Intelligent fire suppression sprinkler

LAB: EC Design Problem

Date: 2023-12-19

Author/Partner: YunKi Noh / EunChan Kim

Demo Video: Demo Video

I. Introduction

Overview

Sprinklers installed in the factory, when detecting a fire, extinguish all areas within the sprinkler's range. This leads to the problem of areas not affected by the fire, including equipment and materials, getting soaked, resulting in economic losses.

Therefore, we aim to create a concentrated spray-type sprinkler that can suppress only the fire point. Based on fire detection sensors and motors, the fire point is detected, and only the specific point is sprayed with extinguishing agent for concentrated suppression. By spraying extinguishing agent only in areas requiring suppression, we can protect the equipment.

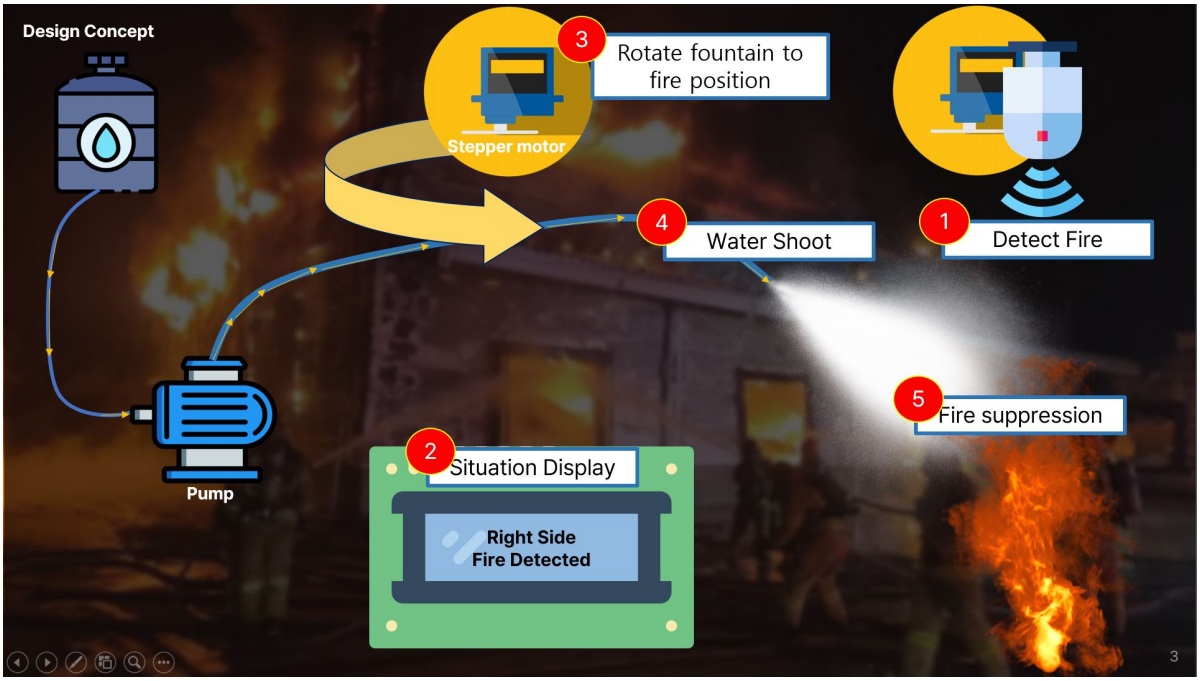

Figure1. Concept Design

Requirements

Hardware

Item	Model/Description	Qty
STM MCU	NUCLEO -F411RE	1
Arduino MCU	Arduino Uno R3	2
Analog Sensor	Fire Detection Sensor(NS-FDSM)	1
Digital Sensor	Fire Detection Sensor(NS-FDSM)	1
Actuator	Servo Motor(SG90)	2
Display	LCD(SZH-EK101)	1
Communication	Bluetooth Module(SZH-EK010)	1
Water Pump	SZH-GNP155	1
Ultra Sonic Sensor	HC-SR04	1
Buzzer	FQ-010	1
Motor Driver	L9110s	1

Software

- Keil uVision IDE
- Arduino IDE
- EC_HAL

II. Problem

Problem Description

We have physically implemented three modes for fire detection and water dispersal. The first is the auto mode, in which the motor traverses its entire operational range, with the attached sensor automatically detecting a fire. The second mode is manual, which allows the user to manually control the motor to spray water at a desired location. The third mode sends a notification to refill the water when the water level in the tank is low.

1. **Auto Mode**

MODE	Description
Auto Mode	By pressing the 'N' button, the system switches to auto mode. The motor then traverses all areas for fire detection.

2. **Manual Mode**

MODE	Description
Manual Mode	Press 'M' to activate manual mode. Use the 'A, S, D, W' buttons to control up, down, left, and right movements.

3. Water Level Alarm

MODE	Description
Water Level	The ultrasonic sensor measures the water level in the tank and sends an alarm to refill the water with LCD when it drops below a certain level.

Sensor Unit: MCU_1

Function	Sensor	Configuration	Comments
Fire Detection	Fire Detection Sensor	It receives sensor information at intervals of 0.05 seconds and detects fire.	In auto mode, it uses both analog and digital sensors, detecting a wide range of fire locations with the analog fire detection sensor and pinpointing precise fire locations with the digital fire detection sensor.
Water Level Check	Ultra Sonic Sensor	When a rising edge is detected, it retrieves the distance value.	

MCU Configuration

Functions	Register	PORT_PIN	Configuration
System Clock	RCC	-	PLL 84MHz
delay_ms	SysTick	-	
Bluetooth	USART1	TXD: PA9 RXD: PA10	No Parity, 8-bit Data, 1-bit Stop bit 9600 baud-rate
Timer Interrupt	TIMER2	PA5	1sec period, priority 3
Timer Interrupt	TIMER3	PA1	0.5sec period, priority 2
Timer Interrupt	TIMER4	PA6, PB6	0.5sec period, priority 4

Circuit Diagram

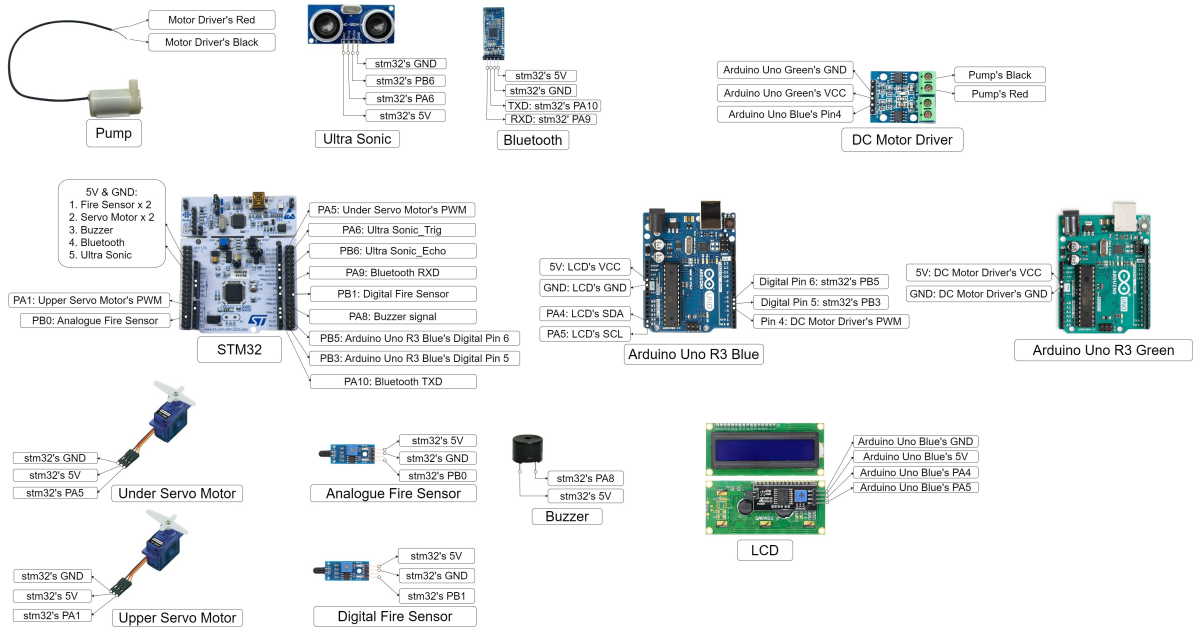

Figure2. Circuit Diagram

III. Algorithm

Logic Design

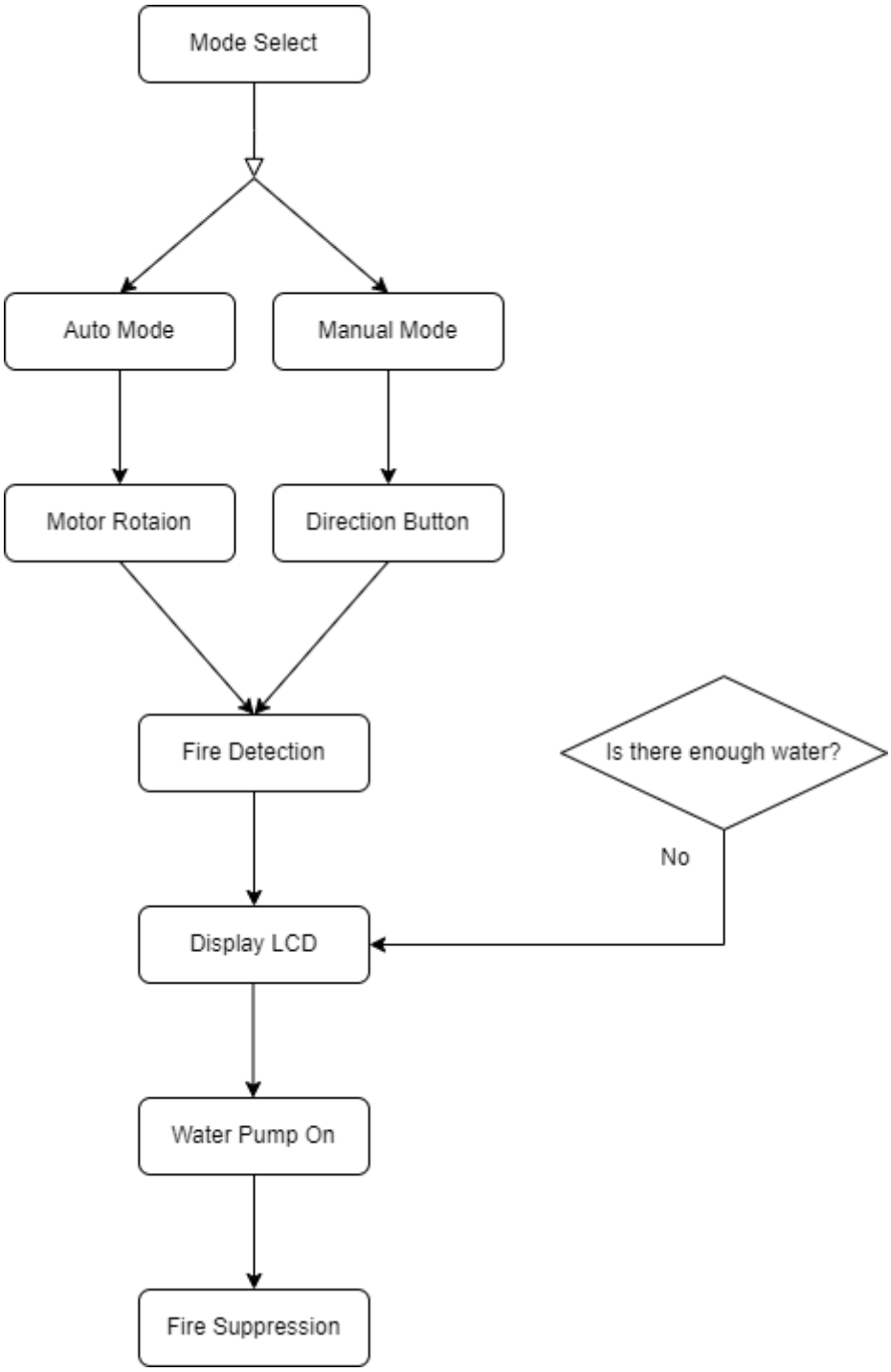

Figure3. Flow Chart

The logic is composed of two modes: auto and manual.

Auto Mode

Auto Mode can be accessed by pressing the 'N' button. When Auto Mode is selected, two servo motors operate within the detectable area. The servo motors are combined vertically, with the top motor responsible for the left and right areas and the bottom motor adjusting the up and down. They move in all directions to detect a fire in front.

The analog fire detection sensor increases its value as it gets closer to the flame. Conversely, the digital sensor changes its value when it aligns precisely with the fire point. Using these two features, the analog fire detection sensor first detects fire points in a wide area. Once detected, the speed of the motor's vertical movement is increased to detect the area more finely.

If the value changes in the digital sensor, it is determined that the precise fire point within the wide area has been detected, and the water pump is operated to suppress the fire. At this point, it sends a HIGH signal to the Arduino board indicating that a fire has been detected, and the LCD is activated.

Manual Mode

Manual Mode is activated by pressing the 'M' button. The analog and digital sensors operate to detect a fire, just like in Auto Mode. The difference with Auto Mode is that the user can access the fire point using the keyboard and move to the desired location. If a fire is detected through the sensor, the pump automatically operates.

Water Level Check

The water level is measured through the ultrasonic sensor installed in the water tank. If the water level drops below a certain value, the pump cannot draw water, so a message to refill the water is sent through the LCD.

Code

initialization

The variables necessary for operating the sensor have been declared.

```
#include "ecSTM32F411.h"

//----------------------------Fire Detection Value----------------------------//
//uint32_t Fire_Detection; // fire sensor value
uint32_t Fire_Signal = 0;  // signal for fire detection


//----------------------------Servo Motor cnt----------------------------//
int cnt = 0;    // count value for upper servo motor's working
int cnt2 = 0;   // count value for under servo motor's working
int cnt_u = 0;  // count value for manual mode's under direction
int cnt_r = 0;  // count value for manuala mode's right direction


//----------------------------Ultra Sonic Virable----------------------------//
uint32_t ovf_cnt = 0;
float distance = 0;
float timeInterval = 0;
float time1 = 0;
float time2 = 0;


//----------------------------Define Variable----------------------------//
#define PWM_PIN1 PA_1 // Upper servo
#define PWM_PIN2 PA_5 // Under servo

#define TRIG PA_6     // Ultra Sonic's Trig pin
#define ECHO PB_6     // Ultra Sonic's Echo pin

#define FIRE1 PB_0    // Fire Sensor's Analog pin
```

```
#define FIRE2 PB_1    // Fire Sensor's Digital pin


//----------------------------IR Variable----------------------------//
static volatile uint32_t ADC_Analogue, ADC_Digital; // Analogue value & Digital
value of Fire Sensor
int flag = 0;                                                    // flag
for reading two IR Sensor's values
PinName_t seqCHn[2] = {PB_0, PB_1};                        // Array for two IR
Sensor's value

static volatile uint8_t BT_Data = 0; // Bluetooth data
static volatile char mode;                 // Mode of Fire Sprinkler


//----------------------------Function Declaration----------------------------//
void setup(void);                              // Setting Function
void ADC_IRQHandler(void);                  // Fire Sensor Function
void EXTI15_10_IRQHandler(void);       // Button Function
void ServoSetting();                           // Setting for Servo Motor Function
void TIM3_IRQHandler(void);              // Upper Servo Motor Function
void TIM2_IRQHandler(void);              // Under Servo Motor Function
```

main

In the main code, it uses the ultrasonic sensor to detect distance, and when the distance exceeds 6.5cm, it sends a HIGH signal to the Arduino (PB5) under the assumption that the water level is low and the pump will not operate. In the opposite case, it sends a LOW signal to stop the operation of the LCD. It also prints the values from the fire detection sensors and the mode.

```
int main(void) {

   // Initialiization --------------------------------------------------------
   setup();

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

      /*-------------------Ultra Sonic Working--------------------*/
      distance = (float) timeInterval * 0.034 / 2.0;    // [mm] -> [cm]

       if(distance>=1 && distance<=6.5){

          if(distance>=6.0){
               GPIO_write(GPIOB,5,1);
            }

            else if(distance < 6.0){
               GPIO_write(GPIOB,5,0);
            }
      }

      /*--------------------Printing Sector--------------------*/
```

```
        /*----------ADC----------*/
        printf("/*-----ADC value*-----/ \r\n");
        printf("Fire Analogue: %d \r\n", ADC_Analogue);
        printf("Fire Digital: %d \r\n", ADC_Digital);
        printf("\r\n");

        /*----------Mode----------*/
        printf("/*-----Current Mode-----*/ \r\n");
        if(mode == 'n' || mode == 'N') printf("Current Mode: Auto Mode \r\n");
        else if(mode == 'm' || mode == 'M') printf("Current Mode: Manual Mode
\r\n");
        printf("\r\n");
        delay_ms(1000);
    }
}
```

Initialization

1. **Clock setting and timer initialization**: Set the system clock to 84MHz through the `RCC_PLL_init()` function and initialize system time-related functions through `SysTick_init()`.
2. **ADC Initialization**: Initializes the analog-to-digital converter (ADC) and sets two pins (`PB_0` and `PB_1`) as ADC input pins.
3. **GPIO Settings**: Set the digital input/output pins and set the initial value to LOW. Arduino pins (`PA_6` of `GPIOA`, `PB_5`, `PB_3`, `PA_8` of `GPIOB`) correspond to this.
4. **Serial communication initialization**: Initialize UART1 and UART2, and set each communication speed to 9600. Also, configure USART1 by connecting it to specific pins (pins 9 and 10 of `GPIOA`).
5. **Stepper and servo motor settings**: Set the stepper motor and servo motor. Within the code they are marked with comments saying "Stepper Setting" and "Servo Setting".
6. **Ultrasonic and PWM Settings**: Set the ultrasonic sensor and PWM settings to set the PWM signal for the `TRIG` pin and input capture for the `ECHO` pin. Used to process trigger and echo signals from ultrasonic sensors.
7. **Buzzer Settings**: Initialize the buzzer and set the corresponding pin.

```
// Initialiization
void setup(void)
{
      RCC_PLL_init(); // System Clock = 84MHz
      SysTick_init(); // SysTick Init

      // ADC Init
      ADC_init(PB_0); // Analogue Pin
      ADC_init(PB_1); // Digital Pin

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

      // BT serial init
      UART1_init();
```

```
	UART1_baud(BAUD_9600);

	UART2_init();
	UART2_baud(BAUD_9600);

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

	// Servo setting
	ServoSetting();

	// Ultra Sonic 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 conf iguration ----------------------------------------
-------------------------
	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

	// Sending Signal from stm32 to Arduino by using Output pin(stm32) & Input
pin(Arduino)
	GPIO_setting(GPIOB,5,OUTPUT,EC_PUSH_PULL,EC_PU,EC_MEDIUM);
	GPIO_write(GPIOB,5,LOW); //water
	GPIO_setting(GPIOB,3,OUTPUT,EC_PUSH_PULL,EC_PU,EC_MEDIUM);
	GPIO_write(GPIOB,3,LOW); // fire

	//Buzzer
	GPIO_setting(GPIOA,8,OUTPUT,EC_PUSH_PULL,EC_PU,EC_MEDIUM);
	GPIO_write(GPIOA,8,LOW);
}
```

ADC

If the value from the analog fire detection sensor exceeds a certain threshold, it's determined that a fire has been detected and the speed of the motor is increased. If a fire is detected by the digital sensor, all motor timers are stopped to allow the sensor to face the location of the fire.

Upon detection of a fire, a HIGH signal is sent to provide a flag for displaying a warning message on the Arduino, and the water pump is also activated.

1. **Overflow handling**: Use the `is_ADC_OVR()` function to check whether an ADC overflow has occurred, and if so, clear the overflow through the `clear_ADC_OVR()` function.
2. **Check completion of conversion**: Use the `is_ADC_EOC()` function to check whether the ADC conversion is complete.
3. **Read ADC value**: When conversion is complete, read the analog value through the `ADC_read()` function.
4. **Action according to condition**: Performs a specific action according to the read analog value. For example, if the `ADC_Analogue` value exceeds 2000, change the `deno`, `range`, and `mole` values.

5. **Conditional branching based on ADC value**: Checks multiple conditions according to the `ADC_Digital` value and performs the corresponding action. For example, if it is below 1000, disable the timer and set a HIGH level on that specific pin. By sending a HIGH signal to the Arduino, it causes a fire alarm to be displayed on the LCD.
6. **Flag Toggle**: Toggle the `flag` variable to determine which ADC value will be read next.

```
// ADC Interrupt
void ADC_IRQHandler(void){
   if(is_ADC_OVR())
      clear_ADC_OVR();

   if(is_ADC_EOC()){                // after finishing sequence
      if (flag==0)
         ADC_Analogue = ADC_read(); // upper_Analogue
      else if (flag==1)
         ADC_Digital = ADC_read();  // under_Digital

      if(ADC_Analogue > 2000){ // Fire not detected
      }
      else if(ADC_Analogue <= 2000){
         if(ADC_Digital < 1000){
               TIM_UI_disable(TIM2);
               TIM_UI_disable(TIM3);

      }
        else if(ADC_Digital >= 1500){ // Fire not detected
               TIM_UI_enable(TIM2);
               TIM_UI_enable(TIM3);
               GPIO_write(GPIOB,3, LOW); // Send LOW signal to Arduino
               GPIO_write(GPIOA,8,LOW);  // Turn off Buzzer
      }
      }

    if(ADC_Digital < 1000){ // Fire Detected
         TIM_UI_disable(TIM2);       // Pause Under Servo Motor
         TIM_UI_disable(TIM3);       // Pause Over Servo Motor
         GPIO_write(GPIOB,3, HIGH);  // Send HIGH signal to Arduino
         GPIO_write(GPIOA,8,HIGH);   // Turn on Buzzer
    }
    else { // Fire not Detected
         TIM_UI_enable(TIM2);           // Working Uner Servo Motor
         TIM_UI_enable(TIM3);           // Wokring Over Servo Motor
         GPIO_write(GPIOA,8,LOW);     // Send LOW signal to Arduino
         GPIO_write(GPIOB,3, LOW);   // Turn off Buzzer
    }

      flag =! flag;      // flag toggle
   }
}
```

EXTI

When the button is pressed, it adjusts the 'PWM_duty' to return Servo Motor to the original position.

```
void EXTI15_10_IRQHandler(void) {
   if (is_pending_EXTI(BUTTON_PIN)) {
        // Stop Servo Motor
        PWM_duty(PWM_PIN1, (float)0.025);
        PWM_duty(PWM_PIN2, (float)0.025);
        cnt = 0;
      clear_pending_EXTI(BUTTON_PIN); // cleared by writing '1'

   }
}
```

Servo Motor

1. **Servo and LED/BUTTON pin settings**:
   - Using the `GPIO_init()` function, set pin 1 of GPIOA and pin 8 of GPIOB to AF (Alternative Function) and initialize them as pins for the servo motor.
   - Also set the LED and BUTTON pins to initialize GPIOA's `LED_PIN` for output and GPIOC's `BUTTON_PIN` for input.
2. **Timer Settings**:
   - Initialize TIM3 and TIM2 using the `TIM_init()` function. TIM3 is set to a 500ms period, and TIM2 is set to a 1000ms period.
   - Activate the Update Interrupt of each timer and activate the interrupt request of the corresponding timer through NVIC. Also sets the interrupt priority.
3. **External interrupt settings**:
   - Use the `EXTI_init()` function to convert pin 13 of the GPIOC to the EXTI signal.

```
void ServoSetting(){

   // Servo pin setting
    GPIO_init(GPIOA, 1, EC_AF);
    GPIO_init(GPIOA, 5, EC_AF);

    // LED/BUTTON pin setting
    GPIO_init(GPIOC, BUTTON_PIN, INPUT);

    // Timer setting
    TIM_init(TIM3, 500);            // TIMER period = 500ms
    TIM3->DIER |= 1;                // Update Interrupt Enabled
    NVIC_EnableIRQ(TIM3_IRQn);      // TIM3's interrupt request enabled
    NVIC_SetPriority(TIM3_IRQn, 2); // set TIM3's interrupt priority as 2

    TIM_init(TIM2, 500);                  // TIMER period = 500ms
    TIM2->DIER |= 1;                       // Update Interrupt Enabled
    NVIC_EnableIRQ(TIM2_IRQn);      // TIM2's interrupt request enabled
    NVIC_SetPriority(TIM2_IRQn, 3); // set TI2's interrupt priority as 3
```

```
    // Button Setting
    EXTI_init(GPIOC, 13, FALL, 0);       // set C_port's 13_pin as signal of EXTI
    NVIC_EnableIRQ(EXTI15_10_IRQn);      // enable request interrupt
    NVIC_SetPriority(EXTI15_10_IRQn, 3); // set priority

    // PWM setting
    PWM_init(PWM_PIN1);           // set PA1 PWM's output pin
    PWM_period(PWM_PIN1, 20);  // 20 msec PWM period

    PWM_init(PWM_PIN2);           // set PA5 as PWM's output pin
    PWM_period(PWM_PIN2, 20);  // 20 msec PWM period
}
```

TIM3

As `TIM3` is activated, it causes the upper servo motor to move left and right. When it reaches 180 degrees, it rotates the motor in the opposite direction, continuing to move the motor left and right.

1. **Check interrupt flag**: Check if the Update Interrupt Flag of TIM3 is set.
2. **Servo motor control**:
    - Used to rotate the servo motor from 0 to 180 degrees.
    - Use the `PWM_duty()` function to adjust the duty cycle of PWM_PIN1. This value changes depending on changes in the `cnt` variable.
    - Determines the rotation direction of the motor according to the `cnt` value and controls it to rotate in the opposite direction at 0 degrees and 180 degrees.
3. **Clear Interrupt Flag**: When the task is finished, clear the Update Interrupt Flag of TIM3 to put the next interrupt into a waiting state.

```
/*-------------Over Servo Motor-------------*/
void TIM3_IRQHandler(void){
   if((TIM3->SR & TIM_SR_UIF) == 1){
      /*----0.5ms = 0dgree, 1.5ms = 90degree, 2.5ms = 180degree----*/
         if(mode == 'N'){

      static int direction1 = 1;// Add a direction variable to control the
movement

      PWM_duty(PWM_PIN1, (float)0.025*(cnt*4/100.0 + 1)); // Working

      cnt = cnt + direction1; // coutning

      if(cnt >= 100.0) direction1 = -1;  // Rotate to the left when reaching 180
degrees
      else if(cnt <= 0) direction1 = 1;  // Rotate to the right when reaching 0
degrees
         }
      // clear by writing 0
      TIM3->SR &= ~ TIM_SR_UIF;
   }
}
```

TIM2

As `TIM2` is activated, it engages the lower motor. Similar to the upper motor, it alternates motion in the vertical direction. The duty value was adjusted differently from the upper motor to control the range of vertical motion.

1. **Check interrupt flag**: Check whether an interrupt occurred by checking the Update Interrupt Flag of TIM2.
2. **Servo motor control**:
    - This code also controls the servo motor, just like TIM3's handler.
    - Use the `PWM_duty()` function to adjust the duty cycle of PWM_PIN2. Controls the rotation of the motor according to the value of the `cnt2` variable.

```
/*-------------Under Servo Motor-------------*/
void TIM2_IRQHandler(void){
   if((TIM2->SR & TIM_SR_UIF) == 1){
       /*----0.5ms = 0dgree, 1.5ms = 90degree, 2.5ms = 180degree----*/
       if(mode == 'N'){

        static int direction2 = 1; // Add a direction variable to control the
movement

        PWM_duty(PWM_PIN2, (float)0.05*(cnt2*0.5/30 + 1)); // 0.06
        cnt2 = cnt2 + direction2; // counting

        if(cnt2 >= 30) direction2 = -1;  // Rotate to the left when reaching 180
degrees
        else if(cnt2 <= 0) direction2 = 1;  // Rotate to the right when reaching
0 degrees
       }

      // clear by writing 0
      TIM2->SR &= ~ TIM_SR_UIF;
    }
}
```

TIM4

When `TIM4` is activated, it fetches the input and output values of the ultrasonic waves using the echo and trigger pins. It calculates the time interval to provide a basis for calculating distance in the main script.

1. **Update Interrupt Handling**:
    - Checks for the update interrupt flag (`UIF`) of TIM4. Increments an overflow count variable (`ovf_cnt`) and clears the update interrupt flag.
2. **Input Capture Handling**:
    - Checks for two capture/compare interrupt flags (`CCIF`) associated with TIM4 Channel 1 (IC1) and Channel 2 (IC2).

- When the rising edge is detected (`is_CCIF(TIM4, 1)`), it captures the timestamp (`time1`) and clears the corresponding capture/compare interrupt flag.
- On detecting the falling edge (`is_CCIF(TIM4, 2)`), it captures another timestamp (`time2`). Then, it calculates the time interval (`timeInterval`) between the rising and falling edges of the echo pulse using the captured timestamps and the overflow count.

```
// Ultra Sonic
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
   }
}
```

USART

In the `USART1_IRQHandler`, it sets the manual or auto mode for the motor using Bluetooth communication and communicates the actions to be taken according to the manual mode.

1. **Data Reception**:
   - Reads the received data using `USART1_read()` and stores it in the `BT_Data` variable.
   - Echoes back the received data by writing it back through `USART1_write()`.
2. **Mode Selection**:
   - Changes the mode based on certain received characters:
     - 'm' or 'M': Sets `mode` to 'M', disables TIM3 and TIM2.
     - 'n' or 'N': Sets `mode` to 'N', enables TIM3 and TIM2.
3. **Servo Control**:
   - Controls servo motors based on specific characters:
     - 'S' or 's': Rotate the motor to Down side. Increases the `cnt_u` variable and adjusts the duty cycle of PWM_PIN2 to control the servo position.
     - 'W' or 'w': Rotate the motor to Up side. Decreases the `cnt_u` variable and adjusts the duty cycle of PWM_PIN2.

- 'D' or 'd': Rotate the motor to Right side. Increases the `cnt_r` variable and adjusts the duty cycle of PWM_PIN1.
- 'A' or 'a': Rotate the motor to Left side. Decreases the `cnt_r` variable and adjusts the duty cycle of PWM_PIN1.

```
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';
            TIM_UI_disable(TIM3);
            TIM_UI_disable(TIM2);
      }
         else if(BT_Data=='n'||BT_Data=='N'){
            mode='N';
            TIM_UI_enable(TIM3);
            TIM_UI_enable(TIM2);
         }

      else if(BT_Data=='S'||BT_Data=='s'){
         cnt_u++;
         //if(cnt_u > 50) cnt_u = 50; // Limit the maximum value to prevent
exceeding the servo range
         PWM_duty(PWM_PIN2, (float)0.025*(cnt_u*1/50.0 + 1));
      }
      else if(BT_Data=='W'||BT_Data=='w'){
         cnt_u--;
         //if(cnt_u < 0) cnt_u = 0; // Limit the minimum value to prevent
exceeding the servo range
         PWM_duty(PWM_PIN2, (float)0.025*(cnt_u*1/50.0 + 1));
      }
      else if(BT_Data=='D'||BT_Data=='d'){
            cnt_r++;
         //if(cnt_r > 50) cnt_r = 50; // Limit the maximum value to prevent
exceeding the servo range
         PWM_duty(PWM_PIN1, (float)0.025*(cnt_r*1/50.0 + 1));
      }
      else if(BT_Data=='A'||BT_Data=='a'){
         cnt_r--;
         //if(cnt_r < 0) cnt_r = 0; // Limit the minimum value to prevent
exceeding the servo range
         PWM_duty(PWM_PIN1, (float)0.025*(cnt_r*1/50.0 + 1));
      }
   }
}
```

Arduino

Through the STM board, it receives fire alarms or water level warning messages based on sensor values. It displays different warning messages depending on the pin that receives the HIGH signal.

1. **Library and Pin Initialization**:
    - Includes necessary libraries (`Wire.h` for I2C communication and `LiquidCrystal_I2C.h` for LCD display).
    - Specifies a digital pin `6` as the pin connected to the Flame Sensor.
    - Initializes the `sensorValue` variable to store the sensor reading.
    - Sets up the LiquidCrystal display with its address (`0x27`) and size (`20` columns by `4` rows) and initializes it.
2. **Setup Function**:
    - Starts serial communication at a baud rate of `9600`.
    - Configures the digital pin connected to the Flame Sensor as an input.
    - Initializes and turns on the backlight for the LCD display.
3. **Loop Function**:
    - Reads the digital value from the specified pin (`digitalPin`) where the Flame Sensor is connected.
    - Clears the LCD display to prepare for the new output.
    - Checks the sensor value:
        - If the sensor reads `HIGH` (value of `1`), it displays "Warning!" and "Fire Detection!" on the LCD.
        - If the sensor reads `LOW` (value of `0`), it displays "Safe" on the LCD.
    - Delays the program for `1000` milliseconds (1 second) before looping and reading the sensor value again.

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

const int digitalPin = 6;   // 디지털 핀 번호 지정
int sensorValue;           // 읽은 값을 저장할 변수

LiquidCrystal_I2C lcd(0x27, 20, 4);

void setup() {
  Serial.begin(9600);       // 시리얼 통신 시작
  pinMode(digitalPin, INPUT);  // 디지털 핀을 입력 모드로 설정

  lcd.init();  // LCD 초기 설정
  lcd.backlight(); // LCD 초기 설정
}

void loop() {
  // 디지털 핀에서 값을 읽어옴
  sensorValue = digitalRead(digitalPin);

  // LCD에 값 출력
  lcd.clear();  // 이전 값 지우기

  // SensorValue가 1이면 "Warning!"과 "Fire Detection!" 출력, 그렇지 않으면 "Safe" 출
력
```

```
  if (sensorValue == 1) {
    lcd.setCursor(0, 0);
    lcd.print("Warning!");
    lcd.setCursor(0, 1);
    lcd.print("Fire Detection!");
  } else {
    lcd.setCursor(0, 0);
    lcd.print("Safe");
  }

  delay(1000);  // 1초 대기
}
```

IV. Results and Demo

As can be seen in the video, both auto and manual modes are confirmed to operate properly. The motor rotates, scanning all areas in front of the sensor, and when a fire is detected by the fire detection sensor, the motor stops to focus on the location of the fire. Subsequently, the motor operates while a fire alarm is displayed on the LCD to extinguish the fire.

Analyze

It was found through experimentation that when the size of the flame decreases to about 1cm, it is not detected by the fire detection sensor, causing a problem. It is believed that this could be improved by replacing it with a fire detection sensor that has a further detection range.

In addition, it was observed that the water was being sprayed off target even when the exact point was detected, due to the misalignment of the fire detection sensor and the water hose. This issue could be resolved by creating a hardware structure to precisely align the sensor and the hose.

Lastly, a problem occurred where water continued to flow even when the water pump stopped. Upon inspection, it was confirmed that water passes through the inlet and outlet even when the power to the pump used is cut off. If a pump that can completely stop the flow of water is used, it is expected that water can be sprayed only at the point when a fire is detected.

V. Reference

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

Appendix

entire code

```
/*------------------------
/----------------------------------------------------------------/
- Class: 23-2 Embeded Contorller Final Project
- Team Students: EunChan Kim(21801017), YunKi Noh(21800226)
- Project name: Intelligent Fire Suppression Sprinkler
- Advisor: YungKeun Kim Professor
- Date: 23/12/19
/----------------------------------------------------------------/
---------------------------*/

#include "ecSTM32F411.h"

//----------------------------Fire Detection Value----------------------------//
//uint32_t Fire_Detection; // fire sensor value
uint32_t Fire_Signal = 0;  // signal for fire detection


//----------------------------Servo Motor cnt----------------------------//
int cnt = 0;    // count value for upper servo motor's working
int cnt2 = 0;   // count value for under servo motor's working
int cnt_u = 0;  // count value for manual mode's under direction
int cnt_r = 0;  // count value for manuala mode's right direction


//----------------------------Ultra Sonic Virable----------------------------//
uint32_t ovf_cnt = 0;
```

```
float distance = 0;
float timeInterval = 0;
float time1 = 0;
float time2 = 0;


//----------------------------Define Variable----------------------------//
#define PWM_PIN1 PA_1 // Upper servo
#define PWM_PIN2 PA_5 // Under servo

#define TRIG PA_6     // Ultra Sonic's Trig pin
#define ECHO PB_6     // Ultra Sonic's Echo pin

#define FIRE1 PB_0    // Fire Sensor's Analog pin
#define FIRE2 PB_1    // Fire Sensor's Digital pin


//----------------------------IR Variable----------------------------//
static volatile uint32_t ADC_Analogue, ADC_Digital; // Analogue value & Digital
value of Fire Sensor
int flag = 0;                                                    // flag
for reading two IR Sensor's values
PinName_t seqCHn[2] = {PB_0, PB_1};                        // Array for two IR
Sensor's value

static volatile uint8_t BT_Data = 0; // Bluetooth data
static volatile char mode;                 // Mode of Fire Sprinkler


//----------------------------Function Declaration----------------------------//
void setup(void);                            // Setting Function
void ADC_IRQHandler(void);                 // Fire Sensor Function
void EXTI15_10_IRQHandler(void);       // Button Function
void ServoSetting();                         // Setting for Servo Motor Function
void TIM3_IRQHandler(void);              // Upper Servo Motor Function
void TIM2_IRQHandler(void);              // Under Servo Motor Function


int main(void) {

   // Initialiization ---------------------------------------------------------
   setup();

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

      /*-------------------Ultra Sonic Working-------------------*/
      distance = (float) timeInterval * 0.034 / 2.0;    // [mm] -> [cm]

       if(distance>=1 && distance<=6.5){

          if(distance>=6.0){
              GPIO_write(GPIOB,5,1);
            }

            else if(distance < 6.0){
```

```
                GPIO_write(GPIOB,5,0);
            }
        }

        /*--------------------Printing Sector--------------------*/
        /*----------ADC----------*/
        printf("/*-----ADC value*-----/ \r\n");
        printf("Fire Analogue: %d \r\n", ADC_Analogue);
        printf("Fire Digital: %d \r\n", ADC_Digital);
        printf("\r\n");

        /*----------Mode----------*/
        printf("/*-----Current Mode-----*/ \r\n");
        if(mode == 'n' || mode == 'N') printf("Current Mode: Auto Mode \r\n");
        else if(mode == 'm' || mode == 'M') printf("Current Mode: Manual Mode
\r\n");
        printf("\r\n");
        delay_ms(1000);
    }
}

// Initialiization
void setup(void)
{
      RCC_PLL_init(); // System Clock = 84MHz
      SysTick_init(); // SysTick Init

      // ADC Init
      ADC_init(PB_0); // Analogue Pin
      ADC_init(PB_1); // Digital Pin

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

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

      UART2_init();
      UART2_baud(BAUD_9600);

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

      // Servo setting
      ServoSetting();

      // Ultra Sonic 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 conf iguration ----------------------------------------
--------------------------
      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

      // Sending Signal from stm32 to Arduino by using Output pin(stm32) & Input
pin(Arduino)
      GPIO_setting(GPIOB,5,OUTPUT,EC_PUSH_PULL,EC_PU,EC_MEDIUM);
      GPIO_write(GPIOB,5,LOW); //water
      GPIO_setting(GPIOB,3,OUTPUT,EC_PUSH_PULL,EC_PU,EC_MEDIUM);
      GPIO_write(GPIOB,3,LOW); // fire

      //Buzzer
      GPIO_setting(GPIOA,8,OUTPUT,EC_PUSH_PULL,EC_PU,EC_MEDIUM);
      GPIO_write(GPIOA,8,LOW);
}

// ADC Interrupt
void ADC_IRQHandler(void){
   if(is_ADC_OVR())
      clear_ADC_OVR();

   if(is_ADC_EOC()){                // after finishing sequence
      if (flag==0)
         ADC_Analogue = ADC_read(); // upper_Analogue
      else if (flag==1)
         ADC_Digital = ADC_read();  // under_Digital

      if(ADC_Analogue > 2000){ // Fire not detected
      }
      else if(ADC_Analogue <= 2000){
         if(ADC_Digital < 1000){
               TIM_UI_disable(TIM2);
               TIM_UI_disable(TIM3);

      }
        else if(ADC_Digital >= 1500){ // Fire not detected
               TIM_UI_enable(TIM2);
               TIM_UI_enable(TIM3);
               GPIO_write(GPIOB,3, LOW); // Send LOW signal to Arduino
               GPIO_write(GPIOA,8,LOW);  // Turn off Buzzer
      }
      }

    if(ADC_Digital < 1000){ // Fire Detected
         TIM_UI_disable(TIM2);       // Pause Under Servo Motor
         TIM_UI_disable(TIM3);       // Pause Over Servo Motor
         GPIO_write(GPIOB,3, HIGH);  // Send HIGH signal to Arduino
         GPIO_write(GPIOA,8,HIGH);   // Turn on Buzzer
    }
    else { // Fire not Detected
         TIM_UI_enable(TIM2);           // Working Uner Servo Motor
         TIM_UI_enable(TIM3);           // Wokring Over Servo Motor
         GPIO_write(GPIOA,8,LOW);     // Send LOW signal to Arduino
         GPIO_write(GPIOB,3, LOW);   // Turn off Buzzer
    }

      flag =! flag;      // flag toggle
   }
```

```
}

void EXTI15_10_IRQHandler(void) {
   if (is_pending_EXTI(BUTTON_PIN)) {
        // Stop Servo Motor
        PWM_duty(PWM_PIN1, (float)0.025);
        PWM_duty(PWM_PIN2, (float)0.025);
        cnt = 0;
      clear_pending_EXTI(BUTTON_PIN); // cleared by writing '1'

   }
}

void ServoSetting(){

   // Servo pin setting
    GPIO_init(GPIOA, 1, EC_AF);
    GPIO_init(GPIOA, 5, EC_AF);

    // LED/BUTTON pin setting
    GPIO_init(GPIOC, BUTTON_PIN, INPUT);

    // Timer setting
    TIM_init(TIM3, 500);            // TIMER period = 500ms
    TIM3->DIER |= 1;                // Update Interrupt Enabled
    NVIC_EnableIRQ(TIM3_IRQn);      // TIM3's interrupt request enabled
    NVIC_SetPriority(TIM3_IRQn, 2); // set TIM3's interrupt priority as 2

    TIM_init(TIM2, 500);                   // TIMER period = 500ms
    TIM2->DIER |= 1;                        // Update Interrupt Enabled
    NVIC_EnableIRQ(TIM2_IRQn);      // TIM2's interrupt request enabled
    NVIC_SetPriority(TIM2_IRQn, 3); // set TI2's interrupt priority as 3

    // Button Setting
    EXTI_init(GPIOC, 13, FALL, 0);       // set C_port's 13_pin as signal of EXTI
    NVIC_EnableIRQ(EXTI15_10_IRQn);      // enable request interrupt
    NVIC_SetPriority(EXTI15_10_IRQn, 3); // set priority

    // PWM setting
    PWM_init(PWM_PIN1);           // set PA1 PWM's output pin
    PWM_period(PWM_PIN1, 20);  // 20 msec PWM period

    PWM_init(PWM_PIN2);           // set PA5 as PWM's output pin
    PWM_period(PWM_PIN2, 20);  // 20 msec PWM period
}

/*-------------Over Servo Motor-------------*/
void TIM3_IRQHandler(void){
   if((TIM3->SR & TIM_SR_UIF) == 1){
      /*----0.5ms = 0degree, 1.5ms = 90degree, 2.5ms = 180degree----*/
         if(mode == 'N'){

      static int direction1 = 1;// Add a direction variable to control the
movement

      PWM_duty(PWM_PIN1, (float)0.025*(cnt*4/100.0 + 1)); // Working
```

```
      cnt = cnt + direction1; // coutning

      if(cnt >= 100.0) direction1 = -1;  // Rotate to the left when reaching 180
degrees
      else if(cnt <= 0) direction1 = 1;  // Rotate to the right when reaching 0
degrees
         }
      // clear by writing 0
      TIM3->SR &= ~ TIM_SR_UIF;
   }
}

/*-------------Under Servo Motor-------------*/
void TIM2_IRQHandler(void){
   if((TIM2->SR & TIM_SR_UIF) == 1){
       /*----0.5ms = 0dgree, 1.5ms = 90degree, 2.5ms = 180degree----*/
       if(mode == 'N'){

        static int direction2 = 1; // Add a direction variable to control the
movement

        PWM_duty(PWM_PIN2, (float)0.05*(cnt2*0.5/30 + 1)); // 0.06
        cnt2 = cnt2 + direction2; // counting

        if(cnt2 >= 30) direction2 = -1;  // Rotate to the left when reaching 180
degrees
        else if(cnt2 <= 0) direction2 = 1;  // Rotate to the right when reaching
0 degrees
       }

      // clear by writing 0
      TIM2->SR &= ~ TIM_SR_UIF;
    }
}



// Ultra Sonic
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
   }
}

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';
            TIM_UI_disable(TIM3);
            TIM_UI_disable(TIM2);
      }
         else if(BT_Data=='n'||BT_Data=='N'){
            mode='N';
            TIM_UI_enable(TIM3);
            TIM_UI_enable(TIM2);
         }

      else if(BT_Data=='S'||BT_Data=='s'){
         cnt_u++;
         //if(cnt_u > 50) cnt_u = 50; // Limit the maximum value to prevent
exceeding the servo range
         PWM_duty(PWM_PIN2, (float)0.025*(cnt_u*1/50.0 + 1));
      }
      else if(BT_Data=='W'||BT_Data=='w'){
         cnt_u--;
         //if(cnt_u < 0) cnt_u = 0; // Limit the minimum value to prevent
exceeding the servo range
         PWM_duty(PWM_PIN2, (float)0.025*(cnt_u*1/50.0 + 1));
      }
      else if(BT_Data=='D'||BT_Data=='d'){
             cnt_r++;
         //if(cnt_r > 50) cnt_r = 50; // Limit the maximum value to prevent
exceeding the servo range
         PWM_duty(PWM_PIN1, (float)0.025*(cnt_r*1/50.0 + 1));
      }
      else if(BT_Data=='A'||BT_Data=='a'){
         cnt_r--;
         //if(cnt_r < 0) cnt_r = 0; // Limit the minimum value to prevent
exceeding the servo range
         PWM_duty(PWM_PIN1, (float)0.025*(cnt_r*1/50.0 + 1));
      }
   }
}
```