칼만필터를 이용한 객체추적

- openCV 를 활용한 영상처리 프로젝트 -

이름: 성수호

Problem 1: 촬영된 영상을 보정(calibration) 작업 후 파라미터 값을 통해 3 차원 좌표 추출

1. 문제의 개요

본 프로그램을 간략히 설명하면 다음과 같다.

- HSV 칼라로 빨간색의 범위를 설정한다.
- 마스크를 생성하여 영상 속 빨간 공을 찾아 바운딩 박스를 그린다.
- 칼만필터를 이용해 바운딩 박스가 그려진 빨간 공을 추적한다.

2. 알고리즘

본 프로그램 작성을 위한 알고리즘을 Pseudo 코드 형태로 나타내면 다음과 같다.

2.1. 마스크 생성

- HSV 칼라를 이용해 빨간색의 범위를 지정
- 영상 속에서 칼라 범위를 이용해 빨간 공을 찾는다.

2.2. 칼만필터 적용

- 칼만 필터의 변수들을 초기화 한다.
- 시간에 따른 모델의 추정 값, 예측과 측정값을 이용한 상태 정정을 반복하여 모델의 상태 벡터를 추정한다.

3. 프로그램 구조 및 설명

3.1. 마스크 생성

```
ret, frame = cap.read()
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

lowb = np.array([0,100,100])
highb = np.array([10,255,255])

#마스크생성
mask = cv2.inRange(hsv,lowb,highb)

#배경추출
gaus = cv2.GaussianBlur(mask,(5,5),0)
image = fgbg.apply(gaus)

#바운딩박스그리기
nlables, labels, stats, centroids = \
    cv2.connectedComponentsWithStats(image)
for index, centroid in enumerate(centroids):
    if stats[index][0]==0 and stats[index][1]==0:
        continue
    if np.any(np.isnan(centroid)):
        continue

    x,y,width,height,area = stats[index]

    if area > 100:
        x,y,w,h,area = stats[index]
```

- HSV 로 컬러의 범위를 조정해 찾고자 하는 빨간색의 범위를 찾는다.

- inRange() 함수를 사용하여 영상에 색의 범위를 지정하여 영상 출력 시 빨간 공만 나오게 한다.

- connectedComponentsWithStats() 함수로 영상에서 바운딩박스 정보와 무게 중심 좌표를 가져온다.

- stats 에서 area 는 면적, 픽셀의수를 가져오는데 특정 임계 값을 넘었을 때 해당 바운딩 박스 정보를 다시 가져와 output 영상에서 출력되는 박스 이미지를 최소로 한다.

3.2. 칼만 필터 적용.

```
#칼만필터셋업
q = 1e-5 #pnc
r = 0.01 #mnc
dt = 1
KF = cv2.KalmanFilter(4,2,0)
KF.transitionMatrix = np.array([[1,0,dt,0],
                                [0,1,0,dt],
                                [0,0,1,0],
                                [0,0,0,1]],np.float32) #A
KF.measurementMatrix = np.array([[1,0,0,0],
                                 [0,1,0,0]],np.float32) #H
```

```
    #칼만필터 시작
    KF.processNoiseCov = q * np.eye(4,dtype=np.float32) #Q
    KF.measurementNoiseCov = r * np.eye(2,dtype=np.float32) #R
    KF.errorCovPost = np.eye(4,dtype=np.float32) #P0 = I

    KF.statePost = np.array([[x],[y],[0.],[0.]],dtype=np.float32)

    tracking_start = True

    if tracking_start :
        predict = KF.predict()

        #칼만정정
        z = np.array([[x],[y]],dtype=np.float32) #measurement
        estimate = KF.correct(z)
        estimate = np.int0(estimate)

        #바운딩박스
        x2,y2 = estimate[0][0],estimate[1][0]
        cv2.rectangle(frame,(x2,y2),(x2+w,y2+h),(255,0,0),2)
```

- 프로세스 잡음과 측정잡음 공분산 행렬에서 사용할 q, r 을 초기화 한다. 상태 벡터 차원 4, 측정 벡터 차원 2, 외부 컨트롤을 사용하지 않는 칼만

필터 객체 KF 를 생성

- 4x4 상태변환 행렬 A, 2x2 측정행렬 H 를 각각 초기화 한다.
- 칼만 필터의 초기값 설정하고, errorCovPost 에 4x4 로 초기화 한다. 바운딩 박스에서 반환한 좌표 x, y 를 이용하여, statePost 에 4x1 벡터로 초기화 한다.
- 객체 추적이 시작되면 predict()로 predict 에 프로세스 상태를 예측 계산 한다.
- 위 좌표 x, y 를 이용하여 측정 벡터 z 를 생성하고, correct(z)로 상태 벡터를 정정하여 estimate 에 계산한다.

4. 결론

- openCV 교재를 통해서 배운 컬러 공간 변환과 칼만 필터 물체 추적을 이용하여 특정 객체 추적을 수행했다. 칼만 필터는 적용 전보다 추적에서 개선된 모습을 보인다.

5. 개선방향

- 칼만 필터 적용 후 데이터를 기반으로 헝가리안 알고리즘을 사용하여 객체 트래킹하는 과정에서 장애물이 있어도 트래킹 정확도를 높여야 한다.

- 딥러닝을 이용하여 detection 한 객체에 칼만 필터와 헝가리안 알고리즘을 적용하면 트래킹 성능이 훨씬 좋을 것 같다.

- 유사한 객체가 겹치거나 부딪혔다 튕겨난 상황에 트래킹 에러가 많이 발생하는데, 최종목표는 동일한 색상과 같은 유사한 객체가 있어도 개별 id 가 부여돼 트래킹 에러를 없애는 방식이다.