 60](#_Toc170079142)

[Figura 3.3: Atributos COCOMO 61](#_Toc170079143)

[Figura 3.4: Grafico de ventas de Xplane 11 y 12 63](#_Toc170079144)

[Figura 3.5: Comparación de las ventas de los diferentes simuladores 64](#_Toc170079145)

[Figura 3.6: Comparación del pico máximo de usuarios de cada simulador 64](#_Toc170079146)

# Descripción del proyecto

## 1.1 Introducción

El mundo de la aviación se encuentra en constante cambio incorporando mejoras y nuevas tecnologías para aumentar la seguridad y la eficiencia de las aeronaves.

En sus orígenes los pilotos carecían de sistemas de vuelos sofisticados, dependían de sus sentidos de la orientación y de la observación visual para orientarse. Todo esto cambio trans el fin de la primera guerra mundial cuando el incremento de la complejidad de los aparatos y la duración de los vuelos se hizo evidente la necesidad de diseñar instrumentos que pudieran proporcionar información más precisa sobre la actitud y el rendimiento de la aeronave.

Los primeros instrumentos de vuelo de las aeronaves eran completamente mecánicos y funcionaban mediante principios físico como la gravedad, presión o principios mecánicos como los giroscopios. En la década de los 70 en pleno auge de la Guerra fría, se vio la necesidad de minimizar el tamaño de los sistemas electromecánicos de las aeronaves militares. Apareciendo los primeros circuitos integrados que aglutinaban múltiples funciones en una sola unidad.

En los años 80 con el avance de la tecnología y la evolución de los microprocesadores, empezaron a emerger los primeros sistemas operativos en tiempo real, que se extendieron rápidamente a la industria aeroespacial y automovilística. Permitiendo una gestión eficiente de los sistemas y asegurando que las funciones críticas se ejecuten dentro de los tiempos especificados. Todo esto junto a la aparición de las primeras pantallas LCD, facilito la aparición de las denominadas “glass cockpits”, que fueron reemplazando a los instrumentos de vuelo analógicos, permitiendo mostrar toda la información relevante de la aeronave a través de monitores.

En la actualidad con incremento de la potencia de cálculo de los microprocesadores y la reducción de sus costes de producción, han permitido que la gran mayoría de aeronaves modernas hagan uso de algún tipo de RTOS su gestión y control. Mejorando la precisión, fiabilidad y eficiencias que son factores críticos para la seguridad y el éxito de todas las operaciones de vuelo.

Con este trabajo de fin de grado se busca implementar con éxito un RTOS para la gestión de los instrumentos de vuelos de una aeronave dentro de un microcontrolador.

Este documento se compone de tres capítulos. Los cuales siguen la siguiente estructura:

***Capítulo 1:*** En este primer capítulo se definirán los objetivos que se quieren cumplir en este proyecto, también se realizara un estudio previo del estado del arte y un análisis de requisitos y riesgos para este proyecto.

***Capítulo 2:*** En segundo capitulo se realizara el desarrollo del proyecto donde se explicara de forma individual los diferentes módulos y tecnologías forman este trabajo.

***Capítulo 3:*** Se describe la planificación que se ha seguido para el proyecto, junto a un análisis del mercado y de los costes del producto.

## 1.2 Objetivos del proyecto

El objetivo principal de este proyecto es el desarrollo de un sistema empotrado para la representación de gráficos en tiempo real. En particular se pretende desarrollar los instrumentos de vuelo enfocado para el uso de en sistemas de simulación.

* Obtener los datos generados por el simulador
* Crear un puente de comunicación entre el simulador y el microcontrolador
* Diseño de interfaces gráficas
* Control en tiempo real de las interfaces gráficas mediante el uso de un sistema operativo en tiempo real
* Reproducción de alertas dependiendo del estado de la Aeronave

## 1.2 Estado del Arte

En ámbito de los sistemas empotrados con interfaces gráficas, se ha producido un gran avance en los últimos años, la industria produce cada vez microcontroladores más grandes y con mayor potencia gráfica. Lo que facilita la creación de aplicaciones con gráficos dinámicos. Esto ha permitido que diferentes empresas comercialicen productos similares al que se ha creado con este proyecto.

El primer producto que se va a analizar será el **Logitech G Pro Flight Instrument** Panel. Este dispositivo ha sido creado por Logitech, una empresa de renombre internacional en el sector de periféricos y accesorios para ordenador. Logitech pose una línea de producto especializada en simuladores de vuelos que proporcionan una experiencia inmersiva.

Este producto destaca por los siguientes elementos:

* Pantalla LCD de 3,5 pulgadas
* Soporta los principales simuladores de vuelos
* Se pueden alternar entre 15 opciones de pantalla
* Modularidad y compatibilidad con otros dispositivos
* Buena calidad de la carcasa y el acabado

Tiene ciertas desventajas en comparación con nuestro producto, su alto precio de 142 € lo hace poco atractivo para la mayoría de bolsillo, además por ese precio tan solo ofrecen una pantalla de 3,5 pulgadas. En cambio, con nuestro producto ofrecemos una pantalla de 4,3 pulgadas por 96,75€

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Figura 1.1: Logitech G Pro Flight Instrument Panel

Otro producto similar que se ha encontrado es el **Class Echo**, este producto es muy peculiar porque se oferta como un kit por piezas de forma que el usuario sé a quién lo ensamble. Este kit está formado por:

* Arduino AT Mega 2560,
* Pantalla táctil de 4,3 pulgadas
* Software SPAD neXt.

Si lo comparamos con nuestro producto, podemos ver que por un precio de 130€ el Arduino que se utiliza en el kit es muy inferior en prestaciones al microcontrolador que hemos utilizado en nuestro dispositivo. Provocando que tan solo se pueda utilizar la pantalla para dibujar gráficos simples, si a esto le añadimos que hay que ensamblar los distintos componentes e instalar el firmware en el Arduino. Esto no lo hace acto para las personas sin conocimientos previos en el mundo de los microcontroladores.

Imagen que contiene electrónica, monitor

Descripción generada automáticamente

Figura 1.2: Class Echo Flight Panel

## 1.4 Requisitos y análisis de riesgos

En este apartado se definirán los requisitos del sistema tanto hardware como software y se realizara un minucioso estudio de posibles fallos que se podrían ocasionarse durante el desarrollo y como solventarlos.

### 1.4.1 Requisitos Hardware

Se implementarán los siguientes elementos en el hardware del proyecto:

* Sistema de comunicación serie para crear un puente entre el simulador y el microcontrolador.
* Microcontrolador con potencia suficiente como para reproducir animaciones complejas.
* El microcontrolador debe ser capaz de almacenar las imágenes que se van a usar en la memoria Flash.
* Almacenamiento externo de archivos de audios.
* Sistema de alerta a través de un altavoz.
* Pantalla táctil que permita la interacción con el dispositivo.
* El microcontrolador deberá ser compatible con sistema operativo FreeRTOS

### 1.4.2 Requisitos Software

Los requisitos softwares se encuentran dividido en dos grupos uno para el Firmware del microcontrolador y otro para el cliente

Requisitos Firmware:

* Uso de un protocolo de comunión para la recepción de los datos del simulador
* Sistema de gestión de paquetes de datos, se encargará de clasificar y almacenar los datos que se vayan recibiendo.
* Reproducción de alertas según el estado de la aeronave
* Gráficos dinámicos que se actualicen con la información entiempo real del simulador de vuelo
* Implementación en el sistema operativo en tiempo real FreeRTOS

Requisitos Cliente:

* Lectura del socket donde el simulador vuelva los datos
* Separa los datos en paquetes distintos
* Notificar si se producen fallos de lectura del socket
* Establecer una comunicación con el microcontrolador para transmitir los datos del simulador

### 1.4.3 Análisis de riesgo

En la siguiente tabla se realizará un análisis de posibles riesgos, al que podría ser sometido este proyecto. Con el fin de ver cuál sería su impacto y posibles soluciones a dicho problema.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Descripción** | **Probabilidad** | **Impacto** | **Mitigación** |
| RI-01 | Compatibilidad de Hardware y Software. | Baja | Alto | Realizar pruebas de compatibilidad durante la fase de inicio del proyecto. |
| RI-02 | Problemas de rendimiento en la pantalla. | Media | Alta | Utilizar técnicas de optimización para reducir el número de píxel que se van a renderizar. |
| RI-03 | Retrasos en la comunicación de los datos. | Media | Medio | Utilizar protocoles de comunicación eficiente junto buffers para proporcionar una transmisión fluida. |
| RI-04 | Comportamiento no esperado de las tareas del RTOS | Baja | Alto | Hacer uso de las herramientas que nos proporciona el RTOS para la sincronización y control de tareas. |
| RI-05 | Corrupción o perdida del proyecto | Baja | Alto | Uso de herramientas de control de versiones como Git |
| RI-06 | Falta de documentación oficial sobre alguna de las tecnologías empleadas | Alta | Medio | Buscar otras fuentes no oficiales sobre su funcionamiento o buscar una alternativa a dicha tecnología |

# 2 Ejecución del proyecto

## 2.1 Introducción

En esta sección, se mostrará de manera general cada uno de los módulos que componen el proyecto y como los diferentes módulos interaccionan entre ellos.

Como podemos ver en la siguiente imagen, el proyecto se ha dividido en dos partes:

## 2.2 Diseño del sistema

Interfaz de usuario gráfica

Descripción generada automáticamente

Figura 2.1 Diagrama de bloques de la arquitectura del proyecto

1. **Sistema Windows:** Representa a las llamadas del sistema operativo y APIs de Windows, de las que depende la aplicación ClienteXplane para funcionar. Dentro de la arquitectura global actúa como un intermediario facilitado la comunicación entre el simulador y la aplicación y permitido el envío de datos al microcontrolador.
   * Xplane11: Es simulador de vuelo que se va a utilizar, se encargara de publicar los datos del simulador en localhost del ordenador.
   * ClienteXplane: Es una aplicación que he desarrollado para permitir la comunicación Simulador-Microcontrolador, su trabajo será el de acceder al localhost y leer los mensajes que se vayan publicado, al mismo tiempo los enviará a la placa a través una conexión USB.
2. **STM32F746G:** Es el microcontrolador que se ha utilizado para el desarrollo del proyecto, el cual ejecuta siguientes tareas para controlar los diversos periféricos que se tienen conectados.
   * Serial\_Rx: Esta tarea será la encargada de recibir y procesar los paquetes de datos que se envíen desde el clienteXplane a través de una conexión USB con el ordenador. Una vez haya recibido los datos, comunicará dichos datos al resto de tareas que se estén ejecutando.
   * SoundTask: Es la responsable de comprobar si se cumplen una serie de condiciones, según que prioridad tengan llamara a la tarea play\_waw\_file para reproducir el archivo de audio acorde a la situación.
   * play\_waw\_file: Es una tarea one-shot que se encarga de transferir los archivos de audio que hay guardados en una memoria externa al descodificador de audio para reproducido por el altavoz
   * TouchGFXTask: Es la tarea que controla que se dibuja en pantalla, esta tarea recibe los datos que van llegando del simulador y actualiza la pantalla para mostrar
   * Render\_time: Es la tarea encargada de medir el retraso entre cada frame que renderiza el sistema e imprimir los datos en la pantalla.

## 2.3 Tecnologías empleadas

### 2.3.1 Introducción

En este capítulo hablaremos de las tecnologías utilizadas durante el desarrollo del proyecto. Empezaremos hablando del simulador de vuelo Xplane11 y todas las herramientas que se han empleado para poder obtener los datos de vuelo del simulador. Y concluiremos con una visión en profundidad de la placa de desarrollo STM32F746G.

### 2.3.2 Xplane 11

Logotipo

Descripción generada automáticamente

Xplane11 es una herramienta de simulación de vuelo desarrollado por Laminar Research.

Lo utilizan pilotos profesionales y entusiastas de la aviación.

Xplane fue creado en los 90, como proyectó personal de Austin Meyer, un estudiante de informática de la universidad de Minnesota, que deseaba crear un simulador que proporcionara una experiencia más realista y autentica que los simuladores que existían en aquella época.

* Tiene un motor de vuelo realista y detallada, que le permite simular con precisión el comportamiento de las aeronaves en diferentes condiciones atmosféricas y de vuelo.
* Es una herramienta multiplataforma soportada por Windows, Linux, Mac os, iOS y Android
* Dispone de cientos de opciones para que los usuarios puedan personalizar y modificar aviones, escenarios y condiciones atmosféricas según sus preferencias
* Gráficos y escenarios detallados. Xplane utiliza texturas de alta definición junto a efecto de iluminación realistas para proporcionar una inmersión completa al usuario

Xplane está desarrollado en C++, un lenguaje de programación de alto rendimiento utilizado en los sistemas de simulación. Xplane además también hace uso de otros lenguajes de programación como Python o Lua para ciertas funciones como la creación de scripts y plugins que se utilizan en el desarrollo de contenido adicional y modificaciones

### 2.3.3 XplaneConnect

Xplane Connect(XPC) es una herramienta de investigación de código abierto desarrollada por la NASA para interactuar con Xplane. Permite al usuario controlar y recibir información en tiempo real sobre el estado de las aeronaves, que están dentro de la simulación. XPC nos permite hacer uso de programas personalizados o de terceros a través de C, Python, MATLAB, Java. Esta herramienta además usa protocolos de comunicación permitiendo que el cliente y el simulador se encuentren en máquinas distintas.

### 2.3.4 C++



Es un lenguaje de programación de alto rendimiento, creado en los años 80 por

Bjarne Stroustrup, como extensión del lenguaje C, se caracteriza para acceder a las características de bajo nivel del ordenador.

Las características principales son:

* **Orientado a objetos:** C++ hace uso de clases y objetos, facilitando la modularidad y la reutilización del código.
* **Eficiencia:** C++ destaca por su eficiencia en tiempo de ejecución y en el uso de recursos, permitiendo un control preciso de la memoria y la optimización del código.
* **Portabilidad:** Los programas desarrollados en C++ pueden ser compilados y ejecutado en un gran número de plataformas diferentes.
* **Bibliotecas:**  C++ dispone de una gran variedad de bibliotecas, que facilitan el desarrollo del código, como Win32 y librería estándar de C++.
* **Compatibilidad con C:** C++ permite ejecutar y compilar código escrito en C.

### 2.3.5 Visual Studio 2022

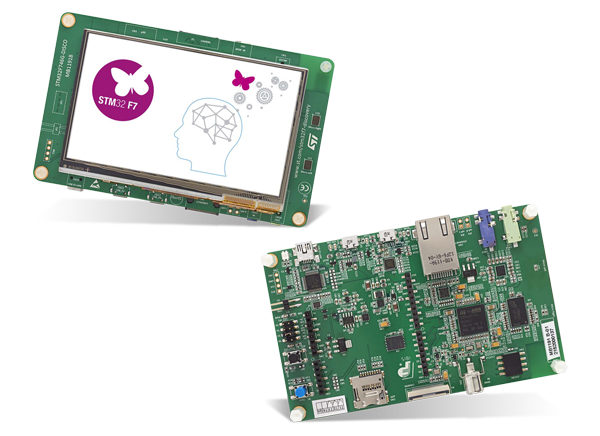
Es un entorno de desarrollo creado por Microsoft, está diseñado para ofrecer herramientas y funcionalidades avanzadas para el desarrollo de software en una amplia gama de plataformas y lenguajes de programación. Sus características principales son:

* **Rendimiento**: Optimizado para manejar proyectos de cualquier tamaño y complejidad, gracias a un entorno de desarrollo de 64 bits.
* **Soporte de múltiples lenguajes y plataformas**: Como C#, C++, Visual Basic y plataformas como .NET, Unity, Azure.
* **Productividad**: Integración de herramientas de control de versiones como GIT, que permiten a los desabolladores trabajar sobre sus repositorios.
* **Depuración**: Herramientas de depuración mejoradas con la capacidad de analizar el rendimiento de la aplicación y el uso de la memoria en tiempo real.

### 2.3.5 Win32

Win32 API, es la plataforma original para aplicaciones de Windows en C y C++ que requieren acceso directo a Windows y al hardware. Proporcionan una amplia gama de funciones para manejar ventanas, gestionar archivos, creación de hilos y manipulación de recursos del sistema.

### 2.3.6 STM32F746G Discovery



Es un kit de desarrollado y demostración creada por STMicroelectronics para el microcontrolador STM32F746NGH6. A continuación enumeraremos todas sus características principales:

* **Microcontrolador STM32F746NGH6:** Este microcontrolador está basado en el núcleo ARM Cortex-M7 de 32 bits, tiene 1Mbyte de memoria Flash y 340 Kbyte de memoria RAM.
* **Memoria:** 128 Mbits de Quad-SPI Flash y 128 Mbit de SRAM (solo son accesible 64Mbits).
* **Pantalla:** Una pantalla de color LCD-TFT de 4,3 pulgadas con una resolución de 480\*272 pixeles
* **Conectores:** Cámara, microSD card, USB Type-B ST-LINK/V2-1, 2x micro USB, Ethernet RJ45, SPDIF RCA de entrada, Audio Jack, ARDUINO UNO v3
* **Interfaces de comunicación:** BUS CAN, UART, SPI, I2C

* **Botones y Leds:** 2 botones, uno de reset y otro para el usuario, además de múltiples leds de diferentes colores.
* **Conectores de expansión de Arduino:** La placa hace uso del formato de conectores ARDUINO Uno V3 que nos permite acceder a una gran cantidad de accesorios para la placa.

Gráfico de dispersión

Descripción generada automáticamente

Figura 2.2: Esquema de los componentes del microcontrolador

### 2.3.7 FreeRTOS



Es un sistema operativo en tiempo real con kernel de código abierto. Esta diseñado de manera que sea simple y pequeño, haciéndolo idóneo para su uso en un amplio rango de aplicaciones empotradas, desde aplicaciones simples de control de sensores, hasta aplicaciones más complejas como dispositivos IoT o automatización industrial.

Aspectos destacables de FreeRTOS:

* Planificador en tiempo real: FreeRTOS nos proporciona un planificador basado en prioridades, permitiéndonos ejecutar las tareas en función de su prioridad, generando un comportamiento determinista crítico.
* Gestión de tareas: El sistema operativo nos proporciona las herramientas para crear diferentes tareas cada una con su propia pila de llamada y nivel de prioridad.
* Gestión de recursos: FreeRTOS proporciona mecanismos para gestionar recursos como mutex, semáforos, colas y temporizadores de software, facilitando la sincronización y comunicación entre tareas.
* Gestión de memoria:
* Portabilidad:

### 2.3.8 TouchGFX



Es un framework de desarrollo de interfases de usuario de alto rendimiento, optimizado para los microcontroladores STM32. Destaca por los siguientes elementos.

* Alto rendimiento: Optimizado para aprovechar al máximo los recursos hardware del microcontrolador.
* Diseño visual e intuitivo: Permite la creación de interfaces graficas sofisticadas con animaciones fluidas y gráficos de alta calidad.
* Integración con las herramientas de desarrollo: Compatible con STM32cubeMX y con los entornos de desarrollo de stm32cubeIDE y Keil.
* Biblioteca de widgets: Proporciona una amplia variedad de widgets como botones, gráficos, cuadro de textos que podemos configurar según las necesidades de nuestro proyecto.

### 2.3.9 TouchGFX Designer

Es una herramienta de diseño grafica que forma parte del framework de TouchGFX, esta herramienta permite a los desarrolladores crear interfaces graficas de usuario de forma sencilla y rápida. TouchGFX Designer permite generar automáticamente el código necesario para la interfaz de forma que se ahorre tiempo y se reduzca la posibilidad de errores.

### 2.3.10 STM32cubeIDE



Es una plataforma avanzada de desarrollo C/C++ que nos proporciona STMicroelectronics, está diseñado específicamente para los microcontroladores STM32. Este entorno de desarrollo combina múltiples herramientas del ecosistema STM32 como, configuración del hardware, desarrollo software, depuración, compilación y análisis del código en una única plataforma facilitando y mejorando el desarrollo de firmwares de los microcontroladores STM32.

## 2.4 Desarrollo

### 2.4.1 Introducción

Como ya comentamos al comienzo de este capítulo, proyecto se encuentra dividido en dos partes, la primera corresponde con el ordenador donde se está ejecutando la simulación y la segunda con el microcontrolador. Cada parte está compuesta de múltiples módulos que se desarrollaron a continuación.

### 2.4.2 Xplane11

Es el simulador que se ha seleccionado para realizar este proyecto, su principal ventaja respecto a las otras alternativas que se estuvieron barajando al comienzo del desarrollo del proyecto. Es el soporte oficial para publicar y recibir datos desde el simulador algo que el resto de los simuladores no poseen o hay que pagar un extra por ello.

Xplane11 hace uso de los protocolos de red UDP-IP para publicar dichos datos. Para habilitar la publicación de los datos el simulador cuenta con una ventana de configuración, que nos permite seleccionar la IP y puerto donde se van a enviar los paquetes de datos y seleccionar que datos van a ser los que enviemos.

Se utilizará la dirección IP 127.0.0.1 que corresponde con el localhost de mi equipo y el puerto 49003.

Interfaz de usuario gráfica

Descripción generada automáticamente

Figura 2.3: Interfaz de configuración de Xplane11

Cada paquete que envíe XPlane11 tendrá la forma de una matriz de 9x10 donde cada fila seguirá la siguiente estructura, Data[0] es el índice del dataset que podemos observar en la Figura 2.1 y todos los elementos desde Data [1] hasta Data[9] serán el contenido del dataset formado por números de tipo Float.

### 2.4.3 ClienteXplane

Es una aplicación especifica que se ha desarrollado para este proyecto, ha sido desarrollada en C++ para hacer uso de las funciones que trae la API de Windows y de la librería XplaneConnect, permitiendo la comunicación entre el microcontrolador y la simulación.

Para acceder al socket donde se están publicando los datos, se utilizan las funciones que trae la Liberia XplaneConnect, a la que le pasamos los siguientes parámetros:

* **Dirección IP:** Se utilizará la dirección del localhost (127.0.0.1).
* **Puerto 1:** Puerto donde publica Xplane11 el dataset (49003).
* **Puerto 2:** Puerto que utiliza XplaneConnect para escuchar (49007).
* **Float Data [10][9]:** Un Buffer para almacenar los datos.
* **Filas:** El número de filas que se quieren leer.

Si la operación se ha realizado con éxito, obtenemos los datos dentro del buffer y se procederá a cerrar el socket.

De todos los elementos del dataset solo se van a utilizar doce, los cuales corresponden con:

* **Velocidad:** Velocidad respecto al suelo del avión.
* **Alabeo, Cabeceó y Dirección:** Valores de los ejes rotación del avión.
* **Altitud y Altitud radar:** La primera es la altura respecto al nivel de mar y la segunda es la distancia al suelo.
* **RPM y Temperatura:** Revoluciones por minuto del y temperatura a la que se encuentra el motor.
* **Flujo de combustible:** Cantidad de combustible que está consumiendo el motor.
* **Tanque de combustibles:** Peso del combustible que hay dentro de los tanques.
* **Flaps:** Posición en la que se encuentran los flaps.
* **VVI:** Incremento de la velocidad vertical del avión.
* **Temperatura y presión del aceite:** Temperatura y presión a la que se encuentra el aceite.

Debido a que la conexión con el microcontrolador se realizar a través un USB operando en modo full speed, los mensajes que se transmitan no podrán ser superior a 64 bytes de tamaño, para solucionar este problema se han repartido entre 2 mensajes las distintas variables del dataset, cada mensaje utiliza el primer Byte como cabecera y separa cada dato mediante un espacio en blanco para facilitar su tratamiento una vez que lleguen al microcontrolador.

0 1 2 63

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cabecera |  | Dato1 |  | Dato2 | Dato3 |  | Dato4 |  | Dato5 |  | Dato6 |  | Dato7 |

Figura 2.2: Esquema de la trama del mensaje

Una vez se ha construido el mensaje, se llama a la función **CreateFile** de la API de Windows, esta función nos permite acceder a **dispositivos de E/S** como si se trataran de archivos almacenados en el disco. A esta función hay que suminístrale el nombre del dispositivo que se quiere abrir y el tipo de operación que se va a realizar, esta puede ser de Escritura, lectura o ambas. En este caso el nombre del dispositivo será L”COM6” y se habilitará tanto la lectura como la escritura. Si se ha creado con éxito, la función nos devolverá el identificador del dispositivo.

Como el dispositivo está haciendo uso de una interfaz serie virtual, hay que configurar ciertos parámetros tanto en el microcontrolador como en el cliente para que se pueda establecer la comunicación con éxito.

Parámetros que configurar:

* **BaudRate:** Velocidad de transmisión del puerto serie para este proyecto se ha establecido la velocidad de 115200 Baudios/segundo.
* **ByteSize:** Numero de Bits en los Bytes que se van a transmitir, se van a utilizar 8 Bits por cada Byte.
* **StopBits:** Número de bits para señalizar el fin del mensaje, solo se va a usar un bit de stop.
* **ParityBit:** Este bit se utilizar para comprobar errores producidos durante la transmisión, para este proyecto se mantendrá desactivado.

Si se ha configurado correctamente el dispositivo, se procederá a transmitir el mensaje haciendo uso de la función WriteFile a la que debemos darle el identificador del dispositivo, el mensaje que se quiere enviar y el número de bytes a enviar.

Diagrama

Descripción generada automáticamente

Figura 2.4: Diagrama de flujo de la aplicación

### 2.4.4 STM32F746G-Discovery

El microcontrolador STM32F746G-Discovery es el corazón de este trabajo, es el encargado de ejecutar las distintas tareas que se ha creado y responder en tiempo real a una serie de eventos que se han definido.

Voy a empezar mostrando la configuración interna utilizada para el microcontrolador y después se explicará de manera detallada el funcionamiento de las distintas tareas.

**Pinout y configuración**

* **Conectividad:**

* + USB\_OTG\_FS:

Como hemos mencionado en punto 2.3.2 el microcontrolador utiliza el protocolo USB para establecer la comunicación con el simulador. El puerto USB utiliza la interfaz Full-Speed (FS), que destaca por operar una velocidad de 12 Mbit/s y usar la especificación USB 2.0. Adicionalmente la funcionalidad OTG (On-The-Go) nos permite configurar el microcontrolador para que funcione como dispositivo USB o como Host USB, en nuestro caso funcionara como dispositivo.

**Diagrama

Descripción generada automáticamente**

Figura 2.5: Esquema del USB-OTG-FS

Para que el USB funcione es necesario configurar su frecuencia del reloj a 48 MHz, a través del divisor PLLQ.

* + SDMMC1:

Es la interfaz de host que se utiliza para la lectura/escritura de tarjetas de memorias SD, puede funcionar hasta una frecuencia máxima de 50 MHz, como podemos ver en la figura 2.6, para este proyecto operara a 48 MHz por las limitaciones del microcontrolador. El bus de datos tiene dos modos de operación, bus de 1 bit de ancho o bus de 4 bits de ancho. Debido a que la documentación proporcionada por STMicroelectronics esta desactualizada y existen fallos en el entorno de desarrollo STM32CUBEIDE, el bus operara en modo 1 bit de ancho.

Diagrama, Esquemático

Descripción generada automáticamente

Figura 2.5: Diagrama de bloques de la interfaz SDMMC

* **Núcleo del sistema:**

* + DMA:

Para liberar al procesador de carga de trabajo, se ha utilizado el acceso directo a memoria. De modo que el rendimiento de la pantalla no se vea mermado por las constantes interrupciones del USB y de la tarjeta SD. Se han utilizado los siguientes canales del DMA:

|  |  |  |  |
| --- | --- | --- | --- |
| Petición DMA | Stream | Dirección | Prioridad |
| USART1\_RX | DMA2 Stream 2 | Periférico a Memoria | Low |
| DMMC1\_RX | DMA2 Stream 3 | Periférico a Memoria | Low |
| DMMC1\_TX | DMA2 Stream 6 | Memoria a Periférico | Low |

* **Multimedia:**
  + SAI:

La interfaz SAI (Serial audio interface) nos va a permitir comunicarnos desde nuestro microcontrolador con procesador de audio externo. Esta interfaz cuenta con dos canales (A y B), los cuales se va a configurar como Maestro y Esclavo síncrono respectivamente. Además, se debe configurar la frecuencia del audio para que funciones a 44.1 KHz.

* + DMA2D:

Una de las peculiaridades de este microcontrolador es que posee un bus DMA especializado que sirve como acelerador gráfico, el cual utilizaremos para renderizar las imágenes que se muestren por pantalla.

* **Middleware:**
  + FREERTOS: Es el RTOS que se ha utilizado para desarrollar este proyecto, se ha configurado para que el sistema operativo tenga reservado 250Kbytes de heap.
  + FATFS: Para que esta librería pueda leer correctamente la tarjeta de memoria SD se han tenido que configurar los siguientes parámetros:
    - USE\_LFN: Se activado en modo buffer dinámico en el stack.
    - MAX\_LFN: Se le ha asignado el valor máximo de 255.
    - MAX\_SS: Se han definido sectores máximos de 4096 Bytes.
    - FS\_EXFAR: Se ha activado para permitir que se puedan leer memorias en formato exFAT.
  + TOUCHGFX: Se ha configurado el framework para que utilice CMSIS\_RTOS\_V2 y la resolución de la pantalla del microcontrolador.

* + USB\_DEVICE: La librería USB se ha configurado para que funcione como un dispositivo de comunicación del tipo puerto virtual COM

**Imagen que contiene Patrón de fondo

Descripción generada automáticamente**

Figura 2.6: configuración de pines del chipset TFBGA216

Diagrama, Esquemático

Descripción generada automáticamente

Figura 2.7: Configuración global del reloj

* **Problemas ocasionados:**
  + DMA y D-Cache:

Originalmente, se estaba utilizando la usart1 para realizar la comunicación con el simulador, pero a medida que fui avanzado con el desarrollo del proyecto se empezaron a producir problemas en la comunicación. Después de realizar consultar los foros de STmicroelectronics y en la documentación del microcontrolador, encontré que el fallo era generado por la cache de datos, la cual no estaba siendo invalidada al usar el DMA. De las cuatro soluciones propuesta por el fabricante no encontré ninguna que me solventase el problema. Por lo tanto, decidí utilizar otra alternativa como la librería USB\_DEVICE.

* + Librería BSP

El STmicroelectronics proporciona una librería genérica para cada familia de microcontroladores, esta librería proporciona funciones y ejemplos de cómo utilizar los distintos componentes del microcontrolador. Para este proyecto se hace uso de uno de sus módulos para controlar el procesador de audio de la placa.

Esta librería y la mayoría de sus ejemplos no ha sido pensado para ser usado en un proyecto con un sistema operativo en tiempo real. Tras analizar el código, vemos que para generar una espera dentro de la librería se utiliza la función HAL\_Delay();. Esto va a generar diferentes problemas en nuestro sistema como pueden ser condiciones de carrera, perdida de rendimiento o retrasos en tareas de alta prioridad.

Para evitar este problema hay que hacer uso de las funciones de FreeRTOS o de CMSIS\_OS-v2 para generar esa espera. Se modifico la librería y se sustituyeron todos los HAL\_Delay(1); por vTaskDelay(1);

### 2.4.5 Tarea 1: Serial\_RX

Esta tarea tiene la función de recibir y almacenar los paquetes que vayan llegando desde el clienteXplane. A medida que los paquetes llegan al microcontrolador, son almacenados en un buffer a la espera de ser procesados. En el momento que el procesador se encuentre disponible, se comprueba el primer byte del buffer para saber qué tipo de mensaje es el que se ha recibido y seleccionar la forma en que debe de ser dividido. Para poder dividir el mensaje se ha uso de la función strtok(), esta función nos proporciona una forma de dividir el mensaje utilizando un delimitador. Una vez que se han extraído los datos, estos se encuentran almacenados como un tipo char, existen dos métodos en la librería estándar de C llamados atoi() y atof() que nos permite transformar de tipo char a int y float.

Esta tarea tiene una alta frecuencia de ejecución para que no acapare todo el tiempo de ejecución y no afecte al rendimiento del resto de las tareas, se le ha asignado una prioridad de valor 1 que corresponde con la menor prioridad posible.

Diagrama

Descripción generada automáticamente

Figura 2.8: Diagrama de flujo de la tarea Serial\_RX

### 2.4.6 Tarea 2: SoundTask

SoundTask es una tarea de baja prioridad que se ejecuta de manera recurrente para comprobar si el botón LND de la pantalla ha sido pulsado, este botón nos sirve para señalizar al microcontrolador que tenemos intención de aterrizar la aeronave.

Si es pulsado, se activará el Sistema de advertencia de proximidad al suelo (GPWS), este sistema comprueba la altura respecto al suelo de la aeronave utilizando una serie de intervalos definidos en el código, si nuestra altura se encuentra dentro de alguno de los intervalos se lanzará la tarea play\_waw\_file.

Para evitar que existan múltiples instancias de la tarea play\_waw\_file al mismo tiempo, se hace uso de las notificaciones entre tareas para bloquear SoundTask cuando se crea play\_waw\_file, Soundtask se desbloqueara cuando se complete la reproducción del archivo de audio o se supere el tiempo de espera de 7000 ms

Diagrama

Descripción generada automáticamente

Figura 2.9: Diagrama de SounTask y del GPWS

### 2.4.7 Tarea 3: Play\_waw\_file

Play\_waw\_file tiene la misión de reproducir una serie de archivos de audios almacenados en la tarjeta de memoria SD. Estos archivos se encuentran almacenado en el formato WAVE PCM, que es un subconjunto de la especificación RIFF de Microsoft para archivos multimedia. Los archivos WAVE están divididos en dos partes. La primera parte del archivo corresponde con el Header o cabecera, como podemos ver en la figura 2.8 consta de una longitud de 44 bytes, donde se almacena los parámetros que se van a utilizar para reconstruir la onda como la velocidad de muestreo, el número de canales, el tamaño de las muestras y el número total de muestras.

Imagen que contiene Tabla

Descripción generada automáticamente

Figura 2.10: Formato de la cabecera de los archivos WAVE

La segunda parte corresponde con las muestras que componen la onda. Haciendo uso de la librería FatFs, la tarea busca el archivo coincida con el nombre que se le paso como parámetro. Si el archivo es encontrado con éxito, guarda en un buffer los primeros 4096 bytes de muestras. A continuación, se comunican estos datos directamente al procesador de audio a través del DMA y se desplaza el puntero del buffer al siguiente bloque de datos. La tarea da por finalizada la lectura del archivo cuando se leen 0 byte, cuando ocurre esto la tarea procede a apagar el procesador de audio seguido de cerrar el archivo que se estaba leyendo y desbloquea la tarea SoundTask para que pueda seguir con su ejecución. Finalmente se destruye a sí misma y finaliza su ejecución.

Originalmente esta tarea tenía un problema que ocasionaba el bloqueo de todo el sistema. Esto se producía si play\_waw\_file estaba siendo ejecutada y el planificador la sustituía por otra tarea, ocasionando un bloqueo del DMA y del resto de las tareas. Como solución decidí asignarle la máxima prioridad del sistema, esto evita que ninguna tarea de menor prioridad pueda adueñarse del procesador e interrumpa la ejecución de play\_waw\_file.

Diagrama

Descripción generada automáticamente

Figura 2.11: Diagrama de flujo de la tarea play\_waw\_file

### 2.4.8 Tarea 5: Render\_Time

Para realizar las pruebas del sistema se ha diseñado una tarea que mide el retraso entre cada fotograma que se genera. TouchGFX utiliza varios GPIOS para generar señales de salida con información que nos permite medir el rendimiento del sistema. En esta tarea se hace uso de la señal **FRAME\_RATE**, cuyo valor de la señal se va alternando cada vez que se actualiza un fotograma. Podemos medir la diferencia de tiempo entre cada fotograma utilizando el Systick del microcontrolador el cual genera un tick cada milisegundo.

Cada vez que se actualiza la pantalla volvemos a leer el valor del Systick, al cual le sustraemos el valor de la iteración anterior. El valor resultante es el número de ticks o latencia entre cada fotograma.

Si dividimos 1000 entre la latencia obtenida, el resultado es número de fotogramas por segundo que está generando el microcontrolador.

### 2.4.9 Tarea 4: TouchGFXTask

Finalmente tenemos la tarea principal de todo el proyecto. Esta tarea hace uso del framework grafico de Touchgfx, este framework incorporar una serie de widgets y contenedores que facilitan la creación de pantallas. Antes de explicar de manera individual el funcionamiento de cada pantalla y como interactúan con el resto de las tareas, se va a explicar el funcionamiento de los gráficos empotrados en este proyecto.

Con el fin de poder crear aplicaciones gráficas, se hace uso de los siguientes componentes de un microcontrolador.

**MCU**

El MCU es el encargado de realizar todos los cálculos necesarios para poder dibujar la imagen. Para ello lee las imágenes almacenadas en la flash y la envía a la CPU para ser procesadas. La CPU realizar la unión de varias imágenes para generar una nueva o el renderizado de diferentes formas geométricas. El resultado obtenido es enviado a la memoria RAM, donde se mantiene a la espera para ser trasladado a la pantalla.

**RAM**

Como ya hemos mencionado en el punto anterior, en la RAM se almacena el resultado de la nueva imagen calculada, ha esto se le denominara el framebuffer.

El framebuffer está formado por posiciones contiguas de la memoria RAM, que generan un mapa de dos dimensiones, donde cada posición de ese mapa corresponde con un píxel de la imagen. Cada posición almacena el color del pixel correspondiente como un valor de 24 bits, esto nos permite generar 224 colores diferentes

Ancho de la imagen

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0,0 | 1,0 | 2,0 | … | x-1,0 |
| 0,1 | 1,1 | … |  | … |
| 0,2 | … |  |  | … |
| … |  |  |  | … |
| 0,y-1 | … | … | … | x-1,y-1 |

Altura de la imagen

Figura 2.12: Framebuffer 2D

**Flash**

En la Flash es donde se almacena todo el contenido estático, como imágenes, textos y fuentes. El MCU se encarga leer su contenido para escribirlo o combinarlo con la RAM.

Para facilitar la lectura, la memoria flash suele estar mapeada, de modo que los pixeles de la imagen se pueden transferir de manera directa desde la flash a la RAM.

**Pantalla**

La pantalla es donde se va a mostrar el contenido de la imagen final que se encuentra en la RAM.

Interfaz de usuario gráfica, Aplicación, Teams

Descripción generada automáticamente

Figura 2.13: Esquema del procesado de imágenes en microcontroladores

Ahora que se ha visto como se generan las imágenes, voy a explicar la comunicación backend de la aplicación. TouchGFX implementa una serie de clases e interfaces que separa mediante una capa de abstracción la aplicación grafica del resto del sistema.

**Clase Model**

Se encarga de definir y almacenar el estado de los datos que van a ser mostrado en pantalla. Esta clase implementa la función tick(), que es llamada de manera automática al comienzo de cada frame para reaccionar ante eventos del microcontrolador.

**Clase View**

Es una interfaz pasiva que muestra en pantalla los datos que se encuentran en la clase Model, además reacciona ante eventos generados por los propios usuarios y enviar esa información al Presenter

**Clase Presenter**

Es el responsable de la lógica de negocio de la pantalla que se encuentre activa en ese momento. Actuará como intermediario entre Model y View cada vez que ocurra un evento.

Interfaz de usuario gráfica

Descripción generada automáticamente

Figura 2.14: Esquema del backend del sistema

Se han diseñado tres interfaces graficas de usurario, cada una de ella simula un instrumento de vuelo de la aeronave.

La primera interfaz que se ha creado simula el horizonte artificial, aquí se le mostrara al usuario la información más relevante. La velocidad y la altitud se mostrarán como valores numéricos en cada extremo de la pantalla, toda la pantalla se ira moviendo en tiempo real, para mostrar los ángulos de alabeo y cabeceo de la aeronave respecto del horizonte. Todas las pantallas incorporan cuatro botones comunes para alternar la vista y activar el indicador de FPS, esta además cuenta con un quinto botón LND que sirve para señalizar al microcontrolador que se va a comenzar la maniobra de aterrizaje.

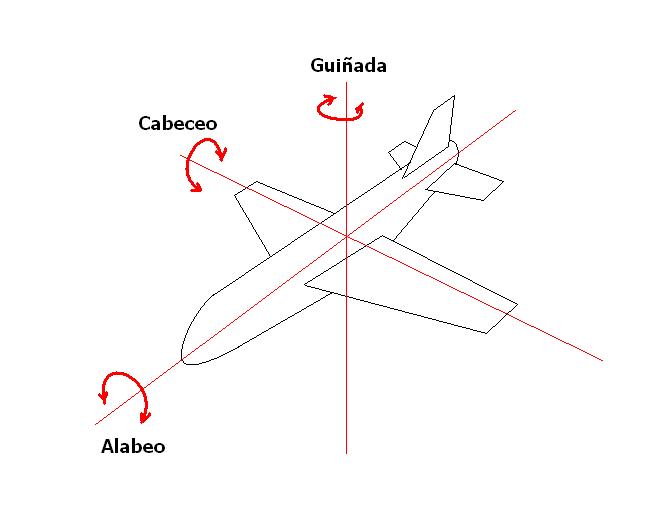


Figura 2.15: Ángulos de una aeronave

Interfaz de usuario gráfica

Descripción generada automáticamente

Figura 2.16: Interfaz gráfica del horizonte virtual

La segunda interfaz será utilizada para la navegación, esta pantalla incorpora una bruja que nos señala el curso actual de la aeronave en la simulación, además se ha implementado dentro de la interfaz dos botones que nos permite señalar un nuevo rumbo.

Imagen de la pantalla de un videojuego con un reloj

Descripción generada automáticamente con confianza baja

Figura 2.17: Interfaz gráfica de navegación

La última interfaz que se ha creado sirve como un cuadro de mando donde se mostrara toda la información importante relativa al estado del motor como puedo ser las RPM, temperaturas, presiones y cantidad de combustible el depósito.

Imagen de la pantalla de un video juego

Descripción generada automáticamente con confianza media

Figura 2.18: Interfaz gráfica del motor

## 2.5 Pruebas del sistema

Con el fin de analizar y medir el rendimiento de cada pantalla del sistema de manera individual, se ha diseñado un plan de pruebas con el cual se pretender medir el rendimiento de las interfaces bajo diferentes condiciones de carga. Además de identificar posibles cuellos de botella que afecte a la fluidez de las animaciones y a la experiencia del usuario.

Haciendo uso de la tarea Render\_Time podemos imprimir en la pantalla en tiempo real las estadísticas del tiempo de respuesta y los FPS que se están generando.

**Pantalla del horizonte artificial**

Esta es la pantalla con mayor coste computacional del sistema, para generar la animación es necesario realizar un gran número de operaciones de rotación y translación para ajustar la pantalla.

Bajo condiciones de normales cargas, el microcontrolador es capaz de generar un promedio de 30 FPS estables.

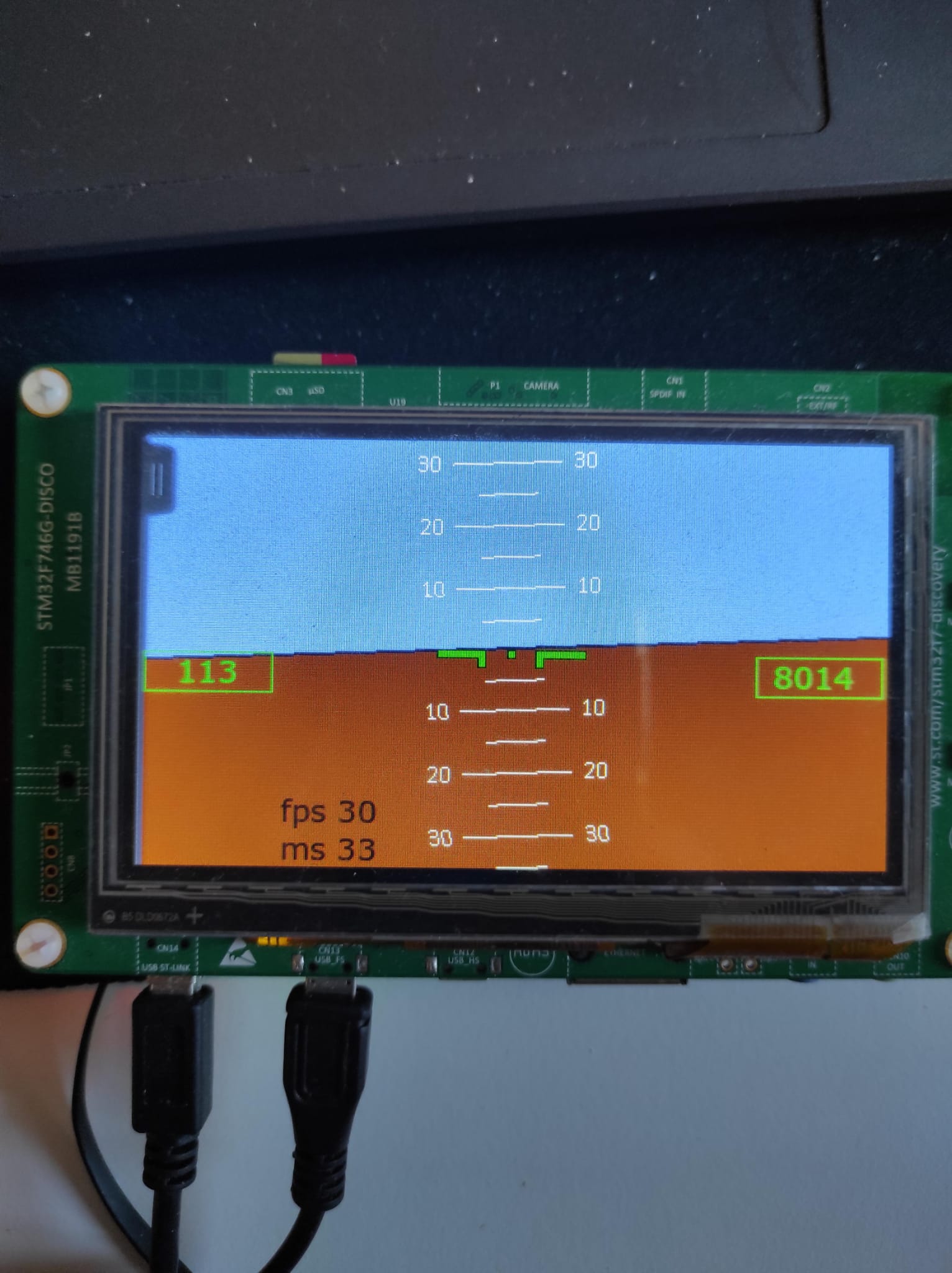


Figura 2.19: Prueba 1 de rendimiento del horizonte artificial.

En condiciones de niveles de carga altos, se pueden observar dos casos distintos dependiendo del tipo de movimiento. Si se produce un movimiento de translación en vertical, el rendimiento prácticamente se mantiene constante. Sin embargo, al realizar una rotación que es una operación más compleja y costosa, se produce una pérdida de rendimiento significativa que afecta directamente a la fluidez de las animaciones.

Imagen de la pantalla de un computador

Descripción generada automáticamente con confianza media

Figura 2.20: Prueba 2 de rendimiento del horizonte artificial.

**Pantalla de navegación**

Aunque esta pantalla es visualmente más simple que la anterior, sufre del mismo problema debido a las rotaciones. En niveles de cargas normales produce 30 FPS constantes. Pero si producen picos de cargas debidos a movimientos espontáneos el rendimiento puede verse afectado durante un breve instante.

Un reloj despertador

Descripción generada automáticamente con confianza media

Figura 2.21: Prueba 1 de rendimiento de la navegación.

Al seguir realizando pruebas, me he percatado de si cambiamos el valor del rumbo “HDG” con los botones de selección, el rendimiento pantalla cae. Es posible que la causa de esto se ha debido a la forma en cómo se calcula la orientación relativa del cursor del rumbo respecto a la brújula. De manera que cuando pulsamos el botón el dispositivo realiza el doble de operaciones de rotación una para calcular la orientación de la brújula y otra para calcular la orientación relativa del cursor respecto a la de la brújula.

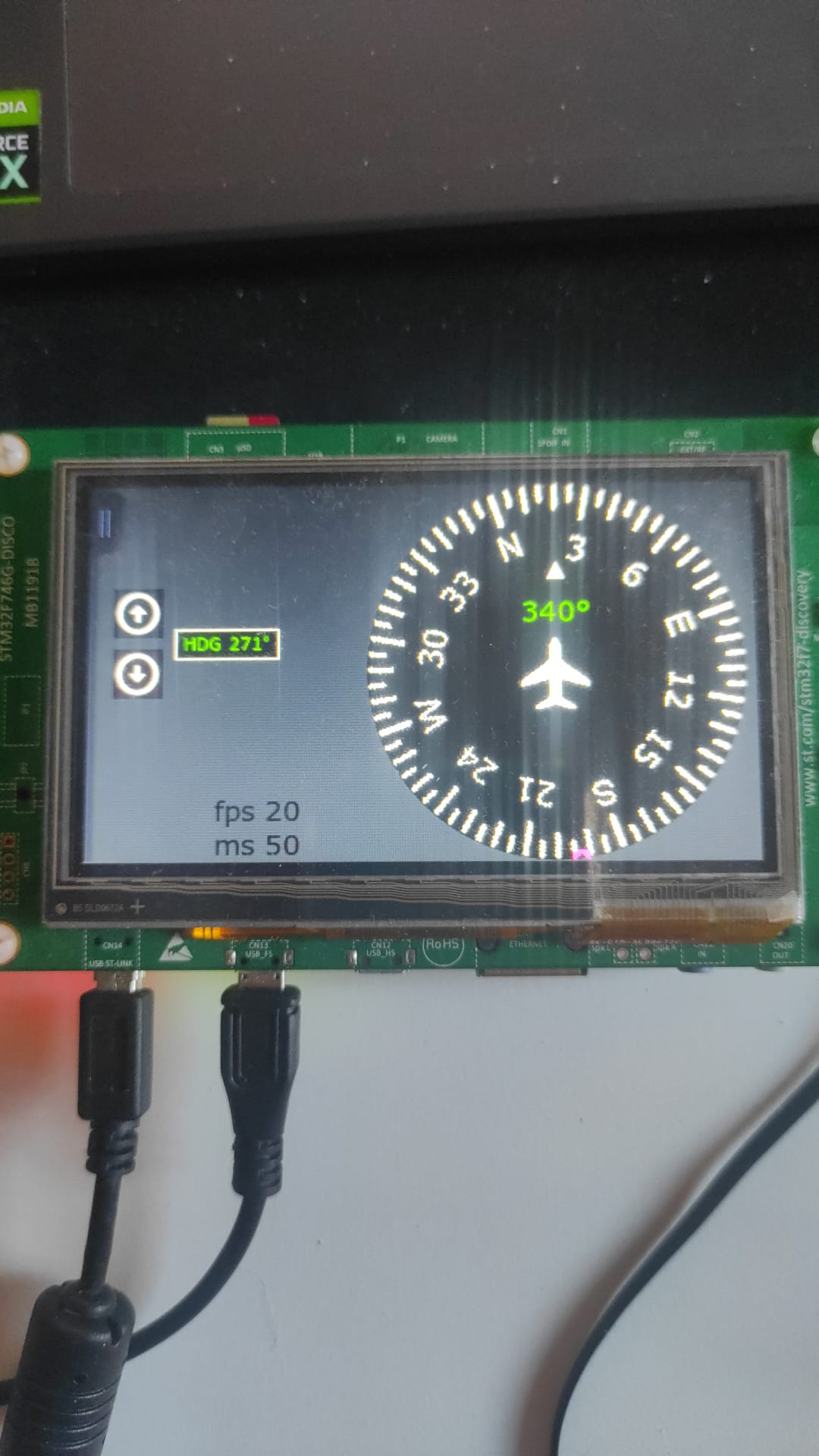


Figura 2.22: Prueba 2 de rendimiento de la navegación.

**Pantalla del rendimiento del motor**

Esta última pantalla, es la que menor coste computacional le supone al dispositivo. Debido a que tan solo realiza operaciones simples, como actualizar la información de los cuadros de texto y mover algunos indicadores. Esto produce un rendimiento constante de 60 FPS y una buena fluidez de las animaciones de la pantalla.

Interfaz de usuario gráfica

Descripción generada automáticamente

Figura 2.23: Prueba 1 de rendimiento de la interfaz del motor.

# 3 Planificación del proyecto

## 3.1 Introducción

En este capítulo se va a realizar la organización del proyecto, de forma que se aproxime a las trescientas horas estipuladas para un trabajo fin de grado. Además, se calculará el coste total de proyecto y se elaborará un plan de comercialización para el producto que se ha creado.

## 3.2 Fases del proyecto

Al comienzo del proyecto, este fue dividido en fases, cada fase consta de una serie de actividades y tareas, que se deben de completar para dar como finalizada la fase.

**Fase 1. Fase de Inicio:**

* A1: Definición de los objetivos y el alcance del proyecto [10 horas]
  + T1: Reuniones con el tutor del TFG [2 horas]
  + T2: Elaboración de los objetivos y el alcance [8 horas]
* A2: Investigación de tecnologías disponibles [2 horas]
  + T1: Elección del simulador de vuelo [1 horas]
  + T2: Selección del microcontrolador [1 hora]
* Hito1**:** Definición de los objetivos y el alcance

**Fase 2. Desarrollo del clienteXplane:**

* A1: Estudio librería XplaneConnect y de la API de Windows (4 horas)
  + T1: Integración en el cliente (4 horas)
* A2: Desarrollo de la aplicación (15 horas)
  + T1: programación de la aplicación (15 horas)
* Hito2**:** Completado el desarrollo del cliente

**Fase 3. Desarrollo del microcontrolador:**

* A1: Implementación del sistema de comunicación (20 horas)
  + T1: Desarrollo de la tarea receptora (16 horas)
  + T2: Prueba de comunicación (4 horas)
* A2: Desarrollo de las interfaces graficas (110 horas)
  + T1: Diseño de las interfaces graficas (30 horas)
  + T2: Implementación dentro del proyecto (80 horas)
* A3: Integración del audio (100 horas)
  + T1: Estudio de su funcionamiento (50 horas)
  + T2: Implementación dentro del proyecto (50 horas)
* Hito3:Completado el desarrollo del microcontrolador.

**Fase 4. Pruebas y ajustes:**

* A1: Pruebas del sistema (10 horas)
  + T1: Prueba de rendimiento (10 horas)
* A2: Corrección de errores (20 horas)
  + T1: Corrección de errores en la pantalla (10 horas)
  + T2: Corrección de errores de sonido (10 horas)
* Hito4: Finalización de pruebas y ajustes.

**Fase 5. Redacción de la documentación:**

* A1: Escritura de la memoria del TFG (40 horas)
  + T1: Escritura de la memoria del TFG (40 horas)
* Hito5:Finalizada la redacción de la memoria y entrega del proyecto.

|  |  |  |
| --- | --- | --- |
| **Fase** | **Tiempo estimado** | **Tiempo** **real** |
| Fase de Inicio | 14 horas | 12 horas |
| Desarrollo del clienteXplane | 19 horas | 21 horas |
| Desarrollo del microcontrolador | 200 horas | 240 horas |
| Pruebas y ajustes | 30 horas | 30 horas |
| Redacción de la documentación | 40 horas | 43 horas |
| **Total Proyecto** | **303 horas** | **346 horas** |

Como podemos observar en la planificación real, se han consumido 346 horas para el desarrollo de este proyecto, la principal causa de este retraso ha sido generado durante la Fase 3 del desarrollo del proyecto. Durante el desarrollo de dicha etapa surgieron una gran cantidad de errores críticos, que impedían continuar con el desarrollo del proyecto hasta que fueran solventados.

### 3.2.1 Diagrama de Gantt

Utilizando la herramienta de gestión y planificación de proyectos OpenProject, se ha creado este diagrama de Gantt para representar de manera visual la planificación del proyecto a lo largo del curso.

Gráfico

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

Figura 3.1: Diagrama de Gantt

## 3.3 Coste del proyecto

A continuación, se van a desarrollar los costes de este proyecto. El coste total se desglosará en coste material y coste del personal.

El coste del personal se ha calculado utilizando el modelo **COCOMO**, este modelo matemático nos permite realizar el cálculo del coste de un proyecto software. COCOMO puede configurarse con distintos nivel y modos, dependiendo de la naturaleza del proyecto y el número de líneas de código que lo forman. Este proyecto se ha calculado en nivel intermedio y modo empotrado.

Utilizando la herramienta **cloc,** que nos permite ver el número de líneas que forman el proyecto, se han obtenido los siguientes datos.

Texto

Descripción generada automáticamente con confianza media

Figura 3.2: Resumen generado por cloc

Estos datos están formados tanto por las líneas de código escritas por mí, como las generadas de manera automática. COCOMO I no tiene forma de diferenciar y medir el coste entre ellas. Por lo tanto, voy a reducir a 1500 las líneas que se van a introducir en la ecuación. Para intentar obtener un resultado que refleje la realidad.

|  |  |  |
| --- | --- | --- |
| **Atributos** | **Valor** | **Multiplicador** |
| Atributos del software | | |
| Fiabilidad | Alto | 1,15 |
| Tamaño de Base de datos | Muy bajo | 1,16 |
| Complejidad | Muy Alto | 1,30 |
| Atributos del hardware | | |
| Restricciones de tiempo de ejecución | Muy alto | 1,11 |
| Restricciones de memoria | Alto | 1,06 |
| Volatilidad de la máquina | Muy bajo | 0,87 |
| Tiempo de respuesta del ordenador | Bajo | 0,87 |
| Atributos del personal | | |
| Capacidad de análisis | Nominal | 1,00 |
| Experiencia en la aplicación | Nominal | 1,00 |
| Calidad de los programadores | Nominal | 1,00 |
| Experiencia en la máquina virtual | Nominal | 1,00 |
| Experiencia en el lenguaje | Nominal | 1,00 |
| Atributos del proyecto | | |
| Técnicas actualizadas de programación | Alto | 0,91 |
| Utilización de herramientas de software | Alto | 0,91 |
| Restricciones de tiempo de desarrollo | Alto | 1,04 |

Figura 3.3: Atributos COCOMO

**Modelos de estimación**

, en personas - mes

, en meses

, en personas

Aplicando los modelos anteriores a los multiplicadores que se han definido en la tabla se han obtenidos los siguientes resultados.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Modo | A | B | C | D | SLOC | Esfuerzo | Duración | Personal |
| Empotrado | 4.822075863263745 | 1.2 | 2.5 | 0.32 | 3100 | 7.8439 | 4.8319 | 1.6236 |

Consideremos que solo hay una persona en el proyecto, la duración de este será de 7.8439 meses. Sobrepasando por un par de meses la duración real del proyecto.

Se ha contratado un Ingeniero de Computadores a jornada completa, con un sueldo bruto de 35.000€ al año, que genera un gasto de 18.22 €/h durante las 346 horas que ha durado el proyecto. Esto nos supone un **coste de personal de 6304.12€.**

Al ingeniero se le ha prestado un equipo de la empresa para trabajar, este equipo tiene un precio inicial de 1500€, con una vida útil de seis años, será usado durante los 6 meses que dura el proyecto. Junto al equipo también se le ha asignado una licencia para poder hacer uso del simulador de vuelo con un valor 19.99 €.

**Cálculo Inicial del proyecto**

|  |  |  |
| --- | --- | --- |
| **Partida** | **Cálculo** | **Total** |
| Coste de personal | 18,22 €/h x 300h | 5466€ |
| STM32F746G-Discovery | 1 x 54,41€ | 54,41€ |
| Cable USB 2.0 Micro B | 1 x 3,29€ | 3,29€ |
| Cable USB 2.0 Mini B | 1 x 3,29€ | 3,29€ |
| Altavoz | 1 x 7,32€ | 7,32€ |
| Amortización del equipo | 1.400€/72 meses x 6 meses | 116,6€ |
| Licencia Xplane11 | 1 x 19.99 € | 19.99€ |
| Tarjeta de memoria microSD | 1 x 5.99 € | 1 x 5.99 € |
|  | **Total previsto:** | **5676,89€** |

**Cálculo Real del proyecto**

|  |  |  |
| --- | --- | --- |
| **Partida** | **Cálculo** | **Total** |
| Coste de personal | 18,22 €/h x 346h | 6304€ |
| STM32F746G-Discovery | 1 x 54,41€ | 54,41€ |
| Cable USB 2.0 Micro B | 1 x 3,29€ | 3,29€ |
| Cable USB 2.0 Mini B | 1 x 3,29€ | 3,29€ |
| Altavoz | 1 x 7,32€ | 7,32€ |
| Amortización del equipo | 1.400€/72 meses x 6 meses | 116,6€ |
| Licencia Xplane11 | 1 x 19.99 € | 19.99€ |
| Tarjeta de memoria microSD | 1 x 5.99 € | 1 x 5.99 € |
|  | **Total real:** | **6514.89€** |

## 3.4 Estudio de mercado

### 3.4.1 Clientes potenciales

El producto que se quiere comercializar tiene un alto nivel de complejidad haciéndolo solamente acto para adolescentes y adultos, que tienen un interés en el mundo de la aviación y ordenadores.

Este proyecto depende gran medida del simulador de vuelo X-plane11, esto nos abre un abanico de oportunidades para vender nuestro producto a los usuarios de este simulador. Las ventas reales de X-plane11 es información privada, que pertenece a Laminar Research, pero utilizando herramientas de análisis como SteamDB, podemos extraer información aproximada de la situación del mercado.

Esta plataforma en línea utiliza la API de Steam para realizar un seguimiento en tiempo real de todos los juegos y jugadores que se encuentran en la plataforma, y a partir de ahí realizar una estimación del número de ventas totales. SteamDB nos proporciona información de tres fuentes distintas sobre el número de ventas.

Figura 3.4: Grafico de ventas de Xplane 11 y 12

Si nos quedamos con la media de las tres fuentes, podemos ver tenemos 336.553 potenciales clientes, que podrían mostrar cierto interés en el producto. Estos datos tan solo son para la plataforma de Steam para la versión digital. X-plane11 también se puede comprar en a través de su página oficial y Amazon en formato físico.

Este parte se va a dejar para el trabajo futuro por falta de tiempo, pero otra forma que tendría de expandir el producto seria dando soporte a otros simuladores como Microsoft Flight Simulator y DCS los cuales poseen una base de usuarios mucho más amplia que X-plane11.

Figura 3.5: Comparación de las ventas de los diferentes simuladores

Figura 3.6: Comparación del pico máximo de usuarios de cada simulador

### 3.4.2 Plan de comercialización

Ahora que ya se conoce la situación del mercado, se va a calcular el precio de venta al público y a realizar un plan de amortización del producto.

Partiendo de los gastos que hemos calculado en el apartado 3.2, se va a suponer que se realiza un pedido al proveedor de 2500 unidades con un 20% de descuento y se quiere obtener un margen de beneficios de 25% por unidad.

|  |  |  |  |
| --- | --- | --- | --- |
| **Descripción** | **Cantidad** | **Precio** | **Total** |
| STM32F746G-Discovery | 1 | 54,41€ | 54,41€ |
| Cable USB 2.0 Mini B | 1 | 3,29€ | 3,29€ |
| Tarjeta de memoria microSD | 1 | 5.99€ | 5,99€ |
| Amortización del proyecto | 1 | 6514,89€/2500 = 2,6€ | 2,6€ |
| **Precio unidad:** | | | **66,09€** |
| **Descuento del 20 %** | | | 52,87€ |
| **Beneficio del 25 %** | | | 79,96€ |
| **IVA 21 %** | | | 96,76€ |
| **Precio de venta final:** | | | **96,75**€ |

El precio de venta es de **96,75€** por cada unidad, tras el análisis de mercado obtuvimos que existían aproximadamente **336.533** potenciales clientes. Si de esta población un 10% acaba comprando nuestro producto. Obtendríamos unos ingresos de **3.255.956,77€** con nos generaría un beneficio de **911.667,89€.**

# Conclusión

El desarrollo de este trabajo de fin de grado ha supuesto ser un proyecto desafiante a la vez que enriquecedor a nivel personal y técnico. Al comienzo de este documento se definieron una serie de objetivos técnicos a alcanzar:

* **Adquisición y comunicación de Datos:** Se ha desarrollado con éxito un cliente que funciona como puente de comunicación entre el microcontrolador y el simulador.
* **RTOS:** Se ha integrado con éxito un sistema operativo en tiempo real, el cual ha sido fundamental para gestionar de manera eficiente las diferentes tareas de este proyecto.
* **Interfaces graficas:** Se han diseñado en incorporado con éxito al proyecto las diferentes interfaces para cada instrumento de vuelo.
* **Alertas de sonido:** Se ha creado con éxito un mecanismo para la lectura y reproducción de archivos de audio almacenados en una memoria externa.

Todos los objetivos se han cumplido con éxito y ha dado como resultado un proyecto funcional que abre muchas puertas a distintas áreas de investigación y desarrollo de las que hablaremos en el próximo apartado.

A nivel personal, me siento satisfecho con este trabajo a pesar de las múltiples adversidades que han ido surgiendo en el proyecto a lo largo de su desarrollo. Me he demostrado a mí mismo de que tengo los conocimientos técnicos para llevar a cabo el desarrollo un proyecto de ingeniería, y la capacidad de solventar cualquier problema que pueda surgir. Este proyecto pone fin a esta etapa de mi vida dentro del Grado de Ingeniería Informática de Computadores y abre el comienzo de la siguiente.

# Trabajo futuro

Aunque los objetivos propuestos se han cumplido con éxito, sin embargo, por falta de tiempo hay varias ideas que fueran desechadas o no se llegaron a implementar completamente dentro del proyecto.

* **Comunicación bidireccional:** Permitir que la placa enviara datos al simulador, el código esta implementado en el cliente, pero por falta de tiempo no se pudo realizar la cuarta interfaz, que hubiese permitido controlar la aeronave desde el microcontrolador.
* **Piloto automático:** Como hemos mencionado en el punto anterior, haber realizado un piloto automático que permitiera estabilizar la aeronave a una altura y velocidad definida por el usuario.
* **Nuevas interfaces:** Crear nuevas interfaces graficas que muestren otros tipos instrumentos de la aeronave.
* **Soporte de múltiples simuladores:** En el estudio de mercado, se hablo de la idea de dar soporte a múltiples simuladores de forma que tuviéramos acceso a un mayor número de clientes.
* **Uso en aeronaves reales:** Expandir el proyecto para integrar el control automático de drones o sustituir los datos del simulador por lecturas reales de los sensores.
* **Mejoras del hardware:** Aunque el microcontrolador que se ha utilizado es suficientemente potente para aplicaciones de domótica e Iot, en este proyecto se ha visto que posee grandes carencias a nivel de hardware como falta de capacidad de procesamiento gráfico.

# Bibliografía

1. https://www.x-plane.com/kb/data-set-output-table/
2. https://developer.x-plane.com/datarefs/
3. https://github.com/nasa/XPlaneConnect
4. https://www.st.com/en/microcontrollers-microprocessors/stm32f7-series/documentation.html
5. https://support.touchgfx.com/docs/introduction/welcome
6. https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-writefile
7. https://cplusplus.com/
8. https://www.st.com/resource/en/application\_note/an4839-level-1-cache-on-stm32f7-series-and-stm32h7-series-stmicroelectronics.pdf
9. https://www.youtube.com/watch?v=3iyVYSjK-ZU&t=195s
10. https://www.keil.com/pack/doc/CMSIS/RTOS2/html/group\_\_CMSIS\_\_RTOS.html
11. https://www.freertos.org/Documentation/161204\_Mastering\_the\_FreeRTOS\_Real\_Time\_Kernel-A\_Hands-On\_Tutorial\_Guide.pdf
12. https://www.st.com/resource/en/reference\_manual/rm0385-stm32f75xxx-and-stm32f74xxx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf
13. http://soundfile.sapp.org/doc/WaveFormat/
14. https://strs.grc.nasa.gov/repository/forms/cocomo-calculation/