공압식 제어

양준석, 이우일

4/22(시험기간) -> 4/15 미팅 목표 (17:00)

1. 카메라 하단 촬영 구성으로 테스트해보기
2. 장갑 폼팩터 개선 -> 카메라 인식률 개선을 위해 손바닥을 보여줘야 하기 때문
3. 기본 연결하는 작업
    - 메카트로닉스 시스템
        - **비례제어 솔레노이드 -> 디지털** https://www.emerson.com/en-us/catalog/automation-solutions/industrial-factory-automation/proportional-valves-sv/asco-202-preci?fetchFacets=true#facet:&partsFacet:&modelsFacet:&facetLimit:&searchTerm:&partsSearchTerm:&modelsSearchTerm:&productBeginIndex:0&partsBeginIndex:0&modelsBeginIndex:0&orderBy:&partsOrderBy:&modelsOrderBy:&pageView:&minPrice:&maxPrice:&pageSize:&facetRange:& Vendor : Emerson
        - 유량센서 (keyence)

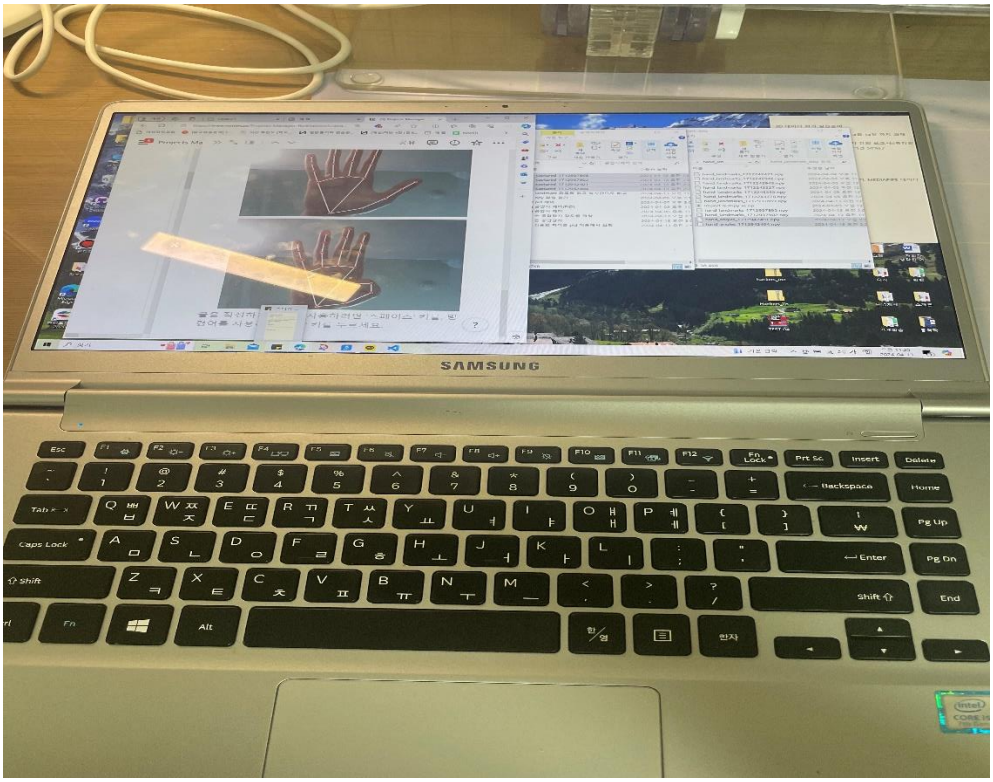
IDEA :

엄지손가락을 접은 상태 캡쳐해서 벡터의 각도를 구하거나 좌표를 쓴다.

그리고 손을 편상태일때의 벡터의 각도나 좌표를 구해서

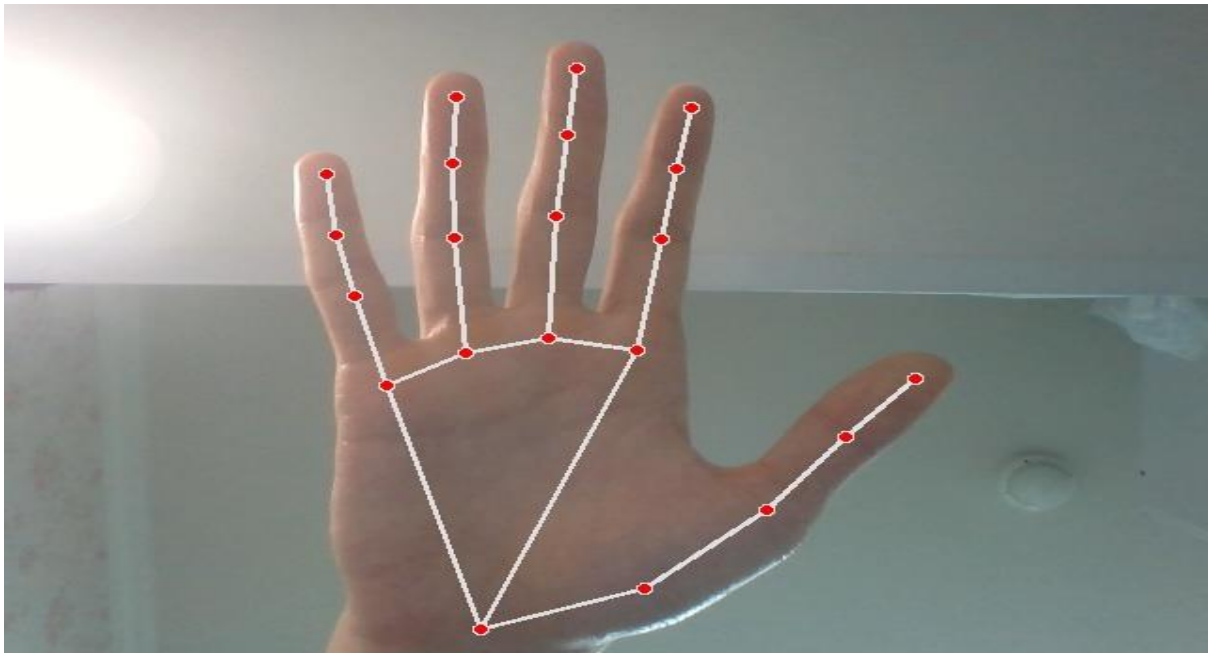
서로의 차이를 구한후 pid 를 이용해서정의한다.

if 문을 이용해서 정확도가 95 정도 될경우에 correct 라 하고 멈춘다.

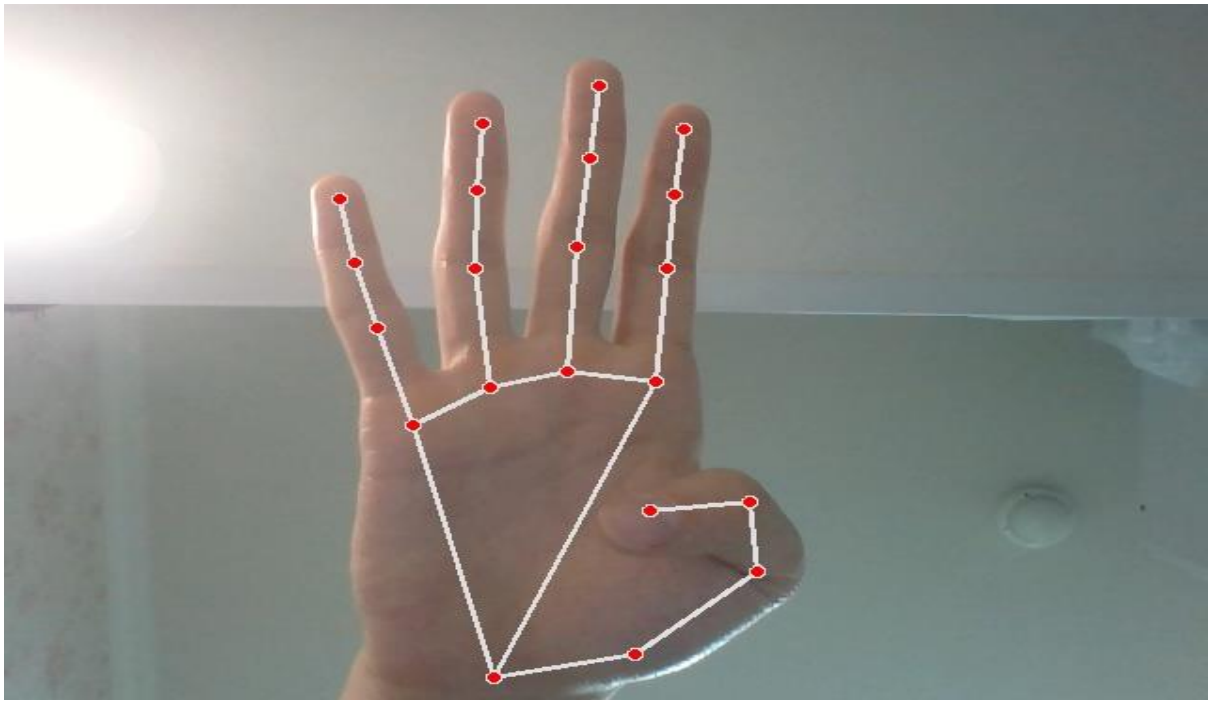


카메라 하단 촬영(mediapipe가 손바닥을 먼저 인식 후 나머지 부위를 인식하기에)

노트북을 180도 만든 후 촬영을 시작함.

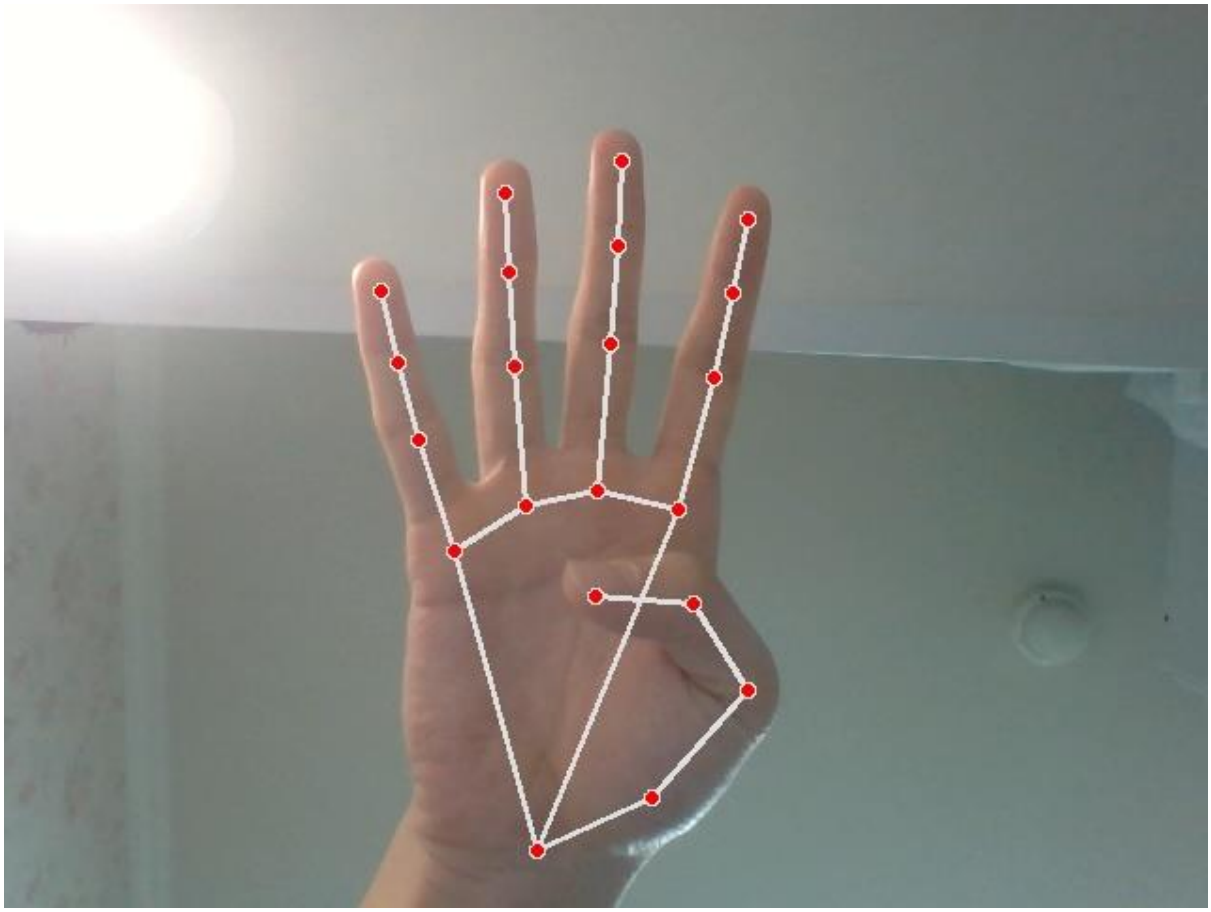


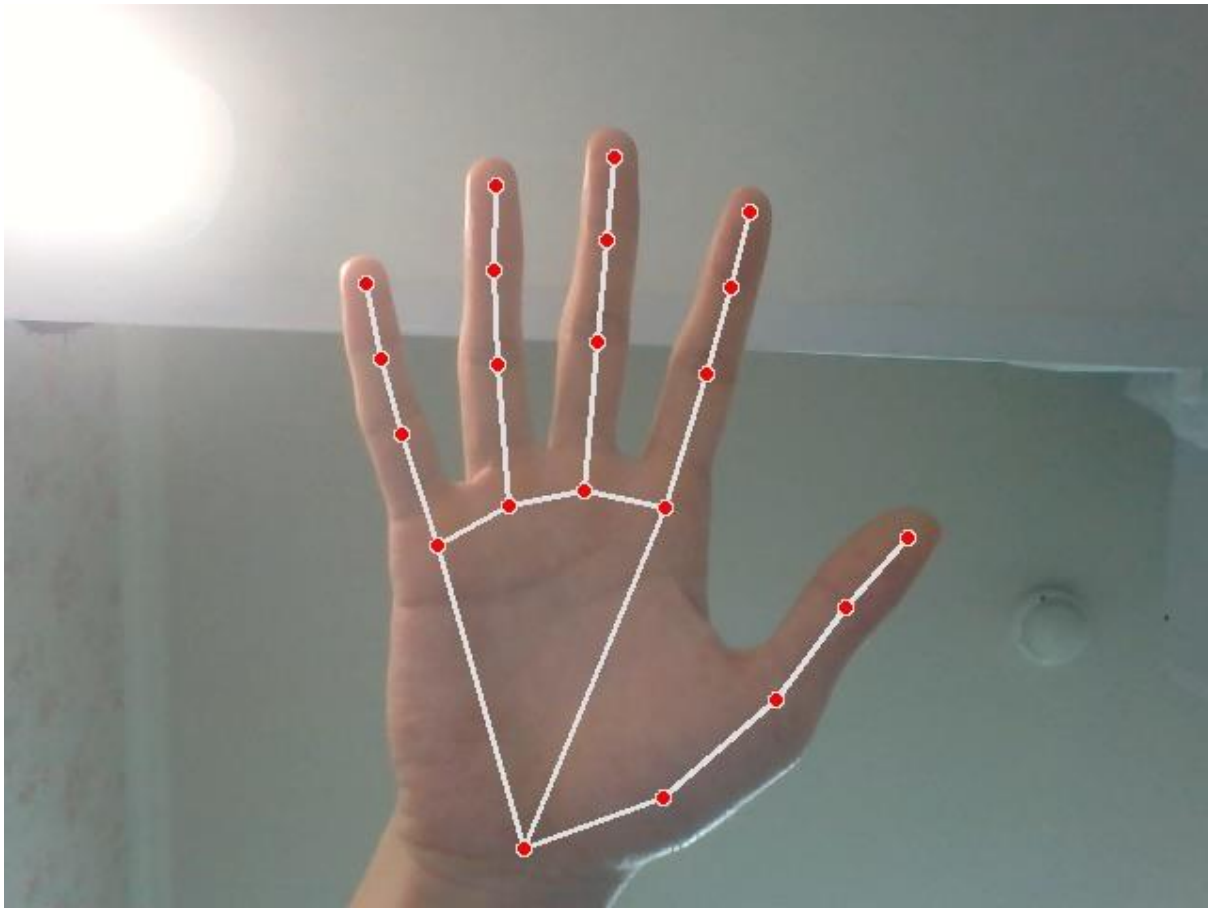
파일명 : hand_landmark_data/hand_landmarks_1712937895.npy



파일명 : hand_landmark_data/hand_landmarks_1712937902

위에 2개는 4차원 배열 (x,y,z,visibility) 좌표만 땀





위에 2개는 벡터를 이용해 각도를 땀.

1번 TASK : 저장된 좌표의 차이를 PID 이용해서 보정

```
import numpy as np

from sklearn.metrics.pairwise import euclidean_distances


# 두 개의 npy 파일에서 손의 landmark 좌표를 읽어옵니다.

hand1_landmarks = np.load('hand_landmark_data/hand_landmarks_1712937902.npy')

hand2_landmarks = np.load('hand_landmark_data/hand_landmarks_1712937895.npy')


error = hand2_landmarks - hand1_landmarks


# 각 좌표 간의 거리를 계산하여 scalar 값을 얻습니다.
```

```
error_magnitude = np.linalg.norm(error, axis=1)


best_similarity = 0.0
best_params = {}


# PID 제어기를 설정합니다.
for kp in np.linspace(0.01, 2, num=20):  # 비례 게인을 0.01부터 2까지 20개로 조정합니다.
    for ki in np.linspace(0.001, 0.5, num=20):  # 적분 게인을 0.001부터 0.5까지 20개로 조정합니다.
        for kd in np.linspace(0.001, 0.5, num=20):  # 미분 게인을 0.001부터 0.5까지 20개로 조정합니다.
            # 각 손의 랜드마크 좌표에 대해 PID 제어기를 적용합니다.
            corrected_hand2_landmarks = np.zeros(hand2_landmarks.shape)
            cumulative_error = np.zeros_like(error_magnitude)
            for i in range(hand2_landmarks.shape[0]):
                # 현재 오차를 기반으로 PID 제어를 수행합니다.
                current_error = error_magnitude[i]

                # PID 제어 알고리즘을 적용합니다.
                pid_output = kp * current_error + ki * np.sum(cumulative_error) + kd * (current_error - cumulative_error[i])

                # 보정된 오차를 사용하여 두 번째 손의 landmark 좌표를 수정합니다.
                corrected_hand2_landmarks[i] = hand2_landmarks[i] - (pid_output * (error[i] / np.linalg.norm(error[i])))

                # 누적 오차를 업데이트합니다.
```

```
                cumulative_error[i] += current_error

            # 유클리드 거리를 사용하여 두 개의 landmark 좌표 간의 유사도를 계산합니다.
            similarity = np.mean(1 / (1 + euclidean_distances(hand1_landmarks,
corrected_hand2_landmarks)))  # 유사도를 0과 1 사이의 값으로 제한

            # 가장 높은 유사도와 해당하는 게인들을 저장합니다.
            if similarity > best_similarity:
                best_similarity = similarity
                best_params = {'kp': kp, 'ki': ki, 'kd': kd}

print(f"가장 높은 유사도: {best_similarity}, 해당하는 게인들: {best_params}")
```

실행한 결과 :

가장 높은 유사도: 0.773349775357674, 해당하는 게인들: {'kp': 0.7431578947368421, 'ki': 0.10605263157894737, 'kd': 0.001}

유사도를 올려보려는 노력을 했으나 더 이상 올라가지 않았다.

1. **PID 제어 알고리즘의 선택**: 현재 코드에서는 간단한 PID 제어 알고리즘을 사용하고 있습니다. 이 알고리즘이 주어진 문제에 적합하지 않거나, 더 복잡한 제어 알고리즘이 필요할 수 있습니다. 더 복잡한 알고리즘을 구현하고 테스트하여 더 나은 결과를 얻을 수 있습니다. → 기본 알고리즘을 적용을 했다!
2. **랜드마크 데이터의 특성**: 손의 랜드마크 데이터가 다양한 패턴을 가지고 있을 수 있습니다. 현재 PID 제어 알고리즘이 이러한 다양성을 고려하지 못할 수 있습니다. 좀 더 유연한 알고리즘을 개발하여 이러한 다양성을 다룰 필요가 있습니다. → 이거는 잘 모르겠다.
3. **다른 제어 매개변수의 조정**: PID 제어에는 비례, 적분, 미분 게인 이외에도 다른 매개변수가 있을 수 있습니다. 예를 들어, 샘플링 주기, 필터링, 제어 출력 제한 등이 있을 수 있습니다. 이러한 매개변수를 조정하여 더 나은 결과를 얻을 수 있습니다.

2번 TASK : LANDMARK 좌표 읽고 실시간으로 비교

```
import cv2
import mediapipe as mp
import numpy as np

# MediaPipe hands model 초기화
mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
hands = mp_hands.Hands(
    max_num_hands=1,
    min_detection_confidence=0.5,
    min_tracking_confidence=0.5)

# 웹캠 실행
cap = cv2.VideoCapture(0)

# 캡쳐한 손의 랜드마크 불러오기
captured_landmarks = np.load('hand_landmark_data₩hand_landmarks_1712937902.npy')

while cap.isOpened():
    ret, img = cap.read()

    # 이미지 좌우 반전
    img = cv2.flip(img, 1)

    # 이미지를 RGB로 변환
```

```
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# 미디어 파이프로 손 감지
results = hands.process(img_rgb)

# 손이 감지되었을 때만 랜드마크 시각화 및 오차 계산
if results.multi_hand_landmarks:
    for hand_landmarks in results.multi_hand_landmarks:
        # 손 랜드마크 시각화
        mp_drawing.draw_landmarks(img, hand_landmarks, mp_hands.HAND_CONNECTIONS)

        # 손 랜드마크를 넘파이 배열로 변환 (x, y, z 좌표 및 가시성 포함)
        landmarks = np.array([[lm.x, lm.y, lm.z, lm.visibility] for lm in hand_landmarks.landmark])

        # 캡처한 손의 랜드마크와 현재 손의 랜드마크 간의 오차 계산
        error = np.linalg.norm(captured_landmarks - landmarks)

        # 오차가 0.1 이하인 경우 "correct" 출력
        if np.mean(error) <= 0.15:
            print("correct")

# 화면에 출력
cv2.imshow('Hand Landmark Detection', img)

# 종료 조건
if cv2.waitKey(1) == ord('q'):
```

```
        break


# 리소스 해제

cap.release()

cv2.destroyAllWindows()
```

실행한 결과 :


bandicam 2024-04-13 11-42


bandicam 2024-04-13 11-45

나의 생각 :

좌표이기 때문에 내가 정해둔 좌표와 비슷한 위치를 해야한다. 다른 위치에서 할 경우 ERROR는 매우 크다.

각도를 이용하는게 다른 위치에서도 동작을 했을경우 인식을 할 수 있는데 ERROR 값을 어떻게 해야 할지 모르겠다..

3번 TASK : LANDMARK 각도 읽고 실시간으로 비교

보통 손목에서 시작해서 다른 관절 또는 끝점까지의 벡터를 사용하여 원점에서 해당 관절 또는 끝점까지의 방향을 나타내는 선분을 그리고, 그것에 대한 각도를 측정합니다. 따라서 손목을 기준으로 각 LANDMARK까지의 벡터를 그려서 해당 벡터에 대한 각도를 계산하는 것이 일반적인 방법입니다.