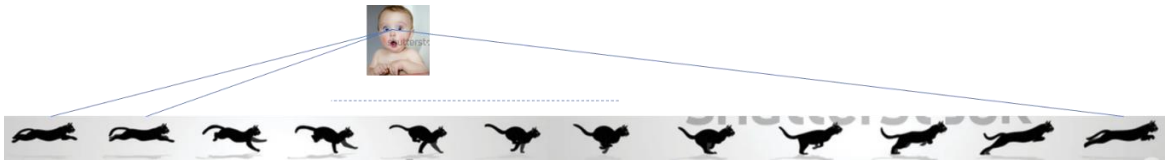
10 동적 비전

사람의 시각은 동적이다. [그림 10-1]에서 갓난아이 눈에 입력되는 고양이 영상은 매우 동적이다. 아이는 연속 영상을 통해 물체가 취하는 무수히 많은 자세를 매 순간 학습한다. 사람의 비전이 심하게 변하는 환경에 강인한 근본 이유다.



[그림 10-1] 사람의 동적 비전

전세계에 설치된 감시 카메라는 10억대에 육박하고 있다. 2019년 기준으로 분당 500시간 분량의 비디오가 유튜브에 업로드된다고 알려져 있다. 컴퓨터의 메모리 용량과 CPU와 GPU 성능이 획기적으로 개선된 탓에 이제 비디오는 일상이 되었다.

이들 비디오 데이터를 자동으로 인식할 수 있다면 무궁무진한 응용을 창출할 수 있다. 9장에서 다룬 분류, 검출, 분할 알고리즘은 비디오 처리의 시작점이다. 하지만 이들 알고리즘은 정지 영상을 독립적으로 처리하기 때문에 한계를 안고 있다. 예를 들어 의심스런 사람을 발견하여 알리는 감시 시스템을 제작한다면, 같은 사람을 연결하여 시간에 따른 몸 동작을 분석하고 행위를 인식할 수 있어야 한다. 추적 도중 다른 물체에 가려 잠시 사라졌다 다시 나타나도 같은 사람으로 확인할 수 있어야 한다.

비디오 인식을 위한 딥러닝 기법을 서베이한 최근 논문으로 [Sharma2021]을 추천한다. 이 논문은 2018년 이후에 발표된 13개의 또다른 서베이 논문을 소개한다. 비디오 인식의 응용 분야, 풀어야 하는 문제, 딥러닝 기법, 17종의 데이터셋, 향후 연구 주제를 체계적으로 설명한다. [Sreenu2019]는 감시 목적의 비디오 처리 연구를 서베이한다.

이 장은 물체의 모션 정보를 분석하는 알고리즘과 물체를 실시간으로 추적하는 알고리즘을 설명한다. 뒤부분에서는 비디오에 나타난 사람을 인식하는 여러 방법을 설명한다. MediaPipe 라이브러리를 가지고 얼굴과 손을 검출하고 사람의 자세를 추정하는 프로그래밍 실습을 수행한다.

 © 전북대학교 오일석

10.3 비디오에서 사람 인식과 MediaPipe를 이용한 프로그래밍

9.6절은 사람 얼굴을 보고 누구인지 인식하고 나이와 성별을 알아내는 여러 알고리즘을 소개했다. [그림 9-40]이 보여주는 바와 같이 얼굴을 인식하려면 먼저 얼굴을 검출하고 정렬하는 전처리 단계를 적용해야 한다. 이 절에서는 얼굴 검출과 얼굴 정렬을 다룬다. 또한 손의 관절을 검출하는 문제를 다룬다.

얼굴 검출과 정렬을 구현한 소스코드가 아주 많은데, 여기서는 MediaPipe이 제공하는 솔루션을 활용한다. MediaPipe는 구글이 개발하여 제공하는 기계학습 개발 프레임워크로서, 여러 가지 유용한 비디오 처리 솔루션을 제공한다 [Lugaresi2019].[1] 이 프레임워크는 비디오의 처리 절차를 그래프로 표현하고 프레임이 시간 순서에 따라 흐르도록 제어한다. 솔루션은 주로 C++ 언어로 개발했고 모바일 장치에서도 실시간 실행할 수 있도록 알고리즘을 설계하고 구현했다.

사용자는 MediaPipe가 제공하는 여러 가지 솔루션을 이용하여 쉽게 응용 프로그램을 개발할 수 있다. 교차 플랫폼$_{\text{cross-platform}}$ 기능을 제공하기 때문에 응용 프로그램을 서버, 데스크톱, 모바일 기기(안드로이드와 iOS) 등의 여러 플랫폼에서 실행할 수 있다. 솔루션을 사용하는 인터페이스 언어로는 파이썬과 자바 스크립트를 주로 사용한다. 현재 MediaPipe 프레임워크는 16개의 솔루션을 제공하는데, 그중 파이썬 인터페이스가 가능한 것은 얼굴 검출$_{\text{face detection}}$, 얼굴 그물망$_{\text{face mesh}}$, 손$_{\text{hands}}$, 셀피 분할$_{\text{selfie segmentation}}$, 자세$_{\text{pose}}$, 몸 전체$_{\text{holistic}}$, 오브젝트론$_{\text{objectron}}$의 7개다.

10.3.1 얼굴 검출

MediaPipe가 제공하는 BlazeFace라는 얼굴 검출 방법을 간략히 설명하고 프로그래밍 실습을 제시한다. MediaPipe는 그래프를 이용하여 비디오의 입력과 출력을 일관성 있게 표현하고 제어한다. 서로 다른 과업인 얼굴 검출, 얼굴 그물망 검출, 손 검출, 자세 추정 등이 일관성 있는 틀 위에서 개발되었기 때문에 MediaPipe로 개발한 프로그램은 간결하고 통일된 패턴을 가진다. BlazeFace의 원리를 간략히 살펴본 다음, 즐거운 마음으로 MediaPipe 프로그래밍을 시작한다.

원리

[1] https://mediapipe.dev/

[그림 9-19]는 물체 검출을 위한 컨볼루션 신경망의 발전 과정을 요약한다. SSD는 한 단계 방법에 해당하는데, [그림 9-22]의 faster RCNN을 개조한 모델이다 [Liu2016(a)]. 실험 결과 faster RCNN의 7 FPS(초당 프레임 처리 수, frames per second)와 YOLO의 45 FPS보다 빠른 59 FPS를 달성하였고 검출 성능도 가장 우수하다.

BlazeFace는 SSD를 얼굴 검출에 맞게 개조했다 [Bazarevsky2019]. 개조 과정에서 특히 처리 속도를 중요하게 고려하였는데 200~1000 FPS를 보장할 수 있게 되었다. SSD는 입력 영상을 점점 줄여 16*16, 8*8, 4*4, 2*2 맵까지 만든다. 맵의 화소에 컨볼루션 연산을 적용함으로써 한 화소에서 크기와 가로세로 비율이 다른 2~6개의 박스를 예측한다. 그런데 얼굴은 가로세로 비율이 1에 가깝기 때문에 BlazeFace는 가로세로 비율은 1로 고정하고 크기만 다른 박스를 여럿 예측한다. 또한 아주 작은 얼굴은 나타나지 않는다고 가정하여 8*8까지만 줄임으로써 속도 향상을 이룬다. SSD는 예측된 박스가 여럿 겹치는 문제를 비최대 억제로 해결하였는데, SSD는 겹친 박스 중에 신뢰도가 최대인 박스를 선택하는 반면 BlazeFace는 여러 박스의 정보를 가중 평균함으로써 정확률을 향상한다. 실험 결과 정확률이 10%만큼 향상되었고 얼굴 위치를 표시하는 박스가 불안정하게 흔들리는 현상이 줄었다고 보고한다.

BlazeFace는 신경망을 추가로 개조하여 얼굴을 표시하는 박스 뿐만 아니라 6개 랜드마크를 같이 출력한다. 랜드마크는 눈의 중심, 귀 구슬점$_{\text{ear tragion}}$, 입 중심, 코 끝이다. 랜드마크를 이용하여 얼굴 특정 부위에 장식을 다는 증강 현실을 구현할 수 있다. 얼굴 분석의 다음 단계인 얼굴 그물망 추정에서 랜드마크 정보를 활용한다.

프로그래밍 실습: BlazeFace로 얼굴 검출

[프로그램 10-4]는 BlazeFace를 이용하여 정지 영상에서 얼굴을 검출한다. 2행은 mediapipe 모듈을 불러온다. 아나콘다 프롬프트에서 pip install mediapipe 명령어로 설치해야 한다. 4행은 OpenCV의 imread 함수로 영상을 읽는다. 6행은 mediapipe 모듈의 solutions에서 얼굴 검출을 담당하는 face_detection 모듈을 읽어 mp_face_detection 객체에 저장한다. 7행은 solutions에서 검출 결과를 그리는데 쓰는 drawing_utils 모듈을 읽어 mp_drawing 객체에 저장한다.

9행은 mp_face_detection이 제공하는 FaceDetection 클래스로 얼굴 검출에 쓸 face_detection 객체를 생성한다. 이때 첫째 인수 model_selection은 0 또는 1을 설정할 수 있는데, 0은 카메라로부터 2미터 이내로 가깝게 있을 때 적합하고 1은 5미터 이내로 조금 멀리 있을 때 적합하다. 이 프로그램은 1을 설정하여 먼 곳으로 설정했다. 두번째 인수 min_detection_confidence는 0~1 사이의 실수를 설정하는데 검출 신뢰도가 설정한 값보다 큰 경우만 검출 성공으로 간주한다. 10행의 process 함수는 실제로 검출을 수행하고 결과를 res 객체에 저장한다. MediaPipe는 OpenCV와 달리 RGB 채널 순서를 사용하기 때문에 BGR로 표현된 img 영상을 RGB로 변환하여 process 함수

에 입력한다.

res 객체는 detections라는 리스트 변수를 가지는데 여기에 검출한 얼굴 정보가 들어있다. print(res.detections)로 확인해보면, 아래와 같이 검출 신뢰도를 나타내는 score와 박스를 나타내는 relative_bounding_box와 함께 6개의 relative_keypoints가 들어 있음을 알 수 있다. 이 프로그램을 실행하면 입력 영상에서 얼굴을 3개 검출했기 때문에 리스트에 아래와 같이 3개 요소가 들어있다.

```
In [1]: print(res.detections)
  [label_id: 0
   score: 0.8932861089706421
   location_data {
     format: RELATIVE_BOUNDING_BOX
     relative_bounding_box { xmin: 0.1996, ymin: 0.2552, width: 0.1337, height: 0.2004}
     relative_keypoints { x: 0.2414, y: 0.3165}
     relative_keypoints { x: 0.2967, y: 0.3340}
     … …
  label_id: 0
   score: 0.8440857529640198
     … …
  label_id: 0
   score: 0.813656747341156
   … …]
```

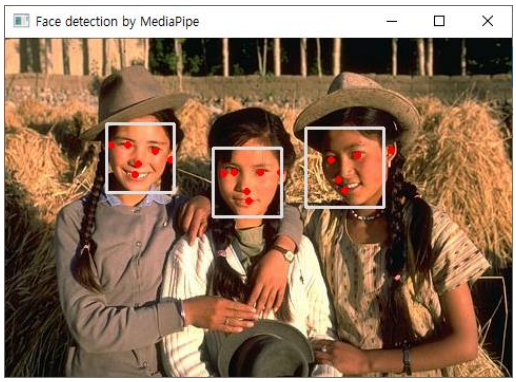
12~13행은 검출한 얼굴이 없으면 실패를 알린다. 14~17행은 얼굴이 있을 때를 처리하는데, 검출된 얼굴 각각을 원본 영상에 표시하고 결과 영상을 윈도우에 디스플레이한다.

[프로그램 10-4] 얼굴 검출 (MediaPipe 솔루션)

```
1   import cv2 as cv
2   import mediapipe as mp
3
4   img=cv.imread('BSDS_376001.jpg')
5
6   mp_face_detection=mp.solutions.face_detection
7   mp_drawing=mp.solutions.drawing_utils
8
9   face_detection=mp_face_detection.FaceDetection(model_selection=1,min_detection_confidence=0.5)
10  res=face_detection.process(cv.cvtColor(img,cv.COLOR_BGR2RGB))
11
```

 © 전북대학교 오일석

```
12  if not res.detections:
13      print('얼굴 검출에 실패했습니다. 다시 시도하세요.')
14  else:
15      for detection in res.detections:
16          mp_drawing.draw_detection(img,detection)
17      cv.imshow('Face detection by MediaPipe',img)
18
19  cv.waitKey()
20  cv.destroyAllWindows()
```



프로그램의 실행 결과를 보면, 세 소녀의 얼굴을 제대로 검출하였다. 얼굴의 6개 랜드마크를 상당히 정확하게 검출했음을 확인할 수 있다. 4행에서 다른 영상을 주고 여러 번 실험하여 성능을 가늠해보기 바란다.

프로그래밍 실습: 비디오에서 얼굴 검출

[프로그램 10-5]는 [프로그램 10-4]의 비디오 버전이다. 7행까지는 [프로그램 10-4]와 같다. 9행은 OpenCV를 이용하여 웹캠과 연결을 시도한다. 12행은 웹캠에서 프레임을 읽는다. 17행은 process 함수로 프레임에서 얼굴을 검출하여 res 객체에 저장한다. 19~21행은 검출된 얼굴을 프레임에 표시한다. 23행은 윈도우에 얼굴 영상을 디스플레이하는데, flip 함수로 좌우 반전하여 거울에 비친 모양을 보여준다.

프로그램 실행 결과를 보면, 비디오에서 얼굴을 제대로 검출했음을 확인할 수 있다. 끊김이 없이 실시간으로 비디오를 처리할 수 있음을 확인할 수 있다.

[프로그램 10-5] 비디오에서 얼굴 검출

 © 전북대학교 오일석

```
1   import cv2 as cv
2   import mediapipe as mp
3
4   mp_face_detection=mp.solutions.face_detection
5   mp_drawing=mp.solutions.drawing_utils
6
7   face_detection=mp_face_detection.FaceDetection(model_selection=1,min_detection_confidence=0.5)
8
9   cap=cv.VideoCapture(0,cv.CAP_DSHOW)
10
11  while True:
12      ret,frame=cap.read()
13      if not ret:
14          print('프레인 획득에 실패하여 루프를 나갑니다.')
15          break
16
17      res=face_detection.process(cv.cvtColor(frame,cv.COLOR_BGR2RGB))
18
19      if res.detections:
20          for detection in res.detections:
21              mp_drawing.draw_detection(frame,detection)
22
23      cv.imshow('MediaPipe Face Detection from video',cv.flip(frame,1))
24      if cv.waitKey(5)==ord('q'):
25          break
26
27  cap.release()
28  cv.destroyAllWindows()
```

 © 전북대학교 오일석

프로그래밍 실습: 얼굴을 장식하는 증강 현실

[프로그램 10-6]은 [프로그램 10-5]에 증강 현실 기능을 추가한 프로그램이다. 달라진 부분을 음영 처리하여 쉽게 구별할 수 있게 한다.

증강 현실augmented reality(AR)은 카메라로 입력된 실제 영상에 가상의 물체를 혼합함으로써 현실감을 증대시키는 기술이다. 여기서는 BlazeFace가 검출한 6개 얼굴 랜드마크에 간단한 장신구를 혼합하는 단순한 가상 현실을 구현한다. 실제 영상에 가상의 물체를 추가할 때 물체의 배경을 투명하게 처리하지 않으면 단지 조각 영상을 붙인 느낌이 들어 현실감이 떨어진다. 영상을 혼합할 때 배경을 투명하게 하려면 RGB 채널 외에 투명도를 나타내는 알파 채널을 추가로 가진 png 파일을 주로 사용한다.

4~6행은 장신구로 사용할 영상을 준비한다. 4행에서 imread 함수로 'dice.png' 파일을 읽는데, cv.IMREAD_UNCHANGED를 인수로 주어 알파 채널을 포함해 4개 채널을 모두 읽어오라고 지시한다.[2] 5행은 영상을 10%로 축소하고, 6행은 너비와 높이를 w와 h에 저장한다.

25~29행은 검출된 얼굴에 장신구를 달고 윈도우에 디스플레이한다. 25행은 get_key_point 함수를 이용하여 검출된 얼굴 정보를 담고 있는 det에서 오른쪽 눈 위치를 꺼내 p 객체에 저장한다. 6개 랜드마크 이름을 알려면, 아래와 같이 dir(mp_face_detection.FaceKeyPoint)을 실행하면 된다.

```
In [7]: dir(mp_face_detection.FaceKeyPoint)

['LEFT_EAR_TRAGION', 'LEFT_EYE', 'MOUTH_CENTER', 'NOSE_TIP', 'RIGHT_EAR_TRAGION', 'RIGHT_EYE', ...]
```

26~27행은 오른쪽 눈을 중심으로 장신구를 배치할 때 장신구 영상의 왼쪽 위와 오른쪽 아래 좌표를 계산한다. 28행은 장신구 영상이 원본 영상 안에 머무는지 확인한다. 안에 있다면 28~29행은 원본 영상의 해당 위치에 장신구 영상을 혼합한다. 장신구를 나타내는 png 영상은 0~2번 채널은 RGB를 나타내고 3번은 투명도를 나타내는 알파 채널이다. 따라서 알파 채널은 dice[:,:,3:]인데, 28행은 [0,255] 범위를 255로 나누어 [0,1]사이로 변환하여 alpha에 저장한다. 29행은 원본 영상 frame에 (1-alpha)를 곱하고 장신구 영상인 dice에 alpha를 곱하고 둘을 더함으로써 둘을 혼합한다.

[프로그램 10-6] 얼굴을 장식하는 증강 현실	
1	import cv2 as cv

[2] 이 png 파일은 위키피디아의 'portable network graphics' 페이지가 제공하는 주사위 영상이다.

```
import mediapipe as mp

dice=cv.imread('dice.png',cv.IMREAD_UNCHANGED) # 증강 현실에 쓸 장신구
dice=cv.resize(dice,dsize=(0,0),fx=0.1,fy=0.1)
w,h=dice.shape[1],dice.shape[0]

mp_face_detection=mp.solutions.face_detection
mp_drawing=mp.solutions.drawing_utils

face_detection=mp_face_detection.FaceDetection(model_selection=1,min_detection_confidence=0.5)

cap=cv.VideoCapture(0,cv.CAP_DSHOW)

while True:
    ret,frame=cap.read()
    if not ret:
        print('프레인 획득에 실패하여 루프를 나갑니다.')
        break

    res=face_detection.process(cv.cvtColor(frame,cv.COLOR_BGR2RGB))

    if res.detections:
        for det in res.detections:
            p=mp_face_detection.get_key_point(det,mp_face_detection.FaceKeyPoint.RIGHT_EYE)
            x1,x2=int(p.x*frame.shape[1]-w//2),int(p.x*frame.shape[1]+w//2)
            y1,y2=int(p.y*frame.shape[0]-h//2),int(p.y*frame.shape[0]+h//2)
            alpha=dice[:,:,3:]/255 # 투명도를 나타내는 알파값
            if x1>0 and y1>0 and x2<frame.shape[1] and y2<frame.shape[0]:
                frame[y1:y2,x1:x2]=frame[y1:y2,x1:x2]*(1-alpha)+dice[:,:,:3]*alpha

    cv.imshow('MediaPipe Face AR',cv.flip(frame,1))
    if cv.waitKey(5)==ord('q'):
        break

cap.release()
cv.destroyAllWindows()
```

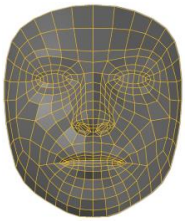
프로그램을 실행하면, 오른쪽 눈에 주사위 장신구가 증강되어 있다. 얼굴을 빠르게 움직여도 지연 없이 비디오가 자연스럽게 디스플레이됨을 확인할 수 있다. MediaPipe는 얼굴 영상을 빠르게 처리할 수 있게 설계되고 구현되어 있기 때문이다. 증강 현실 또는 다른 재미있는 응용 프로그램을 제작할 때 속도는 생각 이상으로 중요한다.

10.3.2 얼굴 그물망 검출

얼굴 영상을 인식하거나 얼굴에 정교하게 증강 현실을 적용하는 등의 응용은 [그림 9-40]이 보여주는 바와 같이 먼저 얼굴 검출$_{\text{face detection}}$과 얼굴 정렬$_{\text{face alignment}}$을 수행해야 한다. 얼굴 검출은 앞 절에서 설명했고, 여기서는 얼굴에서 랜드마크를 검출하는 얼굴 정렬을 다룬다. 얼굴 정렬을 위한 최근 기술을 살필 수 있는 논문으로 [Huang2021]을 추천한다. 여기서는 MediaPipe가 제공하는 FaceMesh 솔루션을 활용한다.

원리

FaceMesh는 [그림 10-12]가 보여주는 468개의 랜드마크를 검출한다 [Kartynnik2019]. 눈과 입, 코에 더욱 조밀하게 랜드마크를 배치하여 매끄러운 얼굴 그물망을 얻는데, 그림에서 보는 바와 같이 랜드마크는 3차원 좌표로 표현된다. 눈과 입을 보다 정확하게 추정하라고 지정하면 주목$_{\text{attention}}$ 알고리즘을 적용하여 보다 정확한 추정을 시도한다 [Grishchenko2020].



[그림 10-12] FaceMesh가 사용하는 468개의 얼굴 랜드마크 [Kartynnik2019]

FaceMesh는 첫 프레임에 BlazeFace를 적용하여 얼굴을 검출한다. 이후에는 이전 프레임에서 검출한 랜드마크와 모션 정보를 활용하여 얼굴을 추적함으로써 처리 시간을 획기적으로 줄인다. 매 프레임에서 예측한 곳에 실제 얼굴이 있는지 신뢰도를 계산하는데, 신뢰도가 임계값보다 낮으면 BlazeFace를 다시 적용한다. 얼굴 추적은 1유로 필터$_{\text{one Euro filter}}$를 활용하여 빠른 속도를 유지한다.

FaceMesh는 3차원 그물망을 추정하려고 3차원 랜드마크 데이터셋으로 학습하였다. 학습에는 모바일 기기로 자연 환경에서 찍은 3만 장의 얼굴 영상을 사용한다. 랜드마크가 3차원 점이므로 레이블링 작업이 아주 까다롭기 때문에 특별한 레이블링 도구를 만들어 사용했다. 이 도구는 얼굴 영상이 입력되면 3차원 표준 그물망 모델로 랜드마크의 초기 좌표를 계산하여 얼굴 영상에 그려준다. 레이블링하는 사람은 정제하는 과정을 통해 정확한 3차원 랜드마크 좌표를 확정한다.

프로그래밍 실습: 얼굴 그물망 검출

[프로그램 10-7]은 FaceMesh를 이용하여 비디오에 나타난 얼굴에서 그물망을 검출하는 프로그램이다. [프로그램 10-5]와 구조가 비슷하다. 단지 BlazeFace로 얼굴을 검출하는 함수가 FaceMesh로 그물망을 검출하는 함수도 대치된 정도이다.

4행은 mediapipe 모듈의 solutions에서 얼굴 그물망 검출을 담당하는 face_mesh 모듈을 읽어 mp_mesh 객체에 저장한다. 5행은 검출 결과를 그리는데 쓰는 drawing_utils 모듈을 읽어 mp_drawing에 저장하고, 6행은 그리는 유형을 지정하는데 쓰는 drawing_styles 모듈을 읽어 mp_styles에 저장한다. 8행은 mp_mesh가 제공하는 FaceMesh 클래스로 얼굴 그물망 검출에 쓸 mesh 객체를 생성한다. 인수를 살펴보면, max_num_faces=2는 얼굴을 2개까지 처리하라고 지시하고, refine_landmarks=True는 눈과 입에 있는 랜드마크를 더 정교하게 검출하라고 지시하고, min_detection_confidence=0.5는 얼굴 검출 신뢰도가 0.5 이상일 때 성공으로 간주하라고 지시한다. 마지막 인수 min_tracking_confidence=0.5는 랜드마크 추적 신뢰도가 0.5보다 작으면 실패로 간주하고 새로 얼굴 검출을 수행하라고 지시한다.

18행의 process 함수는 실제 그물망 검출을 수행하고 결과를 res에 저장한다. 20행은 검출된 얼굴

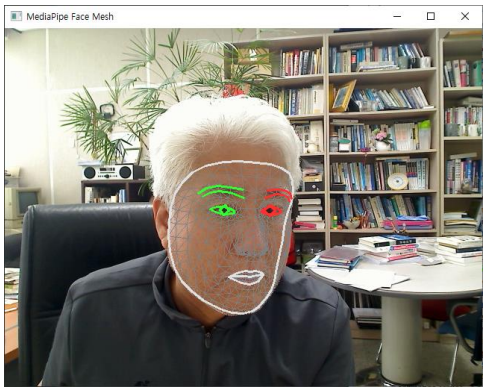
이 있는지 확인하고, 21행은 검출된 얼굴 각각에 대해 그물망을 그리는 일을 반복한다. 22행은 그물망을 그리고, 23행은 얼굴 경계와 눈과 눈썹을 그리고, 24행은 눈동자를 그린다.

[프로그램 10-7] 얼굴 그물망 검출

```
1   import cv2 as cv
2   import mediapipe as mp
3
4   mp_mesh=mp.solutions.face_mesh
5   mp_drawing=mp.solutions.drawing_utils
6   mp_styles=mp.solutions.drawing_styles
7
8   mesh=mp_mesh.FaceMesh(max_num_faces=2,refine_landmarks=True,min_detection_confidence=0.5,min_tracking_confidence=0.5)
9
10  cap=cv.VideoCapture(0,cv.CAP_DSHOW)
11
12  while True:
13      ret,frame=cap.read()
14      if not ret:
15          print('프레인 획득에 실패하여 루프를 나갑니다.')
16          break
17
18      res=mesh.process(cv.cvtColor(frame,cv.COLOR_BGR2RGB))
19
20      if res.multi_face_landmarks:
21          for landmarks in res.multi_face_landmarks:
22
mp_drawing.draw_landmarks(image=frame,landmark_list=landmarks,connections=mp_mesh.FACEMESH_TESSELATION,landmark_drawing_spec=None,connection_drawing_spec=mp_styles.get_default_face_mesh_tesselation_style())
23
mp_drawing.draw_landmarks(image=frame,landmark_list=landmarks,connections=mp_mesh.FACEMESH_CONTOURS,landmark_drawing_spec=None,connection_drawing_spec=mp_styles.get_default_face_mesh_contours_style())
24
mp_drawing.draw_landmarks(image=frame,landmark_list=landmarks,connections=mp_mesh.FACEMESH_IRISES,landmark_drawing_spec=None,connection_drawing_spec=mp_styles.get_default_face_mesh_iris_connections_style())
25
26      cv.imshow('MediaPipe Face Mesh',cv.flip(frame,1)) # 좌우반전
```

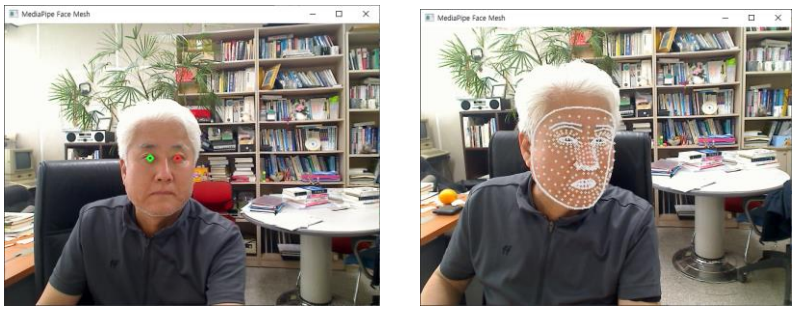

© 전북대학교 오일석

```
27          if cv.waitKey(5)==ord('q'):
28              break
29
30      cap.release()
31      cv.destroyAllWindows()
```



22~24행은 검출한 그물망을 그리는 역할을 하는데, 이 코드를 변형하여 [그림 10-13]과 같이 다양한 형태의 그물망을 그릴 수 있다. [그림 10-13(a)]는 22~24행에서 24행만 남기고 실행한 결과다. [그림 10-13(b)]는 23행만 남기고 23행을 아래와 같이 고치고 실행한 결과다.

```
mp_drawing.draw_landmarks(image=frame,landmark_list=landmarks,connections=mp_mesh.FACEMESH_CONTOURS,landmark_drawing_spec=mp_drawing.DrawingSpec(thickness=1,circle_radius=1))
```

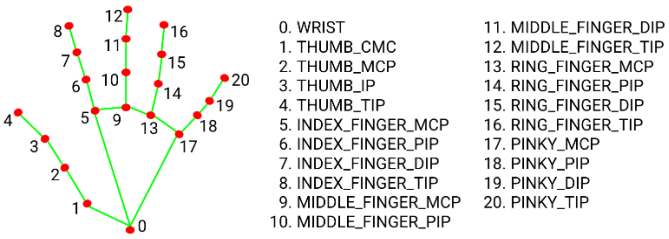


(a) 22~24행에서 24행만 남김 (b) 23행만 남기고 23행을 수정

[그림 10-13] [프로그램 10-7]의 22~24행을 변형하여 얼굴 그물망을 다양하게 그림

10.3.3 손 랜드마크 검출

손 랜드마크를 검출하는 BlazeHand 솔루션은 손을 검출하는 BlazePalm 모듈과 랜드마크를 검출하고 추적하는 모듈로 구성된다. 랜드마크는 [그림 10-14]가 보여주는 14개이다. BlazePlam은 얼굴을 검출하는 BlazeFace와 마찬가지로 SSD 모델을 개조하여 사용하였다 [Zhang2020]. 계산 시간을 절약하기 위해 프레임마다 손 검출을 적용하지 않고 모션 정보를 이용하여 이전 프레임에서 검출한 랜드마크를 현재 프레임에서 예측하는 방법을 사용한다. 검출한 랜드마크의 신뢰도가 임계값보다 낮으면 BlazePalm을 다시 적용하여 손을 새로 검출한다. 얼굴 그물망과 마찬가지로 손 랜드마크도 3차원 좌표로 표현한다. 보다 자세한 내용은 [Zhang2020]을 참조한다.



[그림 10-14] 손을 위한 21개의 랜드마크 [Zhang2020]

프로그래밍 실습: 손 랜드마크 검출

[프로그램 10-8]은 BlazeHand를 이용하여 비디오에 나타난 손의 랜드마크를 검출하는 프로그램이다. [프로그램 10-5]와 [프로그램 10-7]과 구조가 비슷하다.

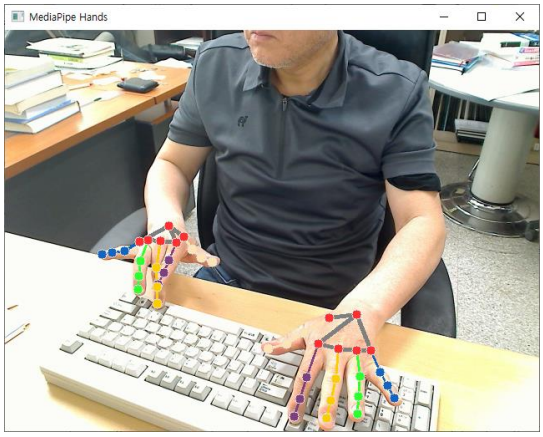
4행은 mediapipe 모듈의 solutions에서 얼굴 그물망 검출을 담당하는 hands 모듈을 읽어 mp_hand 객체에 저장한다. 8행은 mp_hand가 제공하는 Hands 클래스로 손 랜드마크 검출에 쓸 hand 객체를 생성한다. 두번째 인수 static_image_mode=False는 입력을 비디오로 간주하고 첫 프레임에 손 검출을 담당하는 BlazeHand를 적용하고 이후에는 추적을 사용하라고 지시한다. True로 설정하면 매 프레임에 BlazeHand를 적용하기 때문에 정확도는 높아지고 대신 시간은 더 걸린다. 나머지 인수에 대한 설명은 [프로그램 10-7]의 8행과 비슷하다. 18행의 process 함수는 실제 손 랜드마크 검출을 수행하고 결과를 res에 저장한다. 20행은 검출된 손이 있는지 확인하고, 21행은 검출된 손 각각에 대해 그물망을 그리는 일을 반복한다. 22행은 손 랜드마크를 그린다.

[프로그램 10-8] 손 랜드마크 검출

```
1   import cv2 as cv
2   import mediapipe as mp
3
4   mp_hand=mp.solutions.hands
```


© 전북대학교 오일석

```
5   mp_drawing=mp.solutions.drawing_utils
6   mp_styles=mp.solutions.drawing_styles
7
8   hand=mp_hand.Hands(max_num_hands=2,static_image_mode=False,min_detection_confidence=0.5,mi
    n_tracking_confidence=0.5)
9
10  cap=cv.VideoCapture(0,cv.CAP_DSHOW)
11
12  while True:
13      ret,frame=cap.read()
14      if not ret:
15          print('프레인 획득에 실패하여 루프를 나갑니다.')
16          break
17
18      res=hand.process(cv.cvtColor(frame,cv.COLOR_BGR2RGB))
19
20      if res.multi_hand_landmarks:
21          for landmarks in res.multi_hand_landmarks:
22
    mp_drawing.draw_landmarks(frame,landmarks,mp_hand.HAND_CONNECTIONS,mp_styles.get_default_h
    and_landmarks_style(),mp_styles.get_default_hand_connections_style())
23
24      cv.imshow('MediaPipe Hands',cv.flip(frame,1)) # 좌우반전
25      if cv.waitKey(5)==ord('q'):
26          break
27
28  cap.release()
29  cv.destroyAllWindows()
```



 © 전북대학교 오일석

10.4.2 프로그래밍 실습: BlazePose를 이용한 자세 추정

MediaPipe는 자세 추정을 해주는 BlazePose 솔루션을 제공한다. BlazePose는 [그림10-15(a)]가 보여주는 33개 랜드마크로 자세를 표현한다 [Bazarevsky2020]. 깊이 정보까지 추정하여 랜드마크를 3차원 좌표로 표현한다. 17개 랜드마크로 레이블링한 COCO 데이터셋보다 많은 랜드마크를 검출하기 때문에 요가나 운동 자세 분석, 수화 인식과 같은 응용에 쉽게 활용할 수 있는 장점이 있다.

입력 영상이 들어오면 몸을 감싸는 ROI를 검출하여 자세 추정 신경망 모델에 입력한다. 랜드마크를 검출하는 접근방법에는 랜드마크 좌표 자체를 예측하는 좌표 회귀와 [그림 10-16]과 같이 랜드마크에 가우시안을 씌운 열지도를 사용하는 열지도 회귀가 있다. 기존 신경망이 둘 중 하나를 사용하는데, BlazePose는 둘을 모두 써서 성능을 높인다 [Bazarevsky2020]. 학습할 때는 열지도 출력에 대해 손실 함수를 계산함으로써 학습 성능을 높인다. 학습을 마치고 추론하는 과정에서는 열지도 출력 부분을 떼내고, 33*3 텐서 출력을 취한다. 이 텐서는 33개 랜드마크에 대해 (x,y) 좌표와 보이는지 여부를 표시하는 값으로 구성된다.

아주 다양한 자세가 있기 때문에 몸 전체를 포함하는 ROI를 직접 찾는 일은 어렵다. BlazePose는 먼저 BlazeFace로 얼굴을 찾은 다음 두 어깨 중심과 두 엉덩이 중심을 잇는 선을 기준으로 전신 ROI를 검출하는 간접 방법을 사용한다. 빠른 속도를 확보하려고, ROI를 검출하는 일은 첫 프레임에만 적용하고 이후 프레임에서는 이전 프레임에서 알아낸 랜드마크를 활용하여 예측하는 방법을 사용한다. 만일 예측한 ROI에 사람이 없다고 판단되면 다시 ROI 검출을 적용한다.

프로그래밍 실습

[프로그램 10-9]는 BlazePose를 이용하여 비디오에 나타난 사람의 자세를 추정하는 프로그램이다. [프로그램 10-5]와 [프로그램 10-7], [프로그램 10-8]과 구조가 비슷하다.

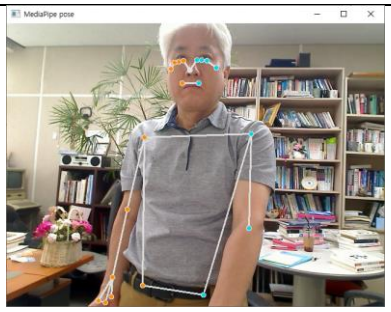
4행은 mediapipe 모듈의 solutions에서 자세 추정을 담당하는 pose 모듈을 읽어 mp_pose 객체에 저장한다. 8행은 mp_pose가 제공하는 Pose 클래스로 손 랜드마크 검출에 쓸 pose 객체를 생성한다. 첫번째 인수 static_image_mode=False는 입력을 비디오로 간주하고 첫 프레임에 ROI 검출을 적용하고 이후에는 추적을 사용하라고 지시한다. 두번째 인수 enable_segmentation=True는 전경과 배경을 분할하라고 지시한다. 이렇게 설정하면 배경을 흐릿하게 만드는 등의 응용에 쓸 수 있다. 나머지 인수에 대한 설명은 [프로그램 10-7]의 8행과 비슷하다. 18행의 process 함수는 실제 자세 추정을 수행하고 결과를 res에 저장한다. 20행은 자세를 원본 영상에 겹쳐 보여준다. 21행은 3차원 공간에 자세를 보여준다.

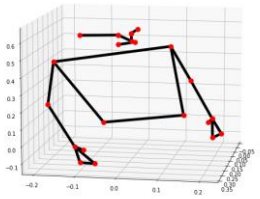
 © 전북대학교 오일석

[프로그램 10-9] BlazePose를 이용한 자세 추정

```
1  import cv2 as cv
2  import mediapipe as mp
3
4  mp_pose=mp.solutions.pose
5  mp_drawing=mp.solutions.drawing_utils
6  mp_styles=mp.solutions.drawing_styles
7
8  pose=mp_pose.Pose(static_image_mode=False,enable_segmentation=True,min_detection_confidence=
   0.5,min_tracking_confidence=0.5)
9
10 cap=cv.VideoCapture(0,cv.CAP_DSHOW)
11
12 while True:
13     ret,frame=cap.read()
14     if not ret:
15        print('프레인 획득에 실패하여 루프를 나갑니다.')
16        break
17
18     res=pose.process(cv.cvtColor(frame,cv.COLOR_BGR2RGB))
19
20
   mp_drawing.draw_landmarks(frame,res.pose_landmarks,mp_pose.POSE_CONNECTIONS,landmark_drawi
   ng_spec=mp_styles.get_default_pose_landmarks_style())
21     mp_drawing.plot_landmarks(res.pose_world_landmarks,mp_pose.POSE_CONNECTIONS)
22
23     cv.imshow('MediaPipe pose',cv.flip(frame,1)) # 좌우반전
24     if cv.waitKey(5)==ord('q'):
25        break
26
27 cap.release()
28 cv.destroyAllWindows()
```

 © 전북대학교 오일석





 © 전북대학교 오일석