UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

GRADO EN INGENIERÍA DEL SOFTWARE

INTERFAZ USANDO ARDUINO PARA LA SIMULACION MECANICA DE INSTRUMENTOS ANALOGICOS EN UN SIMULADOR DE VUELO

INTERFACE USING ARDUINO FOR THE MECHANICAL SIMULATION OF ANALOGIC INSTRUMENTS IN A FLIGHT SIMULATOR

Realizado por
JUAN JESÚS TORREÑO MARTÍN

Tutorizado por
PABLO PÉREZ TRABADO

Departamento
ARQUITECTURA DE COMPUTADORES
UNIVERSIDAD DE MÁLAGA

MÁLAGA, SEPTIEMBRE DE 2020

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERIA DE SOFTWARE

INTERFAZ USANDO ARDUINO PARA LA SIMULACION MECANICA DE INSTRUMENTOS ANALOGICOS EN UN SIMULADOR DE VUELO

INTERFACE USING ARDUINO FOR THE MECHANICAL SIMULATION OF ANALOGIC INSTRUMENTS IN A FLIGHT SIMULATOR

Realizado por
Juan Jesús Torreño Martín

Tutorizado por
Pablo Pérez Trabado

Departamento
Arquitectura de computadores

UNIVERSIDAD DE MÁLAGA

MÁLAGA, SEPTIEMBRE DE 2020

Fecha defensa: de octubre de 2020

Resumen

Un problema habitual a la hora de construir simuladores de vuelo que emulan cabinas de pilotaje de aviones ligeros, es la replicación fidedigna de los instrumentos analógicos de estos aviones, que deben obtener de un simulador de vuelo los datos a representar. La forma habitual de construir estas replicas, implementando artesanalmente para cada instrumento su interfaz al simulador, y su sistema electromecánico, es demasiado laborioso y costoso en tiempo, especialmente en cabinas con decenas de instrumentos.

Este TFG tiene como objetivo el desarrollo de un paquete de software que proporcione un procedimiento e interfaz genérico y estandarizado para la construcción de este tipo de replicas de instrumentos. El software desarrollado en este trabajo permitirá obtener del simulador de vuelo los datos de cualquier instrumento de la cabina y, tras procesarlos, entregarlos a un modulo software que, corriendo sobre hardware Arduino, controle los motores paso a paso con que se implementa el instrumento. La aplicación incluye también interfaces graficas de usuario que simplifican tanto la elección del instrumento a replicar como la configuración de su implementación física.

Palabras clave: Arduino, Simulador de vuelo, .NET Framework, Instrumento, Aviónica.

Abstract

A common problem when building flight simulators that emulate light aircraft cockpits is the reliable replication of the analogic instruments of these aircraft, which must obtain the data to represent from a flight simulator. The usual way of building these replicas, implementing by hand its interface to the simulator and its electromechanical system for each instrument, is too laborious and time consuming, especially in cabins with dozens of instruments.

The goal of this Final Degree Project is to develop a software package that provides a generic and standardized procedure and interface for the construction of this type of instrument replicas. The software developed in this work will make it possible to obtain data from any instrument in the cockpit from the flight simulator and, after processing it, deliver it to a software module that, running on Arduino hardware, controls the stepper motors with which the instrument is implemented. The application also includes graphical user interfaces that simplify both the choice of the instrument to be replicated and the configuration of its physical implementation.

Keywords: Arduino, Flight Simulator, .NET Framework, Instrument, Avionics.

Índice

1

Introducción

1.1 Motivación

Este trabajo surge a raíz de la problemática que presenta la construcción de simuladores de vuelo con cabinas de pilotaje de aviones que usan instrumentación analógica. Estos instrumentos funcionan en las aeronaves reales mediante diales mecánicos movidos por dispositivos que reaccionan a cambios en magnitudes físicas de la nave (presión del aire, altura y orientación del avión, etc.), y cuyo movimiento y operación viene determinado por el tipo de dispositivo (de presión, giroscópico, o magnético).

La simulación software de estos instrumentos en simuladores de vuelo como X-Plane 10 es sencilla, ya que se basa en un valor digital calculado por el juego y que se representa gráficamente mediante una réplica realizada mediante software que emula el funcionamiento del instrumento en cuestión. Sin embargo, la emulación de uno de estos instrumentos en las réplicas físicas de cabinas realizadas en simuladores semiprofesionales es mucho más compleja, pues requiere el uso de un dispositivo electromecánico con una carcasa y diales que reproduzcan fielmente el aspecto del instrumento replicado, y que usen motores eléctricos para mover, en forma convincente, las agujas o diales del instrumento, usando como entrada el valor calculado por el simulador para la magnitud a representar en el instrumento. En particular, la construcción de uno de estos instrumentos replicados plantea dos problemas:

El precio de los dispositivos usados para la réplica del instrumento debe ser reducido, usando siempre que sea posible componentes hardware estándar. Este objetivo de hacer barato el instrumento adquiere especial importancia si el simulador se está construyendo con fondos muy limitados, procedentes de donaciones o aportaciones voluntarias (como ocurre, por ejemplo, en el simulador actualmente en construcción en el museo aeronáutico del aeropuerto de Málaga).

El número de horas de trabajo requeridas para programar y construir el instrumento simulado debe también minimizarse lo más posible (puesto que este trabajo suele ser realizado por voluntarios, trabajando a tiempo parcial).

Esto significa que los instrumentos replicados deben compartir un interfaz común de comunicación con el simulador y de control de los dispositivos electromecánicos; sin embargo, la variada naturaleza de los instrumentos simulados, y de los diales con los que representan la información, hace del diseño de este interfaz común de control una tarea no trivial.

Figura 1.1: Cabina de un Douglas DC3. Fuente: " https://es.wikipedia.org/wiki/Douglas_DC-3" (15 de agosto de 2020).

1.2 Objetivos

El objetivo principal de este TFG es la construcción de un programa que actúe de interfaz con el simulador de vuelo y con el hardware que se use para la replicación de instrumentos, de manera que proporcione una herramienta de base sobre la que sea sencillo y rápido el diseño e implementación de réplicas de cualquier instrumento de cabina.

Para ello se deberá implementar un software (de aquí en adelante, llamado plugin) que se comunique con el simulador de vuelo para realizar la extracción automática y sostenida de los valores más actualizados de los parámetros representados por la instrumentación.
Se necesitará también implementar un programa con interfaz gráfica mediante el cual el usuario seleccionará qué instrumentos va a replicar; este programa tratará los datos proporcionados por el simulador de acuerdo a esta selección y tendrá la capacidad de proporcionar al sistema hardware que representará el instrumento en físico la configuración necesaria para poder hacer uso de estos datos procesados.
Finalmente, se deberá implementar para el hardware que controla los dispositivos electromecánicos de la réplica, un protocolo de comunicación estandarizado y un algoritmo que traduzca, también de forma estandarizada, los datos recibidos en movimientos de estos dispositivos electromecánicos.

1.3 Estado del arte

La construcción de simuladores de vuelo caseros es una actividad muy extendida entre los aficionados al pilotaje de aeronaves, siendo varios los proyectos software y hardware dedicados a facilitar la construcción de estos simuladores, e innumerables los proyectos caseros para montar cabinas lo más realistas posibles.
Es posible adquirir elementos de control tales como palancas y pedales para usar como hardware de entrada al software simulador y controlar con ellos la simulación, pero existen pocas opciones comerciales para instrumentos de cabina o luces indicadoras. A esto hay que añadir que esas opciones comerciales no son reproducciones fieles de instrumentos de aviones reales, pues al ser elementos genéricos están orientados al diseño *"gamer"*, y no están basados en ninguna aeronave en concreto.

Figura 1.2: Instrumentos para simulador TRC Basic Six Starter Pack.

Fuente: " https://www.simkits.com/product/trc-basic-six-starter-pack/*" (22 de septiembre de 2020).*

Es por este motivo que, para ayudar a que las construcciones amateurs o semiprofesionales tengan acabados más fieles, es usual que se construyan artesanalmente. Estas cabinas hechas a mano usan materiales comunes (como placas programables Arduino con motores paso a paso, servos, luces led y potenciómetros) para emular los distintos elementos de una cabina real.

A continuación, se presentan algunos de los proyectos software más conocidos para dar soporte a la construcción de estas cabinas:

1.3.1 LINK2FS

Es un proyecto comenzado en 2007 para la extracción de datos de Microsoft Flight Simulator 2004 y más tarde para Microsoft Flight Simulator 9 y X. Su funcionamiento se basa en un plugin para el simulador de vuelo que hace de interfaz con la aplicación, la cual procesa los datos y las ofrece a Arduino mediante lógica TTL pudiendo configurar el identificador que se le va a dar a cada dato para el envío hacia la placa programable Arduino.

También ofrece una interfaz de entrada de datos para el simulador de manera que facilita el control de la aeronave simulada con elementos de entrada para Arduino.
Este proyecto presenta el problema de que no ofrece un algoritmo para Arduino, siendo el usuario quien tiene que implementar el programa para la placa de acuerdo a la documentación disponible sobre la interfaz de salida de datos. [1]

1.3.2 MobiFlight Modules + FSUIPC

Es de los proyectos más interesantes existentes en la red ya que soporta distintos simuladores de vuelos como X-Plane, Prepar3D o Microsoft Flight Simulator gracias al plugin FSUIPC. Obtiene los datos de este plugin y ofrece una interfaz gráfica para configurar los instrumentos.
Actualmente es completamente funcional con instrumentos de salida que usen luces LED y pantallas de 7 segmentos LED. Como elementos de entrada soporta Switches y Encoders.
En las últimas versiones se ha adentrado en la simulación de instrumentación analógica mediante motores servo y paso a paso, y aunque son funciones que están aún en desarrollo ofrece una gran flexibilidad. El software es gratuito y se mantiene a base de donativos, tiene una gran comunidad detrás y soporta varias placas de la familia Arduino.
Como defecto principal no ofrece scripts para las placas programables por lo que el funcionamiento es poco flexible en cuanto a la interfaz del programa principal. Tampoco ofrece una guía de funcionamiento completa de conexión, limitándose a algunos instrumentos de salida con luces LED y pantallas de 7 segmentos. [2]

1.3.3 SimVim Cockpit

Es el más usado cuando se quiere montar cockpits caseros debido a su sencillez de montaje físico y configuración software. Ofrece un gran catálogo de instrumentos tanto digitales como analógicos extraídos del simulador de vuelto mediante un plugin propio para X-Plane.
Su funcionamiento se basa en renderizar cada instrumento independientemente sobre una pantalla, por lo que para obtener un cockpit realista solo hay que construir una carcasa con materiales baratos, colocar esta carcasa encima de una o varias pantallas y mediante SimVim ubicar sobre las pantallas los instrumentos digitales o analógicos, de manera que queden

centrados en los huecos de la carcasa. El resultado es un panel de instrumentos bastante realista, que se asemeja al avión que se quiere simular.

Este proyecto tiene la gran ventaja de que no requiere conocimientos de electrónica ni placas programables a las que conectarle elementos móviles ya que los instrumentos son simplemente renderizaciones graficas. Tampoco es necesario mucho tiempo de montaje e instalación ya que solo hay que colocar la pantalla, arrastrar los instrumentos a la posición donde deben ir y el programa guarda estas posiciones para futuras sesiones, de manera que cuando se quiera usar el simulador solo haya que iniciar SimVim y el simulador y todo estará listo para volar. [3]

Figura 1.3: Pantalla ejecutando SimVim y resultado con carcasa de madera.

Fuente: " https://simvim.com/simcom/index.php/pit3 *" (15 de agosto de 2020).*

El único inconveniente de SimVim es, por supuesto, el aspecto "plano" de la cabina resultante, que le da sensación de videojuego, y que le resta considerablemente realismo cuando se desea obtener una réplica fidedigna de la cabina de un avión real.

1.4 Metodología, Herramientas y Tecnologías usadas

Para la realización de este proyecto se han usado metodologías ágiles para el proceso de desarrollo software, y en particular, se ha usado *Desarrollo iterativo y creciente* [4], dado que esta metodología tiene grandes ventajas para un desarrollo de este tipo, en el que en un principio se tiene una idea de lo que se quiere desarrollar, pero no se sabe con exactitud a qué nos enfrentamos hasta que se realiza la fase de análisis e investigación, y en donde, además, tras empezar el proyecto, se debe seguir investigando durante todo su desarrollo para solventar los problemas que van apareciendo.

El procedimiento del desarrollo iterativo y creciente soluciona las debilidades del tradicional modelo en cascada, puesto que ofrece flexibilidad al permitir que el proyecto cambie conforme avanza el desarrollo, y se aclaran las necesidades del cliente. Para ello, se divide un conjunto de tareas en pequeñas etapas que se repiten (de aquí el nombre iterativo).

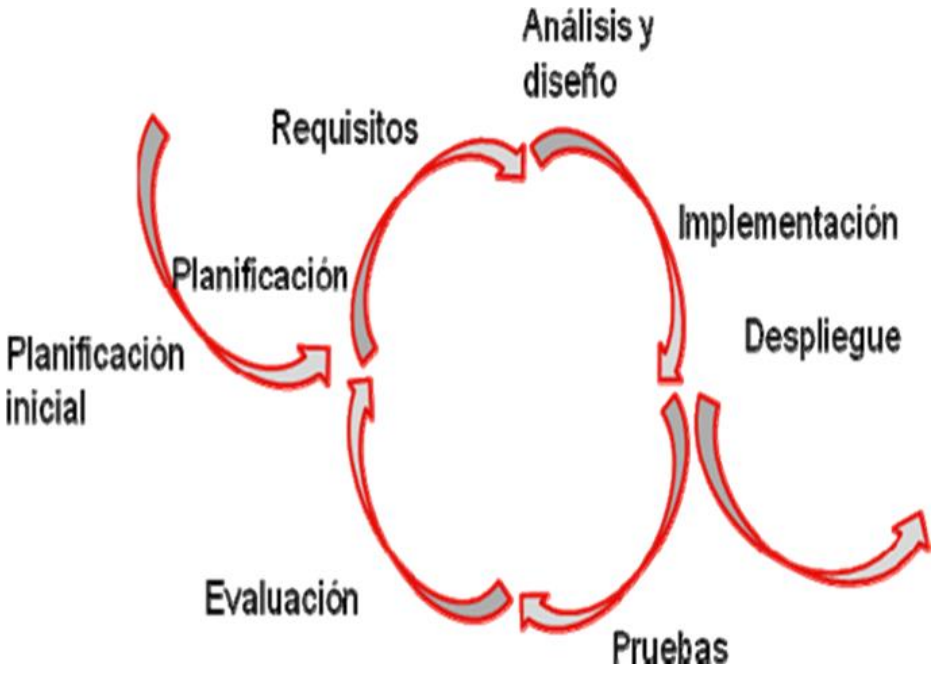

Figura 1.4: Ciclo de trabajo del desarrollo iterativo y creciente.
Fuente: " https://app.emaze.com/@AZRQTRWI#1*" (16 de agosto de 2020).*

Se descompone el desarrollo del sistema en pequeñas partes o módulos que se irán implementando en cada etapa, de manera que tras la finalización de cada etapa se tiene una

parte del sistema que funciona de manera independiente, o dependiente de los desarrollos de iteraciones anteriores. Las etapas del desarrollo son:

- **Etapa de Planificación y requisitos:** Se fija el objetivo que debe cumplirse en esa iteración y los requisitos que se van a implementar y que son necesarios para que la iteración culmine satisfactoriamente.
- **Etapa de Análisis e Implementación:** Una vez completa la planificación de la etapa se realiza la tarea, teniendo en cuenta los posibles riesgos que puedan surgir.
- **Etapa de Pruebas:** Se realizan las pruebas que correspondan para el desarrollo que se ha llevado a cabo en la iteración, pudiendo ser de tipo funcional, estructural, de usabilidad, etc.
- **Etapa de Evaluación:** Se comprueba que el resultado es acorde a los requisitos que se establecieron en la primera etapa de la iteración, y en el caso de que dependa de tareas que se han realizado en iteraciones anteriores se comprueba también su correcto funcionamiento e integración.

Para el desarrollo del proyecto se han usado las siguientes herramientas:

- **Trello**: Herramienta indispensable para la gestión temporal y de tareas del proyecto [5]. El modelo de tarjetas con tableros permite realizar una planificación del proyecto y anotar cualquier incidencia que pueda provocar un cambio en la planificación inicial. En este caso no se ha hecho uso de sus capacidades colaborativas en equipo, pero al ser multidispositivo permite también la gestión de un equipo entero en el proyecto.
- **Microsoft Visual Studio 2015:** Software IDE usado para el desarrollo del plugin para X-Plane 10. La integración del plugin con las librerías y elementos del simulador se ha hecho con el SDK de X-Plane 10 en su versión 3.0.
- **Microsoft Visual Studio 2019:** Software IDE usado para el desarrollo del programa principal que recibe los datos del plugin del simulador y los envía a la placa programable Arduino usando el protocolo de comunicación implementado para tal efecto.
- **Arduino IDE versión 1.8.13:** Software IDE para el desarrollo de algoritmos para placas programables Arduino. Se ha usado para la implementar el código que correrá sobre

el lado hardware del sistema, y que recibe los datos del programa principal a través del protocolo de comunicación diseñado.

- **Librería AccelStepper:** Librería Open Source dedicada a controlar motores paso a paso. Ofrece una interfaz orientada a objetos para los motores que permite controlar parámetros como el avance libre, posición relativa, avance de pasos independientes de la posición, aceleración, velocidad, etc. [6]

Respecto a los lenguajes de programación empleados se hace uso de tres, uno para cada elemento del sistema:

- Para el desarrollo del plugin de X-Plane 10 se ha usado el lenguaje de programación C. La interacción con el software del simulador de vuelo se consigue mediante el SDK de X-Plane [7] y para las comunicaciones por TCP/IP se usa la interfaz Windows Sockets API (WSA o Winsock). La elección de este lenguaje no es opcional porque (aunque hay una versión del SDK para trabajar con Python) C es el lenguaje soportado oficialmente para las extensiones del simulador.
- Para el desarrollo del programa principal se ha usado C#, ya que es un lenguaje orientado a objetos que ofrece un amplio catálogo de librerías que facilitan el trabajo, incluyendo comunicación con sockets, elementos gráficos y comunicación por TTL. Además, forma parte de .NET por lo que ofrece una perfecta integración con la plataforma para la que se ha desarrollado el sistema (Windows).
- Para el desarrollo del algoritmo para la placa programable Arduino se ha usado el lenguaje C++ para Arduino. El lenguaje para estos dispositivos es básicamente un subconjunto de funciones de C y C++; aunque hay funciones propias, en la fase de compilación se traducen al equivalente en C++ y se compilan con un compilador de dicho lenguaje. La elección de este lenguaje es por motivos de eficiencia: aunque es posible usar otros lenguajes que no están contemplados en la documentación oficial, C++ ofrece un rendimiento superior a todos los demás, lo cual es esencial en este tipo de dispositivos que tienen recursos muy limitados.

1.5 Estructura del documento

Esta memoria se estructura en cinco capítulos, en los que se detalla el proyecto, los resultados de la aplicación de la metodología de desarrollo incremental e iterativa y también se justifican las decisiones de diseño que se han tomado para cada uno de los elementos que conforman el sistema.

El primer capítulo (en el que ahora mismo nos encontramos) contiene una introducción al proyecto, así como la motivación que ha dado lugar a este trabajo, y las tecnologías y herramientas usadas.

El segundo capítulo realiza un análisis y especificación formal del proyecto aplicando metodologías de ingeniería del software. El análisis comienza con una descripción del sistema, y los requisitos que debe tener el sistema, tras lo que se realiza una especificación teniendo en cuenta las distintas posibilidades de uso que tendrá el sistema y su funcionamiento.

El tercer capítulo se centra en el diseño del proyecto, como será su arquitectura y cuál es la interacción que tienen los distintos componentes del sistema, especificando como se integran con las plataformas sobre las que se desarrollan y su funcionamiento en detalle.

El cuarto capítulo está dedicado a la implementación, y en él se reflejan las distintas etapas del método de desarrollo incremental e iterativo. y dentro de cada una como se ha implementado cada módulo y su funcionamiento exacto.

En el último capítulo se discuten los resultados del proyecto y su relación con las tecnologías de impresión 3D, los errores que han ido apareciendo a lo largo del desarrollo y como se han solventado, se resume el trabajo en una conclusión y se proponen posibles mejoras que pueden dar lugar a otras líneas para futuros trabajos de fin de grado.

2

Análisis y Especificación

El análisis es una tarea fundamental en el proceso de realización de un proyecto software. Se basa en la identificación de las bases del proyecto, extracción de los requisitos que debe cumplir el producto que se va a desarrollar, y delimitación de qué componentes y qué usuarios estarán involucrados en el desarrollo y uso del producto a lo largo de su vida.
Es una etapa de especial importancia, porque cometer un error en esta fase es crítico, ya que puede retrasar las fases siguientes del proyecto o incluso puede que se tenga que rehacer o desechar parte del trabajo realizado (si el error cometido durante el análisis o especificación es especialmente importante).

2.2 Análisis

El objetivo de este proyecto es crear una interfaz funcional que extraiga los parámetros de toda la instrumentación del simulador de vuelo X-Plane 10 (De aquí en adelante "el simulador de vuelo"), y que mediante un programa el usuario pueda decidir qué instrumento quiere representar usando una placa programable Arduino, eligiéndolo de entre los ofrecidos en el catálogo de los instrumentos que se exportan desde el simulador de vuelo. Por último, el programa principal deberá mediante un interfaz de comunicación enviar los parámetros de configuración del instrumento seleccionado a la placa programable, que será la encargada de

accionar los elementos móviles acorde a la configuración proporcionada por el programa principal. Se observa por lo tanto que hay tres elementos claramente diferenciados:

1) **Plugin o extensión del simulador de vuelo:** Este elemento realiza una única tarea, de forma continua y repetitiva: recoger los parámetros de instrumentación de la aeronave simulada por el simulador de vuelo y exportarlos mediante una interfaz de comunicación fuera del simulador de vuelo, haciéndolos disponibles a cualquier aplicación que sea compatible con dicha interfaz.

   Debido a que el simulador de vuelo se ejecuta sobre su propio espacio de memoria no es posible acceder desde un proceso externo a dichos parámetros; por lo tanto, es necesario que este plugin sea código que pueda ser ejecutado por el propio simulador de vuelo como modulo integrado en su *runtime*, y que opere en paralelo al simulador.
2) **Programa principal:** Este elemento es el más importante del sistema y el que realiza la mayoría de funciones:
   - Debe ser compatible con la interfaz de comunicaciones del plugin que trabaja con el simulador de vuelo, de manera que reciba y procese los datos y parámetros que el plugin envía, y que deben ser almacenados en una estructura diseñada para tal fin. La realización de esta tarea es cíclica y constante.
   - Debe ofrecer una interfaz gráfica sencilla y clara al usuario que permita el uso de las funcionalidades del programa, tales como la visualización de los valores que se recibe del plugin, la selección de un instrumento analógico o la libre configuración de todos los parámetros necesarios para representar un instrumento analógico con un motor paso a paso, pudiendo con estos parámetros establecer el comportamiento que tendrá el instrumento representado a partir de los datos que proporciona el plugin de X-Plane. También debe ofrecer la posibilidad de seleccionar a qué placa programable se conectará para realizar la representación del instrumento, así como indicar la configuración del cableado del interfaz físico de la placa.
   - Debe implementar un protocolo de comunicación con la placa programable para el envío constante de datos para la representación. Este protocolo debe ser tolerante a fallos.

3) **Algoritmo para placa programable Arduino:** Este elemento debe implementar el protocolo de comunicación con el programa principal. En particular, se va a trabajar con motores paso a paso por lo que el algoritmo tiene que estar preparado para recibir e interpretar desde el programa principal los parámetros de configuración que determinarán el comportamiento del motor paso a paso.

   Respecto a la representación del instrumento analógico, el algoritmo deberá solicitar constantemente a la aplicación, e interpretar los datos que describan cual debe ser la posición de los diales del instrumento en ese momento, tarea que debe ejecutarse de forma continua mientras dure la sesión de simulación. El algoritmo debe ser tolerante a los posibles fallos contemplados por el protocolo de comunicación con el programa principal.

2.2.1 Requisitos funcionales

Los requisitos funciones describen las características y comportamientos que un producto software debe cumplir; se puede hablar también de condiciones o capacidades que debe tener un sistema o componente de un sistema para satisfacer las condiciones impuestas por un contrato, especificación u otros documentos impuestos formalmente.

En nuestro TFG los requisitos funcionales que debe cumplir cada componente del sistema son:

Plugin o extensión del simulador de vuelo:

RF.1 Cargar el componente con el simulador de vuelo.

RF.2 Iniciar la interfaz de comunicación al inicio de una simulación.

RF.3 Enviar los parámetros de instrumentación de la aeronave constantemente.

RF.4 Exportar los parámetros de manera que varios consumidores puedan consumir estos datos simultáneamente.

RF.5 Exportar haciendo uso de la interfaz los parámetros de instrumentación especificados en el Apéndice B.

RF.6 El funcionamiento del plugin debe ser independiente al resto del sistema y solo debe depender del software simulador de vuelo.

RF.7 La comunicación entre el plugin de X-Plane y el programa principal se realizará mediante el protocolo TCP/IP.

Programa principal:

RF.8 Recibir parámetros del simulador a través de una interfaz de comunicación.

RF.9 Almacenar los parámetros recibidos en una estructura de datos personalizada.

RF.10 Consultar en tiempo real los parámetros que se están recibiendo desde el simulador de vuelo.

RF.11 Seleccionar o configurar manualmente el sistema de representación de un instrumento.

RF.12 Seleccionar una placa programable Arduino de entre todas las conectadas al ordenador.

RF.13 Implementar un protocolo de comunicación con la placa programable Arduino para el envío de configuración y parámetros.

RF.14 Detectar la desconexión de la placa programable Arduino.

RF.15 Leer y escribir concurrentemente los parámetros de instrumentación recibidos por el simulador de vuelo y enviados a la placa programable Arduino.

RF.16 Refrescar la lista de placas programables Arduino conectadas.

RF.17 Detectar el cese de envío de parámetros por parte del simulador.

RF.18 Tratar errores de comunicación con la placa programable Arduino.

RF.19 Detener el envío de parámetros a la placa programable Arduino manualmente.

RF.20 Mostrar esquema de conexión de cableado al usuario para el correcto funcionamiento de la representación de instrumentos.

RF.21 Iniciar el envío de parámetros de configuración con la placa programable Arduino.

RF.22 Confirmar la correcta configuración de la placa programable Arduino acorde a los parámetros enviados anteriormente.

RF.23 Recibir peticiones de envío de parámetros de instrumentación por parte de la placa programable Arduino.

RF.24 Enviar parámetros de instrumentación a la placa programable Arduino tras la recepción de una petición.

RF.25 La comunicación entre el programa principal y la placa programable Arduino se realizará mediante la interfaz USB.

Algoritmo para placa programable Arduino:

RF.26 Implementar protocolo de comunicación con el programa principal para la recepción de configuración y parámetros.

RF.27 Detectar fallos en la comunicación de la recepción de configuración inicial.

RF.28 Configurar posición inicial de los motores paso a paso mediante detección de esta con elementos físicos.

RF.29 Configurar parámetros tope y comportamiento de los motores paso a paso a partir de la configuración recibida.

RF.30 Recibir la configuración inicial.

RF.31 Establecer un comportamiento en funcion de la configuración inicial recibida.

RF.32 Mandar un mensaje al programa principal para pedir los parámetros de instrumentación más recientes.

RF.33 Recibir los parámetros de instrumentación más recientes que haya recibido el programa principal desde el simulador de vuelo.

RF.34 Mover los motores paso a paso configurados hacia la posición que deben tener según el último parámetro de instrumentación recibido.

RF.35 Detectar cuando se ha completado la operación de movimiento de los motores paso a paso configurados.

RF.36 Detectar perdida de peticiones de envío de parámetros de instrumentación hacia el programa principal

RF.37 Detectar el funcionamiento incorrecto del programa principal

RF.38 Detectar la desconexión del programa principal del protocolo de comunicación por error o finalización de la simulación.

RF.39 Resetear la placa en caso de pérdida total de comunicación o finalización de la sesión de simulación.

RF.40 Esperar nueva configuración inicial tras el reseteo de la placa programable Arduino.

2.2.2 Requisitos no funcionales

Los requisitos no funcionales son restricciones que debe cumplir el sistema en cuanto a su modo de operación o comportamiento. A diferencia de los funcionales no se centra en las funciones que realiza, sino en que características debe tener la operación para considerar

que la funcion que realiza se hace de manera correcta. Estos requisitos suelen estar relacionados con parámetros como el rendimiento, la disponibilidad o accesibilidad.
Los requisitos funcionales que debe cumplir el sistema son:

RnF.1 La interfaz del programa principal ha de ser clara y sencilla de usar

RnF.2 El envío de parámetros por parte del simulador de vuelo se debe hacer con una frecuencia de 10 veces por segundo.

RnF.3 El número de instancias del programa principal que pueden funcionar simultáneamente no está limitado.

RnF.4 Los parámetros de instrumentación recibidos por el programa principal se dividen en categorías.

RnF.5 Una instancia del programa principal solo puede actuar como productor de parámetros de instrumentación de una categoría.

RnF.6 Una placa programable Arduino solo se puede configurar para la recepción de parámetros de instrumentación de una categoría.

RnF.7 Cada instancia del programa principal funciona independientemente pudiendo estas actuar como productores de parámetros de instrumentación de distintas categorías.

RnF.8 Todas las instancias del programa principal reciben los mismos parámetros de instrumentación desde el simulador de vuelo y lo hacen a al mismo tiempo.

RnF.9 Cada instancia del programa principal puede configurar y actuar sobre una sola placa programable Arduino.

RnF.10 El número máximo de motores paso a paso conectados a una placa Arduino es de tres.

RnF.11 La petición y recepción de parámetros de instrumentación de vuelo se hará en modo "pull" o bajo demanda según la capacidad de procesamiento de la placa programable Arduino.

RnF.12 La lectura y escritura concurrente de parámetros de instrumentación se realizará con un sistema de doble buffer.

2.3 Especificación

La fase de especificación de un proyecto software describe cómo será la interacción del usuario con el software, tanto de sus funciones como en la interacción con su interfaz. Hacer una correcta especificación es útil de cara a la implementación, ya que una especificación simple y completa deja el camino hecho al desarrollo sin dejar dudas acerca del funcionamiento y diseño.

Aquí se recoge la especificación del sistema que se va a desarrollar de manera general mediante el modelo de dominio, de manera textual, mediante casos de uso detallados, y mediante gráficos haciendo uso de diagramas de flujo. Se especifica mediante diagramas de flujo también la interacción entre los distintos componentes del sistema.

2.3.1 Modelo de dominio

El modelo de dominio abstrae el sistema. Es una representación de los componentes y las relaciones entre estos en forma de diagrama de clases. Se deriva del análisis realizado previamente y ayuda a comprender los elementos del proyecto de una manera sencilla.

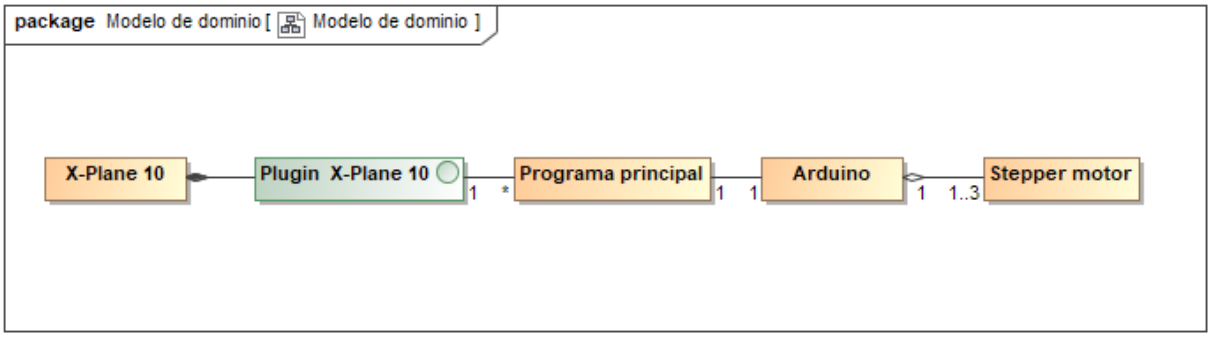

Figura 2.1: Modelo de dominio del sistema

En el diagrama de clases se puede ver como el plugin de X-Plane 10 es una interfaz que vamos a construir sobre el simulador para proporcionar parámetros de instrumentación fuera de este. Nótese que para que el plugin se ejecute es necesaria la existencia y ejecución del simulador debido a la dependencia que tienen.

Puede haber muchas instancias del programa principal consumiendo los datos ofrecidos por la interfaz del simulador, y conectado con cada instancia del programa principal encontramos una placa programable Arduino.

Por último, a cada placa Arduino encontramos conectados de 1 a 3 motores paso a paso que realizan los movimientos para representar el instrumento analógico. La dependencia entre estos dos elementos no es tan fuerte, pero no tiene sentido introducir en el sistema el motor paso a paso si no es conectado y controlado por una placa programable Arduino.

2.3.2 Casos de uso detallados

Los casos de uso listan secuencias de acciones que describen la interacción entre un actor y el sistema. No es necesario que el actor sea el usuario de la aplicación, ya que los casos de uso pueden describir interacción entre dos componentes del sistema o la interacción del sistema con un componente externo al sistema. Habitualmente se seguirá el flujo descrito en el escenario principal, pero puede darse el caso de que algo falle y deba haber un plan de recuperación o contingencia.

Esto se describe en los escenarios alternativos. Son la secuencia de acciones que se deberían seguir naturalmente en el caso de que un paso del escenario principal no se dé por circunstancias de error o mal uso del usuario.

Esta herramienta es muy útil para especificar el funcionamiento de una aplicación, así como el comportamiento del sistema y el usuario en casos de error. Además, es útil para identificar escenarios que pueden llevar a flujos de acción no contemplados inicialmente.

Titulo	Iniciar la aplicación
Descripción	Un usuario inicia la aplicación principal
Precondición	El simulador de vuelo está iniciado Hay una sesión de simulación en marcha El plugin del simulador se ha cargado correctamente
Postcondición	El programa principal se ha iniciado y se muestra la pantalla principal con todos los botones habilitados
Escenario principal	
1. El usuario inicia la aplicación 2. El programa principal se inicia	
Escenario alternativo	
2.a El sistema muestra un aviso de error de comunicación con la interfaz del simulador 1. El usuario reinicia el simulador de vuelo 2. El usuario inicia la aplicación de nuevo.	

Titulo	Iniciar representación de instrumentos relacionados con la categoría "Motores".
Descripción	Un usuario inicia una simulación de uno o varios instrumentos de la categoría motores.
Precondición	El simulador de vuelo está iniciado Hay una sesión de simulación en marcha El plugin del simulador se ha cargado correctamente El programa principal ha iniciado correctamente Encontrarse en la vista de la pantalla principal
Postcondición	El botón de parar simulación se ha activado y el motor paso a paso o motores configurados se han iniciado correctamente y representan los instrumentos seleccionados.

Escenario principal

1. El usuario pulsa el botón "Motores".
2. El sistema muestra los instrumentos disponibles en esta categoría.
3. El usuario selecciona los instrumentos que quiere simular y pulsa siguiente.
4. El sistema muestra una ventana por instrumento seleccionado con los parámetros prestablecidos para el instrumento seleccionado.
5. El usuario selecciona el tipo de instrumento a simular de la lista, modifica la configuración de uno existente o introduce unos parámetros de configuración manualmente.
6. El sistema muestra la lista de puertos a los que hay placas Arduino conectadas
7. El usuario selecciona una de las placas Arduino para representar el instrumento.
8. El sistema pregunta si se necesita calibración de los instrumentos.
9. El usuario indica si se necesita calibración y el método para realizar la calibración de los instrumentos.
10. El sistema muestra el diagrama de conexión de cableado que debe conectar el usuario para que funcione el instrumento.
11. El usuario confirma que ha conectado los cables de esa manera.
12. El sistema inicia el protocolo de comunicación con la placa y comienza la simulación.

Escenario alternativo

6.a La lista de placas conectadas está vacía.

1. El usuario pulsa el botón refrescar lista
2. El sistema comprueba que se ha conectado al menos una placa y se continua en el paso 7.

8.a El sistema muestra un aviso de que la placa seleccionada no está disponible o no funciona correctamente

1. El usuario pulsa aceptar
2. El sistema vuelve al paso 6 mostrando la lista de puertos en los que hay placas conectadas

10.a El sistema muestra un aviso de que no ha podido aplicar la configuración sobre la placa Arduino.

1. El usuario pulsa el botón de RESET de la placa Arduino o espera 9 segundos a que se reinicie y pulsa aceptar.
2. El sistema vuelve al paso 6 mostrando la lista de puertos en los que hay placas conectadas

Titulo	Iniciar representación de instrumentos relacionados con la categoría "Aviónica".
Descripción	Un usuario inicia una simulación de uno o varios instrumentos de la categoría aviónica.
Precondición	El simulador de vuelo está iniciado Hay una sesión de simulación en marcha El plugin del simulador se ha cargado correctamente El programa principal ha iniciado correctamente Encontrarse en la vista de la pantalla principal
Postcondición	El botón de parar simulación se ha activado y el motor paso a paso o motores configurados se han iniciado correctamente y representan los instrumentos seleccionados.

Escenario principal

1. El usuario pulsa el botón "Aviónica".
2. El sistema muestra los instrumentos disponibles en esta categoría.
3. El usuario selecciona los instrumentos que quiere simular y pulsa siguiente.
4. El sistema muestra una ventana por instrumento seleccionado con los parámetros prestablecidos para el instrumento seleccionado.
5. El usuario selecciona el tipo de instrumento a simular de la lista, modifica la configuración de uno existente o introduce unos parámetros de configuración manualmente.
6. El sistema muestra la lista de puertos a los que hay placas Arduino conectadas
7. El usuario selecciona una de las placas Arduino para representar el instrumento.
8. El sistema pregunta si se necesita calibración de los instrumentos.
9. El usuario indica si se necesita calibración y el método para realizar la calibración de los instrumentos.
10. El sistema muestra el diagrama de conexión de cableado que debe conectar el usuario para que funcione el instrumento.
11. El usuario confirma que ha conectado los cables de esa manera.
12. El sistema inicia el protocolo de comunicación con la placa y comienza la simulación.

Escenario alternativo

6.a La lista de placas conectadas está vacía.

1. El usuario pulsa el botón refrescar lista
2. El sistema comprueba que se ha conectado al menos una placa y se continua en el paso 7.

8.a El sistema muestra un aviso de que la placa seleccionada no está disponible o no funciona correctamente

1. El usuario pulsa aceptar
2. El sistema vuelve al paso 6 mostrando la lista de puertos en los que hay placas conectadas

10.a El sistema muestra un aviso de que no ha podido aplicar la configuración sobre la placa Arduino.

1. El usuario pulsa el botón de RESET de la placa Arduino o espera X segundos a que se reinicie y pulsa aceptar.
2. El sistema vuelve al paso 6 mostrando la lista de puertos en los que hay placas conectadas

Titulo	Iniciar representación de instrumentos relacionados con la categoría "Combustible".
Descripción	Un usuario inicia una simulación de uno o varios instrumentos de la categoría combustible.
Precondición	El simulador de vuelo está iniciado Hay una sesión de simulación en marcha El plugin del simulador se ha cargado correctamente El programa principal ha iniciado correctamente Encontrarse en la vista de la pantalla principal
Postcondición	El botón de parar simulación se ha activado y el motor paso a paso o motores configurados se han iniciado correctamente y representan los instrumentos seleccionados.

Escenario principal

1. El usuario pulsa el botón "Combustible".
2. El sistema muestra los instrumentos disponibles en esta categoría.
3. El usuario selecciona los instrumentos que quiere simular y pulsa siguiente.
4. El sistema muestra una ventana por instrumento seleccionado con los parámetros prestablecidos para el instrumento seleccionado.
5. El usuario selecciona el tipo de instrumento a simular de la lista, modifica la configuración de uno existente o introduce unos parámetros de configuración manualmente.
6. El sistema muestra la lista de puertos a los que hay placas Arduino conectadas
7. El usuario selecciona una de las placas Arduino para representar el instrumento.
8. El sistema pregunta si se necesita calibración de los instrumentos.
9. El usuario indica si se necesita calibración y el método para realizar la calibración de los instrumentos.
10. El sistema muestra el diagrama de conexión de cableado que debe conectar el usuario para que funcione el instrumento.
11. El usuario confirma que ha conectado los cables de esa manera.
12. El sistema inicia el protocolo de comunicación con la placa y comienza la simulación.

Escenario alternativo

6.a La lista de placas conectadas está vacía.

1. El usuario pulsa el botón refrescar lista
2. El sistema comprueba que se ha conectado al menos una placa y se continua en el paso 7.

8.a El sistema muestra un aviso de que la placa seleccionada no está disponible o no funciona correctamente

1. El usuario pulsa aceptar
2. El sistema vuelve al paso 6 mostrando la lista de puertos en los que hay placas conectadas

10.a El sistema muestra un aviso de que no ha podido aplicar la configuración sobre la placa Arduino.

1. El usuario pulsa el botón de RESET de la placa Arduino o espera X segundos a que se reinicie y pulsa aceptar.
2. El sistema vuelve al paso 6 mostrando la lista de puertos en los que hay placas conectadas.

Titulo	Iniciar representación de instrumentos relacionados con la categoría "Hora y tiempo".
Descripción	Un usuario inicia una simulación de uno o varios instrumentos de la categoría Hora y tiempo.
Precondición	El simulador de vuelo está iniciado Hay una sesión de simulación en marcha El plugin del simulador se ha cargado correctamente El programa principal ha iniciado correctamente Encontrarse en la vista de la pantalla principal
Postcondición	El botón de parar simulación se ha activado y el motor paso a paso o motores configurados se han iniciado correctamente y representan los instrumentos seleccionados.

Escenario principal

1. El usuario pulsa el botón "Hora y tiempo".
2. El sistema muestra los instrumentos disponibles en esta categoría.
3. El usuario selecciona los instrumentos que quiere simular y pulsa siguiente.
4. El sistema muestra una ventana por instrumento seleccionado con los parámetros prestablecidos para el instrumento seleccionado.
5. El usuario selecciona el tipo de instrumento a simular de la lista, modifica la configuración de uno existente o introduce unos parámetros de configuración manualmente.
6. El sistema muestra la lista de puertos a los que hay placas Arduino conectadas
7. El usuario selecciona una de las placas Arduino para representar el instrumento.
8. El sistema pregunta si se necesita calibración de los instrumentos.
9. El usuario indica si se necesita calibración y el método para realizar la calibración de los instrumentos.
10. El sistema muestra el diagrama de conexión de cableado que debe conectar el usuario para que funcione el instrumento.
11. El usuario confirma que ha conectado los cables de esa manera.
12. El sistema inicia el protocolo de comunicación con la placa y comienza la simulación.

Escenario alternativo

6.a La lista de placas conectadas está vacía.

1. El usuario pulsa el botón refrescar lista
2. El sistema comprueba que se ha conectado al menos una placa y se continua en el paso 7.

8.a El sistema muestra un aviso de que la placa seleccionada no está disponible o no funciona correctamente

1. El usuario pulsa aceptar
2. El sistema vuelve al paso 6 mostrando la lista de puertos en los que hay placas conectadas

10.a El sistema muestra un aviso de que no ha podido aplicar la configuración sobre la placa Arduino.

1. El usuario pulsa el botón de RESET de la placa Arduino o espera X segundos a que se reinicie y pulsa aceptar.
2. El sistema vuelve al paso 6 mostrando la lista de puertos en los que hay placas conectadas

Titulo	Enviar la configuración inicial a la placa Arduino.
Descripción	El programa principal envía la configuración iniciar a la placa programable Arduino.
Precondición	El simulador de vuelo está iniciado Hay una sesión de simulación en marcha El plugin del simulador se ha cargado correctamente El programa principal ha iniciado correctamente El usuario ha confirmado que ha conectado los cables de la manera indicada por el programa principal
Postcondición	La placa se ha configurado correctamente y empieza a pedir los parámetros de instrumentación más recientes disponibles.

Escenario principal

1. El programa principal envía la primera trama con la configuración que la placa Arduino debe aplicar.
2. La placa Arduino guarda la trama recibida y la reenvía de vuelta al programa principal para comprobar que es válida.
3. El programa principal comprueba que la trama recibida de vuelta es la misma que le envió y confirma a la placa Arduino que es correcta y debe aplicar la configuración.
4. La placa Arduino aplica la configuración

Escenario alternativo

3.a La conexión se interrumpe

1. El programa principal agota el tiempo de espera de la trama de confirmación e informa al usuario que algo ha ido mal en la configuración
2. El usuario pulsa aceptar
3. El sistema vuelve a mostrar la pantalla de selección de puertos a los que hay conectados una placa Arduino.

4.a La conexión se interrumpe y la trama de confirmación nunca llega a la placa Arduino

1. La placa Arduino agota el tiempo de espera y se reinicia esperando una nueva configuración
2. El sistema ahora el tiempo de espera y detecta que no hay interacción con el Arduino. Informa al usuario de que ha habido un error al aplicar la configuración y muestra al usuario la pantalla con el listado de puertos en los que hay una placa Arduino conectada.

Titulo	Enviar parámetros de instrumentación más recientes
Descripción	La placa Arduino pide los datos de instrumentación del tipo configurado más recientes.
Precondición	El simulador de vuelo está iniciado Hay una sesión de simulación en marcha El plugin del simulador se ha cargado correctamente El programa principal ha iniciado correctamente La placa Arduino se ha configurado correctamente
Postcondición	Se recibe una trama con los parámetros de instrumentación más reciente disponibles en el programa principal.

Escenario principal

1. La placa Arduino envía una trama pidiendo los parámetros de instrumentación más recientes.
2. El sistema recibe la trama, comprueba que es del tipo petición y le devuelve de vuelta los parámetros de instrumentación más recientes de los que dispone.
3. La placa Arduino recibe los datos y los almacena en la estructura auxiliar para ese tipo de parámetros.

Escenario alternativo

2.a La conexión se interrumpe y la trama no se recibe

1. El programa principal agota el tiempo de espera y detecta que no hay interacción con la placa Arduino. Informa al usuario de que hay un error de comunicación con la placa Arduino o la placa se ha desconectado.
2. La placa Arduino agota su tiempo de espera y se reinicia esperando una configuración iniciar nueva.

3.a La conexión se interrumpe y la trama no se recibe

1. La placa Arduino agota su tiempo de espera y se reinicia esperando una configuración iniciar nueva.
2. El programa principal agota el tiempo de espera y detecta que no hay interacción con la placa Arduino. Informa al usuario de que hay un error de comunicación con la placa Arduino o la placa se ha desconectado.

Titulo	Consultar parámetros de instrumentación en tiempo real
Descripción	Un usuario consulta los parámetros de instrumentación que recibe el programa principal en tiempo real.
Precondición	El simulador de vuelo está iniciado Hay una sesión de simulación en marcha El plugin del simulador se ha cargado correctamente El programa principal ha iniciado correctamente Encontrarse en la vista de la pantalla principal
Postcondición	Se muestra en pantalla los parámetros de instrumentación que se están recibiendo desde el simulador en tiempo real.
Escenario principal	
1. El usuario pulsa el botón "Instrumentos en tiempo real" 2. El sistema muestra una ventana con los parámetros recibidos en tiempo real	
Escenario alternativo	
2.a El sistema muestra un aviso de error de comunicación con la interfaz del simulador 1. El usuario reinicia el simulador de vuelo 2. El usuario reinicia la aplicación	

4.3.3 Diagramas de secuencia

Los diagramas de secuencia muestran como varios componentes interaccionan entre si haciendo visible el intercambio de mensajes que se produce entre ellos a lo largo del tiempo. En este tipo de diagrama se ve también cuando empieza a existir un componente o si la aparición de un componente es fruto de la interacción de otros componentes.

Interacción Programa principal - Interfaz X-Plane 10

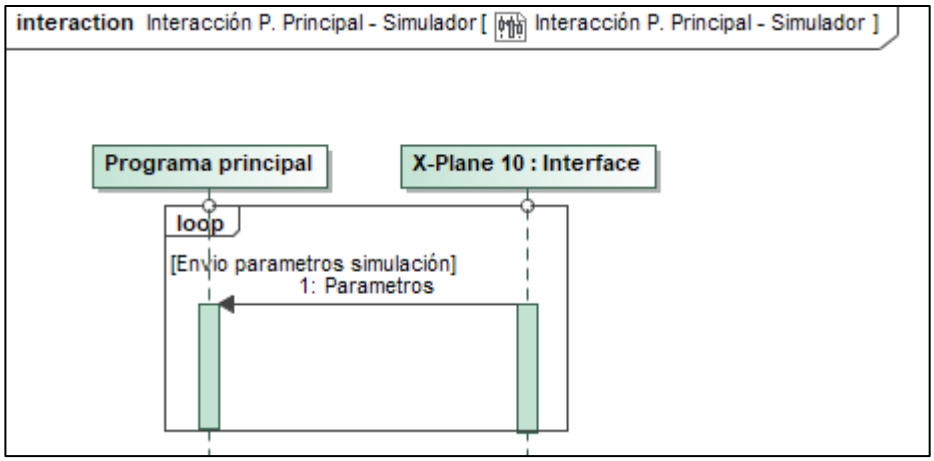

Figura 2.2: Diagrama de secuencia Interacción P. Principal - Interfaz X-Plane 10

La interacción entre estos dos componentes es sencilla: el interfaz que hace el plugin de X-Plane 10 actúa como productor de datos continuamente, mientras que cada instancia del programa principal los consume continuamente de manera pasiva. Como nos interesa disponer siempre del ultimo conjunto de parámetros de la simulación, sin introducir retardos excesivos, no se implementa ningún mecanismo de repetición o reenvío de tramas perdidas: si se diera el caso de que el programa principal no puede consumir alguna trama de datos esta se desecha y se continua con la siguiente. Este mecanismo de descarte de tramas funciona gracias a que la frecuencia de envío es muy alta, por lo que el impacto de una trama perdida es insignificante.

Inicio de una sesión de simulación de instrumentos

Se realiza una abstracción en este diagrama sustituyendo la elección de escoger una categoría concreta para la simulación por una elección genérica "Seleccionar categoría", puesto que el flujo de acciones posteriores a la elección es el mismo independientemente de la categoría elegida.

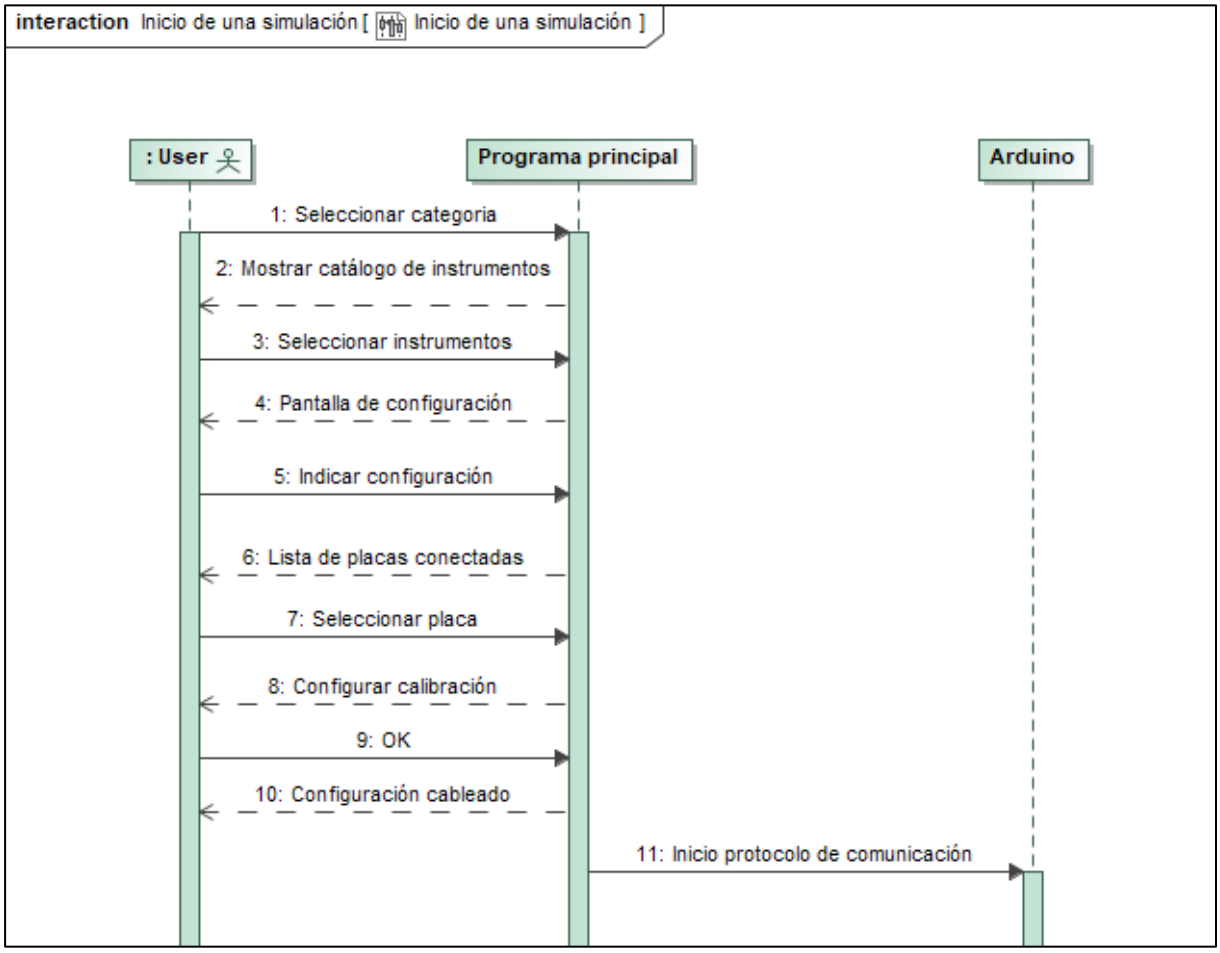

Figura 2.3: Diagrama de secuencia Inicio de una sesión de simulación de instrumentos

Protocolo de comunicación Programa principal - Arduino

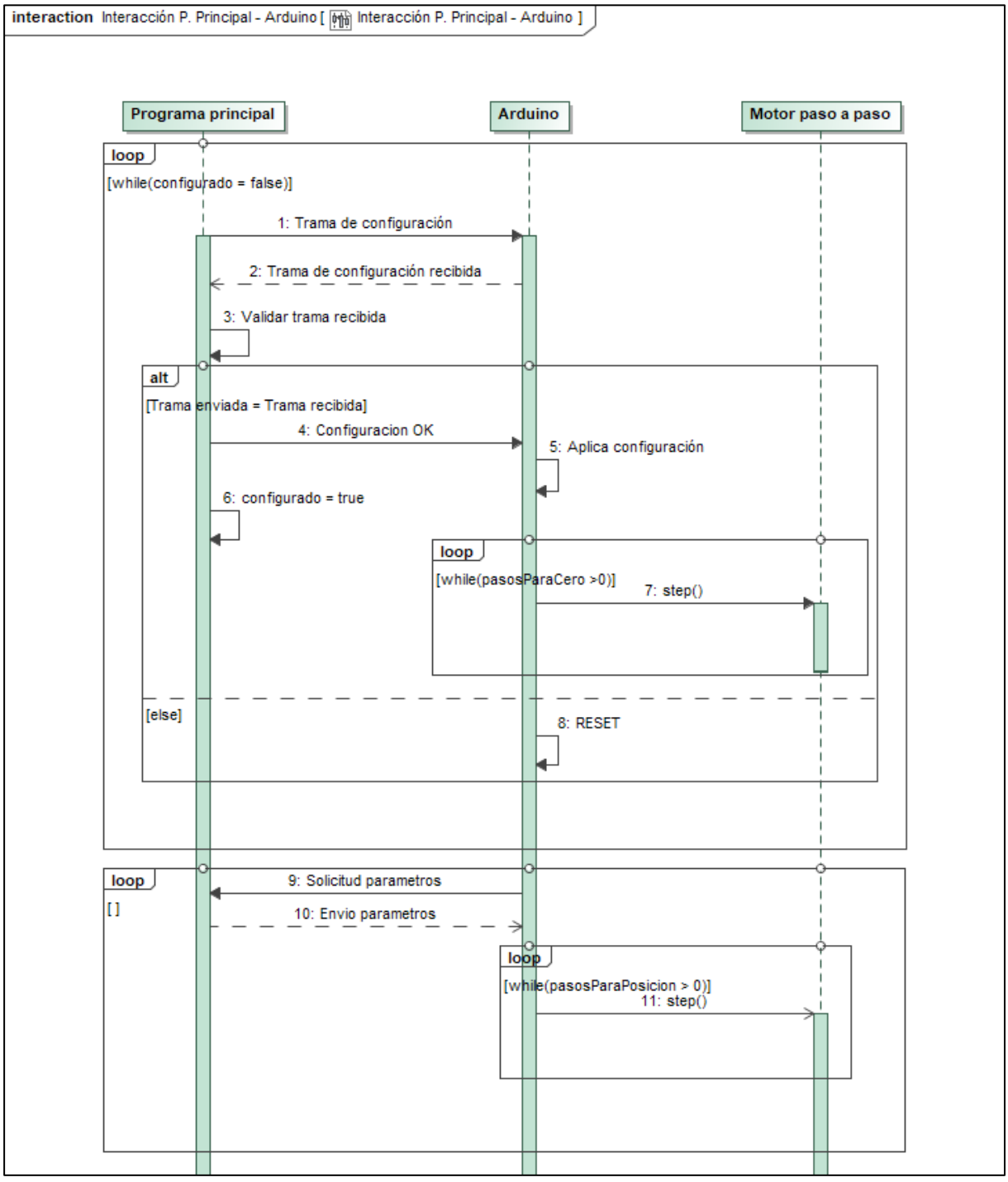

Figura 2.4: Diagrama de secuencia Protocolo de comunicación Programa principal - Arduino

El protocolo de comunicación consiste en el envío de una primera trama con la configuración que se debe aplicar. La placa Arduino devuelve los valores que ha recibido, en el mismo orden que los ha recibido y el programa principal comprueba que se corresponde con lo que le ha enviado. Si la trama que ha recibido es la misma que ha enviado se le indica

a la placa Arduino que todo es correcto y debe aplicar la configuración. De otro modo la placa se resetea pasado un tiempo y se empieza de nuevo.

Si la configuración ha tenido éxito, el programa principal pasa a ser un elemento pasivo, y es la placa Arduino la que irá solicitando periódicamente los datos de instrumentación más actuales disponibles en el programa principal. La actualización se lleva a cabo enviando un mensaje al programa principal, que le responde con los datos más actuales, en base a los cuales el programa en la placa Arduino moverá el motor paso a paso hasta que alcance la posición que debe tener.

3

Diseño

El diseño es la fase que precede a la implementación de todo proyecto software; se apoya en las fases anteriores, y culmina la parte teórica. En la fase de diseño se detallan los componentes del sistema, cómo interaccionan entre ellos, las estructuras de datos y la arquitectura general. El diseño puede tomarse como una guía que deben seguir los programadores a hora de codificar, ya que es la documentación en la que se refleja la especificación del proyecto más cercana al código.

3.1 Arquitectura general del sistema

Como se ha visto en la especificación, hay tres componentes que interactúan entre sí para realizar la simulación mecánica de instrumentos analógicos. El esquema, de manera muy simplificada, queda como muestra la Figura 3.1.

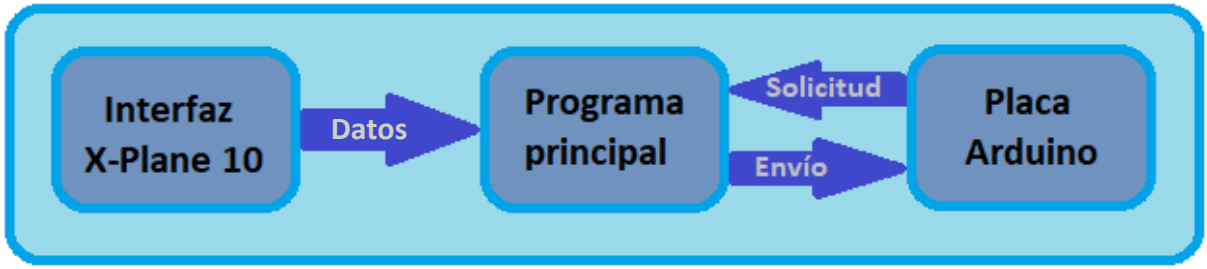

Figura 3.1: Arquitectura simplificada del sistema.

Este modelo conceptual es suficiente cuando se está haciendo un análisis y aún no se saben las características técnicas de cada componente, ni de qué manera interactúan entre ellos cuando están en funcionamiento. Sin embargo, en la fase de análisis y especificación se han descrito las características y los requisitos que debe tener el sistema formalmente, por lo que de cara a diseñar la estructura general del sistema teniendo en cuenta estos requisitos la arquitectura simplificada no es suficiente y debe detallarse.

La interacción entre los componentes puede verse como un sistema productor-consumidor doble, en el que el interfaz de X-Plane 10 es un productor de datos que no hace otra cosa más que enviar datagramas con los últimos parámetros de la simulación, mientras la placa Arduino actúa constantemente como un consumidor de estos parámetros una vez se realiza la configuración.
Es en el programa principal donde encontramos una dualidad que puede llevar a confusión cuando nos referimos a este componente, ya que por un lado consume datos de la interfaz del simulador de vuelo y por otro lado produce datos para la placa Arduino. Por lo tanto, el programa principal debe realizar concurrentemente accesos de lectura y de escritura sobre la estructura que almacena los parámetros de la simulación. En particular, el Requisito no funcional número 12 especifica que la lectura y escritura concurrente se realizara haciendo uso de un sistema de doble buffer. Esto hace más complejo el programa principal y se debe tener en cuenta este dato para el diseño de la arquitectura del sistema.

Para evitar toda confusión y poder hacer una referencia precisa a cada componente del sistema vamos a dividir el programa principal en dos componentes: por un lado, el sistema que consume datos y por otro el que los produce para la placa Arduino. Esto se traduce en que la especificación de la arquitectura debe distinguir dos parejas distintas de entidades productor/consumidor, una para dar servicio el plugin de X-Plane (y, por tanto, esencialmente software), y otra para dar servicio a la placa Arduino (y, por tanto, muy próxima al hardware).

Por lo tanto, para evitar ambigüedades, de ahora en adelante se nombrarán los componentes del sistema como:

- **SW_Productor** *(<<Software productor>>)* para referirnos al plugin de X-Plane que hace de interfaz del simulador y envía los datagramas que contienen los parámetros de la simulación.
- **SW_Consumidor** *(<<Software consumidor>>)* para referirnos a la parte del programa principal que recibe los datos de la interfaz del simulador de vuelo y los almacena en la estructura de datos diseñada para tal fin.
- **HW_Productor** *(<<Hardware productor>>)* para referirnos a la parte del programa principal que proporciona datos a la placa Arduino haciendo uso del protocolo de comunicación diseño para tal efecto.
- **HW_Consumidor** *(<<Hardware consumidor>>)* para referirnos a la placa programable Arduino que consume los datos que le llegan pro la interfaz USB para actuar sobre los elementos mecánicos que realizan la simulación física del instrumento analógico.

Teniendo en cuenta la nomenclatura que se ha establecido y el sistema de doble buffer para el almacenamiento y lectura concurrente de los parámetros de la simulación se obtiene la arquitectura del sistema representada en la Figura 3.2.

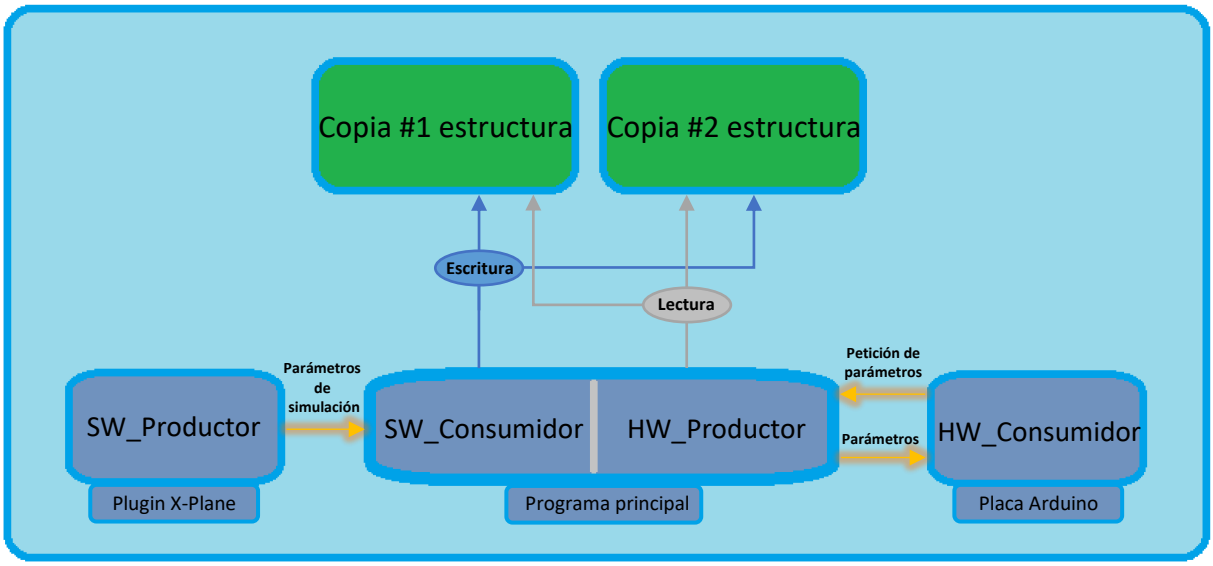

Figura 3.2: Arquitectura del sistema

3.2 Integración del Plugin con X-Plane 10

Para el diseño del plugin que trabajará dentro de X-Plane leyendo los datos de la simulación y enviándolos fuera del juego se debe entender primero como funciona el sistema de extensiones que ofrece este simulador de vuelo.

Figura 3.3: Logo caratula X-Plane 10. Fuente: "https://flyawaysimulation.com/news/4471/" (22 de agosto de 2020).

X-Plane 10 soporta un sistema de plugins o extensiones para todos los sistemas operativos con los que es compatible. La ejecución de los plugins es paralela al simulador de vuelo y necesitan que este esté funcionando para poder funcionar. Un plugin es una librería con código que corre dentro del simulador, haciendo uso del mismo espacio de memoria y tiempos de CPU que el simulador y extiende sus funciones. Son componentes modulares cuyo desarrollo no requiere el código fuente del simulador.

El sistema de plugins en X-Plane (ficheros con extensión xpl) son, esencialmente, DLLs o Bibliotecas de Enlace Dinámico. Estas librerías contienen código y funciones que pueden ser usadas por programas externos en tiempo de ejecución, para lo que, cuando se está preparando la ejecución, el sistema operativo realiza un enlazado entre el ejecutable de X-Plane y la librería dinámica del plugin. (se dice entonces que la librería dinámica exporta sus funciones al programa principal que la usa).

Para realizar el enlazado de X-Plane con los plugin que se le quieran cargar se usa una librería creada por los desarrolladores del simulador, llamada X-Plane Plugin Manager o XPLM.

Cuando se inicia el simulador de vuelo, el ejecutable de X-Plane realiza un enlazado con XPLM y a su vez XPLM se enlaza con todos los plugin que existan en una carpeta dispuesta al efecto. Por lo tanto, el sistema de plugins está formado por 4 elementos: **el simulador X-Plane, la librería XPLM, los plugin desarrollados por terceros y las librerías que usen estos plugins**.

Como toda interacción entre un plugin y el simulador de vuelo requiere de la librería XPLM que está en medio, esta librería exporta un conjunto de funciones prestablecidas para poder realizar acciones sobre el simulador. La empresa desarrolladora de X-Plane pone a disposición de los desarrolladores la librería XPLM, así como su SDK para poder desarrollar plugins y la documentación necesaria.

La estructura básica de un plugin no es más que un archivo con código C que hace uso de las funciones del SDK de X-Plane. Un plugin de X-Plane no puede contener una funcion **main()** como tal o un bucle infinito que realice una tarea constantemente. Esto es debido a que el plugin corre a la vez que el simulador y por lo tanto el desempeño de este tipo de tareas provocaría que no se ejecutaran el resto de tareas necesarias para realizar la simulación. De hecho, X-Plane desactiva un plugin si detecta que su ejecución bloquea al resto de elementos del simulador o si tiene un impacto muy grande sobre el rendimiento de la simulación. Por tanto, el código de un plugin solo puede estar formado por funciones, métodos o rutinas y variables.

El único requisito indispensable para que se reconozca un código como plugin para X-Plane es que implemente las siguientes rutinas:

- **XPluginStart()**

  Esta rutina es llamada cuando se carga el plugin en el simulador. En ella debe de proveerse la información necesaria por el simulador para la ejecución del plugin tales como nombre, signatura y descripción. Además, esta rutina es el punto de entrada a la función que se quiere desempeñar dentro del simulador, por lo que es aquí donde se deben declarar e inicializar los elementos que se vayan a usar, sean estructuras o funciones propias o del marco de desarrollo de X-Plane. El valor de retorno debe ser "true", para indicar al simulador que se ha cargado el plugin correctamente; de otra manera la carga del plugin se aborta.

- **XPluginStop()**

  Esta rutina es llamada cuando se descarga el plugin del simulador. En ella se deben liberar recursos y parar todas las interrupciones y llamadas programadas para el futuro.

- **XPluginEnable()**

  Esta rutina es llamada cuando se ha cargado el plugin y se va a poner en marcha. Dentro de esta rutina también se pueden hacer inicializaciones, pero no es necesario usarla, aunque debe incluirse en el fichero de código fuente. Esta rutina es llamada justo después de que finalice **XPluginStart()** o cuando el usuario habilita el plugin después de haberlo desactivado manualmente.

- **XPluginDisable()**

  Esta rutina es llamada cuando se deshabilita el plugin o previo a la descarga del plugin del simulador. Dentro de esta rutina también se pueden liberar recursos, pero no es necesario usarla, aunque debe incluirse en el fichero de código fuente. Esta rutina es llamada justo antes de que se llame a **XPluginStop()** o cuando el usuario deshabilita el plugin manualmente.

- **XPluginReceiveMessage()**

  Esta rutina es llamada por XPLM cuando se envía un mensaje a un plugin desde el simulador o un mensaje a todos los plugin que haya enlazados con XPLM. Aquí se puede hacer un *handler* o manejador para los mensajes que se reciban y tomar acciones en función de lo que se reciba.

En el plugin desarrollado para este TFG la interfaz implementada en el plugin para enviar los datos fuera del simulador usa el protocolo TCP/IP para realizar la tarea de envío de forma periódica. Siguiendo la estructura diseñada solo es necesario crear un Socket que mediante UDP envíe un datagrama con los datos de la simulación actual. Acorde a los requisitos esta tarea debe tener una frecuencia de al menos 10 veces por segundo, sin usar para ello un bucle continuo, ya que bloquearía los recursos y el simulador deshabilitaría el plugin; es necesario entonces buscar un mecanismo que permita activar la tarea a intervalos regulares y con la frecuencia requerida.

El funcionamiento de X-Plane se basa también en la ejecución de tareas periódicas, pues el dibujado de los gráficos del simulador se hace mediante una rutina interna que se llama periódicamente; entre llamada y llamada de esta rutina de gráficos el simulador ejecuta otras piezas de código tales como el cálculo de las físicas del juego o el código de los propios plugin; el periodo que ocurre entre el inicio del dibujado y su finalización se llama Flight Loop.

Para el desempeño de las tareas periódicas el SDK de X-Plane pone a disposición del desarrollador una funcion para registrar tareas que se invocaran automáticamente entre Flight Loops consecutivos (como si de una interrupción se tratase). Esta funcion se llama XPLMRegisterFlightLoopCallback(), y se le pasa como parámetros un puntero a la funcion que se va a ejecutar periódicamente y el periodo entre llamadas consecutivas.
Cabe destacar que, si el temporizador que controla la siguiente llamada de una rutina llega a 0 durante un Flight Loop, la ejecución de la rutina se produce justo después de la finalización del Flight Loop. La funcion que se llama periódicamente debe devolver el número de segundos que deben pasar hasta la próxima llamada o -1 si lo que se quiere es llamarla justo después del próximo Flight Loop.

Otra cuestión que se debe tener en cuenta es el acceso a los datos de simulación. Estos parámetros se conocen dentro del simulador como DataRefs, y no se puede acceder directamente a ellos. Para su lectura y escritura el SDK de X-Plane ofrece una API llamada XPLMDataAccess y es a través de esta API como se deben localizar para su posterior lectura. Como son referencias a datos hay que buscarlas por su nombre según la lista del Apéndice B, y una vez establecidas las referencias se podrán leer los parámetros. La localización de las referencias por nombre es muy costosa por lo que el plugin debe implementar algún mecanismo de almacenado de las referencias, y realizar la búsqueda solamente en la inicialización.

3.3 Parámetros del servidor

La interfaz que externaliza los parámetros de la simulación será un servidor creado con Sockets. Las guías de desarrollo de X-Plane recomiendan que las tareas entre dos Flight Loop sean lo más rápidas posibles para que no afecten al rendimiento del simulador, por lo que en

la inicialización del plugin se declarará el Socket, que tendrá los siguientes parámetros de configuración:

- Dirección IP Multicast: **239.255.255.230**
- Puerto : **4952**

(El uso de una dirección Multicas facilita, en el futuro, que los datos exportados por el plugin puedan ser recibidos simultáneamente por más de una instancia del receptor).

Tras inicializar el Socket comienza el envío de los parámetros del simulador en un solo datagrama en forma de caracteres ASCII, incluyéndolos en una cadena y usando saltos de línea a modo de elemento separador, lo que hará que el tratamiento de estos datagramas sea más sencillo en el SW_Consumidor. El tamaño de los datos de cada datagrama es de 1024 Bytes; esto es debido a que el simulador de vuelo opera con datos en coma flotante, y por tanto el número de caracteres necesarios para cada valor es considerable.

3.4 Estructura para los datos recibidos por el SW_Consumidor

El diseño del SW_Consumidor se hace de manera análoga al SW_Productor, y consistirá en un cliente con Sockets que lea de la misma dirección y puerto con los que opera el HW_Productor los datagramas que lleguen. Al ser cadenas de texto, para un ordenador es sencillo tokenizar la cadena con los separadores.

Para almacenar los datos se usa una clase Datos cuya única funcion es la de almacenar estos datos y así permitir posteriormente establecer un sistema de bloqueo sobre el objeto que representa la clase para que la lectura y escritura concurrente se produzcan en exclusión mutua. El uso de esta estructura también permite que el acceso desde distintos puntos del código sea sencillo, pues solo se deberá pasar una referencia al objeto y no una referencia a cada parámetro almacenado al que se quiera acceder.

3.5 Lectura y escritura concurrente usando doble buffer

Hay dos componentes que acceden a la estructura que contiene los parámetros de simulación: por un lado, el SW_Consumidor accede cada vez que le llega un datagrama desde el SW_Productor y refresca los datos que la estructura tuviera, y por otro lado, cuando se está realizando la simulación de un instrumento, el HW_Productor (que está atendiendo constantemente las peticiones que le llegan desde el HW_Consumidor) accede a la estructura para leer los últimos parámetros disponibles.

Por lo tanto, es lógico pensar que el acceso a la estructura se debe hacer en exclusión mutua. Esto presenta el inconveniente de que si el SW_Consumidor quiere refrescar los datos de la estructura y justo en ese momento el HW_Productor este leyendo, el primero deba esperar al segundo (y viceversa). Esto merma el rendimiento de la aplicación ya que se producen pausas que serán notables sobre todo cuando el HW_Consumidor pida una nueva trama de parámetros de simulación al HW_Productor y este tenga que esperar a que el SW_Consumidor actualice la estructura.

Para solucionar este problema es conveniente usar la técnica de lectura y escritura mediante doble buffer, técnica ampliamente usada en codificación de video y renderizado de videojuegos. Consiste en duplicar la estructura a la que se va a acceder concurrentemente y alternar entre estas estructuras los accesos de la lectura y escritura, que ya no se producirán simultáneamente sobre las mismas posiciones de memorias. Concretamente, para su operación el componente productor empieza a escribir continuamente en la primera estructura; cuando el componente Consumidor quiere acceder a la estructura, pedirá el acceso a la estructura en la que está escribiendo el Productor, ya que es la que contiene la información más reciente. Es en este momento cuando el Productor, pasa a escribir sobre la segunda estructura, y deja libre la primera con los datos más recientes para el consumidor. Cada vez que el consumidor quiere acceder a los últimos datos se repetirá este proceso con la estructura en la que esté operando el productor, de manera que el consumidor siempre dispondrá de los datos más recientes y ninguno de los dos elementos tiene que interrumpir su tarea.

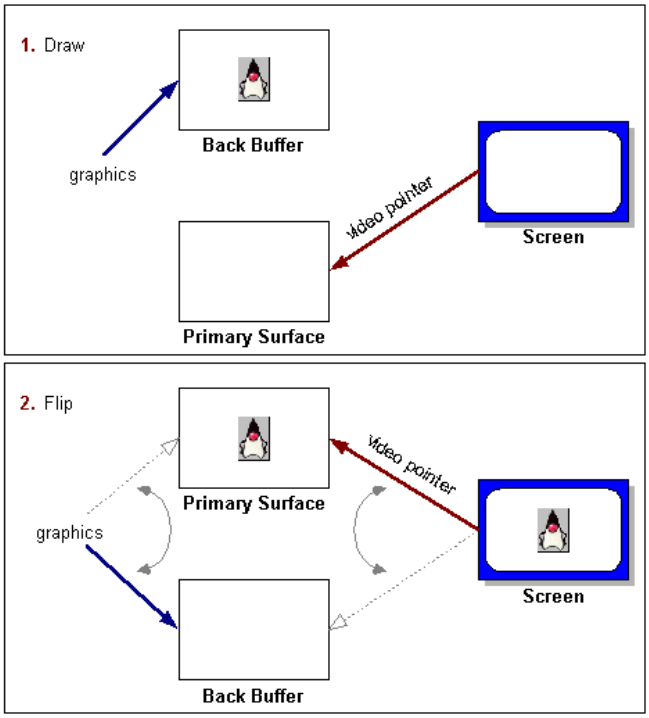

Figura 3.4: Ejemplo de double buffering para renderizado de video en pantalla.
*Fuente: "*https://docs.oracle.com/javase/tutorial/extra/fullscreen/doublebuf.html*" (24 de agosto de 2020).*

3.6 Protocolo de comunicación

La comunicación entre el programa principal o HW_Productor y la placa Arduino o HW_Consumidor se realizará mediante la interfaz USB. Se realizarán envíos sincronizados de tramas de datos diseñadas específicamente para este propósito, por cuestiones de rendimiento, y debido a la escasez de recursos de la placa Arduino, el envío de los datos se realizará directamente en binario; aunque sería más sencillo operar con cadenas de texto de cara a la programación del algoritmo, el rendimiento del sistema se vería mermado cuando la placa Arduino tuviera que hacer el parseo de las cadenas de texto y la subsiguiente conversión de tipos.

El protocolo de comunicación consta de dos fases: en la primera fase el HW_Consumidor recibe los parámetros de configuración para la simulación que se va a realizar, y una vez confirmada la correcta recepción y aplicada la configuración se pasa a la fase de transmisión de datos de simulación para actuar sobre los elementos mecánicos que simulan el instrumento analógico.

3.6.1 Fase de configuración

En un estado inicial el HW_Consumidor se encuentra sin configuración, y no tiene información alguna de que es lo que se va a simular durante la sesión. por lo que esperará de manera pasiva a que le llegue una trama de configuración por la interfaz USB.

Es el HW_Consumidor el que inicia la comunicación enviando una trama de configuración. Esta trama debe contiene todos los parámetros necesarios para actuar sobre los elementos mecánicos de manera que se muevan como lo haría el instrumento real. La trama de configuración debe incluir primero cual es la familia de instrumentos que se va a representar. Esto se hace mediante un identificador asignado a cada familia de instrumentación. Además, dado que con placas programables Arduino UNO se pueden conectar un máximo de 3 motores paso a paso al conjunto de pines digitales de los que dispone, se necesitarán establecer los parámetros para tres motores paso a paso independientemente, ya que cada uno puede representar un instrumento que, aunque pertenezca a la misma categoría, puede tener un comportamiento completamente diferente. Como el objetivo es hacer que la trama sea estándar a cualquier configuración, se debe diseñar la trama de configuración de los motores paso a paso de manera que se puedan emular todos los comportamientos posibles. En consecuencia, los parámetros que debe incluir la trama de configuración son:

- **Índice:** Índice del parámetro dentro de la trama de datos que pedirá el HW_Consumidor al HW_Productor.
- **Tope:** Es el número de unidades que tiene la circunferencia del instrumento en un giro de 360º.
- **Overflow:** Indica si se permite que la aguja siga girando una vez alcanzado los topes máximos o no.
- **Escala:** Debido a que el simulador de vuelo trabaja con valores en coma flotante y este tipo de datos tiene un impacto negativo enorme en el rendimiento de la placa Arduino se va a trabajar con números enteros con factor de escala. Cada instrumento puede estar representado con un factor de escala diferente.
- **Límite inferior:** Indica cual es el tope inferior de la aguja del instrumento en base al sistema de unidades circular que se ha creado con el tope.

- **Límite superior:** Indica cual es el tope superior de la aguja del instrumento en base al sistema de unidades circular que se ha creado con el tope.
- **Dirección de giro:** Indica si la dirección de giro será a izquierda o derecha con los valores positivos y el caso contrario con los negativos.
- **Posición inicial:** Indica cual es la posición inicial de la aguja en el instrumento respecto del sistema de unidades circular que se ha creado con el tope. Esta posición es la que tiene el instrumento cuando la aeronave está completamente parada.

La trama entera se repetirá tres veces, una para cada posible motor paso a paso. Por último, se debe enviar un valor indicando el número de motores paso a paso que se van a usar en la simulación, de manera que, aunque se reciban tres configuraciones solo se operará sobre el número de motores que se indique en este último parámetro.

Tras la recepción de esta trama, el HW_Consumidor devuelve una réplica de la trama al HW_Productor, que comprueba que lo que ha recibido es correcto comparándola con la trama que envió inicialmente, y le confirma al HW_Consumidor que puede aplicar la configuración recibida.
Por último, se realiza la calibración de los instrumentos (si el usuario ha indicado en la configuración que es necesaria). Este proceso puede hacerse manualmente mediante pulsadores, o automáticamente mediante elementos electrónicos. A partir de este paso el HW_Productor pasa a ser pasivo y espera una petición de envío desde el HW_Consumidor, momento en que se pasa a la segunda fase del protocolo de comunicación.

3.6.2 Fase de envío de datos de simulación

Establecida la configuración solo queda empezar el intercambio de los datos de simulación. La mejor manera de hacer esto, teniendo en cuenta la limitación de recursos de la placa Arduino, es mediante un sistema "pull", es decir, va a ser el HW_Consumidor el que demandará los datos de simulación al HW_Productor bajo demanda cuando tenga capacidad de procesarlos.

El HW_Consumidor enviara una trama que indicara al HW_Productor que se está solicitando la trama de datos de instrumentación más reciente de la que se disponga. Tras la recepción y comprobación del tipo de trama se lee de la estructura de almacenamiento la versión más reciente aplicando el mecanismo de cambio del doble buffer. Una vez leída la trama se envía de vuelta por la interfaz USB al HW_Consumidor, que la recibe en una estructura de tamaño determinado por la configuración inicial y usa su contenido para actuar sobre los motores paso a paso y moverlos a la posición que deben tener, según los datos que se han recibido.

Este proceso se repite una vez haya terminado el movimiento de los motores paso a paso. Dado que es la placa Arduino la que solicita el envío de las tramas, está garantizado que siempre tendrá capacidad para procesar si contenido; esto es especialmente importante porque la placa Arduino es mucho más lenta en el procesamiento que el PC, y debe ser ella, por tanto, la que marque el ritmo al que se pueden procesar las tramas.

3.6.3 Métodos de control de detección de errores

Tanto en la fase de configuración como en la de envío de parámetros se usan tiempos de espera acotados a los que no se deberían llegar en un funcionamiento normal. Tras el envío de cada trama durante la fase de configuración se usan estos tiempos de espera o Timeout de manera que es posible detectar fallos de desconexión o falta de sincronización por ambas partes, tanto en el HW_Productor como en el HW_Consumidor.

Durante la fase de envío de parámetros de simulación el HW_Productor espera constantemente peticiones de envío, si se agotase el tiempo de espera se detendría el protocolo de comunicación ya que significaría desconexión de la placa Arduino.
En el lado del HW_Consumidor el protocolo de comunicación envía tramas de petición de envío de parámetros. Si tras una petición no se recibe la trama que se ha pedido y se agota el tiempo de espera, se vuelve a hacer una petición (Ya que esto significa que se ha perdido o bien la trama de petición o la trama con los parámetros de simulación). Tras agotar tres veces consecutivas el tiempo de espera en las peticiones, el HW_Consumidor vuelve a su estado inicial pidiendo una configuración, ya que esto significa que el HW_Productor se ha

desconectado, bien porque se ha desconectado la interfaz USB o bien por qué ha finalizado la sesión de simulación de manera abrupta.

3.6.4 Calibración de instrumentos

La calibración de instrumentos no es más que la fase en la que se posicionan las agujas de los instrumentos en lo que llamaremos *posición 0* o *posición base*. Este proceso es necesario porque los motores paso a paso no tienen registro de su posición actual, y el control de esta posición se debe hacer mediante software. El problema se presenta cuando se produce un reinicio de la placa programable ya que la posición actual respecto al inicio se pierde y no hay manera de saber en qué posición se encuentra la aguja al iniciar una nueva sesión de simulación.

Este inconveniente podría tener fácil solución si cada vez que la placa Arduino moviese un motor paso a paso almacenase seguidamente la cuenta de pasos que lo ha movido en memoria no volátil. Pero la placa Arduino usa memoria EEPROM para almacenar información que no se borra tras el reinicio, por lo que esto no es factible (ya que se desgastaría rápidamente la memoria Flash NAND de la placa Arduino cuya vida útil es de unas 100000 escrituras, cifra rápidamente alcanzable si se escribe en memoria cada vez que se modifica el número de pasos que ha dado el motor paso a paso.

La solución pasa por realizar una calibración cuando los instrumentos no se encuentran en la posición que deben tener al inicio de una sesión de simulación. Este proceso se puede realizar manualmente con pulsadores que, mediante su accionamiento por el usuario hagan que se muevan los motores paso a paso hasta dejarlos en la posición que deben tener, o se puede realizar automáticamente mediante elementos electrónicos de manera automática.

La calibración automática se consigue cerrando un circuito que produzca una corriente eléctrica en una de las entradas del Arduino mediante elementos como interruptores de lengüeta o con un sensor infrarrojo y un emisor de este tipo de luz. Un interruptor de lengüeta o Reed switch es un interruptor eléctrico que se cierra con la proximidad de un campo magnético. Consiste en dos filamentos ferrosos encerrados en un tubo de vidrio sellado al vacío los cuales se encuentran separados en su estado de reposo.

Al acercar un campo magnético al interruptor los dos filamentos se unen cerrando el circuito que haya en las extremidades exteriores de los filamentos. La construcción en un instrumento con aguja requiere de la colocación de este elemento bajo la esfera del instrumento, y de un pequeño imán bajo la aguja del instrumento. De esta manera, para realizar la calibración solo hay que poner a girar el motor paso a paso conectado a la aguja de manera constante y cuando la aguja pase sobre la posición base bajo la cual se encontrará el interruptor de lengüeta, este cierre el circuito por la acción del campo magnético del imán y se produzca una corriente en la entrada de la placa Arduino tal y como se muestra en el esquema de las figuras C2 y C.7.

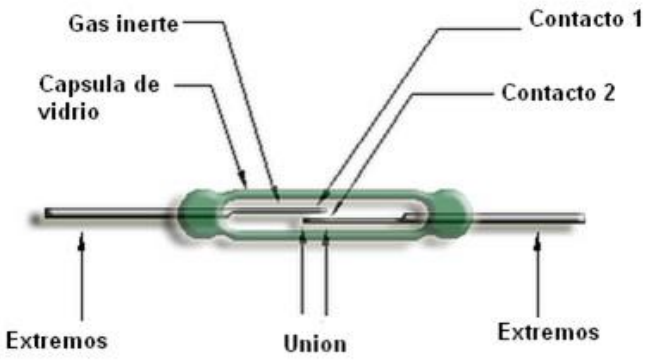

Figura 3.5: Esquema explicativo de un interruptor de lengüeta. Fuente: "http://sistemaselectricosdelautomovil.com/aplicaciones-del-sensor-reed-en-el-automovil/" (26 de agosto de 2020).

Un sensor de obstáculos mediante luz infrarroja consiste en un dispositivo con un LED emisor de luz infrarroja y un fotodiodo que recibe luz. El funcionamiento consiste en la detección mediante el fotodiodo, un comparador y un potenciómetro de la luz reflejada sobre un objeto emitida por el LED. De esta manera colocando el emisor y receptor del sensor en la esfera del instrumento, se podrá crear un circuito de la misma manera que con el Reed switch para que cuando la aguja pase por encima del sensor, esta refleje la luz infrarroja, accione el sensor de obstáculos y este cierre el circuito, generando corriente eléctrica en la entrada de la palca Arduino.

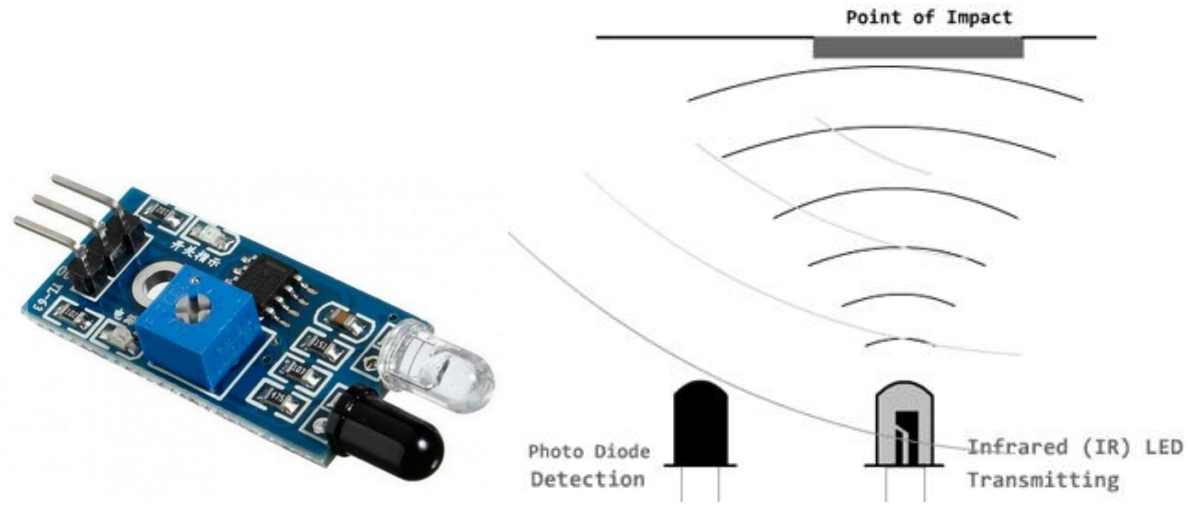

Figura 3.6: Sensor de obstáculos infrarrojo y funcionamiento del sensor mediante reflexión de luz. Fuente: "https://www.luisllamas.es/detectar-obstaculos-con-sensor-infrarrojo-y-arduino/" (26 de agosto de 2020).

4

Implementación

Una vez completadas las fases de especificación y documentación, y de diseño, la implementación crea el código que lleva a la práctica las funcionalidades que se han especificado. La arquitectura del sistema se ha diseñado de forma modular, por lo que estructurar el código a implementar es más sencillo, y es posible la reutilización de código cuando sea posible. Otra ventaja adicional es que si en una de las etapas de implementación se descubre un fallo de especificación es mucho más sencillo solventarlo de lo que sería en un sistema equivalente con estructura monolítica. El desarrollo del código se llevará a cabo por módulos, y la implementación de cada módulo se realizará por etapas, de la forma habitual en un modelo de desarrollo iterativo e incremental.

4.1 Plugin de X-Plane (SW_Productor)

El plugin de X-Plane o SW_Productor realiza la extracción de datos de la simulación, y su envío a un programa externo mediante una interfaz. El módulo se implementa usando el lenguaje de programación C, y la interfaz, como se indica en el capítulo 3, será un servidor creado mediante Sockets que implementa el protocolo TCP/IP. El desarrollo de este módulo se realiza en una sola etapa.

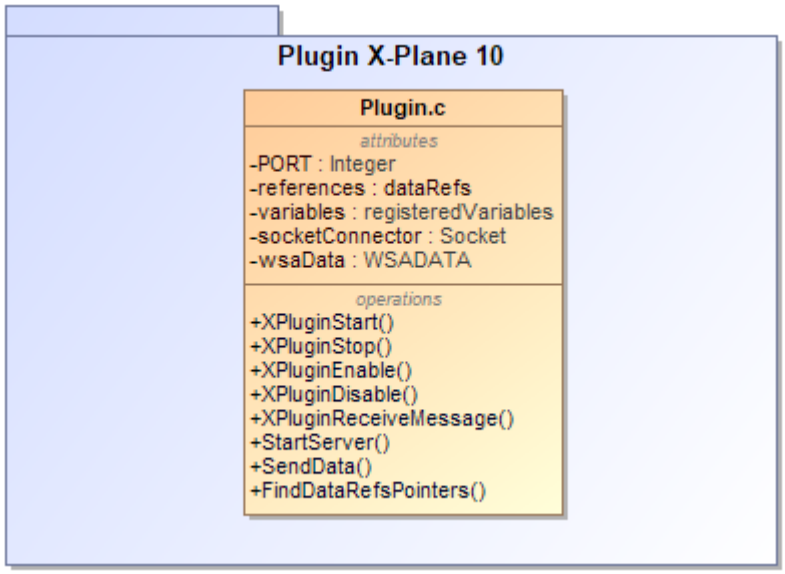

Figura 4.1: Diagrama de clases del módulo Plugin de X-Plane 10

El módulo contiene los métodos descritos en el capítulo 3.2, necesarios para que X-Plane lo reconozca como un plugin. El punto de entrada al cargar el plugin es el método **XPluginStart()**; este método se encarga de inicializar el Socket que actúa de servidor y las estructuras de datos que se va a usar, y además proporciona al simulador los detalles y descripción del plugin.

Dentro de **XPluginStart()** primero se llama al método **FindDataRefsPointers()**. Este método carga en la estructura *references* los punteros que apuntan a los parámetros de simulación que se desean leer, y que localiza por nombre. La lista de parámetros de simulación y sus respectivos nombres para los que se buscan y guardan las referencias en el simulador de vuelo se especifican en el Apéndice B. Como el coste computacional de localizar las referencias a los parámetros por nombre es alto, esta búsqueda se realiza una sola vez y se almacenan las referencias. A continuación, se realiza la inicialización del servidor mediante el método **StartServer()**; en él se inicializa la estructura *wsaData* que contiene los parámetros de los sockets usados, así como la información necesaria para la gestión de errores, tras lo cual se crea el socket para las comunicaciones externas con la dirección de Multicast ***239.255.255.230*** y el puerto ***4952***, que está definido en la variable global PORT. Para crear el servidor mediante Sockets se hace uso de la librería Winsock (librería estandarizada que permite el desarrollo de aplicaciones que hacen uso de comunicaciones por Internet en la plataforma Windows).

Una vez iniciado el servidor y almacenadas las referencias a los parámetros de simulación, se registra el evento de callback que llamara a la funcion **SendData()** entre ciclos Flight Loop cada 0.1 segundos. Esta funcion actualiza en la estructura *variables* los valores de los datos de simulación a los que apuntan las referencias almacenadas en la estructura *references* y realiza el envío metiendo todos esos datos en una cadena que se enviará, mediante el socket que se ha creado, a la dirección de Multicast.

El método **XPluginStop()** libera los recursos usados en memoria, cierra el socket que se ha usado para la comunicación y libera la librería Winsock.

4.2 Programa principal

Como este módulo realiza diversas tareas simultáneamente se puede dividir en módulos más pequeños (el SW_Consumidor y el HW_Productor descritos en el capítulo 3.1) que operan independientemente, pero que en la arquitectura general del sistema funcionan como uno solo. La implementación de cada uno de estos módulos se realiza en una etapa de desarrollo diferente.

4.2.1 Implementación SW_Consumidor

Este módulo trabaja conjuntamente con el SW_Productor; su tarea es recibir continuamente los datagramas que envía el SW_Productor, y actualizar la estructura de memoria correspondiente con los datos de simulación recibidos en el datagrama; esta actualización de datos se realiza usando la técnica de doble buffer, en exclusión mutua con los accesos del HW_Productor.

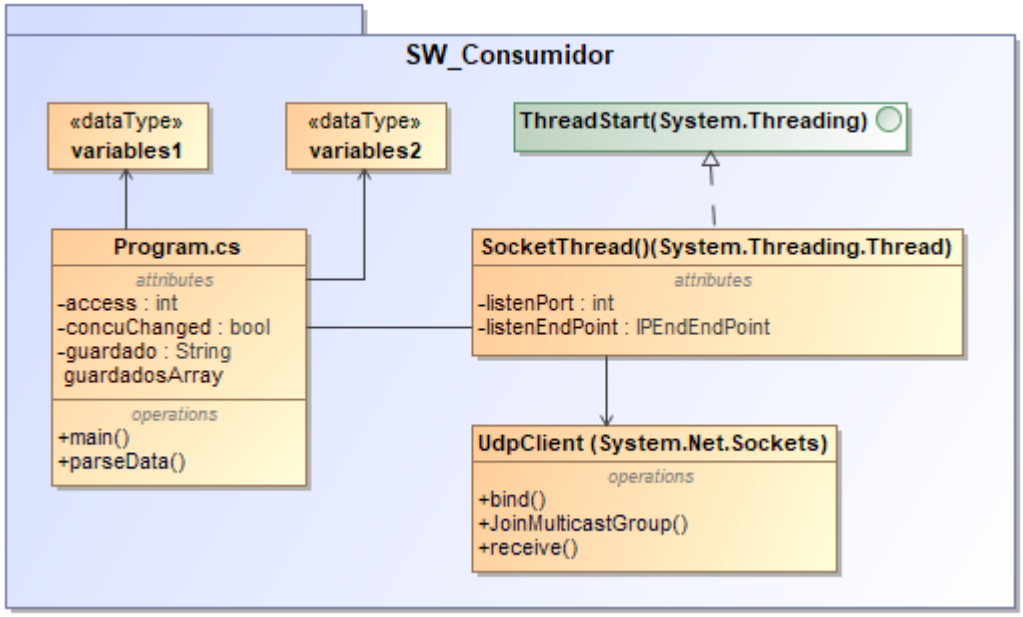

Figura 4.2: Diagrama de clases del módulo SW_Consumidor.

El elemento principal es la clase *Program.cs*, que contiene el método **main()**, que es el que da vida a todo el mecanismo. Inicialmente, **main()** crea una hebra a la cual le pasa la referencia del método que tiene que ejecutar (en este caso **SocketThread()**). Este método crea un cliente UDP que se enlaza a la dirección Multicast y puerto a través de los que está enviando datos el SW_Productor, y se pone a la escucha de cualquier datagrama que pueda llegar.

Cada vez que se recibe un datagrama, este es procesado con el método **parseData()**. En este método se transforman los números en coma flotante que contienen los datagramas en enteros con factor de escala, lo que hará que el rendimiento tanto del programa principal como del HW_Consumidor mejore (ya que la computación con números enteros es más liviana que con números en coma flotante). Finalmente, el resultado de este procesamiento se almacena en la estructura de datos que corresponda de las dos usadas por el mecanismo de doble buffer; a lógica del doble buffer se gestiona, antes de llamar al método **parseData()**, mediante atributos *access* y *concuChanged* que se usan para la sincronización y cambio de la estructura activa, tras lo cual se pasa al método **parseData()** la referencia de la estructura en la que se va a escribir. La lectura y escritura en exclusión mutua se consigue mediante la instrucción *lock()* de C#. Esta instrucción crea un lock sobre el objeto que contiene la estructura de datos en la que se va a escribir de manera que ninguna otra hebra pueda acceder al objeto mientras se produce la escritura.

4.2.2 Implementación HW_Productor

Este módulo contiene las interfaces graficas del programa con las que interaccionara el usuario para seleccionar la configuración, para luego, haciendo uso del protocolo de comunicación diseñado, realizar el envío de datos de simulación al HW_Consumidor.

Para su implementación se usa el lenguaje C#, ya que permite la creación de interfaces de usuario con el framework .NET para Windows (Windows Forms) [8] con el estilo arquitectónico Modelo-Vista-Controlador [9].

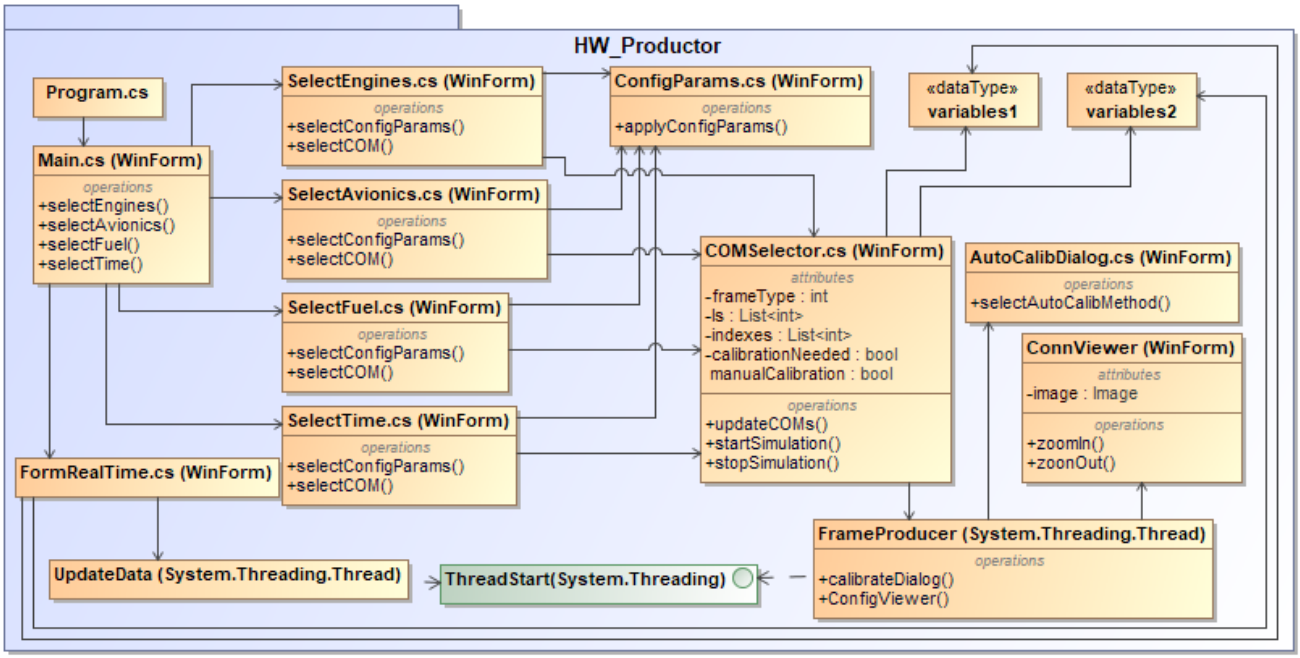

Figura 4.3 Diagrama de clases del módulo HW_Productor.

El punto de entrada, al igual que ocurre en el módulo SW_Consumidor es la clase Program.cs que contiene el método **main()**, pero en este caso el método inicia una hebra que crea la interfaz gráfica de la aplicación la cual está representada por la propia clase Main. Las interfaces graficas con Windows Forms están encapsuladas en clases independientes, que por un lado contienen el diseño y la disposición de los elementos con los que podrá interaccionar el usuario (tales como botones, campos de texto o imágenes), y por otro lado implementan la lógica de negocio asociada a cada componente con el que puede interaccionar el usuario.

Al cargar la interfaz gráfica **Main** el usuario puede elegir ver los parámetros que se están recibiendo en el SW_Consumidor o iniciar la simulación de un tipo de instrumentos mediante las cuatro categorías que se presentan. (En el Apéndice A se pueden ver las pantallas que presenta la aplicación al usuario).

En el caso de que la selección sea mostrar los parámetros de simulación en tiempo real se oculta el menú principal y se genera la interfaz gráfica **FormRealTime**. Esta interfaz contiene una etiqueta de texto por cada parámetro de instrumentación almacenado en el sistema. Como la actualización de las interfaces graficas en el modelo MVC lo realiza el controlador según los datos del modelo y la vista es completamente pasiva, es necesario un elemento que este actualizando la información que se muestra por pantalla constantemente con los parámetros que recibe el HW_Consumidor. Este elemento es implantado con una hebra a la que se le pasa el método **UpdateData()** cuya tarea es la lectura constante de los últimos datos de simulación desde el doble buffer. Esta hebra lee los valores y delega en su creador (es decir, en la interfaz gráfica **FormRealTime**)la actualización de las etiquetas que se le muestran al usuario.

En el caso de que el usuario elija iniciar la simulación de un tipo de instrumentos se muestra la ventana de selección correspondiente a la familia de instrumentos seleccionada; así, para instrumentos relacionados con motores se inicia **SelectEngines**, para instrumentos relacionados con aviónica y navegación se inicia **SelectAvionics**, para instrumentos relacionados con los tanques de combustible se inicia **SelectFuel** y para instrumentos relacionados con la hora y temperatura se inicia **SelectTime**.
En cada una de estas clases se realiza la renderización de una interfaz gráfica que permite al usuario seleccionar los instrumentos que se van a representar. Estas interfaces tienen en cuenta la limitación máxima de 3 instrumentos por placa Arduino UNO, y no permiten al usuario seleccionar más de ese número de instrumentos. Además, la cantidad de instrumentos disponibles para seleccionar varía en funcion de las características de la aeronave simulada en X-Plane (por ejemplo, se limita la cantidad de tanques de combustible o motores posibles al número de estos elementos de que disponga la aeronave simulada).

Una vez realizada la selección de instrumentos como se muestra en la Figura 4.4, se inicia y muestra un formulario de la clase **ConfigParams** en el que el usuario podrá seleccionar un diseño de instrumento analógico adecuado al instrumento seleccionado (por ejemplo, si se selecciona un indicador de velocidad vertical se muestran varios diseños de indicadores de este tipo), y se establecerán sus parámetros de configuración o bien se dará la opción de introducir los parámetros de configuración que el usuario considere oportunos para realizar

la representación. También es posible ver los parámetros de configuración asociados a los instrumentos preestablecidos en la aplicación y modificarlos antes de confirmar la selección. La descripción de esta interfaz se puede observar en la Figura 4.5. Al intentar continuar se pide confirmación antes de establecer los parámetros ya que este paso no se puede deshacer, y corregir un error en los parámetros de configuración requiere el reinicio del proceso.

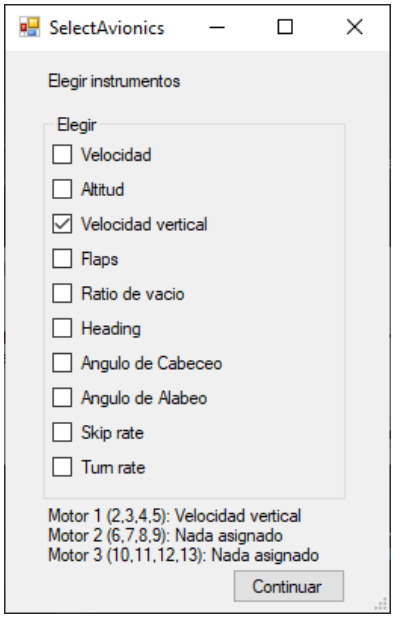

Figura 4.4: Interfaz gráfica de la clase SelectAvionics.

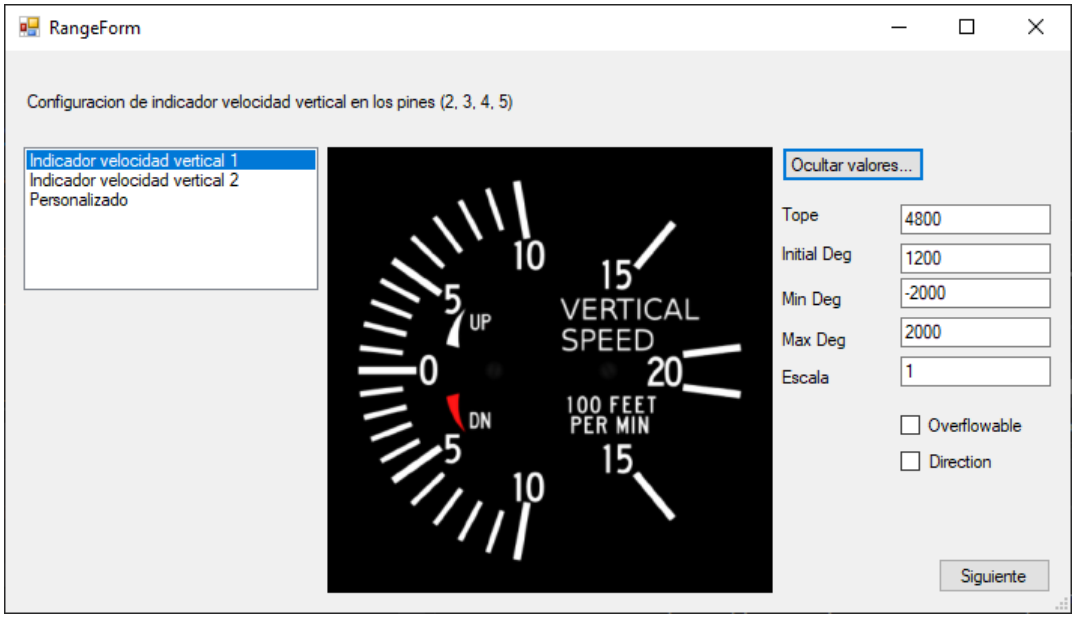

Figura 4.5: Interfaz gráfica de la clase ConfigParams.

Cuando se hayan introducido los parámetros de configuración para cada instrumento seleccionado se inicia la clase **COMSelector,** la cual almacena la configuración establecida hasta ahora (esto es, identificador de la familia de instrumentos que se van a representar en el parámetro *frameType*, parámetros de configuración en la lista *ls*, e índices dentro de cada familia de los instrumentos seleccionados en *indexes*). Esta clase muestra la lista de puertos a los que hay conectado una placa Arduino, y la posibilidad de actualizar esta lista, si el usuario así lo quiere. La interfaz incluye en dos botones que inician o detienen la simulación de los cuales solo uno estará activado en cada momento dependiendo del estado de la simulación.

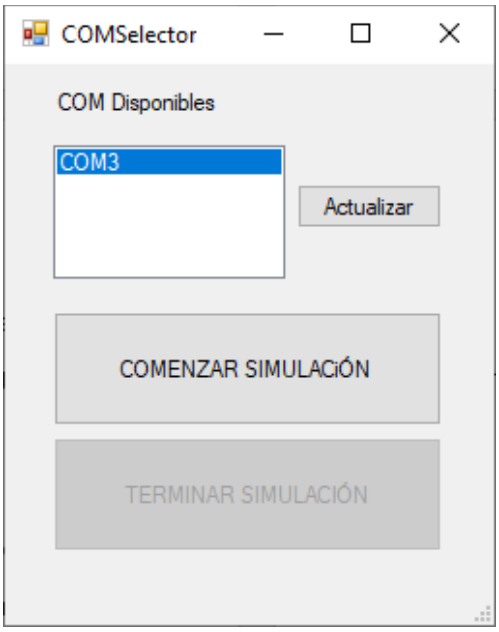

Figura 4.6: Interfaz gráfica de la clase COMSelector.

Los puertos que muestra lista interfaz se corresponden con puertos COM a los que hay una placa Arduino, y se diferencian en que cada uno tiene un número asociado. Cabe señalar para evitar confusión que, aunque la nomenclatura que se le da a la comunicación entre el PC y la placa Arduino se denomina comunicación por puerto serie no tiene nada que ver con conexión por RS-232. La placa Arduino usa un UART [10] para la transmisión y recepción de datos que se realiza por la interfaz USB que tiene la placa por defecto [11].

Al seleccionar un puerto COM y pulsar sobre el botón de inicio de la simulación se crea una hebra que ejecuta el método **FrameProducer()** el cual se encarga se preguntar al usuario si se requiere calibración de los instrumentos o no: la calibración es requerida cuando la posición en la que se encuentran las agujas que mueven los motores paso a paso no es la posición de inicio o posición base.

Si la respuesta es negativa se muestra al usuario el diagrama de conexión del cableado y motores que se requiere implementar para que el sistema funcione. Si la respuesta es afirmativa se pregunta si la calibración será manual, mediante pulsadores, o automática. En este último caso se pregunta también qué tipo de sensor se va a utilizar para realizar la calibración automática mediante el inicio de la clase **AutoCalibDialog**. Tras esto, se muestra el diagrama de conexión del cableado, motores y sensores que corresponda a cada caso según lo elegido por el usuario haciendo uso de la clase **ConnViewer** y se inicia el protocolo de comunicación con el Arduino.

4.3 Algoritmo placa programable Arduino (HW_Consumidor)

Este módulo implementa el protocolo de comunicación diseñado, prepara la placa para operar con la configuración que recibe desde el programa principal, y, una vez configurado, posiciona los instrumentos en la posición inicial que deben tener (llamaremos "calibrado" a este proceso final).

Por último, y en un bucle sin fin mientras dure la sesión de simulación, el módulo pedirá al programa principal tramas con los parámetros de simulación más recientes para actuar sobre los motores paso a paso.

Para la conexión de los motores a la placa Arduino se usan 4 pines de salida que se conectan con 4 entradas del motor mediante una placa controladora. Como se puede observar en la Figura 4.7 cada entrada del motor está conectado a una bobina que actúa de electroimán y provoca el movimiento del mecanismo. El funcionamiento de la placa controladora es asociar un pin de salida de la placa Arduino con una de las bobinas, de manera que cuando reciba corriente por ese pin, pase corriente eléctrica por la bobina asociada. Se puede observar en el Apéndice C como conectar los motores y la placa Arduino con la controladora en los distintos diagramas de interconexión (Figuras C.1 - C.12).

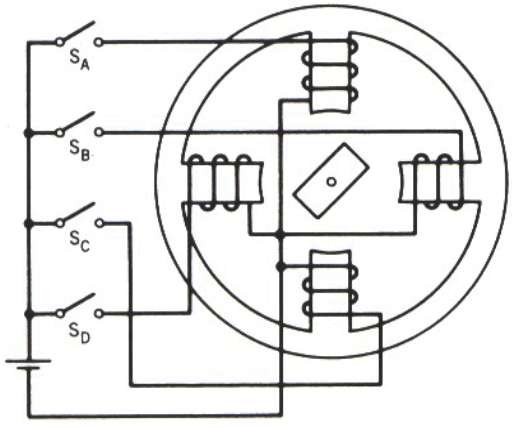

Figura 4.7: Esquemático de un motor paso a paso de 4 bobinas. Fuente: "http://electricalarticle.com/stepper-motor/" (31 de agosto de 2020).

Para actuar sobre los motores paso a paso solo se debe excitar las bobinas asociadas a cada pin en un orden concreto de manera que el movimiento que se produzca sea circular y se produzca en un determinado sentido. Esto se puede hacer manualmente creando una matriz de unos y ceros la cual se recorrerá en un orden especifico excitando las bobinas según los 1 y 0 de cada fila de la matriz (cada fila representa el estado de excitación de los pines asociados al motor paso a paso en un momento determinado.

```
int Paso [ 8 ][ 4 ] =
{ {1, 0, 0, 0},
  {1, 1, 0, 0},
  {0, 1, 0, 0},
  {0, 1, 1, 0},
  {0, 0, 1, 0},
  {0, 0, 1, 1},
  {0, 0, 0, 1},
  {1, 0, 0, 1}
};
```

Figura 4.8: Matriz para el movimiento de un motor paso a paso haciendo medios pasos en cada movimiento.

El movimiento se puede realizar a medios pasos excitando primero una bobina y luego esa bobina y la siguiente para dar un paso completo, como se hace en la matriz de la Figura 4.8 (de manera que, para dar medio paso, hay que cambiar el estado actual que se tenga en los pines por el estado correspondiente al representado por la fila siguiente de la matriz), o de otra manera, realizando pasos completos en cada movimiento activando los pines de dos bobinas cada vez. En este caso, la matriz tendría solo 4 filas y en cada fila habría dos 1 que representarían la activación de bobinas consecutivas.

El inconveniente principal que presenta el método de usar una matriz y hacerlo manualmente es que la manera de dar un paso es bloqueante, es decir, mientras un motor esté moviéndose el Arduino no puede ejecutar otra parte del código (tal como actualizar la trama de parámetros o mover otro motor paso a paso al mismo tiempo), debido a que mover el motor varios pasos requiere de un bucle que recorra las distintas posiciones de la matriz de manera continua hasta que se alcanza la posición deseada en el motor paso a paso.

Para solventar esto se usa la librería AccelStepper [12]. Esta librería, creada por Mike McCauley [6], permite el control de distintos tipos de motores paso a paso, ofreciendo características como control de aceleración, la velocidad en su operación, o el control simultaneo de varios motores paso a paso, pero lo más importante (y la característica clave para este proyecto) es que el control de uno o varios motores paso a paso se realiza de manera no bloqueante, por lo que se podrán controlar varios motores o realizar otras tareas mientras

uno o varios motores se están moviendo. La librería, además, utiliza una interfaz orientada a objetos, que simplifica mucho el código del algoritmo y su legibilidad.

Otro problema que se resuelve en el módulo implementado para el Arduino es la diferencia en la representación en memoria de tipos de datos entre C# y C++ para Arduino. Como se puede observar en las Figuras 4.9 y 4.10, un número entero o *int* en C# está representado en memoria por 4 bytes, es decir, 32 bits, mientras que un número entero en Arduino está representado en memoria por 16 bits, por lo que los enteros enviados desde el HW_Productor deben ser almacenados en el HW_Consumidor usando valores long.

Data Types	Memory Size	Range
char	1 byte	-128 to 127
signed char	1 byte	-128 to 127
unsigned char	1 byte	0 to 255
short	2 byte	-32,768 to 32,767
signed Short	2 byte	-32,768 to 32,767
unsigned Short	2 byte	0 to 65,535
int	4 byte	-2,147,483,648 to 2,147,483,647
signed int	4 byte	-2,147,483,648 to 2,147,483,647
unsigned int	4 byte	0 to 4,294,967,295
long	8 byte	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
signed long	8 byte	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned long	8 byte	0 to 18,446,744,073709,551,615
float	4 byte	$1.5 * 10^{-45}$ to $3.4 * 10^{38}$ (7 Digit)
double	8 byte	$5 * 10^{-324}$ to $1.7 * 10^{308}$
decimal	16 byte	$-7.9 * 10^{28}$ to $7.9 * 10^{28}$

Figura 4.9: Definición de tipos primitivos en el lenguaje C#.
Fuente: "https://www.theengineeringprojects.com/2018/02/introduction-to-data-types-in-c-sharp.html" (31 de agosto de 2020).

Arduino Data Types	Value Assigned	Value Ranges
boolean	8 Bit	True or False
byte	8 Bit	0 to 255
char	8 Bit	-127 to 128
unsigned char	8 Bit	0 to 255
word	16 Bit	0 to 65535
unsigned int	16 Bit	0 to 65535
int	16 Bit	-32768 to 32767
long	32 Bit	-2,147,483,648 to 2,147,483,647
float	32 Bit	-3.4028235E38 to 3.4028235E38

Figura 4.10: Definición de tipos primitivos en el lenguaje C++ para Arduino.
Fuente: " https://www.theengineeringprojects.com/2017/02/arduino-data-types.html *" (31 de agosto de 2020).*

El problema de conversión de tipos en el Arduino se complica, además, porque el protocolo de comunicación USB envía al HW_Consumidor los valores como bytes que luego tienen que agruparse para obtener el valor que representan. Esto se consigue usando el tipo de datos *union*, que permite manejar un dato almacenado en un fragmento de memoria como si fuera distintos tipos de datos a la vez. De esta manera se almacenarán en *arrays* de bytes de 4 elementos los números recibidos y mediante una union se accederá a ellos como un long, tal y como se muestra en la Figura 4.11, consiguiendo así el mismo valor que se envió desde el HW_Productor como *int*.

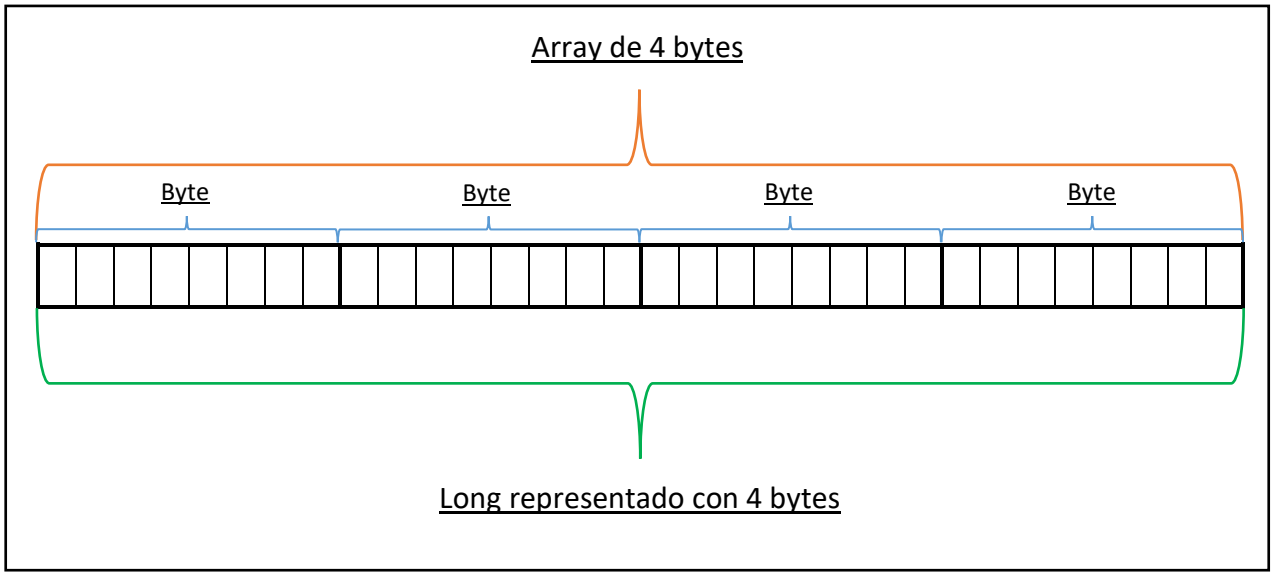

Figura 4.11: Representación gráfica del tipo union en memoria.

Otra complicación usual, que no se suele contemplar, es la aparición de problemas de endianness [13]. Este problema es usual cuando se manipulan datos de tamaño superior a un

byte, entre distintos componentes software ,en bytes independiente; y consiste en que al almacenar o leer estos bytes en memoria se puede hacer de derecha a izquierda o de izquierda a derecha, de manera que, si se lee en el orden inverso en el que se ha almacenado el dato, el valor cambia completamente. En el caso del algoritmo implementado, no se presenta este problema ya que el almacenado de los 4 bytes que componen un entero de 32 bits en el tipo *union* se hace manualmente en cada posición del array en big-endian (esto es, de izquierda a derecha).

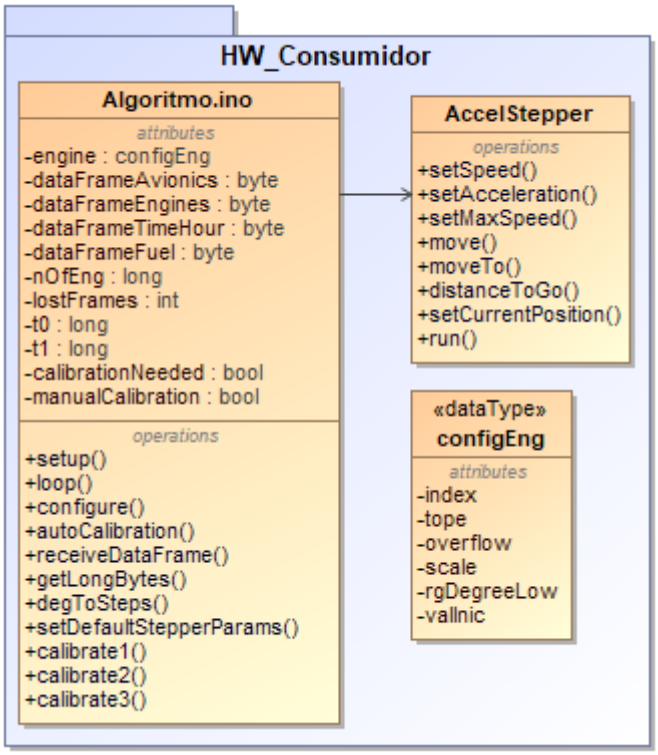

Figura 4.12: Diagrama de clases del módulo HW_Consumidor.

La implementación del módulo con el código para Arduino comienza con la llamada al método **setup()**, en el que se realiza una configuración inicial de los pines, tras la que se llama al método **configure(),** el cual se encarga de esperar pasivamente la llegada de la trama de configuración.

Una vez realizada la configuración se inicia la fase de calibración (si el usuario lo ha seleccionado en el HW_Productor), si la calibración es manual se realiza independientemente en cada motor por los métodos **calibrate1()**, **calibrate2()** y **calibrate3()**, mediante el accionamiento de pulsadores físicos que moverán los motores hasta dejar las agujas en la *posición base*; si la calibración es automática se produce la llamada al método **autoCalibration()**, el cual pone a girar continuamente los motores paso a paso hasta que,

mediante sondeo de un elemento físico (un interruptor *reed* o un sensor óptico infrarrojo) se detecte la posición 0 del sistema, una vez finalizada la calibración se mueven los motores a la posición inicial que debe tener el instrumento representado según la configuración que se ha recibido.

Una vez completa esta inicialización el código llama al método **loop()** el cual implementa la segunda parte del protocolo de comunicación, solicitando una trama actualizada de datos de simulación al HW_Productor. Una vez recibida, se invoca el método **getLongBytes()** (que se usa como auxiliar para acceder a los 4 bytes del parámetro deseado dentro de la trama) y **degToSteps()**, que transforma el valor del parámetro que representa el instrumento a los pasos absolutos que debe dar el motor paso a paso para representar este valor en su instrumento correspondiente según la configuración que tiene el instrumento.

Una vez almacenada la trama de datos más reciente se procede a actuar sobre los motores paso a paso con el método **moveTo()** de la librería AccelStepper que establece la posición en la que se debe encontrar la aguja del instrumento y se invoca al método **run()** de dicha librería, que se usa para mover los motores paso a paso conectados. El código usa **distanceToGo()** para comprobar cuando la aguja de un motor ha llegado a la posición en la que se debería encontrar.

5

Conclusiones, problemas encontrados y futuras líneas relacionadas

Este proyecto propone el desarrollo de un software que permita, de forma sencilla, simplificada y estandarizada, la creación de replicas de instrumentos de vuelo analógicos, usando componentes económicos, para cabinas de simuladores de vuelo. El software se comunica mediante una interfaz con el simulador de vuelo X-Plane, del que recibe parámetros y datos, y tras procesarlos los entrega mediante otro interfaz a un modulo software montado sobre hardware Arduino que se encarga de, usando componentes electromecánicos, replicar el comportamiento del instrumento analógico a la hora de representar esos datos.

El sistema implementado cumple con los objetivos propuestos al principio del proyecto en cuanto a requisitos y funcionalidad. La interfaz implementada para el simulador de vuelo exporta los datos de simulación fuera del simulador de vuelo de forma constante y estandarizada haciendo uso de datagramas TCP/IP usando una dirección Multicast, lo que abre la posibilidad de recibir simultáneamente estos datos en más de una instancia de la aplicación principal, lo que puede ser muy útil para implementar simuladores con múltiples instrumentos analógicos, dando así un equivalente físico de una cabina SimVim.

La aplicación principal proporciona la funcionalidad especificada en los requerimientos, ya que ofrece un catálogo de instrumentos de simulación con una interfaz

amigable, que explica paso a paso como configurar y conectar el sistema para su correcto funcionamiento y, además, permite la personalización de los parámetros de configuración de forma que un mismo valor de los enviados por el simulador de vuelo pueda tener distintas opciones de representación física en instrumentos analógicos.

Finalmente, el protocolo de comunicación y el algoritmo para la placa programable Arduino permite estandarizar la interfaz de comunicación del software con los componentes físicos para cualquier instrumento analógico cuya representación requiera agujas o diales moviéndose en círculos. Al requerirse sólo un envío de tramas muy ligeras por la interfaz USB la carga de trabajo para la placa es poca, lo que permite que el proceso de comunicación sea rápido y constante haciendo así realista el comportamiento del instrumento simulado, cuyo movimiento va a la par con el instrumento virtual correspondiente del simulador de vuelo.
Además, se ha implementado el protocolo de comunicación en modo *"pull"*, controlado por la placa Arduino, de manera que, la carga de trabajo en la placa esté controlada en todo momento y no se produzca sobreescritura de tramas en el Arduino porque los datos de simulación lleguen más rápido de lo que se pueden procesar.
Otro aspecto destacable del sistema es la facilidad de construcción en cuanto a accesibilidad y precio de los componentes, ya que es posible encontrar placas programables Arduino por muy poco dinero, y los motores paso a paso como el 28BYJ-48 (que es el modelo que se ha usado durante el desarrollo del proyecto, ver Figura 5.1) son suficientemente controlables y cuestan menos de dos euros con controladora incluida.

*Figura 5.1: Placa programable Arduino, motor paso a paso 28BYJ-48 y controladora ULN2003. Fuente: "*https://tienda.bricogeek.com/motores-paso-a-paso/969-motor-paso-a-paso-28byj-48-5v-con-driver-uln2003.html*" (5 de septiembre de 2020).*

5.1 Relación con las tecnologías de impresión 3D

Una posibilidad muy interesante ofrecida por el sistema desarrollado en este TFG es que permite simplificar y estandarizar la implementación del sistema electromecánico de simulación de instrumentos analógicos, de forma que la construcción de instrumentos simulados para cabinas realistas pueda limitarse a conseguir el aspecto visual propio de un instrumento analógico haciendo uso de tecnologías de impresión 3D, las cuales no solo permiten obtener la forma y tamaño del instrumento deseado, sino también cambiar el comportamiento del instrumento simulado transformando el movimiento del motor paso a paso en el movimiento de un mecanismo que permita mover, por ejemplo, varias agujas a la vez con un sistema reductor o transformar dicho movimiento circular en lineal.

En particular, mediante el uso de una impresora 3D es posible la creación de los paneles, marcos, biseles, agujas y soportes para que el montaje de los motores paso a paso, y obtener así la apariencia física y comportamiento que tendría un instrumento analógico. En la Figura 5.2 y la Figura 5.3 se presentan algunos ejemplos de lo que es posible conseguir imprimiendo la carcasa de instrumentos analógicos con una impresora de filamento y plástico PLA.

Figura 5.2: Indicador de velocidad vertical impreso en 3D con una impresora doméstica. Fuente "https://www.thingiverse.com/thing:2489322/remixes" (5 de septiembre de 2020).

Figura 5.3: Instrumentos con parámetros de motor impresos con una impresora 3D doméstica. Fuente: "https://www.thingiverse.com/thing:2763499" (5 de septiembre de 2020).

Es importante destacar también la capacidad de personalización y ahorro económico que brinda el uso de la impresión 3D, puesto que puede usarse para construir juegos de engranajes que permitan alterar el modo de operación del motor paso a paso y poder así utilizar motores más económicos que los que serían necesarios sin esos engranajes.

Como ejemplo, se puede crear un sistema reductor o multiplicador para que el movimiento de la aguja se produzca a una velocidad distinta a la que puede dar el motor paso a paso, lo que es especialmente útil cuando se necesita un movimiento de aguja muy rápido y el motor paso a paso que se está empleando no puede proporcionar esas velocidades. En el caso del motor modelo 28BYJ-48 que se ha empleado para el desarrollo del proyecto la velocidad máxima es de 14 vueltas por minuto o un cuarto de vuelta por segundo. Si se requiere una velocidad de giro mayor ya sería necesario, por ejemplo, pasar a un motor 23HS30-2804S cuyo coste es aproximadamente 25 veces el coste del motor usado para este proyecto. Una solución mucho más económica es imprimir un sistema multiplicador que adapte la velocidad del motor usado en este proyecto a una superior, lo que cuesta unos pocos céntimos.

De igual forma, para instrumentos que requieran de varias agujas para su funcionamiento (como pueden ser los altímetros) se puede construir un sistema tal que con el movimiento de un único motor paso a paso se traduzca en el movimiento de dos agujas. Como ejemplo, para un altímetro de tres agujas que representen decenas, cientos y miles de pies, se puede hacer un sistema reductor para que con la aguja de las decenas se muevan las otras como se muestra en la Figura 5.4.

En el caso de movimiento lineal (como los indicadores de viraje o coordinación de giro) se puede transformar el movimiento circular de un motor paso a paso en movimiento lineal para el coordinador de giro mediante un engranaje y una cremallera con dientes como se muestra en la Figura 5.5.

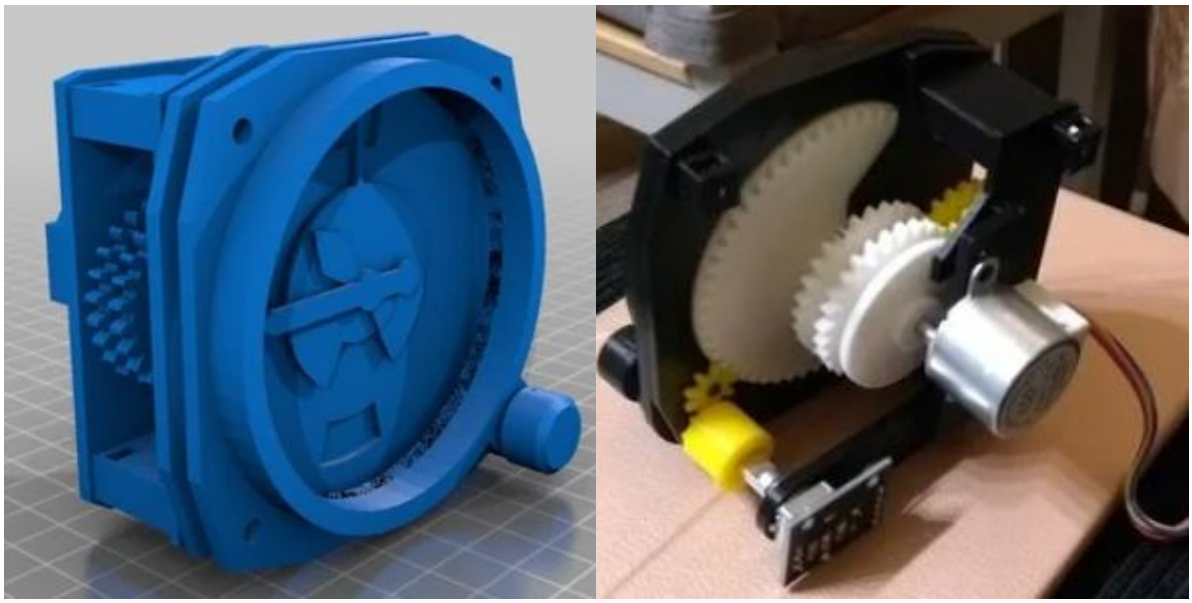

Figura 5.4: Altímetro de tres agujas impreso en 3D con sistema reductor de engranajes. Fuente: "https://www.thingiverse.com/thing:2744359" (5 de septiembre de 2020).

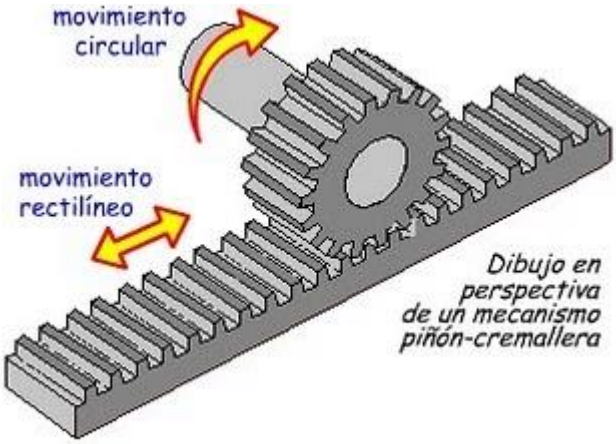

Figura 5.5: Mecanismo engranaje-cremallera para transformar movimiento circular en lineal. Fuente: "https://sites.google.com/site/mecanismos1oima03sap2/elementos-de-maquinas/mecanismo-pinon-cremallera" (5 de septiembre de 2020).

5.2 Problemas encontrados durante la implementación

A lo largo de la implementación se han encontrado varios errores que no se contemplaron en las fases de especificación y diseño o no se implementaron de la manera adecuada. Estos errores se han encontrado en la etapa de pruebas de uso y validación tras cada etapa de desarrollo.

El primer problema surgió testeando la comunicación entre el simulador de vuelo y el programa principal. En la etapa de validación del plugin se crearon pequeños scripts cuya tarea era recibir los parámetros que enviaba el simulador y mostrarlos por consola. La comunicación tenía éxito y la estructura de los datagramas que enviaba el plugin era correcta. Estos pequeños consumidores para el testeo se hicieron usando el lenguaje C, el cual es el mismo que el del plugin. Cuando se implementó el consumidor el C# para el programa principal se utilizó la clase UdpClient y IPendpoint, y aparentemente funcionaba bien, pero cuando se testeó el Requisito no funcional 3 especificado en el capítulo 2.2.2, al iniciar una segunda instancia del consumidor del programa principal este no podía arrancar debido a que el recurso productor se encontraba bloqueado por otra conexión. Tras depurar el funcionamiento se observó que la comunicación no era Multicast, y por lo tanto solo un consumidor podría recibir las tramas que se enviaban desde el plugin. Para solucionarlo se modificó la dirección IP del productor por una Multicast y el consumidor, en vez de asociarse con una sola dirección, escucha de todas las interfaces de red disponibles en la maquina y se suscribe al grupo Multicast cuya dirección es en la que está enviando los datagramas UDP el plugin de X-Plane 10.

El segundo problema detectado se producía en el sistema de detección de desconexión o mal funcionamiento de la placa Arduino. El programa principal detecta un mal funcionamiento si una vez configurada la placa pasan 5 segundos sin recibir ninguna petición de envío de parámetros. Si la sesión en el simulador de vuelo comienza en un aeropuerto con mucha altitud, el altímetro que en su posición inicial está en 0 debe dar tres vueltas para que indique la altitud correcta. Esta inicialización, con el motor paso a paso con el que se ha desarrollado el proyecto requiere al menos 12 segundos; como durante este tiempo no se enviaban peticiones de envío de parámetros mientras se movían los motores, el programa principal detectaba un malfuncionamiento del sistema. Esto se ha solventado cambiando el

tiempo de espera inicial en el programa principal para que a la placa le dé tiempo a mover los motores paso a paso durante la inicialización de los instrumentos del simulador. Tras la recepción de la segunda petición, cuando ya se ha completado la inicialización, el tiempo de Timeout pasa a ser 5 segundos de nuevo.

El tercer problema se detectó en las fases finales de la implementación. Al iniciar una o varias instancias del programa principal todo funcionaba correctamente, pero al cerrarlas, si se prestaba atención a los procesos del sistema operativo se observaba que aún quedaban procesos funcionando en segundo plano creados por la instancia que se acababa de cerrar. Tras realizar la depuración de memoria se observó que las hebras que reciben los parámetros de simulación desde el simulador de vuelo no se cerraban correctamente y se quedaban ejecutándose hasta que se cerraba el simulador de vuelo.
Esto se solucionó cerrando el socket que realiza la comunicación al cerrar la aplicación y por lo tanto cesando la actividad de la hebra.

5.3 Líneas futuras

Con la finalización de este proyecto se resuelve el problema planteado, la construcción de un sistema software para la representación mediante elementos mecánicos de instrumentos analógicos para cabinas de simulación semiprofesionales. Este proyecto puede ser ampliado o modificado en un futuro, en particular, por ejemplo, implementando la interfaz para otros simuladores de vuelo como Prepar3D v5 o Microsoft Flight Simulator 2020.
Otra propuesta interesante puede ser ampliar el sistema para que además de la representación de instrumentación analógica desde el simulador de vuelo hacia elementos físicos, permita conectar elementos analógicos de entrada para completar los paneles de instrumentos de simulación o elementos de control tales como palancas de gases, flaps, switches y botones, de manera que al actuar sobre estos elementos físicamente se produzca la misma acción en el simulador de vuelo. El SDK de X-Plane permite también la entrada de ciertos parámetros de control al simulador por lo que no resultaría muy complicado.

Referencias

[1] «LINK2FS,» [En línea]. Available: http://www.jimspage.co.nz/intro.htm. [Último acceso: 15 septiembre 2020].

[2] «MobiFlight,» [En línea]. Available: https://www.mobiflight.com/en/index.html. [Último acceso: 15 septiembre 2020].

[3] «SimVim,» [En línea]. Available: https://www.mobiflight.com/en/index.html. [Último acceso: 15 septiembre 2020].

[4] «¿Que es el desarrollo iterativo e incremental?,» [En línea]. Available: https://proyectosagiles.org/desarrollo-iterativo-incremental/. [Último acceso: 13 marzo 2020].

[5] ATLASSIAN, «Trello,» [En línea]. Available: https://trello.com/es. [Último acceso: 13 marzo 2020].

[6] «AccelStepper Library,» [En línea]. Available: https://www.airspayce.com/mikem/arduino/AccelStepper/. [Último acceso: 20 Julio 2020].

[7] L. R. «X-Plane SDK Documentation,» [En línea]. Available: https://developer.x-plane.com/sdk/. [Último acceso: 10 Marzo 2020].

[8] Microsoft, «Documentacion de .NET,» [En línea]. Available: https://docs.microsoft.com/es-es/dotnet/. [Último acceso: 10 abril 2020].

[9] M. Á. Álvarez, «¿Que es MVC?,» [En línea]. Available: https://desarrolloweb.com/articulos/que-es-mvc.html. [Último acceso: 7 junio 2020].

[10] «Universal asynchronous receiver-transmitter,» [En línea]. Available: https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter. [Último acceso: 22 septiembre 2020].

[11] E. Gómez, «Cómo funciona el Puerto Serie y la UART,» [En línea]. Available: https://www.rinconingenieril.es/funciona-puerto-serie-la-uart/. [Último acceso: 22 septiembre 2020].

[12] Arduino, «AccelStepper Documentation,» [En línea]. Available: https://www.arduino.cc/reference/en/libraries/accelstepper/. [Último acceso: 22 septiembre 2020].

[13] Microsoft, «Thread Class,» [En línea]. Available: https://docs.microsoft.com/es-es/dotnet/api/system.threading.thread?view=netcore-3.1. [Último acceso: 15 julio 2020].

[14] «C (Lenguage de programación),» [En línea]. Available: https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n). [Último acceso: 20 marzo 2020].

[15] «Arduino,» [En línea]. Available: https://es.wikipedia.org/wiki/Arduino. [Último acceso: 24 mayo 2020].

[16] «C Sharp,» [En línea]. Available: https://es.wikipedia.org/wiki/C_Sharp. [Último acceso: 14 abril 2020].

[17] «C++,» [En línea]. Available: https://es.wikipedia.org/wiki/C%2B%2B. [Último acceso: 14 mayo 2020].

[18] «Reed Switch,» [En línea]. Available: https://es.wikipedia.org/wiki/Reed_switch. [Último acceso: 18 septiembre 2020].

[19] A. A. «Arduino Documentation,» [En línea]. Available: https://www.arduino.cc/en/main/docs. [Último acceso: 14 mayo 2020].

[20] I. Y. M. A. «¿Qué es una memoria EEPROM y cómo funciona?,» [En línea]. Available: https://www.ingenieriaymecanicaautomotriz.com/que-es-una-memoria-eeprom-y-como-funciona/. [Último acceso: 10 julio 2020].

[21] Microsoft, «Lock Statement,» [En línea]. Available: https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/keywords/lock-statement. [Último acceso: 21 junio 2020].

[22] Microsoft, «UdpClient Class,» [En línea]. Available: https://docs.microsoft.com/es-es/dotnet/api/system.net.sockets.udpclient?view=netcore-3.1. [Último acceso: 14 abril 2020].

[23] Microsoft, «Winsock2 Header,» [En línea]. Available: https://docs.microsoft.com/en-us/windows/win32/api/winsock2/. [Último acceso: 17 marzo 2020].

[24] A. Rosario Saavedra, MANUAL DEL PILOTO PRIVADO, Madrid: Pilots, S.A., 1990.

[25] R. Djerf y M. Hammar, «Integrating X-Plane,» Media-Tryck, Lund, 2012.

[26] Microsoft, «Socket Class,» [En línea]. Available: https://docs.microsoft.com/es-es/dotnet/api/system.net.sockets.socket?view=netcore-3.1. [Último acceso: 14 abril 2020].

[27] L. Llamas, «¿QUÉ ES UN DETECTOR DE OBSTÁCULOS IR?,» [En línea]. Available: https://www.luisllamas.es/detectar-obstaculos-con-sensor-infrarrojo-y-arduino/. [Último acceso: 18 septiembre 2020].

[28] I. Sommerville, Ingenieria de software, Madrid: Pearson Educacion, 2005.

[29] D. Lozano Equisoain, Arduino Practico, Madrid: Anaya, 2017.

[30] «X-Plane Datarefs,» [En línea]. Available: http://www.xsquawkbox.net/xpsdk/docs/DataRefs.html.

[31] «Instrumentos de vuelo,» [En línea]. Available: https://es.wikipedia.org/wiki/Instrumentos_de_vuelo. [Último acceso: 18 marzo 2020].

[32] FAA, Instrument Flying Handbook, Oklahoma city: FAA, 2012.

Apéndice A

Manual de uso

Requerimientos:

- Ordenador con capacidad de ejecutar X-Plane 10.
- Simulador de vuelo X-Plane 10.
- Placa Arduino UNO.
- Cable de conexión USB-A a USB-B.
- Motor paso a paso.
- (Recomendado) pulsadores para Arduino.
- (Recomendado) Interruptor *Reed* o sensor infrarrojo.
- Carcasa del instrumento analógico impresa en 3D.
- (Recomendado) Segundo monitor.

Instrucciones de uso:

Colocar el archivo que contiene el plugin de X-Plane en la carpeta de plugins que está dentro de la carpeta de instalación del simulador X-Plane 10. La ruta relativa desde la carpeta de instalación es **/Resources/plugins**. Este paso solo es necesario la primera vez.

Una vez copiado con éxito iniciar el simulador de vuelo y la sesión de simulación.

Cuando el simulador muestre el inicio de la simulación, conectar la placa Arduino al ordenador y ejecutar la aplicación principal "**App.exe**". Se debe mostrar la ventana que muestra la Figura A.1.

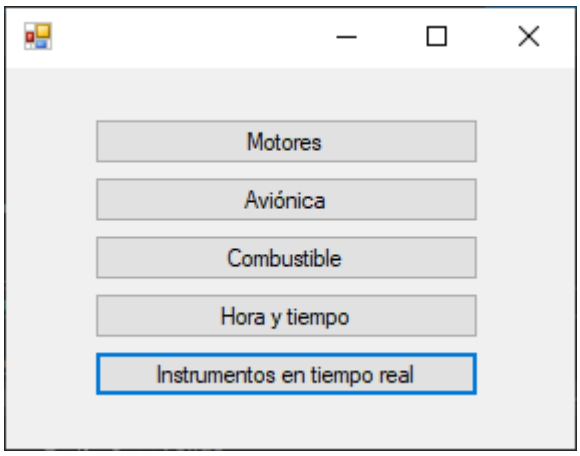

Figura A.1: Pantalla principal de la aplicación.

Aquí podemos elegir mostrar los parámetros de simulación en tiempo real. Aparecen representados por el parámetro que se recibe desde el simulador de vuelo.

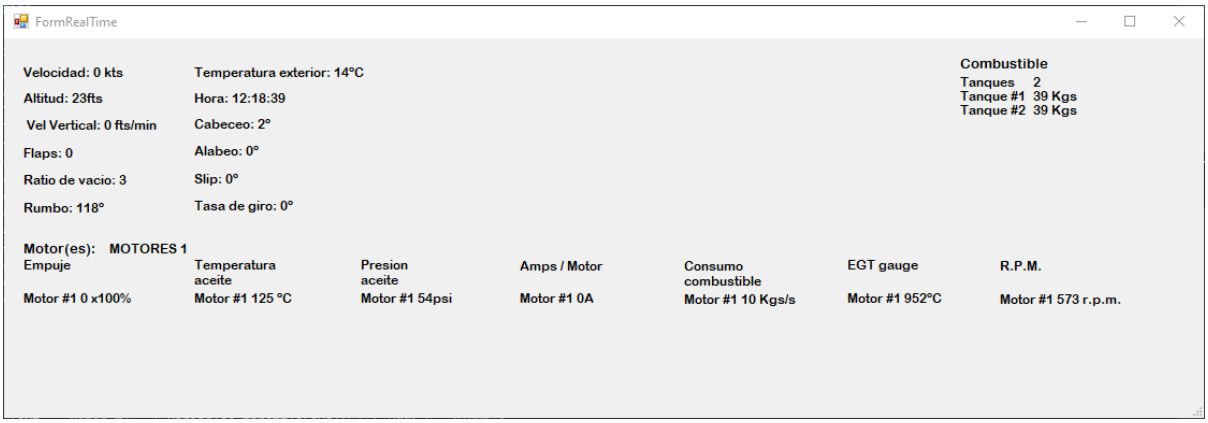

Figura A.2: Ventana con los parámetros de simulación recibidos desde el simulador en tiempo real.

Si por el contrario elegimos iniciar una simulación seleccionando una de las cuatro familias de parámetros de simulación se nos mostrara la ventana asociada. Vamos a seguir este manual seleccionando "Aviónica", pero el proceso es el mismo para todas las familias.

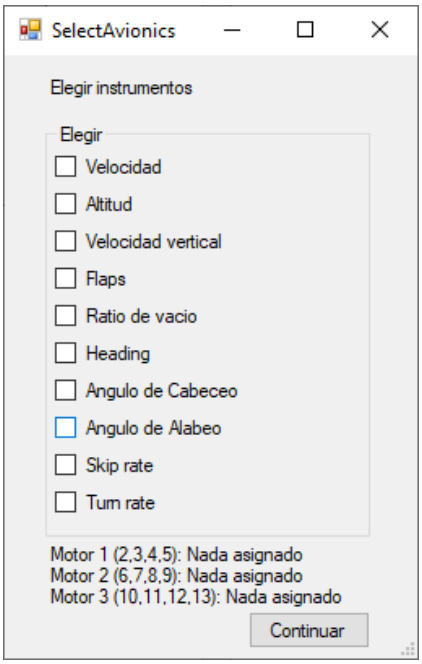

Figura A.3: Ventana de selección de instrumentos de la familia "Aviónica".

Una vez abierta la ventana de selección de instrumentos podremos marcar un máximo de tres instrumentos, que es el número máximo de motores que se pueden conectar a una placa Arduino UNO.

En el caso de superar dicho limite se deshabilitan el resto de instrumentos que no se han seleccionado hasta que hayamos deseleccionado al menos uno.

Nótese que en el área inferior de esta ventana se indica a que motor va a estar asociado cada elemento seleccionado. Siendo el orden de asignación el mismo que el orden de selección y siempre comenzando por el motor conectado a los pines 2, 3 ,4 y 5.

Una vez realizada la selección se abrirá la ventana de configuración de cada instrumento. El contenido de esta ventana es distinto para cada ventana. La aplicación tiene una o varias configuraciones preestablecidas para cada instrumento, pero siempre ofrece la posibilidad de configurar los instrumentos con parámetros de configuración personalizados como se muestra e la Figura A.5.

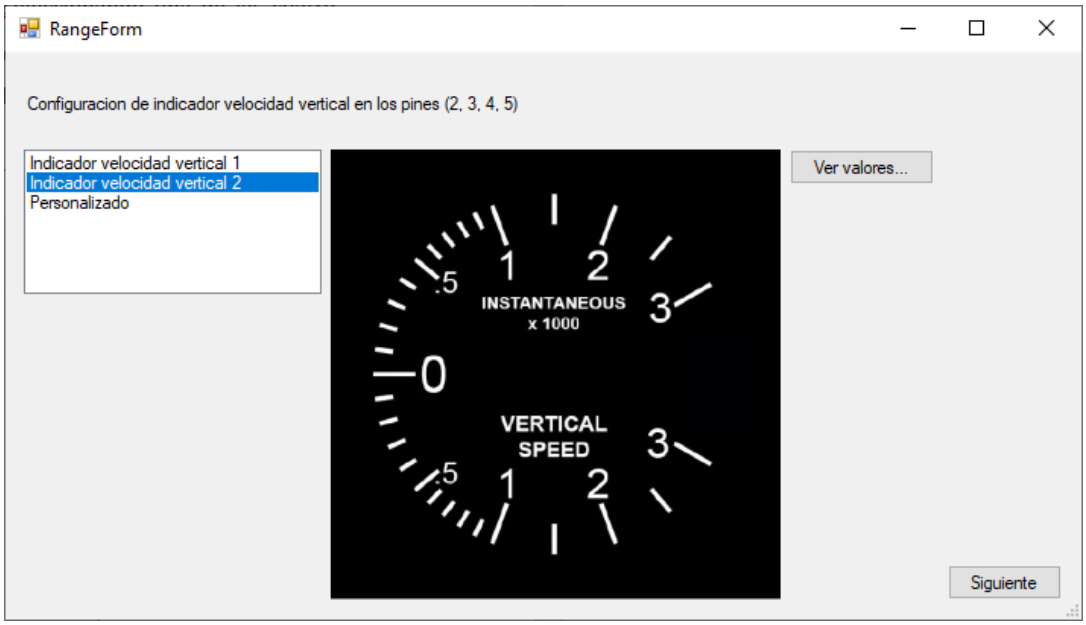

Figura A.4: Ventana de configuración de un indicador de velocidad vertical.

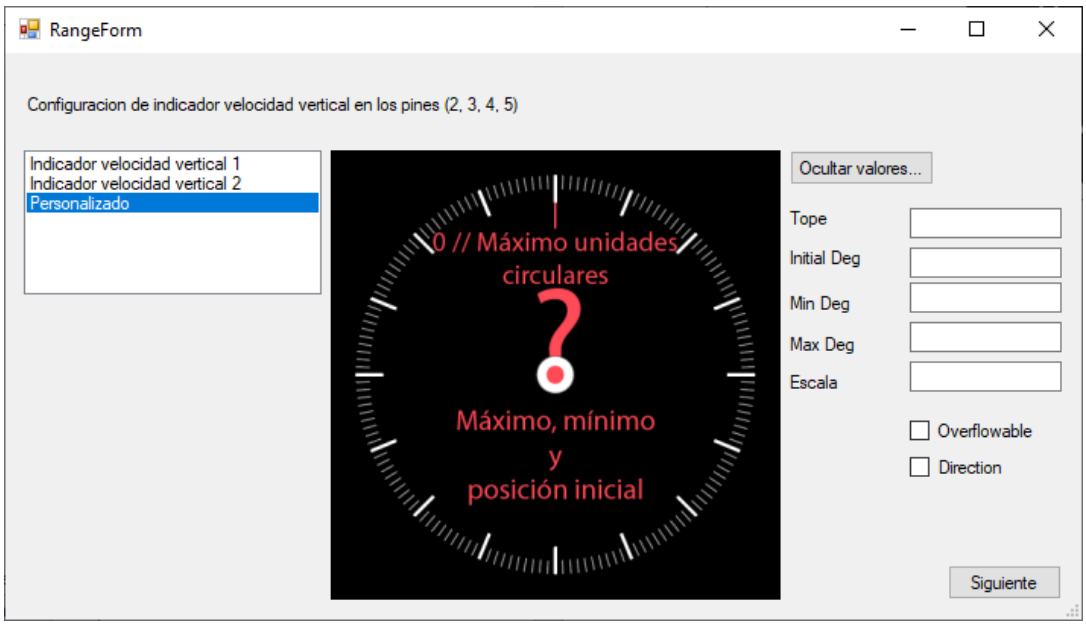

Figura A.5 : Ventana de configuración de un indicador de velocidad vertical personalizado.

Como se puede observar en la Figura A.4 para cada configuración preestablecida se muestra la imagen del indicador asociado a esa configuración. Los valores por defecto no se muestran, aunque hay un botón "Ver valores" que los muestra y permite su medicación.

Para realizar la configuración manual se debe seleccionar "Personalizado" en la lista tal y como se muestra en la Figura A.5. Se mostrará la lista de campos a rellenar. Para conseguir el funcionamiento deseado de giro de un instrumento hay que introducir:

- **El tope o unidades circulares**. Es el equivalente al 12 o 0 de un reloj de agujas. Si el valor indicado en este campo es 360, habría que dar 360 pasos en circulo para llegar a la misma posición.
- **La posición inicial** o número de pasos acorde al valor introducido anteriormente para que la aguja alcance la posición que tiene el instrumento cuando la aeronave esta parada. Esto sería 0 para un altímetro cuya aguja apunta hacia arriba o el número de pasos para que la aguja se mueva 90 grados a la izquierda para un indicador de velocidad vertical.
- **Los valores mínimo y máximo** dentro de los cuales la aguja podrá moverse. Esto en instrumentos que pueden dar vueltas completas y seguir girando es innecesario, pero para un indicador de velocidad o un indicador de giro podremos configurar cual es el tope de un dos o grados máximos permitidos por el indicador.
- **La escala** es un valor que no está relacionado con el instrumento analógico sino con la manera que tiene de almacenar y tratar los parámetros de simulación la aplicación. En vez de tratar con números decimales se almacenan como números de entero con factor de escala. Normalmente la escala es 1000 de manera que podemos tener la precisión de 3 decimales, pero hay parámetros para los que no se escala. Esta información se puede configurar en el Apéndice B.
- **Overflow** indica si llegado a alguno de los topes la aguja puede seguir girando o no.
- **Dirección** sirve para que el movimiento de la aguja ante valores positivos sea a la izquierda en vez de a la derecha como es normal. Esto permite configurar instrumentos como el horizonte artificial con los ángulos de cabeceo y alabeo.

Una vez introducidos los parámetros de configuración o haber elegido una de las establecidas por la aplicación se selecciona "Continuar". Si se hubiesen seleccionado varios instrumentos se repetirá el paso anterior tantas veces como instrumentos se hayan seleccionado.

Finalmente se muestra la pantalla para empezar la comunicación con la placa Arduino tal y como se muestra en la Figura A.6.

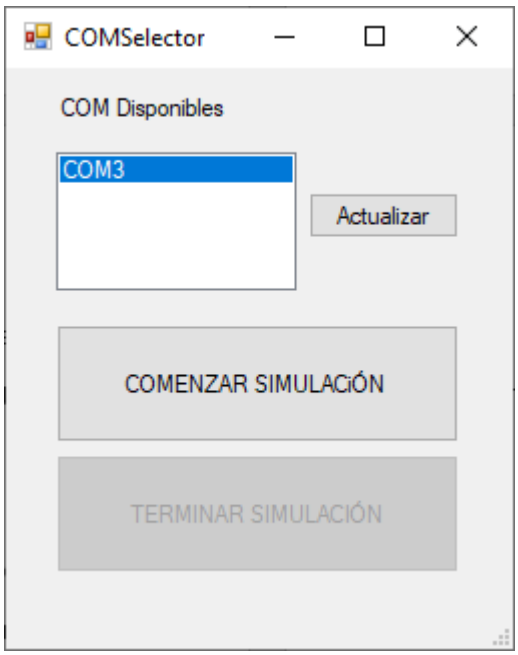

Figura A.6: Ventana de inicio de comunicación con la placa Arduino.

En esta ventana se nos muestran los puertos a los que hay conectados una placa Arduino. Pudiendo en todo momento actualizar la lista para el botón dedicado a ello o iniciar la simulación una vez seleccionado un puerto.

Al comenzar la simulación se nos preguntara si queremos realizar la calibración. En caso afirmativo también se nos preguntara si de manera manual, para lo que se usaran los pulsadores, y los elementos electrónicos para la calibración automática. Una vez indicado todo esto se nos muestra el diagrama de conexión que indica como debemos conectar los componentes con la placa Arduino para el correcto funcionamiento tal y como se muestra en la Figura A.6.

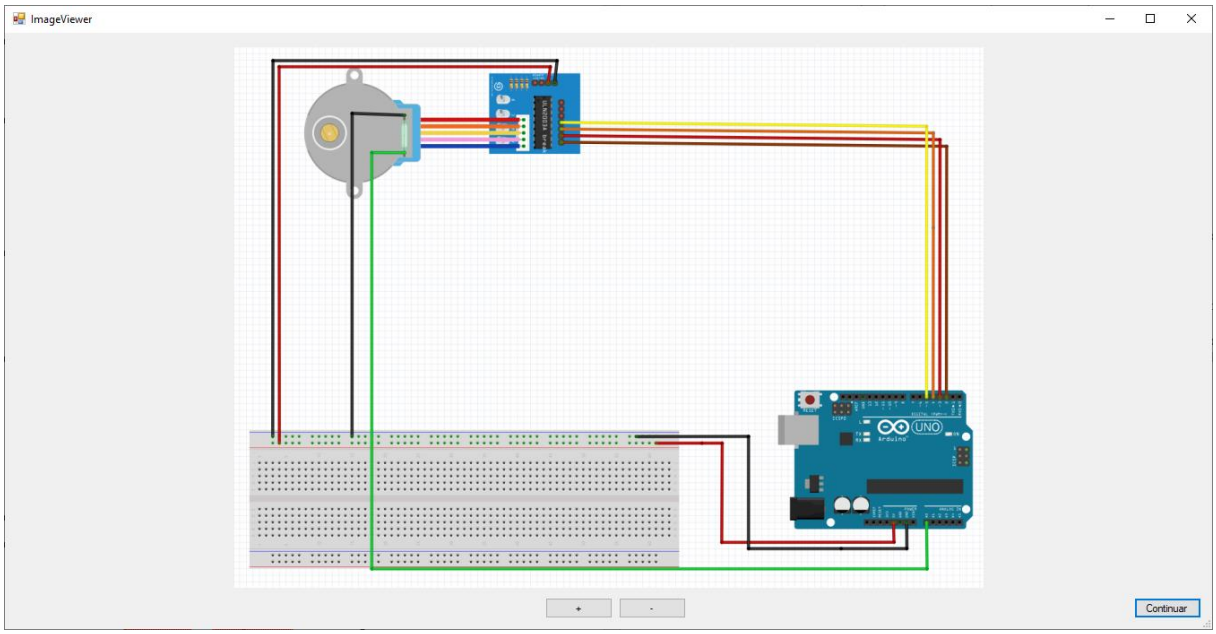

Figura A.7: Ventana que muestra el diagrama de conexión que debemos seguir para que el sistema funcione.

La imagen con el diagrama de conexión se puede ampliar si hay algún detalle que no sea visible de manera clara. Este diagrama es personalizado para cada configuración que haya seleccionado el usuario y siempre mostrara el total de elementos que debe conectar y a donde los debe conectar.

Finalmente, solo hay que pulsar "Continuar" y se inicia la comunicación y simulación con la placa Arduino que hayamos seleccionado.

Si durante la simulación quisiéramos detenerla manualmente se puede hacer con el botón que se ha habilitado ahora se habría habilitado para ello. Este botón es el que en la Figura A.6 aparece desactivado.

Apéndice B

Parametros de simulacion

Aunque la lista de parametros con los nombres de sus DataRefs se pueden conultar en:
"http://www.xsquawkbox.net/xpsdk/docs/DataRefs.html"

Mostramos a continuación la tabla con los nombres, tipos y DataRefs empleados:

Parámetro	Tipo	Nombre DataRef	Escala
Altura	float	sim/cockpit2/gauges/indicators/altitude_ft_pilot	1
Velocidad en nudos	float	"sim/cockpit2/gauges/indicators/airspeed_kts_pilot"	1000
Velocidad vertical	float	"sim/cockpit2/gauges/indicators/vvi_fpm_pilot"	1
Tanques combustible	float	"sim/aircraft/overflow/acf_num_tanks"	1000
Nivel combustible	float[9]	"sim/cockpit2/fuel/fuel_quantity"	1000
Posición flaps	float	"sim/cockpit2/controls/flap_ratio"	1000
Número de motores	float	"sim/aircraft/engine/acf_num_engines"	1000
Nivel de potencia	float[8]	"sim/cockpit2/engine/actuators/throttle_ratio"	1000
Temperatura aceite	float[8]	"sim/cockpit2/engine/indicators/oil_temperature_deg_C"	1000
Amperímetro	float[8]	"sim/flightmodel/engine/ENGN_bat_amp"	1000
Presión aceite en PSI	float[8]	"sim/flightmodel/engine/ENGN_oil_press_psi"	1000
Consumo instantáneo	float[8]	"sim/flightmodel/engine/ENGN_FF_"	1000
Vacío	float	"sim/cockpit/misc/vacuum"	1000
Rumbo	float	"sim/cockpit2/gauges/indicators/heading_electric_deg_mag_pilot"	1000
Temperatura exterior	float	"sim/weather/temperature_ambient_c"	1000
Hora local	float	"sim/time/local_time_sec"	1000
Angulo de cabeceo	float	sim/cockpit2/gauges/indicators/pitch_electric_deg_pilot"	1000
Angulo de alabeo	float	"sim/cockpit2/gauges/indicators/roll_electric_deg_pilot"	1000
Angulo de giro	float	"sim/cockpit2/gauges/indicators/slip_deg"	1000
Angulo deslizamiento	float	"sim/cockpit2/gauges/indicators/turn_rate_heading_deg_pilot"	1000
EGT motores	float	"sim/flightmodel/engine/ENGN_EGT_c"	1000
RPM motores	float	"sim/cockpit2/engine/indicators/engine_speed_rpm"	1000

Tabla B.1: Parámetros de simulación usados en la aplicación y su nombre de referencia DataRef.

Apéndice C

Diagramas de conexión

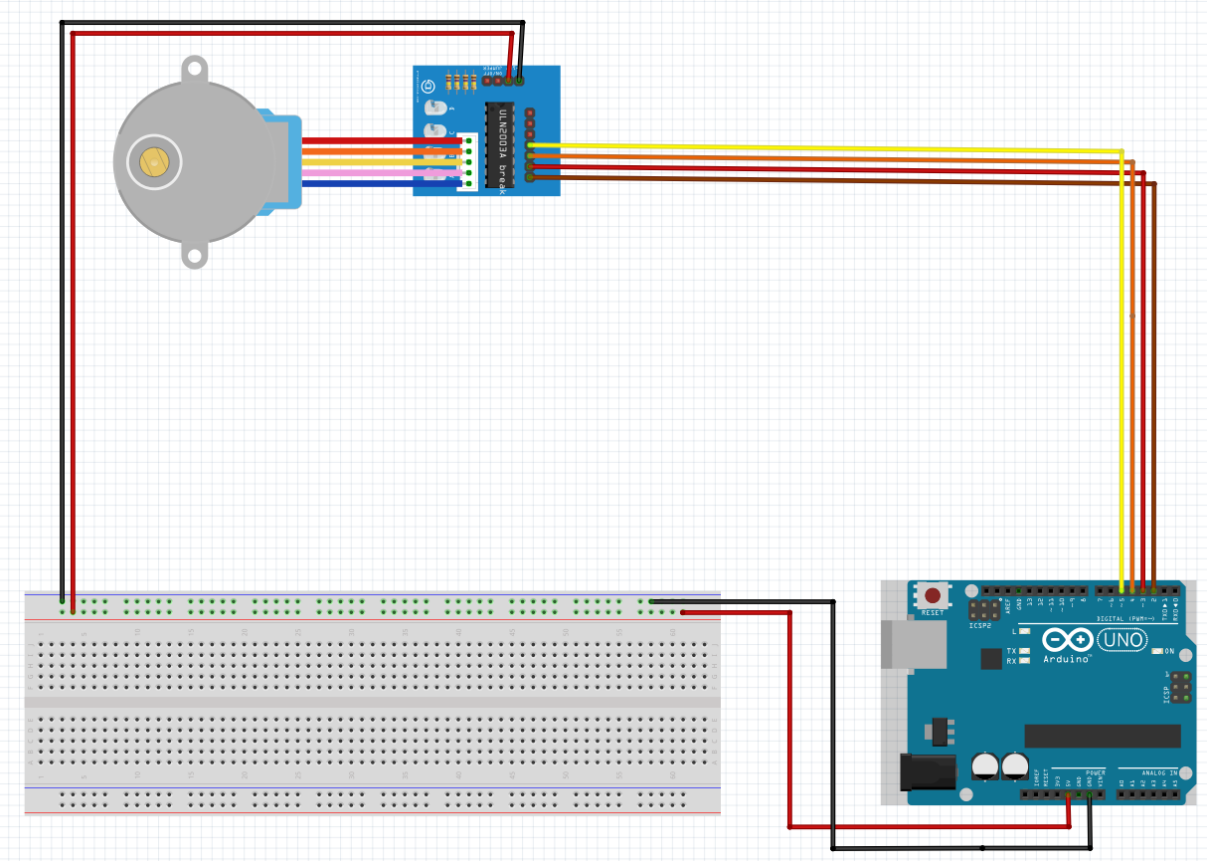

Figura C.1: Diagrama de conexión para un motor sin calibración requerida.

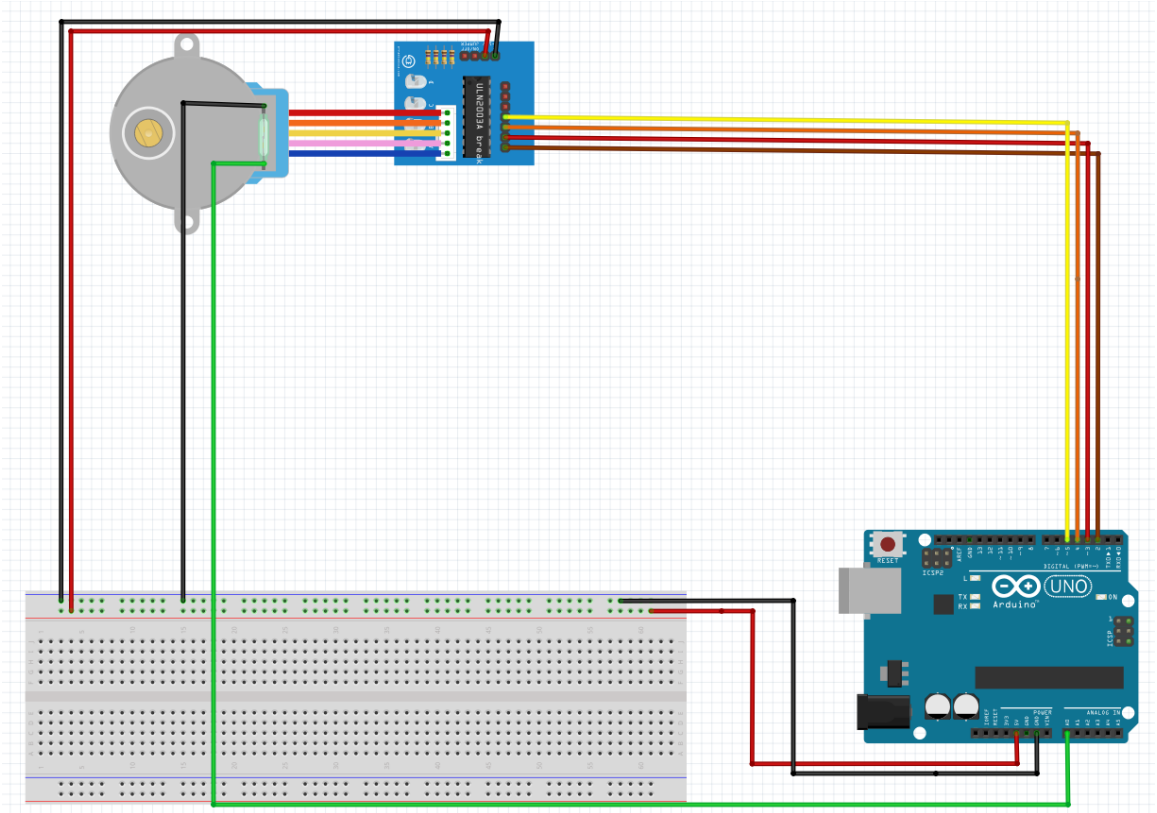

Figura C.2: Diagrama de conexión de un motor con calibración por interruptor magnético o reed.

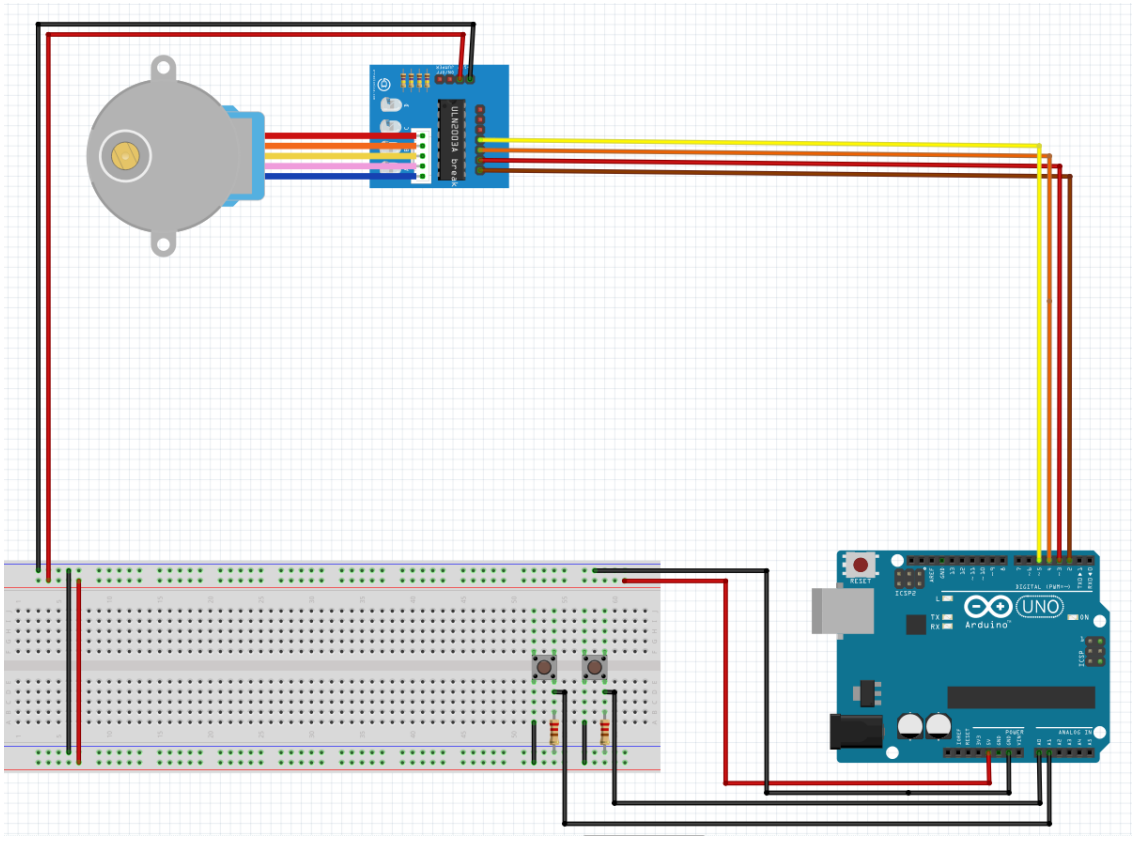

Figura C.3: Diagrama de conexión de un motor con calibración por pulsadores.

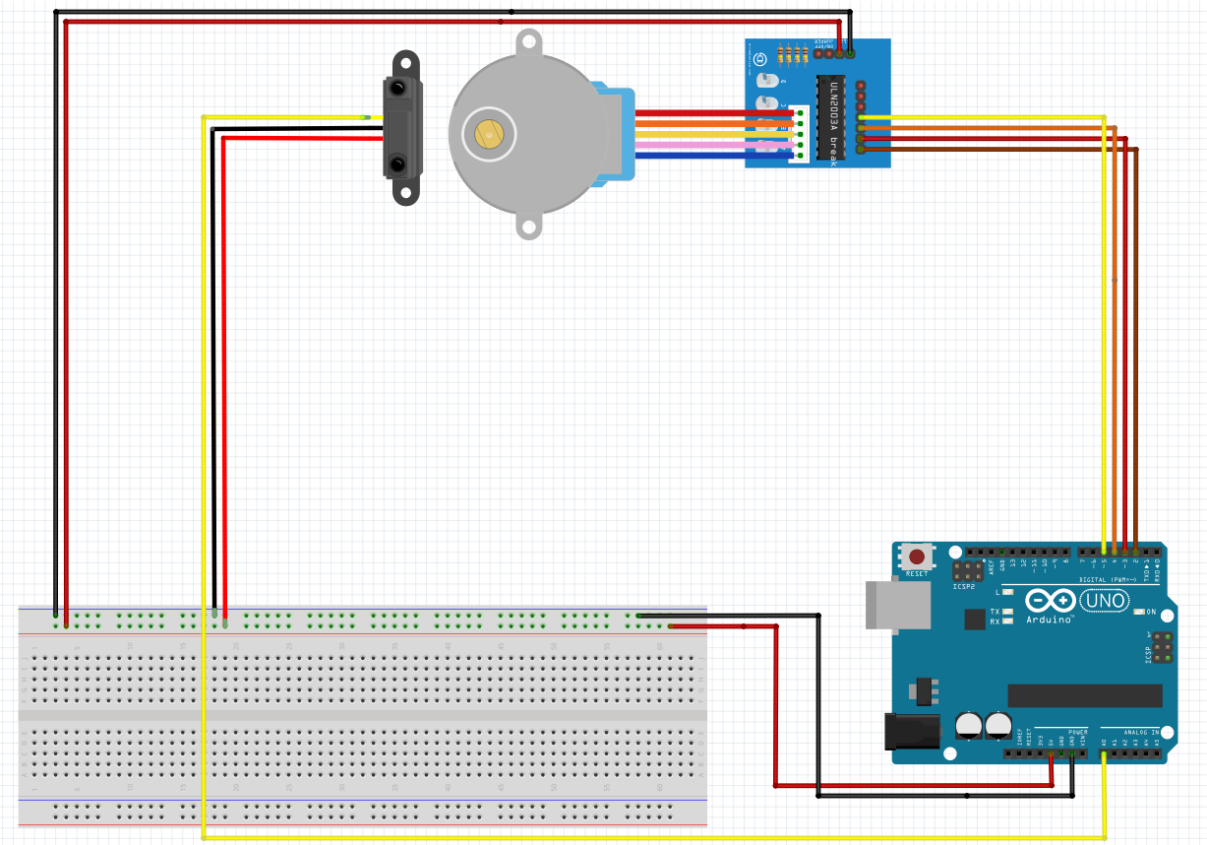

Figura C.4: Diagrama de conexión de un motor con calibración por sensor infrarrojo.

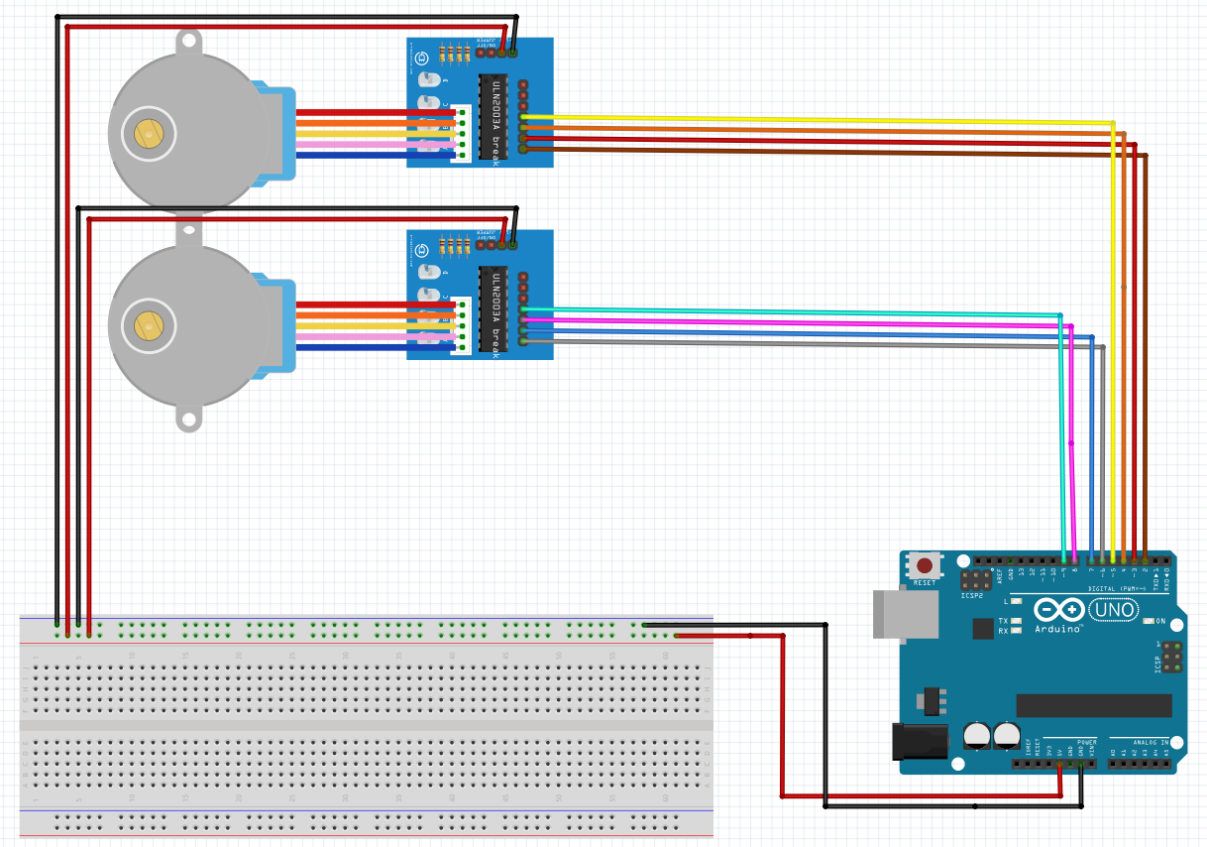

Figura C.5: Diagrama de conexión de dos motores sin calibración requerida.

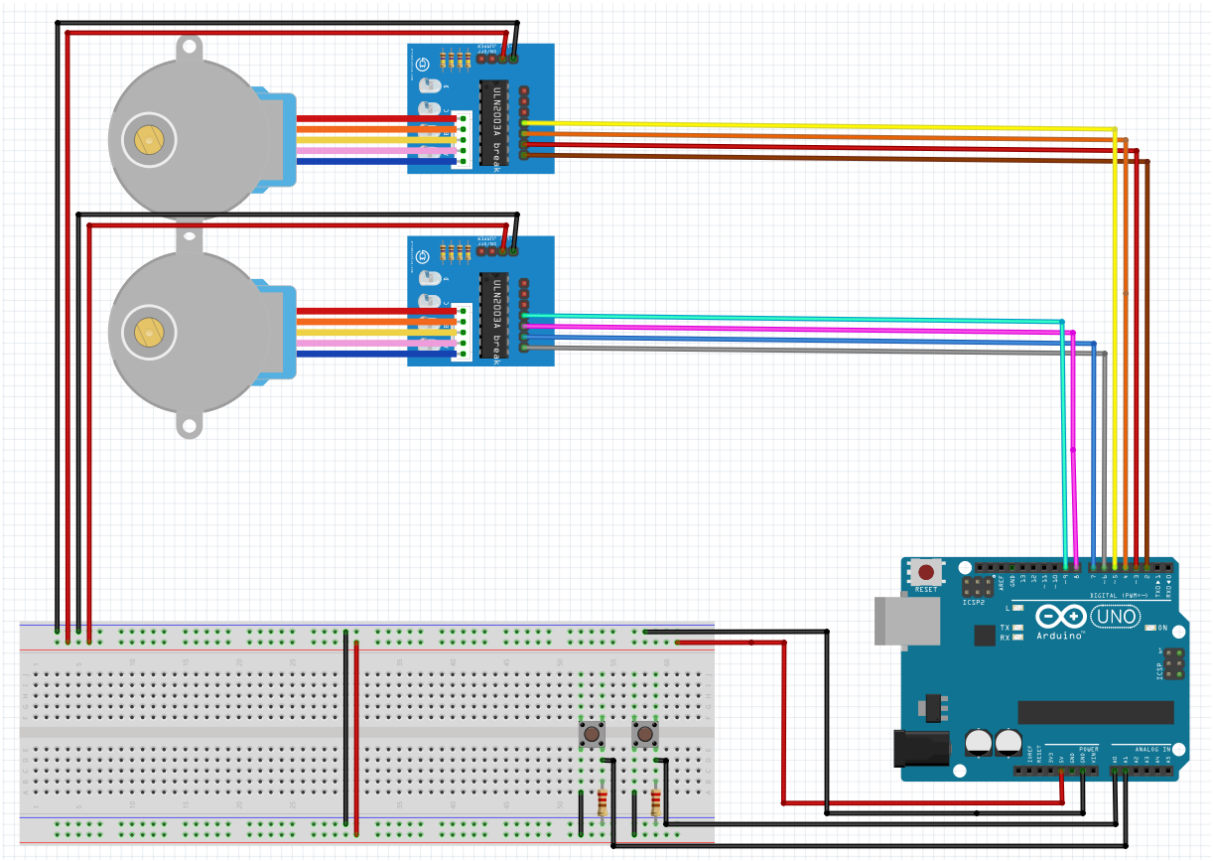

Figura C.6: Diagrama de conexión de dos motores con calibración por pulsadores.

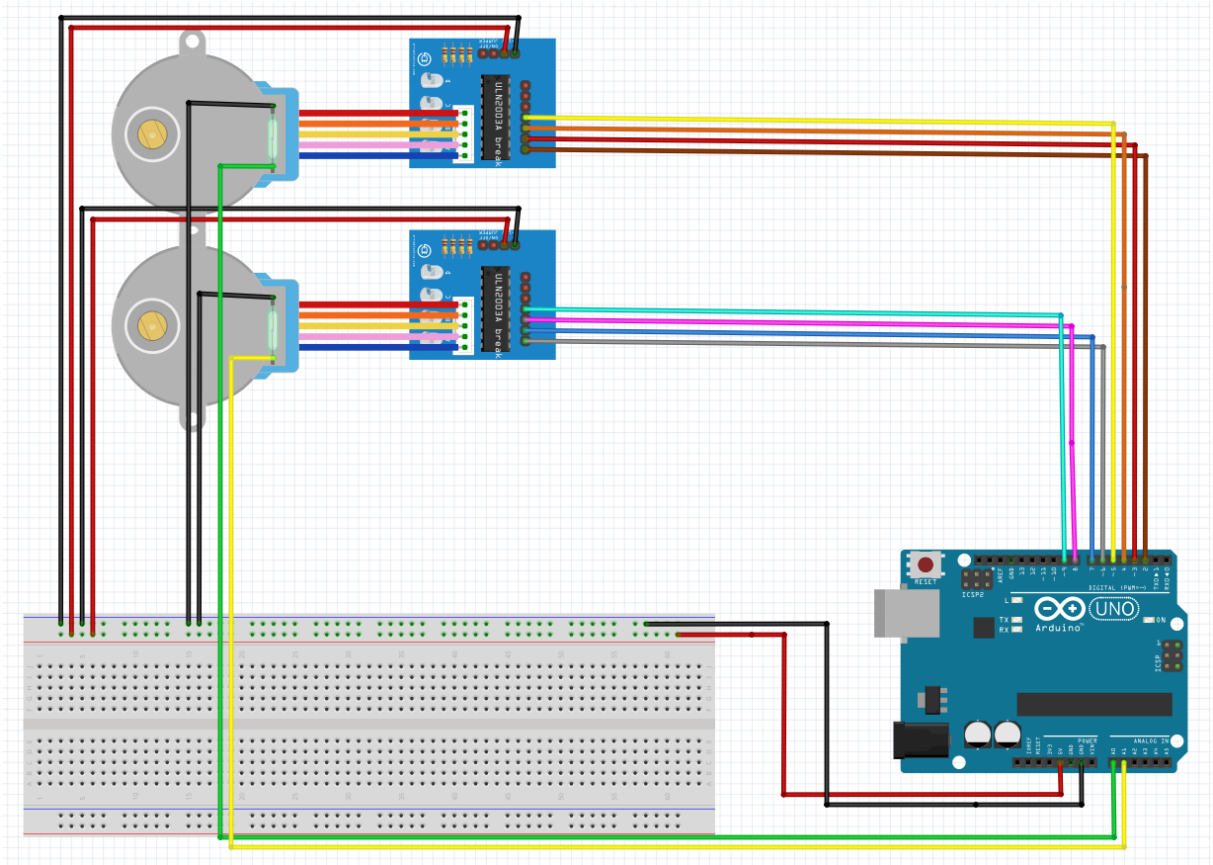

Figura C.7: Diagrama de conexión de dos motores con calibración por interruptor magnético o reed.

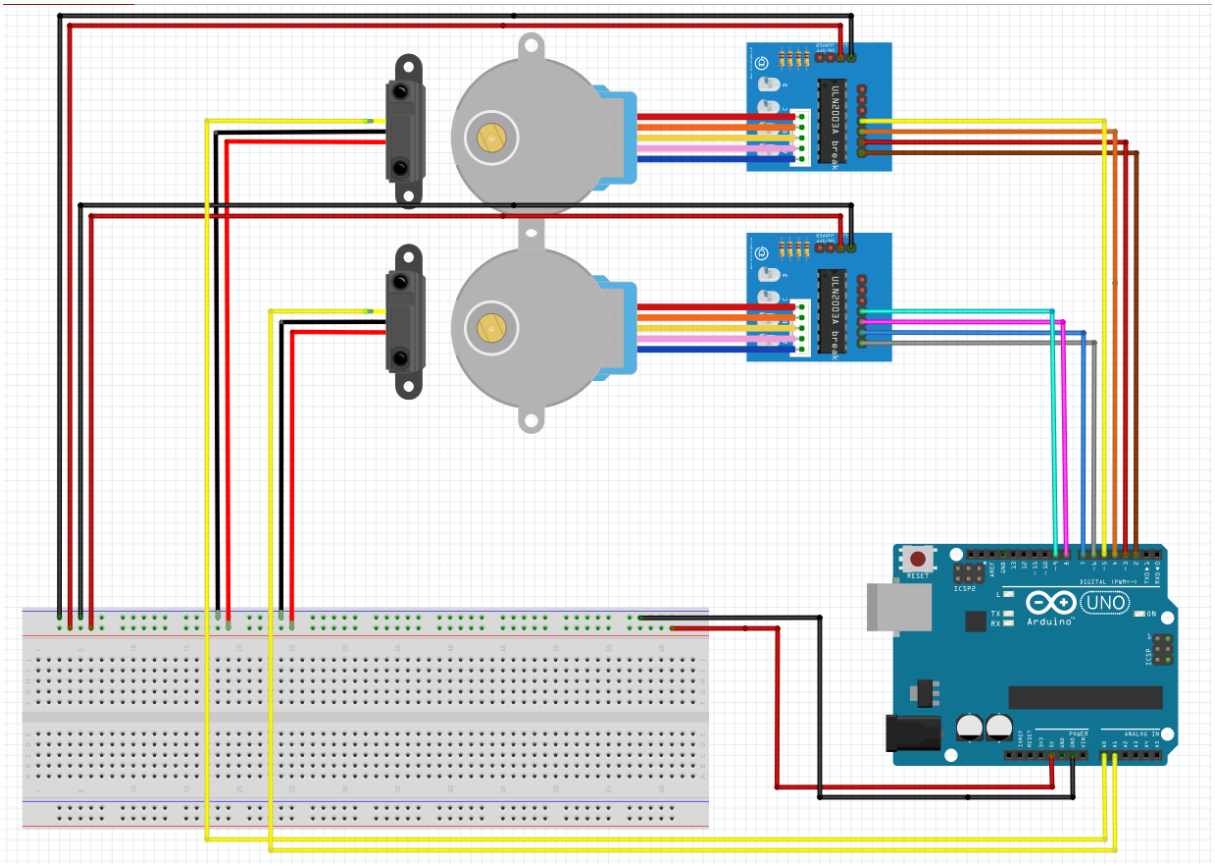

Figura C.8: Diagrama de conexión de dos motores con calibración por sensor infrarrojo.

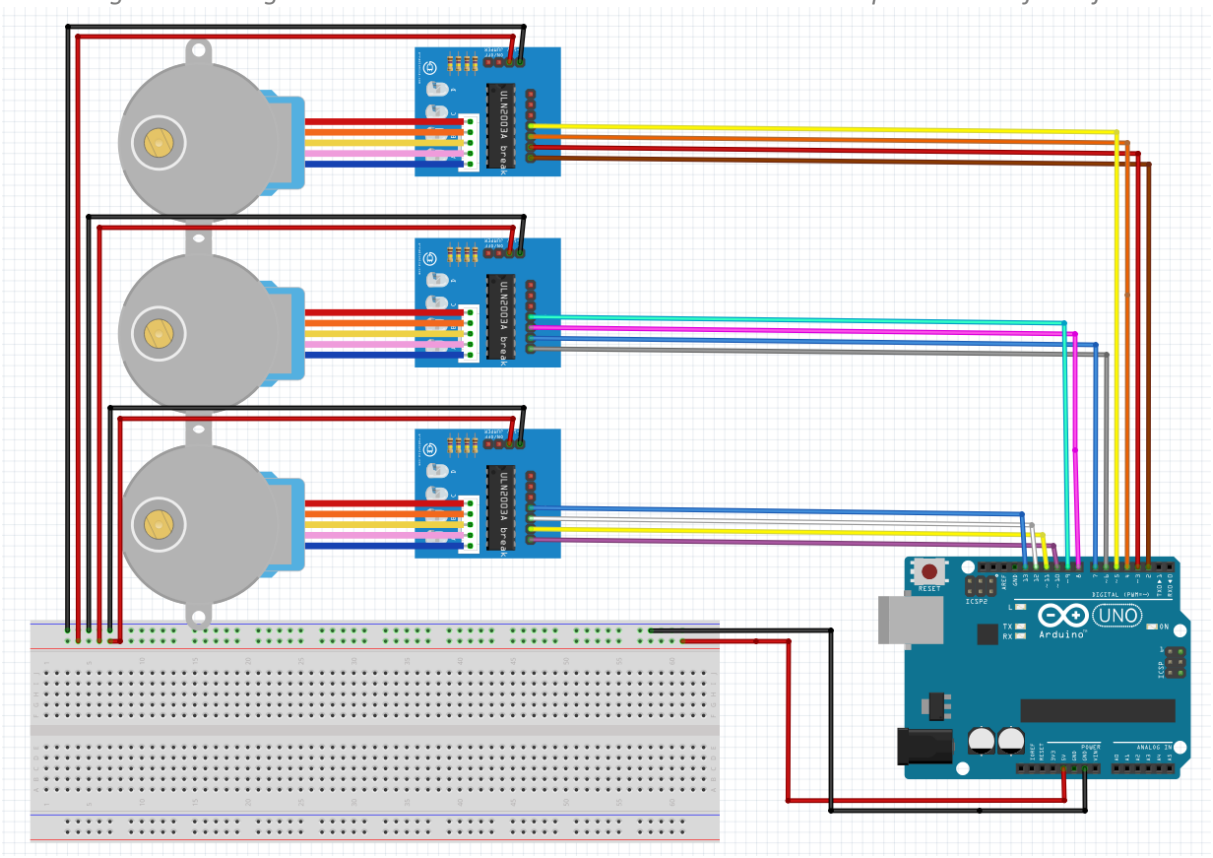

Figura C.9: Diagrama de conexión de tres motores sin calibración requerida.

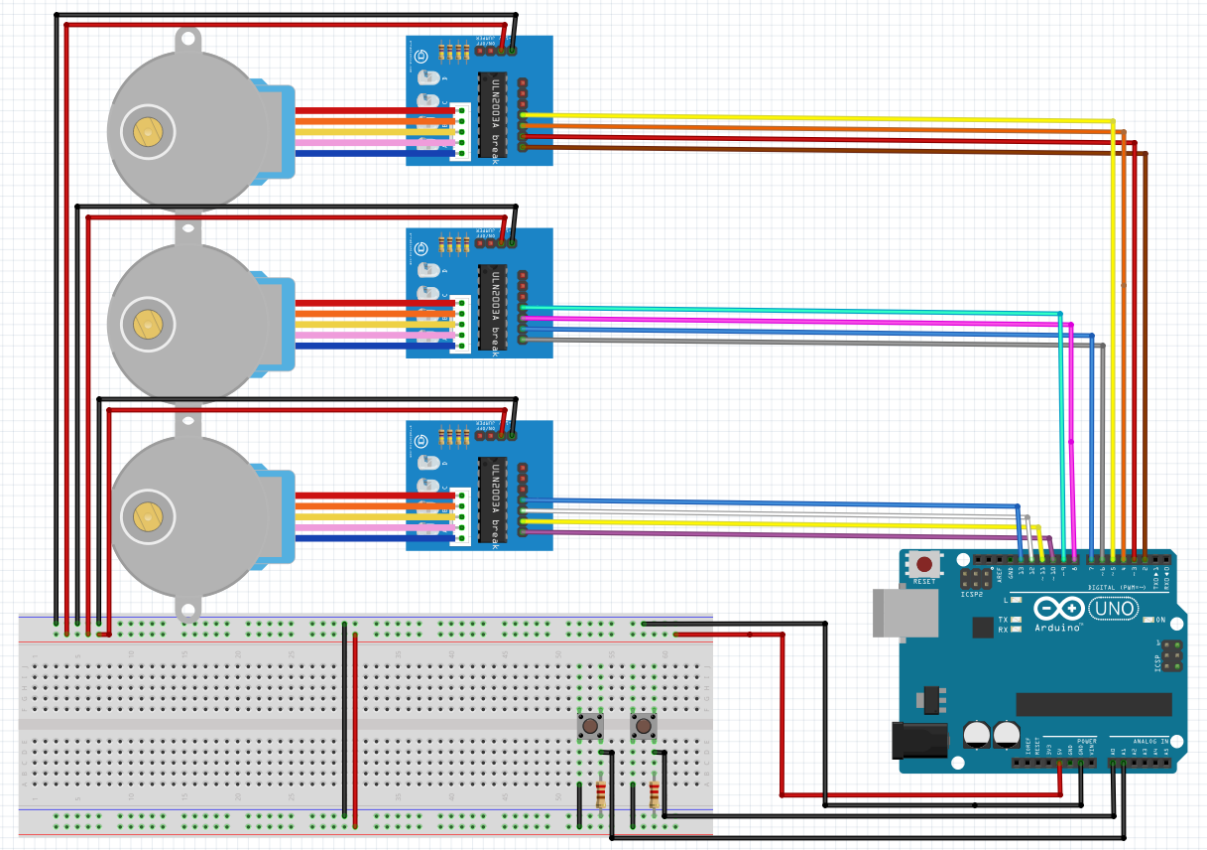

Figura C.10: Diagrama de conexión de tres motores con calibración por pulsadores.

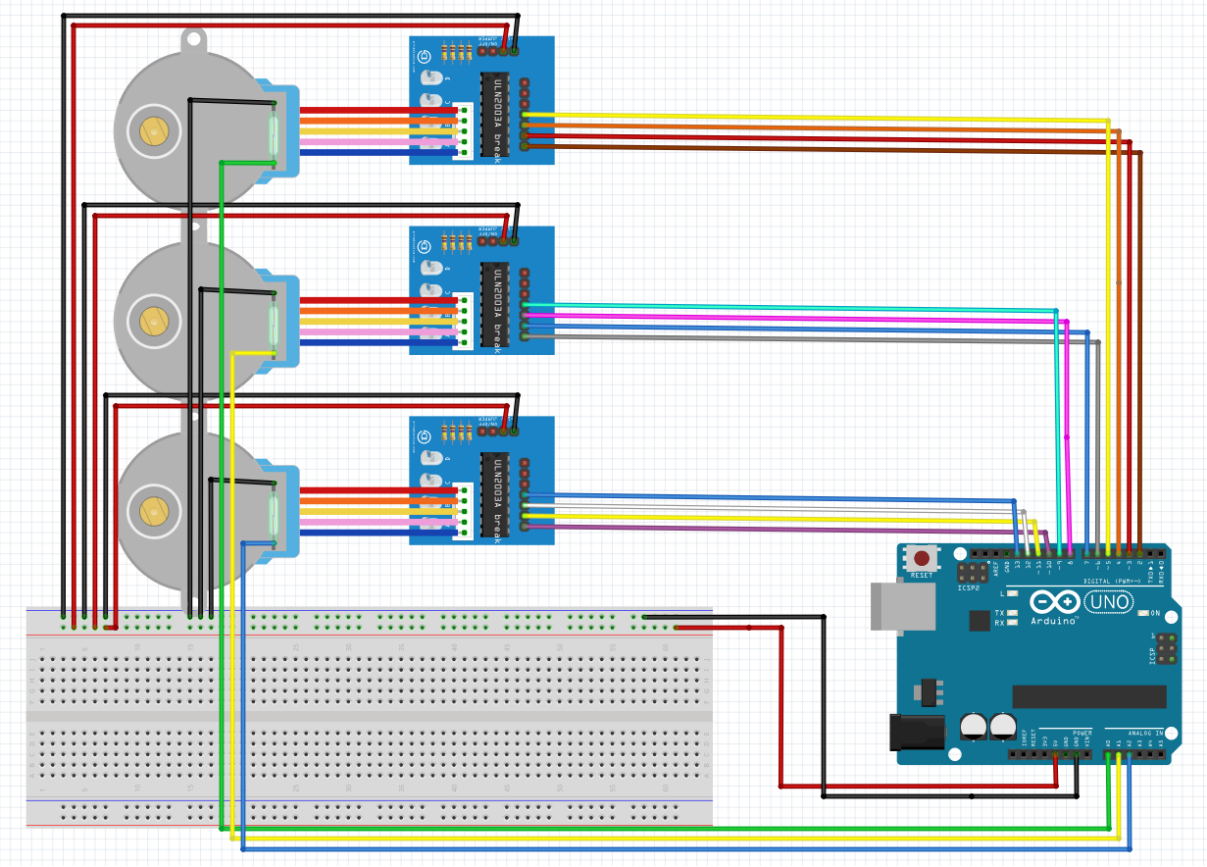

Figura C.11: Diagrama de conexión de tres motores con calibración por interruptor magnético o reed.

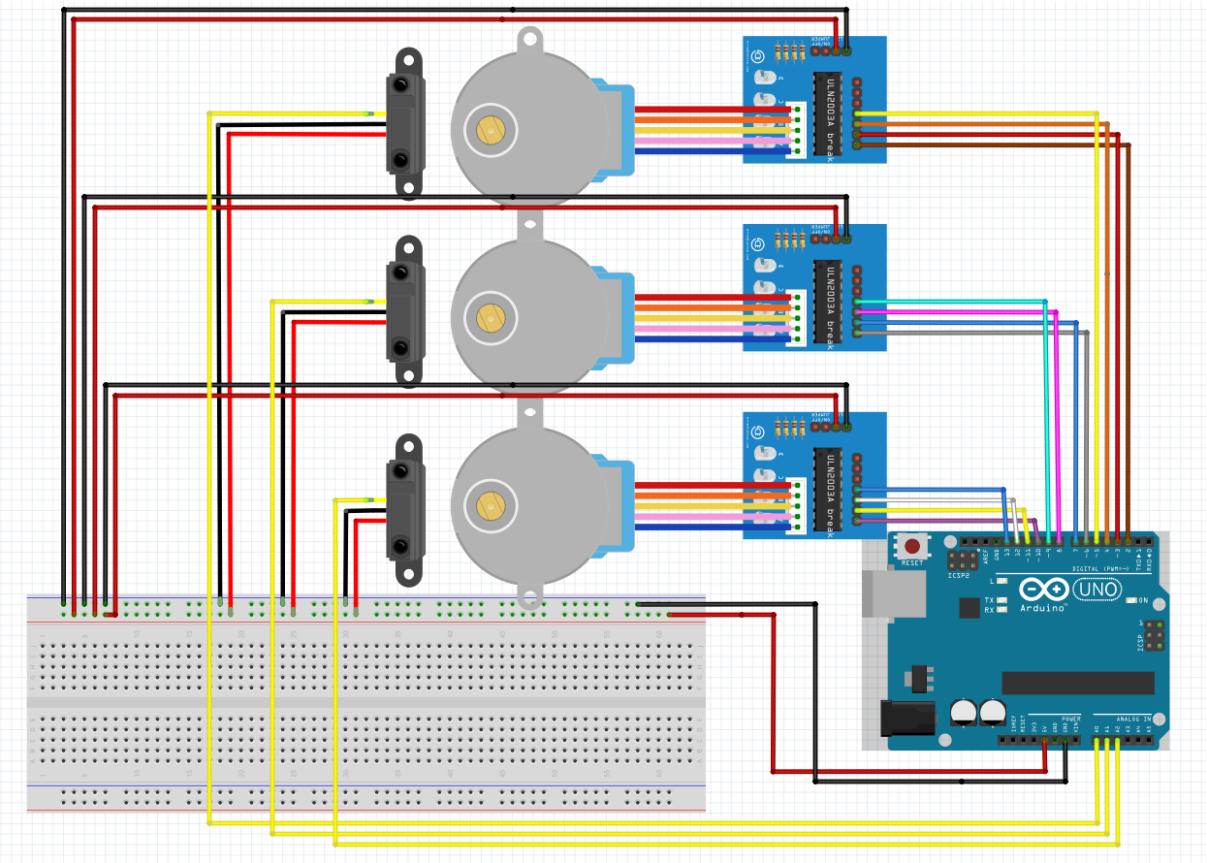

Figura C.12: Diagrama de conexión de tres motores con calibración por sensor infrarrojo.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA