UNIVERSIDAD DE GRANADA

TRABAJO DE FIN DE GRADO
INGENIERÍA INFORMÁTICA

Aplicación de control de un faro de vehículo para demostrador

Creación de un dispositivo portátil capaz de guardar las señales PWM que componen los faros de automóviles de alta gama para el desarrollo e instalación a pie de campo de estos.

Autor
Fº Rubén Sánchez Aguilera

Director
Andrés Roldán Aranda

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, 13 de noviembre de 2023

Aplicación de control de un faro de vehículo para demostrador

Creación de un dispositivo portátil capaz de guardar las señales PWM que componen los faros de automóviles de alta gama para el desarrollo e instalación a pie de campo de estos.

Autor
Fº Rubén Sánchez Aguilera

Director
Andrés Roldán Aranda

Aplicación de control de un faro de vehículo para demostrador

Fº Rubén Sánchez Aguilera

Palabras clave:

Interfaz gráfica, señales PWM, almacenamiento, Arduino, navegación, C, python, QT Designer.

Resumen:

La industria de la automoción es un amplio sector en el que se desarrollan y mejoran constantemente nuevos equipamientos tecnológicos, uno de estos equipos son los faros de los vehículos.

Este proyecto crea un sistema portátil basado en Arduino MEGA, equipado con una pantalla LCD y un menú interactivo operado por rotary encoders. Permitiendo almacenar y ajustar las señales PWM que controlan los faros de nuevas generaciones de vehículos. Adicionalmente, se ha desarrollado una aplicación con interfaz gráfica de usuario, permitiendo a los usuarios modificar las señales PWM de manera más sencilla desde sus computadoras.

Vehicle headlamp control application for demonstrator

Fº Rubén Sánchez Aguilera

Key words:

Graphic interface, PWM signals, storage, Arduino, navigation, C, Python, QT Designer.

Resumen:

The automotive industry is a vast sector where new technological equipment is constantly being developed and improved, one of these pieces of equipment are the vehicle headlights.

This project creates a portable system based on Arduino MEGA, equipped with an LCD screen and an interactive menu operated by rotary encoders. It allows for the storage and adjustment of the PWM signals that control the headlights of new generations of vehicles. Additionally, an application with a graphical user interface has been developed, enabling users to more easily modify the PWM signals from their computers.

D. Andrés María Roldán Aranda, Profesor del departamento de Electrónica y Tecnología de los Computadores de la Universidad de Granada, como director del Trabajo Fin de Grado de D. Francisco Rubén Sánchez Aguilera,

Informa:

Que el presente trabajo, titulado ***Aplicación de control de un faro de vehículo para demostrador***, ha sido realizado bajo su supervisión por **Francisco Rubén Sánchez Aguilera**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 13 de noviembre de 2023.

Fdo: Andrés María Roldán Aranda

Agradecimientos

En primer lugar, quiero agradecer a mis amigos por el inmenso apoyo, honestidad y vitalidad que me han ofrecido en mi vida. También quiero hacer especial mención a los compañeros de mi etapa universitaria, que además de amigos, han sido un gran apoyo necesario para solventar todos los obstáculos enfrentados gracias a su incansable espíritu de compañerismo.

También quiero agradecer a mi familia, por la oportunidad que me han brindado y lo orgulloso que estoy de ellos.

Quiero agradecer también a mi tutor su dedicación y apoyo en la realización de este proyecto Fin de Grado. Sus consejos basados en la experiencia y en su amplio conocimiento me han servido para desarrollar este proyecto de la mejor manera.

Y por último quiero agradecerme a mi mismo, que pese a todos los altibajos que he sufrido, nunca he tirado la toalla y siempre he seguido a delante.

Acknowledgments

First of all, I would like to thank my friends for the immense support, honesty and vitality they have offered me in my life. I would also like to make special mention of my university classmates, who, in addition to being friends, have been a great support necessary to overcome all the obstacles I have faced thanks to their tireless spirit of companionship.

I would also like to thank my family for the opportunity they have given me and how proud I am of them.

I would also like to thank my tutor for his dedication and support in the completion of this Final Degree Project. His advice based on his experience and extensive knowledge has helped me to develop this project in the best way possible.

And finally I would like to thank myself, that despite all the ups and downs I have suffered, I have never given up and have always kept going.

Índice

Tabla de contenido

Índice de figuras.

Lista de acrónimos.

Durante el desarrollo de la memoria utilizaré una serie de acrónimos:

- **Arduino MEGA 2560:** Es el microcontrolador que se utiliza en este proyecto, capaz de llevar a cabo todas las instrucciones y guardar en memoria las diferentes configuraciones.

- **Pantalla LCD:** Pequeña pantalla de 4x16 caracteres en la que se muestra un menú con las diferentes opciones que se pueden utilizar.

- **Rotary encoder:** Dispositivo que convierte el movimiento de un eje para poder desplazarse en el menú mostrado en el LCD.

- **Memoria EEPROM** (EEPROM, s.f.)**:** (**Electrically Erasable Programmable Read-Only Memory**) Una memoria de tamaño reducido que puede ser programada y donde se puede guardar y cargar las señales PWM de los faros.

- **QT designer**: QT Designer es una aplicación que permite diseñar interfaces gráficas de usuario (GUI) de manera visual, facilitando la creación de aplicaciones con ventanas, botones y otros elementos interactivos.

- **C:** El lenguaje C es conocido por su eficiencia y capacidad de bajo nivel. Se utiliza para escribir código de bajo nivel y aplicaciones que requieren un alto grado de control sobre el hardware.

- **Python:** Python es un lenguaje de programación de alto nivel conocido por su legibilidad y facilidad de uso. Se utiliza para desarrollar una amplia variedad de aplicaciones, desde scripts simples hasta aplicaciones de inteligencia artificial.

- **Puerto serie:** Un puerto serie es una interfaz de comunicación que permite la transferencia de datos secuenciales entre dispositivos. Se utiliza comúnmente para la comunicación entre un ordenador y otros dispositivos, como microcontroladores.

- **Señales PWM:** Las señales PWM son señales digitales utilizadas para controlar la potencia de dispositivos electrónicos, como motores o luces. Varían la duración del pulso para ajustar la cantidad de energía entregada.

- **Memoria Flash:** La memoria flash es un tipo de memoria no volátil utilizada para almacenar datos en dispositivos electrónicos. Es rápida, duradera y se utiliza en dispositivos como memorias USB, tarjetas de memoria y microcontroladores.

- **Software:** El software es un conjunto de programas y datos que permiten a un ordenador o dispositivo realizar tareas específicas. Incluye aplicaciones, sistemas operativos, controladores y otros programas.

- **atoi():** una función de la biblioteca estándar de C/C++ que se utiliza para convertir una cadena de caracteres que representa un número en una variable de tipo entero

- **BaudRate:** El "baud rate" (tasa de baudios) es una medida de la velocidad de transmisión de datos en una comunicación serial. Indica la cantidad de símbolos o bits transmitidos por segundo en una línea de comunicación.

- **ISR:** Interrupt Service Routines, permite a un microcontrolador responder rápidamente a ciertos eventos, como la entrada de señales externas o cambios de estado internos, pausando temporalmente el programa principal para atender estos eventos.

Bloque 1: Descripción del proyecto

1. Introducción

En esta primera descripción del proyecto se presentará el trabajo en cuestión, también se expondrán los argumentos de por qué se llegó a este punto, se expondrá el problema a resolver, se explicarán los conceptos y palabras claves a tener en cuenta y se enumerarán los objetivos a alcanzar. logrado al final del proyecto. Además, se considerará la planificación temporal y finalmente se describirá la estructura de la memoria.

1.1 Motivación

En la actualidad, la industria automotriz es un enorme sector donde se trabaja con multitud de componentes, los cuales deben de estar en constante desarrollo y evolución. Estos componentes pueden variar desde el motor del coche, una puerta o un faro.

Siempre he sido un amante de este sector y estoy especializado en la rama de ingeniería de los computadores, que se trabaja con sistemas empotrados, lenguaje de bajo nivel, etc.

Por lo que al encontrar este proyecto lo tuve claro, tenía que crear un equipo completamente nuevo que permitiese a las empresas tener un sistema portátil donde poder guardar todas las configuraciones y parámetros que existen de los diferentes faros de alta gama

1.2. Objetivos

El objetivo del proyecto es facilitar a las empresas un sistema portátil capaz de albergar la información de faros de vehículos de alta gama, por lo cual, se busca crear un sistema sencillo posible de entender para cualquier trabajador de una empresa y que contenga toda la información relacionada con las señales PWM que componen un

faro, y que pueda ser modificada dinámicamente para poder probar diferentes configuraciones.

1.3. Requisitos

Una vez visto los objetivos, salen a la luz los principales requisitos que tendremos en el proyecto:

1. Realizar un sistema portátil compacto y robusto.
2. Implementar un menú en el que se pueda navegar a través de él para elegir diferentes funcionalidades.
3. Capacidad de modificar las propiedades de las señales PWM que componen los faros, las cuales son: Nombre, Frecuencia, Fase, Modo y Duty cycle.
4. Introducción de una contraseña que se pueda modificar para controlar el acceso.
5. Poder almacenar las propiedades de las señales PWM en una memoria no volátil.
6. Añadir un nombre que compone las 8 señales PWM.
7. Diseñar una interfaz gráfica en ordenador que permita tanto la carga y guardado de todas las propiedades de las señales PWM y su modificación.

Estos requisitos serán las tareas principales que se desarrollarán a lo largo de esta memoria.

1.4. Estructura del documento

El documento está dividido en tres grandes bloques claramente diferenciados:

El bloque 1, donde se realiza una breve descripción del proyecto, así como el objetivo de este y los principales problemas que nos encontraremos a la hora de abordarlo, también se explicará el estado con el que se encontró el proyecto, los

requisitos funcionales que el cliente pidió y que tenemos que tener en cuenta antes del bloque 2.

El bloque 2, desarrollo del proyecto. Se tratarán aspectos como el diseño de clases e implementación, en el inicio de cada capítulo se explicará resumidamente la funcionalidad de cada archivo y cómo realiza su función.

En el bloque 3 se explicarán las conclusiones a la que se ha llegado una vez terminado el proyecto, así como futuras mejoras que se pueden incluir en este.

2. Estado del arte (ingeniería inversa).

Este es un producto que ha encargado la empresa VALEO, al laboratorio de GranaSAT, siendo un producto completamente nuevo y que no existe en el mercado, el proyecto lo inició el alumno Luis Sánchez, estudiante de la Escuela Técnica Superior de Ingeniería Informática y Telecomunicaciones (ETSIIT), desarrollando todo el apartado electrónico, creando el PCB y diferentes mecanismos que explicaré a continuación.

En el proceso de desarrollar el producto, me enfrenté a la necesidad de comprender y modificar el código del programa que previamente había desarrollado Luis. Esta etapa de ingeniería inversa implicó desglosar y analizar el código existente para entender su funcionamiento y realizar las adaptaciones necesarias.

Este fue un paso muy importante y llevó más tiempo de lo planeado realizarlo ya que Luis Sánchez era ingeniero electrónico y no un programador, por lo que había bastante falta de documentación y algunas "ideas felices" que me resultaron difícil de comprender y se tuvieron que modificar.

En esta sección se describe el proceso que se siguió para llevar a cabo la ingeniería inversa.

2.1. Análisis de Código Fuente

El primer paso crucial en el proceso de ingeniería inversa fue examinar los archivos existentes e identificar los objetivos de cada uno de ellos. Esto implica abrir y revisar los archivos de código que formaban parte del programa de Arduino.

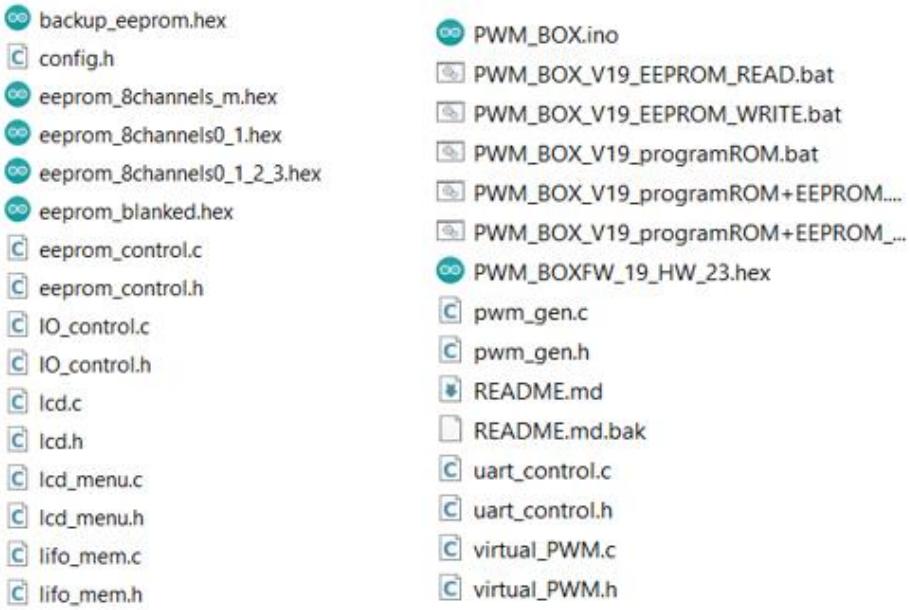

***Figura 1**: Todos los archivos que componen el proyecto*

Después de realizar este paso llegué a la conclusión de que los archivos más importantes eran:

- **Config.h:** En este archivo se encuentran todas las variables, declaraciones y parámetros que necesita el arduino, LCD y Rotary Encoders para poder funcionar.

- **Eeprom_control**: Este archivo se encuentran las estructuras de datos y funciones necesarias para poder guardar y cargar datos en la memoria eeprom del arduino, este archivo tuvo que ser rediseñado entero ya que no funcionaba adecuadamente.

- **IO_control y lifo_menu:** En este archivo están declaradas las funciones para poder manejar las entradas y salidas al producirse una interrupción de la ISR.

- **lcd:** Declaraciones y variables necesarias para que la pantalla LCD pudiera funcionar.

- **lcd_menu**: Este era el archivo más extenso, aquí se controlaba el menú que aparecía en la pantalla del LCD, existiendo dos opciones, uno para el menú principal y otro para el menú de configuración de las señales PWM.
- **PWM_BOX:** El programa principal donde se encuentra el main.
- **pwm_gen y virtual_PWM**: Todas las estructuras y funciones necesarias para poder implementar las señales PWM que componen los faros de los automóviles, aquí es donde hizo un gran trabajo Luis Sánchez.
- **UART_control:** En este archivo se declaran las funciones y estructuras necesarias para poder simular la comunicación serial, ya que este no se podía utilizar debido a las interrupciones para manejar el Rotary Encoder, este paso llevó mucho tiempo debido a que tuve que estudiar el datasheet del arduino MEGA para poder entender los registros que se utilizaban, además se tuvo que modificar para añadir diferentes funcionalidades.

Los demás archivos eran para poder realizar guardados y cargados de datos de las señales PWM previamente definidas.

Quedando este diagrama de componentes para poder ver más claramente en proyecto.

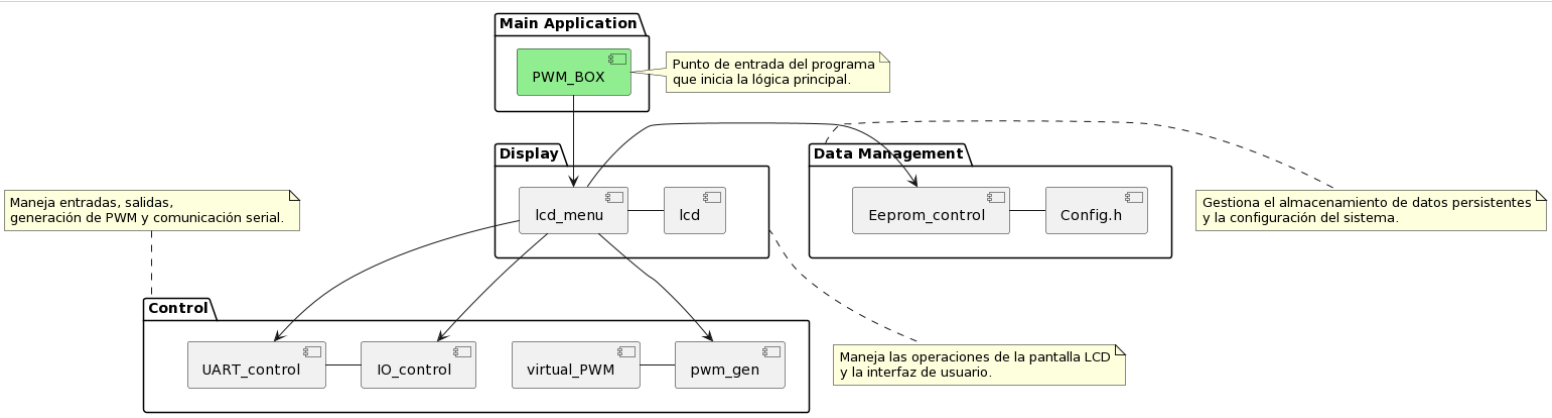

Figura 2: Relación entre los archivos

3. Metodología del desarrollo (ágil, s.f.)

Algo muy importante antes de desarrollar cualquier proyecto es pensar bien la metodología que se va a seguir.

Existen diversas metodologías para la gestión de proyectos, como la metodología en cascada que es la más habitual, pero no es recomendable para este tipo de proyecto, ya que no se prueba el software hasta que una gran parte de él está terminada. Por este motivo busqué una metodología en la que se pudiera ir modificando y actualizando parte del código a medida que se avanza en el proyecto, encontrando finalmente la metodología ágil.

La metodología ágil se basa en la flexibilidad y capacidad de modificar los productos a lo largo del proyecto. Esta metodología divide el proyecto en fases de muy corta duración, el resultado de las cuales es un producto con una serie de funcionalidades que ya permiten que este sea usado. Estas fases se componen por 3 sencillos pasos:

1. **Planificar (Plan)**: Esta es la fase donde se identifican las necesidades del cliente, se establecen las metas para el sprint siguiente y se planifican las tareas para alcanzar esos objetivos.

2. **Desarrollo (Develop)**: En esta etapa, se empieza a construir las características del producto basándose en los requerimientos recogidos y planificados. El desarrollo se lleva a cabo en sprints, que son periodos cortos de tiempo durante los cuales se completa un conjunto predefinido de funciones.

3. **Diseño (Design)**: Aunque se coloca después del desarrollo en el ciclo, en la práctica, el diseño ocurre a menudo al mismo tiempo que el desarrollo. El diseño puede implicar arquitectura de software, diseño de la experiencia del usuario (UX), y la planificación de cómo implementar funcionalidades específicas.

4. **Lanzamiento y prueba** En nuestro proyecto el lanzamiento y prueba se puede juntar, ya que el producto está en desarrollo y no está lanzado al mercado. Se realizan pruebas para asegurar que el producto o las características funcionan como se esperaba y que se cumplen los estándares de calidad. Las pruebas pueden ser automatizadas o manuales y suelen ser continuas durante el ciclo de desarrollo.

5. **Revisión (Review)**: En esta fase, el equipo y, a menudo, los clientes revisan lo que se ha construido, proporcionando retroalimentación que puede usarse para mejorar el producto en el próximo sprint.

6. **Despliegue (Deploy)**: La funcionalidad completa y probada se despliega en el entorno de producción donde los usuarios finales pueden utilizarla.

Esta metodología era ideal para mí ya que al tener que compaginar este proyecto con una jornada laboral completa, me permitía realizar poder realizar un avance con un requisito, terminarlo, comprobar que el resultado fuese correcto y parar durante un tiempo para no saturarme.

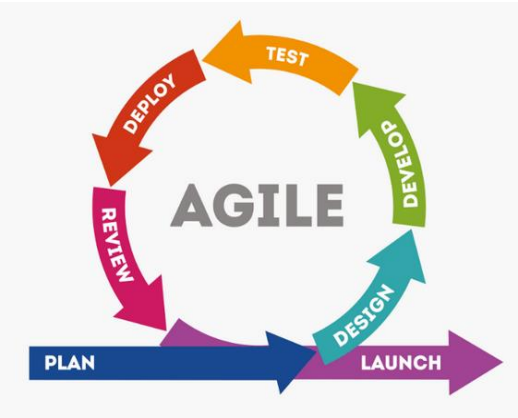

Figura 3: Metodología ágil.

4. Análisis y especificaciones de requisitos

Una vez establecido tanto los requisitos, la planificación temporal y la metodología de trabajo podemos enfocarnos en realizar un análisis de los requisitos para empezar a encontrar posibles soluciones.

Voy a realizar el análisis conforme están definidos los requisitos:

Requisito 1: Realizar un sistema portátil compacto y robusto.

Este requisito implica que el sistema debe ser compacto y portátil en comparación con una computadora. Las posibles soluciones incluyen:

- Utilizar una plataforma de hardware compacta, como Arduino o una placa de desarrollo específica, que sea lo suficientemente pequeña para ser portátil.
- Diseñar una carcasa o envoltura compacta y resistente para alojar el hardware y componentes electrónicos.

<u>Requisito 2:</u> Implementar un menú en el que se pueda navegar para elegir diferentes funcionalidades.

Para implementar un menú navegable, consideré:

- Desarrollar un sistema de menú interactivo en la pantalla LCD o en una interfaz gráfica en un ordenador que permita al usuario navegar y seleccionar diferentes opciones.
- Usar botones físicos o un rotary encoder para la navegación dentro del menú.
- Crear una estructura de datos que represente el menú y sus opciones, y desarrollar un algoritmo que permita al usuario interactuar con él.

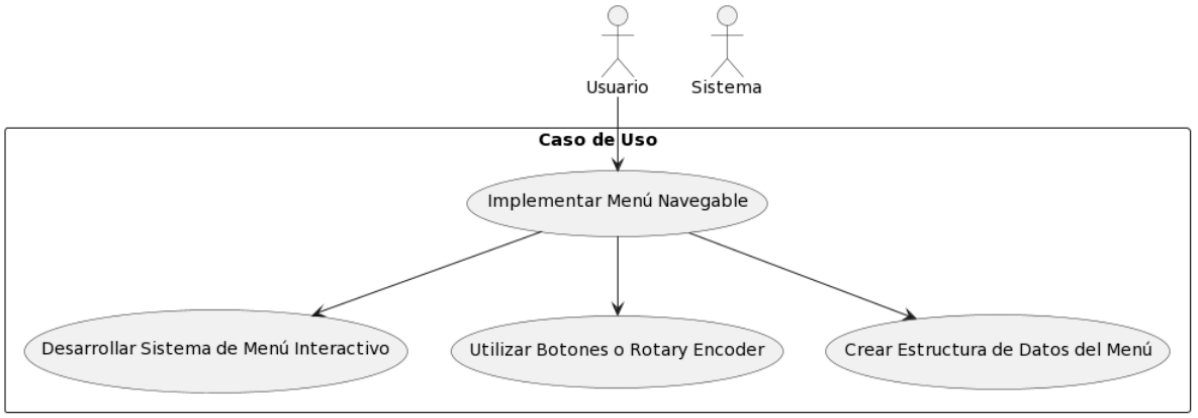

Figura 4: *Diagrama de actividad de los principales usos.*

<u>Requisito 3:</u> Capacidad de modificar las propiedades de las señales PWM (Nombre, Frecuencia, Fase, Modo y Duty cycle).

Para permitir la modificación de propiedades PWM, se pensó:

- Diseñar una interfaz de usuario que permita al usuario editar y ajustar estas propiedades.
- Proporcionar una retroalimentación visual en la pantalla LCD o en la interfaz gráfica para mostrar los cambios realizados.
- Permitir al usuario modificar estos parámetros tanto en el Arduino o en una aplicación de ordenador

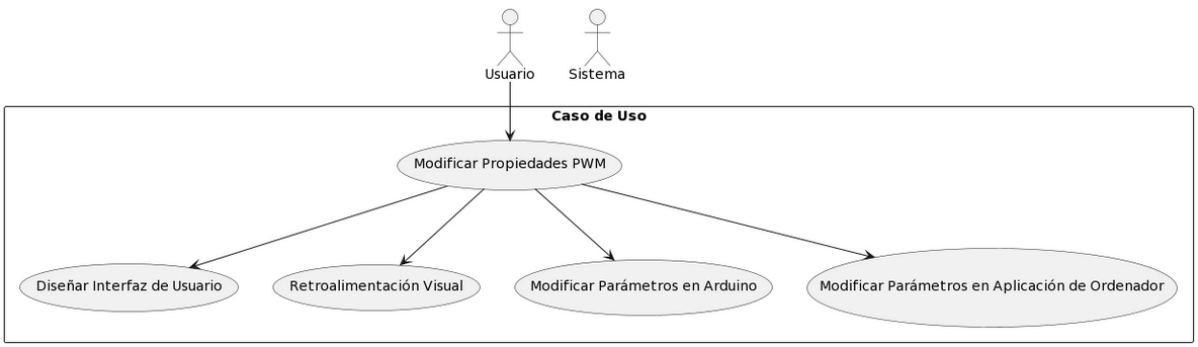

Figura 5: *Diferentes posibilidades de modificación de las señales PWM.*

<u>Requisito 4</u>: Introducción de una contraseña que se pueda modificar para controlar el acceso.

Para poder cumplir con este requisito se pensó en la creación de una pantalla que se accede al intentar modificar una señal PWM en la que se tiene que poner un pin de 4 dígitos para poder finalmente modificar la señal.

<u>Requisito 5</u>: Poder almacenar las propiedades de las señales PWM en una memoria no volátil.

La solución implica:

- Utilizar una memoria no volátil, como EEPROM en Arduino, para almacenar las propiedades PWM de manera permanente, incluso cuando se apaga la alimentación.
- Implementar funciones de lectura y escritura en la memoria no volátil para guardar y recuperar los datos cuando sea necesario.

<u>Requisito 6:</u> Añadir un nombre que compone 8 señales PWM.

Para agregar nombres a las señales PWM:

- Diseñar una interfaz que permita al usuario ingresar y editar nombres para cada señal.
- Utilizar una estructura de datos para almacenar los nombres y relacionarlos con las señales PWM correspondientes.

<u>Requisito 7:</u> Diseñar una interfaz gráfica en un ordenador que permita la carga y guardado de todas las propiedades de las señales PWM.

La solución incluye:

- Crear una aplicación de interfaz gráfica en el ordenador que se comunique con el sistema portátil a través de una conexión serial o USB.
- Implementar la capacidad de cargar, guardar y editar las propiedades de las señales PWM a través de la interfaz gráfica del ordenador.

Una parte fundamental antes de empezar con el desarrollo del proyecto es la planificación, en la siguiente imagen se pondrá el diagrama de Gantt que se ha utilizado.

4.1. Planificación del proyecto

Una parte fundamental antes de empezar con el desarrollo del proyecto es la planificación, en la siguiente imagen se pondrá el diagrama de Gantt que se ha utilizado.

Diagrama de Gantt

Titulo del proyecto	Aplicación de control un faro de vehículo para demostrador	Institución	Universidad de Granada
Lider del proyecto	Fº Rubén Sánchez Aguilera	Fecha	15 de Abril de 2023

					FASE UNO								FASE DOS								FASE TRES			
					ABRIL				MAYO				JUNIO				JULIO				AGOSTO			
WBS NUMBER	Tarea	Fecha de inicio	Fecha de finalización	Duración (semanas)	SEM 1	SEM 2	SEM 3	SEM 4	SEM 1	SEM 2	SEM 3	SEM 4	SEM 1	SEM 2	SEM 3	SEM 4	SEM 1	SEM 2	SEM 3	SEM 4	SEM 1	SEM 2	SEM 3	SEM 4
1	**Inicio y Planificación del proyecto**																							
1.1	Planificación del proyecto	04/04/23	15/04/23	2																				
1.2	Estado del arte	15/04/23	15/05/23	4																				
1.3	Elección de las herramientas	15/05/23	22/05/23	1																				
1.4	Análisis de requisitos	16/05/23	27/05/23	2																				
1.5	Elaboración de Memoria	16/05/23	02/09/23	16																				
2	**Desarrollo del proyecto**																							
2.1	Implementaciones del programa arduino	30/05/23	10/07/23	6																				
2.2	Validaciones de las implementaciones	01/07/23	13/07/23	2																				
2.3	Diseño de la interfaz gráfica	13/07/23	04/08/23	3																				
2.4	Implementación de la lógica	20/07/23	12/08/23	4																				
3	**Pruebas y comparativa de sistemas**																							
3.2	Realización de pruebas	11/08/23	22/08/23	2																				
3.3	Obtención de resultados	25/08/23	30/08/23	1																				

Figura 6: *Planificación del proyecto*

4.2. Coste total del proyecto

Realizar el coste total del proyecto es una tarea muy importante en la que se tiene que considerar múltiples aspectos y puede variar dependiendo de varios factores que se detallarán a continuación.

1. **Costes del personal:** Esta parte es la más significativa del presupuesto, para nuestro proyecto se han necesitado dos programadores, uno junior Rubén Sánchez que ha realizado un aproximado de 300 horas de programación y cada hora se pagaría a 9.00 €. Y un programador senior Andrés Roldán realizando 80 horas de trabajo que se cobrarán a 50.00 € la hora. Esto sería solo para el desarrollo, si se necesitara soporte o mantenimiento después del desarrollo se pagaría aparte.

2. **Costes del hardware**: Para poder desarrollar el proyecto se han necesitado varias piezas que se detallarán a continuación: un Arduino mega 2560, una pantalla LCD de 4x16 caracteres, un rotary encoders.
   Además de los mencionados anteriormente se ha tenido que desarrollar en el laboratorio de GranaSAT una placa PCB personalizada que realiza todas las conexiones de los equipos hardware y una carcasa para que el producto fuese visualmente más vistoso.

3. **Software y licencias:** Para nuestro proyecto se ha requerido el software QT designer, para la creación de la interfaz gráfica, el cual tiene una subscripción de 150.00 € al mes.

4. **Gastos generales y administrativos:** Para el lugar de trabajo GranaSAT ha cedido gratuitamente una estación de trabajo en su laboratorio y los gastos de transporte se ignorarán.

5. **Beneficio industrial:** Debido a que es un producto completamente nuevo en el mercado y una vez creado se cede los derechos a la empresa VALEO para su uso y distribución se ha añadido un beneficio industrial del 15% al costo final del producto.

Una vez analizado todos los factores que intervienen en el coste total del proyecto se realizará una cotización que se entregará al cliente y que se puede ver en la siguiente figura.

COTIZACIÓN

GranaSAT

NÚMERO	1
FECHA	08/11/2023

DESCRIPCIÓN	UNIDADES	PRECIO	TOTAL
Arduino Mega 2560	1	20,00 €	20,00 €
Pantalla LCD 4x16	1	5,00 €	5,00 €
Placa PCB	1	20,00 €	20,00 €
Carcasa	1	50,00 €	50,00 €
Ordenador	1	500,00 €	500,00 €
Mano de obra Junior	300	10,00 €	2.700,00 €
Mano de obra Senior	80	50,00 €	4.000,00 €
Licencia 3 meses QT designer	3	150,00 €	450,00 €
Coste total proyecto			7.745,00 €
Beneficio industrial	15%	1.161,75 €	8.906,75 €

SUB-TOTAL		8.906,75 €
IVA %	21%	1.870,42 €
TOTAL		10.777,17 €

***Figura 6**: Recibo del coste total del proyecto.*

5. Tecnologías utilizadas

En este apartado se mostrarán las herramientas que han sido utilizadas en el desarrollo del proyecto.

5.1. Entornos de desarrollo

5.1.1. Arduino IDE

El Arduino IDE es un entorno de desarrollo integrado (IDE) ampliamente utilizado para programar placas de desarrollo Arduino, como Arduino Uno, Mega, Nano, entre otras. Proporciona una interfaz de usuario amigable que facilita la escritura, carga y depuración de código en lenguaje C para Arduino. Algunas de las características clave incluyen:

- Editor de código con resaltado de sintaxis.
- Integración con bibliotecas estándar de Arduino.

- Herramientas de compilación y carga.
- Monitor serial para la depuración.

5.1.2. Visual Studio

Visual Studio se utilizó junto arduino IDE ya que es un entorno de desarrollo integrado ampliamente utilizado para diversas aplicaciones de desarrollo de software. Es muy útil para describir y depurar código en lenguaje C u gracias a su sus opciones visuales para desarrollar parte del software.

Además permite el uso de GitHub Copilot, la cual es una herramienta de desarrollo desarrollada por GitHub en colaboración con OpenAI. Se basa en tecnologías de lenguaje natural avanzadas y ofrece sugerencias de código en tiempo real mientras se desarrolla la programación. Algunas de sus características son:

- Autocompletar código.
- Generación de comentarios y documentación.
- Ofrece soluciones para problemas comunes.
- Ayuda en la detección de errores y en la escritura de pruebas unitarias.

GitHub Copilot aumentó la productividad y aceleró el desarrollo al proporcionar sugerencias y automatizar partes del proceso de codificación.

Figura 7: Logo GitHub Copilot.

5.1.3. Qt Designer.

Es una herramienta específica para el desarrollo de interfaces gráficas de usuario (GUI) que se integra con el framework Qt. Es utilizado para diseñar y crear interfaces gráficas interactivas para la aplicación de control en el ordenador. Qt Designer es ampliamente utilizado en el desarrollo de aplicaciones de escritorio multiplataforma.

5.2. Sistema operativo

Para llevar a cabo el desarrollo del proyecto se ha utilizado Windows, en concreto la versión 10. Esta elección se debe a que:

- Durante las primeras etapas del desarrollo, sistemas operativos como Linux daba muchos problemas con la comunicación con Arduino.
- Se han utilizado varios puestos de trabajo como mi ordenador personal o el ordenador de sobremesa del laboratorio y para no tener problemas de compatibilidad decidí utilizar Windows como sistema operativo principal.
-

5.3. Lenguajes de programación

Los lenguajes de programación C, C++ y Python son algunos de los más utilizados en el mundo del desarrollo de software. Cada uno tiene sus características únicas y casos de uso donde sobresalen. Voy a presentar las ventajas e inconvenientes de los lenguajes que he escogido y después realizaré una comparación con otros lenguajes existentes.

5.3.1. Lenguaje C

Ventajas:

- **Rendimiento:** C es conocido por generar código muy eficiente, lo que lo hace ideal para sistemas donde el rendimiento es crítico.
- **Nivel de sistema:** Se puede utilizar para escribir software a nivel de sistema como sistemas operativos y drivers.
- **Portabilidad:** Casi todos los sistemas operativos modernos están escritos en C o soportan aplicaciones C.

Desventajas comparadas con otros lenguajes:

- **No es amigable para principiantes:** La sintaxis y los conceptos pueden ser difíciles para los principiantes.
- **Gestión manual de memoria:** A diferencia de lenguajes como Java o Python, C requiere que el desarrollador gestione la memoria manualmente, lo que puede llevar a errores como fugas de memoria.

5.3.2. C++

Ventajas:

- **Orientación a objetos:** C++ es una extensión de C que soporta programación orientada a objetos, lo que facilita la gestión de programas grandes y complejos.
- **Versatilidad:** Se puede utilizar para desarrollar aplicaciones de sistema, aplicaciones de escritorio, juegos, aplicaciones en tiempo real, etc.
- **STL (Standard Template Library):** Viene con una biblioteca de plantillas estándar rica y poderosa.

Desventajas comparadas con otros lenguajes:

- **Complejidad:** Tiene una curva de aprendizaje más pronunciada que otros lenguajes debido a su complejidad y características avanzadas.
- **Menos gestión de memoria que C:** Pero sigue siendo más complicado que en lenguajes con recolección de basura automática como Java.

5.3.3. Python

Ventajas:

- **Facilidad de uso:** La sintaxis de Python es limpia y legible, lo que hace que sea fácil de aprender y popular entre los principiantes y los expertos por igual.
- **Versátil y flexible:** Es ampliamente utilizado en desarrollo web, análisis de datos, inteligencia artificial, ciencia y automación.
- **Gran comunidad y bibliotecas:** Tiene una enorme cantidad de bibliotecas y marcos de trabajo para prácticamente cualquier cosa.

Desventajas comparadas con otros lenguajes:

- **Rendimiento:** Generalmente es más lento que C o C++ debido a que es un lenguaje interpretado.
- **Tipado dinámico:** Aunque puede ser una ventaja, también puede llevar a errores que solo se detectan en tiempo de ejecución.

5.3.4. Contraste con otros lenguajes

Java: A menudo se compara con C++, ya que ambos son orientados a objetos, pero Java tiene gestión de memoria automática y es más fácil de manejar. Sin embargo, C++ suele ser más rápido y ofrece un mayor control a nivel de sistema.

JavaScript: Es el lenguaje principal para el desarrollo web del lado del cliente, algo que Python no está diseñado para hacer nativamente. Python, sin embargo, se usa para el desarrollo de back-end con frameworks como Django o Flask.

Ruby: Similar a Python en términos de facilidad de uso y productividad, pero Python tiene una comunidad más amplia y es más popular en campos como el análisis de datos y la IA.

Finalmente, para nuestro proyecto se llegó a la conclusión de que **C**, **C++** y **Python** eran las mejores opciones ya que son lenguajes poderosos y versátiles que cubren un amplio rango de necesidades de programación.

C y **C++** son inmejorables en rendimiento y nivel de control para poder desarrollar nuestro programa en arduino, mientras que **Python** ofrece una gran rapidez de desarrollo y flexibilidad para realizar la lógica de la interfaz gráfica.

Bloque 2: Desarrollo del proyecto

6. Diseño del código en Arduino

Una vez identificado y analizado los requisitos, siguiendo con la metodología ágil, debemos realizar el diseño del proyecto. Para ello me voy a ayudar de diagramas de flujo, actividad y clase.

Primeramente, voy a realizar un diagrama de componentes para ver la interacción que tienen que tener todos los archivos del proyecto entre ellos

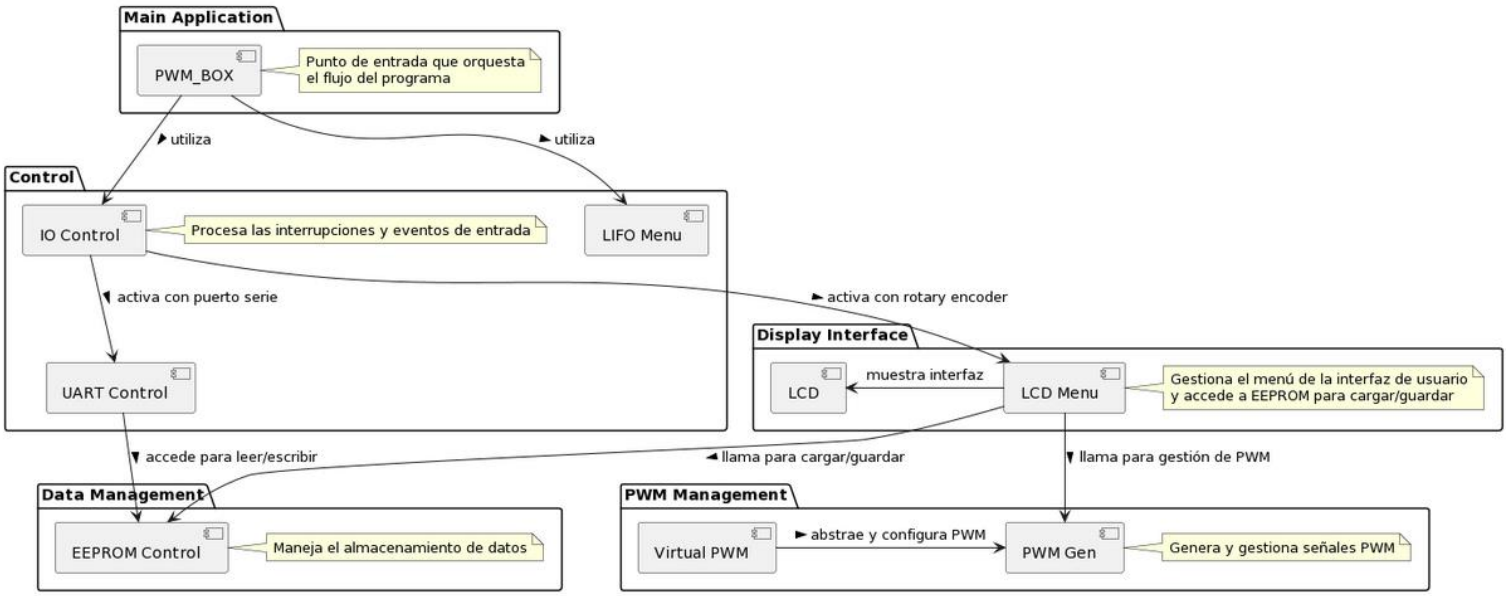

Figura 8: *Diagrama de componentes del nuevo diseño.*

Una vez diseñado la estructura e interacción general del software, se puede ver cómo debería de ser tratada una interacción con el sistema:

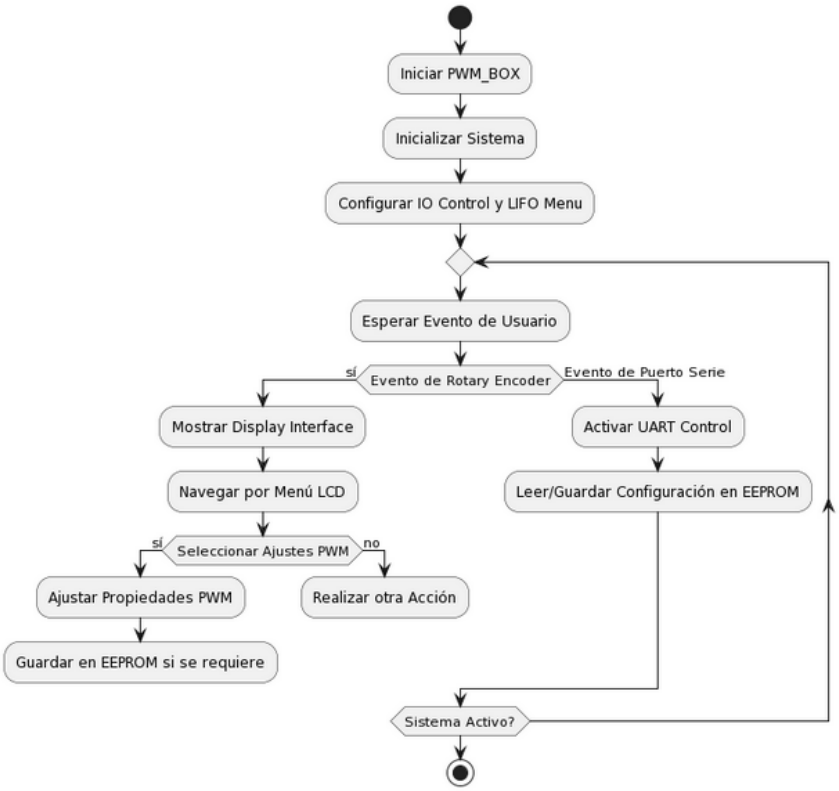

Figura 9: *Diagrama de flujo.*

Una vez que se ha visto cómo deberían de interactuar los componentes del sistema, voy a proceder a diseñar el funcionamiento de los paquetes, control, data management, display interface y PWM managment.

6.1. Diseño del módulo Control.

El paquete diseño de control tiene una función clara, manejar los eventos y ejecutar el correspondiente módulo, este debería de ser el funcionamiento a nivel de usuario.

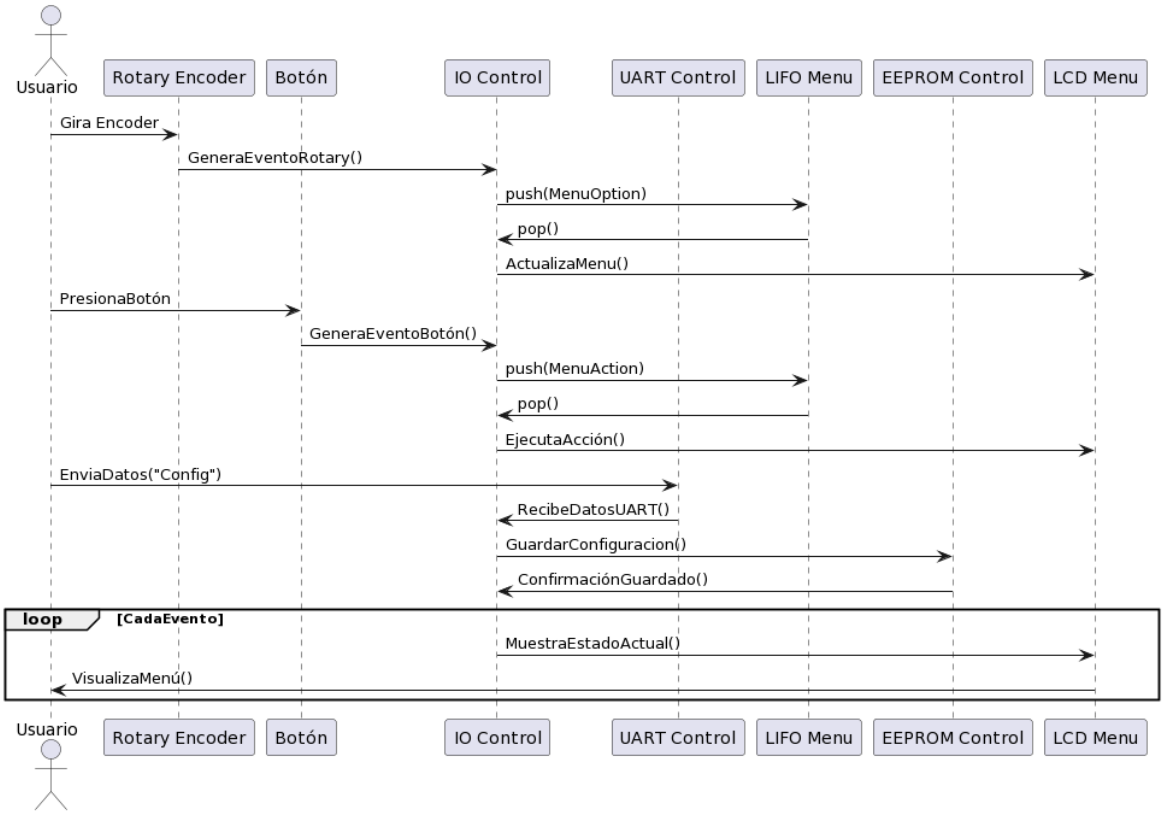

Figura 10: *Diagrama de secuencia de la parte de control.*

6.2. Diseño del módulo Display Interface.

El paquete que maneja el menú mostrado a través del LCD deberá mostrar todas las funciones que puede hacer el sistema, tales como modificar señales PWM, cargar o guardar los datos de las señales, teniendo la siguiente secuencia.

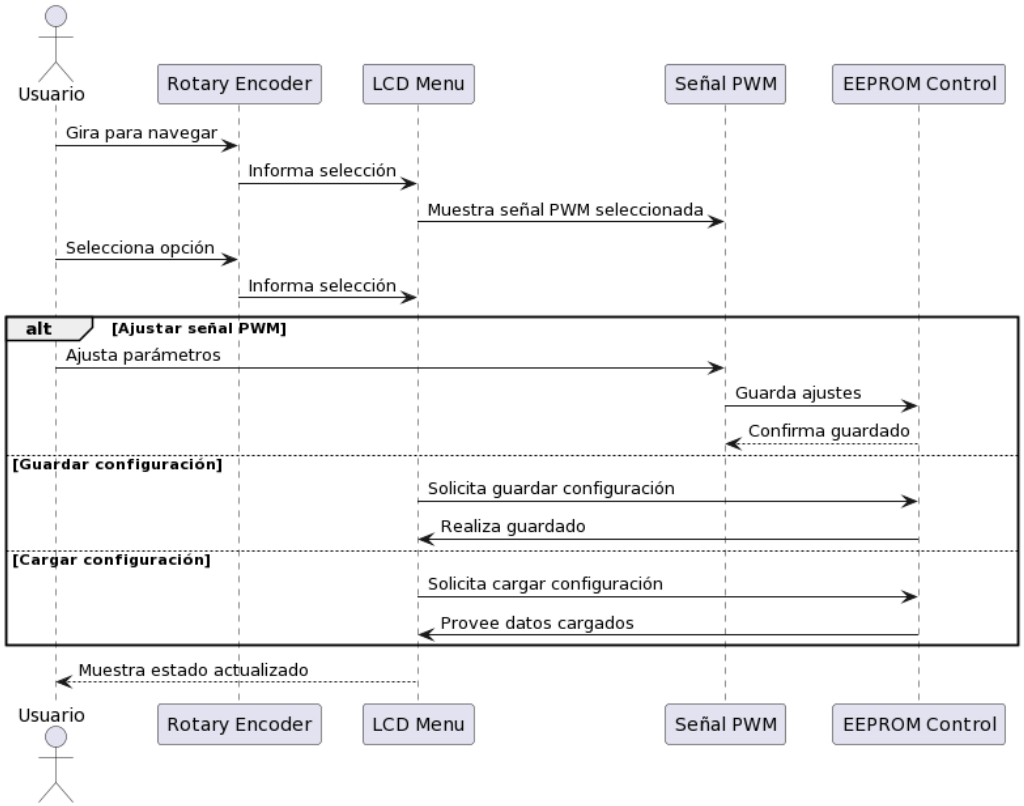

Figura 11: *Diagrama de secuencia del display.*

6.3. Diseño del módulo PWM Management.

Estos archivos fueron diseñados e implementados por Luis Sánchez, por lo que no puedo explicar el diseño que realizó, pero si se explicará su funcionamiento en el apartado de implementación.

6.4. Diseño del módulo Data Management.

Para poder cargar y guardar los datos en la memoria EEPROM del Arduino, el módulo data management debería de tener las siguientes funcionalidades:

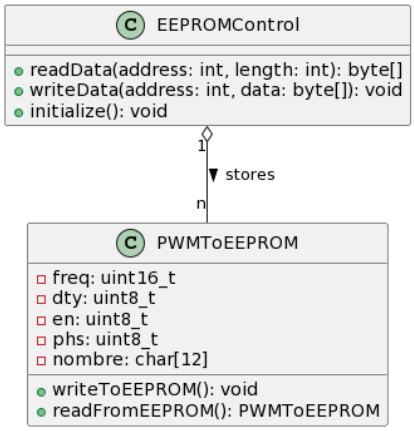

Figura 12: *Diagrama de clases.*

7. Implementación del código en Arduino

Una vez se ha realizado la descripción del proyecto, enumerando y analizando los requisitos y diseñado el sistema, es hora de comenzar la implementación del producto.

Para realizar una descripción del producto más estructurada y sencilla se va a proceder a explicar uno por uno la funcionalidad de los archivos que componen el proyecto, indicando en cada uno de ellos el requisito funcional que resuelven.

7.1. Implementación Main Application (PWM_BOX).

Este es el archivo principal del proyecto, y en él residen las declaraciones y el manejo de interrupciones que más adelante serán necesarias.

Como todos los programas de Arduino contiene la función **setup()**, la cual inicializa todos los datos y componentes que posteriormente se van a utilizar, como la inicialización del LCD, configurar las señales PWM, comprobar que la memoria eeprom está inicializada y si no es así inicializarla, etc.

También contiene la función **main(),** la cual ejecuta un bloque infinito en el que espera que la cola de eventos **lifo_pop** haya una tarea que ejecutar y ejecuta la operación a través de la función **process_evet()**.

```
if(lifo_pop(&event_q, &data))
{
  process_event(data,rx_buf, pwms );
}
```

***Figura 13**: Principal función de PWM_BOX.*

También tiene un contador **seconds_passed_since_ui,** que cuando llega a un total de 1 minuto y 30 segundos sin que haya pasado nada activa en el LCD un protector de pantalla. Al utilizar de nuevo el rotary encoder, se regresaría a la pantalla anterior.

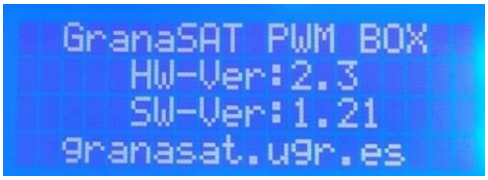

***Figura 14:** Protector de pantalla.*

El manejo de las interrupciones es un apartado esencial ya que permiten a un microcontrolador responder rápidamente a ciertos eventos, como la entrada de señales externas o cambios de estado internos, pausando temporalmente el programa principal para atender estos eventos. Vamos a detallar cada uno de los ISR presentes en el código:

- ISR para PWM:

Esta ISR se activa cuando el Timer2 alcanza un valor específico y regula el estado de las señales PWM llamando a la función **pwm_cycle()**, que se encuentra en el archivo **pwm_gen.**

- ISR para UART:

Esta función es un componente clave para la comunicación serial con el Arduino. Cuando se recibe un byte a través de **USART0** esta función se encarga de leerlo a través de la función **read_uart_rx()**, que lee el registro **UDR0** y lo almacena en el buffer **rx_buf**, cuando el carácter es una nueva línea significa que ya se ha leido la palabra o frase completa creando un **nuevo evento UART_RX** y lo coloca en la cola de eventos para que posteriormente en el main se maneje de la manera oportuna.

```
ISR(USART0_RX_vect){
 event_td uart_event;
 uint8_t c = read_uart_rx();
 if(c != '\r'){
   if(c == '\n'){
     rx_buf[rx_buf_counter] = '\n';
     rx_buf_counter = 0;
     uart_event.event_enum = UART_RX;
    //uart_event.parameter =rx_buf;
    uart_flush();
     lifo_push(&event_q, uart_event);

   }
   else{
     rx_buf[rx_buf_counter] = c;
     rx_buf_counter++;
     //uart_gets(rx_buf);
   }
 }
}
```

Figura 15: *Implementación de la ISR.*

- ISR para rotary encoder.

En este caso la ISR se activa cuando los pines INT1 e INT3 que están conectados al encoder, realizan un cambio de estado. Esta función se encarga de llamar a process_rotary_int(), la cual contiene la lógica para determinar la dirección de la rotación del encoder y crea un nuevo evento de movimiento del encoders y lo coloca en la cola de eventos para su posterior manejo en el main.

```
void process_rotary_int()
{
  seconds_passed_since_ui=0;
  event_td encoder_event;
  uint8_t ro= process_rotary();  //En IO_control.c
  if ( ro==DIR_CW ) { //UP
        encoder_event.event_enum =MOVEMENT_UP;
        encoder_event.parameter2 = ROTARY_SLOW;
        lifo_push(&event_q, encoder_event);
        last_rotary_event_dir = MOVEMENT_UP;
        last_rotary_event = millis();
  }
  else if(ro==DIR_CCW){ //DOWN
        encoder_event.event_enum =MOVEMENT_DOWN;
        encoder_event.parameter2 = ROTARY_SLOW;
        lifo_push(&event_q, encoder_event);
        last_rotary_event_dir = MOVEMENT_DOWN;
        last_rotary_event = millis();
  }
}
```

Figura 16: *Implementación de la ISR para el rotary encoder.*

En resumen, este archivo se utiliza como manejo de interrupciones y como soporte para que los demás archivos puedan cumplir con sus funciones. Para poder realizar esta tarea utiliza una cola de eventos la cual está implementada en el archivo lifo_mem y se explicará a continuación.

7.2. Implementación Control.

7.2.1. LIFO_MEM

En este archivo se encuentra la estructura y funciones para poder realizar el manejo de los eventos con una estructura de datos tipo LIFO (Last In, First Out), comúnmente conocida como pila, con una lógica de buffer circular (o ring buffer). Se encuentra separado en dos partes.

7.2.1.1. LIFO_MEM.h

Este archivo de cabecera define la estructura y las funciones para la manipulación de una memoria tipo LIFO. Las características clave incluyen:

- **Definición de Tamaño Máximo:** LIFO_MAX_SIZE determina el número máximo de elementos que la pila puede contener, en el proyecto hemos decidido que sean 50.
- **Estructura de Evento**: event_td define la estructura de un evento, que incluye un enumerador para el tipo de evento y hasta dos parámetros que pueden ser utilizados cuando se procesa el evento.

  Existen dos tipos de evento, **UART_RX**, para poder controlar la comunicación serial y varias definiciones como **MOVEMENT_UP** o **MOVEMENT_DOWN** para controlar el rotary encoders.

- **Estructura de la Memoria LIFO**: lifo_mem contiene el buffer que almacena los eventos, así como punteros de lectura (rd_ptr) y escritura (wrt_ptr) que gestionan dónde se insertan y se leen los eventos.
- **Funciones de Control**: Se definen cuatro funciones clave para operar la memoria LIFO: is_lifo_empty(), is_lifo_full(), lifo_pop(), lifo_push().

La implementación del comportamiento de las funciones se realiza en el siguiente archivo

7.2.1.2. LIFO_MEM.c

Este archivo de implementación define el comportamiento de las funciones declaradas en lifo_mem.h. Las operaciones son:

- **is_lifo_empty():** Compara los punteros de lectura y escritura; si son iguales, la pila está vacía.
- **is_lifo_full():** Determina si la pila está llena comprobando si el puntero de escritura está justo detrás del puntero de lectura en el buffer circular.
- **lifo_pop():** Extrae un evento de la pila. Si la pila no está vacía, copia el evento en la posición del puntero de lectura al espacio proporcionado por el llamador y luego mueve el puntero de lectura hacia adelante. Si la pila está vacía, retorna 0 para indicar que no hay eventos para procesar.
- **lifo_push():** Inserta un nuevo evento en la pila. Si la pila no está llena, coloca el evento en la posición actual del puntero de escritura y luego avanza este puntero. Si la pila está llena, retorna 0 para indicar que no se pudo agregar el evento.

Una vez que se ha explicado como funciona el programa principal y cómo maneja las interrupciones y eventos, se va a explicar la funcionalidad de los demás archivos.

7.2.2. Config.h

Es un archivo de configuración clave en el firmware. Se definen las configuraciones para diferentes versiones de placas de circuito impreso (PCBs), los pines utilizados para la generación de PWM, y otras constantes críticas para el funcionamiento del firmware.

7.2.3. IO_control

El módulo IO_control encapsula la lógica funcional para la gestión de eventos de entradas/salidas (E/S) y constituye un componente esencial para la interacción entre el usuario y el sistema.

Se integra estrechamente con otros componentes del sistema, como **lcd_menu** para la interfaz de usuario y **eeprom_control** para el almacenamiento persistente. La estandarización de los tipos de eventos y la gestión centralizada de estos permite una interacción fluida y coherente entre los distintos módulos del firmware.

A continuación, se detalla la implementación de cada las funciones y su contribución al proyecto:

7.2.3.1. Inicialización

En la función **setup_IO_interrupts()** se configura los pines del microcontrolador para permitir las interrupciones en las acciones del usuario, como girar el codificador rotativo o presionar un botón. Cada interrupción invoca una función específica (ISR) que procesa el evento y actualiza el estado del sistema de manera oportuna. La configuración precisa evita el polling continuo de los pines, reduciendo la carga sobre el procesador y permitiendo una respuesta más rápida a las acciones del usuario.

```
void setup_IO_interrupts(){
    cli();
    pinMode(20, INPUT);
    pinMode(18, INPUT);
    pinMode(19, INPUT);

    EICRA = (1<<ISC10) | (1<<ISC21)  | (1<<ISC20) | (1<<ISC31)  | (1<<ISC30); //Interrupt 1 any edge, Interrupt 2 on rising edge
    EIMSK = (1<<INT1) | (1<<INT2) ; // Enabled both interrupt 1 and 2 and 3
    sei();
}
```

Figura 17: *Implementación de la función iniciadora.*

7.2.3.2. Manejo del rotary encoders

La función **process_rotary()** permite la interpretación de las señales del codificador rotativo para determinar la dirección y velocidad de rotación. Se utiliza un algoritmo de máquina de estados para decodificar la secuencia de pulsos generados por el codificador, distinguiendo entre giros horarios y anti-horarios, así como la velocidad de giro. Esta información se emplea para navegar a través de un menú LCD o ajustar parámetros en tiempo real.

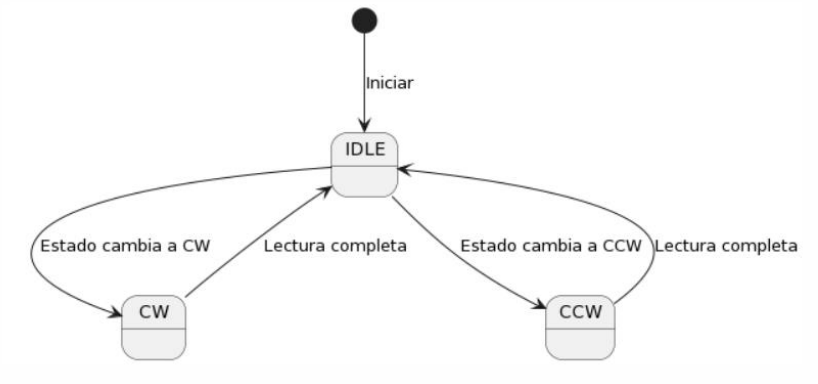

Figura 18: *Diagrama de estados.*

7.2.3.3. Manejo de Eventos

La función **process_event()** actúa como el distribuidor central de eventos, recibiendo datos de eventos capturados por las interrupciones o por la comunicación serial y tomando acciones según sea necesario. Esto asegura que todas las entradas del usuario y las señales del sistema sean consideradas y gestionadas en un solo lugar, lo cual simplifica la lógica del programa principal y facilita la mantenibilidad.

En caso de ser un evento del rotary encoder, llama a las funciones declaradas en los archivos **lcd_menu** para poder navegar a través del menú que se muestra en el LCD.

En caso de ser un evento causado por el envío de mensajes a través del puerto serie, entra en acción la lógica que se ha diseñado para tratar la comunicación entre la interfaz gráfica y el guardado/cargado de los datos en la memoria EEPROM.

Tanto el envío como recibimiento de la información se hace a través del array **rx_buf,** el cual se pasa como atributo al principio de la función y se obtiene en la función **PWM_BOX.ino** al manejar las interrupciones con ISR, contiene una cadena de caracteres codificada con los datos para guardar en la memoria EEPROM o el código para enviar los datos guardados en la memoria.

La lógica para procesar esta codificación o decodificación depende del primer elemento del array, el segundo está reservado para la posición de memoria. A continuación se mostrará cómo se tratan los datos dependiendo del primer elemento del array:

1. **Código A, carga y envío de las señales PWM del micro al PC**

Este caso se trata del envío de los datos de las señales PWM, para ello se hace uso de la función **eeprom_load_pwms**, pasando como parámetros la estructura de las señales PWM y la posición de la memoria que se encuentran gracias al parámetro **rx_buf[1]**, una vez se cargan los en las diferentes variables se concatenan en un único array y se envía a través del puerto serie gracias al uso de la función **uart_puts().**

```
pwm_to_eeprom_t aux[8];

eeprom_load_pwms(aux, pos);

for(i = 0; i < 8; i++){
    freq = aux[i].freq;
    dty = aux[i].dty;
    en = aux[i].en;
    phs = aux[i].phs;
    strcpy(nombre, aux[i].nombre);

    //Concatenacion de los datos y envio
    //uart_puts(nombre);
    sprintf(respuesta, "%s,%u,%04u,%03u,%03u\r\n", nombre, en, freq, dty, phs);
    uart_puts(respuesta);
}
```

Figura 19: *Lógica para el envió de datos.*

2. Código B, recibo y guardado de las señales PWM del PC al micro

Para el guardado de las señales PWM, se recibe a través del array **rx_buf**, todas las 8 señales PWM codificadas, separando las 8 señales PWM entre almohadillas y los datos de estas entre comas, para poder entender más sencillamente cómo se realiza este proceso he creado el siguiente diagrama de flujo.

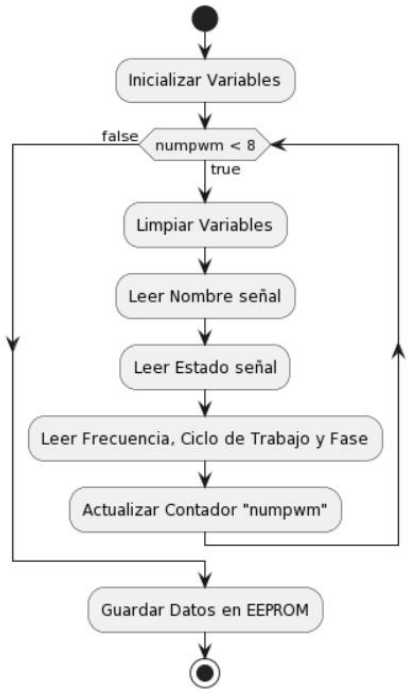

Figura 20: *Diagrama de flujo.*

Una vez decodificado el mensaje, se guardan las señales en la memoria gracias a la función **eeprom_save_pwms().**

3. Código C y D, enviar y guardar la contraseña micro al PC

Este apartado es muy sencillo, para el código C, envío de contraseña se utiliza la función **eeprom_pass_read()** y luego se envía gracias a la función **uart_puti()**.

Para el caso de guardar contraseña como los datos vienen en tipo char, se utiliza la función **atoi()**, devolviendo un número numérico y almacenando en la memoria eeprom.

4. Código E, F y G, tratamiento nombre conjuntos PWM

Debido al requisito número 5 el conjunto de las señales PWM debían estar compuestas por un nombre, normalmente el nombre del vehículo que componen las señales PWM, en este apartado se realiza las funciones de guardado y escritura de estos nombres.

Para el código E, envío del nombre del coche, se usa la función **obtenerNombreCoche()** implementada en el archivo eeprom_control y una vez cargado el nombre se envía gracias a la función **uart_puts()**, implementada en el archivo uart_control.

Para el código G a parte del envío del coche también se realiza el envío del número de coches que hay guardados en memoria.

En el caso del código F, se realiza el guardado del nombre del coche gracias a la función **guardadoNombreCoche()**.

7.2.4. uart_control

Este archivo es una implementación personalizada de las funciones UART (Universal Asynchronous Receiver-Transmitter) para el microcontrolador ATmega2560 del Arduino Mega.

A pesar de ser el archivo con menos líneas de código, fue uno de los más difíciles de implementar debido a la complejidad de tratar con los registros del

microcontrolador del Arduino, teniendo que estudiar el **datasheet** de este hasta encontrar en la página 207 los registros adecuados.

22.6 Data Transmission – The USART Transmitter

The USART Transmitter is enabled by setting the *Transmit Enable* (TXEN) bit in the UCSRnB Register. When the Transmitter is enabled, the normal port operation of the TxDn pin is overridden by the USART and given the function as the Transmitter's serial output. The baud rate, mode of operation and frame format must be set up once before doing any transmissions. If synchronous operation is used, the clock on the XCKn pin will be overridden and used as transmission clock.

A continuación, se describirá el comportamiento de las funciones en este archivo:

1. **uart_init(void)**:
    - Desactiva las interrupciones con **cli()**.
    - Establece la tasa de baudios del UART. Utiliza una definición de BAUD de 57600 y calcula el valor adecuado para el registro **UBRR0**.
    - Habilita el receptor y el transmisor UART. También habilita las interrupciones de recepción con **RXCIE0**.
    - Establece el formato de datos a **8 bits**.
    - Reactiva las interrupciones con **sei()**.

```
void uart_init(void){
    cli();
    UBRR0 = UB;
    UCSR0B = (1<<RXEN0) | (1<<TXEN0) | (1<<RXCIE0);
    UCSR0C = (1<<UCSZ01) | (1<<UCSZ00);
    sei();
}
```

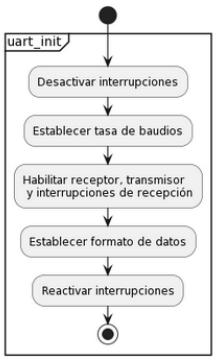

Figura 21: *Implementación de la función de inicialización y diagrama de actividad.*

2. **uart_putc(unsigned char data)**:
    - Espera hasta que el registro de datos de transmisión (**UDR0**) esté libre.
    - Una vez libre, carga el dato data en el registro **UDR0** para transmitirlo.

3. **uart_getc(void)**:
    - Espera hasta que haya datos disponibles para ser leídos en el UART.
    - Lee y devuelve el dato recibido de **UDR0**.

Figura 22: Diagrama de actividad de las funciones putc y getc.

4. **uart_available(void)**:
    - Verifica si hay datos disponibles para leer en el UART y devuelve el estado.

5. **uart_flush(void)**:
    - Limpia el buffer de recepción del UART leyendo y descartando todos los datos disponibles.

7.3. Implementación Display Interface.

7.3.1. lcd_menu

Para poder implementar el requisito 2 y realizar un menú interactivo que se pueda navegar por él y acceder a todas las funcionalidades deseadas, he implementado dos archivos **lcd_menu.h** donde se realizan todas las declaraciones de las estructuras y funciones, y **lcd_menu.c** donde se implementa la lógica.

Además, me he ayudado del archivo **lcd**, el cual tiene los componentes necesarios para el uso de la pantalla LCD 4x16.

Estos archivos son los más grandes del proyecto con mayor número de lineas de código, por lo que para poder explicarlos mejor me ayudaré de diagramas de clase y diagramas de actividad.

7.3.1.1. lcd_menu.h

Está compuesto por estructuras que contienen los diferentes menús disponibles, todos ellos tienen un atributo en común que es el local_menu_pointer, el cual sirve para indicar al usuario donde se encuentra situado en los menús.

Para este producto se han desarrollado los siguientes menús:

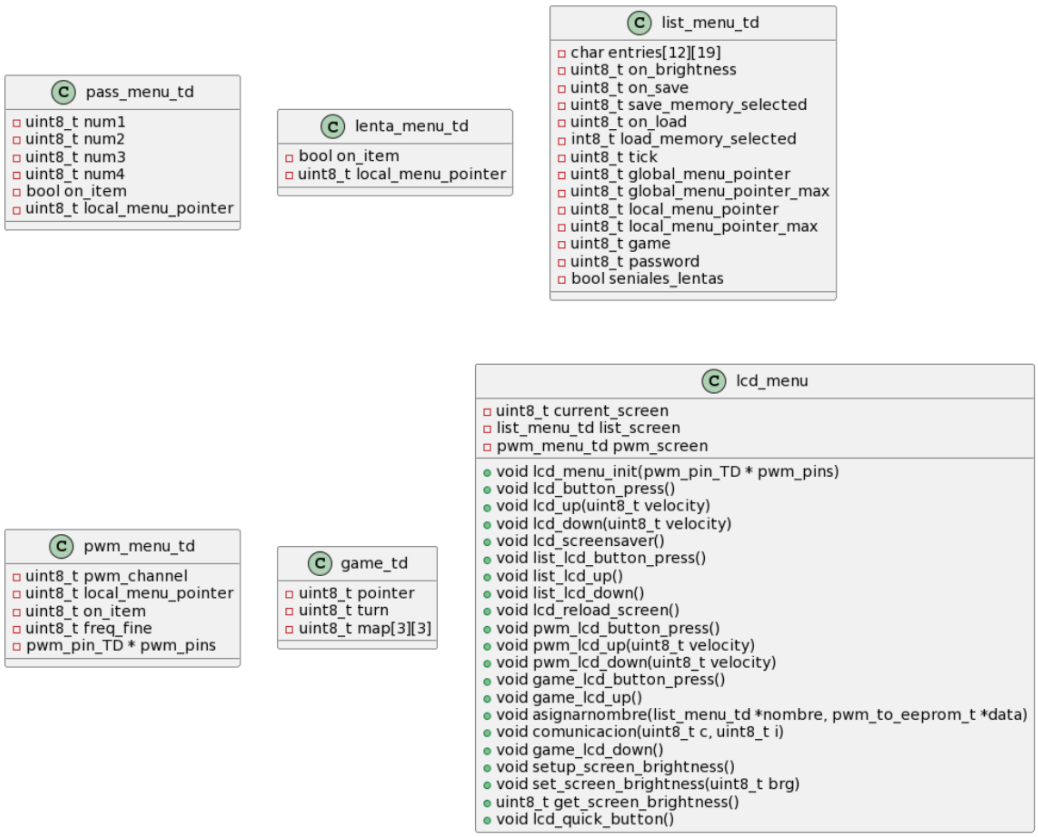

Figura 23: Diagrama de clases del display.

1. **List_menu_td**

Es el menú general que actúa como punto de entrada para las diferentes funciones. Permite al usuario cargar y guardar configuraciones en la memoria EEPROM, ajustar el brillo de la pantalla LCD, acceder al menú de señales PWM y a las señales lentas. Su estructura incluye diferentes punteros para permitir la navegación global y local, así como indicadores para las distintas funciones activas.

Figura 24: *Menú principal que se muestra en el LCD.*

2. **Pwm_menu_td**

Este menú controla la configuración de las señales PWM, permite al usuario modificar los parámetros que componen estas señales, tales como: Frecuencia, modo, ciclo de trabajo y la fase.

Figura 25: *Menú señales PWM.*

Si se intenta modificar por primera vez uno de estos parámetros, como medida de seguridad, saltará el menú de contraseña.

3. **Pass_menu_td**

Este menú gestiona la entrada de una contraseña de seguridad. Consiste en cuatro dígitos (**num1** a **num4**) y permite al usuario navegar y seleccionar cada dígito para su edición. La contraseña debe ser ingresada correctamente para acceder a funciones restringidas, como la modificación de las configuraciones de señal PWM.

Figura 26: *Menú contraseña.*

4. **Game_menu_td**

Este menú es un pequeño easter egg que lo implementó el anterior estudiante y he decidido mantenerlo, se trata del juego en raya.

Todos estos menús están gestionados e integrados en lcd_menu, el cual se explicará a continuación.

7.3.1.2.lcd_menu.c

Este archivo se encarga de realizar la lógica para poder navegar en el menú, para explicar más sencillamente las funcionalidades voy a agruparlas en 4 funciones y luego usaré diagramas de actividad para enseñar gráficamente cómo funciona la navegación.

Las funciones de los diferentes menús se podrían resumir en 4, todas estas funciones se activan al accionar el rotary encoders.

1. **Movimiento hacia izquierda del rotary encoders**

Al realizar este movimiento se activarán todas las funciones (menú correspondiente)**_down()**, las cuales dependiendo si la variable **on_item**.

Esta variable indica si está seleccionado algún elemento de los respectivos menús, en caso de que esté activado, reduce el parámetro seleccionado en 1, en caso de que no esté activado se mueve por el menú.

2. **Movimiento hacia derecha del rotary encoders**

Al realizar este movimiento se activarán todas las funciones (menú correspondiente)**_up()**, las cuales dependiendo si la variable **on_item**.

Esta variable indica si está seleccionado algún elemento de los respectivos menús, en caso de que esté activado, aumenta el parámetro seleccionado en 1, en caso de que no esté activado se mueve por el menú.

3. **Presionar el botón del rotary encoders**

Al realizar este movimiento se activarán todas las funciones (menú correspondiente)**_press()** y **_refresh()**, las cuales activan o desactivan la variable **on_item.**

Si la variable está activada guarda el valor del parámetro seleccionado, si la variable no está activada, pueden suceder dos cosas, acceder a una funcionalidad, es decir, entrar en el menú PWM, confirmar contraseña, entrar en el menú de las señales lentas… O confirmar el valor del parámetro seleccionado y guardarlo.

Siempre que el botón del rotary encoders es activado al final de todas las funciones se llama a la función **_refresh()** del menú correspondiente, el cual actualiza el nuevo menú que se haya elegido o actualiza los valores de los parámetros del mismo menú.

Para ayudar a comprender la navegación por el menú he diseñado unos diagramas de actividad.

En este primer diagrama se observa la navegación por los diferentes menús y funcionalidades del menú principal.

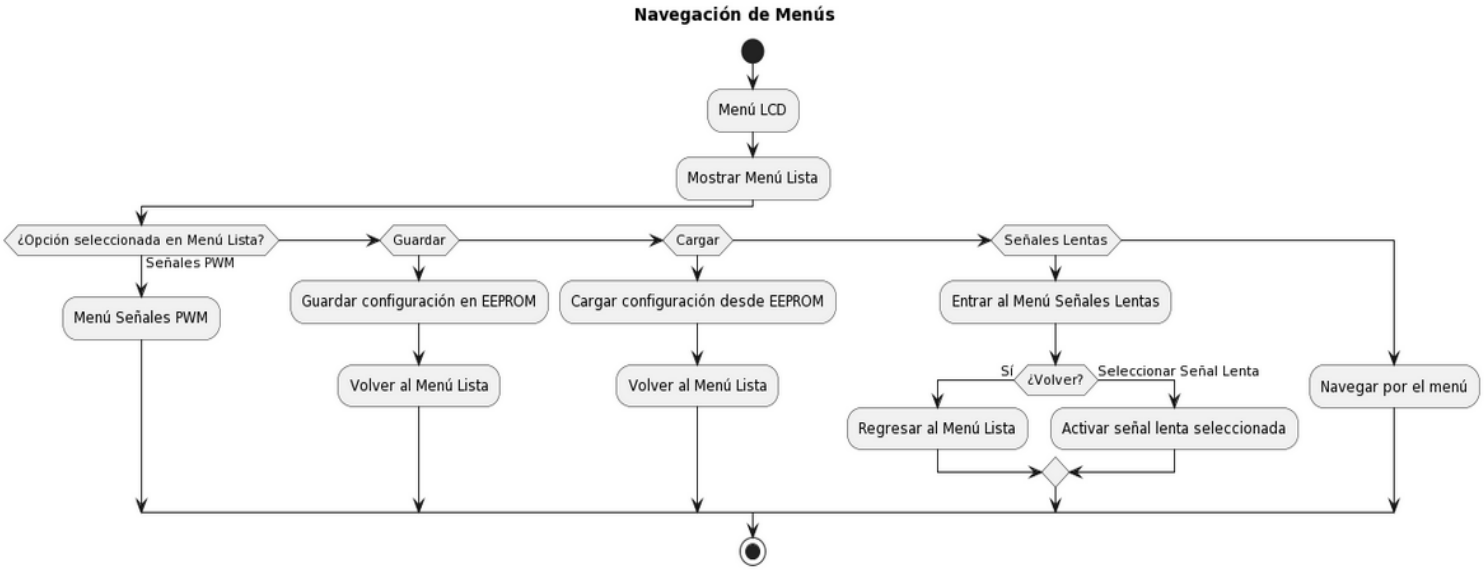

Figura 27: *Diagrama de flujo del menú principal.*

En este segundo diagrama se muestra el menú de las señales PWM

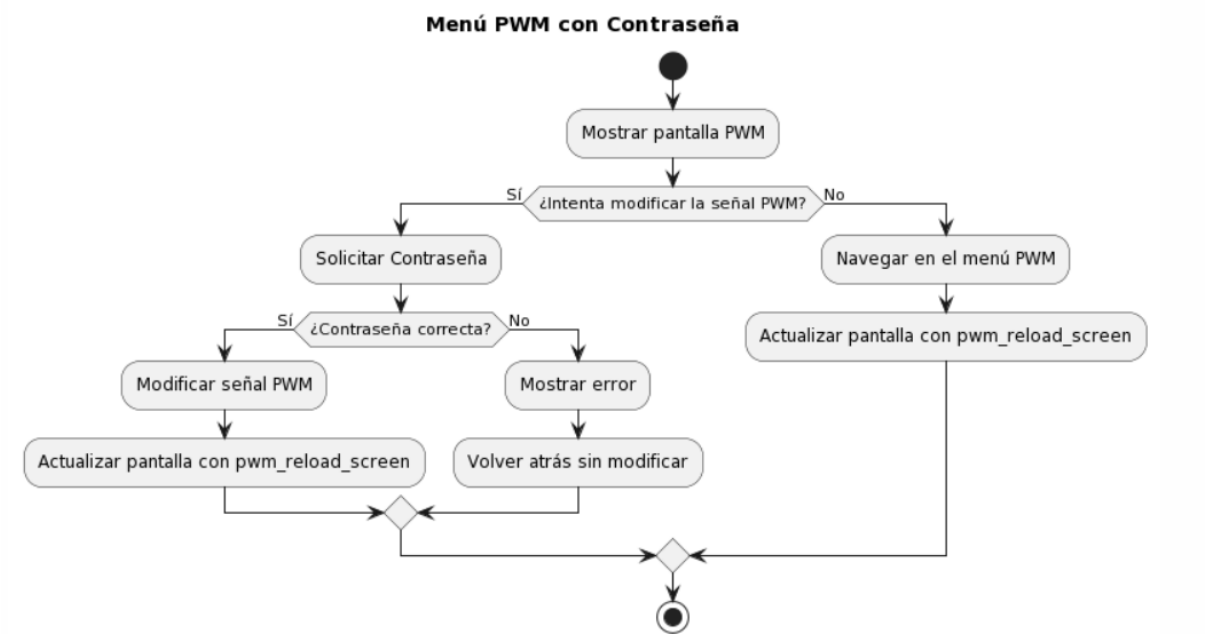

Figura 28: Diagrama de flujo menú señales PWM.

7.4. Implementación del módulo PWM Management.

7.4.1. PWM_gen

El conjunto de archivos **pwm_gen.h** y **pwm_gen.c** son parte del código fuente que permite controlar la generación de señales PWM de una manera muy precisa y configurable. Estos archivos fueron diseñados por el compañero Luis Sánchez y a continuación, se explicará cada parte del código y su función dentro del proyecto.

7.4.1.1. PWM_gen.h

Este archivo de encabezado define la estructura y las funciones que se utilizarán para la generación de las señales PWM.

La estructura **pwm_pin_TD** representa un pin PWM, conteniendo información como el puerto al que está conectado, su configuración, el ciclo de trabajo deseado (**dty**), la frecuencia (**frq**), y la fase (**phase**), además de un contador de ciclos y otros campos de control.

```c
typedef struct pwm_pin{
    /*@{*/
    uint8_t * port; /**< GPIO port in the ATMEGA2560 */
    uint8_t * port_config; /**< GPIO configuration register in the ATMEGA2560 */
    uint8_t pin; /**< GPIO port's bit in the ATMEGA2560 */
    uint8_t en; /**<Is the channel enabled*/
    uint32_t cycles_on; /**< Number of interrupt cycles that the pin shall be HIGH */
    uint32_t cycles_total; /**< Number of interrupt cycles that constitute a period */
    uint32_t cnt; /**< Interrupt cycles counter */
    uint16_t dty; /**< Intended duty cycle for the pin */
    uint16_t frq; /**< Intended frequency for the pin */
    int16_t phase; /**< Intended phase for the pin */
    /*@}*/
} pwm_pin_TD;
```

Figura 29: *Estructura implementada para el módulo PWM.*

Además se realizan las declaraciones que se utilizarán posteriormente en pwm_gen.c, tales como, setup_pwm_interrupt(), turn_off(), start_clock(), stopclock() y pwm_cycle(). Las cuales se explicarán a continuación.

7.4.1.2. PWM_gen.c

Este archivo contiene las implementaciones de las funciones anteriormente mencionadas y que permiten controlar las señales PWM.

- **setup_pwm_interrupt()** configura el temporizador 2 del microcontrolador en modo CTC (Clear Timer on Compare Match) con un prescaler de 8 y un valor de comparación que resulta en una frecuencia de interrupción de 40 kHz. Este es el ritmo al cual las señales PWM serán actualizadas.

- **turn_off()** es una función para apagar un pin rápidamente.

- **start_clock()** y **stop_clock()** habilitan y deshabilitan las interrupciones del temporizador 2 para controlar el inicio y la parada de la generación de PWM.

- **pwm_cycle (pwm_pin_TD * pwm_pins):** itera sobre todos los pines PWM definidos en el vector pwm_pins y actualiza su estado en función de la cantidad de ciclos de interrupción que han pasado, lo cual determina si el pin debe estar en alto o en bajo, implementando así la señal PWM.

7.4.2. Virtual_PWM

Ayudándose de las funciones implementadas en PWM_gen, los archivos virtual_PWM.h y virtual_PWM.c encapsulan las operaciones necesarias para manipular las señales PWM de manera abstracta y conveniente. Permitiendo a los usuarios ajustar las propiedades de las señales de los faros, como la frecuencia y el ciclo de trabajo, con facilidad y precisión. Para ello se ayuda de las siguientes funciones:

- **configure_pwm_pin():** Esta función es el núcleo de la generación de señales PWM. Permite al usuario establecer una frecuencia y un ciclo de trabajo (**duty cycle**) específicos para un pin de salida. La función calcula y establece los valores de temporización necesarios para alcanzar la frecuencia y el ciclo de trabajo deseados basándose en la capacidad del hardware subyacente. Esto resulta en una salida de señal PWM que controla la iluminación, como la intensidad de los faros, con gran precisión.

- **configure_pwm_phase():** Ajusta la fase de la señal PWM para un pin en particular.

- **turn_on_pin():** Esta función de control simplificado activa o desactiva un pin PWM individual, permitiendo a los usuarios controlar de forma rápida y directa la salida de los faros sin tener que ajustar otros parámetros como la frecuencia o el ciclo de trabajo.

- **initialize_all_pins():** Inicializa todos los pines configurados para ser utilizados como pines PWM, estableciendo cada pin con su respectivo puerto y dirección. Esta función es crucial al iniciar el sistema para asegurar que todos los pines estén configurados correctamente y listos para operar.

- **toggle_pwm_pin_enable():** Alterna el estado de un pin PWM entre activado y desactivado. Esta función es útil para controlar rápidamente la salida de los faros sin tener que pasar por un proceso de configuración más detallado.

- **pwm_pin_enable():** Esta función establece explícitamente el estado de habilitación de un pin PWM.

- **sync_pwms():** Sincroniza todos los pines PWM asegurando que sus contadores estén alineados. Esta función es particularmente importante después de cambios en la configuración de la fase o cuando se requiere que los faros operen en un patrón sincronizado.

El conjunto de archivos Virtual_PWM y PWM_gen fueron diseñados por el compañero Luis Sánchez y permiten la implementación del requisito número 3.

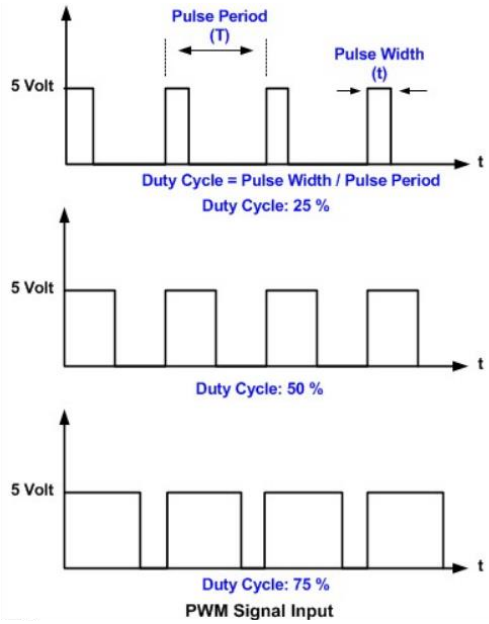

Figura 30: *Ejemplo de la salida de una señal PWM dependiendo de sus características.*

7.4. Implementación del módulo Data Management.

El archivo EEPROM_Control está dividido en dos partes eeprom_control.h y eeprom_contro.c, ambos archivos permiten realizar el **requisito 4 y 5**, almacenando todas las características de las señales PWM junto a su nombre.

A nivel conceptual se ha dividido la memoria en dos partes, dividido por la variable **size_of_pwm_data_area**.

En la primera parte el primer elemento de la memoria está reservado para comprobar si la memoria ha sido inicializada por primera vez, si no es así se añade unas valores de las señales PWM por defecto, los 4 siguientes bytes están reservados para el almacenamiento de la contraseña, el siguiente byte para saber el número de conjuntos de señales PWM que están almacenadas en memoria, y por último se guardan los nombres de estos conjuntos de señales PWM que conforman un automóvil.

En la segunda parte de la memoria se guardan todos los valores que componen las señales PWM.

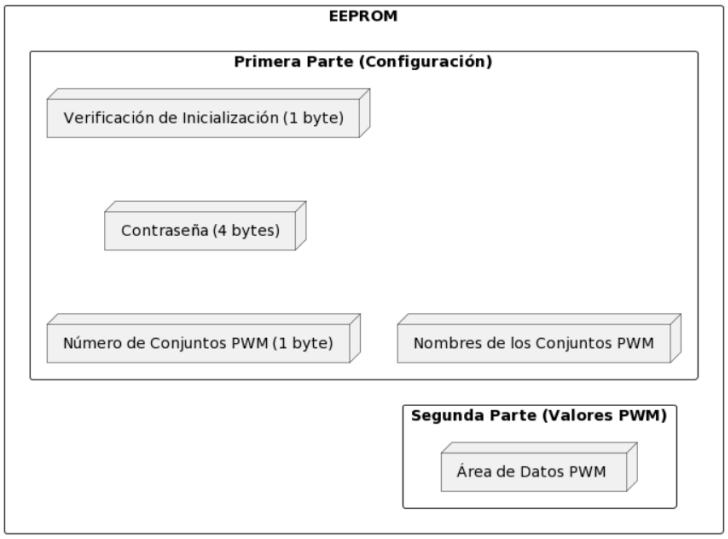

Figura 31: *Distribución de la memoria EEPROM.*

A continuación, se explicará las estructuras y funciones diseñadas para poder realizar el almacenamiento de estos datos.

7.4.1. Eeprom_control.h

El archivo de encabezado **eeprom_control.h** desempeña un papel fundamental en el proyecto al proporcionar las declaraciones de estructuras y funciones necesarias para la gestión de la memoria EEPROM del proyecto. En este apartado se explicarán las estructuras implementadas:

- **Estructura pwm_to_eeprom_t**:

Esta estructura de datos representa una señal PWM (Modulación por Ancho de Pulso) y almacena la siguiente información:

  - **freq**: Frecuencia de la señal PWM.
  - **dty**: Ciclo de trabajo de la señal PWM.
  - **en**: Estado de la señal PWM (activado/desactivado).
  - **phs**: Fase de la señal PWM.
  - **nombre:** Nombre asociado a la señal PWM.

- **Unión eeprom_bridge**:

Esta unión es un componente clave del proyecto, ya que permite almacenar diferentes representaciones de datos PWM. Puede contener:

- Un arreglo de datos PWM, lo que facilita el almacenamiento y recuperación de múltiples señales PWM.
- Un nombre de automóvil, que se asocia a las señales PWM.
- Una representación vectorizada de datos PWM, que puede ser útil para ciertas operaciones.

- **Compatibilidad con C++**:
  - El archivo **eeprom_control.h** está diseñado para ser compatible tanto con código C como con código C++ y utiliza las directivas apropiadas (**#ifdef __cplusplus**) para lograr esta compatibilidad.

En resumen, el archivo **eeprom_control.h** proporciona las declaraciones y estructuras necesarias para gestionar la memoria EEPROM, en el siguiente apartado se explicarán las funciones.

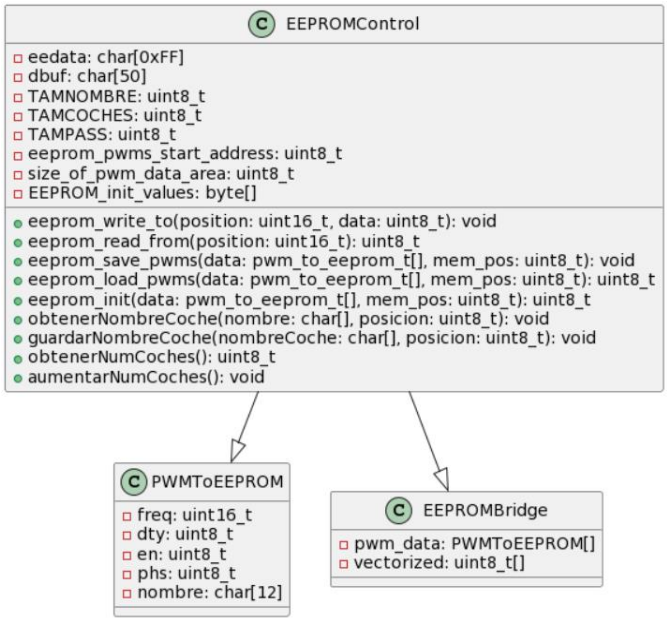

Figura 32: *Diagrama de clases para el módulo data management.*

7.4.2. Eeprom_control.c

En esta sección, se describe las funciones implementadas para poder realizar los guardados y cargados de los datos en la memoria EEPROM.

1. **eeprom_write_to (uint16_t position, uint8_t data):**

Esta función se utiliza para escribir un byte de datos en una posición específica de la EEPROM. Antes de escribir, comprueba si los nuevos datos son diferentes de los datos existentes en esa posición de la EEPROM. Si los datos son diferentes, se realiza la escritura en la EEPROM. Esto ayuda a evitar escrituras innecesarias y a preservar la vida útil de la EEPROM.

```
void eeprom_write_to(uint16_t position, uint8_t data)
{
    if(data!=eeprom_read_from(position) )
    {
        eeprom_write_byte(&eedata[position], data);
    }
}
```

Figura 33: *Implementación de la función que guarda un valor en la memoria EEPROM.*

2. **eeprom_read_from(uint16_t position): uint8_t:**

Esta función se utiliza para leer un byte de datos desde una posición específica de la EEPROM. Devuelve el valor leído como un byte sin procesar.

3. **eeprom_save_pwms (pwm_to_eeprom_t *data, uint8_t mem_pos):**

Esta función se encarga de guardar datos PWM en la EEPROM. Toma como argumentos un puntero a una estructura **pwm_to_eeprom_t** que contiene información de señales PWM y una posición en la EEPROM donde se deben almacenar los datos. Los datos de la estructura se escriben en la EEPROM, y esto permite guardar configuraciones específicas de señales PWM.

```
void eeprom_save_pwms(pwm_to_eeprom_t * data, uint8_t mem_pos)
{
    uint16_t i = 0;
    uint8_t * byte_ptr = (uint8_t *)data;
    for(i = 0; i < sizeof(pwm_to_eeprom_t)*PWM_PINS; i++)
    {
        eeprom_write_to(i + mem_pos * sizeof(pwm_to_eeprom_t)*PWM_PINS
        + size_of_pwm_data_area, byte_ptr[i]);
    }
}
```

Figura 34: *Implementación de la función que guarda las 8 señales PWM en la memoria EEPROM.*

4. **eeprom_load_pwms (pwm_to_eeprom_t *data, uint8_t mem_pos): uint8_t:**

Esta función se utiliza para cargar datos PWM desde la EEPROM y almacenarlos en una estructura **pwm_to_eeprom_t**. Toma un puntero a una estructura y una posición en la EEPROM como argumentos. Lee los datos almacenados en la EEPROM y los copia en la estructura proporcionada.

5. **eeprom_init (pwm_to_eeprom_t *data, uint8_t mem_pos): uint8_t:**

La función eeprom_init se encarga de inicializar la EEPROM y cargar los datos PWM desde la EEPROM en una estructura **pwm_to_eeprom_t**. Si la EEPROM no ha sido inicializada previamente, esta función realiza la inicialización y luego carga los datos almacenados en la EEPROM en la estructura proporcionada. Devuelve un valor indicando el estado de la inicialización.

6. **obtenerNombreCoche (char *nombre, uint8_t posicion):**

Esta función se utiliza para obtener el nombre de un automóvil almacenado en la EEPROM y almacenarlo en un búfer proporcionado. Toma como argumentos un puntero a un búfer (nombre) y la posición del automóvil en la EEPROM. Lee el nombre almacenado en la EEPROM y lo copia en el búfer.

7. **guardarNombreCoche (char *nombreCoche, uint8_t posicion):**

guardarNombreCoche se utiliza para guardar el nombre de un automóvil en la EEPROM. Toma como argumentos un puntero a una cadena de caracteres (nombreCoche) que contiene el nombre del automóvil y la posición en la EEPROM

donde se debe almacenar. Si la posición corresponde a un automóvil existente, se sobrescribe el nombre. Si la posición es la siguiente disponible, se añade un nuevo automóvil y se incrementa el contador de automóviles.

8. **obtenerNumCoches(): uint8_t:**

Esta función se utiliza para obtener el número de automóviles almacenados en la EEPROM. Lee el valor almacenado en la EEPROM que representa el número de automóviles y lo devuelve como un valor entero de 8 bits (uint8_t).

En resumen, estas funciones son esenciales para la gestión de la memoria EEPROM en el proyecto, permitiendo la escritura, lectura, carga y almacenamiento de datos PWM y nombres de automóviles en la EEPROM, lo que facilita la configuración y recuperación de información importante para el sistema de control de automóvil.

8. Diseño de la interfaz gráfica

Para el diseño de la interfaz se han utilizado dos archivos, **lanzador.py** el cual permite realizar toda la lógica y el funcionamiento de la interfaz gráfica además de la comunicación con el Arduino e **interfazgráfica.ui**, la cual es la interfaz gráfica en sí y permite al usuario obtener todos los datos almacenados en el Arduino más sencillamente y eficaz.

Por lo que para abordar cómo se ha desarrollado la interfaz se realizará en dos partes, la primera donde se enseñará cómo ha quedado la interfaz gráfica y cómo el usuario puede interactuar con ella y en la otra parte cómo se produce la lógica y conexión de las peticiones del usuario con el arduino.

8.1. Interfaz gráfica GUI

Para realizar la interfaz gráfica o GUI tuve que instalar el programa QT Designer, como recomendación de mi tutor Andrés. Una vez instalado el programa, debemos abrir una nueva página, en la que incluir los diferentes widgets que permite,

primeramente realizaré una explicación general de cómo funcionan estos objetos y luego entraré más en detalle en cómo los utilicé para realizar mi interfaz gráfica personalizada.

- **Labels (Etiquetas)**: Proporcionan texto informativo en la interfaz de usuario, generalmente no interactúan con el usuario.

- **Push Buttons (Botones de presión):** Los botones de presión ejecutan una acción cuando el usuario hace clic en ellos.

- **Line Edits (Campos de texto):**Permiten a los usuarios ingresar y editar una línea de texto

- **Combo Box (Caja combinada):** Muestra una lista desplegable de opciones que el usuario puede seleccionar.

- **Group Boxes (Cajas de grupo):** Sirven para agrupar visualmente widgets que están relacionados entre sí, a menudo con un título para describir la agrupación.

Aplicando estos widgets a mi interfaz gráfica he construido lo siguiente.

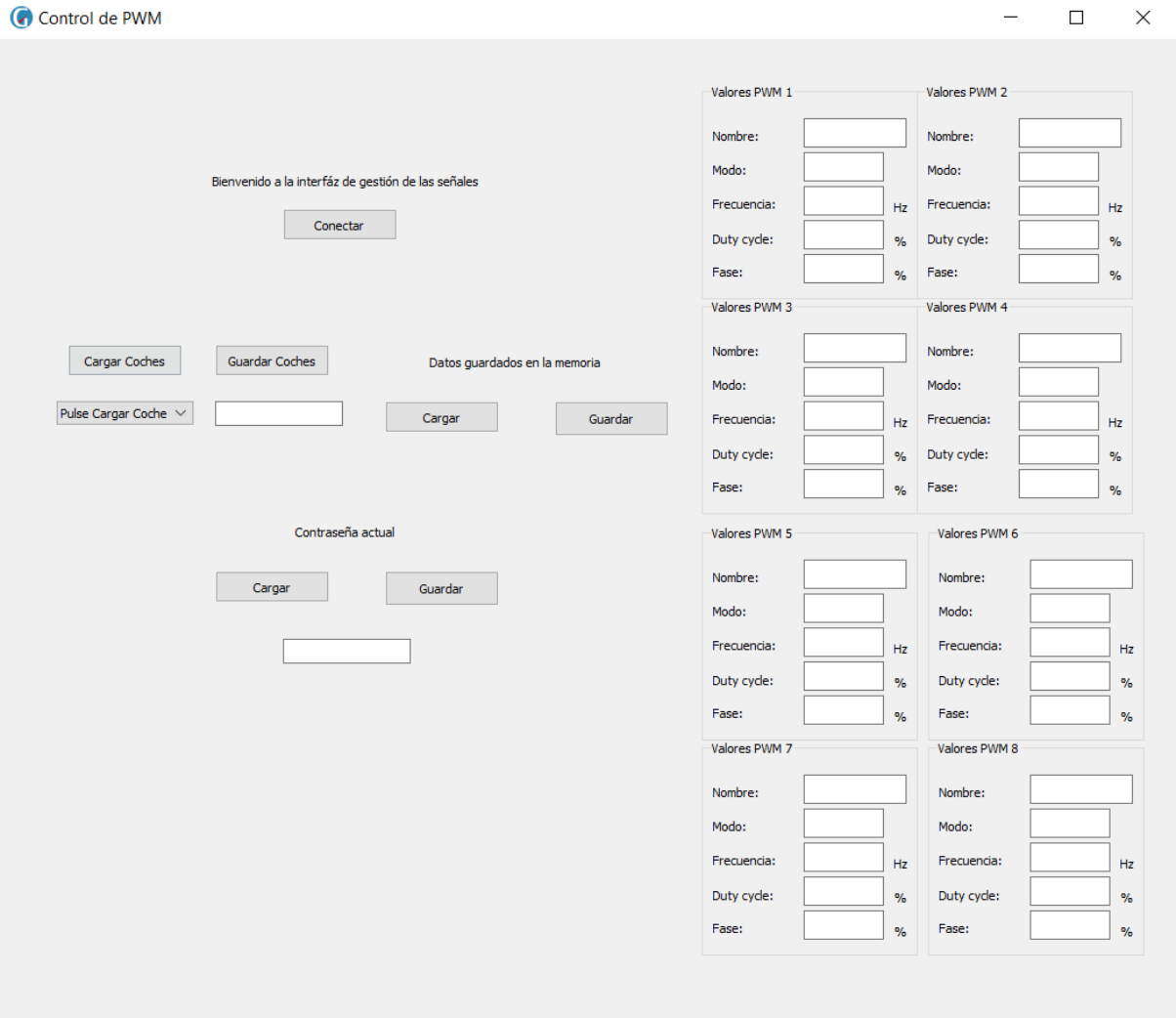

Figura 35: *Diseño de la interfaz gráfica.*

En la parte superior izquierda se muestra el logo de GranaSAT junto con el nombre de la aplicación, un poco más abajo se encuentra una etiqueta para dar la bienvenida al usuario y justo debajo el primer push button que establece la conexión entre el arduino y la interfaz gráfica.

Para acceder a los datos de las señales PWM primeramente se deben pulsar el botón de cargar coches, y al hacerlo, aparecerá el listado de los coches que se encuentran actualmente en la memoria del arduino.

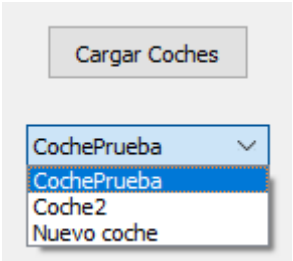

Figura 36: *Ejemplo de los nombres de coche.*

Para añadir un nuevo coche se deberá seleccionar Nuevo coche y añadir en el campo de texto de la derecha el nuevo nombre y pulsar en Guardar Coches. Para modificar una ya existente bastará con seleccionar el coche que queremos modificar.

Una vez tenemos seleccionado el coche que queremos cargar los datos de las señales PWM, pulsaremos el push botton Cargar, apareciendo:

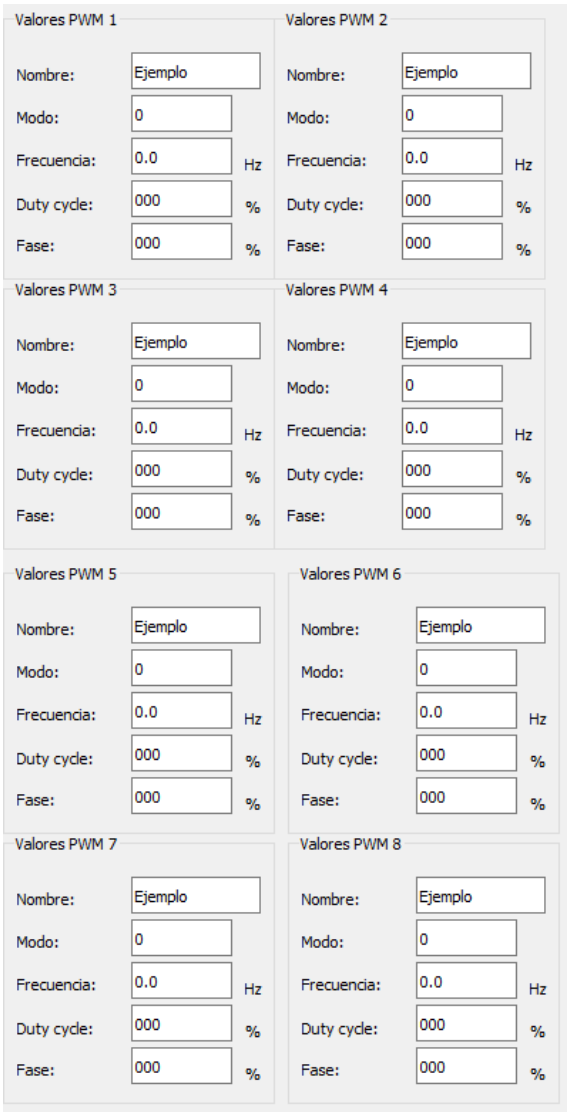

Figura 37*: Ejemplo de los valores de las señales PWM.*

Todos los valores se pueden modificar conforme el **requisito número 3**, para el modo al dejar el ratón sobre el respectivo campo de texto aparece lo siguiente para ayudar al nuevo usuario a elegir correctamente el modo.

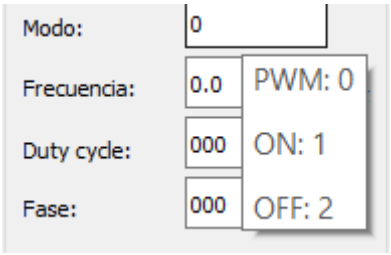

Figura 38: *Ayuda para el usuario al elegir el modo.*

Finalmente para cumplir con el **requisito número 4** he creado un apartado para poder cargar y modificar la contraseña que salta al intentar modificar una señal PWM desde el sistema portátil.

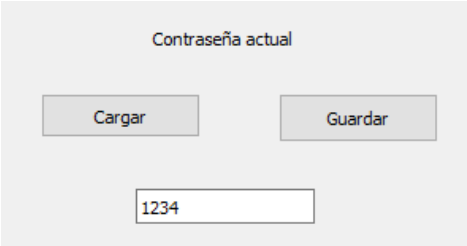

Figura 39: *Apartado para la modificación de la contraseña.*

Una vez vista la interfaz gráfica de manera visual voy a proceder a explicar la lógica que lleva a cabo.

8.2. Implementación de la interfaz gráfica.

Para realizar la lógica de la interfaz gráfica he utilizado el lenguaje **python**, ya que es un lenguaje de alto nivel, muy sencillo y muy utilizado permitiendo utilizar librerías que facilitan el trabajo como son las siguientes:

```
import serial.tools.list_ports
from PyQt5.QtWidgets import QApplication, QMainWindow, QLineEdit, QPushButton, QComboBox, QLabel, QCheckBox, QWidget, QGroupBox
from PyQt5.uic import loadUi
```

Figura 40: *Librerías necesarias para la implementación de la interfaz gráfica.*

Primeramente creamos la clase MainWindow, y definimos la función de inicialización donde declaramos todas las funcionalidades.

```
def __init__(self):
    super(MainWindow, self).__init__()
    loadUi("interfazgrafica.ui", self)  # Carga el archivo.ui

    # COnexión de los botones con las funciones
    self.guardar_button.clicked.connect(self.guardar_datos)
    self.cargar_button.clicked.connect(self.cargar_datos)
    self.conectar_button.clicked.connect(self.conectar_arduino)
    self.guardar_pass.clicked.connect(self.guardar_password)
    self.cargar_pass.clicked.connect(self.cargar_password)
    self.cargar_coche_button.clicked.connect(self.obtener_numCoches)
    self.guardar_coche_button.clicked.connect(self.guardar_coche)
```

Figura 41: *Inicialización de las funciones.*

A continuación explicaré todas las funciones que se han declarado previamente.

- Conectar con el arduino.

Esta es la función principal y que permite la conexión con el arduino, para ello me ayudo de la función **serial.Serial(),** a la cual hay que pasarle por parámetros el puerto serie que está conectado el arduino y el Baud Rate, devolviendo el objeto arduino que nos permite más tarde llamar a las funciones **.write()** y **.read()**.

- Guardar señales PWM.

Para realizar el guardado de los datos en la memoria eeprom tenemos que obtener los datos establecidos en los campos de texto llamando a la función **.text().**

```
# Obtener los valores de la interfaz PWM1
nombre = self.nombre_lineedit.text()
modo = self.modo_lineedit.text()
frecuencia = self.freq_lineedit.text()
duty_cycle = self.duty_lineedit.text()
fase = self.fase_lineedit.text()

frecuencia = frecuencia.replace(".", "")

# Formatear los valores en una cadena
datos_a_enviar1 = f"{nombre},{modo},{frecuencia},{duty_cycle},{fase}"
```

Figura 42: *Lógica implementada para guardar los datos de las señales PWM.*

Una vez que obtenemos los valores, estos se codifican entre comas y las señales PWM entre #. Al principio del array, posición 0, se le añade el código b, y en la posición 1 se carga el valor del combo box para saber que coche se quiere modificar.

```
# Suma total de datos a enviar
index = self.modo_combobox.currentIndex()

datos_a_enviar_total = 'b' + str(index) + datos_a_enviar1 +'#'+ datos_a_enviar2 +'#'+

# Enviar los datos a Arduino
self.arduino.write(datos_a_enviar_total.encode())
```

Figura 43: Suma de los datos para realizar el envío.

Esta codificación se hace para poder realizar la comunicación entre el arduino y la interfaz correctamente y se explica en el archivo **IO_control.** una vez que tenemos todos los archivos realizamos el envío a través de la función **arduino.write()**

- Cargar señales PWM.

La función cargar datos es muy parecida a la de guardar, cambiando el orden de los datos, primeramente envía al arduino el carácter a más el número del coche que queremos obtener los datos.

```
def cargar_datos(self):

    index = self.modo_combobox.currentIndex()

    caracter = 'a' + str(index) + '\n'

    time.sleep(0.5) #Espera para que el arduino no se atore
    self.arduino.write(caracter.encode())
```

Figura 44: Función para cargar los datos en la interfaz gráfica.

Y posteriormente realizamos la lectura de estos datos codificados mediante la función **.readline()**

Una vez obtenidos los datos debemos decodificarlos gracias a la función **.split()** que ofrece python y los añadimos a los respectivos campos de texto.

```
#Decodifico ahora las señales PWM
#PWM1
datos = pwm1.split(",")
nombre = datos[0]
modo = datos[1]
frecuencia = datos[2]
duty_cycle = datos[3]
fase = datos[4]

frecuenciadec = int(frecuencia) /10.0

self.fase_lineedit.setText(fase)
self.freq_lineedit.setText(str(frecuenciadec))
self.duty_lineedit.setText(duty_cycle)
self.nombre_lineedit.setText(nombre)
self.modo_lineedit.setText(modo)
```

Figura 45: *Decodificación de las señales PWM.*

- Cargar y guardar la lista de los coches.

Para obtener la lista de los coches que existe me ayudo de las funciones **obtener_numCoches()** y **cargar_coches()**, y actualizo el valor del combobox con el nuevo resultado, para entenderlo mejor voy a ilustrar un diagrama de flujo.

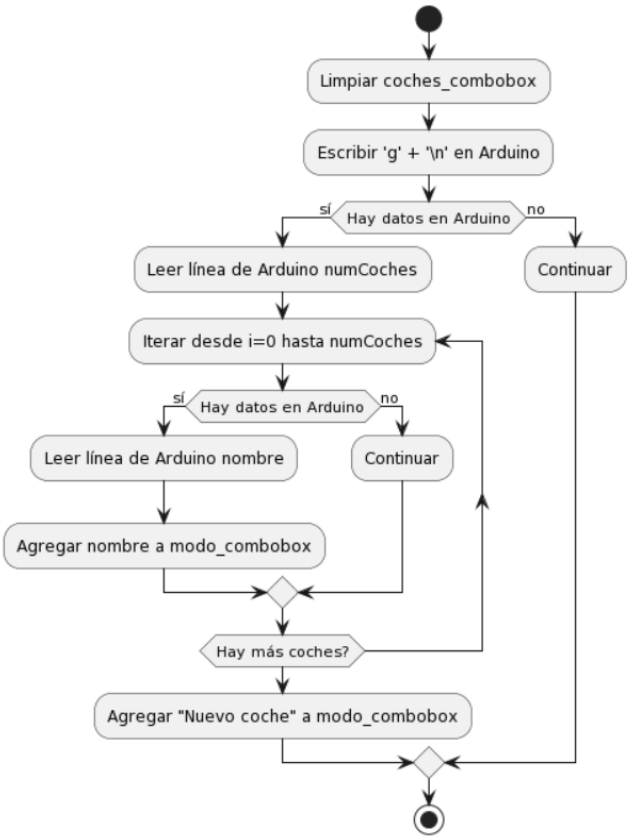

Figura 46: *Diagrama de flujo para guardar la lista de coches.*

- Cargar y guardar la contraseña.

Para la función de guardar y cargar contraseña es muy sencillo, ya que se envía al Arduino el carácter **c** y este responde con la contraseña que se añade al campo de texto de la contraseña y para el guardado se envía el carácter d junto la contraseña del campo de texto.

9. Tests

En el desarrollo nuestro proyecto y software en general, las pruebas representan una fase crítica que asegura la calidad, la fiabilidad y el rendimiento del producto final. Este apartado delineará la estrategia de pruebas integral diseñada para validar cada componente de nuestro sistema, desde módulos individuales como la modificación de las señales PWM, la gestión de la EEPROM, la interfaz de usuario y la lógica de control hasta el sistema completo en funcionamiento.

La metodología de pruebas se ha seleccionado cuidadosamente para adaptarse a nuestro proyecto, tomando en cuenta las restricciones de hardware y los requerimientos de tiempo real. Los objetivos específicos de nuestro enfoque de pruebas son:

1. **Verificación de Funcionalidad:** Asegurar que cada función del sistema opera de acuerdo con los requisitos especificados.
2. **Validación de Rendimiento:** Confirmar que el rendimiento del sistema cumple con las expectativas y tiene un funcionamiento eficiente.
3. **Robustez y Tolerancia a Fallas:** Evaluar la capacidad del sistema para manejar situaciones de error y recuperarse de fallos imprevistos.
4. **Seguridad y Conformidad:** Verificar que el sistema cumple con los estándares de seguridad pertinentes y protege adecuadamente los datos.

El proceso de pruebas abarca desde pruebas unitarias que aíslan y verifican la funcionalidad de pequeñas piezas de código, hasta pruebas de integración que examinan cómo esos componentes trabajan juntos, y pruebas de sistema que evalúan el comportamiento del sistema completo. Además, pruebas de aceptación aseguran que el producto final cumple con los criterios de aceptación del cliente y está listo para su despliegue.

El enfoque iterativo y basado en la **metodología ágil** significa que las pruebas no son una fase posterior al desarrollo, sino un proceso continuo que acompaña cada

etapa del ciclo de vida del software. Esto nos permite identificar y resolver problemas de manera temprana, mejorando la calidad general del software.

En todas las pruebas se ha realizado un vídeo que comprueba a nivel de usuario que se puede realizar todos los requisitos.

9.1. Requisito 1.

Para poder cumplir con este requisito, se ha creado en el laboratorio de GranaSAT una caja de madera de 0,5 cm de grosor capaz de soportar caídas y golpes contundentes, además se ha creado un protector de plástico para la pantalla del LCD para evitar cualquier arañazo o entrada de polvo al sistema.

En cuanto a las pruebas de durabilidad, se ha sometido el Arduino a un benchmarking de estrés para componentes electrónicos y se ha dejado encendido el sistema durante más de 48h, comprobando así que no hubiese ningún problema térmico y pasando este control de calidad.

Figura 47: *Producto final.*

9.2. Requisito 2.

Las pruebas realizadas para verificar la funcionalidad de navegación han confirmado que los usuarios pueden acceder sin problemas a todos los menús y funciones previstos en el diseño. Estas pruebas aseguran una experiencia de usuario fluida y coherente al interactuar con la interfaz del sistema.

9.3. Requisito 3.

En esta prueba se ha comprobado que se pueden modificar los parámetros de las señales PWM, pudiendo alcanzar todos los rangos que las señales aceptan.

Para poder acceder al vídeo en el que se comprueba el funcionamiento (Tests, s.f.).

9.4. Requisito 4.

Para comprobar este requisito se modificó una señal PWM, accediendo al menú contraseña. Una vez dentro, se inserta la contraseña por defecto, 1234, permitiendo modificar la señal PWM. Posteriormente se accede desde la aplicación en el ordenador y se modifica la contraseña, y al repetir el proceso de modificar una señal PWM, aparece el menú de contraseña, al poner la anterior contraseña nos sale una alerta de contraseña errónea y al introducir la nueva, si que nos permite modificar la señal PWM. Cumpliendo por lo tanto con el requisito 4.

8. Añadir un nombre que compone las 8 señales PWM.
9. Diseñar una interfaz gráfica en ordenador que permita tanto la carga y guardado de todas las propiedades de las señales PWM y su modificación.

9.5. Requisito 5.

Este requisito nos permite guardar los datos de las señales PWM para posteriormente cargarlos y utilizarlos. Para cumplir con este requisito se ha guardado en la señal PWM 1 el valor de frecuencia 123.4, la fase 12 y el duty cycle 12 y se ha guardado en el coche Primero, al cargar el coche Segundo, en la señal PWM 1 se nos carga el valor por defecto 0 en todos los datos.

Al volver a cargar el coche Primero, en la señal PWM 1 nos vuelven a aparecer los datos que guardamos previamente, frecuencia 123.4, fase 12 y duty cycle 12. Comprobando así el funcionamiento del requisito 5.

9.6. Requisito 6 y 7.

Para comprobar estos requisitos se ha realizado un vídeo donde se observa los resultados deseados (Tests, s.f.).

Bloque 3: Conclusiones

10. Conclusiones

Este proyecto ha logrado con éxito el desarrollo de un sistema portátil basado en Arduino MEGA, que incorpora una pantalla LCD y un menú interactivo controlado por un rotary encoder. Se destacan dos logros principales: la capacidad de almacenar y ajustar señales PWM para controlar faros de vehículos de nueva generación y el desarrollo de una aplicación con interfaz gráfica para facilitar la modificación de estas señales desde computadoras.

10.1. Evaluación del proyecto

El proyecto ha cumplido sus objetivos iniciales, proporcionando una solución innovadora y práctica para el ajuste de faros de vehículos. El uso de Arduino MEGA ha demostrado ser efectivo, aunque se identificaron desafíos, como la integración de hardware y software, y la precisión en el control de las señales PWM. Las mejoras en la interfaz de usuario de la aplicación han permitido un manejo más sencillo y accesible para los usuarios.

El sistema tiene un potencial significativo para mejorar el ajuste de faros en la industria automotriz, especialmente en vehículos de nueva generación. Su portabilidad y la interfaz gráfica de usuario facilitan su uso, lo que podría llevar a una adopción más amplia por parte de técnicos y entusiastas del automovilismo. Este proyecto aporta un avance notable en el campo del control de iluminación vehicular.

10.2. Relativo a la planificación temporal

La gestión del tiempo ha representado un desafío significativo en este proyecto, principalmente debido a dos factores críticos. En primer lugar, la magnitud del trabajo ha sido considerablemente mayor a cualquier otro que haya abordado anteriormente, lo que dificultó la estimación precisa del tiempo necesario para completar cada sección. En segundo lugar, mi situación personal requirió que equilibrara las responsabilidades del proyecto con un horario laboral completo. Esta circunstancia

limitó mi capacidad para asignar períodos prolongados de atención ininterrumpida a los desafíos emergentes.

Después de realizar el proyecto me he dado cuenta que es muy importante realizar una buena planificación y no abarcar más de lo que uno puede.

10.3. Aprendizaje

El proceso de desarrollo de este proyecto ha sido una experiencia enriquecedora y gratificante a nivel personal y académico. Me ha proporcionado la oportunidad de adentrarme en nuevos horizontes del conocimiento, dominando lenguajes de programación como Python y explorando el fascinante mundo del diseño de interfaces de usuario. También he aprendido a interpretar los documentos datasheet afinando mi habilidad para extraer información pertinente y aplicarla de manera práctica.

11. Trabajo futuro

Una vez terminado el proyecto, hay algunas mejoras que se podrían realizar en futuras versiones para obtener un producto más optimizado (ágil, s.f.).

- **Interfaz gráfica:** Actualmente los usuarios se fijan mucho en la interfaz gráfica de un programa, ya que es lo único que ven, por lo que una mejora podría ser la de realizar una interfaz gráfica más sencilla e intuitiva.

- **Cambio LCD:** Hilando con la interfaz gráfica una posible mejora de cara a una mejor representación de los datos y funcionalidades sería la de mejorar la actual pantalla LCD 4x16 por una de mayor resolución.

- **Optimización memoria EEPROM:** Actualmente para el guardado de las señales PWM en la memora EEPROM se utilizan estructuras de tamaño fijo simulando una **paginación simple** que son más sencillas de utilizar e implementar, pero suponen un consumo mayor de la memoria, impidiendo que se puedan guardar más valores. Una mejora sería implementar una **paginación multinivel**.

Bibliografía

Metodología ágil, M. (n.d.). *https://www.redhat.com/es/devops/what-is-agile-methodology*.

Designer, P. y. (n.d.). From https://hetpro-store.com/TUTORIALES/qt-creator-y-arduino-qserialport/

Designer, Q. (n.d.). *Medium*. From https://medium.com/@hektorprofe/primeros-pasos-en-pyqt-5-y-qt-designer-programas-gr%C3%A1ficos-con-python-6161fba46060

Diagramas. (n.d.). From https://www.planttext.com/

EEPROM. (n.d.). *Wikipedia*. From https://es.wikipedia.org/wiki/EEPROM

GranaSAT. (n.d.). From https://granasat.space/

MEGA, A. (n.d.). From https://lab.bricogeek.com/tutorial/guia-de-modelos-arduino-y-sus-caracteristicas/arduino-mega-2560

Tests, V. (n.d.). *Drive*. From https://drive.google.com/drive/folders/1v-_bUCPdgGzk3hkEdOS8NUUwwveHMlSN?usp=drive_link

UART. (n.d.). From https://docs.arduino.cc/micropython-course/course/serial?queryID=855ebfe2083f449e618e956769587661