



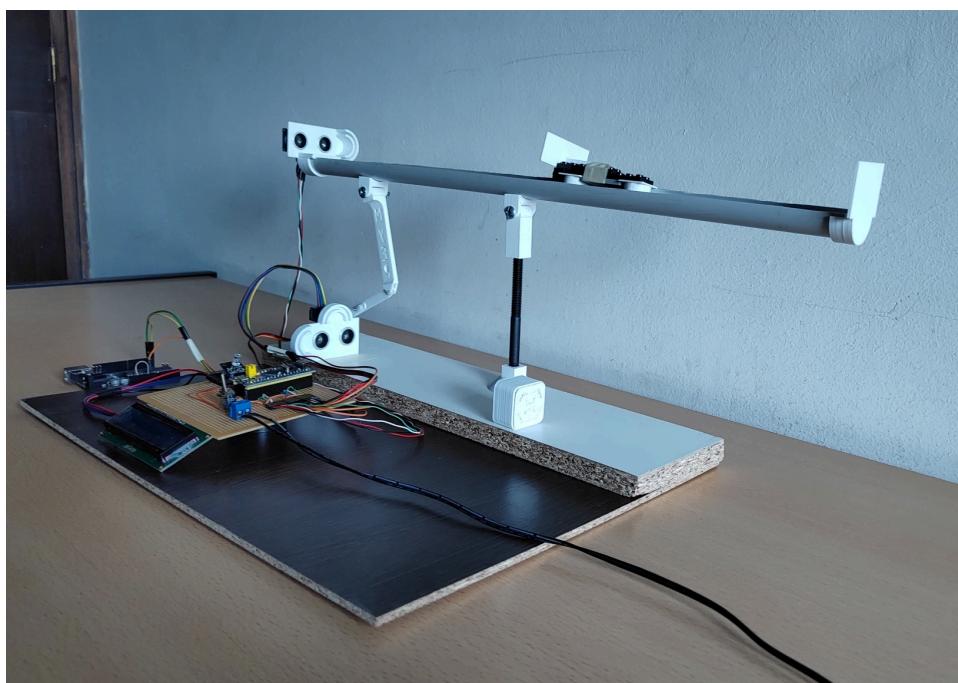
**UNCUYO**  
UNIVERSIDAD  
NACIONAL DE CUYO



**FACULTAD**  
**DE INGENIERÍA**

# PROYECTO FINAL INTEGRADOR

*Sistema de control de posición de carro con microcontrolador “Blue Pill” y control PID.*



**Cátedra**

**MICROCONTROLADORES Y ELECTRÓNICA DE POTENCIA**

**Alumnos**

Dalessandro Francisco  
Panonto Valentín

Legajo N°: 13318  
Legajo N°: 13583

**Profesores a cargo**

Profesor titular:  
Profesor JTP:

Ing. Iriarte Eduardo  
Ing. Cruz Martín

**Año 2024**



# Índice

Índice.....	1
Introducción.....	2
Esquema tecnológico.....	3
Detalle de módulos.....	4
Microcontrolador STM32F103C8 Blue Pill.....	4
Sensor Ultrasónico HC-SR04.....	5
Servomotor SG90.....	7
Panel LCD 16x2 con Módulo I2C (controlador HD44780).....	8
Módulo emisor LED RGB KY-016.....	9
Arduino UNO.....	10
Funcionamiento general.....	11
Configuración del microcontrolador.....	13
Programación.....	16
Funciones.....	25
Etapas de montaje y ensayos realizados.....	39
Resultados y especificaciones finales.....	47
Ensayo de ingeniería de producto.....	48
Conclusiones.....	50
Referencias.....	52
Anexo.....	53



## Introducción

En el presente informe se detalla el proyecto final integrador realizado para la cátedra de “Microcontroladores y Electrónica de Potencia”. El desarrollo se centra en la implementación de un sistema embebido para el control de posición de un carro basado en un mecanismo de barra inclinable. El objetivo es mantener un objeto en una posición específica sobre la barra mediante el ajuste de su inclinación.

La motivación para llevar a cabo este proyecto surgió de la repetida visualización de proyectos similares, los cuales despertaron un gran interés para intentar desarrollar uno desde cero y concluir logrando un funcionamiento óptimo. Enfrentar el desafío de crear un sistema de control dinámico en un entorno controlado representa una oportunidad única para aplicar y experimentar tanto con técnicas de control en sistemas físicos creados, como con el desarrollo del software adecuado y correcta elección del hardware necesario para su implementación.

El sistema desarrollado tiene aplicaciones en diversas áreas donde el control preciso de la posición es crucial. En robótica, por ejemplo, este tipo de control es esencial para la estabilidad y precisión de movimientos que requieren alta exactitud. Además, en la automatización industrial, el sistema puede ser adaptado para controlar mecanismos que deben mantener una posición específica, mejorando la eficiencia y precisión de procesos automatizados.

El sistema también puede ser utilizado como una plataforma de prueba para el desarrollo y validación de algoritmos de control, permitiendo a los ingenieros experimentar y optimizar técnicas antes de aplicarlas en entornos más complejos y costosos. Su capacidad de ajustar la inclinación para mantener un objeto en posición lo hace ideal para aplicaciones que requieren estabilidad dinámica, como en la calibración de equipos sensibles a la posición.

El desarrollo del sistema de control de posición de un carro se basa en varios fundamentos y principios clave de la ingeniería y la física:

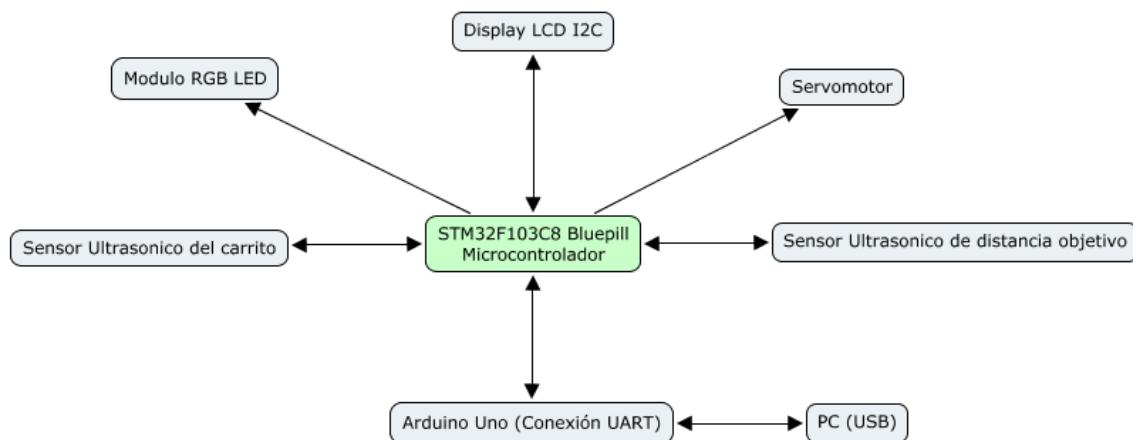
- Control PID (Proporcional-Integral-Derivativo): Técnica de control ampliamente utilizada en sistemas de control automático. Este sistema se utiliza para ajustar la inclinación de la barra en función de la posición del objetivo, utilizando el control proporcional para corregir el error presente, el integral para corregir errores acumulados en el tiempo y el derivativo para anticipar errores futuros basándose en la tasa de cambio de la posición.
- Mecánica: La dinámica del carro sobre la barra inclinable se rige por las leyes de Newton. El movimiento del carro es influenciado por la gravedad y la fuerza aplicada a través del servomotor, que ajusta la inclinación de la barra para mantener la posición deseada.
- Sensores de posición: Se emplean sensores ultrasónicos para medir la posición del carro en la barra. La precisión y la rapidez en la adquisición de datos son cruciales para el correcto funcionamiento del sistema de control, permitiendo realizar ajustes en tiempo real.



- Sistemas Embebidos: El microcontrolador STM32F103C8 actúa como el cerebro del sistema, ejecutando el algoritmo de control PID y gestionando las señales de los sensores y actuadores. La programación de este microcontrolador implica conocimientos de electrónica digital y programación en lenguaje C/C++.
- Retroalimentación: Principio fundamental en los sistemas de control, donde las salidas del sistema (posición del carro) son constantemente monitoreadas y usadas para ajustar las entradas (inclinación de la barra) con el fin de minimizar el error y mantener la estabilidad del sistema.

## Esquema tecnológico

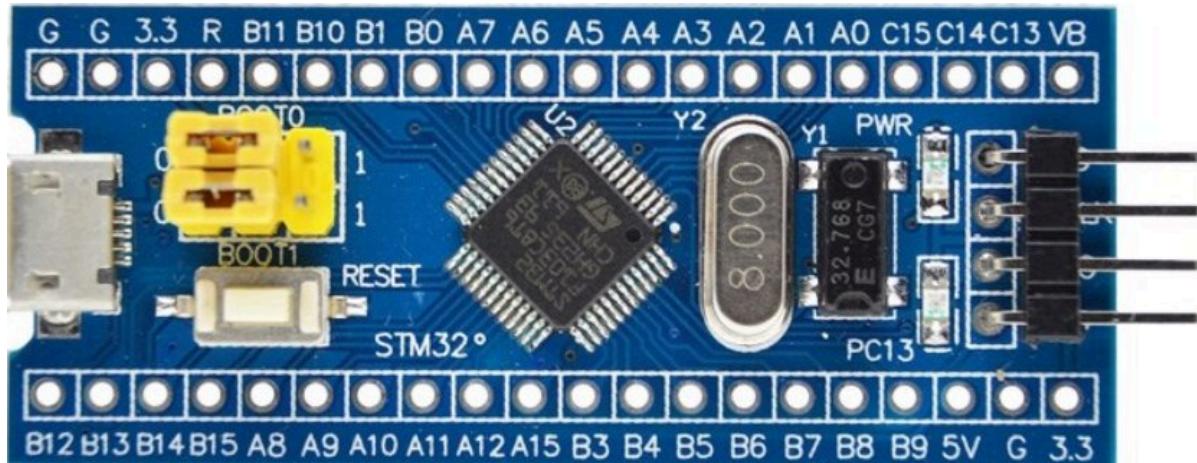
A continuación se muestra un diagrama de bloques que representa el esquema tecnológico del sistema. En el mismo, se observa a grandes rasgos, los principales módulos que componen el sistema. A su vez, se representa la dirección de la comunicación con los sentidos de las flechas.





## Detalle de módulos

### Microcontrolador STM32F103C8 Blue Pill



La placa Blue Pill es una placa de desarrollo basada en el microcontrolador STM32F103C8T6 de la compañía de desarrollo de microcontroladores STMicroelectronics. Es popular por su bajo costo, alto rendimiento y una variedad de características que la hacen adecuada para una amplia gama de proyectos de sistemas embebidos. Algunas de sus características son:

#### 1. Características generales

- **Arquitectura:** ARM Cortex-M3.
- **Frecuencia:** Hasta 72 MHz.
- **Memoria Flash:** 64 KB.
- **RAM:** 20 KB.
- **Número de pines Entrada/Salida (GPIO):** 37
- **Configuración:** Los pines son configurables para funciones especiales como UART, I2C, SPI, PWM y ADC.
- **Tolerancia de Voltaje:** La mayoría de los pines GPIO son tolerantes a 5V, pero no todos los que posee la placa, algunos de estos solo soportan una tensión máxima de 3.3V. Es por esto que se debe tener cuidado al conectar dispositivos de mayor voltaje o al utilizar los pines de la placa como entradas para dispositivos externos.
- **Voltaje de operación:** 2.0 a 3.6V

#### 2. Interfaces de Comunicación

- **UART:** La placa tiene 3 puertos UART (USART1, USART2 y USART3), lo que facilita la comunicación serial para depuración y comunicación con otros dispositivos.
- **SPI:** Soporte para 2 interfaces SPI, útiles para la comunicación con sensores y módulos de alta velocidad.
- **I2C:** 2 Interfaces I2C disponibles para conectar dispositivos como pantallas LCD y sensores.



- **CAN:** Presenta 1 puerto CAN.

### 3. Periféricos Adicionales

- **ADC:** 2 Conversores Analógico-Digital con 10 canales, útiles para leer sensores analógicos.
- **PWM:** Generadores de señal PWM que pueden ser usados para controlar servomotores y LEDs RGB, entre otras cosas.
- **Timers:** 4 temporizadores configurables de 16 bits para tareas de temporización precisa y generación de señales PWM.

### 4. Conectividad

- **USB:** La placa puede ser programada y depurada vía USB utilizando el bootloader integrado o mediante un programador externo (como el que fue utilizado ST-Link), conectado en los pines etiquetados como SWD (Serial Wire Debug).
- **Debug:** Compatible con herramientas de depuración como ST-Link y JTAG. Esta es una gran ventaja pues permite monitorear variables en tiempo real para encontrar posibles funcionamientos erróneos.

La especificación de cada uno de los periféricos que se utilizaron de este microcontrolador, se desarrollará a continuación, según lo requiera cada módulo de hardware en particular que utilice alguno.

## Sensor Ultrasónico HC-SR04

En este proyecto se utilizan dos sensores ultrasónicos HC-SR04 para la medición de distancias. Este sensor opera emitiendo un pulso de onda de sonido ultrasónico a una frecuencia superior a 20 kHz. Un transductor integrado en el propio sensor genera un pulso que se propaga a través del aire, el cual cuando encuentra un obstáculo o un objeto en su camino, se refleja y regresa al sensor, donde es captada por el receptor.



El tiempo transcurrido entre la emisión del pulso y la recepción de su eco, es medido por el microcontrolador. A partir de estos dos datos temporales y utilizando la velocidad del sonido en el aire (aproximadamente de 343 metros por segundo a 20°C), el



microcontrolador calcula la distancia a la cual se encuentra el objeto en el que se reflejó la onda, con la siguiente fórmula:

$$Distancia = \frac{Tiempo\ de\ vuelo \times Velocidad\ del\ sonido}{2} \quad (1)$$

Debido a que el tiempo de vuelo del pulso ultrasónico contempla tanto el viaje de ida como el de vuelta, se divide el número resultante entre dos.

Los sensores son alimentados con una tensión de 5V, provenientes de una fuente de alimentación externa. El primer sensor se encarga de medir la distancia real en vivo del carro, con sus pines Trigger (pin de salida) y Echo (pin de entrada) conectados en los pines PA8 y PB14 respectivamente del microcontrolador. Luego se utiliza un segundo sensor para medir la posición objetivo que se desea, marcada por un pequeño bloque ubicado sobre la base de la maqueta. Este segundo sensor hace uso del pin PB3 para su señal de Echo, y del pin PB4 del microcontrolador para su señal de Trigger.

De la hoja de datos del se extraen a continuación algunas otras especificaciones técnicas relevantes:

- **Fuente de alimentación:** +5V DC
- **Corriente en reposo:** <2mA
- **Corriente de trabajo:** 15mA
- **Ángulo efectivo:** <15°
- **Distancia de rango:** 2-400 cm
- **Resolución:** 0.3 cm
- **Ángulo de medición:** 30°
- **Ancho de pulso de entrada del disparador:** 10uS
- **Dimensiones:** 45mm x 20mm x 15mm
- **Peso:** aproximadamente 10 g

El principio de funcionamiento del sensor ultrasónico es sencillo. Según la hoja de datos, el pin Trigger del sensor debe ponerse en un valor lógico HIGH durante 10 microsegundos (10  $\mu$ s). Inmediatamente después, el pin Trigger vuelve a un valor lógico LOW. Automáticamente, el pin Echo se pondrá en HIGH y el sensor enviará una onda ultrasónica. En este momento, se mide el tiempo hasta que la onda regrese. Cuando esto suceda, el pin Echo volverá a un valor lógico LOW.

El temporizador utilizado para medir todos los tiempos es el Timer 1, configurado con un prescaler de 72. De esta manera, la frecuencia de conteo pasa de 72 MHz a 1 MHz, lo que significa que el contador del temporizador incrementa una unidad cada 1  $\mu$ s.

De esta forma, y aplicando la ecuación (1), el microcontrolador puede calcular la distancia de un objeto hasta el sensor.

Cabe aclarar que el pin Echo genera una tensión de 5V, que hay que tener en cuenta a la hora de elegir su pin en el microcontrolador. Si no se quiere colocar en un pin que tolere 5V, la hoja de datos aconseja reducir ese valor con una resistencia en serie o bien un divisor de tensión.



## Servomotor SG90

Como actuador se utiliza un servomotor SG90, el cual, a partir del control de su posición angular, permite obtener una inclinación específica de la barra móvil a la que está mecánicamente conectado en el sistema.

Este servomotor está compuesto por un pequeño motor de corriente continua (DC) acoplado a un sistema de engranajes de reducción, que disminuye la velocidad del motor y aumenta el par, facilitando un control preciso de la posición del eje. Además, cuenta con un mecanismo de retroalimentación basado en un potenciómetro que mide la posición actual del eje. Cuando el servomotor recibe una señal de modulación por ancho de pulso (PWM) que indica la posición deseada, el potenciómetro compara esta posición con la actual. Si hay una discrepancia, el controlador interno ajusta la dirección del motor para corregir la posición y, una vez alcanzada, mantenerla.



El ciclo de trabajo de la señal PWM, que es el porcentaje de tiempo durante el cual la señal está en estado alto dentro de un periodo, controla la posición del eje del servomotor. Por ejemplo, un ciclo de trabajo de 1 ms generalmente corresponde a una posición de -90 grados, un ciclo de 1.5 ms a la posición central (0 grados), y un ciclo de 2 ms a +90 grados. La frecuencia típica de la señal PWM utilizada es de 50 Hz (o bien, tiene un periodo típico de 20 ms). Esto da como resultado un PWM con un ciclo de trabajo mínimo de 5% y máximo de 10%.

La hoja de datos detalla las siguientes características técnicas:

- **Peso:** 9 g
- **Dimensiones:** aproximadamente 22.2 x 11.8 x 31 mm
- **Torque de estancamiento:** 1.8 kgf·cm
- **Velocidad de operación:** 0.1 s/60 grados
- **Voltaje de operación:** 4.8 V (~5V)
- **Ancho de banda muerto:** 10 µs
- **Rango de temperatura:** 0 °C – 55 °C



- **Conectividad:** Cuenta con tres cables: VCC (rojo), GND (marrón) y señal (naranja).

El cable de señal del servo se conecta al pin A15, correspondiente al Canal 1 del Timer 2, seteado en el modo de generación de PWM. Este Timer, al igual que el Timer 1, tiene un prescaler en 72, y está configurado para contar hasta 19999 (cuenta que tarda exactamente 20 ms, pues comienza desde 0). La generación de PWM está relacionada con el hecho de que el microcontrolador pone en HIGH el pin cuando el contador del Timer 2 está por debajo de cierto valor almacenado en el registro de comparación o CCR, y cuando supera este valor umbral, se cambia a LOW. Entonces, por ejemplo, si ese valor es de 1000, y el Timer cuenta hasta 19999, significa que el pin estará en HIGH un 5% del tiempo (es decir, se generará una señal de PWM con ciclo de trabajo de 5%), pues  $1000/(19999+1) = 0.05$ .

En la ejecución del programa, el valor del CCR es directamente proporcional al ángulo al que el usuario o el controlador PID le pidan que rote.

## Panel LCD 16x2 con Módulo I2C (controlador HD44780)

En este proyecto se emplea un panel LCD 16x2 junto con un módulo de interfaz I2C para la visualización de información.



### Características:

- Este panel se denomina así porque tiene 16 columnas y 2 filas, permitiendo la visualización simultánea de hasta 32 caracteres.
- Utiliza el controlador HD44780, estándar en la mayoría de los módulos LCD, con lo que se facilita la interfaz de comunicación con el microcontrolador.
- Opera a un voltaje de 5V.

Para simplificar la conexión y reducir el número de pines necesarios en el microcontrolador, es que se emplea el módulo I2C HD44780. Este módulo se conecta al panel LCD y permite la comunicación a través del protocolo I2C, utilizando solo dos líneas



para la comunicación (SDA y SCL, líneas conectadas en los pines PB11 y PB10 respectivamente) con el microcontrolador, más las dos correspondientes a la alimentación.

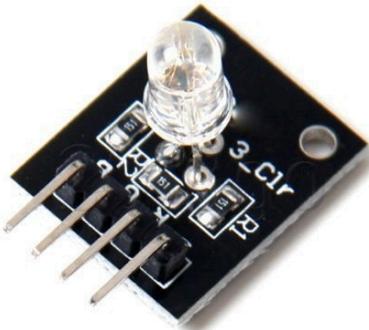
Protocolo I2C: I2C (Inter-Integrated Circuit) es un bus de comunicación serial que utiliza dos líneas:

- SDA (Serial Data): Línea de datos seriales.
- SCL (Serial Clock): Línea de reloj serial.

Se utiliza en este proyecto al panel como esclavo (slave) y el microcontrolador Blue Pill como maestro (master), para mostrar cierta información relevante, como estados del sistema, datos de sensores, y mensajes de error.

El controlador HD44780 en el panel LCD se encarga de interpretar estos comandos y actualizar en consecuencia la información que se observa en el panel. Para simplificar su uso, se utilizó una librería descargada de internet (eziya, 2019).

## Módulo emisor LED RGB KY-016



El módulo emisor LED RGB KY-016 es una unidad versátil que combina tres diodos emisores de luz (LEDs) en un solo paquete: rojo, verde y azul. Este módulo permite la generación de una amplia gama de colores mediante la mezcla aditiva de estos tres colores primarios. Algunas características Características del Módulo LED RGB KY-016

- Tipo de LED: Emisor LED de 5mm con tres colores integrados (Rojo, Verde y Azul).
- Voltaje de Operación:
  - Rojo: 2.0-2.2V
  - Verde: 3.0-3.2V
  - Azul: 3.0-3.2V
- Corriente Máxima: Aproximadamente 20mA por cada color.
- Control de Brillo: El brillo de cada color puede ser controlado mediante modulación por ancho de pulso (PWM).
- Pines del Módulo:
  - GND: Pin de tierra.
  - R: Pin de control del LED rojo.
  - G: Pin de control del LED verde.
  - B: Pin de control del LED azul.

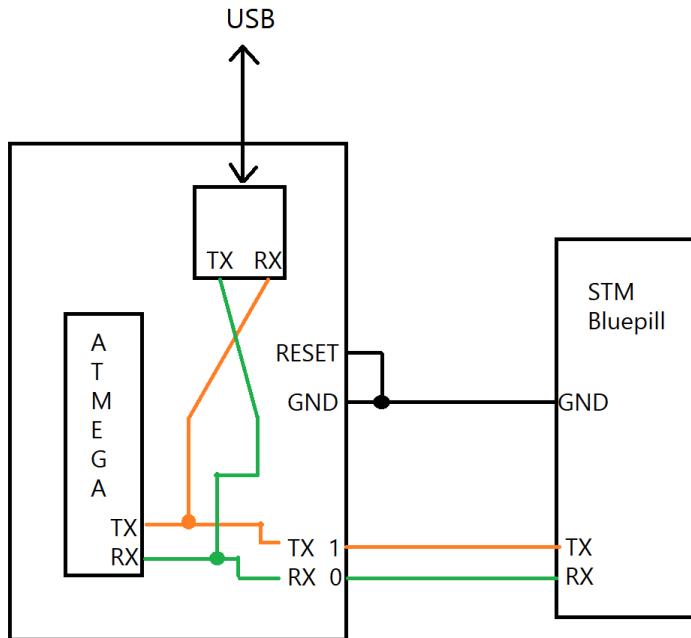


En este proyecto, el módulo LED RGB KY-016 se utiliza para proporcionar una indicación visual del estado del sistema o para crear efectos de iluminación. El microcontrolador controla los LEDs mediante señales PWM, permitiendo la creación de una amplia gama de colores y efectos. Los colores rojo, verde y azul se controlan con los pines A6, A7 y B0 respectivamente.

## Arduino UNO



El Arduino Uno es una placa de desarrollo basada en el microcontrolador ATmega328P. Es una de las placas más populares de la familia Arduino debido a su simplicidad y versatilidad. En este proyecto no se hace uso del microcontrolador Atmega328P, sino de su módulo CH340 para actuar como un puente entre la UART de la placa Blue Pill y el puerto serie de la PC. Esto es debido a que no se disponía de un módulo conversor USB a TTL y la Blue Pill no puede utilizar su puerto USB directamente para comunicarse con la UART. Es por eso que debe conectarse al Arduino como indica el siguiente esquema:



Los pines Tx y Rx de la placa de STM deben ir a los mismos de la Arduino, y no cruzados como uno esperaría. Esto es por la conexión interna que tiene Arduino con respecto al Tx y Rx de su CH340, donde se puede ver en el esquema que sí están conectados de manera cruzada. El pin RESET de Arduino se lo coloca a 0V para que al estar conectado, se ignore cualquier programa escrito en su memoria.

Se usará el pin A2 como Tx y A3 como Rx.

## Funcionamiento general

La idea principal de este proyecto es controlar la posición de un objeto sobre una canaleta mediante un controlador PID, ajustando su ubicación a un setpoint definido. Este setpoint puede ser establecido a través de un comando enviado vía UART o al posicionar manualmente un objeto en una distancia específica detectada por el sensor inferior. El sistema cuenta con varios modos de operación, cada uno con un comportamiento específico. La comunicación con el usuario se realiza tanto por UART como a través del panel LCD, proporcionando la información relevante para cada modo de funcionamiento.

Como se acaba de mencionar, el sistema cuenta con varios modos de funcionamiento que son específicamente 4 y que se detallan a continuación:

- **Modo Energizado:** Este es el modo predeterminado del sistema. En este modo, todos los periféricos están energizados pero no realizan ninguna actividad específica, excepto el servomotor, que si se encuentra en una posición diferente a la horizontal, ajusta la posición para dejar la barra allí para el reposo. Al activarse, el sistema envía un panel de consignas al usuario vía UART, indicando las acciones que se pueden ejecutar. Luego, el sistema entra en un estado de "standby" o bajo consumo, donde permanece



en un bucle cerrado esperando que el usuario ingrese un comando que lleve al sistema a un modo de funcionamiento diferente.

- **Modo Encendido con funcionamiento Automático:** Este es uno de los modos de funcionamiento principales del sistema. Su objetivo es posicionar el carro que se encuentra sobre el tubo superior a la misma distancia que el bloque en la base, utilizando el controlador PID. La distancia del bloque en la base es medida por el sensor ultrasónico inferior. Tanto las lecturas de distancia como la respuesta del controlador PID se realizan en tiempo real, asegurando una precisión constante en el posicionamiento del carro.
- **Modo Encendido con funcionamiento Manual:** Este es el otro modo de funcionamiento principal del sistema. Comparte la mayoría de las funcionalidades del modo automático, con la diferencia clave de que en este modo no se realizan lecturas de distancia con el sensor ultrasónico inferior. En lugar de eso, la distancia objetivo o setpoint del sistema se establece directamente en la consigna cuando se cambia a este estado. Esto permite un control manual preciso del posicionamiento del carro en el tubo superior, utilizando el controlador PID para mantener la posición deseada.
- **Modo Testeo:** Este modo está diseñado para la verificación y prueba del sistema. Permite corroborar que los sensores estén conectados y funcionando correctamente. Además, habilita un comando específico para este modo, el cual permite posicionar manualmente el servomotor en el ángulo deseado mediante comandos (siempre dentro de los parámetros físicos del sistema).

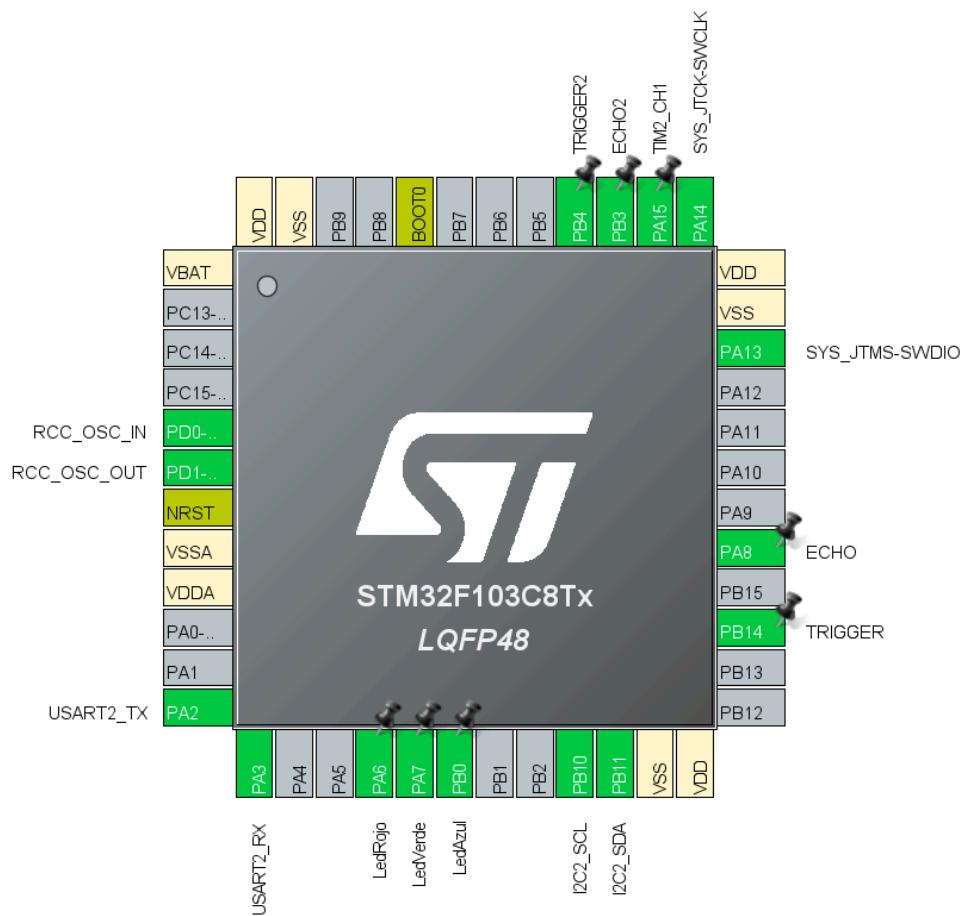
Tanto estos modos, como las demás consignas que se le pueden dar al sistema, se manejan vía UART siguiendo el siguiente protocolo de funcionamiento:

Nombre de comando	Formato	Rango	Observaciones	Ejemplo
Modo Energizado	:A	-	-	-
Modo Encendido funcionamiento Manual	:EM[num]	Números enteros entre 2 y 40	Parámetro numérico opcional. Default: 25	:EM10 :EM
Modo Encendido funcionamiento Automático	:EA	-	-	-
Modo Testeo	:T	-	-	-
Posicionamiento del servo	:S[num]	Números enteros entre 60 y 180	Sólo en modo Testeo	:S160
Ajuste de constantes PID	:KP[num] :KI[num] :KD[num]	Números reales mayores o iguales a 0	-	:KP5.5 :KI100 :KD25.65



# Configuración del microcontrolador

Mediante el software STM32CubeIDE, se realizó la completa configuración de los periféricos y los respectivos pins del microcontrolador STM32F103C8T6 que se encuentra en la placa microcontroladora elegida (Blue Pill) para el desarrollo del proyecto. La asignación resultante se observa a continuación.



Pin N...	Signal on...	GPIO out...	GPIO mode	GPIO Pul...	Maximum...	User Label	Modified
PA6	n/a	Low	Output P...	No pull-u...	Low	LedRojo	<input checked="" type="checkbox"/>
PA7	n/a	Low	Output P...	No pull-u...	Low	LedVerde	<input checked="" type="checkbox"/>
PA8	n/a	n/a	Input mode	No pull-u...	n/a	ECHO	<input checked="" type="checkbox"/>
PB0	n/a	Low	Output P...	No pull-u...	Low	LedAzul	<input checked="" type="checkbox"/>
PB3	n/a	n/a	Input mode	No pull-u...	n/a	ECHO2	<input checked="" type="checkbox"/>
PB4	n/a	Low	Output P...	No pull-u...	Low	TRIGGER2	<input checked="" type="checkbox"/>
PB14	n/a	Low	Output P...	No pull-u...	Low	TRIGGER	<input checked="" type="checkbox"/>
PB10	I2C2_SCL	n/a	Alternate ...	n/a	High		<input type="checkbox"/>
PB11	I2C2_SDA	n/a	Alternate ...	n/a	High		<input type="checkbox"/>
PA13	SYS_JT...	n/a	n/a	n/a	n/a		<input type="checkbox"/>
PA14	SYS_JTC...	n/a	n/a	n/a	n/a		<input type="checkbox"/>
PA15	TIM2 CH1	n/a	Alternate ...	n/a	Low		<input type="checkbox"/>
PA2	USART2...	n/a	Alternate ...	n/a	High		<input type="checkbox"/>
PA3	USART2...	n/a	Input mode	No pull-u...	n/a		<input type="checkbox"/>



Los pines denominados ECHO y TRIGGER corresponden a aquellos del sensor ultrasónico 1, es decir, el que mide la posición del carrito en movimiento. ECHO2 y TRIGGER2 son los pertenecientes al sensor restante (sensor inferior).

Utilizando el oscilador externo de alta velocidad (HSE) con una frecuencia de 8MHz, y multiplicado internamente por PLL (x9) del microcontrolador, determinan en un clock del sistema configurado en 72MHz. Todos los prescalers aplicados en los timers se calcularon tomando el valor de 72MHz como base. A continuación se presenta su configuración y la de los demás periféricos:

- **Timer 1:** Se lo configuró únicamente indicando que su fuente de reloj debe ser el reloj interno, prescaler de valor 71 y periodo de conteo por defecto en 65535. Las demás configuraciones se mantuvieron como se asignan por defecto.

Clock Source Internal Clock

Parameter Settings User Constants NVIC Settings DMA Settings

Configure the below parameters :

Search (Ctrl+F) ⌂ ⌂

Counter Settings

Prescaler (PSC - 16 bits value)	71
Counter Mode	Up
Counter Period (AutoReload Register)	65535
Internal Clock Division (CKD)	No Division
Repetition Counter (RCR - 8 bits value)	0
auto-reload preload	Disable

- **Timer 2:** Se configuró al canal 1 para salida generación de señal PWM, prescaler de valor 71 y periodo de conteo en 19999. Las demás configuraciones se mantuvieron como se asignan por defecto, pero se habilitó su interrupción con capacidad global.

Channel1 PWM Generation CH1

NVIC Settings DMA Settings GPIO Settings

Parameter Settings User Constants

Configure the below parameters :

Search (Ctrl+F) ⌂ ⌂

Counter Settings

Prescaler (PSC - 16 bits value)	71
Counter Mode	Up
Counter Period (AutoReload Register)	19999
Internal Clock Division (CKD)	No Division
auto-reload preload	Disable



NVIC Settings	DMA Settings	GPIO Settings	
Parameter Settings		User Constants	
NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
TIM2 global interrupt	<input checked="" type="checkbox"/>	0	0

- **I2C 2:** A este periférico simplemente se lo habilitó, manteniendo la configuración de sus parámetros por defecto.

NVIC Settings	DMA Settings	GPIO Settings
Parameter Settings		User Constants
Configure the below parameters :		
<input type="text"/> Search (Ctrl+F)	<input type="button"/>	<input type="button"/>
Master Features		
I2C Speed Mode	Standard Mode	
I2C Clock Speed (Hz)	100000	
Slave Features		
Clock No Stretch Mode	Disabled	
Primary Address Length selection	7-bit	
Dual Address Acknowledged	Disabled	
Primary slave address	0	
General Call address detection	Disabled	

- **USART 2:** Se configuró este dispositivo para funcionar en modo asíncrono (UART) a una velocidad de 115200 Bits/s y se habilitó su interrupción global también. Demás parámetros, por defecto.

NVIC Settings	DMA Settings	GPIO Settings
Parameter Settings		User Constants
Configure the below parameters :		
<input type="text"/> Search (Ctrl+F)	<input type="button"/>	<input type="button"/>
Basic Parameters		
Baud Rate	115200 Bits/s	
Word Length	8 Bits (including Parity)	
Parity	None	
Stop Bits	1	
Advanced Parameters		
Data Direction	Receive and Transmit	
Over Sampling	16 Samples	

NVIC Settings	DMA Settings	GPIO Settings	
Parameter Settings		User Constants	
NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
DMA1 channel7 global interrupt	<input checked="" type="checkbox"/>	0	0
USART2 global interrupt	<input checked="" type="checkbox"/>	0	0



- **DMA:** En conjunto con la configuración de la UART, se habilitó el canal para el manejo de la transmisión por DMA (Acceso Directo a Memoria). También se habilitó su interrupción global.

DMA Request	Channel	Direction	Priority
USART2_TX	DMA1 Channel 7	Memory To Peripheral	Low

Cabe destacar también que se configuró la interfaz SWD (Serial Wire Debug) para poder utilizar el programador ST-Link como debugger durante el proceso de desarrollo del programa.

Debug Serial Wire

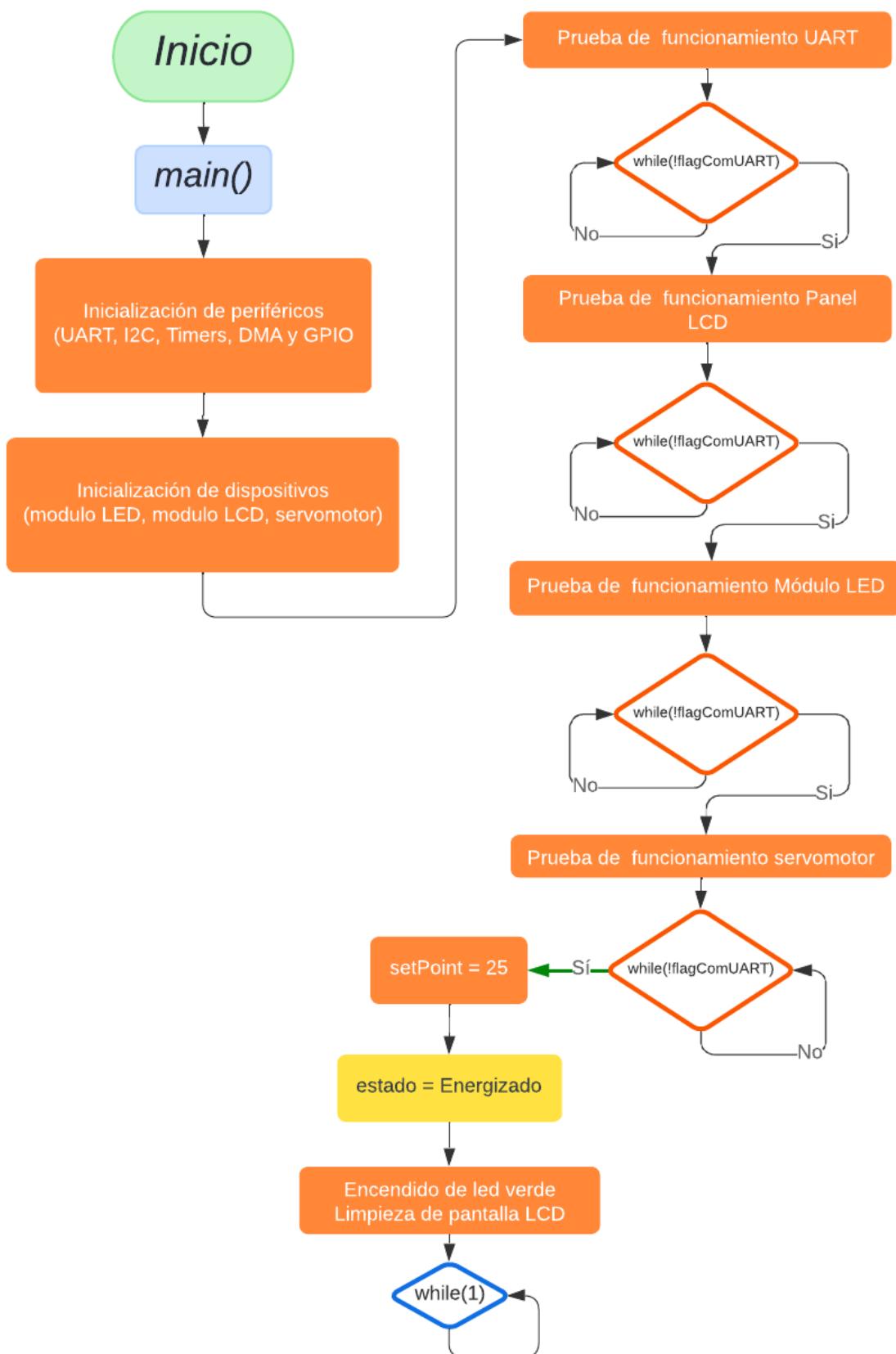
System Wake-Up

Timebase Source SysTick

## Programación

En esta sección se detallará la implementación del código del proyecto, explicando la lógica y las estructuras utilizadas para el correcto funcionamiento del sistema. Se presentará la organización del código, incluyendo las configuraciones iniciales, la lógica de los modos de operación, y la interacción con los distintos periféricos.

En primer lugar, se muestra el siguiente diagrama de flujo que detalla el modo en que inicia el código, comenzando desde el setup o sector de configuración.





```
int main(void)
{
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

    /* MCU Configuration-----*/
    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();

    /* USER CODE BEGIN Init */
    /* USER CODE END Init */

    /* Configure the system clock */
    SystemClock_Config();

    /* USER CODE BEGIN SysInit */
    /* USER CODE END SysInit */

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_TIM1_Init();
    MX_I2C2_Init();
    MX_TIM2_Init();
    MX_DMA_Init();
    MX_USART2_UART_Init();
    /* USER CODE BEGIN 2 */
    HAL_GPIO_WritePin(LedRojo_GPIO_Port, LedRojo_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(LedVerde_GPIO_Port, LedVerde_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(LedAzul_GPIO_Port, LedAzul_Pin, GPIO_PIN_RESET);

    HAL_UART_Receive_IT(&huart2, (uint8_t *)RxBUFFER, 1);
    HAL_TIM_Base_Start(&htim1);
    HAL_TIM_Base_Start_IT(&htim2);
    HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
    __HAL_TIM_SetCompare(&htim2, TIM_CHANNEL_1, 0);

    HD44780_Init(2);

    movServo(ANGMAXS);
}
```

El programa comienza, naturalmente, inicializando cada uno de los periféricos y módulos. Para evitar problemas inesperados y chequear que todo esté conectado y funcionando correctamente, se realiza una prueba donde se le pide al usuario confirmación explícita de que la UART, el módulo LCD, el LED RGB y el servomotor están respondiendo. Para esto, la UART comienza imprimiendo por pantalla un mensaje. Al confirmar el usuario que puede leer ese mensaje, el LCD procede a hacer lo mismo. Luego, el RGB cambia a sus 3 colores, y el servomotor se mueve desde 180 a 90 grados, para luego posicionarse a 160 grados (posición horizontal de la barra).



```
// Pruebas de comunicación y conexión periféricos correcta
sprintf(TxBuffer, "PRUEBA UART - RESPONDA 'Y' PARA NOTIFICAR RECEPCION\r\n");
HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);
while(!(flgComUART));
flgComUART = 0;
sprintf(TxBuffer, "PRUEBA PANEL LCD - RESPONDA 'Y' PARA NOTIFICAR RECEPCION DE TEXTO\r\n");
HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);
HD44780_Clear();
HAL_Delay(250);
sprintf(lcdBuffer, " FUNCIONAMIENTO ");
HD44780_SetCursor(0, 0);
HD44780_PrintStr(lcdBuffer);
sprintf(lcdBuffer, "     CORRECTO      ");
HD44780_SetCursor(0, 1);
HD44780_PrintStr(lcdBuffer);
while(!(flgComUART));
flgComUART = 0;
sprintf(TxBuffer, "PRUEBA MODULO RGB - RESPONDA 'Y' PARA NOTIFICAR VISUALIZACION R - G - B\r\n");
HAL_Delay(400);
HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);
HAL_GPIO_WritePin(LedRojo_GPIO_Port, LedRojo_Pin, GPIO_PIN_SET);
HAL_GPIO_WritePin(LedVerde_GPIO_Port, LedVerde_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(LedAzul_GPIO_Port, LedAzul_Pin, GPIO_PIN_RESET);
HAL_Delay(500);
HAL_GPIO_WritePin(LedRojo_GPIO_Port, LedRojo_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(LedVerde_GPIO_Port, LedVerde_Pin, GPIO_PIN_SET);
HAL_GPIO_WritePin(LedAzul_GPIO_Port, LedAzul_Pin, GPIO_PIN_RESET);
HAL_Delay(500);
HAL_GPIO_WritePin(LedRojo_GPIO_Port, LedRojo_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(LedVerde_GPIO_Port, LedVerde_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(LedAzul_GPIO_Port, LedAzul_Pin, GPIO_PIN_SET);
HAL_Delay(500);
HAL_GPIO_WritePin(LedRojo_GPIO_Port, LedRojo_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(LedVerde_GPIO_Port, LedVerde_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(LedAzul_GPIO_Port, LedAzul_Pin, GPIO_PIN_RESET);
while(!(flgComUART));
flgComUART = 0;
sprintf(TxBuffer, "PRUEBA SERVOMOTOR - RESPONDA 'Y' PARA NOTIFICAR FUNCIONAMIENTO\r\n");
HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);
for (int i = 180; i > 89; --i){
    movServo(i);
    HAL_Delay(30);
}
for (int i = 90; i < 161; ++i){
    movServo(i);
    HAL_Delay(30);
}
while(!(flgComUART));
```

Por defecto, el setpoint inicial es de 25 cm y el sistema pasa a estado Energizado. La pantalla LCD se limpia y el LED se pone de color verde, concluyendo la etapa de setup.



```
distAnt = 25;
setPoint = 25;           // Valor de setPoint inicial (por defecto)
setPointAnt = 25;        // Valor de setPoint inicial (por defecto)

tAnt = HAL_GetTick();   // Tiempo inicial en milisegundos

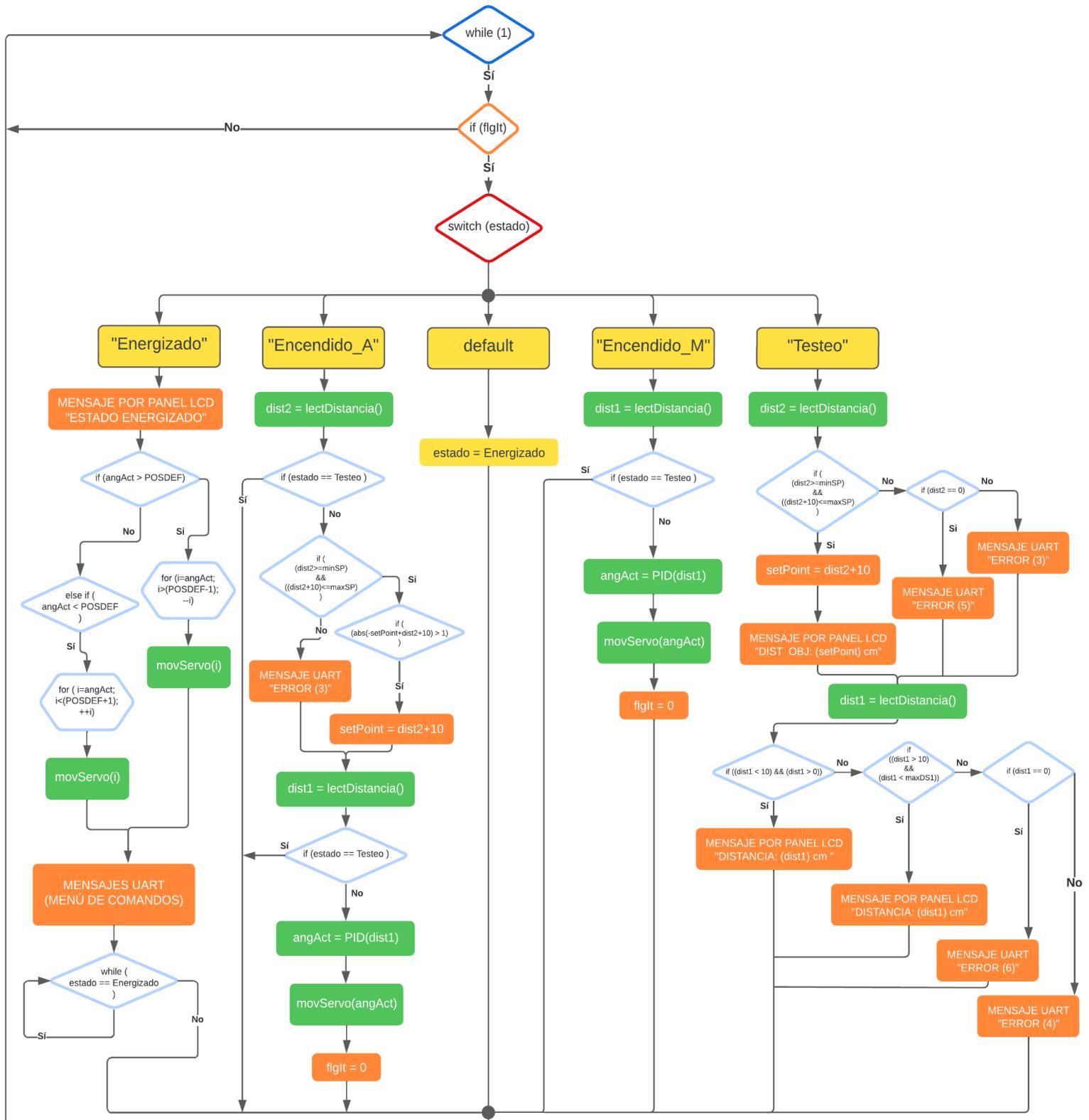
estado = Energizado;
HAL_GPIO_WritePin(LedRojo_GPIO_Port, LedRojo_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(LedVerde_GPIO_Port, LedVerde_Pin, GPIO_PIN_SET);
HAL_GPIO_WritePin(LedAzul_GPIO_Port, LedAzul_Pin, GPIO_PIN_RESET);
HD44780_Clear();

/* USER CODE END 2 */
```

En la etapa de loop, aparece una estructura tipo “switch” que permite elegir entre los 4 modos del sistema: Energizado, Encendido\_A, Encendido\_M y Testeo, entrando en el primero por default. En el estado Energizado, se presenta un menú por el monitor serie que explica con qué comandos se cambia a los demás modos. El servomotor se posiciona a 160 grados y por el LCD se printea el mensaje “ESTADO ENERGIZADO”.

Este menú informa al usuario de cómo es que podrá manejar el sistema.

Con este protocolo en mente, el sistema desarrollará su funcionamiento permanente siguiendo el diagrama de flujo a continuación:





```
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

    if (flgIt){
        switch(estado){
            case Energizado:
                sprintf(lcdBuffer, "      ESTADO      ");
                HD44780_SetCursor(0, 0);
                HD44780_PrintStr(lcdBuffer);
                sprintf(lcdBuffer, "  ENERGIZADO  ");
                HD44780_SetCursor(0, 1);
                HD44780_PrintStr(lcdBuffer);

                if (angAct > POSDEF){
                    for (int i = angAct; i > (POSDEF - 1); --i){
                        movServo(i);
                        HAL_Delay(30);
                    }
                } else if (angAct < POSDEF){
                    for (int i = angAct; i < (POSDEF + 1); ++i){
                        movServo(i);
                        HAL_Delay(30);
                    }
                }
            }

            //Interfaz de usuario
            sprintf(TxBuffer, "\r\n\r\n----- INSTRUCCIONES DEL APLICATIVO ----- \r\n");
            HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);

            sprintf(TxBuffer, "|          PROTOCOLO DE USO:  :[COM][COM/NUM][NUM]          |\r\n");
            HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);

            sprintf(TxBuffer, "|-----|-----|\r\n");
            HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);

            sprintf(TxBuffer, "|      Modo 'Energizado'      ---> :A          |\r\n");
            HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);

            sprintf(TxBuffer, "|      Modo 'Testeo'          ---> :T          |\r\n");
            HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);

            sprintf(TxBuffer, "|      Encendido 'Func_Auto'  ---> :EA         |\r\n");
            HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);

            sprintf(TxBuffer, "|      Encendido 'Func_Manual' ---> :EM[NUM] (XX) |\r\n");
            HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);

            sprintf(TxBuffer, "|      Posicionamiento Servo   ---> :S[NUM] (XX) |\r\n");
            HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);

            sprintf(TxBuffer, "|      Ajuste constantes PID   ---> :K[P/I/D][NUM] (XXX.X) |\r\n");
            HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);

            sprintf(TxBuffer, "-----\r\n");
            HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);

            while (estado == Energizado);
            break;
        }
    }
}
```



```
case Encendido_A:  
  
    // Medicion de la distancia del sensor inferior  
    dist2 = lectDistancia(TRIGGER2_GPIO_Port, TRIGGER2_Pin, ECHO2_GPIO_Port, ECHO2_Pin, 2, flgFS2);  
    if (estado == Testeo) break;  
    if ((dist2 >= minSP) && ((dist2 + 10) <= maxSP)){  
        if ((abs(-setPoint + dist2 + 10) > 1)){  
            setPoint = dist2 + 10;  
        }  
    } else {  
        sprintf(TxBuffer, "ERROR (3). POS OBJETIVO FUERA DE RANGO \r\n");  
        HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);  
    }  
  
    // Medicion de la distancia del sensor superior  
    dist1 = lectDistancia(TRIGGER_GPIO_Port, TRIGGER_Pin, ECHO_GPIO_Port, ECHO_Pin, 1, flgFS1);  
    if (estado == Testeo) break;  
    angAct = PID(dist1);  
    movServo(angAct);  
  
    flgIt = 0;  
    break;
```

En el caso de Encendido\_A o Encendido de funcionamiento Automático, el sensor inferior hace una lectura y se actualiza el setpoint si la distancia medida está dentro de un rango aceptable. Luego el sensor del carrito lee su posición, aplica la función PID y mueve el servo acorde a su salida. Se pone *flgIt* en 0 para indicar que terminó una iteración. Los chequeos de estado de testeo sirven para cortar el programa en este punto si es que la función de lectura de distancia, cambió a dicho estado, por la detección de algún problema con los sensores. De este modo, el programa sale inmediatamente del bucle y pasa en el menor tiempo posible al estado de testeo. Se le suman 10 unidades a *dist2* debido a que la distancia horizontal entre los dos sensores es de 10 cm,y se quiere medir el setpoint desde la posición del sensor superior.

```
case Encendido_M:  
    // Medicion de la distancia del sensor superior  
    dist1 = lectDistancia(TRIGGER_GPIO_Port, TRIGGER_Pin, ECHO_GPIO_Port, ECHO_Pin, 1, flgFS1);  
    if (estado == Testeo) break;  
    angAct = PID(dist1);  
    movServo(angAct);  
  
    flgIt = 0;  
    break;
```

El Encendido\_M, es similar al caso anterior excepto que el sensor de setpoint está deshabilitado y el setpoint fue dado por consigna a través de la UART.



```
case Testeo:  
    // Medicion de la distancia del sensor inferior  
    dist2 = lectDistancia(TRIGGER2_GPIO_Port, TRIGGER2_Pin, ECHO2_GPIO_Port, ECHO2_Pin, 2, flgFS2);  
    if ((dist2 >= minSP) && ((dist2 + 10) <= maxSP)){  
        setPoint = dist2 + 10;  
        sprintf(lcdBuffer, "DIST OBJ: %d cm", setPoint);  
        HD44780_SetCursor(0, 1);  
        HD44780_PrintStr(lcdBuffer);  
    } else if (dist2 == 0){  
        sprintf(TxBuffer, "ERROR (5). ULTS INF DESCONECTADO O FUERA DE RANGO\r\n");  
        HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);  
    } else {  
        sprintf(TxBuffer, "ERROR (3). POS OBJETIVO FUERA DE RANGO \r\n");  
        HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);  
    }  
  
    // Medicion de la distancia del sensor superior  
    dist1 = lectDistancia(TRIGGER_GPIO_Port, TRIGGER_Pin, ECHO_GPIO_Port, ECHO_Pin, 1, flgFS1);  
    if ((dist1 < 10) && (dist1 > 0)){  
        sprintf(lcdBuffer, "DISTANCIA: %d cm ", dist1);  
        HD44780_SetCursor(0, 0);  
        HD44780_PrintStr(lcdBuffer);  
    } else if ((dist1 > 10) && (dist1 < maxDS1)){  
        sprintf(lcdBuffer, "DISTANCIA: %d cm", dist1);  
        HD44780_SetCursor(0, 0);  
        HD44780_PrintStr(lcdBuffer);  
    }  
    else if (dist1 == 0){  
        sprintf(TxBuffer, "ERROR (6). ULTS SUP DESCONECTADO \r\n");  
        HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);  
    } else {  
        sprintf(TxBuffer, "ERROR (4). POS CARRO FUERA DE RANGO \r\n");  
        HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);  
    }  
    break;  
  
default:  
    estado = Energizado;  
    break;  
}  
}  
/* USER CODE END 3 */  
}
```

En el caso del modo Testeo, es un modo en el que ya no se responde a la salida del controlador PID, ni se responde a una consigna de setpoint, sino que es un modo preparado para permitirle al usuario verificar el correcto funcionamiento tanto de los sensores de ultrasonido, como del servomotor. Este estado está preparado para dar los mensajes de error pertinentes en cada caso según sea necesario. También se imprime por el LCD, en directo, la distancia que observa cada sensor o su correspondiente desperfecto, según corresponda. Dicho estado del sistema funciona en gran colaboración con la función de lectura de distancia (lectDistancia), la cual también se enlaza en gran medida con la función de control de funcionamiento de los sensores ultrasónicos (controlFunc).



## Funciones

```
#define PULSE_MIN 550
#define PULSE_MAX 2550

void movServo(uint8_t ang){

    uint16_t pwmServo;
    pwmServo = (uint16_t)((ang-0)*(PULSE_MAX-PULSE_MIN)/(180-0)+PULSE_MIN);
    __HAL_TIM_SetCompare(&htim2, TIM_CHANNEL_1, pwmServo);
    angAct = ang;
}
```

La función `movServo()` está configurada para que el valor del CCR que controla a la señal PWM sea proporcional al ángulo introducido. Por eso, una recta fue creada a partir de los puntos (0, PULSE\_MIN) y (180, PULSE\_MAX). El valor de estas variables se ajustó en 550 y 2550, dando un ciclo de trabajo de entre el 2.8% y el 12.8%, diferente a lo que se indicaba en la hoja de datos, pues estos valores sí corresponden realmente a los ángulos 0° y 180° respectivamente.

La salida de esta recta, entonces, es el nuevo valor del CCR. Una variable global llamada `angAct` almacenará el ángulo del servo en el instante actual.



```
uint16_t lectDistancia(GPIO_TypeDef* TRIGGERPort, uint16_t TRIGGERPin,
    GPIO_TypeDef* ECHOPort, uint16_t ECHOPin, uint8_t ULTS, uint8_t flgFS){

    uint16_t distancia;
    HAL_GPIO_WritePin(TRIGGERPort, TRIGGERPin, GPIO_PIN_SET);
    __HAL_TIM_SET_COUNTER(&htim1, 0);
    while (__HAL_TIM_GET_COUNTER(&htim1) < 10); // ESPERAR 10us
    HAL_GPIO_WritePin(TRIGGERPort, TRIGGERPin, GPIO_PIN_RESET);

    pMillis = HAL_GetTick();
    while (!(HAL_GPIO_ReadPin(ECHOPort, ECHOPin)) && ((pMillis + 10) > HAL_GetTick()));
    val1 = __HAL_TIM_GET_COUNTER(&htim1);

    pMillis = HAL_GetTick();
    while ((HAL_GPIO_ReadPin(ECHOPort, ECHOPin)) && ((pMillis + 50) > HAL_GetTick()));
    val2 = __HAL_TIM_GET_COUNTER(&htim1);

    distancia = (val2 - val1) * 0.034 / 2;
    if ((distancia > maxDS1) && (distancia != 0)){
        if (controlFunc(ULTS, flgFS)){
            if (ULTS == 1) flgFS1 = 0;
            if (ULTS == 2) flgFS2 = 0;
        }
        HAL_Delay(50);
        return 0;
    } else {
        if (ULTS == 1) flgFS1 = 1;
        if (ULTS == 2) flgFS2 = 1;

        HAL_Delay(50);
        return distancia;
    }
}
```

La llamada a la función `lectDistancia()` devolverá la lectura de un sensor ultrasónico. Los parámetros de entrada incluyen los pines Trigger y Echo y sus puertos, del ultrasónico a utilizar, además del número de ultrasónico (1 para el superior y 2 para el inferior), y su flag de falla correspondiente (`flagFS`).

Lo primero que hace es declarar la variable `distancia`, que es el parámetro a devolver. Luego, escribe un HIGH en el pin de Trigger, setea el Timer 1 a 0 y cuenta hasta 10 (equivalente a esperar 10us) para luego escribir un LOW en el mismo pin.

Consecuentemente, la función mide cuánto tiempo tarda en recibir la señal de vuelta en el pin ECHO (receptor). Usa el Timer 1 para esto. Primero, espera a que la señal de vuelta empiece a llegar (esto ocurre cuando el pin ECHO se vuelve alto). Si no llega en los próximos 10 milisegundos, se produce un timeout que saca al programa del bucle while. La variable `val1` almacena el valor del contador del Timer 1 en ese momento.

Luego, espera hasta que el pin ECHO pase de alto a bajo, lo que indica que el pulso de la señal ultrasónica ha terminado. Dicha espera tiene un tiempo máximo de 50ms. Si el pin ECHO no cambia a bajo en este tiempo, el bucle se detiene para evitar que el programa se quede bloqueado. Almacena en `val2` el nuevo valor del contador. La diferencia entre `val2` y `val1` es el tiempo en microsegundos que tarda la onda ultrasónica en ir y volver. Multiplicando por 0.034 cm/us y dividiendo por 2, se obtiene la distancia del objeto.



Si la distancia calculada es mayor que un valor máximo predefinido (*maxDS1*) o si la distancia es cero (lo que indica que no se obtuvo una medición válida), realiza algunas acciones de control basadas en los parámetros *ULTS* y *flgFS*, llamando a la función *controlFunc()*.

Si la distancia es válida (dentro de un rango aceptable), la función devuelve esta distancia. Si no es válida, devuelve 0 y ajusta el flag del sensor correspondiente a 1.

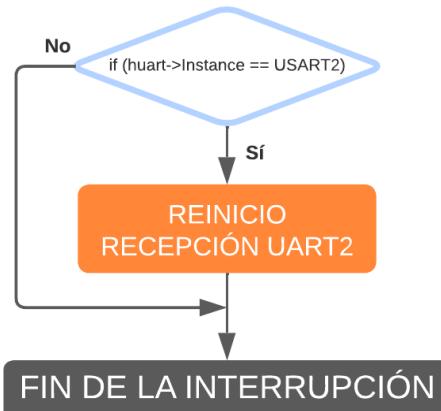
Después de realizar las mediciones y ajustes, la función introduce un pequeño retraso (50 milisegundos) para dar tiempo a que los ajustes se estabilicen antes de continuar.

```
uint8_t controlFunc(uint8_t ULTS, uint8_t flgF){  
  
    if (flgF){  
        estado = Testeo;  
        memset(com, 0, sizeof(com));  
        com[0] = 'T';  
        interpComandos();  
        if (ULTS == 1){  
            sprintf(lcdBuffer, "ULTS SUP FALL/FR");  
            HD44780_SetCursor(0, 0);  
            HD44780_PrintStr(lcdBuffer);  
            movServo(POSDEF);  
        } else if (ULTS == 2){  
            sprintf(lcdBuffer, "ULTS INF FALL/FR");  
            HD44780_SetCursor(0, 1);  
            HD44780_PrintStr(lcdBuffer);  
            movServo(POSDEF);  
        }  
        return 1;  
    } else {  
        return 1;  
    }  
}
```

*ControlFunc()* chequea si la flag de falla de los ultrasónicos está en 1, y si es así, el sistema pasa a modo Test automáticamente e imprime por el LCD un mensaje que indica cuál es el sensor que falla. En todos los casos, el servo regresa a su posición por defecto (160°).



INTERRUPCIÓN POR TRANSMISIÓN DE  
MENSAJE VIA UART



```
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart) {  
    if (huart->Instance == USART2) {  
        HAL_UART_Receive_IT(&huart2, (uint8_t *)RxBuffer, 1);  
    }  
}
```

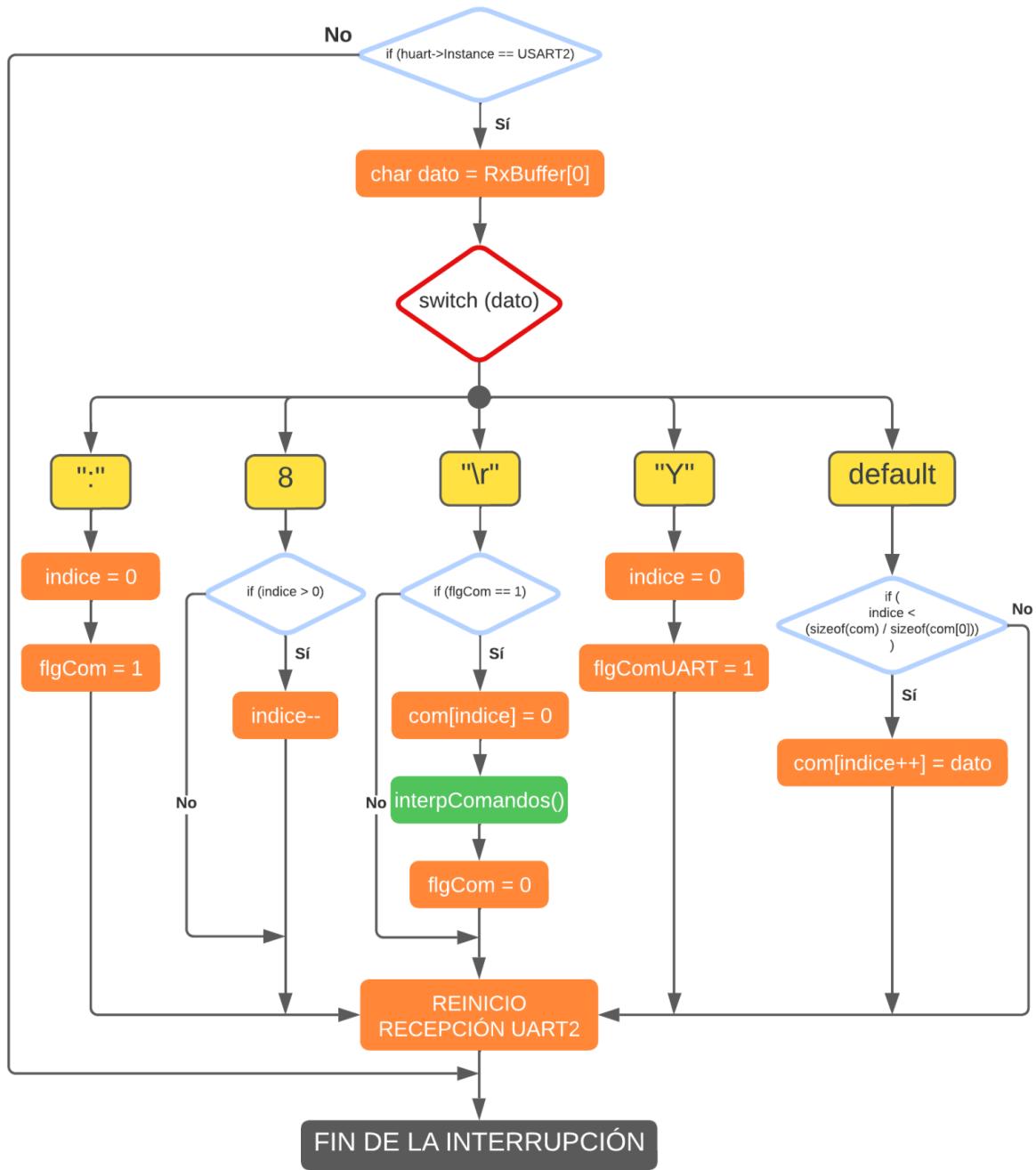
HAL\_UART\_TxCpltCallback() es una función de callback que se llama automáticamente cuando la transmisión de datos UART se completa. La función verifica que la interrupción provenga del canal USART2, y reinicia la recepción de datos para leer el siguiente byte a través de interrupciones.



```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){\n\n    if (huart->Instance == USART2){\n        char dato = RxBuffer[0];\n        switch(dato){\n            case ':':\n                indice = 0;\n                flagCom = 1;\n                break;\n\n            case 8:\n                if (indice > 0){\n                    indice = indice - 1;\n                }\n                break;\n\n            case '\\r':\n                if (flagCom == 1){\n                    com[indice] = 0;\n                    interpComandos();\n                    flagCom = 0;\n                }\n                break;\n\n            case 'Y':\n                indice = 0;\n                flgComUART = 1;\n                break;\n            default:\n                if (indice < (sizeof(com) / sizeof(com[0]))){\n                    com[indice++] = dato;\n                }\n                break;\n        }\n    }\n    HAL_UART_Receive_IT(&huart2, (uint8_t *)RxBuffer, 1);\n}
```



INTERRUPCIÓN POR RECEPCIÓN DE  
MENSAJE VIA UART



HAL\_UART\_RxCpltCallback() es otra función callback que se llama automáticamente cuando se recibe un byte de datos UART. Se encarga de procesar el byte recibido y actualizar el buffer de recepción *RxBuffer*.

Procesamiento de Datos:

- ":" : Cuando se recibe el carácter ':', se reinicia el índice y se activa la bandera *flgCom* para indicar que se ha comenzado un nuevo comando.



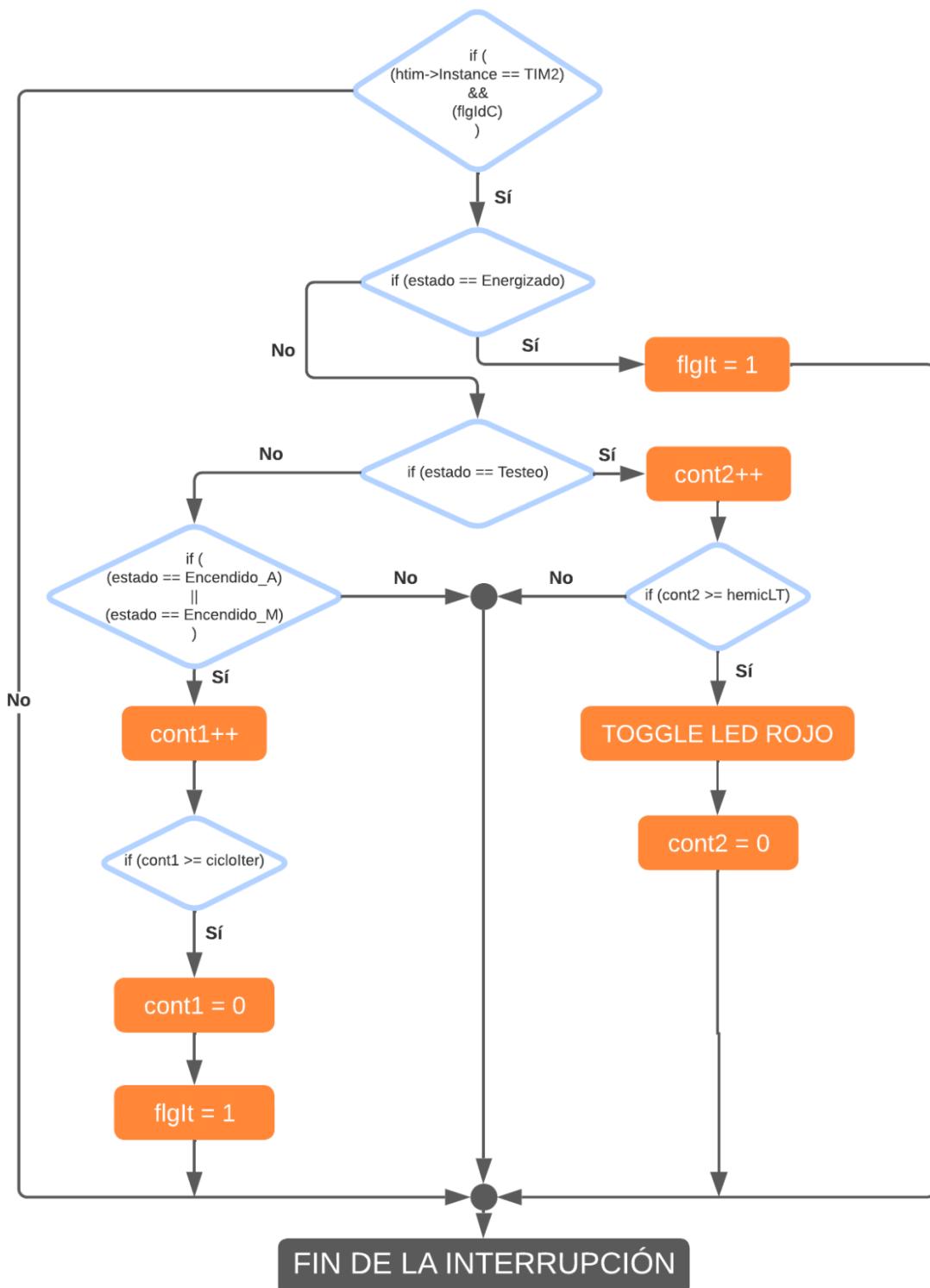
- **8 (Retroceso o Backspace):** Cuando se recibe el carácter de retroceso, si el índice es mayor que 0, se decrementa el índice para eliminar el último carácter del buffer com.
- “\r” (**Enter o Return**): Cuando se recibe el carácter de retorno de carro (“\r”), si *flgCom* está activado, se termina el comando actual, se añade un carácter nulo (0) al final de com para marcar el fin de la cadena, se llama a *interpComandos()* para interpretar el comando y se desactiva *flgCom*.
- “Y”: Cuando se recibe el carácter “Y”, se reinicia el índice y se activa la bandera *flgComUART*. Este carácter solo tiene uso en el setup, cuando se le pide confirmación al usuario del correcto funcionamiento de los dispositivos.
- **Otros caracteres:** Cualquier otro carácter recibido se añade al buffer “com” si el índice no supera el tamaño del buffer.

Finalmente, se vuelve a habilitar la recepción de datos en modo de interrupción (HAL\_UART\_Receive\_IT), asegurando que el sistema continúe recibiendo bytes.

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){  
  
    // Interrupción cada 20ms  
    if ((htim->Instance == TIM2) && (flgIdC)){  
        if (estado == Energizado){  
            flgIt = 1;  
        } else if (estado == Testeo){  
            cont2++;  
            if (cont2 >= hemicLT){          // Ciclo cada 20ms x hemicLT  
                HAL_GPIO_TogglePin(LedRojo_GPIO_Port, LedRojo_Pin);  
                cont2 = 0;  
            }  
        } else if ((estado == Encendido_A) || (estado == Encendido_M)){  
            cont1++;  
            if (cont1 >= cicloIter){      // Ciclo cada 20ms x cicloIter  
                cont1 = 0;  
                flgIt = 1;  
            }  
        }  
    }  
}
```



INTERRUPCIÓN POR CUMPLIMIENTO DE  
PERIODO - TIMER 2



`HAL_TIM_PeriodElapsedCallback` es una función *callback* que se llama automáticamente cada vez que el temporizador Timer 2 alcanza el periodo de interrupción



configurado (en este caso, cada 20 ms). Dependiendo del estado del sistema, realiza diferentes acciones.

La función primero verifica si la interrupción proviene del temporizador Timer 2. Después, se verifica la flag *flgIdC* para asegurarse de que se deben realizar las acciones programadas. Esta flag existe para darle más “prioridad” al intérprete de comandos, pues cuando *interpComandos()* está ejecutándose, la flag vale 0 y la interrupción no ejecuta código.

#### Acciones Basadas en el Estado:

- **Estado Energizado:** Si el estado es Energizado, se establece la bandera *flglT* en 1. Esto influye en el *main()*, haciendo que el loop se ejecute en intervalos controlados, y por ende los sensores miden cada cierto tiempo controlado.
- **Estado Testeo:** Si el estado es Testeo, se incrementa el contador *cont2*. Si *cont2* alcanza o supera el valor de *hemicLT*, se alterna el estado del LED rojo usando *HAL\_GPIO\_TogglePin()*. Esto crea un parpadeo del LED en intervalos definidos por *hemicLT*. Actualmente, *hemicLT* se seteó en 12, para que el LED rojo parpadee cada 240ms. Luego, *cont2* se restablece a 0 para comenzar el ciclo nuevamente.
- **Estado Encendido\_A o Encendido\_M:** Si el estado es Encendido\_A o Encendido\_M, se incrementa el contador *cont1*. Si *cont1* alcanza o supera el valor de *ciclolter*, se restablece *cont1* a 0 y se establece *flglT* en 1. Como *ciclolter* es igual a 6, el loop principal se ejecuta de nuevo cada 120ms, lo que significa que los sensores leen cada ese intervalo.



```
void interpComandos(){

    flgIdC = 0;
    if (flgComUART == 1){
        switch(com[0]){
            case 'A':
                estado = Energizado;
                HAL_GPIO_WritePin(LedRojo_GPIO_Port, LedRojo_Pin, GPIO_PIN_RESET);
                HAL_GPIO_WritePin(LedAzul_GPIO_Port, LedAzul_Pin, GPIO_PIN_RESET);
                HAL_GPIO_WritePin(LedVerde_GPIO_Port, LedVerde_Pin, GPIO_PIN_SET);
                sprintf(TxBuffer, "ESTADO: ENERGIZADO \r\n");
                HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);
                break;

            case 'T':
                estado = Testeo;
                HAL_GPIO_WritePin(LedRojo_GPIO_Port, LedRojo_Pin, GPIO_PIN_RESET);
                HAL_GPIO_WritePin(LedAzul_GPIO_Port, LedAzul_Pin, GPIO_PIN_RESET);
                HAL_GPIO_WritePin(LedVerde_GPIO_Port, LedVerde_Pin, GPIO_PIN_RESET);
                movServo(POSDEF);
                sprintf(TxBuffer, "ESTADO: TESTEO \r\n");
                HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);
                break;

            case 'E':
                HAL_GPIO_WritePin(LedRojo_GPIO_Port, LedRojo_Pin, GPIO_PIN_RESET);
                HAL_GPIO_WritePin(LedAzul_GPIO_Port, LedAzul_Pin, GPIO_PIN_RESET);
                HAL_GPIO_WritePin(LedVerde_GPIO_Port, LedVerde_Pin, GPIO_PIN_RESET);

                if (com[1] == 'M'){
                    if ((atoi(&com[2]) >= minSP) && (atoi(&com[2]) <= maxSP)){
                        estado = Encendido_M;
                        setPoint = atoi(&com[2]);
                        sprintf(TxBuffer, "ESTADO: ENCENDIDO - MODO: MANUAL\r\n");
                        HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);

                        sprintf(lcdBuffer, "ESTADO ENCENDIDO");
                        HD44780_SetCursor(0, 0);
                        HD44780_PrintStr(lcdBuffer);
                        sprintf(lcdBuffer, "MODO: MANUAL ");
                        HD44780_SetCursor(0, 1);
                        HD44780_PrintStr(lcdBuffer);
                    } else {
                        estado = Encendido_M;
                        setPoint = 25;
                        sprintf(TxBuffer, "NUM INVALIDO. POS OBJETIVO POR DEFECTO (25)\r\n");
                        HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);
                    }
                }
        }
    }
}
```



```
    } else if(com[1] == 'A'){
        estado = Encendido_A;
        sprintf(TxBuffer, "ESTADO: ENCENDIDO - MODO: AUTOMATICO\r\n");
        HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);
        sprintf(lcdBuffer, "ESTADO ENCENDIDO");
        HD44780_SetCursor(0, 0);
        HD44780_PrintStr(lcdBuffer);
        sprintf(lcdBuffer, "MODO: AUTOMATICO");
        HD44780_SetCursor(0, 1);
        HD44780_PrintStr(lcdBuffer);

    }else{
        sprintf(TxBuffer, "MODO NO VALIDO \r\n");
        HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);
    }
    break;

case 'S':
    if (estado == Testeo){
        if (com[1] != '\0'){
            angAct= atoi(&com[1]);
            angAct = (angAct > ANGMAXS) ? ANGMAXS : ((angAct < 60) ? 60 : angAct);
            movServo(angAct);
            sprintf(TxBuffer, "NUEVA POSICION SERVOMOTOR: %d \r\n", angAct);
            HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);
        } else{
            sprintf(TxBuffer, "POSICION ACTUAL SERVOMOTOR: %d \r\n", angAct);
            HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);
        }
    }else{
        sprintf(TxBuffer, "ERROR (1). MODO TESTEO NO CONFIGURADO \r\n");
        HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);
    }
    break;

case 'K':
    if ((com[1] != '\0') && (atof(&com[2]) >= 0.0)){
        if (com[1] == 'P'){
            nuevoVal = atof(&com[2]);
            nuevoVal = round(nuevoVal * 100.0) / 100.0;
            Kp = nuevoVal;
        } else if (com[1] == 'I'){
            nuevoVal = atof(&com[2]);
            nuevoVal = round(nuevoVal * 100.0) / 100.0;
            Ki = nuevoVal;
        } else if (com[1] == 'D'){
            nuevoVal = atof(&com[2]);
            nuevoVal = round(nuevoVal * 100.0) / 100.0;
            Kd = nuevoVal;
        } else {
            sprintf(TxBuffer, "ERROR (2.1). LETRA DE CONTROL INCORRECTA \r\n");
        }
    }
```



```
    HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);
    break;
}
sprintf(TxBuffer, "Kp=% .2f , Ki=% .2f , Kd=% .2f \r\n", Kp, Ki, Kd);
HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);

} else {
    sprintf(TxBuffer, "ERROR (2). ENTRADA NULA O NUMERO NEGATIVO \r\n");
    HAL_UART_Transmit(&huart2, (uint8_t*)TxBuffer, strlen(TxBuffer), 100);
}
break;

default:
    printf("COMANDO INCORRECTO \r\n");
    break;

}
flgIdC = 1;
}
```

La flag *flgIdC* se establece en 0 al inicio de *interpComandos()* por lo dicho anteriormente. La función solo procesa comandos si *flgComUART* es 1, que indica que se ha recibido un comando válido por UART. Se usa un switch para determinar qué acción tomar basado en el primer carácter del comando (*com[0]*).

#### Interpretación de Comandos:

- **Comando 'A'**: Cambia el estado a Energizado. Apaga los LEDs rojo y azul, enciende el LED verde. Envía un mensaje por UART indicando el estado Energizado.
- **Comando 'T'**: Cambia el estado a Testeo. Apaga todos los LEDs (pero recordar que por la interrupción del Timer 2, el LED rojo parpadeará) y mueve el servomotor a la posición definida por default (*POSDEF*). Envía un mensaje por UART indicando el estado Testeo.
- **Comando 'E'**: Apaga todos los LEDs. Si el segundo carácter (*com[1]*) es 'M' (Manual), verifica que el valor especificado esté dentro del rango permitido (*minSP* a *maxSP*). Si es válido, establece el estado a *Encendido\_M* y el setPoint a este valor. Si el setpoint no es válido, establece un valor por defecto (25) y notifica al usuario. Si el segundo carácter es 'A' (Automático), establece el estado a *Encendido\_A* y notifica al usuario. Si el modo no es válido, envía un mensaje de error por UART.
- **Comando 'S'**: Solo se procesa si el estado es Testeo. Caso contrario, dará un mensaje de error por UART. Si hay un valor después del 'S', se actualiza la posición del servomotor y envía el nuevo valor por UART. Si no hay valor, envía la posición actual del servomotor por UART. Las posiciones en este caso pueden variar entre 60° y 180°. Si se va de ese rango, el servo se coloca en el límite más próximo.
- **Comando 'K'**: Permite ajustar los valores de las constantes del controlador PID (Kp, Ki, Kd). Verifica que el valor después de 'K' sea un número no negativo. Ajusta el valor correspondiente (Kp, Ki, Kd) y envía los nuevos valores por UART. Si la letra de control es incorrecta o el valor es negativo, envía un mensaje de error por UART.



- **Comando Incorrecto:** Si el comando no coincide con ninguno de los casos, envía un mensaje de comando incorrecto por UART.

*FlgIdC* se establece en 1 al final de la función para indicar que ha terminado el procesamiento del comando.

```
> uint16_t PID(uint16_t Input) {  
  
    // Validación de la lectura del sensor  
    if (abs((int16_t)Input - (int16_t)distAnt) > maxDif) {  
        Input = distAnt;  
    }  
  
    Input = 0.53 * Input + 0.47 * distAnt;  
    setPoint = 0.53 * setPoint + 0.47 * setPointAnt;  
  
    uint16_t Respuesta;  
    tAct = HAL_GetTick();  
    deltaT = (tAct - tAnt) / 1000.0;  
    errAct = -setPoint + Input;  
  
    // Cálculo del término proporcional  
    termP = Kp * errAct;  
    termP = (termP > 180) ? 180 : ((termP < -180) ? -180 : termP); // Saturación del término  
  
    // Cálculo del término derivativo  
    errDer = (errAct - errAnt) / deltaT;  
    termD = Kd * errDer;  
    termD = 0.56 * termD + 0.44 * termDAnt;  
    termD = (termD > 180) ? 180 : ((termD < -180) ? -180 : termD); // Saturación del término  
  
    // Anti-windup para el término integral  
    double respSat = termP + termI + termD;  
    if (respSat > 180) {  
        respSat = 180;  
    } else if (respSat < -180) {  
        respSat = -180;  
    } else {  
        termI += Ki * errAct * (deltaT);  
        termI = (termI > 180) ? 180 : ((termI < -180) ? -180 : termI); // Saturación del término  
    }  
  
    // Cálculo de la salida PID  
    respSat = ((termP + termI + termD) > 180.0) ? 180.0 :  
             (((termP + termI + termD) < -180.0) ? -180.0 : respSat); // Saturación del término  
    Respuesta = (uint16_t)((respSat + ANGMXPID) / 360.0 * (ANGMAXPID - ANGMINPID)) + ANGMINPID;  
  
    // Actualización de variables  
    errAnt = errAct;  
    tAnt = tAct;  
    distAnt = Input;  
    termDAnt = termD;  
}
```



```
setPointAnt = setPoint;

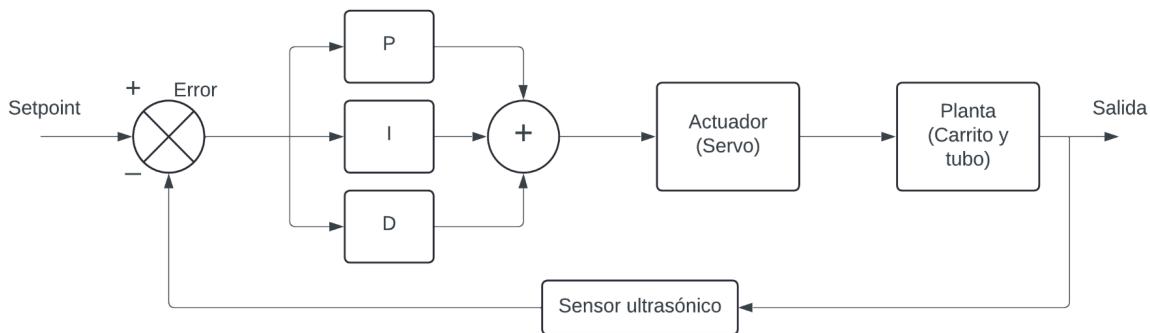
// Control LED
if (Input == setPoint) {
    HAL_GPIO_WritePin(LedAzul_GPIO_Port, LedAzul_Pin, GPIO_PIN_SET);
} else {
    HAL_GPIO_WritePin(LedAzul_GPIO_Port, LedAzul_Pin, GPIO_PIN_RESET);
}

// Transmisión de datos por UART
sprintf(TxBuffer, "O:%d D:%d R:%d\n", setPoint, dist1, Respuesta);
HAL_UART_Transmit_DMA(&huart2, (uint8_t *)TxBuffer, strlen(TxBuffer));

/*sprintf(TxBuffer, "P:%.2f I:%.2f D:%.2f\r\n", termP, termI, termD);
HAL_UART_Transmit_DMA(&huart2, (uint8_t *)TxBuffer, strlen(TxBuffer));*/

return Respuesta;
}
```

El control PID implementado es sencillo y común, con algunas modificaciones. El siguiente esquema representa el funcionamiento general del sistema de control.



Primeramente, se hace una validación. Si la diferencia entre la lectura actual del sensor y la lectura anterior es mayor que *maxDif*, la lectura actual se considera errónea y se usa la lectura anterior (*distAnt*) en su lugar. Esto se debe a que si en dos lecturas contiguas, que están separadas controladamente 120ms, el carrito se movió una distancia grande, significa que hay un error en la lectura pues es físicamente imposible para el carrito alcanzar una velocidad tan grande.

Se aplica luego un filtro para suavizar las lecturas del sensor y el setPoint, usando un filtro de promedio ponderado. Este tipo de filtro se lo conoce como “filtro exponencial”. De esta forma, la salida no será tan brusca frente a cambios grandes.

Se calcula el *deltaT* restando el tiempo en una lectura anterior con el actual y se calcula el error entre la posición actual y el setpoint. Se procede a calcular los términos P y D, este último presenta un filtrado similar al anterior. Se los satura para que estén en un rango entre -180 y 180. Recordar que las ganancias Kp, Kd y Ki son variables globales dentro del código y el usuario puede cambiarlas en cualquier momento.

Para el término integral, se aplica un anti-windup. El anti-windup es una técnica utilizada en los controladores PID para evitar que el término integral se acumule en exceso (lo que se conoce como "windup"), especialmente cuando el actuador ha alcanzado sus



límites físicos y no puede seguir la señal de control. En este caso, se aplica un mecanismo para limitar la acumulación del término integral y así mantener la estabilidad del sistema. Si la respuesta está saturada, el término integral no se acumula, y también satura a los mismos valores que los dos anteriores.

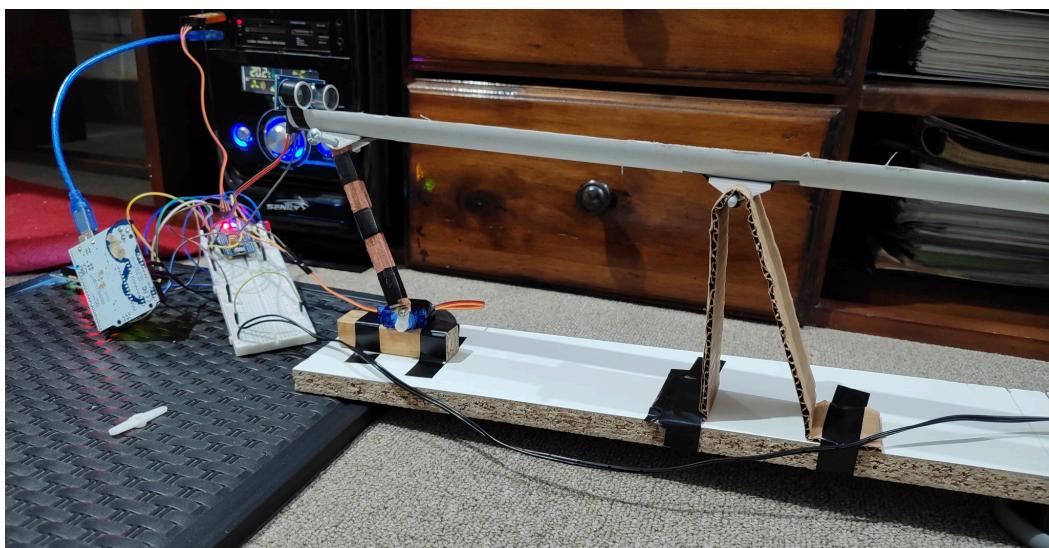
Se aplica la saturación final y se convierte la respuesta PID a un valor dentro del rango de ángulos permitido. Este rango está entre  $111^\circ$  y  $180^\circ$ , pues se comprobó que a esos límites el tubo se inclina un máximo en ambos sentidos.

Por último, se actualizan las variables que son necesarias para la siguiente vez que la función sea llamada. Además, el LED se encenderá en azul cuando el carrito pase por el setpoint, como un indicativo visual. Luego, se envían los datos de salida, setpoint, y respuesta a través de UART. Esto es útil porque es posible graficarlos utilizando el “Serial Plotter” del IDE de Arduino, dando una gráfica en tiempo real.

## Etapas de montaje y ensayos realizados

Los primeros ensayos realizados a priori, fueron aquellos para aprender a utilizar cada uno de los módulos por separado, con la placa Blue Pill. El uso de la librería HAL hizo sencillo la programación y esto resultó en un éxito relativamente rápido en el manejo de cada uno de los periféricos.

Para un primer modelo, luego de programar el comportamiento principal de control, se realizó un montaje improvisado simplemente utilizando objetos que se encontraban en casa en ese momento. El soporte principal se hizo de cartón, y el brazo que conecta al tubo con el servomotor se realizó de una manera improvisada con tres palitos de helado pegados con cola y sujetados con cinta. Para el carril donde debe desplazarse el móvil, se utilizó un tubo de PVC cortado a la mitad. Algo que no se encuentra en casa y que sí se realizó previamente, fueron unas piezas bastante rústicas impresas en 3D con las que se pudo pasar un tornillo y lograr la sujeción de ambas partes. De soporte para el servomotor, se utilizó un pequeño bloque de madera, con lo que se obtuvo como resultado el sistema que se ve a continuación.

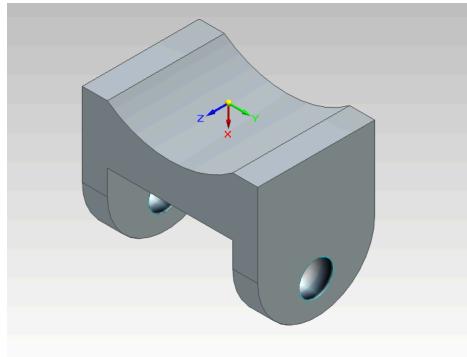
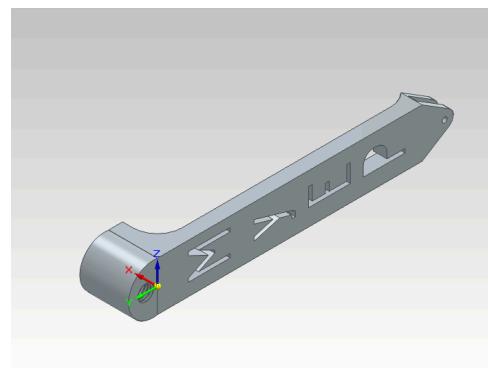
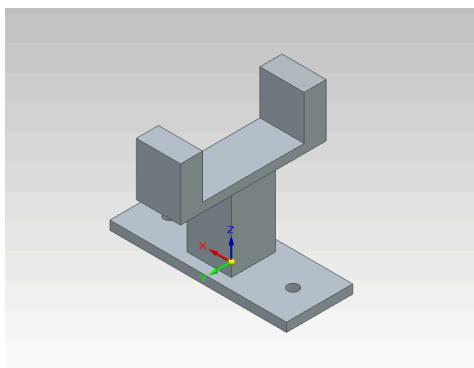


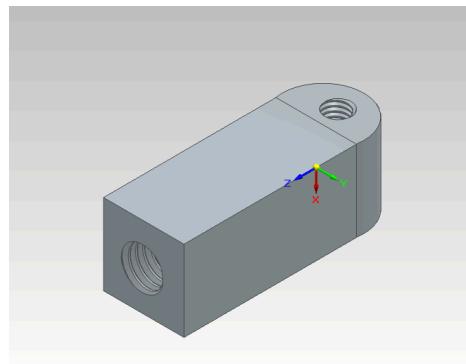
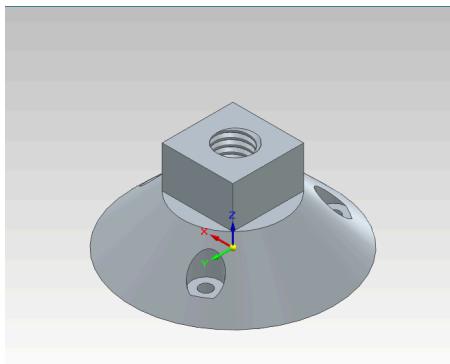


Un primer ensayo demostró un comportamiento incorrecto por parte del sensor. Esto fue debido a que el objeto medido era una pelota (cuya superficie observable era redonda) en lugar de una superficie plana. La pelota, al no poseer una superficie plana, provocaba que las ondas ultrasónicas rebotaran fuera del alcance del sensor, dando así lecturas erróneas. A pesar de ello, al colocar un objeto plano como una regla, se observaba que el servomotor respondía coherentemente según si estaba por encima o por debajo del setpoint. Además, la UART inicialmente devolvía solamente el dato de distancia. Como se puede ver, no existía el sensor inferior para modificar en el valor del setpoint. Todas estas pruebas se realizaron entonces, con el setpoint en la mitad del tubo y este no se podía modificar.

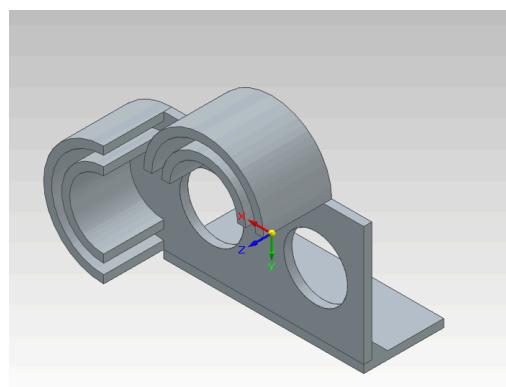
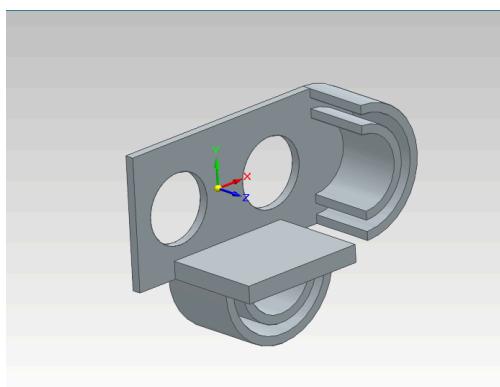
Luego de estas pruebas, se hizo foco en la formación de piezas y partes para robustecer al sistema y que ello permita una confiabilidad mayor del sistema, en lo que respecta a los posibles movimientos inesperados o las malas lecturas de distancias, lo que llevaban al control a responder de manera errónea.

Fue aquí cuando se volvió esencial el diseño e impresión en 3D. Se comenzó por realizar algunas partes tales como un brazo que relate al servomotor con el tubo, un soporte para asegurarla a la base y fijar su posición, también se rediseñaron los agarres del tubo para que sean más fieles y ahora compatibles con las nuevas piezas. Se pensó en una estructura que fuera atornillada a la madera base y que soportase una espiga roscada en su centro haciendo de columna central. En el otro extremo de esa espiga se colocaría una pieza prismática con un roscado interno que le permitiera acoplarse a la espiga, y uno transversal en el otro extremo apto para un tornillo pasante que lo sujetase en conjunto con la pieza en la que se apoya el tubo. Estas piezas fueron concebidas con el objetivo de permitir ajustar la altura del tubo y un fácil agarre también. Los modelos pueden verse a continuación.





Para soportar a los sensores ultrasónicos, se diseñó una pieza capaz de ser encastrada en el extremo del tubo. Esta pieza puede ser colocada horizontal o verticalmente, lo que también ayudó a determinar la mejor posición para el correcto funcionamiento del sensor ultrasónico.



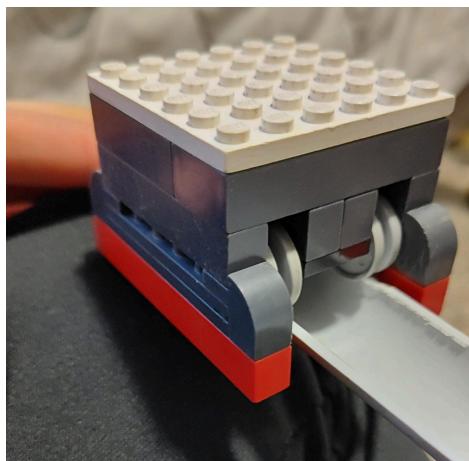


En lo que respecta al carrito a utilizar, éste pasó por muchas formas y modelos, debido principalmente a problemas de fricción que habían con el tubo. Esto provocaba que el sistema prácticamente no funcionase. Fue difícil encontrar la forma de hacer que deslice correctamente y que a su vez mantenga una cara plana para lecturas correctas del sensor.

Se probaron algunos más pesados, otros más livianos, más chicos, más grandes, también se intentó diseñar uno que contase con unos pequeños rodamientos y unas carcasa cónicas para sujetarse del tubo y no caerse, pero resultó ser muy pesado y por eso prácticamente no podía desplazarse.

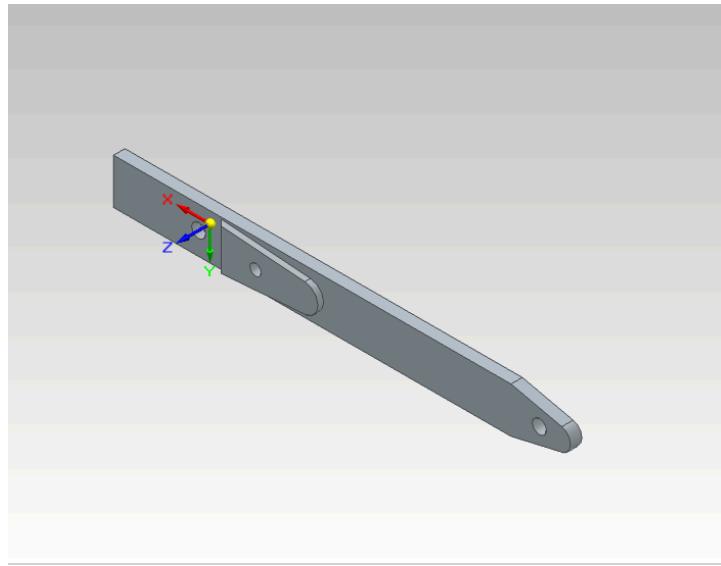
Finalmente el modelo que mejor funcionó, aunque tampoco ideal como se hubiera querido, fue construido con piezas LEGO. Este, a pesar del avance que implicó, poseía ciertos problemas de fricción, que impedían su deslizamiento en numerosas ocasiones.

A continuación, se muestra una pequeña selección de algunos de los carritos que se hicieron.

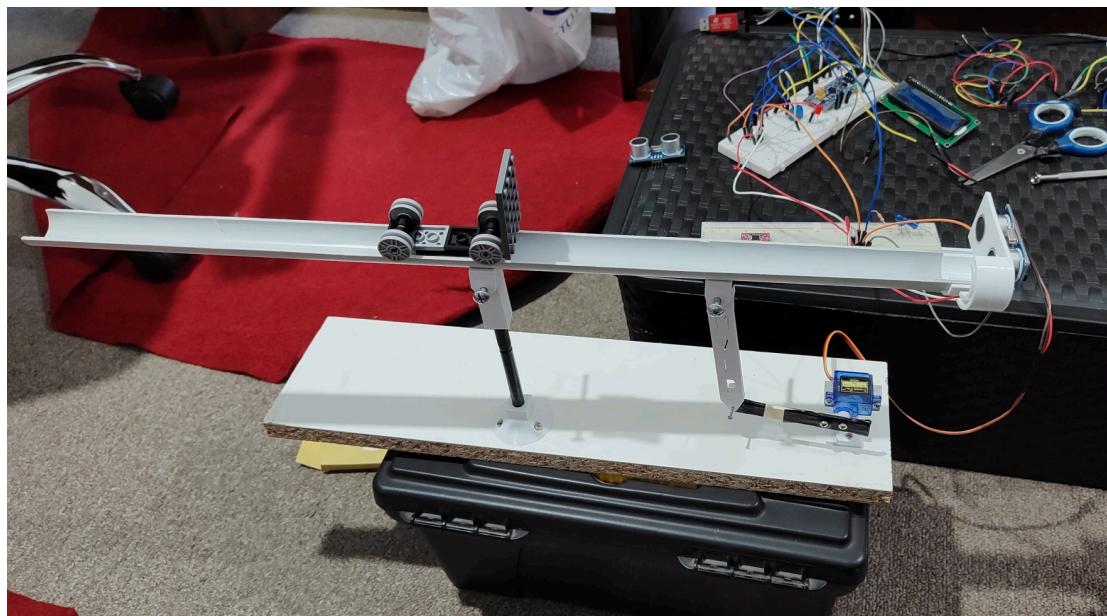




Ante los problemas que poseía el carrito para desplazarse, se realizaron ensayos que demostraban (claramente) que modificando la altura central del tubo, se lograba una mayor inclinación, lo cual conllevaba a que el carrito pudiese moverse en la mayoría de las ocasiones. Para esto, se realizó el cambio de la espiga roscada, por una más larga y que como consecuencia directa, hizo que se tuviera que diseñar e imprimir un segundo brazo para el servomotor. Este nuevo brazo hace las veces de extensión del que trae el propio servomotor y aporta a una conexión que permite que el tubo se pueda desplazar desde una altura superior a la máxima previa, e inferior a la mínima también.

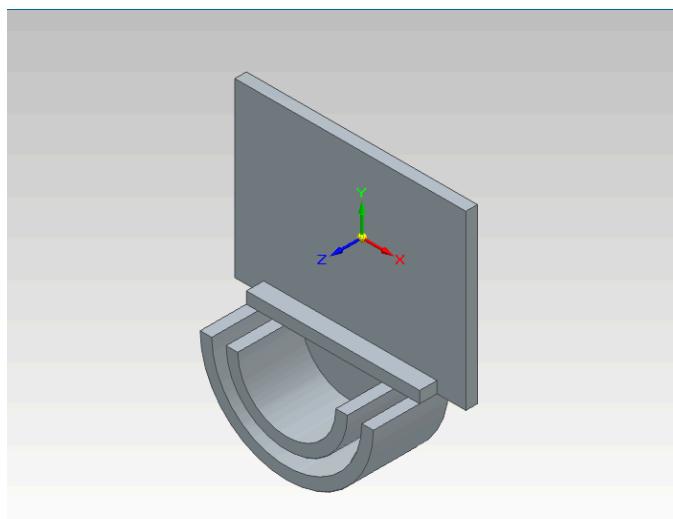


Con un primer intento de estas mejoras ya en acción (el brazo intermedio aún no estaba impreso), la imagen muestra una prueba de uno hecho con un palito de helado, la maqueta del sistema pasó a tener el siguiente aspecto en lo que denominamos la etapa intermedia del proyecto.

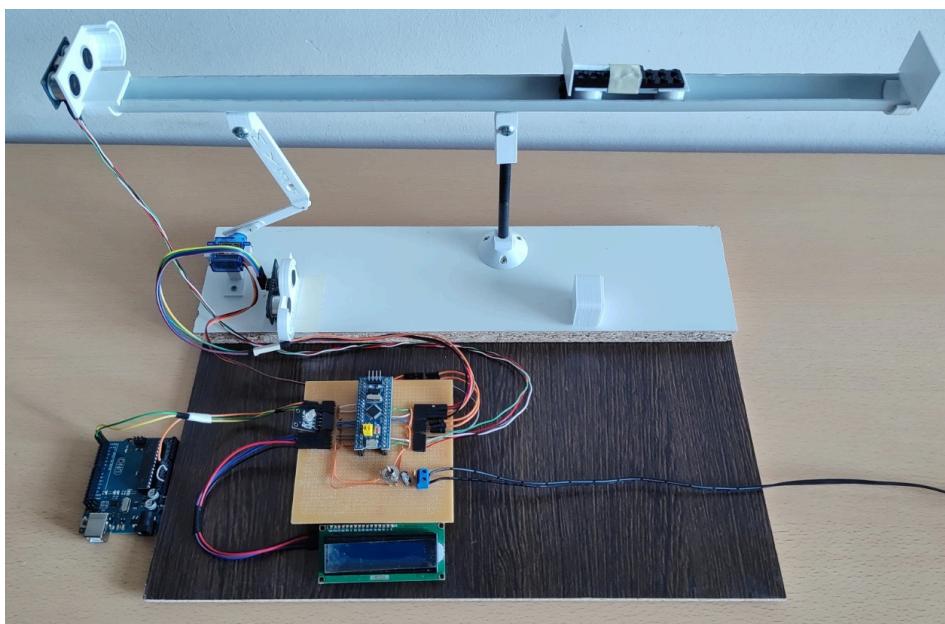




En camino al diseño final, también se diseñó la siguiente pieza. Esta pieza tiene como función principal hacer de tope para que el carrito no se caiga por el extremo del tubo, pero también sirve distancia máxima de medición para el ultrasonido superior, lo cual fue muy útil para el desarrollo de la programación en aquellos casos en los que el carro no se encontraba en el tubo o se lo sacaba repentinamente del mismo.

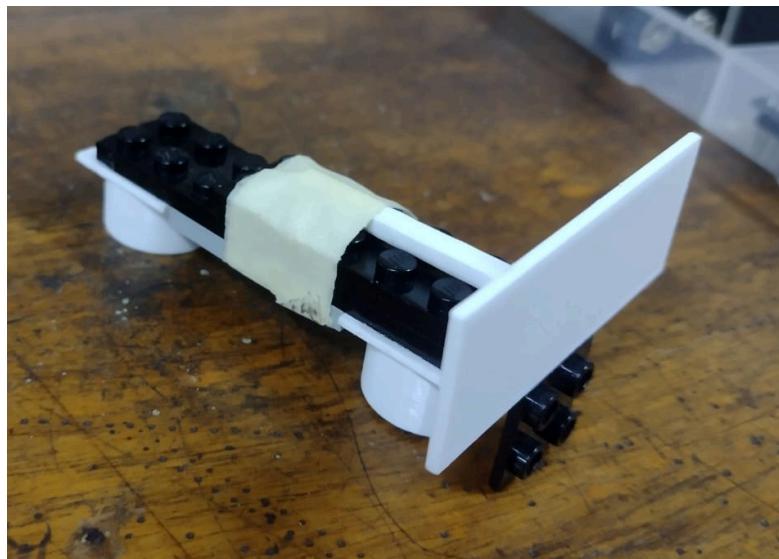
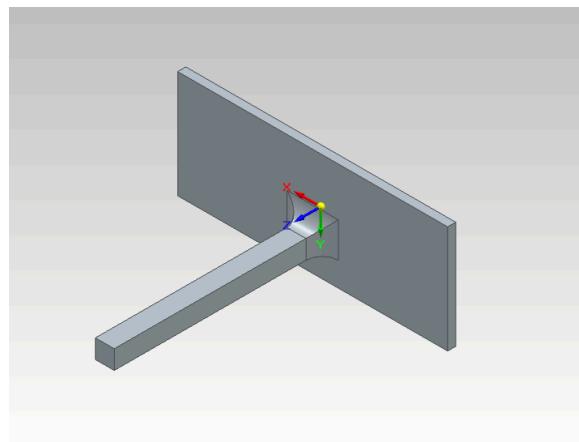
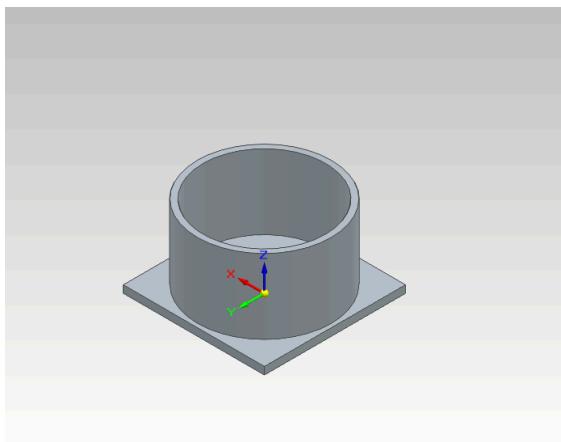


Finalmente, el diseño quedó con la forma y orden que se puede observar en la siguiente imagen, en la cual se incluye la pieza de tope final en el tubo, el sensor ultrasónico inferior con su respectivo soporte, y un pequeño bloque que se utiliza para tomar de valor objetivo en el modo correspondiente. También se puede observar cómo la parte electrónica del sistema ya se encuentra montada y soldada definitivamente sobre una placa perforada en la que se desplegó el circuito (previamente montado en placas protoboard).

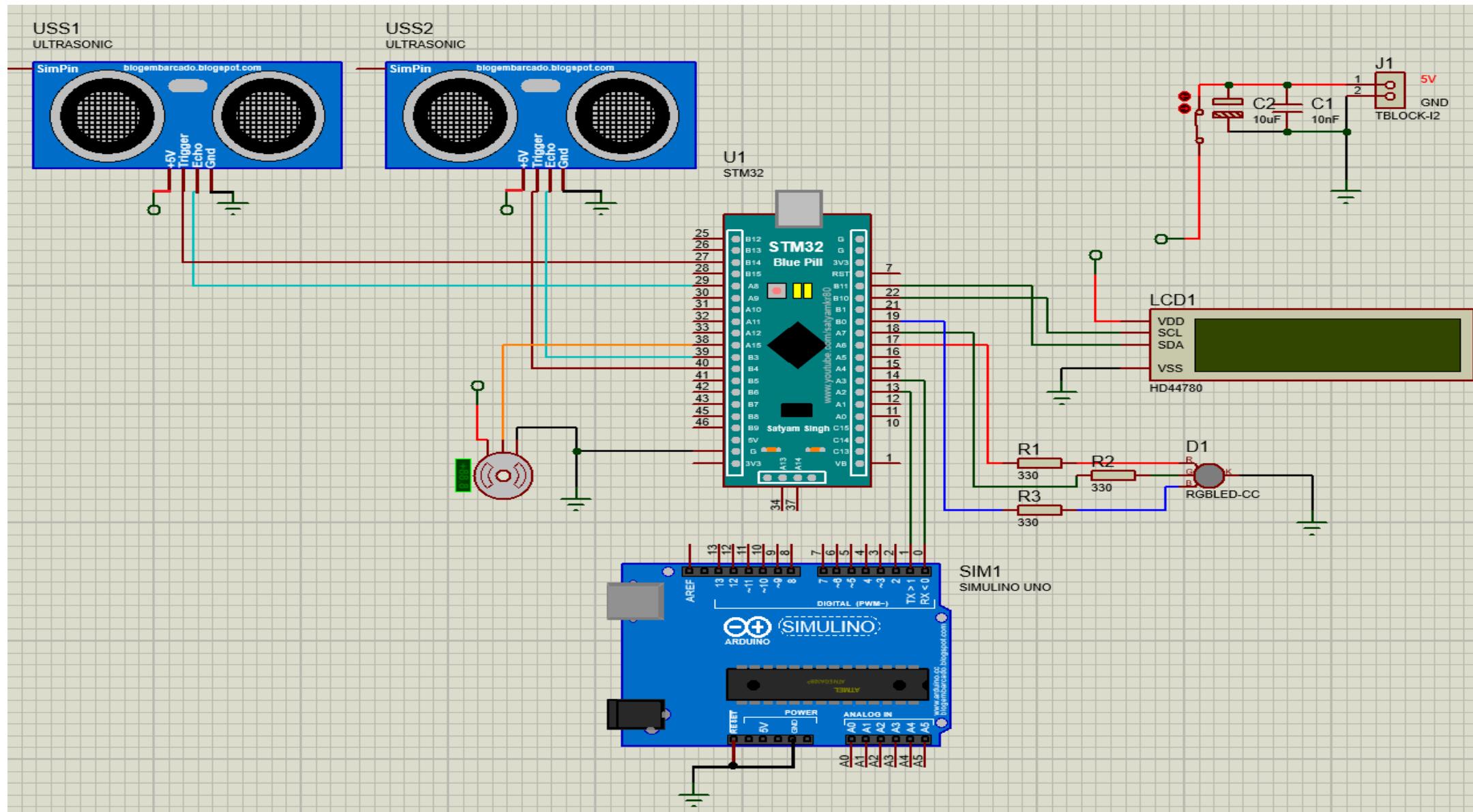




Luego de los múltiples intentos de utilizar pelotas de diferentes tamaños como móvil y no poder lograr su detección (objetos con la mejor capacidad de desplazamiento observado), se optó por conservar esta ventaja. Se hicieron dos soportes los cuales cada uno contiene una bolita permitiendo su rotación dentro de ellos y que además sirven de base para la superficie del carrito. A continuación se pueden observar los diseños de las piezas y el carrito en su versión final, con los dos soportes para las bolitas en la parte inferior y el bracito que sostiene a la superficie plana en su frente.



Por demás de la parte física y mecánica del sistema, en la siguiente imagen se observa un equivalente del circuito eléctrico que se implementó y su correspondiente configuración de pines para la comunicación entre cada periférico y la placa microcontroladora, utilizando el software de simulación Proteus.





El circuito cuenta con una fuente externa que proporciona una tensión de 5V, conectada a una bornera, para la alimentación de energía de los periféricos. A su vez, se colocaron dos capacitores, uno de 10uF y otro de 10nF, los cuales son utilizados para el filtrado de los posibles ruidos eléctricos que pudieran afectar al sistema, específicamente a los sensores ultrasónicos o al servomotor. Como se puede ver, el circuito cuenta con un switch que sólo desactiva los módulos que están conectados a la fuente externa y tanto el Arduino UNO, como la placa Blue Pill, están energizados a través de sus puertos USB (o el programador ST-Link en el caso de la Blue Pill). Tener en cuenta que se respeta una única masa compartida en entre todos los elementos del circuito.

## Resultados y especificaciones finales

En esta sección, se evaluará el grado de cumplimiento de las metas y objetivos planteados inicialmente, proporcionando un análisis detallado de los resultados obtenidos. A continuación, se presenta una lista cronológica de las metas planteadas y cumplidas durante el desarrollo del proyecto, reflejando el progreso alcanzado en cada etapa:

1. Manejo de GPIO y periféricos del microcontrolador: Implementación del control de pines y periféricos del STM32F103C8 utilizando el software STM32CubeIDE.
2. Detección de distancia con sensores ultrasónicos: Desarrollo del código necesario para la correcta medición de distancia mediante los sensores ultrasónicos, asegurando lecturas precisas.
3. Control del servomotor: Implementación del control de posición del servomotor a través de señales PWM generadas adecuadamente.
4. Maqueta del sistema y carro: Construcción de una maqueta funcional que permite la interacción y prueba del sistema en condiciones reales.
5. Comunicación y control vía UART: Establecimiento de la comunicación y control del sistema mediante comandos UART, permitiendo la interacción con el usuario.
6. Control de posicionamiento del carro: Integración de un controlador PID para lograr el posicionamiento preciso del carro sobre la barra, ajustando la inclinación de la misma para alcanzar la posición objetivo deseada.
7. Optimización del controlador PID: Ajuste y calibración de las constantes del controlador PID (Proporcional, Integral y Derivativo) para mejorar la precisión en la corrección de la posición del carro.
8. Interfaz de usuario intuitiva: Desarrollo de una interfaz de usuario accesible mediante comandos UART y visualización en un panel LCD, facilitando la interacción y operación del sistema.
9. Funcionamiento en múltiples modos: Implementación de diversos modos de operación (Energizado, Automático, Manual y Testeo), cada uno con funcionalidades específicas para adaptarse a diferentes necesidades del usuario.



#### Especificaciones técnicas del sistema:

- Placa microcontroladora Blue Pill
  - Consumo de corriente en modo Energizado: 26.5mA
  - Consumo de corriente en modo Automático: 22-25mA
  - Consumo de corriente en modo Testeo: 28.5mA
- Servomotor
  - Paso mínimo: 1°
  - Rango del dispositivo: 0 - 180°
  - Rango de uso en modo de funcionamiento A/M: 111 - 180°
  - Rango de uso en modo de funcionamiento Testeo: 60 - 180°
  - Consumo de corriente en reposo: 2mA
  - Consumo de corriente máxima en funcionamiento: 270mA
  - Periodo de ciclo de trabajo: 20ms
  - Rango de ciclo de trabajo PWM: 2.8% - 12.8% (ver anexo)
- Sensor ultrasónico
  - Rango del dispositivo: 2 - 400cm
  - Rango de uso sensor inferior: 2 - 30cm
  - Rango de uso sensor superior: 2 - 51cm
  - Precisión en el uso: 1cm
  - Consumo de corriente en reposo: 2.5mA
  - Consumo de corriente máxima en funcionamiento: 3.4mA
  - Periodo de lecturas: 120 ms (ver anexo)
- LED RGB
  - Colores utilizados: Rojo (100%), Verde (100%), Azul (100%)
  - Consumo de corriente aproximado: 2 - 4mA
- Pantalla LCD + Módulo I2C
  - Consumo de corriente en funcionamiento: 32.6mA
- Controlador PID
  - Tiempo de respuesta: Menos de 1ms

## Ensayo de ingeniería de producto

Para llevar a cabo una implementación del producto pero a un nivel comercial, una aplicación posible para este proyecto en particular, puede ser la del desarrollo en el ámbito educativo. Este proyecto es un ejemplo básico, pero claramente representativo, de cómo responde el control PID en la búsqueda de posiciones objetivos que puedan cambiar durante la misma búsqueda de la posición y con constantes de control también posibles de modificar.

En este escenario, se elegiría hacer principalmente los siguientes cambios al sistema (dados a partir de la experiencia obtenida en el desarrollo actual) para potenciar su



confiabilidad. Como cambios principales, se comienza por utilizar un sensor distinto al ultrasónico tipo HC-SR04, debido a que este demostró ser muy dependiente de la forma que tiene la superficie a detectar. En vez de este, se optaría por utilizar un sensor de tipo infrarrojo por ejemplo, el cual tiene un diseño físico distinto, especialmente en lo que respecta a la posición entre el elemento emisor y el receptor, lo que permitiría una flexibilización importante a la hora de elegir el objeto a detectar. Este objeto podría idealmente ser esférico y de esta manera hacer posible observar con mucha mayor claridad cómo es que el sistema aporta cada tipo de componente del control en la respuesta que otorga.

Componente	Precio (USD)
Blue Pill + ST-Link V2	10 USD
Conversor USB a TTL CH340	3 USD
Sensor ultrasónico HC-SR04	2.5 USD
Sensor infrarrojo GP2Y0E03	13 USD
Servomotor SG90	2 USD
LCD 1602 + módulo I2C	8 USD
Módulo LED RGB KY-016	1.5 USD
Filamento e impresión de piezas en 3D	8 USD
Placa perforada, switch, bornera y cables	6 USD (aproximado)
Base de madera, tornillos, tuercas y tubo	5 USD (aproximado)
Fuente de alimentación de 5V	8 USD

Considerando los materiales utilizados en este proyecto, se calcula un coste de aproximadamente 56.5 USD (considerando dos sensores ultrasónicos HC-SR04), solo en materiales, para el desarrollo realizado.

Para el paso a un producto comercial, se estima un coste de 67 USD (considerando la compra de un sensor ultrasónico y uno infrarrojo), solo en materiales.

Actividad	Horas invertidas
Investigación	20
Diseño de piezas en 3D	8
Impresión de las piezas en 3D	7
Programación y pruebas de funcionamiento	120
<b>TOTAL</b>	<b>155</b>



Una alternativa de producto comercial con una implementación necesariamente más sofisticada, podría ser la de un dron. Si bien no aparecen el tubo y el carrito, la idea en el aspecto del control es la misma. Los drones logran un vuelo estable y preciso con la ayuda del control PID, el cual, similarmente a nuestro proyecto, ajusta variables en motores. Para una posible selección de componentes, se podría utilizar un microcontrolador PixHawk o ESP32, motores brushless, controladores de velocidad electrónicos y acelerómetros.

## Conclusiones

### Reflexión sobre el Desarrollo Realizado

El proyecto de control del carrito en el tubo utilizando un controlador PID ha sido una experiencia enriquecedora que ha permitido explorar tanto las capacidades como las limitaciones de los sistemas de control automático en un entorno físico específico. A través de la implementación y ajuste del sistema PID, se han alcanzado varios objetivos clave, pero también se han identificado áreas para mejora y optimización.

### Logros y Conclusiones

Uno de los logros más significativos del proyecto ha sido la capacidad del sistema para mantener el carrito en la posición deseada dentro del tubo. Gracias a la correcta implementación del controlador PID, se ha logrado un control estable y preciso del carrito, incluso en presencia de algunas perturbaciones leves y variaciones en el entorno. Esto demuestra la efectividad del PID para gestionar sistemas dinámicos y responder adecuadamente a los cambios.

Se ha conseguido filtrar buena parte de los ruidos que pudieran haber en los sensores, gracias a haber identificado el por qué se producían y qué es lo que los producían. El modo debug con ST-Link fue una herramienta clave en este proceso.

La interfaz de usuario creada es intuitiva y simple. Es también muy interesante el poder graficar el estado del sistema en tiempo real usando el IDE de Arduino. Gracias al DMA, este proceso no afecta en términos de tiempo al controlador PID, logrando que este responda más rápidamente.

### Problemas Encontrados

Durante el desarrollo del proyecto, se enfrentaron varios problemas técnicos que impactaron en el rendimiento del sistema. Uno de los principales desafíos fue el ajuste del controlador PID. Aunque se logró una buena precisión en el posicionamiento del carrito, se observó una respuesta lenta en ciertas condiciones, afectando la estabilidad general del sistema. Además, las perturbaciones o cambios muy fuertes provocan que el carrito salga disparado, lo que subraya la dificultad de ajustar los parámetros en función de las condiciones cambiantes del entorno.



Otro problema significativo fue la fricción en el carrito, que complicó aún más el ajuste del PID. Además, los sensores de posición, aunque efectivos en general, presentaron limitaciones en cuanto a su precisión y velocidad de respuesta. Estos problemas demostraron la necesidad de ajustar continuamente los parámetros del sistema para mantener un rendimiento óptimo en condiciones variables.

#### Aprendizajes y Mejoras

A partir de estos desafíos, se han aprendido valiosas lecciones sobre la importancia de un ajuste fino del controlador PID y la necesidad de seleccionar componentes de alta precisión. La experiencia ha mostrado que un análisis más profundo y un ajuste más sofisticado del PID podrían mejorar significativamente la respuesta del sistema. Además, la selección de sensores con mejor rendimiento y la optimización del diseño del prototipo son áreas que ofrecen oportunidades significativas para mejorar el proyecto.

#### Perspectivas de Mejora

Para mejorar el prototipo, se recomienda explorar técnicas avanzadas de sintonización del PID, como el método Zieger-Nichols, el uso de otros algoritmos de optimización o el ajuste adaptativo en tiempo real. Asimismo, la incorporación de sensores de mayor precisión podría incrementar la eficacia del sistema. Un ejemplo puede ser un sensor infrarrojo, que funciona similarmente a los ultrasónicos, pero utilizando un haz de luz infrarroja, lo que lo hace más rápido y que permitiría la utilización de objetos como pelotas, en lugar de un carrito, reduciendo en gran magnitud también el problema de la fricción. También podría ser posible aumentar aún más la velocidad del sistema, enviando mensajes a una interfaz gráfica con la que se relacione el usuario.



## Referencias

STMicroelectronics. (s.f.). *Mainstream Performance line, Arm Cortex-M3 MCU with 64 Kbytes of Flash memory, 72 MHz CPU, motor control, USB and CAN.*  
<https://www.st.com/en/microcontrollers-micropocessors/stm32f103c8.html>

Alldatasheet. (s.f.). *HCSR04 Datasheet (PDF) - List of Unclassified Manufacturers.*  
<https://pdf1.alldatasheet.com/datasheet-pdf/view/1132204/ETC2/HCSR04.html>

Alldatasheet. (s.f.). *SG90 Datasheet (PDF) - List of Unclassified Manufacturers.*  
<https://pdf1.alldatasheet.com/datasheet-pdf/view/1572383/ETC/SG90.html>

Components101. (2021, mayo 30). *16x2 LCD Module.*  
<https://components101.com/displays/16x2-lcd-pinout-datasheet>

Alldatasheet. (s.f.). *HD44780 Datasheet (PDF) - Hitachi Semiconductor*  
<https://pdf1.alldatasheet.com/datasheet-pdf/view/63673/HITACHI/HD44780.html>

eziya. (2019, marzo 9). *STM32\_HAL\_I2C\_HD44780* [Repositorio de GitHub].  
[https://github.com/eziya/STM32\\_HAL\\_I2C\\_HD44780](https://github.com/eziya/STM32_HAL_I2C_HD44780)

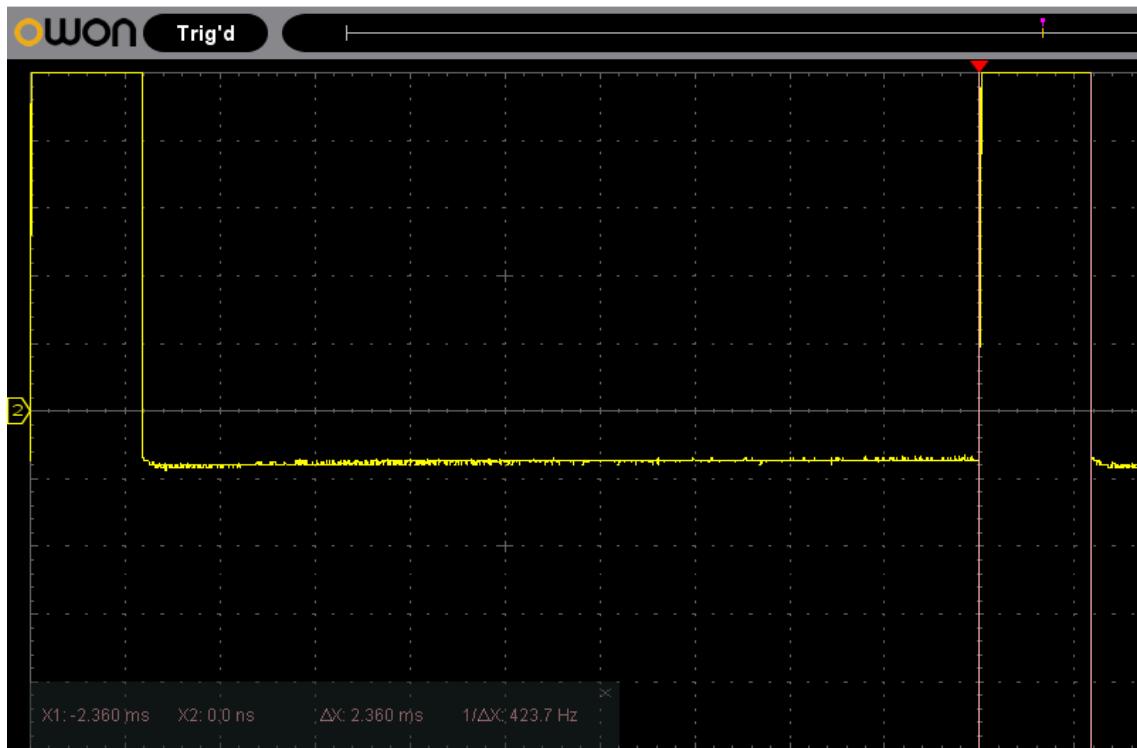
Semiconductors. (s.f.). *KY-016 Joy-IT RGB 5mm LED module Datasheet.*  
<https://semiconductors.es/datasheet-pdf/1402027/KY-016.html>

Arduino. (s.f.). *Arduino Uno Rev3 (PDF).* En Arduino Docs.  
<https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf>



## Anexo

Señal de PWM recibida por el servomotor. Capturas tomadas de osciloscopio.  
PWM para consigna de  $160^\circ$  (2ms/div).  $\Delta x: T_{ON}$

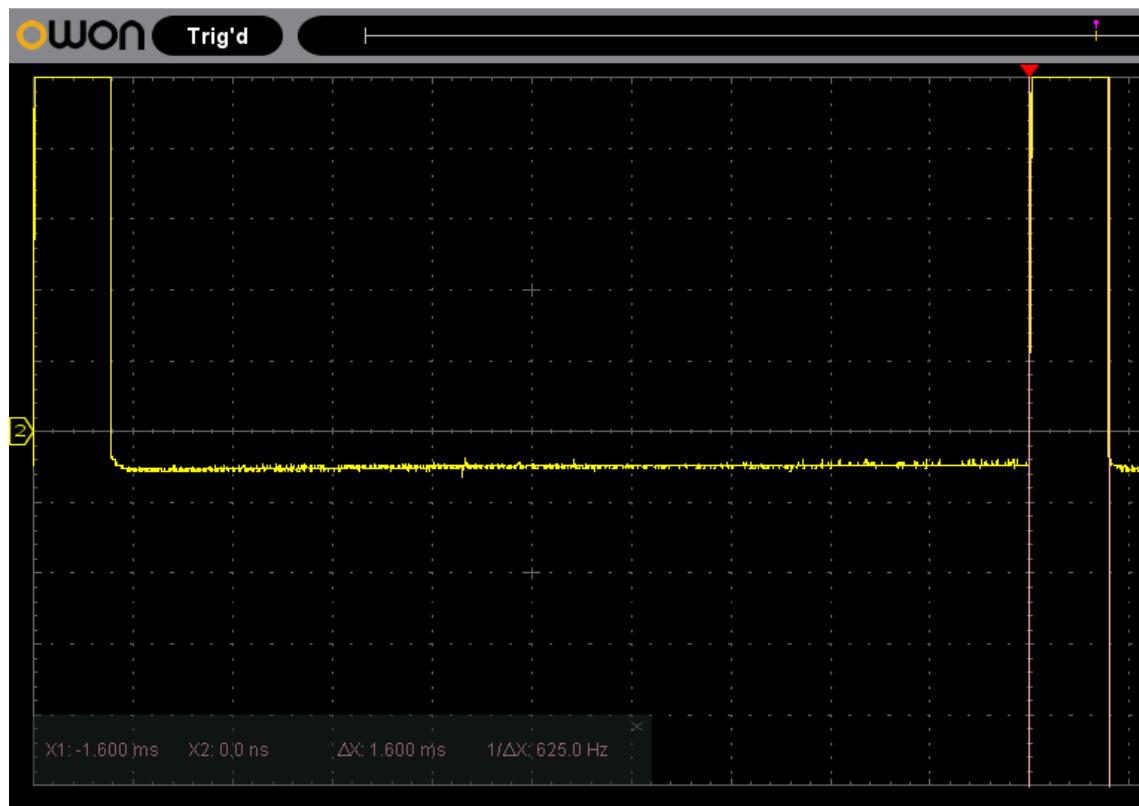


PWM para consigna de  $180^\circ$  (2ms/div).  $\Delta x: T_{ON}$

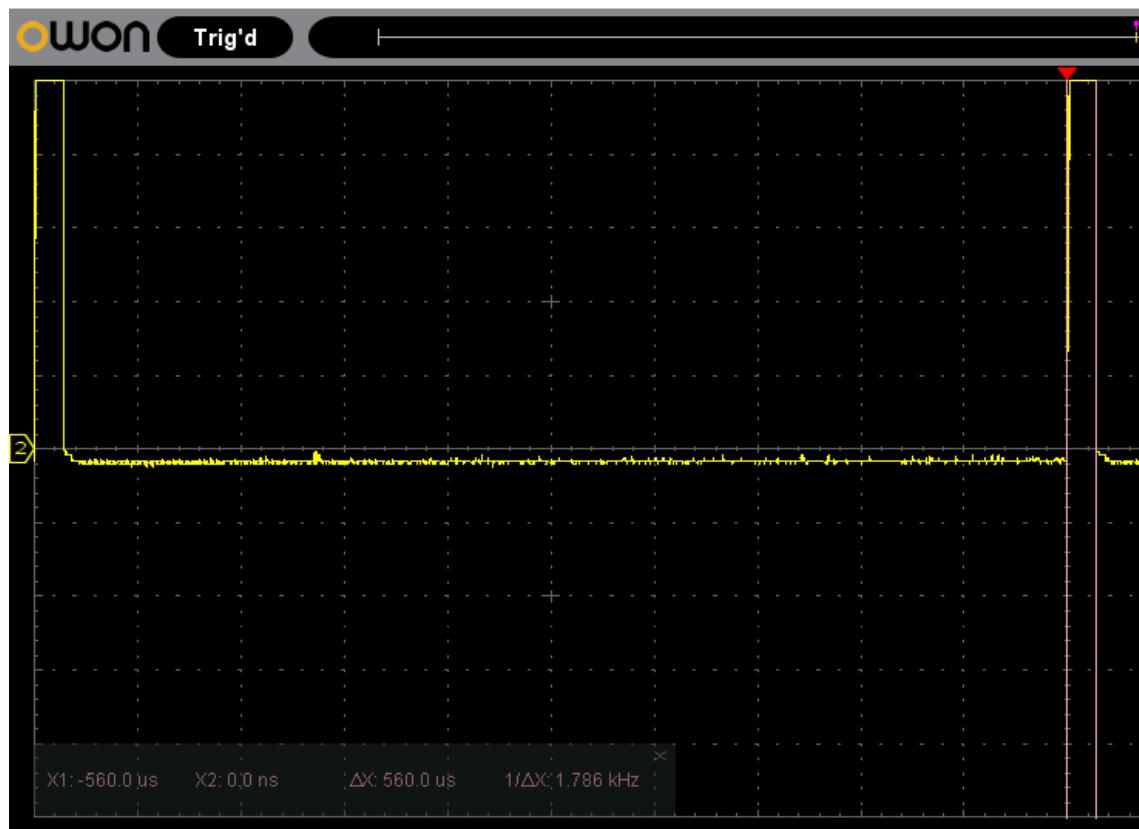




PWM para consigna de 90° (2ms/div). Δx: T\_ON

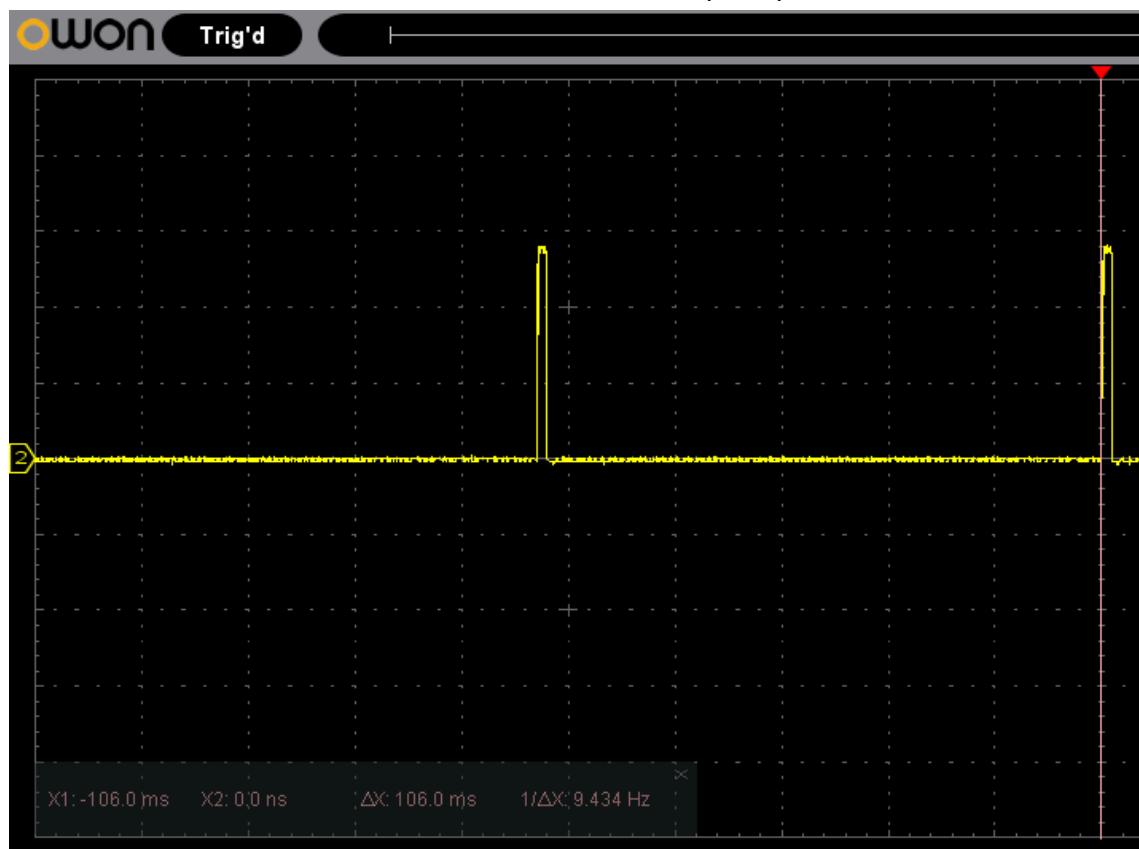


PWM para consigna de 0 grados (2ms/div). Δx: T\_ON





Periodo no controlado entre dos iteraciones del bucle principal. Modo automático.



Periodo controlado entre dos iteraciones del bucle principal. Modo automático.

