LAB: Line Tracing RC Car

Date: 2023-11-24

Author/Partner: GyeonhealAn / HanminKim

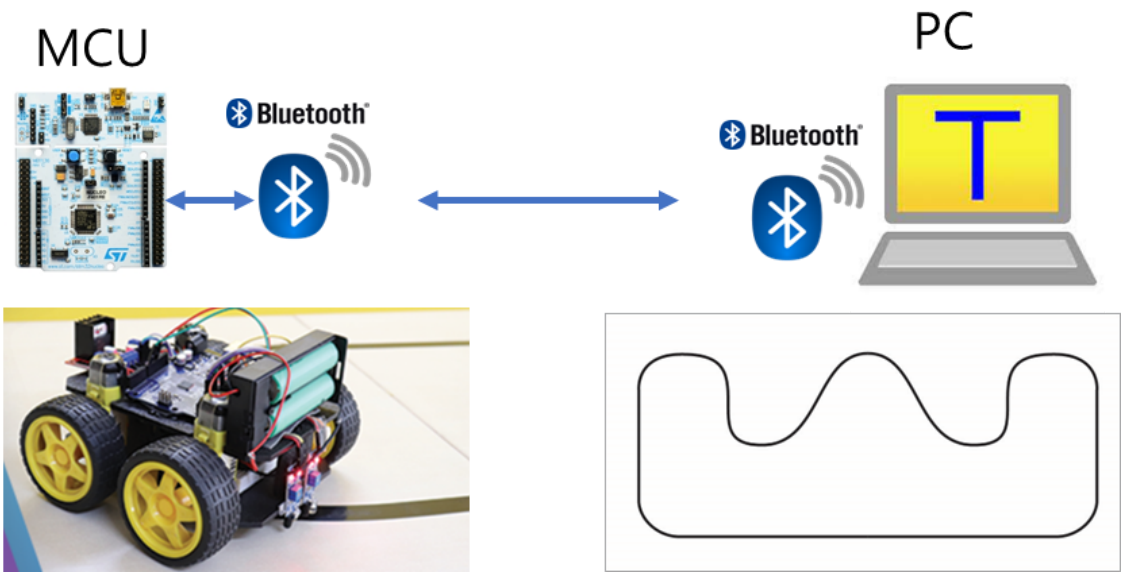
Github: https://github.com/AnGyeonheal/Embedded_Control_GH

Demo Video: https://www.youtube.com/watch?v=S_XWER2FtZE

I. Introduction

Design an embedded system to control an RC car to drive on the racing track. The car is controlled either manually with wireless communication or automatically to drive around the track. When it sees an obstacle on the driving path, it should temporarily stop until the obstacle is out of the path.

> There can be more missions to complete.



i. 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
  - additional sensor/actuators are acceptable

Software

- Keil uVision, CMSIS, EC_HAL library

II. Problem Definition

Design your RC car that has the following functions:

1. Line tracing on the given racing track
2. has 2 control modes: **Manual Mode** to **AUTO Mode**
3. stops temporally when it detects an object nearby on the driving path

- On the PC, connected to MCU via bluetooth
- Print the car status every 1 sec such as " (" MOD: A DIR: F STR: 00 VEL: 00 ")

i. Manual Mode

- Mode Change(MOD):
  - When 'M' or 'm' is pressed, it should enter **Manual Mode**
  - LD2 should be ON in Manual Mode
- Speed (VEL):
  - Increase or decrease speed each time you push the arrow key "UP" or "DOWN", respectively.
  - You can choose the speed keys
  - Choose the speed level: V0 ~ V3
- Steer (STR):
  - Steering control with keyboard keys
  - Increase or decrease the steering angles each time you press the arrow key "RIGHT" or "LEFT", respectively.
  - Steer angles with 3 levels for both sides: e.g: -3, -2, -1, 0, 1, 2, 3 // '-' angle is turning to left
- Driving Direction (DIR)
  - Driving direction is forward or backward by pressing the key "F" or "B", respectively.
  - You can choose the control keys
- Emergency Stop
  - RC car must stop running when key "S" is pressed.

ii. Automatic Mode

- Mode Change:
  - When 'A' or 'a' is pressed, it should enter **AUTO Mode**
- LD2 should blink at 1 second rate in AUTO Mode
- It should drive on the racing track continuously
- Stops temporall when it detects an object nearby on the driving path
- If the obstacle is removed, it should drive continuously

III. Procedure

1. Discuss with the teammate how to design an algorithm for this problem
2. In the report, you need to explain concisely how your system works with state tables/diagram or flow-chart.

   ● Listing all necessary states (states, input, output etc) to implement this design problem.

   ● Listing all necessary conditional FLAGS for programming.

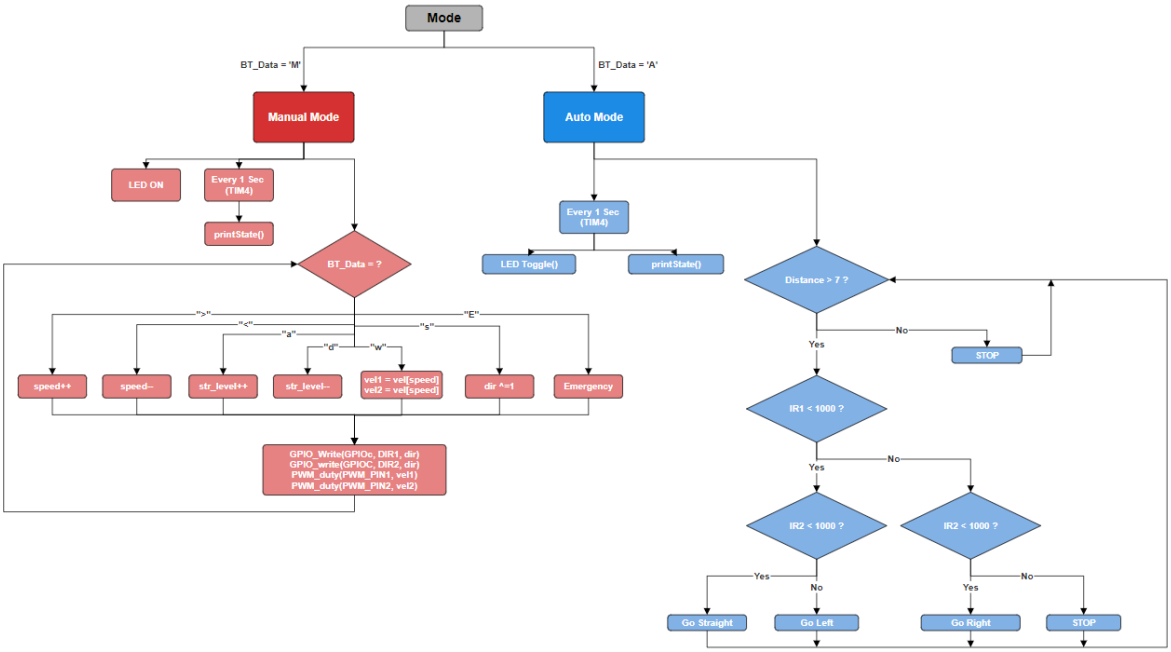
   ● Showing the logic flow from the initialization

   and more

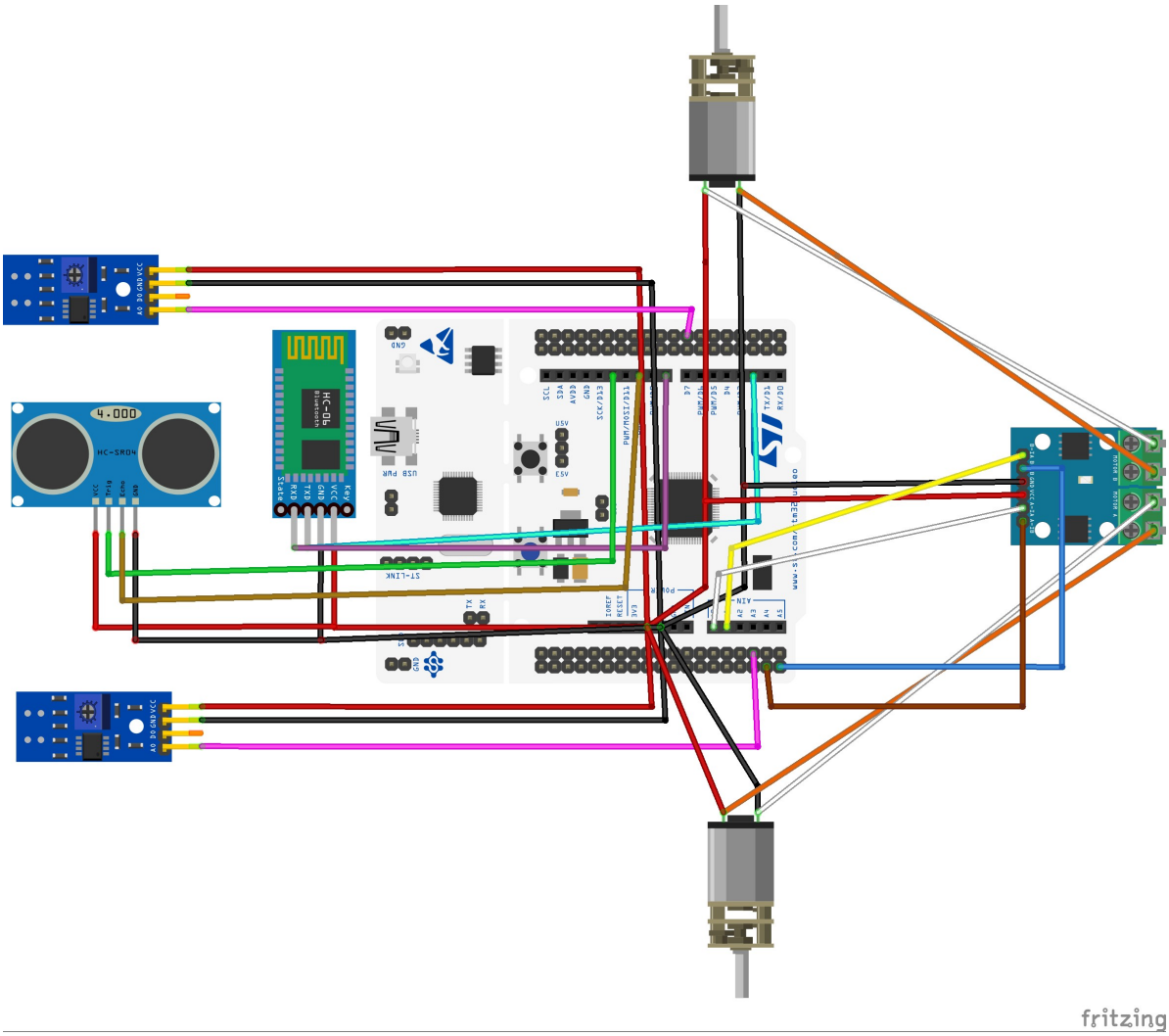
3. Select appropriate configurations for the design problem. Fill in the table.

Functions	Register	PORT_PIN	Configuration
System Clock	RCC		PLL 84MHz
LED		PA5	Output, Medium, Push-Pull, No Pull-up Pull-down
Motor DIR1	Digital Out	PC2	
Motor DIR2	Digital Out	PC3	
Motor PWM1	TIM2_CH1	PA0	AF, PUSH_PULL, PULL-UP, FAST, period : 6ms
Motor PWM2	TIM2_CH2	PA1	AF, PUSH_PULL, PULL-UP, FAST, period : 6ms
Ultra Sonic PWM	TIM3_CH1	TRIG : PA6	AF, Push-Pull, No Pull-up Pull-down, Fast, period: 50ms, pulse width: 10us
	TIM4_CH1	ECHO : PB6	AF, No Pull-up Pull-down, Counter Clock : 0.1MHz
ADC	ADC1_CH8	PB0	Analog Mode No Pull-up Pull-down
	ADC1_CH9	PB1	Analog Mode No Pull-up Pull-down
ADC sampling trigger	TIM3		ADC Clock Prescaler /8 12-bit resolution, right alignment Continuous Conversion mode Scan mode: Two channels in regular group External Trigger (Timer3 Trigger) @ 1kHz Trigger Detection on Rising Edge
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

i. Flow Chart



ii. Circuit Diagram



ii. Code

Your code goes here: GITHUB

Parameter Definition

```c
#include "stm32f4xx.h"
#include "ecSTM32F411.h"
#include "math.h"
#include "stdio.h"
// Pin define
#define DIR_PIN1 2
#define DIR_PIN2 3
PinName_t PWM_PIN1 = PA_0;
PinName_t PWM_PIN2 = PA_1;
#define TRIG PA_6
#define ECHO PB_6

// Velocity, Direction define
#define EX 1
#define v0 0.7
#define v1 0.5
#define v2 0.25
#define v3 0
#define F 1
#define B 0

// PWM period define
float period = 6;

// TIM4 count define
uint32_t _count = 0;

// UltraSonic parameter define
uint32_t ovf_cnt = 0;
float distance = 0;
float timeInterval = 0;
float time1 = 0;
float time2 = 0;

// IR parameter define
uint32_t value1, value2;
int flag = 0;
PinName_t seqCHn[2] = {PB_0, PB_1};

// USART1(Bluetooth) parameter define
static volatile uint8_t PC_Data = 0;
static volatile uint8_t BT_Data = 0;

// Other parameter define
int i=0;        // speed level
char mode;      // mode = 'Manual' or 'Auto'
double vel[4] = {v0, v1, v2, v3};   // velocity levels
int str_level = 0;  // Steer level
double vel1 = 1;    // 1st DC motor duty ratio
double vel2 = 1;    // 2nd DC motor duty ratio
uint8_t dir = 1;    // Direction

// char for printState
char DIR;
char VEL[2];
char STR[2];

// Function Defines
```

```
void setup(void);
double str_angle(int str_level);
void printState(void);
void speedUP();
void speedDOWN();
void M_right();
void M_left();
void M_straight();
void E_stop();
void M_back();
void LED_toggle();
```

This is parameter definition. I set velocity as four levels. The DC motor we used has its maximum speed when the direction is set to 1 and the duty cycle is 0, and its minimum speed occurs when the duty cycle is 1.

Main Function

```
void main(){
    setup();
    // Initial State (STOP)
    GPIO_write(GPIOC, DIR_PIN1, dir);
    GPIO_write(GPIOC, DIR_PIN2, dir);
    PWM_duty(PWM_PIN1, vel1);
    PWM_duty(PWM_PIN2, vel2);
    while(1){
        if(mode == 'A'){    // Auto mode
            distance = (float) timeInterval * 340.0 / 2.0 / 10.0;   // [mm] ->
[cm]
            if(value1 < 1000 && value2 < 1000){ // Move Straight
                vel1 = 0;
                vel2 = 0;
            }
            else if(value1 > 1000 && value2 < 1000){    // Turn right
                vel1 = 0;
                vel2 = 0.6;
            }
            else if(value1 < 1000 && value2 > 1000){    // Turn left
                vel1 = 0.6;
                vel2 = 0;
            }
            else if(value1 > 1000 && value2 > 1000){    // STOP
                vel1 = 1;
                vel2 = 1;
            }
            if(distance < 7){   // Obstacle detected
                E_stop();
            }
            else if(distance > 3000){   // Ignoring dummy value
                continue;
            }
            if(_count >= 1){    // printing state, toggling every 1 second
                LED_toggle();
                printState();
                _count = 0;
            }
            // DC motor operate
```

```
            GPIO_write(GPIOC, DIR_PIN1, dir);
            GPIO_write(GPIOC, DIR_PIN2, dir);
            PWM_duty(PWM_PIN1, vel1);
            PWM_duty(PWM_PIN2, vel2);
        }
        if(mode == 'M'){    // Manual mode
            if((_count >= 2) &(BT_Data!='E')){  // printing state every 1 second
                printState();
                _count = 0;
            }
            GPIO_write(GPIOA, LED_PIN, 1);  // LED ON
            // DC motor operate
            GPIO_write(GPIOC, DIR_PIN1, dir);
            GPIO_write(GPIOC, DIR_PIN2, dir);
            PWM_duty(PWM_PIN1, vel1);
            PWM_duty(PWM_PIN2, vel2);
        }
    }
}
```

The main function executes the setup() function to initialize pins and outputs the initial motor state, which is the stopped state. Within the loop, it distinguishes between Automation mode and Manual mode. In Auto mode, it performs Line Tracing based on the values obtained from two IR sensors (value1, value2). Additionally, when the distance value from the Ultrasonic sensor is less than 7cm, the RC car comes to a stop. To prevent issues arising from sensing errors resulting in extremely large distance values, values exceeding 3000cm are ignored. The MCU's LED blinks every second based on the _count value obtained through TIM4, displaying the current state of the car. Finally, the DC motor is operated using the dir, vel1, and vel2 values obtained earlier.

USART1_IRQHandler

```
void USART1_IRQHandler(){                       // USART2 RX Interrupt :
Recommended
    if(is_USART1_RXNE()){
        BT_Data = USART1_read();        // Send Data PC to Bluetooth
        if(BT_Data == 'M') {                // Manual mode
            USART1_write("Manual Mode\r\n",13);
            mode = 'M';
        }
        else if(BT_Data == 'A'){        // Auto mode
            USART1_write("Auto Mode\r\n",11);
            mode = 'A';
        }
        if(mode == 'M') {
            if (BT_Data == '>'){        // Speed Up
                speedUP();
            }
            else if (BT_Data == '<'){       // Speed Down
                speedDOWN();
            }
            else if (BT_Data == 'd') {      // Turn right
                M_right();
            } else if (BT_Data == 'a') {    // Turn left
                M_left();
            } else if (BT_Data == 'w') {    // Move straight
                M_straight();
```

```
            }
            else if (BT_Data == 's') {      // Move back
                M_back();
            }
            else if (BT_Data == 'E'){           // Emergency STOP
                E_stop();
                USART1_write("Emergency\r\n", 11);
            }
        }
    }
}
```

This function performs commands when values are detected through Bluetooth. It obtains the BT_Data using the USART1_read() function and determines the mode. While Auto mode is executed within the main function, Manual mode is handled within this function. It allows for adjusting the car's speed and steering angle, ranging from -3 to 3. The function also implements forward and backward movement and activates the Emergency Stop function upon reading 'E'.

IR sensor

```
// IR sensor Handler
void ADC_IRQHandler(void){
    if(is_ADC_OVR())
        clear_ADC_OVR();

    if(is_ADC_EOC()){      // after finishing sequence
        if (flag==0)
            value1 = ADC_read();
        else if (flag==1)
            value2 = ADC_read();
        flag =! flag;      // flag toggle
    }
}
```

This function reads the values of IR sensors using the ADC handler functionality. (value1, value2)

TIM4_IRQHandler()

```
// TIM4 Handler (Ultra Sonic)
void TIM4_IRQHandler(void){
    if(is_UIF(TIM4)){                     // Update interrupt
        ovf_cnt++;                                                  // overflow
count
        _count++;                                                       // count
for 1sec
        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) + (TIM4->ARR+1) * ovf_cnt) * 0.01;  //
(10us * counter pulse -> [msec] unit) Total time of echo pulse
        ovf_cnt = 0;                           // overflow reset
        clear_CCIF(TIM4,2);                                    // clear
capture/compare interrupt flag
    }
}
```

This function is the TIM4_IRQHandler() function for the Ultrasonic sensor. Through this function, you can obtain the values of timeInterval and _count.

printState()

```
// Print the state (Manual or Auto mode)
void printState(void){
    if(mode == 'M'){    // Manual Mode
        // Changes int to char
        sprintf(VEL, "%d", i);
        sprintf(STR, "%d", str_level);
        // Print state on PC screen
        USART1_write("MOD: ", 5);
        USART1_write(&mode, 1);
        USART1_write(" DIR: ", 6);
        USART1_write(&DIR, 1);
        USART1_write(" STR: ", 6);
        USART1_write(&STR, 2);
        USART1_write(" VEL: ", 6);
        if(str_level == 0){
        USART1_write("V", 1);
        USART1_write(&VEL, 2);
        }
        USART1_write("\r\n", 2);
    }
    else if(mode == 'A'){   // Automation mode
        if(distance < 7){
            USART1_write("Obstacle Infront\r\n", 18);
        }
        else{
            if(value1 < 1000 && value2 < 1000){
                USART1_write("Straight\r\n",10);
            }
            else if(value1 > 1000 && value2 < 1000){
                USART1_write("Turn right\r\n", 13);
            }
            else if(value1 < 1000 && value2 > 1000){
                USART1_write("Turn left\r\n", 12);
            }
        }
    }
}
```

This function sends the status of the RC car in Manual and Auto modes to a PC via Bluetooth and outputs it. It utilizes the `USART1_write()` function for display, but since this function only transmits strings, `sprintf()` is used to convert speed and steering angle values into strings for output.

RC Car Function

```
void speedUP(){
    i++;
    if(i>=3) i=3;
    vel1 = vel[i];
    vel2 = vel[i];
}
void speedDOWN(){
    i--;
    if(i<=0) i=0;
    vel1 = vel[i];
    vel2 = vel[i];
}
void M_right(){
    str_level--;
    if(str_level<-3) str_level=-3;
    str_angle(str_level);
}
void M_left(){
    str_level++;
    if(str_level>3) str_level=3;
    str_angle(str_level);
}
void M_straight(){
    str_level = 0;
    dir = F;
    vel1 = vel[i];
    vel2 = vel[i];
    DIR = 'F';
}
void M_back(){
    str_level = 0;
    dir = B;
    vel1 = vel[1];
    vel2 = vel[1];
    DIR = 'B';
}
void E_stop(){
    dir = F;
    vel1 = EX;
    vel2 = EX;
}
double str_angle(int str_level){
    if(str_level == -1){
        vel1 = v2;
        vel2 = v1;
    }
    else if(str_level == -2){
        vel1 = v2;
        vel2 = v0;
    }
    else if(str_level == -3){
        vel1 = v3;
        vel2 = v0;
    }
    else if(str_level == 1){
```

```
        vel1 = v1;
        vel2 = v2;
    }
    else if(str_level == 2){
        vel1 = v0;
        vel2 = v2;
    }else if(str_level == 3){
        vel1 = v0;
        vel2 = v3;
    }
    else if(str_level == 0){
        vel1 = vel[i];
        vel2 = vel[i];
    }
}
```

These functions are related to the operation of an RC car in Manual Mode. Pressing the ">" and "<" buttons increases and decreases the speed level (i) to determine vel1 and vel2 values. Additionally, there are functions for moving forward and backward, as well as a function for setting the steering angle. Pressing "a" or "d" buttons changes the str_level, and based on str_level, the str_angle function adjusts the speeds of the left DC motor and right DC motor. When str_level is 0, it maintains the speed of the original forward motion.

LED Toggle Function

```
void LED_toggle(void){
    static unsigned int out = 0;
    if(out == 0) out = 1;
    else if(out == 1) out = 0;
    GPIO_write(GPIOA, LED_PIN, out);
}
```

This function enables the blinking of the MCU's built-in LED in Auto mode.

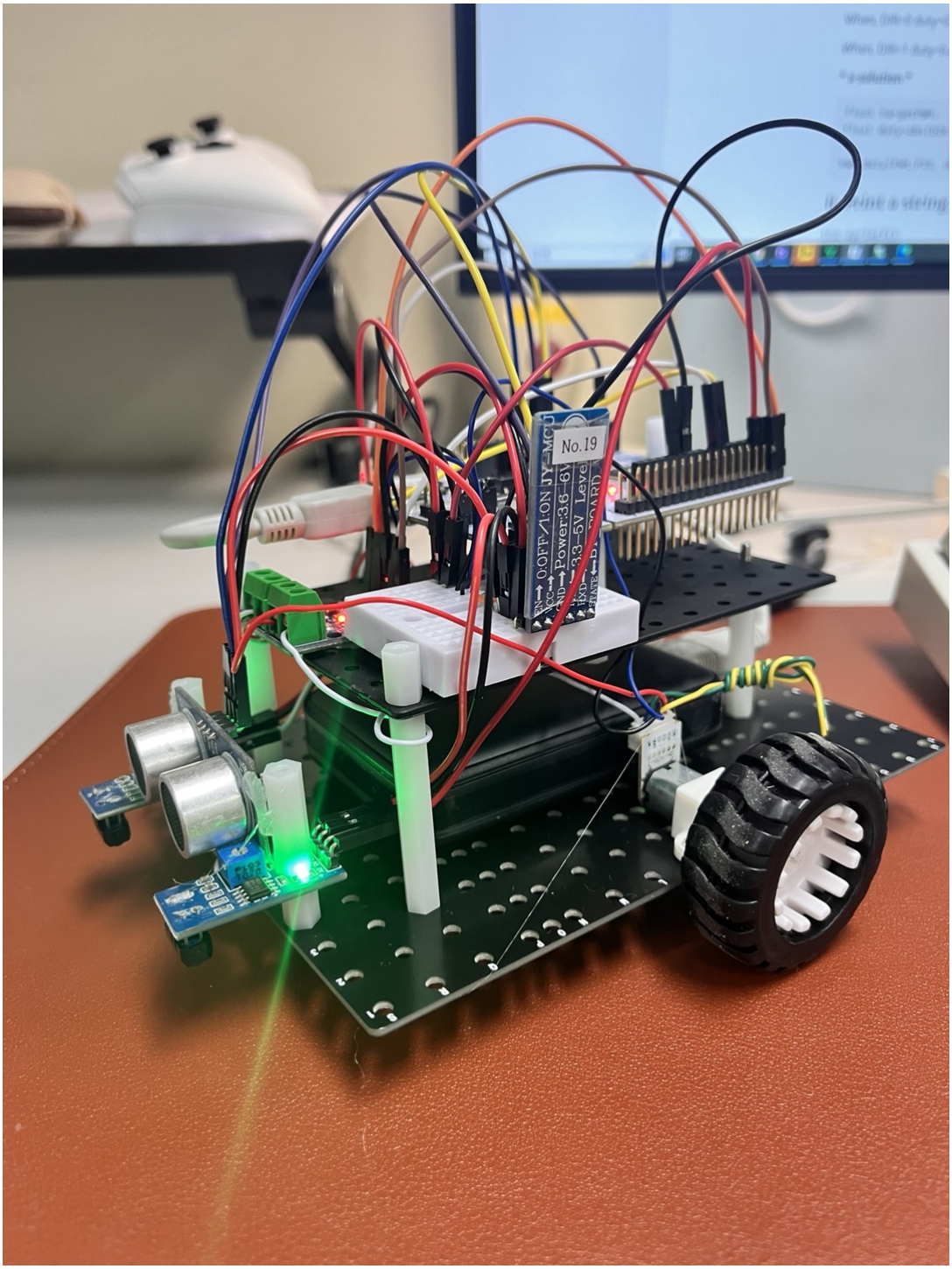
setup Function

```
void setup(void){
    RCC_PLL_init();
    SysTick_init();                     // SysTick Init
    UART2_init();
    // LED
    GPIO(GPIOA, LED_PIN, OUTPUT, EC_MEDIUM, EC_PUSH_PULL, EC_NONE);

    // BT serial init
    UART1_init();
    UART1_baud(BAUD_9600);

    // DIR1 SETUP
    GPIO_init(GPIOC, DIR_PIN1, OUTPUT);
    GPIO_otype(GPIOC, DIR_PIN1, EC_PUSH_PULL);

    // DIR2 SETUP
    GPIO_init(GPIOC, DIR_PIN2, OUTPUT);
    GPIO_otype(GPIOC, DIR_PIN2, EC_PUSH_PULL);

    // ADC Init
```

```
    ADC_init(PB_0);
    ADC_init(PB_1);

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

    // PWM1
    PWM_init(PWM_PIN1);
    PWM_period_ms(PWM_PIN1, period);

    // PWM2
    PWM_init(PWM_PIN2);
    PWM_period_ms(PWM_PIN2, period);

    // PWM configuration ------------------------------------------------------
-------------
    PWM_init(TRIG);         // PA_6: Ultrasonic trig pulse
    PWM_period_us(TRIG, 50000);    // PWM of 50ms period. Use period_us()
    PWM_pulsewidth_us(TRIG, 10);   // PWM pulse width of 10us

    // Input Capture configuration ------------------------------------------
------------------------
    ICAP_init(ECHO);        // PB_6 as input caputre
    ICAP_counter_us(ECHO, 10);      // ICAP counter step time as 10us
    ICAP_setup(ECHO, 1, IC_RISE);  // TIM4_CH1 as IC1 , rising edge detect
    ICAP_setup(ECHO, 2, IC_FALL);  // TIM4_CH2 as IC2 , falling edge detect

}
```

This function configures the basic settings of sensors and motors used in the RC car. Details can be found in the above configuration.

IV. Results



demo video link

V. Reference

LAB - EC (gitbook.io)

VI. Troubleshooting

i. motor PWM duty ratio for different DIR

When, DIR=0 duty=0.8--> PWM 0.8 // actural PWM

When, DIR=1 duty=0.8--> PWM 0.2 // actural PWM

*** a solution ***

```
float targetPWM;  // pwm for motor input
float duty=abs(DIR-targetPWM); // duty with consideration of DIR=1 or 0

PWM_duty(PWM_PIN, duty);
```

ii. Print a string for BT (USART1)

Use `sprintf()`

```
#define _CRT_SECURE_NO_WARNINGS    // sprintf 보안 경고로 인한 컴파일 에러 방지
#include <stdio.h>     // sprintf 함수가 선언된 헤더 파일

char BT_string[20]=0;

int main()
{
    sprintf(BT_string, "DIR:%d PWM: %0.2f\n", dir, duty);    // 문자, 정수, 실수를 문자열로 만듦
    USART1_write(BT_string, 20);
    // ...
}
```

https://dojang.io/mod/page/view.php?id=352

iii. Check and give different Interrupt Priority

We set Priority ADC = 1 / TIM = 2 / UART = 3

I prioritized obtaining values from IR sensors for immediate steering response, assigning priority in the order of TIM and UART.

iv. Preprocessing values obtained from an ultrasonic sensor.

```
else if(distance > 3000){   // Ignoring dummy value
    continue;
}
```

This is for ignoring dummy value.