개별연구

(CS 노인을 위한 제스처 인식 시스템 설계)

컴퓨터공학과 2020112057 이지호

Part 1. 개요

1) 연구 목표

2) 기대 효과

본 연구에서는 일반적인 컴퓨터 환경에 익숙하지 않고, 거동이 불편한 노인과 같은 사용자를 위하여 특화된 **비전 기반 제스처 인식 시스템**을 설계하고, 실험하고자 한다.

1) 연구 목표

터치스크린 방식의 정보전달 시스템인 기존의 키오스크는 시력 저하와 같은 신체적 노화, 기계에 대한 부정적 인식을 가진 노인에게 불편함을 초래한다.
따라서 사용자의 손동작(제스처)을 인식하여 동작하는 키오스크를 제작하고자 한다.

2) 기대효과

글자를 읽고 해당 글자를 터치하는 것 대신, 글자를 보거나 듣고 적절한 손동작을 취하는 것으로 의사 표현을 하여 노인이나 시각장애인도 키오스크를 쉽게 이용할 수 있다. 본 연구를 통해 디지털 정보 격차 문제가 해소되기를 기대한다.

Part 2. 연구 배경

1) 키오스크란?
2) 문제 인식
3) 상황 분석
4) 해결책 제안

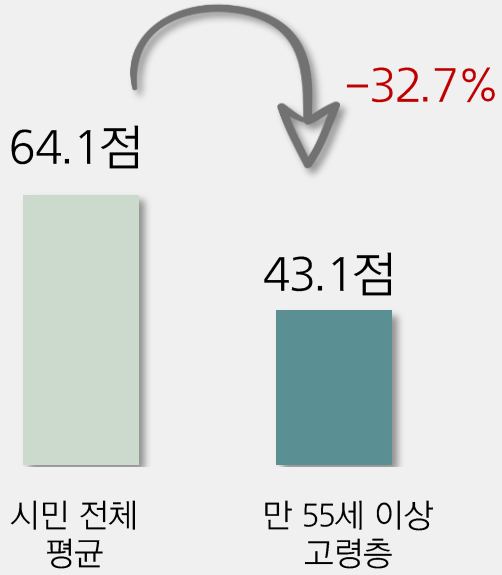
1) 키오스크란?

정보 서비스와 업무의 무인·자동화를 통해 대중들이 쉽게 이용할 수 있도록 공공장소에 설치한 무인 단말기로, 인터렉티브 키오스크라고 한다. 대개 터치 스크린이 탑재된 안내기기 또는 주문기를 말한다.

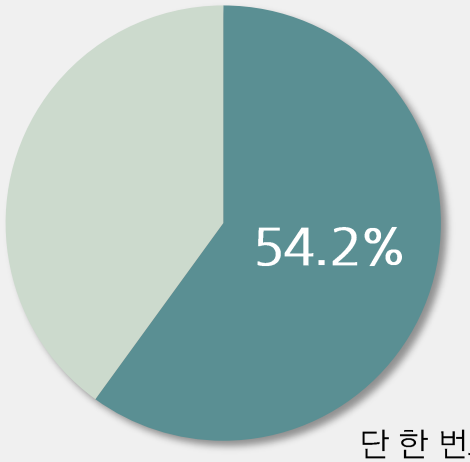
2) 문제 인식

출처 : 서울디지털재단

서울시민 디지털 역량조사



고령층의 키오스크 이용 경험



단 한 번도
사용해본 적 없음.

출처 : 소비자학연구

글씨가 너무 작아서 잘 안보입니다. 화면을 가까이 들여다보아야 하니까 몸도 피곤하고, 주문 시간이 오래 걸리네요. 지문이 닳아서 그런 것인지 화면 만지는 것도 잘 안 돼요. 그냥 키오스크 사용 안 하는 식당 갈래요.

이O창(67세, 남)

3) 상황 분석

고령층이 키오스크를 어려워하는 이유

기계에 대한 부정적 심리

낯섦, 정보의 부족, 위축됨

다른 소비자의 존재

사람들의 부정적 시선,
빨리 해야 한다는 부담감

비직관적인 UI

과도한 애니메이션,
복잡한 UI,
작은 글씨와 영문 글씨,
주문을 방해하는 광고

신체적 조건

시력 저하, 비주얼 크라우딩,
화면의 높이,
낮은 터치 인식률

3) 상황 분석

Visual Crowding(비주얼 크라우딩)

사람은 나이가 들면서 시각, 인지 기능이 자연스럽게 줄어들고, 이로 인해 시각 정보가 좁은 공간에 여러 개 있을 때 이를 명확하게 인지하지 못하는 “비주얼 크라우딩(시각적 혼잡)”이 나타난다.

4) 해결책 제안

정전식 터치패드가 아닌 카메라를 통해 손동작(제스처)을 인식하여 동작하는 키오스크

글자를 읽고 해석해 터치하는 것이 어렵다면, 글자를 읽어주고 제스처로 의사 표현을 하게끔 하면 된다. 키오스크에 NUI(Natural User Interface)를 도입하여 마치 청각 장애인이 수화로 의사소통 하듯, 우리는 키오스크와 제스처로 의사소통 하는 것이다.

NUI(Natural User Interface)란?

사용자가 최대한 자연스럽게 사용할 수 있는 사용자 인터페이스를 의미한다. 센서 기술과 인공지능 기술이 발전함에 따라 사용자의 행동을 직접 입력 받을 수 있게 되며 주목받고 있는 개념이다. 학습하는 데에 장벽이 없고 접근성이 높아 디지털 격차를 줄일 수 있다. 그러나 보안에 좋지 않고 복잡한 처리가 불가능하다는 단점이 있다.

Part 3. 구체화

1) 문제 정의
2) 실험 계획
3) 사용 기술 및 언어
4) 세부 일정

1) 문제 정의

“터치가 아닌 제스처 인식(NUI)으로 동작하는 키오스크를 제작한다.”

1. 노인을 위한 SLOW ZONE에 있다고 가정하여 키오스크의 UI, 제스처 모두 노인 친화적으로 제작한다.
2. 제스처 인식 시스템의 입력 값은 사용자의 손 제스처, 출력 값은 이에 해당하는 동작이다.

2) 실험 계획

1. 키오스크에 각각 다른 위치에 부착된 카메라와 손의 각도에 따른 인식률을 실험한다.
2. 키오스크와 사용자의 거리에 따른 인식률을 실험한다.
3. 제스처 별 인식률을 비교한다.

3) 사용 기술 및 언어

Python Visual Studio Code OpenCV MediaPipe

4) 세부 일정

Week	1	2	3	4	5	6	7	8	9	10	11	12	13
연구 방향 설정 및 자료, 오픈소스 조사													
1차 개발 (손 검출 단계)													
2차 개발 (제스처 인식 단계)													
3차 개발 (스크린 연동)													
실험 및 결과 분석													
논문 작성													

Part 4. 시스템 및 알고리즘 설계

1) 시스템 설계
2) 제어 명령에 할당할 제스처
3) 키오스크 제어 알고리즘
4) 제스처 인식 알고리즘

1) 시스템 설계

가상의 키오스크 UI

노인에게 적합한 키오스크 화면을 가상으로 구성하였고, 이에 맞는 동작(터치)들을 특정 제스처와 매칭하고자 한다.

특징

- 한 화면에는 하나의 질문만
- 불필요한 정보, 광고 제거
- 글씨는 크고 굵게
- 외래어는 되도록 우리말로
- 사진과 그림을 적극적으로 활용

* 본 모니터 화면은 대한민국디자인전람회의 노인 키오스크 우수 디자인(모두 UI)을 참고하여 재구성하였습니다.

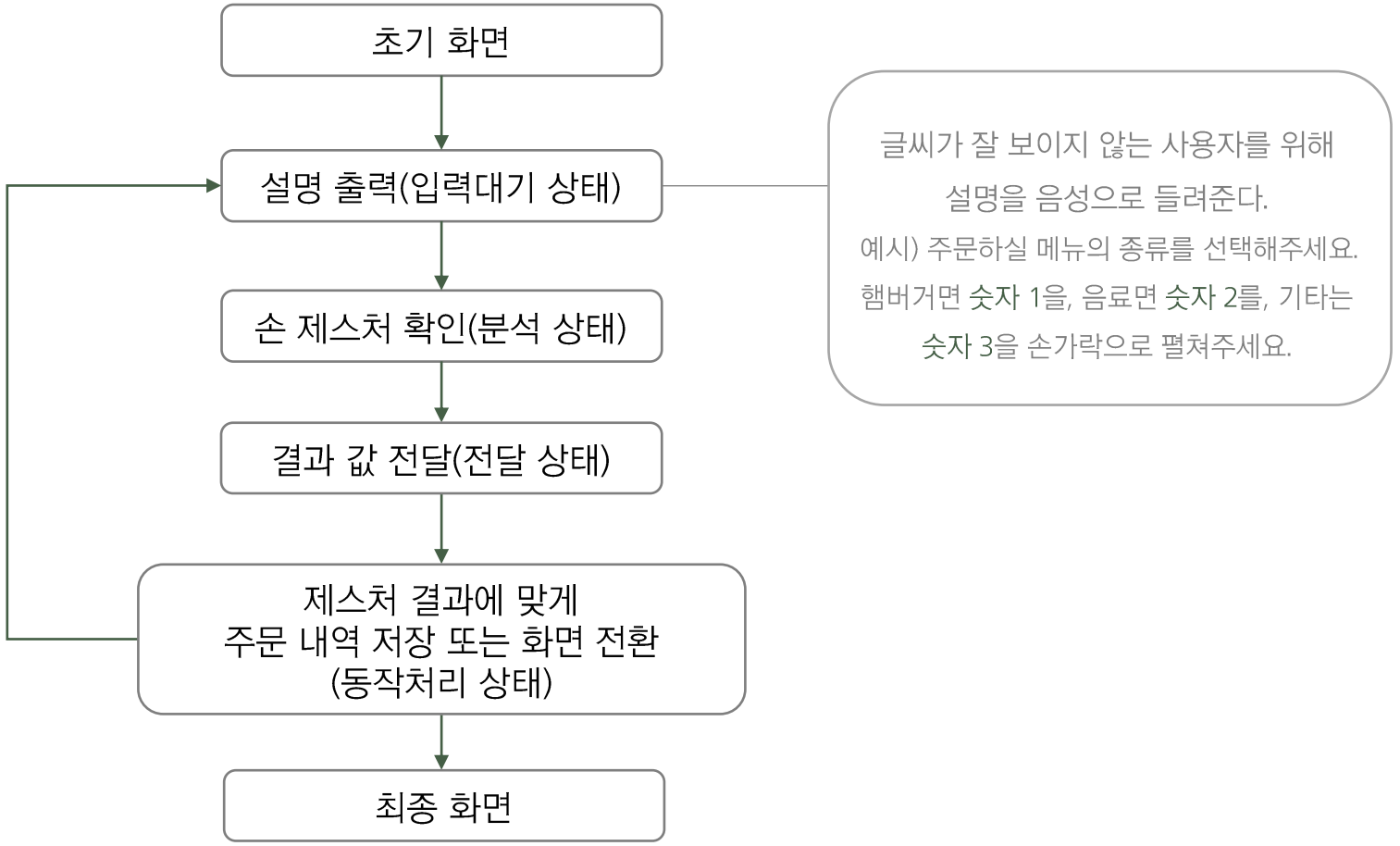
2) 제어 명령에 할당할 제스처

키오스크 제어 동작

제스처	명령
OK	예 / 다음 화면으로
손가락 숫자(5)	아니오
손가락 숫자(0,1,2)	해당하는 선택지

노인의 경우 전신으로 제스처를 취하기 보다는 손으로, 두 손보다는 한 손으로 제스처를 취하는 것이 쉬우며,
손놀림의 정확성이 떨어져 단추를 채우거나 글씨를 쓰는 등의 세밀한 동작은 어렵다.

3) 키오스크 제어 알고리즘



4) 제스처 인식 알고리즘

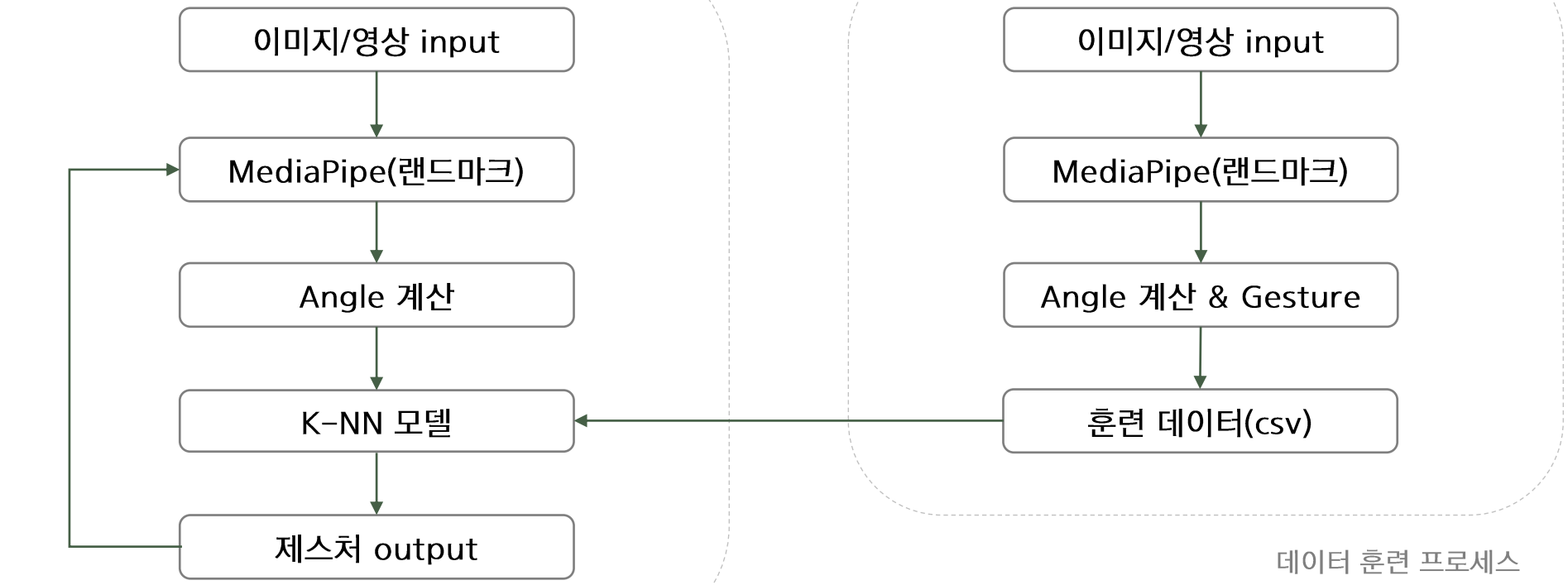
MediaPipe

- 손 제스처를 인식하기 위하여 실시간 미디어를 위한 머신러닝 솔루션인 MediaPipe 활용
- 사용자 손 위치, 좌표 정보 획득
- 훈련 데이터를 csv 파일 형태로 생성

K-NN(K-Nearest Neighbor) 알고리즘

- 테스트 데이터와 가장 가까이 있는 k개의 학습 데이터를 찾아 분류 또는 회귀를 수행하는 알고리즘
- 입력된 이미지에서 손의 좌표를 입력 받아 각도를 계산
- k-NN 알고리즘을 거쳐서 해당 손 제스처 값 출력

4) 제스처 인식 알고리즘



손 제스처 인식 프로세스

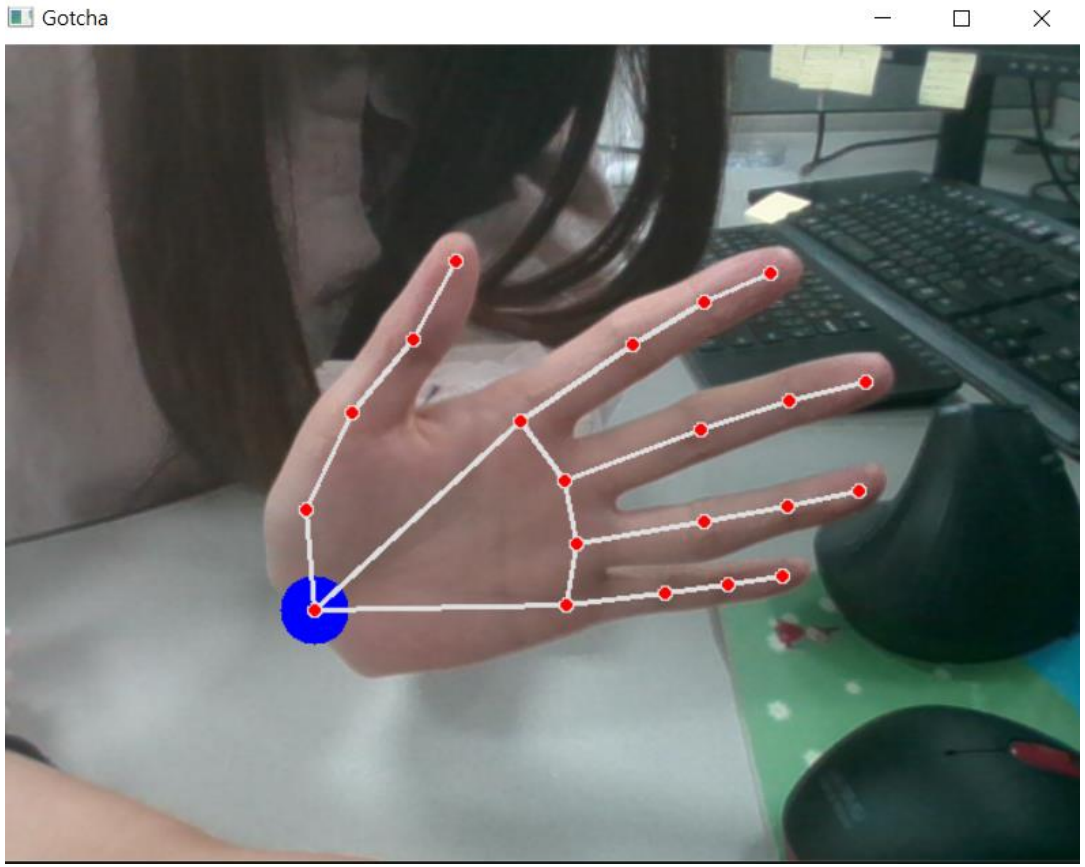
데이터 훈련 프로세스

Part 5. 결과물

1) 손 인식
2) 제스처 데이터 수집
3) 제스처 인식
4) 키오스크 화면 제어
5) 데모 영상

1) 손 인식

```python
import cv2
import mediapipe as mp
import time

cap = cv2.VideoCapture(0)

mp_drawing = mp.solutions.drawing_utils # 손 위에 그림을 그릴 수 있는 메소드
mp_hands = mp.solutions.hands

hands = mp_hands.Hands(
    max_num_hands = 1, # 인식할 손모양의 갯수, 생략하면 2가 지정된다.
    min_detection_confidence = 0.5, # 성공적인 것으로 간주되는 최소 신뢰도 값. 0.0 ~1.0사이로서 기본값은 0.5이다.
    min_tracking_confidence = 0.5) # 손 랜드마크가 성공적으로 추적된 것으로 간주되는 최소 신뢰도 값. 0.0 ~1.0 사이로서 기본값은

while True:
    success, img = cap.read()
    if not success:
        continue
    # OpenCV 영상은 BGR 형식인데 MediaPipe에서는 RGB 형식을 사용하므로 영상형식을 변환해 준다.
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    # MediaPipe의 hands 모듈을 이용해서 손동작을 인식한다. 손동작 인식 AI모델이 작동되고 결과 값이 result로 저장된다.
    results = hands.process(imgRGB)
    # MediaPipe용 RGB 형식으로 변환했던 것을 OpenCV 영상처리를 위해 다시 BGR형식으로 되돌린다.
    imgRGB = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)

    # result값이 정상인 경우에만 후속 작업 처리한다.
    if results.multi_hand_landmarks:
        # 실제 손이 인식될 때 점(랜드마크)을 찍는다.
        for handLms in results.multi_hand_landmarks:
            # results로 반환된 landmark 데이터를 사용한다. 인식된 손가락 모양은 index값을 가지는 배열로 제공된다.
            for id, lm in enumerate(handLms.landmark):
                h, w, c = img.shape
                cx, cy = int(lm.x*w), int(lm.y*h)
                print(id, " :" , cx, cy)
                if id == 0:
                    cv2.circle(img, (cx,cy), 20, (255,0,0), cv2.FILLED)

            # MediaPipe에 내장된 유틸리티 기능을 이용해서 구해진 손가락 모양을 서로 연결한 그림을 그려준다.
            mp_drawing.draw_landmarks(img, handLms, mp_hands.HAND_CONNECTIONS)

    cv2.imshow("Gotcha", img)
    cv2.waitKey(1)
```



2) 제스처 데이터 수집

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

max_num_hands = 1 # 인식할 수 있는 손 개수
gesture = {
    0:'fist', 1:'one', 2:'two', 3:'three', 4:'four', 5:'five',
    6:'six', 7:'rock', 8:'spiderman', 9:'scissors', 10:'ok'
} # 11가지의 제스처, 제스처 데이터는 손가락 관절의 각도와 각각의 라벨을 포함한다.

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

# Gesture recognition data
file = np.genfromtxt('C:/Users/LJH/Desktop/2022-IDPCD/project/data/test_train.csv', delimiter=',')
print(file.shape)

cap = cv2.VideoCapture(0)

def click(event, x, y, flags, param):
    global data, file
    if event == cv2.EVENT_LBUTTONDOWN:
        file = np.vstack((file, data))
        print(file.shape)

cv2.namedWindow('Dataset')
cv2.setMouseCallback('Dataset', click)

while cap.isOpened():
    ret, img = cap.read()
    if not ret:
        continue

    img = cv2.flip(img, 1)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    result = hands.process(img)

    img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)

    if result.multi_hand_landmarks is not None:
        for res in result.multi_hand_landmarks:
            joint = np.zeros((21, 3))
            for j, lm in enumerate(res.landmark):
                joint[j] = [lm.x, lm.y, lm.z]

            # Compute angles between joints
            v1 = joint[[0,1,2,3,0,5,6,7,0,9,10,11,0,13,14,15,0,17,18,19],:] # Parent joint
            v2 = joint[[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20],:] # Child joint
            v = v2 - v1 # [20,3]
            # Normalize v
            v = v / np.linalg.norm(v, axis=1)[:, np.newaxis]

            # Get angle using arcos of dot product
            angle = np.arccos(np.einsum('nt,nt->n',
                v[[0,1,2,4,5,6,8,9,10,12,13,14,16,17,18],:],
                v[[1,2,3,5,6,7,9,10,11,13,14,15,17,18,19],:])) # [15,]

            angle = np.degrees(angle) # Convert radian to degree

            data = np.array([angle], dtype=np.float32)

            data = np.append(data, 11) #

            mp_drawing.draw_landmarks(img, res, mp_hands.HAND_CONNECTIONS)

    cv2.imshow('Dataset', img)
    if cv2.waitKey(1) == ord('q'):
        break

np.savetxt('data/gesture_train_fy.csv', file, delimiter=',')
```

```
32.647995,27.334458,18.777239,32.558145,149.876268,29.578624,37.800344,133.458050,45.085468,32.342115,142.727088,37.366495,26.056262,139.415512,40.519754,0.000000
45.166793,26.117019,18.645196,31.567098,138.965945,20.022370,33.176819,132.806630,21.381263,28.201580,138.854737,14.808465,26.329570,128.394781,23.400235,0.000000
20.826588,35.926397,39.943409,20.257126,88.651968,29.781060,17.812546,102.683664,16.751548,11.247996,103.264677,18.000063,15.584646,82.515342,23.045704,0.000000
31.327256,24.482902,27.073693,22.894414,64.537885,47.775269,16.256466,89.803293,40.146627,9.443338,91.327471,35.448822,14.446064,71.702363,35.047523,0.000000
28.164625,31.028619,23.568194,21.468457,141.750809,21.386416,29.323112,129.772949,38.852031,24.264641,138.309162,33.321451,23.231644,129.452243,30.819853,0.000000
39.493933,30.825017,37.798715,48.789407,101.569152,29.543310,62.408973,102.175635,65.566150,59.301602,112.161872,55.427656,60.565796,102.860590,49.489659,0.000000
46.636410,27.871526,25.808261,22.547515,69.819060,42.556670,16.971363,85.544017,36.496952,11.288500,75.542530,39.251253,16.539423,48.637377,29.019384,0.000000
45.042568,19.961871,11.893033,13.603903,32.278803,27.388893,7.747610,17.797417,5.794515,2.940872,16.492116,1.481793,9.127502,8.934992,6.483260,0.000000
47.299112,32.659637,25.455925,33.674011,52.766652,43.792789,29.175726,97.063906,24.384931,27.473441,104.583461,24.572468,32.064351,94.460182,22.230902,0.000000
46.483112,30.871050,36.277547,40.493482,96.787611,39.591184,38.159227,110.260392,30.181509,30.411843,119.519070,30.006762,26.418003,122.606342,33.310417,0.000000
27.792747,27.346865,18.358880,13.459867,12.675573,14.827358,21.851454,94.371765,35.972556,19.283318,104.185629,31.780360,26.073177,81.116557,33.292727,1.000000
35.320273,26.381613,20.191984,14.668020,7.185043,4.248832,25.126893,121.084548,26.620646,24.078772,117.775156,30.698631,30.541078,89.432634,37.903796,1.000000
26.048195,19.402587,26.436642,6.365824,9.542588,3.241575,28.086456,113.535227,26.468682,20.288474,116.216192,24.670197,26.922405,95.790580,20.527646,1.000000
27.221395,15.414500,25.939750,6.623381,7.608257,2.696879,28.052599,108.763997,27.351137,21.551970,108.780979,26.114410,25.929242,91.324413,21.103923,1.000000
32.014172,10.991831,28.415461,5.728636,6.481493,0.889479,23.010316,90.874638,34.321755,12.829574,92.857042,35.219209,11.395936,68.408045,41.629204,1.000000
35.391177,18.592058,33.007410,4.140826,6.493375,1.912261,25.824411,100.550619,29.616766,19.956852,99.451776,31.702370,24.273458,82.648994,25.282382,1.000000
33.667543,32.918064,19.468711,7.488233,2.724066,7.948471,28.506697,49.497494,13.477383,55.206680,70.580958,23.938099,54.585635,72.790658,27.310205,1.000000
34.297057,31.368970,12.617624,12.278204,1.644287,8.980053,30.469056,26.478147,7.106024,52.598374,70.892878,25.786772,42.428934,65.808129,34.571704,1.000000
37.283942,26.503899,38.901662,12.471588,1.771188,4.433059,38.654754,109.571409,23.595078,35.738030,105.880785,31.167126,25.614191,107.024708,29.663542,1.000000
37.309748,20.163841,29.692652,7.735964,3.484735,2.936218,33.402019,102.752624,25.379006,27.139984,100.369156,26.013738,25.387794,95.077939,19.624609,1.000000
27.064267,4.116995,11.001622,16.486113,10.552898,11.181537,14.660537,7.230117,6.805212,5.992467,1.279764,12.949280,2.583446,5.751485,12.190489,9.000000
31.191134,4.290191,11.017077,14.137684,3.406932,3.979965,14.172432,62.540096,5.982379,4.602033,33.828657,3.623022,7.488203,11.515055,14.876723,9.000000
32.675400,6.051246,9.601219,16.191259,6.049887,3.021566,23.574714,34.476614,52.095947,20.944558,28.680245,10.530155,21.732707,9.850545,3.993437,9.000000
36.983640,10.596499,14.782542,8.568285,5.152676,2.903961,20.725178,85.560392,20.407994,15.760408,73.182706,27.857098,16.365531,47.081374,18.958339,9.000000
28.553979,4.386473,14.044008,14.800291,8.822664,4.434532,12.653980,85.107713,5.685620,4.419841,57.803916,8.104906,9.925122,21.634206,28.194048,9.000000
43.905587,4.436173,6.406616,7.984581,3.269949,4.945221,21.120639,23.389945,11.310605,16.343349,24.295424,15.957736,9.628379,15.356038,23.239797,9.000000
47.227182,2.093504,13.185004,10.198872,9.016928,4.762649,30.488841,87.460182,19.246491,31.417656,83.437891,35.377278,26.301992,71.631192,41.428162,9.000000
38.963707,3.312926,14.764106,13.346643,2.701027,3.122325,15.783439,70.890089,24.271086,16.162595,55.105191,27.125569,14.065205,31.765137,30.163877,9.000000
38.235214,9.765067,12.204059,13.054985,3.186906,6.008764,30.496132,105.006685,19.614097,37.621662,101.843967,33.236817,36.027421,96.934316,33.416261,9.000000
44.894136,5.703140,18.672275,16.862815,7.250208,1.171276,27.349864,91.238617,15.085455,31.288754,83.884860,30.505573,32.303599,72.093530,33.306199,9.000000
33.233165,5.251370,12.361472,7.406677,2.403716,1.809087,14.060716,1.544563,4.514234,31.659243,100.935772,25.510835,35.251030,89.933031,19.148129,3.000000
52.796238,11.266403,14.081873,4.255632,1.900488,2.120873,8.555830,4.423145,2.657591,32.922917,102.226827,22.889829,35.501908,85.247861,24.733276,3.000000
45.530282,6.599183,2.705272,11.851240,11.457306,8.534295,10.935138,3.143677,7.689091,40.015122,108.650040,20.131360,44.474448,95.466861,23.438341,3.000000
40.178526,5.883763,11.293230,9.412355,5.010292,3.067079,17.325150,5.933416,5.999398,39.241775,112.753544,23.859172,37.437907,104.919452,24.762976,3.000000
41.220657,6.328683,11.057584,8.445178,5.794255,1.084988,11.758416,8.098485,5.679773,36.361657,116.513918,16.134616,37.406455,101.434675,18.718483,3.000000
58.029470,7.936083,18.771053,7.850386,3.486047,9.073247,6.786294,4.949442,3.565229,34.004830,84.326025,32.755850,41.777266,74.807435,32.737475,3.000000
47.655365,6.263117,10.347858,4.807681,4.465829,8.093180,5.107651,1.721795,4.607089,33.309515,102.655187,23.337544,38.665966,107.032195,18.660136,3.000000
37.110564,9.127239,10.668107,9.388768,7.970887,9.143346,12.781739,13.336964,6.699117,2.049043,44.085116,10.394698,5.635468,36.843559,23.718730,3.000000
38.684006,11.977840,10.109062,4.396552,1.626858,0.703276,12.755652,4.171569,4.843316,33.498154,83.193397,31.964912,32.269514,74.880485,33.148861,3.000000
37.850077,11.219031,11.298609,4.051946,5.171304,1.238764,9.662268,7.351379,3.868443,29.601931,85.170091,34.670269,24.091245,67.618521,43.369632,3.000000
33.375924,45.897643,44.959032,4.206236,2.510320,3.222605,6.785700,5.948434,5.084050,4.488179,10.262063,2.284405,10.728425,9.843519,2.926731,4.000000
42.745151,37.643024,50.160862,5.435130,4.830251,6.826986,5.556171,3.002356,0.205582,4.943730,4.228434,1.315191,14.777243,1.532173,2.396397,4.000000
36.649501,39.702291,44.822909,3.642930,8.181224,8.751886,3.456824,9.219551,4.474447,5.561067,16.236727,1.138055,18.158290,4.719707,7.470625,4.000000
33.732343,45.422586,55.822462,7.638146,2.152629,4.033472,1.464054,4.268573,1.663308,2.984584,7.626433,3.389159,6.941995,4.292772,5.763352,4.000000
38.116837,42.441980,54.622501,5.664158,2.892497,5.785950,4.736157,4.173156,2.245456,4.465982,7.503835,2.509713,12.375703,4.960036,2.874021,4.000000
24.033897,40.088233,38.787525,9.386012,9.464247,10.413310,5.538884,8.901695,6.428157,5.607933,8.132257,8.182374,12.767122,4.234681,9.934021,4.000000
```

3) 제스처 인식

```python
import cv2 # 웹캠 제어 및 ML 사용
import mediapipe as mp # 손 인식을 할 것
import numpy as np

result='result'
max_num_hands = 1 # 손은 최대 1개만 인식
gesture = { # **11가지나 되는 제스처 라벨, 각 라벨의 제
    0:'fist', 1:'one', 2:'two', 3:'three', 4:'four',
    6:'six', 7:'rock', 8:'spiderman', 9:'scissors', 1
}
kiosk_gesture = {0:'zero', 1:'one', 2:'two', 10:'ok'}

# MediaPipe hands model
mp_hands = mp.solutions.hands # 웹캠 영상에서 손가락
mp_drawing = mp.solutions.drawing_utils # 웹캠 영상에

 # 손가락 detection 모델을 초기화
hands = mp_hands.Hands(
    max_num_hands=max_num_hands, # 최대 몇 개의 손을
    min_detection_confidence=0.5, # 0.5로 해두는 게
    min_tracking_confidence=0.5)

# 제스처 인식 모델
file = np.genfromtxt('C:/Users/LJH/Desktop/2022-IDPC
angle = file[:,:-1].astype(np.float32) # 각도
label = file[:, -1].astype(np.float32) # 라벨
knn = cv2.ml.KNearest_create() # knn(k-최근접 알고리즘
knn.train(angle, cv2.ml.ROW_SAMPLE, label) # 학습!

cap = cv2.VideoCapture(0)

while cap.isOpened(): # 웹캠에서 한 프레임씩 이미지를
    ret, img = cap.read()
    if not ret:
        continue

    img = cv2.flip(img, 1)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    result = hands.process(img)

    img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
```

```python
# 각도를 인식하고 제스처를 인식하는 부분
if result.multi_hand_landmarks is not None: # 만약 손을 인식하면
    for res in result.multi_hand_landmarks:
        joint = np.zeros((21, 3)) # joint == 랜드마크에서 뽑은 점, joint는 21개가 있고 x,y,z 좌표니까 21,3
        for j, lm in enumerate(res.landmark):
            joint[j] = [lm.x, lm.y, lm.z] # 각 joint마다 x,y,z 좌표 저장

        # Compute angles between joints joint마다 각도 계산
        # **공식문서 들어가보면 각 joint 번호의 인덱스가 나옴**
        v1 = joint[[0,1,2,3,0,5,6,7,0,9,10,11,0,13,14,15,0,17,18,19],:] # Parent joint
        v2 = joint[[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20],:] # Child joint
        v = v2 - v1 # [20,3]짜리벡터
        # Normalize v
        v = v / np.linalg.norm(v, axis=1)[:, np.newaxis] # 벡터 정규화(크기 1 벡터) = v / 벡터의 크기

        # Get angle using arcos of dot product **내적 후 arcos으로 각도를 구해줌**
        angle = np.arccos(np.einsum('nt,nt->n',
            v[[0,1,2,4,5,6,8,9,10,12,13,14,16,17,18],:],
            v[[1,2,3,5,6,7,9,10,11,13,14,15,17,18,19],:])) # [15,]

        angle = np.degrees(angle) # Convert radian to degree

        # Inference gesture 학습시킨 제스처 모델에 참조를 한다.
        data = np.array([angle], dtype=np.float32)
        ret, results, neighbours, dist = knn.findNearest(data, 3) # k가 3일 때 값을 구한다!
        idx = int(results[0][0]) # 인덱스를 저장!

        # Draw gesture result
        if idx in kiosk_gesture.keys():
            cv2.putText(img, text=kiosk_gesture[idx].upper(), org=(int(res.landmark[0].x * img.shape[1]), int(res.landmark[0].y * img.shape[0] + 20)), fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=1, color=(255, 255, 255), thickness=2)
            # save gesture result
            result = kiosk_gesture[idx].upper()



        # Other gestures 모든 제스처를 표시한다면
        # cv2.putText(img, text=gesture[idx].upper(), org=(int(res.landmark[0].x * img.shape[1]), int(res.landmark[0].y * img.shape[0] + 20)), fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=1, color=(255, 255, 255), thickness=2)

        mp_drawing.draw_landmarks(img, res, mp_hands.HAND_CONNECTIONS) # 손에 랜드마크를 그려줌

cv2.imshow('Game', img)
if cv2.waitKey(1) == ord('q'):
    break
```

4) 키오스크 화면 제어

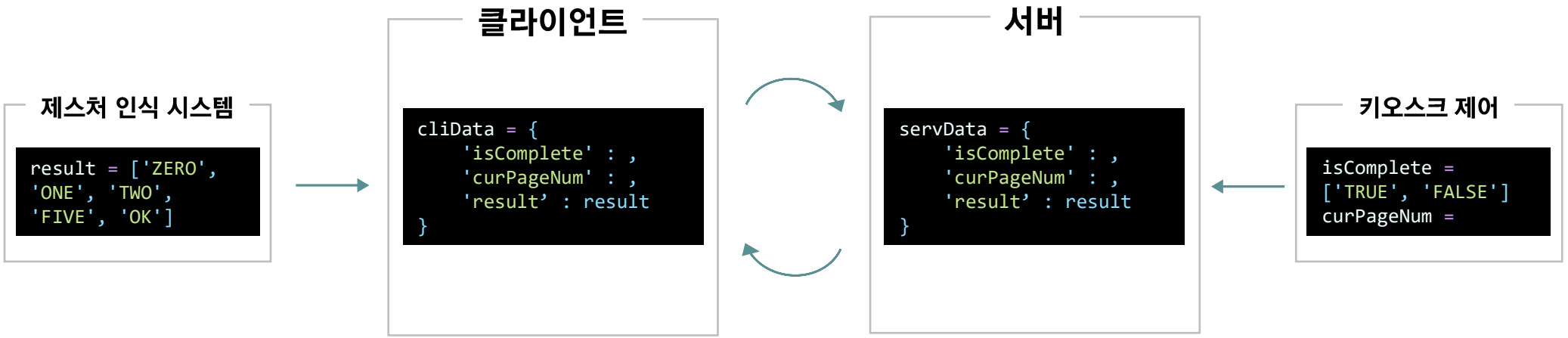
OpenCV와 소켓 통신

서버

```
PS C:\Users\LJH\Desktop\2022-IDPCD>  & 'C:\Users\LJH\AppData\Local\Programs\Python\Python37\python.exe' 'c:\Users\LJH\.vscode\extensions\ms-python.python-2022.18.2\pythonFiles\lib\python\debugpy\adapter/
../..\debugpy\launcher' '52244' '--' 'c:\Users\LJH\Desktop\2022-IDPCD\project\server.py'
Connected by ('127.0.0.1', 52258)
Received from ('127.0.0.1', 52258) hello
```

클라이언트

```
PS C:\Users\LJH\Desktop\2022-IDPCD>  & 'C:\Users\LJH\AppData\Local\Programs\Python\Python37\python.exe' 'c:\Users\LJH\.vscode\extensions\ms-python.python-2022.18.2\pythonFiles\lib\python\debugpy\adapter/
../..\debugpy\launcher' '52256' '--' 'c:\Users\LJH\Desktop\2022-IDPCD\project\client.py'
input data :hello
```



4) 키오스크 화면 제어

클라이언트

recog_gesture() 로부터 인식 결과를 받아오고,

소켓 통신을 이용해 server에게 전달

```python
# client
# 역할 : server로부터 페이지 번호와 전환 여부를 받아 올바른 제스처를 인식한 후 결과를 전송한다.

import socket, datetime, json

import sys
sys.path.append("/recog_gesture")
import recog_gesture as recog_gesture

HOST = '127.0.0.1' # local 호스트 사용
PORT = 10000 # 10000번 포트 사용
# 소켓 생성
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# 연결
client_socket.connect((HOST, PORT))

cliData = {
    'isComplete' : False,
    'curPageNum' : 1,
    'result' : ''
}
while cliData.get('isComplete') == False :
    # 제스처 인식 결과 가져오기
    result = recog_gesture.recog_gesture()

    # 데이터 서버에 전송
    cliData.update(result=result)
    sendData = json.dumps(cliData)
    client_socket.send(bytes(sendData, 'utf-8'))

    # 서버에게 데이터 받음
    data = client_socket.recv(1024)
    print(data)
    recvData = json.loads(data)
    print('recvData : ',recvData, datetime.datetime.now())
    cliData.update( isComplete=bool(recvData.get('isComplete')), curPageNum=int(recvData.get('curPageNum')))


client_socket.close()
```

4) 키오스크 화면 연동

서버

client로부터 제스처 인식 결과를 받아오고,

해당 결과를 파라미터로 하여 키오스크 제어 함수 호출

이후 제어 결과를 client에게 전달

```python
def binder(client_socket, addr):
    # 커넥션이 되면 접속 주소가 나온다.
    print('Connected by', addr)
    try:
        initData = {
            'isComplete' : False,
            'curPageNum' : 0,
            'result' : 'SERVER'
        }
        ctrlKiosk(initData)
        while True :

            # 클라이언트에게 데이터 받음
            data = client_socket.recv(1024)
            recvData = json.loads(data)
            print('recvData : ',recvData, datetime.datetime.now())

            # 키오스크 화면 제어
            servData = ctrlKiosk(recvData)

            # 클라이언트에게 데이터 전송
            sendData = json.dumps(servData)
            client_socket.send(bytes(sendData, 'utf-8'))

    except:
    # 접속 해제시 except
        print("except : " , addr)
    finally:
    # 종료
        client_socket.close()


# 소켓 생성
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

# 10000번 포트 사용
server_socket.bind(('127.0.0.1',10000))

server_socket.listen(1)

try:
    # 클라이언트가 접속하기 전까지 서버는 실행되어야하기 때문에 무한 루프 사용
    while True:
        client_socket, addr = server_socket.accept()
    # 쓰레드 사용해서 대기
        th = threading.Thread(target=binder, args = (client_socket,addr))
        th.start()
except:
    print("server")
finally:
    # 종료
    server_socket.close()
```

4) 키오스크 화면 연동

키오스크 제어

openCV를 활용하여 파라미터 값에 따라 동작 수행

수행 결과를 server에게 전달

```python
def ctrlKiosk(recvData) -> json :
    global orders
    global pickUpYn
    global cashYn
    global openImage
    curPageNum = int(recvData.get('curPageNum'))
    nextPageNum = curPageNum + 1
    result = recvData.get('result')
    src = ""

    if curPageNum == 0 : # 초기 화면 띄우기
        if result == 'SERVER' :
            src = "C:/Users/NICE-DNB/Desk
    elif curPageNum == 1 : # 초기 화면
        if result == 'OK' : # 다른 화면으로
            src = "C:/Users/NICE-DNB/Desk
    elif curPageNum == 2 : # 주문 메뉴의 종
        if result == 'ZERO' :
            src = "C:/Users/NICE-DNB/Desk
        # elif result == 'ONE' :
        #     src = ""
        # elif result == 'TWO' :
        #     src = ""
    elif curPageNum == 3 :
        if result == 'ZERO' :
            order = {
                'menu': '치즈버거',
                'count' : 1,
                'price' : 3900
            }
            orders.append(order)
            src = "C:/Users/NICE-DNB/Desk
        # elif result == 'ONE' :
        #     src = ""
        # elif result == 'TWO' :
        #     src = ""
    elif curPageNum == 4 :
        if result == 'OK' :
            order = {
                'menu': '콜라, 감자튀김',
                'count' : 1,
                'price' : 2000
            }
            orders.append(order)
            src = "C:/Users/NICE-DNB/Desktop/2022-IDPCD/project/kiosk/image/005.png"
        elif result == 'FIVE' :
            src = "C:/Users/NICE-DNB/Desktop/2022-IDPCD/project/kiosk/image/005.png"
    elif curPageNum == 5 :
        if result == 'OK' :
            src = "C:/Users/NICE-DNB/Desktop/2022-IDPCD/project/kiosk/image/002.png"
            nextPageNum = 2
        elif result == 'FIVE' :
            src = "C:/Users/NICE-DNB/Desktop/2022-IDPCD/project/kiosk/image/006.png"
    elif curPageNum == 6 :
        if result == 'ONE' :
            src = "C:/Users/NICE-DNB/Desktop/2022-IDPCD/project/kiosk/image/007.png"
        elif result == 'TWO' :
            pickUpYn = True
            src = "C:/Users/NICE-DNB/Desktop/2022-IDPCD/project/kiosk/image/007.png"
    elif curPageNum == 7 :
        if result == 'ONE' :
            src = "C:/Users/NICE-DNB/Desktop/2022-IDPCD/project/kiosk/image/008.png"
        elif result == 'TWO' :
            cashYn = True
            src = "C:/Users/NICE-DNB/Desktop/2022-IDPCD/project/kiosk/image/008.png"
    elif curPageNum == 8 :
        if result == 'OK' :
            print('주문 내역 :',orders)
            recvData.update(isComplete=True)

    image = cv2.imread(src, cv2.IMREAD_UNCHANGED)
    if image is None :
        print('Image load failed...Please gesture one more time.')
        recvData.update(result='FAIL')
    else :
        cv2.destroyAllWindows
        openImage = cv2.resize(image, dsize=(300,420), interpolation=cv2.INTER_AREA)
        recvData.update(curPageNum=int(nextPageNum))
        cv2.imshow("kiosk", openImage)
        cv2.waitKey(1)
    return recvData
```

5) 데모 영상

```
Connected by ('127.0.0.1', 56550)
recvData :  {'isComplete': False, 'curPageNum': 1, 'result': 'OK'} 2022-12-04 01:17:43.932393
recvData :  {'isComplete': False, 'curPageNum': 2, 'result': 'ZERO'} 2022-12-04 01:17:47.206552
recvData :  {'isComplete': False, 'curPageNum': 3, 'result': 'ZERO'} 2022-12-04 01:17:50.776514
recvData :  {'isComplete': False, 'curPageNum': 4, 'result': 'OK'} 2022-12-04 01:17:54.350382
recvData :  {'isComplete': False, 'curPageNum': 5, 'result': 'OK'} 2022-12-04 01:18:02.280479
recvData :  {'isComplete': False, 'curPageNum': 2, 'result': 'ZERO'} 2022-12-04 01:18:05.407264
recvData :  {'isComplete': False, 'curPageNum': 3, 'result': 'ZERO'} 2022-12-04 01:18:08.135003
recvData :  {'isComplete': False, 'curPageNum': 4, 'result': 'FIVE'} 2022-12-04 01:18:10.874425
recvData :  {'isComplete': False, 'curPageNum': 5, 'result': 'FIVE'} 2022-12-04 01:18:13.734883
recvData :  {'isComplete': False, 'curPageNum': 6, 'result': 'TWO'} 2022-12-04 01:18:16.820206
recvData :  {'isComplete': False, 'curPageNum': 7, 'result': 'ONE'} 2022-12-04 01:18:19.601855
recvData :  {'isComplete': False, 'curPageNum': 8, 'result': 'OK'} 2022-12-04 01:18:22.332660
주문 내역 : [{'menu': '치즈버거', 'count': 1, 'price': 3900}, {'menu': '콜라, 감자튀김', 'count': 1, 'price': 2000},
 {'menu': '치즈버거', 'count': 1, 'price': 3900}]
```

Part 6. 실험 및 결과 분석

1) 실험 설계 및 방법
2) 카메라 위치 및 각도
3) 카메라와 사용자 간 거리
4) 결과 분석

1) 실험 설계 및 방법

실험 주제	목적
카메라 위치 및 각도	키오스크에 부착될 카메라의 위치에 따라 보여지는 손의 각도가 다르므로 이에 따른 인식률을 비교하여 최적의 조건을 찾는다.
카메라와 사용자 간 거리	사용자마다 키오스크 모니터 및 카메라와의 거리가 다르므로 이에 따른 인식률을 비교하여 최적의 조건을 찾는다.
제스처 별로 비교	각 조건 하에서 제스처마다의 인식률을 비교하여 인식률이 낮은 원인을 분석한다.

제안하는 시스템의 최적화된 조건을 확인하고자 각기 다른 조건 하에서 인식률 실험을 실시하였다.
실험은 각기 다른 조건을 설정하고 해당 조건 하에서 특정 제스처를 취한 후, 예상 결과와 비교하는 방식으로 수행하였다.

1) 실험 설계 및 방법

```python
import sys
sys.path.append("/recog_gesture")
import recog_gesture as recog_gesture
import time

kiosk_gesture = ['ZERO', 'ONE', 'TWO', 'FIVE', 'OK']

def compare(gesture, count) -> str :
    isCorrect = 0
    isIncorrect = 0
    isFail = 0
    for i in range(0,count) :
        result = recog_gesture.recog_gesture()
        print(i+1, '. return value : ', result , time.time())
        if result == gesture : isCorrect += 1
        elif result == 'fail' : isFail += 1
        else : isIncorrect += 1
    return '인식률 ' + str(isCorrect / (count) * 100) + '% | 불일치률 ' + str(isIncorrect / count * 100) + '% | 실패률 ' + str(isFail / (count) * 100) + '%'

count = 120
for gesture in kiosk_gesture :
    print(gesture, ':', compare(gesture, count))
```

2) 카메라 위치 및 각도

해당 실험은 15cm 이내의 거리에서 진행하였음.

1. 카메라가 아래에 있을 경우 N = 120 (시행 횟수)

제스처	인식률	불일치율	실패율
ZERO	98.3%	1.7%	0%
ONE	100%	0%	0%
TWO	99.2%	0.84%	0%
FIVE	100%	0%	0%
OK	99.2%	0.84%	0%

2. 카메라가 왼쪽에 있을 경우 N = 120 (시행 횟수)

제스처	인식률	불일치율	실패율
ZERO	100%	0%	0%
ONE	100%	0%	0%
TWO	90.8%	9.2%	0%
FIVE	99.2%	0.84%	0%
OK	98.3%	1.7%	0%

3. 카메라가 오른쪽에 있을 경우 N = 120 (시행 횟수)

제스처	인식률	불일치율	실패율
ZERO	100%	0%	0%
ONE	100%	0%	0%
TWO	90.8%	9.2%	0%
FIVE	99.2%	0.84%	0%
OK	85.8%	14.2%	0%

4. 카메라가 위에 있을 경우 N = 120 (시행 횟수)

제스처	인식률	불일치율	실패율
ZERO	95.8%	4.2%	0%
ONE	88.3%	11.7%	0%
TWO	84.2%	15.8%	0%
FIVE	95%	5%	0%
OK	82.5%	7%%	0%

3) 카메라와 사용자 간 거리

해당 실험은 카메라를 아래에 놓고 진행하였음.

1

2

3

1. 거리가 15cm 이내인 경우 N = 120 (시행 횟수)

제스처	인식률	불일치율	실패율
ZERO	98.3%	1.7%	0%
ONE	100%	0%	0%
TWO	99.2%	0.84%	0%
FIVE	100%	0%	0%
OK	99.2%	0.84%	0%

2. 거리가 15cm~30cm 인 경우 N = 120 (시행 횟수)

제스처	인식률	불일치율	실패율
ZERO	98.3%	0%	1.7%
ONE	97.5%	0%	2.5%
TWO	90.8%	9.2%	0%
FIVE	99.2%	0.84%	0%
OK	95.8%	1.7%	2.5%

3. 거리가 30cm 이상인 경우 N = 120 (시행 횟수)

제스처	인식률	불일치율	실패율
ZERO	78.46%	0.84%	11.7%
ONE	82.5%	1.7%	15.8%
TWO	85.8%	5%	9.2%
FIVE	97.46%	1.7%	0.84%
OK	76.6%	9.2%	14.2%

4) 결과 분석

실험 1. 카메라 위치 및 각도에 따른 인식률 비교

1. 카메라가 아래에 있을 경우 N = 120 (시행 횟수)

제스처	인식률	불일치율	실패율
ZERO	98.3%	1.7%	0%
ONE	100%	0%	0%
TWO	99.2%	0.84%	0%
FIVE	100%	0%	0%
OK	99.2%	0.84%	0%

2. 카메라가 왼쪽에 있을 경우 N = 120 (시행 횟수)

제스처	인식률	불일치율	실패율
ZERO	100%	0%	0%
ONE	100%	0%	0%
TWO	90.8%	9.2%	0%
FIVE	99.2%	0.84%	0%
OK	98.3%	1.7%	0%

3. 카메라가 오른쪽에 있을 경우 N = 120 (시행 횟수)

제스처	인식률	불일치율	실패율
ZERO	100%	0%	0%
ONE	100%	0%	0%
TWO	90.8%	9.2%	0%
FIVE	99.2%	0.84%	0%
OK	85.8%	14.2%	0%

4. 카메라가 위에 있을 경우 N = 120 (시행 횟수)

제스처	인식률	불일치율	실패율
ZERO	95.8%	4.2%	0%
ONE	88.3%	11.7%	0%
TWO	84.2%	15.8%	0%
FIVE	95%	5%	0%
OK	82.5%	7%%	0%

1. 카메라가 아래에 설치되어 있는 경우 인식률이 가장 높음.
2. 왼쪽과 오른쪽에서 제스처를 인식하는 경우, 손 모양이 방향에 따라 바뀌는 ‘TWO’, ‘OK’의 인식률이 줄어듦.
3. 카메라가 위에 설치되어 있는 경우 불일치율이 다른 조건보다 높음. 키오스크의 높이가 높은 만큼 위에서 손을 바라보기 때문에 모양이 정확하지 않은 것으로 추정됨.

4) 결과 분석

실험 2. 거리에 따른 인식률 비교

1. 거리가 15cm 이내인 경우 N = 120 (시행 횟수)

제스처	인식률	불일치율	실패율
ZERO	98.3%	1.7%	0%
ONE	100%	0%	0%
TWO	99.2%	0.84%	0%
FIVE	100%	0%	0%
OK	99.2%	0.84%	0%

2. 거리가 15cm~30cm 인 경우 N = 120 (시행 횟수)

제스처	인식률	불일치율	실패율
ZERO	98.3%	0%	1.7%
ONE	97.5%	0%	2.5%
TWO	90.8%	9.2%	0%
FIVE	99.2%	0.84%	0%
OK	95.8%	1.7%	2.5%

3. 거리가 30cm 이상인 경우 N = 120 (시행 횟수)

제스처	인식률	불일치율	실패율
ZERO	78.46%	0.84%	11.7%
ONE	82.5%	1.7%	15.8%
TWO	85.8%	5%	9.2%
FIVE	97.46%	1.7%	0.84%
OK	76.6%	9.2%	14.2%

1. 거리가 15cm 이내인 경우 인식률이 가장 높음.
2. 거리가 멀어질수록 인식률이 떨어지고 실패율(손을 인식하지 못하는 경우)이 높아짐.
3. 특히 30cm 이상으로 거리가 멀어지면, 평균 인식률이 84.164%로, 현저히 낮아짐.

Part 7. 결론

1) 발전 가능성

2) 연구 소감

1) 발전 가능성

현재 사용자 손 제스처 인식 알고리즘은 단순 손 제스처 의미 인식에 그쳤지만, 추후 손가락의 상대적인 위치 및 영상 내 객체 연동을 고려하여 복잡한 순서도를 가진 키오스크에도 적용할 수 있는 제스처 인식 알고리즘 연구를 진행할 필요가 있다.

2) 연구 소감

이번 연구를 통해 키오스크 사용에 어려움을 겪는 노인들의 입장에서 생각해볼 수 있었다. 그러나 본 연구에서는 키오스크 기계와의 연동, 다양한 키오스크 UI 및 동작을 고려하지 않아 실현 가능성이 적다. 처음 계획과는 달리 결과물이 부족하여 아쉽지만 추후 기술력이 더해져 본 연구가 어느 분야이든 활용되기를 바란다.

감사합니다