

Grado Universitario en Ingeniería Informática  
Curso Académico 2024-2025

*Trabajo Fin de Grado*

Diseño de un salpicadero virtual de un vehículo en un PC usando microcontrolador

Jose Crespo Cantador

Tutores

Jose Enrique Suarez Pascual

Colmenarejo, 08 de Septiembre de 2025



Esta obra se encuentra sujeta a la licencia Creative Commons **Reconocimiento – No Comercial – Sin Obra Derivada**

**RESUMEN**

Blablablalbalblabla

**AGRADECIMIENTOS**

**ÍNDICE DE CONTENIDOS**

[1 1. INTRODUCCIÓN 14](#_Toc207216503)

[1.1 Motivación del Trabajo. 14](#_Toc207216504)

[1.2 Objetivos 14](#_Toc207216505)

[1.2.1 Microcontrolador 14](#_Toc207216506)

[1.2.2 Conexión Alámbrica 14](#_Toc207216507)

[1.2.3 Salpicadero Virtual 14](#_Toc207216508)

[2 ESTADO DEL ARTE 15](#_Toc207216509)

[3 MARCO REGULADOR 16](#_Toc207216510)

[4 DESARROLLO DEL SISTEMA 17](#_Toc207216511)

[4.1 Requisitos 17](#_Toc207216512)

[4.2 Componentes 17](#_Toc207216513)

[4.2.1 Microcontrolador 17](#_Toc207216514)

[4.2.2 Sensor de ultrasonidos 18](#_Toc207216515)

[4.2.3 Sensor de temperatura 19](#_Toc207216516)

[4.2.4 Sensor fotoeléctrico 20](#_Toc207216517)

[4.2.5 Sensor de aceleración 20](#_Toc207216518)

[4.2.6 potenciómetro 21](#_Toc207216519)

[4.3 Implementación 21](#_Toc207216520)

[4.3.1 Microcontrolador 21](#_Toc207216521)

[4.3.2 Salpicadero digital 25](#_Toc207216522)

[4.4 Resultados 30](#_Toc207216523)

[5 ANÁLISIS DEL PROYECTO 31](#_Toc207216524)

[5.1 Contexto socioeconómico 31](#_Toc207216525)

[5.2 Presupuesto 31](#_Toc207216526)

[6 CONCLUSIONES 31](#_Toc207216527)

**ÍNDICE DE FIGURAS**

[Figura 4.1 Microcontroladora STM32F407G-DISC1 17](#_Toc207216488)

[Figura 4.2 Sensor de ultrasonidos HC-SR04 18](#_Toc207216489)

[Figura 4.3 Diagrama de tiempos sensor HC-SR04 18](#_Toc207216490)

[Figura 4.4 Sensor de temperatura PT100 19](#_Toc207216491)

[Figura 4.5 Conversor de señal RTD a digital 19](#_Toc207216492)

[Figura 4.6 Sensor fotoeléctrico Daokai LM393 20](#_Toc207216493)

[Figura 4.7 Rueda perforada con 20 ranuras 20](#_Toc207216494)

[Figura 4.8 Vista de los ejes de detección de aceleración 21](#_Toc207216495)

[Figura 4.9 Manómetro del salpicadero digital 26](#_Toc207216496)

[Figura 4.10 Indicador de distancia de seguridad del salpicadero digital 27](#_Toc207216497)

[Figura 4.11 Indicador de distancia recorrida del Cuadro de información 28](#_Toc207216498)

[Figura 4.12 Indicador de temperatura del habitáculo del Cuadro de información 28](#_Toc207216499)

[Figura 4.13 Manómetro de nivel de combustible del Cuadro de información 28](#_Toc207216500)

[Figura 4.14 Manómetro de temperatura del motor del Cuadro de información 28](#_Toc207216501)

[Figura 4.15 Salpicadero digital completo 30](#_Toc207216502)

**ÍNDICE DE TABLAS**

**LISTA DE ABREVIATURAS**

# 1. INTRODUCCIÓN

## Motivación del Trabajo.

Desde hace aproximadamente dos décadas los vehículos han ido introduciendo el uso de pantallas y otros elementos digitales para proporcionar más y mejor información al conductor y sus ocupantes. Este proceso de digitalización del vehículo se ha visto acelerado por la introducción de tecnologías que permiten la comunicación ente nuestros dispositivos móviles y las consolas de los vehículos, hasta el punto de sustituir elementos analógicos como el cuentarrevoluciones o el velocímetro.

Aunque los elementos del salpicadero hayan sido digitalizados, los sensores analógicos siguen formando parte de todos los vehículos y ahora más que nunca es importante conocer como funciona el proceso de recolección y tratamiento de estos datos, esencial para el desarrollo de tecnologías de conducción autónoma y seguridad vial.

Para que el vehículo sea seguro y el conductor tenga toda la información relevante, esta debe ser mostrada de una forma clara y actualizada. Y para esto tanto el sistema encargado de mostrar dicha información y el encargado de recogerla deben trabajar de forma sincronizada y robusta.

Este proyecto tiene como objetivo recoger información de sensores a través de un microcontrolador programable y presentarlo en un salpicadero digital en otro dispositivo con una conexión alámbrica aplicando conocimientos adquiridos en el grado de Ingeniería Informática

## Objetivos

El proyecto se puede dividir en tres objetivos

### Microcontrolador

Este objetivo tiene como fin la configuración adecuada del microcontrolador, así como del desarrollo del código necesario para la obtención de la información requerida para el salpicadero virtual.

### Conexión Alámbrica

En este objetivo buscamos desarrollar la capacidad de comunicar de forma alámbrica los dispositivos de los que disponemos, es decir, el microcontrolador y el PC donde se dibujara el salpicadero virtual.

### Salpicadero Virtual

Por último, este objetivo desarrollara una interfaz simple que muestre la información obtenida por el microcontrolador de forma clara y rápida.

# ESTADO DEL ARTE

En 1968 Volkswagen introduce un sistema de inyección electrónica en su modelo Tipo III, que usaba unos 25 transistores para realizar un control preciso de la inyección de la gasolina al motor. No es hasta los años 70 que se empieza a usar microcontroladores de forma más general, para cumplir con el aumento de regulaciones a las emisiones de los vehículos y la eficiencia de los motores.

Actualmente los vehículos constan de varios ordenadores distintos que controlan características diferentes del vehículo. Estos ordenadores están interconectados y controlan desde las medidas de seguridad como el ABS hasta los sistemas de conducción autónoma.

La hibridación y electrificación de los vehículos modernos que intentan buscar un futuro sin emisiones han creado nuevos retos. Como la gestión de un motor térmico junto con uno eléctrico en el caso de los híbridos o la administración de las baterías en el caso de ambas tecnologías.

# MARCO REGULADOR

Para el desarrollo del proyecto es importante entender y tener en cuenta el marco regulador que nos afecta en el diseño y funcionamiento del sistema.

El Reglamento 2019/2144 de la Unión Europea establece los requisitos de homologación para los distintos sistemas de los vehículos. Centrándose en los sistemas avanzados de asistencia a la conducción y la protección de las personas.

También debemos considerar la ISO 26262 que marca un estándar para la seguridad funcional de vehículos y los sistemas Eléctricos y Electrónicos.

Dado que la mayor parte del proyecto se centra en el desarrollo de código debemos tener en cuenta estándares establecidos en la industria. El estándar MISRA C establece una serie de mejores prácticas para el desarrollo de código en el lenguaje C dirigido a la industria de la automoción. Por otro lado, se debe tener en cuenta el estándar PEP 8 dirigido al desarrollo de código en el lenguaje Python. El cual trata de establecer una serie de buenas prácticas para la creación de código más mantenible y seguro.

# DESARROLLO DEL SISTEMA

Una vez establecido el contexto del proyecto, se pueden empezar a desarrollar las distintas partes que conforman el sistema.

En este apartado se identifican los requisitos establecidos para el desarrollo del proyecto, se indican los componentes necesarios y se expone el proceso seguido para el desarrollo.

## Requisitos

Para cumplir con el objetivo del proyecto se debe de presentar la siguiente información al usuario, y dicha información debe de provenir del microcontrolador de forma alámbrica.

* Velocidad
* Distancia Recorrida
* Revoluciones por minuto
* Temperatura exterior
* Temperatura interior
* Señal de seguridad
* Nivel de llenado de deposito

## Componentes

### Microcontrolador

Se estableció que se debía usar un microcontrolador programable para el desarrollo de la parte relacionada con la obtención de las mediciones de sensores. Esto descartaba placas como la raspberry pi, ya que, aunque era posible usarse como placa para el desarrollo del proyecto se decidió que era demasiado potente para este proyecto.

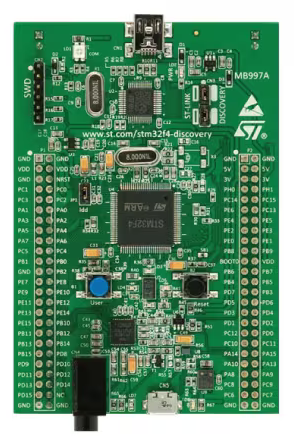
Por eso se deicidio usar la placa de desarrollo STM32F407G Discovery. Las principales razones para elegir esta placa son: la inclusión de una herramienta de reprogramación y depuración llamada ST-LINK, la gran cantidad de pines GPIO, y la posibilidad de configurar una conexión serie.

Figura 4.1 Microcontroladora STM32F407G-DISC1

### Sensor de ultrasonidos

Se usarán los sensores de ultrasonidos HC-SR04 para simular los sensores de proximidad de los vehículos, proporcionando de esta manera la señal de seguridad. Se usarán 2 unidades de este sensor para recoger la distancia que haya ha obstáculos como otros vehículos, personar u objetos. Estos componentes representaran los distintos grupos de sensores que se suelen encontrar en la parte delantera y trasera de los vehículos.



Figura . Sensor de ultrasonidos HC-SR04

Estos sensores tienen un rango efectivo que va de los 2cm a los 400cm, con una precisión de 3mm. Esto los convierte en una opción razonable y asequible para el desarrollo del proyecto.

El sensor consta de 4 pines, Vcc donde debe recibir una alimentación de 5V, un pin Gnd para conectarlo a tierra, un pin Trig o Trigger para accionar el sensor y un pin Echo donde se recibirá la respuesta.

Para explicar su funcionamiento debemos ver el diagrama de tiempos proporcionado por el fabricante.

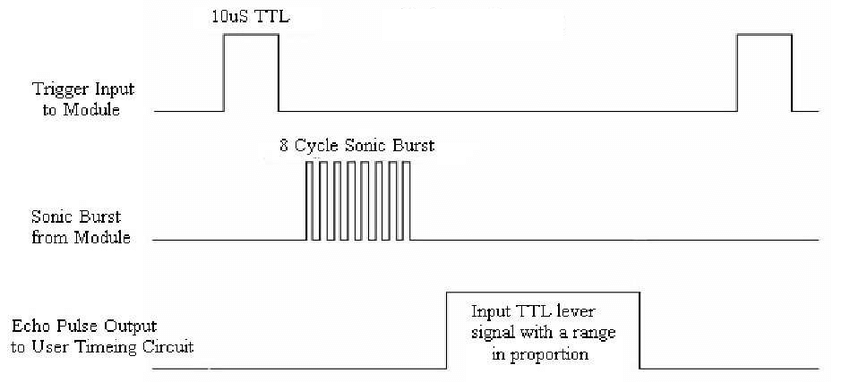


Figura . Diagrama de tiempos sensor HC-SR04

Para iniciar la medición por parte del sensor deberemos de mandar una señal de 10μs al pin Trig, tras esto el sensor lanza 8 pulsos de ultrasonidos a una frecuencia de 40KHz. Por ultimo el sensor devuelve una señal en el pin Echo que indica el tiempo que ha tardado en recibir el pulso de vuelta. Entre cada señal se mande al sensor deben de transcurrir al menos 60 milisegundos según el fabricante.

Con esta información podemos calcular la distancia en cm que han recorrido los pulsos de ultrasonidos emitidos por los sensores. Para ello usaremos la siguiente formula.

Siendo *tiempo* el valor en microsegundos que ha durado la señal recibida en el pin echo y *1/58* el factor de conversión a cm proporcionado por el fabricante.

### Sensor de temperatura

Para el sensor de temperatura se barajó el uso del modelo DHT11 muy usado en proyectos de este tipo. Pero se descarta por su alta latencia, ya que generaba un bloqueo en la ejecución de más de un segundo.

Finalmente se decidió usar un termistor, es decir, una resistencia variable, cuyo valor de resistencia varía en función de la temperatura de esta. El termistor usado se trata de un PT100 de 3 pines. Al tener 3 pines mitiga parte del error introducido por la resistencia añadida del cable que va de la punta del sensor donde se encuentra la resistencia hasta los terminales donde se conecta el sensor.

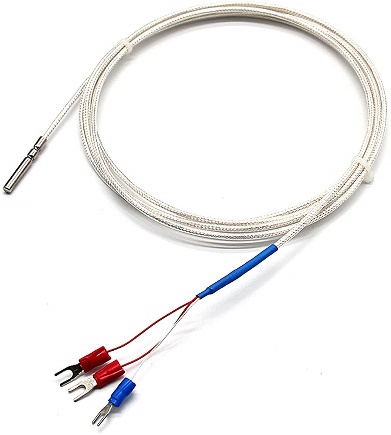


Figura . Sensor de temperatura PT100

Para poder usar este tipo de sensor se opto por usar un conversor de valor de resistencia a digital, conocido como Conversor RTD.

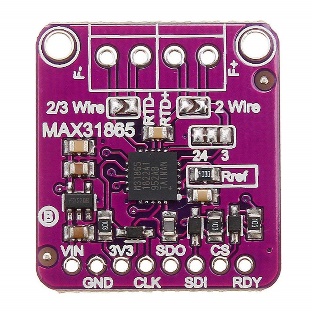


Figura . Conversor de señal RTD a digital

### Sensor fotoeléctrico

Este sensor se usará para medir las revoluciones por minuto a las que gira el motor. Se trata de un sensor DAOKAI LM393, este usa un haz de luz infrarroja emitido contra un receptor para detectar las interrupciones del haz.

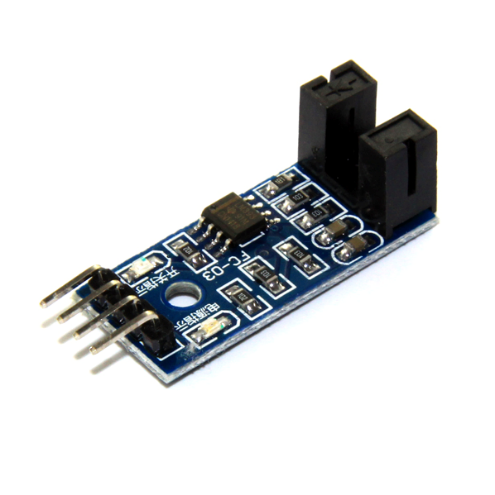


Figura . Sensor fotoeléctrico Daokai LM393

Si añadimos una rueda perforada con 20 huecos para interrumpir el haz del sensor, sabremos cuando ocurre una revolución.



Figura . Rueda perforada con 20 ranuras

La rueda se conectará a un motor de corriente continua, que no se incluye como parte del proyecto ya que no será controlado por ninguno de los sistemas a desarrollar y simplemente se usará para comprobar el correcto funcionamiento de este sensor concreto.

### Sensor de aceleración

Para obtener información como velocidad y distancia recorrida se va a usar el sensor de aceleración de 3 ejes del que dispone la placa. El acelerómetro de 3 ejes LIS3DSHTR, el cual proporciona la aceleración detectada en 3 ejes (x, y, z).

Diagrama, Dibujo de ingeniería

El contenido generado por IA puede ser incorrecto.

Figura . Vista de los ejes de detección de aceleración

Para poder hacer uso de la aceleración primero debemos de tener en cuenta la aceleración ejercida por la gravedad terrestre. Además, hay que tener en cuenta el factor del ruido en las mediciones del acelerómetro, el cual habrá que mitigar.

### potenciómetro

Para simular el sensor que mide la cantidad de combustible que hay en un tanque vamos a usar un potenciómetro. Ya que si reducimos los componentes de este tipo de sensor llegamos a que básicamente se trata de un potenciómetro, es decir, una resistencia variable conectada a un flotador.

## Implementación

### Microcontrolador

Para el desarrollo del código se ha empleado el entorno de desarrollo integrado STM32CubeIDE. El cual incluye las herramientas necesarias para compilar el código C y permite configurar más fácilmente el microcontrolador

#### **Sensores de proximidad**

Como se expone en el apartado 4.2.2 donde se describe el funcionamiento del sensor HC-SR04, hay que esperar un mínimo de 60 ms entre cada medición del sensor. Para tener un margen de sobra la lógica que ejecuta el ciclo de lectura del sensor se ejecuta solo si han pasado más de 100 ms. Para esto se hace uso de la función *HAL\_GetTick()*, que forma parte de la capa de abstracción de Hardware proporcionada como librería por parte del entorno de desarrollo. La función devuelve el número de milisegundos que han pasado desde el inicio del sistema. Cada vez que se ejecuta el código se guarda el valor actual que devuelve la función mencionada, de este modo en cada iteración del bucle principal se puede comparar el valor en ese instante con el de la última ejecución de la mencionada sección de código.

*if (HAL\_GetTick() - delay\_hcsr04 >= 100)*

A ambos sensores se les manda la misma señal de Trigger, de esta forma ambos toman la medición de una forma coordinada. Esta señal se manda a través del Timer TIM11 de la placa. Este Timer se ha configurado para que un tick dure 2.5μs y el pulso se mantenga a lo largo de 4 ticks, lo que equivale a un pulso de 10μs. El Timer realiza este proceso cada 100ms.

Estos son los cálculos utilizados para configurar el Timer correctamente:

Donde *APB2* es la frecuencia el reloj al que está conectado el Timer TIM11 y *Ttick* es el tiempo que dura un tick. *fTIM11* es la frecuencia con la que queremos que se complete un ciclo, en este caso es 10Hz, que son 100ms. Tras fijar esos valores se calcula el *Prescaler* y el *CounterPeriod* que representan los valores de configuración del Timer que tiene el mismo nombre. De esta forma quedaría configurado el comportamiento del Trigger.

Para la medición de la duración de la señal que indica la distancia medida por los sensores necesitaremos usar otro Timer. En este caso el TIM8 el cual se configura con *Reset Mode* en el apartado *Slave Mode,* esto quiere decir que se configura para reiniciarse cada vez que detecta una señal, en este caso la señal mandada por el TIM11. De esta forma siempre estará listo para recibir la señal generada por el sensor. Al tener dos sensores también se configuran 2 canales de los cuatro disponibles como *Input Capture* *direct*, lo que significa que usaremos los pines predefinidos como puntos de lectura. Para el TIM8 estos pines son PC7 y PC6.

Finalmente llamaremos a la función *HAL\_TIM\_ReadCapturedValue()*, donde se especifican como parámetros la referencia del Timer que se va a leer y el canal de dicho Timer. Tras dividir el valor leído entre 58 se obtiene la distancia en centímetros que hay delante del sensor.

#### **Sensores de temperatura**

Los sensores de temperatura están conectados a un conversor RTD como ya se explicó previamente. La comunicación entre estos conversores y la placa se realiza a través de la interfaz de comunicación *SPI*. La temperatura del interior del vehículo es recibida a través de la interfaz *SPI2*, mientras que la temperatura del motor es obtenida de la interfaz *SPI3*. Para configurar estas interfaces hay que poner el modo *Full-Duplex Master* y configurar la opción *Clock Polarity* como *Low* junto con *Clockl Phase* como *2 Edge*. Esto significa que leeremos los datos durante el flanco de bajada de la señal.

Se usa la librería *max31865* que nos permite inicializar fácilmente la interfaz SPI y realiza la conversión del valor de resistencia a Celsius directamente. Desde el bucle principal se llama a la función *MAX31865\_readTempC()* que da la medición actual, se comprueba que el valor no es incorrecto y se pasa por un filtro para compensar posibles fallos de lectura y así suavizar los valores de temperatura.

#### **Sensor de aceleración**

El acelerómetro incluido en la placa está conectado al *SPI1*, para su lectura solo es necesario configurarlo en modo *Full-Duplex Master*. Para interactuar con el acelerómetro y leer los valores se usa la librería *LIS3DSH* que permite la interacción con el sensor de aceleración.

Para obtener la aceleración actual se llama a la función *LIS3DSH\_GetDataScaled()* que nos devuelve los valores de la aceleración en cada eje. Como se ve en la Figura 4.8 y por la orientación del sensor en la placa, el movimiento del vehículo se ejerce en el eje *y*.

La lectura de la aceleración se devuelve como la milésima parte de *g*. Para la conversión a *m/s2* solo es necesario multiplicar por 9.81 y dividir entre 1000. En este caso se multiplica por 0.00981 simplificando el código.

Para obtener una lectura sin interferencias de la gravedad terrestre la alineación de la placa debería de ser perpendicular a esta. Es prácticamente imposible cumplir con esto, para mitigar esto se ignoran las aceleraciones que estén por debajo de los 0.5m/s2.

Además, para mitigar el posible ruido del sensor y suavizar las lecturas se aplica un filtro de paso bajo, atenuando los posibles fallos de lectura que suelen presentarse como picos en los valores de aceleración, ya sea con valor positivo o negativo.

Una vez se tiene un valor aceptable de la aceleración, se pasa a el cálculo de la información derivada de la aceleración. Vamos a derivar la velocidad en el eje *y*, gracias a esto podremos obtener la distancia recorrida.

Para el cálculo de la velocidad simplemente se obtiene el diferencial de tiempo entre cada calculo, para multiplicarlo por la aceleración y de esta forma conseguir la velocidad. La velocidad se guarda en una variable que acumula el valor, de esta manera se aplican los cambios de velocidad cuando se acelera o frena.

Una vez se tiene la velocidad se puede calcular la distancia recorrida, solo hay que obtener el diferencial de tiempo de cada iteración del bucle y multiplicarlo por el valor absoluto de la velocidad, ya que la velocidad negativa solo indica la dirección del movimiento.

#### **Sensor fotoeléctrico**

Para el sensor fotoeléctrico solo hace falta configurar un pin como *External Interrupt*, es decir que cuando el sensor detecte un corte en el haz de luz mandara una interrupción al pin que se ha configurado.

En el archivo generado por el propio entorno de desarrollo encargado de las rutinas para las interrupciones, se declara una variable como contador, a la que se va sumando valores cada vez que ocurre una interrupción en el pin configurado.

Finalmente, en el archivo principal declaramos una función que se ejecute cada vez que se genera una interrupción de algún Timer. En este caso el TIM6, que está configurado para generar una interrupción cada segundo. Cuando eso ocurra solo tendremos que multiplicar el numeró de interrupciones detectadas por el sensor por 60 segundos y dividirlo entre las 20 ranuras que tiene una revolución. Esto último se reduce a una multiplicación por 3. Por último se reinicia a 0 el contador de interrupciones del sensor fotoeléctrico.

#### **Potenciómetro**

El potenciómetro dará el valor para el nivel de combustible para esto hay que usar un conversor de analógico a digital o *ADC.* *HAL\_ADC\_GetValue()*

#### **Comunicación alámbrica**

Para la comunicación alámbrica se usa la interfaz *USART2* en modo asíncrono. Y se configura con los siguientes parámetros. *Baud Rate* en 115200 Bits/s, un *Word Length* de 8 Bits, sin Bit de paridad y el parámetro de *Stop Bits* a uno.

Vamos a transmitir un total de 8 variables, de las cuales 6 son de tipo *float* que ocupan 4 bytes y las otras 2 son de tipo *uint32\_t* que ocupa un tamaño de 8 bytes. En total transmitiremos un total de 40 bytes. Los datos se transmitirán como un *string* de caracteres separados por comas, para ello guardaremos los valores en un *buffer* de 41 bytes. Ese byte extra se añade para cubrir el espacio que ocupen las comas y otros caracteres especiales. Las mediciones siguen el siguiente orden: distancia frontal, distancia trasera, temperatura exterior, temperatura interior, revoluciones por minuto, velocidad, nivel de combustible, distancia recorrida.

### Salpicadero digital

El salpicadero digital se desarrolla con *Visual Studio Code* y esta desarrollado en el leguaje Python. Se usa este leguaje por la familiaridad que se posee con su sintaxis y por integrar la mayoría de las herramientas necesarias para el desarrollo del proyecto, como las librerías para crear una interfaz gráfica. Se usa la librería estándar de Python *tkiner* para esta interfaz.

Las herramientas principales que se usan de esta librería son la estructura *grid* que permite ordenar las partes de nuestra interfaz siguiendo un patrón de filas y columnas. El otro componente será el denominado *Canvas* que es la forma en la que agruparemos los distintos objetos gráficos que forman cada pieza de la interfaz.

#### **Manómetro de rpm y velocidad**

Para representar las revoluciones por minuto y la velocidad obtenida a través de la comunicación con el microcontrolador, se usa la figura del manómetro. Ambas magnitudes se representan de esta manera en el mundo real, por eso se usa la misma representación.

Con el fin de generar esta representación se crea una clase llamada *GaugeCanvas* que hereda de la clase *Canvas* definida en la librería que estamos usando. Esta representación está formada por un arco con la escala y una aguja que indica el valor en esa escala.

La librería incluye una función *create\_arc()* que permite dibujar una elipse dando como parámetros dos tuplas de puntos que definan el rectángulo que la contiene. También se define que parte del arco de la elipse se quiere dibujar, en este caso se quiere obtener algo similar a un semicírculo, es decir, queremos la parte superior de la elipse que está situada en un arco de 0º a 180º.

La otra parte de este arco es la escala, para dibujarla se colocan líneas usando la función *create\_line()*. Para poder dibujar las marcas es necesario calcular las coordenadas de los puntos que indican el inicio y el fin de cada recta, y para poder hacer ese cálculo es necesario obtener el ángulo al que va colocada cada marca. Esto se hace usando la siguiente formula:

Donde *θmarca* es el ángulo de la recta que representa la marca, *θinicial*indica donde empieza el arco, *i* representa el valor que se va a representar en la escala, *N* representa el valor máximo que puede tener la escala, y *Δθ* indica la extensión en grados del arco.

El resultado se transforma a radianes, ya que la librería *math* de Python espera radianes para realizar cálculos y no grados.

Para calcular la distancia aplicamos las siguiente formulas:

Donde *x* e *y* son las coordenadas del punto, *x0* e *y0* son las coordenadas del centro del arco, y *θ* es el ángulo previamente calculado.

De esta forma obtenemos los puntos sobre el arco que se dibuja, y para obtener los puntos que completan las rectas que representan la escala solo debemos hace el mismo calculo anterior, modificando el radio para crear un arco más pequeño. Por último, para añadir las unidades en la escala se usan las mismas formulas, pero con un radio menor para que estas se puedan leer.

Si se aplican estas fórmulas, primero para calcular el ángulo para un valor de velocidad o rpm, y luego se calcula un punto para ese ángulo con un radio menor que los anteriores, se puede dibujar una recta desde el centro hasta ese punto. Obteniendo de esta manera la aguja del manómetro.

Un conjunto de letras blancas en un fondo blanco

El contenido generado por IA puede ser incorrecto.

Figura . Manómetro del salpicadero digital

Por último se declara un método para actualizar la posición de la aguja asegurándonos de que la aguja no sobrepase el valor máximo.

#### **Indicador de distancia de seguridad**

Se sigue el mismo procedimiento para dibujar el indicador de la distancia de seguridad, se crea una clase que contendrá los distintos elementos de la representación. La clase llamada *DistanceCanvas* y que contiene los métodos para crear la representación hereda como la clase creada para el manómetro de la clase *Canvas*.

En este caso se usa una imagen para representar al vehículo y se dibujan arcos que a medida que la distancia disminuye alertando de la proximidad de objetos. Para conseguir este comportamiento debemos limitar la distancia a partir de la cual se empezarán a dibujar estos arcos, en este caso 50cm, es decir, cualquier medición por encima de 50cm no se indicará, y cualquier medida que este por debajo tendrá una representación. Esto se realiza con la función *min()*, que devuelve el valor más pequeño de los que se le dan como argumento. En este caso a la función se le da el valor de la distancia actual que queremos representar y 50, que es la distancia a partir de la cual se dibuja una representación. Para obtener el número de arcos a dibujar se utilizan las siguiente formulas:

Donde *dacotada*es la distancia obtenida a través de la función *min()*, *dlim* es la distancia limite a partir de cual se empiezan a dibujar arcos, en este caso 50cm. La variable *ratio* indica en un valor entre 0 y 1 la proporción que hay entre la distancia a mostrar y la máxima. Donde *arcos* indica el número de arcos que se dibujaran y se obtiene de la función *round()*, a la que se le pasa como parámetro el producto del *ratio* y el número máximo de arcos representado por *arcosmax*. De esta forma se obtiene el número de arcos que se deben dibujar en la interfaz.

Para dibujar los arcos se aumenta el tamaño del rectángulo que define le elipse por una cantidad fija, de este modo conseguimos un aumento de tamaño y dejamos un espacio visual que marca el intervalo de los arcos.

Icono

El contenido generado por IA puede ser incorrecto.

Figura . Indicador de distancia de seguridad del salpicadero digital

#### **Cuadro de información del vehículo**

Este elemento del salpicadero virtual contiene la información respectiva a la distancia recorrida, nivel del depósito de combustible, temperatura del motor y temperatura del interior del habitáculo.

Como en los apartados anteriores los elementos que forman parte del cuadro se han agrupado en una clase. Aunque en este caso también se ha creado una clase para especificar el comportamiento de los indicadores de combustible y temperatura del motor, ya que se usa un manómetro para representarlos.

Los otros dos elementos de este cuadro son simplemente indicadores numéricos con las magnitudes correspondientes, es decir, a nivel de código son dos cuadros de texto que se actualizan con los nuevos valores que se van recibiendo.



Figura . Indicador de distancia recorrida del Cuadro de información



Figura . Indicador de temperatura del habitáculo del Cuadro de información

Los manómetros están representados por un arco que se extiende 270º, este acro empieza a partir de los 135º. Una vez creado el arco se dibuja la escala, la cual divide el arco en arcos de 67.5º, es decir, se divide el arco en cuartos. Para el cálculo de las marcas de la escala se usan las fórmulas (4.8), (4.9) y (4.10).

Los valores que se pueden representar en el manómetro tendrán que estar comprendidos entre 0 y 1. Para el indicador del nivel de combustible 0 indica vacío y 1 lleno. Para el indicador de temperatura del motor 0 indica frio y 1 caliente.

Imagen que contiene Forma

El contenido generado por IA puede ser incorrecto.

Figura . Manómetro de nivel de combustible del Cuadro de información

Imagen que contiene Forma

El contenido generado por IA puede ser incorrecto.

Figura . Manómetro de temperatura del motor del Cuadro de información

#### **Comunicación alámbrica**

Para la comunicación alámbrica se hace uso de la librería *pyserial* la cual proporciona la funcionalidad necesaria para la transferencia de información a través de un puerto serie. La configuración necesaria se guarda en un archivo de variables de entorno, lo cual facilita ejecutar el programa en sistemas distintos.

La configuración debe de ser la misma que la utilizada en el microcontrolador, a excepción del puerto serie. Este vendrá determinado por la interfaz a la que conectamos el microcontrolador y del sistema operativo. En este caso durante el desarrollo en Windows el adaptador usado es detectado como COM6. Con esta configuración se declara un objeto de tipo *Serial* que representa la interfaz serie, el cual declaramos como global para poder ser usado en la función que se encarga de leer y actualizar los valores de la interfaz.

La función *process\_data()* se encarga de leer los valores que llegan al puerto serie y utiliza los métodos específicos para actualizar cada parte de la interfaz. Del puerto serie se leen 41 bytes, que es el mismo número que se manda desde el microcontrolador.

Antes de actualizar los datos, se tienen que procesar. Para esto decodifican los bytes como un string, luego se eliminan los caracteres que no necesitamos como el salto de línea o valores nulos. Por ultimo se usa la función *split()* que permite transformar un string en una lista, donde cada elemento se extrae de la cadena usando el separador definido como argumento de la función.

#### **Función principal**

Todos los componentes y funcionalidades del salpicadero virtual se integran en una función principal que se encarga de inicializar todos los componentes del salpicadero y de la comunicación alámbrica.

Se declaran las variables globales, el puerto serie, y los componentes de la interfaz. Primero se declara el objeto *window* que hace referencia a la ventana donde se dibujan los elementos de la interfaz. Se debe de definir el tamaño de la ventana y la posición en la que aparece en la pantalla. El tamaño de la pantalla se obtiene de variables de entorno que indican el ancho y el alto de la ventana. Con esta información podemos calcular la posición del punto que indica el inicio de la ventana con respecto de la pantalla. Teniendo en cuenta que el eje de la pantalla, o inicio de las coordenadas se encuentra en la esquina superior izquierda, y que el punto que se usa para indicar el comienzo de la ventana coincide con la esquina superior izquierda de esta, se usa la siguiente formula para que la ventana aparezca de una forma centrada en la pantalla.

Donde *c* es la coordenada del punto que queremos obtener, *s* es el la altura o el ancho dependiendo de la coordenada que queramos calcular, y *w* es la altura o ancho de la ventana que vamos a dibujar. Tras hacer esta calculo para el ancho y la altura obtenemos el punto que dibuja la ventana centrada en nuestra ventana.

Tras esto se definen los elementos de la interfaz en el siguiente orden: cuentarrevoluciones, velocímetro, indicador de distancia de seguridad y cuadro de información del vehículo.

Por último se crea un hilo para la función de lectura y actualización de los valores en la interfaz y se llama a la función *mainloop* del objeto *window*, bloqueando el hilo de ejecución principal y dibujando la interfaz en la pantalla.

Imagen que contiene medidor, reloj

El contenido generado por IA puede ser incorrecto.

Figura . Salpicadero digital completo

## Resultados

Para validar el funcionamiento de cada componente se desarrollaron las siguientes pruebas:

* Prueba cuentarrevoluciones: Para verificar el correcto funcionamiento del cuentarrevoluciones se conecta un motor de corriente continua a un pin de 5V del microcontrolador, el motor cuenta con la rueda mostrada en la *Figura 4.7*. Las especificaciones del motor indican un máximo de 11600rpm. Se pudo comprobar que la medición obtenida por el microcontrolador y la representación mostrada en el salpicadero se acercaba a ese límite máximo.
* Prueba velocímetro y distancia recorrida: Dado que el microcontrolador no esta montado en una plataforma móvil, se instaló temporalmente en un vehículo teledirigido. Junto con unos cables de longitud considerable se mantuvo la conexión alámbrica al ordenador, y se trató de circular en línea recta. La velocidad máxima del vehículo radio control es de 8km/h o 2m/s. Se pudo observar como el microcontrolador indico una velocidad de 1.8m/s y el velocímetro quedo cerca de la marca de 2m/s. Aprovechando que el microcontrolador estaba montado sobre el vehículo se circulo un rato hasta alcanzar la distancia de 2m en el indicador de distancia recorrida.
* Prueba distancia de seguridad: Se coloca una caja de cartón delante de cada sensor junto con un metro para observar si se cumple el comportamiento esperado. Se comprueba que a la mínima distancia hay 4 arcos, teniendo en cuenta que hay como máximo 4 arcos y la distancia máxima configurada es 50cm deberíamos ver que aproximadamente cada 12cm se elimina un arco hasta llegar a los 50cm donde deberían de desaparecer todos. A una distancia de 13cm desaparece el primer arco, a los 24cm desaparece el siguiente, a los 38 desaparece el tercer arco y a los 49cm desaparece el ultimo arco, verificando el correcto funcionamiento de los sensores y el salpicadero.
* Prueba nivel de llenado del deposito de combustible: Se comprueba que para el valor mínimo de resistencia del potenciómetro se muestra como vacío el tanque de combustible, para el valor máximo que el tanque se muestra lleno, para la mitad del valor máximo se muestra el tanque a la mitad y que la transición entre estos valores no es demasiado abrupta. Verificando así la precisión del valor medido y el dibujado en la interfaz.
* Prueba temperatura motor y habitáculo: con un mechero se calientan los sensores hasta que se alcanzan los 50ºC, para el indicador del habitáculo se comprueba que se dibuja 50ºC en la interfaz. Y para el indicador de la temperatura del motor se comprueba que la aguja indica la mitad del manómetro, ya que el manómetro está configurado para mostrar valores entre 0 y 100.

# ANÁLISIS DEL PROYECTO

## Contexto socioeconómico

## Presupuesto

# CONCLUSIONES