INFORME PRACTICA DE LABORATORIO 7

Santiago Nomesque – Jair Munevar

Universidad Sergio Arboleda

¿QUE USAMOS?

- Microcontrolador STM32F411CEU.
- ST LINK V2.
- Programa STMCUBE IDE.
- Motor DC.
- Sensores IR de herradura.
- Puente HTB6612FNG.
- Programa QT Creator.

PROTOCOLO PARA ENVIAR RPM Y TIEMPO

Para este laboratorio se implementó un protocolo de comunicación básico creado por nosotros, donde se utilizó un array de siete posiciones, donde la primera posicion indica el inicio del envió de datos con un valor 0x0F, luego en la segunda posicion se establece el tamaño del array según la información enviada, de siete bytes, para los datos que deseamos enviar se usan las posiciones tres, cuatro y cinco, de modo que en la tres y cuatro se envían las RPM y en la cinco el valor del TIEMPO. La posición seis se usa para el CHK, en el cual establecimos una operación XOR entre los datos de la posición tres, cuatro, cinco y el numero decimal 15. Para finalizar nuestra comunicación en la posición siete se envía el valor 0xFF que funciona como indicativo de que el array de información llego a su fin.

QT CREATOR

QT Creator es un entorno de desarrollo integrado (IDE) para desarrollar aplicaciones utilizando el framework Qt. Se usa para crear aplicaciones gráficas de escritorio y móviles, gracias a su capacidad para manejar interfaces de usuario y soporte para múltiples plataformas. Este software se utilizó para crear una interfaz gráfica en la que se visualizará el valor de las RPM de nuestro motor y se tendrá la posibilidad de controlar el sentido de este mediante unos botones.

¿QUE SE REALIZÓ?

En este laboratorio primeramente se enviaron datos por medio del puerto USB y estos se recibieron con el programa “Hercules”, estos datos se utilizaron para realizar una gráfica en Excel de las RPM vs TIEMPO, posteriormente se buscó obtener la gráfica con el menor ruido posible y una vez se obtuvo esto, se le aplico un filtro digital de tipo FIR para así tener una señal lo más limpia posible. Luego de haber aplicado el filtro, se procedió a realizar una gráfica donde se evidenciarán las RPM máximas de acuerdo con el voltaje suministrado, en nuestro caso empezaríamos en 1 voltio y terminaríamos en 7 voltios con un incremento de 1 cada vez que se obtuviera la respectiva curva de cada voltaje y evidenciarlo en una misma grafica. Posteriormente teniendo en cuenta las RPM maximas de acuerdo con el voltaje, se realizó una gráfica de RPM vs VOLTAJE y a esta grafica se le determino la ecuación de la recta que luego se le hallo la función inversa para así tener una ecuación que nos permita ingresar el valor de RPM deseado y saber a qué voltaje debería operar nuestro motor. Con esto también se implementaron dos funciones en el código de la STM, una que nos permitía controlar el ciclo útil del PWM para asi poder suministrar el voltaje que se requiere a nuestro motor y otra que de acuerdo con el valor de RPM, determinaba el valor de voltaje necesario, estas dos funciones se integraron de modo que el sistema desde QT envía el valor de las RPM deseadas y una función recibe este valor, calcula el voltaje y lo envía a la otra función que recibe dicho valor y con base en este, establece el valor del ciclo útil de nuestro PWM.

También se configuro una interfaz en QT, la cual nos permitía, establecer nuestro motor en “encendido” y “apagado”, digitar el valor que deseamos obtener de RPM de nuestro motor, graficar las RPM vs TIEMPO y saber el tiempo que se demoró el motor en llegar a las RPM máximas y el valor de estas.

¿QUÉ SUCEDIÓ?

Al momento de graficar los primeros datos obtenidos de nuestro motor, vimos que los datos estaban muy dispersos, es decir la gráfica presentaba mucho ruido tal como se observa en la imagen 1 y 2.

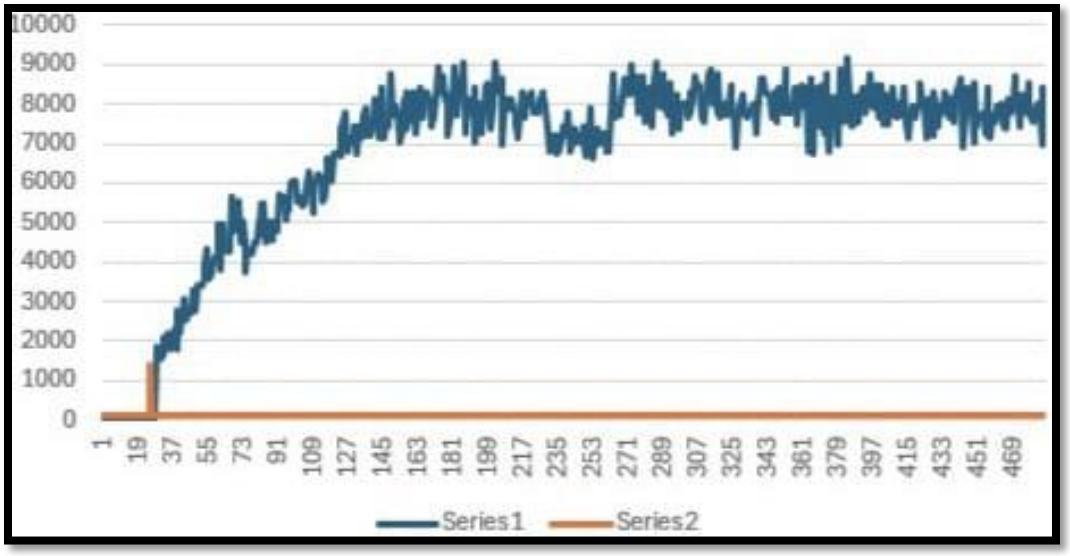

Imagen 1: Curva RMP sin aplicar filtros.

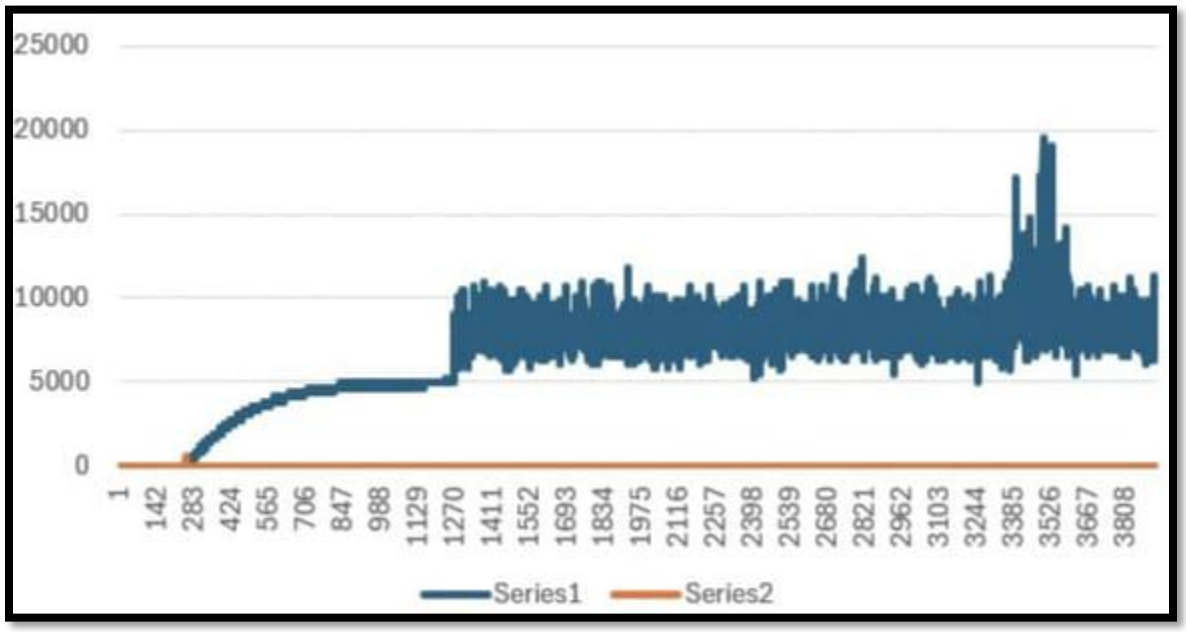

Imagen 2: Curva RMP sin aplicar filtros.

Entonces para solucionar esto, nos dimos cuenta de que la señal arrojada por el sensor no era la más optima, esta debía ser cuadrada y prácticamente era una señal de picos, lo que nos llevó a cambiar las resistencias que alimentaban nuestro sensor y de esta manera se logró optimizar el funcionamiento del sensor. Con esto ya solucionado volvimos a tomar los datos y realizar la gráfica y se obtuvo una gráfica mucho mejor, ya tenía un ruido menor y se veía más estable, tal como se observa en la imagen 3.

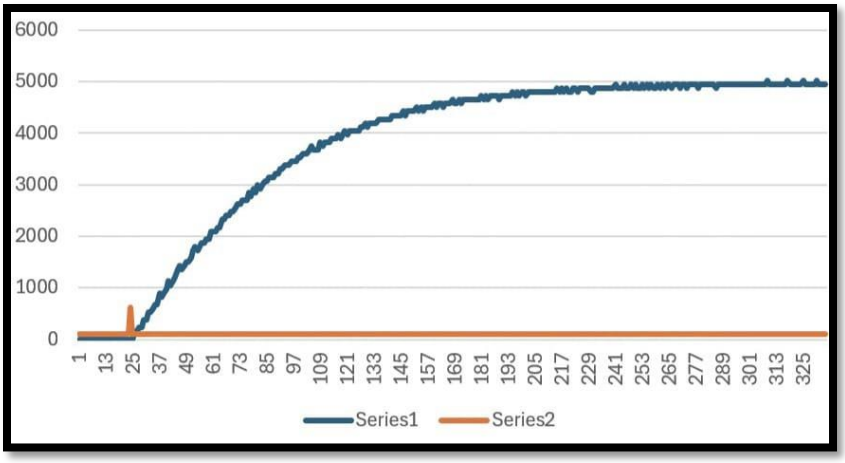

Imagen 3: Curva RPM con la optimización del sensor

Ahora para la suavización de dicho ruido se implementó el filtro FIR, primero este se aplicó con un total de 5 coeficientes y la ventana usada era Hann, esta nos arrojaba una gráfica bien durante las primeras RPM pero cuando se estabilizaban presentaban mucho ruido, así que decidimos aumentar los coeficientes de 5 en 5 hasta un total de 15, donde vimos que la gráfica ya empezaba a alejarse mucho de la original, entonces establecimos el valor de coeficientes en 15, sin embargo el ruido seguía en las RPM máximas, entonces ahora se procedió a probar con los diferentes tipos de ventana, llegando al mejor resultado haciendo uso de la ventana de Bartlett, de esta manera se obtuvo la gráfica que se observa en la imagen 4. Cabe aclarar que el valor de los coeficientes calculados se realizó con un programa sencillo en PYTHON, el cual es el siguiente:

“import numpy as np

from scipy.signal import firwin

numtaps = 15

cutoff = 0.1

coefficients = firwin(numtaps, cutoff, window='bartlett')

print(coefficients)”

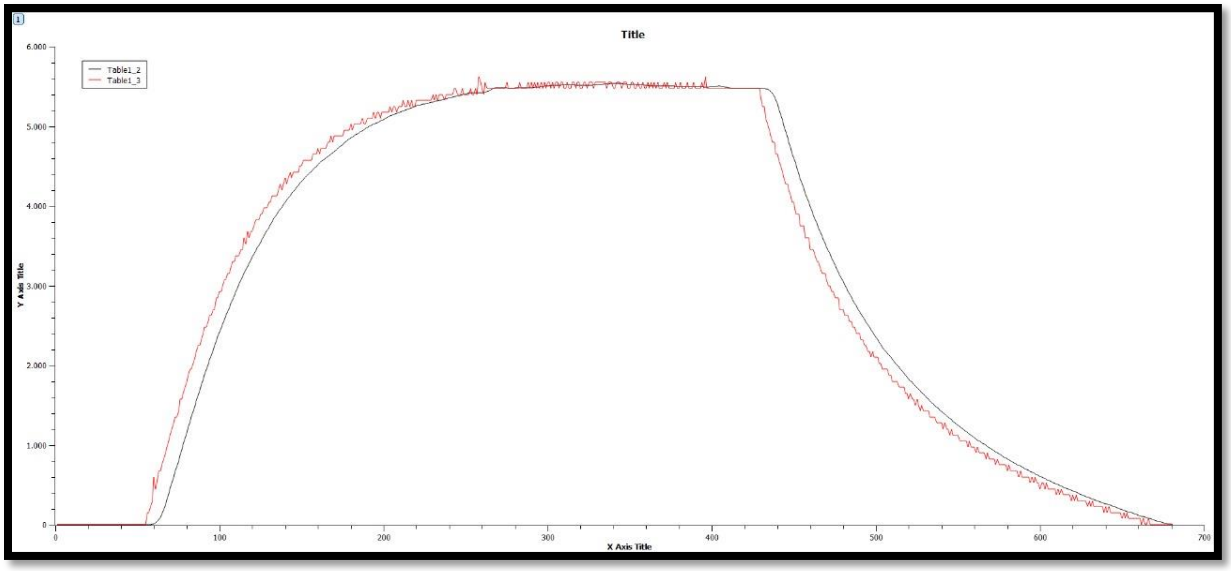

Imagen 4: Grafica RPM con el filtro FIR aplicado

Una vez obtenido los resultados esperados, procedimos con la gráfica de las RPM de acuerdo con el voltaje suministrado, para ello se implementó la función que controlaba el ciclo útil del PWM, esta función básicamente es una regla de 3, teniendo en cuenta que nuestro 100% de ciclo útil es 7,5 voltios y el 0% es 0 voltios, de esta manera se calculaba el valor del ciclo útil y adicionalmente se agregó la funcionalidad de que si llegábamos a poner el valor en negativo del voltaje, eso indicaba que el motor debía girar en sentido contrario.

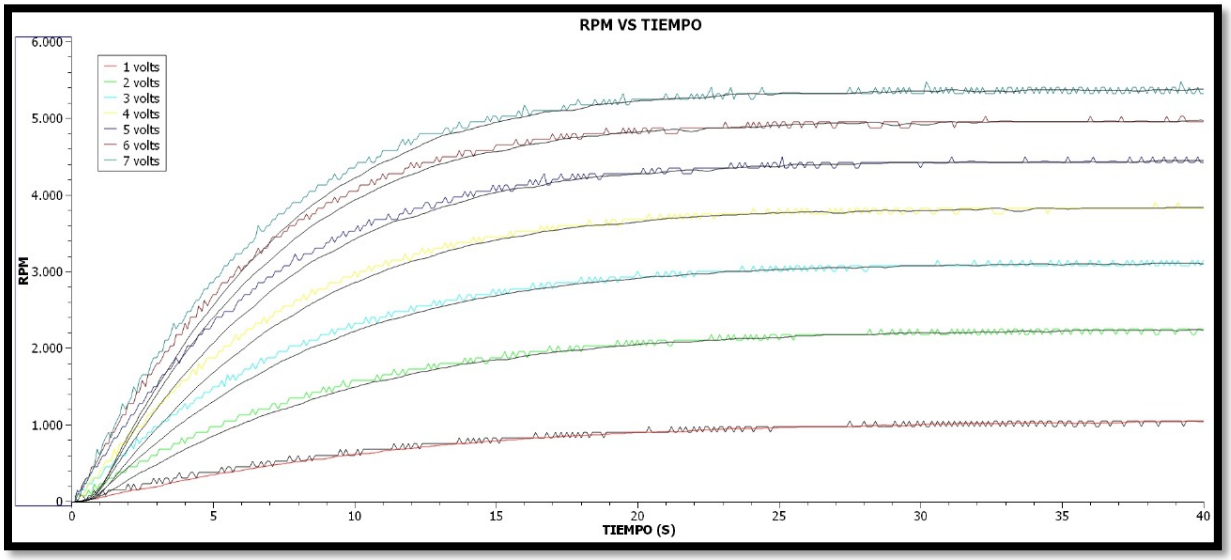

Imagen 5: Grafica RPM vs Tiempo de 1v a 7v.

Usando la gráfica anterior, determinamos las revoluciones máximas de acuerdo con el voltaje y estos datos los utilizamos para realizar la gráfica que se observa en la imagen 6.

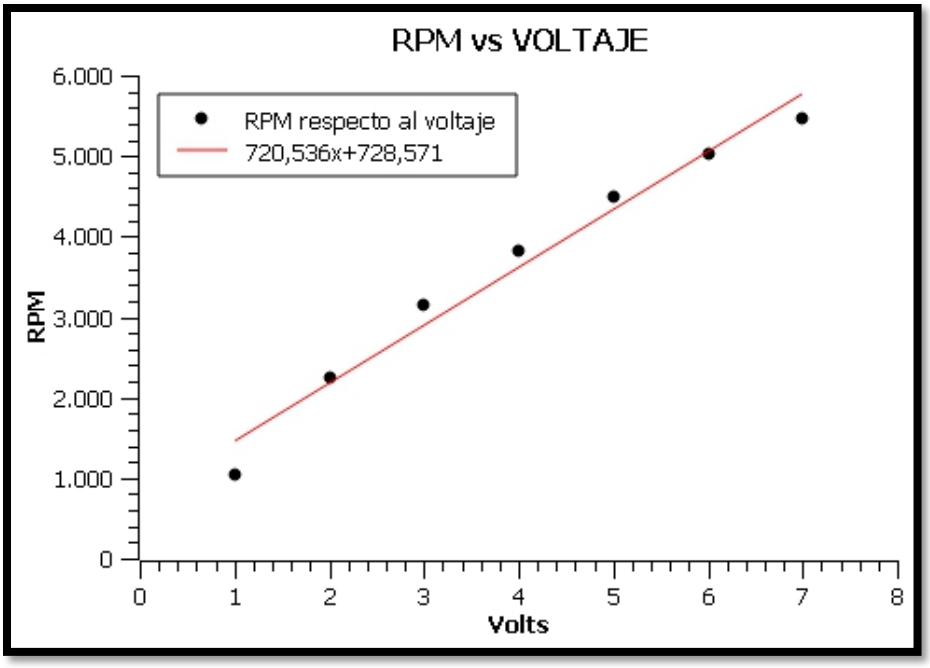

Imagen 6: Grafica RPM vs Voltaje.

Y, por último, teniendo en cuenta la gráfica anterior y su ecuación de recta, se le hallo la función inversa a dicha ecuación y esta se utilizó para crear una función la cual nos permitió ingresar el valor de las revoluciones que deseáramos y se calculaba el voltaje al cual debía alimentarse el motor, logrando así combinar la función creada anteriormente que tomaba el valor del voltaje con la función de las RPM.

Una vez se obtuvo todo lo anterior se procedió con la interfaz gráfica de QT, donde se recibían los datos el tiempo y las revoluciones, estos datos se mostraban en tiempo real en la interfaz, también se agregó la funcionalidad de que se calculara en QT el valor de las RPM máximas y el tiempo que se demoró el sistema en alcanzar dicho valor, para ello inicialmente se estaban utilizando variables que no se habían creado como globales y esto no nos permitió el cálculo adecuado y presentaba errores, eso se solucionó creando las variables como globales en el archivo widget.h y de esta manera ya se pudieron realizar los cálculos de manera adecuada y así mismo mostrar estos datos en la interfaz. Posteriormente se utilizó un ejemplo compartido por el docente para implementar la gráfica de las RPM en nuestra interfaz, acá fue donde presentamos más problemas, debido a que inicialmente no entendíamos del todo bien cómo funcionaba por sí solo el ejemplo enviado, una vez logramos entender su funcionamiento lo implementamos en nuestra interfaz, pero algunas cosas no funcionaban ya que nos arrojaba errores, esto se debió esencialmente a que no habíamos agregado las librerías correctamente y también no se había agregado en el widget.pro la herramienta para la impresión de la gráfica en la interfaz. Esto se logró arreglar agregando las librerías adecuadamente y en el archivo widget.pro, a la línea de

código “*greaterThan(QT_MAJOR_VERSION, 4): QT += widgets*” se complementó con “*printsupport*”, quedando de la siguiente manera “*greaterThan(QT_MAJOR_VERSION, 4): QT += widgets printsupport*” y asi ya se logró que el programa corriera la interfaz gráfica. Hasta este punto pensamos que ya todo estaba de manera correcta, pero al correr nuestra interfaz y realizar la conexión con nuestro puerto “COM” la interfaz colapsaba y se cerraba, con ayuda del docente nos dimos cuenta de que no estábamos realizando la correcta configuración de la función “setupPlot” en nuestro programa principal y era por ello por lo que no funcionaba correctamente, se agregó esa función en la función de Widget principal y se logró solucionar el problema y ahora sí, el programa funcionaba de manera óptima. La interfaz gráfica obtenida se puede observar en la imagen 7.

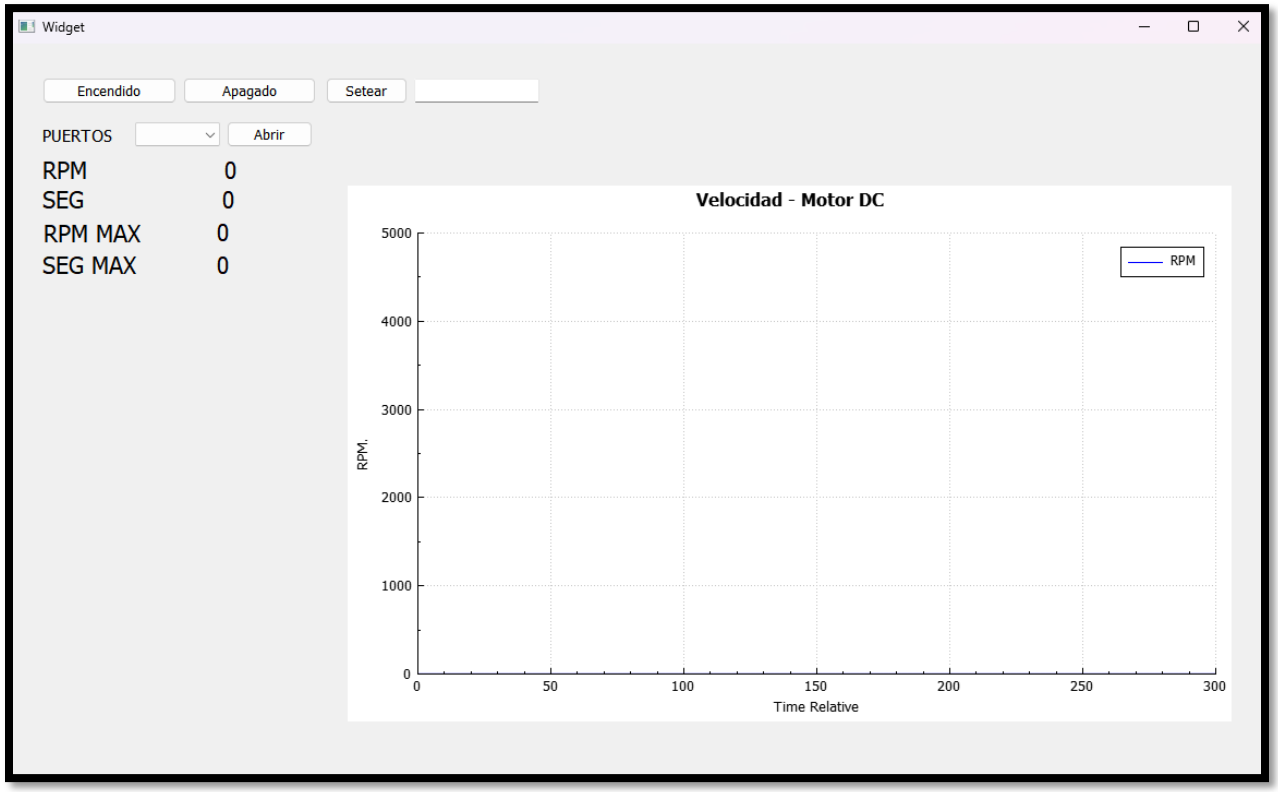

Imagen 7: Imagen de la interfaz gráfica obtenida en QT

DIAGRAMAS DE FLUJO

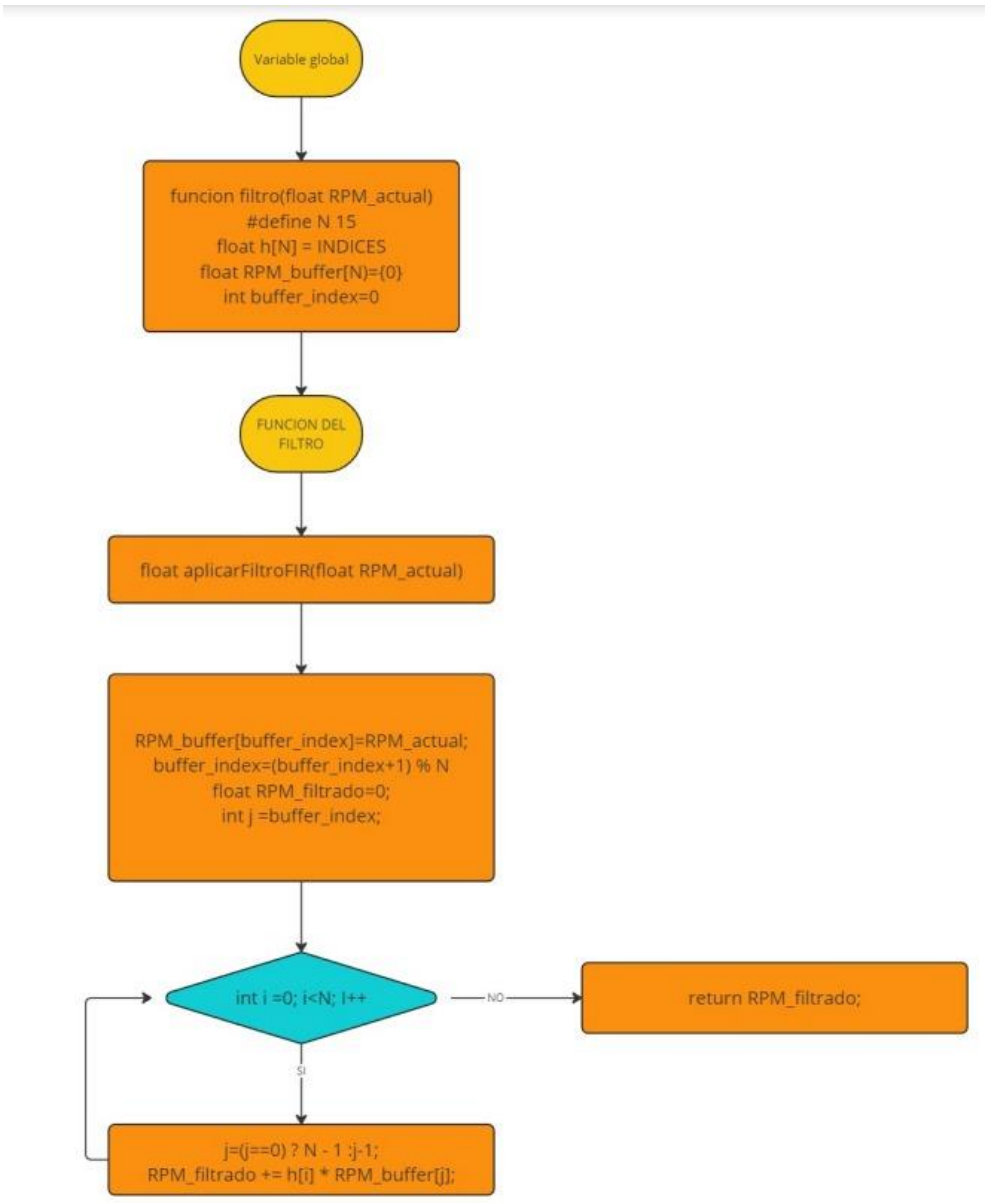

Imagen 8: Diagrama de flujo del código de la STM32 para filtrar la señal

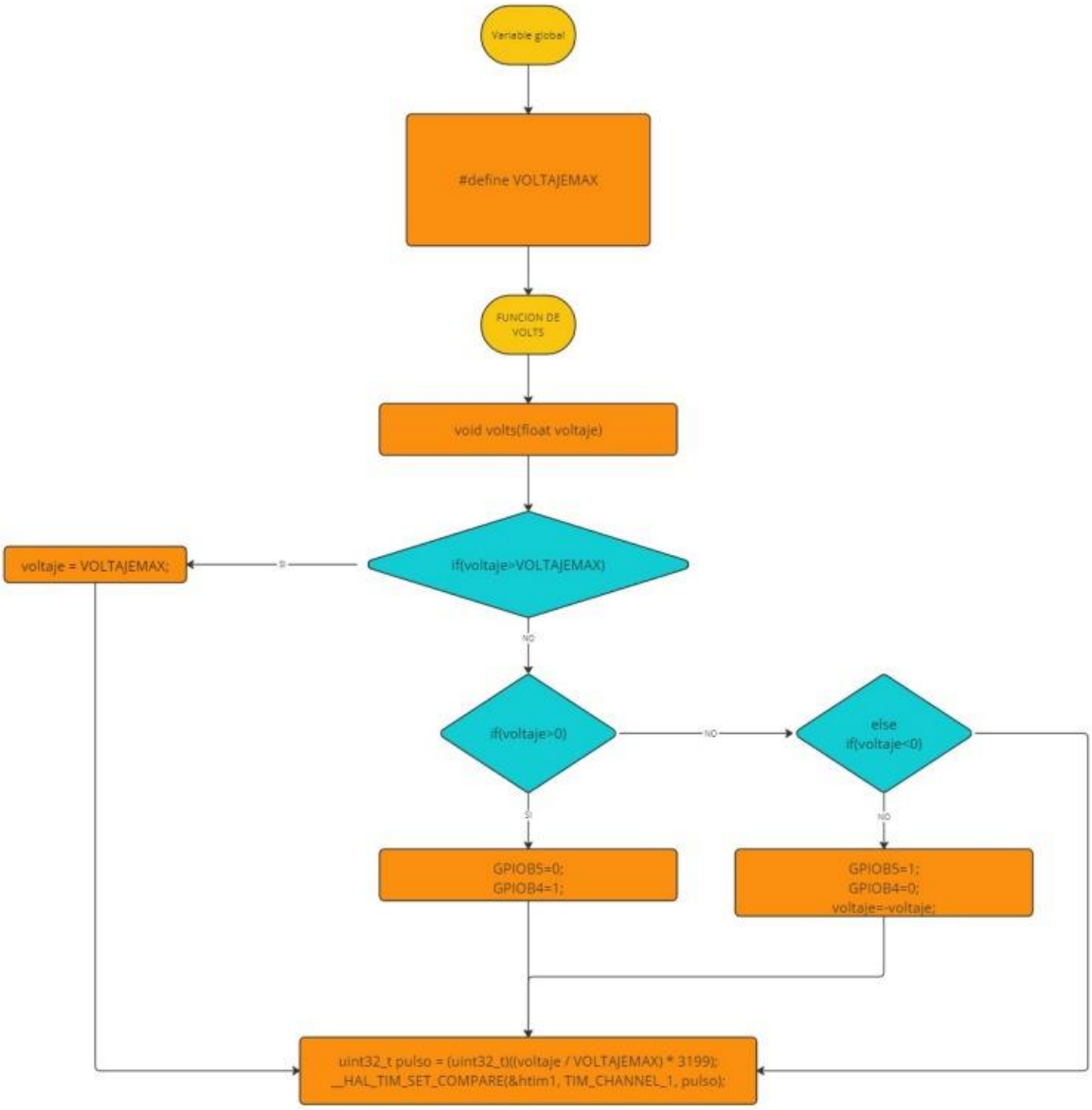

Imagen 9: Diagrama de flujo del código de la STM32 para la función del ciclo útil del PWM

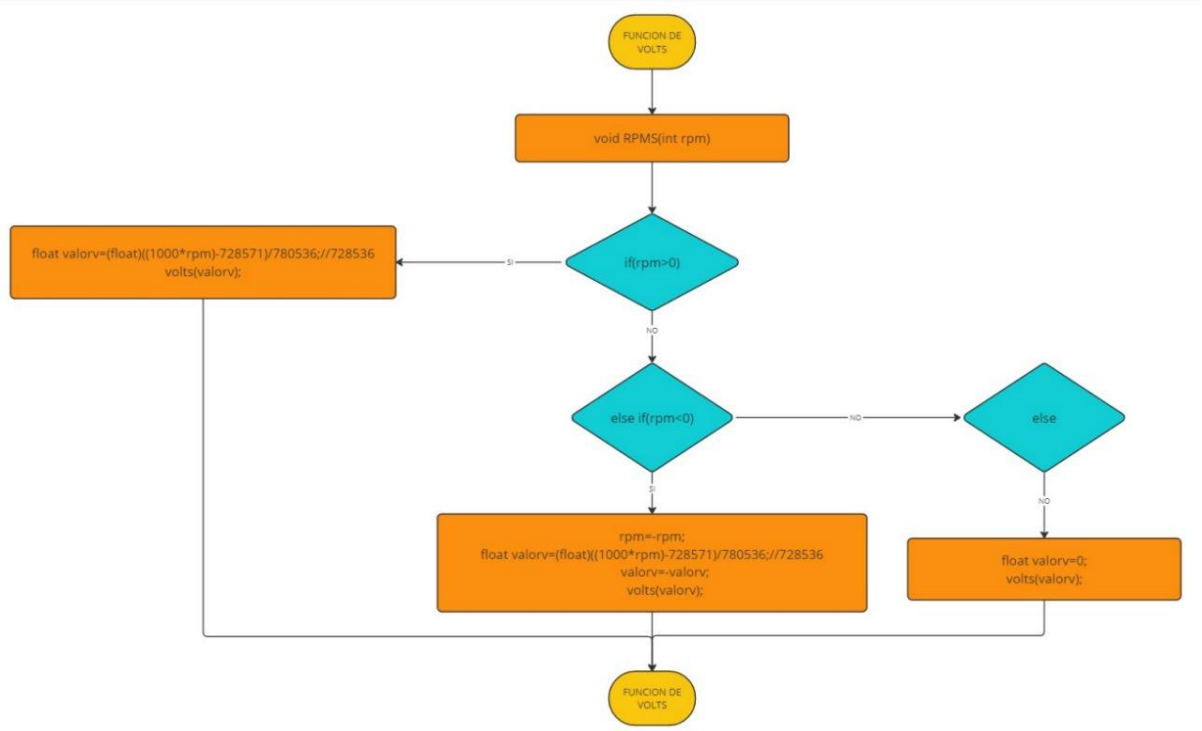

Imagen 10: Diagrama de flujo del código de la STM32 de la función para el cálculo del voltaje de acuerdo con las RPM

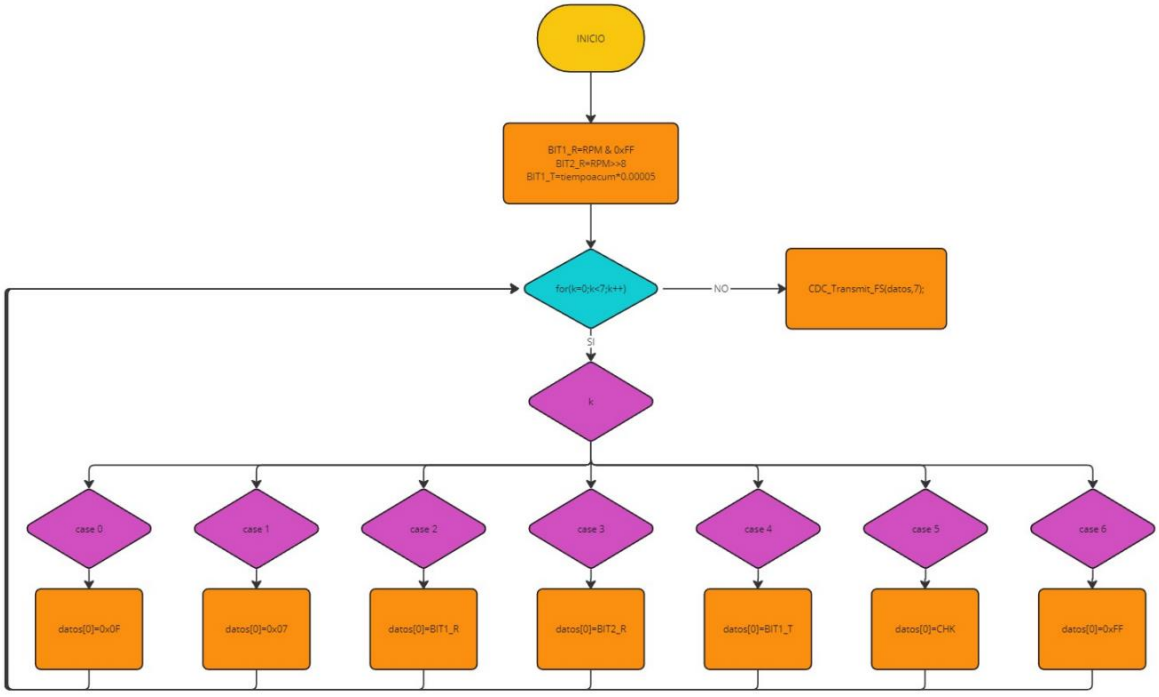

Imagen 11: Diagrama de flujo del código de la STM32 para él envió de datos por medio del USB

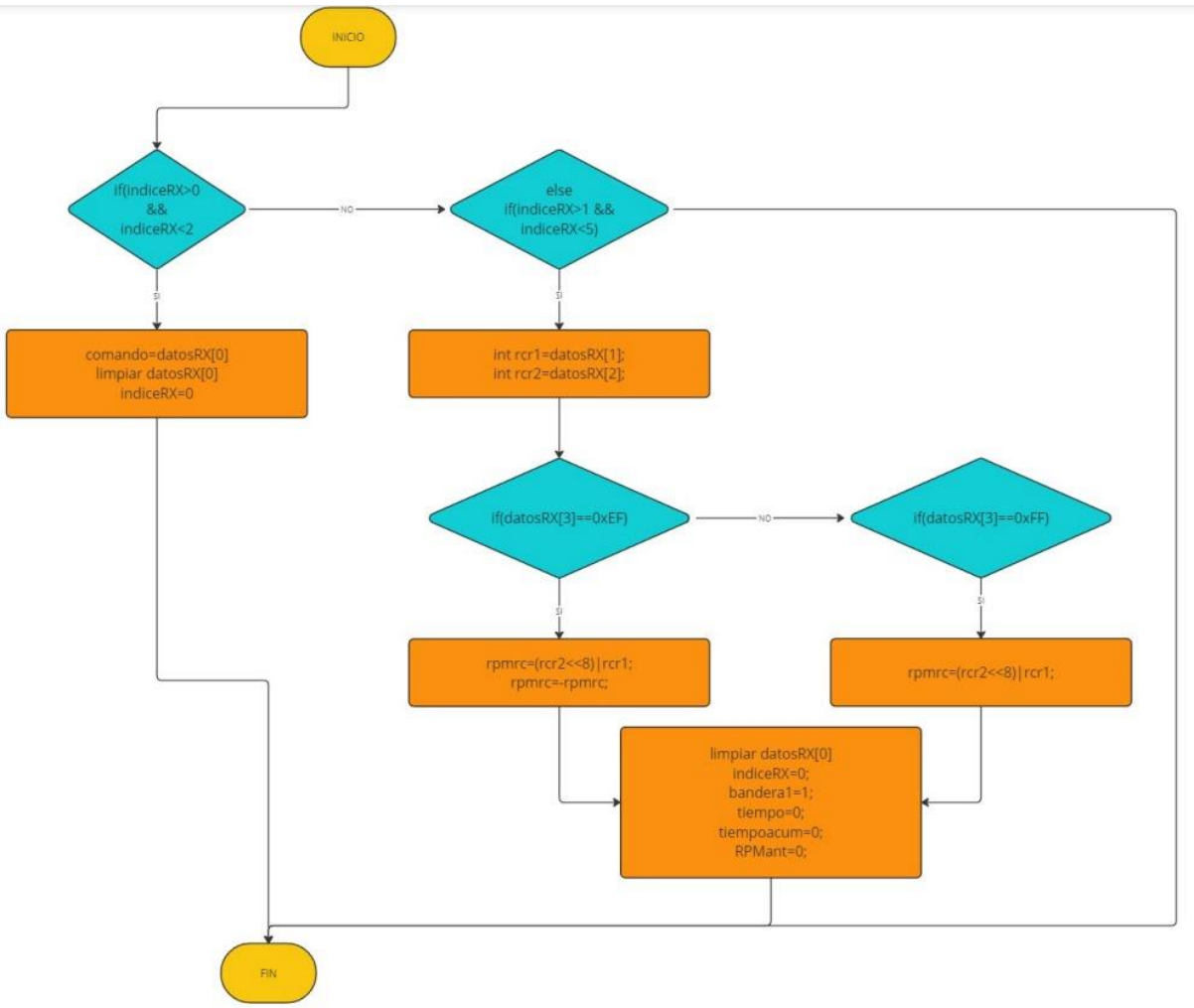

Imagen 12: Diagrama de flujo del código de la STM32 para la recepción de datos

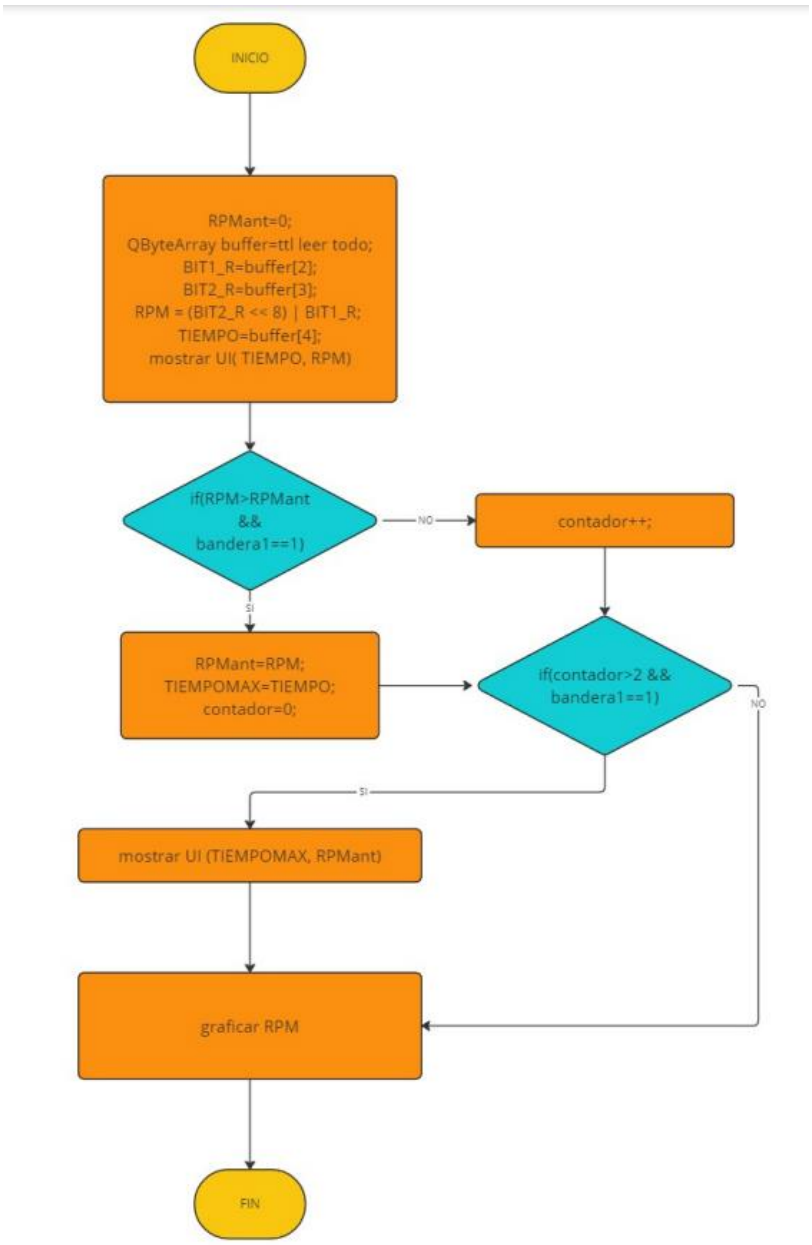

Imagen 14: Diagrama de flujo del código de la interfaz de QT para la lectura y visualización de datos

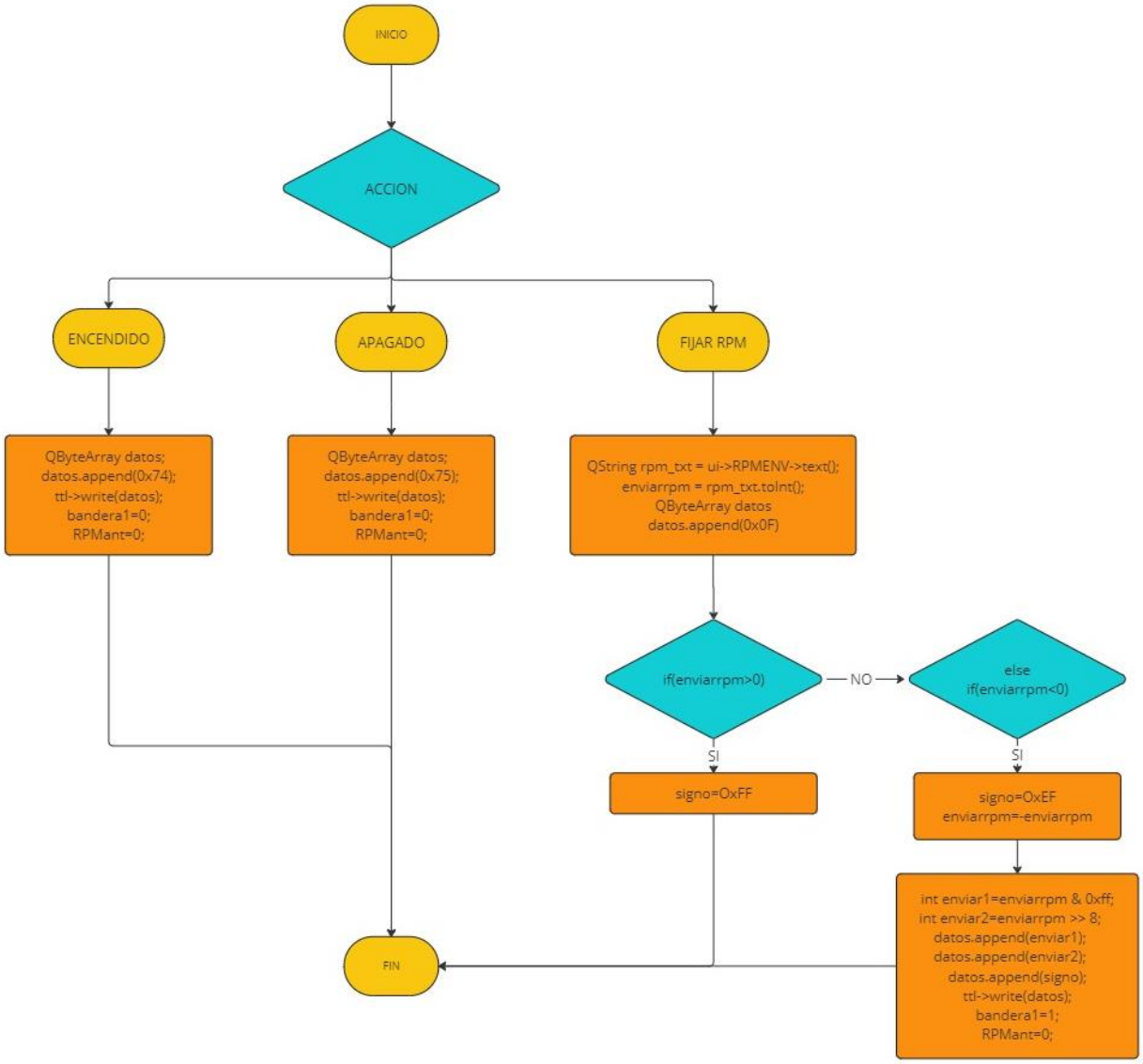

Imagen 15: Diagrama de flujo del código de la interfaz de QT para el envió de datos

ESQUEMÁTICO

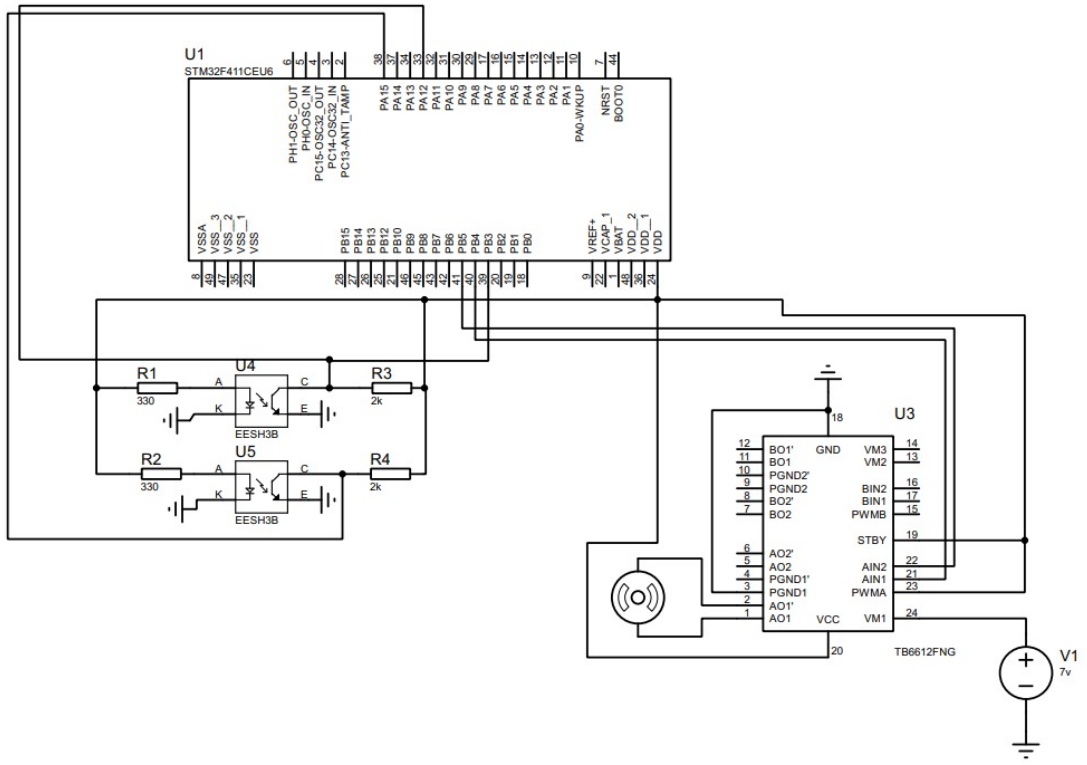

Imagen 16: Esquemático usado en la elaboración del laboratorio.

CONCLUSIONES

- Si se va a utilizar un codigo de ejemplo para la implementación de alguna funcionalidad propia, se debe primero entender correctamente el código ejemplo, ya que, a la hora de cualquier error o inconveniente, se puede saber qué es lo que está pasando, de esta manera no se “copia y pega” esperando que funcione de manera óptima debido a que el ejemplo funcionaba perfectamente.
- La primera fase del laboratorio mostró que la señal del sensor tenía un alto nivel de ruido. Al ajustar las resistencias que alimentaban el sensor, se logró estabilizar la señal, permitiendo obtener datos más precisos. Esto resalta la importancia de la calibración de sensores en sistemas de control, donde una señal limpia es fundamental para mediciones confiables.
- El uso del filtro digital FIR fue crucial para limpiar la señal de RPM. Inicialmente, los resultados fueron insatisfactorios, pero al aumentar los coeficientes y experimentar con diferentes ventanas, se encontró que la ventana de Bartlett proporcionaba los

mejores resultados. Este proceso destacó la eficacia del filtrado en el mejoramiento de señales en sistemas de control.

- Al graficar RPM versus voltaje, se estableció una relación clave que permite calcular el voltaje necesario para alcanzar RPM deseadas. Esto es fundamental para el control en tiempo real mediante PWM. La capacidad de ajustar el voltaje según las RPM deseadas muestra la aplicabilidad práctica de los sistemas de control.
- La creación de una interfaz gráfica en QT facilitó el control del motor y la visualización de datos en tiempo real. Aunque se presentaron desafíos técnicos, su resolución subrayó la importancia de la depuración en el desarrollo de software. La interfaz no solo mejora la usabilidad, sino que también proporciona un feedback visual esencial para el monitoreo efectivo del sistema.