YOLO 알고리즘과 HSV 색상 모델을 활용하여 환경 변화에 강인한 실시간 손 추적 알고리즘 설계 및 구현

이영재, 김현규, 나은찬, 최성률, 이상준
{ youngjewall, dhk0561, eunchan0492, chltjd513 }@naver.com, sangjun@ssu.ac.kr
숭실대학교 소프트웨어학부

Design and implementation of a real-time hand tracking algorithm that is robust to environmental changes using YOLO algorithm and HSV color model

Youngjae Lee, Hyeongyu Kim, Eunchan Na, Seongryul Choi, Sangjun Lee
School of Software, Soongsil University

요약

가상현실 및 증강현실은 현대인에게 매우 친숙한 기술이 되어가고 있으며 이러한 기술의 발전에 맞추어 점점 경량화된 HMD(Head Mounted Device)도 등장하고 있다. 이에 따라 증강현실 제어에 대한 새로운 방식으로 손을 직접 사용하는 방법을 고안하였다. 이 목표는 컴퓨터 비전 기술을 사용하여 손의 모양과 동작을 인식하는 것에서 시작할 수 있다. 본 논문에서는 이를 위한 첫 단계로 실시간으로 손을 인식하고 추적하는 방법에 대해 제안한다. 크게 인식과 추적의 단계로 나뉜다. 주변 환경 변화에 강인한 대응을 위해 YOLO 알고리즘을 사용하여 손을 인식하고, 실시간의 빠른 속도로 손의 모양을 인식하고 추적하기 위해 HSV 색상 모델을 사용한다. 추가적으로 가상의 3D 공간 내에서 손을 움직일 수 있도록 가상 손을 생성하는 단계까지 진행하였다.

ABSTRACT

Virtual reality and augmented reality are becoming very familiar technologies to modern people, and in accordance with the development of these technologies, head mounted devices (HMDs) that are becoming lighter in weight are also appearing. Accordingly, a new method for augmented reality control was devised to use the hand directly. The goal can begin with recognizing the shape and motion of the hand using computer vision technology. In this paper, we propose a real-time hand recognition and tracking method as the first step. It is largely divided into stages of recognition and tracking. To robustly respond to changes in the surrounding environment, the YOLO algorithm is used to recognize the hand, and the HSV color model is used to recognize and track the shape of the hand in real time. In addition, the process of creating a virtual hand to move the hand in a virtual 3D space was progressed.

1. 서론

AR과 VR은 4차 산업 혁명의 핵심기술로 손꼽힌다. 게임부터 시작해서 교육, 건강, 국방, 공학 등 다양한 산업으로 퍼지고 있다. 현재는 AR 시장보다는 VR 시장이 크다. 하지만 2018년 이후 스마트 디바이스 기반의 AR 서비스 및 콘텐츠가 다양한 산업 영역에 활용되어 시장을 주도할 것이라고 정보통신산업진흥원은 예상하였다.

기술의 발전이 이어질수록 사용자는 더욱 편리한 생활환경을 제공받는다. 이는 AR 분야에 있어서도 마찬가지이다. 초기 AR은 한손으로 스마트폰을 들고 AR 환경을 체험하는 형태였다. 그러나 Google Glass 등의 등장으로 사용자는 자유로운 두 손으로 AR 환경을 접할 수 있게 되었다. 이에 따라 사용자는 스크린 터치 등이 아닌 방식으로 증강현실을 제어할 수 있는 새로운 방법을 필요로 한다.

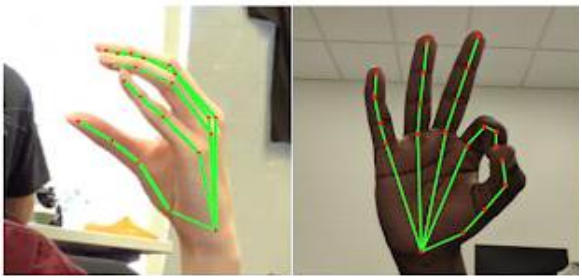
손의 모양과 움직임을 인식하고 추적하는 기술은 증강현실의 디지털 컨텐츠에 물리적인 작용을 통한 정보의 전달로 이어질 수 있다. 본 프로젝트는 이러한 기술을 통해 AR 환경을 직접 손으로 직접 제어할 수 있는 사용자 경험을 제공하고자 한다. 이를 위한 첫 단계로 스마트폰 카메라를 통해 비춰지는 손의 움직임을 통해 화면에 보이는 AR 물체를 만지고 건드리는 방식 등으로 제어할 수 있게 하는 것이 첫 번째 목표이다. 컴퓨터 비전 기술을 사용하여 손의 움직임을 감지하고 그 움직임에 따라 여러 기능을 제공한다. 예를 들어 AR 물체의 크기를 조절하거나 위치를 이동시킬 수 있고 새로운 물체 등을 만들 수 있다.

2. 관련 연구

2.1 Google MediaPipe Hands

MediaPipe는 구글사에서 개발한 컴퓨터 비전 기술로 모바일, 데스크탑, 웹 및 IoT 기기를 위한 머신러닝 솔루션을 제공한다.

MediaPipe Hands는 그 중 손 추적 솔루션으로 모바일 기기에서 실시간 추적의 성능을 보이며 조명이나 기타 환경 등의 변화에도 높은 정확성을 보인다. 또한 다중 손에 대한 추적도 지원하고 있다.



[그림 n] MediaPipe Hands 실행 예

2.2 Leap Motion

매우 정교한 손동작 인식 센서를 지닌 사용자 인터페이스를 제공하는 장치이다. 사용자의 손동작을 인식하여 컴퓨터를 통해 게임, 시뮬레이션 등 다양한 작업을 할 수 있다. Airspace라는 전용 앱 스토어를 가지고 있어 Leap Motion 전용 애플리케이션을 설치할 수도 있다.

Leap Motion은 컴퓨터에 유선으로 연결하여 그 위에서 손을 움직여서 손동작을 인식한다. 따라서 손동작 인식을 위해서는 기기를 휴대하고 다녀야 하며, 유선으로 컴퓨터에 연결해서 사용하기 때문에 HMD와 연결해 자유롭게 사용하기에는 어려움이 있다.

[그림 n] Leap Motion 실행 예

3. 구현 내용

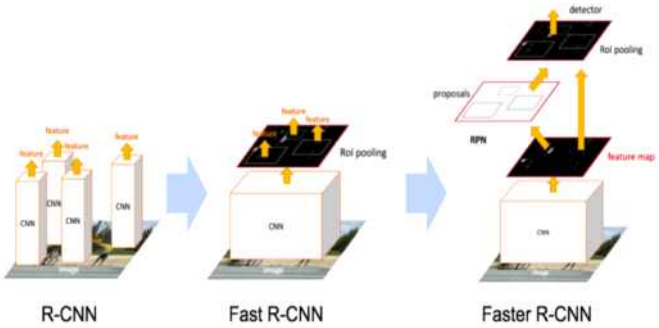
3.1 YOLO

3.1.1 YOLOv2

AR 오브젝트를 제어하기 위해서는 스마트폰의 카메라를 통해 사용자의 손을 인식하고 추적해야 한다. 움직이는 손을 정확하게 인식해야 하기 때문에 딥러닝을 사용한다. 딥러닝을 통해 손을 인식하면 색상 모델의 HSV 임계값을 설정하여 움직이는 사용자의 손을 추적하게 된다.

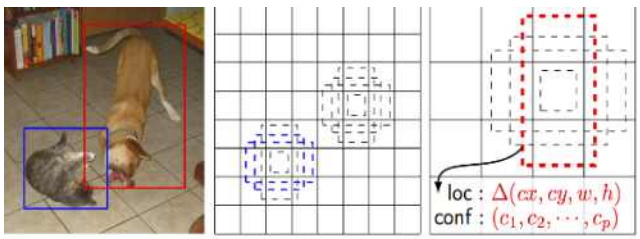
손을 인식하는 딥러닝의 종류로는 Faster-RCNN, SSD(Single Shot Detector), YOLOv2가 있다.

Faster-RCNN은 Region Proposal 단계에서 발생하는 병목 현상을 Selective search가 아닌 RPN(Region Proposal Network)를 사용함으로써 제거한다. 그렇기 때문에 RCNN, Fast-RCNN보다 정확하고 겹치거나 작은 사물에 대한 인식률이 높다. 하지만 속도 면에서는 느리기 때문에 실시간으로 사용자의 손을 인식하기에는 부적절한 방법이다.



[그림 n] Faster R-CNN 처리 과정

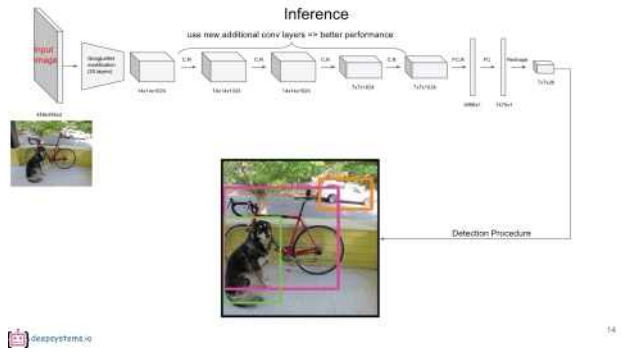
SSD는 사진의 변형 없이 한 장으로 훈련하고 검출을 한다. feature map을 여러 크기로 만들어 큰 map에서는 작은 물체를 검출하고, 작은 map에서는 큰 물체를 검출하여 위치 추정과 입력 이미지의 resampling 과정을 없앴기 때문에 Faster-RCNN, YOLO보다 빠르고 높은 정확도를 가지게 된다. 하지만 너무 가깝거나 너무 작은 물체를 감지하는데 문제가 있다. 또한, YOLO에 비해 사용 방법이 쉽지 않은 단점이 있다.



[그림 n] SSD map

YOLOv2는 YOLO의 다음 버전이다. YOLO의 기존 성능과 속도를 모두 개선 시켰고, 9000개의 오브젝트 카테고리를 검출할 수 있다. 네트워크의 크기를 조절하여 FPS(Frame Per Second)와 MaP(Mean average Precision)을 균형 있게 조절할 수 있는 장점도 가졌다. YOLO를 실행시키기 위해서는 Darknet 또한 필요하다. Darknet은 DNN을 학습시키고 실행시킬 수 있는 Framework이다.

YOLOv2를 본 프로젝트에서 사용한 가장 큰 이유는 앞서 소개했던 Faster-RCNN과 SSD보다 속도가 빠르고 정확하기 때문이다. 또한, 실시간으로 처리하는 속도도 SSD보다 높게 측정된다. 학습시킬 이미지 중에서 미리 손 영역을 라벨링한다. 라벨링 된 이미지들을 YOLOv2에 딥러닝 시켜 스마트폰 카메라를 통해 들어온 이미지 중에서 손 영역을 추출하도록 한다. 라벨링 과정도 쉽게 할 수 있고, 딥러닝 과정도 SSD에 비해 쉽기 때문에 YOLOv2를 선택하였다.



[그림 n] YOLOv2 과정

3.1.2 데이터 라벨링

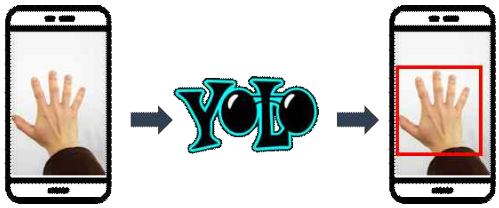
YOLOv2를 사용하여 손을 인식하게 하려면 이미지들을 학습시켜야 한다. 학습시킬 이미지 내에서 어떤 부분이 손인지 라벨링을 한 후, 딥러닝을 진행한다. 카메라로 들어온 이미지를 서버로 전송하여 손 부분을 추출한다.

약 3000장의 이미지 중에서 손 영역을 직접 라벨링한다. 사람마다 피부색이 다르기 때문에 다양한 이미지를 사용하였다. 또한, 어떠한 제스처를 취해도 손으로 인식을 해야 하므로 여러 제스처를 취하고 있는 손도 라벨링을 하여 학습을 시켰다.

[그림 n] 손 영역 라벨링

3.1.3 손 영역 검출

이렇게 생성된 YOLO모델을 이용하여 이미지에서 손 영역을 검출한다. 입력 이미지를 S x S grid로 나누고 각 cell에서 bounding box를 그리고 confidence score를 예측한다. 이를 통해 손을 인식하고 손의 Boundary를 검출할 수 있다.



[그림 n] YOLO모델을 통한 손 영역 검출

3.2 HSV 색상 모델

3.2.1 피부색 추출

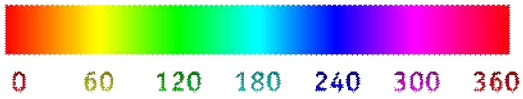
서버로부터 영상 내 손 영역의 사각형 Boundary 정보를 전달받으면 손이라고 판단되는 영역을 추측하는 Mask 이미지를 만들어 피부색 영역만을 추출한다. 이렇게 얻은 이미지에서 픽셀의 값이 0이 아닌 픽셀, 즉 손의 영역이라고 판단되는 픽셀에 대해 RGB 평균값을 구한다. 이 값을 현재 환경에서의 피부색이라고 지정하고 HSV 색상을 설정한다.

[그림 n] 손 영역 색 추출

3.2.2 HSV 색상값 설정

HSV 모델은 Hue(색조), Saturation(채도), Value(명도)의 3가지 성분으로 색을 표현한다. 사람의 피부는 대부분 붉은색 계열을 띄기 때문에 이에 대한 다양한 피부색의 편차에 유연하게 대응하기 위해 HSV 모델을 사용한다. 색상 정보에 좀 더 초점을 두되 이로 인한 과적합을 막기 위해 밝기 정보도 적절히 사용한다.

앞서 피부색 추출을 통해 얻은 RGB 색상 값을 HSV 색상 값으로 변환한다. HSV 모델에서 색상에 대한 정보만을 표시하면 [그림 n]과 같이 나타낼 수 있는데 피부색은 붉은 계열이므로 약 300 ~ 60 범위의 값을 갖게 된다. 이에 따라 현재 얻은 Hue값으로부터 일정한 범위만큼 피부색 영역이라 판단할 수 있도록 lowHue와 highHue를 설정하였다.



[그림 n] Hue 값에 따른 색상

3.2.3 손 영역 Mask 이미지 생성

추출해낸 피부색에 대해 손 영역 Mask 이미지를 최대한으로 추출해내기 위해 이미지를 스무딩하는 작업이 필요했다. 가우시안 블러 처리를 통해 이 작업을 수행하였다. 이후 지정한 HSV 범위를 이용하여 영상을 이진화하였다. 이렇게 생성한 이미지에는 앞서 수행한 가우시안 블러를 통해 처리되지 않은 잡음이 존재했으므로 Morphological 연산인 Opening과 Closing 기법을 수행하여 최종적인 Mask 이미지를 생성하였다.

[그림 n] HSV 변환을 거쳐 생성한 Mask 이미지

3.2.4 손 윤곽선 및 꼭짓점 검출

원본 이미지와 mask 이미지의 AND 연산을 통해 피부 영역에 대한 이미지를 검출하고 이 이미지에 대하여 Gray Scale을 수행, 색상을 반전시킨다. 이후 Canny Edge 검출 기법을 통해 손의 가장자리를 검출한다.

해당 이미지에 존재하는 여러 contour들을 찾고 그 각각의 contour에 대하여 블록 껍질 convex hull을 찾는다. 이때 그 블록 껍질 내에 존재하는 defect도 찾는다. 이렇게 찾은 가장 countour 중 가장 큰 contour와 그에 해당하는 convex hull, defect를 출력하면 [그림 n]과 같은 결과를 얻게 된다.

[그림 n] 손에 대한 contour, convex hull, defect

그러나 손가락 끝과 엄지손가락 주변에 꼭짓점이 필요 이상으로 많이 존재하는 것을 볼 수 있다. 본 프로젝트에서 추출해야 하는 점은 손가락마다 한 개씩이었으므로 가까이에 존재하는 여러 점을 하나의 점으로 통합해야 했다. 일정 거리(d)만큼 인접한 점이라면 같은 점으로 판단해 통합을 진행하였다. 통합해야 할 두 점 $P1(x_1,y_1)$ $P2(x_2,y_2)$의 조건은 다음과 같다.

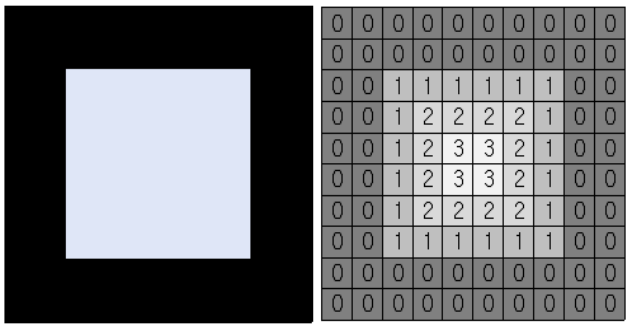
$$\sqrt[2]{(x_1-x_2)^2+(y_1-y_2)^2} < d$$

통합을 위해 그룹 테이블을 하나 생성한다. 현재 검사하는 꼭짓점이 어느 그룹에 속해 있지 않다면 나머지 꼭짓점들과 거리값을 비교하고 일정 거리 d보다 작을 경우 비교한 꼭짓점이 속한 그룹에 추가하거나 새로운 그룹을 만들어 통합을 진행하였다.

이때의 거리 d를 특정 상수값으로 정의한다면 손의 크기가 달라짐에 따라, 예를 들어 손이 카메라로부터 멀어졌을 때 대부분의 점이 하나로 통합되어버리는 문제가 발생하였다. 따라서 카메라에 비춰지는 손의 크기에 따라 이 d가 변해야 했으므로 손의 크기를 정하는 방법이 필요했다.

3.2.5 손 크기(반지름) 계산

손의 크기(카메라와 손의 거리)를 결정하는 반지름을 구하기 위해서 OpenCV의 distanceTransform 함수를 사용하였다. 이 거리 변환 함수는 이진 이미지에서 픽셀 값이 0인 배경으로부터의 오브젝트 영역의 픽셀까지의 거리를 표시하는 방법으로 배경으로부터 멀리 떨어져 있을수록 큰 값을 갖는다. 원리는 [그림 n]과 같다.

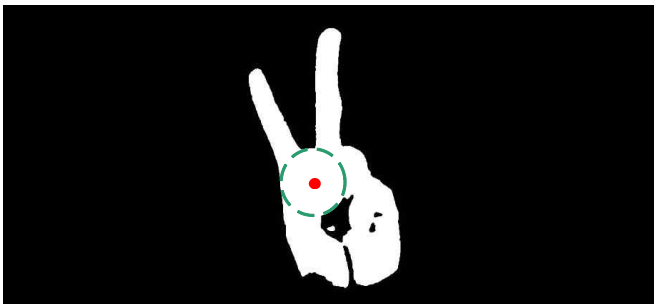


0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	0	0
0	0	1	2	2	2	2	1	0	0
0	0	1	2	3	3	2	1	0	0
0	0	1	2	3	3	2	1	0	0
0	0	1	2	2	2	2	1	0	0
0	0	1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

[그림 n] 거리 변환 행렬

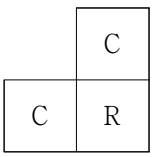
이렇게 생성된 거리 변환 행렬의 픽셀에 대해 가장 큰 값이 손의 반지름값이 된다. 또한, 가장 큰 값을 갖는 픽셀은 손의 중앙이 된다.

그러나 위와 같은 방식으로 손의 중앙을 구하고 반지름을 구할 때 손의 이진 mask 이미지에 음영이 생기면 제대로 된 값을 구할 수 없었다. [그림 n]과 같이 손 안에 그림자가 생겨 피부로 제대로 검출되지 않는 경우 손의 중앙과 반지름이 다음과 같이 생성되는 문제가 생긴다.



[그림 n] 음영으로 인해 잘못 계산된 중앙과 반지름

이 문제를 해결하기 위해서는 기존의 이진 이미지의 contour 내부의 영역은 모두 255값으로 채워주어야 했다. contour 내부의 하나의 픽셀을 선정하고 이 픽셀을 기준으로 floodFill 연산을 하여 contour 내부의 영역을 채운다. 어떤 픽셀이 contour 내부의 픽셀인지 검사할 때 현재 픽셀을 기준으로 왼쪽과 위쪽 픽셀이 contour를 이루는 픽셀이라면 현재 픽셀은 contour 내부에 있는 픽셀이라고 판단하였다.



[그림 n] contour 내부 픽셀 검사 조건

이후 이렇게 구한 픽셀에 대해 floodFill을 하면 다음과 같은 이미지를 얻을 수 있다. 이 이미지에 대해 거리 변환 연산을 수행하여 더욱 정확한 손의 반지름과 중앙점을 구할 수 있다.

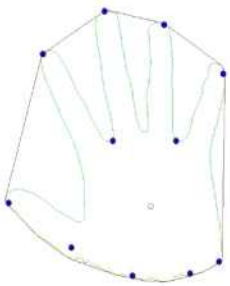
[그림 n] 내부의 음영을 제거한 이미지

3.2.6 손가락 끝 점 추출

앞서 손에 대해 꼭짓점을 구하면 수많은 점이 검출되었다. 이를 줄이기 위해 일정 거리 d만큼 인접한 점에 대해 통합하는 과정이 필요했다. 반지름을 이용하여 이 d값을 다음과 같이 계산하였다.

$$d = radius \times 0.4$$

인접한 꼭짓점을 통합하면 [그림 n]과 같은 결과를 얻을 수 있다.



[그림 n] 인접한 점끼리 통합

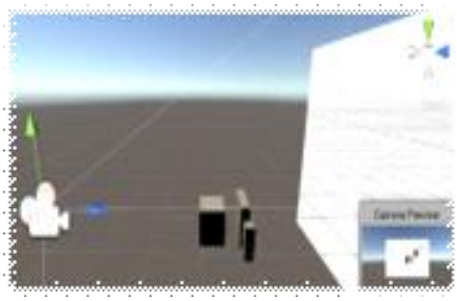
그러나 여전히 불필요한 점이 존재한다. 최종적으로 손가락 끝에 위치하는 5개의 점만 남기기 위해 이전에 구했던 손의 중앙점을 사용했다. 손가락 끝 점은 반지름을 구할 때 손의 중앙으로부터 그려지는 가상의 원의 외부에 위치하고 손가락 길이로 인해 그 거리가 다른 점보다 멀기 때문에 손의 중앙과 가장 멀리 있는 점 5개를 손가락 끝점으로 인식하여 이 점만 남겼다.

[그림 n] 최종적으로 구한 5개의 손가락 끝 점

3.3 기타 환경 설정

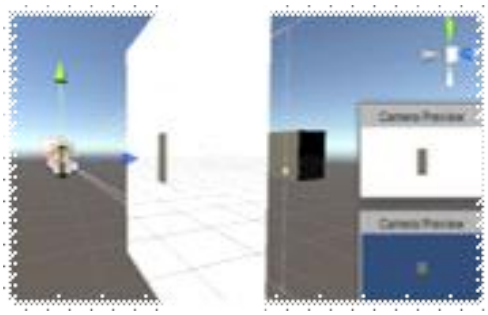
3.3.1 AR 카메라 구성

[그림 n]는 Unity 환경에서 AR 카메라로 작동하도록 카메라 1대에 오버레이한 2D Texture를 담고 있는 Canvas를 3D 공간에 두어 카메라와 상대적인 Z 좌표축을 조절하여 화면에 꽉 차도록 설계하였다. 하지만 PC, Mobile 카메라의 해상도에 따라 Z 좌표축의 거리를 달리 해야 하는 번거로움과 카메라와 Texture의 사이에만 있어야 3D 가상 오브젝트를 제어할 수 있는 한계가 있다.



[그림 n] 기존 AR 카메라 구성 방식

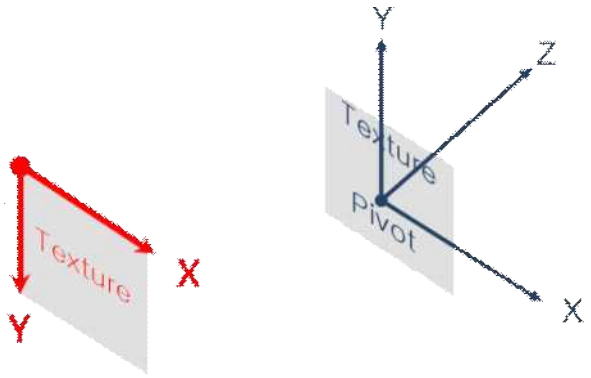
수정된 AR 카메라 구성 방식은 카메라 2대를 활용하는 것이다. 카메라 1대는 Unity 가상 환경의 카메라, 나머지 1대는 현실 환경을 비추는 카메라로 구성하였다. 현실 환경을 비추는 카메라는 프레임(60fps)마다 Texture 1장씩 빠르게 스크린에 비춘다. 이와 같은 방법으로 Unity 가상 환경의 카메라도 프레임(60fps)마다 가상 환경을 Rendering하는 Texture로 뽑아낸다. 이 두 Texture를 Layer로 우선 순위를 정해 하나의 Canvas에 오버레이 하여 두 세계를 제어 할 수 있도록 환경을 구성하였다.



[그림 n] 수정된 AR 카메라 구성 방식

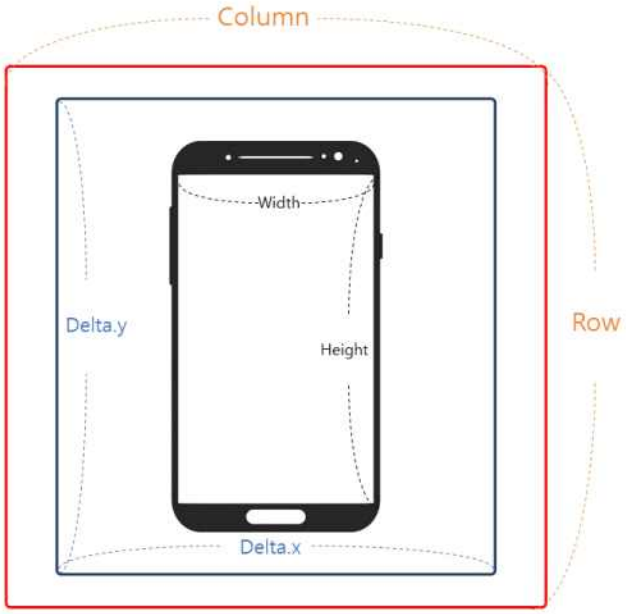
3.3.2 Unity 3D 좌표 맵핑

[그림 n]와 같이, 두 세계가 나타나는 Texture의 좌표 체계는 다르다. 행(Row : y), 열(Column : x)로 구성된 현실 카메라의 Texture는 (y, x) 좌표 체계고, Unity 가상 좌표는 Pivot에 따른 (X, Y, Z) 체계다. 손 인식과 추적에 사용되는 (y, x) 체계로 계산을 하고, 3D 가상 오브젝트를 제어하기 위해서 Unity 가상 좌표계에 이를 맵핑한다.



[그림 n] 현실 카메라의 Texture 좌표 체계(좌),
가상 카메라의 Texture 좌표 체계(우)

각 카메라와 화면의 Size는 [그림 n]와 같다. 각 카메라를 화면 해상도에 동일하게 비추도록 절차적으로 맵핑해야 한다.



[그림 n] 현실 카메라의 Texture Size(빨강), 가상
카메라의 Texture Size(파랑), 화면 Display Size(검정)

① 현실 카메라의 Texture Size를 가상 카메라의 Texture Size에 맵핑
② 현실 카메라 Texture의 상대 위치를 고려하여 가상 세계에 맵핑
③ 가상 카메라의 Texture Size를 화면 Display Size에 맵핑

①의 전처리 과정으로서 Pivot의 위치를 고정해야 한다. Pivot은 사용자 설정에 따라 변경할 수 있다. 하지만, PC의 화면과 카메라는 사용자 기준 같은 방향이고, 모바일의 화면과 뒷면 카메라는 다른 방향을 가리키기 때문에 기기에 따라 x 좌표축이 180°바뀌며, Unity 가상 세계에서 현실 카메라 Texture의 상대 위치까지 변할 수 있다. 따라서, 좌표축은 바뀌더라도 상대 위치는 일정할 수 있도록 Pivot을 중앙 위치로 고정한다.

Pivot을 중앙 위치에 고정시킨 후, 기기에 따른 현실 카메라 Texture의 좌표 $P(u,v)$의 원점을 Pivot과 일치시킨다. 이는 식으로 다음과 같이 나타낼 수 있다.

$$P'(u',v') \ : \ u' = (u/\text{cols} - 0.5)\times \text{Delta.x}$$
$$v' = (0.5 - v/\text{rows})\times \text{Delta.y}$$

여기서 가상 세계에서의 현실 카메라 Texture의 위치 Scale x좌표를 sx, y좌표를 sy라고 할 때, ②의 과정을 적용하면 식으로 나타내면 다음과 같다.

$$P'(u',v') \ : \ u' = (u/\text{cols} - 0.5)\times \text{Delta.x}\times sx$$
$$v' = (0.5 - v/\text{rows})\times \text{Delta.y}\times sy$$

마지막으로 ③과정은 Unity tool 내부의 Component에서 화면 Display에 오버레이를 할 수 있는 기능을 적용한다.

다음과 같은 식들은 모바일에서의 예시며 PC에서는 y 좌표계는 동일하며 x 좌표계가 다음과 같은 식으로 바꾸어 변환해야 한다.

$$u' = (0.5 - u/\text{cols})\times \text{Delta.x}\times sx$$

위와 같은 과정을 거쳐 최종적으로 맵핑된 결과는 [그림 n]과 같다.

[그림 n] 가상 손 생성 결과

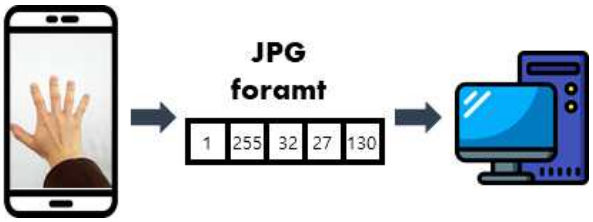
3.3.3 JPG 전송 및 손 영역 정보 수신

다양한 환경 변화에 강인한 손 인식을 위해 YOLO 알고리즘을 수행하였다. 그러나 이는 스마트폰과 같은 디바이스에서 연산하기에 연산량이 많아 시간이 오래 걸리기 때문에 PC에 서버를 두고 서버에서 연

산하였다. 따라서 스마트폰 카메라로부터 입력되는 서버로 전송한다. 환경은 지속적으로 변할 수 있기 때문에 1000 프레임마다 계속해서 서버로 영상을 전송하였다.

보통 스마트폰 카메라의 해상도는 1920×1080이며 한 픽셀이 가지고 있는 정보가 3Byte라고 하면 전송해야 할 영상의 크기는 약 6MB가 된다. 이는 전송하기에 너무 크므로 JPG 포맷으로 인코딩하였다. JPG로 인코딩하면 영상을 약 49～50KB의 크기로 압축할 수 있다.

입력 영상을 전송하기 전에 먼저 전송할 영상의 크기를 서버로 보낸다. 서버에서 영상을 수신 후 제대로 수신했는지 확인하기 위함이다. JPG로 변환된 영상을 전송할 때는 서버에서 안정적으로 수신하기 위해 1024Byte, 즉 1KB씩 분할 하여 전송하였다.

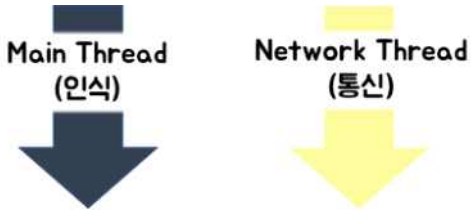


[그림 n] 입력 영상 JPG 인코딩 후 서버 전송

서버에서 연산이 완료되면 서버로부터 연산 결과를 받게 되는데 서버로부터 받은 데이터가 1Byte라면 인식이 제대로 이루어지지 않았다는 것이다. 인식을 완료하고 손 영역의 범위에 대한 데이터를 정상적으로 수신하면 3.2.1의 과정을 수행하게 된다.

[그림 n] 연산 결과(손 영역) 수신

서버에서 연산을 수행하는데 소요되는 시간은 약 3～4초로 단일 스레드 환경에서 서버에 영상을 전송하고 연산 결과를 수신하게 되면 서버에서 연산하는 동안 다음 입력 영상이 화면에 표시되지 못해 사용자의 입장에서는 애플리케이션이 정상적으로 작동하지 않는 것처럼 보인다. 이는 서버로부터 연산 결과를 수신할때까지 Block된 상태로 대기하기 때문이다. 따라서 서버에 영상을 송신하고 결과를 수신하는 작업은 카메라로부터 입력 영상을 받고 HSV 색상 모델을 통해 손을 추적하는 연산과 병렬적으로 이루어져야 했기 때문에 통신을 위한 스레드를 생성하였다.

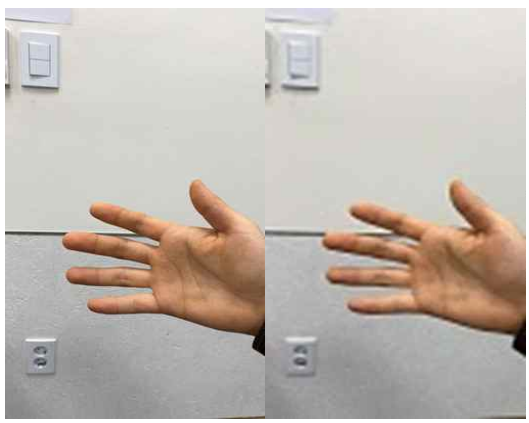


[그림 n] 인식과 통신 병렬 처리

3.3.4 이미지 Resize를 통한 속도 향상

PC에서는 HSV 색상 모델을 통해 손을 추적하는 작업이 실시간으로 이루어졌지만 스마트폰에서 같은 연산 수행에 대해 오랜 시간이 걸리는 것을 확인할 수 있었다. 스마트폰 CPU의 성능이 PC에 비해 떨어지기 때문이었다. 또한 스마트폰 카메라의 해상도가 PC의 웹캠보다 훨씬 크다는 것도 있었다. 해상도가 커지면서 입력 영상의 크기가 커지면 늘어난 픽셀의 개수만큼 처리해야 할 연산량이 늘어난다. 스마트폰 카메라로부터 입력된 원본 영상의 크기는 1920×1080이다. 이는 영상을 한 번 처리할 때 픽셀을 2,073,600번 접근하게 된다. 스마트폰에서도 실시간으로 손을 추적하기 위해서는 연산량을 줄일 필요가 있었다.

주 연산이 색상을 통해 손을 추적하는 것이므로 해상도가 인식의 정확도에 큰 영향을 미치지 않는다. 따라서 연산량을 줄이기 위해 입력 영상의 크기를 재조정하여 연산량을 대폭 감소시킬 수 있다. 원본 영상의 비율이 16:9이므로 이 비율에 맞추어 적절한 resize 크기를 설정하였다. 입력 영상의 크기를 최소한으로 줄이면서 허용할 수 있는 정확도를 얻을 수 있도록 resize 크기를 240×135로 설정하였다.



[그림 n] resize 전 영상(좌)과 resize 후 영상(우)

4. 결과 분석

YOLO 모델을 통한 HSV 색상 모델 재설정은 주변 환경에 기존 색상 모델보다 강인하다. 이에 대한 성능은 다음과 같이 측정한다.

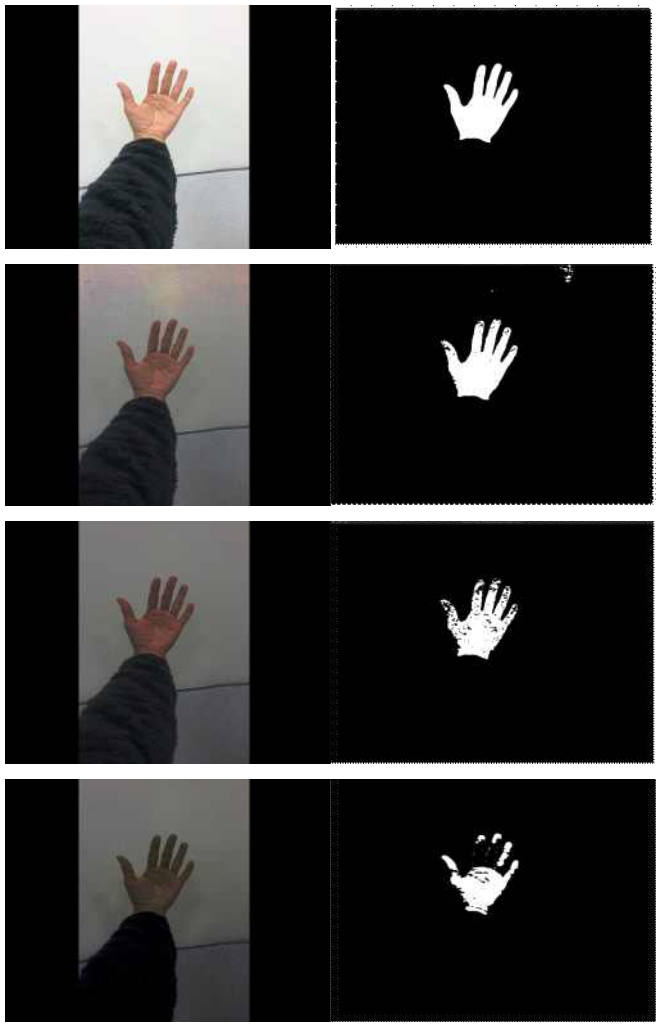
주변 환경은 밝기 조절을 통해 변화를 준다. 주변 밝기가 다양한 환경을 실제로 구성하기 어렵기 때문에 실험 사진의 밝기를 조절하며 진행하였다.

[그림 n] YOLO-HSV 통합 모델 성능 측정에 사용된 실험 이미지(밝기 100)

위의 사진을 밝기-100 이미지라고 할 때 밝기 조절된 이미지들은 다음과 같다.

[그림 n] 밝기 조절된 실험 이미지
(왼쪽 위=밝기 100, 오른쪽 위=밝기 50
왼쪽 밑=밝기 30, 오른쪽 밑=밝기 10)

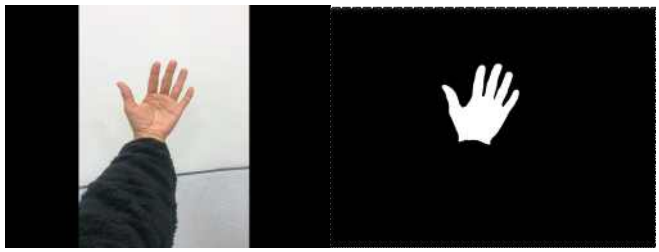
상수화된 임계값을 사용하던 기존의 색상 모델을 사용해 추출한 마스크 이미지는 다음과 같다.

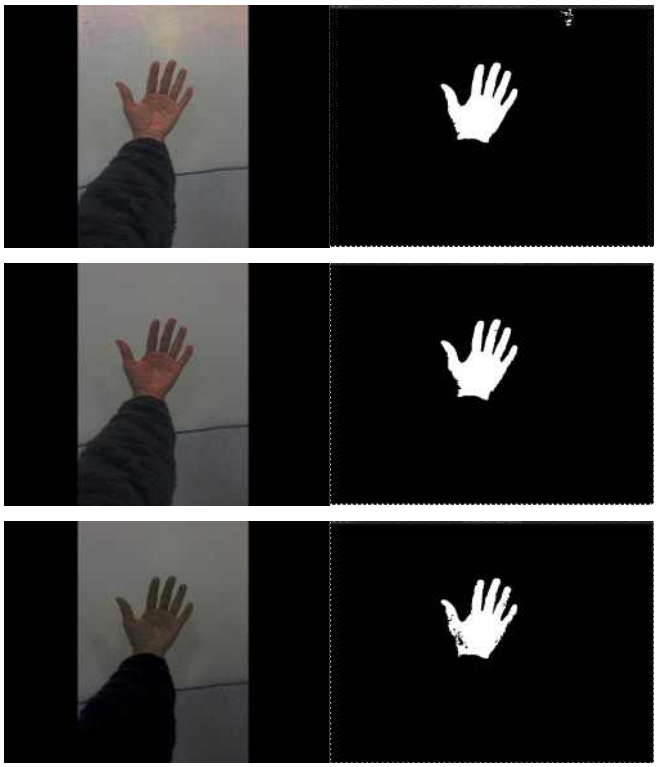


[그림 n] 기존 색상 모델을 통한 피부색 검출

밝고 손 영역이 선명한 사진에서는 좋은 성능을 보여주지만 사진이 어두워지면서 잡음이 많아지는 모습을 보였고 손의 모양 역시 제대로 표시되지 않는 것을 볼 수 있다. 사진의 밝기에 따라 성능이 크게 차이 나는 것을 육안으로도 확인할 수 있다.

다음은 YOLO-HSV 통합 모델을 사용했을 때의 피부색 마스크 이미지이다.





[그림 n] YOLO-HSV 통합 모델을 통한 피부색 추출

YOLO-HSV 통합 모델에서는 기존의 색상 모델보다 잡음이 많이 없어지고 손의 모양도 계속 유지가 되는 것을 확인할 수 있다. 그렇기 때문에 우리가 YOLO를 통해 강화한 색상 모델은 주변의 밝기 등에 대한 환경 변화에 큰 영향을 받지 않고 좋은 성능을 보여줄 수 있을 것으로 기대된다.

5. 결론

본 논문에서는 YOLO 알고리즘을 통해 주변 환경 변화에도 정확하게 손을 인식하고 HSV 색상 모델을 사용하여 실시간으로 추적하는 방법에 대해 제시하였다. 최종적으로 증강현실에서 가상의 3D 물체를 손으로 직접 건드릴 수 있었다. 속도와 정확도가 향상될수록 AR 환경에 대한 제어가 용이해질 것으로 판단한다. 이는 Google Glass와 같이 점점 경량화되면서 상용화되고 있는 HMD에 내장 소프트웨어로 적용되면 사용자에게 AR 환경 제어에 있어 더욱 편리한 사용자 경험을 제공할 것으로 기대된다. 또한 스마트폰이나 컴퓨터 웹캠을 통해서도 [그림 n]과 같이 다양한 오락 기능으로 발전하여 게임에 새로운 패러다임을 제시할 것이다.

[그림 n] 농구 게임(좌)과 인형뽑기 게임(우)

참고 문헌

[1] Pei Xu, *A Real-time Hand Gesture Recognition and Human-Computer Interaction System*, 2017

[2] In-kyu Choi · Jisang Yoo, *Hand shape recognition based on geometric feature using the convex-hull,* 2014

[3] Soohwan Chae, Kyungkoo Jun, *HSV Color Model based Hand Contour Detector*, 2015

[4] ANKIT SACHAN, *Zero to Hero: Guide to Object Detection using Deep Learning: Faster R-CNN, YOLO, SSD*, https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/

[5] 이호성, *Fast hand gesture recognition using CNN andedge detection*, 2018

[6] Wei Liu, *Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. BergSSD: Single Shot MultiBox Detector*, 2016

[7] Joseph Redmon, Ali Farhadi, *YOLO9000: Better, Faster, Stronger*, 2016

[8] Seongkyun Han's blog, Jan 06 2019, *[Object detector] R-CNN/Fast R-CNN/Faster R-CNN/SSD 가볍게 알아보기* (June 28 2020), https://seongkyun.github.io/papers/2019/01/06/Object_detection/

[9] 꾸준희, Oct 16 2019, [Object Detection] *darknet custom 학습하기* (May 18, 2020), https://eehoeskrap.tistory.com/370

[10] Jonathan Hui, Mar 28 2018, *What do we learn from single shot object detectors (SSD, YOLOv3), FPN & Focal loss (RetinaNet)?* (June 28 2020), https://medium.com/@jonathan_hui/what-do-we-learn-from-single-shot-object-detectors-ssd-yolo-fpn-focal-loss-3888677c5f4d

[11] 이상현, Dec 26 2017, *Deep Learning(YOLO) 기반의 Smart Scarecrow (May 20 2020)*, https://pgmrlsh.tistory.com/category/Deep%20Learning%28YOLO%29기반의%20Smart%20Scarecrow