LAB: Line Tracing RC Car

Date: 2023-11-24

Author/Partner: NohYunKi / Moon Hyron ho

Demo Video: https://youtu.be/ao4916Agozc // please choose 4k

Introduction

In this lab, we made RC car using several sensors and communication device. There are two types of problem. First is Manual mode.

Requirement

Hardware

- MCU
  - NUCLEO-F411RE
- Actuator/Sensor/Others: Minimum
  - Bluetooth Module(HC-06)
  - DC motor x2, DC motor driver(L9110s)
  - IR Reflective Sensor (TCRT 5000) x2
  - HC-SR04

Software

- Keil uVision, CMSIS, EC_HAL library

Problem

1. Manual Mode

- Speed/Angles/Direction/Obstacle detecting and Displaying state
  - speed:
    1. When pressing 'up' key and 'down' key(customized key), four types of speed states(V0<V1<V2<V3) must be displayed using bluetooth communication.
  - steer:
    1. When pressing 'right', 'left' key(customized key), two types of direction informations must be displayed using bluetooth.
    2. Also, steering angle must be three levels. (-3, -2, -1, 0, 1 , 2, 3 // From left to right). This level informations must be displayed.
  - driving direction:
    1. Driving direction is two types. When pressing 'F' key, car will go straight and when pressing 'B' key, car will go back.
    2. These direction informations also must be displayed.

2. Automatic Mode

- LED blinking/Line tracing/Obstacle detecting and Displaying state
    - LED:
    1. When choosing AUTO MODE, car's LD2 must be blinked per 1 second.
    - Line tracing:
    1. Using IR sensor, RC car must go forward tracing black line.
    - Obstacle detecting
        1. Using Ultra Sonic sensor when detecting obstacle from the front side, RC car must stop. In addition to this, when obstacle is removed from front side, RC car must go forward again.

These two types of modes must be changed when entering ''Manual Mode' key and 'Auto Mode' key. These key also are chosen.

Configuration

Functions	Register	PORT_PIN	Configuration
System Clock	RCC		PLL 84MHz
delay_ms	SysTick		PLL 84MHz
Motor DIR	Digital Out	PC2, PC3	OUTPUT
TIMER	TIMER2		1 msec period
	TIMER3		50 msec period
Timer Interrupt	TIMER4		50 msec PWM period 10 usec PWM pulse width
ADC	ADC		50 msec
Ultra Sonic_PWM	PWM3	PA6	50 msec
Ultra Sonic_ECHO	TIMER4	PB6	10 usec
DC Motor Speed	PWM2	PA0, PA1	1 msec
ADC sampling trigger	PWM3	PB0, PB1	50 msec
RS-232 USB cable(ST-LINK)	USART2		No Parity, 8-bit Data, 1-bit Stop bit 38400 baud-rate
Bluetooth	USART1	TXD: PA9 RXD: PA10	No Parity, 8-bit Data, 1-bit Stop bit 9600 baud-rate

Circuit Diagram

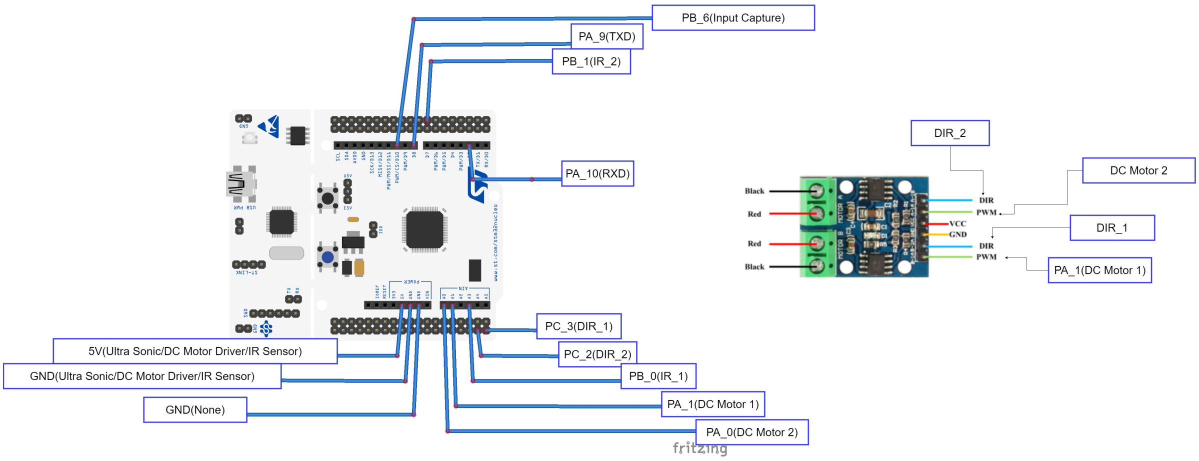

Figure 1. RC car circuit diagram

Flow Chart

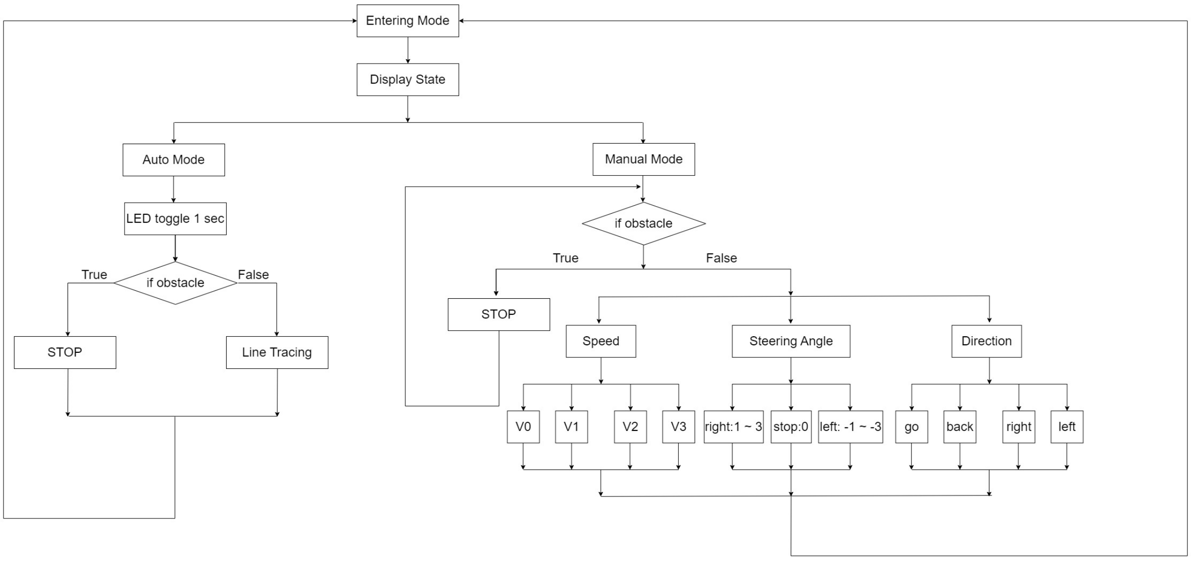

Figure 2. Flow Chart

Code

Entire Code

```
/*
******************************************************************************
* @author   Noh Yun Ki
* @Mod      2023/11/24
* @brief    Embedded Controller:  LAB_RCcar
*
******************************************************************************
*/

#include "ecSTM32F411.h"

// State Signal
#define GO               0x49  //I
#define BACK           0x4B  //K
#define RIGHT         0x4C  //L
```

```
#define LEFT          0x4A  //J
#define STOP           0x53  //S
#define FORWARD        0x46  //F
#define BACKWORD     0x42  //B
#define REST            0x52  //R
#define Control         0x4D  //M
#define AUTO            0x41  //A

// Ultra Sonic
#define TRIG PA_6
#define ECHO PB_6

// Auto Mode display signal
#define auto_stop       0
#define auto_up   1
#define auto_right   2
#define auto_left   3

static volatile int speed = 0; // speed value
static volatile int direc = 1; // Forward/Backward
static volatile int STR = 3; // Steering
static volatile int MODE = 0; // Auto/Manual

static volatile int cnt = 0; // Counting variable for blink LED
uint32_t ovf_cnt = 0; // Ultra Sonic Overflow Count
uint32_t auto_cnt = 0; // Auto Mode Display periond
static volatile float distance = 0; // Ultra Sonic Distnace
float timeInterval = 0; // Ultra Sonic timeInterval
float time1 = 0; // Rising capture
float time2 = 0; // Falling capture
int display_auto = 0; // Counting variable for Auto Display

uint8_t btData = 0;

int flag = 0;
PinName_t seqCHn[2] = {PB_0, PB_1};

static volatile uint32_t right_IR, left_IR;

void MCU_init(void);
void Direction_display(uint8_t direction);
void Display_MState(void);
void Display_AState(void);
void Control_mode(void);
void Auto_mode(void);
void USART1_IRQHandler(void);
void TIM4_IRQHandler(void);
void ADC_IRQHandler(void);



int main(void) {
   // Initialiization --------------------------------------------------------
   MCU_init();
   // Inifinite Loop ----------------------------------------------------------
   while (1){
       //Auto_Display_State(MODE, drirec)


       //Manual_Set_State(MODE, direc, drc, speed);
        if(btData == Control){
            MODE=1;
            speed = 0;
            STR = 3;
            GPIO_write(GPIOC, 2, 1);
            GPIO_write(GPIOC, 3, 1);
            PWM_duty(PA_0, (float)(1));
            PWM_duty(PA_1, (float)(1));
            GPIO_write(GPIOA, 5, 1);
         }else if(btData == AUTO){
            MODE=2;
            speed = 0;
            STR = 3;
            GPIO_write(GPIOC, 2, 0);
            GPIO_write(GPIOC, 3, 0);
         }else if(btData == REST){
            MODE=0;
            speed = 0;
            STR = 3;
            GPIO_write(GPIOC, 2, 1);
            GPIO_write(GPIOC, 3, 1);
            PWM_duty(PA_0, (float)(1));
            PWM_duty(PA_1, (float)(1));
            GPIO_write(GPIOA, 5, 0);
         }


         if(MODE ==1){
            Control_mode();
            Display_MState();

         }else if(MODE ==2){
            Auto_mode();
            Display_AState();

         }else if(MODE ==0){
            GPIO_write(GPIOC, 2, 1);
            GPIO_write(GPIOC, 3, 1);
            PWM_duty(PA_0, (float)(1));
            PWM_duty(PA_1, (float)(1));
         }


   }
}


void USART1_IRQHandler(){         //USART1 INT
   if(is_USART_RXNE(USART1)){
     Direction_display(btData);

   }
```

```
}

void TIM4_IRQHandler(){
   if(is_UIF(TIM4)){                     // Update interrupt

      distance = timeInterval * ((float)(340.0 / 2.0 / 10.0));   // [mm] -> [cm]

      ovf_cnt++;         // overflow count
      cnt++;
      auto_cnt++;

      clear_UIF(TIM4);                           // clear update interrupt flag
   }

   if(is_CCIF(TIM4, 1)){                         // TIM4_Ch1 (IC1) Capture Flag.
Rising Edge Detect
      time1 = TIM4 -> CCR1;                           // Capture TimeStart
      clear_CCIF(TIM4, 1);                // clear capture/compare interrupt flag
   }
   else if(is_CCIF(TIM4, 2)){                            // TIM4_Ch2 (IC2)
Capture Flag. Falling Edge Detect
      time2 = TIM4 -> CCR2;                           // Capture TimeEnd
      timeInterval = ((time2 - time1) + ovf_cnt * (TIM4->ARR + 1))/100;    //
(10us * counter pulse -> [msec] unit) Total time of echo pulse
      ovf_cnt = 0;                        // overflow reset
      clear_CCIF(TIM4,2);                          // clear capture/compare
interrupt flag
   }
}

void ADC_IRQHandler(void){
   if(is_ADC_OVR())
      clear_ADC_OVR();

   if(is_ADC_EOC()){      // after finishing sequence
      if (flag==1)
         right_IR = ADC_read();
      else if (flag==0)
         left_IR = ADC_read();

      flag =! flag;      // flag toggle
   }
}



//Print the current direction state based on the pressed key
void Direction_display(uint8_t direction){
     btData = USART_read(USART1);

   if(btData == GO){
      USART_write(USART1, (uint8_t*) "SPEED UP", 8);
   }
    else if(btData == BACK){
      USART_write(USART1, (uint8_t*) "SPEED DOWN", 10);
   }
   else if(btData == RIGHT){
      USART_write(USART1, (uint8_t*)"TURN RIGHT", 10);
   }
   else if(btData == LEFT){
      USART_write(USART1, (uint8_t*) "TURN LEFT", 9);
   }
   else if(btData == STOP){
      USART_write(USART1, (uint8_t*) "STOP", 4);
   }
    else if(btData == FORWARD){
      USART_write(USART1, (uint8_t*) "GO", 2);
   }
    else if(btData == BACKWORD){
      USART_write(USART1, (uint8_t*) "BACK ", 4);
   }
    else if(btData == Control){
          USART_write(USART1, (uint8_t*) "CONTROL MODE", 12);
    }
    else if(btData == AUTO){
          USART_write(USART1, (uint8_t*) "LINE TRACE MODE", 15);
       }
    else if(btData == REST){
          USART_write(USART1, (uint8_t*) "Select a Mode", 13);
       }
   USART_write(USART1, "\r\n", 2);
}



void Control_mode(void){

               if(btData == FORWARD){
                     direc=1;
               }else if(btData == BACKWORD){
                     direc=0;
               }else if(btData == GO){
                     speed++;
                     if(speed>3){
                        speed=3;
                     }
               }else if(btData == BACK){
                     speed--;
                     if(speed<0){
                           speed=0;
                     }
               }else if(btData == LEFT){
                     STR--;
                     if(STR<0){
                           STR=0;
                     }
               }else if(btData == RIGHT){
                     STR++;
                     if(STR>6){
                           STR=6;
                     }
               }else if(btData == STOP){
                     direc=1;
                     speed=0;
                     STR=3;
               }

               btData =0x35; // clear btData
               GPIO_write(GPIOC, 2, direc);  //Direction out=1 CW Direction out=0 CCW
               GPIO_write(GPIOC, 3, direc);

               if(direc == 1){
              if(speed == 0 && STR == 3 || distance < 15){
                    PWM_duty(PA_0, (float)(1));
                    PWM_duty(PA_1, (float)(1));
              }else if(speed==1 && STR==3){
                    PWM_duty(PA_0, (float)(1*0.7));
                    PWM_duty(PA_1, (float)(1*0.7));
              }else if(speed==2 && STR==3){
                    PWM_duty(PA_0, (float)(1*0.4));
                    PWM_duty(PA_1, (float)(1*0.4));
              }else if(speed==3 && STR==3){
                    PWM_duty(PA_0, (float)(1*0.1));
                    PWM_duty(PA_1, (float)(1*0.1));
             }else if(STR==2){                    //LEFT
                    PWM_duty(PA_0, (float)(1*0.6));
                    PWM_duty(PA_1, (float)(1*0.4));
             }else if(STR==1){
                   PWM_duty(PA_0, (float)(1*0.7));
                    PWM_duty(PA_1, (float)(1*0.4));
             }else if(STR==0){
                   PWM_duty(PA_0, (float)(1*0.9));
                   PWM_duty(PA_1, (float)(1*0.4));
             }else if(STR==4){                   //RIGHT
                    PWM_duty(PA_0, (float)(1*0.4));
                   PWM_duty(PA_1, (float)(1*0.6));
             }else if(STR==5){
                    PWM_duty(PA_0, (float)(1*0.4));
                    PWM_duty(PA_1, (float)(1*0.7));
              }else if(STR==6){
                    PWM_duty(PA_0, (float)(1*0.4));
                    PWM_duty(PA_1, (float)(1*0.9));
                    }
        }else if(direc == 0){
              if(speed==0 && STR==3 || distance<15){
                    PWM_duty(PA_0, (float)(0));
                     PWM_duty(PA_1, (float)(0));
              }else if(speed==1 && STR==3){
                    PWM_duty(PA_0, (float)(1*0.3));
                   PWM_duty(PA_1, (float)(1*0.3));
              }else if(speed==2 && STR==3){
                    PWM_duty(PA_0, (float)(1*0.6));
                     PWM_duty(PA_1, (float)(1*0.6));
              }else if(speed==3 && STR==3){
                     PWM_duty(PA_0, (float)(1*0.9));
                    PWM_duty(PA_1, (float)(1*0.9));
              }else if(STR==2){                    //LEFT
```

```
                     PWM_duty(PA_0, (float)(1*0.4));
                     PWM_duty(PA_1, (float)(1*0.6));
               }else if(STR==1){
                     PWM_duty(PA_0, (float)(1*0.4));
                     PWM_duty(PA_1, (float)(1*0.7));
              }else if(STR==0){
                      PWM_duty(PA_0, (float)(1*0.4));
                     PWM_duty(PA_1, (float)(1*0.9));
              }else if(STR==4){                   //RIGHT
                      PWM_duty(PA_0, (float)(1*0.4));
                      PWM_duty(PA_1, (float)(1*0.4));
              }else if(STR==5){
                     PWM_duty(PA_0, (float)(1*0.7));
                      PWM_duty(PA_1, (float)(1*0.4));
               }else if(STR==6){
                      PWM_duty(PA_0, (float)(1*0.9));
                      PWM_duty(PA_1, (float)(1*0.4));
               }
         }
}


void Auto_mode(void){

   // Stop
        if(distance < 15){
            PWM_duty(PA_0, (float)0);
            PWM_duty(PA_1, (float)0);
                  display_auto = auto_stop;
            }
            else if(distance > 15){
               if(right_IR < 1300 && left_IR < 1300){
            PWM_duty(PA_0,(float)1); // 80%
            PWM_duty(PA_1,(float)1); // 80%
                 display_auto = auto_up;
               }
                     // Right
        else if(right_IR > 1300 && left_IR < 1300){
            PWM_duty(PA_0,(float)0.2); // 50%
            PWM_duty(PA_1,(float)1);  // 80%
                  display_auto = auto_right;
               }
                           // Left
        else if(right_IR < 1300 && left_IR > 1300){
            PWM_duty(PA_0,(float)1);  // 80%
            PWM_duty(PA_1,(float)0.2);   // 50%
                  display_auto = auto_left;
               }
            }
}

void Display_MState(void){
   if(auto_cnt > 1){
         USART_write(USART1, (uint8_t*)"MOD: M ", 7);
         if(direc == 1){
               USART_write(USART1, (uint8_t*)"DIR: F ", 7);
         }else if(direc == 0){
               USART_write(USART1, (uint8_t*)"DIR: B ", 7);
         }
         if(STR == 0){
                  USART_write(USART1, (uint8_t*)"STR: -3 ", 8);
                  USART_write(USART1, (uint8_t*)"VEL: V3", 7);
         }else if(STR == 1){
                  USART_write(USART1, (uint8_t*)"STR: -2 ", 8);
                  USART_write(USART1, (uint8_t*)"VEL: V2", 7);
         }else if(STR == 2){
                  USART_write(USART1, (uint8_t*)"STR: -1 ", 8);
                  USART_write(USART1, (uint8_t*)"VEL: V1", 7);
         }else if(STR == 3){
                  USART_write(USART1, (uint8_t*)"STR: 0 ", 7);
         }else if(STR == 4){
                  USART_write(USART1, (uint8_t*)"STR: 1 ", 7);
                  USART_write(USART1, (uint8_t*)"VEL: V1", 7);
         }else if(STR == 5){
                  USART_write(USART1, (uint8_t*)"STR: 2 ", 7);
                  USART_write(USART1, (uint8_t*)"VEL: V2", 7);
         }else if(STR == 6){
                  USART_write(USART1, (uint8_t*)"STR: 3 ", 7);
                  USART_write(USART1, (uint8_t*)"VEL: V3", 7);
         }
         if(speed == 0 && STR == 3){
               USART_write(USART1, (uint8_t*)"VEL: V0", 7);
         }else if(speed == 1 && STR == 3){
               USART_write(USART1, (uint8_t*)"VEL: V1", 7);
         }else if(speed == 2 && STR == 3){
               USART_write(USART1, (uint8_t*)"VEL: V2", 7);
         }else if(speed == 3 && STR == 3){
               USART_write(USART1, (uint8_t*)"VEL: V3", 7);
         }
         USART_write(USART1, "\r\n", 2);
         auto_cnt = 0;
  }
}

void Display_AState(void){


         if(cnt > 1){
         if(MODE==2){
            LED_toggle();


         }
            cnt=0;
      }
   if(auto_cnt > 1){
       USART_write(USART1, (uint8_t*) "MODE : Line Tracer   ", 21);
      // STOP
      if(display_auto == auto_stop) USART_write(USART1, (uint8_t*) "STOP", 4);
      // UP
      else if(display_auto == auto_up) USART_write(USART1, (uint8_t*) "GO", 2);
      // RIGHT
      else if(display_auto == auto_right) USART_write(USART1, (uint8_t*)"TURN RIGHT", 10);
      // LEFT
      else if(display_auto == auto_left)USART_write(USART1, (uint8_t*) "TURN
LEFT", 9);
      USART_write(USART1, "\r\n", 2);
      auto_cnt = 0;
  }
}


// Initialiization
void MCU_init(void){
   RCC_PLL_init();
   SysTick_init();

  // BT serial init
   UART1_init();   // PA9 - TXD , PA10 - RXD
   UART1_baud(BAUD_9600);
   UART2_init();
   UART2_baud(BAUD_38400);

          // ADC Init
   ADC_init(PB_0);
   ADC_init(PB_1);

    // ADC channel sequence setting
   ADC_sequence(seqCHn, 2);


   GPIO_init(GPIOA, LED_PIN, OUTPUT);    // calls RCC_GPIOA_enable()
   GPIO_otype(GPIOA, LED_PIN, 0);        //  GPIOA LED_PIN Output Type: Push-Pull
   GPIO_ospeed(GPIOA, LED_PIN, EC_FAST);      // GPIOA Speed LED_PIN Fast

   // PWM of 1 msec:  TIM2_CH1 (PA_0 AFmode)
   PWM_init(PA_0);
   GPIO_otype(GPIOA, 0, EC_PUSH_PULL);     //PWM PA_0 Push-Pull
   GPIO_pupd(GPIOA, 0, EC_NONE);           //PWM Pin No pull-up, pull-down
   GPIO_ospeed(GPIOA, 0, EC_FAST);         //PWM Pin Fast
   PWM_period(PA_0, 1);

   // PWM of 1 msec:  TIM2_CH2 (PA_1 AFmode)
   PWM_init(PA_1);
   GPIO_otype(GPIOA, 1, EC_PUSH_PULL);     //PWM Pin Push-Pull
   GPIO_pupd(GPIOA, 1, EC_NONE);           //PWM Pin No pull-up, pull-down
   GPIO_ospeed(GPIOA, 1, EC_FAST);         //PWM Pin Fast
   PWM_period(PA_1, 1);

   //Moter Direction Pin   PC_2
   PWM_init(PC_2);
   GPIO_init(GPIOC, 2, OUTPUT);


   //Moter Direction Pin  PC_3
   PWM_init(PC_3);
   GPIO_init(GPIOC, 3, OUTPUT);

   // PWM:  TIM4_CH2 (PA_6 AFmode)
   PWM_init(TRIG);
   GPIO_otype(GPIOA, 6, EC_PUSH_PULL);   //PWM Pin Push-Pull
   GPIO_pupd(GPIOA, 6, EC_NONE);         //PWM Pin NO Pull-up, Pull-down
   GPIO_ospeed(GPIOA, 6, EC_FAST);       //PWM Pin Fast
   PWM_period_us(TRIG, 50000);        // 50 msec PWM period
   PWM_pulsewidth_us(TRIG,10);        // 10 usec PWM pulse width

   // input Capture:  TIM4_CH1 (PB_6 AFmode)
   ICAP_init(ECHO);
   GPIO_pupd(GPIOB, 6, EC_NONE);         //PWM Pin NO Pull-up, Pull-down
   ICAP_counter_us(PB_6, 10);            //Counter Clock : 0.1MHz (10us)
   ICAP_setup(PB_6, 1, IC_RISE);         //TI4 -> IC1 (rising edge)
   ICAP_setup(PB_6, 2, IC_FALL);         //TI4 -> IC2 (falling edge)

}
```

Setting: 'void MCU_init(void)'

To use bluetooth connection, we set USART1. And to use 'print' function, we set USART2.

```
// BT serial init
   UART1_init();   // PA9 - TXD , PA10 - RXD
   UART1_baud(BAUD_9600);
   UART2_init();
   UART2_baud(BAUD_9600);
```

In this code, we set IR environment.

```
 // ADC Init
   ADC_init(PB_0);
   ADC_init(PB_1);
```

LED setting

```
GPIO_init(GPIOA, LED_PIN, OUTPUT);    // calls RCC_GPIOA_enable()
   GPIO_otype(GPIOA, LED_PIN, 0);        //  GPIOA LED_PIN Output Type: Output open drain (1)
   GPIO_ospeed(GPIOA, LED_PIN, EC_FAST);      // GPIOA Speed LED_PIN Medium speed(01)
```

DC Motor

```
// PWM of 1 msec:  TIM2_CH1 (PA_0 AFmode)
   PWM_init(PA_0);
   GPIO_otype(GPIOA, 0, EC_PUSH_PULL);     //PWM PA_0 Push-Pull
   GPIO_pupd(GPIOA, 0, EC_NONE);           //PWM Pin No pull-up, pull-down
   GPIO_ospeed(GPIOA, 0, EC_FAST);         //PWM Pin Fast
   PWM_period(PA_0, 1);

   // PWM of 1 msec:  TIM2_CH2 (PA_1 AFmode)
   PWM_init(PA_1);
   GPIO_otype(GPIOA, 1, EC_PUSH_PULL);     //PWM Pin Push-Pull
   GPIO_pupd(GPIOA, 1, EC_NONE);           //PWM Pin No pull-up, pull-down
   GPIO_ospeed(GPIOA, 1, EC_FAST);         //PWM Pin Fast
```

DC Motor Driver

```
//Motor Direction Pin   PC_2
   PWM_init(PC_2);
   GPIO_init(GPIOC, 2, OUTPUT);



   //Motor Direction Pin  PC_3
   PWM_init(PC_3);
   GPIO_init(GPIOC, 3, OUTPUT);
```

Ultra Sonic_Trig Echo

```
// PWM:  TIM4_CH2 (PA_6 AFmode)
// Ultra Sonic_Trig
   PWM_init(TRIG);
   GPIO_otype(GPIOA, 6, EC_PUSH_PULL);   //PWM Pin Push-Pull
   GPIO_pupd(GPIOA, 6, EC_NONE);         //PWM Pin NO Pull-up, Pull-down
   GPIO_ospeed(GPIOA, 6, EC_FAST);       //PWM Pin Fast
   PWM_period_us(TRIG, 50000);        // 50 msec PWM period
   PWM_pulsewidth_us(TRIG,10);        // 10 usec PWM pulse width
   // Ultra Sonic_Echo
   // input Capture:  TIM4_CH1 (PB_6 AFmode)
   ICAP_init(ECHO);
   GPIO_pupd(GPIOB, 6, EC_NONE);         //PWM Pin NO Pull-up, Pull-down
   ICAP_counter_us(PB_6, 10);            //Counter Clock : 0.1MHz (10us)
   ICAP_setup(PB_6, 1, IC_RISE);         //TI4 -> IC1 (rising edge)
   ICAP_setup(PB_6, 2, IC_FALL);         //TI4 -> IC2 (falling edge)
```

TIMER Interrupt: 'void TIM4_IRQHandler()'

We set codes for distance estimation & change direction of Auto mode RC car and LED toggle per 1 second, we use TIMER 4 Interrupt.

- Dircetion

This is direction code for Auto mode. By checking whether right IR value and left IR value are under 1300 or not, we give direction signal.

For example when both side of IR values are under 1300, RC car will go forward.

```
else if(right_IR < 1300 && left_IR < 1300){
            USART_write(USART1, (uint8_t*) "Forward", 7);
               }
```

When Right IR value is over 1300 and Left IR value is under 1300, RC car will turn right.

```
 else if(right_IR > 1300 && left_IR < 1300){
            USART_write(USART1, (uint8_t*) "Turn Right", 10);
               }
```

On the contrary, when right value is under 1300 and left value is over 1300, RC car will turn left.

```
else if(right_IR < 1300 && left_IR > 1300){
            USART_write(USART1, (uint8_t*) "Turn Left", 9);
               }
```

- LED toggle

Because LED should be blinked per 1 second, we set LED toggle function not in while loop, but in TIMER 4 Interrupt.

```
void TIM4_IRQHandler(){
   if(is_UIF(TIM4)){
       ...
      cnt++;
       ...
      if(cnt>1){
         if(MODE==2){
            LED_toggle();
         }
       ...
    }
       cnt=0;
  }
 clear_UIF(TIM4);
 }
```

- Ultra Sonic(Distance)

Count CCR value when Rising signal arises.

```
if(is_CCIF(TIM4, 1)){                         // TIM4_Ch1 (IC1) Capture Flag.
Rising Edge Detect
      time1 = TIM4 -> CCR1;                           // Capture TimeStart
      clear_CCIF(TIM4, 1);                // clear capture/compare interrupt flag
   }
```

Count CCR value when Falling signal arises.

```
else if(is_CCIF(TIM4, 2)){                            // TIM3_Ch1 (IC2) Capture
Flag. Falling Edge Detect
      time2 = TIM4 -> CCR2;                           // Capture TimeEnd
      timeInterval = ((time2 - time1) + ovf_cnt * (TIM4->ARR + 1))/100;    //
(10us * counter pulse -> [msec] unit) Total time of echo pulse
      ovf_cnt = 0;                        // overflow reset
      clear_CCIF(TIM4,2);                          // clear capture/compare
interrupt flag
   }
```

Calculate distance value.

```
distance = timeInterval * ((float)(340.0 / 2.0 / 10.0));   // [mm] -> [cm]
```

Manual Mode: 'void Control_mode(void)'

We make 'void Control_mode(void)' for Manual Mode. By using USART 1, we give signal which determine the state of RC car.

- Forward/Backward: By giving 'FORWARD',' 'BACKWARD' signal, we determine the direction of RC car.

```
if(btData == FORWARD){
                     direc=1;
               }else if(btData == BACKWORD){
                     direc=0;
               }
```

- Speed: By giving 'GO',' 'BACK' signal, we determine the speed of RC car.

```
else if(btData == GO){
                     speed++;
                     if(speed>3){
                        speed=3;
                     }
               }else if(btData == BACK){
                     speed--;
                     if(speed<0){
                           speed=0;
                     }
```

- Right/Left and Angle: By giving 'LEFT',' 'RIGHT' signal, we determine the direction of RC car. In addition to this, when entering right of left signal more, RC car's rotating angle is also changed('STR').

```
else if(btData == LEFT){
                     STR--;
                     if(STR<0){
                           STR=0;
                     }
               }else if(btData == RIGHT){
                     STR++;
                     if(STR>6){
                           STR=6;
                     }
```

- Stop: By giving 'STOP' signal, we can make RC car puased.

```
else if(btData == STOP){
                     direc=1;
                     speed=0;
                     STR=3;
               }
```

By determine RC car's state signal, we can actually change RC car's movement like this.

```
 if(direc == 1){
              if(speed == 0 && STR == 3 || distance < 15){
                    PWM_duty(PA_0, (float)(1));
                    PWM_duty(PA_1, (float)(1));
                    ...
}
}else if(direc == 0){
              if(speed==0 && STR==3 || distance<15){
                    PWM_duty(PA_0, (float)(0));
                     PWM_duty(PA_1, (float)(0));
                    ...
}
```

Auto Mode: 'void Auto_mode(void)'

At first, when obstacle detected, RC car will be paused.

```
if(distance < 15){
            PWM_duty(PA_0, (float)0);
            PWM_duty(PA_1, (float)0);
                        display_auto = auto_stop;
            }
```

When obstacle is not detected, RC car will go.

```
else if(distance > 15){
               if(right_IR < 1300 && left_IR < 1300){
            PWM_duty(PA_0,(float)0.75); // 80%
            PWM_duty(PA_1,(float)0.75); // 80%
                      display_auto = auto_up;
               }
                     // Right
```

```
        else if(right_IR > 1300 && left_IR < 1300){
            PWM_duty(PA_0,(float)0.2); // 50%
            PWM_duty(PA_1,(float)1);  // 80%
                        display_auto = auto_right;
               }
                           // Left
        else if(right_IR < 1300 && left_IR > 1300){
            PWM_duty(PA_0,(float)1);  // 80%
            PWM_duty(PA_1,(float)0.2);   // 50%
                        display_auto = auto_left;
               }
            }
```

Display State: 'void Display_MState(void)' & 'void Display_AState(void)'

By using USART 1 Communication, we send RC car state of Manual mode and Auto mode.

```
void Display_MState(void){
   if(auto_cnt > 1){
         USART_write(USART1, (uint8_t*)"MOD: M ", 7);
         ...
}

void Display_AState(void){


         ...
         if(auto_cnt > 1){
       USART_write(USART1, (uint8_t*) "MODE : Line Tracer   ", 21);
   }
}
```

Results

Link: https://youtu.be/ao4916Agozc

// please choose 4k

TroubleShooting

1. Auto Mode RC car doesn't work immediately

Because Auto Mode RC Car responded slowly, we can make RC car's speed over 0.8.

So, we used 'Display_AState' function by 'TIMER 4 Interrupt' not 'polling'.

To call 'Display_AState' function as interrupt, we updated 'cnt' and 'auto_cnt' in the TIM4_IRQHandler and use these two signals at the 'Display_AState' function.

In addition to this, we set ADC timer's period bigger than before. (1 msec --> 50 msec).

```
// ADC_init
void ADC_init(PinName_t pinName){
...
ADC_trigger(TIM3, 50, RISE);
...
}
```

```
// TIM_period
void ADC_trigger(TIM_TypeDef* TIMx, int msec, int edge){
...
TIM_period_ms(TIMx, msec);
...
}
```