Hand tracking 과 Motion tracking 을 이용한 Digital Puppet

고강연, 문주은, 윤희찬, 한예원

요 약

가상환경의 3D 데이터 컨트롤은 2D 기반의 기존 인터페이스보다는 공간상의 손의 움직임이나 표정 등 복합적인 Human-computer interaction 을 활용하는것이 필수적이다. 이때, 다양한 환경에서 사용할 수 있도록 간단한 센서만을 이용하여 범용성을 높이는 것 또한 중요하다. 따라서 본 논문은 Virtual puppet 이란 가상 환경에서 구현된 인형으로 2 개의 RGB camera 를 사용해서 가상의 인형을 조종한다. RGB camera 을 이용해 손의 움직임을 추적하여 인형이 손가락의 움직임에 맞게 움직일 수 있도록 한다. 또한, 특정 손동작을 인식해서 정해진 행동을 취할 수 있도록 한다. 추가적으로, 다른 RGB camera 에서는 사람의 얼굴을 촬영한다. 캡쳐한 사진으로 사람의 표정을 인식하고 이를 인형의 표정에 대입하여 다양한 표정을 구사할 수 있도록 한다.

1. 서 론

1.1. 연구배경

보통 가상환경에서 인형을 조종하기 위해서는 3D motion capture 또는 Remote manipulator 와 같은 Motion control 을 위한 특수한 센서를 사용했다. 하지만 이 장비들은 가격 등의 측면에서 진입장벽이 높아 일반 사용자들이 사용하기에는 많은 어려움이 있다. 본 논문에서는 이러한 문제를 해결하기 위해서 사용자들이 좀 더 쉽게 다가갈 수 있는 장비인 2 대의 RGB camera 를 사용하여 실감나게 인형을 조종한다. RGB camera 로 손의 움직임을 추적하여 인형이 손가락의 변화에 맞게 움직일 수 있도록 하고, 특정 손동작을 인식해서 정해진 행동을 취할 수 있도록 하여 가상으로 구현한 인형을 조종한다. 또한 다른 RGB camera 에서는 사람의 얼굴을 촬영한다. 촬영한 얼굴을 CNN(Convolution Neural Network)에 넣어 사용자의 표정을 인식해서 기쁨, 슬픔 같은 감정들을 추출하고 이에 맞게 인형의 표정에 적용한다.

본 프로젝트에서 구현한 Virtual puppet 에 사용된 Hand Skeleton Tracking, Hand Gesture Estimation, Emotion Recognition 은 RGB camera 로만 구현되어 컴퓨터, 스마트폰, 스마트 tv 등 웹캠과 같은 카메라를 이용할 수 있는 다양한 전자기기에서 사용할 수 있도록 범용성을 높였다. 또한 Virtual puppet 뿐 아니라 어린이 fun 용, 메타버스 아바타 컨트롤, 전문가용 3d 객체 컨트롤과 같은 다양한 분야에 적용할 수 있다.

특히, 현재는 3D 형태의 데이터를 마우스를 이용하여 2D 평면상에서만 제어할 수 있지만 본 논문에서 구현한 인식 및 트래킹 기능을 이용하면 건축 전문가들이 전체적인 설계도의 3D 단면을 직접 손을 이용해 돌려보며 확인하거나 의사들이 복잡한 종류의 수술을 위해서 3D MRI 혹은, 엑스레이 촬영 자료를 이용할 때 3 차원으로 폭 넓게 확인하는등 다양한 분야에 큰 기여를 할 수 있다.

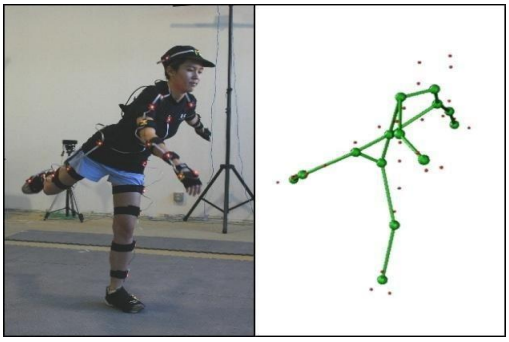

그림 1. 3D motion capture

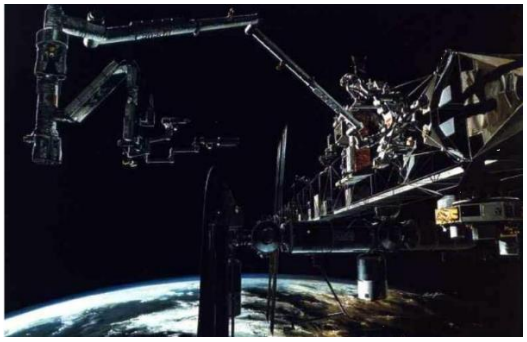

그림 2. Remote manipulator

1.2. 연구목표

3D motion capture, remote manipulator 와 같은 기존의 사용자들이 사용하기 어려운 장비를 이용해서 인형을 조정하는 것이 아니라 2D 동영상을 통하여 특정 손모양이나 손가락의 움직임만으로 인형을 쉽고 간단하게 조종할 수 있도록 구현한다. 이 뿐만 아니라 2D 비디오를 통하여 사람의 표정 변화를 인식하고 이를 인형에 적용한다. 또한, 센서를 이용하게 된다면 여러가지 환경적인 변수로 인하여 정상적인 데이터를 받지 못하는 경우가 많다. 센서에서 받은 불안정한 데이터 값을 그대로 사용하게 된다면 인형은 사용자가 움 직이는 대로 움직이지 않는 경우가 발생한다. 따라서, 단순히 데이터 값을 그대로 이용하는 것이 아닌, 센서에서 받아온 데이터를 정제하는 단계를 추가하여 더 자세하고 안정적인 tracking 과 estimation 을 할 수 있도록 한다. 이를 통해 여러 추가적인 장비 없이, 정확하고 안정적인 Digital Puppet 을 만들 수 있도록 한다.

2. 관련 연구

2.1. 기존 연구

2.1.1. Hand tracking

손의 움직임을 실시간으로 사용자가 추적할 수 있어야 한다. Hand Tracking 이란 사용 자의 손의 움직임을 카메라를 통해서 실시간으로 추적하는 것을 말한다. 제스쳐 인식과 이미지 처리 분야에서 Hand tracking 은 고해상도 기술로 사용자의 손가락의 연속적 위치를 파악하여 객체를 3D 로 표현한다. Hand tracking 은 사용자와 데이터 간의 상호작용에 중점을 두고 있으며 이는 사용자가 표현하는 3D 데이터를 손가락으로 인식하여 가상 데이터 이미지로서 상호작용한다. Hand tracking 은 사용자와 데이터 사이의 상호작용을 통해 손의 움직임을 좀 더 직관적으로 표현하는 것에 초점을 두고 있다. 이때 발생하는 잡음에 의한 혼선이나 순간적인 움직임의 포착이 Hand tracking 의 주요한 과제 중 하나이다. 이러한 과제를 위해 Hand tracking 에서는 기존의 움직임을 기억하여 가중치를 줘서 순간적인 움직임과 기존의 움직임을 크게 벗어나는 값을 통제하고 smoothing 알고리즘을 통해서 잡음에 의한 혼선을 줄인다. 우리는 이러한 Hand tracking 을 2D 이미지를 통해 3D joint 값을 추정하는 MediaPipe 라이브러리를 이용하여 구현하였다.

2.1.2. Emotion recognition

사용자의 얼굴 표정을 통해서 감정을 인식할 수 있어야 한다. Emotion recognition 은 카메라에서 비치는 사용자의 얼굴 데이터를 통해서 감정을 인식하는 기술이다. 표정 뿐 아니라 목소리 등의 매체를 통해서도 판단되기도 한다. 사람의 감정을 표정을 통해서 명확하게 판단하는 이론은 아직 확립되어 있지 않지만 보통은 다양한 표본에서 얻은 자료로 각각의 감정마다 일반적인 표정을 알아내는데 중점을 두고 있다. 다양한 표본들을 활용하기 위해서 주로 머신 러닝을 이용하여 구현되며 svm, rf , knn 등의 알고리즘이 주로 사용되고 있고, 서로 다른 action unit 을 기반으로 얼굴의 감정과 강도를 측정한다. 또한 emotion recognition 은 감정의 구분을 위해서 심리학에 기반한 분류 역시 함께 활용되고 있다. 이중에서 단일 모델 머신 러닝에 기반한 emotion recognition 에 비해 다중 모델 머신 러닝을 이용한 방식이 좀 더 emotion recognition 을 잘 수행해낼 수 있다. 또한 얼굴의 주요 부위에 n 개의 point 를 표현하여 학습시키는 방식도 존재한다.

2.1.3. Hand pose estimation

사용자의 손 모양을 인식할 수 있어야 한다. Hand pose estimation 은 카메라에 비치는 사용자의 손 데이터를 통해서 손의 형태를 인식하는 기술이다. Hand pose estimation 은 우선 사람의 손을 인식하고, 각 손가락을 파악한다. 이후, 각 손가락의 위치와 자세를 이용하여 현재 손이 어떠한 모습을 하고 있는지 판단한다. 이를 나타내는 방식은 여러 가지가 있으며 주로 쓰이는 방식은 손의 중심을 표현한 뒤 중심을 기준으로 각각의 손가락 관절마다 점을 이용하여 상대 좌표계로 표현해주는 방식이 있고 또 다른 방식으로는 각각의 손가락 정보를 모은 후에 각각의 손가락마다 시작점과 끝점 그리고 방향을 표현하여 나타내는 방식이 있다. 주로 2D 이미지를 이용하여 손의 포즈를 판단하지만, 2D 이미지 이외에도 depth 이미지나 point set 들을 이용하여 3D 환경에서 구현하는 방식 역시 사용되고 있다.

그림 3. Emotion estimation

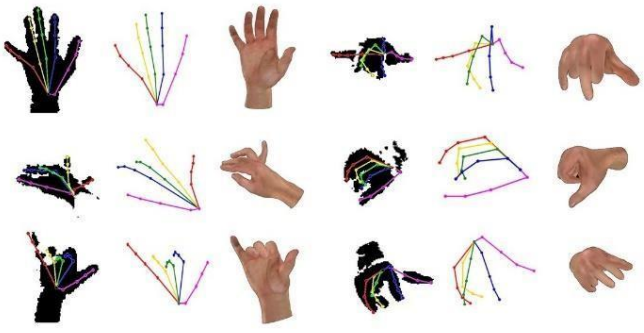

그림 4. Hand pose estimation

2.2. 기존 연구의 문제점 및 해결 방안

2.2.1. 연구의 문제점

위에서 언급한 기술들을 통해서 손 모양의 tracking 과 estimation 을 통해 손의 움직임에 따라서 조종할 수 있으며 얼굴 표정의 tracking 과 recognition 을 통해서 얼굴 표정을 표현할 수 있는 virtual puppet 을 구현한다. 손의 tracking 과 estimation 을 최대한 noise 없이 구현해야 하며 빠르고 순간적인 변화를 포착하여 연속적으로 표현해줄 수 있어야 한다. 또한 virtual puppet 이 사용자의 표정을 명확하게 표현할 수 있도록 얼굴 표정의 estimation 이 정밀하게 수행되어야 한다. 또한 얻어진 정보가 플랫폼에 손실 없이 전송될 수 있어야 하며 플랫폼에서는 얻어진 정보를 토대로 명확한 virtual puppet 의 형성 및 연동이 필요하다.

2.2.2. 해결 방안

2.2.2.1. 2D camera tracking

현재 2D video 를 이용하여 hand tracking 을 수행하는 다양한 모델들이 존재한다. 따라서 이러한 모델들을 사용하여 적절하게 noise 를 통제하고 순간적이거나 모호한 표현을 수정해준다면 hand tracking 과 hand pose estimation 을 수행할 수 있다. 해당 프로젝트에서는 다양한 모델들 중, MediaPipe 를 이용하여 Hand tracking 을 진행하기로 하였다. 또한 2D data 를 통하여 3D 정보를 얻는 것이기 때문에, 데이터를 완전하게 신뢰하지 못한다. 따라서 모델을 통해 얻은 Data 의 Noise 를 제거하고 Outlier 를 없애는 등, 데이터 전처리를 필수적으로 진행해야 한다. 이때 Noise 를 제거하는 방법으로는 우선 첫번째로 값이 이전 값들과 확연하게 차이가 나면 이를 데이터 표본에서 제거하도록 한다. 또한 정제된 데이터들을 이용하여 조금 더 안정적인 값을 얻을 수 있도록, 이전 값들의 경향성과 현재 데이터 값을 비교하여 비슷한 경향성이 나타나도록 변환한다. 이렇게 보정된 Hand Tracking 데이터를 통해서 손의 중심인 palm 좌표와 각각의 손가락들의 관절을 상대좌표로 나타내는 방식 혹은 손가락의 시작과 끝 그리고 각도를 표현해주는 방식을 통해서 Hand pose estimation 을 할 수 있다. 이렇게 얻어진 데이터를 정제해줘서 바로 사용하거나 정확성 향상을 위해서 머신 러닝 모델을 통해 학습시켜서 정밀한 Hand pose estimation 을 수행할 수 있다.

2.2.2.2. 2D camera Facial Emotion Recognition

표정 인식의 정확도를 높이기 위해 영상에서 얼굴 인식이 먼저 수행되고, 인식된 얼굴 부분을 잘라낸 후 표정 인식을 한다. 이 두 종류의 인식 결과를 영상 싱크에 맞게 실시간으로 출력해야 하므로 연산량을 줄이기 위해 매개변수를 줄여 경량화한 CNN 모델[8]을 사용한다. 이 모델은 Facial Expression Recognition 2013 Dataset 으로 Pre-train 되어 있는데 해당 데이터셋의 표정 모델은 대부분 서양인이라 한국인과 인종이 달라 실제로 사용했을 때 민감도가 떨어진다는 단점이 있다. 따라서 Korean Drama Multi-Label Facial Emotion Recognition Dataset[9]을 변형 및 가공하여 Fine-tuning 을 진행했다. 또한, 기존 데이터에서 전체적인 맥락 없이 단순한 얼굴 표정만으로 설명하기 어려운 disgust, scared 와 같이 모호한 감정 표현은 제외하여 사용하였다.

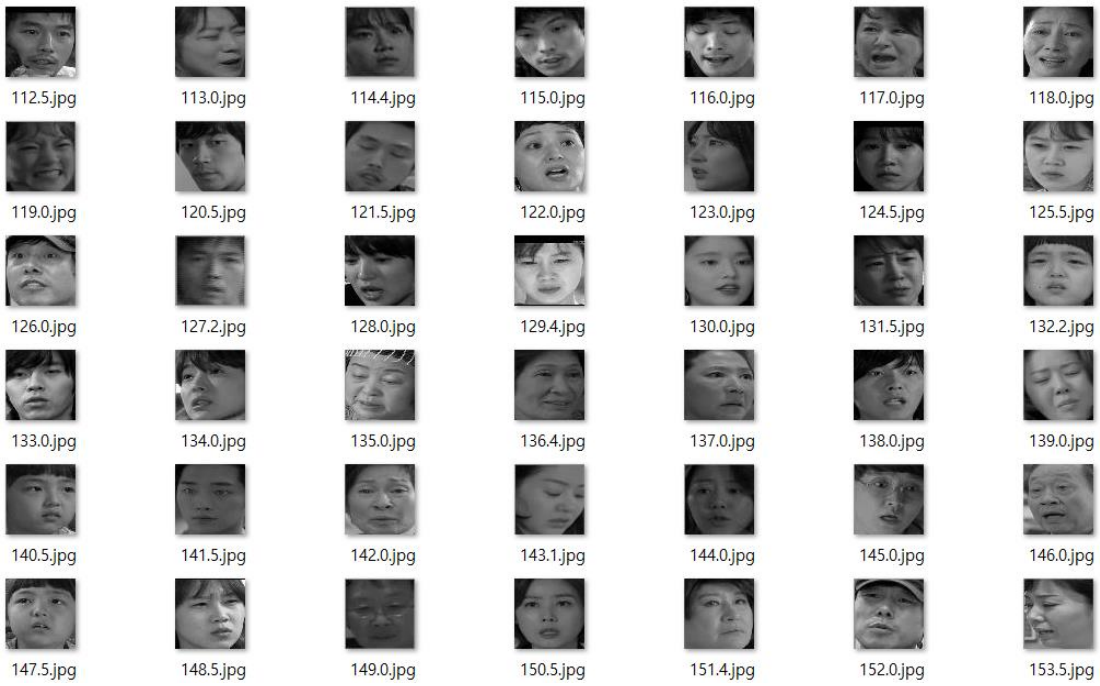

그림 5. CNN 모델 input 에 맞게 변형한 Korean Drama Multi-Label Facial Emotion Recognition Dataset 중 일부

3. 프로젝트 내용

3.1. 시나리오

1) RGB 카메라에서 hand joint 값과 이미지, 얼굴 이미지를 받아온다.

2) 센서를 통해 받은 데이터 값에서 극단적으로 변화한 데이터 값은 outlier 로 판단하여 제거한다. 이후 inlier 데이터들이 비슷한 경향성을 가지며 변화하도록 이전 데이터값들과 현재 데이터 값을 적절한 weight 로 합쳐 좀 더 안정적인 데이터 값을 구한다.

3) 정제된 data 들을 이용하여 puppet 을 조종할 수 있는 여러 값을 구한다. 우선 hand joint 값은 puppet 의 joint 와 매핑해 움직임과 위치를 조정할 수 있도록 한다. 또한, 손의 이미지는 Hand pose estimation 모델을 통하여 hand gesture 를 인식해 puppet 이 특정한 포즈를 취할 수 있도록 한다. 마지막으로, 얼굴 이미지와 depth 값으로 사용자의 감정을 기쁨, 슬픔, 놀람, 화남, 무표정 등으로 분류해 puppet 의 표정을 사용자의 표정과 일치하게 표현할 수 있도록 한다.

4) 이전 단계에서 구한 hand gesture 값, 표정, tracking 값을 소켓 통신을 이용하여 전송, 이를 실시간으로 행동, 표정 등을 반영하는 digital puppet 을 구현한다.

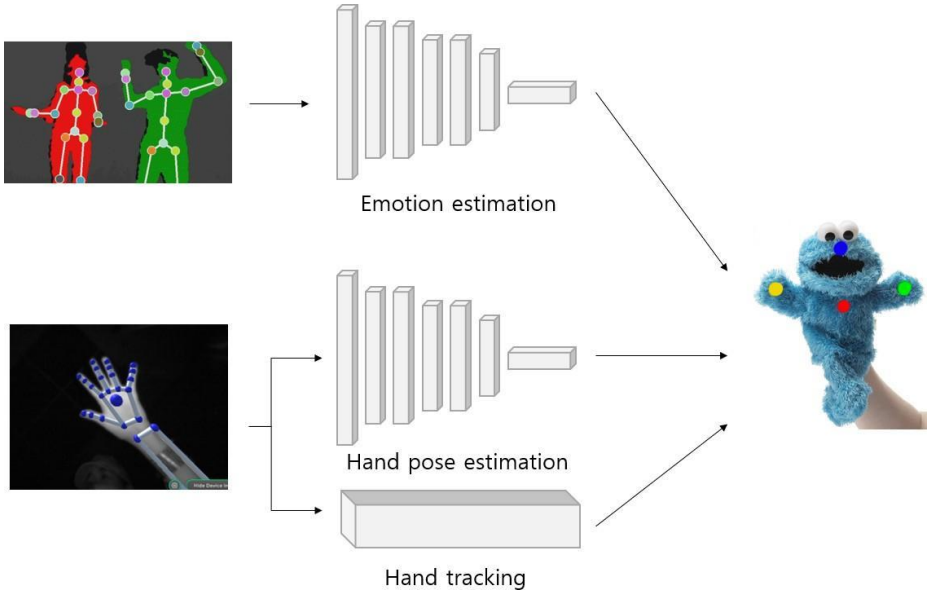

그림 6. 전체적인 모델 구성

3.2. 요구사항

1) RGB camera 에서 추출한 raw data 는 실제 움직임과 다르게 noise 가 존재하며, 주변 값들과 다르게 튀는 값이 존재하기도 한다. 따라서 이를 실제 값과 비슷하도록 적절히 보정하기 위해 gaussian blur 등의 후처리를 해야 한다.

2) puppet 의 움직임이 어색하지 않도록 puppet joint 와 매핑한 hand joint 값을 적절히 조정한다. 움직임이 기괴해지지 않도록 관절의 가동범위를 고려해 값을 보정한다. hand gesture / 표정 인식 모델의 경우, 기존에 존재하는 데이터를 충분히 활용하되, 실험 환 경에 맞는 새로운 데이터를 만들어 모델을 최적화한다.

3) Unity 플랫폼에서 2D camera 에서 얻은 비디오를 통해 얻은 Data 값을 이용하여 virtual puppet 의 움직임과 표정을 구현한다. 적절한 virtual puppet 의 형상을 구현하여 손의 움직임과 표정의 변화가 virtual puppet 과 잘 매칭 될 수 있도록 한다.

4) Puppet 을 움직이기 위한 데이터들을 Unity 로 전송할 때, 소켓 통신을 이용하여 전송할 수 있도록 한다. 이때 데이터 손실을 줄이기 위하여 TCP 통신을 이용하여 데이터를 전송하고, 받도록 한다.

3.3. 시스템 설계

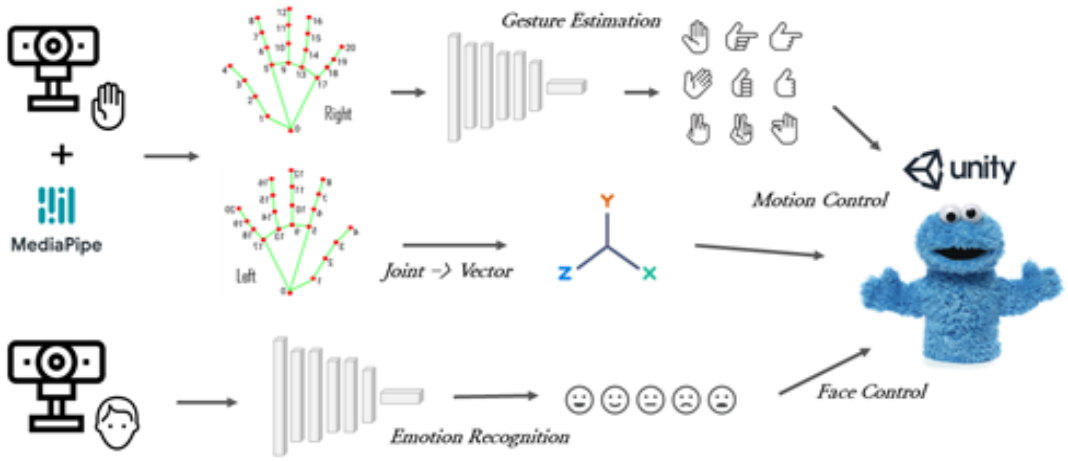

그림 7. 시스템 설계

3.4. 구현

프로젝트에서 필요한 데이터를 주고받는 데에 있어서, 소켓 통신을 이용하여 데이터를 주고받는다. 따라서 해당 프로젝트의 구성은 크게 서버와 클라이언트로 되어있다. 서버에서는 손과 표정을 인식하여 필요한 데이터를 얻은 후, 이를 클라이언트로 전송하고, 클라이언트에 해당하는 유니티는 해당 데이터들을 받아서 인형을 움직인다.

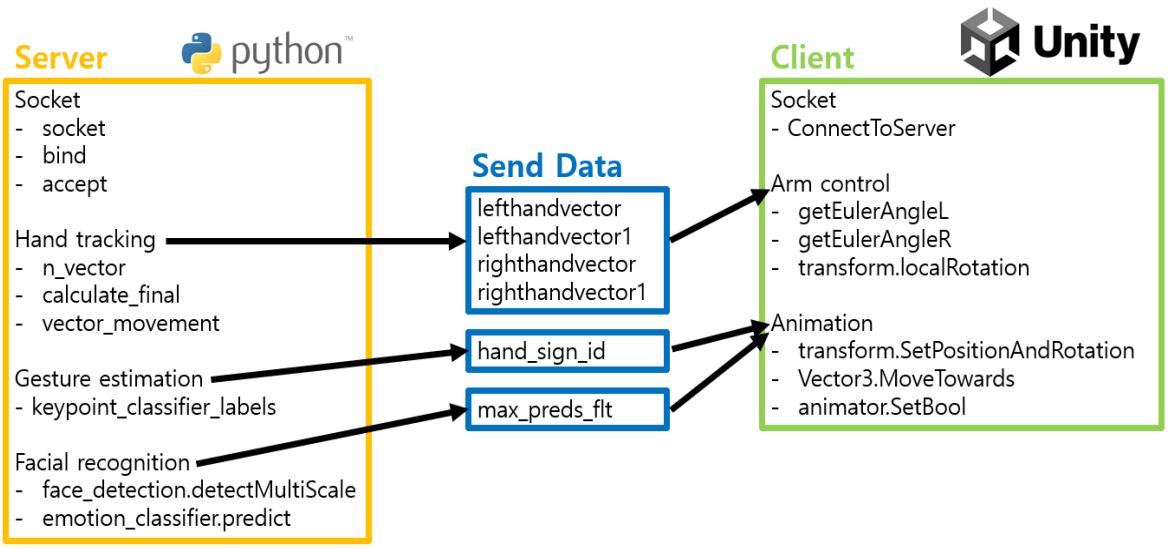

그림 8. 시스템 구조

3.4.1. 서버

우선 서버에서 돌아가는 파트는 Hand tracking & Gesture estimation 파트와 Facial recognition 파트로 구성되어 있다. 소켓으로 데이터를 전송할 때 모든 데이터 값을 한번에 묶어서 전송하기 때문에, 실시간으로 데이터 값의 길이를 변경해주는 것이 쉽지 않았다. 따라서 데이터를 전송하는데에 있어 만약 값이 존재하지 않는 경우에는 -1 값을 넣어주어 유니티 상에서 예외처리를 할 수 있도록 구성 하였고, Nan 값이 들어가는 경우에는 예외처리를 두어 소켓이 전송되지 않도록 구현하였다.

3.4.1.1. Hand tracking

Hand tracking 을 통하여 사용자는 캐릭터를 컨트롤 할 수 있다. 이때 캐릭터를 컨트롤 하고 있는 사용자의 왼쪽 손을 RGB 카메라를 통해 촬영하고 이때 획득한 이미지를 Mediapipe hand tracking 모델에 입력하여 Hand skeleton 정보를 결과 값으로 얻는다. 획득한 정보를 바탕으로 첫째, 엄지, 검지, 중지 손가락의 Skeleton 정보를 중점적으로 사용하여 캐릭터의 신체 일부분을 조종한다. 먼저 캐릭터의 오른쪽 팔, 캐릭터의 상반신, 캐릭터의 왼쪽 팔 부분에 엄지, 검지 그리고 중지를 각각 할당한다. 그 다음 각 손가락 Joint 값을 이용하여 해당 손가락의 방향 벡터를 구하고, 이 벡터값을 이용하여 캐릭터를 조종할 수 있도록, 해당 값을 클라이언트에 넘겨준다.

3.4.1.2. Gesture estimation

왼손이 캐릭터 신체 부분을 컨트롤 한다면, 오른손은 정해진 제스처를 취해서 게임 캐릭터가 해당 제스처에 해당하는 모션을 행동하도록 한다. 우선 카메라를 통해 사용자의 오른손 이미지를 받아오고, 이를 Hand tracking 모델에 넘겨준다. 그러면 Hand tracking 모델이 오른손의 Skeleton 을 추출하고 추출된 Skeleton 의 Joint 부분을 Pre-trained 된 인공신경망에 입력 데이터로 넣어주면 인공신경망은 현재 어떤 제스처를 취하고 있는지 판단한다. 현재 모델에 학습시킨 제스처의 종류는 총 10 가지로, 학습을 하는 데에 사용된 데이터의 개수는 표 1 과 같다. 이때 제스처들 중에서 손을 모두 피는 제스처와 손을 오므린 제스처는 캐릭터 조종의 시작과 종료를 시스템에게 알려주는 역할을 한다. 또한 제스처를 이용하여 캐릭터를 앞뒤, 좌우로 이동할 수 있게 구현하였다. 만약 프로그램을 종료를 하게 되면 이전에 저장되어 있는 데이터들은 모두 초기화하여 처음부터 다시 시작할 수 있도록 구현하였다. 이후 모델이 판단한 제스처를 소켓을 통해 유니티로 전송하여 사용자가 특정 제스처를 취하고 있다고 알리면 유니티는 해당 제스처에 해당하는 모션을 찾고, 캐릭터가 이 모션을 취할 수 있도록 한다.

종류	Hand Open	Hand Close	Point	Rock	Check	Thumbs Up	Down	Up	Left	Right
개수	1110	1262	1050	1145	1055	935	2243	1577	2001	1602

표 1. 제스처 학습 데이터 개수

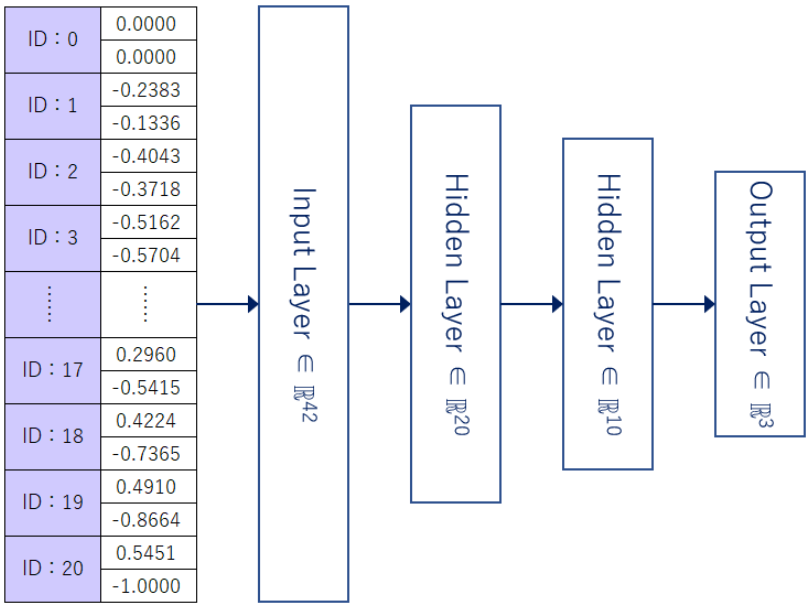

그림 9. 모델 아키텍쳐

3.4.1.3. Facial recognition

웹캠을 이용해 실시간으로 찍고 있는 얼굴 영상을 캡쳐한다. 이 캡쳐 영상에서 Haar Cascade 모델을 이용해 얼굴 부분의 이미지를 검출하여 잘라낸다. 이를 Facial Expression Recognition 2013 Dataset 과 Korean Drama Multi-Label Facial Emotion Recognition Dataset 로 Pre-train 된 CNN 모델에 넣어 사용자의 현재 표정 정보[angry, happy, sad, neutral]에 대한 결과값을 리턴받는다. 이 결과값을 유니티 클라이언트에 전송하여 캐릭터가 해당 표정에 맞는 얼굴 애니메이션을 수행할 수 있도록 한다.

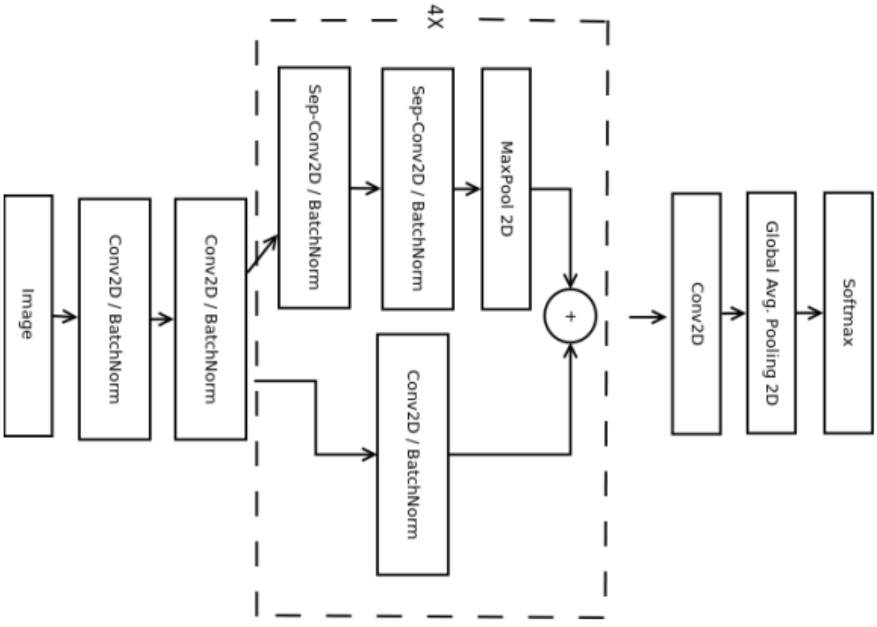

그림 10. 표정 인식 모델 아키텍쳐

3.4.1.4. 안정화를 위한 데이터 전처리

RGB 카메라로 손을 트래킹하는 과정에서 트래킹에 실패하거나 추출된 손의 Skeleton 정보가 비정상적으로 나올 가능성이 있다. 이러한 경우 사용자와 캐릭터 간의 인터랙션을 방해할 수 있으므로, 유니티 상으로 값이 넘어가기 전에 이러한 값들을 전처리해주는 것이 중요하다. 이를 위하여 각 프레임에서 얻은 값을 전후 프레임과 비교하여 현재 값이 적절한 값인지를 판단하고, 만약 적절하지 않다면 이를 조절할 수 있도록 다음과 같이 구현하였다.

우선 트래킹 실패를 확인하기 위하여 다음과 같은 방식을 적용하였다. Frame 상에서 손의 Skeleton 값이 구해지지 않는 경우는 두가지 존재하는데, 첫번째로는 실제로 손이 없는 경우, 두번째로는 일시적으로 트래킹에 실패하는 경우이다. 전자의 경우에는 프레임 상에서 손의 Skeleton 정보를 얻지 못하는 것이 정상적이지만, 후자의 경우에는 손의 Skeleton 정보가 구해져야만 한다. 따라서 먼저 손이 카메라 상에서 보이지 않는 경우인지 아니면 실제로 트래킹에 실패한 경우인지 판단해야 한다. 이를 판단하는 방법으로 우선 각 프레임에서 얻은 손 Skeleton 정보를 Queue 에 넣고, Queue 의 크기를 일정하게 유지하기 위해서 순차적으로 들어온 Skeleton 정보를 차례로 Queue 에서 내보낸다. 매번 Queue 에서 뽑은 데이터들은 유니티로 값을 전송한 후에 이를 이전 Frame 의 데이터 값을 저장해 놓는 곳에 업데이트 하며 값을 저장한다. 이때 만약 Queue 에서 뽑아낸 Skeleton 정보에 아무런 값이 없다면, Queue 에 있는 모든 Skeleton 데이터 정보들을 확인한다. 만일 Queue 에 있는 모든 Skeleton 값 역시 존재하지 않는다면 이는 손이 카메라에서 사라졌다는 것을 의미하지만, Queue 에 있는 skeleton 정보들이 값을 가지고 있다면 이는 일시적인 트래킹 실패를 의미하게 된다. 따라서 이러한 경우에는 미리 저장해 두었던 이전 Frame 의 데이터 값을 불러와 이 데이터와 동일하게 Skeleton 값을 저장하고, 이를 유니티로 전송해준다. 마지막으로 현재 프레임의 값을 이전 Frame 데이터 값으로 업데이트 한다.

다음으로는 Outlier 데이터를 제외시키기 위하여 일차적으로 모델을 통해 얻은 데이터값의 신뢰도가 50% 미만이면 해당 값은 사용하지 않고, 이전 Frame 의 데이터값을 넘겨주게 된다. 신뢰도가 50%를 넘겼지만, 여전히 값이 Outlier 인 경우에는, 이를 해결하기 위하여 위와 동일하게 Queue 와 이전 Frame 의 Skeleton 값을 이용한다. 만약 서로 인접하는 프레임(이전 Frame 과 이후 Frame)과 현재 Frame 의 Skeleton 손가락 벡터 차이가 모두 역치를 벗어나게 된다면, 이는 적절하지 않은 데이터라고 판단하고, 해당 데이터 값을 이전 Frame 과 이전 Frame 의 Skeleton 데이터 값의 평균으로 설정한다.

마지막으로 실험을 진행하는 과정에서, 손이 움직이지 않는데에도 불구하고 데이터의 값이 일정하게 나오지 않는 것을 확인할 수 있었다. 이에 따라 데이터의 값이 조금만 바뀌는 경우라도 캐릭터의 팔이 과도하게 떨리거나 움직이는 상황이 발생한다. 따라서 이러한 현상을 줄이기 위하여 Kalman filter 를 적용하여 이전데이터들을 통해 예측 값을 구하고, 실제 값과 이를 비교하여 Smoothing 한 Joint 값을 얻고자 하였다. 이때 예측 값은 이전 값에서 지금까지 한 프레임당 각 Joint 의 변화 정도를 계산하여 구하였고, 분산의 경우 모델을 통해 얻은 데이터값의 신뢰도를 이용하여 분산을 정하였다. 이를 통하여 Smoothing 하기 전보다 더욱 안정된 캐릭터의 움직임을 확인하였다.

```python
def vector_movement(moveMatrix, successframe, preMatrix, curMatrix):

    curmove = abs(curMatrix - preMatrix)
    moveMatrix = moveMatrix + ((curmove-moveMatrix)/successframe)

    return moveMatrix

def calculate_final(moveMatrix,preMatrix,curMatrix,confidence,successframe):

    # single point prediction update
    posMatrix = np.zeros((12,3))
    for j, pPoint in enumerate(zip(preMatrix,curMatrix)):
        for i in range(3):
            if  pPoint[1][i] >pPoint[0][i] :
                posMatrix[j][i] = 1
            else:
                posMatrix[j][i] = -1

     # position = np.array([ 1 if i >j else -1  for cjoint, pjoint in
enumerate(zip(preMatrix,curMatrix)) for i, j in enumerate(zip(cjoint, pjoint))])
    predict = moveMatrix*posMatrix + preMatrix

    # Compare current vector and predict vector and update
     curMatrix = (predict * confidence[0] + curMatrix * confidence[1]) /
(confidence[0]+confidence[1])

    moveMatrix = vector_movement(moveMatrix,successframe,predict,curMatrix) #
Single movement vector update

        confidence[0]   =   (confidence[1]*confidence[0])/(confidence[1]   +
confidence[0])

    preMatrix = curMatrix
```

그림 11. Kalman filter 를 비슷하게 적용한 smoothing

3.4.2. 클라이언트

클라이언트에서는 서버로부터 받은 손의 위치에 대한 벡터 값, 얼굴 표정과 제스쳐에 대한 값을 모두 서버로부터 받아왔다. 이 때 서버에서 데이터를 전송할 때 바이트 형식으로 전송하므로 받아올 때는 플로트 자료형을 바이트 단위로 끊어서 값을 읽어와야 정확한 값을 읽어 올 수 있다. 또한 받아올 값이 없는 경우에는 update 를 하지 않는 방식을 통해 예외 처리를 하였으며 서버와의 협업을 통해서 -1 값이 들어올 시에는 변화가 없도록 구현하였다. 이렇게 받아온 값을 처리하여 유니티상에서 다양한 동작과 움직임 표정 변화 등을 구현하였다.

```
public void ConnectToServer()
{
    // 이미 연결되었다면 함수 무시
    if (socketReady) return;

    // 기본 호스트/ 포트번호
    string ip = "192.168.0.3"; //"192.168.0.40", 192.168.0.3
    int port = 8889;
```

```
    // 소켓 생성


    socket = new TcpClient(ip, port);
    stream = socket.GetStream();
    writer = new StreamWriter(stream);
    reader = new StreamReader(stream);
    socketReady = true;
}

void Update()
{
    //transform.rotation = Quaternion.Euler(new Vector3(30, 0, 0));
    if (socketReady && stream.DataAvailable)
    {
        receivedBuffer = new byte[100];
        stream.Read(receivedBuffer, 0, receivedBuffer.Length); // stream에 있던 바이트배열

        float msg = BitConverter.ToSingle(receivedBuffer, 0);  // float는 4바이트...
        float msg2 = BitConverter.ToSingle(receivedBuffer, 4);
        float msg3 = BitConverter.ToSingle(receivedBuffer, 8);
        float msg4 = BitConverter.ToSingle(receivedBuffer, 12);  // float는 4바이트...
        float msg5 = BitConverter.ToSingle(receivedBuffer, 16);
        float msg6 = BitConverter.ToSingle(receivedBuffer, 20);

        float msg7 = BitConverter.ToSingle(receivedBuffer, 24);  // float는 4바이트...
        float msg8 = BitConverter.ToSingle(receivedBuffer, 28);
        float msg9 = BitConverter.ToSingle(receivedBuffer, 32);
        float msg10 = BitConverter.ToSingle(receivedBuffer, 36);  // float는 4바이트...
        float msg11 = BitConverter.ToSingle(receivedBuffer, 40);
        float msg12 = BitConverter.ToSingle(receivedBuffer, 44);
```

그림 12. 클라이언트(유니티)에서의 데이터 수신

3.4.2.1. 캐릭터 팔 조종

손가락을 사용해서 캐릭터의 팔을 조종하는 가장 직관적인 방법은 그림 12 과 같이 매핑을 하는 방법이 있다.

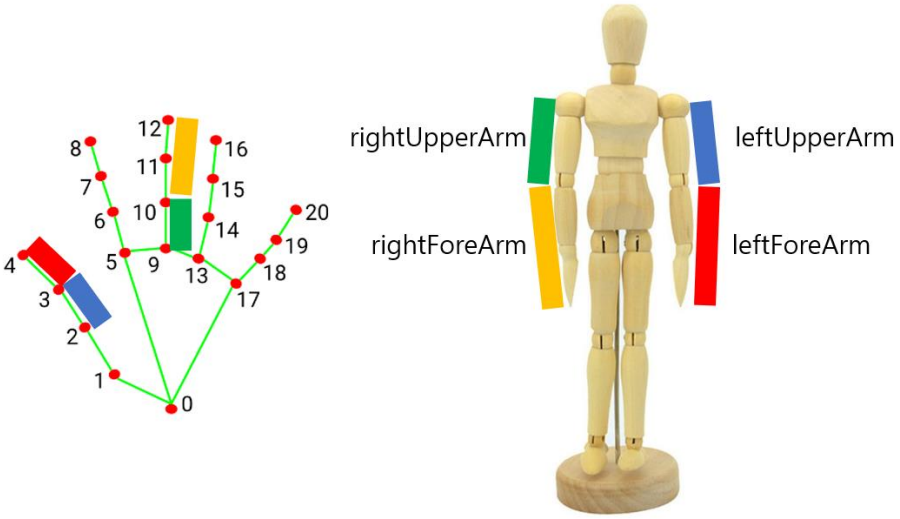

그림 12. 왼손과 캐릭터 팔 사이의 매핑

캐릭터의 왼쪽 상완 부분은 왼손 엄지손가락의 2-3 번 사이의 마디를, 전완 부분은 3-4 번 사이의 마디를, 오른쪽 상완은 중지손가락의 9-10 번 마디, 마지막으로 전완은 10-12 번 마디를 매핑한다. 코드상에서는 msg, msg2, msg3 가 서버로부터 받아온 2-3 번 사이의 벡터값의 x, y, z 값에 해당한다. 비슷한 방식으로 msg4-6 은 3-4 사이 벡터의 x, y, z 값이고, msg7-9 은 9-10 사이 벡터의 x, y, z 값, 마지막으로 msg10-12 은 10-12 사이 벡터의 x, y, z 값이다.

왼손과 캐릭터의 팔 부분이 서로 매핑되었다면 다음은 왼손의 엄지와 중지의 움직임에 따라서 캐릭터의 팔이 동일하게 움직여야한다. 그러기 위해서 현재의 벡터와 바로 이전의 벡터 이 둘의 사잇각을 구해서 그 각만큼 팔을 회전시키는 방법을 사용한다. 회전각을 구하기 위해서 외적을 이용한 아래의 식을 사용한다.

$$\theta_x = \sin^{-1}\left[\frac{|\vec{a}\times\vec{b}|}{|\vec{a}||\vec{b}|}\right]$$

그림 13. 외적을 이용한 두 벡터의 사잇각을 구하는 수식

3 차원 상의 회전을 적용하기 위해서 x, y, z 축에 대한 각각의 θ 값을 구한다. 이를 구하는 코드는 아래와 같다.

```
// get angle using cross product
angleX = Mathf.Rad2Deg * Mathf.Asin(magnX /
    (Mathf.Sqrt((past.y * past.y) + (past.z * past.z)) *
    Mathf.Sqrt((cur.y * cur.y) + (cur.z * cur.z))));

angleY = Mathf.Rad2Deg * Mathf.Asin(magnY /
    (Mathf.Sqrt((past.x * past.x) + (past.z * past.z)) *
    Mathf.Sqrt((cur.x * cur.x) + (cur.z * cur.z))));

angleZ = Mathf.Rad2Deg * Mathf.Asin(magnZ /
    (Mathf.Sqrt((past.x * past.x) + (past.y * past.y)) *
    Mathf.Sqrt((cur.x * cur.x) + (cur.y * cur.y))));

if(float.IsNaN(angleX) || float.IsNaN(angleY) || float.IsNaN(angleZ))
{
    angleX = 0;
    angleY = 0;
    angleZ = 0;
}

if(weight == 1.5f) return new Vector3(0, angleZ, weight * angleX); // Upper Arm
else return new Vector3(0, angleY, 0); // Fore Arm
```

그림 14. 캐릭터 팔 회전에 필요한 두 벡터간의 사잇각을 구하는 코드

위의 그림은 캐릭터의 왼팔을 회전하기 위해 구현한 함수 getEulerAngleL(float weight, Vector3 past, Vector3 cur)의 일부분이다. 여기에서 magnX, Y, Z 는 외적의 크기를 나타낸다. 또한 사잇각을 구하다가 NaN 값이 나올 수 있으므로 이에 대한 예외 처리를 if 문을 통해 처리한다.

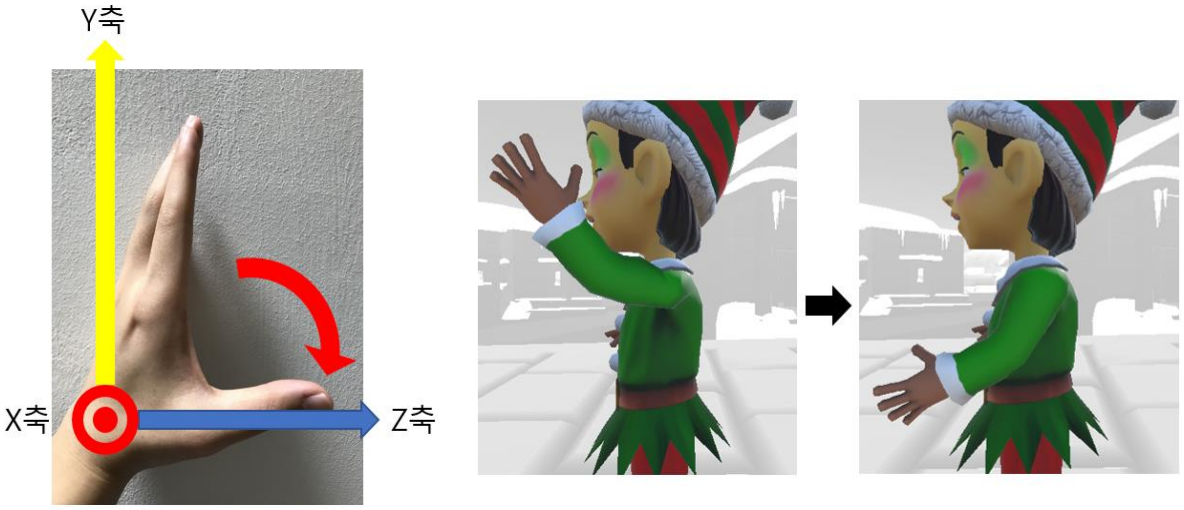

그림 15. 엄지 손가락 회전에 따른 캐릭터 팔 회전

mediapipe 에서는 위의 왼쪽 그림처럼 x 축-빨간색, y 축-노란색, z 축-파란색과 같이 좌표축이 설정되어 있는데 x 축을 중심으로 빨간색 화살표를 따라 엄지손가락을 회전한 값을 유니티에 그대로 넣게 되면 캐릭터의 팔이 전혀 다르게 회전하는 문제가 있다. 문제의 원인은 mediapipe 상에서

설정한 좌표축과 유니티상에서의 캐릭터 팔에 해당하는 좌표축이 다르다는 점인데 위 그림처럼 캐릭터 팔을 아래로 내리기 위해서는 x 축이 아닌 z 축 회전이 사용된다. 이를 해결하기 위해서 위의 코드의 if 문 리턴값과 같이 mediapipe 상의 x 축 회전값인 angleX 를 Vector3 의 z 성분에 넣는다. 반면에 팔꿈치를 굽혀서 전완을 움직이는 부분은 mediapipe 와 유니티상에서 모두 y 축 회전으로 동일하므로 else 리턴값과 같이 mediapipe 상의 y 축 회전값인 angleY 를 Vector3 의 y 성분에 넣는다.

코드상에서 weight 는 가중치로 엄지와 중지가 움직일 수 있는 회전반경은 한정되어 있기 때문에 캐릭터가 차렷 자세를 취하는 것이 불가능하다. 따라서 회전값에 1.5 배의 가중치를 줘서 손가락이 움직이는 것보다 더 많이 캐릭터 팔이 움직일 수 있게 해서 차렷 자세가 가능하도록 구현하였다.

이러한 방식으로 캐릭터의 오른팔부분인 getEulerAngleR 도 동일하게 구현한다.

```
angleLeftUpper = getEulerAngleL(1.5f, pastVecLeftUpper, curVecLeftUpper);
angleLeftFore = getEulerAngleL(1, pastVecLeftFore, curVecLeftFore);

angleRightUpper = getEulerAngleR(1.5f, pastVecRightUpper, curVecRightUpper);
angleRightFore = getEulerAngleR(1, pastVecRightFore, curVecRightFore);

leftUpperArm.transform.localRotation *= Quaternion.Euler(angleLeftUpper);
rightUpperArm.transform.localRotation *= Quaternion.Euler(angleRightUpper);
```

그림 16. localRotation 에 적용

왼팔, 오른팔에 대한 3 차원 회전각이 모두 구해졌다면 이 값을 위 그림처럼 Quaternion Euler 함수에 넣어 quaternion 값으로 변환하고 이를 localRotation 에 적용한다.

3.4.2.2. 표정 연결

표정 역시 유니티상에서 캐릭터의 얼굴 객체의 구성요소의 transform 과 rotation 을 조정해주고 서버에서 값을 받아올 때마다 얼굴 객체의 transfrom 과 rotation 을 각 표정에 맞게 설정해주는 과정을 통해서 표정을 구현하였다.

그림 21. 애니메이션을 통해 구현한 표정 (왼쪽부터 sad / happy / angry)

3.4.2.3. 제스쳐 동작 연결

제스쳐 동작 연결을 위해 서버로부터 받아온 제스쳐 값을 통해서 일반 idle 상태와 다양한 애니메이션 상태를 구현하였으며 이를 코드에서 bool 값을 통해서 확인할 수 있도록 하였다. 또한 애니메이션이 계속해서 true 상태를 유지해야만 실행되기에 애니메이션이 실행 될 때 count 값을 이용하여 특정한 업데이트 count 를 만족해야만 애니메이션 상태를 변경 할 수 있도록 하여 애니메이션이 잘 동작 할 수 있게 하였다.

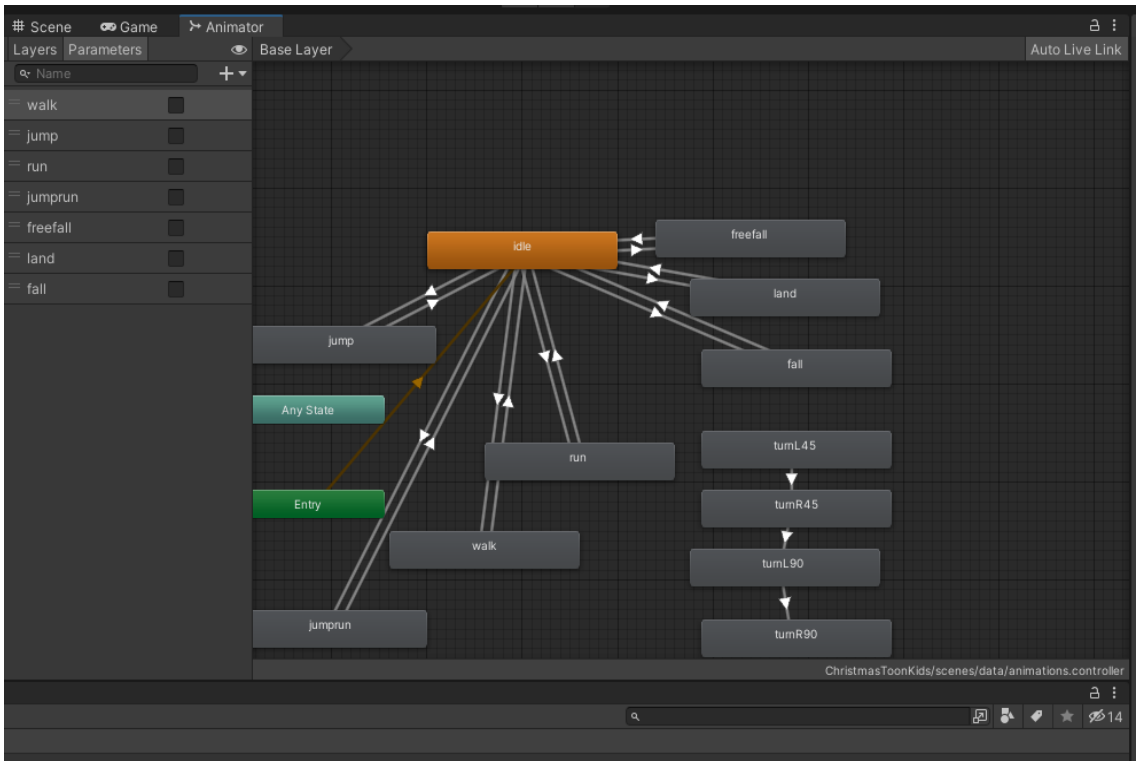

그림 22. 제스쳐 실행 순서도

4. 프로젝트 결과

4.1. 연구 결과

4.2. 성능 평가

4.2.1. Hand gesture estimation

Hand gesture 를 위하여 직접 각 Class 마다 데이터를 표 1 과 같이 뽑아내어 모델을 학습시켰다. 이후 Test 이미지들을 이용하여 각 Class 마다 정확도를 측정하였다. Hand gesture estimation 의 결과는 그림 17 에서 알 수 있듯이, 대부분 정확하게 맞추었지만, 주먹을 어느정도 쥐는 제스처들이 간혹 다른 제스처로 인식이 되는 경우가 발생했다. 이를 위해서 더 많은 데이터를 넣고 다시 모델을 학습시켰지만 결과는 비슷하게 나오는 것을 확인하였다. 이로 인해 간혹 모델이 종료 제스처로 인식하여 프로그램이 종료되는 상황이 발생했지만, 이러한 경우를 제외하고는 모두 잘 인식하여 제스처를 통해 캐릭터를 수월하게 컨트롤 하는 것을 확인하였다.

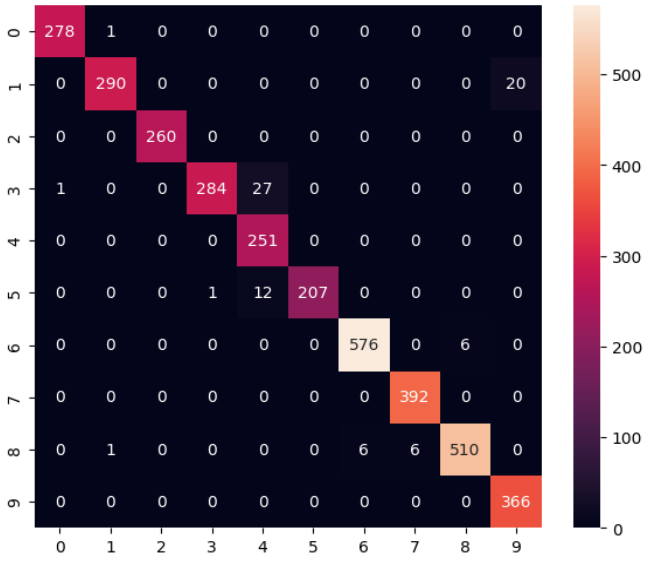

그림 17. Hand gesture estimation 결과

4.2.2. Hand smoothing

Smoothing 의 성능을 평가하기 위하여 Smoothing 을 진행한 경우와 진행하지 않는 경우를 측정하였다. Smoothing 을 하지 않는 경우, 손을 가만히 있음에도 불구하고 그림 18 과 같이 손의 Joint 값이 변화하였지만, Smoothing 을 한 후에는 그림 19 과 같이 손의 Joint 값이 변화하지 않았음을 확인할 수 있었다. 이를 통하여 캐릭터의 팔의 떨림이 이전보다는 나아진 모습을 확인하였지만, 여전히 조금 흔들리는 것은 조절하지 못하였다.

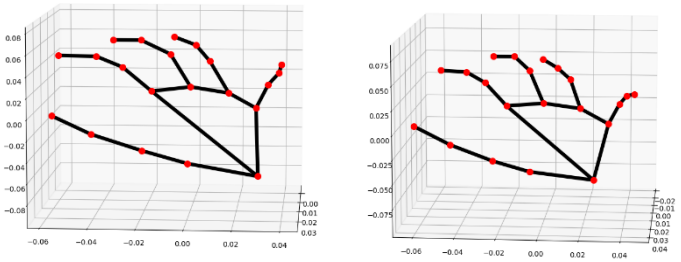

그림 18. Smoothing 을 수행하지 않은 경우

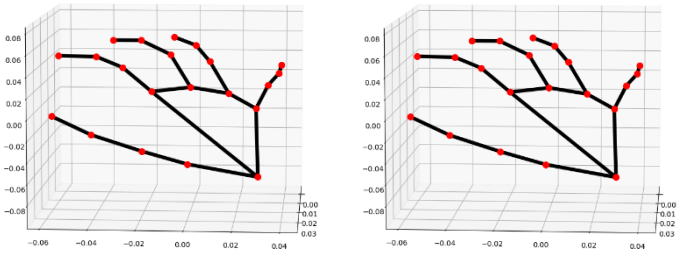

그림 19. Smoothing 을 수행한 경우

4.2.3. Facial recognition

표정 인식의 성능을 높이기 위해 Korean Drama Multi-Label Facial Emotion Recognition Dataset 으로 Fine-tuning 을 진행하으나 데이터 부족 및 모호한 label 분류로 인해 큰 차이를 실감하기는 어려웠다. 이에 기존 데이터의 angry, disgust, scared, happy, sad, surprised, neutral 7 가지의 표정 분류 중 전체적인 맥락 없이 단순한 얼굴 표정만으로 설명하기 어려운 disgust, scared, surprised 를 제외하고 확실하게 분류 가능한 angry, happy, sad, neutral 만을 이용하여 성능을 체감 가능하도록 변형하였다.

5. 결 론

5.1. 기대 효과

본 프로젝트에서 사용자의 손의 움직임을 통해 자연스럽게 움직이고 사용자의 표정 변화를 따라하는 가상 인형을 만들었다. 기존의 트래킹 모델과 인식 모델이 가지고 있는 잡음으로 인한 부정확함을 줄이고 순간적인 변화에 보다 더 잘 대처하는 모델을 만들었다. 이를 통해서 기존의 tracking model 과 recognition model 이 가지고 있는 부정확함을 줄일 수 있으며 또한 사용자는 자신의 손과 얼굴 표정의 변화를 통해 변화하는 virtual puppet 을 확인함으로써 기존의 vr/ar 보다도 다양한 체험을 할 수 있다. 또한, 특수한 장비를 없이 카메라 만을 통한 컨트롤 방식이 더 발전하게 된다면, 다양한 분야의 3D data 를 2D 상이 아닌 3D 상에서의 컨트롤을 적용할 수 있다.

5.2. 추후 연구 방향

현재까지 진행한 전처리 및 Smoothing 을 보완하여 추후에는 가만히 있어도 손 관절의 joint 좌표들이 떨리지 않고 그대로 유지되도록 한다. 또한, 유니티상에서 보여지는 캐릭터의 팔도 떨리지 않고 더 자연스럽게 움직이도록 만든다. 표정 인식의 경우 주변 환경의 영향과 인종의 영향을 덜 받도록 fine-tuning 을 추가로 진행한다.

6. 참고 문헌

[1] Goebl, W.; Palmer, C. (2013). Balasubramaniam, Ramesh (ed.). "Temporal Control and Hand Movement Efficiency in Skilled Music Performance"

[2] https://learn.microsoft.com/en-gb/azure/kinect-dk/body-joints

[3] Liyu Meng, Yuchen Liu, Xiaolong Liu, Zhaopei Huang, Yuan Cheng, Meng Wang, Chuanhe Liu, Qin Jin : Multi-modal Emotion Estimation for in-the-wild Videos

[4] https://towardsdatascience.com/emotion-detection-a-machine-learning-project-f7431f652b1f

[5] Ali Erol , George Bebis , Mircea Nicolescu , Richard D. Boyle , Xander Twombly : Vision- based hand pose estimation: A review

[6] Cem Keskin, Furkan Kırac¸, Yunus Emre Kara and Lale Akarun : Real Time Hand Pose Estimation using Depth Sensors

[7] Liuhao Ge , Yujun Cai , Junwu Weng , Junsong : YuanHand PointNet: 3D Hand Pose Estimation using Point Sets

[8] Octavio Arriaga, Matias Valdenegro-Toro, and Paul Plöger. Real-time convolutional neural networks for emotion and gender classification. arXiv preprint arXiv:1710.07557, 2017.

[9] H. Cho, W. K. Kang, Y. -S. Park, S. G. Chae and S. -j. Kim Multi-Label Facial Emotion Recognition Using Korean Drama Video Clips, 2022 IEEE International Conference on Big Data and Smart Computing (BigComp), 2022.