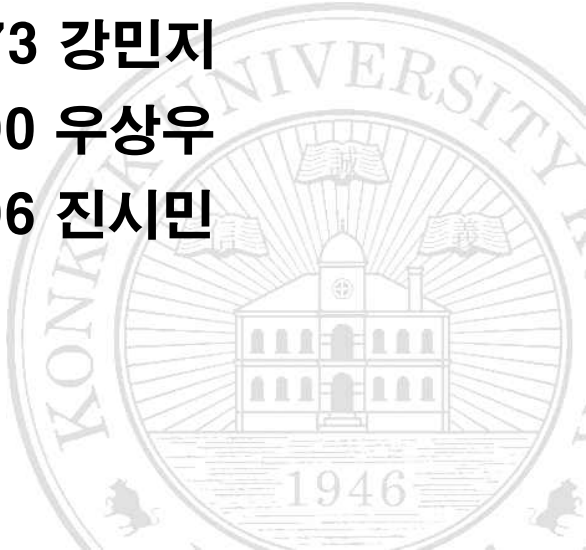
〈목요일 1조〉

전기전자심화설계및실습
〈최종텀프로젝트〉

담당교수 : 조용범 교수님
실험날짜 : 2020. 12. 18
조 : 목요일 1조
조원 : 201810773 강민지
201814090 우상우
201611396 진시민



1. Title

2. Name

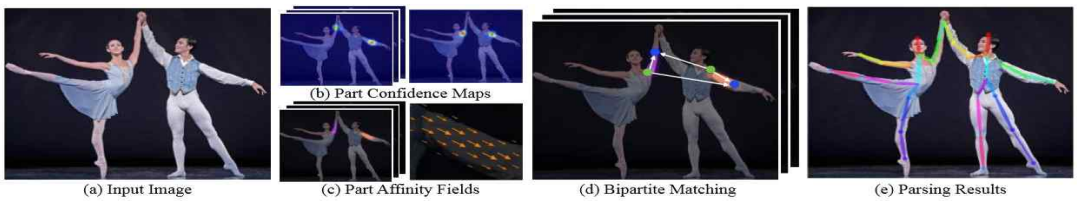
1조 강민지 201810773
1조 우상우 201814090
1조 진시민 201611396

3. Abstract

4. Background

A. OpenPose

[1] OpenPose



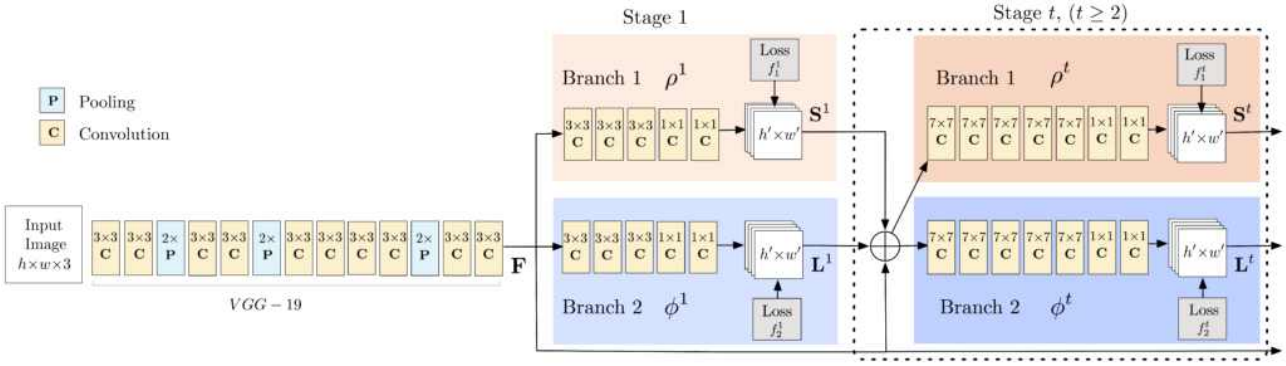
* Ouput에서 feature을 강조한 상태로 출력하게 된다.
* 첫 번째 output으로는 신체 부위 중 팔꿈치, 무릎 등에 사용된다.
* 반복되는 Stage에 따라 가지를 거쳐서 confidence map과 affinity field를 구하게 된다.
  - confidence map : 인간의 관절 구조 확인
  - affinity field : 추출된 관절 구조가 어떤 객체의 것인지 확인
* VGG는 CNN 망(신경망) 중 하나로, 성능에 비해 내부 구조가 비교적 간단하여 활용도가 높은 특성을 가지고 있다.

* 쉽게 말해서 처음에는 feature을 통해서 팔꿈치, 무릎, 어깨 등 인체에서 꺽이는 부분을 확인한다. 반복적인 stage를 통해 해당 부분이 인체와 비교하여 Keypoint와 이를 연결해서 인체 모델로 만든다.

[2] Pose-estimation (luvimperfection)이란?

* Pose Estimation은 Computer Vision의 한 분야로, 사물의 position과 orientation을 detect하고자 하는 분야입니다. 소위 KEypoint detection라고 불리는데, 사물을 특징지을 수 있는 'keypoint'의 위치를 detect하고자 하기 때문입니다.

[3] 구조

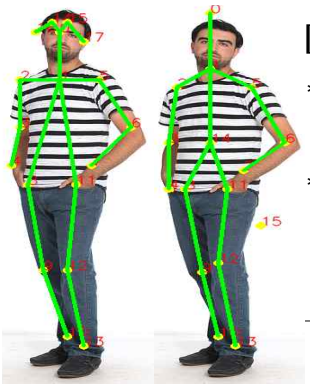


[4] 진행 과정

Step 1) 입력한 image나 video로부터 VGGNet을 이용하여 feature map을 생성합니다.

Step 2) Part Confidence Maps, Part Affinity Fields 출력

* CNN은 두 개의 branch로 구성되어 있는데, 첫 번째 branch는 Part Confidence Maps를 출력하고, 두 번째 branch는 Part Affinity Fields를 출력합니다.
* 첫 번째 branch는 특정 신체 부위 keypoint를 출력합니다. (무릎이나 팔꿈치와 같은 관절이 있을 것으로 예상되는 위치를 나타냅니다.)
* 두 번째 branch에서는 첫 번째 branch로 얻어낸 keypoint를 특정 관절끼리 연결하여 출력하도록 합니다.



[5] COCO / MPI 차이점

* COCO(좌측)

  눈과 귀까지 포함해서 총 18개의 keypoints를 표시합니다.

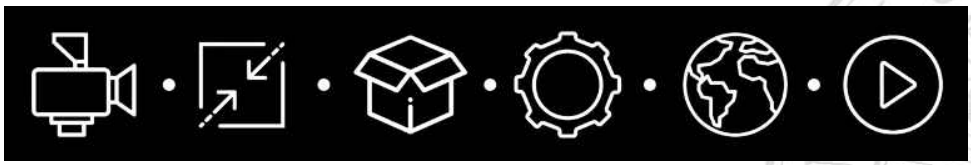
* MPII(우측)

  눈과 귀를 제외하고 허리까지 포함해서 총 15개의 keypoints를 표시합니다.

→ 자세한 자세 판단을 위해서 허리도 인식하는 MPII dataset 사용한다.

B. Python streamlink

Streaming이란 인터넷에서 데이터를 실시간 전송, 구현할 수 있게 하는 기술이다.



* '캡처->비디오 코덱 및 인코딩->패키징 및 프로토콜->인제스트 및 트랜스 코딩->전달->재생'
  위의 단계를 통하여 실시간 비디오를 확인할 수 있습니다.

* Streamlink는 다양한 서비스의 비디오 스트림을 VLC와 같은 비디오 플레이어로 파이프하는 CLI 유틸리티이다. Streamlink가 작동하는 방식은 스트림을 추출하고 전송하는 수단 일 뿐이며 재생은 외부 비디오 플레이어에 의해 수행된다.

```
>>> import  streamlink
>>> streams  =  streamlink . 스트림 ( "url" )
```

* 다음과 같은 명령어를 사용하여 플러그인을 찾고 url로부터 스트림을 추출하는데 사용한다.
* 반환된 값은 stream objects를 포함한 dict이다.

```
 >>> streams
{'best': <HLSStream('http://...')>,
 'high': <HLSStream('http://...')>,
 'low': <HLSStream('http://...')>,
 'medium': <HLSStream('http://...')>,
 'mobile': <HLSStream('http://...')>,
 'source': <HLSStream('http://...')>,
 'worst': <HLSStream('http://...')>}
```

* 미세한 조정을 원할 경우 session object를 사용하면 다양한 옵션을 설정할 수 있으며 스트림을 두 번 이상 추출할 때 효율적이다.

```
>>> from  streamlink  import  Streamlink
>>> session  =  Streamlink ()

>>> streams = session.streams("url")
```

* 일부 스트림을 추출한 결과에서 일부 데이터를 읽어올 수 있습니다.

```
>>> stream = streams["source"]
>>> fd = stream.open()
>>> data = fd.read(1024)
>>> fd.close()
```

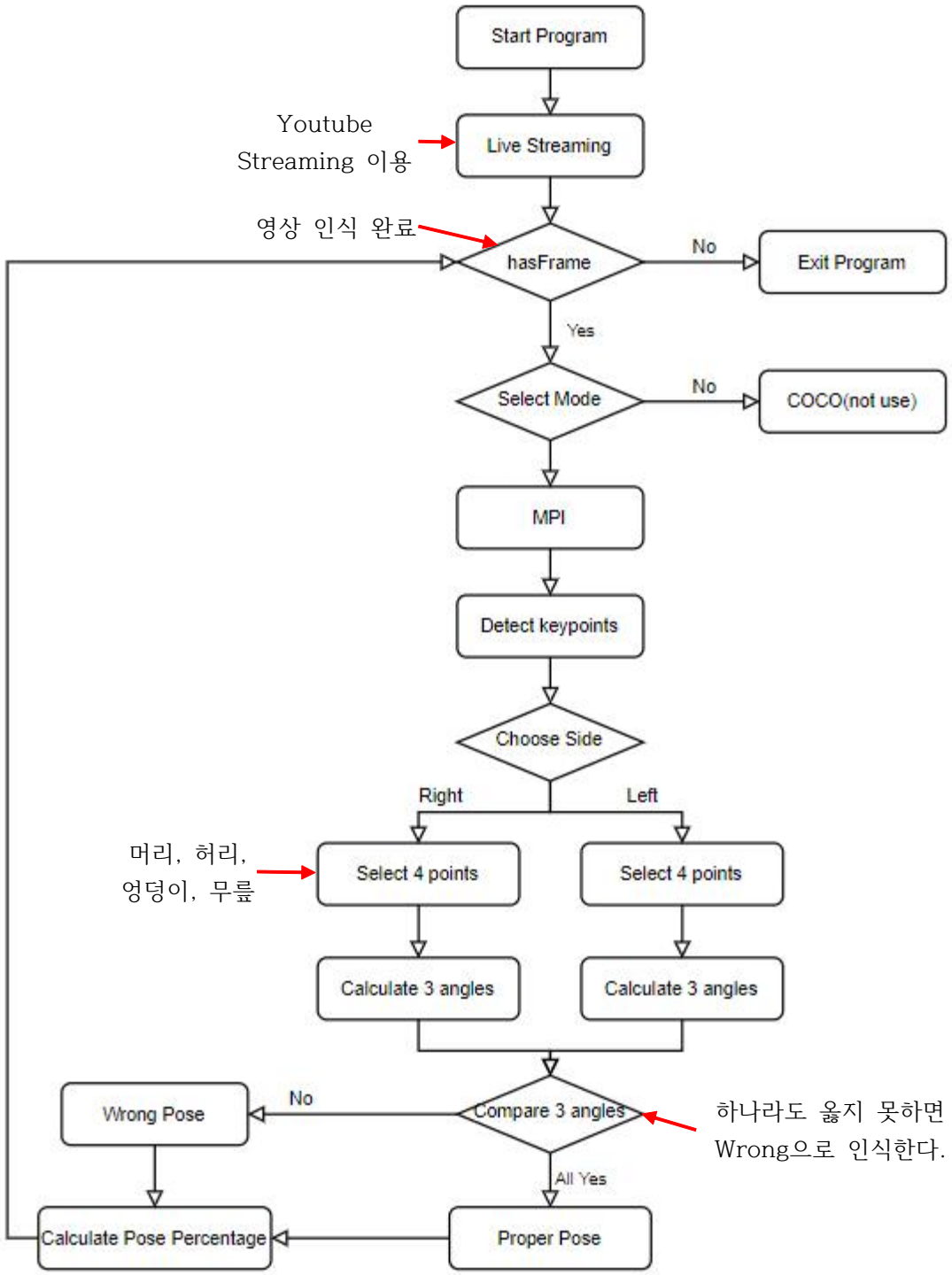
C. 바른 자세로 앉기

* 허리가 너무 뒤로 젖혀져있거나 앞으로 쏠리지 않도록 반듯하게 피는 것이 핵심입니다. 그래야 머리를 지탱하는 근육과 허리를 받쳐주는 근육에 무리가 덜 가기 때문입니다. 머리의 위치를 최대한 중력선과 가깝게 한다고 생각하시면 됩니다.

5. Experimental Result

1. 동작 구현 과정

A. Flow Chart



B. Python Code

(1) Setting for python

```
import cv2              # for image processing
import time             # for FPS
import numpy as np
import argparse         # for 인자
import streamlink       # for video streaming (python library)
import datetime         # for recording -txt file

url = 'https://youtu.be/myWD76xcp5g' # using live stream video -> use url to get the video file
streams = streamlink.streams(url)     # live stream video is saved in this 'streams'

# 인자
parser = argparse.ArgumentParser(description='Run keypoint detection')
parser.add_argument("--device", default="cpu", help="Device to inference on") # 거의 픽스해서 사용
side = 1 # left:1 / right:2 -> 앉는 자세별로 위치 조정
args = parser.parse_args()

MODE = "MPI" # 모드 픽스하기

if MODE is "COCO":
    protoFile = "pose/coco/pose_deploy_linevec.prototxt"
    weightsFile = "pose/coco/pose_iter_440000.caffemodel"
    nPoints = 18
    POSE_PAIRS = [[1,0],[1,2],[1,5],[2,3],[3,4],[5,6],[6,7],[1,8],[8,9],[9,10],[1,11],[11,12],[12,13],[0,14],[0,15],[14,16],[15,17]]

# MODE
elif MODE is "MPI" :
    protoFile = "pose/mpi/pose_deploy_linevec_faster_4_stages.prototxt"
    weightsFile = "pose/mpi/pose_iter_160000.caffemodel" # using trained weight model
    nPoints = 15                                         # MPI model's point
    POSE_PAIRS = [[0,1], [1,2], [2,3], [3,4], [1,5], [5,6], [6,7], [1,14], [14,8], [8,9], [9,10], [14,11], [11,12], [12,13] ]
    POSE_PAIRS_left = [ [0,1],[1,14],[14,11],[11,12] ]
    POSE_PAIRS_right = [ [0,1],[1,14],[14,8],[8,9] ]
    # head = 0, neck = 1, waist = 14, hip = 8
```

```
# for faster time -> reduce size of input
inWidth = 368
inHeight = 368
threshold = 0.34 # 적절한 쓰레시 값
net = cv2.dnn.readNetFromCaffe(protoFile, weightsFile) # 일단 신경망은 오픈소스 그대로 사용
if args.device == "cpu":
    net.setPreferableBackend(cv2.dnn.DNN_TARGET_CPU)
    print("Using CPU device")
elif args.device == "gpu":
    net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
    net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)
    print("Using GPU device")
```

* url : youtube 실시간 영상 주소 → streamlink.streams(url)로 실시간 영상을 입력한다.
* Mode에서 COCO 대신에 MPI를 사용한 이유?
    * COCO에서는 목에서 엉덩이 양쪽으로 직결되어있고, 추가적으로 얼굴 표정이 표시되어 있다. 그와 달리 MPI에서는 중간에 허리 중심이 keypoints로 표시되고, 머리 꼭대기는 표시되어있지만, 얼굴 표정은 표시되지 않아서, 불필요한 부분을 제거합니다.
* POSE_PAIRS_left/right : 자세 인식을 위해서 옆면에서 촬영하는데, 왼쪽과 오른쪽이냐에 따라서 출력 결과에 오차가 생깁니다. 또한, 머리-목-허리-엉덩이-무릎까지의 각도만 필요하기 때문에 나머지는 제거합니다.
* inWidth, inHeight, threshold 값 선정 기준 : 일정 기준의 정확성을 기준으로 최대한 빠른 속도
* CPU/GPU:빠른 속도를 지닌 GPU를 사용하려했지만, 지원되지 않아서, CPU 사용

(2) Video Input 처리, OpenPose 준비

```
#input_source = args.video_file         #사용자에게 input 받은 비디오 파일 사용
#cap = cv2.VideoCapture(input_source)
cap = cv2.VideoCapture(streams["360p"].url) # 비디오 스트림 사용
hasFrame, frame = cap.read()                    # 스트림 읽어서 적용
# 동영상 처리
while cv2.waitKey(1) < 0:
    t = time.time()               # 동영상 시작 시간
    hasFrame, frame = cap.read()
    frameCopy = np.copy(frame)
    #if not hasFrame:
    if 0xff == ord('q'):
        #cv2.waitKey()
        print("finished detection") # 탈출 확인
        cv2.destroyAllWindows()      # 다 close 하기
        break                        # 종료
```

```
    frameWidth = frame.shape[1]
    frameHeight = frame.shape[0]

    inpBlob = cv2.dnn.blobFromImage(frame, 1.0 / 255, (inWidth, inHeight),
                                    (0, 0, 0), swapRB=False, crop=False)
    net.setInput(inpBlob)
    output = net.forward()

    H = output.shape[2]
    W = output.shape[3]

    # 특징점들 초기화
    point_x = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
    point_y = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

    # Empty list to store the detected keypoints
    points = []
```

* 구한 영상을 인식하고 싶으면 input_source를 이용하고, 실시간으로 인식하고 싶으면 stream을 이용하면되고, read()를 통해 읽은 값들을 hasFrame과 frame에 저장한다.
* 'if 0xff == ord('q')' 구문을 통해 q를 입력하면 종료된다.
  (하지만 virtual box 이요의 한계점으로 사용이 되지 않는다.→강제 종료 이용)
* OpenPose(MPI)를 실행하며 필요한 저장소를 초기화하고, 계산을 위한 값들을 준비합니다.

(3) OpenPose 실행, 출력 결과를 통해 값 도출

```
for i in range(nPoints):
        # confidence map of corresponding body's part.
        probMap = output[0, i, :, :] # map checking point

        # Find global maxima of the probMap.
        minVal, prob, minLoc, point = cv2.minMaxLoc(probMap)

        # Scale the point to fit on the original image
        x = (frameWidth * point[0]) / W
        y = (frameHeight * point[1]) / H

        #  for문 동안 각 포인트들 매치 (프레임당 변화됨)
        point_x[i]=x
        point_y[i]=y
```

```
        if prob > threshold :
            if side == 1:
                if i in [0,11,12,14]:
                    cv2.circle(frameCopy, (int(x), int(y)), 8, (0, 255, 255), thickness=-1,
lineType=cv2.FILLED)
            elif side == 2:
                if i in [0,8,9,14]:
                    cv2.circle(frameCopy, (int(x), int(y)), 8, (0, 255, 255), thickness=-1,
lineType=cv2.FILLED)

            # Add the point to the list if the probability is greater than the threshold
            points.append((int(x), int(y)))
        else :
            points.append(None)

    # detecting angle between points
    def __angle_between(p1,p2):
        ang1=np.arctan2(*p1[::-1])
        ang2=np.arctan2(*p2[::-1])
        res=np.rad2deg((ang1-ang2)%(2*np.pi))
        return res

    def getAngle3P(p1,p2,p3):
        pt1=(point_x[p1]-point_x[p2], point_y[p1]-point_y[p2])
        pt2=(point_x[p3]-point_x[p2], point_y[p3]-point_y[p2])
        product=np.dot(pt1,pt2)
        normu=np.linalg.norm(pt1)
        normv=np.linalg.norm(pt2)
        cost=product/(normu*normv)
        res=np.rad2deg(np.arccos(cost))
        return res

    if side == 1:
        print("\nangle1:{}".format(getAngle3P(0,1,14)))      # head-neck-waist
        print("angle2:{}".format(getAngle3P(1,14,11)))       # neck-waist-hip
        print("angle3:{}\n".format(getAngle3P(14,11,12)))     # waist-hip-knee

    elif side == 2:
        print("\nangle1:{}".format(getAngle3P(0,1,14)))      # head-neck-waist
        print("angle2:{}".format(getAngle3P(1,14,8)))       # neck-waist-hip
        print("angle3:{}\n".format(getAngle3P(14,8,9)))      # waist-hip-knee
```

* OpenPose를 통해 나온 지점들을 구합니다.(point_x/y)
* 왼쪽/오른쪽에 따라 영상이나 사진에 저장할 지점을 지정합니다. (머리, 목, 허리, 엉덩이, 무릎)
* __angle_between으로 두 지점 사이의 벡터를 구하고, getAngle3P를 통해 두 벡터의 사이각을 cos공식을 이용하여 구합니다. (출력 결과는 0~180 각이 나옵니다.)
* 올바른 앉은 자세 판단을 위해서 필요한 3 각도를 출력합니다.

(4) 결과를 토대로 영상과 텍스트 파일 제작

```
# save output
f=open("sitting_output.txt",'w')
now=datetime.datetime.now()          # write the time when this is recorded
nowDate=now.strftime('%Y-%m-%d')
f.write('{}\n'.format(nowDate))
a=time.time()                        # current time
# ********************** 프레임에 skeleton 그림 ******************** #
    # Draw Skeleton
    if side == 1:
        for pair in POSE_PAIRS_left:
            partA = pair[0]
            partB = pair[1]

            if points[partA] and points[partB]:
                if (140 <= getAngle3P(0,1,14) <= 180) and (140 <= getAngle3P(1,14,11) <= 180)
and (80 <= getAngle3P(14,11,12) <= 130) :
                    cv2.putText(frame, "proper pose", (30,30),
cv2.FONT_HERSHEY_COMPLEX, .8, (255, 50, 0), 2, lineType=cv2.LINE_AA)
                    proper_steck=proper_steck+1
                else:
                    cv2.putText(frame, "wrong pose", (30,30), cv2.FONT_HERSHEY_COMPLEX,
.8, (255, 50, 0), 2, lineType=cv2.LINE_AA)
                    wrong_steck=wrong_steck+1
                percentage=(proper_steck)/(proper_steck+wrong_steck)*100
                cv2.line(frame, points[partA], points[partB], (0, 255, 255), 3, lineType
=cv2.LINE_AA)
                cv2.circle(frame, points[partA], 8, (0, 0, 255), thickness=-1, lineType
=cv2.FILLED)
                cv2.circle(frame, points[partB], 8, (0, 0, 255), thickness=-1, lineType
=cv2.FILLED)

    elif side == 2:
        for pair in POSE_PAIRS_right:
```

```
            partA = pair[0]
            partB = pair[1]

            if points[partA] and points[partB]:
                if (160 <= getAngle3P(0,1,14) <= 180) and (165 <= getAngle3P(1,14,8) <= 180) and (100 <= getAngle3P(14,8,9) <= 120) :
                    cv2.putText(frame, "proper pose", (30,30), cv2.FONT_HERSHEY_COMPLEX, .8, (255, 50, 0), 2, lineType=cv2.LINE_AA)
                else:
                    cv2.putText(frame, "wrong pose", (30,30), cv2.FONT_HERSHEY_COMPLEX, .8, (255, 50, 0), 2, lineType=cv2.LINE_AA)

                cv2.line(frame, points[partA], points[partB], (0, 255, 255), 3, lineType=cv2.LINE_AA)
                cv2.circle(frame, points[partA], 8, (0, 0, 255), thickness=-1, lineType=cv2.FILLED)
                cv2.circle(frame, points[partB], 8, (0, 0, 255), thickness=-1, lineType=cv2.FILLED)

    cv2.putText(frame, "time taken = {:.2f} sec".format(time.time() - t), (50, 150), cv2.FONT_HERSHEY_COMPLEX, .8, (255, 50, 0), 2, lineType=cv2.LINE_AA)

    # angle print
    cv2.putText(frame, "angle:{:.0f}".format(getAngle3P(0,1,14)), (int(point_x[1]), int(point_y[1])), cv2.FONT_HERSHEY_PLAIN, 0.8, (0, 0, 0), 2, lineType=cv2.LINE_AA)
    cv2.putText(frame, "angle:{:.0f}".format(getAngle3P(1,14,8)), (int(point_x[14]), int(point_y[14])), cv2.FONT_HERSHEY_PLAIN, 0.8, (0, 0, 0), 2, lineType=cv2.LINE_AA)
    cv2.putText(frame, "angle:{:.0f}".format(getAngle3P(14,8,9)), (int(point_x[8]), int(point_y[8])), cv2.FONT_HERSHEY_PLAIN, 0.8, (0, 0, 0), 2, lineType=cv2.LINE_AA)

    now=datetime.datetime.now()
    nowTime=now.strftime('%H:%M:%S')
    record_time = round(time.time()-a,2) # code run time
    f.write('\nNow time : {}\n'.format(nowTime))
    f.write('Now percentage : {}%\n'.format(percentage))
    f.write('Recoding Time : {}s\n'.format(record_time))

    vid_writer.write(frame)
cap.release()
vid_writer.release()
```

* 결과 저장할 수단 : 텍스트(txt), 영상(avi)
* txt 파일에는 현재 날짜와 시간, 현재까지 올바른 자세의 비율, 측정 시간을 입력합니다.
* 영상에서는 첫 번째, 왼쪽/오른쪽으로 분류합니다.(사용된 keypoints가 다르다.)
* 구한 3 각도인 angle1, angle2, angle3를 각각 140~180, 140~180, 80~130)을 기준으로 'proper' 인지, 'wrong'인지 판단합니다.

  (구간마다 범위를 넓게 설정한 이유: 각 사람의 특징과 주위 환경으로 인하여 인식된 지점에 오차가 발생합니다. 또한, 올바른 자세 측정에 요구되는 지점과 OpenPose를 통해 인식된 지점에 차이가 있어서 오차가 발생할 수 밖에 없습니다.)
* 'proper'로 인식할 때와 'wrong'으로 인식할 때의 횟수를 기준으로 측정 시간까지의 비율을 구합니다. → percentage(%)
* 자세 판단에 필요한 지점들과 그 지점들을 있는 직선을 frame에 저장합니다.

  → 해당 frame은 video에 저장되면서 종료됩니다.

C. Result

[1] Text File Result

```
 GNU nano 2.9.3                    sitting_output.txt

2020-12-17

Now time : 16:18:22
Now percentage : 100.0%
Recoding Time : 4.07s

Now time : 16:18:26
Now percentage : 100.0%
Recoding Time : 8.07s

Now time : 16:18:30
Now percentage : 100.0%
Recoding Time : 12.09s

Now time : 16:18:34
Now percentage : 100.0%
Recoding Time : 16.06s

Now time : 16:18:38
Now percentage : 100.0%
Recoding Time : 20.04s
```

* 측정된 시간을 확인하여 한번 실행할 때마다 4초 정도 소모된 것을 확인할 수 있다.
* 현재까지 측정된 자세는 모두 옳다고 판단되서 100%가 발생한다.
* 해당 부분은 txt 파일의 극소수 부분으로 1초 영상인 결과라도 매우 많은 결과 값들이 입력됩니다.

[2] Video File Result

* 유튜브의 live streaming을 통해서 영상을 실시간으로 OpenPose.py로 전송한다.
* 전송된 영상은 명령대로 자세를 판단하여, 옳은 자세의 비율을 최신화합니다.
* 현재 상태는 머리-목-허리:166 / 목-허리-엉덩이:150 / 허리-엉덩이-무릎:90 정도의 결과가 나와서 'Proper Pose'로 판단합니다.

[3] Problems

(1) OpenPose 실행 시간 : $fps \approx 4s$

→ 1 frame당 대략 4초의 시간이 소모되는데, 1초에 30 frames으로 매 frame마다 OpenPose를 실행하면 실시간이랑 인식 속도의 차이가 심각하게 발생한다.

(2) Error 발생

→ OpenPose 실행 과정에서 자세를 판단하는데 필요한 지점들을 전부 인식하지 못하거나, 잘못 인식해서, 실제 결과와는 다른 결과가 발생하는 경우가 있다.

* Why? 사람 신체와 주위 사물을 구분하지 못한다.(기대거나 색상으로 구분하기 힘들다.)
* Why? 영상에서 신체 전부를 확인해야 정상적인 값이 나온다.

[4] How to solve?

(1) OpenPose 실행 시간 단축 : fps↓

* How? CPU보다는 GPU 사용하여 속도를 향상시킨다. 프로젝트에서는 원격 접속으로 인하여 제한되서 실현되기 힘들다.
* How? Threshold 값을 조정한다. 정확성이 떨어지지만, 일정 정확성보다 낮아지지 않은 시점까지 계속 조절하여 속도를 높인다.
* How? 매 frame마다 OpenPose를 실행하지 않고, 실행이 끝난 시점에서의 영상을 OpenPose 실행한다.(기존에서는 실행하면 영상이 멈추지만, 영상이랑 OpenPose 실행이 같이 흐른다.)

(2) Error를 줄인다.

* How? 주위 사물과 구분되기 쉽게 의상을 입는다.
* How? 등받이, 팔걸이 등, 사람을 인식하는데 방해된 요소를 제거한다.

6. References

https://github.com/CMU-Perceptual-Computing-Lab/openpose
https://hanryang1125.tistory.com/2
https://m.blog.naver.com/shino1025/221607197982
https://m.blog.naver.com/sowon622/221474947843