

2022년도 전자공학과 졸업작품 결 과 보 고 서

시각장애인의 폭넓은 대인관계를 위한 사람 인식 어플리케이션

2022년 11월 10일

지도 교수 : 임정수 교수님

팀 명 : 히사이시조

참 여 자 1 : 1626077 장휘재

참 여 자 2 : 1729039 추은호

목 차

I. 작품 개요	3
II. 관련 이론	4
1. OpenCV	4
2. Face Detection	6
3. 음성 인식	7
4. 음성 합성	8
III. 개발 방법	9
1. 소프트웨어	9
2. 하드웨어	9
IV. 개발 내용	10
1. 소프트웨어	10
2. 하드웨어	14
V. 결론	15
부록	3

I. 작품 개요

1. 작품 개발 배경

시각 장애인이 사람을 구별할 때, 목소리에 의존하는 경우가 많아 주변 소음이 큰 경우, 목소리를 통해 사람을 구별하기 어렵고 기억에 의존하여 사람을 구별하기에, 어려움을 느낄 수 있다.

시각 장애인이 상대방의 얼굴 생김새 혹은 특징에 대한 궁금증을 갖는 경우, 이러한 궁금증을 해소하기 위해서는 상대방의 얼굴을 직접 구석구석 만지거나 목소리로만 유추가능하며 시각장애인이 유추한 상대방의 얼굴과 실제 얼굴간의 차이를 시각장애인이 흥미로운 시각으로 바라보는 영상들이 있으며 지인의 변화되는 모습을 직접 보고 싶어 한다고 한다. (시각장애인 사진작가 김우림씨 인터뷰와 유튜브 ‘원샷한술’)

뿐만 아니라 시각장애인의 고충을 여러 매체를 통해 조사한 결과 눈이 보이게 된다면 가장 먼저 보고 싶은 것으로 본인의 얼굴, 사랑하는 가족의 얼굴, 주변 지인들의 얼굴을 선택했다. (유튜브 ‘원샷한술’, ‘이정화’ 씨 SBS 뉴스 인터뷰)

시각 장애가 생긴 이후, 사회에서 독립적으로 살아가는 과정, 특히 새로운 인간관계 형성에 어려움을 겪어 이에 관한 고민을 하고 있다. (유튜브 ‘당장만나’의 시각장애인 ‘우령’ 씨 인터뷰)

위의 이러한 문제와 궁금증들을 작게나마 해소하고 싶어 다음과 같은 주제로 작품의 개발을 시작하였다.

2. 작품의 목적

작품의 목적은 다음과 같았다.

1. 시각장애인들의 상대방의 외적인 모습에 대한 궁금증 해결
2. 시각장애인들의 자신감 있는 사회생활 도모 및 인간관계 형성
3. 오랜만에 만난 지인들의 모습 확인 및 기록
4. 소음이 심한 곳에서의 사람 구별

3. 최종 목표

작품의 최종 목표는 해당 작품을 사용함으로 문제없이 상대방을 인식하고 외적인 모습을 알아내 이를 통해 시각장애인들의 여러 궁금증이나 불편함 등을 해소하는 것과 동시에, 기존의 제품과 비교하였을 때, 제공하는 정보의 양, 편리성, 저장 가능성 등의 성능에서 차별성과 경쟁력을 갖는 것을 최종 목표로 설정하였다.

II. 관련 이론

1. OpenCV 영상처리

OpenCV는 오픈 소스 컴퓨터 비전 라이브러리로 실시간 컴퓨터 비전, 이미지 프로세싱을 목적으로 한 프로그래밍 라이브러리이다. 실시간 처리에 중점을 두고 설계되었기에 빠른 속도와 효율성을 자랑하며 기반언어는 C++로 멀티 코어 프로세서를 활용할 수 있다.

1.1 cascade classifier

OpenCV가 제공하는 대표적인 상위레벨 API로 다수의 객체 이미지(positive image)와 객체가 아닌 이미지(negative image)를 cascade함수로 트레이닝 시켜 객체 검출을 달성하는 머신러닝 기반의 접근법으로 직접 머신러닝 학습 알고리즘을 사용하지 않고도 훈련된 검출기를 xml파일 형태로 제공하여 객체를 검출할 수 있도록 한다. 처음 구현될 때, Haar feature 기반으로 구현되어 Haar cascade로도 알려져 있지만 다른 feature를 직접 작성해 사용할 수도 있다.

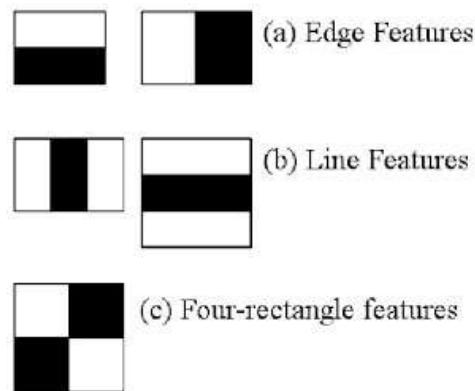


그림 1 Haar features

Haar feature은 convolutional 커널과 비슷한 아래와 같은 이미지를 이용하며, 각 특징(feature)은 검은색 사각형 아래 픽셀 값의 합에서 흰색 사각형 아래 픽셀 값의 합을 빼서 얻은 값이다.

이때, 이미지에서 특징들을 검출하기 위해 가능한 모든 크기의 커널을 이미지의 모든 부분에 적용해야하는데, 이는 비효율적이므로, 모든 과정을 여러 단계로 그룹화하여 적용하는 것이 cascade of classifier의 원리이다. 이를 통해 어떤 단계를 통과하지 못하면 그 다음 단계를 진행하지 않도록 하여 효율성을 높인다.

1.2 LBP

LBP는 Local Binary Pattern의 약자로 지역적인 이진 패턴을 계산하여 이미지의 질감 표현 및 얼굴 인식 등에 활용되는 간단하지만 효율적인 알고리즘이다.

3x3셀 내에서 중심의 픽셀과 이웃하는 8개의 픽셀의 값을 비교하여 중심의 픽셀 값보다 크거나 같으면 1, 작으면 0으로 threshold해준다. 즉, 중심의 픽셀 값을 임계점으로 threshold한다. 이를 순서대로 나열하면 10110010과 같이 8비트(8자리 이진수)의 값을 얻을 수 있고 이를 십진수 값으로 계산하면 0부터 255 사이의 값을 얻을 수 있고, 모든 픽셀에 대해 계산하여 histogram을 구성한다. 즉, 이 과정으로 하나의 이미지(영상)의 질감을 256차원의 벡터(특징 벡터)가 만들어진다.

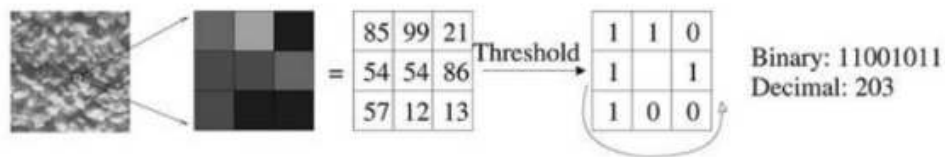


그림 2 LBP의 원리

이러한 LBP histogram은 한 픽셀을 중심으로 주변 픽셀을 비교하여 계산되기 때문에 단순한 밝기(명암)변화에 강인하다는 특성을 갖는다.

OpenCV의 LBPHFaceRecognizer는 이러한 LBPH의 특성을 이용하여 만들어진 특징벡터를 분류기의 학습 데이터로 사용해 사용자의 얼굴을 분류한다.

1.3 색 공간

색 공간(color space)는 색 표시계(color system)를 3차원으로 표현한 공간 개념을 의미하며 색 표시계의 모든 색들은 이 색 공간에서 3차원 좌표로 표시된다. 어떤 색 공간을 사용하는 지에 따라 각 축이 의미하는 요소가 달라진다.

OpenCV에서는 다양한 색 공간으로 변환할 수 있으며 대표적으로 RGB 색 공간과 HSV 색 공간이 있다.

RGB 색 공간은 색을 혼합하면 명도가 올라가는 가산 혼합 방식(빛의 삼원색)으로 색을 표현한다. 각 축은 삼원색인 적색(RED), 녹색(GREEN), 청색(Blue)을 의미하며 세 가지 채널의 밝기를 기준으로 값을 지정한다. 보편적으로 사용되고 있다.

OpenCV에서는 RGB가 아닌 BGR의 순서로 색상을 인식하며 0~255 사이의 정수로 밝기를 표현한다. 예를 들어, 원색의 빨간색은 RGB 값으로 (255, 0, 0)이지만, BGR 값으로는 (0, 0, 255)의 값을 갖는다.

HSV 색 공간은 색조(Hue), 채도(Saturation), 명도(Value)를 기준으로 색을 구성한다. 색조는 색의 계열로 해당 색이 붉은색 계열인지 푸른색 계열인지를 나타내며, 채도는 해당 색이 얼마나 선명한(순수한) 색인지를 의미해 낮을수록 무채색, 높을수록 선명한 색이 된다. 명도는 밝기(intensity)를 의미하며 낮을수록 색상이 어두워지고 높을수록 밝아진다.

OpenCV에서 H는 0~179 사이의 정수로, S와 V는 0~255 사이의 정수로 표현된다. 색조가 0~179의 범위로 표현되는 이유는 일반적인 HSV에서 H는 0~360°의 각도로 표현되지만 해당

값을 저장하는 자료형은 256이상의 정수를 표현할 수 없기 때문에 각도를 2로 나눈 값을 H로 설정하여 값을 구성한다.

RGB와 HSV의 이러한 차이점 때문에 색의 검출과 분리와 같은 영상처리에 대해서는 HSV 색 공간을 사용하는 것이 RGB보다 직관적이고 효과적일 수 있다.

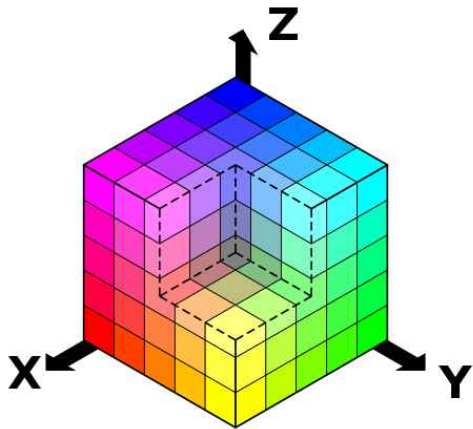


그림 3 RGB 색 공간

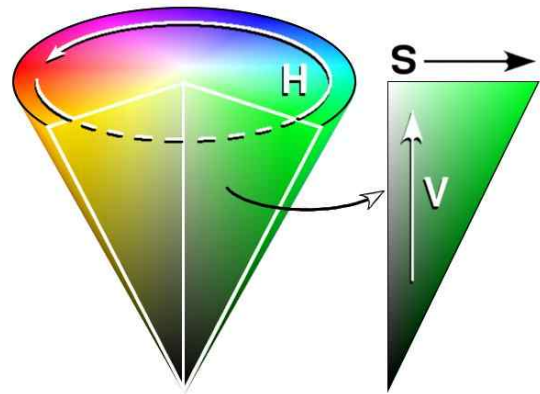


그림 4 HSV 색 공간

2. Face Detection

2.1 Dlib를 이용한 특징점 검출

이미지 처리 및 기계 학습, 얼굴 인식 등을 진행할 수 있는 C++로 개발된 고성능의 라이브러리이다. OpenCV와 비슷한 기능을 갖는 듯 보이지만, 얼굴인식에 관해서는 dlib가 강력한 기능을 보이며, dlib의 기능을 사용하기 위한 영상처리에 대해서는 opencv가 많은 역할을 하기에 두 라이브러리를 함께 사용하는 것이 좋다. 이 중에 얼굴 검출 기능은 눈, 코, 입, 윤곽과 같은 주요 지점에서 특징점(land mark)을 뽑아 좌표를 반환하는 역할을 하며 작품 내에서 눈, 코, 입의 크기 측정을 위해 사용되었다.

2.1.1 HOG(Histogram of Oriented Gradients) 특성

HOG는 픽셀값의 변화로 파악할 수 있는 영상 밝기 변화의 방향을 그래디언트(gradient)로 표현한 것인데, 이로부터 객체의 형태를 찾아낼 수 있다. 우선 영상은 기본적으로 픽셀들의 집합으로 되어있다. 이러한 픽셀들을 묶어서 소그룹을 만들면 Cell 이고, 이러한 Cell 들을 다시 묶어서 그룹을 만들면 Block 이 되는 것을 먼저 알아야 한다. HOG는 대상 영역을 일정 크기의 셀로 분할하고, 각 셀마다 edge 픽셀(gradient magnitude(변화도 광도)가 일정 값 이상인 픽셀)들의 방향에 대한 히스토그램을 구한 후 이들 히스토그램 bin 값들을 일렬로 연결한 벡터이다. 즉, HOG는 edge의 방향 히스토그램 템플릿으로 볼 수 있다. 템플릿 매칭(template matching)의 경우에는 원래 영상의 기하학적 정보를 그대로 유지하며 매칭을 할 수 있지만 대상의 형태나 위치가 조금만 바뀌어도 매칭이 잘 안되는 문제가 있다. 반면에 히스토그램 매칭은 대상의 형태가 변해도 매칭을 할 수 있지만 대상의 기하학적 정보를 잃어버리고 단지

분포(구성비) 정보만을 기억하기 때문에 잘못된 대상과도 매칭이 되는 문제가 있다.

HOG는 템플릿 매칭과 히스토그램 매칭의 중간 단계에 있는 매칭 방법으로 볼 수 있으며 블록 단위로는 기하학적 정보를 유지하되, 각 블록 내부에서는 히스토그램을 사용함으로써 로컬한 변화에는 어느정도 강인한 특성을 가지고 있다.

또한 HOG는 edge의 방향정보를 이용하기 때문에 일종의 edge기반 템플릿 매칭 방법으로도 볼 수 있다. Edge는 기본적으로 영상의 밝기 변화, 조명 변화 등에 덜 민감하므로 HOG 또한 유사한 특성을 갖는다고 생각할 수 있다. 또한 HOG는 물체의 실루엣(윤곽선) 정보를 이용하므로 사람, 자동차 등과 같이 내부 패턴이 복잡하지 않으면서도 고유의 독특한 윤곽선 정보를 갖는 물체를 식별하는데 적합한 영상 feature이다.

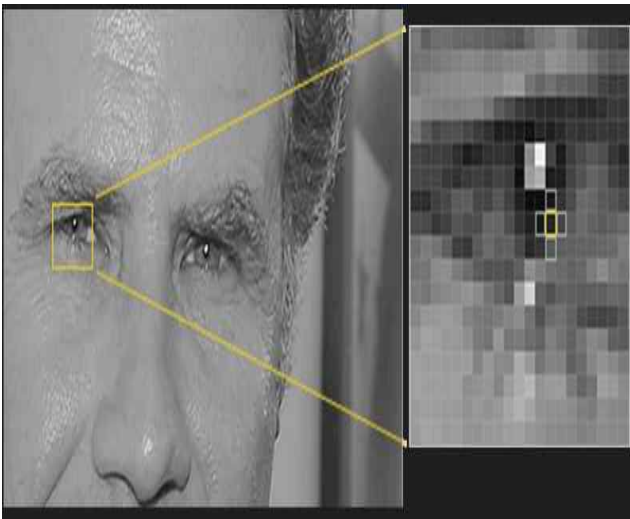


그림 1. 이미지 단일 픽셀1

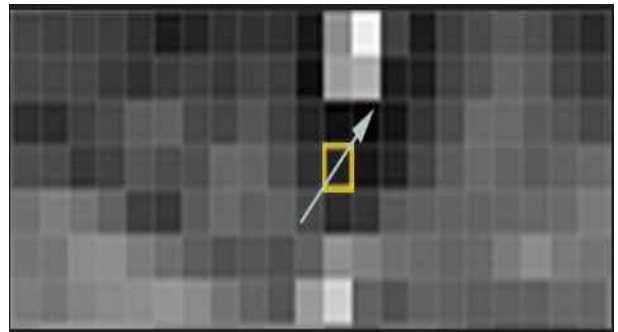


그림 2. 이미지 단일 픽셀2

위 그림에서처럼 이미지의 모든 픽셀에 대해 이 프로세스를 반복하면 결국 모든 픽셀이 화살표로 바뀌게 된다. 이러한 픽셀들을 그래디언트(gradients)라고 부르고, 이를 통해 전체 이미지에서 밝은 부분으로부터 어두운 부분으로의 흐름을 알 수 있다. 픽셀을 그래디언트로 바꾸는 이유는 픽셀을 직접 분석하면, 동일한 사람의 어두운 이미지와 밝은 이미지는 전혀 다른 픽셀값을 갖게 될 것이다. 그러나 밝기가 변하는 방향만 고려하면 어두운 이미지와 밝은 이미지에 대한 완전히 동일한 표현을 얻게 되어 문제를 더 훨씬 쉽게 해결할 수 있다. 그러나 모든 단일 픽셀에 대해 그래디언트를 저장하면 너무 자세하기 때문에 이미지의 기본 패턴을 알 수 있도록, 높은 수준에서 밝음/어두움의 기본 흐름만을 보는 것이 더 좋다. 이를 위해 이미지를 각각 16x16 픽셀의 작은 정사각형들로 분해하고, 각 정사각형에서 그래디언트가 주요 방향(우상향)을 얼마나 가리키고 있는지 세어 본 다음에 이 사각형들을 가장 강한 화살표 방향으로 바꾼다.

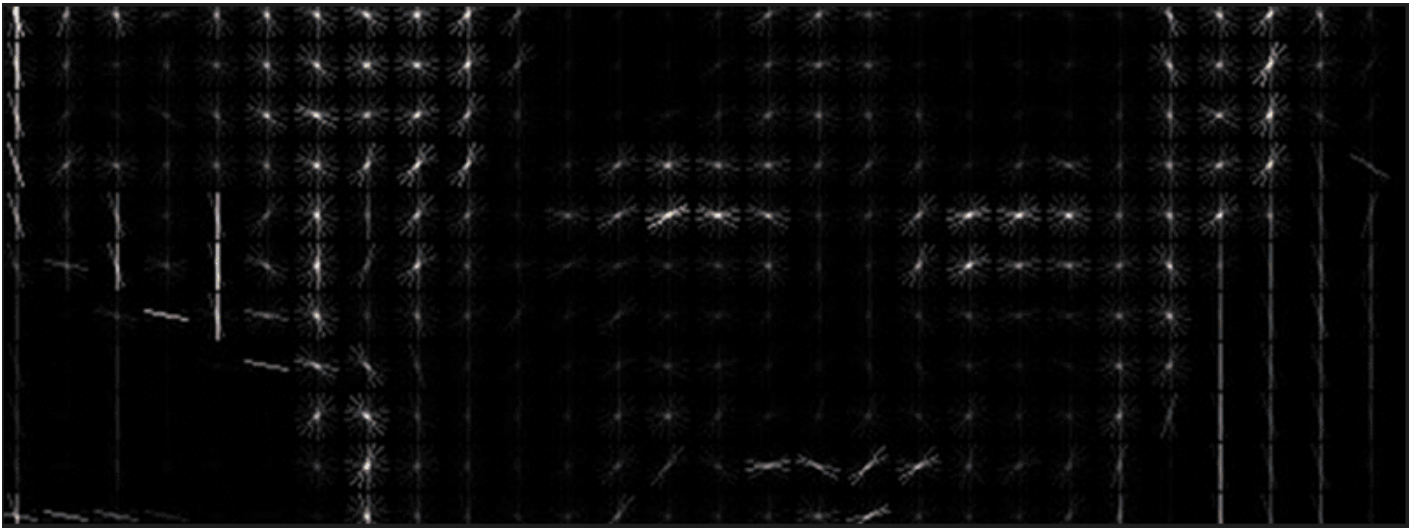


그림 ? 그래디언트를 이용해서 HOG 표현을 한 이미지



그림 ? HOG 표현 과정

첫 번째 단계는 감마 보정이다. 여기서 감마 값을 조정하는 것은 밝기(혹은 휘도)를 조절하는 기능으로, 픽셀 값을 비선형적으로 보정한다. 다음과 같은 수식을 통해서 적용할 수 있다.

$$V_{out} = A V_{in}^{\gamma}$$

두 번째 단계에서는 픽셀의 gradient를 구한다. 이 과정에서는 가장 간단한 1D Kernel $[-1, 0, 1]$, $[-1, 0, 1]^T$ 을 사용한다. 이와 같은 커널을 적용시켜 x와 y의 변화량에 대해서 구한다. 이렇게 얻은 바탕으로 dy/dx 값을 구하고 \tan^{-1} 즉, arctangent를 거치면 gradient의 방향이 나오게 된다. 여기서 arctangent를 사용하는 이유는 sin과 cos는 대각의 값을 구해야 하고 그 과정에서 루트와 같은 비싼 연산이 등장하기 때문이라고 한다.

세 번째 단계는 두 번째 과정에서 gradient를 병합하는 단계이다. 우선 이미지를 $\square \times \square$ 크기의 셀로 나눈다. 이렇게 셀로 나누어 표현하는 이유로는 표현이 compact 해지고, 잡음에 강해지기 때문이다.

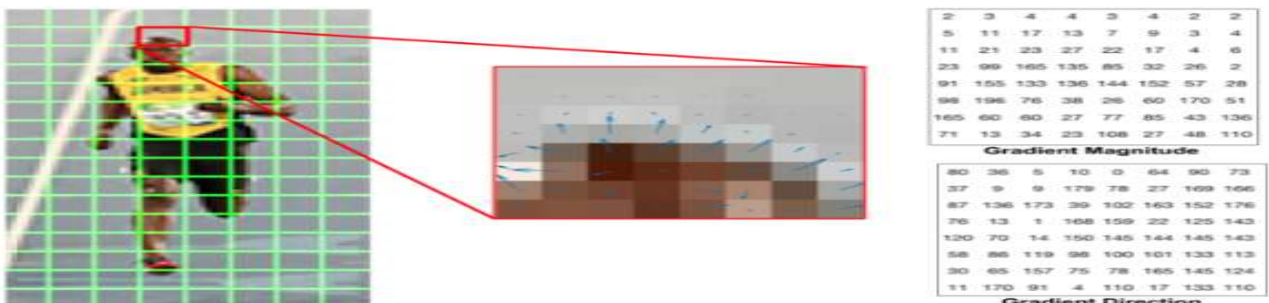


그림 ? 8x8 크기로 셀을 나눈 후, 하나의 셀에 대한 gradient 값과 방향을 표현한 이미지

이제 gradient를 각도의 범위 별로 나누어 경향성을 histogram에 vote할 것이다. 각도의 범위는 0~180으로 20도씩 나누는 것이 실험적으로 좋다고 하는데, 일반적으로 각도의 범위는 360도이지만, 180도와 360도는 같은 직선 내에 있기 때문에 0~360이 아닌 -180~180으로 보고 절대값을 취한 범위만 이용하면 되기 때문이다. 다음은 vote 과정이다.

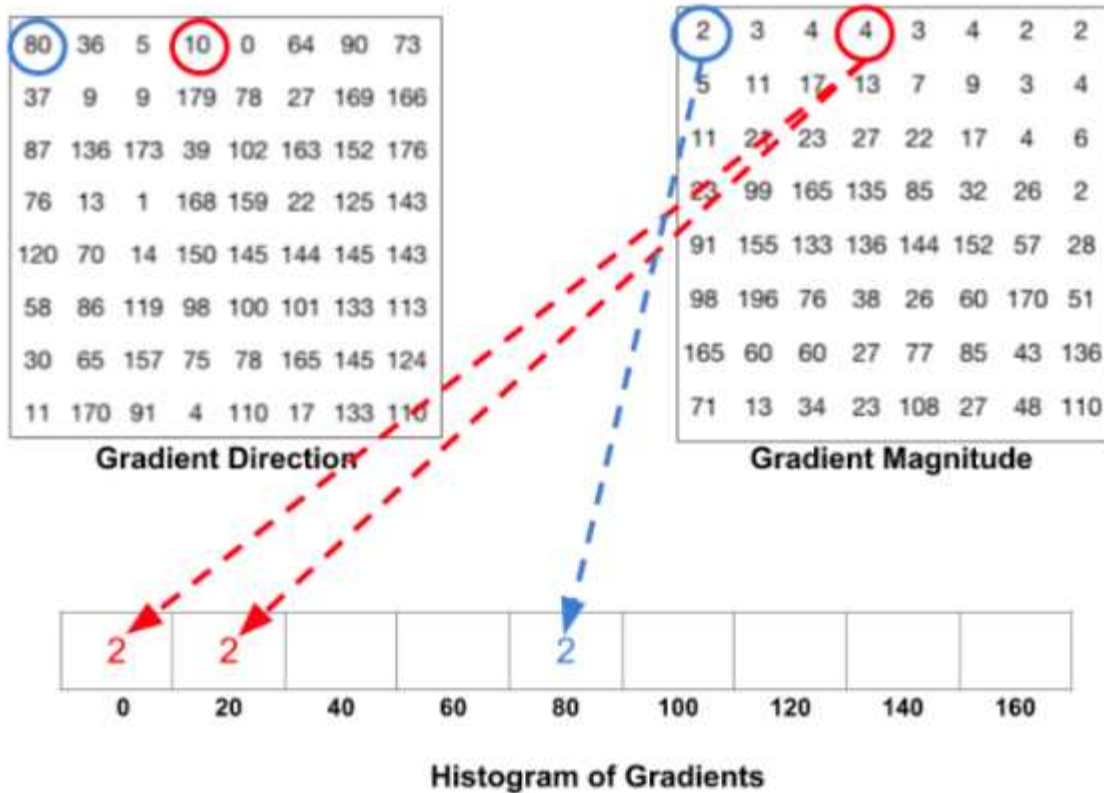


그림 ? 히스토그램에 vote 하는 과정

네 번째 단계에서는 히스토그램을 정규화 시켜준다. gradient는 조명에 민감하기 때문에 만약에 값이 조금만 바뀌더라도 히스토그램의 경향성이 급변하기 때문에 조명에 따라 값이 잘 바뀌지 않게 해주기 위함이다.

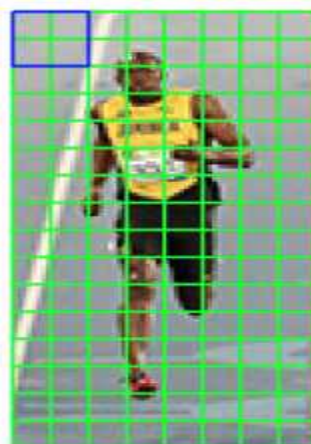


그림 ? 히스토그램을 정규화해준 이미지

다섯 번째 단계는 detection window에 대해서 HOG를 얻는다. 이미지를 보면 사람의 몸통 방향에 맞게 HOG가 모아졌다.



그림 ? (36크기의 히스토그램) X (가로 7개 X 세로 105개) = 3780차원 벡터를 얻은 이미지

여섯 번째 단계는 분류기 실행이다. SVM(Support Vector Machine)은 인공지능의 기계학습 분야 중 하나로, 패턴인식이나 자료분석을 위한 지도학습 모델이다. 즉, 2개의 범주를 분류하는 이진 분류기이다. 주로 분류와 회귀 분석을 위해 사용되며, SVM 알고리즘은 주어진 데이터 집합을 바탕으로 하여 새로운 데이터가 어느 카테고리에 속할 것인지 판단하는 비확률적 이진 선형 분류 모델을 만들게 된다. SVM의 기본적 원리는 흰색 바둑알과 검은색 바둑알이 학습용 데이터로 주어졌다고 했을 때, 두 그룹에서 각각의 데이터 간 거리를 측정하여 두 개의 데이터 사이의 중심을 구한 후에 그 가운데에서 최적의 초평면(Optimal Hyper Plane)을 구함으로써 흰색과 검은색 그룹을 나누는 방법을 학습하게 된다. 여기서 직선으로 나눌 수 있다면 선형 분류 모델을 적용하고, 직선으로 나눌 수 없는 경우 비선형 분류 모델을 사용하게 된다.

이와 같은 학습 분류기를 통하여 “사람이다” 또는 “사람이 아니다” 라는 것을 판별할 수 있게 해준다.

2.1.2 Face Landmark

위 과정을 통해 이미지에서 얼굴들만 분리해 낸다. 이때, 얼굴이 정면이 아닌 다른 방향을 보고 있으면 컴퓨터에게는 전혀 다르게 인식하기 때문에 이를 해결하기 위해서 각각의 사진을 비틀어 눈과 입술 등 얼굴의 랜드마크가 항상 표준 위치에 올 수 있도록 위치를 교정해줘야 한다. 이렇게 하기 위해선 face landmark estimation 이라고 하는 알고리즘을 사용한다.

[연구논문] One Millisecond Face Alignment with an Ensemble of Regression Trees - Vahid Kazemi and Josephine Sullivan KTH, Royal Institute of Technology Computer Vision and Active

Perception Lab Teknikringen 14, Stockholm, Sweden 참조.

기본적인 아이디어는 모든 얼굴에 존재하는 68개의 landmarks라 부르는 특정 포인트들을 찾아내는 것인데, dlib가 제공하는 기계 학습 알고리즘을 훈련시킨 라이브러리를 사용하여 68개의 특정 포인트들을 찾을 수 있다.

2.2 Mediapipe를 이용한 Face Mesh 검출

MediaPipe는 구글에서 제공하는 AI 프레임워크로, 비디오 형식 데이터를 이용한 다양한 기능을 파이프라인 형태로 손쉽게 사용할 수 있도록 한다. AI 모델 및 수많은 데이터셋을 이용한 학습이 완료된 상태로 제공되므로 라이브러리를 불러 사용하듯 간편하게 호출하여 사용하면 되는 형태로, 비전 AI기능을 개발할 수 있다.

이 중, Face Mesh 기능은 모바일 기기에서도 실시간으로 468개의 3D 얼굴 랜드마크를 추정할 수 있는 솔루션이다. 머신 러닝을 사용하여 3D 얼굴 표면을 추론하므로 단일 카메라 입력만 있으면 사용가능하다. 작품에서는 얼굴형 검출을 위한 방법으로 사용되었다. 일반적인 얼굴 키포인트 감지는 68개(x,y)를 계산하지만 Face Mesh 는 468개의 포인트(x,y,z)를 제공한다.

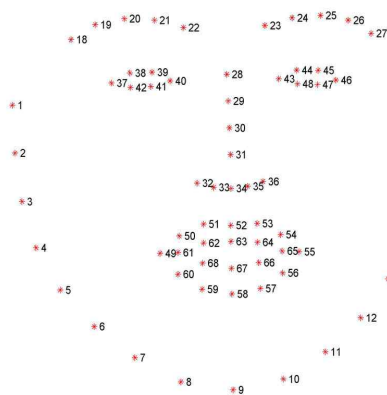


그림 13 Dlib를 사용하여 뽑아낸
68개 랜드 마크

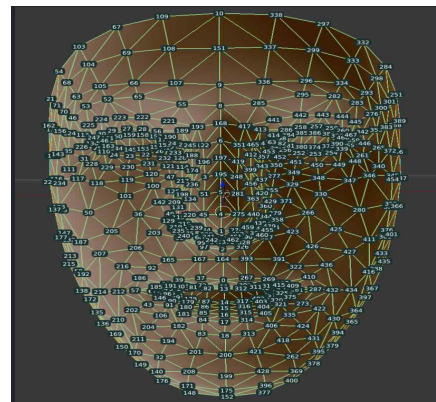


그림 14 MediaPipe를 이용해
찾아낸 468개의 랜드 마크

얼굴 랜드마크 모델은 화면 좌표 공간에서 단일 카메라 얼굴 랜드마크 감지를 수행한다. X와 Y 좌표는 정규화된 화면 좌표이며, Z 좌표는 상대적이며 약한 투시 투영 카메라 모델에서 X 좌표로 스케일링된다. 이 형식은 일부 애플리케이션에 적합하지만 가상 3D 객체를 감지된 얼굴과 정렬하는 것과 같은 증강 현실(AR) 기능의 전체 스펙트럼을 직접 사용할 수는 없다. 얼굴 변환 모듈은 화면 좌표 공간에서 메트릭 3D 공간으로 이동하며 감지된 얼굴을 일반 3D 개체로 처리하는 데 필요한 기본 요소를 제공한다. 설계상 투시 카메라를 사용하여 얼굴 랜드마크 위치가 변경되지 않도록 보장하면서 최종 3D 장면을 화면 좌표 공간에 투영할 수 있다.

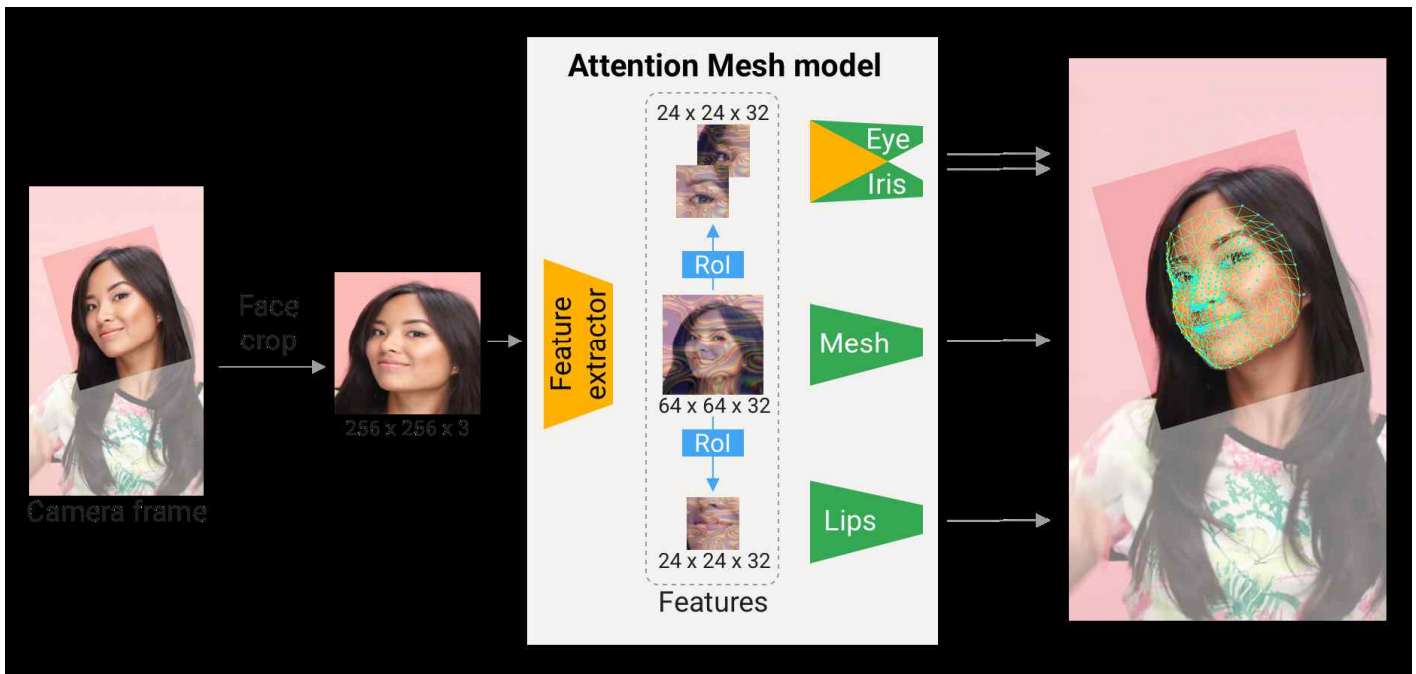


그림 ? Attention mesh model 원리

주요 개념으로는 Metric 3D space 이 있다. 이는 얼굴 변환 모듈 내에 설정된 메트릭 3D 공간은 right-handed 또는 법선 메트릭 3D 좌표 공간이다. 공간 내에는 공간 원점에 위치하고 Z축의 음의 방향을 가리키는 가상 투시 카메라가 있다. 현재 파이프라인에서 입력 카메라 프레임은 정확히 이 가상 카메라에 의해 관찰되므로 나중에 화면 랜드마크 좌표를 메트릭 3D 공간으로 다시 변환하는 데 사용된다고 가정한다. 가상 카메라 매개 변수는 자유롭게 설정할 수 있지만 더 나은 결과를 얻으려면 가능한 실제 물리적 카메라 매개 변수에 가깝게 설정하는 것이 좋다.

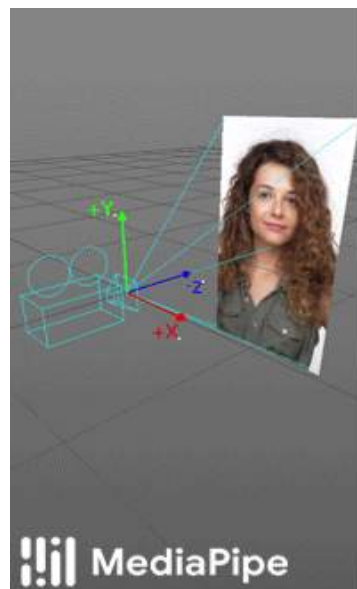


그림 ? Metric 3D space의 여러 주요 요소들의 시각화

Canonical face model(표준 얼굴 모델)은 얼굴 랜드마크 모델의 468 3D 얼굴 랜드마크 토폴로지를 따르는 사람 얼굴의 정적 3D 모델이다. 모델에는 두 가지 중요한 기능이 있다. 첫 번째는 메트릭 단위 정의이다. 표준 면 모델의 척도는 메트릭 3D 공간의 메트릭 단위를

정의하고, 기본 표준 얼굴 모델에서 사용되는 미터법 단위는 cm이다.

두 번째는 정적 공간과 런타임 공간의 연결이다. 얼굴 포즈 변환 매트릭스는 실제로 표준 얼굴 모델에서 각 프레임에 추정된 런타임 얼굴 랜드마크 세트까지의 선형 맵이다. 이러한 방식으로 표준 얼굴 모델을 중심으로 모델링된 가상 3D 자산에 얼굴 포즈 변환 매트릭스를 적용하여 가상 캐릭터가 따라하게 할 수 있다. 구성요소로는 geometry pipeline, effect renderer가 있다.

Geometry pipeline(지오메트리 파이프라인)은 메트릭 3D 공간 내에서 얼굴 변환 개체를 추정하는 주요 구성 요소이다. 각 프레임에서 다음 단계가 지정된 순서대로 실행된다.

1. 얼굴 랜드마크 화면 좌표가 메트릭 3D 공간 좌표로 변환.
2. 얼굴 포즈 변환 매트릭스는 둘 사이의 차이를 최소화하는 방식으로 표준 얼굴 메트릭 랜드마크 세트에서 런타임 얼굴 메트릭 랜드마크 세트로 엄격한 선형 맵핑으로 추정.
3. 얼굴 mesh는 런타임 얼굴 메트릭 랜드마크를 정점 위치(XYZ)로 사용하여 생성되며, 정점 텍스처 좌표(UV)와 삼각 위상은 모두 표준 얼굴 모델에서 상속.

Effect renderer(효과 렌더러)는 얼굴 효과 렌더러의 작업 예로 사용되는 구성 요소인데, OpenGL ES 2.0 API를 대상으로 모바일 장치에서 실시간 성능을 제공하며 다음과 같은 렌더링 모드를 지원한다.

1. 3D 객체 렌더링 모드 : 가상 객체를 감지된 얼굴 위에 그리며 얼굴에 부착된 객체(안경 같은)를 그린다.
2. 얼굴 그물망 렌더링 모드 : facemesh 표면 위에 텍스처를 확장하여 페이스 페인팅 기법을 모방한다.

2.2.1 BlazeFace

모바일 GPU 추론에 맞춘 가볍고 성능이 좋은 얼굴 검출기이다. 플래그십 장치에서 200-1000FPS 이상의 속도로 실행되고, 이 초실시간 성능을 통해 2D/3D 얼굴 키포인트 또는 기하학적 추정, 얼굴 특징 또는 표정 분류, 얼굴 영역 분할과 같은 작업별 모델에 대한 입력으로 정확한 얼굴 영역이 필요한 증강 현실 파이프라인에 적용할 수 있다.

Model	Average Precision	Inference Time, ms (iPhone XS)
MobileNetV2-SSD	97.95%	2.1
Ours	98.61%	0.6

표 ? BlazeFace 모델을 사용했을 때와 MobileNetV2를 사용했을 때의 정확성, 추론 시간 비교
BlazeFace에서는 컨볼루션 연산에 5x5 크기의 가중치 행렬을 사용하며, 이는 더 많은 채널을

사용하는 것보다 비교적 더 저렴하고 특정 크기의 수용 영역 크기에 도달하기 위해 필요한 레이어의 개수를 줄일 수 있다. 또한, BlazeFace는 Bottleneck layer의 중간 채널 수 경량화를 위해 입력 채널 수를 감소하고 출력 채널을 늘려서 사용한다.

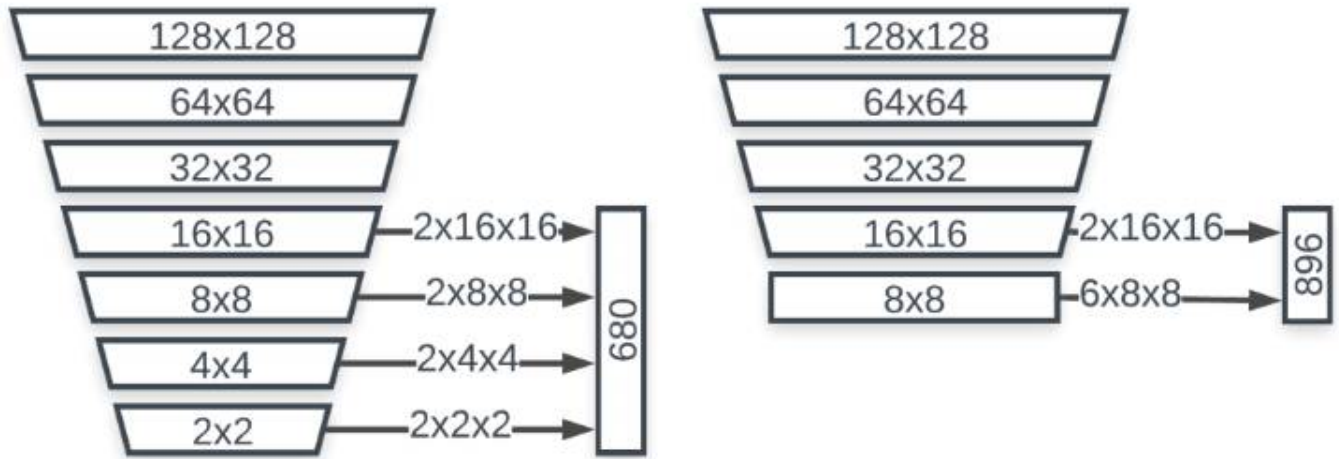


그림 ? Anchor computation: SSD(left) vs BlazeFace(right)

위 그림에서처럼 BlazeFace에서는 8x8 크기 이하로 그리드 크기를 줄이지 않았고, 2x2, 4x4, 8x8 크기의 피쳐맵에서 각각 2개의 anchors를 8x8 크기 피쳐맵에서의 6개의 anchors로 대체하였다. 결과적으로 16x16 크기의 피쳐맵에서 각 픽셀마다 2개의 anchors, 8x8 크기의 피쳐맵에서 6개의 anchors를 사용하여 예측하게 되며 총 896개의 바운딩 박스를 사용해 객체를 검출한다.

[연구논문] BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs 참조

2.3 역삼각함수(아크탄젠트)

직각 삼각형에서 직각을 포함하는 두 변의 길이를 R_x , R_y 라고 할 때, arc tangent를 사용하여 각도 θ 를 구하는 함수이다. 이때, θ 의 단위는 rad(라디안)이다.

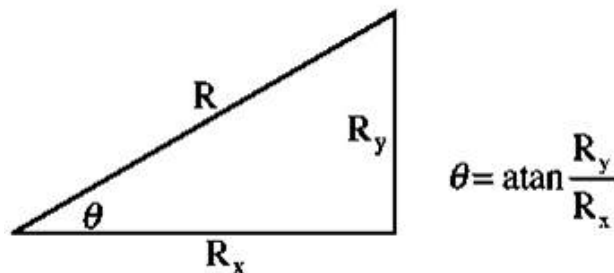


그림 19 아크탄젠트 공식

3. 음성 인식

음성인식이란 사람이 말하는 음성언어를 컴퓨터가 해석해 그 내용을 문자 데이터로 전환하는

신호처리를 말하며 ASR(auto speech recognition), STT(speech to text)라고도 부른다.

음성 인식 기술의 원리는 기본적으로 음성에 대한 파형을 분석하는 것에서부터 시작한다.

0.001초 단위로 나누고 그 시점에 있는 약 0.02초로 짧은 길이의 음성파형인 음편(unit)을 가져와 신호 처리를 거쳐 10개 이상의 숫자들을 출력한다. 이렇게 출력된 결과(특징 벡터)들이 바로 그 시점에서의 성대와 성도(소리길)의 상태를 의미한다.

발성시점의 성대 진동 횟수와 입 모양을 나타내는 숫자들은 바로 해당 소리가 되기에 알맞은 패턴을 찾아내야한다. 패턴을 찾는 과정은 모든 후보 단어에 대한 가능성을 열어두고, 인식기가 음성을 듣다가 정답이 아닌 것 같은 후보들을 탈락시켜, 신호 처리가 완료되었을 때 경쟁에서 살아남은 최종 단어가 최종 인식 결과가 된다.

이때, 음성의 신호처리는 시간 축에서 처리하려면 복잡하기에 주파수 축으로 축 변환하고 서로 다른 주파수 성분을 갖는 사인파들로 나누어준 후 신호처리를 진행해준다.

최근 음성인식 기술에 딥러닝 기술이 적용되면서 해당 기술의 성능이 비약적으로 향상되었다.

4. 음성 합성

음성 합성은 말소리의 음파를 기계가 자동으로 만들어내는 기술 즉, 인위적으로 사람의 소리를 합성해 텍스트를 음성으로 변환하는 기술로 TTS(text to speech)라고도 한다. 음성합성 기술은 크게 4가지로 분류할 수 있다. 조음 합성, 포먼트 합성, 연결 합성, 통계기반 파라미터 합성이 그것이다.

이 기술들 중 현재 가장 보편적으로 사용되고 있는 기술은 연결 합성(Concatenative synthesis)과 통계기반 파라미터 합성(Statistical parametric speech synthesis) 기술이다.

연결 합성은 USS(Unit Selection Synthesis)라고도 부르며 단어 또는 문장 단위로 녹음된 음성 데이터를 어떤 기준에 의해 음소 단위로 나누어 음편DB로 만들고, 역으로 음성을 합성할 때, 이 DB에서 전체 발화에 적합한 음편을 찾아 이어 붙이는 기술이다. DB에서 내가 만들고자 하는 음성을 위한 최적의 음편을 선택하는 기술과, 선택한 음편을 자연스럽게 이어 붙이는 능력의 기술이 필요하다.

통계기반 파라미터 합성은 음성 신호처리 기술에 기반하고 있다. 음성은 조음기관을 거치며 어떤 특성을 갖게 되는데, 이 특성을 신호처리 기술을 활용해 음성 데이터로부터 추출해 모델링하는 방식이다. 이때 데이터로부터 추출된 음성 특징들을 흔히 파라미터라고 부른다. 특징 파라미터들을 추출해 통계 모델링하는 훈련과정과, 텍스트가 입력되면 통계 모델로부터 해당되는 파라미터를 생성하고 음성 신호 처리 과정을 통해 적절한 음성으로 재구성되는 합성 과정으로 구성된다.

최근 10여년 사이에는 음성인식 분야와 마찬가지로 음성합성 기술 분야에도 딥러닝 기술이 활용되어 비약적인 성능 향상이 이뤄졌는데, 딥러닝 기반의 음성합성 기술은 위의 두 기술이 가지는 장점을 모두 가지며 두 기술이 가지는 단점을 모두 극복하였다. 뿐만 아니라 학습을 기반으로 다양한 사람의 발화 스타일을 직접 학습하기에 길지 않은 녹음 데이터만으로 여러 감정이나 스타일 등의 표현이 가능한 음성 합성기를 만들어 낼 수 있다는 점에서 보다 높은 가치를 지닌다고 말할 수 있다.

Ⅲ. 개발 방법

처음 이 작품을 구상할 때, 휴대용 기기로 설계가 되었고 이에 따라 하드웨어와 소프트웨어의 두 가지 방향으로 작품 설계가 진행되었다.

1. 소프트웨어

소프트웨어를 이용한 작품 구성은 작품에서 원하는 기능을 제공하는 오픈소스를 찾아내어 인용한 뒤 작품에 맞게 변형, 수정을 가할 수 있도록 계획하였다. 소프트웨어의 경우, 설계를 통해 크게 두 흐름으로 나누어 진행하는 것이 결정되었다.



그림 20 간단히 설계한 인물 촬영(학습) 및 인물 검색(특징 출력) 알고리즘

해당 설계과 하드웨어 개발에 대한 계획을 통해 프로그래밍 언어는 파이썬으로 진행하고 동시에 기본적인 영상처리 기능이 필요하다고 판단되어 OpenCV 패키지를 사용하여 계획을 진행해준다

소프트웨어 부분을 완성한 후, 라즈베리파이에 해당 코드를 이식하여 작품을 구현을 완성하는 것을 계획하였다

2. 하드웨어

하드웨어의 경우, 간단한 영상처리를 계획했었기에 라즈베리파이를 이용하여 설계 및 작품 구성을 진행한다. 버튼, 카메라, 소리 입력장치와 출력장치를 구성하여 필요한 입출력 장치들을 연결하여 완성된 회로 및 보드를 휴대 및 보호할 수 있도록 외부를 구성하여 마무리한다.

이에 따라 라즈베리파이4 보드, 전용 보이스 키트, 전용 카메라 모듈, 마이크와 이어폰, 스위치, 리본케이블, 점퍼케이블, SD카드를 준비하고 설계를 진행한다.

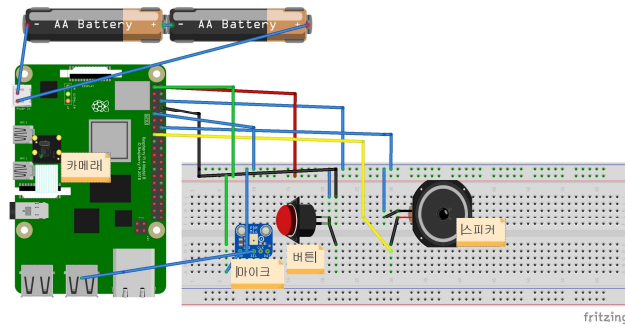


그림 21 간단히 설계한 하드웨어 부분 회로도

회로도는 다음과 같이 구상하였으며 해당 회로도에서 스피커 대신 이어폰을 대체하여 여러 환경에 대체할 수 있도록 한다. 작품의 외적인 부분은 가능하면 3D모델링 및 프린팅을 통해 구성하는 것을 계획하였다.

IV. 개발 내용

1. 소프트웨어

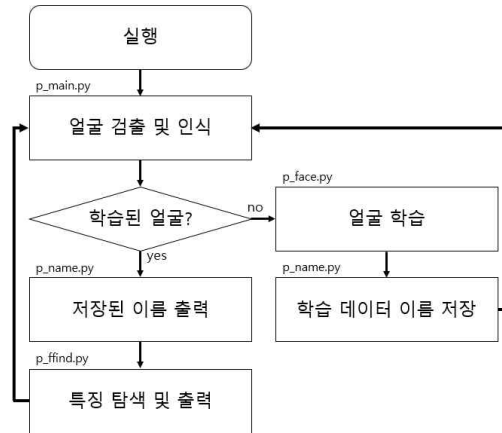


그림 22 설계한 대략적인 순서도

1.1 얼굴 검출 및 인식(p_main.py)

OpenCV의 CascadeClassifier함수를 이용하여 얼굴 정면을 검출하고 LBPHFaceRecognizer 함수를 이용해 학습시켜놓은 yml 얼굴 인식 파일의 데이터와 현재 검출된 얼굴이 동일 인물의 얼굴인지 비교하여 해당 인물의 id와 그 인물일 확률을 저장해놓는다.

버튼을 눌렀을 때, 검출된 얼굴이 학습된 얼굴들 중 하나와 일치할 경우, 해당 id에 해당하는 이름을 출력(1.3)하고 곧바로 얼굴 특징을 출력한다. 출력할 때, gTTS를 이용하여 음성신호로 출력한다.

만약 검출된 얼굴이 데이터와 일치하지 않을 경우, 얼굴 학습을 실행시켜 새로운 얼굴 데이터를 학습하여 저장하고 그 인물에 해당하는 이름을 음성신호로 입력받아 저장한다. 저장한 후에는 학습 데이터들을 읽어서 적용시킨 후 다시 얼굴의 검출과 인식을 반복한다.

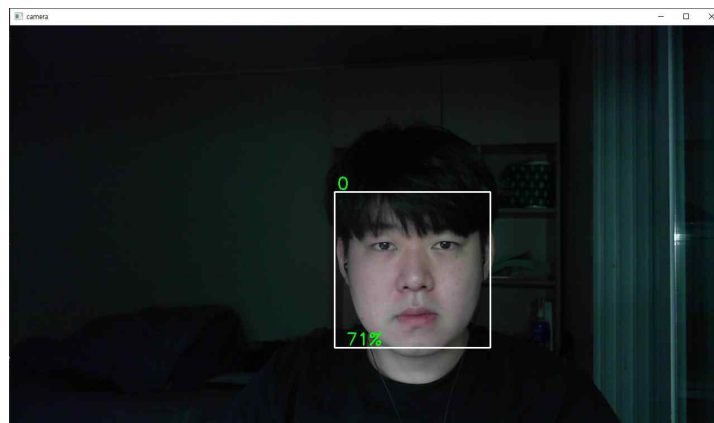


그림 23 얼굴 검출 및 인식 실행 시

1.2 얼굴 학습 모듈(p_face.py)

검출된 얼굴과 일치하는 데이터가 없을 때, 버튼을 누르면 실행된다. 먼저 검출된 얼굴의 이미지를 ROI(얼굴) 부분만을 잘라 짧은 시간에 100장을 새로운 id를 이름으로 저장한다. 저장한 이미지들을 LBPHFaceRecognizer함수를 이용해 LBP matrix의 패턴을 yml파일 형태로 저장해 학습한다. 이후 저장한 이미지들을 제거하여 저장 공간을 확보한다.



그림 24 학습 전 상황

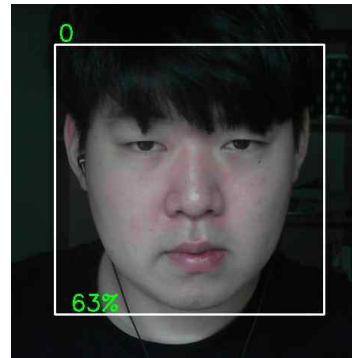


그림 25 학습 후 상황

처음 코드를 작성해주었을 때는 하나의 yml파일을 만들었기에 학습된 얼굴의 개수가 증가하면 새롭게 yml파일을 만드는 시간(학습 시간)이 늘어나고 저장공간이 부족한 문제를 야기하였다. 이를 해결하기 위해 서로 다른 얼굴을 학습하여 다른 yml파일 분할하여 저장하는 방법으로 코드를 작성하였고 위의 문제를 해결하였다.

1.3 이름 저장과 출력(p_name.py)

이름을 음성 인식(SpeechRecognition)패키지와 음성 합성(gTTS)패키지를 이용하여 음성신호 형태로 저장하고 출력하는 함수를 가지는 모듈을 구성하였다.

setName(id) 함수는 저장을 원하는 인물에 해당하는 이름을 음성 입력 장치에 입력하면 음성인식을 통해 텍스트로 바꾼 후 다시 해당 텍스트를 음성 합성하여 id에 해당하는 음성신호 파일로 저장한다.

getName(id) 함수는 출력을 원하는 id를 통해 해당 id의 이름이 저장된 음성파일을 출력 장치로 출력한다.

음성 합성을 진행할 때, 출력되는 음성 파일이 mp3파일로 구성되었기 때문에 해당 파일의 재생을 위해 mp3파일을 열 수 있는 playsound패키지를 이용해주었다.

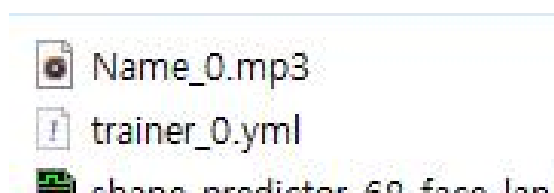


그림 26 생성된 yml 파일과 이름 mp3파일

1.4 얼굴 특징 출력(p_ffind.py)

확장성을 위해서 얼굴의 특징을 찾아내는 여러 함수를 이 모듈을 통해 실행하도록 구성해주었다. 해당 모듈에서 얼굴의 특징을 찾는 함수를 실행시키고 결과들을 모으고 정렬하여 gTTS를 이용하여 음성신호로 출력한다.

1.5 머리 색 검출(p_ff_hair.py)

검출된 얼굴의 범위를 이용해서 머리카락이 있을 것 같은 범위(ROI)를 대략적으로 정한 후 HSV로 색 공간을 변환한 후, 어떤 범위의 색이 가장 많은 지(cv2.inRange()) 측정한 후, 해당 색을 머리카락 색으로 지정하여 출력한다.

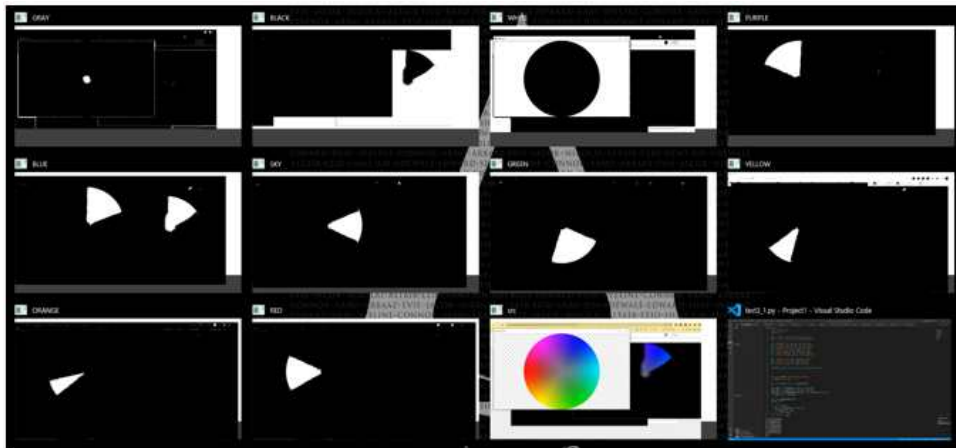


그림 27 지정된 영역에서 특정 색 범위 추출

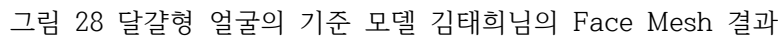
1.6 얼굴형 검출(p_ff_face.py)

사람들의 얼굴형은 크게 5가지로 구분할 수 있다. 사람마다 얼굴 골격이 다르고 살집도 다르기 때문에 다양한 얼굴형이 검출이 되는데 그 중에서도 가장 보편적으로 사람들이 가지고 있는 검출형 5가지를 조사하였다.

얼굴형 종류 중에서 각진 얼굴을 검출하기 위해 아크탄젠트 공식을 사용하였다. 파이썬에서 사용하는 아크탄젠트 함수는 atan(), atan2() 인데, atan() 함수는 두 점 사이의 탄젠트 값을 받아 리턴값이 $-\pi/2 \sim \pi/2$ 의 라디안 값을 범위를 가지고, atan2() 함수는 두 점 사이의 상대좌표(x,y)를 받아 $-\pi \sim \pi$ 의 범위를 가져서 사람 얼굴을 검출할 때 어느 위치, 각도에서든 각도를 정확하게 계산할 수 있게 atan2() 함수를 사용하였다.

파이썬에서 각도에 대한 함수를 쓰면 그 값이 라디안값으로 계산되기 때문에 우리가 흔히 사용하는 0~360도(dgree)로 나타내주기 위해 radian을 degree로 변환해 주는 함수를 사용하였다. radian을 도(degree)로 변환하는 법은 $\text{rad} * 180 / \pi$ 인데, 여기서 180을 추가로 더해줘야 우리가 원하는 양수의 각도를 구할 수 있게 된다. atan2() 함수가 상대좌표를 기준으로 계산하기 때문에 텍선의 위치상 데카르트 좌표에서 각도가 음수가 나오기 때문이다. 이러한 공식들과 Face Mesh 를 이용한 486개의 랜드마크로 기준이 되는 얼굴 부위의 좌표를

얼굴형 판단을 위해 각 얼굴형의 대표되는 인물들을 선정해 얼굴 부위 랜드마크를 기준으로
 잰 길이끼리 서로 비교하여 평균 낸 값을 다른 얼굴형들과 또 비교하여 평균 낸 값을 판별의
 기준이 되는 비교값으로 설정함. ex) 달걀형 -> 김태희, 카리나, 한가인. 긴얼굴형 -> 김우빈,
 노홍철, 장휘재. 각진형 -> 안젤리나 졸리, 노홍철, 김우빈. 둥근형 -> 조정석, 한가인, 카리나.
 역삼각형 -> 한예슬, 장휘재, 노홍철.



눈, 코, 입 크기 판별은 Dlib 를 사용하여 68개의 랜드마크를 통해 각 부위에 점을 찍어 두 점 사이의 거리를 구하는 공식을 사용하여 길이와 높이를 각각 계산하였다.

눈, 코, 입의 크기 비교를 위해 사용한 기준 값들은 얼굴형 검출에서 사용했던 기준들과 마찬가지로 각 얼굴형들의 대표적인 인물들 눈, 코, 입 비율을 구한 뒤, 성형외과 의사들이 통상적으로 황금비율을 가진 얼굴의 대표적 인물이라고 말하는 ‘김태희’를 기준으로 삼아 얼굴 비율의 최댓값-기준값, 기준값-최솟값의 평균을 내서 그 평균보다 작으면 ‘작음’, 크면 ‘큼’, 최댓값-기준값의 평균과 기준값-최솟값의 평균 사이면 ‘보통’으로 분류했다.

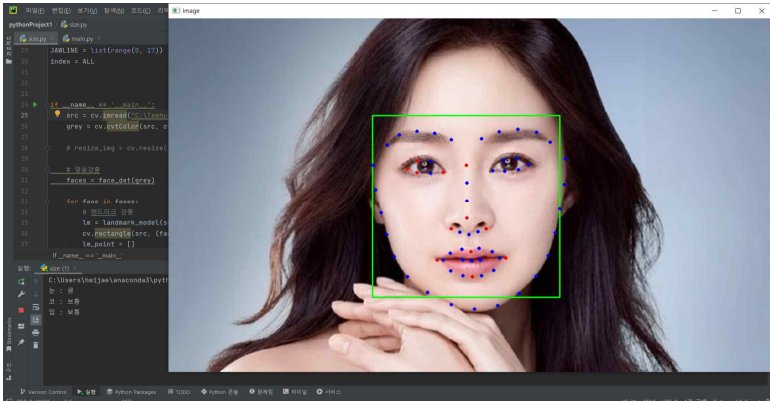


그림 29 얼굴 면적을 구할 때, 적용한 초록 사각형과 랜드마크

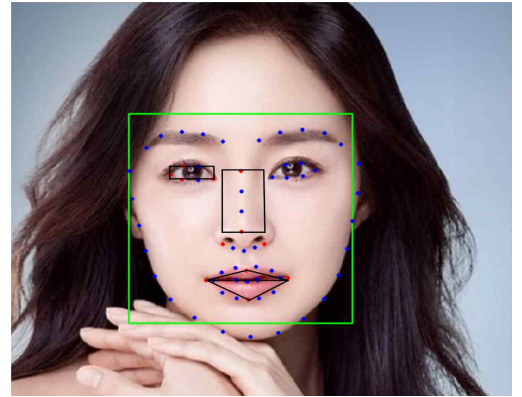


그림 30 면적 계산에 적용된 사각형 예시

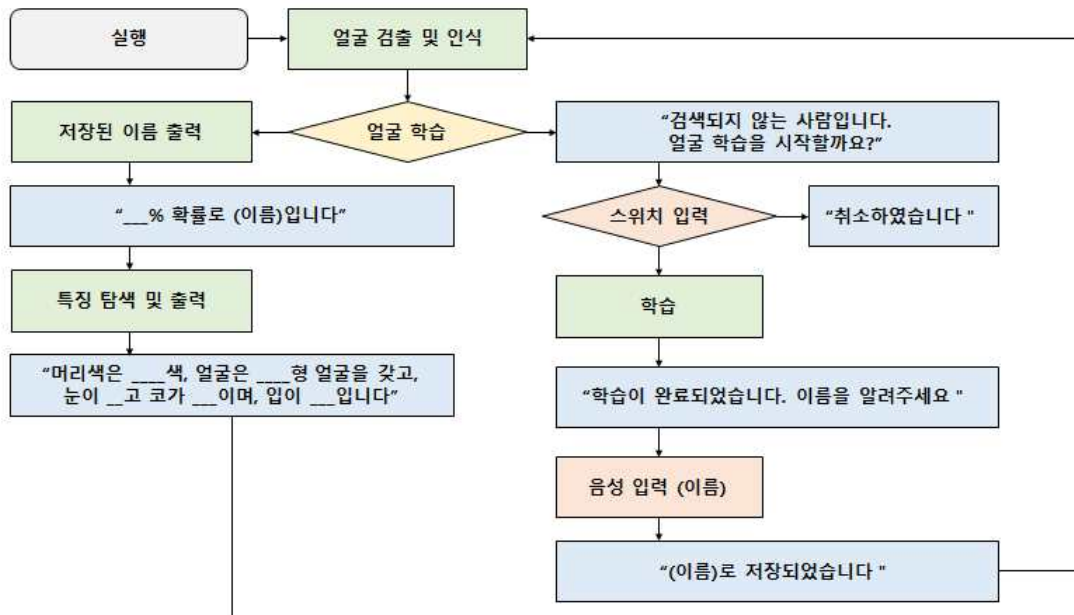


그림 31 개발 결과 순서도 (출력되는 TTS 음성 포함 표현)

2. 하드웨어

개발 방법에서 본래에 제작한 소프트웨어를 실행시킬 보드로 라즈베리파이의 사용을 계획하였으나, 라즈베리파이 운영체제에서 python, OpenCV등 필요한 환경과 패키지를 구축하는 데에 있어 계속되는 오류와 실패로 결국 하드웨어적인 결과물은 만들어내지 못한 채로 작품의 개발을 완료하게 되었고, 결과적으로 처음 목표한 핵심적인 기능을 담당하는 프로그램의 소프트웨어만 완성하게 되었다.

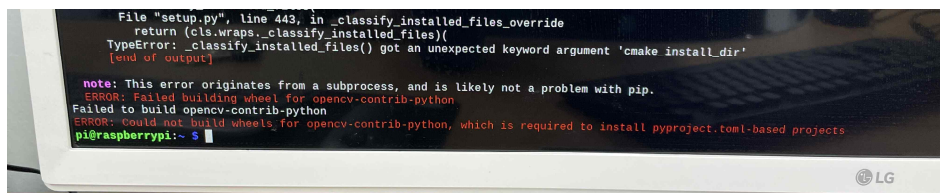


그림 32 여러 수단을 사용해도 발생하는 오류와 문제점

V. 결론

1. 평가 및 교훈

먼저 처음에 목표로 했던 것은 라즈베리파이와 파이썬, OpenCV를 이용해 시각장애인분들을 위한 얼굴 특징 및 인물 판별 디바이스 였는데, 라즈베리파이에선 원하는 환경을 구성하기 위해 여러 패키지들과 도구들을 설치하는 과정에서 예상치 못한 오류들이 많이 발생하였다. 이 때문에 결국 라즈베리파이에선 개발을 진행하지 못하고 디바이스를 만들려던 목표를 코드 개발과 소스코드 구성하는 것으로 변경하고 말았다. 이에 대해서는 아직 어떤 문제에 대한 대처, 해결방식이 많이 부족하다는 것을 깨달았다. 라즈베리파이에 쏟은 시간을 생각해보면, 라즈베리파이에 패키지나 툴을 설치하는 시간과 오류를 해결하는 시간이 코딩을 하는 시간보다 더 오래 걸린 것 같다.

소프트웨어의 관점에서도 아쉬운 점이 많이 존재하였다. 얼굴의 검출과 인식의 경우에는 높은 정확도를 얻을 수 있었지만, 얼굴의 특징을 도출해내는 과정에서의 결과는 정확도가 많이 낮음을 확인할 수 있었다. 먼저 머리카락의 색 판별 과정의 경우, 원래에는 평균 이동이나 watershed 등의 segmentation 기법을 사용하려고 하였지만 머리카락의 형태가 사람마다 다르거나 이미지가 찍히는 배경에 따라 발생하는 오차가 너무 많아 그저 ROI를 설정하여 구하는 방식에 그치고 말았다. 또한 얼굴형이나 얼굴 구성요소의 크기를 검출하는 부분에서 또한 오차범위가 허용되는 오차보다 너무 넓다는 것을 확인할 수 있었다.

비슷한 기능의 패키지를 중구난방하게 사용한 점 또한 확인할 수 있던 문제점 중 하나였다. 이러한 과정을 통해 얻은 교훈은 어떠한 일을 진행함에 있어 생길 수 있는 오류들을 미리 예측하여 그에 맞는 해결책과 계획을 미리 수립함으로 그 일이 더 원활하게 진행될 수도 있다는 점과 어떤 문제를 해결함에 있어 목표로 가는 방향이나 방식은 내가 생각한 것보다 많기에 고집을 부리기보단 다른 방법도 고려해보긴 해야 한다는 것을 다시한번 깨달을 수 있었다. 이를 계기로 발생할 수 있는 여러 문제에 대해 미리 대비하는 개발자가 될 수 있기를 바란다.

2. 향후 개선 방안

위 평가에서 확인한 문제점들을 개선하기 위해서는 얼굴 특징에 관한 부분은 더 정확한 판별 기준이 필요하다는 점과 머리카락과 같은 부분에 대해서는 더 세밀한 ROI설정이 필요하다는 것을 확인했다. 이러한 문제의 해결을 위해 비교대상과 학습데이터를 더 증가시키고 이용하여 더 자세하게 비교를 진행하여 그에 대한 결과 계산에 대해서도 더 정확한 방법을 모색하는 과정을 가져야한다. 또한 라즈베리파이에 대한 문제 해결을 다시 진행함으로, 휴대가 가능한 디바이스를 만든다는 목적에 집중하여 현 개발 결과의 아쉬운 점을 개선해 갈 예정이다.

부록

프로그램 소스코드

p_main.py

```
import cv2
import numpy as np
from playsound import playsound

#--import user module
import p_init
import p_face as pf
import p_ffind as pff
import p_name as pn

#--variables setting
path_trainer = './trainer'
path_Cascade = './trainer/haarcascade_frontalface_default.xml'

confidence_print = 0
firstId = 0
nextId = 0
loop = []
id_array=[]
confidence_array=[]
id_print = "none"

#--cv2 setting
detector = cv2.CascadeClassifier(path_Cascade)

font = cv2.FONT_HERSHEY_SIMPLEX

cap = cv2.VideoCapture(0)
cap.set(3,1280)
cap.set(4,720)

recognizer = []
Id = 0
flag = False
```



```

#--Read trainer file(recognizer setting)
while True:
    recognizer.append(cv2.face.LBPHFaceRecognizer_create())
    id_array.append(0)
    confidence_array.append(0)
    try:
        recognizer[nextId].read(path_trainer + '/trainer_'+str(nextId)+'.yaml')
        nextId+=1
    except:
        loop = range(firstId,nextId)
        break

print("ID checked      " + str(Id))
print("ID for next learn is " + str(nextId))
print("loop:          " + str(loop))
#print("id_array:      " + str(id_array))
#print("confidence_array:  " + str(confidence_array))
print("recognizer address: " + str(recognizer))

#--main code(Loop)
while __name__ == '__main__':
    ret, src = cap.read()
    img = src.copy()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = detector.detectMultiScale(
        gray,
        scaleFactor = 1.2,
        minNeighbors = 5,
        minSize = (128, 128)
    )

    for (x,y,w,h) in faces:
        cv2.rectangle(img, (x,y), (x+w,y+h), (255,255,255), 2)

        # predict??
        # 이미 만들어진 yaml 파일을 불러와서 만든 recognizer를 이용해서 받아온 이미지와
        # 일치하는 값이 있는지 알아보고 id, confidence(작을수록 얼굴 일치)반환
        confidence = 100
        id_index = -1

```

```

for i in loop :
    id_array[i], confidence_array[i] = recognizer[i].predict(gray[y:y+h,x:x+w])
    if confidence > confidence_array[i]:
        confidence = confidence_array[i]
        id_index = id_array[i]

    if (confidence < 45):          # 일치하는 얼굴 존재 시,
        confidence_print = " {0}%".format(round(100 - confidence))
        id_print = str(id_index)
    else:                         # 일치하는 얼굴 없을 시,
        confidence_print = " {0}%".format(round(100 - confidence))
        id_print = "unknown"
    buf = img.copy()
    flag = True

    # 일치 확률과 이름을 화면에 출력
    cv2.putText(img, id_print, (x+5,y-5), font, 1, (0,255,0), 2)
    cv2.putText(img, str(confidence_print), (x+5,y+h-5), font, 1, (0,255,0), 2)

#try:
#    cv2.imshow('buf', buf)
#except:
#    NULL
if flag:
    if cv2.waitKey(10)==32:
        if id_print == "unknown":
            playsound('tts_unknown.mp3')
        if cv2.waitKey() == 32:
            playsound('tts_startL.mp3')
            currentId, nextId, new_recognizer = pf.startLearning(nextId, cap)
            recognizer.append(new_recognizer)
            recognizer[currentId].read(path_trainer + '/trainer_'+str(currentId)+'.yml')
            loop = range(firstId, nextId)
            id_array.append(0)
            confidence_array.append(0)
            playsound('tts_complete.mp3')
            pn.setName(currentId)
        else:
            playsound('tts_cancel.mp3')

```

```

        else:
            #print(confidence_print)
            #print(id_print)
            pn.getName(id_print, confidence_print)
            pff.findFeature(buf, x, y, w, h, id_print)
            flag = False
    else:
        if cv2.waitKey(10) == 32:
            playsound('tts_miss.mp3')

    cv2.imshow('camera', img)
    # 최대한 자주 Key를 획득할 수 있도록 wait time을 줄임
    if cv2.waitKey(10) == 27: # [esc] 누르면 메인 루프 종료 및 프로그램 종료
        break

print("Terminate Program.")
cap.release()
cv2.destroyAllWindows()

```

p_face.py

```

import cv2
import numpy as np
from PIL import Image
import os
from glob import glob

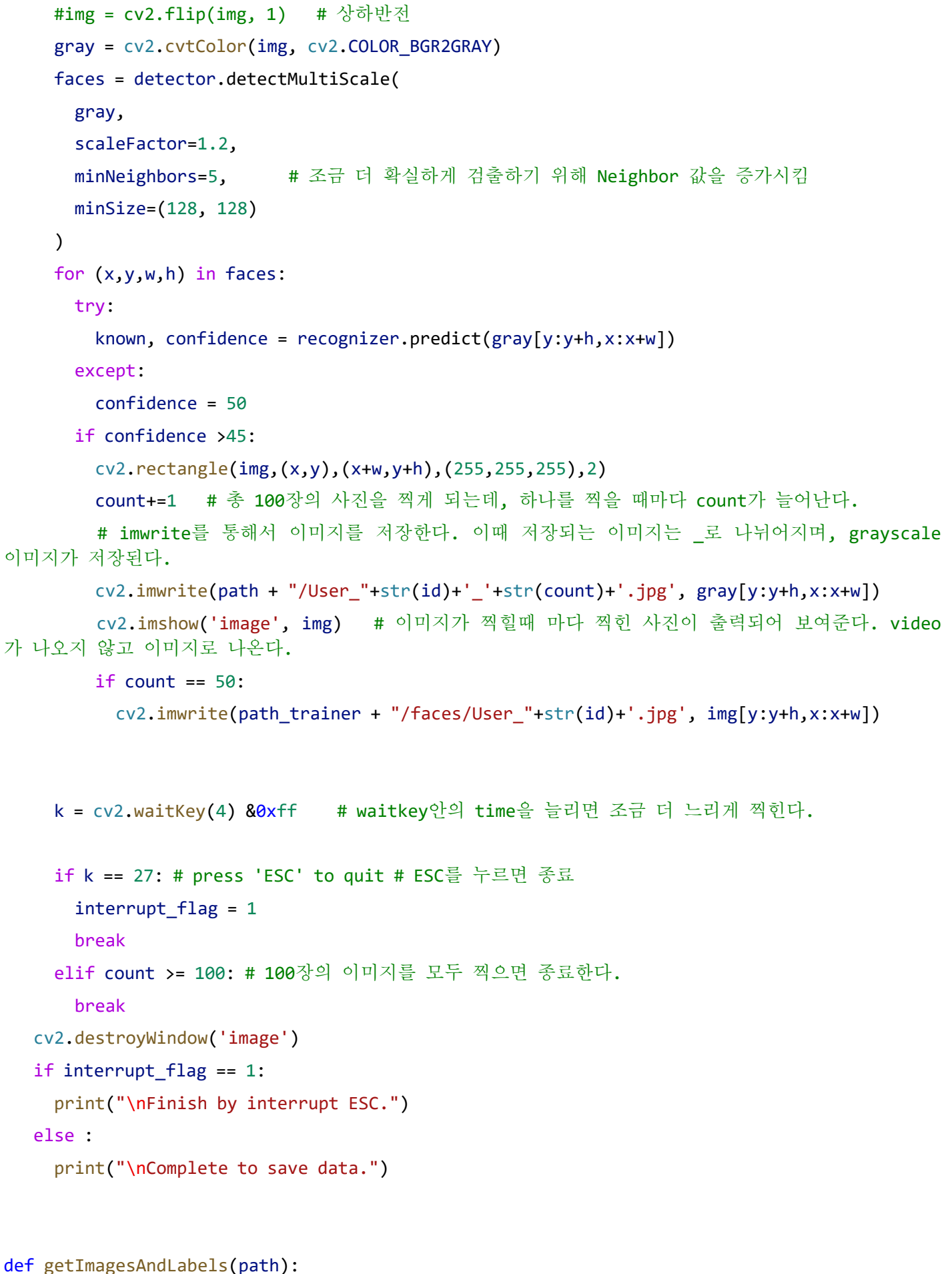
path_trainer = './trainer'
path = path_trainer + '/PhotoLabel'
path_Cascade = './trainer/haarcascade_frontalface_default.xml'

detector = cv2.CascadeClassifier(path_Cascade)
recognizer = cv2.face.LBPHFaceRecognizer_create()

#현재 저장될 아이디를 받고 100장의 사진 찍어서 저장
def extractIm(id, cap):
    count = 0
    interrupt_flag = 0
    while True:
        ret, img = cap.read()

```

```


#img = cv2.flip(img, 1) # 상하반전
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
faces = detector.detectMultiScale(
    gray,
    scaleFactor=1.2,
    minNeighbors=5, # 조금 더 확실하게 검출하기 위해 Neighbor 값을 증가시킴
    minSize=(128, 128)
)
for (x,y,w,h) in faces:
    try:
        known, confidence = recognizer.predict(gray[y:y+h,x:x+w])
    except:
        confidence = 50
    if confidence >45:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,255,255),2)
        count+=1 # 총 100장의 사진을 찍게 되는데, 하나를 찍을 때마다 count가 늘어난다.
        # imwrite를 통해서 이미지를 저장한다. 이때 저장되는 이미지는 _로 나뉘어지며, grayscale
이미지가 저장된다.
        cv2.imwrite(path + "/User_"+str(id)+'_'+str(count)+'.jpg', gray[y:y+h,x:x+w])
        cv2.imshow('image', img) # 이미지가 찍힐때 마다 찍힌 사진이 출력되어 보여준다. video
가 나오지 않고 이미지로 나온다.
        if count == 50:
            cv2.imwrite(path_trainer + "/faces/User_"+str(id)+'.jpg', img[y:y+h,x:x+w])

k = cv2.waitKey(4) &0xff # waitkey안의 time을 늘리면 조금 더 느리게 찍힌다.

if k == 27: # press 'ESC' to quit # ESC를 누르면 종료
    interrupt_flag = 1
    break
elif count >= 100: # 100장의 이미지를 모두 찍으면 종료한다.
    break
cv2.destroyAllWindows()
if interrupt_flag == 1:
    print("\nFinish by interrupt ESC.")
else :
    print("\nComplete to save data.")

def getImagesAndLabels(path):

```

```

imagePaths = [os.path.join(path,file) for file in os.listdir(path)]    # 이미지 파일들을
안에 넣음
faceSamples=[]                # 각 이미지의 얼굴 값을 array uint8 형태로 저장한것을 dictionary 형
태로 저장
ids = []                      # 여러개의 id값을 배열로 저장
for imagePath in imagePaths:    # 이미지 파일을 하나씩 받아 옴
    PIL_img = Image.open(imagePath).convert('L') # image를 grayscale로 변환 시킨다고 함 (굳
이...? 이미 되어있는데)
    img_numpy = np.array(PIL_img,'uint8')      # np.array로 img 파일을 int형으로 변환시켜 저
    id = int(os.path.split(imagePath)[-1].split("_")[1]) # 파일의 id를 추출
    faces = detector.detectMultiScale(img_numpy)      # 다시 얼굴 이미지에서 또 얼굴을 추출
(얼굴의 크기를 알기 위함)
    for (x,y,w,h) in faces:
        faceSamples.append(img_numpy[y:y+h,x:x+w]) # img를 int형으로 바꾼 sample들을 넣은 배열
        ids.append(id)                # id값을 쪽 넣어서 배열로 만듦
return faceSamples, ids

```

#이미지가 저장된 파일을 받고 얼굴 훈련, yml파일 생성, 얼굴의 갯수 반환

```

def trainIm(id):
    print("\nPlease wait for a second...")
    faces,ids = getImagesAndLabels(path)
    recognizer.train(faces, np.array(ids))                # LBP matrix를 만듦.
    try:
        os.chmod(path_trainer,777)
    except:
        NULL
    recognizer.write(path_trainer + '/trainer_'+str(id)+'.yaml') # 만든 LBP matrix를 yaml 파일
형태로 저장
    nextid = id + 1
    print("\n {0} faces trained\n".format(nextid))      # ids 배열의 개수만큼 훈련되었다고 표시함.
    return nextid ,recognizer

```

```

def startLearning(id, cap):
    print("\ntraining started")
    extractIm(id, cap)
    nextid, recognizer = trainIm(id)
    print("nextid is "+str(nextid))
    [os.remove(f) for f in glob(path + '/*.jpg')]
    print("\ntraining ended")

```

```
return id, nextid, recognizer
```

p_name.py

```
import speech_recognition as sr
import cv2
from gtts import gTTS
from playsound import playsound
import os

path_trainer = './trainer'

r = sr.Recognizer()

def setName(id):
    filename = path_trainer + '/Name_'+str(id)+'.mp3'
    while True:
        with sr.Microphone() as source:
            #print("Say Something")
            speech = r.listen(source)

        try:
            audio = r.recognize_google(speech, language="ko-KR")
            #print("Your speech thinks like\n " + audio)
            tts = gTTS(
                text=audio,
                lang='ko', slow=False
            )
            tts.save(filename)
            playsound(filename)
            playsound('tts_save.mp3')
            if cv2.waitKey() == 32:
                playsound(filename)
                playsound('tts_savecomp.mp3')
                break
            else:
                os.remove(filename)
                pass

        except sr.UnknownValueError:
```

```

        print("Your speech can not understand")
        pass

    except sr.RequestError as e:
        print("Request Error!; {0}".format(e))
        pass
    playsound('tts_retry.mp3')

def getName(id, conf):
    tts = gTTS(
        text = str(conf)+' 확률로',
        lang='ko', slow=False
    )
    tts.save('tts_conf.mp3')

    playsound('tts_conf.mp3')
    try:
        playsound(path_trainer + '/Name_'+str(id)+'.mp3')
    except:
        pass
    playsound('tts_check.mp3')
    os.remove('tts_conf.mp3')

```

p_ffind.py

```

from gtts import gTTS
from playsound import playsound
import os

#--import user module
import p_ff_hair as ffh
import p_ff_face as fff
import p_ff_size as ffs

def findFeature(src,x,y,w,h,id):
    haircolor = ffh.hairColor(src,x,y,w,h)
    faceshape = fff.shapeFace(src)
    facesize = ffs.faceSize(src)

    featuretext = '머리색은 ' + str(haircolor) + ', 얼굴은 ' + str(faceshape) + '을 갖고, ' +
    str(facesize)

```

```

tts = gTTS(
    text= featuretext,
    lang = 'ko', slow = False
)
tts.save('Feature.mp3')
playsound('Feature.mp3')
os.remove('Feature.mp3')

```

p_ff_hair.py

```

import cv2
import numpy as np

```

```

def setRoi(src, x, y, w, h):

```

```

    #source Image
    #cv2.imshow('Original img', src)

```

```

    #set ROI Image
    x0 = x
    y0 = int(y-h/3 if y-h/3 >0 else 0)
    x1 = int(x+w)
    y1 = y

```

```

    roi =src[y0:y1, x0:x1]
    #cv2.imshow('roi', roi)
    return roi

```

머리색을 찾는 함수 (input: src: 머리 부분 컬러 이미지, output: str형 색깔 정보)

```

def findColor(src):
    RED_L = ("붉은색", (0, 30, 30), (8, 255, 255) )
    RED_H = ("붉은색", (168, 30, 30), (180, 255, 255))

    ORG = ("주황색", (8, 30, 30), (20, 255, 255))
    YEL = ("노란색", (20, 30, 30), (35, 255, 255))
    GRN = ("초록색", (35, 30, 30), (80, 255, 255))
    SKY = ("하늘색", (80, 30, 30), (95, 255, 255))
    BLU = ("파란색", (95, 30, 30), (130, 255, 255))
    PUR = ("보라색", (130, 30, 30), (150, 255, 255))
    PNK = ("분홍색", (150, 30, 30), (167, 255, 255) )

```



```

WHT = ("하얀색", (0, 0, 215), (180, 30, 255) )
GRY = ("회색", (0, 0, 40), (180, 30, 215))
BLK = ("검은색", (0, 0, 0), (180, 255, 40))

colorlist = (ORG, YEL, GRN, SKY, BLU, PUR, PNK, WHT, GRY, BLK)

hsv = cv2.cvtColor(src, cv2.COLOR_BGR2HSV)

lower_RED = cv2.inRange(hsv, RED_L[1], RED_L[2])
upper_RED = cv2.inRange(hsv, RED_H[1], RED_H[2])
added_RED = cv2.addWeighted(lower_RED, 1.0, upper_RED, 1.0, 0.0)

stk = np.average(added_RED)
colorD = RED_L

for i in colorlist:
    mask = cv2.inRange(hsv, i[1], i[2])
    p = np.average(mask)
    #print({p})

    if p > stk:
        stk = p
        colorD = i

return colorD[0]

def hairColor(src, x, y, w, h):
    img = setRoi(src,x,y,w,h)
    Color = findColor(img)
    return Color

```

p_ff_face.py

```

import mediapipe as mp
import math

```

```

# 아크탄젠트(탄젠트의 역함수)를 통해 각도 구하기
def cal_rad(x1, x2, y1, y2):
    rad = math.atan2(y2-y1,x2-x1)
    return rad

def radTodeg(rad):
    PI = math.pi
    deg = 180 + (rad*180)/PI
    # print(deg)
    # if deg >= 40:
    #     print("얼굴은 V라인입니다.")
    return deg

# Face Mesh
mp_face_mesh = mp.solutions.face_mesh
face_mesh = mp_face_mesh.FaceMesh()

def shapeFace(image):
    height, width, _ = image.shape

    # Facial landmarks
    result = face_mesh.process(image)

    #cv2.imshow('image', image)      # 잘라낸 부분 크기 확대해서 나옴 -> 코드 수정했음(신경x 해도 됨)

    for facial_landmarks in result.multi_face_landmarks:
        for i in range(0, 468):
            pt1 = facial_landmarks.landmark[i]
            x = int(pt1.x * width)
            y = int(pt1.y * height)

            #cv2.circle(image, (x, y), 2, (255, 255, 255), -1)

            if i == 54:
                x11 = x
                y11 = y
                #cv2.circle(image, (x11, y11), radius=2, color=(255, 255, 0), thickness=2)

            if i == 284: # 오른쪽 이마 끝

```

```

x12 = x
y12 = y
#cv2.circle(image, (x12, y12), radius=2, color=(255, 255, 0), thickness=2)

if i == 172: # 좌측 하악
    x9 = x
    y9 = y
    #print("왼쪽턱 좌표 = ", [x9, y9])
    #cv2.circle(image, (x9, y9), radius=2, color=(255, 255, 0), thickness=2)

if i == 397: # 우측 하악
    x10 = x
    y10 = y
    #cv2.circle(image, (x10, y10), radius=2, color=(255, 255, 0), thickness=2)

if i == 9: # 이마 가운데
    x8 = x
    y8 = y
    #cv2.circle(image, (x8, y8), radius=2, color=(0, 0, 255), thickness=2)

if i == 365: # 우측 턱
    x6 = x
    y6 = y
    #cv2.circle(image, (x6, y6), radius=2, color=(0, 0, 0), thickness=2)

if i == 67: # 좌측 이마
    x7 = x
    y7 = y
    #cv2.circle(image, (x7, y7), radius=2, color=(0, 255, 0), thickness=2)

if i == 152: # 맨 아래 턱
    x3 = x
    y3 = y
    #print("턱끝 좌표 = ", [x3, y3])
    #cv2.circle(image, (x3, y3), radius=2, color=(0, 255, 0), thickness=2)

if i == 4: # 코 끝 좌표
    x1 = x
    y1 = y
    #print("코 끝 좌표 = " , [x1, y1]) # 코 끝 좌표 출력

```

에 빨간점
#cv2.circle(image, (x1, y1), radius=2, color=(0, 0, 255), thickness=2) # 코 끝 좌표

```
if i == 10: # or i == 152:
    x2 = x
    y2 = y
    #print("이마 좌표 = ", [x2, y2])
    #cv2.circle(image, (x2, y2), radius=2, color=(0, 255, 0), thickness=2)
```

```
if i == 227: # 좌측 관자놀이 랜드마크 번호
    x4 = x
    y4 = y
    #print("좌측관자놀이 좌표 = ", [x4, y4])
    #cv2.circle(image, (x4, y4), radius=2, color=(255, 0, 0), thickness=2)
```

```
if i == 454: # 우측 관자놀이 랜드마크 번호
    x5 = x
    y5 = y
    #print("우측관자놀이 좌표 = ", [x5, y5])
    #cv2.circle(image, (x5, y5), radius=2, color=(255, 0, 0), thickness=2)
```

```
r1 = math.sqrt((x2 - x3) ** 2 + (y2 - y3) ** 2) # 얼굴의 높이 계산
#print("이마 ~ 턱까지의 길이(얼굴의 높이) = ", r1)
```

```
r2 = math.sqrt((x4 - x5) ** 2 + (y4 - y5) ** 2) # 얼굴의 넓이 계산
#print("좌측관자 ~ 우측관자까지의 길이(얼굴의 넓이) = ", r2)
```

```
r3 = math.sqrt((x6 - x7) ** 2 + (y6 - y7) ** 2) # 얼굴의 대각선 계산
#print("왼쪽이마 ~ 오른쪽턱 길이(얼굴의 대각선) = ", r3)
```

```
r4 = math.sqrt((x2 - x8) ** 2 + (y2 - y8) ** 2) # 이마의 높이 계산
#print("미간 ~ 이마중앙 길이(이마의 높이) = ", r4)
```

```
r5 = math.sqrt((x11 - x12) ** 2 + (y11 - y12) ** 2) # 이마의 길이 계산
#print("왼쪽이마 ~ 오른쪽이마 길이(이마의 길이) = ", r5)
```

```
r6 = math.sqrt((x3 - x9) ** 2 + (y3 - y9) ** 2) # 턱선의 길이 계산
#print("좌측턱 ~ 아래턱 길이(턱선의 길이) = ", r6)
# print(r6*2)
```

```

r7 = math.sqrt((x9 - x10) ** 2 + (y9 - y10) ** 2)    # 턱~턱 길이 계산
#print("좌측턱 ~ 우측턱 길이(턱~턱의 길이) = ", r7)

rad = cal_rad(x3, x9, y3, y9)
radTodeg(rad)
degree = radTodeg(rad)
if degree >= 40:
    return '역삼각형 얼굴'
if r1 > r2 and r5 > (r6*2.1):    # 얼굴의 높이 >얼굴의 넓이, 이마의 넓이 >턱선의 길이 * 2.1
    return '달걀형 얼굴'
if r1 > r2*1.1865:    # 얼굴의 높이 >얼굴의 넓이 * 1.1865
    return '긴 얼굴'
if (r2-r5)+r7 > r5 or (r2-r5)+r7 > r2:    # (얼굴너비-이마너비)+턱너비 >이마너비 or (얼굴너비-
이마너비)+턱너비 >얼굴너비
    return '각진 얼굴'
if r2 > r5:    # 얼굴의 넓이 >이마의 넓이
    return '둥근 얼굴'

```

p_ff_size.py

```

import cv2
import dlib
import numpy as np
import math

path_shape_prediction = './trainer/shape_predictor_68_face_landmarks.dat'

face_det = dlib.get_frontal_face_detector()
landmark_model = dlib.shape_predictor(path_shape_prediction)

# range는 끝값이 포함안됨
ALL = list(range(0, 68))
RIGHT_EYEBROW = list(range(17, 22))
LEFT_EYEBROW = list(range(22, 27))
RIGHT_EYE = list(range(36, 42))
LEFT_EYE = list(range(42, 48))
NOSE = list(range(27, 36))
MOUTH_OUTLINE = list(range(48, 61))
MOUTH_INNER = list(range(61, 68))

```

```
JAWLINE = list(range(0, 17))
```

```
index = ALL
```

```
def faceSize(src):
```

```
    gray = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
```

```
    # resize_img = cv2.resize(src, (552, 588))
```

```
    # 얼굴검출
```

```
    faces = face_det(gray)
```

```
    for face in faces:
```

```
        # 랜드마크 검출
```

```
        lm = landmark_model(src, face)
```

```
        #cv2.rectangle(src, (face.left(), face.top()), (face.right(), face.bottom()), color=(0, 255, 0), thickness=2)
```

```
        lm_point = []
```

```
        #print(face)  # 출력시 좌표는 (왼쪽위모서리 x,y값) (오른쪽아래모서리 x,y값)
```

```
        r = face.right()-face.left()
```

```
        #print(face.left(),face.right())
```

```
        #print(r)
```

```
        for p in lm.parts():
```

```
            lm_point.append([p.x, p.y])
```

```
        lm_point = np.array(lm_point)
```

```
        i = 0
```

```
        for p in lm_point:
```

```
            i+=1
```

```
            cv2.circle(src, (p[0], p[1]), radius=2, color=(255,0,0), thickness=2)
```

```
            if i == 37: # 눈 길이(왼)
```

```
                x1 = p[0]
```

```
                y1 = p[1]
```

```
                #print(p)
```

```
                #cv2.circle(src, (p[0], p[1]), radius=2, color=(0, 0, 255), thickness=2)
```

```
            if i == 40: # 눈 길이(오)
```

```
                x2 = p[0]
```

```
                y2 = p[1]
```

```
                #print(p)
```

```
                #cv2.circle(src, (p[0], p[1]), radius=2, color=(0, 0, 255), thickness=2)
```

```
            if i == 38: # 눈 높이(위)
```

```
                x3 = p[0]
```

```

y3 = p[1]
#print(p)
#cv2.circle(src, (p[0], p[1]), radius=2, color=(0, 0, 255), thickness=2)
if i == 42: # 눈 높이(아래)
    x4 = p[0]
    y4 = p[1]
    #print(p)
    #cv2.circle(src, (p[0], p[1]), radius=2, color=(0, 0, 255), thickness=2)
if i == 49: # 입 길이(왼)
    x5 = p[0]
    y5 = p[1]
    #print(p)
    #cv2.circle(src, (p[0], p[1]), radius=2, color=(0, 0, 255), thickness=2)
if i == 55: # 입 길이(오)
    x6 = p[0]
    y6 = p[1]
    #print(p)
    #cv2.circle(src, (p[0], p[1]), radius=2, color=(0, 0, 255), thickness=2)
if i == 52: # 입 높이(위)
    x7 = p[0]
    y7 = p[1]
    #print(p)
    #cv2.circle(src, (p[0], p[1]), radius=2, color=(0, 0, 255), thickness=2)
if i == 58: # 입 높이(아래)
    x8 = p[0]
    y8 = p[1]
    #print(p)
    #cv2.circle(src, (p[0], p[1]), radius=2, color=(0, 0, 255), thickness=2)
if i == 31:      # 코 끝 좌표(코 길이(아래))
    x9 = p[0]
    y9 = p[1]
    #print(p)
    #cv2.circle(src, (p[0], p[1]), radius=2, color=(0, 0, 255), thickness=2)
if i == 28:     # 코 길이(위)
    x10 = p[0]
    y10 = p[1]
    #cv2.circle(src, (p[0], p[1]), radius=2, color=(0, 0, 255), thickness=2)
if i == 32:      # 코 너비(왼)
    x11 = p[0]
    y11 = p[1]

```

```

    #print(p)
    #cv2.circle(src, (p[0], p[1]), radius=2, color=(0, 0, 255), thickness=2)
if i == 36:    # 코 너비(오)
    x12 = p[0]
    y12 = p[1]
    #cv2.circle(src, (p[0], p[1]), radius=2, color=(0, 0, 255), thickness=2)

r1 = math.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2) # 눈 길이
r2 = math.sqrt((x3 - x4) ** 2 + (y3 - y4) ** 2) # 눈 높이
r3 = math.sqrt((x5 - x6) ** 2 + (y5 - y6) ** 2) # 입 길이
r4 = math.sqrt((x7 - x8) ** 2 + (y7 - y8) ** 2) # 입 높이
r5 = math.sqrt((x9 - x10) ** 2 + (y9 - y10) ** 2) # 코 길이
r6 = math.sqrt((x11 - x12) ** 2 + (y11 - y12) ** 2) # 코 너비

#### 얼굴전체면적(네모박스면적) : 눈코입 면적 = 100 : ??? #### 비례식 이용해서 비율 계산 ####
#### ??? = 눈코입이 얼굴에서 차지하는 퍼센트(비율) ####
x = (r1*r2*100)/(r**2)    # 직사각형의 넓이공식 : 가로x세로    : 눈 비율
y = (r5*r6*100)/(r**2)    # 직사각형의 넓이공식 : 가로x세로    : 코 비율
z = ((r3*r4/2)*100)/(r**2) # 마름모의 넓이 공식 : (가로x세로)/2    : 입 비율
# print(x,y,z)

# 크기 비교의 기준이 되는 값은 기준인물들의 눈코입 비율을 구한 뒤 비교하여 넣은 값임 #
text_result = ''

if x <= 0.81:
    text_result += '눈이 작고 '
elif x >0.81 and x <0.99:
    text_result += '눈이 보통이고 '
else:
    text_result += '눈이 크고 '

if y <= 5.24:
    text_result += '코가 작으며, '
elif y >5.24 and y <6.01:
    text_result += '코가 보통이며, '
else:
    text_result += '코가 크며, '

if z <= 2.2:
    text_result += '입이 작습니다'

```



```

elif z >2.2 and z <2.7:
    text_result += '입 크기가 보통입니다'
else:
    text_result += '입이 큼니다'

return text_result

```

p_init.py

```

from gtts import gTTS

tts = gTTS(
    text='인식되는 얼굴이 존재하지 않습니다. 다시 시도해주세요',
    lang='ko', slow=False
)
tts.save('tts_miss.mp3')

tts = gTTS(
    text='검색되지 않는 사람입니다. 얼굴 학습을 시작할까요?',
    lang='ko', slow=False
)
tts.save('tts_unknown.mp3')

tts = gTTS(
    text='학습을 시작합니다',
    lang='ko', slow=False
)
tts.save('tts_startL.mp3')

tts = gTTS(
    text='취소하였습니다',
    lang='ko', slow=False
)
tts.save('tts_cancel.mp3')

tts = gTTS(
    text='학습이 완료되었습니다. 이름을 알려주세요.',
    lang='ko', slow=False
)
tts.save('tts_complete.mp3')

```

```

tts = gTTS(
    text='다시 말씀해주세요',
    lang='ko', slow=False
)
tts.save('tts_retry.mp3')

tts = gTTS(
    text = '로 저장할까요?',
    lang='ko', slow=False
)
tts.save('tts_save.mp3')

tts = gTTS(
    text = '로 저장되었습니다',
    lang='ko', slow=False
)
tts.save('tts_savecomp.mp3')

tts = gTTS(
    text = '입니다',
    lang='ko', slow=False
)
tts.save('tts_check.mp3')

```

출처

[사이트] Open Source Computer Vision - <https://docs.opencv.org/>

[사이트] MediaPipe - <https://google.github.io/mediapipe/>

[연구논문] 한국인 성인 남녀의 머리 및 얼굴 부위 측정치 통합분석 - 전은경,
문지현(울산대학교 생활과학부 의류학 전공)

[연구논문] 한국인 입에 대한 생체계측학적 연구 - 김순흠 외 8명(건국대학교 의학전문대학원
성형외과학교실, 해부학교실)

[연구논문] 중앙면에 대한 생체계측학적 연구; 노화에 따른 코의 계측치 변화 - 김순흠 외
8명(건국대학교 의학전문대학원 성형외과학교실, 산부인과학교실, 해부학교실)

[사이트] 한국인 인체지수조사 - <https://sizekorea.kr/>

[사이트] Facial point annotations - <https://ibug.doc.ic.ac.uk/resources/facial-point-annotations/>

[사이트] 300 Faces In-the-Wild Challenge(300-W), ICCV 2013 -
<https://ibug.doc.ic.ac.uk/resources/300-W/>

- [연구논문] One Millisecond Face Alignment with an Ensemble of Regression Trees - Vahid Kazemi and Josephine Sullivan KTH, Royal Institute of Technology Computer Vision and Active Perception Lab Teknikringen 14, Stockholm, Sweden
- [연구논문] Analysis and Improvement of Facial Landmark Detection - Philipp Kopp Semester Project Report February 2019 Dr. Derek Bradley, Dr. Thabo Beeler, Prof. Dr. Markus Gross
- [연구논문] BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs - Valentin Bazarevsky, Yury Kartynnik, Andrey Vakunov, Karthik Raveendran, Matthias Grundmann, Google Research, 1600 Amphitheatre Pkwy, Mountain View, CA 94043, USA
- [연구논문] Real-time Facial Surface Geometry from Monocular Video on Mobile GPUs - Yury Kartynnik, Artsiom Ablavatski, Ivan Grishchenko, Matthias Grundmann, Google Research, 1600 Amphitheatre Pkwy, Mountain View, CA 94043, USA