EE 426 Final Project

Team 6

Samantha Connolly
Tayshawn Williams
Brandon Hubbell
Hannah Siegel

December 13, 2023

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this project is to engineer and program an autonomous vehicle utilizing brushless DC motors and ultrasonic sensors to navigate a designated path from a specified start to end point. This vehicle must autonomously traverse a 75 feet distance while maintaining a lateral distance within +-2 feet from the central trajectory and avoiding contact with surrounding obstacles. By integrating hardware components and sophisticated software algorithms, the aim is to demonstrate precise navigation, obstacle avoidance, and adherence to specified constraints in an autonomous vehicle system.

1.2 Scope

The project scope encompasses the design, assembly, and programming of an autonomous vehicle utilizing provided materials including brushless motors, ESCs, propellers, ultrasonic sensors, and a Cortex Arm7 processor board. The vehicle's construction involves creating a base frame within the dimensions of 15cmx15cm to 38cmx38cm using varied materials like cardboard, plywood, or plexiglass. The software development involves implementing control algorithms to ensure the vehicle autonomously navigates a straight 75 feet path, staying within +-2 feet laterally from the central trajectory, while avoiding obstacles. Testing and refinement to ensure accurate sensor readings, precise navigation, and obstacle avoidance within the specified parameters are essential components of this project's scope.

1.3 Overview

The project initiates with a comprehensive understanding of the specified requirements and constraints, including provided materials and components, delineating the scope and performance expectations for the autonomous vehicle. Subsequently, a meticulous design and planning phase ensues, outlining the base frame's configuration within prescribed dimensions and strategically plotting the placement of motors, ESCs, sensors, and the processor board for optimal functionality. With the design finalized, the project transitions into the mechanical construction phase, constructing the base frame using selected materials and securely affixing the motors, ESCs, and sensors according to the planned layout. Following this, electronics integration commences, entailing meticulous wiring to link the brushless motors, ESCs, and processor board to ensure seamless electrical functionality and control system operation. Simultaneously, the software development phase unfolds, configuring the Cortex Arm7 processor board, writing code to interface with ultrasonic sensors for data acquisition, and formulating precise control algorithms for speed regulation and obstacle avoidance. Subsequent stages involve rigorous testing and calibration to validate sensor readings, fine-tune control algorithms, and ensure accurate navigation and obstacle avoidance. Integrated testing and iterative refinement further enhance the vehicle's performance, concluding with comprehensive testing to assess its adherence to lateral boundaries and obstacle avoidance proficiency. Finally, documentation encapsulating the design process, encountered challenges, implemented solutions, and a detailed report or presentation showcasing the project methodology and results culminate the project.

1.4 Reference Material

Ultrasonic Sensor Reference Manual:
https://web.eece.maine.edu/~zhu/book/lab/HC-SR04%20User%20Manual.pdf

QWinOut A2212 1000KV Brushless Motor Reference Manual:
https://www.rhydolabz.com/documents/26/BLDC_A2212_13T.pdf

STM32F76xxx Reference Manual:
https://www.st.com/resource/en/reference_manual/rm0410-stm32f76xxx-and-stm32f77xxx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

Interfacing with BLDC Motor with STM32:
https://controllerstech.com/how-to-interface-bldc-motor-with-stm32/

2. Requirements

The project mandates the utilization of specific hardware components including two QWinOut A2212 1000KV Brushless Outrunner Motors, two QWinOut 2-4S 30A RC Brushless ESCs equipped with Simonk Firmware, and two QwinOut 10x4.5" 1045 1045R CW CCW Plastic Props. Additionally, three HC-SRO4 ultrasonic sensor modules, three 1 Inch Self-Adhesive Caster Wheels, and a Nucleo-F767zi Cortex Arm7 processor board are provided. Teams are tasked with constructing a base frame, employing materials like cardboard, plywood, or plexiglass, adhering to dimensions greater than 15cmx15cm but less than 38cmx38cm. The autonomous vehicle's functional requirements encompass traversing a 75 feet distance from a designated start to end point, maintaining a lateral precision within +-2 feet of the central trajectory, and executing obstacle avoidance to evade contact with walls positioned along the path. Teams have the option to enhance the vehicle's capabilities by integrating supplementary sensors and electronics as deemed necessary for improved performance or functionality.

3. Overview of the Implementation

3.1 Hardware – Sensors, Actuators, Processor, Schematic / Circuit Diagram / Block Diagram

HC-SRO4 Ultrasonic Sensor Modules:

Function: Ultrasonic sensors are used for distance measurement based on sound waves. HC-SRO4 sensors emit ultrasonic waves and measure the time taken for the waves to bounce back after hitting an obstacle, thereby determining the distance.

QWinOut A2212 1000KV Brushless Outrunner Motors:

Type:Brushless outrunner motors are known for their high torque and efficiency, commonly used in various RC applications.
Specifications: The A2212 motors have a 1000KV rating, denoting their rotational speed per volt. Higher KV motors generally offer higher speeds with lower torque.
Usage: These motors will drive the propellers to propel the autonomous vehicle forward. They convert electrical energy from the ESCs into mechanical motion, providing the necessary propulsion.

QWinOut 2-4S 30A RC Brushless ESC Simonk Firmware Electric Speed Controllers:

Type: Electronic Speed Controllers (ESCs) regulate the speed and direction of brushless motors by adjusting the voltage and current supplied to them.
Specifications: These ESCs support 2-4 cell LiPo batteries (2S-4S) and have a 30A current rating, indicating their capacity to handle current flow.
Firmware: Simonk Firmware is a popular choice for ESCs, known for its responsive and smooth motor control.

3.2 Software – Modules, Procedures, Algorithm

Speed Control

In order to control and adjust speed, closed-loop control is used. Feedback from the ultrasonic sensors is used to maintain a constant distance from obstacles and regulate speed based on environmental conditions. This is done by applying a coefficient to a differential distance calculation. This coefficient is less than 0.9, because a coefficient of 1 results in no change in speed or direction.

Obstacle Avoidance
In order for the vehicle to avoid obstacles basic logic was implemented to steer the vehicle away from detected obstacles by adjusting motor speeds. For instance, if an obstacle is detected on the right side, decrease the speed of the right motor to turn the vehicle left. This logic is the same as the speed control where the speed is determined by a coefficient applied to an equation taking a calculation of a distance measured by one sensor with a distance measurement taken by another.

4. Results and Discussion

The motors functioned successfully, responding accurately to control signals, demonstrating their capability to function autonomously. Additionally, the integrated sensors provided reliable data, enabling effective navigation and obstacle avoidance algorithms.

However, an obstacle arose concerning the vehicle's independence from an external voltage generator. Despite successful individual functionality, the vehicle encountered issues operating autonomously due to challenges in connecting to battery packs. The root cause appeared to stem from potential misconfigurations while setting up multiple batteries in series, leading to incorrect voltage supply or the possibility of exceeding the ESC's amperage tolerance.

Regrettably, this issue resulted in rendering one set of ESCs unusable, necessitating their replacement. Although this setback impacted the project's timeline, the replacement ESCs resolved the issue, allowing for continued experimentation and refinement of the autonomous vehicle's functionalities.

Moving forward, a more comprehensive approach to battery setup and voltage regulation should be implemented, ensuring compatibility with ESCs and preventing potential damage.

4.1 Weekly Progress Reports

Week 1 - 11/15/23

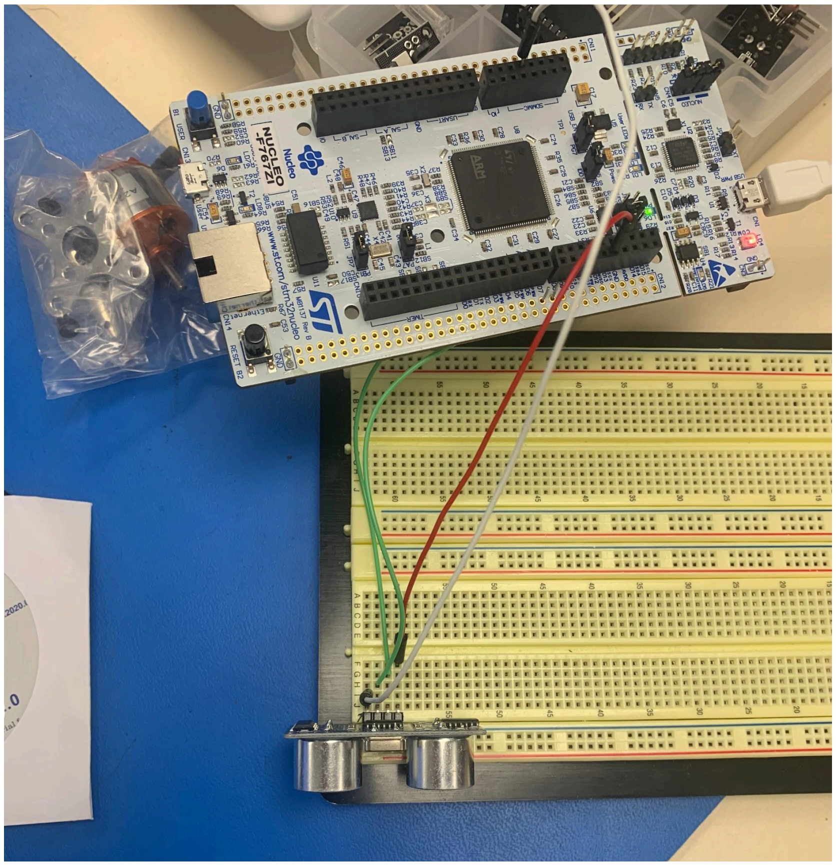
Rough design plan:

1. Ultrasonic sensor Wk 1
2. Motors Wk 2
3. Speed controller Wk 3
4. Combine Wk 4 + 5
5. Test Wk 5

We planned to code the ultrasonic sensors this week. We were able to finalize the code for individual working ultrasonic sensor, which we will copy for each sensor in the final code.

Functions:

- Measures distance in cm with a delay of 50 milliseconds
- A9 is trigger, A8 is echo (for this test, each sensor will hve its own pins)



Final pins for each sensor:

1. A8, A9 Forward sensor: a8-ECHO(GPIO input)

2. A10, A11 Left Sensor: A10-echo(input) —
3. A4, A5 Right Sensor: A4-echo(input)

Week 2 - 11/22/23

Function code info, integrating 3 sensors at once

Put in user code section

```
void sensorFunction( TRIG_PORT, TRIG_PIN, pMillis, ECHO_PORT, ECHO_PIN, Value1,
Value2, Distance)
{
while (1)
 {
                HAL_GPIO_WritePin(TRIG_PORT, TRIG_PIN, GPIO_PIN_SET);  // pull the TRIG pin HIGH
                __HAL_TIM_SET_COUNTER(&htim1, 0);
                while (__HAL_TIM_GET_COUNTER (&htim1) < 10);  // wait for 10 us
                HAL_GPIO_WritePin(TRIG_PORT, TRIG_PIN, GPIO_PIN_RESET);  // pull the TRIG pin low
                pMillis = HAL_GetTick(); // used this to avoid infinite while loop  (for timeout)
               // wait for the echo pin to go high
               while (!(HAL_GPIO_ReadPin (ECHO_PORT, ECHO_PIN)) && pMillis + 10 >
HAL_GetTick());
               Value1 = __HAL_TIM_GET_COUNTER (&htim1);
                pMillis = HAL_GetTick(); // used this to avoid infinite while loop (for timeout)
                // wait for the echo pin to go low
                while ((HAL_GPIO_ReadPin (ECHO_PORT, ECHO_PIN)) && pMillis + 50 > HAL_GetTick());
                Value2 = __HAL_TIM_GET_COUNTER (&htim1);
                Distance = (Value2-Value1)* 0.034/2;
                HAL_Delay(50);
}
}
```

Define function prior to main code as

```
void sensorFunction( TRIG_PORT, TRIG_PIN, pMillis, ECHO_PORT, ECHO_PIN, Value1,
Value2, Distance);
```

https://controllerstech.com/how-to-interface-bldc-motor-with-stm32/

https://www.youtube.com/watch?v=OwlfFp8fPN0 pwm

Week 3 + 4 11/28/23 and 12/6/23

Establish working motors and combine into full code.

ESC CODE - In a separate code file

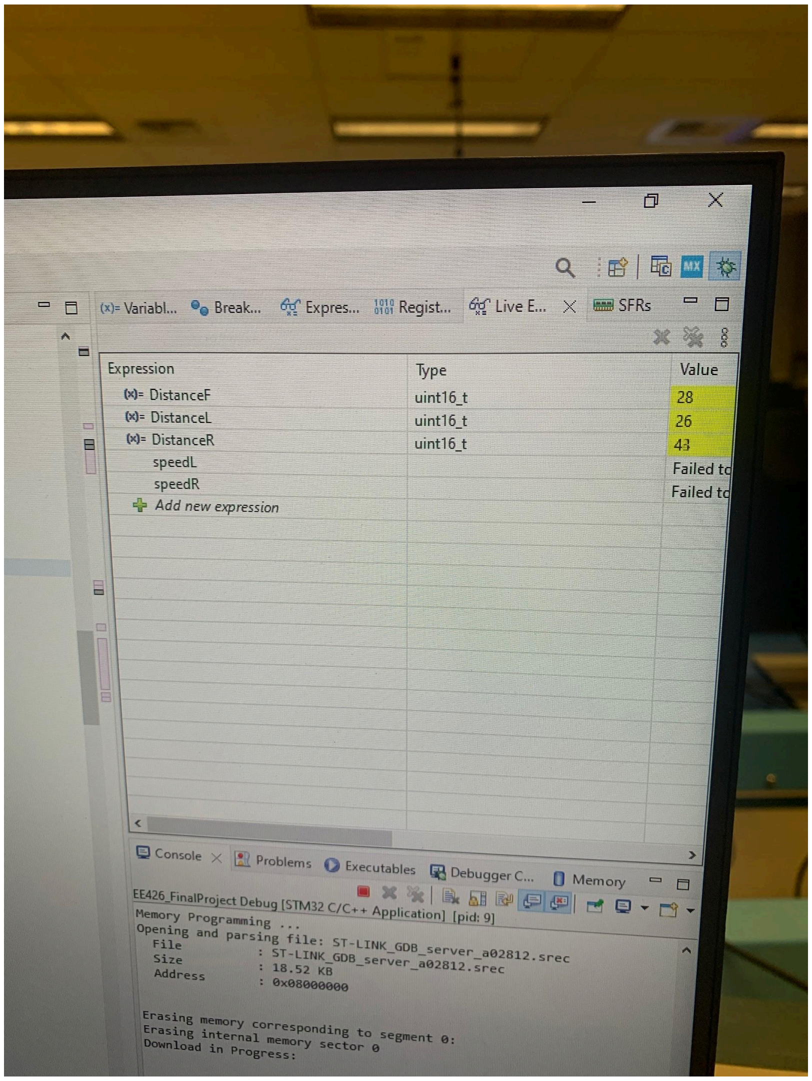
```
#define Calibrate 1 // started our calibration
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM1_Init(void);
int SpeedL= 0;
int SpeedR= 0;
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
        SpeedL= 65;
        TIM1->CCR1 = SpeedL;
        SpeedR= 65;
        TIM1->CCR2 = SpeedR;
}
int main(void)
{
 HAL_Init();
 SystemClock_Config();


 HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
 HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
#if Calibrate  // Calibration Setup
 TIM1->CCR1 = 100;  // Set the maximum pulse (2ms)
 HAL_Delay (2000);  // wait for 1 beep
 TIM1->CCR1 = 50;   // Set the minimum Pulse (1ms)
 HAL_Delay (1000);  // wait for 2 beeps
 TIM1->CCR1 = 0;    // reset to 0, so it can be controlled via ADC

 TIM1->CCR2 = 100;  // Set the maximum pulse (2ms)
 HAL_Delay (2000);  // wait for 1 beep
 TIM1->CCR2 = 50;   // Set the minimum Pulse (1ms)
 HAL_Delay (1000);  // wait for 2 beeps
 TIM1->CCR2 = 0;    // reset to 0, so it can be controlled via ADC

#endif
 while (1)
 {
 }
```

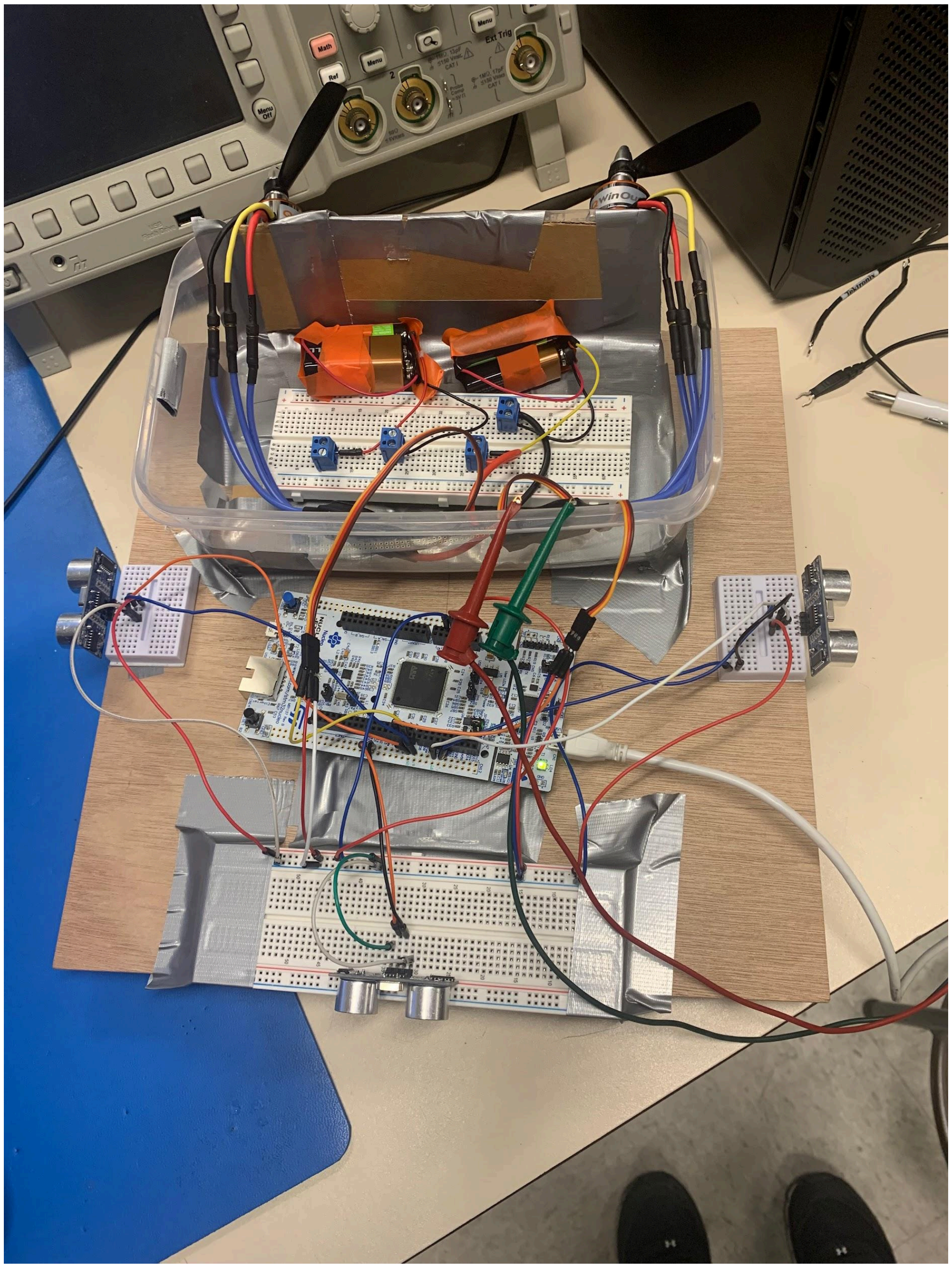
Reference: This code used a potentiometer to control speed, but I used
https://www.youtube.com/watch?v=qUFhDOTz160



Expression	Type	Value
DistanceF	uint16_t	28
DistanceL	uint16_t	26
DistanceR	uint16_t	43
speedL		Failed to
speedR		Failed to
Add new expression		

```
EE426_FinalProject Debug [STM32 C/C++ Application] [pid: 9]
Memory Programming ...
Opening and parsing file: ST-LINK_GDB_server_a02812.srec
  File          : ST-LINK_GDB_server_a02812.srec
  Size          : 18.52 KB
  Address       : 0x08000000


Erasing memory corresponding to segment 0:
Erasing internal memory sector 0
Download in Progress:
```



Main CODE

```
#include "main.h"
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;
I2C_HandleTypeDef hi2c1;
TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
//Front sensor
#define TRIG_PINF GPIO_PIN_3
#define TRIG_PORTF GPIOA
#define ECHO_PINF GPIO_PIN_2
#define ECHO_PORTF GPIOA
//Left sensor
#define TRIG_PINL GPIO_PIN_7
#define TRIG_PORTL GPIOA
#define ECHO_PINL GPIO_PIN_6
#define ECHO_PORTL GPIOA
//Right Sensor
#define TRIG_PINR GPIO_PIN_8
#define TRIG_PORTR GPIOA
#define ECHO_PINR GPIO_PIN_9
#define ECHO_PORTR GPIOA
uint32_t pMillisF;
uint32_t pMillisL;
uint32_t pMillisR;
uint32_t Value1F = 0;
uint32_t Value2F = 0;
uint32_t Value1L = 0;
uint32_t Value2L = 0;
uint32_t Value1R = 0;
uint32_t Value2R = 0;
uint16_t DistanceF  = 0;
uint16_t DistanceL  = 0;
uint16_t DistanceR  = 0;
/* USER CODE END PV */
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_TIM1_Init(void);
static void MX_TIM2_Init(void);
static void MX_ADC1_Init(void);
static void MX_I2C1_Init(void);
static void MX_TIM3_Init(void);
int i=0; // Initiated our while loop sequence
int main(void)
{
//  /* USER CODE BEGIN 1 */
            void mStart()       // Motor calibration
            {
              TIM1->CCR1 = 100;  // Set the maximum pulse (2ms)
              HAL_Delay (2000);  // wait for 1 beep
              TIM1->CCR1 = 65;   // Set the minimum Pulse (1ms)
              HAL_Delay (1000);  // wait for 2 beeps

              TIM3->CCR1 = 100;  // Set the maximum pulse (2ms)
              HAL_Delay (2000);  // wait for 1 beep
              TIM3->CCR1 = 65;   // Set the minimum Pulse (1ms)
              HAL_Delay (3000);  // wait for 2 beeps

            }

            //  SENSOR FUNCTIONS
            void sensorFunctionF()
            {
                                    HAL_GPIO_WritePin(TRIG_PORTF, TRIG_PINF, GPIO_PIN_SET);  // pull the TRIG
pin HIGH
                            __HAL_TIM_SET_COUNTER(&htim2, 0);
                            while (__HAL_TIM_GET_COUNTER (&htim2) < 10);  // wait for 10 us
                            HAL_GPIO_WritePin(TRIG_PORTF, TRIG_PINF, GPIO_PIN_RESET);  // pull the TRIG pin
low
                            pMillisF = HAL_GetTick(); // used this to avoid infinite while loop  (for timeout)
                            // wait for the echo pin to go high
                            while (!(HAL_GPIO_ReadPin (ECHO_PORTF, ECHO_PINF)) && pMillisF + 10 >
HAL_GetTick());
                            Value1F = __HAL_TIM_GET_COUNTER (&htim2);
                            pMillisF = HAL_GetTick(); // used this to avoid infinite while loop (for timeout)
                            // wait for the echo pin to go low
                            while ((HAL_GPIO_ReadPin (ECHO_PORTF, ECHO_PINF)) && pMillisF + 50 >
HAL_GetTick());
                            Value2F = __HAL_TIM_GET_COUNTER (&htim2);
                            DistanceF = (Value2F-Value1F)* 0.034/2;
                            HAL_Delay(50);
                        //    return DistanceF;
            }
            void sensorFunctionL()
            {
                                    HAL_GPIO_WritePin(TRIG_PORTL, TRIG_PINL, GPIO_PIN_SET);  // pull the TRIG
pin HIGH
                            __HAL_TIM_SET_COUNTER(&htim2, 0);
                            while (__HAL_TIM_GET_COUNTER (&htim2) < 10);  // wait for 10 us
                            HAL_GPIO_WritePin(TRIG_PORTL, TRIG_PINL, GPIO_PIN_RESET);  // pull the TRIG pin
low
                            pMillisL = HAL_GetTick(); // used this to avoid infinite while loop  (for timeout)
                            // wait for the echo pin to go high
                            while (!(HAL_GPIO_ReadPin (ECHO_PORTL, ECHO_PINL)) && pMillisL + 10 >
HAL_GetTick());
                            Value1L = __HAL_TIM_GET_COUNTER (&htim2);
                            pMillisL = HAL_GetTick(); // used this to avoid infinite while loop (for timeout)
                            // wait for the echo pin to go low
                            while ((HAL_GPIO_ReadPin (ECHO_PORTL, ECHO_PINL)) && pMillisL + 50 >
HAL_GetTick());
                            Value2L = __HAL_TIM_GET_COUNTER (&htim2);
                            DistanceL = (Value2L-Value1L)* 0.034/2;
                            HAL_Delay(50);
                        //    return DistanceL;
            }
            void sensorFunctionR()
            {
                                    HAL_GPIO_WritePin(TRIG_PORTR, TRIG_PINR, GPIO_PIN_SET);  // pull the TRIG
pin HIGH
                            __HAL_TIM_SET_COUNTER(&htim2, 0);
                            while (__HAL_TIM_GET_COUNTER (&htim2) < 10);  // wait for 10 us
                            HAL_GPIO_WritePin(TRIG_PORTR, TRIG_PINR, GPIO_PIN_RESET);  // pull the TRIG pin
low
                            pMillisR = HAL_GetTick(); // used this to avoid infinite while loop  (for timeout)
                            // wait for the echo pin to go high
                            while (!(HAL_GPIO_ReadPin (ECHO_PORTR, ECHO_PINR)) && pMillisR + 10 >
HAL_GetTick());
                            Value1R = __HAL_TIM_GET_COUNTER (&htim2);
                            pMillisR = HAL_GetTick(); // used this to avoid infinite while loop (for timeout)
                            // wait for the echo pin to go low
                            while ((HAL_GPIO_ReadPin (ECHO_PORTR, ECHO_PINR)) && pMillisR + 50 >
HAL_GetTick());
                            Value2R = __HAL_TIM_GET_COUNTER (&htim2);
                            DistanceR = (Value2R-Value1R)* 0.034/2;
                            HAL_Delay(50);
                        //    return DistanceR;
            //
            }
   HAL_Init();
   SystemClock_Config();
   MX_GPIO_Init();
   MX_DMA_Init();
   MX_TIM1_Init();
   MX_TIM2_Init();
   MX_ADC1_Init();
   MX_I2C1_Init();
   MX_TIM3_Init();
   /* USER CODE BEGIN 2 */
   HAL_TIM_Base_Start(&htim2);
   /* USER CODE END 2 */
    HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
    HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);
   /* Infinite loop */
    HAL_GPIO_WritePin(TRIG_PORTF, TRIG_PINF, GPIO_PIN_RESET);  // pull the TRIG pin low
    HAL_GPIO_WritePin(TRIG_PORTL, TRIG_PINL, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(TRIG_PORTR, TRIG_PINR, GPIO_PIN_RESET);

   while (1)
   {
                while (i!= 1)
                           {
                           mStart();
                           i=1;
                           }
                                                 sensorFunctionF(); //  PA7 Output, PA15 Input
                                                 sensorFunctionL(); // PC9 Output, PC8 Input
                                                 sensorFunctionR(); // PC7 Output, PC6 Input
                                                if((DistanceF > 180) && (DistanceL > 30) && (DistanceR > 30)) // Motor run
                                                                                               {
                                                                                                           TIM3->CCR1 =
80;

HAL_Delay(25);
                                                                                                           TIM1->CCR1 =
80;

                                                                                               }
                                                                    else if (DistanceF < 180) // Front sensor stopping
                                                                                {
                                                                                          TIM3->CCR1 =0;
                                                                                          HAL_Delay(25);
                                                                                          TIM1->CCR1 =0;
                                                                                }
                                                                    else if ((DistanceL < 30) && (DistanceF > 180)) // Left
sensor correction

                                                                    {
                                                                                TIM3->CCR1 = 100;
                                                                                HAL_Delay(25);
                                                                                TIM9 ->CCR1 = 80;
                                                                    }
                                                                    else if ((DistanceR <30) && (DistanceF > 180)) // Right
sensor correction

                                                                    {
                                                                                TIM3 ->CCR1 = 80;
                                                                                HAL_Delay(25);
                                                                                TIM9 -> CCR1 = 100;
                                                                    }


   }
}
```