취약계층을 위한 스마트 공유 냉장고 시스템: 관리 자동화 및 접근 제어

김두현, 김관영, 홍상훈(전기전자공학부), 이소정(보건환경융합과학부)

[Abstract]

공유냉장고는 누구나 음식을 기부하고 필요한 사람이 가져갈 수 있는 시스템으로, 음식 낭비를 줄이고 취약계층에게 도움을 제공하는 사회적 목적을 가지고 있다. 그러나 기존 모델은 음식의 재고 관리와 냉장고 접근 통제에 있어 관리자의 부담이 크며, 이로 인해 설치 장소가 주로 공공기관에 한정되는 등의 문제점이 있다. 본 프로젝트는 이러한 문제를 해결하기 위해 음식 수령 대상을 취약계층으로 한정하고자 RFID 를 활용한 잠금장치 구성한다. 그리고 음식의 재고 관리 자동화를 위해 실시간 이미지 분석을 3 축 이동장치와 결합한 모델을 제안한다. 이를 통해 공유냉장고는 설치 장소의 제약을 줄이고, 기부자의 접근성을 극대화하며, 사회적 가치 창출에 기여할 것으로 기대된다.

1. 서론

공유냉장고는 누구나 음식물을 넣고 가져감으로써 취약계층들에게 무료로 음식을 제공하는 시스템이다. 이 시스템은 음식 낭비를 감소시키고 경제적 취약계층에게 직접적인 도움을 줄 수 있는 방법을 제공하여 먹거리 사각지대를 해소할 수 있다. 이와 같은 장점으로 인해 2018 년 1 월 수원시 권선구 고색동에 처음 설치되어 [1] 현재는 전국에서 3000 개 이상 사용되고 있다. 그러나 기존의 공유냉장고 모델은 음식이 꼭 필요한 취약계층이 아니더라도 공유 냉장고에 접근할 수 있다는 점과 음식의 재고를 파악해야 한다는 점 때문에 이를 관리해줄 사람이 필요하다. 따라서 주로 동사무소와 같은 공공기관에 설치되어 직원이 직접 재고를 관리해왔다. 하지만 위 방법은 관리자의 부담이 커 사업의 확장성에 제약을 가하는 단점이 있다. 관리의 부담이 크기 때문에 냉장고 설치 장소가 공공기관 등으로 제한되며, 이로 인해 음식 기부자들의 접근성이 저하될 수 있다.

본 프로젝트에서는 이러한 문제를 해결하기 위해 음식을 수령하는 대상을 취약계층으로 한정하고 음식의 재고 관리를 중앙화 및 부분 자동화함으로써 관리의 부담을 최소화하고 냉장고 설치 장소의 제약을 줄이는 방안을 제시한다.

음식 수령 대상을 한정하기 위해, 투입구와 수령구를 분리한다. 투입구에서는 누구나 음식을 기부할 수 있고 투입된 음식은 자동으로 빈 자리에 보관된다. 수령구는 신원이 확인된 취약계층만 음식에 접근할 수 있도록 잠금장치와 RFID 카드를 설치하여 보안을 강화한다. 또한

냉장고 내 음식의 재고를 실시간으로 파악하고 관리하기 위해 카메라를 사용한다. 음식의 입출고 시마다 카메라가 냉장고 내부의 사진을 촬영하고 이를 통해 재고의 변화 상태를 판단하여 추적한다. 이와 같은 개선된 공유냉장고 모델은 관리 부담을 최소화시켜 설치장벽을 낮추며, 기부자들의 접근성을 극대화시키는 등의 사회적 가치를 창출할 것으로 기대된다.

2. 본론

제안된 냉장고 모델은 다음과 같은 방식으로 작동한다.

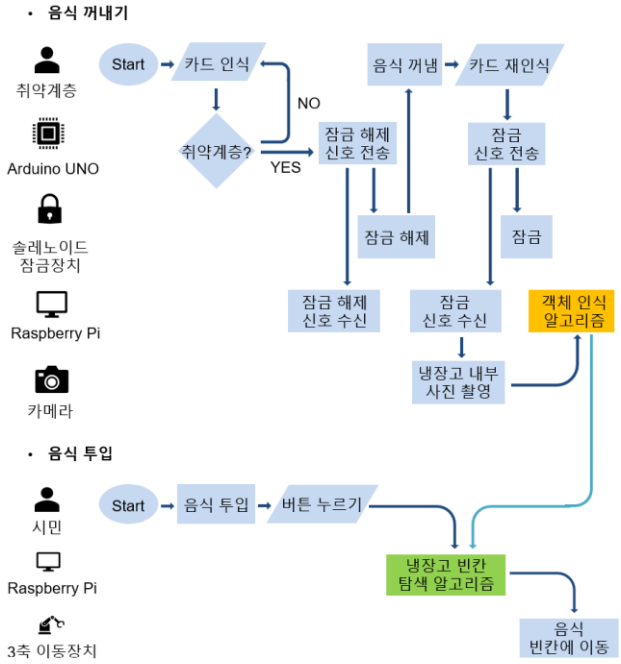


그림 1. 구동 과정

A. 냉장고 제작

1) 외형 틀은 나무 합판과 아크릴 판으로 제작하여, 2 층의 선반과 2 개의 문(투입구와, 배출구)을 구현한다.
2) 잠금부는 솔레노이드 잠금장치, RFID 리더기를 아두이노에 연결하여 구현한다. 이미지 촬영을 위해 ESP32 카메라를 냉장고 윗 문에 부착하여, 냉장고 내부 전체가 촬영될 수 있도록 한다.
3) 3 축장치 구성을 위해 3D CAD 와 3D 프린터로 부품을 출력하였다. 3 축 장치의 좌우 움직임을 위해 소형모터 회전부에 풀리와 타이밍벨트를 연결시켜 음식판이 3D 프린터로 만든 좌우레일을 따라 움직이며 음식을 이동한다. 상하운반을 위해서는 스태핑 모터에 부착된 3D 프린터로 만든 38cm 리드나사가 돌아가며 좌우움직임 장치가 위아래로 이동할 수 있다.

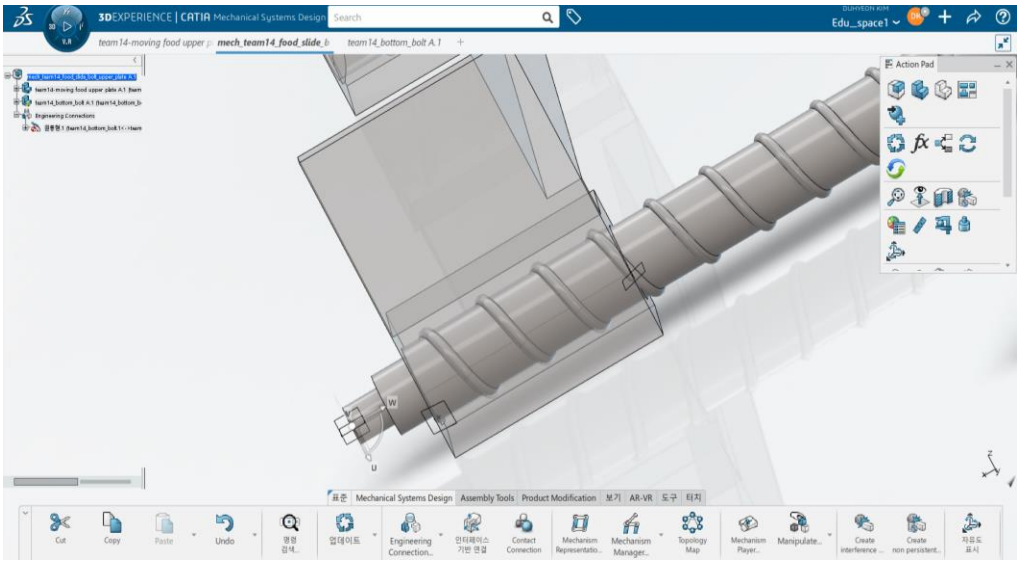


그림 2. 38cm 리드 나사 3D CAD 로 제작 중인 화면(by. 3Dexperience platform)

음식 판 3축 이동장치 정면 및 평면 사진

그림 33.차원 음식 이동장치

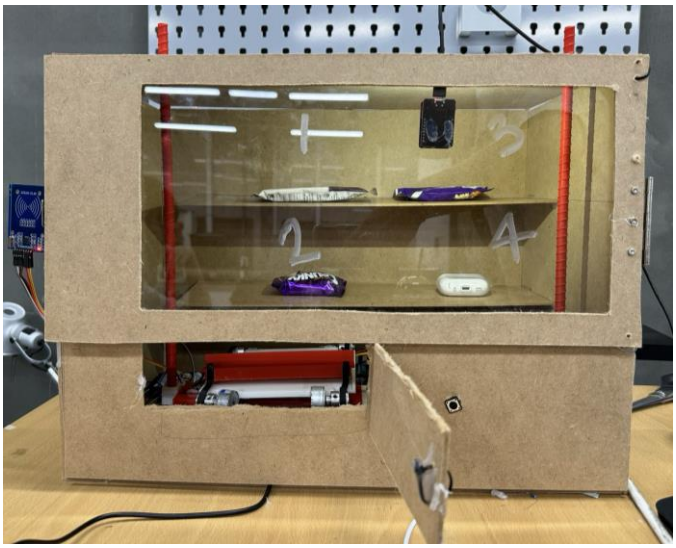


그림 44. 냉장고 완성 모델

B. 작동 과정(코드 설명)

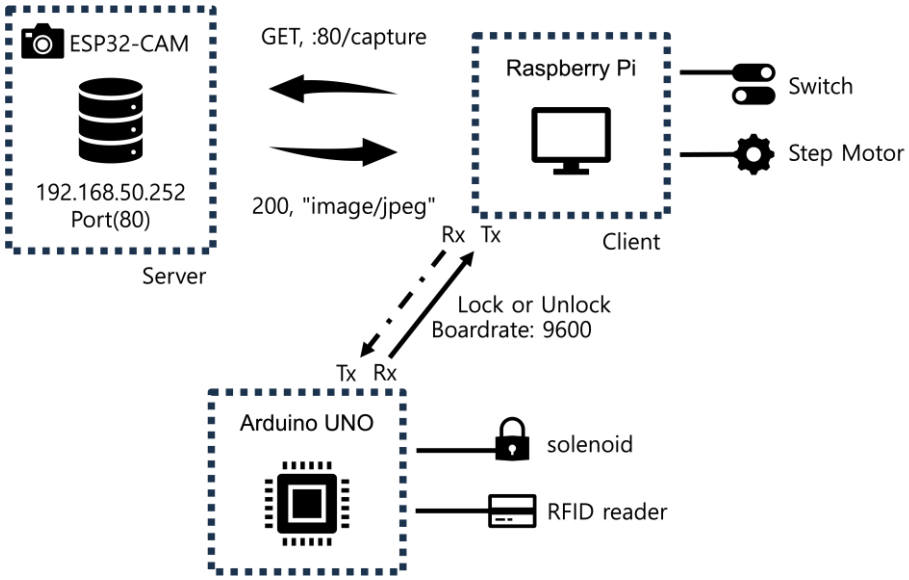


그림 55. 하드웨어 구성도

a. RFID+솔레노이드+사진 캡쳐 알고리즘(잠금장치, 카메라서버)

1) 하드웨어 연결

아두이노는 RFID 카드 리더기, 릴레이모듈, 솔레노이드, 라즈베리파이와 연결된다. Esp32 cam 은 아두이노로 코드를 미리 업로드한 후 전원을 공급한다.

2) Esp32 cam

Esp32 cam 은 서버를 개설하여 /capture 경로로 들어오는 요청을 기다리고, 해당 요청이 들어오면 handleCapture 함수를 실행하여 현재 화면을 캡처한 후 그 이미지를 클라이언트에게 전송한다.

3) 아두이노: RFID, 솔레노이드, 릴레이모듈

RFID 카드 리더기에 올바른 UID 를 가진 카드가 태그될 때마다 릴레이 모듈을 활성화/비활성화 하여 솔레노이드 잠금장치를 해제하거나 잠근다. (솔레노이드는 12V 전압을 사용하기 때문에 릴레이 모듈을 통해 제어됨.) 이와 동시에 라즈베리파이와 시리얼 통신을 통해 "U"(Unlock) 또는 "L"(Lock) 신호가 전송된다.

4) 라즈베리파이

"L" 신호를 수신하면, 라즈베리파이는 Esp32 cam 서버(192.168.50.252)에 /capture 경로로 GET 요청을 보내고, 응답으로 받은 이미지 파일을 저장한다.

Esp32 cam 코드

1. handleCapture 함수

1-1. 카메라에서 사진을 캡처하고 이미지 데이터를 저장한다.

```
void handleCapture() {
  clearBuffer();  // 오래된 버퍼를 비우고 시작
  camera_fb_t * fb = esp_camera_fb_get();
  Serial.println("capture_try");// 카메라에서 사진 캡처 시도
```

1-2. 캡처한 이미지 데이터를 클라이언트에게 전송한다.

```
Serial.println("request get");
server.setContentLength(fb->len);
server.send(200, "image/jpeg", "");
WiFiClient client = server.client();
client.write(fb->buf, fb->len);
esp_camera_fb_return(fb);
```

서버에서 클라이언트로부터 capture 경로로 요청이 오면 handleCapture 함수를 실행시켜 사진을 캡처하고 이미지 데이터를 클라이언트로 전송해준다.

2. Esp32 cam 주요 설정들

2-1. 포트 80 에 웹 서버 객체를 생성하였다.

```
WebServer server(80);
```

2-2. frame buffer 를 1 로 설정하고 이미지 화질을 VGA(640x480)로 설정하여 이미지의 딜레이를 예방하였다.

```
if (psramFound()) {
  config.frame_size = FRAMESIZE_VGA;
  config.jpeg_quality = 10;
  config.fb_count = 1;
} else {
  config.frame_size = FRAMESIZE_VGA;
  config.jpeg_quality = 12;
  config.fb_count = 1;
}
```

아두이노 코드

1. 취약계층 카드의 UID 를 등록한다.

```
byte knownUID[] = {0x60, 0x72, 0x88, 0x21}; // 올바른 카드의 UID
```

2. 리더기에 태그된 카드의 UID 를 인식한다.

```
rfid.PICC_ReadCardSerial()
```

3. 취약계층 UID 카드인 경우 (60 72 88 21)

3-1. 솔레노이드 잠금장치가 활성화되어 있으면,
릴레이를 비활성화하여 솔레노이드 잠금장치 잠금 및 serial 포트를 통해 라즈베리파이에 "L"신호를 전달한다.

3-2. 솔레노이드 잠금장치가 비활성화되어 있으면,
릴레이 활성화하여 솔레노이드 잠금장치 잠금 및 serial 포트를 통해
라즈베리파이에 "U"신호를 전달한다.

라즈베리파이 코드

아두이노에서부터 "L"신호가 오면 192.168.50.252 의 80 번 포트(ESP32-CAM)로 get request 를 보내고 응답으로 받은 이미지 파일을 저장한다.

```
if line == "L":
  print("Access granted. Door locked.")
  try:
      response = requests.get('http://192.168.50.252:80/capture')
      if response.status_code == 200:
          current_time = datetime.now().strftime("%Y%m%d_%H%M%S")
          image_filename = os.path.join(folder_path, f'image_{current_time}
          with open(image_filename, 'wb') as f:
              f.write(response.content)
          print(f"Image captured and saved as {image_filename}")
```

b. 빈칸 인식 알고리즘(Opencv, Computervision))

우리가 사용하는 하드웨어의 특성을 고려하여 빈칸 인식 알고리즘을 선정한다.
기존에 torch 기반의 다양한 object detection + localization 모델이 존재한다. 다만, 이 모델을 라즈베리파이에서 돌리기에는 라즈베리파이 연산능력의 한계가 존재한다.
메모리의 한계보다는 모델이 가지고 있는 높은 GFLOPS(Giga Floating Point Operations Per Second)의 문제이다.
또한 해당 모델은 모든 상황에 대한 높은 정확도를 가지는 모델이지만, 우리는 현재 고정된 카메라, 고정된 이미지에 대해서 이미지를 분석하는 것이기 때문에
ML 모델보다는 opencv 내장 라이브러리만을 이용한 고전 컴퓨터비전을 활용하였다.

알고리즘

1. 이미지 전처리 및 조정

1-1) 크롭

- 크롭(Cropping): 주어진 `crop_box` 좌표를 사용하여 냉장고 내부 음식 배치 가능 구역만을 추출한다.

1-2) 밝기 조절

- 밝기 계산: `ImageStat.Stat`를 사용하여 이미지의 평균 밝기를 계산한다.

- 조정: 타겟 밝기에 맞추어 이미지의 밝기를 조정한다. 이는 `lambda` 함수를 통해 각 픽셀 값에 비례적으로 조정을 가하며, 이미지 간의 밝기를 통일시켜 처리 효과를 균일화한다.

2. 백그라운드 서브트랙션 및 이미지 비교

2-1) 블러링 및 차이 계산

- 블러링(Blurring): `cv2.GaussianBlur`를 사용하여 이미지에 가우시안 블러를 적용함으로써, 노이즈를 줄이고 주요 객체의 경계를 부드럽게 처리한다.
- 차이 이미지(Difference Image): 블러 처리된 배경과 객체 이미지 간의 절대 차이를 계산한다.

2-2) 이진화 및 섹션 분할

- 이진화(Thresholding): 차이 이미지를 그레이스케일로 변환한 뒤, `cv2.threshold`을 사용하여 이진화를 수행한다.
- 영역 분할(Segmentation): 이진화된 이미지를 네 개의 섹션으로 나누어 각 섹션에서의 변화 정도를 계산한다. 이는 냉장고 내부의 칸과 1 대 1 대응한다.

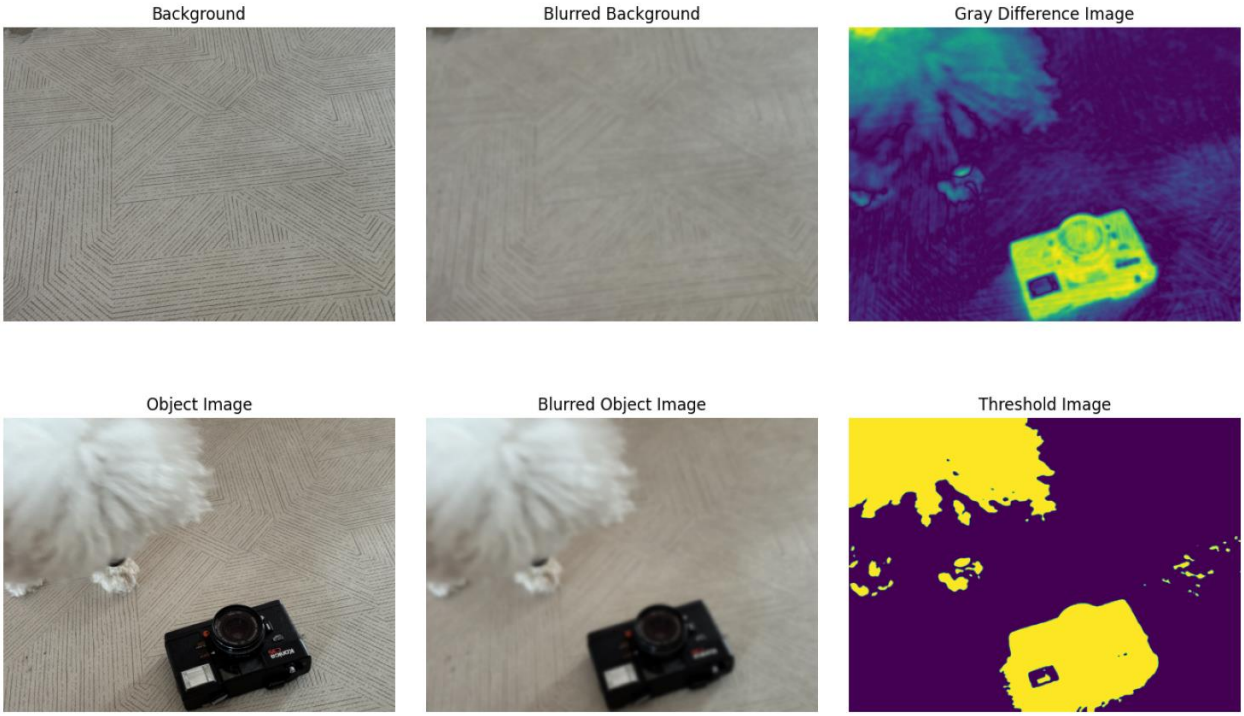


그림 66. 백그라운드 서브트랙션 과정

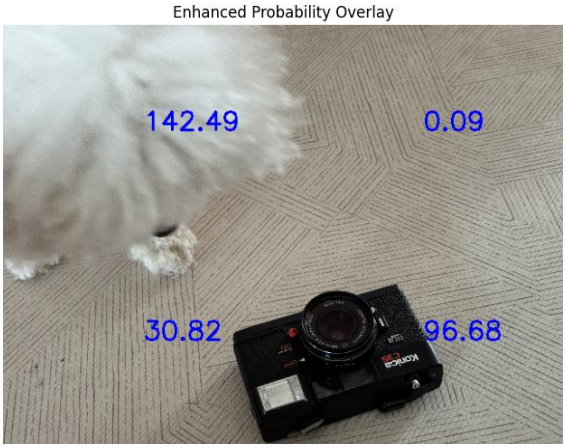


그림 77. 결과값

c. 모터 이동 설계(병력적 드라이버 출력 활용, 지정된 좌표까지 이동)

우리는 b. 빈칸 인식 알고리즘(Opencv, Computervision))을 활용해서, 2x2 냉장고 내부의 빈칸을 찾을 수 있도록 하였다.

해당 결과는 main함수에서 monitoring_thread를 통해서 얻어지고, q= queue.LifoQueue()에 put된다. 우리는 3가지 thread가 동시에 라즈베리파이에서 동작하기 때문에, LIFO queue를 활용해서 빈칸정보를 주고 받는다. 빈칸정보는 아래와 같은 형식으로 queue에 저장되어 있다.

```
New image detected: ./refridge_images/image_20240809_070352.jpg
Image captured and saved as refridge_images/image_20240809_070352.jpg
Probabilities: [[np.float64(0.0), np.float64(29.0029296875), np.float64(35.773046875), np.float64(0.0)]]
```

Probabilities : [1 번칸, 2 번칸, 3 번칸, 4 번칸] -> 숫자가 높을 수록 full

사용자의 음식이 투입되고 호출되면 motor.to_blank()가 호출된고, motor 는 동작을 시작한다. main.py 에서 선언한 queue 의 값을 get 해서 냉장고 내부의 상황을 파악하고, 아래 step map 을 바탕으로 모터를 동작시킨다. 이때, step motor 별로 step 간 다른 delay 를 사용한다.

Step Map steps = (x축, y축, z축)

(15, 4753, 63)	(90, 4753, 63)
(15, 1740, 63)	(90, 1740, 63)

그림 88. Step map

d. main.py 구성

공유냉장고 자동화를 위해서는 총 3 가지의 병렬적인 프로그램이 라즈베리파이에서 작동해야된다.

1) 냉장고 내부 사진 수신여부 모니터링을 시작하는 스레드 (monitoring_thread)

2) 아두이노와 시리얼 통신을 시작하는 스레드 (serial_thread)

3) 음식 투입 및 모터 구동을 하는 스레드 (food_thread)

또한 해당 스레드 간의 데이터 공유를 위해 LifoQueue 를 사용한다. 특히 monitoring_thread 에서 냉장고 내부 상황을 분석한 probabilities 의 값을 queue 통해서 food_thread 에서 모터의 동작에서 활용 가능하다.

코드

Pin setup

```
8   def setup_gpio():
9       GPIO.setmode(GPIO.BCM)  # 전역적으로 한 번만 설정
10
11      # 핀 번호 설정
12      DIR_td = 20   # 방향 핀 (Top-Down)
13      STEP_td = 16  # 스텝 핀 (Top-Down)
14      DIR_lr = 3    # 방향 핀 (Left-Right)
15      STEP_lr = 2   # 스텝 핀 (Left-Right)
16      DIR_foodplate = 27   # 방향 핀
17      STEP_foodplate = 17  # 스텝 핀
18
19      # GPIO 핀 설정
20      GPIO.setup(DIR_td, GPIO.OUT)
21      GPIO.setup(STEP_td, GPIO.OUT)
22      GPIO.setup(DIR_lr, GPIO.OUT)
23      GPIO.setup(STEP_lr, GPIO.OUT)
24
25      GPIO.setup(DIR_foodplate, GPIO.OUT)
26      GPIO.setup(STEP_foodplate, GPIO.OUT)
```

Thread 설정

```python
def main():
    setup_gpio()

    # Queue 생성
    q = queue.LifoQueue()

    # image_in_checker 모듈에서 디렉토리 모니터링을 시작하는 스레드 설정
    image_directory = "./refridge_images/"
    monitoring_thread = threading.Thread(target=new_image_checker.start_monitoring, args=(image_directory, q))

    # receive_message 모듈에서 시리얼 통신을 시작하는 스레드 설정
    serial_thread = threading.Thread(target=lock_check.receive_message)

    # food_in_checker 모듈에서 음식 감지를 시작하는 스레드 설정
    food_thread = threading.Thread(target=food_in_checker.food_detected, args=(q,))

    # 스레드 시작
    monitoring_thread.start()
    serial_thread.start()
    food_thread.start()

    try:
        monitoring_thread.join()
        serial_thread.join()
        food_thread.join()
    except KeyboardInterrupt:
        print("Program terminated by user.")
    finally:
        GPIO.cleanup()  # Cleanup GPIO only once here
    # 메인 스레드에서 스레드가 종료될 때까지 기다림

if __name__ == "__main__":
    main()
```

e. 오류 작업(스위치 디바운싱, 카메라 capture delay)

1) GPIO SWITCH_PIN 입력 튀는 현상

우리의 의도와 달리는 rfid 를 인식하고, 아두이노가 라즈베리파이로 serial 통신을 하는 과정에서 food_detected 코드가 실행되는 오류가 발생하였다. 본래 food_detected 는 사용자가 24 번 스위치를 눌렀을 때만, 작동하여 음식의 투입을 인지하고 모터를 움직이는 함수를 호출하는 역할이다. 하지만 rfid 를 tag 하는 순간에 gpio 의 불안정한 input 으로 모터가 움직이는 문제가 발생하였다.
이를 해결하기 위해 debouncing logic 을 추가하였다. 스위치에서의 **debouncing** 은 스위치의 접촉 불안정성으로 인해 발생하는 노이즈나 오작동을 제거하는 기술 또는 과정을 의미한다. 사용한 debouncing logic 은 아래 코드와 같다.

```python
def food_detected(q):
    setup()
    last_pressed = 0
    debounce_time = 0.1  # 디바운스 시간 조정

    while True:
        try:
            input_state = GPIO.input(SWITCH_PIN)
            if input_state == GPIO.HIGH and (time.time() - last_pressed) > debounce_time:
                last_pressed = time.time()  # 마지막 트리거 시간 업데이트
                time.sleep(0.1)  # 짧은 지연 시간 추가
                if GPIO.input(SWITCH_PIN) == GPIO.HIGH:  # 재확인
                    if not q.empty():
                        print("Food detected! Activating motor...")
                        motor.to_blank(q)
                        time.sleep(5)  # 재트리거 방지
                    else:
                        continue
        except KeyboardInterrupt:
            print("Interrupted by the user")
            break
        except Empty:
            print("Queue is empty, skipping...")
            time.sleep(1)  # 큐가 비어 있는 경우, 다시 확인하기 전에 잠시 대기
```

SWITCH_PIN 의 input GPIO.HIGH 를 한 번만 인식하는 것이 아니라, 0.1 delay 후에도 GPIO input 을 체크함으로써 해당 HIGH signal 이 전기적인 노이즈인지 물리적인(사람) 스위치에 의한 작동인지를 구분할 수 있도록 하였다.
이를 통해 하드웨어 적인 bouncing 을 소프트웨어적으로 debouncing 할 수 있었다.

2) Capture delay 발생 문제

본래 취약계층이 음식을 수령 후 ID 카드를 찍고 문을 잠근다면, 해당 순간의 냉장고 내부 사진이 라즈베리파이로 전송되어야 한다. 하지만 몇 번의 시뮬레이션이 진행됨에 따라 해당 전송 속도가 현저히 느려졌다. 대략 40 초 정도의 지연이 발생한 이미지가 전송되었다. 이는 실시간으로 냉장고 내부 상황을 분석하고, 음식을 투입하는 과정에 상당히 오류를 범할 수 있다. 이를 해결하기 위해서 ESP32-CAM 에 camera bufferclear 함수를 구현하였다.

```
30  void clearBuffer() {
31      camera_fb_t *fb = NULL;
32      while ((fb = esp_camera_fb_get()) != NULL) {
33          esp_camera_fb_return(fb);
34      }
35  }
```

동작 원리

- camera_fb_t *fb 포인터를 사용하여 프레임 버퍼를 저장한다.
- while 루프를 통해 esp_camera_fb_get() 함수가 NULL 이 아닐 때까지 반복 실행된다.

- 각 반복에서 esp_camera_fb_get()는 사용 가능한 프레임 버퍼를 반환하며, esp_camera_fb_return()을 호출하여 프레임 버퍼를 반환하고 메모리를 해제한다.

C. 결과 분석(시연 연상 혹은 사진)

유효한 RFID 카드를 리더기에 인식했을 때 솔레노이드 잠금장치가 잘 작동하는지 확인한다.

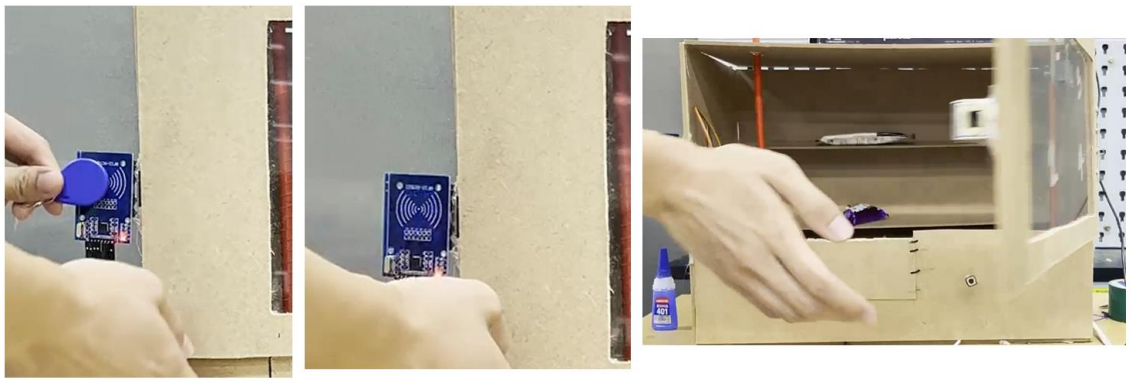


그림 9**9.** RFID 인식 후 솔레노이드 잠금장치 잠금해제

잠금장치의 작동을 확인한 후 빈칸 분석 알고리즘이 잘 실행되는지 확인하기 위해 3 가지 경우로 나누어 실험을 하였다.

1) 아무 음식을 투입하지 않은 상태에서 음식 투입

음식이 아직 진열되지 않은 상황에서 빈칸 인식 알고리즘에서는 기본적으로 슬롯 1, 2, 3, 4 순으로 빈공간을 채운다. 음식을 두 차례 투입하였을 때 음식들이 차례대로 슬롯 1, 슬롯 2 로 이동하는 것을 확인할 수 있다.

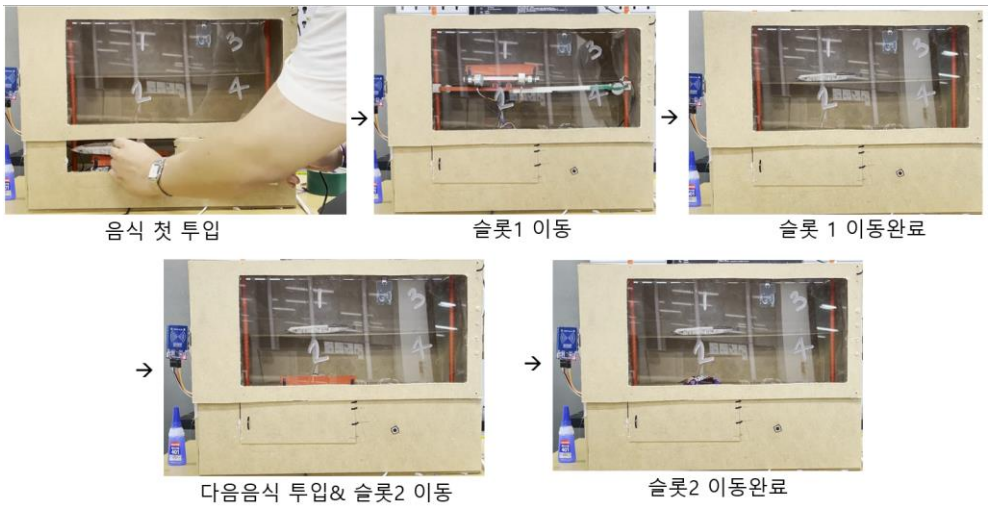


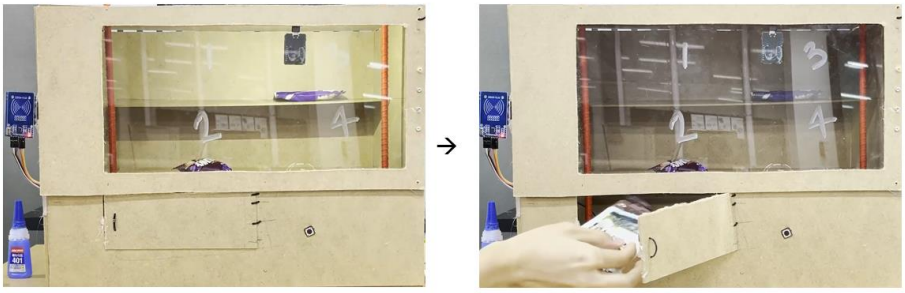
그림 1010. (1) 실험 과정

```
(siwon_venv) team14@raspberrypi:~/Documents/final_siwon $ python main.py
Connected to Arduino on /dev/serial0
Received message: U
Access granted. Door unlocked.
Received message: L
Access granted. Door locked.
New image detected: ./refridge_images/image_20240809_070130.jpg
Image captured and saved as refridge_images/image_20240809_070130.jpg
Probabilities: [[np.float64(0.0), np.float64(0.0), np.float64(0.0), np.float64(0.0)]]
Food detected! Activating motor...
Moving to slot 1
:) Moving finished, returning to original position
Returned to original position
Food detected! Activating motor...
Moving to slot 2
:) Moving finished, returning to original position
Returned to original position
Received message: U
Access granted. Door unlocked.
Received message: L
Access granted. Door locked.
New image detected: ./refridge_images/image_20240809_070352.jpg
Image captured and saved as refridge_images/image_20240809_070352.jpg
Probabilities: [[np.float64(0.0), np.float64(29.0029296875), np.float64(35.773046875), np.float64(0.0)]]
```

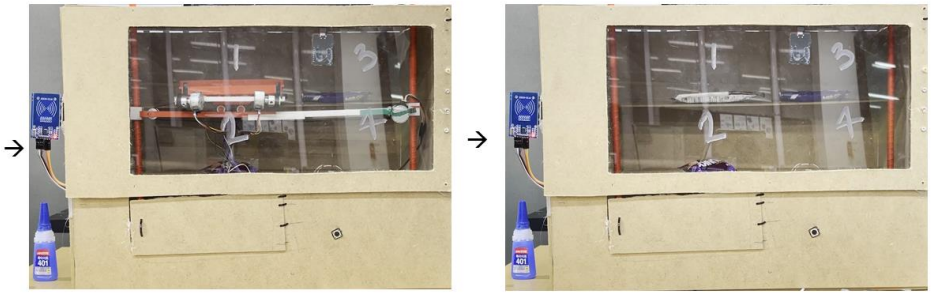
그림 1111. (1) 실험 로그

2) 음식이 어느 정도 진열된 상태에서 음식 투입

슬롯 2, 3 이 찬 상태에서 빈칸 찾기 알고리즘은 음식을 투입할 때 슬롯 1, 4 순서대로 투입되야 한다. 먼저 음식을 한 번 투입하였을 때 빈 공간인 슬롯 1 로 음식을 이동시킨 것을 확인했다.



슬롯 2, 3 음식 있는 상태에서 음식 투입



빈 공간인 슬롯 1로 음식 운반 완료

그림 1212. (2) 실험 과정

일단 슬롯 1 에 음식이 채워졌을 때 빈칸 찾기 알고리즘에서 슬롯 1 의 객체 인식 결과값이 0 에서 45 로 올라간 것을 확인할 수 있다. 따라서 다음 음식을 투입할 때에는 슬롯 4 로 음식을 보내는 것을 코드 하단부의 'Moving to slot 4 로 확인할 수 있다.

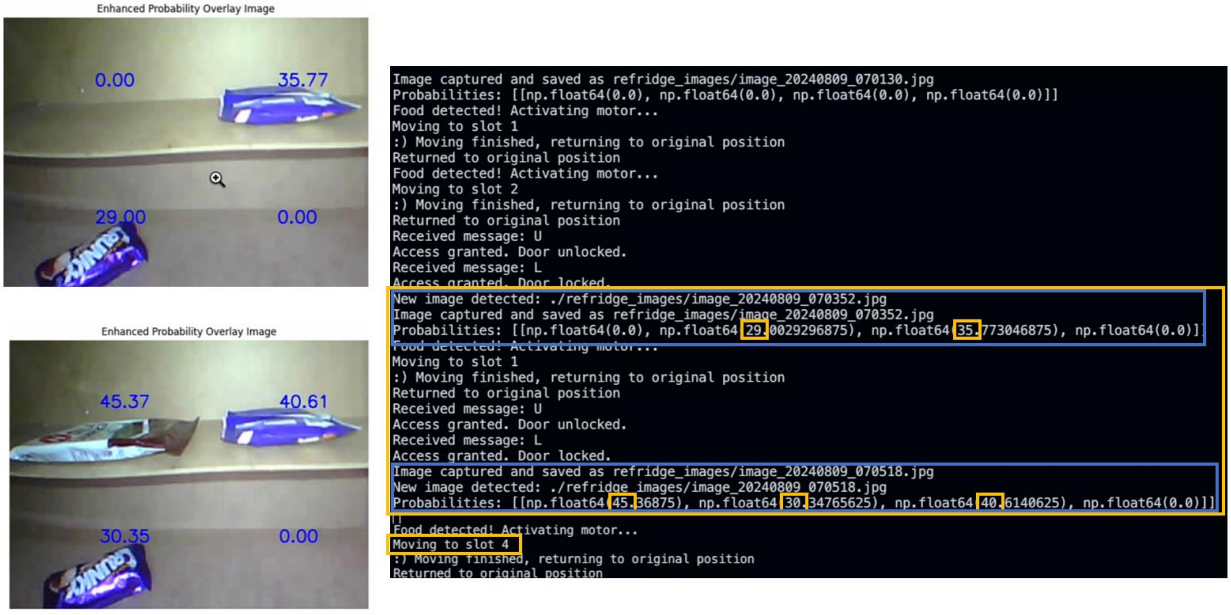
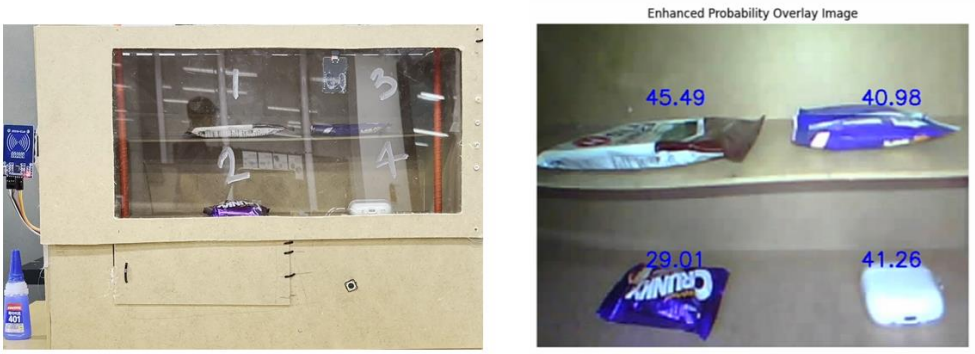


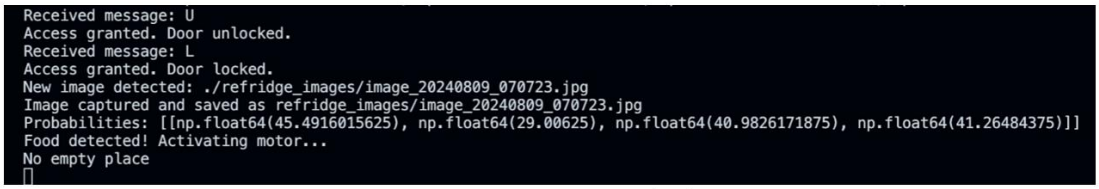
그림 1313. (2) 객체인식 결과값 사진들 및 실험 로그

3) 음식이 꽉찬 상태에서 음식 투입

슬롯 1, 2, 3, 4 가 가득 찬 상태에서 음식 투입 후 빈칸 찾기 알고리즘을 실행했을 때, 효과적으로 네 슬롯의 객체 인식 결과값을 인식하고, 'No empty place' 이후 음식 운반을 하지 않으면서 음식을 다시 회수할 수 있다.



슬롯 가득 찬 상태



No empty 뜨고, 음식 이동 X

그림 1414. (3) 실험 과정 및 로그

3. 결론

본 프로젝트에서는 특정 대상이 냉장고에 접근 가능하고, 냉장고 음식 재고 관리를 자동화할 수 있는 시스템을 설계하였다. RFID 를 이용한 사용자 식별 시스템을 통해 특정 대상이 냉장고 음식에 접근할 수 있고, Opencv 와 ESP32-CAM 을 활용하여 임베디드 환경에서 실시간을 처리할 수 있는 빈칸 탐색 알고리즘을 고안하였다. 또한 3D 프린터를 활용하여 3 축 이동장치를 설계하였다. 해당 3 가지 기술들이 유기적으로 작동되는 것을 확인하였으며, 공유 냉장고의 접근남용을 방지하고 관리의 효율성을 증가시킬 수 있을 것으로 기대된다. 이는 더불어 공유 냉장고 설치장벽을 낮출 수 있을 것으로 생각된다.

향후 이 시스템의 성능을 더욱 향상시키기 위해서, 공유 냉장고에서 수집된 데이터를 중앙 데이터베이스로 통합하여 분석하는 관리 중앙화를 도입할 수 있다. 이렇게 함으로써 재고 변화를 실시간으로 파악하여 취약계층에게 필요한 정보를 제공할 수 있다. 또한, 이러한 중앙 관리 시스템이 구축된다면 추가적으로 사회적 가치가 있는 공공사업으로 발전할 수 있을 것이다.

4. 참조문헌

[1] 이정하, "누구나 가져고가 채워놓고" …호응 뜨거운 수원 '공유냉장고', 한겨레, 2021.12.19, https://www.hani.co.kr/arti/area/capital/1023880.html

5. Appendix

영상 시연 자료

코드 전문

https://github.com/dudududukim/siwon_Hands