Weekly diary 8주차(2024.05.13 ~ 2024.05.19)

팀 푸바오

팀장 임베디드시스템공학과 201901752 서정인

팀원 임베디드시스템공학과 201701726 권오찬

팀원 임베디드시스템공학과 201901747 류제현

팀원 임베디드시스템공학과 202001697 박성빈

[작업 기록]

- **서정인**

  2024.05.13 13:48 ~ 14:48 (1시간)

  2024.05.13 19:07 ~ 21:07 (2시간)

  2024.05.15 13:57 ~ 18:15 (4시간 18분)

  2024.05.17 16:40 ~ 20:07 (3시간 27분)

  2024.05.18 11:03 ~ 15:40 (4시간 37분)

  2024.05.19 15:25 ~ 19:54 (4시간 29분)

  총합 시간 20시간 51분

- **권오찬**

  2024.05.13 13:48 ~ 16:00 (2시간 12분)

  2024.05.13 17:50 ~ 21:00 (3시간 10분)

  2024.05.15 11:40 ~ 18:14 (6시간 34분)

  2024.05.16 12:30 ~ 15:59 (3시간 29분)

  2024 05.17 17:23 ~ 20:10 (2시간 47분)

  2024.05.18 11:11 ~ 15:40 (4시간 29분)

2024.05.19 15:25 ~ 19:57 (4시간 32분)

총합 시간 27시간 13분

- **류제현**

  2024.05.14 11:45 ~ 15:52 (4시간 7분)

  2024.05.15 15:15 ~ 18:15 (3시간 1분)

  2024.05.17 15:46 ~ 21:32 (5시간 46분)

  2024.05.19 15:45 ~ 19:57 (4시간 12분)

  총합 시간 17시간 4분

- **박성빈**

  2024.05.13 16:48 ~ 21:03 (4시간 15분)

  2024.05.14 13:26 ~ 14:31 (1시간 5분)

  2024.05.15 11:38 ~ 15:58 (4시간 20분)

  2024.05.16 12:07 ~ 15:51 (3시간 44분)

  2024.05.18 11:05 ~ 15:38 (4시간 33분)

  2024.05.19 15:14 ~ 19:56 (4시간 42분)

  총합 시간 22시간 39분

작업 내용

2024.05.13

- 압력감지 주석 처리 돼있는 것들 코드화
- 라즈베리파이 보드 전체 초기화 및 OS, 패키지 재설치

2024.05.14

- 나무판에 대고 캘리브레이션 - 나무판을 조금만 움직여도 무게변화 → 외관 작업 진행 후 추가 보정작업 진행 예정

2024.05.15

- 외관작업을 진행하려 했으나 흔들림이 심하여 개선방법 고안
- 두께에 맞는 추가 부품 구매하여 보완 예정
- 블루투스 모듈도 고장으로 작업 불가능

- 나무 합판을 사용해 제작한 임시 거치대에 카메라 고정 및 촬영
- 멀티 스레딩을 사용한 코드로 두 카메라 촬영 화면 동시 송출

2024.05.16

- 중간점검 자료 제작
- 아두이노와 라즈베리파이 간 통신 공부

2024.05.17

- 중간점검 제출용 영상 제작
- 발판 외관 보충
- 라즈베리파이 카메라 캘리브레이션 준비

2024.05.18

- 발판 외관 보충
- 블루투스 모듈 연결 확인 및 압력감지 코드 수정
- 라즈베리파이 블루투스 모듈 연결 코드화
- 라즈베리파이 카메라 캘리브레이션 진행
- 스테레오 비전 캘리브레이션 진행

2024.05.19

- 미디어파이프, 텐서플로우 라즈베리파이 보드에 설치 시도
- AI Hub 데이터셋 검색 및 분석
- 제출 기한 임박 마무리 작업 일정 회의
- 카메라 외관 회의

압력 감지

2024.05.13

압력감지 알고리즘 중 코드가 떠오르지 않아 주석처리 했던 부분들중 일부를 코드화 후 나머지 부분들에 대하여 공부를 진행하였다.

발판 압력감지가 시작될 때나 종료될 UI와 라즈베리파이가 통신 후 라즈베리파이가 받은 값을 아두이노에서 받아 변수로 사용하고 싶은데 그 부분에 대해서는 추가 공부가 필요하다.

2024.05.14

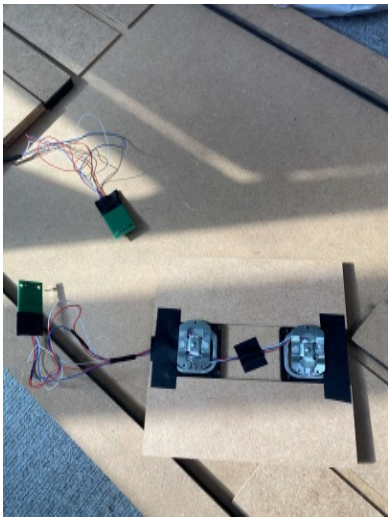
초반부에 캘리브레이션을 1kg 아령으로 진행해두었으나 프로젝트 진행 도중 센서가 눌린 정도가 달라서인지 갑자기 값이 튀는 현상이 발생하였다. 그래서 우리가 계획중인 외관과 최대한 유사한 상태로 5kg 원판으로 캘리브레이션을 다시 진행하였다. 센서들에 따라 수치가 이전과 조금 달라졌으나 바닥 나무판의 배치에 따라 무게가 달라지는 현상이 발생하여 외관을 빠르게 완성한 뒤 그 조건에 맞는 캘리브레이션 수치를 찾아야 할 거 같다.

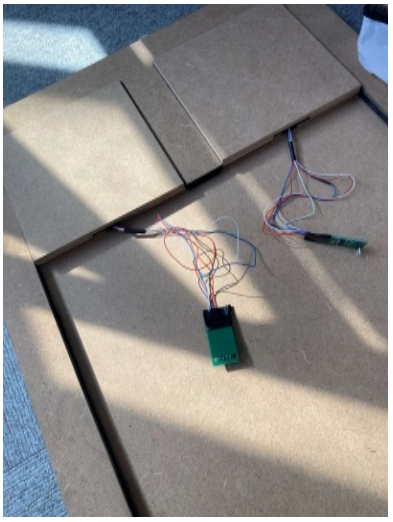
2024.05.15

외관을 생각한대로 구성해보았으나 생각보다 위의 합판이 뜨는 공간이 많이 발생하여 흔들림이 많이 발생하였다. 얇은 나무 합판을 추가로 구매하여 뜨는 공간을 최대한 메꿔서 흔들림을 최소화 할 예정이다.

2024.05.18

로드셀 두께와 위치에 맞춰 발판 하부와 외관 프레임을 제작하여 안정성을 높였다. 사용자가 발판에 올라설 수 있는 사이즈의 합판을 재단하여 발판 상부를 완성한 후에 로드셀을 고정시킬 계획이다.





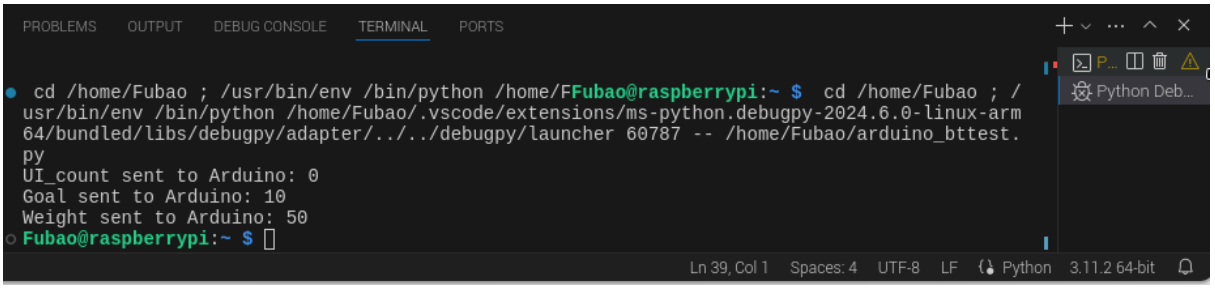
라즈베리파이와 블루투스 모듈 연결

라즈베리파이와 블루투스 모듈을 연결하기 위해 라즈베리파이 터미널에서 해야 하는 설정들을 공부 한 후 따로 정리해두었다.

아래 코드는 라즈베리파이와 블루투스 모듈 간의 통신을 코드화 해둔 부분이다 UI와 라즈베리파이를 연결하는 방법은 공부해 나가며 개발할 예정이다.

```
import serial
import time

# 시리얼 통신 설정 (블루투스 포트와 통신 속도에 맞게 설정)
ser = serial.Serial('/dev/rfcomm0', 9600)  # 블루투스 포트와 맞게

def send_UI_count_to_arduino(UI_count):
    ser.write(bytes(str(UI_count) + '\n', 'utf-8'))  # count
    print("UI_count sent to Arduino:", UI_count)

def send_goal_to_arduino(goal):
    ser.write(bytes(str(goal) + '\n', 'utf-8'))  # goal 값을 바
    print("Goal sent to Arduino:", goal)

def send_weight_to_arduino(weight):
    ser.write(bytes(str(weight) + '\n', 'utf-8'))  # weight 값
    print("Weight sent to Arduino:", weight)

def send_realtime_count_to_arduino(realtime_count):
    ser.write(bytes(str(realtime_count) + '\n', 'utf-8'))  #
    print("Realtime_count sent to Arduino:", realtime_count)

# 값 설정 -> UI와 연결 필요 UI에서 사용자가 선택한 값을 초기에 받아야 한다
UI_count = 0 # UI에서 목표 횟수로 입력 받은 횟수
goal = 10
weight= 50


# 값을 아두이노로 전송
send_UI_count_to_arduino(UI_count)
time.sleep(1)  # UI_count 값을 전송한 후 잠시 대기

send_goal_to_arduino(goal)
```

```
time.sleep(1)  # goal 값을 전송한 후 잠시 대기

send_weight_to_arduino(weight)
time.sleep(1)  # weight 값을 전송한 후 잠시 대기



while(True):
    realtime_count = 0 # 카메라에서 실시간으로 세고 있는 운동의 횟수
    send_realtime_count_to_arduino(realtime_count)
    if(realtime_count==UI_count):
        break # 계속 운동 횟수를 카운트 하다가 목표 횟수에 도달하면 종료

# 시리얼 통신 종료
ser.close()
```



라즈베리파이에서 아두이노로 송신

아두이노에서 수신한 화면

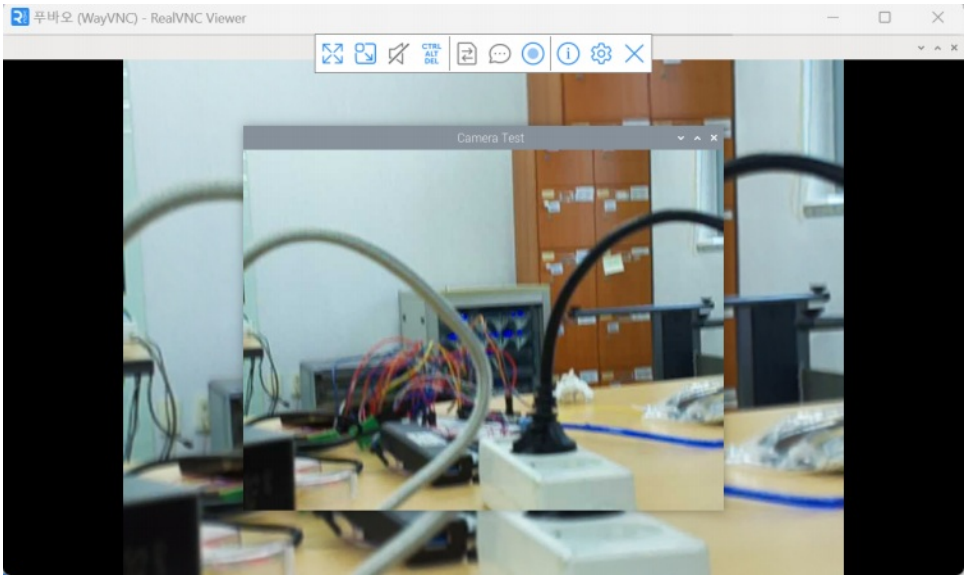
블루투스 연결 후 라즈베리파이에서 목표한 세가지 숫자를 송신하고 아두이노 보드에서 수신하여 출력이 되는 것을 확인하였다.

모션 감지

지속된 openCV 패키지 설치 및 카메라 인식 오류로 인해 보드 전체 초기화 및 전체 패키지 재설치를 진행하였다. raspberry Imager에 캐시되어있는 OS 파일까지 삭제한 뒤 처음부터 설치를 진행하였고 libcamera를 사용한 코드로 실시간 카메라 화면을 출력하는 코드를 작성하였다.

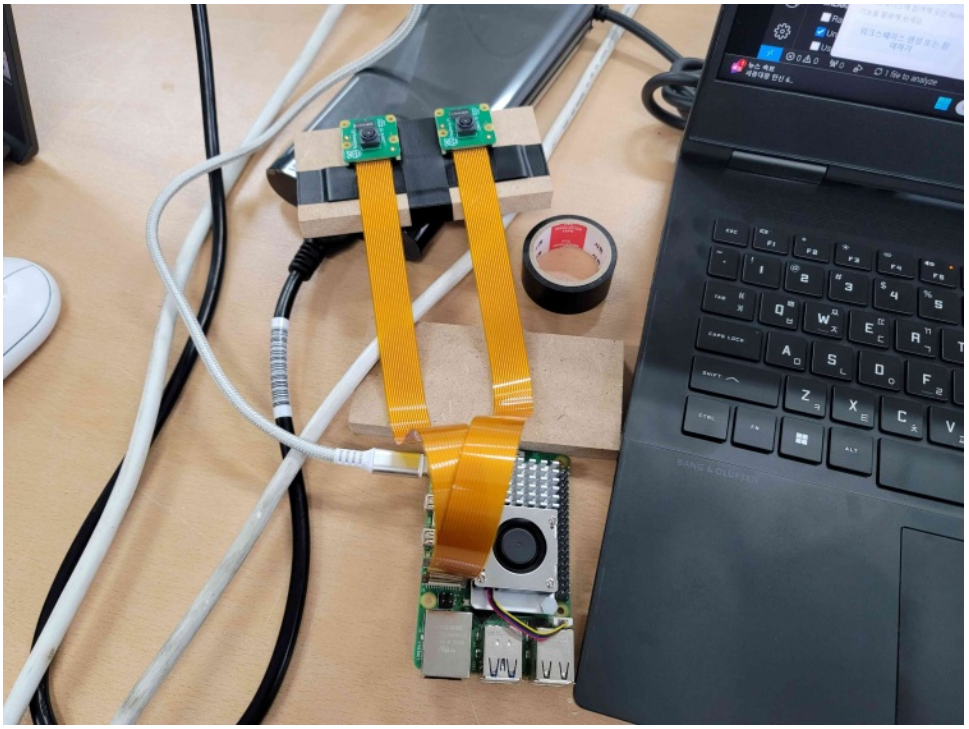
```
import cv2
import numpy as np
import subprocess

# libcamera-vid를 사용하여 MJPEG 스트림을 생성
command = ['libcamera-vid', '--codec', 'mjpeg', '--width', '6
process = subprocess.Popen(command, stdout=subprocess.PIPE, b

# MJPEG 스트림을 읽어와서 OpenCV를 통해 화면에 표시
if process.stdout is not None:
    # 비디오 스트림을 읽기 위한 파일 객체 생성
    bytes_buffer = bytes()
    try:
        while True:
            # 스트림에서 데이터 읽기
            bytes_buffer += process.stdout.read(1024)
            a = bytes_buffer.find(b'\xff\xd8')  # JPEG 시작
            b = bytes_buffer.find(b'\xff\xd9')  # JPEG 끝

            if a != -1 and b != -1:
                jpg = bytes_buffer[a:b+2]  # JPEG 이미지 추출
                bytes_buffer = bytes_buffer[b+2:]  # 버퍼에서 사

                # JPEG 이미지 디코딩
                frame = cv2.imdecode(np.frombuffer(jpg, dtype=

                # 화면에 이미지 표시
                cv2.imshow('Camera Test', frame)

                # 'q' 키를 누르면 종료
                if cv2.waitKey(1) & 0xFF == ord('q'):
                    break
    finally:
```

```
        # 자원 해제
        process.stdout.close()
        process.terminate()
        process.wait()
        cv2.destroyAllWindows()
```



카메라 한 대가 촬영한 화면을 모니터에 스트리밍

카메라 두 대를 사용하여 스테레오 비전을 구현하는 것이기 때문에 카메라 각도 조정과 고정을 용이하게 하기 위해서 라즈베리파이 카메라 거치대를 주문하였으며, 수령 전까지 임시로 사용할 수 있도록 나무 합판을 사용해 카메라를 고정시켰다.

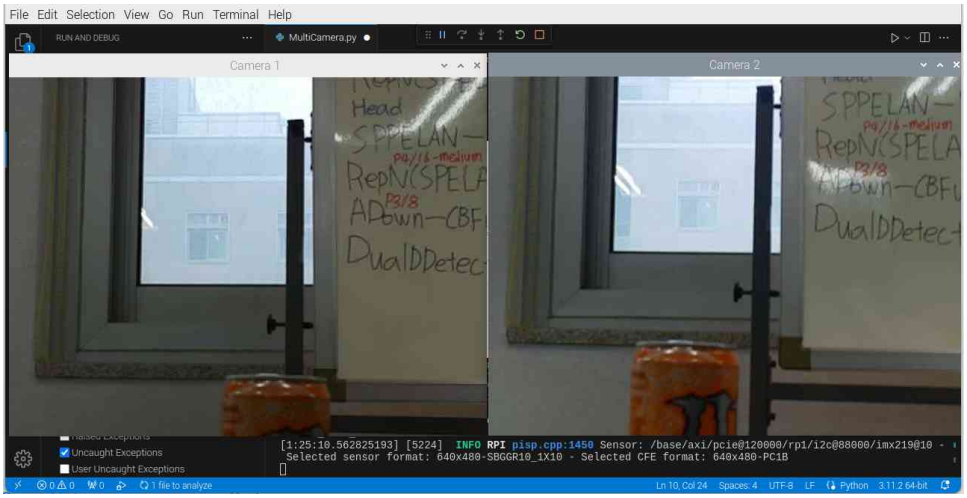


카메라 두 대가 모두 정상적으로 촬영이 이루어지고, 촬영한 화면을 동시에 모니터에 출력하여 확인할 수 있도록 코드를 작성하였다. 단일 프로세스에서는 카메라 두 대의 촬영 화면을 모두 띄울 수 없기 때문에 멀티 스레드를 활용하여 화면이 동시에 출력되는 것처럼 보이게 할 수 있었다. 초기화 및 재설치 이전에도 멀티 스레딩을 활용하여 화면 출력을 시도하였지만 카메라를 인식하지 못하는 문제가 지속적으로 발생했었는데, 이번 초기화와 재설치 이후엔 카메라 두 대를 모두 정상적으로 인식하는 것을 확인할 수 있었다.

```
import cv2
import numpy as np
import subprocess
import threading

def camera_stream(camera_index, window_name):
    command = [
    'libcamera-vid',
    '--camera', str(camera_index),
    '--codec', 'mjpeg',
    '--width', '640',
    '--height', '480',
    '-t', '0',
    '-o', '-',
```

```
    '--exposure', 'normal',
    '--awb', 'auto'
]
    process = subprocess.Popen(command, stdout=subprocess.P
IPE, bufsize=10**8)

    bytes_buffer = bytes()
    try:
        while True:
            bytes_buffer += process.stdout.read(1024)
            a = bytes_buffer.find(b'\xff\xd8')
            b = bytes_buffer.find(b'\xff\xd9')
            if a != -1 and b != -1:
                jpg = bytes_buffer[a:b+2]
                bytes_buffer = bytes_buffer[b+2:]
                frame = cv2.imdecode(np.frombuffer(jpg, dty
pe=np.uint8), cv2.IMREAD_COLOR)

                if frame is not None:
                    cv2.imshow(window_name, frame)

                    if cv2.waitKey(1) & 0xFF == ord('q'):
                        break
    finally:
        process.stdout.close()
        process.terminate()
        process.wait()
        cv2.destroyWindow(window_name)

threads = []
for index in range(2):
    window_name = f"Camera {index+1}"
    thread = threading.Thread(target=camera_stream, args=(i
ndex, window_name))
    thread.start()
    threads.append(thread)
```

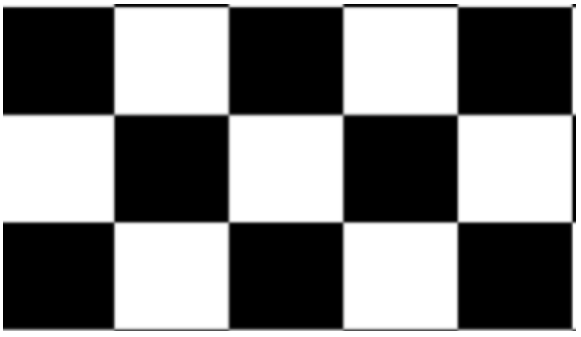
```
for thread in threads:
    thread.join()
```



임시로 두 카메라를 고정한 것이기 때문에 각도가 정밀하게 맞지 않는 부분은 있지만 비슷한 위치를 촬영하고 있는 모습을 확인할 수 있다.

Stereo Camera Calibration과 Stereo Camera Rectification의 절차

(현재 라즈베리파이5에서 Camera V2 모노카메라 2대를 사용하여 스테레오비전을 구현하려고함)



체커보드(Checker Board)

체스 게임의 게임판 보드 패턴과 유사하게 검정색과 흰색 사각형이 번갈아가며 교차된 형식의 격자 무늬 패턴. 카메라 캘리브레이션과 같은 컴퓨터 비전 작업에서 많이 사용된다.

카메라 캘리브레이션에서 체커보드가 많이 쓰이는 이유

제작 비용이 저렴하고 데이터셋 생성에 용이하며 체커보드 패턴의 규칙성과 기하학적 특성이 3D 공간에서 2D 이미지로 변환하는 과정의 왜곡을 분석하는데 유리하기 때문

Stereo Calibration

Stereo Calibration은 두 카메라를 통해 3차원 정보를 얻을 때 두 카메라의 내적 및 외적 파라미터를 추정하여 기하학적 관계를 파악하는 과정

이를 통해 카메라의 왜곡을 보정하고, 두 카메라 간의 정확한 관계를 파악할 수 있다

`내적 파라미터란?`

각 카메라의 내부 파라미터로, 렌즈의 초점 거리(focal length), 주점(principal point), 렌즈 왜곡(distortion coefficients)등

`외적 파라미터란?`

두 카메라 간의 상대적인 위치와 방향을 나타내는 파라미터로 회전 행렬(rotation matrix)과 변환 벡터(translation vector)등

`카메라 2대를 사용하기 때문에 일단 개별적으로 캘리브레이션을 해준 뒤에 각 카메라에 대한 내부 매개변수를 구하고 스테레오 캘리브레이션으로 두 카메라 사이의 상대적 위치와 방향을 알아낸다`

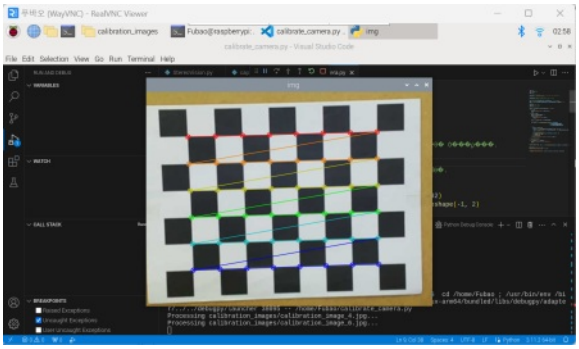
```
import cv2
import numpy as np
import subprocess
import os

# 캡처한 이미지를 저장할 디렉토리
capture_dir = 'calibration_images'
if not os.path.exists(capture_dir):
    os.makedirs(capture_dir)

def capture_images_from_camera(camera_index, window_name):
    # libcamera-vid 명령어 설정
    command = [
        'libcamera-vid',
        '--camera', str(camera_index),  # 카메라 인덱스 설정
        '--codec', 'mjpeg',             # 코덱 설정
        '--width', '640',               # 프레임 너비 설정
        '--height', '480',              # 프레임 높이 설정
        '-t', '0',                      # 무제한 시간 설정
        '-o', '-',                      # 표준 출력으로 스트림 설정
        '--exposure', 'normal',         # 노출 설정
        '--awb', 'auto'                 # 자동 화이트 밸런스 설정
    ]
    process = subprocess.Popen(command, stdout=subprocess.PIPE
```

```
    bytes_buffer = bytes()
    image_counter = 0
    try:
        while True:
            bytes_buffer += process.stdout.read(1024)  # 스트림
            a = bytes_buffer.find(b'\xff\xd8')  # JPEG 시작 지점
            b = bytes_buffer.find(b'\xff\xd9')  # JPEG 종료 지점
            if a != -1 and b != -1:
                jpg = bytes_buffer[a:b+2]  # JPEG 이미지 추출
                bytes_buffer = bytes_buffer[b+2:]  # 버퍼 업데이
                frame = cv2.imdecode(np.frombuffer(jpg, dtype=

                if frame is not None:
                    cv2.imshow(window_name, frame)  # 프레임을

                    if cv2.waitKey(1) & 0xFF == ord('c'):  #
                        image_counter += 1
                        image_path = os.path.join(capture_dir
                        cv2.imwrite(image_path, frame)  # 프레
                        print(f"Captured {image_path}")

                    if cv2.waitKey(1) & 0xFF == ord('q'):  #
                        break
    finally:
        process.stdout.close()  # 프로세스의 표준 출력을 닫음
        process.terminate()     # 프로세스 종료
        process.wait()          # 프로세스 종료 대기
        cv2.destroyWindow(window_name)  # 창 닫기

# 0번 카메라에서 이미지 캡처
capture_images_from_camera(0, 'Camera 0')
# 1번 카메라에서 이미지 캡처 (주석 처리됨)
# capture_images_from_camera(1, 'Camera 1')
```

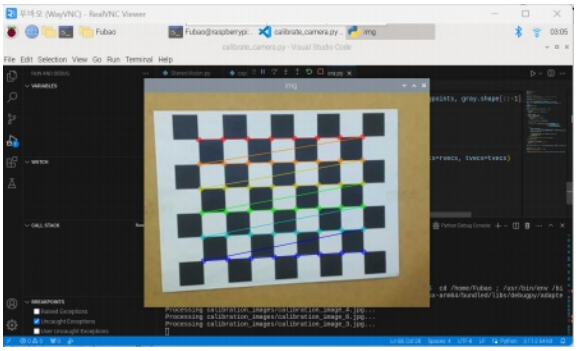
```
import cv2
import numpy as np
import glob
```

```
import os

capture_dir = 'calibration_images'  # 캡처된 이미지가 저장된 디렉토

CHECKERBOARD = (8, 6)  # 체커보드 패턴의 크기 (가로, 세로)

# 체커보드의 3D 좌표 준비
objp = np.zeros((CHECKERBOARD[0] * CHECKERBOARD[1], 3), np.fl
objp[:, :2] = np.mgrid[0:CHECKERBOARD[1], 0:CHECKERBOARD[0]].

objpoints = []  # 3D 공간에서의 점들
imgpoints = []  # 이미지 평면에서의 점들

# 캡처 디렉토리에서 모든 jpg 이미지 파일 로드
images = glob.glob(os.path.join(capture_dir, '*.jpg'))

if not images:
    print(f"No images found in {capture_dir}.")
    exit()  # 이미지가 없으면 종료

for fname in images:
    print(f"Processing {fname}...")
    img = cv2.imread(fname)
    if img is None:
        print(f"Failed to load {fname}.")
        continue  # 이미지를 로드할 수 없으면 다음 파일로 넘어감

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # 이미지를 그

    # 체커보드 코너 찾기
    ret, corners = cv2.findChessboardCorners(gray, CHECKERBOA
                                             cv2.CALIB_CB_ADA
                                             + cv2.CALIB_CB_F
                                             + cv2.CALIB_CB_N

    if ret:
        objpoints.append(objp)  # 3D 점 추가
        # 더 정확한 코너 위치 찾기
```

```
            corners2 = cv2.cornerSubPix(gray, corners, (11, 11),
                                        (cv2.TERM_CRITERIA_EPS +
            imgpoints.append(corners2)  # 2D 점 추가

            # 이미지에 체커보드 코너 그리기
            cv2.drawChessboardCorners(img, CHECKERBOARD, corners2
            cv2.imshow('img', img)  # 이미지를 창에 표시
            cv2.waitKey(500)  # 0.5초 대기
        else:
            print(f"Checkerboard corners not found in {fname}.")

    cv2.destroyAllWindows()  # 모든 OpenCV 창 닫기

    if not objpoints or not imgpoints:
        print("Not enough points for calibration.")
        exit()  # 충분한 점이 없으면 종료

    # 카메라 캘리브레이션 수행
    ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints,

    # 결과 출력
    print("Camera matrix: \n", mtx)
    print("Distortion coefficients: \n", dist)

    # 캘리브레이션 데이터를 파일에 저장
    # np.savez('camera0_calibration_data.npz', mtx=mtx, dist=dist
    np.savez('camera1_calibration_data.npz', mtx=mtx, dist=dist,
```



카메라0 캘리브레이션 화면



카메라1 캘리브레이션 화면

camera_calibration_data.npz에 저장되는 캘리브레이션 결과값

```
import cv2
import numpy as np
import subprocess
import threading
import os

# 이미지 저장 디렉토리 설정
left_dir = 'left'
right_dir = 'right'

# 디렉토리 생성
if not os.path.exists(left_dir):
    os.makedirs(left_dir)
if not os.path.exists(right_dir):
    os.makedirs(right_dir)

# 이미지 카운터 초기화
left_image_counter = 0
right_image_counter = 0

# 캡처 플래그 설정
capture_flag_left = threading.Event()
capture_flag_right = threading.Event()

def camera_stream(camera_index, window_name, output_dir, coun
    command = [
        'libcamera-vid',
        '--camera', str(camera_index),
        '--codec', 'mjpeg',
```

```
        '--width', '640',
        '--height', '480',
        '-t', '0',
        '-o', '-',
        '--exposure', 'normal',
        '--awb', 'auto'
    ]

    process = subprocess.Popen(command, stdout=subprocess.PIP

    bytes_buffer = bytes()
    try:
        while True:
            bytes_buffer += process.stdout.read(1024)
            a = bytes_buffer.find(b'\xff\xd8')  # JPEG 시작
            b = bytes_buffer.find(b'\xff\xd9')  # JPEG 끝
            if a != -1 and b != -1:
                jpg = bytes_buffer[a:b+2]
                bytes_buffer = bytes_buffer[b+2:]
                frame = cv2.imdecode(np.frombuffer(jpg, dtype

                if frame is not None:
                    cv2.imshow(window_name, frame)

                    if capture_flag.is_set():  # 캡처 플래그가 설
                        counter[0] += 1
                        image_path = os.path.join(output_dir,
                        cv2.imwrite(image_path, frame)
                        print(f'Captured {image_path}')
                        capture_flag.clear()  # 플래그 초기화

                    if cv2.waitKey(1) & 0xFF == ord('q'):  #
                        break
    finally:
        process.stdout.close()
        process.terminate()
        process.wait()
        cv2.destroyWindow(window_name)
```

```
# 이미지 카운터 리스트 (mutable type 사용)
left_counter = [left_image_counter]
right_counter = [right_image_counter]

# 카메라 스트림 스레드 시작
thread_left = threading.Thread(target=camera_stream, args=(0,
thread_right = threading.Thread(target=camera_stream, args=(1

thread_left.start()
thread_right.start()

while True:
    key = cv2.waitKey(1) & 0xFF
    if key == ord('c'):  # 'c' 키로 왼쪽 카메라 이미지 캡처
        capture_flag_left.set()
    elif key == ord('v'):  # 'v' 키로 오른쪽 카메라 이미지 캡처
        capture_flag_right.set()
    elif key == ord('q'):  # 'q' 키로 종료
        break

thread_left.join()
thread_right.join()
```

```
import cv2
import numpy as np
import glob

# 카메라 캘리브레이션 데이터 로드
with np.load('camera0_calibration_data.npz') as X:
    mtx0, dist0 = [X[i] for i in ('mtx', 'dist')]
with np.load('camera1_calibration_data.npz') as X:
    mtx1, dist1 = [X[i] for i in ('mtx', 'dist')]

# 체커보드 패턴 크기
CHECKERBOARD = (8, 6)

# 3D 점 배열 생성
```

```
objp = np.zeros((CHECKERBOARD[0] * CHECKERBOARD[1], 3), np.fl
objp[:, :2] = np.mgrid[0:CHECKERBOARD[1], 0:CHECKERBOARD[0]].

objpoints = []  # 3D 점
imgpoints0 = []  # camera0의 2D 점
imgpoints1 = []  # camera1의 2D 점

# 체커보드 이미지 파일 경로
images_left = sorted(glob.glob('left/*.jpg'))
images_right = sorted(glob.glob('right/*.jpg'))

# 이미지 처리
for img_left, img_right in zip(images_left, images_right):
    imgL = cv2.imread(img_left)
    imgR = cv2.imread(img_right)
    grayL = cv2.cvtColor(imgL, cv2.COLOR_BGR2GRAY)
    grayR = cv2.cvtColor(imgR, cv2.COLOR_BGR2GRAY)

    retL, cornersL = cv2.findChessboardCorners(grayL, CHECKER
    retR, cornersR = cv2.findChessboardCorners(grayR, CHECKER

    if retL and retR:
        objpoints.append(objp)
        corners2L = cv2.cornerSubPix(grayL, cornersL, (11, 11
                                     (cv2.TERM_CRITERIA_EPS +
        corners2R = cv2.cornerSubPix(grayR, cornersR, (11, 11
                                     (cv2.TERM_CRITERIA_EPS +
        imgpoints0.append(corners2L)
        imgpoints1.append(corners2R)

# 스테레오 카메라 캘리브레이션 수행
ret, mtx0, dist0, mtx1, dist1, R, T, E, F = cv2.stereoCalibra
    objpoints, imgpoints0, imgpoints1, mtx0, dist0, mtx1, dis
    criteria=(cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_I
    flags=cv2.CALIB_FIX_INTRINSIC
)

# 결과 저장
```

```
np.savez('stereo_calibration_data.npz', mtx0=mtx0, dist0=dist

print("Stereo calibration done.")
print("R: ", R)
print("T: ", T)
```

각 카메라를 개별 캘리브레이션 한 값을 이용해 스테레오 캘리브레이션을 진행하였다.

캘리브레이션이 완료되어 캘리브레이션 값이 stereo_calibration_data.npz파일에 저장되고 회전 행렬(R)과 번역 벡터(T)가 출력되는 것을 확인하였지만 깊이 맵이 정상적으로 출력되지 않는 문제가 발생하여 이미지 크기 등의 파라미터를 조율하며 재시도해볼 예정이다.

향후 계획

1. 카메라 - 라즈베리파이 보드 고정할 외관 디바이스 제작
2. UI 디자인 개선
3. 압력감지 HW 제작 마무리
4. 5.29 실사용 테스트(헬스 트레이너 섭외)