```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2023 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 */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define TRIGGERPULSE 15
#define HALFSECONDPERIOD 500
#define NINETYDEGREETURN 480
#define HUNDREDEIGHTYDEGREETURN 970
/* USER CODE END PD */

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

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc;

TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;

UART_HandleTypeDef huart1;

/* USER CODE BEGIN PV */
unsigned short distance = 0;                // Distance in cm
unsigned short echoStartTime = 0;           // Time at which the echo is
triggered
int echoTotalTime = 0;                      // Total time received by echo
unsigned short cautionMode = 0;             // Depending on this var, the robot
will take various speeds
unsigned short maxSpeed = 50;               // maxSpeed ~= DC
unsigned short turnMode = 1;
unsigned short prevTurnMode = 0;
unsigned short tim3Time = 0;
```

```
//UART VARIABLES
unsigned short globalMode = 1;
uint8_t received[7];
uint8_t lineBreak[7];
uint8_t answer[7] = {0,0,0,0,0,0,0};
uint8_t choose[7] = {67, 104, 111, 111, 115, 101, 32};
uint8_t aMode[7] = {97, 32, 109, 111, 100, 101, 32};
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM3_Init(void);
static void MX_TIM4_Init(void);
static void MX_ADC_Init(void);
static void MX_USART1_UART_Init(void);
/* USER CODE BEGIN PFP */
void goForward(void);
void goBackward(void);
void forwardBrake(void);
void Stop(void);
void goRight(void);
void goLeft(void);

void lineBreakFunction(void);
void chooseAMode(void);
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
void lineBreakFunction(void) {
  lineBreak[0] = 10;
  HAL_UART_Transmit(&huart1, lineBreak, 1, 10000);
  lineBreak[0] = 13;
  HAL_UART_Transmit(&huart1, lineBreak, 1, 10000);
}
void chooseAMode(void) {
  HAL_UART_Transmit(&huart1, choose, 7, 10000);
  HAL_UART_Transmit(&huart1, aMode, 7, 10000);
}

void goForward(void) {
  GPIOA->BSRR = (1 << 11)<<16;
  GPIOA->BSRR = (1 << 12)<<16;
  TIM4->CCR3 = (maxSpeed-(maxSpeed/10)); // DC initialize to 99 (100%)
  TIM4->CCR4 = maxSpeed; // DC initialize to 99 (100%)
}

void goBackward(void) {
  GPIOA->BSRR = (1 << 11);
  GPIOA->BSRR = (1 << 12);
  TIM4->CCR3 = 99-(maxSpeed-(maxSpeed/10)); // DC initialize to 99 (100%)
  TIM4->CCR4 = 99-maxSpeed; // DC initialize to 99 (100%)
}

void forwardBrake(void) {
  GPIOA->BSRR = (1 << 11)<<16;
  GPIOA->BSRR = (1 << 12)<<16;
  TIM4->CCR3 = (((maxSpeed-maxSpeed/10)-35)*(distance-10)/10)+35;
  TIM4->CCR4 = ((maxSpeed-35)*(distance-10)/10)+35;
}
```

```
void Stop(void) {
  GPIOA->BSRR = (1 << 11)<<16;
  GPIOA->BSRR = (1 << 12)<<16;
  TIM4->CCR3 = 0; // DC initialize to 0 (0%)
  TIM4->CCR4 = 0; // DC initialize to 0 (0%)
}

void goRight(void) {
  GPIOA->BSRR = (1 << 11);
  GPIOA->BSRR = (1 << 12)<<16;
  TIM4->CCR3 = 0;
  TIM4->CCR4 = 99;
}

void goLeft(void) {
  GPIOA->BSRR = (1 << 11)<<16;
  GPIOA->BSRR = (1 << 12);
  TIM4->CCR3 = 99;
  TIM4->CCR4 = 0;
}


void ADC1_IRQHandler(void) {
  if((ADC1->SR & (1 << 1)) != 0) {
    maxSpeed = (ADC1->DR) * (99 - 50) / (4095) + 50;
  }
  ADC1->SR = 0;
}

void TIM3_IRQHandler(void) {
  if((TIM3->SR & 0x0002) != 0) {
    TIM3->CCR1 += HALFSECONDPERIOD;

    // LAUNCH ADC
    ADC1->CR2 |= 0x40000000;

    // LAUNCH TRIGGER
    GPIOD->BSRR = (1 << 2);
    TIM2->CCR2 = TIM2->CNT + TRIGGERPULSE;
    TIM2->DIER |= (1 << 2);          // Enable DIER Ch2

    if(globalMode == 1) {
      Stop();
    }
    if(globalMode == 2) {
      goForward();
    }

    if(globalMode == 3) {
      goBackward();
    }

    if(globalMode == 4) {
      goRight();
    }

    if(globalMode == 5) {
      goLeft();
    }

    if(globalMode == 6) {    // AUTOMATIC MODE
      // DECIDE MODE
      if(cautionMode == 0) {       // CASE DISTANCE < 10 cm
        if(turnMode == 0) {
```

```
          GPIOA->BSRR = (1 << 1);
          Stop();
          TIM2->DIER &= ~(1 << 1);  // Disable Echo Ch1
          TIM3->DIER |= (1 << 2);
          TIM3->CCR2 = TIM3->CNT + HALFSECONDPERIOD;
          turnMode = 1;
        }
        else {
          GPIOA->BSRR = (1 << 1);
          Stop();
        }
      }
      else if(cautionMode == 1) { // CASE DISTANCE >= 10cm && <= 20cm

        forwardBrake();

        if((GPIOA->ODR & (1 << 1)) != 0) {
          GPIOA->BSRR = (1 << 1)<<16;
        }
        else {
          GPIOA->BSRR = (1 << 1);
        }
      }
      else {                      // CASE DISTANCE > 20 cm
        GPIOA->BSRR = (1 << 1)<<16;
        goForward();

      }
    }
    received[0]=0;
    TIM3->SR &= ~(0x0002);
  }

  if((TIM3->SR & 0x0004) != 0) {

      if(prevTurnMode == 4) {
        turnMode = 2;
      }
      else if(prevTurnMode == 2) {
        turnMode = 3;
      }
      else if(prevTurnMode == 3) {
        turnMode = 4;
      }
      prevTurnMode = turnMode;
      tim3Time = TIM3->CNT;
      TIM3->DIER &= ~(1 << 2);
      TIM3->SR &= ~(0x0004);

  }
}

void TIM2_IRQHandler(void) {

  // TRIGGER (CHANNEL 2)
  if((TIM2->SR & 0x0004) != 0) {      // TIM2 CHANNEL  2
    GPIOD->BSRR = (1 << 2) << 16;     // Trigger -> 0
    TIM2->DIER &= ~(1 << 2);          // Disable DIER Ch2
    TIM2->SR &= ~(0x0004);            // Clear the flag
  }

  // ECHO (CHANNEL 1)
  if((TIM2->SR & 0x0002) != 0) {      // TIM2 CHANNEL  1
```

```
    if((GPIOA->IDR&(1 << 5)) != 0) {
      echoStartTime = TIM2->CCR1;
    }
    else {
      echoTotalTime = TIM2->CCR1 - echoStartTime;
      if(echoTotalTime < 0) echoTotalTime += 0xFFFF;
      distance = (echoTotalTime) / 90;
      // Decide buzzer Mode
      if(distance < 10) {
        cautionMode = 0;
        turnMode = 0;
        uint8_t answer[7] = {60, 32, 49, 48, 32, 99, 109};
        HAL_UART_Transmit(&huart1, answer, 7, 10000);
        lineBreakFunction();
      }
      if(distance >= 10 && distance <= 20) {
        cautionMode = 1;
        prevTurnMode = 4;
        uint8_t answer[7] = {79, 98, 115, 116, 97, 99, 32};
        HAL_UART_Transmit(&huart1, answer, 7, 10000);
        uint8_t answer1[7] = {49, 48, 60, 120, 60, 50, 48};
        HAL_UART_Transmit(&huart1, answer1, 7, 10000);
        lineBreakFunction();
      }
      if(distance > 20) {
        cautionMode = 2;
      }
    }

    TIM2->SR &= ~(0x0002);            // Clear the flag
  }

}
/* 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_TIM2_Init();
  MX_TIM3_Init();
  MX_TIM4_Init();
  MX_ADC_Init();
  MX_USART1_UART_Init();
  /* USER CODE BEGIN 2 */

  // CONFIG BUZZER GPIO
  GPIOA->MODER &= ~(1 << (1*2 +1));   // PA1 as Digital Output (01)
  GPIOA->MODER |= (1 << (1*2));

  // CONFIG TRIGGER GPIO
  GPIOD->MODER &= ~(1 << (2*2 +1));   // PD2 as Digital Output (01)
  GPIOD->MODER |= (1 << (2*2));

  // CONFIG ECHO GPIO
  GPIOA->MODER |= (1 << (5*2 +1));    // PA5 as AF (10)
  GPIOA->MODER &= ~(1 << (5*2));
  GPIOA->AFR[0] &= ~(0x00F00000);
  GPIOA->AFR[0] |= (1 << 5*4);        // Select the AF1 for PA5

  // MOTOR 1 CONFIG
  GPIOB->MODER |= (1 << (8*2 +1));    // PB8 as AF (10)
  GPIOB->MODER &= ~(1 << (8*2));
  GPIOB->AFR[1] |= (1 << (0*4 +1));     // PB8 as AF2 (linked to TIM4)
  GPIOA->MODER &= ~(1 << (11*2 +1));  // PA11 as Digital Output(01)
  GPIOA->MODER |= (1 << (11*2));

  // MOTOR 2 CONFIG
  GPIOB->MODER |= (1 << (9*2 +1));    // PB8 as AF (10)
  GPIOB->MODER &= ~(1 << (9*2));
  GPIOB->AFR[1] |= (1 << (1*4 +1));     // PB9 as AF2 (linked to TIM4)
  GPIOA->MODER &= ~(1 << (12*2 +1));  // PA12 as Digital Output(01)
  GPIOA->MODER |= (1 << (12*2));

  // POTENCIOMETER CONFIG
  GPIOA->MODER |= (1 << (4*2 +1));    // PA4 as analog (11)
  GPIOA->MODER |= (1 << (4*2));

  // ADC CONFIG
  ADC1->CR2 &= ~(0x00000001);
  ADC1->CR1 |= 0x00000020;
  ADC1->CR2 |= 0x00000400;
  ADC1->SQR1 = 0x00000000;
  ADC1->SQR5 = 0x00000004;

  ADC1->CR2 |= 0x00000001;

  // TIM2 CONFIG (TRIGGER ECHO)
  // Select the internal clock
  TIM2->CR1 = 0x0000;     // ARPE = 0 (only for PWM); CEN = 0 (counter disabled
for configuration)
  TIM2->CR2 = 0x0000;     // All zeros
  TIM2->SMCR = 0x0000;    // All zeros

  // Counter behavior setting
  TIM2->PSC = 31;               // freq_Counter = 32 MHz / 32 = 1 MHz -->>
T_Counter = 1 us
  TIM2->CNT = 0;                // Initialize the counter at 0
  TIM2->ARR = 0xFFFF;           // Maximum value
  TIM2->CCR2 = TRIGGERPULSE;    // CCR2 = Trigger Pulse (~12us)

  // Setting IRQ or not
  TIM2->DIER &= ~(0xFF);
```

```
  TIM2->DIER |= (1 << 1);       // Enable IRQ for Channel 1 (CC1E)
  TIM2->DIER &= ~(1 << 2);      // Disable DIER Ch2

  // Output mode
  TIM2->CCMR1 &= ~(0xFFFF);   // Clear CCMR1 register
  TIM2->CCMR1 |= 0x0001;      // CC1S = 01 (TIC)
                              // CC2S = 00 (TOC)
                              // OC2PE = 0 (only for PWM)
                              // OC2M = 000 (No output)
  TIM2->CCER = 0x000b;        // CC1E = 1 (Enable capture)
                              // CC2E = 0 (Disable hardware output)
                              // CC1NP:CC1P = 11 (Both edges)

  // Counter enabling
  TIM2->CR1 |= 0x0001;        // CEN = 1 -->> Start counter
  TIM2->EGR |= 0x0001;        // UG = 1 -->> Update all registers
  TIM2->SR = 0x0000;          // Clear counter flags


  // TIM3 CONFIG (PPal Loop)
  // Select the internal clock
  TIM3->CR1 = 0x0000;         // ARPE = 0 (only for PWM); CEN = 0 (counter
disabled for configuration)
  TIM3->CR2 = 0x0000;         // All zeros
  TIM3->SMCR = 0x0000;        // All zeros

  //Counter behavior setting
  TIM3->PSC = 31999;              // freq_Counter = 32 MHz / 32000 = 1 kHz -->>
T_Counter = 1 ms -->> CCR = 500
  TIM3->CNT = 0;                  // Initialize the counter at 0
  TIM3->ARR = 0xFFFF;             // Set to the maximum
  TIM3->CCR1 = HALFSECONDPERIOD;  // CH1 CCR1 = 500

  // Setting IRQ or not
  TIM3->DIER |= (1 << 1);    // We enable an IRQ in channel 1
  TIM3->DIER &= ~(1 << 2);

  // Output mode
  TIM3->CCMR1 &= ~(0xFFFF);   // Clear CCMR1 register
  TIM3->CCMR1 = 0x0000;       // CC1S = 0 (TOC); OC1M = 000 (no output); OC1PE =
0 (No preload)
  TIM3->CCER = 0x0000;        // CC1NP = 0; CC1P = 0; CC1E = 0 (Disable output)

  // Counter enabling
  TIM3->CR1 |= 0x0001;        // CEN = 1 -->> Start counter
  TIM3->EGR |= 0x0001;        // UG = 1 -->> Update all registers
  TIM3->SR = 0x0000;          // Clear counter flags

  // TIM4 CONFIG
  // Internal clock selection: CR1, CR2, SMRC
  TIM4->CR1 = 0x0080; // ARPE = 1 -> Is PWM; CEN = 0; Counter OFF
  TIM4->CR2 = 0x0000; // Always 0 in this course
  TIM4->SMCR = 0x0000; // Always 0 in this course

  // Counter setting: PSC, CNT, ARR and CCRx
  TIM4->PSC = 319; // Pre-scaler=320 -> f_counter=32000000/320 = 100000
steps/second -> T = 10us
  TIM4->CNT = 0; // Initialize counter to 0
  TIM4->ARR = 99; // PWM Frequency to 1000 Hz and 100 steps
  TIM4->CCR3 = maxSpeed; // DC initialize to 99 (50%)
  TIM4->CCR4 = maxSpeed; // DC initialize to 99 (50%)

  // Select, or not, IRQ: DIER
  TIM4->DIER = 0x0000; // No IRQ when counting is finished -> CCyIE = 0
```

```
// Output mode
TIM4->CCMR1 &= ~(0xFFFF); // Clear CCMR1
TIM4->CCMR2 |= (0x6868);     // CCyS = 0 (TOC, PWM)
                          // OCyM = 110 (PWM starting in 1)
                          // OCyPE = 1 (with preload)
TIM4->CCER |= 0x1100;      // CCyP = 0 (always in PWM)
                          // CCyE = 1 (hardware output activated)

// Counter enabling
TIM4->CR1 |= 0x0001;  // CEN = 1 -> Start counter
TIM4->EGR |= 0x0001;  // UG = 1 -> Generate update event
TIM4->SR = 0;         // Counter flags cleared

// Enabling ADC1_IRQ at NVIC
NVIC->ISER[0] |= (1 << 18);

// Enabling TIM2_IRQ at NVIC
NVIC->ISER[0] |= (1 << 28);

// Enabling TIM3_IRQ at NVIC
NVIC->ISER[0] |= (1 << 29);

HAL_UART_Receive_IT(&huart1, received, 1);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{


    if(turnMode == 2) {
      uint8_t answer[7] = {82, 105, 103, 104, 116, 0, 0};
      HAL_UART_Transmit(&huart1, answer, 7, 10000);
      lineBreakFunction();
      while(TIM3->CNT < tim3Time + NINETYDEGREETURN) {
        goRight();
      }
      TIM2->DIER |= (1 << 1);  // Enable Echo Ch1
      turnMode = 1;
      cautionMode = 0;
    }
    else if(turnMode == 3){
      uint8_t answer[7] = {76, 101, 102, 116, 0, 0, 0};
      HAL_UART_Transmit(&huart1, answer, 7, 10000);
      lineBreakFunction();
      while(TIM3->CNT < tim3Time + HUNDREDEIGHTYDEGREETURN) {
        goLeft();
      }
      TIM2->DIER |= (1 << 1);  // Enable Echo Ch1
      turnMode = 1;
      cautionMode = 0;
    }
    else if(turnMode == 4){
      uint8_t answer[7] = {76, 101, 102, 116, 0, 0, 0};
      HAL_UART_Transmit(&huart1, answer, 7, 10000);
      lineBreakFunction();
      while(TIM3->CNT < tim3Time + NINETYDEGREETURN) {
        goLeft();
      }
      TIM2->DIER |= (1 << 1);  // Enable Echo Ch1
      turnMode = 1;
    }
```

```
    /* 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_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** 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.PLLMUL = RCC_PLL_MUL6;
  RCC_OscInitStruct.PLL.PLLDIV = RCC_PLL_DIV3;
  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_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief ADC Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC_Init(void)
{

  /* USER CODE BEGIN ADC_Init 0 */

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

```
  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC_Init 1 */

  /* USER CODE END ADC_Init 1 */

  /** Configure the global features of the ADC (Clock, Resolution, Data
Alignment and number of conversion)
  */
  hadc.Instance = ADC1;
  hadc.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
  hadc.Init.Resolution = ADC_RESOLUTION_12B;
  hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc.Init.EOCSelection = ADC_EOC_SEQ_CONV;
  hadc.Init.LowPowerAutoWait = ADC_AUTOWAIT_DISABLE;
  hadc.Init.LowPowerAutoPowerOff = ADC_AUTOPOWEROFF_DISABLE;
  hadc.Init.ChannelsBank = ADC_CHANNELS_BANK_A;
  hadc.Init.ContinuousConvMode = DISABLE;
  hadc.Init.NbrOfConversion = 1;
  hadc.Init.DiscontinuousConvMode = DISABLE;
  hadc.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc.Init.DMAContinuousRequests = DISABLE;
  if (HAL_ADC_Init(&hadc) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure for the selected ADC regular channel its corresponding rank in
the sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_4;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_4CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC_Init 2 */

  /* USER CODE END ADC_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};

  /* USER CODE BEGIN TIM2_Init 1 */

  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
```

```
  htim2.Init.Prescaler = 0;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 65535;
  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();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != 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 = 0;
  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 TIM4 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM4_Init(void)
{

  /* USER CODE BEGIN TIM4_Init 0 */

  /* USER CODE END TIM4_Init 0 */

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

  /* USER CODE BEGIN TIM4_Init 1 */

  /* USER CODE END TIM4_Init 1 */
  htim4.Instance = TIM4;
  htim4.Init.Prescaler = 0;
  htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim4.Init.Period = 65535;
  htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim4) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM4_Init 2 */

  /* USER CODE END TIM4_Init 2 */

}

/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART1_UART_Init(void)
{

  /* USER CODE BEGIN USART1_Init 0 */

  /* USER CODE END USART1_Init 0 */
```

```
  /* USER CODE BEGIN USART1_Init 1 */

  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 9600;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */

  /* USER CODE END USART1_Init 2 */

}

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

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

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  HAL_UART_Receive_IT(huart, received, 1); // Vuelve a activar Rx por haber
acabado el buffer
  if(received[0] != '1' && received[0] != '2' && received[0] != '3' &&
received[0] != '4' && received[0] != '5' && received[0] != '6') {
    uint8_t answer[7] = {69, 114, 114, 111, 114,0,0}; // Error message
    lineBreakFunction();
    HAL_UART_Transmit(&huart1, answer, 7, 10000);
  }
  else {
    if(received[0] == '1') {   // A 1 is received -> STOP
      globalMode = 1;
      uint8_t answer[7] = {83, 116, 111, 112, 112, 101, 100};
      lineBreakFunction();
      HAL_UART_Transmit(&huart1, answer, 7, 10000);
    }
    if(received[0] == '2') {   // A 2 is received -> FORWARD
      globalMode = 2;
```

```
      uint8_t answer[7] = {70, 111, 114, 119, 97, 114, 100};
      lineBreakFunction();
      HAL_UART_Transmit(&huart1, answer, 7, 10000);
    }
    if(received[0] == '3') {   // A 3 is received -> BACK
      globalMode = 3;
      uint8_t answer[7] = {66, 97, 99, 107, 0, 0, 0};
      lineBreakFunction();
      HAL_UART_Transmit(&huart1, answer, 7, 10000);
    }
    if(received[0] == '4') {   // A 4 is received -> RIGHT
      globalMode = 4;
      uint8_t answer[7] = {82, 105, 103, 104, 116, 0, 0};
      lineBreakFunction();
      HAL_UART_Transmit(&huart1, answer, 7, 10000);
    }
    if(received[0] == '5') {   // A 5 is received -> LEFT
      globalMode = 5;
      uint8_t answer[7] = {76, 101, 102, 116, 0, 0, 0};
      lineBreakFunction();
      HAL_UART_Transmit(&huart1, answer, 7, 10000);
    }
    if(received[0] == '6') {   // A 6 is received -> AUTOMATIC
      globalMode = 6;
      uint8_t answer[7] = {65, 117, 116, 111, 110, 111, 109};
      lineBreakFunction();
      HAL_UART_Transmit(&huart1, answer, 7, 10000);
    }
  }
  lineBreakFunction();
  chooseAMode();
}
/* 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 */
```