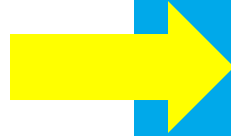


# Landmarks 전송 방식

파일 읽기/쓰기를 이용한  
Landmarks 전송



소켓통신을 이용한  
Landmarks 전송

# Landmarks 변환 함수

```
# -----  
# 랜드마크 추출 함수  
# 정규 표현식(x, y, z) 추출  
# landmarks에서 x, y, z만 추출  
# ([+-]?\d+\.\d+) 부분이 각각의 x, y, z 값을 추출하는데 사용  
# - 또는 + 부호의 유무와 상관없이 소수점이 포함된 숫자를 추출  
pattern = r"landmark {\s+x: ([+-]?\d+\.\d+)\s+y: ([+-]?\d+\.\d+)\s+z: ([+-]?\d+\.\d+)"  
  
def transformer(data, part):  
    # 랜드마크 전처리  
    matches_list = []  
  
    # 전처리 전의 데이터를 matches_list에 하나씩 저장  
    for i in range(len(data)):  
        matches_list.append(re.findall(pattern, data[i]))  
  
    # face 제외하고 일단 사전형 생성  
    output_data = {part[0]: [], part[1]: [], part[2]: []}  
  
    # left - right - pose 순  
    for i, matches in enumerate(matches_list):  
        for _, match in enumerate(matches):  
            x, y, z = map(float, match)  
            output_data[part[i]].append({"x": round(x, 3), "y": round(y, 3), "z": round(z, 3)})  
  
    output_json = json.dumps(output_data, indent=2, ensure_ascii=False)  
  
    return output_json
```

Mediapipe를 통해 Landmarks 추출 후 Json 형식 변환

# 소켓 통신

The image shows a dual-screen development environment. The left screen displays a Python script named `server.py` in a code editor. The script is part of a project named `mediapipe_socket` and is located in the `transformer` directory. The code includes comments in Korean and Python code for processing face data. Below the code editor, there is a `PROBLEMS` panel showing four errors related to camera recognition. At the bottom of the left screen, a window titled `MediaPipe Holistic` shows a video feed of a person's face with a 3D mesh overlay, indicating face tracking.

The right screen displays a C# script named `client_notclass.cs` in a code editor. The script is also part of the `mediapipe_socket` project and is located in the `client_notclass.cs` directory. The code includes comments in Korean and C# code for connecting to a server via sockets. Below the code editor, there is a `TERMINAL` panel showing the output of the program, which is a JSON object representing face pose data.

```
server.py
19 # 랜덤하게 가져옴
20 matches_list = []
21
22 # 전처리 전의 데이터를 matches_list에 하나씩 저장
23 for i in range(len(data)):
24     matches_list.append(re.findall(pattern, data[i]))
25
26 # face 제외하고 일단 사전형 생성
27 output_data = {part[0]: [], part[1]: [], part[2]: []}
28
29 # left - right - pose 순
```

```
client_notclass.cs
1 using System;
2 using System.Net;
3 using System.Net.Sockets;
4 using System.Text;
5
6 // 서버의 IP 주소와 포트 번호 설정
7 string serverIP = "127.0.0.1"; // 서버의 실제 IP 주소로 대체
8 int serverPort = 25001; // 파이썬 서버와 동일한 포트 번호
9
10 // try
11 // {
12 //     서버에 연결
13 TcpClient client = new TcpClient(serverIP, serverPort);
```

```
{
  "left": [],
  "right": [],
  "pose": [
    {
      "x": -0.015,
      "y": -0.576,
      "z": -0.334
    },
    {
      "x": -0.001,
      "y": -0.613,
      "z": -0.326
    },
    {
      "x": -0.002,
      "y": -0.613,
      "z": -0.328
    },
    {
      "x": -0.001,
      "y": -0.613,
      "z": -0.326
    },
    {
      "x": -0.04,
      "y": -0.614,
      "z": -0.326
    },
    {
      "x": -0.038,
      "y": -0.615,
      "z": -0.326
    }
  ]
}
```

# 유니티 데이터 수신

## Unity에서 소켓통신을 통해 데이터 수신

MediaPipe에서 보내주는 데이터를 수신하고 관리할 클래스 생성

Json 형식의 데이터를 유니티에서 활용 가능한 Vector3[] 형태로 변환

유니티의 Update() 함수가 호출될 때마다 데이터 수신, 전처리 과정이 이루어지도록 함

```
[System.Serializable]
3 references
public class Data
{
    public Vector3[] pose;
    public Vector3[] face;
    public Vector3[] left;
    public Vector3[] right;
}
```

```
private void Update()
{
    Load();
    preprocessing();
}
```

# 데이터 처리

## MediaPipe 좌표 대칭 및 스케일 변환

좌표계의 방향이 달라 유니티에서 그대로 사용하기엔 부적합

$(x, y, z) \Rightarrow (x, -y, -z)$  로 변환하여 사용자가 보는데 불편함을 해소

MediaPipe에서 z 좌표는 x, y좌표와는 약간 다른 스케일을 가지고 있기 때문에 사용자가 z 스케일 정도를 조절할 수 있도록 설정

```
private void Load()
{
    bytesRead = stream.Read(buffer, 0, buffer.Length);
    dataReceived = Encoding.UTF8.GetString(buffer, 0, bytesRead);
    //Debug.Log(dataReceived);

    // 받은 데이터의 길이가 5보다 작으면 루프 종료
    if (dataReceived.Length < 5)
        Debug.Log("Received data is too short. please check data.");

    data = JsonUtility.FromJson<Data>(dataReceived);
}
```

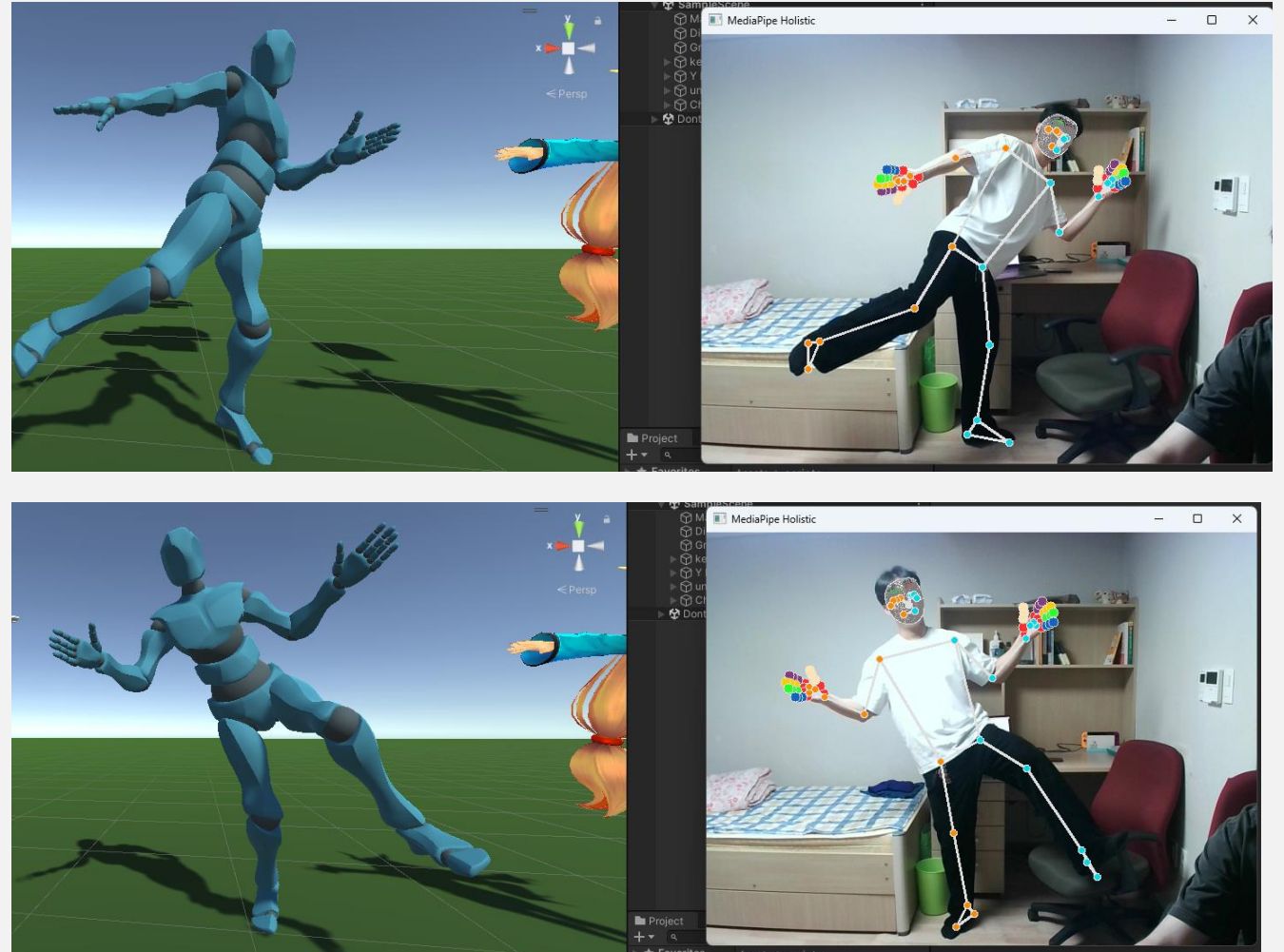
```
public void preprocessing()
{
    for (int i = 0; i < data.pose.Length; ++i)
        data.pose[i] = new Vector3(
            data.pose[i].x * 10,
            -data.pose[i].y * 10,
            -data.pose[i].z * resize_z);
}
```

# 아바타 관절 제어

## 리깅된 아바타의 관절 제어

Landmarks의 위치값을 이용해 회전값을 계산

유니티의 HumanBodyBones API를 사용해  
각 계산된 회전값으로 관절 제어



# Rotation 계산

## 회전 값 계산을 위한 방법

유니티 내장 함수 이용

- Quaternion.FromToRotation()
- Quaternion.LookRotation()

아바타의 각 관절의 Local 좌표계를 구해  
Look 벡터, Up 벡터를 회전 연산에 사용

```
private void SetShoulder()
{
    animator.GetBoneTransform(HumanBodyBones.UpperChest).rotation =
        Quaternion.FromToRotation(Vector3.right, pose[11] - pose[12]);
}
```

```
private void SetArm()
{
    Vector3 leftUp = pose[14] - pose[12];
    Vector3 leftRight = Vector3.Cross(leftUp, pose[16] - pose[12]);
    Vector3 leftForward = Vector3.Cross(leftUp, leftRight);
    animator.GetBoneTransform(HumanBodyBones.LeftUpperArm).rotation =
        Quaternion.LookRotation(leftForward, leftUp);
}
```

# Yolo v5 사용 시도

1

기존엔 한 사람만 인식해 랜드마크를 추출

2

YOLO v5 모델을 이용해 객체(사람)을 감지

3

감지된 수만큼 반복해 landmark 추출



# Yolo v5 사용 시도

