Microcontroller & Interfacing

CE205T

	CLO-2		CLO-3					Total
Part	B	C	A	D	E	F	G	
Marks	50	50	50	50	50	50	50	
Obt.								

Project Name

1: Bluetooth RC Car

Group Number

1: Abdullah Basit Awan (BSCE23013)

2: Fozan Khan (BSCE23033)

3: Muhammad Umar Akbar (BSCE23061)

A. Overview [CLO-3, 50 Marks]

The objective of this project is to design and implement a wirelessly controlled Remote-Control (RC) car that can be operated via a mobile phone using Bluetooth communication. The system is built around the STM32 Black Pill microcontroller, a powerful and affordable ARM Cortex-M4-based development board, which serves as the brain of the vehicle.

The RC car is designed to move in multiple directions—forward, backward, left, and right—based on commands sent from a smart phone. Control is enabled through a Bluetooth module (HC-05), which receives signals wirelessly from a mobile phone running a Bluetooth controller app. These signals are interpreted by the STM32 Black Pill microcontroller, which then drives the four DC motors through an L298N motor driver module. However, due to the compatibility limitations of the HC-05 module, the system only supports Android devices and is not operable via iOS-based phones. This setup serves as a practical implementation of a low-level embedded system, showcasing key concepts such as wireless communication via bluetooth, motor control, microcontroller-based interfacing, and real-time signal processing.

GOALS

- Designed and built a Bluetooth-controlled RC car powered by the STM32 Black Pill microcontroller.
- Implemented precise motor control logic using GPIOs and PWM to enable smooth movement in all directions.
- Interfaced key external components including four DC motors, an L298N motor driver, and an HC-05 Bluetooth module for wireless control.
- Demonstrated real-time embedded system integration using UART and PWM to control and coordinate the car's functionality via an Android smartphone.

B. List of Components Used [CLO-2, 50 Marks]

#	Components	Quantity	Price (per item)	Data Sheet
1	Black Pill STM32F411CEU6	1	PKR 1250	Black Pill Datasheet
2	HC-05 Bluetooth Module	1	PKR 750	HC-05 Datasheet
3	L298N Motor Drivers	2	PKR 400	L298N Motor Driver
4	DC Motors	4	PKR 130	Specifications of DC Motor
5	18650 Lithium Ion Cell	6	PKR 250	Lithium Ion Cell 18650 2500mAh Battery Datasheet
6	Power Bank	1	PKR 3000	N/A
7	Cell Holder 3x	2	PKR 80	N/A
8	Jumper Wires	20-30	PKR 200	N/A
9	ST-Link V2	1	PKR 1050	ST- Link V2 Programmer Specs
10	White LEDs	4	PKR 4	N/A
11	Red LEDs	4	PKR 4	N/A

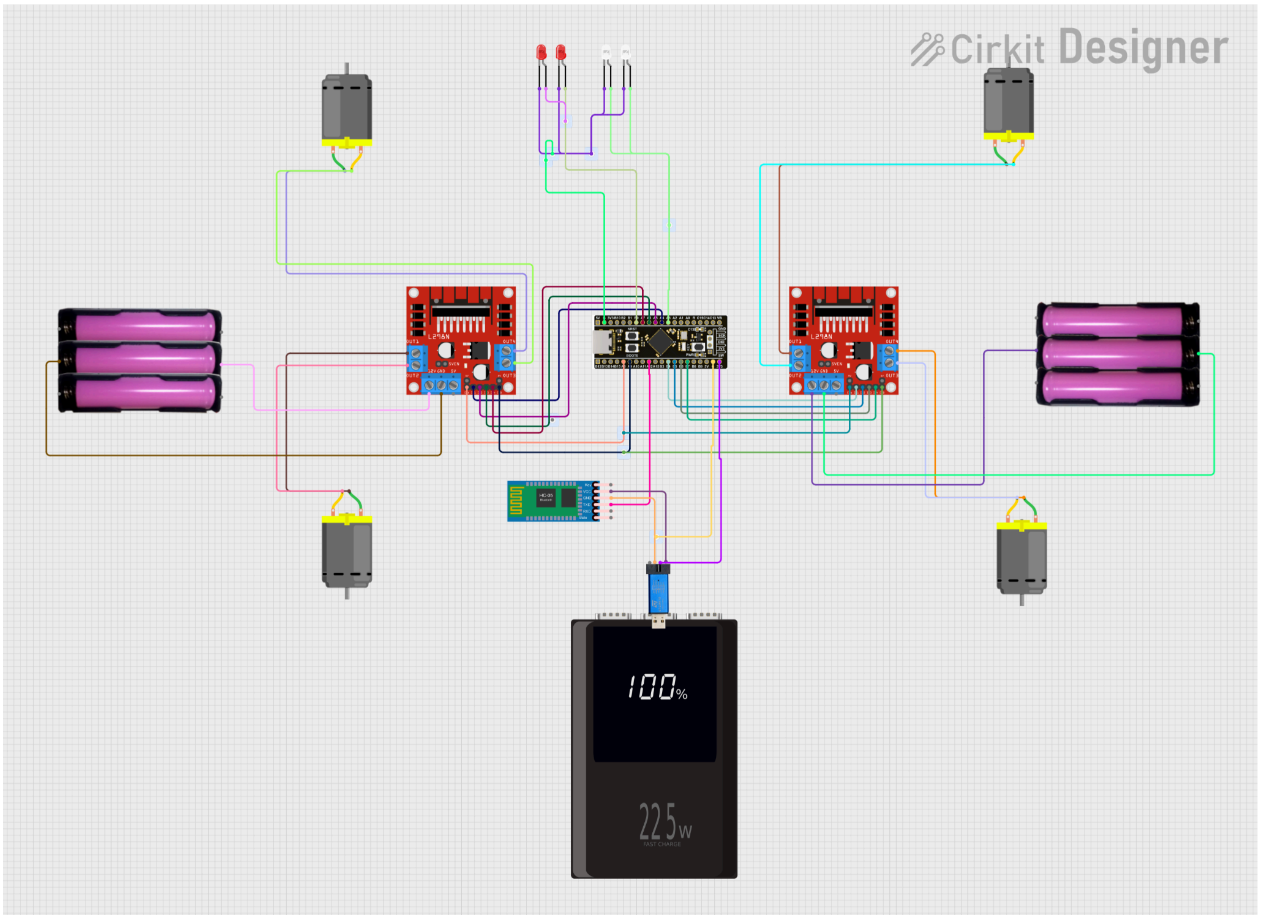
Additionally, we are using an Android application called UBRControl to send signals to Black Pill MCU via HC-05 and control the car.

C. Peripherals of STM Microcontroller being used [CLO-2, 50 Marks]

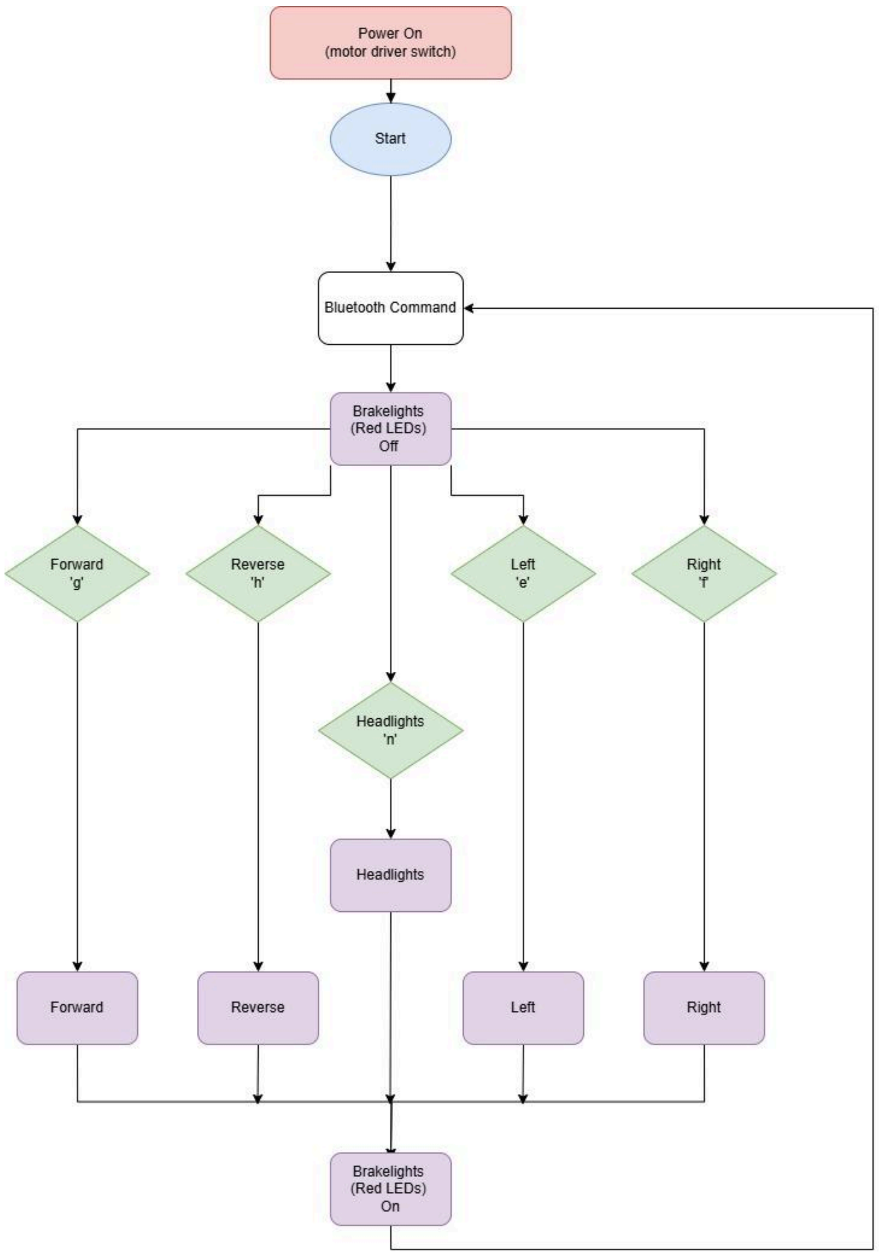
For the Bluetooth RC Car, following peripherals of the STM32 Black Pill were utilized:

- UART
- PWM
- PA4-PA9
- PA12
- PB4-PB7
- PA3
- PB0

D. Block Diagram/Schematic [CLO-3, 50 Marks]



E. Flow Chart (Required at the time of final submission) [CLO-3, 50 Marks]



F. CEP (Project Complexity) Attributes - Describe Briefly [CLO-3, 50 Marks]

Attribute	Description	Complexity Level in your project
WP1: Depth of knowledge	The project shall involve in-depth engineering knowledge related to the area of Microprocessors, Microcontrollers & Interfacing [WK-4, Engineering Specialization].	• Enabled us to set motor directions by controlling the input pins of the L298N motor driver. • Used hardware timers to generate PWM signals for controlling the speed of the DC motors via the motor driver's enable pins. • Applied knowledge of UART communication to interface the STM32 with the Bluetooth module for receiving control commands wirelessly.
WP2: Range of conflicting requirements	The project has multiple conflicting requirements in terms of optimal usage of peripheral resources available on a Microcontroller.	Some GPIO pins on the STM32 serve multiple functions (e.g., PWM, UART, I2C), making it challenging to assign pins without overlap. For example, PA9 and PA10 are used for UART (Bluetooth), but are also potential PWM outputs — requiring thoughtful peripheral planning.
WP5 Extent of applicable codes	The projects expose the students to broadly defined problems which require the development of codes that may be partially outside those encompassed by well-documented standards.	Our code is simple to understand as for the most part, we have if-else checks to call different functions that initiate the movement of the car in different directions.
WP7 Interdependence	The projects shall have multiple components at the hardware and software level.	The main challenge was managing limited GPIO pins and timers on the STM32 Black Pill. Controlling four motors and connecting the Bluetooth module required careful pin selection, as some pins had overlapping functions. Additionally, we had to figure out how Black Pill should be powered. We used powerbank and ST-link to power the Black Pill.

G. Code [CLO-3, 50 Marks]

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2025 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 */

/* USER CODE END PD */

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

/* Private variables ---------------------------------------------------------*/
TIM_HandleTypeDef htim1;

UART_HandleTypeDef huart6;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM1_Init(void);
static void MX_USART6_UART_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM1_Init();
  MX_USART6_UART_Init();
  HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
  HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
  uint8_t led_state = 0;
  while (1)
```

```
{
  uint8_t rx_data;
  if (HAL_UART_Receive(&huart6, &rx_data, 1, 100) == HAL_OK)
  {
   HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
    if (rx_data == 'g')
    {
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_SET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);
      __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, 1000);
      __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 1000);
      HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
    }
    else if (rx_data == 'h')
    {
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_SET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, GPIO_PIN_SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);
      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_SET);
      __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, 1000);
      __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 1000);
      HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
    }
    else if (rx_data == 'e')
    {
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, 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(GPIOB, GPIO_PIN_6, GPIO_PIN_SET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);
      __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, 1000);
      __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 1000);
      HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
    }
    else if (rx_data == 'f')
    {
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);
      __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, 1000);
      __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 1000);
      HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
    }
    else if (rx_data == 'n')
    {
      led_state = !led_state;
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, led_state ? GPIO_PIN_SET : GPIO_PIN_RESET);
    }
  }
  else
  {
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, 0);
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 0);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, 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(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
```

```
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
    }
  }
}
/**
  * @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_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_NONE;
  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_HSI;
  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_0) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief TIM1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM1_Init(void)
{

  /* USER CODE BEGIN TIM1_Init 0 */

  /* USER CODE END TIM1_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};
  TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};

  /* USER CODE BEGIN TIM1_Init 1 */

  /* USER CODE END TIM1_Init 1 */
  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 500;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 999;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
  if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
  sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
  if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
  {
    Error_Handler();
  }
  sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
  sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
  sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
  sBreakDeadTimeConfig.DeadTime = 0;
  sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
  sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
  sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
  if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM1_Init 2 */

  /* USER CODE END TIM1_Init 2 */
```

```
  HAL_TIM_MspPostInit(&htim1);

}

/**
  * @brief USART6 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART6_UART_Init(void)
{

  /* USER CODE BEGIN USART6_Init 0 */

  /* USER CODE END USART6_Init 0 */

  /* USER CODE BEGIN USART6_Init 1 */

  /* USER CODE END USART6_Init 1 */
  huart6.Instance = USART6;
  huart6.Init.BaudRate = 9600;
  huart6.Init.WordLength = UART_WORDLENGTH_8B;
  huart6.Init.StopBits = UART_STOPBITS_1;
  huart6.Init.Parity = UART_PARITY_NONE;
  huart6.Init.Mode = UART_MODE_TX_RX;
  huart6.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart6.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart6) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART6_Init 2 */

  /* USER CODE END USART6_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_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);

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

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6
                          |GPIO_PIN_7, GPIO_PIN_RESET);

  /*Configure GPIO pin : PC13 */
  GPIO_InitStruct.Pin = GPIO_PIN_13;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : PA3 PA4 PA5 PA6
                           PA7 */
  GPIO_InitStruct.Pin = GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6
                          |GPIO_PIN_7;
  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 : PB0 PB4 PB5 PB6
                           PB7 */
  GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6
                          |GPIO_PIN_7;
  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);

/* 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 */
```

H. References [Negative Marking of 20% if this section is skipped]

As the Bluetooth RC car project is common with Arduino MCU, we took help from Arduino-related videos too.

- Bluetooth RC Car With STM32F103C and L293D -- Inexpensive : 5 Steps - Instructables
- Arduino Bluetooth Control Car With Servo Steering
- Black Pill Datasheet