LAB: Line Tracing RC Car

Date: 2023-11-24

Author/Partner: Eunchan Kim / Heejun Lee

Demo Video: Youtube link

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.

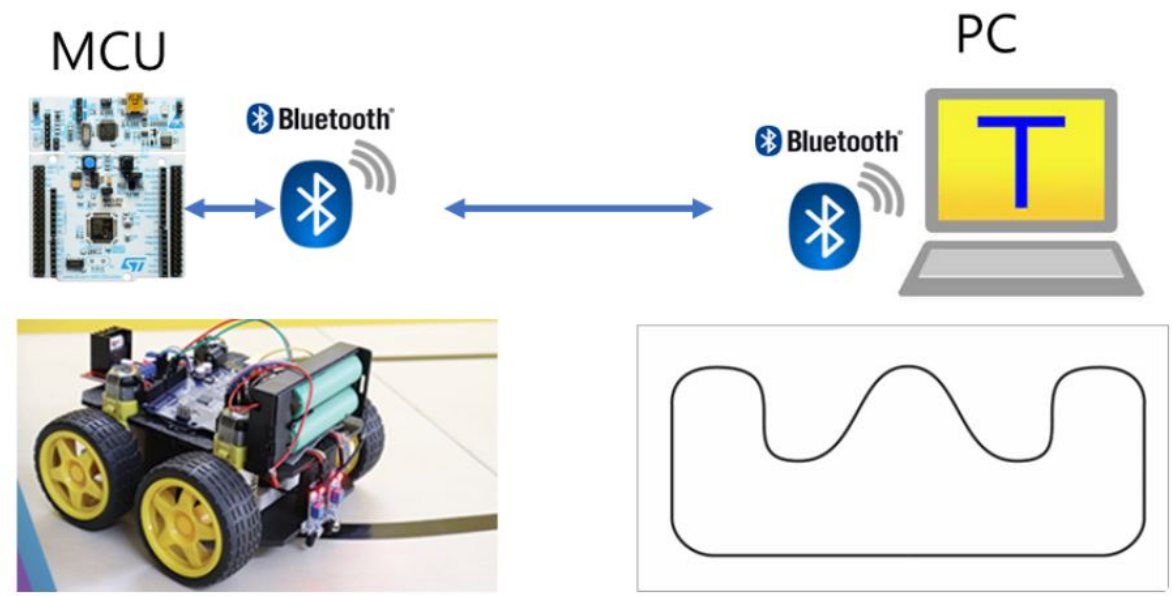


Figure1. Overview of RC Car-PC communication

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

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 ”)

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.

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 temporally when it detects an object nearby on the driving path
- If the obstacle is removed, it should drive continuously

Configuration

Functions	Register	PORT_PIN	Configuration
System Clock	RCC		PLL 84MHz
delay_ms	SysTick		
Motor DIR	Digital Out	PC2, PC3	FWD: 0, BWD: 1
TIMER	TIMER3		500 msec, SetPriority -> 2
	TIMER4		
ADC	ADC	PB0, PB1	SetPriority -> 3
DC Motor Speed	PWM2	PA0, PA1	initialized with a period of 1ms
Ultra-Sonic sampling trigger	PWM3	PA6,PB_6	Initialized with a period of 50ms, pulse width of 10us, Initialized with a count step of 10us, IC1 set for rising edge detection, IC2 set for falling edge detection
RS-232 USB cable(ST-LINK)	USART2		No Parity, 8-bit Data, 1-bit Stop bit 9600 baud-rate
Bluetooth	USART1	TXD: PA9 RXD: PA10	No Parity, 8-bit Data, 1-bit Stop bit 9600 baud-rate

State Table & Flow

A state table according to all angles and speeds was manufactured. Based on this, the vehicle is operated in the manual mode.

Angle	Speed (State)	Next State				Output	
		W	A	S	D	Left(dir-j)	Right(dir-i)
-3	0(S0)	S1	S0	S0	S4	0	0
	1(S1)	S2	S1	S0	S5	0.2	0.5
	2(S2)	S3	S2	S1	S6	0.3	0.7
	3(S3)	S3	S3	S2	S7	0.4	0.9
-2	0(S4)	S5	S0	S4	S8	0	0
	1(S5)	S6	S1	S4	S9	0.3	0.5
	2(S6)	S7	S2	S5	S10	0.4	0.7
	3(S7)	S7	S3	S6	S11	0.5	0.9
-1	0(S8)	S9	S4	S8	S12	0	0
	1(S9)	S10	S5	S8	S13	0.4	0.5
	2(S10)	S11	S6	S9	S14	0.5	0.7
	3(S11)	S12	S7	S10	S15	0.6	0.9
0	0(S12)	S13	S8	S12	S16	0	0
	1(S13)	S14	S9	S12	S17	0.5	0.5
	2(S14)	S15	S10	S13	S18	0.7	0.7
	3(S15)	S15	S11	S14	S19	0.9	0.9
1	0(S16)	S17	S12	S16	S20	0	0
	1(S17)	S18	S13	S16	S21	0.5	0.4
	2(S18)	S19	S14	S17	S22	0.7	0.5
	3(S19)	S19	S15	S18	S23	0.9	0.6
2	0(S20)	S21	S16	S20	S24	0	0
	1(S21)	S22	S17	S20	S25	0.5	0.3
	2(S22)	S23	S18	S21	S26	0.7	0.4
	3(S23)	S23	S19	S22	S27	0.9	0.5
3	0(S24)	S25	S20	S24	S24	0	0
	1(S25)	S26	S21	S24	S25	0.5	0.2
	2(S26)	S27	S22	S25	S26	0.7	0.3
	3(S27)	S27	S23	S26	S27	0.9	0.4

Table1. State Table for Manual Mode

The RC Car drive flow is as follows. The mode flag is different according to the manual mode and the auto mode so that each corresponding operation is performed in the main and handler. In manual mode, speed, rotation, forward and backward according to keyboard input are determined. In the auto mode, it is responsible for rotation, stopping, and progression according to IR sensor values and ultrasonic sensor values.

1. **Initialization**
2. **Main Loop**
    - **'M' mode** : For Manual Mode
    - **'N' mode**: For Auto Mode
3. **Interrupt Handlers**
    - **TIM3_IRQHandler**
    - **USART1_IRQHandler**
    - **ADC_IRQHandler**
    - **TIM4_IRQHandler**

Circuit Diagram

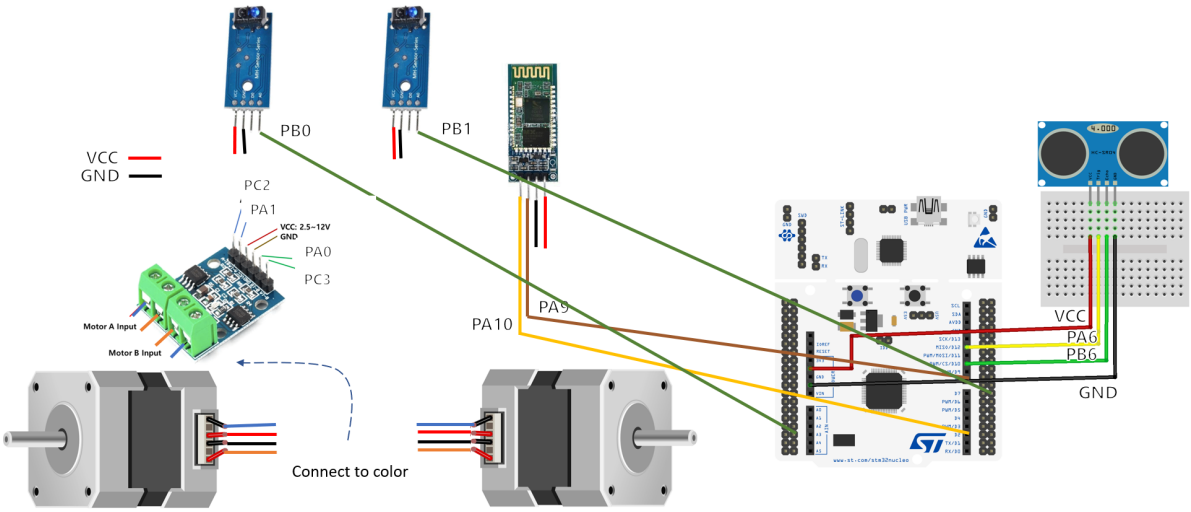


Figure2. Curcuit Diagram of RC Car Sensors

Code

The main flow of the code is as follows.

1. **Initialization**: The program begins by initializing various hardware components such as System Clock, UART, SysTick, ADC, GPIO, PWM, and Timers. It also sets up the input capture for ECHO and PWM for TRIG.
2. **Main Loop**: In the main loop, the program checks for two modes: 'M' and 'N'.
    - In **'M' mode**, the program sets the duty cycle of motors based on the direction and sends the current state to the USART1.
    - In **'N' mode**, the program uses IR sensors to control the motors. If the difference between the two IR sensors is larger than a certain threshold, it adjusts the angle and speed of the motors. If an object is detected within a certain distance (30 cm), it stops the motors.
3. **Interrupt Handlers**: The program has several interrupt handlers:
    - **TIM3_IRQHandler**: This handler is triggered every 500ms. It updates the PWM duty cycle for the motors if in 'M' mode and toggles an LED every 4 seconds. In 'N' mode, it sends the current state to USART1 if no object is detected.
    - **USART1_IRQHandler**: This handler is triggered when data is received from UART1. It updates the mode, direction, speed, and angle based on the received data.
    - **ADC_IRQHandler**: This handler is triggered when the ADC conversion is complete. It reads the ADC value alternately from two IR sensors.
    - **TIM4_IRQHandler**: This handler is triggered when a rising or falling edge is detected on the ECHO pin or when the timer overflows. It calculates the time interval of the echo pulse, which is used to calculate the distance to the object.

The variables to be used first were defined.

```
static volatile uint8_t BT_Data = 0;
```

```
static volatile float i=0;
static volatile float j=0;
static volatile int dir=0;
static volatile int speed=0;
static volatile int angle=0;
static volatile char mode;
static volatile char direction;
static volatile float iduty=0;
static volatile float jduty=0;
static volatile int state=0;
char s1[30];

uint32_t ovf_cnt = 0;
float distance = 0;
float timeInterval = 0;
float time1 = 0;
float time2 = 0;

#define TRIG PA_6
#define ECHO PB_6

//IR parameter//
uint32_t value1, value2;
int flag = 0;
PinName_t seqCHn[2] = { PB_0, PB_1 };
int IR_diff = 0;
int Stop_flag = 0;
int cnt = 0;
int LED_flag = 0;
```

The code was constructed according to the State Table.

```
static volatile float duty[28][2]={
    {0,0},
    {0.2,0.5},
    {0.3,0.7},
    {0.4,0.9},
    {0,0},
    {0.3,0.5},
    {0.4,0.7},
    {0.5,0.9},
    {0,0},
    {0.4,0.5},
    {0.5,0.7},
    {0.6,0.9},
    {0,0},
    {0.5,0.5},
    {0.7,0.7},
    {0.9,0.9},
    {0,0},
    {0.5,0.4},
    {0.7,0.5},
    {0.9,0.6},
    {0,0},
    {0.5,0.3},
    {0.7,0.4},
    {0.9,0.5},
```

```
    {0,0},
    {0.5,0.2},
    {0.7,0.3},
    {0.9,0.4}
};
```

First, the `setup()` function is called to initialize necessary hardware and software settings. Then, an infinite loop starts, mainly handling two modes ('M' and 'N').

- 'M' mode(Manual):
  - Calculate the difference between `dir`, `i`, and `j` to set `iduty` and `jduty`.
  - Output the value of `dir` to pins 2 and 3 of GPIOC.
  - Store information about the current mode, direction, angle, and speed in the string `s1` and transmit it via USART1.
  - Wait for 1 second.
- - 'N' mode(Auto):
  - Calculate the difference between values read from two IR sensors.
  - If `Stop_flag` is 0 (i.e., not in a stopped state), adjust the angle based on the difference in IR sensor values and change the PWM duty cycle of the motor.
  - Use an ultrasonic sensor to calculate distance. If the distance is less than 30cm and greater than 0, set `Stop_flag` to 1 and set the PWM duty cycle of the motor to 0 to stop the vehicle. Otherwise, keep `Stop_flag` as 0.

```
void main(){
    setup();
  while(1){
        if(mode=='M'){
            iduty=fabs(dir-i);
            jduty=fabs(dir-j);
            GPIO_write(GPIOC,2,dir);
            GPIO_write(GPIOC,3,dir);
            //TIM_UI_init(TIM3,10);
            sprintf(s1,"\r\nMOD:%c DIR:%c STR:%.2d
VEL:%.2d\r\n",mode,direction,angle,speed);
            USART1_write(s1,30);
            delay_ms(1000);

        }

        else if (mode == 'N') {
            IR_diff = value1 - value2;
            if (Stop_flag == 0) {
                if (IR_diff > 400) {
                    angle = 1;
                    PWM_duty(PA_0, 1);
                    PWM_duty(PA_1, 0.25);
                }
                else if (IR_diff < -400) {
                    angle = -1;
                    PWM_duty(PA_0, 0.25);
                    PWM_duty(PA_1, 1);
                }
                else {
                    angle = 0;
                    PWM_duty(PA_0, 1);
```

```
                    PWM_duty(PA_1, 1);
                }


            }
            distance = (float)timeInterval * 0.034 / 2;      // [mm] -> [cm]
            if (distance < 30 && distance>0) {
                Stop_flag = 1;
                PWM_duty(PA_0, 0);
                PWM_duty(PA_1, 0);

            }
            else {
                    Stop_flag = 0;
            }

        }
    }
}
```

Sensors and timers were initialized according to the configuration.

```
void setup(void){
  RCC_PLL_init();
    SysTick_init();

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

    UART2_init();
  UART2_baud(BAUD_9600);

    USART_setting(USART1, GPIOA, 9, GPIOA, 10, BAUD_9600);

    SysTick_init();

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

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

    GPIO_init(GPIOA,5,OUTPUT);
    GPIO_init(GPIOC,2,OUTPUT);
    GPIO_init(GPIOC,3,OUTPUT);
    GPIO_write(GPIOC, 2, LOW);
    GPIO_write(GPIOC, 3, LOW);

    PWM_init(PA_0);
    PWM_period_ms(PA_0, 1);
    PWM_init(PA_1);
    PWM_period_ms(PA_1, 1);

    // Timer setting
```

```
    TIM_UI_init(TIM3, 500);          // TIM3 Update-Event Interrupt every 500
msec
    TIM_UI_enable(TIM3);
    NVIC_EnableIRQ(TIM3_IRQn);   // TIM3 interrupt request enabled
    NVIC_SetPriority(TIM3_IRQn, 3); // TIM3 interrupt priority



    // 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 primarily controls the PWM duty cycle of the motor in 'M' mode and controls the LED state while periodically transmitting the status of the automatic mode. It increments the counter `cnt`. If the current mode is 'M', it sets two PWM outputs to `iduty` and `jduty`. It toggles the LED_flag whenever cnt is a multiple of 80. If the current mode is 'N', it outputs the value of LED_flag to pin 5 of GPIOA. Additionally, if Stop_flag is 0, it stores the status of the automatic mode in the string s1 whenever cnt is a multiple of 80 and transmits it via USART1. Finally, it clears the UIF of TIM3 to reset the interrupt.

```
void TIM3_IRQHandler(void) {
        if ((TIM3->SR & TIM_SR_UIF) == TIM_SR_UIF) {
            cnt++;
            if(mode=='M'){
                PWM_duty(PA_0, iduty);
                PWM_duty(PA_1, jduty);
            }
            if (cnt % 80 == 0) {
                LED_flag ^= 1; // flag switching
            }
            if (mode == 'N') {
                GPIO_write(GPIOA, 5, LED_flag);
                if(Stop_flag == 0){
                    if (cnt % 80 == 0){
                        sprintf(s1,"\r\nMOD:Auto DIR:FWD STR:%.2d
VEL:1\r\n",angle);
                        USART1_write(s1,30);
                    }

                }
            }

            TIM3->SR &= ~TIM_SR_UIF;
    }
```

```
}
```

This function primarily controls the direction, speed, and mode of the vehicle based on commands received through Bluetooth. The function `is_USART1_RXNE()` checks if the USART1 receive data register is empty. If data is available, it reads and stores it in the `BT_Data` variable and then transmits the stored data back through USART1.

- Upon receiving 'm' or 'M', it sets GPIOA pin 5 to HIGH and sets the mode to 'M'.
- Upon receiving 'F' or 'f', it sets the vehicle's direction to forward, initializing motor duty cycle, angle, and speed to 0.
- Upon receiving 'B' or 'b', it sets the vehicle's direction to reverse, initializing motor duty cycle, angle, and speed to 0.
- Upon receiving 'W' or 'w', it increases the speed and adjusts the duty cycle to match the new speed. However, if the speed is already at its maximum (3), no action is taken.
- Upon receiving 'S' or 's', it decreases the speed and adjusts the duty cycle to match the new speed. However, if the speed is already at its minimum (0), no action is taken.
- Upon receiving 'A' or 'a', it decreases the angle and adjusts the duty cycle to match the new angle. However, if the angle is already at its minimum (-3), no action is taken.
- Upon receiving 'D' or 'd', it increases the angle and adjusts the duty cycle to match the new angle. However, if the angle is already at its maximum (3), no action is taken.
- Upon receiving 'P' or 'p', it sets the motor's duty cycle to 0.
- Upon receiving 'N' or 'n', it sets the mode to 'N'.

```
void USART1_IRQHandler(){                // USART2 RX Interrupt : Recommended
    if(is_USART1_RXNE()){
    BT_Data = USART1_read();         // RX from UART1 (BT)
        USART1_write(&BT_Data,1);
        if(BT_Data=='m'||BT_Data=='M'){
            GPIO_write(GPIOA,5, 1);
            mode='M';
        }
        else if(BT_Data=='F'||BT_Data=='f'){
            dir=0;
            i=0;
            j=0;
            direction='F';
            state=12;
            angle=0;
            speed=0;
        }
        else if(BT_Data=='B'||BT_Data=='b'){
            dir=1;
            i=0;
            j=0;
            direction='B';
            state=12;
            angle=0;
            speed=0;
        }
        else if(BT_Data=='W'||BT_Data=='w'){
            if(speed==3){
                return;
            }
            else{
```

```
                state++;
                speed++;
                j=duty[state][0];
                i=duty[state][1];
            }
        }
        else if(BT_Data=='S'||BT_Data=='s'){
            if(speed==0){
                return;
            }
            else{
                state--;
                speed--;
                j=duty[state][0];
                i=duty[state][1];
            }
        }
        else if(BT_Data=='A'||BT_Data=='a'){
            if(angle==-3){
                return;
            }
            else{
                angle--;
                if(dir==0){state-=4;}
                else if(dir==1){state+=4;}
                j=duty[state][0];
                i=duty[state][1];
            }
        }
        else if(BT_Data=='D'||BT_Data=='d'){
            if(angle==3){
                return;
            }
            else{
                angle++;
                if(dir==0){state+=4;}
                else if(dir==1){state-=4;}
                j=duty[state][0];
                i=duty[state][1];
            }
        }
        else if(BT_Data=='P'||BT_Data=='p'){
            i=0;
            j=0;
        }
        else if (BT_Data == 'N' || BT_Data == 'n') {
            mode = 'N';
        }
    }
}
```

This function is called when the ADC conversion is completed or when an overflow occurs.

The `is_ADC_OVR()` function checks the overflow status of the ADC. If an overflow has occurred, it calls the `clear_ADC_OVR()` function to clear the overflow flag.

Next, the `is_ADC_EOC()` function checks if the ADC conversion is complete. If the conversion is finished, it checks the value of the `flag` variable. Depending on its value (0 or 1), it stores the converted value in either `value1` or `value2`. As the `flag` toggles between 0 and 1, it alternates storing the ADC values in these two variables.

```
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 primarily detects the start and end of echo pulses from an ultrasonic sensor to calculate the total time. This calculated time can be used to measure the distance between an object and the sensor.

The `is_UIF(TIM4)` function checks if the Update Interrupt Flag (UIF) of TIM4 is set. If it is set, it increments the overflow counter `ovf_cnt` and clears the UIF.

Next, the `is_CCIF(TIM4, 1)` function checks if a Rising Edge is detected on Channel 1 (IC1) of TIM4. If detected, it saves the captured time in `time1` and clears the CCIF.

Finally, the `is_CCIF(TIM4, 2)` function checks if a Falling Edge is detected on Channel 2 (IC2) of TIM4. If detected, it saves the captured time in `time2`. It calculates the total time of the echo pulse in `timeInterval`, resets the overflow counter, and clears the CCIF.

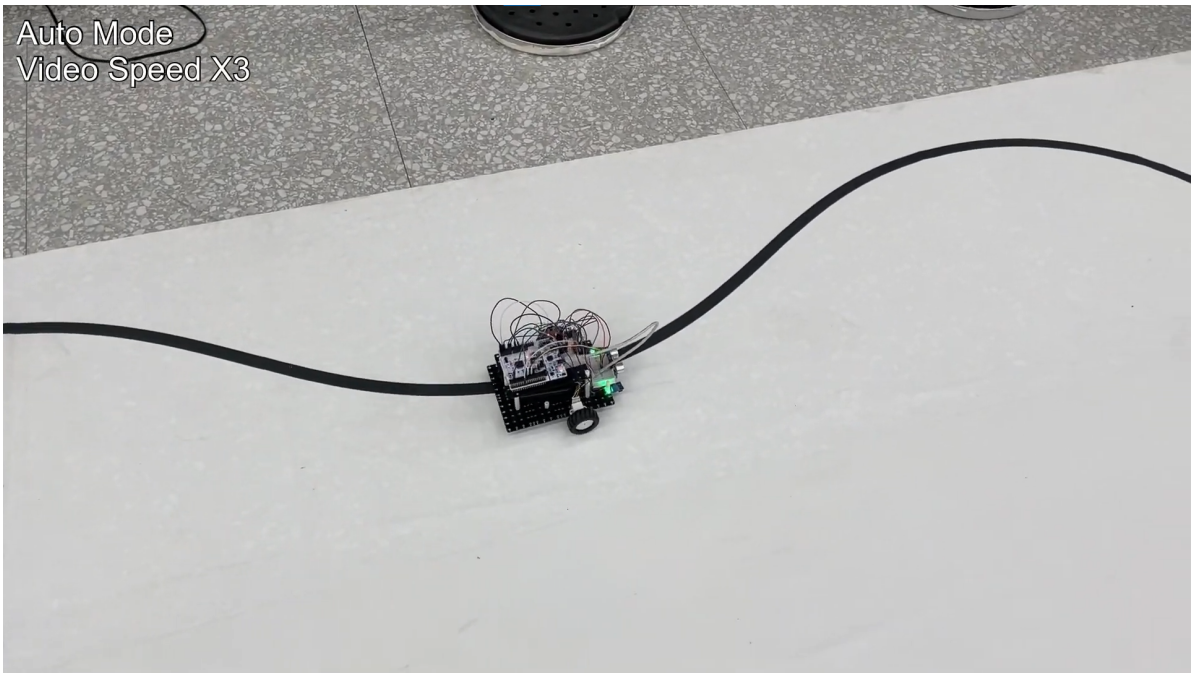
```
void TIM4_IRQHandler(void) {
    if (is_UIF(TIM4)) {                     // Update interrupt
        ovf_cnt++;                                              // overflow
count
        clear_UIF(TIM4);                                // clear update
interrupt flag
    }
    if (is_CCIF(TIM4, 1)) {                                 // TIM4_Ch1 (IC1)
Capture Flag. Rising Edge Detect
        time1 = ICAP_capture(TIM4, IC_1);                                   //
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 = ICAP_capture(TIM4, IC_2);                                   //
Capture TimeEnd
        timeInterval = 10 * ((ovf_cnt * (TIM4->ARR + 1)) + (time2 - time1));
// (10us * counter pulse -> [msec] unit) Total time of echo pulse
        ovf_cnt = 0;                            // overflow reset
        clear_CCIF(TIM4, 2);                                    // clear
capture/compare interrupt flag
```

```
    }
}
```

Results

As can be seen from the image, it can be seen that the forward and backward, left and right rotations work well in the manual mode. In auto mode, the vehicle moves along a black line and stops when an obstacle appears in front of it.

Demo video↓



Reference

- STM32 Cortex®-M4 MCUs and MPUs programming manual Download Link
- STM32 Cortex®-M4 MCUs and MPUs reference manual Download Link

Troubleshooting

1. 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

2. Motor does not run under duty 0.5

When duty was less than 0.5, there was a problem that the motor did not move. This is a problem that occurs because the PWM period of the motor is too fast, and it operates normally when it is adjusted to 1 kHz.

3. Check and give different Interrupt Priority

There was a case where the IR sensor did not work properly. As a result of confirming the cause, it was a problem that occurred because the priority of TIM3 and ADC handlers was the same as 2. By changing the priority of the ADC handler to 3, the problem could be solved.

4. Ultrasonic sensor outliers

The problem of stopping occasionally occurred even though there were no obstacles in front of the vehicle. The cause was due to an outlier in the ultrasonic sensor and was solved by correcting the code to stop the distance between 0 and 30.