Resize code 분석 보고서

박종명

목차

1.목적
2.코드 설명
3.전체 코드

1. 목적

이 코드의 목적은 주어진 비디오에서 원활한 분석을 위해 피사체의 크기를 적당한 사이즈로 변환하여 생성하는 것입니다. 이 프로그램은 포즈 감지를 위해 Mediapipe 툴을 사용하여 비디오 프레임에서 33개의 포즈 랜드마크(주요 신체 지점)를 추적합니다. 코드는 감지된 포즈 랜드마크를 기반으로 다양한 계산을 수행하여, 프레임의 크기를 조정하고 원하는 영역을 잘라내어 골프 스윙의 초기 및 최종 프레임을 식별하고 해당 비디오 클립을 생성합니다.

2. 코드 설명

코드는 resize.py, Function.py, Calc.py 세 파일로 분리했습니다. resize.py에는 main문이 있습니다. Function.py에는 영상 선택, 손목 변화 여부 확인, 영상 크기 조정, 초기와 최종 프레임 찾기 그리고 영상을 화면에 보여주는 함수들로 구성되어 있습니다. Calc.py에는 팔이 구부러진 각도와 두 손의 거리를 계산하는 함수가 있는데, 스윙 분석을 할 때 많이 사용되지만 시작과 종료 시점 조건을 찾기 위해서도 필요합니다.

-resize.py 코드 설명

```
import cv2
import mediapipe as mp
import numpy as np
import time
from moviepy.editor import *
import Function
import Calc
```

import cv2: OpenCV 라이브러리를 임포트합니다. 영상 및 이미지 처리를 위해 사용됩니다.

import mediapipe as mp: Mediapipe 라이브러리를 임포트합니다. 머신러닝 기반 미디어 처리 알고리즘을 포함하고 있으며, 주로 포즈, 얼굴, 손, 그리고 감지와 추적 등에 사용됩니다.

import numpy as np: NumPy 라이브러리를 임포트합니다. 과학적 계산을 위해 사용되는 파이썬 라이브러리로, 다차원 배열을 다루는데 유용합니다.

import time: time 모듈을 임포트합니다. 시간 관련 기능을 사용하기 위해 포함시키는 것으로 보입니다.

from moviepy.editor import *: moviepy 라이브러리에서 모든 것을 임포트합니다. 영상 편집을 위한 라이브러리로 사용됩니다.

import Function: Function.py 파일에서 정의된 함수를 사용하기 위해 모듈을 임포트합니다.

import Calc: Calc.py 파일에서 정의된 함수를 사용하기 위해 모듈을 임포트합니다.

```
def main():

    #variable
    pTime = 0
    pTime2 = 0
    mark = np.zeros((33, 3))
    resize_info = np.zeros(7)
    before_wrist = np.zeros((2, 3))

    init_frame = 0
    final_frame = 0

    count_init = 0
    max_finish = None
    state_finish = 0

    file = Function.user()

    mpPose = mp.solutions.pose
    pose = mpPose.Pose(
        model_complexity = 1,
        min_detection_confidence = 0.5,
        min_tracking_confidence = 0.5
    )
    mpDraw = mp.solutions.drawing_utils

    cap = cv2.VideoCapture(file[0])
```

Main 함수를 정의합니다. 각 변수들을 0으로 초기화 시켜 줍니다. 각 배열들을 생성 시켜줍니다.

file = Function.user(): Function.py 파일에 정의된 user 함수를 호출하고, 그 결과를 file 변수에 저장합니다. (#1에 추가설명)

mpPose = mp.solutions.pose: Mediapipe에서 제공하는 포즈 모델을 mpPose 변수에 저장합니다.

pose = mpPose.Pose(...): 포즈 모델(pose)을 초기화하고, 모델의 복잡도, 최소 탐지 신뢰도, 최소 추적 신뢰도를 설정합니다.

mpDraw = mp.solutions.drawing_utils: Mediapipe에서 제공하는 drawing_utils를 mpDraw 변수에 저장합니다.

cap = cv2.VideoCapture(file[0]): file 변수에 저장된 영상 파일 경로로부터 영상을 불러와 cap 변수에 저장합니다. 이제 해당 영상을 사용하여 추후 작업을 수행할 수 있습니다.

#1. Function에 user 함수

```
def user():

    print("Select input video\n")
    print("1. yphong0125")

    file_src = ""
    file_out = ""
    num = input()

    if num == '1':
        file_src = 'C:\\Users\\TG\\Desktop\\hong\\yphong0125.mp4'
        file_out = 'C:\\Users\\TG\\Desktop\\hong\\yphong0125_resize.mp4'
    else:
        print("input false")

    clip1 = "C:\\Users\\TG\\Desktop\\hong\\clip_start_to_end.mp4"

    return file_src, file_out, clip1
```

user() 함수는 사용자에게 영상 파일을 선택하도록 안내하고, 선택한 파일의 경로를 반환하는 함수입니다. 코드에서는 사용자가 1을 입력하면 특정 영상 파일에 대한 경로 정보를 설정하고, 1 이외의 값이 입력되면 "input false"를 출력합니다. 이후에는 두 개의 변수(file_src, file_out)와 문자열 변수 clip1이 초기화됩니다.

```
while True:
    success, img = cap.read()  # Import one image(frame)
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Recolor image to RGB(becasue of OpenCV characteristic)
    results = pose.process(imgRGB)  # Detect 33 pose landmark using MediaPipe
    curr_frame = cap.get(cv2.CAP_PROP_POS_FRAMES)  # Get frame number

    if results.pose_landmarks:  # Check if any landmark are found
        mpDraw.draw_landmarks(img, results.pose_landmarks, mpPose.POSE_CONNECTIONS)  # Draw Pose landmarks(line) on image
        for id, lm in enumerate(results.pose_landmarks.landmark):  # In one frame, this 'for loop' loops 33 times.
            h, w, c = img.shape  # Get the Height(y), Width(x), depth from Camera(z) of the input image(frame)
            cx, cy = int(lm.x * w), int(lm.y * h)
            cv2.circle(img, (cx, cy), 5, (255, 0, 0), cv2.FILLED)  # Draw circles on landmark coordinate
            mark[id][0] = lm.x
            mark[id][1] = lm.y
            mark[id][2] = lm.z
```

첫 번째 루프 시작 부분입니다. 첫 번째 루프는 영상 시작 프레임과 종료 프레임을 찾고, 두 번째 루프는 분석 영상을 만들기 위해 존재합니다.

```
distance_wrist = Calc.distance(mark[15][0], mark[15][1], mark[16][0], mark[16][1])
lba = Calc.bend(h, w, mark[15][1], mark[13][1], mark[11][1], mark[15][0], mark[13][0], mark[11][0])
rba = Calc.bend(h, w, mark[16][1], mark[14][1], mark[12][1], mark[16][0], mark[14][0], mark[12][0])
```

두 손목 사이 거리, 왼팔이 구부러진 각도, 오른팔이 구부러진 각도를 계산합니다. 계산식의 전체코드(Calc.py)는 #2 에서 볼 수 있습니다.

```
change_wrist = Function.check_change_wrist(before_wrist, mark[15:17])
before_wrist[0] = mark[15]
before_wrist[1] = mark[16]
```

시작 프레임을 정하는 조건 중 손목 위치 변화량이 있도록 했습니다.

Function.check_change_wrist 함수는 이전 손목 좌표와 현재 손목 좌표를 비교하여 손목의 변화 여부를 판단합니다. 손 목의 변화 여부를 체크하고, 다음 프레임에서 손목 좌표를 업데이트하여 두 프레임 간의 변화를 추적하는 작업이 이루어집니다.(#3에 추가 설명)

#2. Calc.py

```
import math


def bend(h, w, wrist_y, elbow_y, shoulder_y, wrist_x, elbow_x, shoulder_x):
    angle = abs(math.atan2(elbow_y*h - shoulder_y*h, shoulder_x*w - elbow_x*w)
                - math.atan2(elbow_y*h - wrist_y*h, wrist_x*w - elbow_x*w))*180/math.pi
    if angle >= 180:
        angle -= 180
    else:
        angle = 180 - angle

    return angle


def distance(left_x, left_y, right_x, right_y):
    result = math.sqrt(math.pow(right_x - left_x, _y: 2) + math.pow(right_y - left_y, _y: 2))

    return result
```

두 가지 함수, bend와 distance 의 정의 입니다. 이 두 함수는 포즈 랜드마크의 좌표 정보를 기반으로 각도와 거리를 계산하는데 사용됩니다.

Bend 함수는 각도를 계산하여 결과를 반환합니다. 계산 과정은 다음과 같습니다. 두 선분 간의 각도를 계산하

는 방법으로, math.atan2() 함수를 사용합니다. 이 함수는 아크탄젠트(역탄젠트) 값을 계산하며, 각도의 단위는 라디안(rad)입니다. 먼저, 어깨에서 팔꿈치까지의 선분과 팔꿈치에서 손목까지의 선분 간의 각도를 계산합니다. 이 때, 좌표값은 이미지 크기 h와 w로 조정됩니다. 각도를 구하기 위해 math.atan2(elbow_y*h - shoulder_y*h, shoulder_x*w - elbow_x*w)와 math.atan2(elbow_y*h - wrist_y*h, wrist_x*w - elbow_x*w)를 계산하고, 두 각도의 차를 구합니다. 최종적으로, 각도를 180도로 정규화하여 결과를 반환합니다.

Distance 함수는 두 점 유클리드 거리를 계산하여 결과를 반환합니다.

#3. Function. Check _change _wrist 함수

```
def check_change_wrist(before, after):
    count = 0
    change = 0

    for row in range(2):
        for col in range(3):
            change = before[row][col] - after[row][col]
            if abs(change) < 0.04:
                count += 1

    if count == 6:
        return 1
    else:
        return 0
```

이 함수는 이전 손목 좌표와 현재 손목 좌표를 비교하여 손목의 변화 여부를 판단하는 함수입니다. 이 함수는 두 개의 2x3 배열인 before와 after를 인자로 받습니다. 각 배열은 손목의 x, y, z 좌표를 나타냅니다.

if abs(change) 〈 0.04: 변화 값의 절대값이 0.04보다 작은 경우, 손목의 변화가 없다고 판단합니다. 이 값은 사용자가 적절하게 조정할 수 있는 임계 값으로, 좌표 값의 작은 변화를 무시하기 위한 목적입니다.

count += 1: 변화가 발생한 좌표의 개수를 증가시킵니다.

if count == 6: 변화가 발생한 좌표 개수가 6인 경우, 즉 양쪽 손목의 x, y, z 좌표 모두에서 변화가 발생한 경우에 손목의 변화로 판단합니다.

이유는 for row in range(2)와 for col in range(3) 반복문을 통해 양쪽 손목의 모든 좌표를 비교하기 때문에, count가 6이면 모든 좌표에서 변화가 발생한 것입니다.

return 1: 손목의 변화가 있으면 1을 반환합니다.

```
if init_frame == 0:
    if Function.find_init_frame(distance_wrist, mark[23:25], lba, rba, change_wrist):
        count_init += 1

    else:
        count_init = 0

    if count_init == 20:
        init_frame = curr_frame
        resize_info = Function.resize(h, w, mark[0][1], mark[31][1], mark[23][0], mark[24][0])
```

Function.find_init_frame() 함수를 호출하여 초기 스윙 프레임을 찾는데 사용합니다. (#4. 상세설명)

count_init 변수가 20이 되었을 때, 즉 연속적으로 20번의 프레임에서 Function.find_init_frame() 함수가 참을 반환한 경우에 실행됩니다.

Function.resize() 함수를 호출하여 이미지 리사이징에 필요한 정보를 설정합니다. (#5. 상세설명)

위의 코드 블록은 초기 스윙 프레임을 찾고, 초기 스윙 프레임에 따른 리사이징 정보를 설정하는 작업을 수행합니다. count_init 변수는 Function.find_init_frame() 함수가 연속적으로 참을 반환한 횟수를 나타내며, 이 횟수가 20이 되면 init_frame 변수에 현재 프레임 번호가 할당되어 초기 스윙 프레임이 설정됩니다. 이후 리사이징 정보를 활용하여 영상 처리 작업을 수행할 수 있습니다.

#4. Function. Find _init _frame() 함수

```
def find_init_frame(distance_wrist, hip, lba, rba, change_wrist):
    count = 0

    if distance_wrist < abs(hip[0][0] - hip[1][0]):
        if lba < 30 and rba < 30:
            if change_wrist:
                count = 1

    return count
```

이 함수는 손목 사이 거리가 엉덩이 사이의 거리보다 작고 왼팔 오른팔의 각도가 30도 미만인 경우 손목의 위치가 변경되면 count가 1이 된다. 이때를 초기 스윙 프레임을 찾았음을 의미한다.

#5. Function. resize() 함수

```python
def resize(h, w, nose, foot, left, right):
    height = foot - nose
    width = abs(left - right)

    ratio = int(1080 * 0.7 * w / (h * height))
    resize_x = int(1080 * 0.7 * w / (h * height))  # output_y = 1080
    resize_y = int(1080 * 0.7 / height)

    img_left = int(1080 * 0.7 * w / (h * height) * (right - 3 * width))
    img_right = int(1080 * 0.7 * w / (h * height) * (left + 3 * width))
    img_top = int(1080 * (0.7 * nose / height - 0.2))
    img_bottom = int(1080 * (0.7 * foot / height + 0.1))

    #exception
    if img_left <= 0:
        img_left = 0
    if img_right >= resize_x:
        img_right = resize_x
    if img_top <= 0:
        img_top = 0
    if img_bottom >= resize_y:
        img_bottom = resize_y

    return ratio, resize_x, resize_y, img_left, img_right, img_top, img_bottom
```

이 함수는 이미지 크기를 조정하는데 사용되는 매개변수들을 입력 받고, 조정된 이미지의 크기와 위치를 결정하는 여러 변수들을 계산하여 반환합니다.

이미지 크기를 조정하기 위해 가로와 세로에 1080 * 0.7과 height를 고려하여 계산합니다. 그리고 출력 이미지에서 얼굴의 왼쪽 오른쪽 끝과 얼굴 상단과 하단의 위치를 고려하여 위치 조정 계산을 합니다.

출력 이미지가 왼쪽 끝, 오른쪽 끝, 상단, 하단에 겹치는 부분이 발생하지 않도록 예외 처리를 합니다. img_left와 img_right는 resize_x를 벗어나면 안 되며, img_top과 img_bottom은 resize_y를 벗어나면 안 됩니다. 따라서, 해당 조건을 만족하지 않으면 값을 조정하여 출력합니다.

```python
else:
    check = Function.find_final_frame(mark[11:13], mark[13:15], mark[15:17], state_finish, max_finish)

    if check == 1:
        final_frame = curr_frame
        max_finish = mark[13][0]
        state_finish = 1

    elif check == 2:
        break

    if curr_frame == cap.get(cv2.CAP_PROP_FRAME_COUNT):
        final_frame = curr_frame
        break
```

Function.find_final_frame() 함수는 어깨, 팔꿈치, 손목의 랜드마크 정보를 사용하여 스윙 동작의 최종 프레임을 찾아냅니다. (#5. 상세 설명) 그리고 최종 프레임을 찾으면 해당 루프를 종료하고, 찾지 못한 경우 다음 프레임으로 계속 반복하게 됩니다.

#5. Function. find _final _frame() 함수

```
def find_final_frame(shoulder, elbow, wrist, state, max_finish):
    result = 0

    if state != 2:
        if shoulder[0][0] < shoulder[1][0]:
            if (wrist[0][1] + wrist[1][1]) / 2 < (shoulder[0][1] + shoulder[1][1]) / 2:
                if wrist[0][1] < elbow[0][1] and wrist[1][1] < elbow[1][1]:
                    if wrist[0][0] < elbow[1][0] and wrist[1][0] < elbow[1][0]:
                        if max_finish is None or elbow[0][0] < max_finish:
                            result = 1

        if max_finish is not None:
            if shoulder[0][0] > shoulder[1][0] and wrist[0][1] > shoulder[0][1]:
                result = 2

    return result
```

이 함수는 주어진 랜드마크들을 기반으로 스윙 동작의 최종 스윙 프레임을 찾아내고, 해당 프레임 번호를 반환합니다. 만약 최종 스윙 프레임을 찾지 못한 경우 0을 반환하며, 최종 스윙 프레임을 찾은 경우 1 또는 2를 반환합니다. 1은 현재 프레임을 최종 스윙 프레임으로 판별한 경우이고, 2는 이미 이전에 찾은 최종 스윙 프레임보다 더 오른쪽에 있는 경우를 의미합니다.

```
    cTime = time.time()
    fps = 1 / (cTime - pTime)
    pTime = cTime
    Function.show(img, cap)

cv2.destroyWindow("Image")
```

전체적으로 이 코드는 비디오 프레임을 처리하면서 프레임 속도를 계산하고 이미지를 화면에 표시합니다. 프로그램이 끝나면 "Image" 창을 닫아 프로그램이 깔끔하게 종료될 수 있도록 합니다.

```
"""second loop"""
cap2 = cv2.VideoCapture(file[0])
cap2.set(cv2.CAP_PROP_POS_FRAMES, init_frame) # start at initial swing frame
fourcc = cv2.VideoWriter_fourcc(*'mp4v') #select codec
clip_start_to_end = cv2.VideoWriter(file[2], fourcc, 30.0,
                                    (resize_info[4]-resize_info[3], resize_info[6]-resize_info[5]))
```

이 코드는 마지막 스윙 프레임부터 비디오를 처리하기 위해 두 번째 루프를 설정합니다. 입력 비디오에서 프레임을 읽기 위해 새로운 비디오 캡처 개체를 만들고, 시작 프레임을 최종 스윙 프레임으로 설정하고, 지정된

코덱과 프레임 속도를 사용하여 처리된 프레임을 출력 비디오 파일에 저장하기 위해 새 비디오 작성기 개체를 만듭니다.

```
while True:
    success, img2 = cap2.read()
    imgRGB2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)
    results2 = pose.process(imgRGB2)
    curr_frame = cap2.get(cv2.CAP_PROP_POS_FRAMES)
```

이 코드는 무한 루프를 사용하여 비디오를 계속해서 읽고, 이미지 데이터를 RGB 형식으로 변환한 후 포즈 랜드마크를 감지합니다. 프레임 번호를 계속 추적하면서 비디오의 모든 프레임을 처리합니다.

```
img2 = cv2.resize(img2, (resize_info[1], resize_info[2]))  # Resize output video
output = img2[resize_info[5]: resize_info[6], resize_info[3]: resize_info[4]]
```

이 코드는 출력 비디오를 지정된 크기로 리사이즈하고, 해당 비디오에서 관심 영역만을 추출하여 output에 저장합니다.

```
        if results2.pose_landmarks:
            clip_start_to_end.write(output)  # Save the frames as a sequence(video)S
            if curr_frame >= final_frame:
                clip_start_to_end.release()  # Quit saving the frames
                break

        cTime2 = time.time()
        fps2 = 1 / (cTime2 - pTime2)
        pTime2 = cTime2
        Function.show(output, cap2)

    """clip videos -> output video"""
    clip_start_to_end = VideoFileClip(file[2])  # import clip video
    clip_start_to_end.write_videofile(file[1])  # Combine multiple clip videos


main()
```

이 코드는 두 번째 루프에서 클립 비디오를 저장하고, 저장된 클립 비디오를 하나의 출력 비디오 파일로 합치는 작업을 수행합니다.

3. 전체 코드

3-1. resize.py

```python
import cv2
import mediapipe as mp
import numpy as np
import time
from moviepy.editor import *
import Function
import Calc

def main():

    #variable
    pTime = 0
    pTime2 = 0
    mark = np.zeros((33, 3))
    resize_info = np.zeros(7)
    before_wrist = np.zeros((2, 3))

    init_frame = 0
    final_frame = 0

    count_init = 0
    max_finish = None
    state_finish = 0

    file = Function.user()

    mpPose = mp.solutions.pose
    pose = mpPose.Pose(
        model_complexity = 1,
        min_detection_confidence = 0.5,
        min_tracking_confidence = 0.5
    )
    mpDraw = mp.solutions.drawing_utils
```

```
cap = cv2.VideoCapture(file[0])

while True:
    success, img = cap.read()  # Import one image(frame)
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Recolor image to RGB(becasue of OpenCV characteristic)
    results = pose.process(imgRGB)  # Detect 33 pose landmark using MediaPipe
    curr_frame = cap.get(cv2.CAP_PROP_POS_FRAMES)  # Get frame number

    if results.pose_landmarks:  # Check if any landmark are found
        mpDraw.draw_landmarks(img, results.pose_landmarks, mpPose.POSE_CONNECTIONS)  # Draw Pose landmarks(line) on image
        for id, lm in enumerate(results.pose_landmarks.landmark):  # In one frame, this 'for loop' loops 33 times.
            h, w, c = img.shape  # Get the Height(y), Width(x), depth from Camera(z) of the input image(frame)
            cx, cy = int(lm.x * w), int(lm.y * h)
            cv2.circle(img, (cx, cy), 5, (255, 0, 0), cv2.FILLED)  # Draw circles on landmark coordinate
            mark[id][0] = lm.x
            mark[id][1] = lm.y
            mark[id][2] = lm.z

        distance_wrist = Calc.distance(mark[15][0], mark[15][1], mark[16][0], mark[16][1])
        lba = Calc.bend(h, w, mark[15][1], mark[13][1], mark[11][1], mark[15][0], mark[13][0], mark[11][0])
        rba = Calc.bend(h, w, mark[16][1], mark[14][1], mark[12][1], mark[16][0], mark[14][0], mark[12][0])
        change_wrist = Function.check_change_wrist(before_wrist, mark[15:17])
        before_wrist[0] = mark[15]
        before_wrist[1] = mark[16]

        if init_frame == 0:
            if Function.find_init_frame(distance_wrist, mark[23:25], lba, rba, change_wrist):
                count_init += 1

            else:
                count_init = 0

            if count_init == 20:
                init_frame = curr_frame
                resize_info = Function.resize(h, w, mark[0][1], mark[31][1], mark[23][0], mark[24][0])
```

```
            else:
                count_init = 0

            if count_init == 20:
                init_frame = curr_frame
                resize_info = Function.resize(h, w, mark[0][1], mark[31][1], mark[23][0], mark[24][0])

        else:
            check = Function.find_final_frame(mark[11:13], mark[13:15], mark[15:17], state_finish, max_finish)

            if check == 1:
                final_frame = curr_frame
                max_finish = mark[13][0]
                state_finish = 1

            elif check == 2:
                break

            if curr_frame == cap.get(cv2.CAP_PROP_FRAME_COUNT):
                final_frame = curr_frame
                break

    cTime = time.time()
    fps = 1 / (cTime - pTime)
    pTime = cTime
    Function.show(img, cap)

cv2.destroyWindow("Image")

"""second loop"""
cap2 = cv2.VideoCapture(file[0])
cap2.set(cv2.CAP_PROP_POS_FRAMES, init_frame) # start at initial swing frame
fourcc = cv2.VideoWriter_fourcc(*'mp4v') #select codec
clip_start_to_end = cv2.VideoWriter(file[2], fourcc, 30.0,
                                    (resize_info[4]-resize_info[3], resize_info[6]-resize_info[5]))
```

```
    while True:
        success, img2 = cap2.read()
        imgRGB2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)
        results2 = pose.process(imgRGB2)
        curr_frame = cap2.get(cv2.CAP_PROP_POS_FRAMES)

        img2 = cv2.resize(img2, (resize_info[1], resize_info[2]))  # Resize output video
        output = img2[resize_info[5]: resize_info[6], resize_info[3]: resize_info[4]]

        if results2.pose_landmarks:
            clip_start_to_end.write(output)  # Save the frames as a sequence(video)S
            if curr_frame >= final_frame:
                clip_start_to_end.release()  # Quit saving the frames
                break

        cTime2 = time.time()
        fps2 = 1 / (cTime2 - pTime2)
        pTime2 = cTime2
        Function.show(output, cap2)

    """clip videos -> output video"""
    clip_start_to_end = VideoFileClip(file[2])  # import clip video
    clip_start_to_end.write_videofile(file[1])  # Combine multiple clip videos


main()
```

3-2. Function.py

```python
import cv2


def user():

    print("Select input video\n")
    print("1. yphong0125")

    file_src = ""
    file_out = ""
    num = input()

    if num == '1':
        file_src = 'C:\\Users\\TG\\Desktop\\hong\\yphong0125.mp4'
        file_out = 'C:\\Users\\TG\\Desktop\\hong\\yphong0125_resize.mp4'
    else:
        print("input false")

    clip1 = "C:\\Users\\TG\\Desktop\\hong\\clip_start_to_end.mp4"

    return file_src, file_out, clip1
```

```python
def check_change_wrist(before, after):
    count = 0
    change = 0

    for row in range(2):
        for col in range(3):
            change = before[row][col] - after[row][col]
            if abs(change) < 0.04:
                count += 1

    if count == 6:
        return 1
    else:
        return 0


def resize(h, w, nose, foot, left, right):
    height = foot - nose
    width = abs(left - right)

    ratio = int(1080 * 0.7 * w / (h * height))
    resize_x = int(1080 * 0.7 * w / (h * height))  # output_y = 1080
    resize_y = int(1080 * 0.7 / height)

    img_left = int(1080 * 0.7 * w / (h * height) * (right - 3 * width))
    img_right = int(1080 * 0.7 * w / (h * height) * (left + 3 * width))
    img_top = int(1080 * (0.7 * nose / height - 0.2))
    img_bottom = int(1080 * (0.7 * foot / height + 0.1))
```

```python
    #exception
    if img_left <= 0:
        img_left = 0
    if img_right >= resize_x:
        img_right = resize_x
    if img_top <= 0:
        img_top = 0
    if img_bottom >= resize_y:
        img_bottom = resize_y

    return ratio, resize_x, resize_y, img_left, img_right, img_top, img_bottom
```

```python
def find_final_frame(shoulder, elbow, wrist, state, max_finish):
    result = 0

    if state != 2:
        if shoulder[0][0] < shoulder[1][0]:
            if (wrist[0][1] + wrist[1][1]) / 2 < (shoulder[0][1] + shoulder[1][1]) / 2:
                if wrist[0][1] < elbow[0][1] and wrist[1][1] < elbow[1][1]:
                    if wrist[0][0] < elbow[1][0] and wrist[1][0] < elbow[1][0]:
                        if max_finish is None or elbow[0][0] < max_finish:
                            result = 1

        if max_finish is not None:
            if shoulder[0][0] > shoulder[1][0] and wrist[0][1] > shoulder[0][1]:
                result = 2

    return result
```

```python
def show(img, cap):
    cv2.putText(img, str(cap.get(cv2.CAP_PROP_POS_FRAMES)), (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 3)
    # Display information during analysis only
    cv2.imshow("Image", img)  # Show image during analysis
    cv2.waitKey(1)  # No stop during analysis
```

3-3. Calc.py

```python
import math


def bend(h, w, wrist_y, elbow_y, shoulder_y, wrist_x, elbow_x, shoulder_x):
    angle = abs(math.atan2(elbow_y*h - shoulder_y*h, shoulder_x*w - elbow_x*w)
                - math.atan2(elbow_y*h - wrist_y*h, wrist_x*w - elbow_x*w))*180/math.pi
    if angle >= 180:
        angle -= 180
    else:
        angle = 180 - angle

    return angle


def distance(left_x, left_y, right_x, right_y):
    result = math.sqrt(math.pow(right_x - left_x, _y: 2) + math.pow(right_y - left_y, _y: 2))

    return result
```