3조 IoT 프로젝트 결과보고서

3D 프린팅 기술을 활용한 Realtime DB – AI Classifier 기반 스마트 배달 알림 시스템

개발 기간 : 2021.08.10 ~ 2021.08.24

발표자: 2017312605 김요셉

2018315533 고영선, 2016310411 김영진, 2019315601 이현석, 2019315404 서유림

IoT 프로젝트 결과보고서
조/작성자 : 3조 / 김요셉
작성일자 : 2021.08.24

IoT 프로젝트 결과보고서
조/작성자 : 3조 / 김요셉
작성일자 : 2021.08.24

1. 개요

◦ 제　　목 : 3D프린팅 기술을 활용한 Realtime DB – AI Classifier 기반 스마트 배달 알림 시스템
◦ 개발 기간 : 2021.08.10 ~ 2021.08.24
◦ 조원 및 담당역할 :
김요셉: 개발 총괄, 전체 구조 및 회로 설계, 각 파트 기술자문, 클라우드-앱 통합체계 구현, C/C++기반 하드웨어 제어, Base64-jpeg 상호 변환 구현, DB읽기-쓰기 및 블루투스 CSV 데이터 파싱 구현
고영선: 모션감지 이미지 전송 디바이스 하드웨어 구현 (ESP32-Cam, OV2640, HC-SR501), 소프트웨어 가변저항 (2채널 릴레이) 구현, 동영상 및 사진 촬영
김영진: 외부 알림 디바이스 하드웨어 구현(I2C_Crystal, 3핀 릴레이 Buzzer구현, RTC), 출력형식구현
이현석: 시스템 제어 App 구현, HC-06(블루투스) 연결 및 CSV데이터 송수신 구현, Realtime DB처리
서유림: 외관 치수 측정 및 디자인 및 인벤터 모델링구현, 3D프린터로 외관 구현, Realtime DB처리

2. 개발 목표 달성 수준

2.1 개발 목표

2.1.1 도입

코로나19로 인한 배달 및 비대면 수령이 증가함에 따라 배달원의 배달도착 문자 부재로 인해 음식 만족도가 하락하는 문제점이 발생하고 있습니다. 따라서 사용자가 직접 배달 음식의 도착 여부를 확인할 수 있는 시스템이 필요하다고 생각하게 되었습니다. 이 시스템을 통해 배달원 문자 유무와 관계없이 배달 완료를 확인할 수 있기 때문에 음식 만족도 향상 및 사용자 편의성 확대에 기여할 수 있을 것이라 생각합니다.

2.1.2 개발목표

2.1.2.1 Firebase를 통해 App으로 이미지 전송
2.1.2.2 HD급 이미지 전송
2.1.2.3 Firebase의 이미지 데이터 읽기
2.1.2.4 jpeg파일을 DL모델로 분류
2.1.2.5 Delivery class인 경우 배달완료
2.1.2.6 App을 통한 외부 알림부 제어

2.2 목표 달성 수준

2.2.1 Firebase를 통해 App으로 이미지 전송

OV2640 카메라 모듈을 통해 esp_cam_fb 라는 구조체 타입에서의 파일 버퍼에 JPEG 이미지데이터를 바이너리 형태로 저장할 수 있음. 하지만 Firebase에서 제공하는 API 중 아두이노에서 사용 가능한 것은 없으므로 String 전송 API를 사용하기 위해 Jpeg바이너리 파일을 Base64 포맷 스트링으로 인코딩한 후 앱인벤터 스타일 String으로 변환하여 Firebase 클래스의 setString()메소드로 전송 구현 성공했습니다.

2.2.2 HD급 이미지 전송

VGA급 이미지 전송 성공, 그 이상의 경우 String 분해 후 전송을 시도했으나 Arduino에서 제공하는String 클래스는 C스타일의 문자열과 달리 분해의 과정에서 특정 index의 포인터를 제공하는 것이 아니라 새로운 String을 duplication하는 과정이 수반되어 메모리 부족을 야기했습니다. 이는 향후 SD카드를 가상메모리로 사용한다면 해결 가능할 것으로 기대됩니다.

2.2.3 Firebase의 이미지 데이터 읽기

적절한 형식의 String데이터를 앱인벤터에 적용 가능한 KIO4_Base64 extension으로 base64 string으로부터 jpeg파일로 변환하였습니다. 이를 통해 이미지 데이터 읽기 구현을 성공했습니다.

2.2.4 jpeg파일을 DL모델로 분류

Personal Image Classifier로 DL모델을 학습하려고 했으나 지나치게 많은 시간이 소요되어 기존에 이미 구현된 Look Extension의 포팅으로 대체하였습니다. Delivery 클래스라고 볼 수 있는 것은 plastic bag과 light bulb의 적절한 조합임을 발견하여 이를 통해 구현에 성공했습니다.

2.2.5 Delivery class인 경우 배달완료

분류 모델에 의한 결과값에 따라 사전에 정의한 end() 함수를 호출하여 알림부와 파이어베이스, 그리고 센서모듈 전체에 배달완료 시그널과 데이터를 보내고 어플리케이션 내부에서 음성데이터로 배달완료임을 공지하는 구현을 성공했습니다.

2.2.6 App을 통한 외부 알림부 제어

외부 알림부 제어를 위해 블루투스를 사용하기로 했는데 CSV 형태의 스트링 데이터를 전송하여 정보를 전달하는 것이 유효할 것을 판단되어 데이터 형식을 정의하고 알림부 조건문 형태로 제어하는 형태로 논리를 만들어 봤습니다. 이것이 좋은 결과를 만들어 구현에 성공했습니다.

3. 결과물

3.1 구현 세부 사항

3.1.1 소프트웨어

(소스코드: https://github.com/yspkm/delivery-notification-system)

3.1.1.1 센서부 소프트웨어

3.1.1.1.1 camera

카메라의 촬영은 espcam 라이브러리에 들어있는 esp_camera_fb_get()을 통해 OV2640 카메라 모듈을 작동시킵니다. 먼저, 촬영에 앞서서 setCameraConfig() 메소드를 사용하여 여러 설정을 진행하는데, cam_config.pixel_format = PIXFORMAT_JPEG;를 통해 픽셀 포멧을 jpeg으로 설정하면 이미지 데이터가 camera_fb_t 구조체의 (uint8_t *) buf에 jpeg의 형태로 저장됩니다. 저는 이 get API를 저의photo2Base64메소드와 결합하여 getPhotoThenSendToFirebase() 메소드로 묶었습니다. 이것은 Fireabse의 setString을 포함합니다.

```
void getPhotoThenSendToFirebase(void)
{
  camera_fb_t *cam_fb = NULL;
  cam_fb = esp_camera_fb_get();

  photo2Base64(cam_fb);
```

```
if (is_authenticated && Firebase.ready())
{
  if (Firebase.setString(firebase_data, photo_path.c_str(), photo_data))
  {
    Serial.println("PASSED");
    Serial.println("PATH: " + firebase_data.dataPath());
    Serial.println("TYPE: " + firebase_data.dataType());
    Serial.println("ETag: " + firebase_data.ETag());
    Serial.print("VALUE: ");
    Serial.println("complete");
    printResult(firebase_data); //see addons/RTDBHelper.h
    Serial.println("------------------------------------");
    Serial.println();
  }
  else
  {
    Serial.println("FAILED");
    Serial.println("REASON: " + firebase_data.errorReason());
    Serial.println("------------------------------------");
    Serial.println();
  }
}
```

3.1.1.1.2 firebase

3.1.1.1.2.1 전송

Firebase의 API는 이해하는 것을 목적으로 했고, 큰 수정 없이 바로 사용했습니다. setString메소드는 자동으로 String클래스를 FirebaseJSON클래스의 인스턴스로 변환하고 이를 정해진 path변수의 값에 맞는 위치로 전송하게 됩니다. 이에 앞서서 firebaseInint()메소드로 다양한 설정 및, Authentication, sign up, token처리 등을 해 주는 절차가 아두이노의 setup에서 진행됩니다.

3.1.1.1.2.2 수신

Firebase의 데이터를 수신하는 과정은 getString()메소드 (또는 getInt()메소드)를 사용하게 되며, 사전에 global variable로 선언된 FirebaseData 인스턴스에 저장됩니다. 이 값은 stringData() 메소드를 통해 불러올 수 있습니다. 저희는 이런 데이터의 송수신과정에서 타입에 관한 여러 에러를 접할 수 있었는데 이것은 앱인벤터에서 파이어베이스에 저장하는 데이터가 대부분 String으로 인식된다는 점에 기인합니다. 따라서 boolena타입이나 Integer 타입의 변수로 생각했으나 String타입인 경우가 많기 때문에 가능하면 데이터 전송에서는 문자열 형태로 취급해야 한다는 것을 알 수 있습니다.

3.1.1.1.3 wifi

Wifi의 연결은 Arduino에서 제공하는 WiFi 클래스를 사용합니다. 구동에 앞서, setup()함수에서 wifiInit()을 작동시키는데, 이것은 static WiFi인스턴스를 생성하는 것 같습니다. 사전에 private_info.h 헤더파일에 저장된 SSID와 Password를 통해 공유기와 연결되게 됩니다.

3.1.1.1.4 base64

처음에 저희가 직면한 문제는 어떻게 이미지를 Firebase의 Realtime DB에 전송하는가에 대한 문제였습니다. 파이어베이스에서 제공하는 API에는 Jpeg파일을 전송하는 것이 없습니다. 그러므로 저희는 일반적으로 이메일 및 URL등에 활용되는 Base64 인코딩을 이용하여 String으로 변환하고 이를 setString()으로 전송해주는 방법을 택했습니다.

<photo2Base64메소드>

```
void photo2Base64(camera_fb_t *cam_fb)
{
  photo_data.clear();
  photo_data.concat("\\\"");
  char *input = (char *)cam_fb->buf;
  char output[base64_enc_len(3)];

  for (int i = 0; i < cam_fb->len; i++)
  {
    base64_encode(output, (input++), 3);
    if (i % 3 == 0)
      photo_data.concat(String(output));
  }
  photo_data.concat(String(output));
  photo_data.concat("\\\"");
}
```

3.1.1.1.5 motion

모션 가동에 앞서서 loop구문에서는 getString메소드를 통해 sensor_control 태그(key)의 값을 읽어오게 됩니다(아마도 딕셔너리 형태의 해쉬맵을 이용하는 것 같습니다.). 이 값은 앱인벤터에서 배달시작을 할 때 "true"값으로, 배달 종료시에 "false"값으로 설정되는데, 이를 통해 센서를 불필요하게 on하지 않습니다.

3.1.1.2 외부알림모듈

3.1.1.2.1 LiquidCrystal_I2C (LCD2004) 제어

LCD 모듈의 제어는 LiquidCrystal_I2C 클래스를 이용해서 하게 됩니다. 이 클래스의 인스턴스는 주소값(0x27), column값, 그리고 row값으로 이루어져 있습니다. (row-column 순이 아닌 이유는 행렬보다 좌표계의 개념으로 접근하기 때문인 것 같습니다.) 내부 API가 매우 쉽게 사용할 수 있는 형태이므로 적절히 출력할 수 있는 정규화 된 형태의 String을 만드는 메소드를 통해 (sprintf를 포함하는 getTimeFormString) 현재 시간 및 배달 도착 예정시간을 출력할 수 있습니다. 또한 lcdPrintStatus 메소드에서는 LcdControl enumerator 타입을 활용해 다양한 상황에 대한 출력이 제어되도록 설계했습니다.

```
void lcdPrintStatus(LcdControl lcd_control)
{
  /* LCD Display
  LCD Display Example
  NOW:08:15|2021-08-21
  EXP:08:45|2021-08-21
  MOTION DETECTED!
  DELIVERY COMPLETE!!! */
  switch (lcd_control)
  {
  case LCDINIT:
    lcd.setCursor(0, 0); // setCursor(col, row)
    getTimeFormStringNow();
```

```
    lcd.print("NOW:" + time_form_str); // Display Current Time
    lcd.setCursor(0, 1);
    getTimeFormStringArrv(arrv_time);
    lcd.print("EXP:" + time_form_str); // Display estimated arrival time
    lcd.setCursor(0, 2);
    lcd.print("MOTION CHECKING..."); // Display if motion is detected
    break;
  case TIME_NOW_LINE0:
    lcd.setCursor(0, 0);
    getTimeFormStringNow();
    last_minute = rtc.GetDateTime().Minute();
    lcd.print("NOW:" + time_form_str); // Display Current Time
    break;
  case MOTION_LINE2:
    lcd.setCursor(0, 2);
    lcd.print("MOTION DETECTED!!"); // Motion Detection Notification
    break;
  case DELIVERY_END_LINE3:
    lcd.setCursor(0, 3);
    lcd.print("DELIVERY COMPLETE!!"); // Delivery End Notification
    break;
  default:
    Serial.println("error: lcd control");
  }
}
```

3.1.1.2.2 SoftwareSerial (HC06, 블루투스) 제어

블루투스 모듈의 제어는 아두이노의 기본 통신인 직렬통신을 이용하게 됩니다. 블루투스로 받는 정보는 오직 String으로 한정했습니다 그 이유는 무엇보다도 Occam's Razor를 따르고자 복잡성을 최소화하여 모든 것을 구현하고 싶었기 때문입니다. 또한 Unix운영체제의 시스템콜이 바이트 단위로 처리를 한다고 배웠는데, 분명 SoftwareSerial의 많은 메소드들은 은 어떤 임베디드 환경에서의 Unix 느낌의 운영체제 하에 시스템콜을 사용할 것이라고 생각되어서 바이트 단위와 일치하는 문자열을 버퍼로 이용하는 스트링클래스만을 사용하는 것이 더 안전할 것이라고 생각했기 때문에 저는 더욱 확고히 String만을 효율적으로 다루고 싶었습니다.

일반적으로 파일 스트림이나 다양한 네트워크 프로토콜에서 사용하는 방식을 개념적으로 모방하여 "모듈이름, 항목이름, 값₩n"의 형태로 전송하되, SoftwareSerial 클래스의 readUntil()메소드를 사용하여 개행 문자 직전까지 읽어오도록 하고, 간단한 parseString()메소드를 통해 3개의 String 값을 전역변수에 저장했습니다. 전역변수를 사용한 이유는 어디서 어디까지가 main인지 알 수가 없기 때문입니다. 자세한 것을 알 수 없지만 main내부에서 loop() 함수의 블록이 반복적으로 실행되어 local variable이 갱신되는 것 같았습니다.

<parseString 메소드>

```
void parseString(void)
{
  byte idx_module = received_str.indexOf(','); //finds location of ','
  byte idx_item = received_str.indexOf(',', idx_module + 1);
```

```
  header_module = received_str.substring(0, idx_module);
  //module name
  header_item = received_str.substring(idx_module + 1, idx_item);
  //item name
  header_value = received_str.substring(idx_item + 1, received_str.length())
; // value

  header_module.replace(',', ' ');
  header_item.replace(',', ' ');
  header_value.replace(',', ' ');

  header_module.trim();
  header_item.trim();
  header_value.trim();
}
```

3.1.1.2.3 소프트웨어 가변저항

외부에서 소프트웨어적인 가변저항을 구현해보았습니다. 아날로그 아웃을 사용하는 방법도 있었지만 릴레이를 통해 논리적인 제어를 구현해 보고 싶었기 때문입니다. 부저 자체를 LOW하는 것으로 0단계 mute를 표현했고, 1번 릴레이로 1단계 음량, 1번릴레이와 2번릴레이로 2단계 및 3단계 음량을 구현했습니다. 부저 모듈의 경우에는 passive 부저가 tone()메소드를 사용해야 한다는 문제로 인해 부득이하게도 active 부저를 사용하게 되었는데, active부저의 경우 3핀 모듈이 없어서 제가 1채널 릴레이를 활용하여 3핀부저를 만들었습니다. 부저를 제어하는 핵심 함수는 다음과 같습니다.

```
void turnOnBuzzer(void)
{
  //Buzzer noise occurs at different levels of volume
  //level0 is mute
  if (buzzer_level == LEVEL0)
  {
    digitalWrite(buzzer_pin, LOW);
  }
  else if (buzzer_level == LEVEL1)
  {
    digitalWrite(relay_pin[0], HIGH);
    digitalWrite(buzzer_pin, HIGH);
  }
  else if (buzzer_level == LEVEL2)
  {
    digitalWrite(relay_pin[0], LOW);
    digitalWrite(relay_pin[1], HIGH);
    digitalWrite(buzzer_pin, HIGH);
  }
  else if (buzzer_level == LEVEL3)
  {
    digitalWrite(relay_pin[0], LOW);
    digitalWrite(relay_pin[1], LOW);
    digitalWrite(buzzer_pin, HIGH);
  }
```

```
  else
  {
    Serial.println("buzzer level error");
  }
}
```

3.1.1.2.4 Realtime Clock (RTC모듈) 제어

RTC모듈 또한 LiquidCrystal_I2C클래스처럼 RtcDS1302 클래스로 제어가 됩니다. RtcDS1302클래스의 인스턴스를 생성하게 되는데 이 타입은 generics를 포함합니다. 저희는 ThreeWire클래스를 넣었고, 각가 data, clock, reset핀을 의미합니다. RTC모듈에서의 시간데이터는 RtcDataTime 클래스의 변수에 저장되는데, 이 값을 getTime()메소드로 시분초년월일 각각의 변수에 저장하고, 이를 getTimeFormString에서 문자열로 변환한 후에 lcd에 출력합니다. 여기서 전역변수가 상당히 많이 쓰이는데, 이것은 아두이노의 함수의 특성상 main함수의 범위를 알기 어렵다 보니 부득이한 것으로 간주됩니다. 대다수의 아두이노 프로젝트에서도 관행적으로 전역변수를 사용 중인 것 같습니다.

```
void getTimeNow(uint16_t *hour, uint16_t *min, uint16_t *day, uint16_t *month, u
int16_t *year)
{
  RtcDateTime date_time = rtc.GetDateTime();
  *hour = date_time.Hour();
  *min = date_time.Minute();
  *day = date_time.Day();
  *month = date_time.Month();
  *year = date_time.Year();
}

void getTimeFormString(uint16_t hour, uint16_t min, uint16_t day, uint16_t month
, uint16_t year)
{
  char *time_fstr = NULL;
  time_fstr = (char *)malloc(18 * sizeof(char));
  sprintf(time_fstr, "%02u:%02u|%4u-%02u-%02u", hour, min, year, month, day);
  time_form_str = String(time_fstr);
  Serial.println("time is: " + time_form_str);
  free(time_fstr);
}

void getTimeFormStringNow(void)
{
  uint16_t hour, min, day, month, year;
  getTimeNow(&hour, &min, &day, &month, &year);
  getTimeFormString(hour, min, day, month, year);
}

void getTimeFormStringArrv(uint16_t arrv_minute)
{
  uint16_t min_new, hour_new;
  uint16_t hour, min, day, month, year;
  getTimeNow(&hour, &min, &day, &month, &year);
  min += arrv_minute;
  min_new = min % 60;
  hour_new = hour + (min / 60);
  getTimeFormString(hour_new, min_new, day, month, year);
```

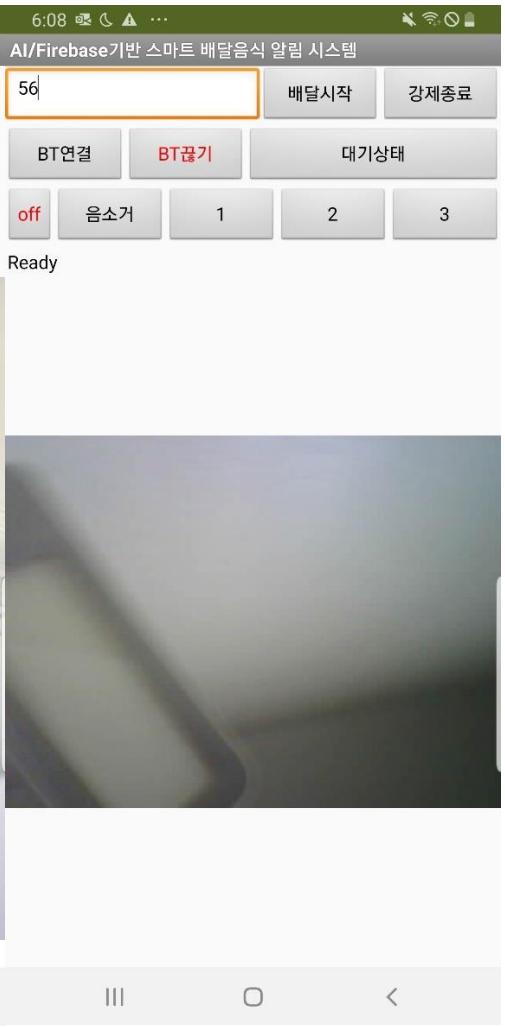
```
}
```

3.1.1.3 어플리케이션


뷰어에 숨겨진 컴포넌트 나타내기
전화 크기 (505,320)
9:48
AI/Firebase기반 스마트 배달음식 알림 시스템
배달시작
강제종료
BT연결
BT끊기
대기상태
off
음소거
1
2
3
기다리는 중
현관 이미지
보이지 않는 컴포넌트
Look1
블루투스클라이언트1
알림1
파이어베이스DB1
KIO4_Base641
음성변환1

예시1-이미지 데이터 로드

예시-2 이미지 데이터 분류

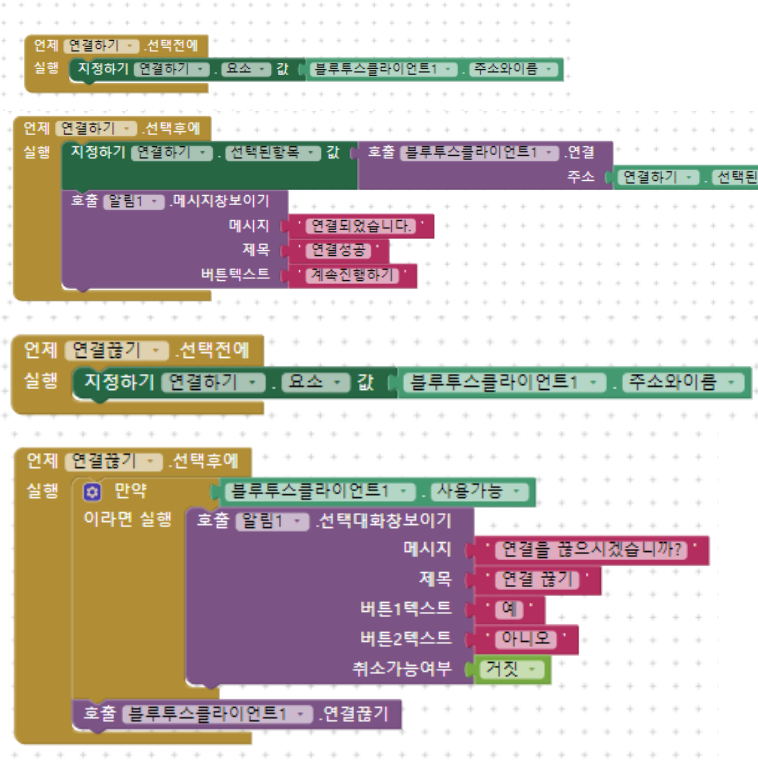


3.1.1.3.1 파이어베이스에서 imgdata를 받아오는 부분

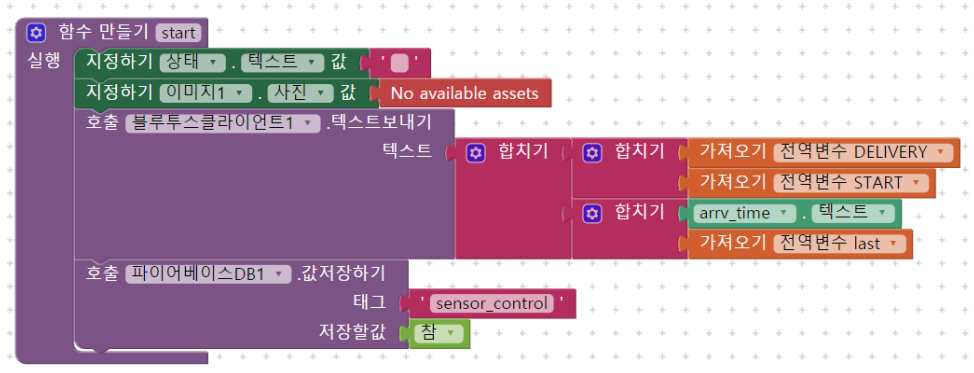
3.1.1.3.2 imgdata를 jpeg으로 변환 후 AI Classification하는 부분

3.1.1.3.3 블루투스 연결을 통해 external notification device와 송수신하는 부분

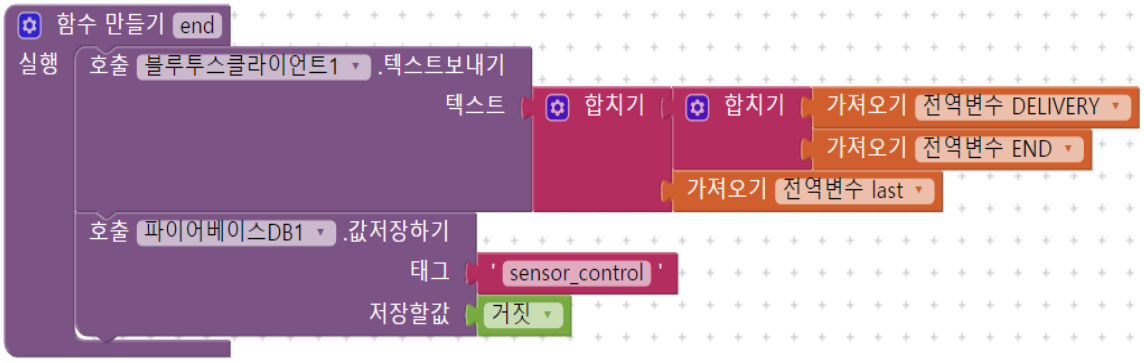
3.1.1.3.3.1 블루투스 연결부분 구현



3.1.1.3.3.2 배달 시작 구현



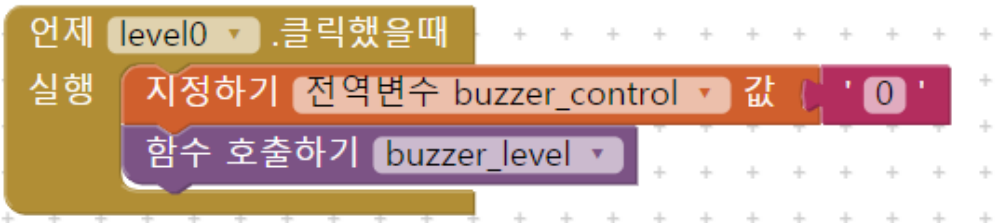
3.1.1.3.3.3 배달 종료 구현



3.1.1.3.3.4 대기상태 구현

3.1.1.3.3.5 buzzer 레벨 설정 구현

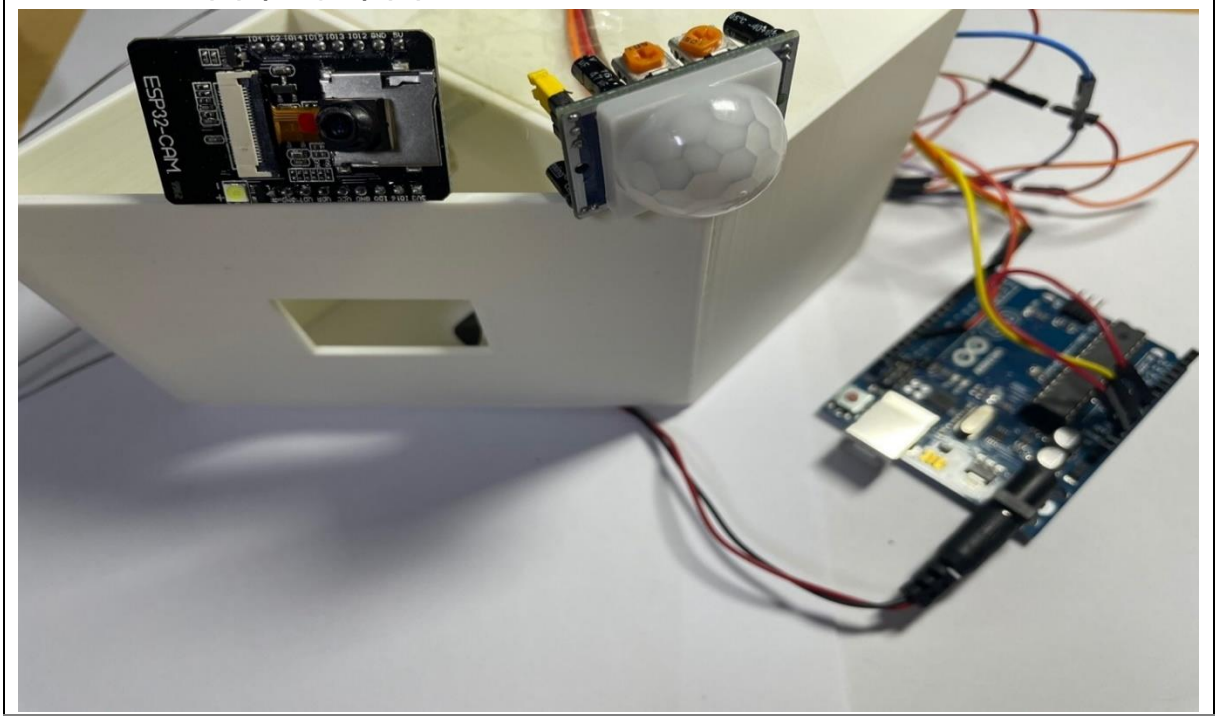
사용 예시)



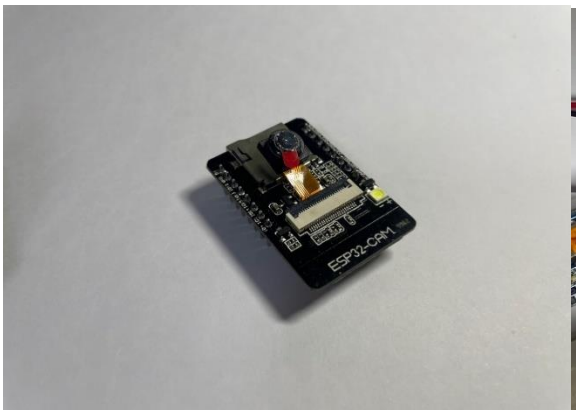
3.1.1.3.4 전역변수 값

3.1.2 하드웨어

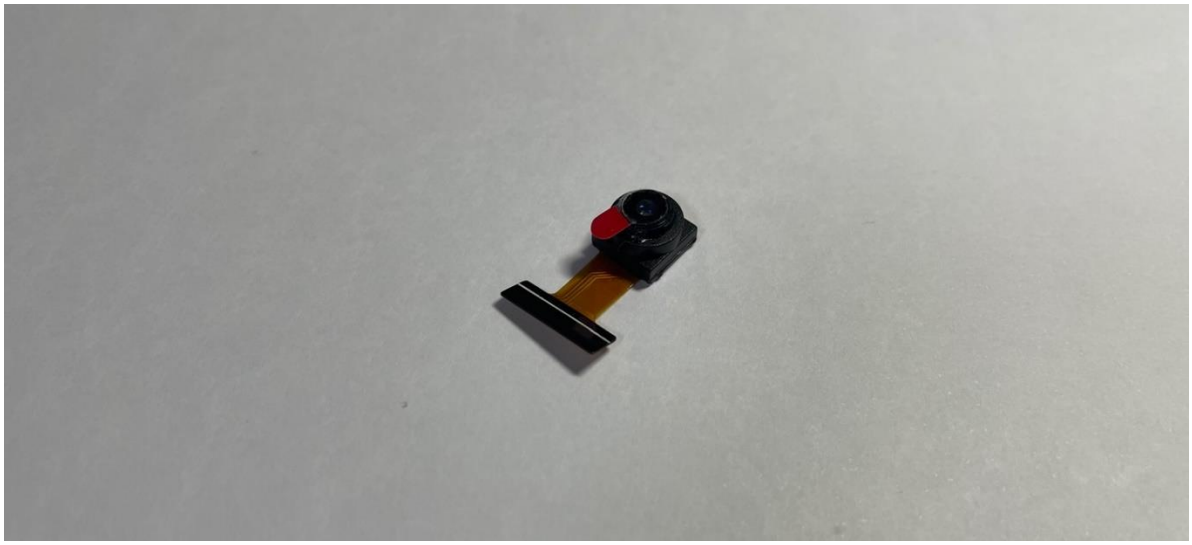
3.1.2.1 이미지 전송 디바이스



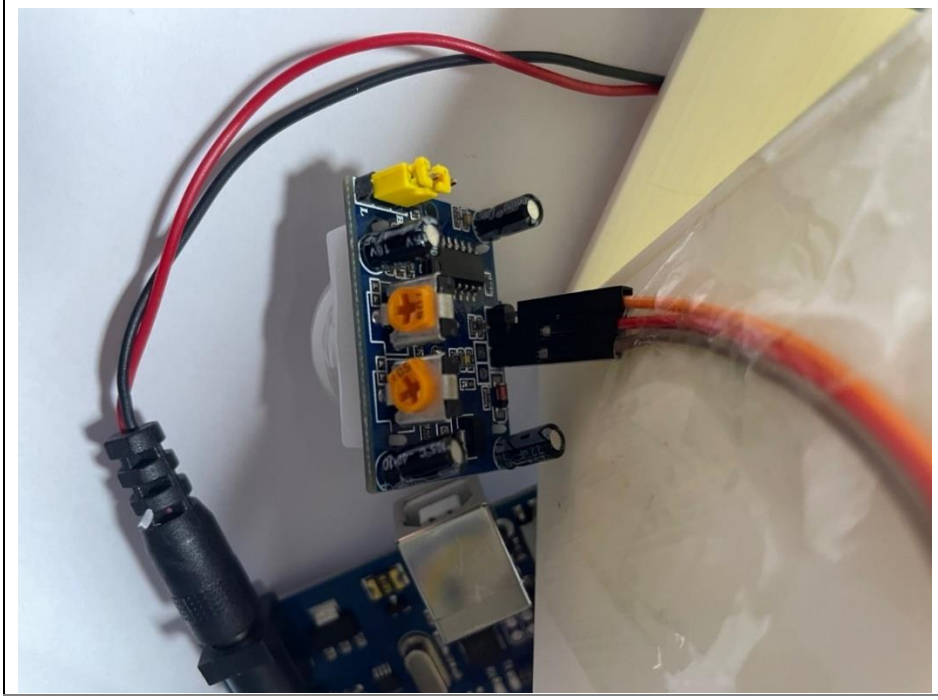
3.1.2.1.1 ESP32



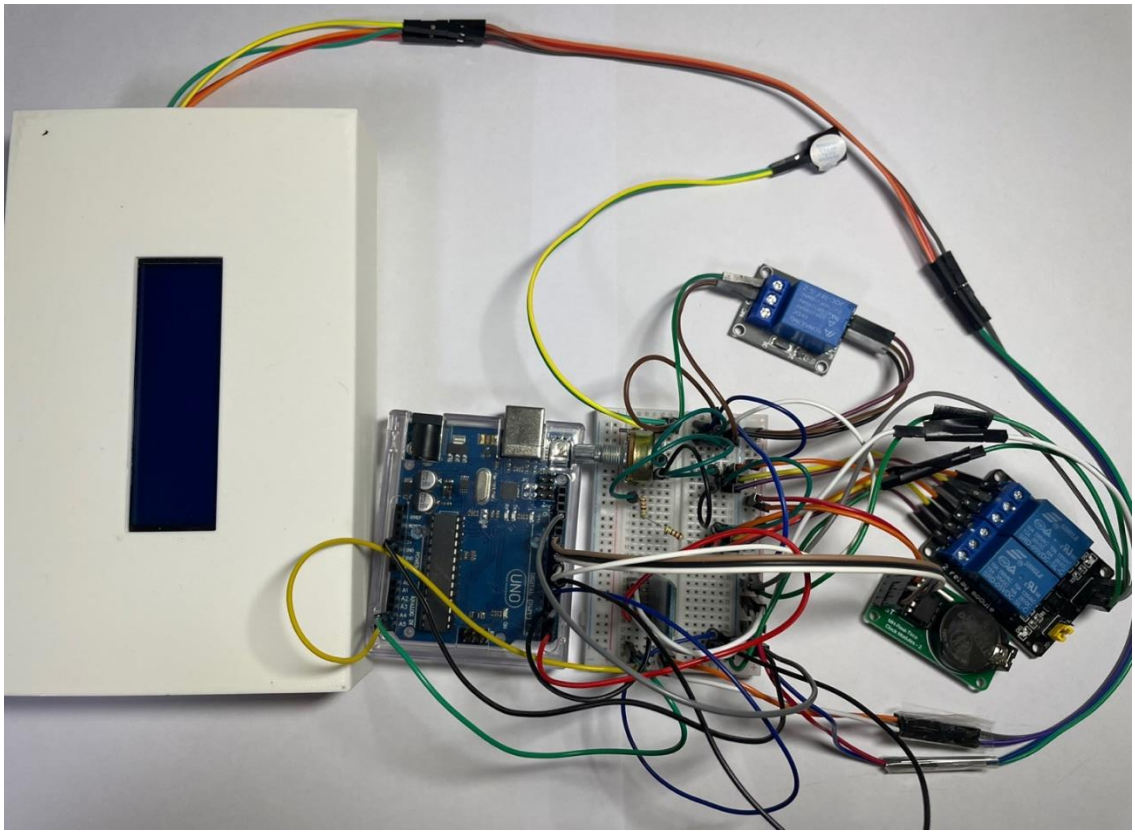
3.1.2.1.2 OV2640



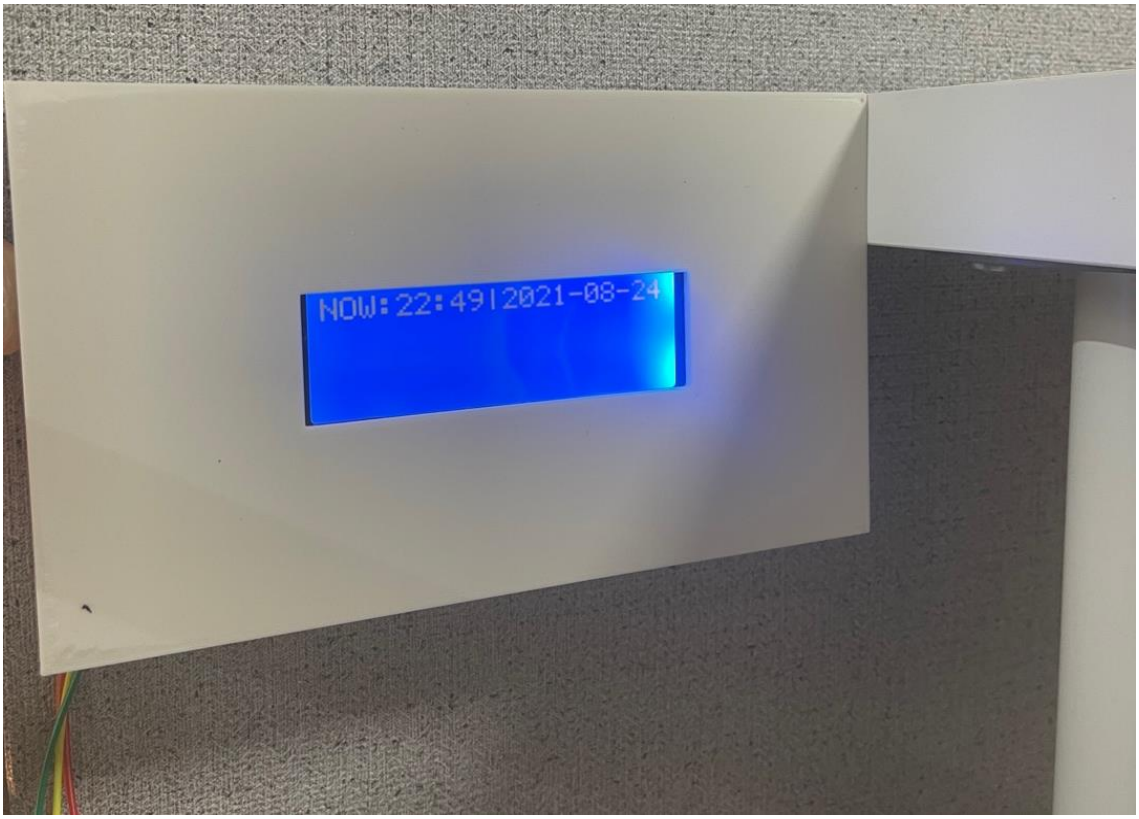
3.1.2.1.3 HC-SR501



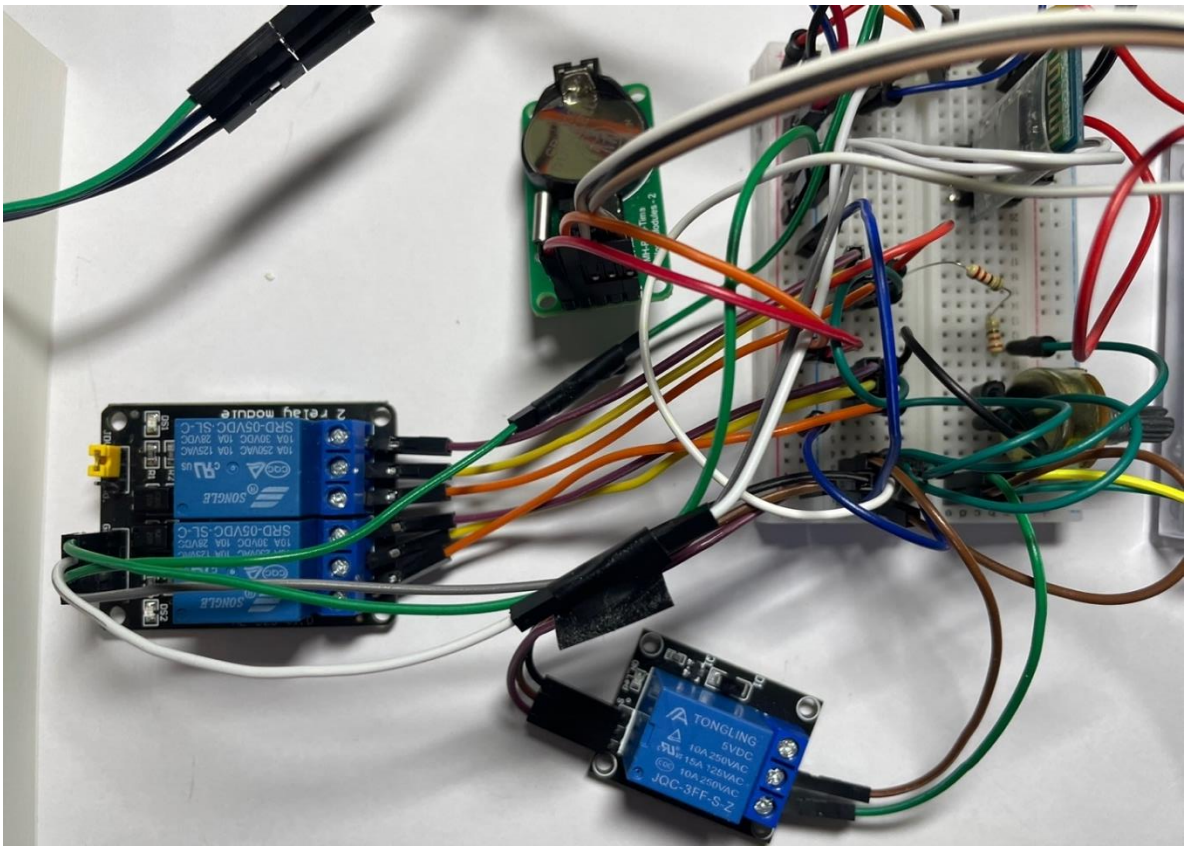
3.1.2.2 외부 알림 디바이스



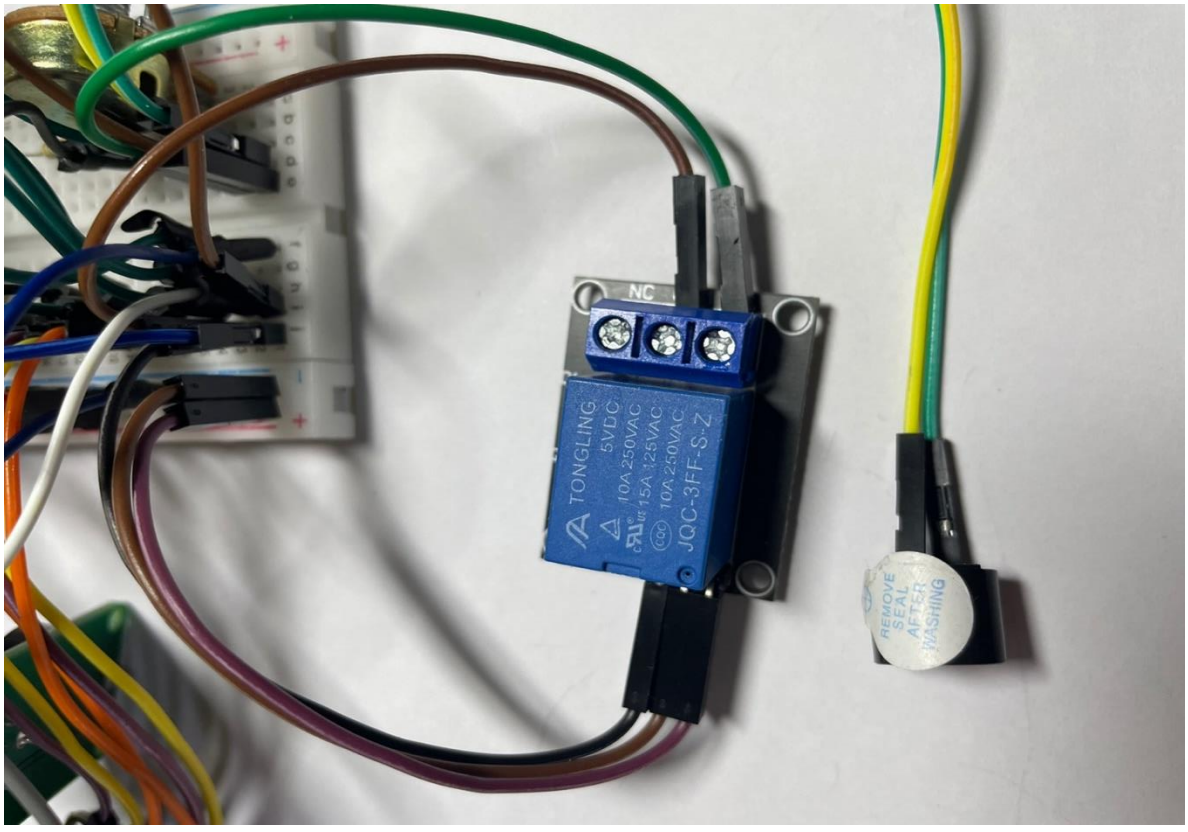
3.1.2.2.1 LiquidCrystal_I2C



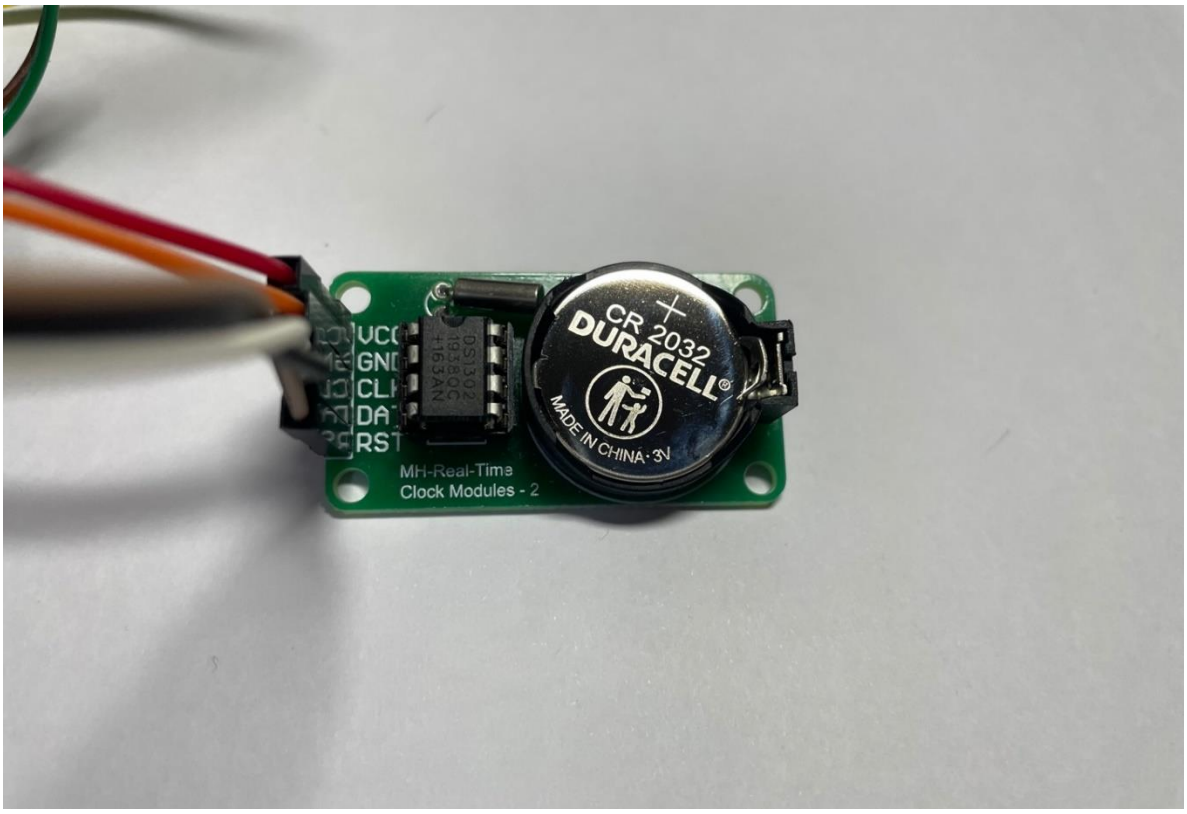
3.1.2.2.2 소프트웨어 제어 가변저항



3.1.2.2.3 3핀 액티브 buzzer



3.1.2.2.4 RTC 모듈

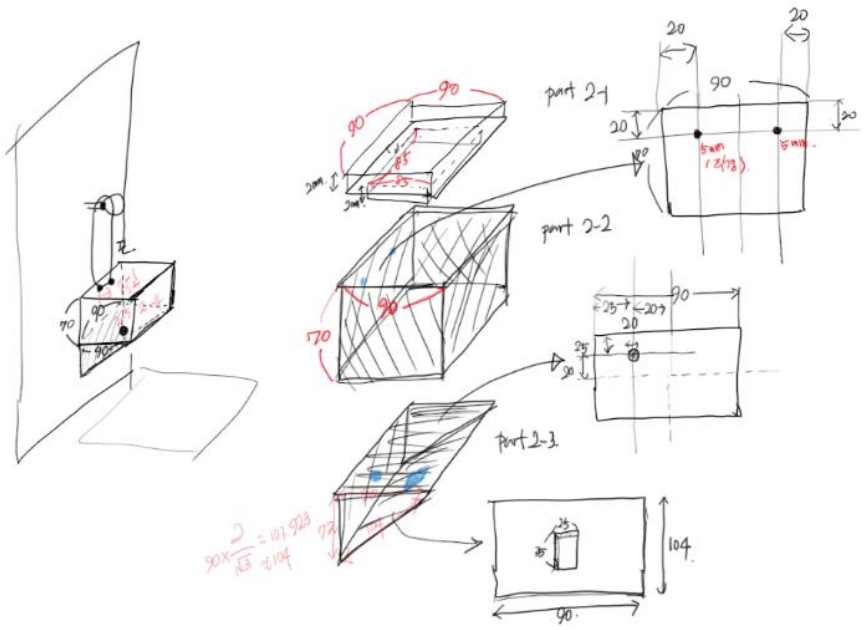


3.1.3 디자인 및 모델링

3.1.3.1 Inventor 모델링

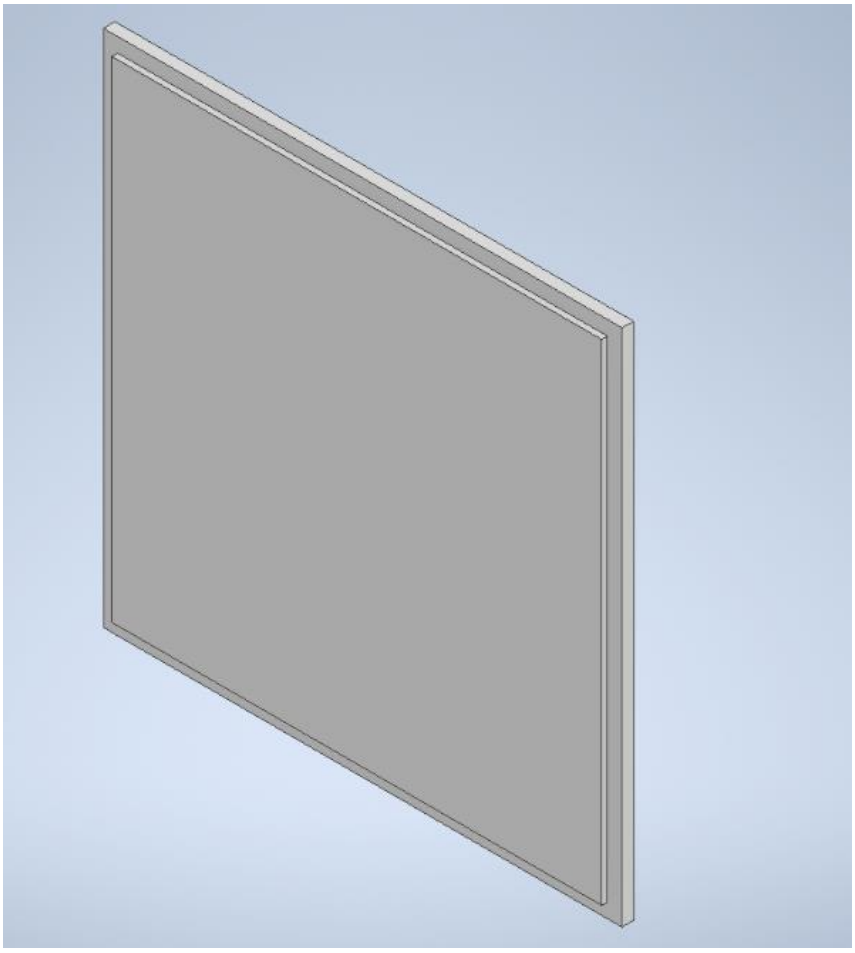
3.1.3.1.1 이미지 전송 디바이스

3.1.3.1.1.1 개요도

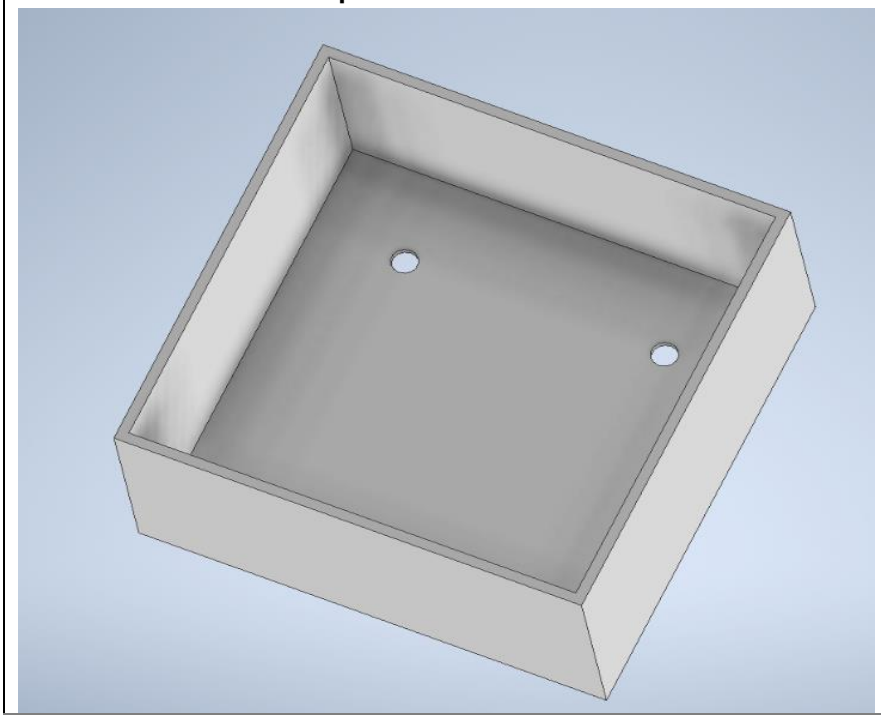


3.1.3.1.1.2 모델링 이미지

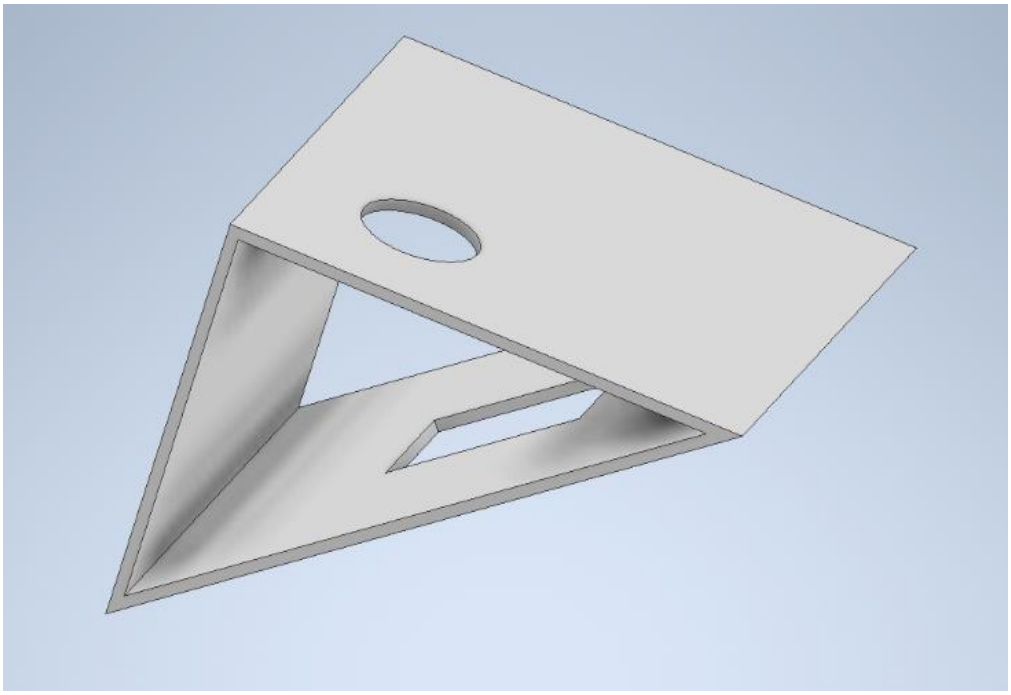
3.1.3.1.1.2.1 part 2-1



3.1.3.1.1.2.2 part 2-2

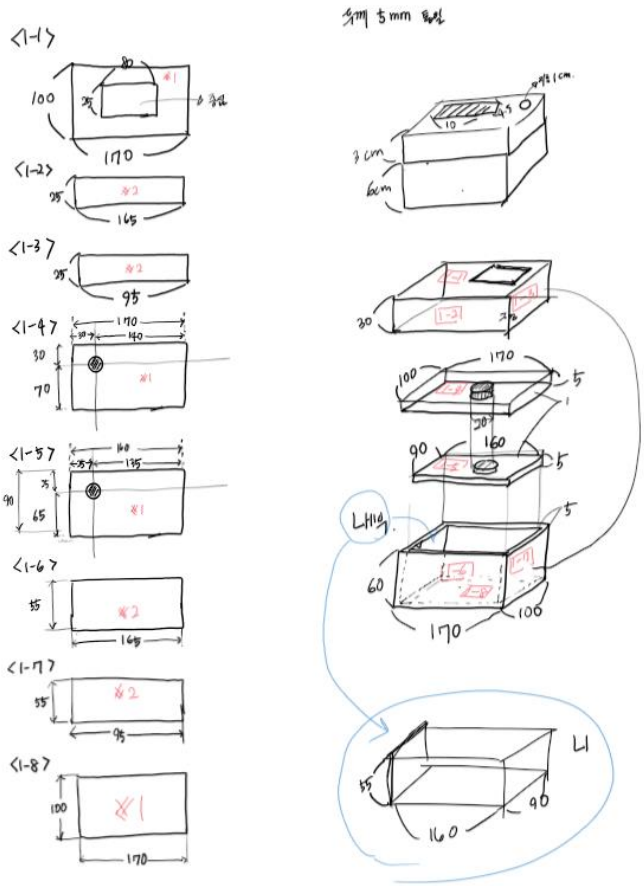


3.1.3.1.1.2.3 part 2-3



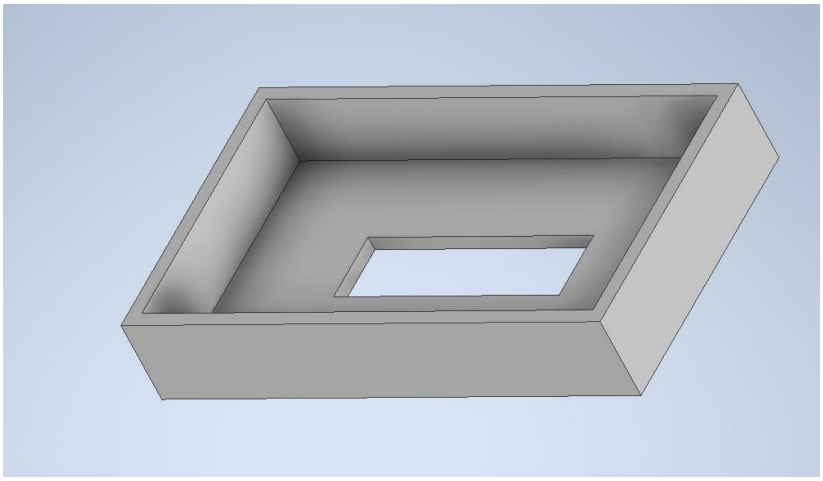
3.1.3.1.2 외부 알림부 모듈

3.1.3.1.2.1 개요도

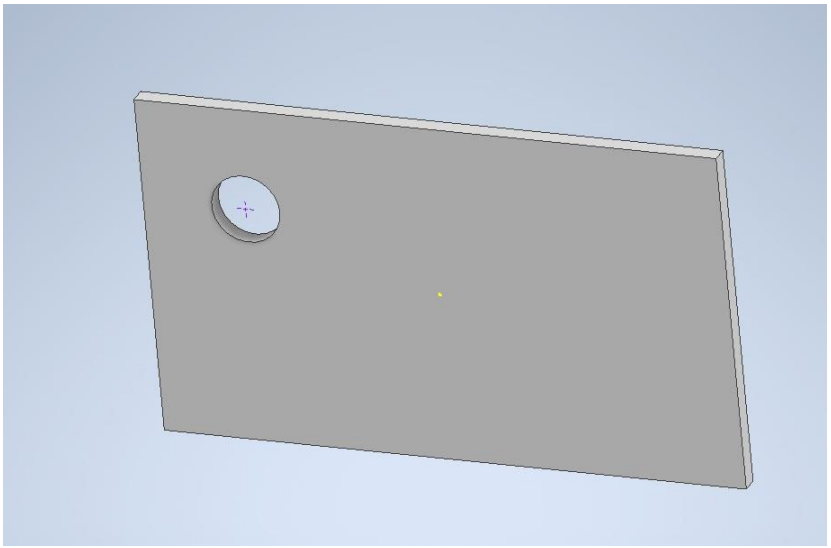


3.1.3.1.2.2 모델링 이미지

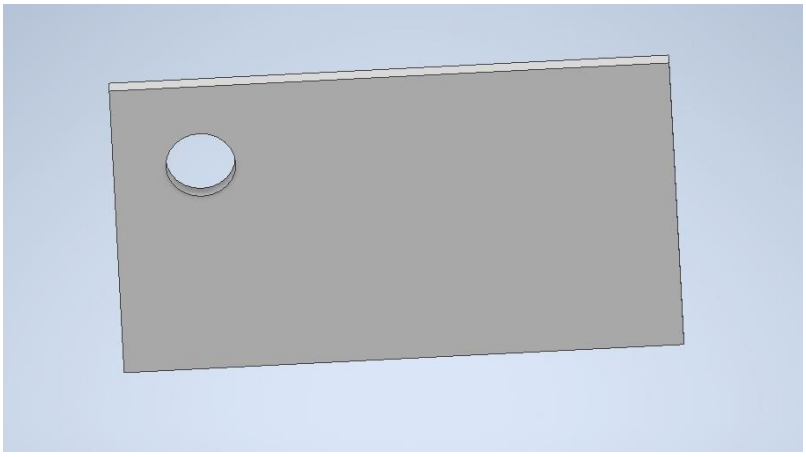
3.1.3.1.2.2.1: 1-1, 1-2, 1-3



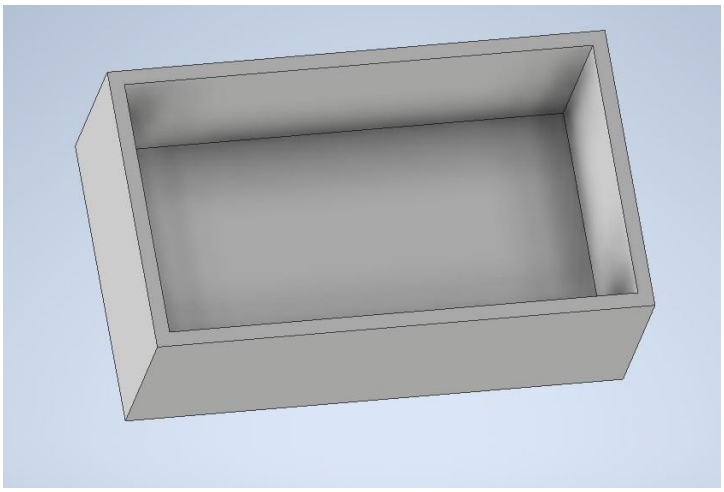
3.1.3.1.2.2.2: 1-4



3.1.3.1.2.2.2: 1-5

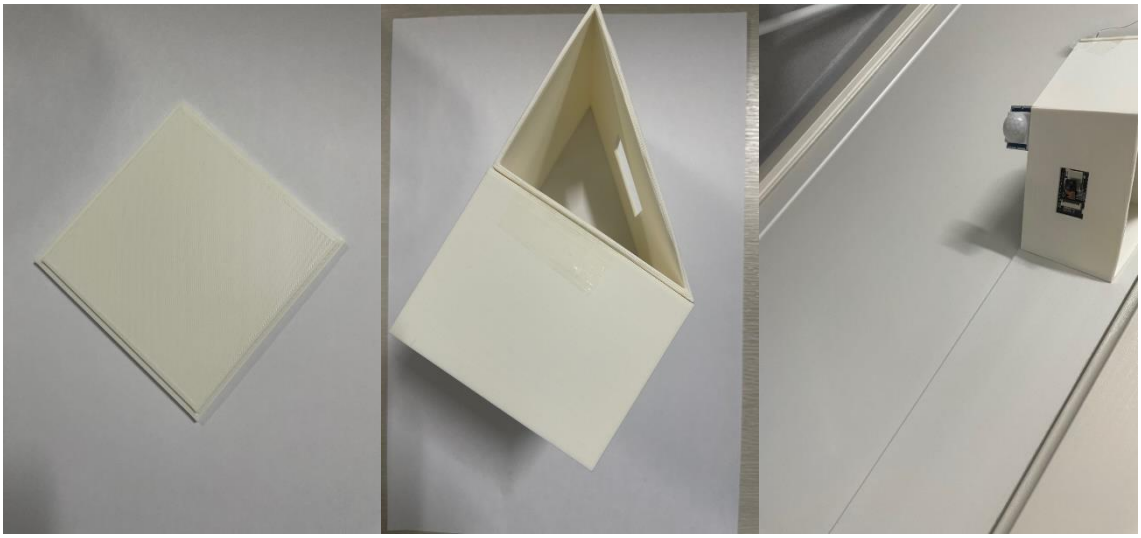


3.1.3.1.2.2.2: 1-6, 1-7, 1-8

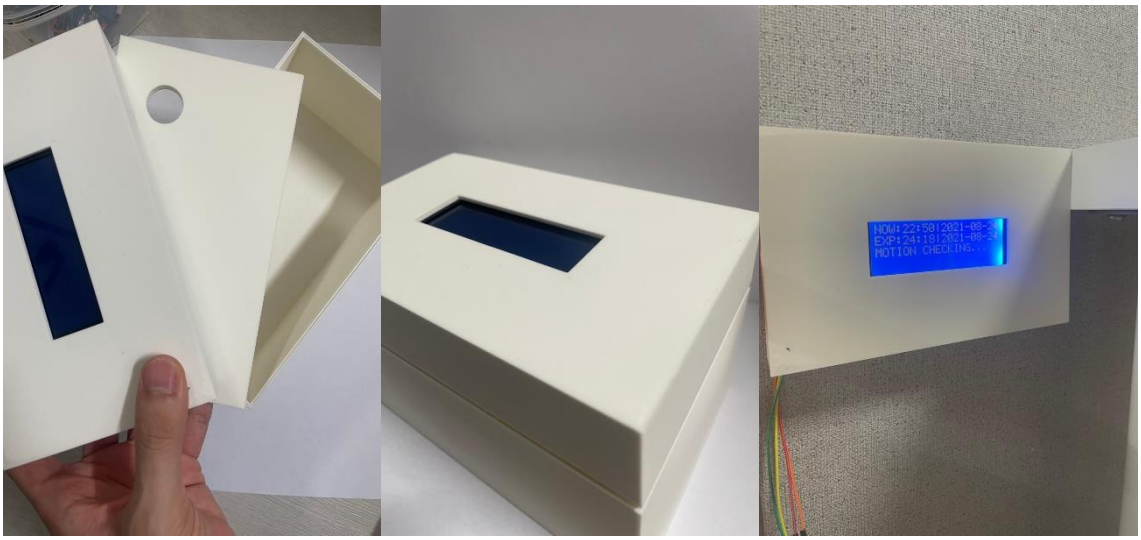


3.1.3.2 3D프린팅 부품

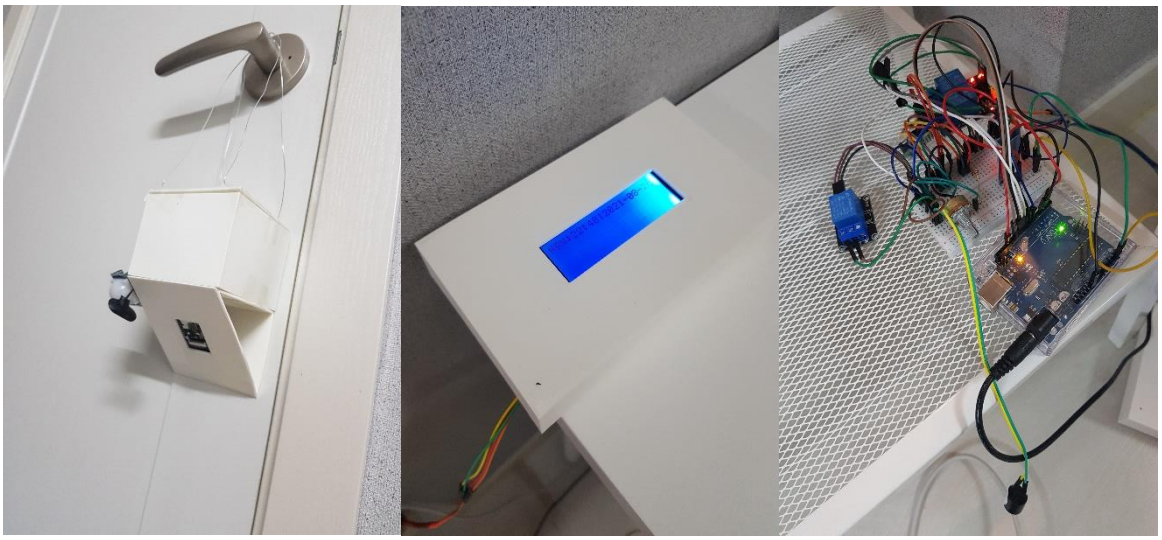
3.1.3.2.1 이미지 전송 디바이스



3.1.3.2.2 외부 알림 디바이스



3.1.3.3 최종 완성 디자인



3.2 기술적 문제 및 해결 과정

3.2.1 소프트웨어적 결함

3.2.1.1 이미지 전송 이슈

3.2.1.1.1 Jpeg file의 base64 encoding 문제

Esp32 라이브러리 상에서 jpeg 파일을 다루는 방식은 camera_fb_t라는 C언어 Structure를 사용합니다. 해당 구조체 내부에서 (uint8_t *) buf는(char랑 동일) uint8_t 포인터로 구성되어 있습니다 (바이트 단위) 따라서 이 값을 각 element 단위로 base64 스트링으로 변환하여 기존의 Firebase class의 setString() API를 큰 변경 없이 사용할 수 있도록 하여 개발시간을 단축했습니다. 처음에는 setJSON이라는 API를 수정하여 jpeg에 해당되는 char배열을 전송하려고 했지만 이것에서 너무 많은 기술적 문제가 발생했기 때문에 기존에 이메일 전송 및 web URL등에서 사용되는 기술 Base64를 도입하게 되었습니다.

3.2.1.1.2 base64 string의 firebase DB로의 전송 문제

저희는 String 클래스 (C++의 string클래스를 아두이노에 맞게 수정한 클래스)를 자동으로 다양한 언어가 공유할 수 있는 JSON 구조에 맞게 변형하여 Realtime DB로 전송하려고 했습니다. 그러나 이것을 FirebaseJSON 클래스 형태로 바꾸거나 전송에 관련된 API의 내부 구조를 뜯어고치기 어려웠고, 이에 대한 대안으로 Firebase클래스의 setString()메소드를 사용하여 전송에서의 문제를 해결했습니다.

3.2.1.1.3 firebase DB에서 app inventor로 base64 string 전송

App inventor는 Firebase DB의 key(tag)를 읽어올 때, value에서 확장 문자 W"를 시작과 끝 부분에서 인식되는 경우에만 이것을 String 타입으로 인식합니다. 따라서 photo2Base64메소드에서 이미지 데이터의 시작과 끝 부분에 해당 escape character를 concat() 메소드로 붙여서 base64스트링을 만들게 됩니다. 이렇게 전송된 String은 "₩"스트링내용₩""의 형태가 되며, 비로소 app inventor에서 String 타입으로 인식된다는 것을 확인할 수 있었습니다.

3.2.1.1.4 app inventor에서 base64 string 처리 문제

3.2.1.1.4.1 파일 경로 문제

안드로이드 운영체제는 일반적인 Unix 디렉토리 구조와 달리 루트 path로 접근이 불가능한 것 같

습니다. 내부에 /mnt/emulated/o/sdcard/혹은 어떤 다른 경로로 저장되는 것 같은데 이 부분에서 상당히 다양한 문제점이 제기되었습니다. 그러나 KIO4_Base64 extension에서는 /sdcard/라는 가상의 경로에 디코딩 된 jpeg파일을 저장했고, Look extension(인공지능 분류)에서도 동일한 형태의 경로를 활용한다는 특성을 파악하게 되어 가볍게 이 문제를 해결할 수 있었습니다.

3.2.1.1.4.2 디코딩 문제

처음에 저희는 Base64 라이브러리를 app inventor 블록의 형태로 포팅하는 것을 생각해 보았으나 다행스럽게도 KIO4_Base64 확장기능을 발견하여 해결할 수 있었습니다. 해당 확장에서는 StringToFile이라는 메소드가 있는데, 이를 활용하면 앱인벤터의 text타입(String)을 사용자가 원하는 형식의 파일로 저장해줍니다.

3.2.1.2 이미지 처리 이슈

3.2.1.2.1 DL분류모델을 생성에서 클래스 수의 문제

처음 프로젝트를 계획할 무렵, 저희는 App inventor에서 제공하는 Personal Image Classifier를 이용하고자 했습니다. 확률적인 측면에서 제공되는 데이터가 얼마나 그 이미지와 일치되는지에 대한 정보라고 판단했기 때문입니다. 그러나 이것은 각 class에 속할 확률에 해당한다는 사실을 뒤늦게 알게 되었습니다. 따라서 저희는 delivery가 아닌 것, 즉 not delivery 클래스에 해당되는 수 많은 클래스를 학습시킬 필요가 있었습니다. 이것은 너무나 큰 시간적 이슈를 불러일으키므로 기존에 MIT에서 공식 extension으로 제공 중이던 look extension에서 만들어진 모델을 사용하게 되었습니다. 다행스럽게도 Plastic bag class와 light bulb 클래스 사이의 확률을 적절히 조합하면 무언가 들어있는 비닐봉지로 분류할 수 있다는 사실을 발견했습니다.

3.2.1.2.2 스마트폰 기종에 따른 tensorflow 미지원 문제 (error code -1)

모든 안드로이드 스마트폰이 Look extension을 지원하지는 않는다는 점을 알게 되었습니다. 지원하지 않는다면 error 코드 -1이 발생하는데 이 부분에서 많은 시간을 소요하였으나 look extension이 지원중인 갤럭시 S8 공기계를 얻게 되어 해결했습니다.

3.2.1.2.3 image인식모드/video 인식모드 전환 문제 (error code -4)

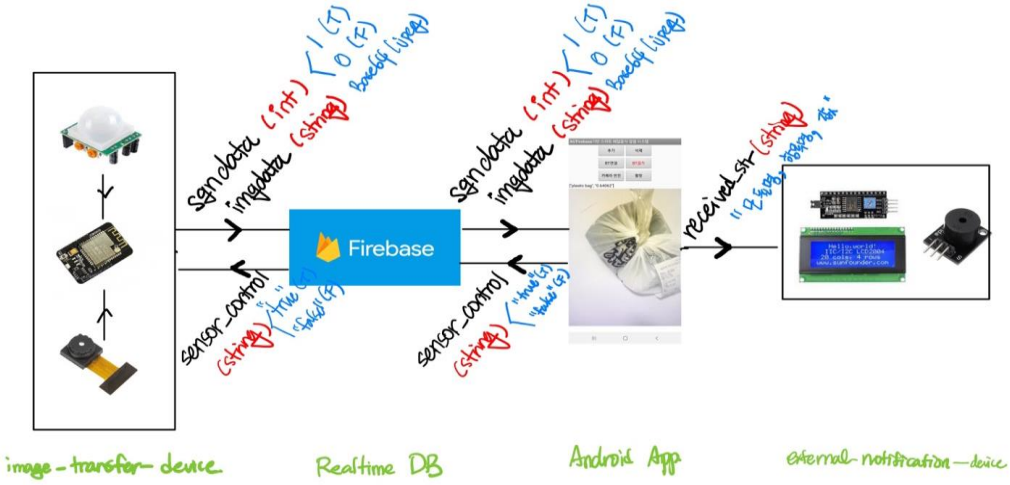
Video 이미지를 분류하는 경우와 Jpeg이미지를 분류하는경우의 매커니즘이 다른 것 같습니다. 따라서 초기에 Look 확장을 실행할 때의 디폴트 값인 video분류 모드를 image 분류 모드로 바꿔주지 않으면 jpeg 파일을 분류할 때 -4에러, 즉 video분류 불가능 에러가 발생하게 됩니다. 이런 점을 몰라서 초반에 많은 시간을 낭비하였지만, 모드 변경 기능을 알게 되어 해결할 수 있었습니다.

3.2.2 문제점 해결 과정 요약

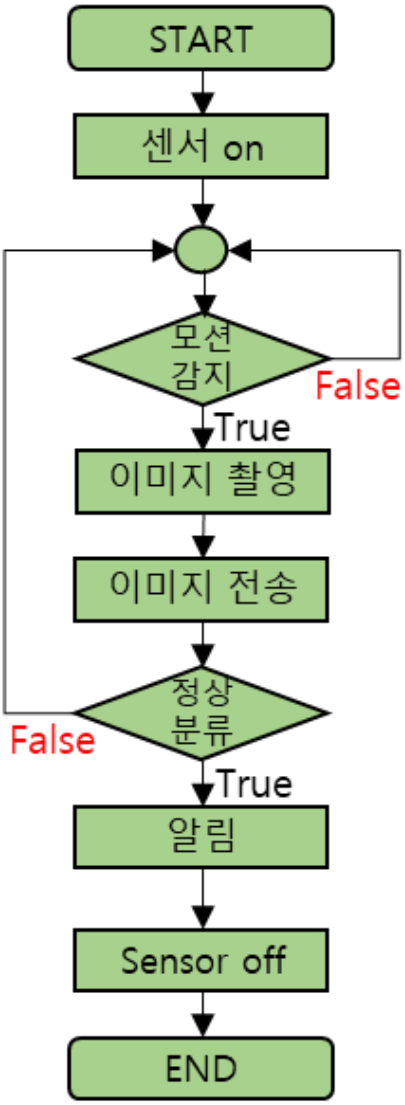
문제점	적용 해결 방법	해결 과정 및 결과
이미지 전송 API가 없음	Base64 스트링 변환 뒤 전송	구현 성공
앱인벤터와 Firebase의 불일치	String값에 '₩"'를 붙임	구현 성공
이미지 데이터 전송시 Wifi끊김	사이즈 축소 or 데이터 분할	사이즈 축소로 구현
파일경로 문제	가상 경로 사용	루트경로 대신가상경로로 구현
DL시에 -1에러와 -4에러	-1에러는 기종, -4에러는 설정	S8로 변경, image모드로 변경으로 구현 성공

4. 시스템 설계 및 상세설계

4.1 프로젝트 전체 구성도



4.2 기능별 내용별 논리 흐름



4.2.1 이미지 전송 디바이스

```
Loop:
Firebase.getString();
String sensor_control = FirebaseData.stringData();
IF sensor_control == "true"
        GET sgndata from motion sensor
        SEND sgndata TO Firebase
        IF sgndata == true
            GET esp_cam_fb image data
            TRANSFORM esp_cam_fb TO Base64 String imgdata
            SEND imgdata TO Firebase
        END
END
```

4.2.2 어플리케이션

```
GET arrv_time
SET sensor_control = "true"
SET is_finished = false
sendBTString("delivery, start, String(arrv_time)")
FOR !is_finished:
    FOR sgndata == 1:
    END
    GET imgdata
    String2File(imgdata, "/imgdata.jpg")
    getAIClassification("/imgdata.jpg")
    IF class == "delivery":
        is_finished = true;
        sendBTString("delivery, end, ")
    END
END
```

4.2.3 외부 알림 모듈

```
Loop:
   GET received_str // "모듈이름, 항목이름, 값"
   PARSE header_module, header_item, header_value
   SWITCH header module:
      CASE "delivery"
          "start", "end"가 있습니다. Start는 예상 시간에 맞게 출력합니다.
          "end"에 해당되면 해당 상태를 출력하고 부저를 울립니다.
      CASE "buzzer"
           "level"과 "on/off"가 있습니다. Level은 0~3으로 설정됩니다.
      CASE "standby"
          현재시간을 표시하는 대기상태입니다.
```

5. 프로젝트 결과 평가

◦ 목표 기술 달성도(2에서 설정한 세부 목표에 대한 기술 구현 정도 및 달성도 기입)

목표 기술	구현 기술	기술 달성도(%)[1]
Firebase를 통해 App으로 이미지 전송	Base64 스트링 변환 후 전송	100%
HD급 이미지 전송	VGA급 이미지 전송	75% (메모리 용량 부족)
Firebase의 이미지 데이터 읽기	앱인벤터에서 구현 완료	100%
jpeg파일을 DL모델로 분류	Look 확장을 통한 구현 완료	100%
Delivery class인 경우 배달완료	Plastic bag클래스로 대체	75% (비닐봉지 오탐지 有)
App을 통한 외부 알림부 제어	CSV String을 통한 무선제어	100%

[1] 100%: 원안과 동일 구현 혹은 대체 구현 완벽, 75% 다소 미흡한 구현, 0%~50%: 구현 실패