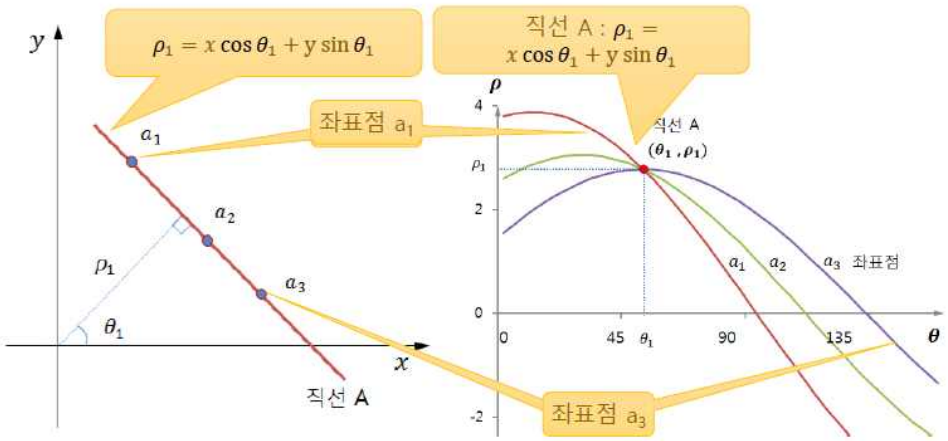
W12 영상분할 (Hough변환)

[redacted]

이론 설명 및 활용방안.

Hough 변환은 직선이나 모서리를 검출하는 알고리즘으로 공간 구조 분석에 유용하게 사용되는 알고리즘이다.

허프변환은 직교 좌표계로 표현한 영상에서 에지 점을 극좌표계로 이동하여 검출하고자 하는 파라메터 Rho와 Theta를 추출한다.



Rho : 픽셀간격으로, 선의 해상도에 영향을 미친다.
Theta : 각도 간격드로, 각도의 해상도에 영향을 미친다.
Threshold : 선을 검출하기 위해 필요한 최소한의 교차점 수다.

현재 위의 사진에서는 연속적인 그래프를 보이고 있으나 실제 PC에서는 discrete한 값들이기 때문에 어떤 해상도로 나타낼 것인지가 중요하다.

사용될 사진 설명

Hough 변환이 사용되는 예시는 자율주행 자동차의 차선감지에 와 같이 공간 구조의 분석에 사용된다. 사용한 사진은 야구장을 선택하였다.
야구장의 파울라인을 검출하는 것이 목적으로 축구의 경우 var등으로 오프사이드를 확인할 때 선을 그어 확인이 쉽다. 야구의 경우 특히 홈런타구와 피울의 비디오 판독의 경우 흐리거나 카메라로 돌려보아도 판독이 어려운 경우가 존재하며 팬들이 납득하지 못하는 경우가 많다.
축구와 같이 파울라인에 영상에 선을 그어줄 수 있다면 판정도 쉬워지고 납득할 수 있을 것이라 생각되어 해당 선을 검출하는 것을 목표로 선정하였다.

cv2.HoughLines 사용

threshold = 160

threshold = 140

threshold = 120

threshold = 100

목표했던 파울라인의 선을 검출하는 것은 threshold가 140일 때 가장 정확했다. 하지만 좌측의 파울라인과 다르게 우측은 내야의 엣지선을 직선으로 그린 것을 확인 할 수 있었다. 140 이상의 threshold에서는 파울라인 부근의 선이 검출되지 않았으며 140 이하의 값에서는 원하지 않는 선들도 검출된 것을 확인할 수 있다.

```
# Canny edge detection
edges = cv2.Canny(Y, 150,250)
```

위와 같이 canny filter의 파라메터를 수정하여 노이즈같이 보이는 엣지를 줄이고 야구장의 윤곽만을 검출 할 수 있도록 수정해 주었다.

Threshold = 140

Threshold = 150

canny_filter의 파라메터 수정전 120의 threshold값에서의 결과와 수정후 140의 값에서 결과가 같아진 것을 확인 할 수 있으며 우측의 파울라인에 대한 선을 얻지 못하였다. 이는 파울라인의 선이 완벽한 직선이 검출되는것이아닌 내야와 외야의 경계면에서 한번 끊어져 있어 그런 것으로 생각된다. 따라서 선의 해상도를 줄여보았다.

수정된 canny_filter에 Rho = 3 , Threshold = 160

수정된 canny_filter에 Rho = 2 , Threshold = 160

Rho를 높여 해상도를 낮추었을 때 선이 거의 검출되지 않던 160의threhold값에서 원하던 좌우측의 파울라인이 검출된 것을 확인 할 수 있다. Rho를 조금 낮춰서 불필요한 선을 조금 줄일 수 있었으나 다른 파라메터를 조정해 검출된 선을 제거하려 시도하면 원하는 선을 검출 할 수 없었다.

<Thres = 110>

또한 다음과 같이 원하는 직선의 엣지검출이 잘 이루어지지 않은 경우 메인 직선으로 판단되지 않는다. 또한 파울라인의 선을 검출 하기위해 Threshold를 낮춘다면 조금의 엣지만 검출되어도 모두 직선으로 반환되어 무수히 많은 선들이 검출되는 문제가 있다.

다음 실험인 HoughLinesP는 해당 문제를 어느정도 해결할 수 있도록 길이와 Gap을 지정해 줄 수 있다.

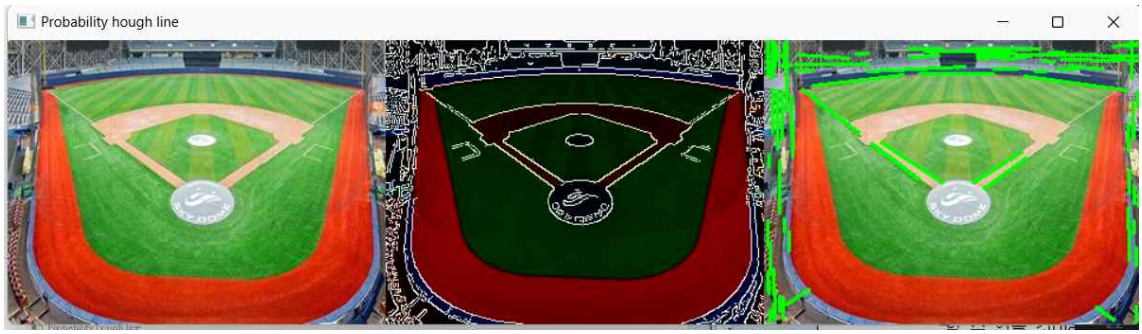
cv2.HoughLinesP 사용

HoughLines함수를 사용할 때 원하는 라인의 검출은 Canny_filter를 수정하고 Rho=2로 변경해 주었을 때 Threshold=160에서 확인 할 수 있었다.
HoughLinesP의 경우 파라메터를 통해 검출되는 Line간의 간격과 길이를 지정해 줄 수 있으므로 파울라인과 내야의 선이 검출될 때 이를 조정하여 원하는 선만을 검출할 수 있을 것이라 생각된다.

```
edges = cv2.Canny(Y, 150, 250)
edge_color = cv2.cvtColor(cv2.merge((edges, Cr, Cb)), cv2.COLOR_YCrCb2BGR)


lines = cv2.HoughLinesP(edges, 2, np.pi/180, Thres, None, 10, 5)
```

<HoughLines에서 가장 잘나왔을 때와 동일한 파라메터로 수정>

<Threshold=160, minLineLength=10, maxLineGap=5>



<Threshold=130, minLineLength=10, maxLineGap=5>

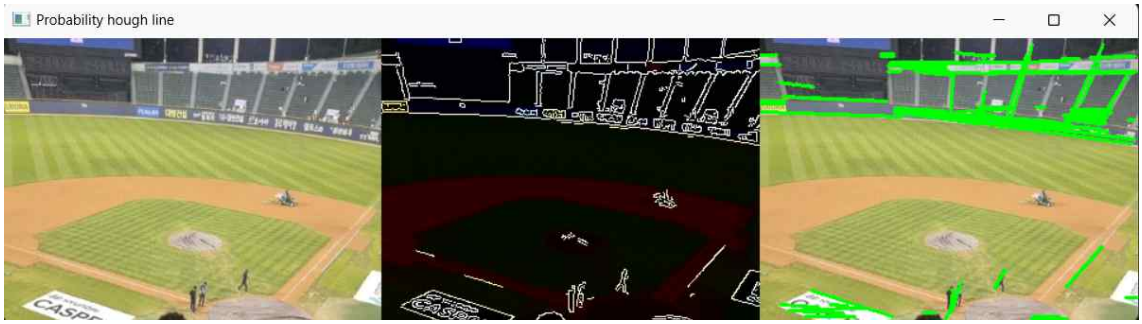
파울라인과 내야의 직선이 모두 검출 되기 때문에 maxLineGap 파라메터를 늘려 두 직선중 하나의 선만이 나오도록 수정을 시도해 보았다.

<Threshold=140, minLineLength=10, maxLineGap=40>

<Threshold=140, minLineLength=10, maxLineGap=20>

예상과는 다르게 maxLineGap 파라메터의 크기를 매우 크게 하여도 분리가 되지 않았으며 오히려 적당한 값에서 분리가 되었다.

이는 해당 파라메터로 지정된 값 이상 떨어질 경우 독립적인 선으로 보는데 각도로 인해 부분적으로는 해당 파라메터가 지정한 Gap을 만족하기 때문에 독립적인 선으로 판단된 것이 아닐까 생각된다.



<Threshold=35, minLineLength=5, maxLineGap=10>

이전 실험에서 결과가 안좋았던 사진역시 원하던 파울라인을 검출할 수 는 있었지만 원본 이미지의 엣지검출이 부족한 문제로 Threshold를 35 까지 낮추어야 했으며 이로인해 불필요한 직선도 검출된 것을 확인할 수 있다.

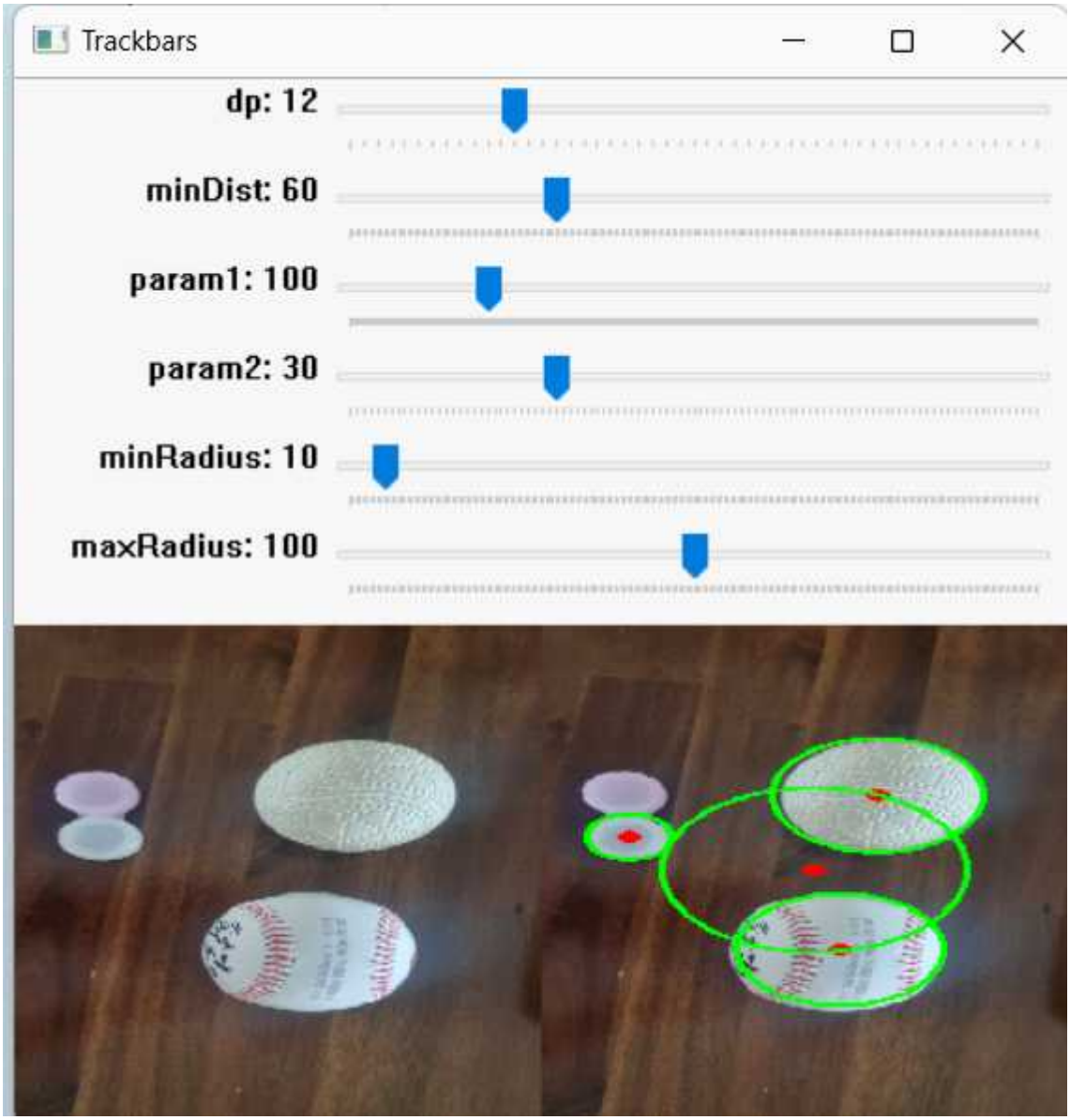
cv2.HoughCircles 사용

파라메터가 많아 각 파라메터의 적절한 조합을 찾기위해 각 파라메터를 trackbar로 만들었다.

```
    # 윈도우 생성 및 초기 이미지 표시
    cv2.namedWindow('Trackbars', cv2.WINDOW_NORMAL)
    cv2.resizeWindow('Trackbars', new_width * 2, new_height + 50)  # 윈도우 크기 조

    # 트랙바 생성
    cv2.createTrackbar('dp', 'Trackbars', 10, 50, on_trackbar)  # dp 값을 1.0 ~ 5
    cv2.setTrackbarPos('dp', 'Trackbars', 12)  # 초기값 설정 (1.2)

    cv2.createTrackbar('minDist', 'Trackbars', 1, 200, on_trackbar)
    cv2.setTrackbarPos('minDist', 'Trackbars', 80)

    cv2.createTrackbar('param1', 'Trackbars', 0, 500, on_trackbar)
    cv2.setTrackbarPos('param1', 'Trackbars', 100)

    cv2.createTrackbar('param2', 'Trackbars', 1, 100, on_trackbar)
    cv2.setTrackbarPos('param2', 'Trackbars', 30)

    cv2.createTrackbar('minRadius', 'Trackbars', 0, 200, on_trackbar)
    cv2.setTrackbarPos('minRadius', 'Trackbars', 10)

    cv2.createTrackbar('maxRadius', 'Trackbars', 0, 200, on_trackbar)
    cv2.setTrackbarPos('maxRadius', 'Trackbars', 100)
```

```
on_trackbar(pos):
    dp = cv2.getTrackbarPos('dp', 'Trackbars') / 10.0
```

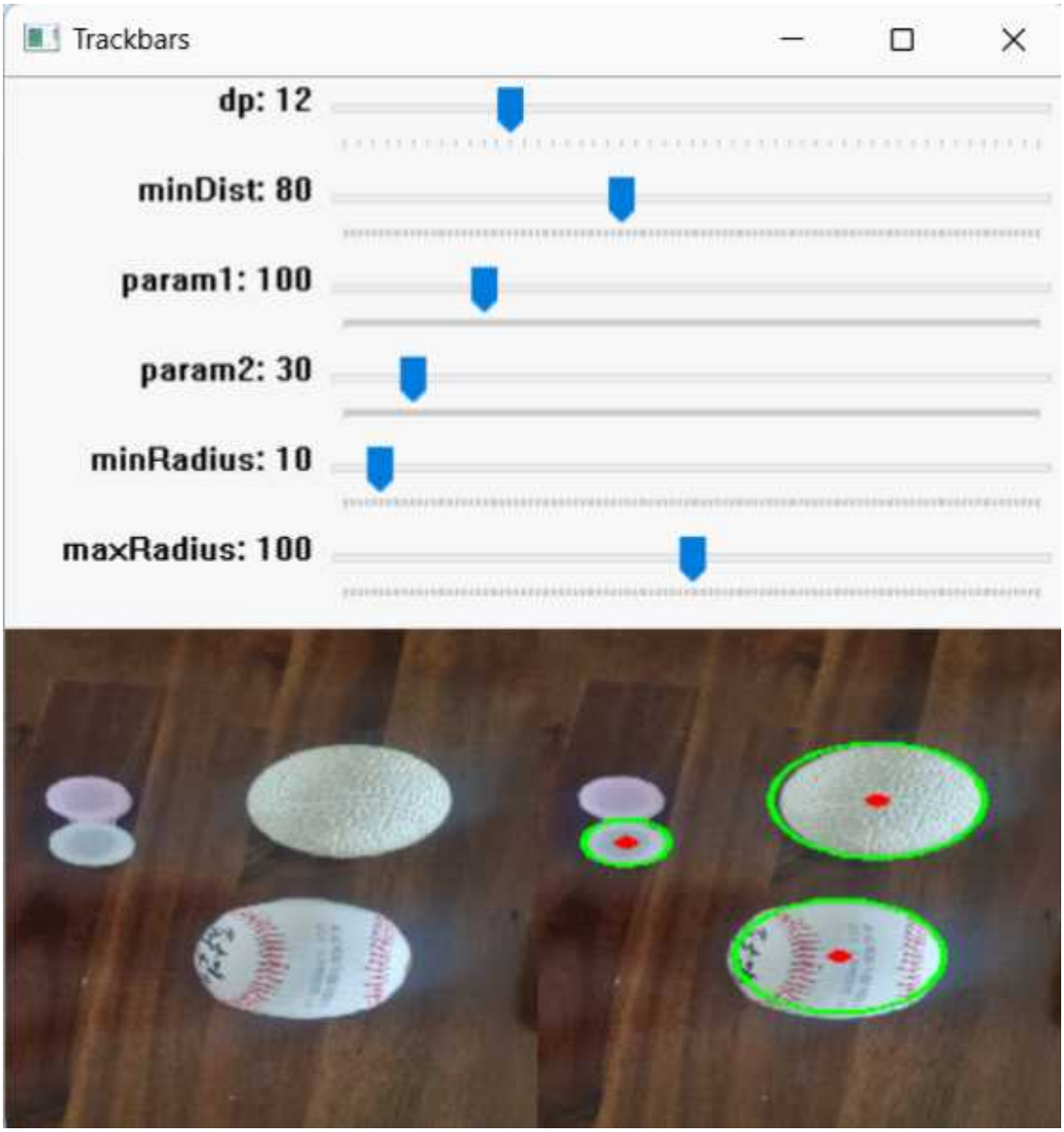
dp의 경우 10 으로 나누어 주어 계산되도록 하였다.



<dp = 1.2 , minDisr 60, param1 = 100, param2 = 30, minRadius = 10 MaxRadius = 100>

minDist의 경우 원들 중심간의 최소거리다.
가운데 가장 큰원은 실제론 잘못된 원이므로 해당 파라메터를 조정하면 없어질 것이다.

<dp = 1.2 , minDisr 60, param1 = 100, param2 = 30, minRadius = 10 MaxRadius = 100>

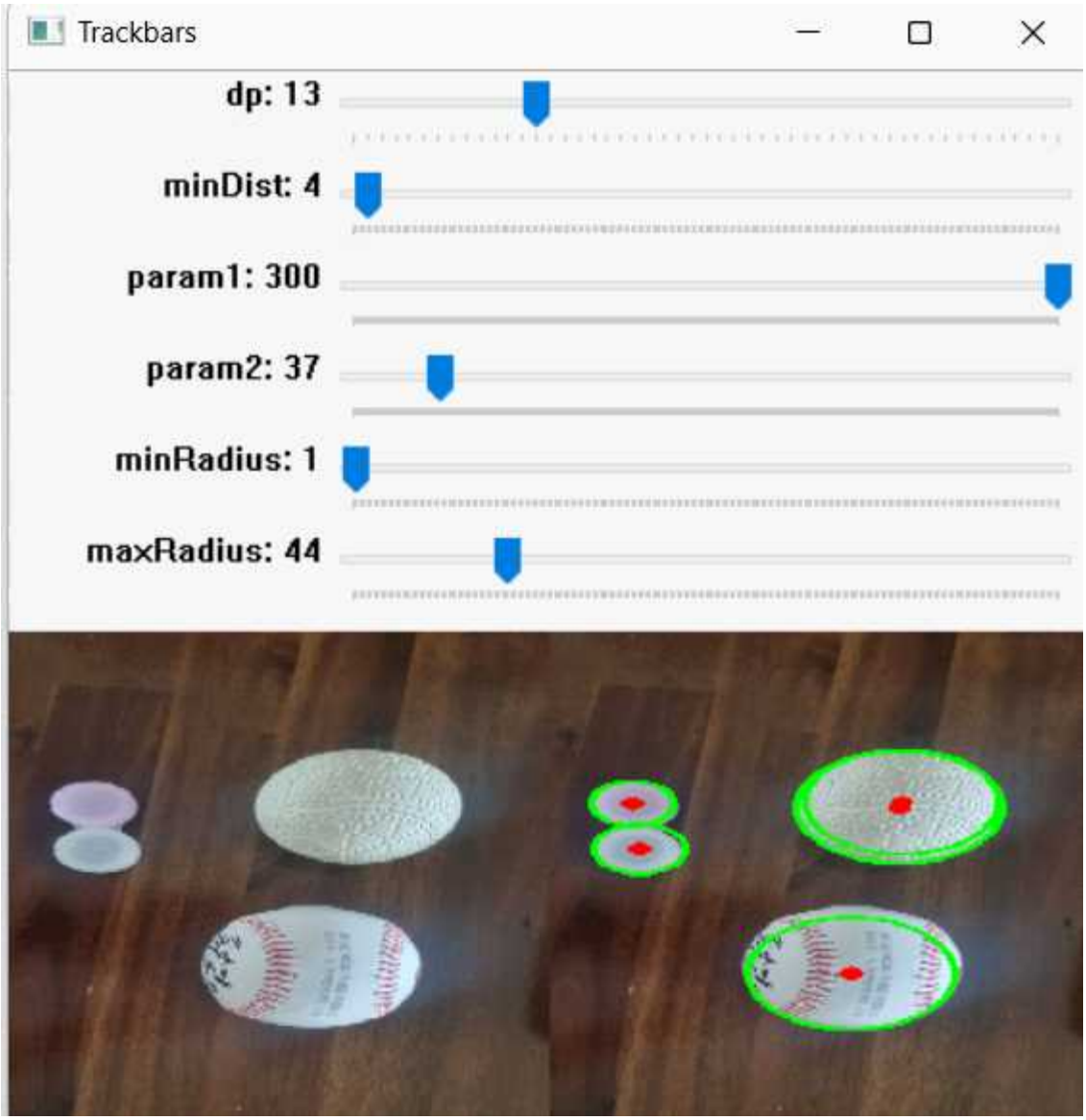


<dp = 1.2 , minDist 80, param1 = 100, param2 = 30, minRadius = 10 MaxRadius = 100>

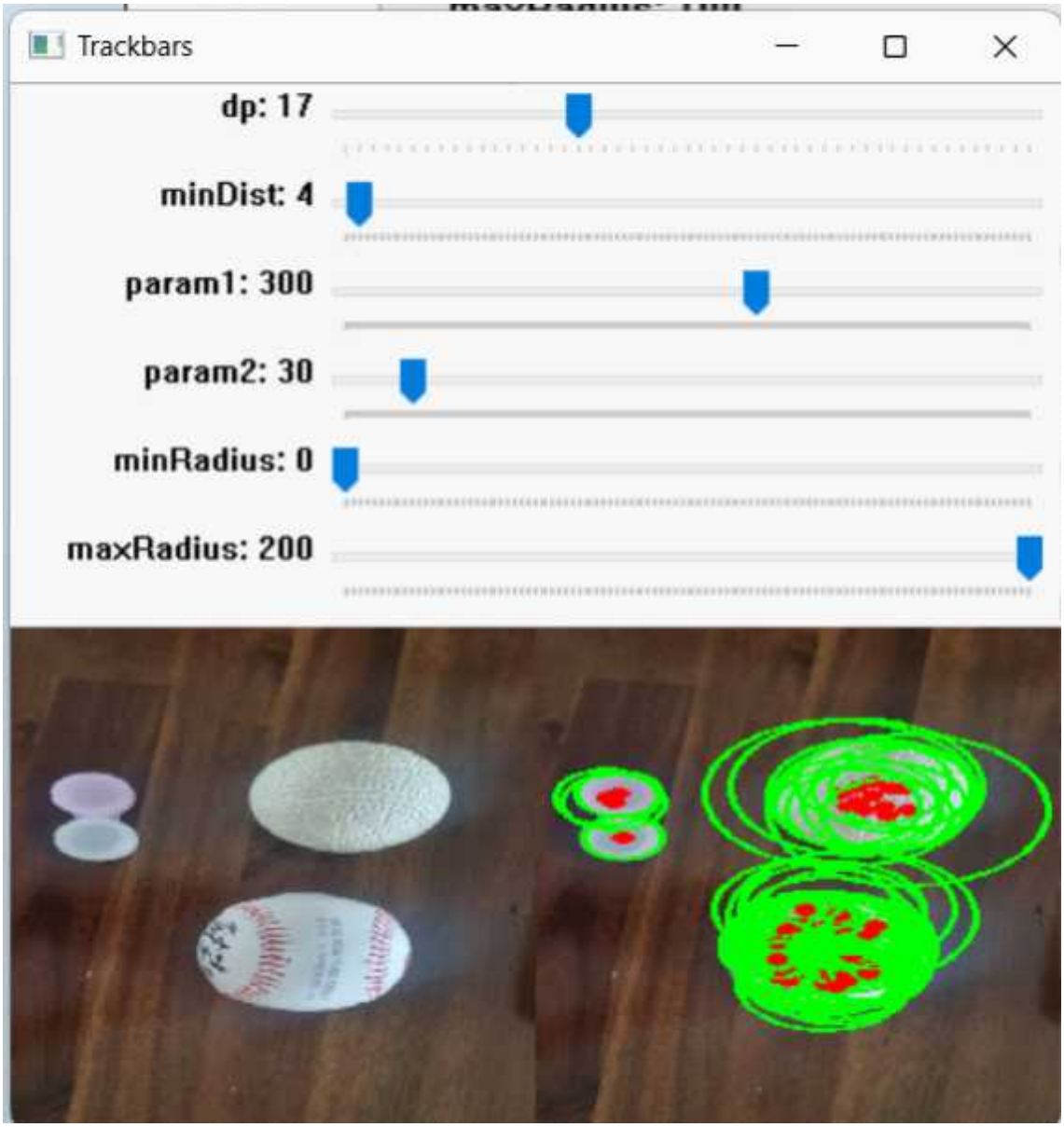
minDist를 80으로 늘리자 가운데에 있던 가장큰 원이 사라졌다.
예상대로 원의 중심간의 최소거리에 영향을 받아 잘못된 원을 제거할 수 있었다.
하지만 렌즈통의 원중 하나만 검출되는 이유 역시 minDist 값이 크기 때문이다.
따라서 minDist 값을 낮추고 다른 파라메터를 수정하였다.

<dp = 1.2 , minDist 4, param1 = 300, param2 = 35, minRadius = 0, MaxRadius = 44>

나머지 원들이 잘 검출 되었으나 야구공의 경우 살짝 작은 원이 검출되었다.



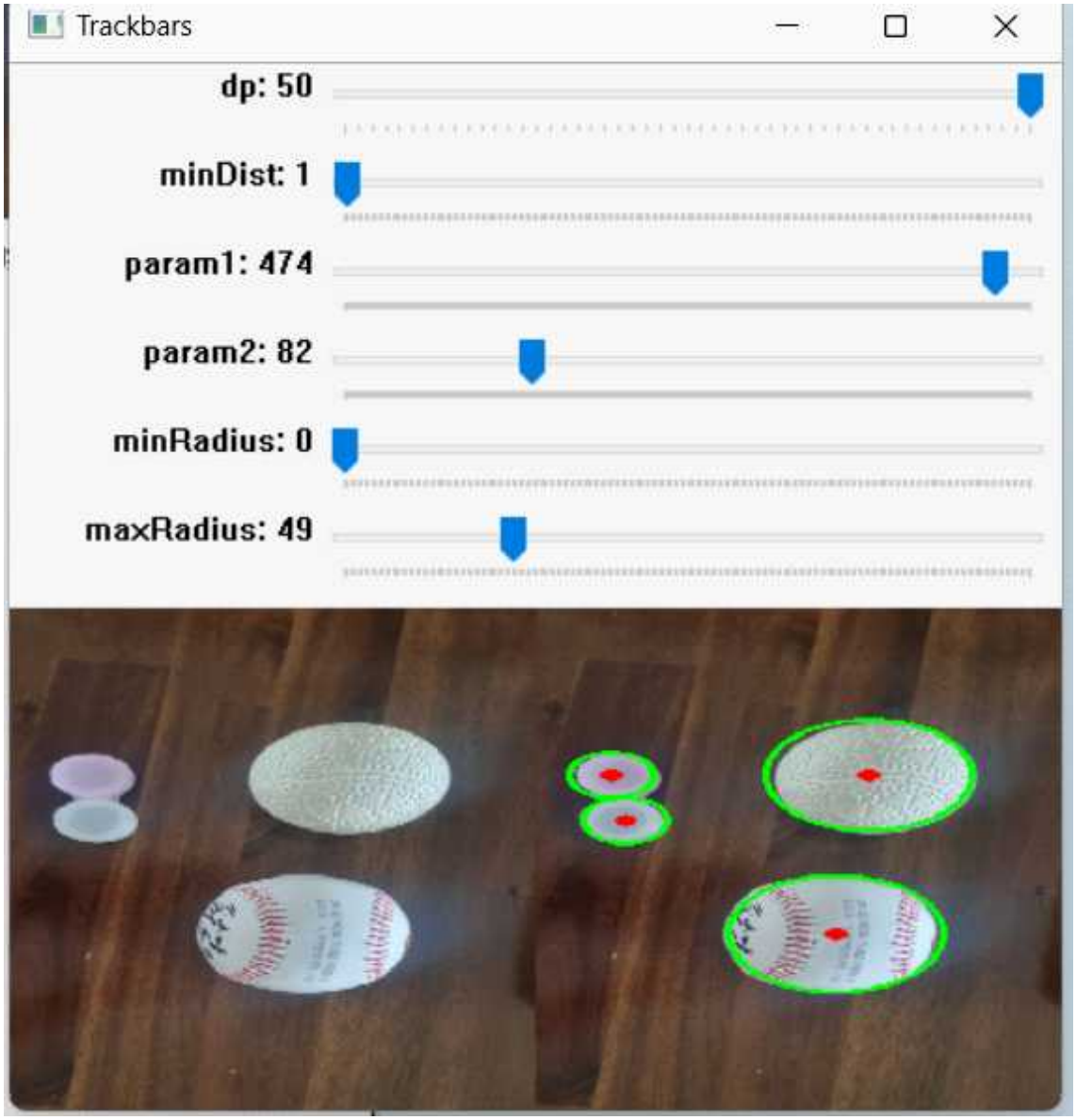
<dp = 1.3 , minDist 4, param1 = 300, param2 = 37, minRadius = 1 MaxRadius = 44>



<dp = 1.7 , minDist 4, param1 = 300, param2 = 30, minRadius = 0 MaxRadius = 200>

가장 작은원인 렌즈통의 두 개의 원을 검출하기위해 minDist와 minRadius의 값을 낮췄다. 야구공의 실밥과 배팅볼의 홈으로 인해 많은 동심원들이 검출되는 것을 볼 수 있다.

비슷한 조건에서 dp값을 올리자 동심원들이 늘어나는 결과를 확인 할 수 있었다.



<dp = 5.0 , minDist 1, param1 = 474, param2 = 82, minRadius = 0 MaxRadius = 49>

여러번의 파라메터 튜닝 끝에 가장 좋은 결과를 얻는 파라메터 조합은 이와 같았다. 각 파라메터의 값이 다른 파라메터 값이 무엇이냐에 따라 값이 1만 변경되어도 크게 변화할때도 있고 10이상의 변화가 있어도 변화가 없는 경우가 존재하였다.

따라서 이와같이 여러 가지의 파라메터에 영향을 받는 경우 Trackbar를 통해 각각의 값을 변경해줄때의 변화를 바로바로 보며 파라메터를 튜닝하는 방식이 효율적이라고 생각된다.