2024 임베디드 시스템 설계 및 실험
수요일 분반 3조 Term Project 결과보고서

2024년 12월 22일 조수현, 김민경, 김범모, 최두환

요약
임베디드 시스템을 이용한 엘리베이터 개폐 감지 시스템 모형 구현 결과보고서로 프로젝트의 목표와 시나리오, 구현 내용, 동작 설명, 결론 등을 포함한다.

<목차>

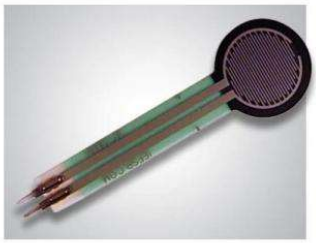
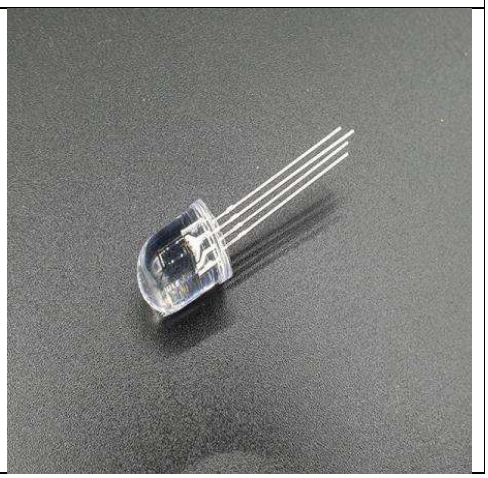
1. 프로젝트 소개

- 수업시간에 배운 여러 센서 및 보드의 기능을 이용하여 하드웨어를 개발한다.

- bluetooth 및 통신 관련 기능을 이용하여 하드웨어를 개발한다.

- 압력 센서를 통해 엘리베이터의 혼잡도를 확인하고, 출입문을 제어하여 이용자들의 안전을 확보한다.

2. 주요 센서

<table>
<tr><td colspan="2">2-1) 모션 인식부분</td></tr>
<tr><td>압력센서 FSR 402 Solder Tabs [30-81794] 1개
(https://www.devicemart.co.kr/goods/view?no=33870)
Actuation Force
≈0.2N min
Force Sensing Range
≈0.2N - 20N
Force Repeatability Single Part
±2%
Force Repeatability Part to Part
±6% (single batch)
No-Load Resistance
〉10 MΩ
Hysteresis
+5% (RF+ - RF-)/RF+
Long Term Drift at 1kg for 35 days
〈 5% log10(time)</td><td>  </td></tr>
<tr><td>[SMG] RGB LED 10파이(CA) 투명 1개
(https://www.devicemart.co.kr/goods/view?no=12501933)
- 사이즈 : 10 파이
- 전압 : 2V~3.2V
- 전류 : 20mA
- 색상 : RGB
- 공통단자 : 애노드 공통</td><td>  </td></tr>
</table>

4-2) 출입문 자동 제어 부분

MG90S 메탈기어 디지털 서보모터 1개
(https://www.devicemart.co.kr/goods/view?no=12503480)

- Weight: 13.4g
  Dimension: 22.8×12.2×28.5mm
  Stall torque: 1.8kg/cm (4.8V); 2.2kg/cm (6.6V)
  Operating speed: 0.10sec/60degree (4.8V); 0.08sec/60degree (6.0V)
  Operating voltage: 4.8V
  Temperature range: 0℃_ 55℃
  Dead band width: 1us
  Power Supply: Through External Adapter
  servo wire length: 25 cm
  Servo Plug: JR (Fits JR and Futaba)

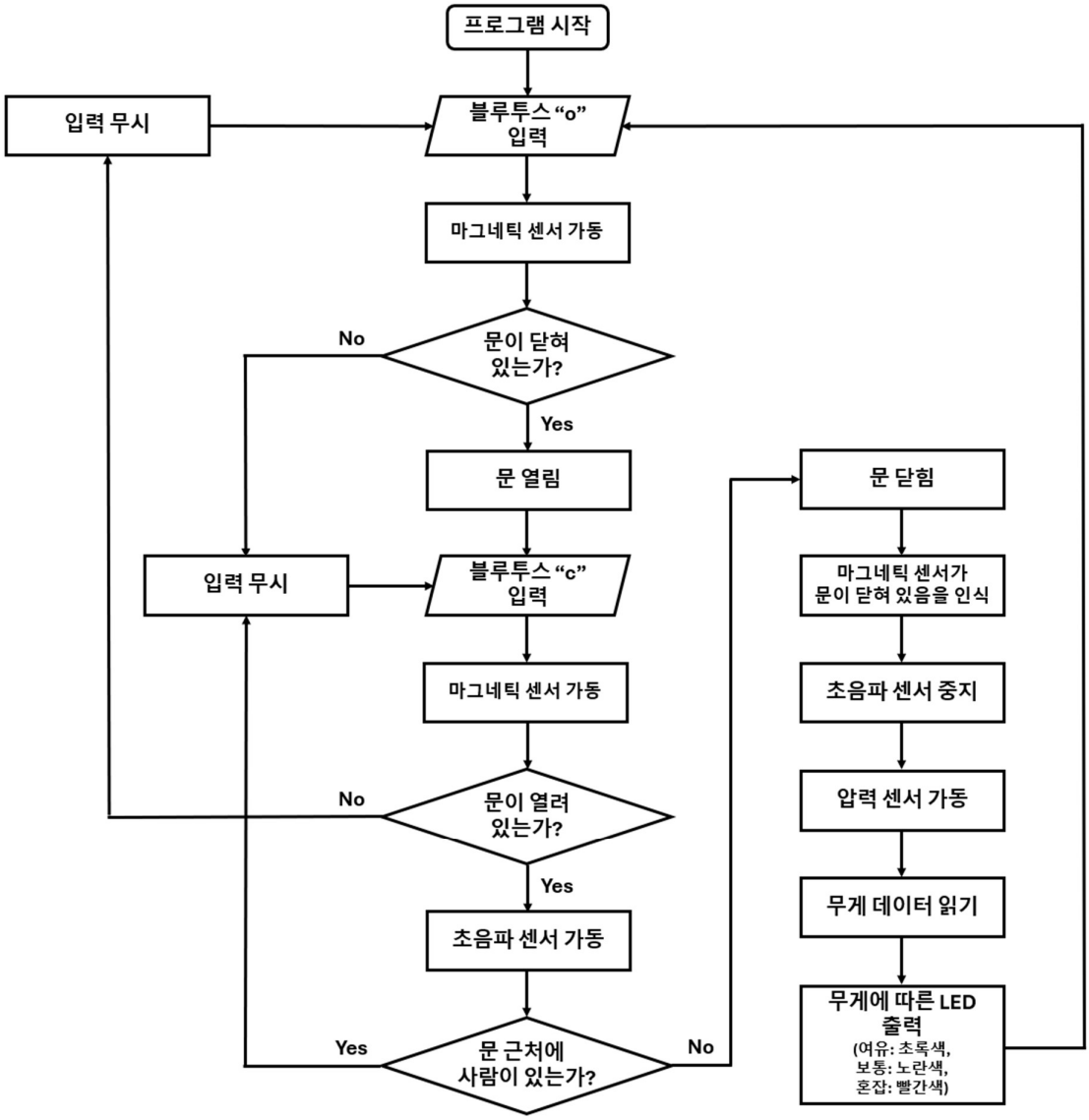
[SMG-A] HC-SR04P 3.3V/5V 호환 초음파 거리센서 모듈 [SZH-USBC-004] 1개
(https://www.devicemart.co.kr/goods/view?no=1323062)

- Wide voltage operation: 3V-5.5V
- With HC-SR04 software and hardware size is fully compatible
- Detecting distance:
- 5V: 2cm - 450cm
- 3.3V: 2cm - 400cm
- Detection angle: <15
- Using industrial grade MCU, working temperature: -20 - 80 degrees Celsius

[e-con] 마그네틱 문열림 감지용 센서 (NO) 5C-39 1개, 3900원
(https://www.devicemart.co.kr/goods/view?no=1341891)

- **Gap: 15~20mm**

3. 흐름도



(제안서에서 변경된 로직에 따라 플로우 차트도 변경)

4. 제약 사항 준수 및 구현

4.1. 인터럽트 활용

다음 코드는 프로젝트에서 인터럽트를 활용한 예시로, USART 와 ADC 변환 완료 시 인터럽트 핸들러를 구현하여 실시간 데이터를 처리하는 방법을 보여준다.

```
// USART1 인터럽트 핸들러
void USART1_IRQHandler() {
```

```
    if(USART_GetITStatus(USART1,USART_IT_RXNE)!=RESET){
        flagUART1 = 1;
        // the most recent received data by the USART1 peripheral
        wordFromUART1 = USART_ReceiveData(USART1);
        // clear 'Read data register not empty' flag
        USART_ClearITPendingBit(USART1,USART_IT_RXNE);
    }
}

// USART2 인터럽트 핸들러
void USART2_IRQHandler() {

    if(USART_GetITStatus(USART2,USART_IT_RXNE)!=RESET){
        flagUART2 = 1;
        // the most recent received data by the USART1 peripheral
        wordFromUART2 = USART_ReceiveData(USART2);
        // clear 'Read data register not empty' flag
        USART_ClearITPendingBit(USART2,USART_IT_RXNE);
    }
}

// ADC 변환 완료 인터럽트 핸들러
void ADC1_2_IRQHandler(void) {
    if (ADC_GetITStatus(ADC1, ADC_IT_EOC) != RESET) {
        // ADC 변환 값 읽기
        pressureValue = ADC_GetConversionValue(ADC1);

        // 인터럽트 플래그 클리어
        ADC_ClearITPendingBit(ADC1, ADC_IT_EOC);
    }
}
```

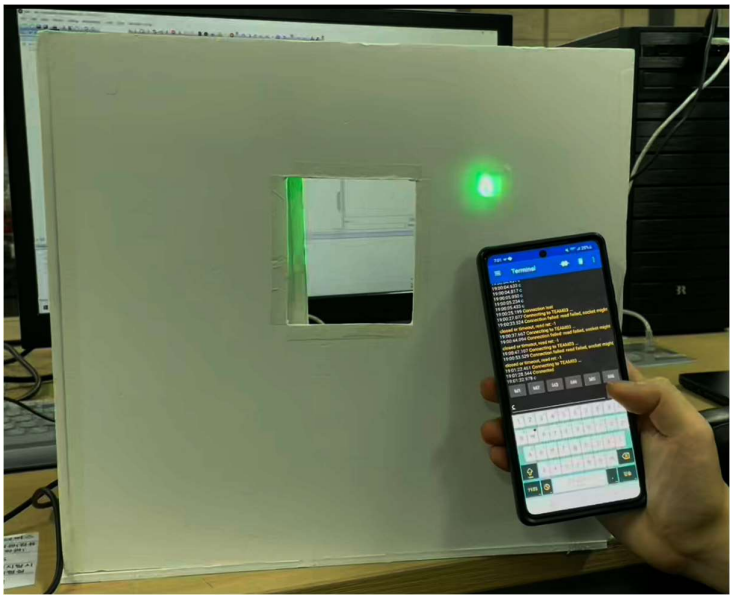
4.2. 센서 간의 의존성 (센서 2 개 이상, 각자 폴링 방식으로 동작하지 말고 한 센서 값이 다른 센서 이용을 호출하는 시나리오)

다음 코드는 마그네틱 센서와 초음파 센서 간의 의존성을 보여준다. isOpen() 함수에서 마그네틱 센서 값을 감지하여 문이 열렸을 때만 detectPerson() 함수에서 초음파 센서를 호출하여 사람을 감지하는 방식으로 동작한다.

```
// 마그네틱 센서로 문 닫힘 감지
int isOpen(void) {
    if(~GPIOD->IDR & GPIO_Pin_12) {
        return 0; // 문이 닫혔다고 표시
    }
    else {
        return 1; // 문이 열려 있다고 표시
    }
}

// 초음파 센서로 사람 감지
```

```
int detectPerson(void) {
   if(isOpen() == 1) { // 문이 열려 있을 때만 감지 (즉, 마그네틱 값이 초음파 이용을 호출)
        int distanceThr = 150;   // 감지 거리 임계값
        dist = HCSR04GetDistance(); // 초음파 센서를 통한 거리 측정
        if(distanceThr > dist) {
            return 1;   // 사람 감지됨
        }
        else {
            return 0;   // 사람 없음
        }
    }
}
```

4.3. 블루투스 연동



휴대폰과 블루투스가 연결되어 있는 모습이다.

4.4. DC 모터를 사용하는 시나리오일 경우 릴레이 모듈이 아닌 모터 드라이버 활용

본 프로젝트에서는 DC 모터 대신 서보 모터를 사용하였기 때문에, 위 제약 사항에는 해당되지 않는다.

5. 시나리오에 따른 구현 과정

5.1. 시스템을 켠 후 핸드폰과 블루투스로 연결

이 코드는 시스템을 켠 후 핸드폰과 블루투스를 연결하기 위해 USART2를 초기화하고, 데이터를 송수신하는 기능을 구현한다. USART2_Init() 함수는 USART2 설정을 진행하며, USART2_IRQHandler()에서 블루투스 장치로부터 수신된 데이터를 처리한다. 또한 sendDataUART2()를 통해 데이터를 송신한다.

```
// USART2 초기화 함수
void USART2_Init(void)
{
    USART_InitTypeDef USART2_InitStructure;

    // Enable the USART2 peripheral
    USART_Cmd(USART2, ENABLE);

    // TODO: Initialize the USART using the structure 'USART_InitTypeDef' and the function 'USART_Init'
    // Initialize the USART using the structure 'USART_InitTypeDef' and the function 'USART_Init'
    USART2_InitStructure.USART_BaudRate = 9600;
    USART2_InitStructure.USART_WordLength = (uint16_t)USART_WordLength_8b;
    USART2_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    USART2_InitStructure.USART_Parity = (uint16_t)USART_Parity_No;
    USART2_InitStructure.USART_StopBits = (uint16_t)USART_StopBits_1;
    USART2_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;

    USART_Init(USART2, &USART2_InitStructure);
    // TODO: Enable the USART2 RX interrupts using the function 'USART_ITConfig' and the argument value
'Receive Data register not empty interrupt'
    USART_ITConfig(USART2, USART_IT_RXNE, ENABLE);
}

// USART2 인터럽트 핸들러
void USART2_IRQHandler() {

    if(USART_GetITStatus(USART2,USART_IT_RXNE)!=RESET){
        flagUART2 = 1;
        // the most recent received data by the USART1 peripheral
        wordFromUART2 = USART_ReceiveData(USART2);
        // clear 'Read data register not empty' flag
        USART_ClearITPendingBit(USART2,USART_IT_RXNE);
    }
}

void sendDataUART2(uint16_t data) {
    USART_SendData(USART2, data);
}
```

5.2. 출입문의 현재 개폐 여부에 따른 출입문의 다음 상태 결정

아래 코드는 휴대폰의 입력에 따라 문 열림/닫힘을 조절하는 기능을 구현한다. main() 함

수에서는 UART로 받은 데이터에 따라 문을 여는 동작(open())과 닫는 동작(close())을 수행한다.

```
int main(void) {
    // 중략

    while (1) {
        if (flagUART1 == 1){
            sendDataUART2(wordFromUART1);
            flagUART1 = 0;
        }
        else if (flagUART2 == 1){
            sendDataUART1(wordFromUART2);
            flagUART2 = 0;

            // 'o'를 받으면 문 열기
            if(wordFromUART2 == 'o') {
                open();
            }
            // 'c'를 받으면 문 닫기
            else if(wordFromUART2 == 'c') {
                close();
            }
            else if(wordFromUART2 == 's') {
                moveMotor(1500);
            }

        }
        // 중략
}
```

그리고 모터를 움직일 때는 아래의 함수를 활용한다.

```
// 모터 회전 함수 (PWM 제어)
void moveMotor(uint16_t var)
{
    TIM_OCInitTypeDef TIM_OCInitStructure;
    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 = var;
    TIM_OC3Init(TIM4, &TIM_OCInitStructure);
}
```

5.2.1. 출입문을 열어야 하는 경우

isOpen() 함수는 마그네틱 센서를 이용해 문이 닫혔는지 확인하고, open() 함수는 문이 닫혀 있을 때만 서보 모터를 작동시켜 문을 연다.

```
// 마그네틱 센서로 문 닫힘 감지
```

```
int isOpen(void) {
    if(~GPIOD->IDR & GPIO_Pin_12) {
        return 0; // 문이 닫혔다고 표시
    }
    else {
        return 1; // 문이 열려 있다고 표시
    }
}


// 문 열기 동작
void open() {//문 여는 함수
    if (isOpen() == 0) { //문이 닫혀 있을 때만 모터 동작
        moveMotor(1000);   // 모터 회전
        for(int i=0; i < 10000000; i++) {}   // 딜레이는 실험적으로 지정
    }
    moveMotor(1500); //모터 정지
}
```

5.2.2. 출입문을 닫아야 하는 경우

5.2.1.의 isOpen() 함수를 마그네틱 센서를 통해 문이 열려 있는지 확인하고, detectPerson() 함수는 초음파 센서를 이용해 문 앞에 사람이 있는지를 감지한다. close() 함수는 문이 열려 있고 사람이 없을 때만 모터를 작동시켜 출입문을 닫는다.

```
// 초음파 센서로 사람 감지
int detectPerson(void) {
  if(isOpen() == 1) { // 문이 열려 있을 때만 감지 (즉, 마그네틱 값이 초음파 이용을 호출)
      int distanceThr = 150;   // 감지 거리 임계값
      dist = HCSR04GetDistance(); // 초음파 센서를 통한 거리 측정
      if(distanceThr > dist) {
          return 1;   // 사람 감지됨
      }
      else {
          return 0;   // 사람 없음
      }
    }
}

// 문 닫기 동작
void close() {
    if (!detectPerson()) { // 문이 열려 있고(detectPerson()에서 확인) 사람이 없을 때만 모터 동작
        moveMotor(2000);   // 모터 회전
        for(int i=0; i < 10000000; i++) {}   // 딜레이는 실험적으로 지정
    }
    //모터 off
    moveMotor(1500);
 }
```

5.3. 압력 센서가 탑승 인원들의 무게 데이터를 읽어 혼잡도에 따른 색상을 led에 출력

ADC1_2_IRQHandler() 함수는 압력 센서로부터 변환된 데이터를 읽어 pressureValue에 저장하며, main() 함수에서는 이 값을 기준으로 혼잡도 상태를 결정하여, 해당 상태에 맞는 LED 색상을 설정한다. setRGBLED() 함수는 입력받은 flag 값에 따라 RGB LED의 각 색상(빨강, 초록, 파랑)을 제어하여, 혼잡도에 맞는 색상을 출력하도록 설정하는 함수다.

```
// ADC 변환 완료 인터럽트 핸들러
void ADC1_2_IRQHandler(void) {
    if (ADC_GetITStatus(ADC1, ADC_IT_EOC) != RESET) {
        // ADC 변환 값 읽기
        pressureValue = ADC_GetConversionValue(ADC1);

        // 인터럽트 플래그 클리어
        ADC_ClearITPendingBit(ADC1, ADC_IT_EOC);
    }
}


// RGB LED 설정 함수
void setRGBLED(int flag) {
    switch (flag) {
    case 1: // 초록색
        GPIOD->ODR |= (GPIO_Pin_2);   // 빨간색 끔
        GPIOB->ODR &= ~(GPIO_Pin_2);    // 초록색 켬
        GPIOC->ODR |= (GPIO_Pin_5);   // 파란색 끔
        break;
    case 2: // 노란색 (빨간 + 초록)
        GPIOD->ODR &= ~(GPIO_Pin_2);   // 빨간색 켬
        GPIOB->ODR &= ~(GPIO_Pin_2);   // 초록색 켬
        GPIOC->ODR |= (GPIO_Pin_5);   // 파란색 끔
        break;
    case 3: // 빨간색
        GPIOD->ODR &= ~(GPIO_Pin_2);   // 빨간색 켬
        GPIOB->ODR |= (GPIO_Pin_2);   // 초록색 끔
        GPIOC->ODR |= (GPIO_Pin_5);   // 파란색 끔
        break;
    default: // 하얀색 (빨간, 초록, 파란색 모두 켬)
        GPIOD->ODR |= GPIO_Pin_2;
        GPIOB->ODR |= GPIO_Pin_2;
        GPIOC->ODR |= GPIO_Pin_5.
        break;
}
}

int main(void) {
```

```
    // 중략

        // 압력 센서값에 따른 상태 설정
        if(pressureValue < pressureThreshold1) {
            flag = 1;   // 상태 1 (초록색)
        }
        else if(pressureValue < pressureThreshold2) {
            flag = 2;   // 상태 2 (노란색)
        }
        else {
            flag = 3;   // 상태 3 (빨간색)
        }

        setRGBLED(flag); // 상태에 맞는 LED 색 설정

    }
     return 0;
}
```

5.4. 우선순위 설정

이 부분은 시나리오와 직접적으로 관련되지는 않지만, 시스템의 안정적인 동작을 위해 인터럽트 우선순위를 설정하는 중요한 부분이다.

NVIC_Configure() 함수는 시스템의 인터럽트 우선순위를 설정한다. 우선순위 그룹을 NVIC_PriorityGroup_2로 설정하고, 각 인터럽트에 대해 Preemption Priority와 SubPriority를 지정하여 우선순위를 설정한다.

```
// 인터럽트 우선순위 설정: USART1 > USART2 > TIM4 > ADC1
void NVIC_Configure(void) {
    NVIC_InitTypeDef NVIC_InitStructure;

    // NVIC Priority Group 설정
    // 우선순위 그룹을 2로 설정하여 Preemption Priority가 2비트, SubPriority가 2비트로 나누어짐
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);

    // ADC1 인터럽트
    NVIC_EnableIRQ(ADC1_2_IRQn);                    // ADC1_2 인터럽트 활성화
    NVIC_InitStructure.NVIC_IRQChannel = ADC1_2_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x03;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x00;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    // USART1 인터럽트
    NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x00;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x00;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
```

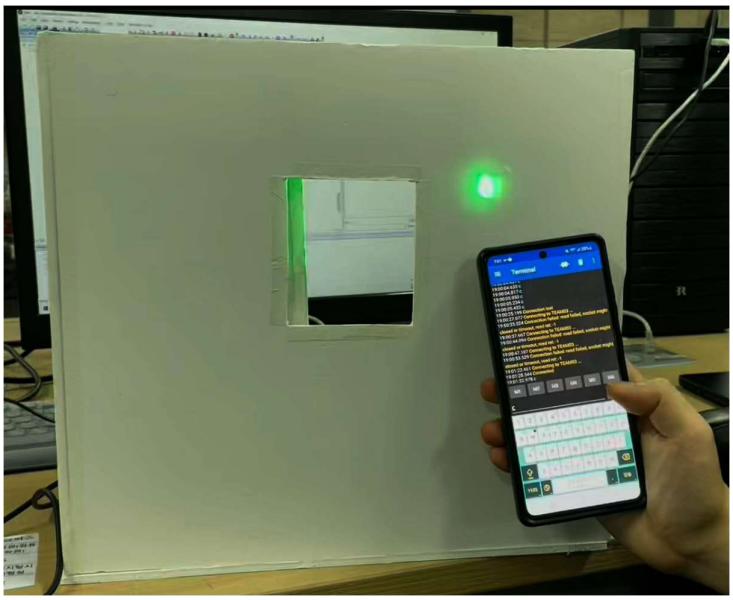
```
    NVIC_Init(&NVIC_InitStructure);

    // USART2 인터럽트
    NVIC_InitStructure.NVIC_IRQChannel = USART2_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x01;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x00;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    // TIM4 인터럽트
    NVIC_InitStructure.NVIC_IRQChannel = TIM4_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x02;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x00;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

}
```

6. 구현

아래의 링크에서 구현 영상들을 확인할 수 있다.

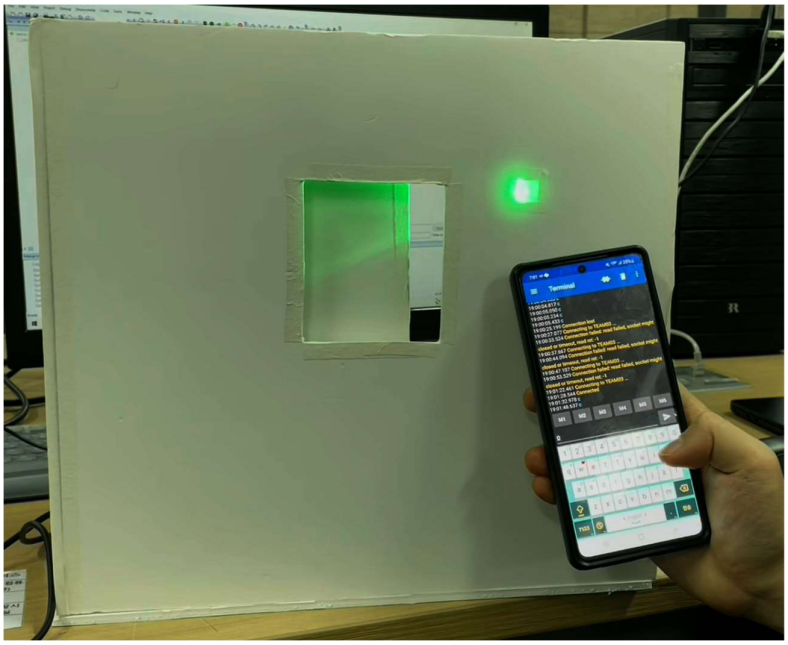
https://confirmed-sulfur-5eb.notion.site/2024-3-1650df77f77f80c9b7b2ff5912ac3b50?pvs=4

6.1 휴대폰과 블루투스로 연결



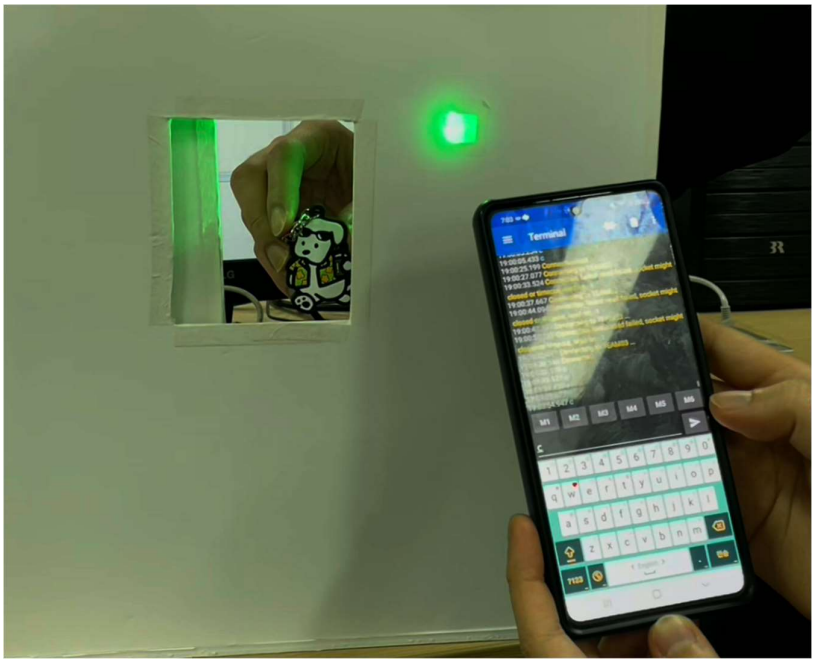
휴대폰이 블루투스를 통해 보드와 연결되어 있는 모습이다.

6.2 문닫힘



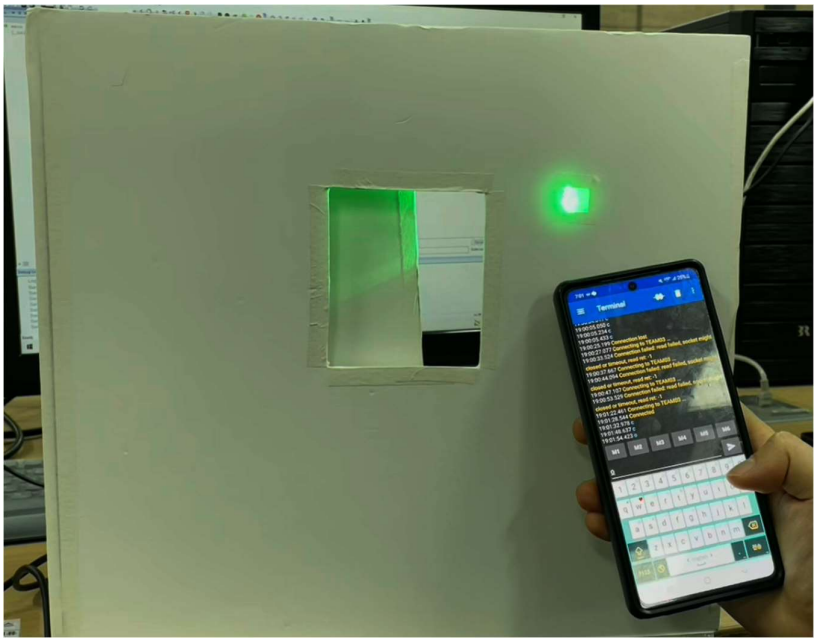
블루투스 통신으로 보드로의 "c"입력을 통해 문이 닫히는 모습이다. 문이 닫히는 경우 초음파 센서를 통해 문 앞에 사람이 있는지 확인하며 앞에 사람이 없어야 한다. 또한 문열림 감지 센서를 통해 문이 열려 있어야 한다.

6.2.1 문 앞에 사람이 있는 경우



문 앞에 사람이 있는 경우 초음파 센서를 통해 인식한 후 문이 닫히지 않는 모습이다.

6.3 문열림

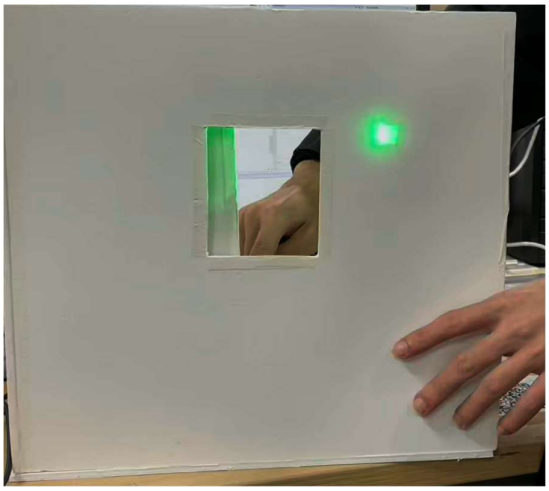


블루투스 통신으로 보드로의 "o"입력을 통해 문이 열리는 모습이다. 문열림 감지 센서를 통해 문이 닫혀 있어야 한다.

6.4 탑승 가능 인원에 따른 LED 점등

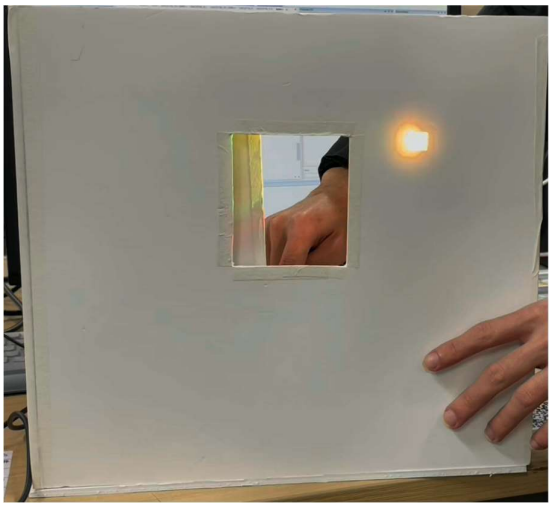
엘리베이터 안의 바닥에 설치된 압력센서를 통해 현재 탑승 인원의 무게를 측정하여 탑승 가능 인원을 LED 색상으로 표시한다.

6.4.1 원활한 경우



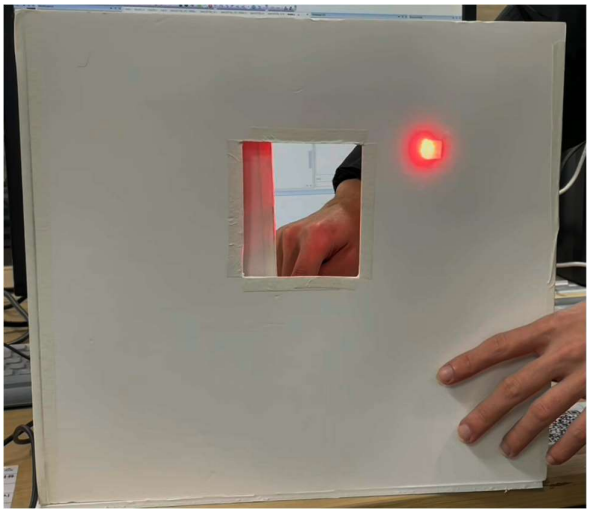
무게 측정 결과 엘리베이터의 탑승한 인원이 적어 탑승이 원활히 가능한 경우 LED를 초록색으로 점등한다.

6.4.2 혼잡한 경우



엘리베이터가 혼잡한 경우 배려를 유도하기 위해 LED를 노란색으로 점등한다.

6.4.3 만원인 경우



엘리베이터가 만원이어서 탑승이 불가능함을 나타내기 위해 빨간색으로 점등한다.

7. 한계점 및 어려웠던 부분

- 문의 개폐를 역학적으로 원활하게 하기 위해 문틀과 서보모터의 고정이 힘들었다.

- 압력센서의 경우 입력 받을 수 있는 면적이 좁아 엘리베이터의 바닥 전체를 대상으로 입력 받기 어려웠다.

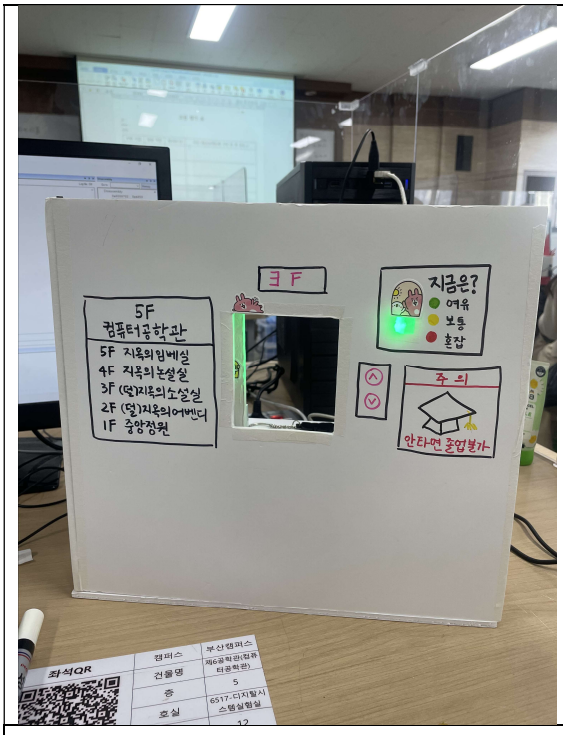
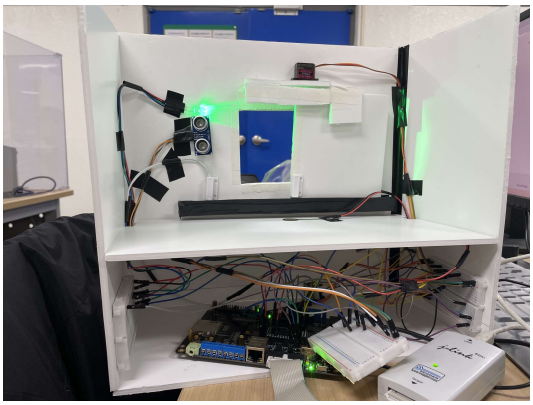
- 초음파 센서가 자체적으로 사용하는 타이머와 서보모터의 타이머 사이에 충돌이 있었는데, 타이머 번호와 채널을 다르게 하여 해결하였다.

- 핀들의 정확한 고정이 불가능하기에 인지하지 못한 핀의 빠짐이 프로젝트 진행을 지체했다.

- 문 끼임의 경우 미세한 접촉을 인식할 수 있는 센서를 찾지 못했다.

- ADC IRQ Handler 함수의 리턴 타입을 int형으로 지정했었는데, NVIC Configure가 진행되지 않았다. 따라서 리턴 타입을 void형으로 지정하여 해결하였다.

8. 최종 결과물

	
앞모습	뒷모습

9. 제안서의 기능에서 달라진 사항과 그 이유

기능 1. 출입문이 닫히는 경우 초음파 센서를 이용하여 출입문으로 다가오는 물체를 감지할 경우 문을 다시 열리게 한다. (초음파는 출입문이 닫히는 상황에서만 작동한다.)

- 변경: 출입문이 열린 상태에서 마그네틱 센서가 문이 열려 있는 것을 인식하면, 초음파 센서를 호출한다. 초음파 센서가 주변에 물체가 있다는 것을 확인하면 문을 닫지 않도록 한다.

- 이유: 출입문을 닫다가 다시 여는 과정에서, 로직을 올바르게 하여 코드를 작성하였음에도 출입문을 닫은 만큼 다시 열지 않는 현상이 발생하였다. 따라서 문 닫기를 시작하기 전에 주변에 사람이 있는지 확인하여, 사전에 개폐를 차단하고자 하였다.

기능 2. 엘리베이터의 바닥에 압력 센서를 이용하여 승객들의 무게를 측정하여 현재 상황에서 탑승가능한 인원수를 출력한다.

기능 3. STM32 보드와 블루투스 연결된 휴대폰에서 엘리베이터의 혼잡도를 확인할 수 있다.

⇒ 제안서 기능 2, 3를 새로운 기능으로 통합하였다.

- 변경: 엘리베이터 바닥 압력 센서가 인식한 값으로 엘리베이터의 혼잡도를 LED(초록, 노랑, 빨강)로 구분하여 표시한다.

- 이유: 휴대폰을 켜지 않아도 LED를 통해 엘리베이터 외부에서 직관적으로 혼잡도를 확인할 수 있도록 하여, 출퇴근 시간과 같은 혼잡한 시간대에 대기 시간을 줄일 수 있도록 하였다.