LAB: ADC - IR sensor

Date: 2022-11-31

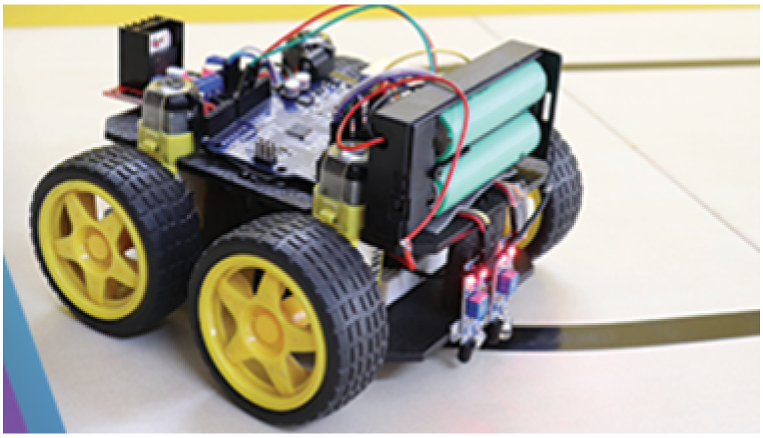
Author/Partner: Dongmin Kim, Jinho Kook

Github: https://github.com/DongminKim21800064/EC_dmkim-064

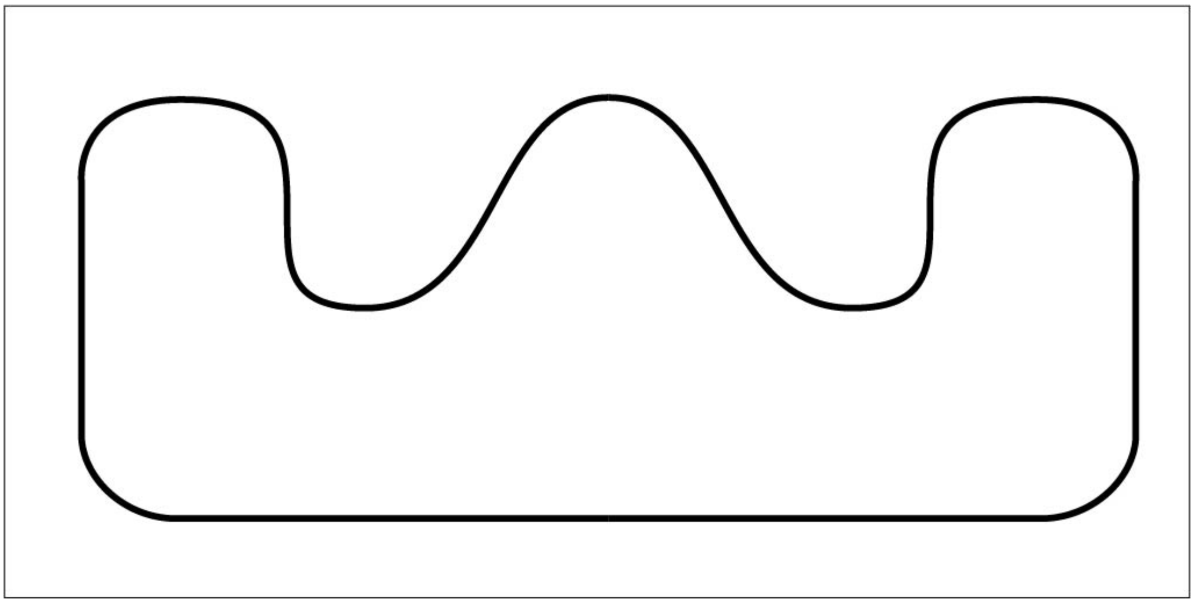
Demo Video: : https://youtu.be/jyC4ZVEwGiI

Introduction

In this lab, you are required to create a simple application that uses ADCs to implement the line tracing mission for an RC car. The analog measurement of reflection values from two IR reflective sensors are used . The ADCs are triggered by a timer of given sampling rate.



image



Track

You must submit

- LAB Report (*.md & *.pdf)
- Zip source files(main*.c, ecRCC.h, ecGPIO.h, ecSysTick.c etc...).

- Only the source files. Do not submit project files

Requirement

Hardware

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

Software

- Keil uVision, CMSIS, EC_HAL library

Problem 1: Create HAL library

Create HAL library

Declare and Define the following functions in your library. You must

update your header files located in the directory `EC \lib\`.

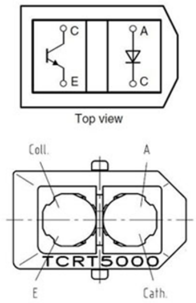
ecADC.h

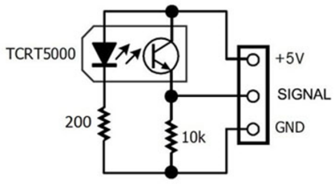
```
// ADC setting

void ADC_init(GPIO_TypeDef *port, int pin, int trigmode);   // trigmode : SW , TRGO
void ADC_continue(int contmode);                            // contmode : CONT, SINGLE / Operate both ADC,JADC
void ADC_TRGO(TIM_TypeDef* TIMx, int msec, int edge);       // set the TIMx TRGO (TIM2, TIM3 only)
void ADC_sequence(int length, int *seq);                    // configure the order of ADC channels
void ADC_start(void);
uint32_t ADC_read(void);
uint32_t is_ADC_EOC(void);
uint32_t is_ADC_OVR(void);
void clear_ADC_OVR(void);
uint32_t ADC_pinmap(GPIO_TypeDef *port, int pin);
```

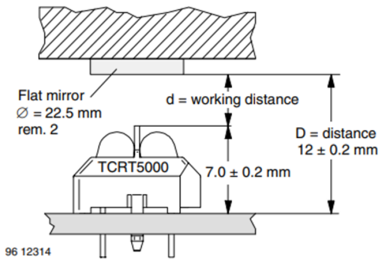
Problem 2: IR Reflective Sensor (TCRT 5000)

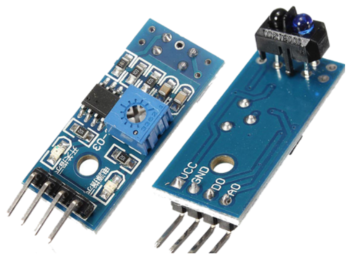
IR Reflective Sensor(TCRT 5000): [Spec Sheet]()

The TCRT5000 and TCRT5000L are reflective sensors which include an infrared emitter and phototransistor in a leaded package which blocks visible light.









The HC-SR04 Ultrasonic Range Sensor Features:

- Input Voltage : 5V
- Detector type: phototransistor
- Operating range within > 20 % relative collector current: 0.2 mm to 15 mm
- Emitter wavelength: 950 nm

APPLICATIONS

- Position sensor for shaft encoder
- Detection of reflective material such as paper, IBM cards, magnetic tapes etc.
- Limit switch for mechanical motions in VCR
- General purpose - wherever the space is limited

Procedure

1. Create a new project under the directory `\repos\EC\LAB\LAB_ADC_IR`

- The project name is "**LAB_ADC_IR**"
- Create a new source file named as "**LAB_ADC_IR.c**"

> You MUST write your name on the source file inside the comment section.

2. Include your updated library in `\repos\EC\lib\` to your project.

- **ecGPIO.h, ecGPIO.c**
- **ecRCC.h, ecRCC.c**
- **ecTIM.h**, **ecTIM.c**
- **ecUART.h, ecUART.c**
- **ecADC.h, ecADC.c**

Configuration

TIMER	ADC	GPIO
TIM3	ADC1_CH8 (1st channel) ADC1_CH9 (2nd channel)	PB_0 PB_1
Up-Counter, Counter CLK 1kHz OC1M(Output Compare 1 Mode) : PWM mode 1 Master Mode Selection: (TRGO) OC1REF	ADC Clock Prescaler /812-bit resolution, right alignment Single Conversion mode Scan mode: Two channels in regular group External Trigger (Timer3 TRGO) @ 1kHz Trigger Detection on Rising Edge	Analog Mode No Pull-up Pull-down

Line Tracing

- Create a logic to trace a dark line on white background surface for your RC car.
- Use 2 IR reflective sensors to detect if the black line is in between the sensors. It should display whether the system needs to move **Left or Right** to keep the line between sensors.
- Set the ADC sampling rate trigger to be 1KHz, to decrease burden to your CPU.
- Determine the threshold value to differentiate dark and white surface of the object.
- Display (1) and (2) on serial monitor of Tera-Term. Print the values every second.

  (1) reflection value of IR1 and IR2

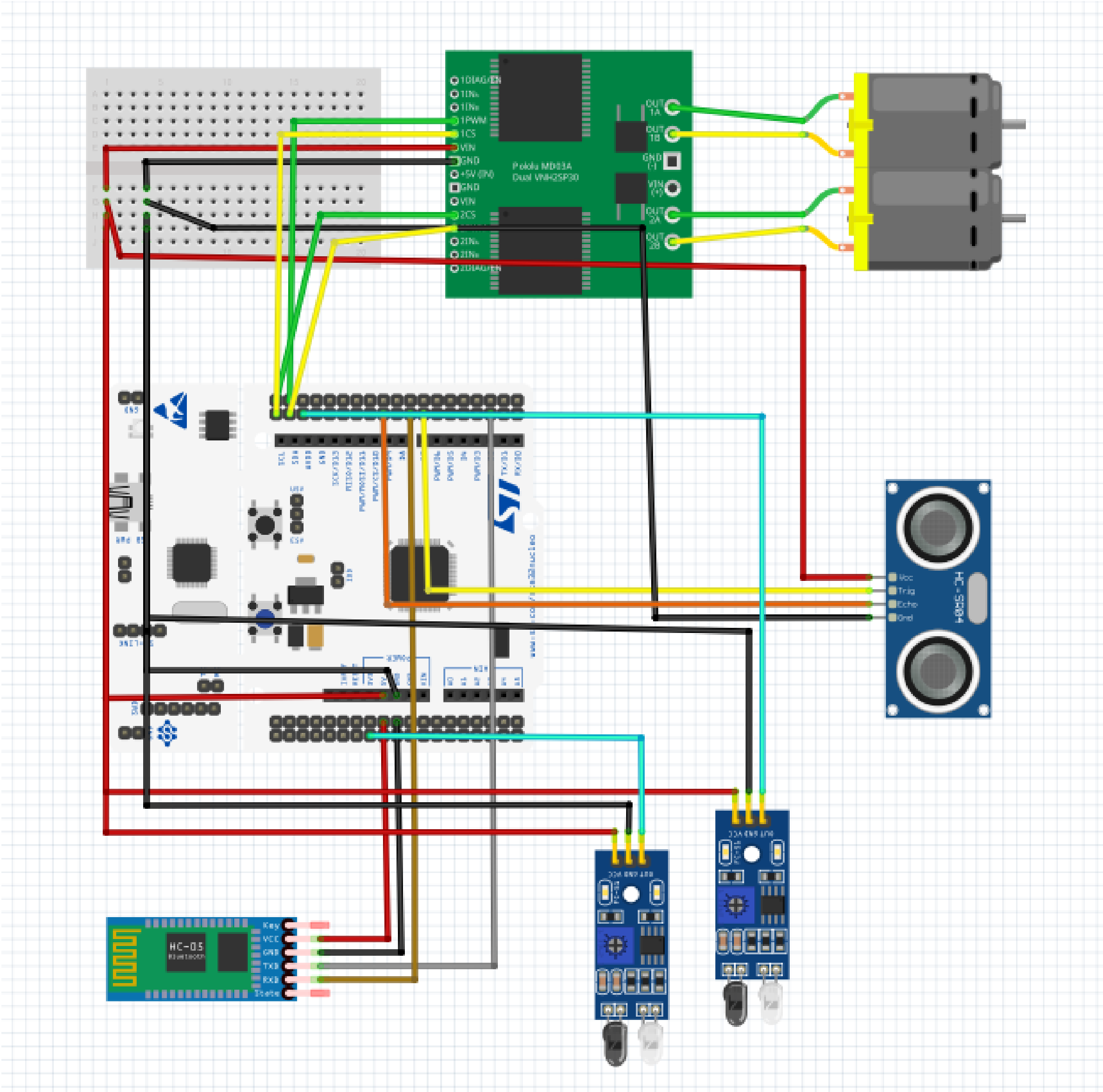
  (2) print 'GO LEFT' or 'GO 'RIGHT'

Display Example

```
IR1 = 3582
IR2 = 219
GO LEFT

IR1 = 220
GO RIGHT
IR2 = 3449

IR1 = 898
GO RIGHT
IR2 = 3913

IR1 = 1952
IR2 = 269
GO LEFT

IR1 = 756
GO RIGHT
IR2 = 3911

IR1 = 3057
IR2 = 3785

IR1 = 2397
IR2 = 3406

IR1 = 264
GO RIGHT
IR2 = 3589
```

Circuit Diagram

> You need to include the circuit diagram

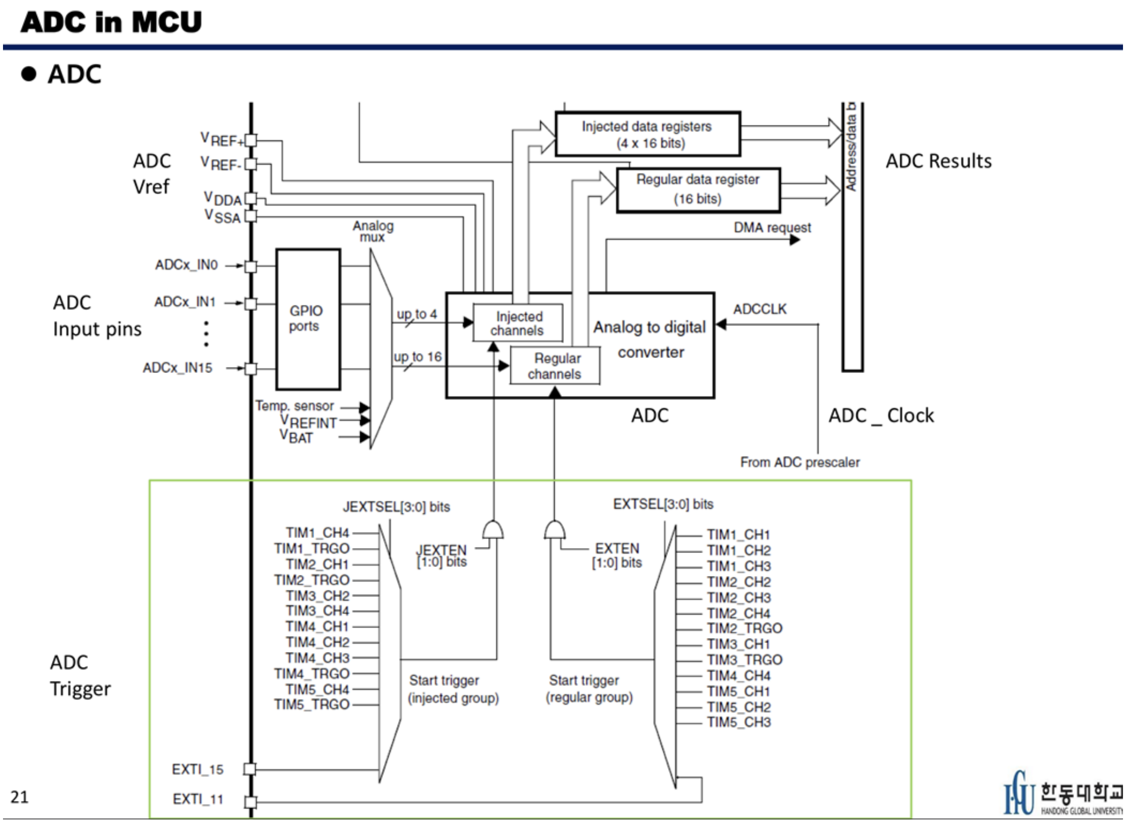


Discussion

1. How would you change the code if you need to use 3 Analog sensors?

> 1. Call **ADC_init(GPIO port, pin, TRGO)** once again
> 2. Modify **int sequence[2]** to **int sequence[3]**
> 3. Modify **ADC_sequence(2, sequence)** to **ADC_sequence(3, sequence)**
> 4. Modify ADC_IRQHandler(void) – Add additional EXTI_flag state
>
> After above the setting, modify **int main()**. Using by 3 Analog sensors, more accurate line tracing can be possible.

2. Which registers should be modified if you need to use Injection Groups instead of regular groups for 2 analog sensors?



Injected group

This mode is enabled by setting the JDISCEN bit in the ADC_CR1 register. It can be used to convert the sequence selected in the ADC_JSQR register, channel by channel, after an external trigger event.

When an external trigger occurs, it starts the next channel conversions selected in the ADC_JSQR registers until all the conversions in the sequence are done. The total sequence length is defined by the JL[1:0] bits in the ADC_JSQR register.

Example:

n = 1, channels to be converted = 1, 2, 3
1st trigger: channel 1 converted
2nd trigger: channel 2 converted
3rd trigger: channel 3 converted and JEOC event generated
4th trigger: channel 1

When all injected channels are converted, the next trigger starts the conversion of the first injected channel. In the example above, the 4th trigger reconverts the 1st injected channel 1.

It is not possible to use both the auto-injected and discontinuous modes simultaneously.

Discontinuous mode must not be set for regular and injected groups at the same time. Discontinuous mode must be enabled only for the conversion of one group.

Like this way, injected group mode is enabled by setting the JDISCEN bit in the ADC_CR1 register. It can be used to convert the sequence selected in the ADC_JSQR register, channel by channel, after an external trigger event.

Code

Your code goes here: https://github.com/DongminKim21800064/EC_dmkim-064

Explain your source code with necessary comments.

```
#include "stm32f411xe.h"
```

```c
#include "ecGPIO.h"
#include "ecRCC.h"
#include "ecTIM.h"
#include "ecSysTick.h"
#include "ecUART.h"
#include "ecADC.h"
#include "ecPWM.h"

//IR parameter//
uint32_t IR1, IR2;
//int flag = 0;
int seqCHn[16] = {8,9,};


#define END_CHAR 13
#define A 0
#define B 1
#define MAX_BUF 100


uint8_t pcData = 0;
uint8_t mcu2Data = 0;
uint8_t btData = 0;
uint8_t buffer[MAX_BUF] = {0, };
uint8_t buffer2 = '\r\n';
int bReceive = 0;
int idx = 0;
int flag = 5;
int dis_flag =0;
int dirA =0;
int dirB =0;
int R_DC_velocity =0;
int L_DC_velocity =0;
float R_duty = 0.f;
float L_duty = 0.f;



uint16_t flagidx =0;



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

void setup(void);

_Pin dcPwmPin[2] = {
   {GPIOC, 9}, // TIM3 Ch3 A-IA
   {GPIOC, 8}   // TIM3 Ch4 B-IA
};

PWM_t dcPwm[2];

_Pin dcDirPin[2] = {
```

```
    {GPIOB, 8}, //A-IB
    {GPIOC, 6}   //B-IB
};


int main(void) {

    // Initialiization --------------------------------------------------------
    setup();
      printf("Hello Nucleo\r\n");

//USART_write(USART1,(uint8_t*) "Hello bluetooth\r\n",17);


    // Inifinite Loop ----------------------------------------------------------
    while(1){


          // For ignore Initial sensor malfunction
          //distance = (timeInterval*340/(2*1000000) ) * 100;
          // ( timeInterval[us]*340[m/s]/(2*1000000) ) * 100 ==> distance[cm]
          distance = (float) timeInterval * 340.0 / 2.0 / (10.0 * 100);
                printf("raw_distance = %f [cm]\r\n", distance); delay_ms(500);
          //if(distance>2 && distance<400) {
          //printf("\r%f [cm]",distance);

                if(distance < 10.0) {
                 //dis_flag =1;
                         printf("disflag is on\n"); delay_ms(30);


                PWM_period_ms(&dcPwm[A], 40);
                PWM_period_ms(&dcPwm[B], 40);

                PWM_duty(&dcPwm[A], 1);
                PWM_duty(&dcPwm[B], 1);
                       GPIO_write(dcDirPin[A].port, dcDirPin[A].Pin, 1);
                       GPIO_write(dcDirPin[B].port, dcDirPin[B].Pin, 1);
                   delay_ms(100);


                         }
                            else {


       printf("IR1 = %d \r\n",IR1);
       printf("IR2 = %d \r\n",IR2);
       printf("\r\n");

             PWM_period_ms(&dcPwm[A], 40); //R
             PWM_period_ms(&dcPwm[B], 40); //L

             PWM_duty(&dcPwm[A], 0.7);
             PWM_duty(&dcPwm[B], 0.7);

             GPIO_write(dcDirPin[A].port, dcDirPin[A].Pin, 1);
         GPIO_write(dcDirPin[B].port, dcDirPin[B].Pin, 1);
```

```
        if (IR1 > 1000){
           printf("GO LEFT\r\n");

           PWM_period_ms(&dcPwm[A], 40);
        PWM_period_ms(&dcPwm[B], 40);

        PWM_duty(&dcPwm[A], 0.5);
        PWM_duty(&dcPwm[B], 1);

         GPIO_write(dcDirPin[A].port, dcDirPin[A].Pin, 1);
           GPIO_write(dcDirPin[B].port, dcDirPin[B].Pin, 1);
              delay_ms(10);
        }

        if (IR2 > 1000) {
           printf("GO RIGHT\r\n");
        printf("\r\n");


           PWM_period_ms(&dcPwm[A], 40);
        PWM_period_ms(&dcPwm[B], 40);

        PWM_duty(&dcPwm[A], 1);
        PWM_duty(&dcPwm[B], 0.5);

         GPIO_write(dcDirPin[A].port, dcDirPin[A].Pin, 1);
         GPIO_write(dcDirPin[B].port, dcDirPin[B].Pin, 1);
              delay_ms(10);
        }
        }
        }

     }


  // Initialiization
  void setup(void)
  {
     RCC_PLL_init();                         // System Clock = 84MHz
     UART2_init();
     SysTick_init(1);
     TIM_INT_enable(TIM4);

     GPIO_pupd(GPIOA, 8, 0x00);
     GPIO_pupd(GPIOB, 6, 0x00);
     GPIO_otype(GPIOA, 8, 0);
     GPIO_ospeed(GPIOA, 8, 2);

     // ADC setting
      ADC_init(GPIOB, 0, TRGO);
     ADC_init(GPIOB, 1, TRGO);

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

     // ADON, SW Trigger enable
     ADC_start();
```

```
    // PWM configuration ---------------------------------------------------------
-------------

    PWM_t trig;                                  // PWM1 for trig
   PWM_init(&trig, GPIOA, 8);                       // 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 --------------------------------------------------
---------------------
  IC_t echo;                                 // Input Capture for echo
   ICAP_init(&echo, GPIOB, 6);          // PB10 as input caputre
    ICAP_counter_us(&echo, 10);         // ICAP counter step time as 10us
   ICAP_setup(&echo, 1, IC_RISE);      // TIM2_CH3 as IC3 , rising edge detect
   ICAP_setup(&echo, 2, IC_FALL);      // TIM2_CH3 as IC4 , falling edge detect


         PWM_init(&dcPwm[A], dcPwmPin[A].port, dcPwmPin[A].Pin);
         PWM_init(&dcPwm[B], dcPwmPin[B].port, dcPwmPin[B].Pin);



   for (int i = 0; i < 2; i++){
      GPIO_init(dcDirPin[i].port, dcDirPin[i].Pin,OUTPUT);
      GPIO_pupd(dcDirPin[i].port, dcDirPin[i].Pin, 01);
      GPIO_otype(dcDirPin[i].port, dcDirPin[i].Pin, 0);
      GPIO_ospeed(dcDirPin[i].port, dcDirPin[i].Pin, 11);
   }
}

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

   if(is_ADC_EOC()){       //after finishing sequence
         if (flag==0){
            IR1 = ADC_read();
         }
         else if (flag==1){
            IR2 = ADC_read();
         }
      flag =! flag;
   }
}


void TIM4_IRQHandler(void){
   if(is_UIF(TIM4)){
      ovf_cnt++;
      clear_UIF(TIM4);
   }
```

```
    if(is_CCIF(TIM4, 1)){
        time1 = TIM4->CCR1;
        clear_CCIF(TIM4, 1);
    }
    else if(is_CCIF(TIM4, 2)){
        time2 = TIM4->CCR2;
        timeInterval = ( (time2 - time1) + ( (TIM4->ARR) + 1 )*ovf_cnt );
        ovf_cnt = 0;
        clear_CCIF(TIM4, 2);
    }
}
```

code discription

main function

Find the distance using timeInterval in the while statement. Stop the DC motor when the distance measured by the UltraSensor is less than 10 cm. In normal times, adjust the PWM value to advance at a slow speed. When the IR sensor detects a color change, it adjusts the PWM value of the left and right DC motors to rotate.

setup

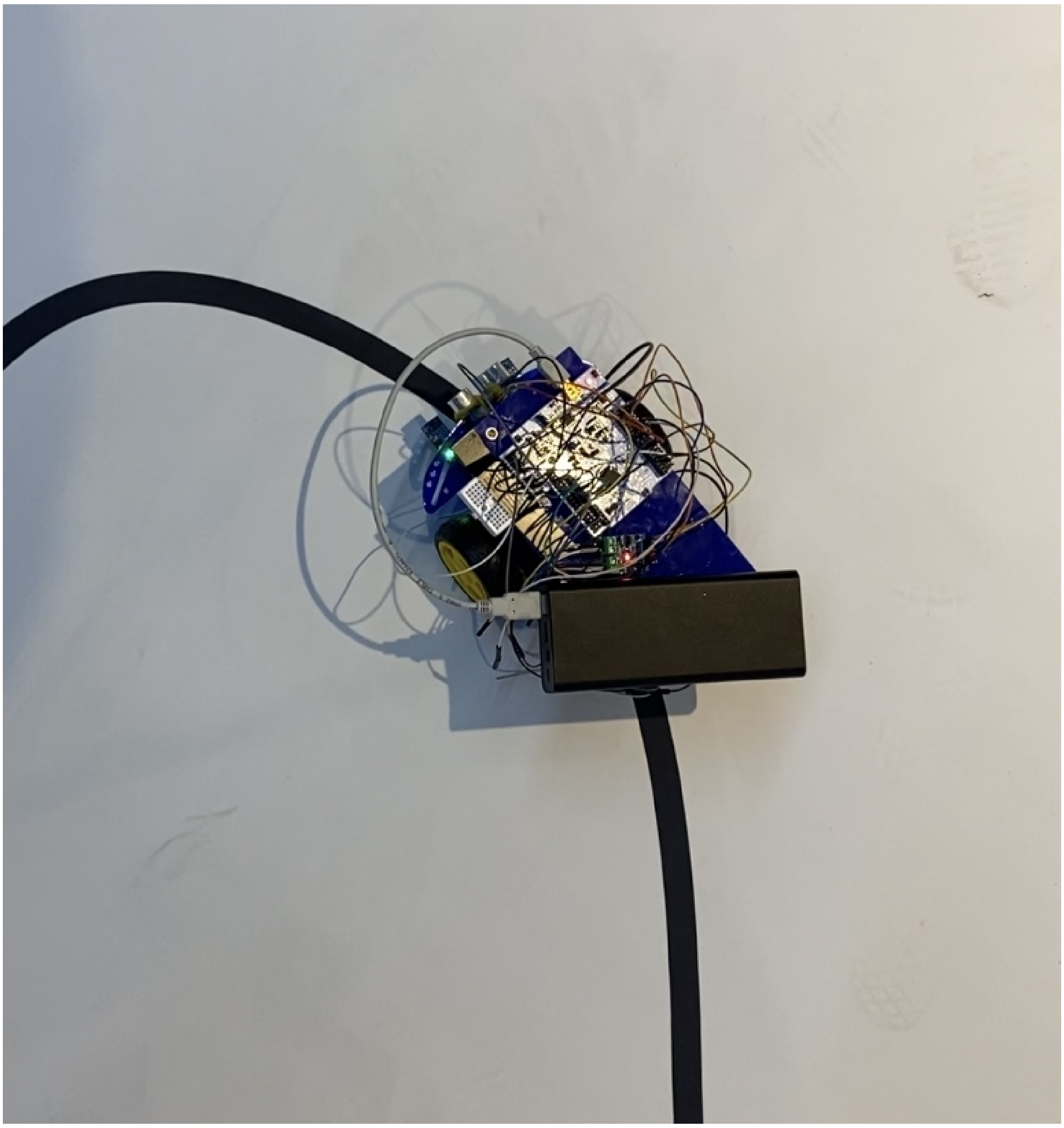
Perform basic initialization. ADC settings, PWM configuration, Input Capture configuration, and PWM initialization.

Results

Experiment images and results

> Display results

```
raw_distance = 90.440002 [cm]
IR1 = 222
IR2 = 3993

GO RIGHT

raw_distance = 81.599998 [cm]
IR1 = 216
IR2 = 4095

GO RIGHT

raw_distance = 81.430000 [cm]
IR1 = 4095
IR2 = 269

GO LEFT
raw_distance = 72.589996 [cm]
IR1 = 4095
IR2 = 281

GO LEFT
raw_distance = 72.080002 [cm]
IR1 = 4095
IR2 = 283

GO LEFT
raw_distance = 71.739998 [cm]
IR1 = 228
IR2 = 4095

GO RIGHT

raw_distance = 82.790001 [cm]
IR1 = 223
IR2 = 4095

GO RIGHT

raw_distance = 88.739998 [cm]
IR1 = 2456
IR2 = 311

GO LEFT
raw_distance = 71.059998 [cm]
IR1 = 4095
IR2 = 268

GO LEFT
raw_distance = 70.889999 [cm]
IR1 = 4095
IR2 = 4095

GO LEFT
GO RIGHT

raw_distance = 476.850006 [cm]
```



Demo video Link : https://youtu.be/jyC4ZVEwGiI

Reference

Complete list of all references used (github, blog, paper, etc)

Troubleshooting

1. There was a problem that occurred while setting several timers. The pin allocated to the new timer was selected by referring to the pinmap of the existing code file and connected to the MCU.
2. Selection of PWM period and duty ratio suitable for our device was important in order to rotate the specified course to a stable state. In this experiment, a heavy auxiliary battery was attached to this device. Since it takes a large load, it was possible to set an appropriate value for try and error.
3. Using an ultrasonic sensor, the motor was stopped when the obstruction approached. If there is too much delay in the main function, it stops narrowly or hits an obstacle. Therefore, the code had to be devised to reduce the delay as much as possible and stop the obstruction as soon as it sensed it