5. Code

```
/* 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 "stdio.h"
#include "string.h"
#include "stdlib.h"
/* 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 ---------------------------------------------------------
*/
ADC_HandleTypeDef hadc1;

TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim6;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------
*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM1_Init(void);
static void MX_TIM6_Init(void);
/* USER CODE BEGIN PFP */

// A utiliser après la partie démarrage des timers
void tourner();
void avancer();
void avancer_test(float avancement,int sens);
void set_servo_angle(float angle);

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------
*/
/* USER CODE BEGIN 0 */
#define NB_char 60
char msg[NB_char];
char msg2[NB_char];

#define SERVO_MIN_PULSE_WIDTH 670
#define SERVO_MAX_PULSE_WIDTH 4030
```

```
volatile uint16_t CCR;
volatile uint16_t Vbatt;
volatile uint16_t Tbatt;
volatile uint16_t distance;
volatile uint16_t angle_robot;

volatile uint16_t valeur_sonarF;
volatile uint16_t valeur_sonar;
volatile uint16_t compteur=0;

volatile uint16_t avancement=0;
volatile uint16_t temps=0;
volatile uint16_t start=0;
volatile uint16_t mesure_temps=0;


void set_servo_angle(float angle)
{
  uint16_t pulse_width = SERVO_MIN_PULSE_WIDTH + ((SERVO_MAX_PULSE_WIDTH -
SERVO_MIN_PULSE_WIDTH) * angle) / 180;
  __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4, pulse_width);
}


void tourner() //a utiliser apres la partie demarrage timers

{
  HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin,GPIO_PIN_RESET);
  HAL_GPIO_WritePin(DIRG_GPIO_Port, DIRG_Pin,GPIO_PIN_SET);
  CCR=16000;
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1,CCR);
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4,CCR);
}

void avancer() //a utiliser apres la partie demarrage timers

{
  HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin,GPIO_PIN_SET);
  HAL_GPIO_WritePin(DIRG_GPIO_Port, DIRG_Pin,GPIO_PIN_SET);
  CCR=10000;
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1,CCR);
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4,CCR);
}

void avancer_test(float avancement,int sens) //a utiliser apres la partie
demarrage timers
// temps en ms, avancement en cm, avancer : sens=1 sinon reculer : sens=-1
{
  CCR=10000;
```

```
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1,CCR);
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4,CCR);

  int temps = 5000 * avancement/100 ;

  if(sens==1){
    HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin,GPIO_PIN_SET);
    HAL_GPIO_WritePin(DIRG_GPIO_Port, DIRG_Pin,GPIO_PIN_SET);
  }
  else if (sens==-1){
    HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin,GPIO_PIN_RESET);
    HAL_GPIO_WritePin(DIRG_GPIO_Port, DIRG_Pin,GPIO_PIN_RESET);
  }

  HAL_Delay(temps);

  CCR=0;
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1,CCR);
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4,CCR);
}

/* 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_USART2_UART_Init();
  MX_ADC1_Init();
  MX_TIM2_Init();
  MX_TIM1_Init();
  MX_TIM6_Init();
  /* USER CODE BEGIN 2 */

  // démarrage des timers

  // Pour le SONAR
  HAL_TIM_Base_Start(&htim1);
  HAL_TIM_IC_Start_IT(&htim1, TIM_CHANNEL_2); // Reception
  HAL_GPIO_WritePin(Trig_sonar_GPIO_Port, Trig_sonar_Pin , GPIO_PIN_SET);
//Emission du SONAR
  HAL_TIM_Base_Start_IT(&htim2);
  HAL_TIM_Base_Start_IT(&htim6);

  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_4);
  HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_4);
  // fin de démarrage des timers

  //test Sonar
  //Calibrage du sonar
    /*for (int i=0;i<180;i++){
      set_servo_angle(i);
    }*/
  angle_robot=86; //regarder par rapport au capteur et non pas à la tête,
mettre sonar légèrement vers la droite dans le pire des cas
  set_servo_angle(angle_robot);
  HAL_Delay(1000);
  /* USER CODE END 2 */

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

  if(mesure_temps>=5){ //surveillance batterie toutes les 50 ms
    HAL_ADC_Start(&hadc1);
    HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
    Vbatt=HAL_ADC_GetValue(&hadc1);
    Tbatt=Vbatt*3300/4095;
```

```
    if(Vbatt==4095){
      sprintf(msg,"Tension de la batterie : au moins %d mV\r\n",Tbatt);
      HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    }
    else {
      sprintf(msg,"Tension de la batterie : %d mV\r\n",Tbatt);
      HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    }

    mesure_temps=0;
  }

    //test moteur robot
    /*avancer_test(100, 1);  //avancer d'1m en 5s puis attendre 5s
    HAL_Delay(5000);*/

    if(start){

    if (20*101<valeur_sonar && valeur_sonar<=150*101 && compteur==0){
      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1,0);
      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4,0);
      compteur++;
    }

    if (compteur>0){
      avancer();
      compteur=0;
      sprintf(msg,"Vitesse du robot : 20 cm/s  \r\n");
      HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    }

    else if (valeur_sonar>150*101 && compteur==0) {
      tourner();
        compteur=0;
        sprintf(msg,"Robot en rotation (sens horaire) \r\n");
        HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    }
    else if (valeur_sonar<=20*101 && compteur==0){
      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1,0);
      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4,0);
      HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin,GPIO_PIN_SET);
      HAL_GPIO_WritePin(DIRG_GPIO_Port, DIRG_Pin,GPIO_PIN_SET);
      compteur=0;
      sprintf(msg,"Vitesse du robot : 0 cm/s \r\n");
        HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    }
    }

    else{
```

```
      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1,0);
      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4,0);
      HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin,GPIO_PIN_SET);
      HAL_GPIO_WritePin(DIRG_GPIO_Port, DIRG_Pin,GPIO_PIN_SET);
      sprintf(msg,"Robot à l'arrêt\r\n");
      HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    }
    /* 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
  */
  if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
  {
    Error_Handler();
  }

  /** 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 = 1;
  RCC_OscInitStruct.PLL.PLLN = 10;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV7;
  RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
  RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
  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_4) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief ADC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_MultiModeTypeDef multimode = {0};
  ADC_AnalogWDGConfTypeDef AnalogWDGConfig = {0};
  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */

  /** Common config
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  hadc1.Init.LowPowerAutoWait = DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.DMAContinuousRequests = DISABLE;
```


```
  hadc1.Init.Overrun = ADC_OVR_DATA_PRESERVED;
  hadc1.Init.OversamplingMode = DISABLE;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure the ADC multi-mode
  */
  multimode.Mode = ADC_MODE_INDEPENDENT;
  if (HAL_ADCEx_MultiModeConfigChannel(&hadc1, &multimode) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Analog WatchDog 1
  */
  AnalogWDGConfig.WatchdogNumber = ADC_ANALOGWATCHDOG_1;
  AnalogWDGConfig.WatchdogMode = ADC_ANALOGWATCHDOG_SINGLE_REG;
  AnalogWDGConfig.Channel = ADC_CHANNEL_14;
  AnalogWDGConfig.ITMode = ENABLE;
  AnalogWDGConfig.HighThreshold = 4095;
  AnalogWDGConfig.LowThreshold = 3723;
  if (HAL_ADC_AnalogWDGConfig(&hadc1, &AnalogWDGConfig) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_14;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_640CYCLES_5;
  sConfig.SingleDiff = ADC_SINGLE_ENDED;
  sConfig.OffsetNumber = ADC_OFFSET_NONE;
  sConfig.Offset = 0;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}

/**
  * @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_SlaveConfigTypeDef sSlaveConfig = {0};
  TIM_IC_InitTypeDef sConfigIC = {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 = 46-1;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 0xFFFF;
  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();
  }
  if (HAL_TIM_IC_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  sSlaveConfig.SlaveMode = TIM_SLAVEMODE_RESET;
  sSlaveConfig.InputTrigger = TIM_TS_TI1FP1;
  sSlaveConfig.TriggerPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
  sSlaveConfig.TriggerPrescaler = TIM_ICPSC_DIV1;
  sSlaveConfig.TriggerFilter = 5;
  if (HAL_TIM_SlaveConfigSynchro(&htim1, &sSlaveConfig) != HAL_OK)
```


```
{
  Error_Handler();
}
sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
sConfigIC.ICFilter = 5;
if (HAL_TIM_IC_ConfigChannel(&htim1, &sConfigIC, TIM_CHANNEL_1) != HAL_OK)
{
  Error_Handler();
}
sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_FALLING;
sConfigIC.ICSelection = TIM_ICSELECTION_INDIRECTTI;
if (HAL_TIM_IC_ConfigChannel(&htim1, &sConfigIC, TIM_CHANNEL_2) != HAL_OK)
{
  Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterOutputTrigger2 = TIM_TRGO2_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.OCFastMode = TIM_OCFAST_DISABLE;
sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_4) != 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.BreakFilter = 0;
sBreakDeadTimeConfig.Break2State = TIM_BREAK2_DISABLE;
sBreakDeadTimeConfig.Break2Polarity = TIM_BREAK2POLARITY_HIGH;
sBreakDeadTimeConfig.Break2Filter = 0;
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 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_OC_InitTypeDef sConfigOC = {0};

  /* USER CODE BEGIN TIM2_Init 1 */

  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 1-1;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 16000;
  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_PWM_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();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_4) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_Init 2 */
  HAL_TIM_MspPostInit(&htim2);

}

/**
  * @brief TIM6 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM6_Init(void)
{

  /* USER CODE BEGIN TIM6_Init 0 */

  /* USER CODE END TIM6_Init 0 */

  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM6_Init 1 */

  /* USER CODE END TIM6_Init 1 */
  htim6.Instance = TIM6;
  htim6.Init.Prescaler = 13-1;
  htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim6.Init.Period = 61538;
  htim6.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim6) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
```


```
  if (HAL_TIMEx_MasterConfigSynchronization(&htim6, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM6_Init 2 */

  /* USER CODE END TIM6_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_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_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, DIRG_Pin|Trig_sonar_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin, GPIO_PIN_RESET);

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

  /*Configure GPIO pin : LD2_Pin */
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : DIRG_Pin Trig_sonar_Pin */
  GPIO_InitStruct.Pin = DIRG_Pin|Trig_sonar_Pin;
  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 : DIRD_Pin */
  GPIO_InitStruct.Pin = DIRD_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(DIRD_GPIO_Port, &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 */
void HAL_ADC_LevelOutOfWindowCallback(ADC_HandleTypeDef *hadc)
{
  /* Prevent unused argument(s) compilation warning */
  if(hadc->Instance==ADC1){
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
  }
}


void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
  if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2){
      valeur_sonar = (uint16_t) HAL_TIM_ReadCapturedValue(&htim1,
TIM_CHANNEL_2);
      distance =  (uint16_t) valeur_sonar/101;
  }
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if (htim->Instance == TIM6){
    mesure_temps; //TIM6 de période 10 ms
  }
}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  if (GPIO_Pin == B1_Pin){
    start=!start;
  }
}

/* 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 */
```