교육부 Ministry of Education

NRF 한국연구재단 National Research Foundation of Korea

OpenCV를 활용한 객체 추적

COSS 인공지능 Convergence and Open Sharing System

전남대학교 CHONNAM NATIONAL UNIVERSITY

성균관대학교 SUNGKYUNKWAN UNIVERSITY

KNU 경북대학교 KYUNGPOOK NATIONAL UNIVERSITY

전주대학교 JEONJU UNIVERSITY

서울시립대학교 UNIVERSITY OF SEOUL

서울과학기술대학교 SEOUL NATIONAL UNIVERSITY OF SCIENCE & TECHNOLOGY

영진전문대학교 YEUNGJIN UNIVERSITY

Contents

- 외곽선 검출을 활용한 차선 인식
- 얼굴인식

외곽선 검출을 활용한 차선 인식

외곽선 검출을 활용한 차선 인식 개요

❖이미지 처리 기법을 활용하여 이미지 또는 영상에서 차선을 인지하는 프로그램을 작성

❖차선 인식을 위한 이미지 처리 순서

① 이미지 불러오기 ☞ 카메라가 차선을 인식하도록 아래로 향한다.

② 경계선(edge) 검출(Canny Edge Detection)

- 색상 변환 (RGB -> Gray) 및 가우시안(Gaussian) 필터를 활용한 노이즈 필터링 포함

③ 관심 구역(ROI : Region of Interest) 설정

- 화면의 관심 구역 이외의 영역은 제거(crop)

④ 관심 구역 內 조건을 충족하는 선 찾기

⑤ 이미지에 차선 그리기

① 이미지 불러오기

❖**OpenCV의 imread()를 통해 차선 인식에 활용할 이미지 불러오기**

- 인식된 차선 등의 계산을 위해 numpy를 활용

```
01:  import cv2
02:  import numpy as np
03:
04:  image = cv2.imread("test.png")
05:  cv2.imshow("test",image)
```

② 경계선(edge) 검출

❖이미지의 색상을 회색톤으로 변경(RGB -> Gray)

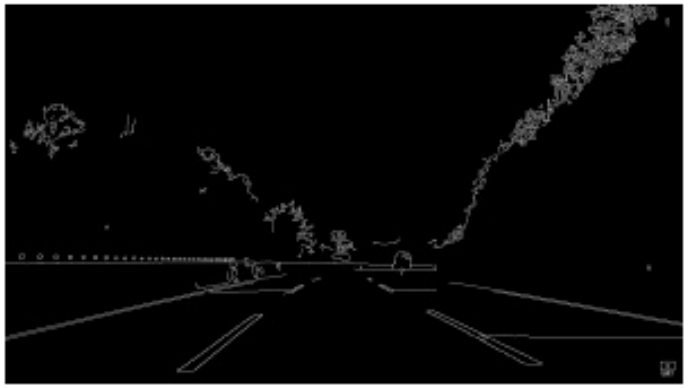
❖가우시안 필터를 통한 노이즈 제거

- 5x5 크기의 가우시안 필터 사용

❖엣지 검출

- Canny(이미지, 최소 Threshold, 최대 Threshold)
  - 최소/최대 Threadshold: 에지 여부를 판단하는 임계값. 작을수록 에지가 검출되기 쉽고, 클수록 검출되기 어렵다.

```
01:    def edge_detector(frame):
02:        gray = cv2.cvtColor(frame,
cv2.COLOR_RGB2GRAY)
03:        blur = cv2.GaussianBlur(gray, (5, 5), 0)
04:        canny = cv2.Canny(blur, 50, 150)
05:        return canny
06:
07:    canny_image = edge_detector(image)
08:    cv2.imshow("canny",canny_image)
```



③ 관심 구역(ROI) 설정

❖에지 검출한 이미지에서 차선 검출에 활용할 구역을 설정

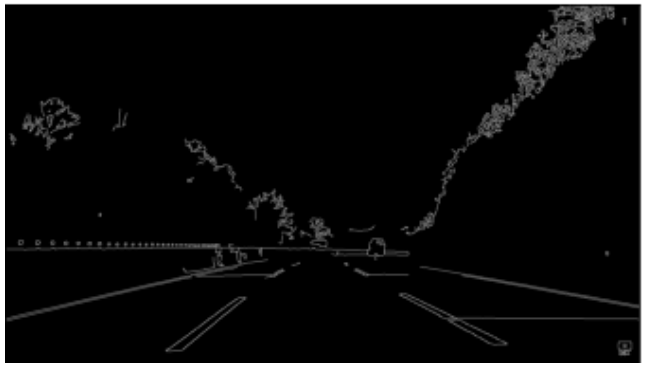
- ROI는 이미지 크기, 검출하려는 선의 형태 등에 따라 달라질 수 있음

❖ROI 이외 부분은 검정색인 이미지를 생성

❖생성된 검정색 이미지와 에지 검출에 활용할 이미지의 비트 연산을 진행 ☜ ROI 이외 부분을 제거

```
01:    def roi_mask(image):
02:        height = image.shape[0]
03:        width = image.shape[1]
04:        polygons = np.array([[(0, height), (round(width/2), round(height/2)), (1000,
height)]])
05:        mask = np.zeros_like(image)
06:        cv2.fillPoly(mask, polygons, 255)
07:        masked_image = cv2.bitwise_and(image, mask)
08:        return masked_image
09:
10:    cropped_image = roi_mask(canny_image)
11:    cv2.imshow("roi_image",cropped_image)
```

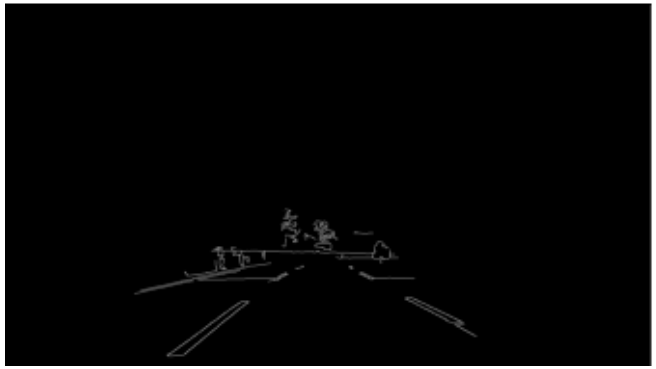
③ 관심 구역(ROI) 설정 –cont.



Edge Image

bitwise and

ROI Mask Image



④ 관심 구역 內 조건을 충족하는 선 찾기

❖조건을 충족하는 선 찾기

- 허프 변환(Hough Transform)을 통해 ROI 내의 선을 검출
  - 허프 변환은 이미지에서 모양을 찾는 기법, 직선, 원 등의 다양한 모양을 인식 할 수 있음
  - 기본적인 허프 변환은 직선의 방정식을 활용
- cv2.HoughLines()
  - 입력 이미지: ROI 설정까지 완료된 cropped image
    - 이외의 이미지를 활용할 경우 이진화된 이미지를 활용 해야함
  - rho: 거리의 정밀도 (ρ값의 간격)
  - theta: 각도의 정밀도 (Θ값의 간격)
  - threshold: 누적배열에서 직선으로 판단할 임계값
  - (옵)lines: 직선 정보(Θ,ρ)를 담고있는 배열
  - (옵)min_theta: 검출되는 선의 최소 각도
  - (옵)max_theta: 검출되는 선의 최대 각도
  - 결과값: Θ와 ρ로 직선 파라미터 정보

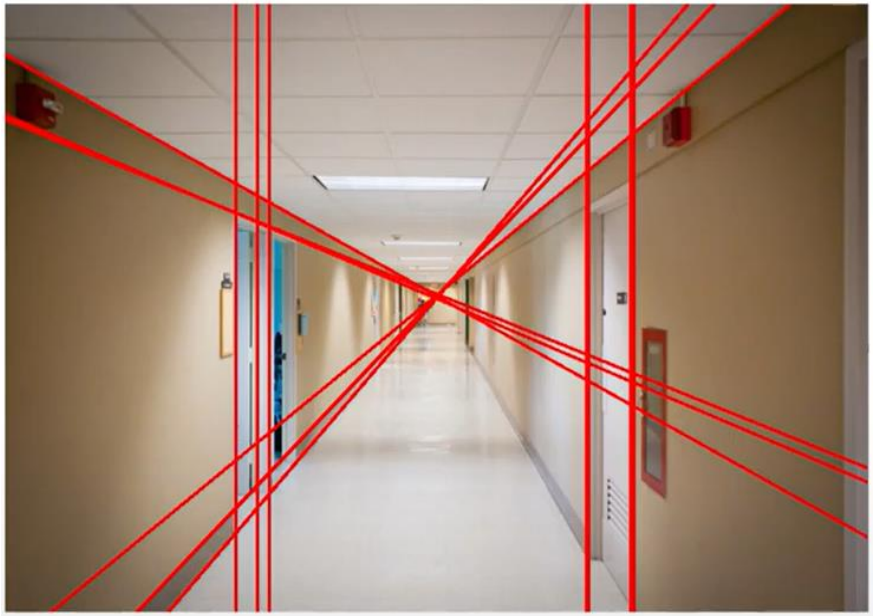
```
results = cv2.HoughLines(
          cropped_image,
          rho=2,
          theta=np.pi / 180,
          threshold=200)
```

④ 관심 구역 內 조건을 충족하는 선 찾기 –cont.

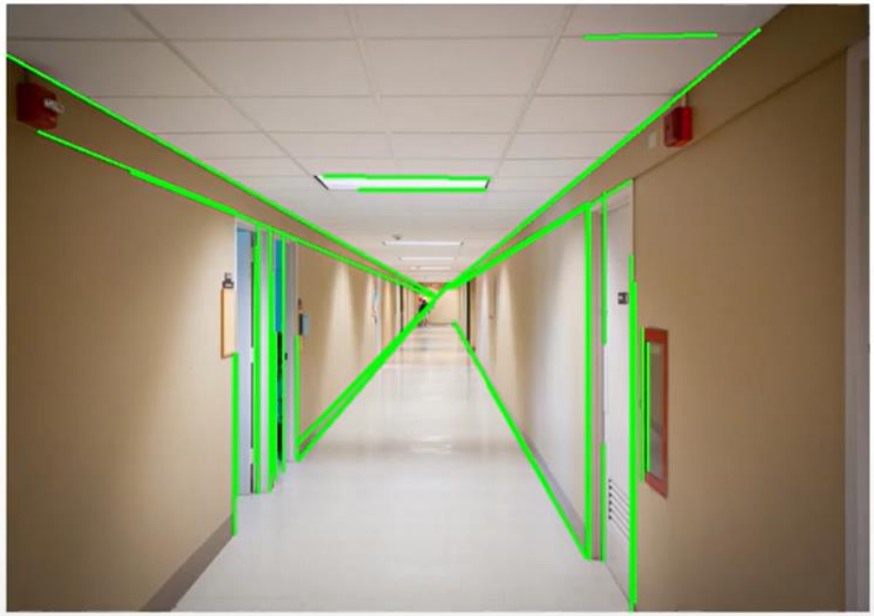
- cv2.HoughLinesP()
  - 확률적 허프변환 함수
  - cv2.HoughLines()는 모든 점에 대해 수많은 선을 그어 직선을 찾기 때문에 많은 연산을 필요
    - cv2.HoughLinesP()는 모든 점을 고려하지 않고 무작위 픽셀을 선정하여 허프변환을 수행하여 점점 그 수를 증가시키는 방법
  - 대부분 입력값은 cv2.HoughLines()와 동일하나, 아래만 다름
    - minLineLength: 검출되는 선의 최소 길이
    - maxLineGap: 직선으로 간주할 점들의 최대 간격
  - 결과값으로 직선의 시작과 끝 정보 제공
  - cv2.HoughLines() 함수에 비해 선 검출이 적게 되므로, 엣지를 강하게 하고 threshold 값을 낮게 지정해야 함

```
results = cv2.HoughLinesP(
            cropped_image,
            rho=2,
            theta=np.pi / 180,
            threshold=100,
            lines=np.array([]),
            minLineLength=40,
            maxLineGap=25
        )
```

④ 관심 구역 內 조건을 충족하는 선 찾기 –cont.



cv.HoughLines



cv.HoughLinesP

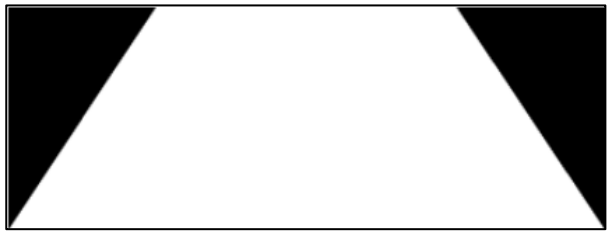
④ 관심 구역 內 조건을 충족하는 선 찾기 –cont.

❖조건을 충족하는 선 찾기 –cont.

- 허프 변환 조건에 충족한 선들의 데이터를 토대로 왼쪽 차선과 오른쪽 차선을 검출
- (x1,y1) 과 (x2,y2)를 지나는 직선 그래프를 계산
- 직선의 기울기를 토대로 차선인지 여부와 위치를 판단
  - 일반적인 차선은 차량 내부에서 바라보는 경우 차선의 기울기는 30 ~ 60도 사이 형성
- 조건이 충족된 선들의 평균을 통해 검출된 선들의 중간 값을 계산
- 검출된 선의 화면에 표시할 때 활용할 점 좌표를 계산

전체코드 1/3

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import copy

def get_mask(img):   # ROI 구역을 가져올 수 있는 mask 생성 및 반환
    mask = np.zeros_like(img)
    if len(img.shape) > 2:
        channel_count = img.shape[2]
        ignore_mask_color = (255,) * channel_count
    else:
        ignore_mask_color = 255
    imshape = img.shape
 vertics = np.array([[(0, int(imshape[0])-1),(int(imshape[1]/4)-1, 0),
(int(imshape[1]*3/4)-1, 0), (int(imshape[1])-1, int(imshape[0])-1)]])
    cv2.fillPoly(mask, vertics, ignore_mask_color)
    return mask

# 이미지에 대한 허프변환을 실시하고 라인을 그린 영상을 반환
def hough_lines(crop_img, edge_img, rho=2, theta=np.pi/180, threshold=30,
min_line_len=20, max_line_gap=10):
    lines = cv2.HoughLinesP(edge_img, rho, theta, threshold, min_line_len,
max_line_gap)
    if lines is not None:
        for line in lines:
            for x1,y1,x2,y2 in line:
                cv2.line(crop_img, (x1,y1), (x2,y2), (0,0,255), 3)
    return crop_img
```



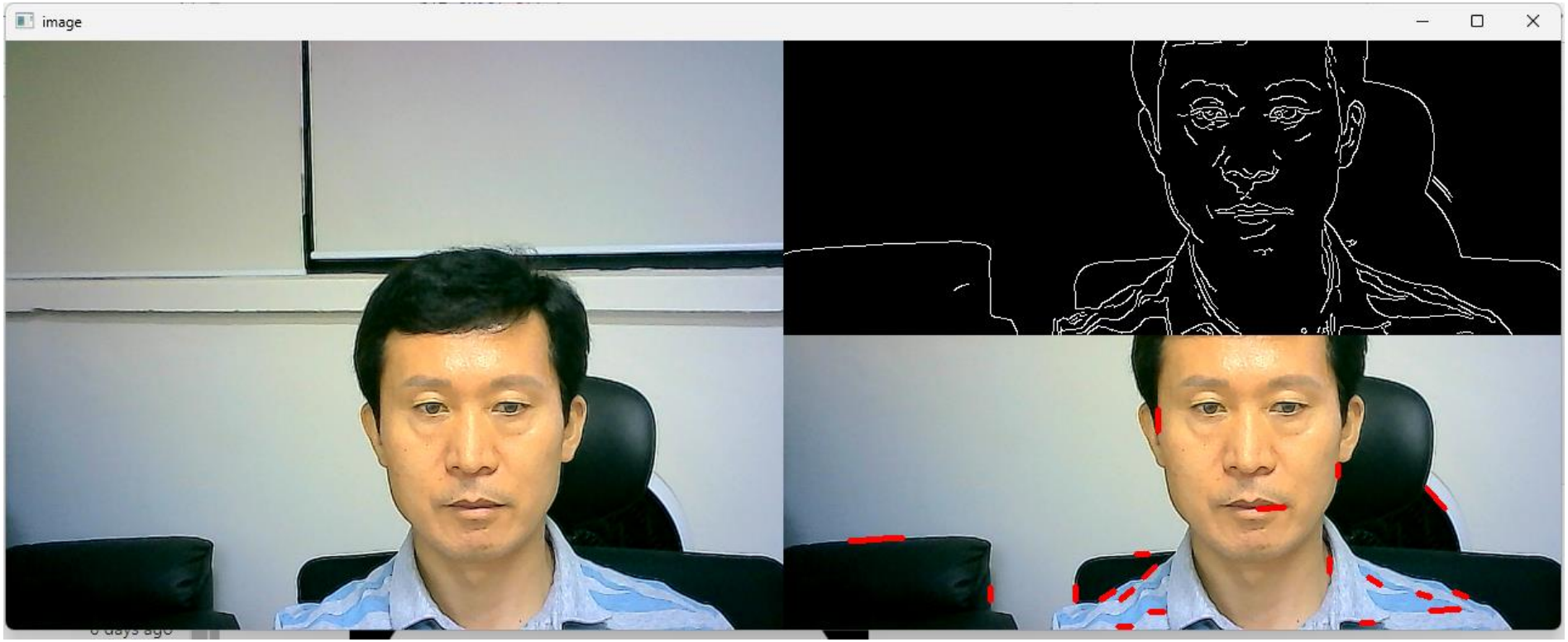
전체코드 2/3

```
# 영상 가져오기 초기화
camera = cv2.VideoCapture(0)
if not camera.isOpened():
    print('not found camera')
width = camera.get(cv2.CAP_PROP_FRAME_WIDTH)
height = camera.get(cv2.CAP_PROP_FRAME_HEIGHT)

flag = True   # 마스크 이미지를 한번 보여주기 위한 flag

# 카메라로부터 프레임 가져오기
while True:
    ret, frame = camera.read()
    if not ret:
        break
    crop_img = copy.deepcopy(frame[int(height/2):, :])   # 세로 중간 아래 배열들만 가져옴

    # 경계선(에지) 검출
    gray_img = cv2.cvtColor(crop_img, cv2.COLOR_BGR2GRAY)
    blur_img = cv2.GaussianBlur(gray_img, (5,5), 0)
    edge_img = cv2.Canny(blur_img,60, 100)
    # ROI 가져오기
    mask = get_mask(edge_img)
```

전체코드 3/3

```
    # 처음에 한번 마스크 이미지를 보여주기
    if flag:
        plt.figure(figsize =(5,4))
        plt.axis('off')
        plt.imshow(mask, cmap='gray')
        plt.show()
        flag = False

    # 경계선 영상과 마스크 영상을 bitwise_and 연산
    masked_img = cv2.bitwise_and(edge_img, mask)

    #  ROI 내 허프변환에 의한 선이 표시된 이미지 그리기
    hough_img = hough_lines(crop_img, masked_img)

    # hough_img 칼라영상과 edge_img 흑백영상을 수직 병합하고 원본영상과는 수평 병합하여 출력
    edge_img_3d = cv2.cvtColor(edge_img, cv2.COLOR_GRAY2BGR) # 칼라영상과 병합위해 3채널화
    cv2.imshow('image', cv2.hconcat([frame, np.vstack((edge_img_3d, hough_img))]))

    ''' 이동제어 루틴 코드 영역 '''

    if cv2.waitKey(10) == 27: break     # ESC 화면 종료

camera.release()
cv2.destroyAllWindows()
```

출력결과



[참고] 이미지에 차선 그리기

❖검출된 선의 좌표를 토대로 원본 이미지에 검출된 선을 표시

```
01:   def draw_lines(image, lines, thickness):
02:     line_image = np.zeros_like(image)
03:     color=[0, 0, 255]
04:     if lines is not None:
05:         for x1, y1, x2, y2 in lines:
06:             cv2.line(line_image, (x1, y1), (x2, y2), color, thickness)
07:     combined_image = cv2.addWeighted(image, 0.8, line_image,
1.0, 0.0)
08:     return combined_image
09:
10:  combined_image = draw_lines(image, averaged_lines, 5)
11:  cv2.imshow("combined_image",combined_image)
```

얼굴인식

얼굴 인식

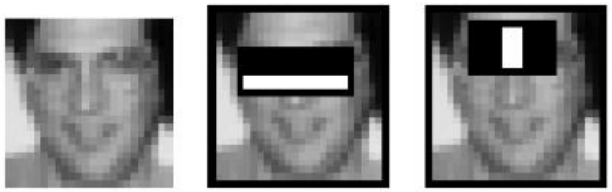
❖OpenCV 라이브러리에는 얼굴인식 관련 데이터 모델과 얼굴 인식 알고리즘을 기본 제공

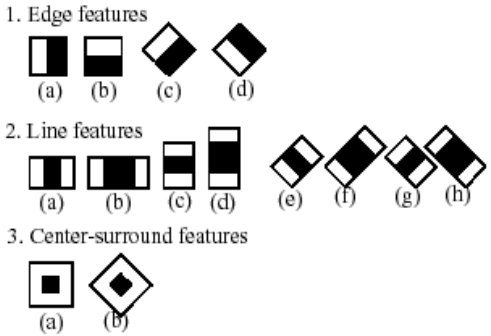
❖카메라로 입력된 영상에서 사람 얼굴을 인지하는 프로그램 작성 가능

Haar Cascades 얼굴 검출 개요

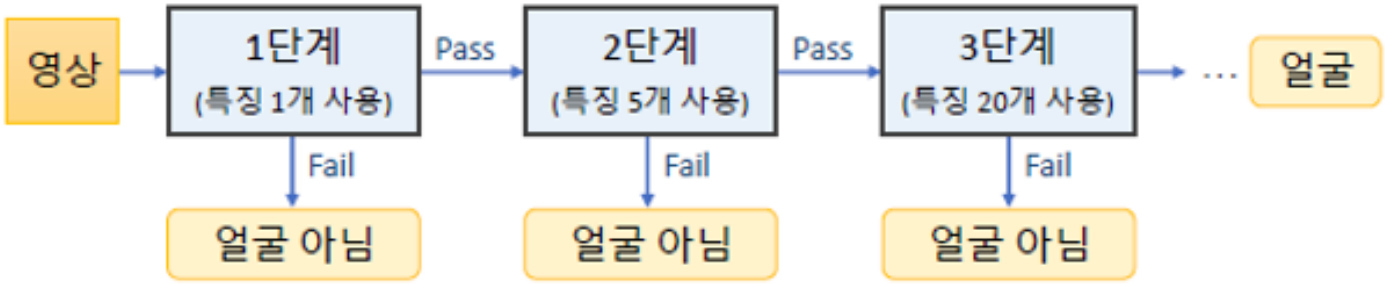
❖직사각형 영역으로 구성되는 Haar 특징을 사용

- 픽셀단위로 객체를 검출하는 방법보다 동작 속도 측면에서 검출 속도가 빠름
- 이미지의 대부분의 공간은 얼굴이 없는 영역이기 때문에 얼굴영역인지를 판단 → 속도 증가
  - 유사 Haar 특징 (Haar-like features)





❖단계별로 얼굴인지를 체크하는 Cascade 식별 절차 적용



Haar Cascades 얼굴 검출 절차 동영상

Haar Cascades 얼굴 검출 분류기

❖OpenCV에서는 Haar Cascades 알고리즘으로 생성된 분류기를 xml 파일로 제공

파일명	검출 대상
haarcascade_frontalface_default.xml haarcascade_frontalface_alt.xml haarcascade_frontalface_alt2.xml haarcascade_frontalface_alt_tree.xml	정면 얼굴 검출
haarcascade_profileface.xml	측면 얼굴 검출
haarcascade_smile.xml	웃음 검출
haarcascade_eye.xml haarcascade_eye_tree_eyeglasses.xml haarcascade_lefteye_2splits.xml haarcascade_righteye_2splits.xml	눈 검출
haarcascade_frontalcatface.xml haarcascade_frontalcatface_extended.xml	고양이 얼굴 검출
haarcascade_fullbody.xml	사람의 전신 검출
haarcascade_upperbody.xml	사람의 상반신 검출
haarcascade_lowerbody.xml	사람의 하반신 검출
haarcascade_russian_plate_number.xml haarcascade_licence_plate_rus_16stages.xml	러시아 자동차 번호판 검출

❖다운로드

- https://github.com/opencv/opencv/tree/master/data/haarcascades

Haar Cascades 얼굴 검출 프로그래밍 절차

❖**설치된 분류기를 사용할 때는 CascadeClassifier 활용**

- 인자로 미리 학습된 분류기의 경로를 넣을 경우 로드 후 사용 가능
- 상당한 시간이 소요되는 학습과정 없이 빠르게 활용 가능

❖**CascadeClassifier(cascPath): 분류기를 로드하여 반환**

- cascPath : 분류기 파일의 경로

❖**detectMultiScale(image, scaleFactor, minNeighbors, minSize):**

- image: 입력 영상 파일
- scaleFactor: 영상 확대 비율. 1보다 커야 함. (기본값 1.1)
- minNeighbors: 얼굴로 인식되기 위해 필요한 검출되는 이웃 사각형의 개수(기본값 3)
- minSize: 최소 객체(얼굴) 크기(w, h)
- maxSize: 최대 객체(얼굴) 크기(w, h)
- 결과값: 검출된 객체의 사각형 정보(x, y, w, h)

얼굴 인식 예제 1/4

❖OpenCV의 앞면 인식 Haar Cascade 분류기 로드

```
01:     import cv2
02:
03:     haar_face = 'haarcascade_frontalface_default.xml'
04:     face_cascade = cv2.CascadeClassifier(haar_face)
```

❖VideoCapture 객체 생성

```
05:     camera = cv2.VideoCapture(0)
06:
07:     if not camera.isOpened():
08:         print("Not found camera")
09:
```

얼굴 인식 예제 2/4

❖카메라 프레임별 반복처리 루틴

- 입력받은 프레임을 회색톤으로 변환
- detectMultiScale() 메소드로 얼굴 검출
  - scaleFactor의 값을 줄일 경우
    - 정확도가 늘어날 수가 있지만 속도가 느려짐
  - minNeighbors의 값을 늘릴 경우
    - 정확도가 늘어날 수 있지만 해상도가 떨어지는 이미지에서는 검출 실패 가능성 있음

```
10:          while True:
12:              ret, img = camera.read()
13:              if ret:
13:                  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) ☜ 옵션
14:                  faces= face_cascade.detectMultiScale(gray,
scaleFactor=1.3 ,minNeighbors=1,minSize=(100,100))
```

얼굴 인식 예제 3/4

❖카메라 프레임별 반복처리 루틴 -계속

- 원본 이미지에서 찾은 객체들의 위치에 사각형 그리기
- 완성된 이미지 출력

```
15:                 for (x,y,w,h) in faces:
16:                     cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)
17:                 cv2.imshow('img', img)
18:             if cv2.waitKey(0)) == 27: break    # ESC
```

❖객체 반환 및 종료

```
19:     camera.release()
20:     cv2.destroyAllWindows()
```

얼굴 인식 예제 4/4

❖전체 코드

```
01:         import cv2
02:
03:             haar_face='haarcascade_frontalface_default.xml'
04:             face_cascade = cv2.CascadeClassifier(haar_face)
05:             camera = cv2.VideoCapture(0)
06:
07:             if not camera.isOpened():
08:                 print("Not found camera")
09:
10:                 while True:
12:                 ret, img = camera.read()
13:                 if ret:
13:                     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
14:                     faces= face_cascade.detectMultiScale(gray, scaleFactor=1.3 , minNeighbors=1,
minSize=(100,100))
15:                     for (x,y,w,h) in faces:
16:                         cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)
17:                     cv2.imshow('img', img)
18:                 if cv2.waitKey(0)) == 27: break    # ESC
19:             camera.release()
20:             cv2.destroyAllWindows()
```

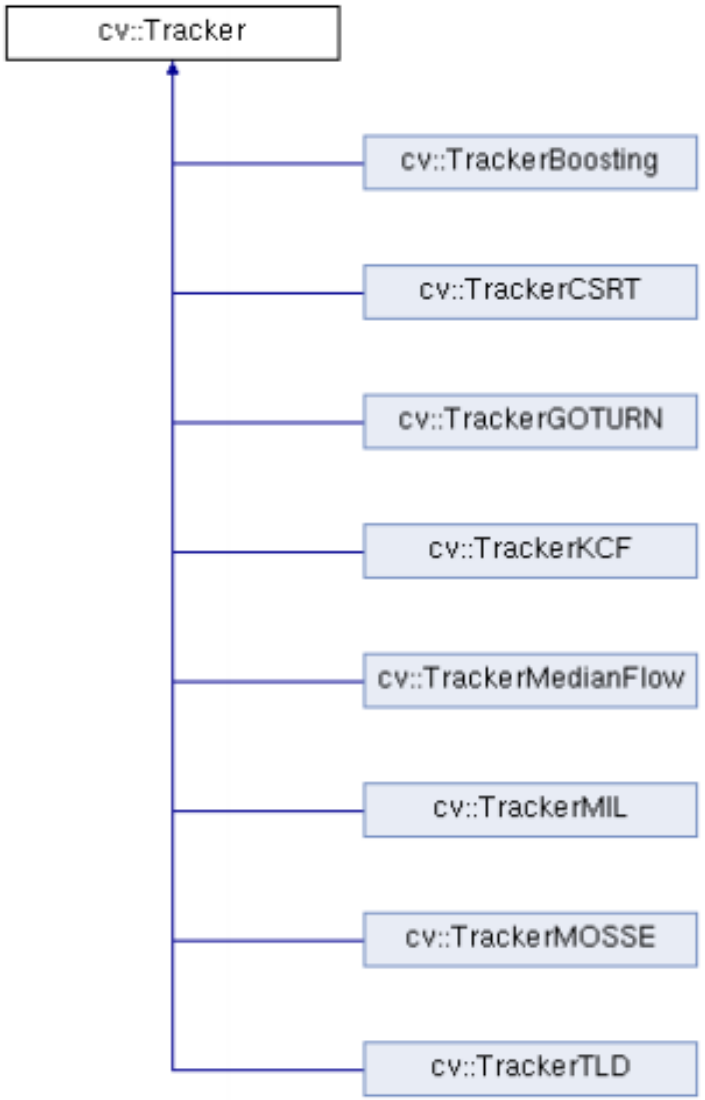
객체 추적

OpenCV 객체 추적 알고리즘 1/2

❖**OpenCV의 extra 모듈을 설치 해야 사용가능**

❖**알고리즘**

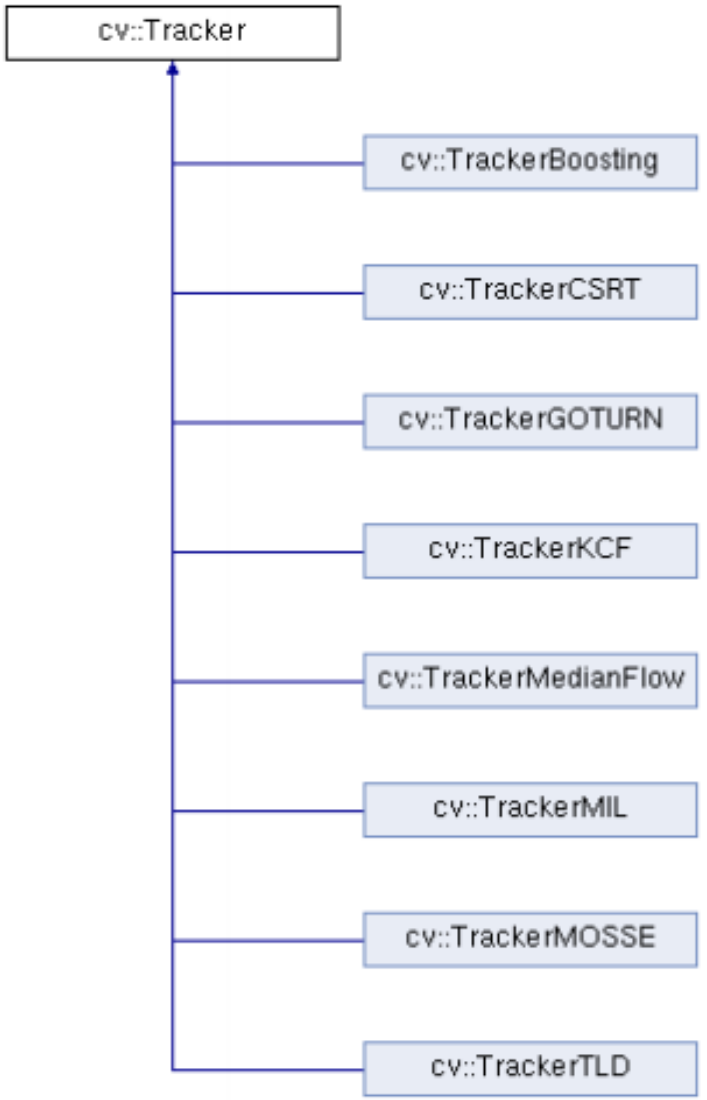
- Boosting: AdaBoost 알고리즘 기반이며 성능이 보통이며 느림
- CSRT : 연산은 느리지만 강인하게 추적
- GOTURN : 딥러닝 기반, 가중치 파일을 다운받아 저장해야 작동 가능
- KCF : 단일 개체를 추적하는 데 매우 잘 빠르게 작동
- MedianFlow: 객체의 전/역방향을 추적해서 불일치성을 측정



OpenCV 객체 추적 알고리즘 2/2

❖알고리즘 -계속

- MIL: Boosting 추적기를 개선
- MOSSE : 빠르게 작동
- TLD: 이전 프레임의 위치를 사용하여 추적하면서 학습



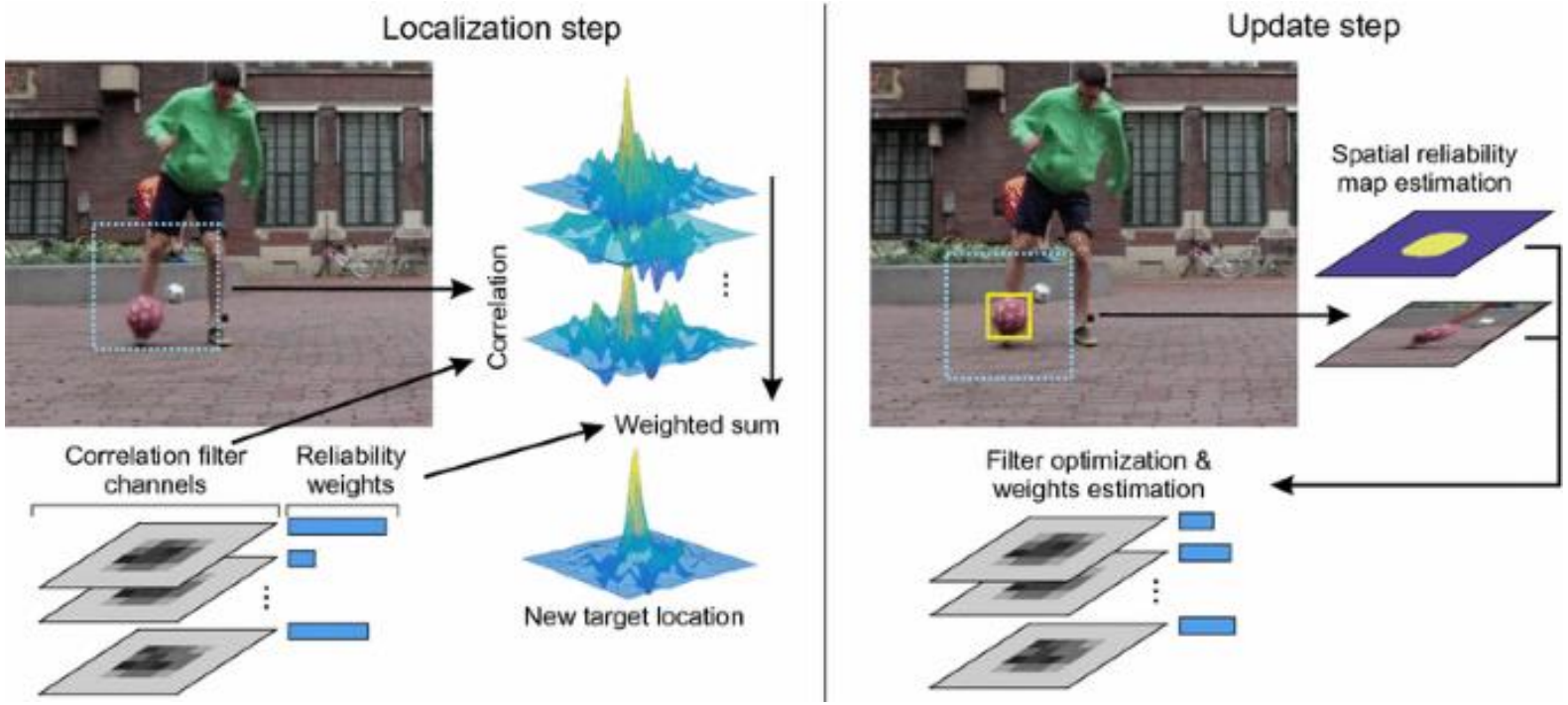
❖사용법

- 클래스 객체 생성
  - cv2.TrackerXXX_create()
- 객체 초기화
  - cv2.Tracker.init(image, boundingBox)
- 정보 업데이트
  - cv2.Tracker.update(image)

CSRT (Channel and Spatial Reliability Tracking)

❖객체를 추적하기 위해 채널과 공간적인 신뢰도를 사용

- 채널 신뢰도: 객체의 특징과 색상 변화를 측정하기 위한 RGB 차원의 신뢰도를 사용
- 공간 신뢰도: 추적하는 객체 주위의 텍스처 및 특징을 고려하여 움직임을 예측하는 공간적인 신뢰도를 사용



TLD (Tracking-Learning-Detection)

❖추적하면서 학습

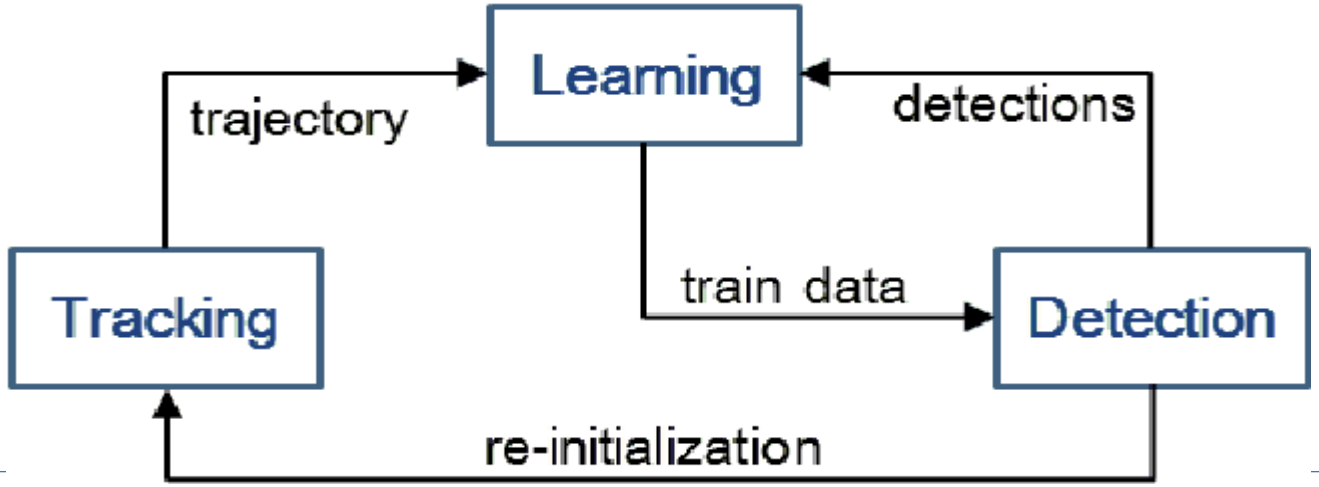
① 검출: CNN 기반의 객체 검출(식별)하여 ROI 구역 반환

② 학습: 검출된 객체를 추적하도록 추적 모델을 학습

③ 추적: 학습된 추적 모델을 사용하여 객체를 실시간 추적

- 이전 프레임에서 객체의 위치와 특징을 사용하여 현재 프레임에서 객체의 새로운 위치를 예측

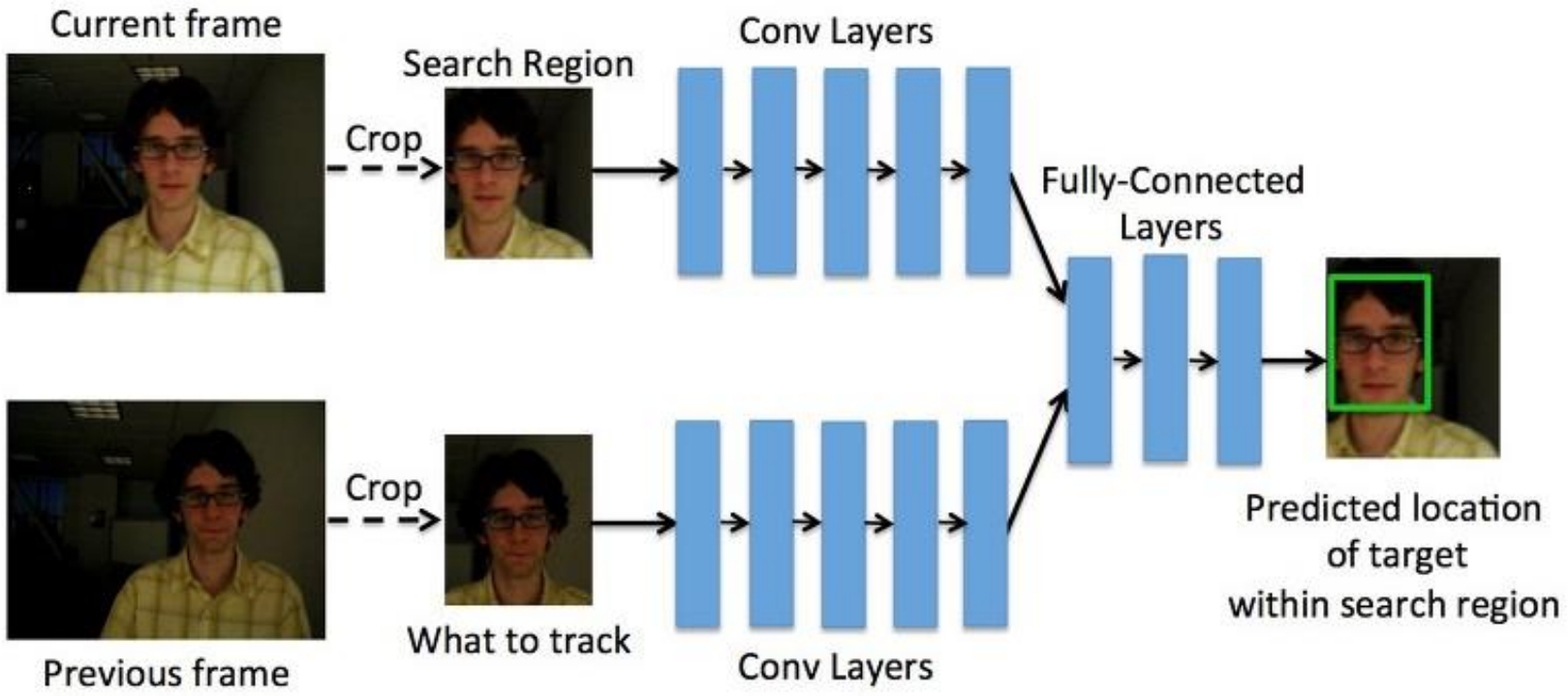
- 객체 추적의 정확도와 신뢰성을 높이기 위해 지속적으로 학습하면서 객체를 추적



GOTURN (Generic Object Tracking Using Regression Networks)

❖딥러닝 기반의 물체 추적방식

- 이전 프레임의 ROI를 가지고 현재 프레임에서 가장 유사한 구역을 비교하여 찾음.
- 실행을 위해서는 딥러닝 학습된 데이터 파일 필요
  - caffemodel, prototxt: https://github.com/Mogball/goturn-files
  - 압축풀기: 윈도우에서는 cat 대신 type 명령어



OpenCV 객체 추적 알고리즘 비교

알고리즘	지원버전	설명	비고
Boosting	3.0.0	가장 오래된 추적알고리즘으로 성능 떨어짐	
MIL	3.0.0	Boosting에 비해 정확도 개선	
KCF	3.1.0	Boosting과 MIL보다 속도 빠름	
CSRT	3.5.2	KCF보다 정확(높은 정확도)하지만 느림	
MedianFlow	3.0.0	성능은 좋으나 빠르면 놓침	
TLD	3.0.0	성능은 좋으나 느림	
MOSSE	3.4.1	성능은 떨어지지만 매우 빠름	
GOTURN	3.2.0	딥러닝 기반으로 추가모델 파일이 필요함	

OpenCV 모듈 재설치

❖**OpenCV의 tracker 모듈을 활용하려면, opencv-python은 삭제하고 opencv-contrib-python만을 설치해야 함**

- pip uninstall opencv-python
- pip install opencv-contrib-python
- jupyterlab 재시작

❖**예제**

- tracker1 = cv2.legacy.TrackerBoosting_create()
- tracker2 = cv2.legacy.TrackerCSRT_create()
- tracker3 = cv2.legacy.TrackerKCF_create()
- tracker4 = cv2.legacy.TrackerMIL_create()
- tracker5 = cv2.legacy.TrackerMOSSE_create()
- tracker6 = cv2.legacy.TrackerMedianFlow_create()
- tracker7 = cv2.legacy.TrackerTLD_create()

⇒ 위에서 일부는 legacy를 생략해도 가능

객체 추적 예제코드

```
import cv2, sys

# 카메라 장치 열기
cap = cv2.VideoCapture(0)

if not cap.isOpened():
    print('Video open failed!')
    sys.exit()

# 트래커 객체 생성
#tracker = cv2.TrackerKCF_create()
#tracker = cv2.TrackerMOSSE_create()
#cv2.legacy.TrackerTLD_create()
tracker = cv2.TrackerCSRT_create()

#GOTURN
#import os
#if (os.path.isfile('goturn.caffemodel') and
os.path.isfile('goturn.prototxt')):
#    tracker = cv2.TrackerGOTURN_create()

# 첫 번째 프레임에서 추적 ROI 설정
ret, frame = cap.read()
if not ret:
    print('Frame read failed!')
    sys.exit()

# ROI를 선택하고 Enter키를 치면 추적 시작
bbox = cv2.selectROI(frame, False)
tracker.init(frame, bbox) # 객체 초기화

# 매 프레임 처리
while True:
    ret, frame = cap.read()
    if not ret:
        print('Frame read failed!'); sys.exit()

    # 추적 ROI 사각형(bbox) 업데이트
    # 매 프레임마다 update하고 bbox값 받아옴
    ret, bbox = tracker.update(frame)

    # floate 형태로 bbox값을 받으므로 int로 변
환해서 list로 감싸고 tuple로 변환
    bbox = tuple([int(_) for _ in bbox])
    cv2.rectangle(frame, bbox, (0, 0, 255), 2)
    cv2.imshow('Tracking APIs', frame)
    if cv2.waitKey(20) == 27: break # ESC 종료

cv2.destroyAllWindows()
```

객체 추적 결과(예시)

Q & A

교육부 Ministry of Education

NRF 한국연구재단 National Research Foundation of Korea

thank you

본 과제(결과물)는 교육부와 한국연구재단의 재원으로 지원을 받아 수행된
디지털신기술인재양성 혁신공유대학사업의 연구결과입니다.

Convergence and Open Sharing System COSS 인공지능 전남대학교 CHONNAM NATIONAL UNIVERSITY 성균관대학교 SUNGKYUNKWAN UNIVERSITY KNU 경북대학교 KYUNGPOOK NATIONAL UNIVERSITY 전주대학교 JEONJU UNIVERSITY 서울시립대학교 UNIVERSITY OF SEOUL 서울과학기술대학교 SEOUL NATIONAL UNIVERSITY OF SCIENCE & TECHNOLOGY 영진전문대학교 YEUNGJIN UNIVERSITY