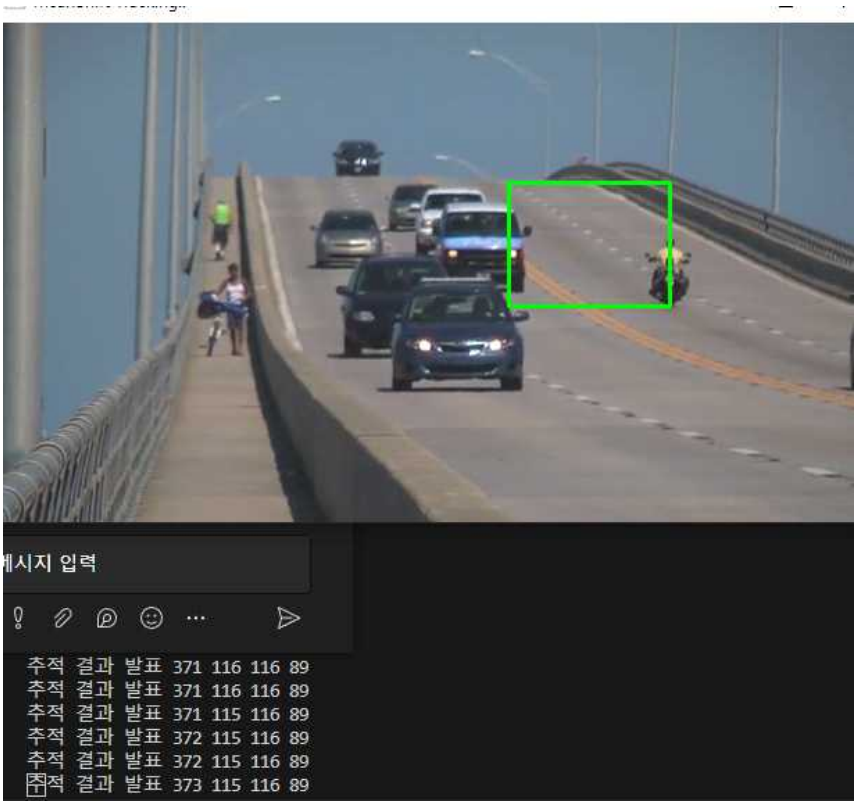
객체 추적 실습 코드 리뷰
전진희

실습01. 평균 이동 객체 추적

```
import cv2
# 평균 이동 추척을 위한 초기 사각형 설정
track_window =None # 객체의 위치 정보를 담을 변수로 초기값은 None
roi_hist =None # 히스토그램을 저장할 변수로 초기값 설정 (roi: region of interest, 관심영역)
trem_crit =(cv2.TERM_CRITERIA_EPS |cv2.TERM_CRITERIA_COUNT, 10, 1)
 #trem_crit은 알고리즘의 종료 기준을 정의하는 변수(최소 변경량|최대 반복량|수렴 조건 판단 최소횟수(1))조건을 사용

# 비디오 파일 열기
cap =cv2.VideoCapture('./data/slow_traffic_small.mp4')
# 첫 번째 프레임에서 추적할 객체 선택
ret, frame =cap.read()
x,y,w,h =cv2.selectROI('selectROI', frame, False, False) # 매개변수: (창의 이름, 프레임, 영역선택 가능여부, 영역이 선택되었을 때 영역
을 표시할지 여부)
# selectROI() 함수는 ROI(관심있는 영역)를 선택할 수 있는 창을 생성하고, 선택한 영역의 좌표와 크기를 반환한다.
print("선택한 박스 좌표 >>", x, y, w, h)
# esc 키를 누를 때까지 추적할 객체 선택 후 ROI 프린트된다

# 선택한 객체에 대한 초기 히스토그램 계산
roi =frame[y:y+h, x:x+w] # 선택한 객체의 좌표와 크기를 이용하여 프레임에서 객체 영역을 추출한다.
hsv_roi =cv2.cvtColor(roi, cv2.COLOR_BGR2HSV) # 추출한 객체 영역을 HSV 색공간으로 변환한다.
roi_hist =cv2.calcHist([hsv_roi], [0], None, [180], [0,180]) # 변환한 HSV 색공간에서 H 채널에 대한 히스토그램을 계산한다. 매개변수
는 (입력영상, 채널번호, 마스크, 히스토그램 크기, 히스토그램 범위)
cv2.normalize(roi_hist, roi_hist, 0, 255, cv2.NORM_MINMAX) # 계산한 히스토그램을 정규화한다.매개변수는 (입력히스토그램, 출력히스
토그램, 정규화시작값, 정규화종료값, 정규화타입)
# 추적할 객체의 초기 윈도우 설정
track_window =(x,y,w,h) # 추적할 객체의 위치 정보를 저장한다.
# cv2.imshow('roi test', roi)
# esc 키 누르면 roi 영역만 잘려 나온다
cv2.waitKey(0)
while True:
    # 프레임 읽기
    ret, frame =cap.read() # cap.read() 함수는 프레임을 읽어서 반환한다. 프레임을 읽었으면 ret에 True가 저장된다.
    if not ret:
        break # 프레임을 읽지 못하면 종료
   # 추적할 객체의 히스토그램 역투명 계산
    hsv =cv2.cvtColor(frame, cv2.COLOR_BGR2HSV) # 읽어온 프레임을 HSV 색공간으로 변환한다.
    dst =cv2.calcBackProject([hsv], [0], roi_hist, [0,180], 1) # 변환한 HSV 색공간에서 H 채널에 대한 히스토그램 역투영을 계산한다.
매개변수는 (입력영상, 채널번호, 히스토그램, 히스토그램 범위, 스케일)
   # 히스토그램 역투명은 프레임간 색상 등 유사도를 히스토그램을 이용하여 비교하여 이미지에서 특정 객체를 찾고자 하는 것(안써도 되지만 쓰면 더 정확도
올라감)



    # 평균 이동 알고리즘을 통해 객체 위치 추정
    _, track_window =cv2.meanShift(dst, track_window, trem_crit) # 평균 이동 알고리즘을 이용하여 객체의 위치를 추정한다. 매개
변수는 (입력영상, 추정위치, 알고리즘 종료 기준)
   # 추적 결과를 사각형으로 표시
    x,y,w,h =track_window # 추정한 객체의 위치 정보를 가져오기
```

```
    print("추적 결과 발표", x, y, w, h)
    cv2.rectangle(frame, (x,y), (x+w, y+h), (0,255,0), 2)
    # 프레임 출력
    cv2.imshow("MeanShift Tracking..", frame)
    # q 키를 누르면 종료
    if cv2.waitKey(1) &0xFF ==ord('q'):
        exit()
# 자원 해제
cap.release()
cv2.destroyAllWindows()
"""
계산된 역투명 히스토그램은 추적하려는 객체의 위치를 보다 정확하게 표시합니다. 평균 이동 알고리즘 등의 방법을 사용하여 이 역투명 히스토그램을 이용해
객체의 위치를 추정하고 업데이트하는 것이 일반적인 객체 추적 방법입니다.
"""
```



실습02. 칼만 필터 이용한 객체 추적

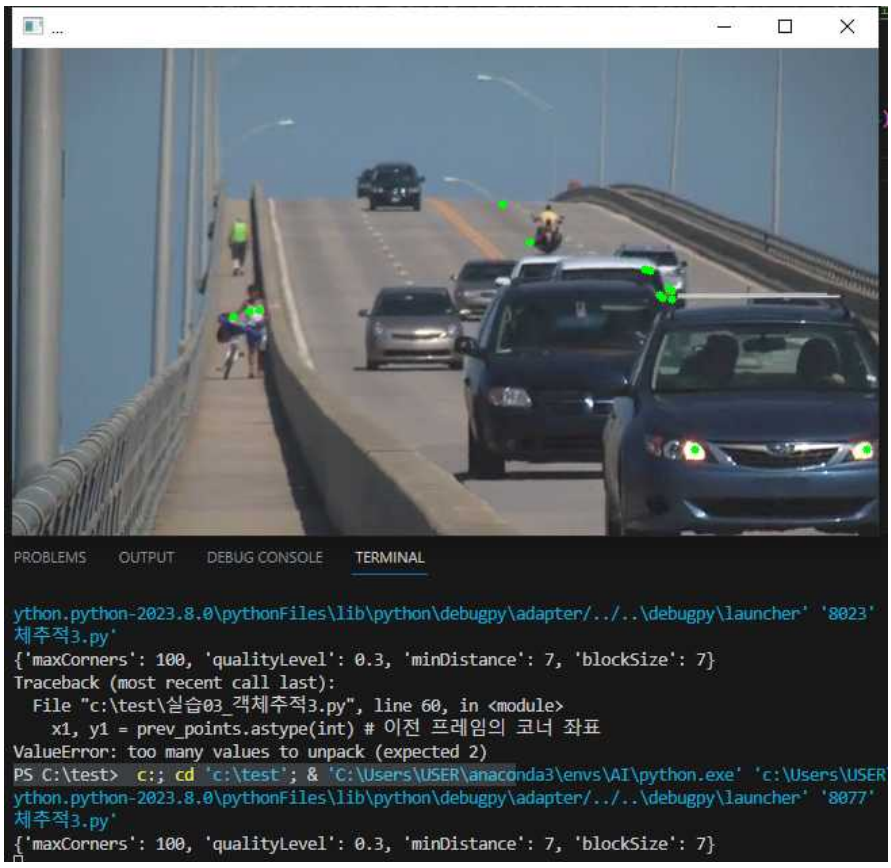
```
"""
칼만 필터 추적
1. 칼만 필터는 상태 추정에 사용되는 필터링 기술로 객체 추적에도 적용
2. 초기 예측과 실제 관측을 기반으로 객체의 위치와 속도 추정
3. 예측과 관측 간의 오차를 고려하여 정확한 위치를 업데이트
4. 이를 통해 움직이는 객체의 경로를 추적하고 예측을 통해 불확실성을 줄인다
"""
import cv2
import numpy as np
# 칼만 필터 초기화
kalman =cv2.KalmanFilter(4, 2) # 4: 상태 백터 크기 , 2: 측정 백터 크기,
kalman.measurementMatrix =np.array([[1, 0, 0, 0],
[0, 1, 0, 0]], np.float32) # 측정 행렬
kalman.transitionMatrix =np.array([[1, 0, 1, 0],
[0, 1, 0, 1],
[0, 0, 1, 0],
[0, 0, 0, 1]], np.float32) # 전이 행렬
kalman.processNoiseCov =np.array([[1, 0, 0, 0],
[0, 1, 0, 0],
[0, 0, 1, 0],
[0, 0, 0, 1]], np.float32) *0.03 # 공분산 행렬
"""
측정 벡터는 실제로 측정되는 관측값
전이 행렬은 현재 상태 정보를 가지고 다음 상태 벡터로 예측하기 위해 사용
전이 행렬은 시스템의 동역학에 대한 정보를 담고 있으며, 칼만 필터는 이를 이용하여 현재 상태 벡터를 예측
하고 측정 정보와 결합하여 추정하는 과정을 수행합니다. 올바른 전이 행렬을 설정하여 시스템의 동역학을
정확하게 모델링해야 정확한 예측과 추정이 가능합니다
공분산 행렬을 설정합니다. 이 행렬은 시스템의 불확실성을 모델링합니다.
예제에서는 모든 상태 변수에 대한 공분산을 동일하게 설정하고,
* 0.03을 통해 불확실성의 크기를 조정
"""
# 칼만 필터 추적 실습
# 동영상 불러오기
cap =cv2.VideoCapture('./data/slow_traffic_small.mp4')
# 첫 프레임에서 추적할 객체 선택
ret, frame =cap.read()
print(ret, frame) # 경로가 잘못된 경우 -> false, None / 정상적인 경우 -> true, frame /[..,..,..]
bbox_info =cv2.selectROI("Select Object", frame, False, False) # ROI 선택
print("box info >> ", bbox_info) # box info >>  (287, 186, 116, 82)
# 객체 주적을 위한 초기 추정 위치 설정
kalman.statePre =np.array([[bbox_info[0]],
[bbox_info[1]],
[0],
[0]], np.float32) # 초기 예측 상태
"""
-bbox_info[0]: 객체의 초기 x 좌표입니다.
-bbox_info[1]: 객체의 초기 y 좌표입니다.
-0: 객체의 초기 x 속도입니다. 초기에는 객체의 속도를 알 수 없으므로 0으로 설정합니다.
-0: 객체의 초기 y 속도입니다. 초기에는 객체의 속도를 알 수 없으므로 0으로 설정합니다.


-"객체의 속도"라는 표현은 일반적으로 동영상에서 객체의 움직임 속도를 의미합니다. Kalman 필터를 사용하여 객체의 위치와 속도를 추정하는 경우, 이는 객
체가 프레임 간에 이동하는 속도를 의미합니다.
-일반적으로 속도는 "픽셀 단위/프레임" 또는 "미터 단위/초"와 같은 형태로 표현됩니다.
-Kalman 필터는 초기 상태 벡터를 사용하여 초기 추정을 수행하고, 이후 추정 단계에서는 이전 상태와 관측 데이터를 기반으로 다음 상태를 예측합니다. 초기
상태 벡터는 추정 과정의 시작점을 정의하는데 사용
```

```
"""
# 칼만 필터 추적 실습
while True:
    # 프레임 읽기
    ret, frame =cap.read()
    if not ret:
        print("프레임 읽기 실패")
        break
    # 칼만 필터를 사용하여 객체 위치 추정
    kalman.correct( # 추정 단계, 측정값을 사용하여 추정값을
      # 객체의 바운딩 박스 중심의 x와 y좌표로 구성된다.
        np.array( # 측정 벡터
[
[np.float32(bbox_info[0] +bbox_info[2] /2)], # x 좌표
[np.float32(bbox_info[1] +bbox_info[3] /2)], # y 좌표
]
)
)
    kalman.predict() # 추정값을 사용하여 다음 추정값을 예측
    # 칼만 필터로 추정된 객체 위치
    predicted_bbox =tuple(map(int, kalman.statePost[:2,0])) # 추정된 객체 중심점 좌표
    # 추정된 객체 위치를 사각형 표시
    cv2.rectangle(
                frame,
(predicted_bbox[0] -bbox_info[2] //2, predicted_bbox[1] -bbox_info[3] //2), # 중심점 좌표라 이렇게 해줘야함
(predicted_bbox[0] +bbox_info[2] //2, predicted_bbox[1] +bbox_info[3] //2),
(0, 255, 0),
                  2,
)
    cv2.imshow("kalman Traking", frame)
    if cv2.waitKey(30) &0xFF ==ord('q'):
        exit()
cap.release()
cv2.destroyAllWindows()
```

실습03. 특정점 기반 추적

```
"""
초기화 단계:
1. 첫 번째 프레임을 읽고 흑백 이미지로 변환합니다.
2. Shi-Tomasi 코너 검출기 또는 다른 특징점 검출 방법을 사용하여 초기 추적 지점을 선택합니다.
3. 추적할 특징점의 초기 위치를 저장합니다.
"""
import cv2
# 동영상 파일 읽기
cap =cv2.VideoCapture('./data/slow_traffic_small.mp4')
# 코너 검출기 파라미터 설정
feature_params =dict(maxCorners=100, # 검출할 코너 개수
                     qualityLevel=0.3, # 코너로 간주할 최소한의 품질 수준
                     minDistance=7, # 코너 사이의 최소 거리
                     blockSize=7 # 코너 검출을 위한 블록 크기
)
print(feature_params) # 초기값을 딕셔너리 형태로 만든 것 확인 가능


# 광학 흐름 계산을 위한 파라미터 설정
lk_params =dict(winSize=(15, 15), # 검색 윈도우 크기
                maxLevel=2, # 이미지 피라미드 레벨 수 (0이면 피라미드 사용 안 함)
                criteria=(cv2.TERM_CRITERIA_EPS |cv2.TERM_CRITERIA_COUNT, 10, 0.03) # 계산 중지 조건
)
# 첫 프레임 읽기
ret, prev_frame =cap.read() # 첫 프레임 읽기, ret: 읽기 성공 여부, prev_frame: 읽은 프레임
prev_gray =cv2.cvtColor(prev_frame, cv2.COLOR_BGR2GRAY)
# 초기 추적 지점 선택
prev_corner =cv2.goodFeaturesToTrack(prev_gray, mask=None, **feature_params) # 코너 검출 함수(이전 프레임에서 추적
할 코너 검출), **: 딕셔너리 형태로 전달
prev_points =prev_corner.squeeze() # 이전 프레임에서 검출한 코너 좌표를 저장
"""
squeeze() 는 NumPy 배열에서 크기가 1인 차원을 제거하는 함수입니다.
"""
# 추적 결과 표시하기 위한 색상 설정
color =(0, 255, 0)


while True:
    # 다음 프레임 읽기
    ret, frame =cap.read()
    if not ret:
        print("프레임 읽기 실패")
        break
    # 현재 프레임 변환 -> 그레이 스케일 변환(for 광학 계산)
    gray =cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # 광학 흐름 계산
    next_points, status, _=cv2.calcOpticalFlowPyrLK( # 광학 흐름 계산 함수로 이전 프레임과 현재 프레임을 사용하여 광학 흐름 계산
        prev_gray, gray, prev_points, None, **lk_params
) # 이전 프레임과 현재 프레임을 사용하여 광학 흐름 계산, 매개변수는 (이전 프레임, 현재 프레임, 이전 프레임에서 추적할 코너 좌표, 검출된 코너의 상태,
```

```
광학 흐름 계산을 위한 파라미터)

    # 추적 결과를 표시
    for i, (prev_point, next_point) in enumerate(zip(prev_points, next_points)): # zip() 함수는 동일한 개수로 이루어진 자료형을 묶어주는 역할을 하는 함수
        x1, y1 =prev_point.astype(int) # 이전 프레임의 코너 좌표
        x2, y2 =next_point.astype(int) # 현재 프레임의 코너 좌표
        cv2.circle(frame, (x2, y2), 3, color, -1) # 추적 점 그리기
    # 프레임 출력
    cv2.imshow('...', frame)
    # 다음 프레임을 위해 변수 업데이트
    prev_gray =gray.copy() # 현재 프레임을 이전 프레임으로 저장
    prev_points =next_points # 현재 프레임의 코너 좌표를 이전 프레임의 코너 좌표로 저장

    if cv2.waitKey(30) &0xFF ==ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

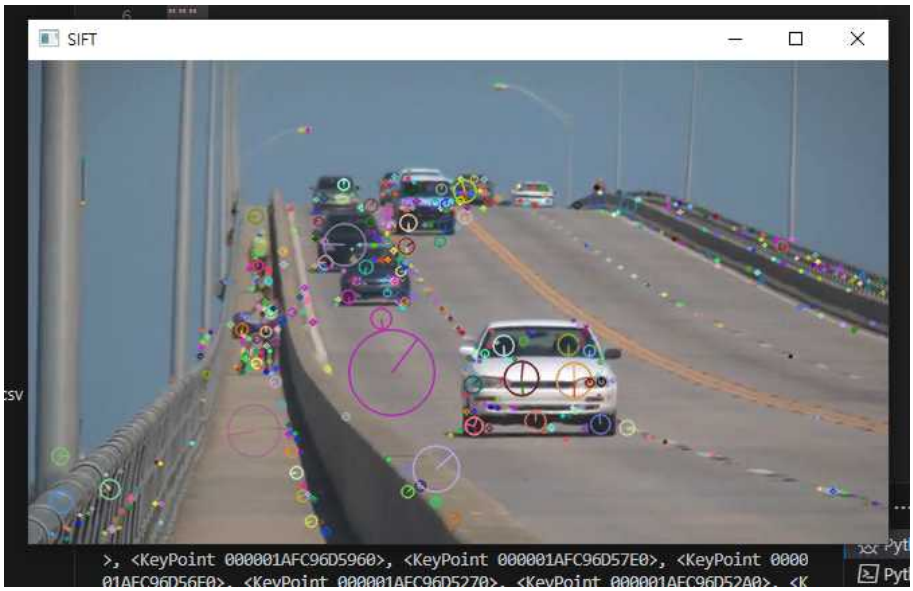


실습04. zip() 함수 알아보기

```
# zip 간단하게 보고 넘어가기
a =['a', 'b', 'c']
b =[1, 2, 3]
c =['#', '$', '%']


result =zip(a, b, c)
#for i in result:
#    print(i)
#for i in zip(a, b, c):
#    print(i)
#
"""
result:
('a', 1, '#')
('b', 2, '$')
('c', 3, '%')
"""
list_temp =list(zip(a, b, c))
print(list_temp)
"""
result:
[('a', 1, '#'), ('b', 2, '$'), ('c', 3, '%')]
"""
```

실습05. SIFT 객체 추적

```
"""
SIFT, SURF, ORB 등 특징점 검출 알고리즘
SIFT:
1. 크기 및 회전에 불변하는 특징점 검출
"""
import cv2
# 동영상 파일 열기
cap =cv2.VideoCapture('./data/slow_traffic_small.mp4')
# SIFT 객체 생성
sift =cv2.SIFT_create()
while True:
    # 프레임 읽기
    ret, frame =cap.read()
    if not ret:
        break
    #  그레이 스케일로 변환
    gray =cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # 특징점 검출
    keypoints, descriptors =sift.detectAndCompute(gray, None)    #keypoints는 이미지에서 검출된 특징점들의 위치와 특징을 담고 있는 변수이고, descriptor는 특징점들의 주변 정보를 기반으로 생성된 특징 기술자
    # 특징점 그리기
    frame =cv2.drawKeypoints(
        frame, keypoints, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS
) #cv2.drawKeypoints() 함수는 이미지와 특징점들을 인자로 받아들여 특징점을 그림,이때 flags 인자를 통해 특징점의 크기와 방향을 표시
    # 프레임 출력
    cv2.imshow('SIFT', frame)
    # q를 누르면 종료
    if cv2.waitKey(30) &0xFF ==ord('q'):
        break
    # 자원 해제
cap.release()
cv2.destroyAllWindows()
```



실습06. SIFT 객체 추적-특징점 제한

```
import cv2
cap =cv2.VideoCapture('./data/slow_traffic_small.mp4')
# SIFT 객체 생성
sift =cv2.SIFT_create(contrastThreshold=0.02) # contrastThreshold는 특징점 검출을 위한 임계값, 픽셀간 대비의 밝기 차이를 이용함
# 특징점 개수 제한 설정
max_keypoints =100 # 원하는 최대 특징점 개수
while True:
    # 프레임 읽기
    ret, frame =cap.read()
    if not ret:
        break
    # 그레이 스케일로 변환
    gray =cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # 특징점 검출
    keypoints, descriptors =sift.detectAndCompute(gray, None)
    print(keypoints,descriptors)
    # 특징점 개수 제한
    if len(keypoints) >max_keypoints:
        keypoints =sorted(keypoints, key=lambda x: -x.response)[:max_keypoints]

    # 특징점 그리기
                        frame              =cv2.drawKeypoints(frame,              keypoints,              None,
flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
    # 프레임 출력
    cv2.imshow('SIFT', frame)
    if cv2.waitKey(30) &0xFF ==ord('q'):
        break
# 자원 해제
cap.release()
cv2.destroyAllWindows()
```

실습07. ORB 객체 추적

```
import cv2
# 동영상 파일 열기
cap =cv2.VideoCapture('./data/slow_traffic_small.mp4')
# ORB 객체 생성
orb =cv2.ORB_create()
while True:
    # 프레임 읽기
    ret, frame =cap.read()
    if not ret:
        break
    #  그레이 스케일로 변환
    gray =cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # 특징점 검출
    keypoints =orb.detect(gray, None)
    # 특징점 그리기
    frame =cv2.drawKeypoints(
        frame, keypoints, None, (0,150,220) ,flags=0) # flags=0 : 특징점의 크기와 방향을 표시하지 X
    # 출력
    cv2.imshow('ORB', frame)
    if cv2.waitKey(30) &0xFF ==ord('q'):
        exit()
    # 자원 해제
cap.release()
cv2.destroyAllWindows()
```

실습08. ORB 객체 추적2

```
import cv2
import numpy as np
# 동영상 파일 읽기
cap =cv2.VideoCapture('./data/slow_traffic_small.mp4')
# ORB 객체 생성
orb =cv2.ORB_create()
# 특징점 최소 크기 설정,
min_keypoint_size =10
# 중복 특징점 저거 기준 거리
duplicate_threshold =10
while True:
    # 프레임 읽기
    ret, frame =cap.read()
    if not ret:
        print('프레임 읽기 실패')
        break
    # 그레이 스케일로 변환
    gray =cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # 특징점 검출
    keypoints =orb.detect(gray, None)
    # 특징점 크기가 일정 크기 이상인 것만 남기기
    keypoints =[kp for kp in keypoints if kp.size >min_keypoint_size] # kp.size는 특징점의 크기를 의미, 키포인트의 크기는 특징의 중요도
    """
<List comprehension>
파이썬에서 리스트를 간결하게 만드는 방법 중 하나
new_list = [expression for item in iterable if condition == True]
반복문과 조건문을 한 줄에 사용할 수 있음, 조건문을 만족하는 경우에만 expression이 실행되어 리스트에 추가됨
"""
    # 중복된 특징점 제거
    mask =np.ones(len(keypoints), dtype=bool) # len(keypoints)만큼의 크기를 가진 배열을 생성하고, 모든 원소를 True로
    for i, kp1 in enumerate(keypoints):
        if mask[i]:
            for j, kp2 in enumerate(keypoints[i +1:]): #
                if(
                    mask[i +j +1]
                    and np.linalg.norm(np.array(kp1.pt) -np.array(kp2.pt)) # pt는 키포인트의 위치를 나타낸다
                    <duplicate_threshold # 두 키포인트의 거리가 duplicate_threshold보다 작으면 중복된 키포인트로 판단
):
                    mask[i +j +1] =False # 중복된 키포인트는 False
    keypoints =[kp for i, kp in enumerate(keypoints) if mask[i]] #  True인 키포인트만 남기기


    # 특징점 그리기
    frame =cv2.drawKeypoints(
    frame, keypoints, None, (0, 200, 150), flags=0)
    # 출력
    cv2.imshow("ORB", frame)
    if cv2.waitKey(30) &0xFF ==ord("q"):
        break
"""
.pt 는 OpenCV의 키포인트(KeyPoint) 객체의 속성(Attribute) 중 하나입니다
```

```
kp1.pt와 kp2.pt는 각각 두 개의 키포인트 객체인 kp1과 kp2의 위치 좌표를 나타냅니다.
"""



# 자원 해제
cap.release()
cv2.destroyAllWindows()
```