Laboratorio # 4 Laboratorio control de motor DC y ISR

S. Herran.

Universidad Sergio Arboleda
Escuela de ciencias exactas e ingeniería
Sistemas Embebidos
5 de Septiembre del 2024

1. Problema

Utilizando los conocimientos adquiridos en clase y los componentes listados a continuación, escribir un programa para el microcontrolador para medir la velocidad de un Motor DC y sus tiempos de respuesta usando el control IR para su manejo.

Procedimiento:

1. Montar el sistema del **Laboratorio#3** (control IR) para recibir correctamente los comandos del control.
2. Conectar el sistema en donde el driver del motor (Puente H) sea controlado por dos salidas digitales GPIO con las respectivas conexiones del módulo (Ver diagrama conexión motor).
3. Conectar las salidas de los sensores de herradura a entradas GPIO.
4. Configurar el microcontrolador para realizar un "arrancar" y "detener" del motor haciendo uso de una función o método, dicho método se debe operar desde un botón del control remoto y en la pantalla debe aparecer el mensaje de "Encendido" o "Apagado" según sea el caso.
5. Agregar la funcionalidad de escoger el sentido de giro deseado haciendo uso de dos botones adicionales del control remoto.
6. Desarrollar la funcionalidad para realizar una medición de R.P.M. (revoluciones por minuto) utilizando el sensor de IR de cruce herradura. Este valor se debe visualizar en la pantalla en tiempo real (mínimo 0.5s de refresco).
7. Desarrollar la funcionalidad de detección de sentido de giro. El sistema debe visualizar en pantalla si está girando a la izquierda o a la derecha. **Esto debe ser detectado y no visualizado por el comando enviado, ya que la rueda se podría girar con la mano y el sistema debe visualizar tanto las RPM como el sentido de giro.**
8. Visualizar en pantalla el valor de máximas RPM y actualizarlo en tiempo real.
9. Calcular y visualizar en pantalla el tiempo transcurrido en milisegundos entre el comando de "arrancar" hasta cuando alcance la velocidad máxima.
10. Hacer pruebas y determinar el TIEMPO DE RESPUESTA DEL MOTOR.

11. Realizar el correspondiente informe con mínimo 5 conclusiones reales basadas en la experiencia de la realización del laboratorio.

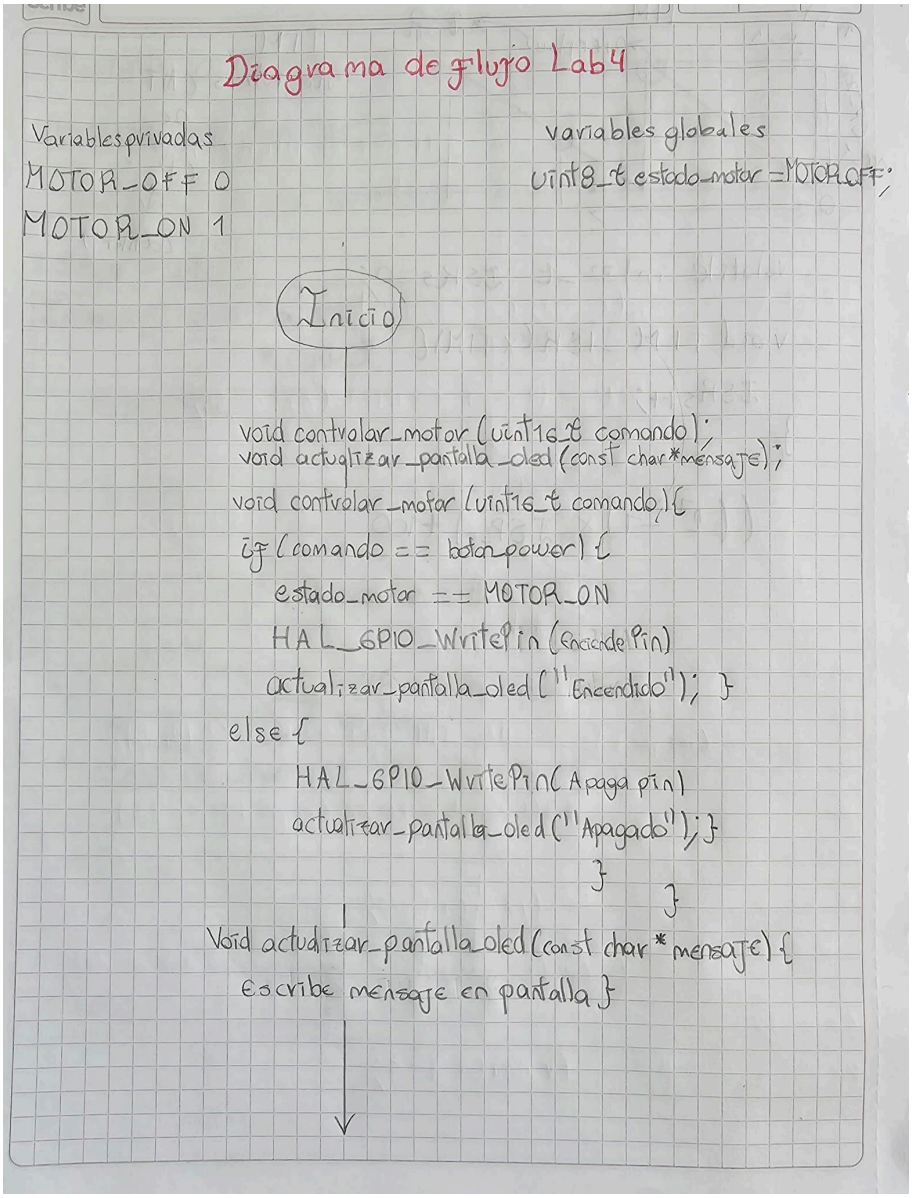
2. Materiales

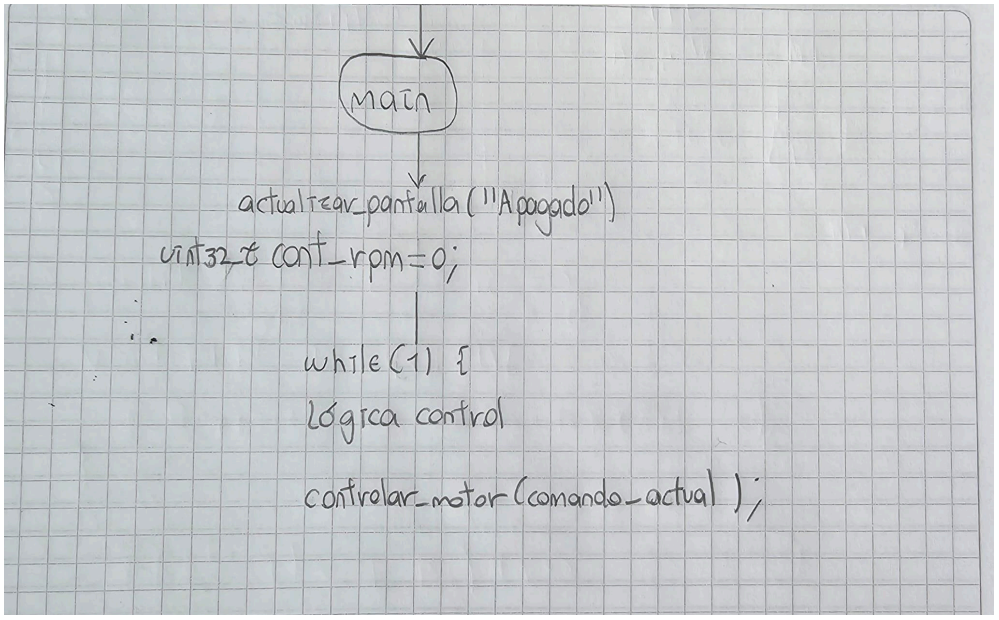
- ☐ **Microcontrolador STM32F411**
- ☐ **Motor DC**
- ☐ **2 sensores IR de herradura**
- ☐ **Pantalla OLED**
- ☐ **Rueda con disco y dos ranuras**
- ☐ **Cables, resistencias, transistores**
- ☐ **Sensor IR**
- ☐ **Control remoto**
- ☐ **Puente H (motor driver) TB6612FNG**

3. Resolución del problema

Primero se une el código de la pantalla oled con el del control remoto, luego se crea una función que controle el motor y que se encienda/apague con el botón power, luego con otros dos botones se pueda invertir el giro o que gire en el sentido de las manecillas del reloj. Luego en la pantalla se tiene que mostrar los RPM medidos por el sensor de herradura y si se está girando a la derecha o izquierda. Visualizar en pantalla el tiempo transcurrido entre arrancar y velocidad máxima. Determinar el tiempo de respuesta del motor.

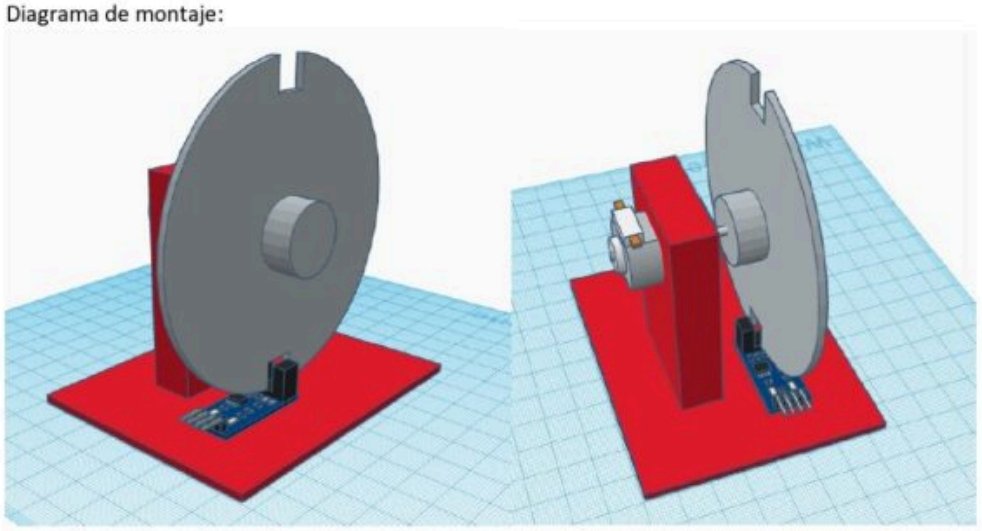
4. Diagrama de flujo



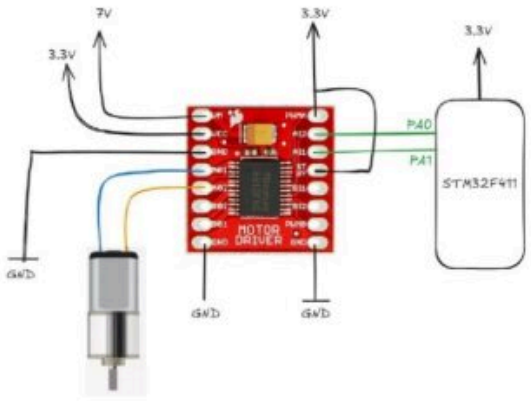


5. Esquemático /montaje

- Diagrama de montaje:



- Diagrama Driver Motor



6. **Problemas al desarrollar el laboratorio**

   - **El problema más grave de este laboratorio fueron los componentes, por problemas internos del grupo se desperdicio bastante tiempo donde no se pudo avanzar el laboratorio. Esto perjudicó a la entrega del laboratorio, sin añadir problemas adicionales por los cuales se perjudicó la entrega**

7. **Código principal**

```
/* USER CODE BEGIN Header */
/**

******************************************************************************
  * @file           : main.c
  * @brief          : Main program body

******************************************************************************
  * @attention
  *
  * Copyright (c) 2024 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *

******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "ssd1306.h"
#include "ssd1306_fonts.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
```

```
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define MOTOR_ON 1
#define MOTOR_OFF 0

volatile uint32_t rpm_actual = 0;
volatile uint32_t rpm_anterior = 0;
volatile uint32_t contador_pulsos = 0;  // Contador de pulsos
volatile uint32_t tiempo_ultimo_pulso = 0; // Tiempo del último pulso
volatile uint32_t tiempo_pulsos = 0;     // Tiempo entre pulsos en milisegundos
uint32_t rpm = 0;                    // Variable para almacenar rpm
uint32_t periodo = 1000;              // Intervalo de medición en milisegundos
uint32_t tiempo_inicio_motor = 0;       // Tiempo cuando se enciende el motor
uint32_t tiempo_maxima_velocidad = 0;   // Tiempo cuando se alcanza la velocidad máxima
uint32_t tiempo_transcurrido_max_vel = 0;  // Tiempo transcurrido para alcanzar la
velocidad máxima
uint8_t motor_estado = MOTOR_OFF;      // Estado del motor
uint8_t direccion_giro = 0;           // Dirección de giro, 0: Derecha, 1: Izquierda
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
I2C_HandleTypeDef hi2c1;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
```

```
static void MX_TIM2_Init(void);
static void MX_TIM3_Init(void);
/* USER CODE BEGIN PFP */
void verificar_y_cambiar_estado(uint16_t valor_trama);
void controlar_motor(uint16_t comando);
void actualizar_rpm_pantalla(uint8_t estado_motor, uint8_t direccion_giro);
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
uint16_t trama[32];
uint16_t tiempo;
uint16_t tiempo_bits;
uint16_t comando_actual = 0;
// Declarar el comando de los botones del 1 al 8
uint16_t boton_1 = 0x88;
uint16_t boton_2 = 0x48;
uint16_t boton_3 = 0xC8;
uint16_t boton_4 = 0x28;
uint16_t boton_5 = 0xA8;
uint16_t boton_6 = 0x68;
uint16_t boton_7 = 0xE8;
uint16_t boton_8 = 0x18;
uint16_t boton_power = 0x10; // Comando para el botón de encendido/apagado del motor
uint16_t boton_der = 0x60; // flecha derecha del control
uint16_t boton_izq = 0xE0; // flecha izquierda del control
uint8_t estado_motor = MOTOR_OFF;



int i;
int j;

// Función de interrupción que se activa cuando los sensores detectan una vuelta
/*void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
   if (GPIO_Pin == GPIO_PIN_12 ) {  // pines de los sensores

      //static __HAL_TIM_SET_COUNTER(&htim3, 0);

      if(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_13) == 1){
                        // sentido de giro 1
      } else{
```

```
            //sentido de giro = 0;
        }
        contador_vueltas++;  // suma 1 vuelta
    }
}
*/
/* Función de interrupción que se activa cuando los sensores detectan un pulso */
/*void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
    if (GPIO_Pin == GPIO_PIN_12 || GPIO_Pin == GPIO_PIN_13) {  // pines de los sensores
        uint32_t tiempo_actual = __HAL_TIM_GET_COUNTER(&htim3);

        // Medir el tiempo transcurrido desde el último pulso
        if (tiempo_ultimo_pulso != 0) {
            tiempo_pulsos = tiempo_actual - tiempo_ultimo_pulso;
        }

        // Actualiza el tiempo del último pulso
        tiempo_ultimo_pulso = tiempo_actual;

        contador_pulsos++;
    }
    actualizar_rpm_pantalla(motor_estado, direccion_giro);
}
*/

/* Interrupción del temporizador para calcular las RPM cada periodo */
/*void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM3) {
        // Calcular RPM
        uint32_t revoluciones = contador_pulsos / 2;  // Cada 2 pulsos es 1 revolución
        rpm = (revoluciones * 60000) / periodo;  // RPM = (revoluciones / tiempo_en_minutos)

        // Resetear el contador
        contador_pulsos = 0;

        // Verificar si el motor está encendido y calcular el tiempo transcurrido
        if (motor_estado == MOTOR_ON) {

            if (rpm >= velocidad_maxima_umbral) {

                if (tiempo_maxima_velocidad == 0) {
```

```
            // Almacena el tiempo cuando se alcanza la velocidad máxima
            tiempo_maxima_velocidad = __HAL_TIM_GET_COUNTER(&htim3);

          }
        }

        if (tiempo_maxima_velocidad != 0) {

          // Calcula el tiempo transcurrido desde el arranque hasta alcanzar la velocidad
máxima
          tiempo_transcurrido_max_vel = tiempo_maxima_velocidad -
tiempo_inicio_motor;
        }
      }

      // Actualizar la pantalla OLED
      actualizar_rpm_pantalla(motor_estado, direccion_giro);
    }
}
*/

void actualizar_rpm_pantalla(uint8_t estado_motor, uint8_t direccion_giro) {
    char mensaje[64];

    // Limpiar pantalla
    ssd1306_Fill(Black);

    // Mostrar las RPM en la primera línea
    sprintf(mensaje, "RPM: %lu", rpm);
    ssd1306_SetCursor(2, 0);  // Establecer el cursor en la parte superior
    ssd1306_WriteString(mensaje, Font_11x18, White);

    // Mostrar el estado del motor en la segunda línea
    ssd1306_SetCursor(2, 20);  // Mover el cursor hacia abajo
    if (estado_motor == MOTOR_ON) {
      ssd1306_WriteString("ON ⚡", Font_11x18, White);  // Mostrar el símbolo de
encendido
    } else {
      ssd1306_WriteString("OFF", Font_11x18, White);   // Motor apagado
    }
```

```
    // Mostrar la dirección de giro en la tercera línea
    ssd1306_SetCursor(2, 40);
    if (direccion_giro == 0) {
        ssd1306_WriteString("Der. >", Font_11x18, White);  // Derecha
    } else {
        ssd1306_WriteString("Izq. <", Font_11x18, White);  // Izquierda
    }

    // Mostrar el tiempo transcurrido para alcanzar la velocidad máxima
    /*
    ssd1306_SetCursor(2, 60);
    sprintf(mensaje, "T. Max Vel: %lu ms", tiempo_transcurrido_max_vel);
    ssd1306_WriteString(mensaje, Font_11x18, White);
      */

    // Actualizar la pantalla OLED
    ssd1306_UpdateScreen();
}

void verificar_y_cambiar_estado(uint16_t valor_trama) {
    // Apagar todos los LEDs
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);

    // Encender el LED correspondiente
    switch (valor_trama) {
        case 0x88:
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET); // Botón 1 - Pin A1
            break;
        case 0x48:
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_SET); // Botón 2 - Pin A2
            break;
        case 0xC8:
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_SET); // Botón 3 - Pin A3
            break;
        case 0x28:
```

```
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET); // Botón 4 - Pin A4
            break;
        case 0xA8:
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET); // Botón 5 - Pin A5
            break;
        case 0x68:
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET); // Botón 6 - Pin A6
            break;
        case 0xE8:
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_SET); // Botón 7 - Pin A7
            break;
        case 0x18:
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET); // Botón 8 - Pin A8
            break;
        default:
            // No coincide con ningún botón
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, trama[17] ? GPIO_PIN_SET :
GPIO_PIN_RESET);
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, trama[18] ? GPIO_PIN_SET :
GPIO_PIN_RESET);
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, trama[19] ? GPIO_PIN_SET :
GPIO_PIN_RESET);
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, trama[20] ? GPIO_PIN_SET :
GPIO_PIN_RESET);
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, trama[21] ? GPIO_PIN_SET :
GPIO_PIN_RESET);
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, trama[22] ? GPIO_PIN_SET :
GPIO_PIN_RESET);
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, trama[23] ? GPIO_PIN_SET :
GPIO_PIN_RESET);
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, trama[24] ? GPIO_PIN_SET :
GPIO_PIN_RESET);
            break;
    }
}

// Función para controlar el motor y actualizar la pantalla OLED
void controlar_motor(uint16_t comando) {
    // Cambia el estado del motor solo si se presiona el botón de encendido
    if (comando == boton_power) {
        // Cambia el estado del motor
        estado_motor = (estado_motor == MOTOR_OFF) ? MOTOR_ON : MOTOR_OFF;
```

```
    }

    // Control del motor basado en el estado actual
    if (estado_motor == MOTOR_ON) {
        // Encender el motor
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_SET);  // Arranca el motor

        // Configurar dirección del motor
        if (comando == boton_der) {
            // Giro a la derecha
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_RESET); // Configura
dirección a derecha
            direccion_giro = 0;
        } else if (comando == boton_izq) {
            // Giro a la izquierda
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_RESET);   // Configura
dirección a izquierda
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_SET);
            direccion_giro = 1;
        }
    } else {
        // Apagar el motor
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_RESET);  // Apaga el
motor
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_RESET);  //
    }

    // Actualizar la pantalla OLED
    actualizar_rpm_pantalla(estado_motor, direccion_giro);
}



// Interrupción del temporizador para actualizar las RPM cada 0.5 segundos
/*void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM3) {
       // Calcular las rpm cada 0.5 segundos
            rpm = (contador_vueltas * 120);  //
            contador_vueltas = 0;  // reiniciar el contador para el siguiente período
             actualizar_rpm_pantalla(estado_motor, direccion_giro);
    }
}
```

```
*/
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
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_I2C1_Init();
  MX_TIM2_Init();
  MX_TIM3_Init();
  ssd1306_Init();
  /* USER CODE BEGIN 2 */
  //HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12 | GPIO_PIN_13, GPIO_PIN_RESET); //
configura los pines de los sensores de
  // herrradura para contar la vuelta

  /* USER CODE END 2 */
```

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  // se inician los temporizadores
  HAL_TIM_Base_Start(&htim2);
  HAL_TIM_Base_Start_IT(&htim3);


  tiempo_inicio_motor = 0;
  tiempo_maxima_velocidad = 0;
  tiempo_transcurrido_max_vel = 0;
  actualizar_rpm_pantalla(estado_motor,direccion_giro);
  while (1)
  {

      /* Empieza el código, se hace un bucle que espere cuando el PIN 9 del  puerto B cambie de estado*/
      while(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_9)== 1);
      /*Se agrega un delay para el rebote y capturar los bits correctamente*/
      HAL_Delay(200);
      /* Se pone el contador del TIM2 en 0 para que empiece a contar el tiempo  */
      __HAL_TIM_SET_COUNTER(&htim2, 0);
      /* Espera a que el PIN 9 deje de estar en 0 y luego cuando vuelva a 1 espere de nuevo*/
      while(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_9)== 0);
      while(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_9)== 1);
      /* Se crea una variable tiempo, esta obtiene el tiempo del TIM2*/
      tiempo = __HAL_TIM_GET_COUNTER(&htim2);
      /* Se crea un bucle for que inicia en 0 y acaba a las 32 veces de recorrido, va sumando 1 cada vez
       * que acaba el bucle*/
            for(i=0 ;i<32 ;i++)
            {

                  /*Se pone el TIM2 en 0 para volver a contar*/
                  __HAL_TIM_SET_COUNTER(&htim2, 0);
                  /*Se hace un bucle que mientras el pin este en 0 espere y luego que cuando este en 1 espere*/
                  while(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_9)== 0);
                  while(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_9)== 1);
                  /*Se crea una variable tiempo_bits, que obtiene el valor del contador TIM2 */
                  tiempo_bits = __HAL_TIM_GET_COUNTER(&htim2);
```

```
                    /* Se crea una condición, si el tiempo_bits esta entre 100 y 120 el arreglo
trama en el indice
                     * correspondiente sea igual a 0, esto se hace para que guarde el valor
lógico de acuerdo con
                     * el protocolo NIC*/
                    /*100 equivale a 1ms y 120 a 1.2ms*/
                    if (tiempo_bits >=100 && tiempo_bits<=120 )
                    {
                            trama[i]=0;
                    }
                    /*De la misma forma se hace una condición pero esta vez con las
restricciones para que almacene
                     * un 1 lógico*/
                    if (tiempo_bits >=200 && tiempo_bits<=240 )
                    {
                            trama[i]=1;
                    }


            }

            // Convertir los bits de trama[17] a trama[24] en un valor hexadecimal
                    comando_actual = 0;
                    for (i = 0; i < 8; i++) {
                            comando_actual |= (trama[17 + i] << (7 - i)); // Combinando los
bits en un solo valor
                    }

                  if (motor_estado == MOTOR_ON) {
                      if (tiempo_inicio_motor == 0) {
                         // Registrar el tiempo cuando se enciende el motor
                         tiempo_inicio_motor = __HAL_TIM_GET_COUNTER(&htim3);
                      }
                     actualizar_rpm_pantalla(motor_estado, direccion_giro);
                   }
                    // Verificar y cambiar el estado de los pines según el valor de
comando_actual
                    verificar_y_cambiar_estado(comando_actual);
                    controlar_motor(comando_actual);

   /* USER CODE END WHILE */
```

```
  /* USER CODE BEGIN 3 */

 }
 /* USER CODE END 3 */
}

/**
 * @brief System Clock Configuration
 * @retval None
 */
void SystemClock_Config(void)
{
 RCC_OscInitTypeDef RCC_OscInitStruct = {0};
 RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

 /** Configure the main internal regulator output voltage
 */
 __HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE
1);

 /** Initializes the RCC Oscillators according to the specified parameters
 * in the RCC_OscInitTypeDef structure.
 */
 RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
 RCC_OscInitStruct.HSIState = RCC_HSI_ON;
 RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
 RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
 RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
 RCC_OscInitStruct.PLL.PLLM = 8;
 RCC_OscInitStruct.PLL.PLLN = 100;
 RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
 RCC_OscInitStruct.PLL.PLLQ = 4;
 if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
 {
  Error_Handler();
 }

 /** Initializes the CPU, AHB and APB buses clocks
 */
```

```
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_3) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
```

```
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */

}

/**
  * @brief TIM2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM2_Init(void)
{

  /* USER CODE BEGIN TIM2_Init 0 */

  /* USER CODE END TIM2_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_IC_InitTypeDef sConfigIC = {0};

  /* USER CODE BEGIN TIM2_Init 1 */

  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 1000-1;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 4294967295;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_IC_Init(&htim2) != HAL_OK)
  {
```

```
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
  sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
  sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
  sConfigIC.ICFilter = 0;
  if (HAL_TIM_IC_ConfigChannel(&htim2, &sConfigIC, TIM_CHANNEL_1) !=
HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_Init 2 */

}

/**
  * @brief TIM3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM3_Init(void)
{

  /* USER CODE BEGIN TIM3_Init 0 */

  /* USER CODE END TIM3_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM3_Init 1 */

  /* USER CODE END TIM3_Init 1 */
  htim3.Instance = TIM3;
```

```
  htim3.Init.Prescaler = 1000-1;
  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim3.Init.Period = 65535;
  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM3_Init 2 */

  /* USER CODE END TIM3_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
```

```
  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4
                    |GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8,
GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14|GPIO_PIN_15, GPIO_PIN_RESET);

  /*Configure GPIO pins : PA1 PA2 PA3 PA4
                         PA5 PA6 PA7 PA8 */
  GPIO_InitStruct.Pin = GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4
                    |GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pins : PB12 PB13 */
  GPIO_InitStruct.Pin = GPIO_PIN_12|GPIO_PIN_13;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pins : PB14 PB15 */
  GPIO_InitStruct.Pin = GPIO_PIN_14|GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pin : PB9 */
  GPIO_InitStruct.Pin = GPIO_PIN_9;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
```

```
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

8. **Breve explicación del código**

   Se tienen 3 funciones importantes:

- ☐ **Verificar y cambiar de estado**
- ☐ **Controlar motor**
- ☐ **Actualizar pantalla**

La primera se utiliza para la lógica del control, la segunda se utiliza para modificar el estado del motor dependiendo de lo que el usuario requiera y la tercera se utiliza para actualizar la pantalla según se necesite o cuando requiera de algún cambio.

9. Conclusiones

- ❖ **Se pudo entender que para realizar acciones simples como encender y apagar el motor se utiliza GPIO y esto se puede aplicar no solo para un motor sino para muchos otros componentes la lógica simplificada va a ser habilitar una salida del Gpio o deshabilitarla, 0 o 1**

- ❖ **Si se quiere medir ciertas características para el motor como el tiempo que está prendido y la velocidad máxima se pueden utilizar interrupciones y de este modo obtener un valor preciso puesto que las interrupciones son prioritarias en el programa**

- ❖ **Siempre se debe soldar los componentes que necesitan una conexión estable, pues si no se sueldan como en el caso del puente H no van a funcionar de una forma óptima**

- ❖ **Cuando se utiliza un método o cualquier factor que haga qué cambie los valores en la pantalla al final siempre se tiene que actualizar la pantalla para que ese valor o cambio se vea reflejado en la pantalla.**

- ❖ **El sistema se analiza dividiendo en 2 partes el problema principal:**
  1. **Contar pulsos**
  2. **Revisar tiempo**

  De esta forma se puede resolver el problema con 2 métodos:

1. **Contar cuántos pulsos lleva en un tiempo específico y obtener el tiempo de la vel máx y el tiempo que lleva prendido**

2. **Medir el tiempo de un pulso y con esto calcular cuándo la velocidad sea máx y el tiempo que lleve prendido**