텀프로젝트 결과보고서

10조 (001분반)

과목명 : 임베디드시스템설계및실험

담당 교수 : 정상화 교수님

담당 조교 : 최호진 조교님

조원 : 201924603 하규승(조장)

201727102 강준혁

202023139 박지원

제출 날짜 : 2023.12.21

목차

1. 실험목표

- 수업시간에 배운 여러 센서 및 보드의 기능을 이용하여 하드웨어를 개발한다.
- 블루투스 밑 통신 관련 기능을 이용하여 하드웨어를 개발한다.
- 사람이 들어갈 수 없는 좁은 공간을 스스로 탐색 또는 조작할 수 있는 소형 자동 주행 시스템을 개발한다.

2. 세부목표

3) 사용자는 스마트폰으로 조작 모드(자동 주행, 수동 주행)를 선택할 수 있다.

3-1) 수동 주행 모드일 시, 사용자는 블루투스를 통해 RC카와 연결된 스마트폰으로 전진과 후진, 조향과 정지 조작이 가능하다.

3-2) 자동 주행 모드일 시, RC카는 초음파 센서로부터 기기와 전방과 좌우측 장애물 사이의 거리를 수신 받아 현재 주행상태(전진, 후진, 좌/우회전, 정지)를 결정한다.

3-3) 추가기능

- 시스템의 현재 설정(자동 주행, 수동 주행) 을 LCD에 표시한다.
- 충돌 감지 모듈을 사용하여 만약 시스템이 충돌을 감지할 시, 부저 경고음과 함께 시스템을 5초간 정지한다. 그 후 조작 모드가 자동 주행일 시 수동 주행 상태로 변경한다.

3. 실험 기구 및 사용센서

- STM32F107 보드
- 만능 기판
- TFT LCD
- 납땜 기구
- 센서(목록은 아래쪽에 작성)

<table>
<tr><td colspan="2">3-1) 모션 인식부분</td></tr>
<tr><td>- [SMG-A] HC-SR04P 3.3V/5V 호환 초음파 거리센서 모듈 [SZH-USBC-004]

• Operating Voltage: 3V – 5.5V
• Detecting distance : 5V(2cm – 450cm), 3.3V (2cm-400cm)
• Detection angle : <15

수량 : 3개</td><td> 
</td></tr>
<tr><td>- SW-18010P 진동센서모듈 [SZH-EK023]

• Working voltage : 3.3V – 5V
• Output forms : Digital switching output
• Wiring instructions : VCC, GND, DO, AO

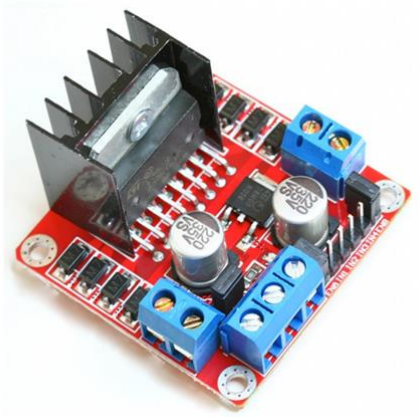
수량 : 1개</td><td>  </td></tr>
</table>

3-2) 모터

- [SMG] 2A L298 모터드라이버 모듈 (아두이노 호환) [SZH-EK001]

  - Drive voltage : 5V – 35V
  - Logical current : 0mA – 36mA
  - Max power : 25W
  - ENA enable IN1 IN2 control OUT1 OUT2
  - ENA enable IN3 IN4 control OUT3 OUT4

수량 : 2개



- 모터 : 기어박스장착모터 (NP01D-288)

  1- Operating voltage : 3V – 12V
  2- Max current : 0.17A
  3- Max speed : 19r/min
  4- Max output : 0.18W

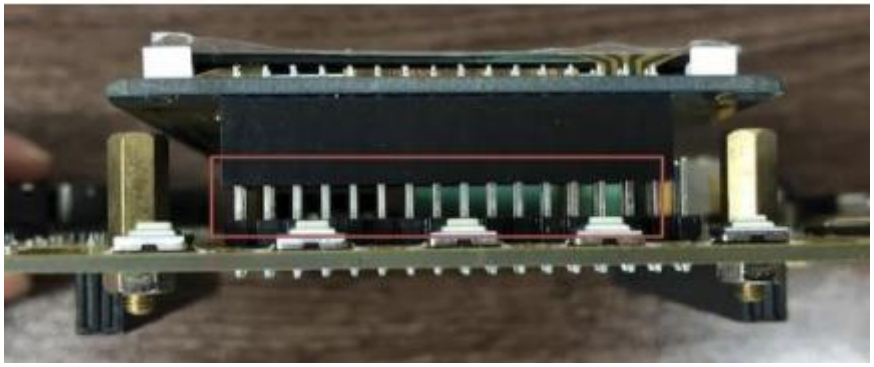
수량 : 4개

- 모터 : [SMG] 바퀴 66파이

수량 : 4개

4. 실험과정

4-1. 프로젝트 세팅



[그림 1] TFT LCD

4-1-1. LCD 기판을 보드의 핀과 정확히 맞도록 연결한다.

4-1-2. 만능 기판에 핀 맵에 따라 납땜을 한다. 핀 맵은 다음과 같다.

모터드라이버 -> d1, d2, d4, d7 / d8, d9, d10, d12

lcd -> b2, b9 / c5~c6, c8, c10~c12 / d11, d13~d15 / e0 ~ e15 / nc reset

초음파 -> c1, c3 / c2, c4 / c13, c14

진동센서 -> c0

블루투스 모듈 관련 -> a2, a3

버저 -> b0

4-1-3. 10주차 파일을 참고하여 코딩에 필요한 소스 및 헤더 파일을 사용해 프로젝트 디렉토리를 구성한다.

4-2. main.c 구현

```
#include "stm32f10x_exti.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_usart.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_tim.h"
#include "stm32f10x_adc.h"
#include "stm32f10x.h"
#include "core_cm3.h"
#include "misc.h"
#include "lcd.h"
#include "touch.h"

#define PC_ODR          *(volatile uint32_t *)0x4001100c
#define PC_IDR          *(volatile uint32_t *)0x40011008

TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
TIM_OCInitTypeDef TIM_OCInitStructure;

uint16_t input = '₩0';
int drive_mode = 0; // 0 for Auto,  1 for Manual
int now_go = 1; // 1 for front,  2 for backward,  3 for stop
int stuck = 0;
```

4-2-1. Project Setting

프로젝트 내에서 사용할 헤더 파일 및 전역변수들을 선언한다.

```
void RCC_Configure(void) {
  // TODO: Enable the APB2 peripheral clock using the function 'RCC_APB2PeriphClockCmd'
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2 , ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
}
```

4-2-2. RCC_Configure method

각종 센서 및 모터를 연결할 포트와 TIM 레지스터, 그리고 USART2 포트와 AFIO 의 RCC 를 활성화한다.

```
void GPIO_Configure(void) {
  GPIO_InitTypeDef GPIO_InitStructure;

  /* Motordriver 1 pin(d2, d3, d4, d7) setting */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_1 | GPIO_Pin_4 | GPIO_Pin_7;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOD, &GPIO_InitStructure);

  /* Motordriver 2 pin(d8, d9, d10, d12) setting */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10 | GPIO_Pin_12;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOD, &GPIO_InitStructure);

  /* Motordriver pwm pins setting*/
  GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_6;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOA, &GPIO_InitStructure);
  GPIO_Init(GPIOB, &GPIO_InitStructure);

  /* ultrasonic trigger pin(c1, c2, c13) setting */
  GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_13;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOC, &GPIO_InitStructure);

  /* ultrasonic echo pin(c3, c4, c14) setting */
  GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_14;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOC, &GPIO_InitStructure);

  /* hit sensor pin(c0) setting */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
  GPIO_InitStructure.GPIO_Speed = 0;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
  GPIO_Init(GPIOC, &GPIO_InitStructure);

/* usart2 pin() setting */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
GPIO_Init(GPIOA, &GPIO_InitStructure);
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
GPIO_Init(GPIOA, &GPIO_InitStructure);

/* buzzer pin(b0) setting*/
GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_0;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AF_PP;
GPIO_Init(GPIOB, &GPIO_InitStructure);
```

4-2-3. GPIO_Configure method

각 센서, 포트와 연결된 핀의 GPIO 를 활성화한다. 핀맵은 다음과 같다.

모터드라이버 -> d1, d2, d4, d7 / d8, d9, d10, d12

lcd -> b2, b9 / c5~c6, c8, c10~c12 / d11, d13~d15 / e0 ~ e15 / nc reset

초음파 -> c1, c3 / c2, c4 / c13, c14

진동센서 -> c0

블루투스 모듈 관련 -> a2, a3

버저 -> b0

```
void TIM_Configure(void) { // ??          ?  ?    ?     TIM2  ?
  uint16_t prescale = (uint16_t) 7200 - 1;

  TIM_TimeBaseStructure.TIM_Period = 10000-1;
  TIM_TimeBaseStructure.TIM_Prescaler = prescale;
  TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
  TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);

  TIM_ARRPreloadConfig(TIM2, ENABLE);
  TIM_Cmd(TIM2, ENABLE);
  // TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);

  prescale = (uint16_t) 1000-1;

  TIM_TimeBaseStructure.TIM_Period = 36-1;
  TIM_TimeBaseStructure.TIM_Prescaler = prescale-1;
  TIM_TimeBaseStructure.TIM_ClockDivision = 0;
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Down;
  TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
  TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
  TIM_OCInitStructure.TIM_Pulse = 0; // us
  TIM_OC3Init(TIM3, &TIM_OCInitStructure);

  TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);
  TIM_OC3PreloadConfig(TIM3, TIM_OCPreload_Disable);
  TIM_ARRPreloadConfig(TIM3, ENABLE);
  TIM_Cmd(TIM3, ENABLE);
}
```

4-2-3. TIM_Configure method

마이크로 초 딜레이를 구현하기 위해 TIM2 를, buzzer 의 pwm 을 구현하기 위해 TIM3 를 사용한다.

```
void NVIC_Configure(void) {
   NVIC_InitTypeDef NVIC_InitStructure;
   NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);

   // Sensor Active
   NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn;
   NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
   NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
   NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
   NVIC_Init(&NVIC_InitStructure);

   // USART2
   NVIC_EnableIRQ(USART2_IRQn);
   NVIC_InitStructure.NVIC_IRQChannel = USART2_IRQn;
   NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
   NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
   NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
   NVIC_Init(&NVIC_InitStructure);

}
```

4-2-4. NVIC_Configure method

충돌센서 작동 시 동작할 인터럽트와 USART2 통신을 위한 인터럽트를 설정한다. 이때, 인터럽트의 우선순위는 충돌센서가 1 순위, USART 통신이 2 순위 이다.

```
void USART2_Init(void) {
  USART_InitTypeDef USART_InitStructure;
  // Enable the USART2 peripheral
  USART_Cmd(USART2, ENABLE);

  USART_InitStructure.USART_BaudRate = 9600;
  USART_InitStructure.USART_WordLength = USART_WordLength_8b;
  USART_InitStructure.USART_StopBits = USART_StopBits_1;
  USART_InitStructure.USART_Parity = USART_Parity_No;
  USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
  USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
  USART_Init(USART2, &USART_InitStructure);

  // Enable the USART2 RX interrupts
  USART_ITConfig(USART2, USART_IT_RXNE, ENABLE);
}
```

4-2-5. USART_Init method

블루투스 통신을 위한 USART2 사용을 하기 위해 USART 를 초기화한다.

```
void bluetooth(void) {
   // drive mode
   if(input == 'a') { // auto
     drive_mode = 0;
     now_go = 3;
   }
   if(input == 'm') { // manual
     drive_mode = 1;
     now_go = 3;
   }
   // direction
   if(input== 'f') { // front
     now_go = 1;
   }
   if(input == 'b') { // back
     now_go = 2;
   }
   if(input == 's') { //stop
     now_go = 3;
   }
//    input = '₩0';
}

void USART2_IRQHandler(void) {
   if (USART_GetITStatus(USART2, USART_IT_RXNE) != RESET) {
       input = USART_ReceiveData(USART2);
       bluetooth();
       USART_ClearITPendingBit(USART2,USART_IT_RXNE);
   }
}
```

4-2-6. USART Communication method

USART2 를 이용해 블루투스 통신을 구현한다. 이때, 통신을 위한 버튼은 매크로로 설정된 단어를 보내며, 이를 인터럽트가 들어왔을 때 처리한다.

```
/* Motor */
void motorfl(int n) {
  // Uses d2, d3
  // n == 0 front move high/low
  if(n == 0) {
    GPIO_SetBits(GPIOD, GPIO_Pin_2);
    GPIO_ResetBits(GPIOD, GPIO_Pin_1);
  }
  // n == 1 back move low/high
  if(n == 1) {
    GPIO_ResetBits(GPIOD, GPIO_Pin_2);
    GPIO_SetBits(GPIOD, GPIO_Pin_1);
  }
  // n == 2 stop low/low
  if(n == 2) {
    GPIO_ResetBits(GPIOD, GPIO_Pin_2);
    GPIO_ResetBits(GPIOD, GPIO_Pin_1);
  }
}
void motorfr(int n) {
  // Uses d4, d7
  // n == 0 front move high/low
  if(n == 0) {
    GPIO_SetBits(GPIOD, GPIO_Pin_4);
    GPIO_ResetBits(GPIOD, GPIO_Pin_7);
  }
  // n == 1 back move low/high
  if(n == 1) {
    GPIO_ResetBits(GPIOD, GPIO_Pin_4);
    GPIO_SetBits(GPIOD, GPIO_Pin_7);
  }
  // n == 2 stop low/low
  if(n == 2) {
    GPIO_ResetBits(GPIOD, GPIO_Pin_4);
    GPIO_ResetBits(GPIOD, GPIO_Pin_7);
  }
}
```

```
void motorbl(int n) {
  // Uses d8, d9
  // n == 0 front move high/low
  if(n == 0) {
    GPIO_SetBits(GPIOD, GPIO_Pin_8);
    GPIO_ResetBits(GPIOD, GPIO_Pin_9);
  }
  // n == 1 back move low/high
  if(n == 1) {
    GPIO_ResetBits(GPIOD, GPIO_Pin_8);
    GPIO_SetBits(GPIOD, GPIO_Pin_9);
  }
  // n == 2 stop low/low
  if(n == 2) {
    GPIO_ResetBits(GPIOD, GPIO_Pin_8);
    GPIO_ResetBits(GPIOD, GPIO_Pin_9);
  }
}
void motorbr(int n) {
  // Uses d10, d12
  // n == 0 front move high/low
  if(n == 0) {
    GPIO_ResetBits(GPIOD, GPIO_Pin_10);
    GPIO_SetBits(GPIOD, GPIO_Pin_12);
  }
  // n == 1 back move low/high
  if(n == 1) {
    GPIO_SetBits(GPIOD, GPIO_Pin_10);
    GPIO_ResetBits(GPIOD, GPIO_Pin_12);
  }
  // n == 2 stop low/low
  if(n == 2) {
    GPIO_ResetBits(GPIOD, GPIO_Pin_10);
    GPIO_ResetBits(GPIOD, GPIO_Pin_12);
  }
}
```

```
void setFront() {
  motorfl(0);
  motorfr(0);
  motorbl(0);
  motorbr(0);
}
void setBack() {
  motorfl(1);
  motorfr(1);
  motorbl(1);
  motorbr(1);
}
void setRight() {
  motorfl(0);
  motorfr(2);
  motorbl(0);
  motorbr(2);
}
void setLeft() {
  motorfl(2);
  motorfr(0);
  motorbl(2);
  motorbr(0);
}
void setStop() {
  motorfl(2);
  motorfr(2);
  motorbl(2);
  motorbr(2);
}
```

4-2-6. Motor Activation method

모터의 움직임을 구현한다. fl, fr, bl, br 은 각각 왼쪽 앞바퀴, 오른쪽 앞바퀴, 왼쪽 뒷바퀴, 오른쪽 뒷바퀴의 동작을 구현한다.

```c
/* Ultrasonic */
void delay_us(uint32_t us) {
  if(us > 1) {
    uint32_t count = us * 8 - 6;
    while(count--);
  }
  else {
    uint32_t count = 2;
    while(count--);
  }
}

void trig_pulse(uint16_t trigPin)  {
  PC_ODR |= (1 << trigPin); // GPIO_SetBits
  delay_us(11);
  PC_ODR &= ~(1 << trigPin); // GPIO_ResetBits
  delay_us(11);
}

uint32_t getDistance(uint16_t trigPin, uint16_t echoPin) {
  uint32_t echo;
  trig_pulse(trigPin); // give trig pulse to u_sonic sensor

  while((PC_IDR & (1 << echoPin)) != (1 << echoPin)); // wait echo pin status turns HIGH
  echo = TIM2->CNT;

  while((PC_IDR & (1 << echoPin)) == (1 << echoPin));
  echo = TIM2->CNT - echo;

  return echo;
}
```

4-2-7. Ultrasonic sensor method

초음파 센서 동작을 구현한다. trig_pulse()에서는 delay_us()를 이용해 trigger pin 에 10us 의 pulse 를 입력하고, getDistance()에서는 echo 핀에서 출력된 신호를 통해 거리를 계산하여 반환한다.

```c
/* Buzzer */
void alert_buzzer(void) {
  for(int i = 0; i < 5; i++) {
    TIM3->PSC = 1000;
    TIM3->CCR1 = TIM3->ARR / 2;
    delay_us(700000);
    TIM3->PSC = 0;
    TIM3->CCR1 = 0;
    delay_us(300000);
  }
}
```

4-2-8. Buzzer Activation method

PWM 을 통해 버저 출력을 구현한다. alert_buzzer() 함수가 호출되면 버저는 5 초간 5 번의 비프음을 발생한다.

```
/* Crash sensor */
void EXTI_Configure(void) {
   EXTI_InitTypeDef EXTI_InitStructure;

   GPIO_EXTILineConfig(GPIO_PortSourceGPIOC, GPIO_PinSource0);
   EXTI_InitStructure.EXTI_Line = EXTI_Line0;
   EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
   EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
   EXTI_InitStructure.EXTI_LineCmd = ENABLE;
   EXTI_Init(&EXTI_InitStructure);
}

void EXTI0_IRQHandler(void) {
  if (EXTI_GetITStatus(EXTI_Line0)!= RESET) {
    if (GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_0) == Bit_RESET) {
      now_go = 3;
      drive_mode = 1;
      stuck = 1;
    }
    EXTI_ClearITPendingBit(EXTI_Line0);
  }
}
```

4-2-9. Crash sensor method

충돌센서의 동작을 구현한다. EXTI 를 통해 충돌이 감지 될 시, drive_mode 를 수동 주행으로, 진행 방향을 정지로 설정하고, 전역변수인 stuck 의 값을 1 로 변경해 충돌이 감지되었음을 알려준다.

```c
/* main function */
int main(void) {
  SystemInit();
  RCC_Configure();
  GPIO_Configure();
  NVIC_Configure();
  EXTI_Configure();
  TIM_Configure();
  USART2_Init();
  LCD_Init();
  Touch_Configuration();
  LCD_Clear(WHITE);
  TIM3->PSC = 0;
  TIM3->CCR1 = 0;

  while (1) {
    // TODO: implement

    // Front move
    uint32_t fr_sensor = getDistance(13, 14); // trig / echo
    uint32_t rt_sensor = getDistance(1, 3);
    uint32_t lt_sensor = getDistance(2, 4);

    LCD_ShowNum(40, 40, fr_sensor, 10 , RED, WHITE);
    LCD_ShowNum(40, 60, rt_sensor, 10 , RED, WHITE);
    LCD_ShowNum(40, 80, lt_sensor, 10 , RED, WHITE);
    LCD_ShowNum(40, 100, drive_mode, 10 , RED, WHITE);
    LCD_ShowNum(40, 120, now_go, 10 , RED, WHITE);

    if(stuck == 1) {
      setStop();
      alert_buzzer();
      stuck = 0;
    }
    if(drive_mode == 0) {
      if(fr_sensor < 5) {
        setBack();
        delay_us(1000000);
      }
      else if(fr_sensor < 15) {
        if(rt_sensor < lt_sensor) {
          setLeft();
        }
        else if(lt_sensor < rt_sensor) {
          setRight();
        }
      }
      else if(lt_sensor < 5) {
        setRight();
      }
      else if(rt_sensor < 5) {
        setLeft();
      }
      else {
        setFront();
      }
    }
    else {
      if(now_go == 1) { setFront(); }
      else if(now_go == 2) { setBack(); }
      else if(now_go == 3) { setStop(); }
    }
  }

  return 0;
}
```

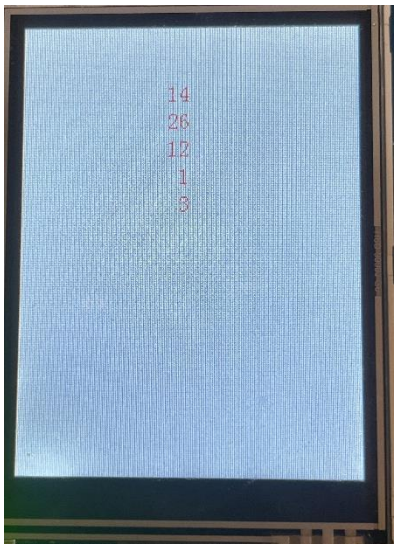
4-2-10. Main method

위에서 구현한 함수를 통해 전체적인 움직임을 설정한다. LCD 에는 지속적으로 전방/우측/좌측 초음파 센서의 반환값을 표시하고, 현재 주행모드와 진행 방향 또한 미리 설정된 값으로 표시한다. 부딪혀서 stuck 값이 1 이 되면 정지하고 alert_buzzer()를 호출하여 버저를 작동하고, 값을 초기화한다. 이외의 경우엔 drive_mode 의 값에 따라 0 이면 자동 주행, 1 이면 수동 주행을 한다. 자동 주행일 시, 센서의 threshold 값에 따라서 전방에 장애물이 있으면 후진, 전방 벽과 거리가 얼마 안 남은 상태에서 좌측 센서의 threshold 값이 더 작으면 우회전, 그렇지 않으면 좌회전을 하고, 만약 왼쪽에 장애물이 너무 가까이 있다면 우회전을, 오른쪽에 장애물이 너무 가까이 있다면 좌회전을 한다. 수동 주행 모드일시, Bluetooth 통신을 통해 폰에서 방향을 지정함에 따라 전진, 후진, 정지 중 1 가지 동작을 한다.

5. 실험 결과

[그림 2] Bluetooth Terminal 화면

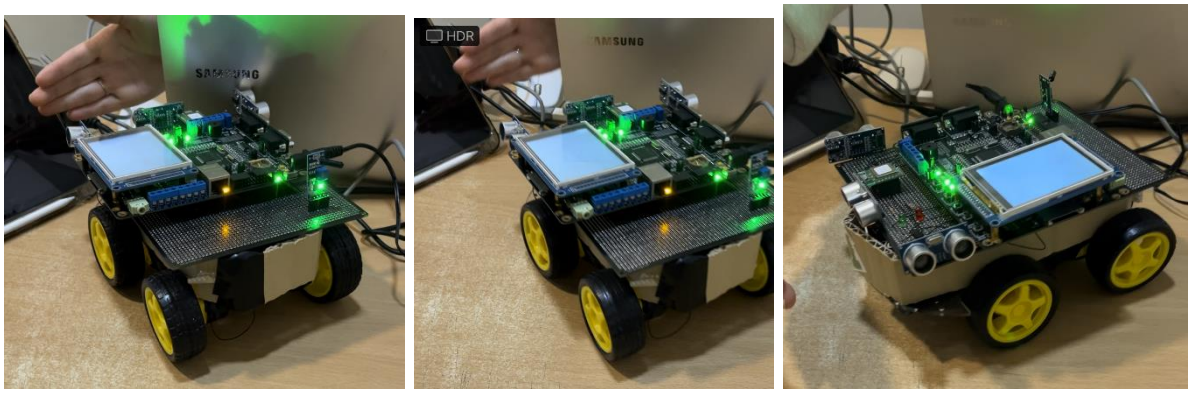
주행에 앞서 블루투스를 통해 스마트폰과 연결을 시도하여 연결에 성공한 것을 확인하였다. 또한, TFT LCD에 RC카의 앞, 좌, 우에 부착된 초음파 센서를 통해 주변의 장애물과의 거리를 측정하여 표시하는 것을 확인할 수 있었다.



[그림 3] TFT LCD 화면(초음파 센서 거리 값, 주행 모드 상태 표시)

<자동 주행 모드>

전원을 인가 한 후 자동 주행 모드로 주행하는 것을 확인할 수 있었다. 초음파 센서를 통해 전방에 장애물이 있는지 확인한 후 장애물이 없을 시에는 계속 직진을 하고 장애물이 존재할 경우 장애물과의 거리를 받아와 거리가 5cm 이하일 때 후진하는 것을 알 수 있었다. 후진을 일정 거리 진행한 후 장애물과의 거리가 15cm이하일 때 좌우의 초음파 센서를 통해 장애물이 있는지 판단하고 초음파 센서로 측정한 거리가 더 긴 방향으로 모터 조작을 통해 진행하는 것을 확인할 수 있었다.



[그림 4] RC 자동 주행 모드

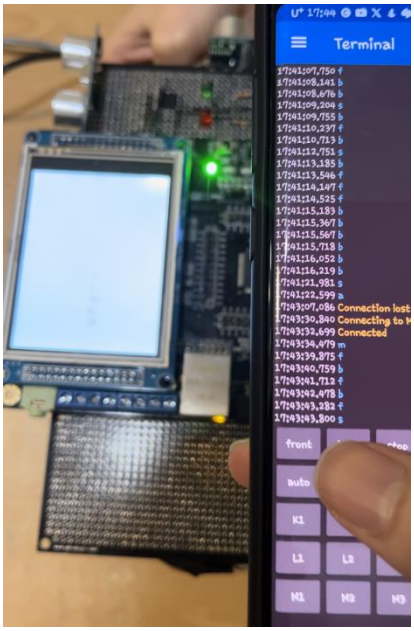
자동 주행 모드로 주행 시 진동 모듈 센서에 충격이 감지되면 부저가 5초동안 울리며 수동 주행 모드로 전환된 것을 확인할 수 있었다. 이때 초음파 센서 또한 정지한 것을 알 수 있었다.

[그림 5] Bluetooth Terminal keyboard 화면

자동 주행 모드로 주행 도중 블루투스를 통해 연결된 스마트폰에서 수동 주행 모드로 변경하라는 신호인 'm'(manual)을 보내면 수동 주행 모드로 변경되는 것을 확인할 수 있었다.

<수동 주행 모드>

수행 주행 모드 시 블루투스를 통해 연결된 스마트폰에서 'f'(front) 신호를 보내면 앞으로 직진하는 것을 알 수 있었고 'b'(back) 신호를 보내면 후진하는 것을 알 수 있었다. 또한 's'(stop) 신호를 보내면 모터를 정지시켜 멈춘 것을 확인할 수 있었다.



[그림 6] 수동 주행 모드

수행 주행 모드 시 블루투스를 통해 연결된 스마트폰에서 다시 자동 주행 모드로 변경하라는 신호인 'a'(auto)를 보내면 자동 주행 모드로 변경되는 것을 확인할 수 있었다.

6. 결과에 대한 논의 및 결론

위에서 확인한 실험 결과를 바탕으로 사용자가 스마트폰으로 조작 모드(자동 주행, 수동 주행)를 선택할 수 있는 기능이 잘 작동하는 것을 알 수 있었다. 먼저 자동 주행 모드시, RC 카가 초음파 센서로부터 기기와 전방과 좌우측 장애물 사이의 거리를 수신 받아 현재 직진할 것인지 후진할 것인지에 대한 주행상태를 결정하는 동작이 정상작동하는 것을 알 수 있었다. 또한, 수행 주행 모드시, 사용자가 블르투스로 연결된 스마트폰을 통해 RC 카의 전진, 후진, 정지 조향이 가능한 것을 알 수 있었으며 결과적으로 초음파 센서, 모터에 관한 동작이 목표에 따라 잘 작동하는 것을 알 수 있었다. 추가적으로 초음파 센서로 읽어온 거리 값과 시스템의 현재 설정 (자동 주행, 수동 주행)을 LCD 에 표시하여 사용자가 현재 상태를 확인할 수 있도록 하며, 진동 모듈 센서를 통해 시스템이 충돌을 감지할 시 부저 경고음을 5 초간 울리며 시스템이 정지하고 자동 주행 모드에서 수동 주행 모드로 바뀌는 동작 또한 잘 작동하는 것을 확인할 수 있었다.

원래 세부 목표를 정상적으로 가동하는 것을 확인하여 시스템을 정상적으로 설계 및 구현한 것을 알 수 있었고 이에 따라 실험 목표를 수행할 수 있었다.