

크레용 포장 불량 검사 알고리즘



R1/R2 접근 방법 또는 아이디어

공통 아이디어

1. 이미지 사이즈의 통일
2. 이미지의 회전을 통해 각각의 크레용을 같은 방향으로 두고 검출

R1

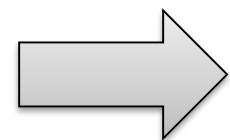
생각해본 접근 방법들

1. 뒤집혔을 때와 정상일 때 픽셀 값을 비교
2. 특정 부분 crop을 통해 불량 검출 후
원이 검출되지 않을 경우 크레용이 뒤집혔다고 판단.

R2

생각해본 접근 방법들

1. 크레용이 회전된 경우 생기는 틈의 픽셀 값으로 판단.
2. 특정 부분 crop을 통해 불량 검출 후
원이 검출된 경우 크레용이 회전되었다 판단.



R1과 R2 모두 두번째 방법으로 알고리즘 작성

R1/R2 접근 방법 또는 아이디어

R1, R2 따로 검출하지 않고 같이 검출한 이유

R1과 R2를 따로 검출하는 경우

R1을 검출하기 위한 전처리 > 검출  알고리즘 ①

R2를 검출하기 위한 전처리 > 검출  알고리즘 ②

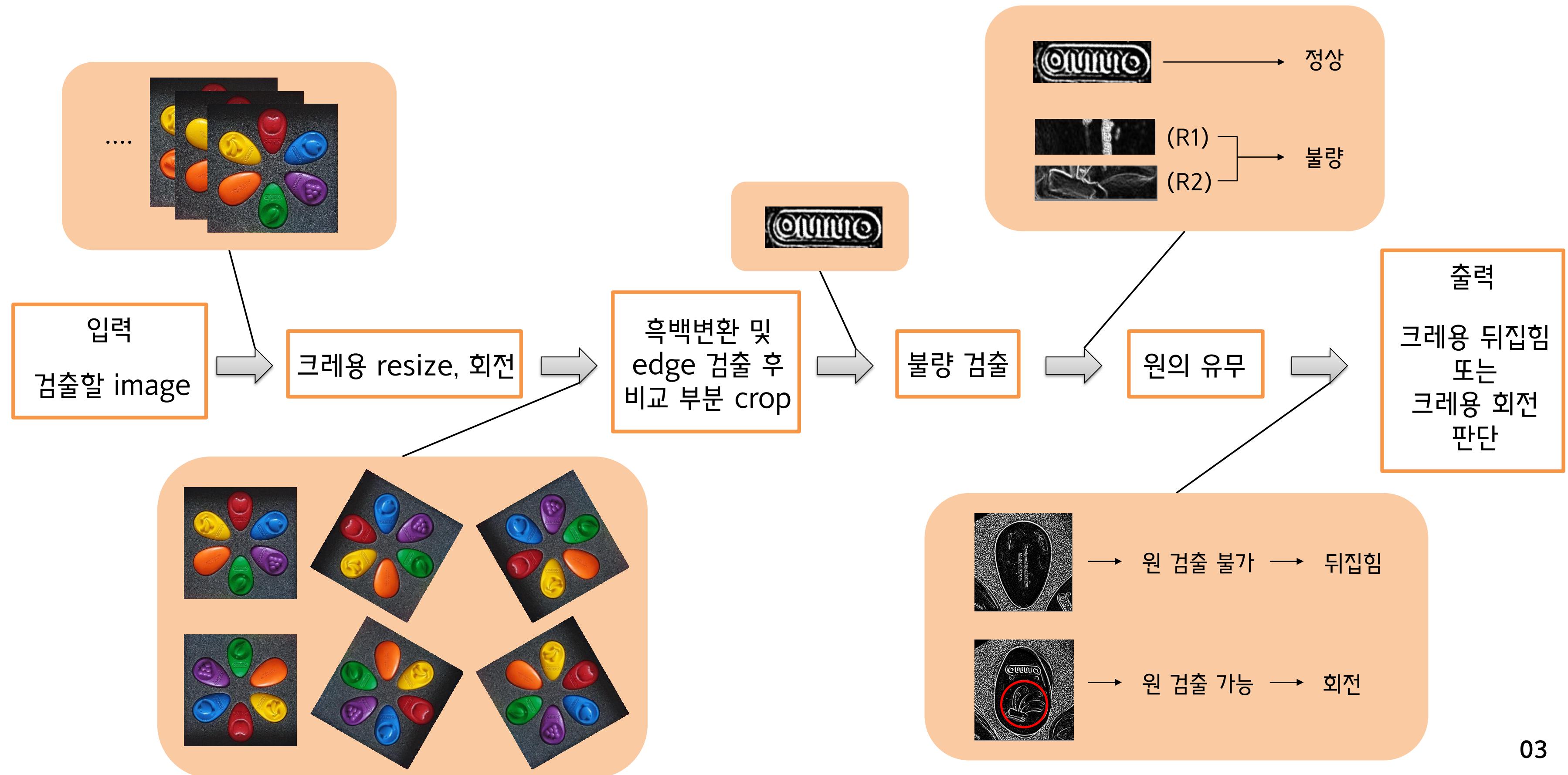
R1과 R2를 함께 검출하는 경우

불량 검출(R1, R2) > 두 경우의 차이점으로 R1, R2 구분  알고리즘 ①

따라서

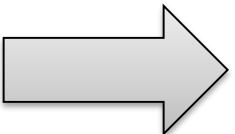
함께 검출하는 경우 알고리즘이 보다 간단해지므로 효율적인 검출작업을 수행할 수 있으며,
작업량이 많을 때 검출장비의 부하를 줄일 수 있다.

R1/R2 전체 흐름도



R1/R2 단계 별 알고리즘 설명

1) 검출할 이미지 입력

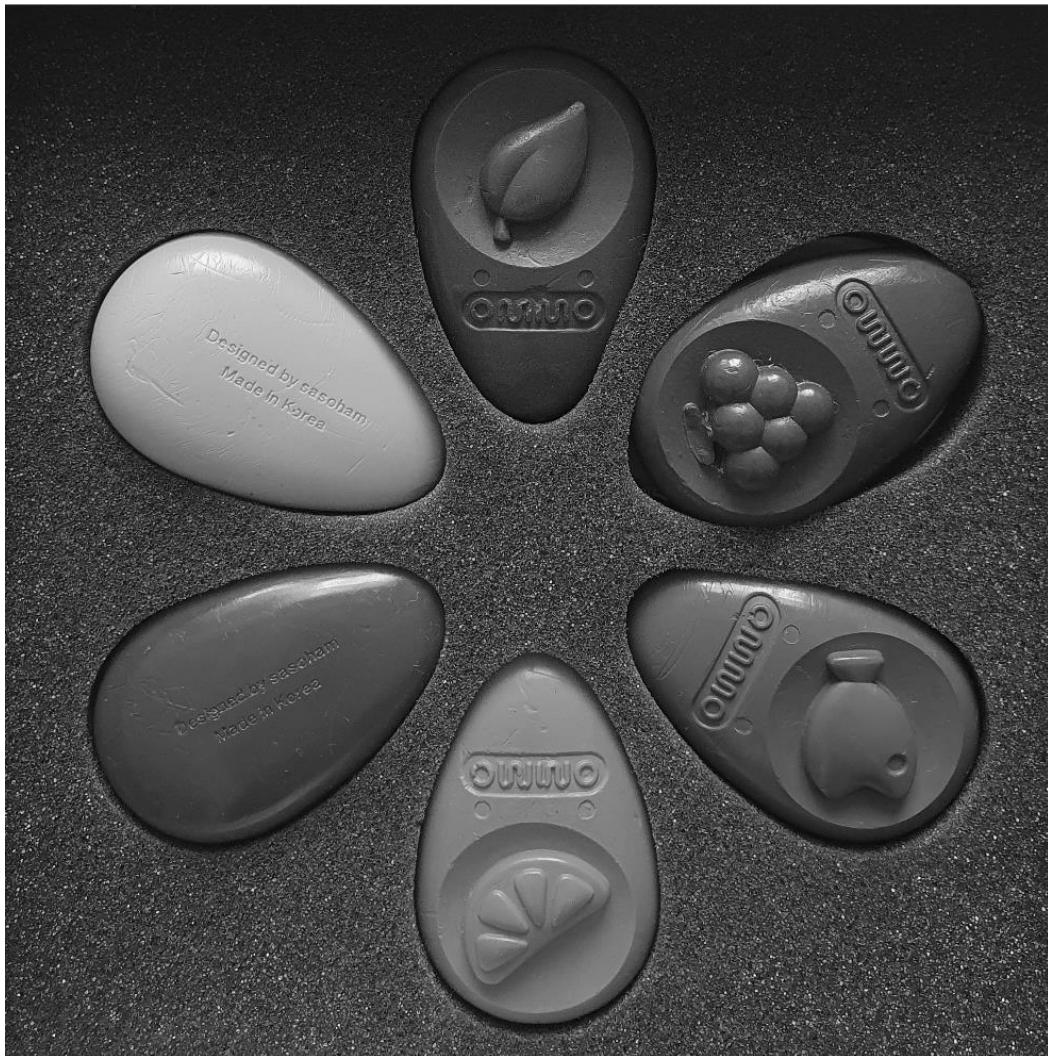


2) 이미지 크기 resize 및 rotation



R1/R2 단계 별 알고리즘 설명

3) 흑백 변환 후 edge 검출



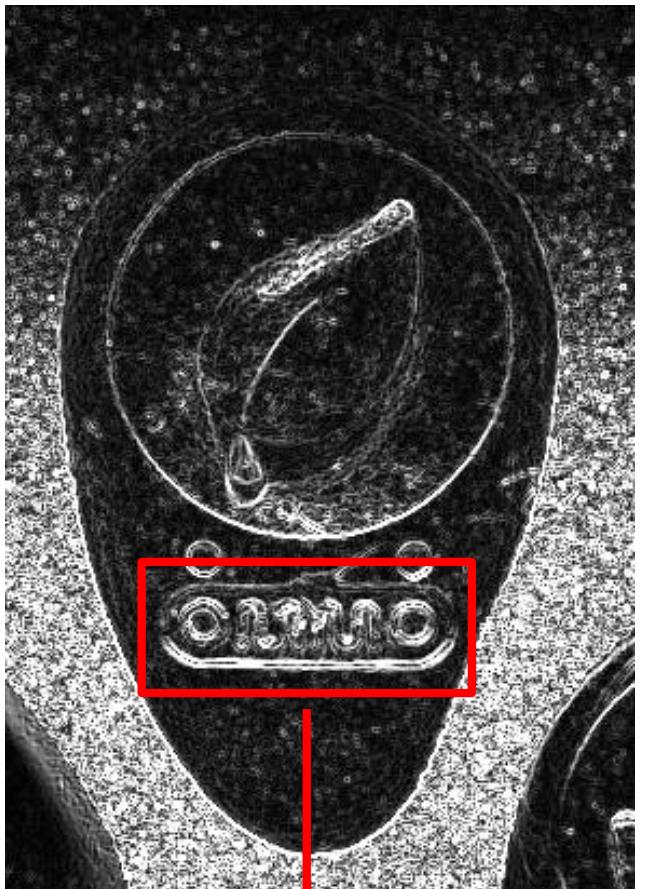
```
img_gray=cv2.cvtColor(img_re,cv2.COLOR_BGR2GRAY)
```

```
img_edge_x = cv2.Sobel(img_gray, cv2.CV_64F, 1, 0, ksize=3)
img_edge_x = cv2.convertScaleAbs(img_edge_x)
img_edge_y = cv2.Sobel(img_gray, cv2.CV_64F, 0, 1, ksize=3)
img_edge_y = cv2.convertScaleAbs(img_edge_y)
img_edge = cv2.addWeighted(img_edge_x, 1, img_edge_y, 1, 0)
```

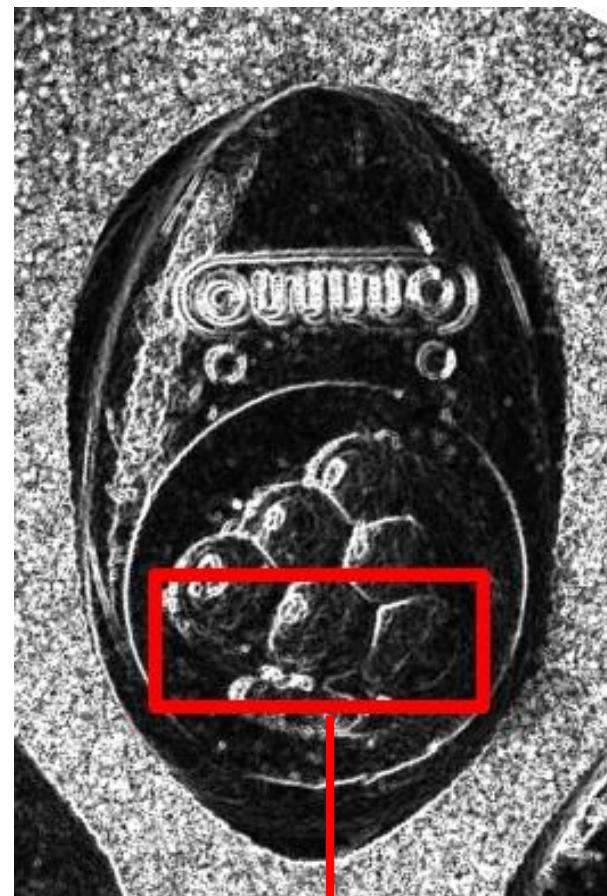
R1/R2 단계 별 알고리즘 설명

4) 정상인 이미지와 비교할 부분 crop

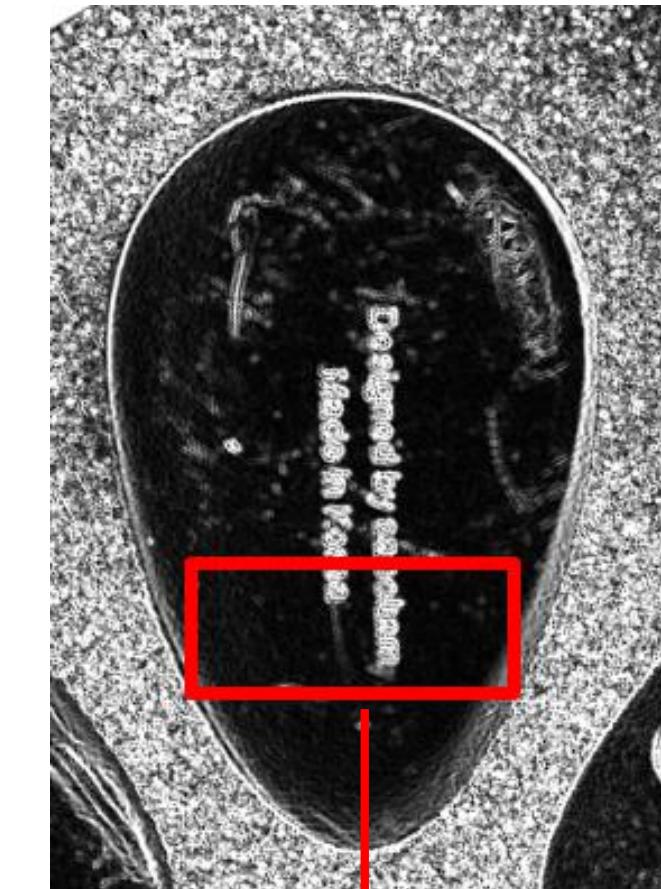
정상인 경우



회전된 경우



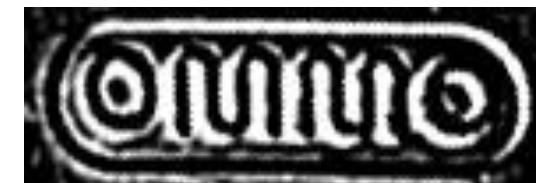
뒤집힌 경우



R1/R2 단계 별 알고리즘 설명

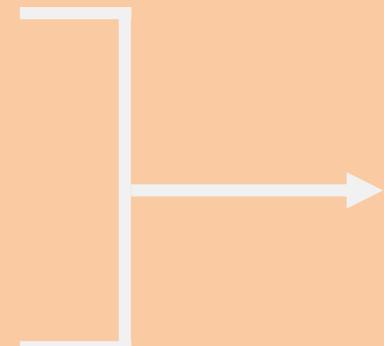
5) 불량(R1, R2)검출

정상인 경우



edge 검출이 많이 됨

뒤집힌 경우



edge 검출이 거의 안됨



불량(R1, R2) 검출

회전된 경우



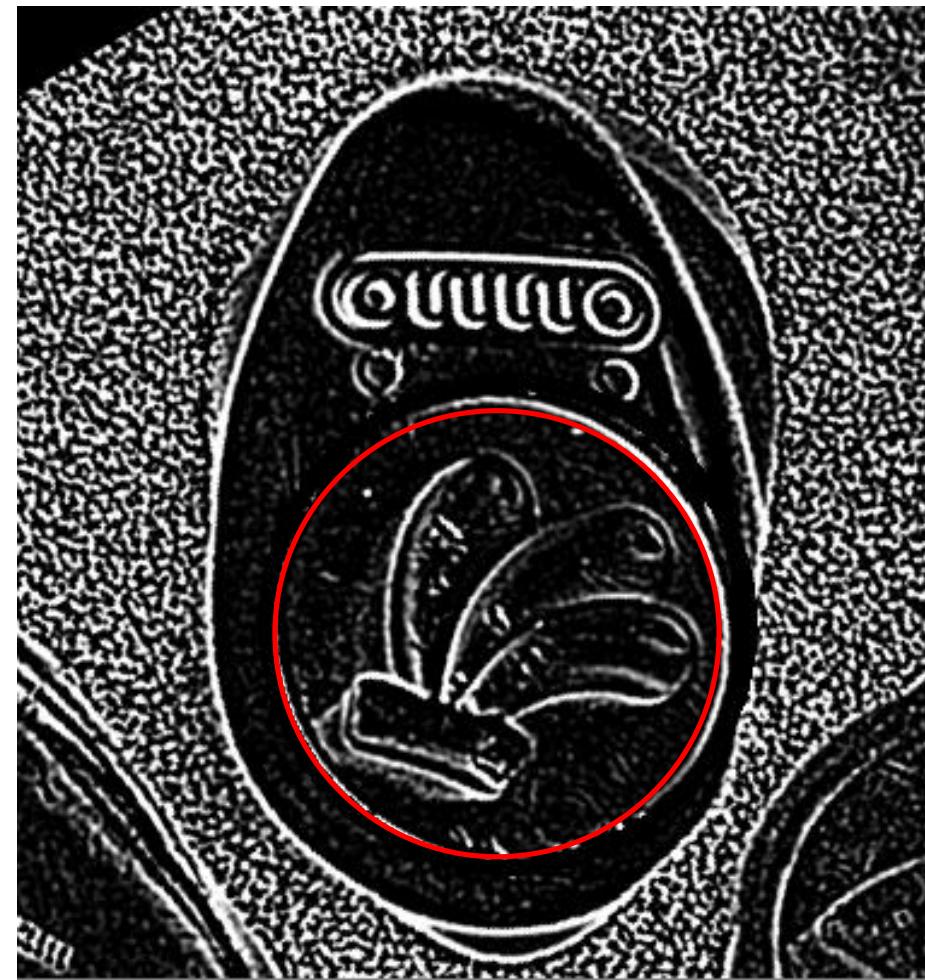
R1 단계 별 알고리즘 설명

6) 불량 검출된 크레용들에 한해 원 검출 → 원이 검출되지 않으면 뒤집힌 크레용

```
img_gray_cir = cv2.GaussianBlur(img_gray, (5,5), 0) #가우시안 블러  
img_edge_cir = cv2.Laplacian(img_gray_cir, cv2.CV_8U, ksize=5) #라플라시안 엣지
```



뒤집힌 경우
→ 원 검출이 되지 않는다.



회전된 경우
→ 원 검출이 된다.

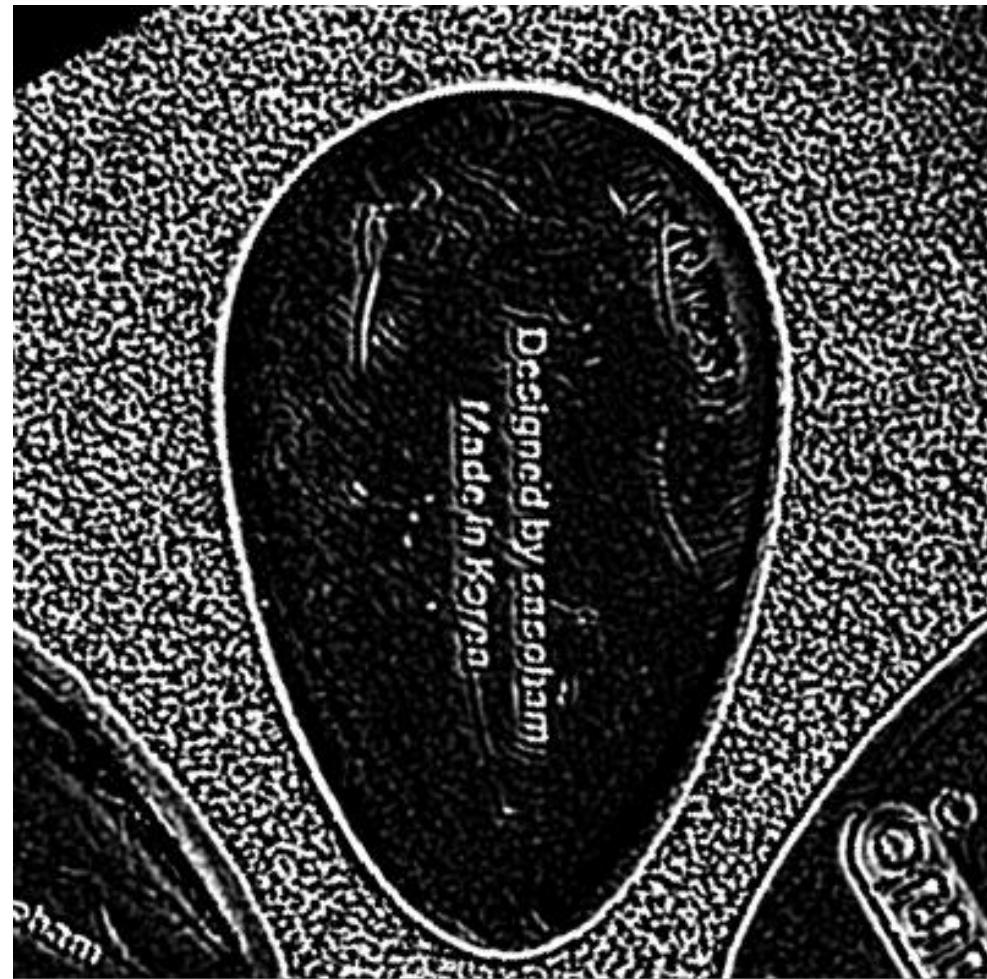


원 검출이 되지 않은 것들을
뒤집힌 경우(R1)이라 판단

R2 단계 별 알고리즘 설명

6) 불량 검출된 크레용들에 한해 원 검출 → 원이 검출되면 회전된 크레용

```
img_gray_cir = cv2.GaussianBlur(img_gray, (5,5), 0) #가우시안 블러  
img_edge_cir = cv2.Laplacian(img_gray_cir, cv2.CV_8U, ksize=5) #라플라시안 엣지
```



뒤집힌 경우
→ 원 검출이 되지 않는다.

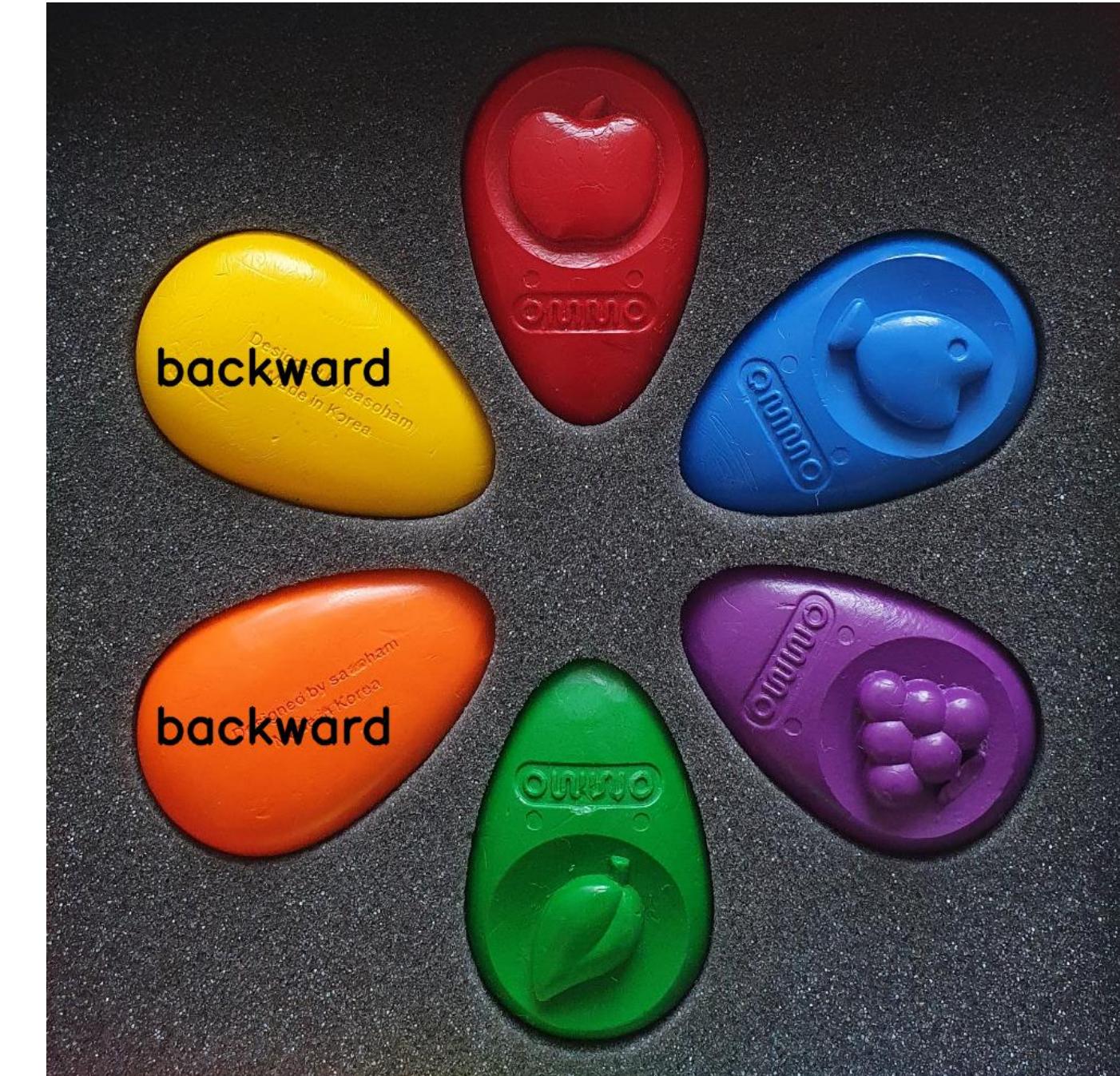


원 검출이 된 것들을
회전된 경우(R2)이라 판단

R1 검사 결과



정상일 경우 검사 결과



R1, R3 비정상 / R2 정상일 경우
검사 결과

R2 검사 결과



정상일 경우 검사 결과



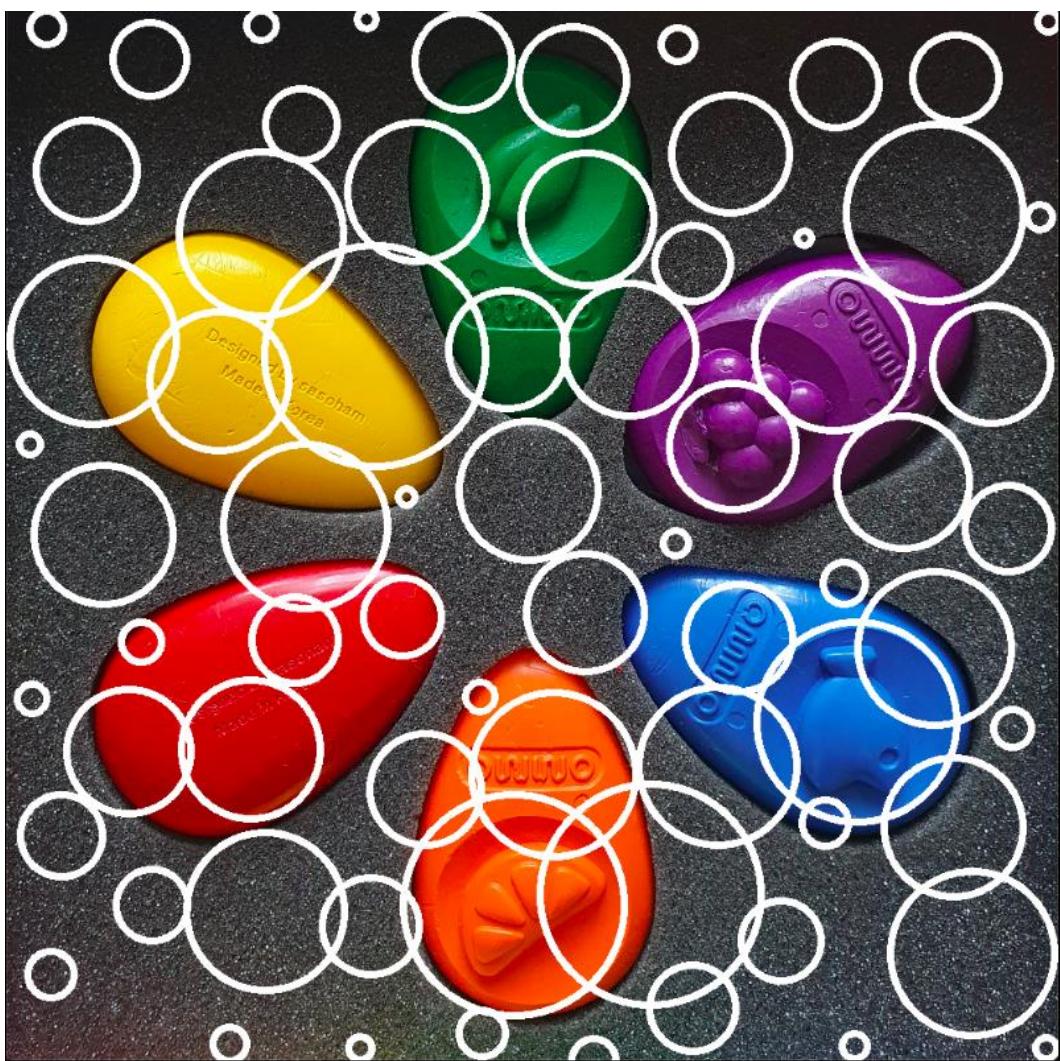
R2 정상 / R1, R3 비정상일 경우
검사결과



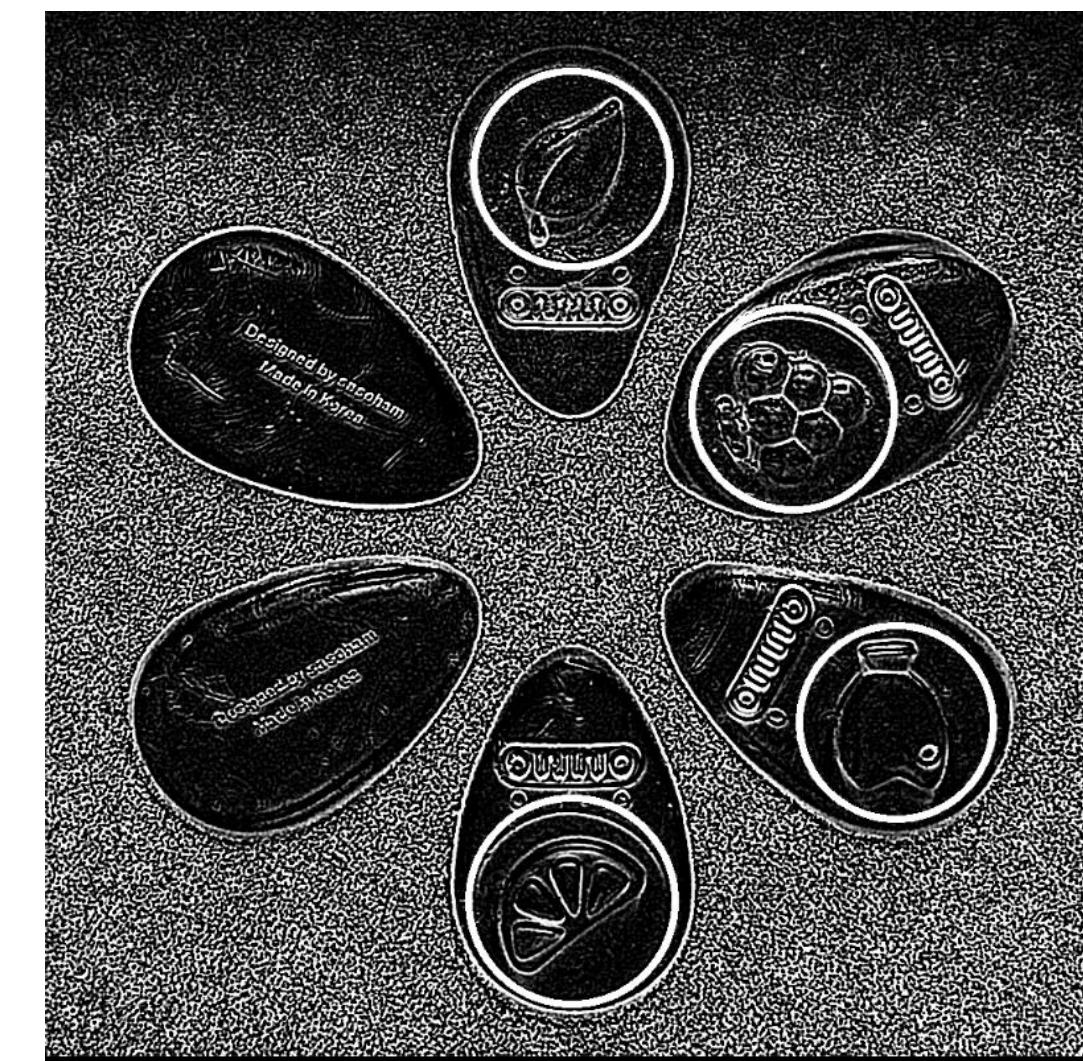
R1, R2, R3 비정상일 경우
검사결과

R2 검사 결과

원 검출에서 전처리가 필요한 이유



가우시안 전처리를 하지 않고 원 검출 한 경우



가우시안 전처리를 하고 원 검출 한 경우

R3 접근 방법 또는 아이디어

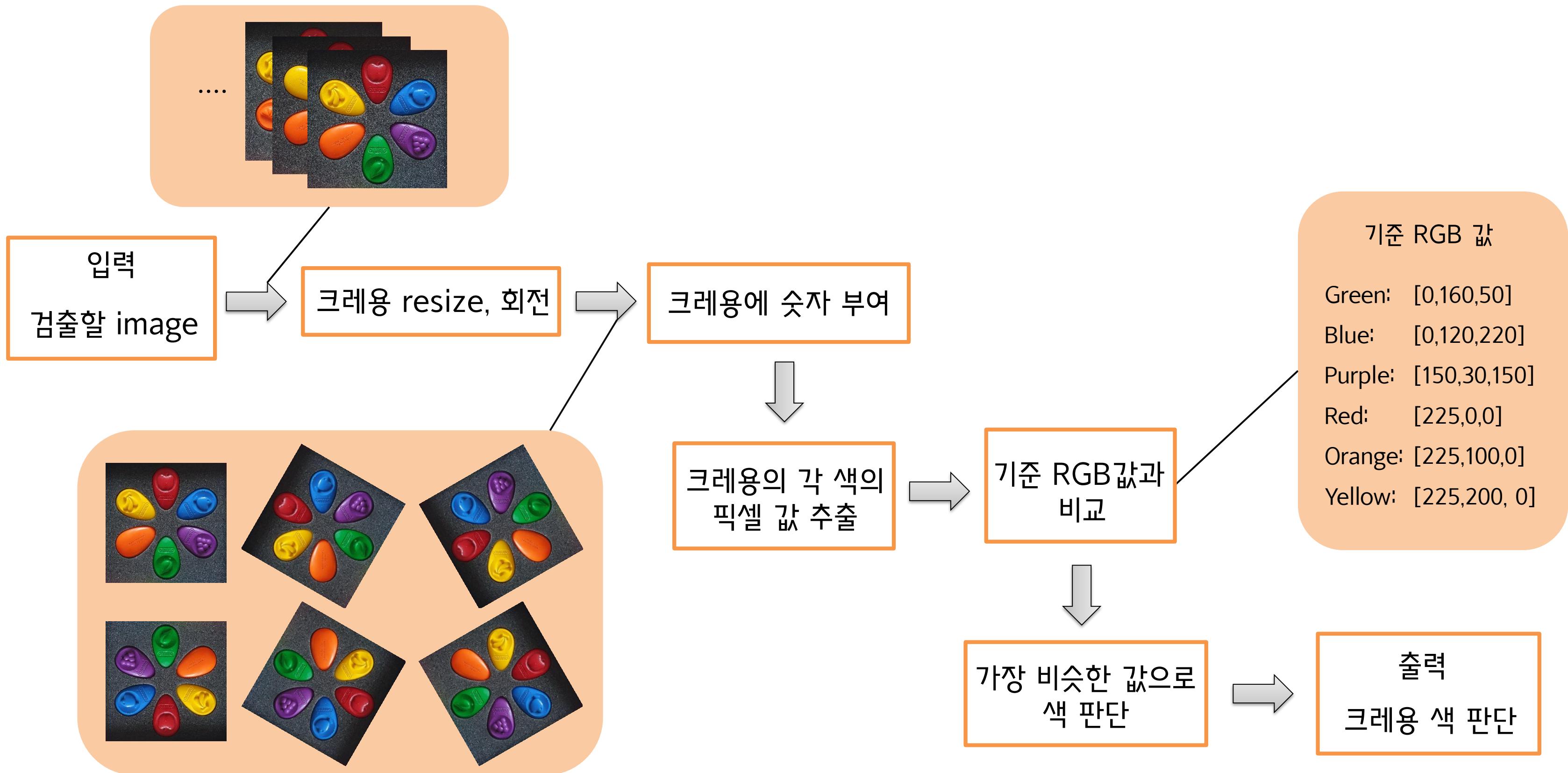
아이디어

1. 이미지 사이즈의 통일
2. 이미지의 회전을 통해 각각의 크레용을 같은 방향으로 두고 검출

생각해본 접근 방법

1. 색깔 별로 기준 RGB 값을 설정하여, 검출된 각각의 크레용 색과 비교 후 가장 가까운 색으로 크레용의 색을 판단한다.

R3 전체 흐름도

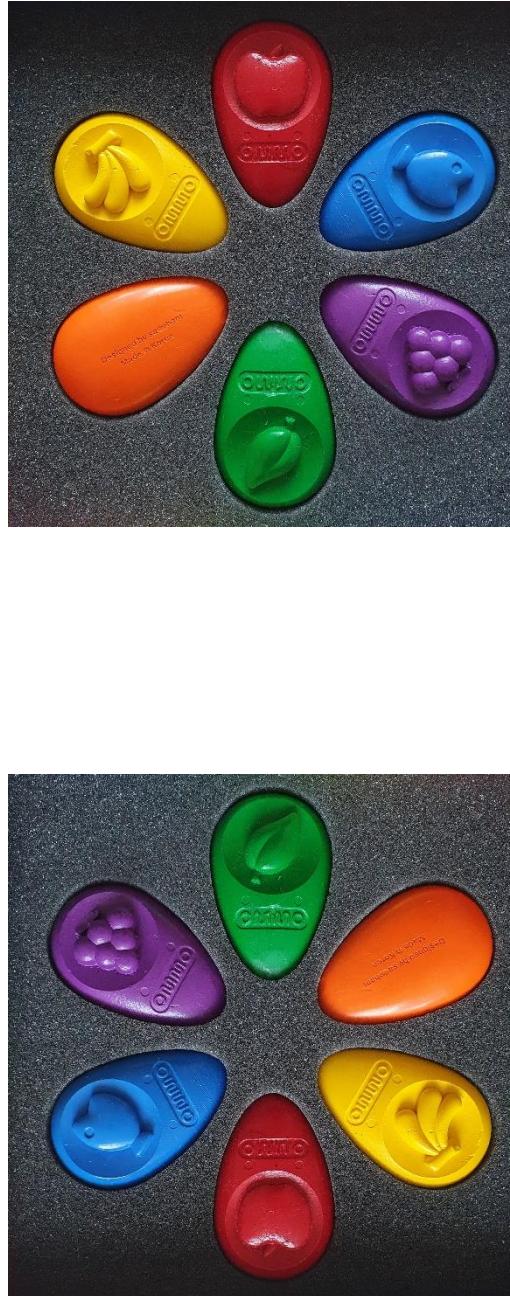


R3 단계 별 알고리즘 설명

1) 검출할 이미지 입력



2) 이미지 크기 resize 및 rotation



R3 단계 별 알고리즘 설명

3) 크레용에 순서 부여



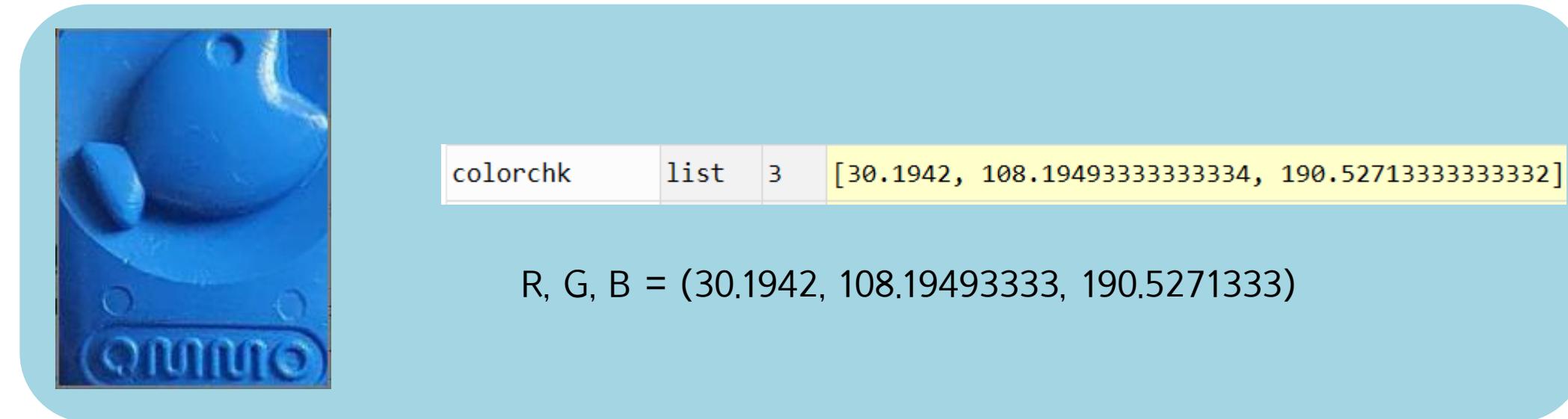
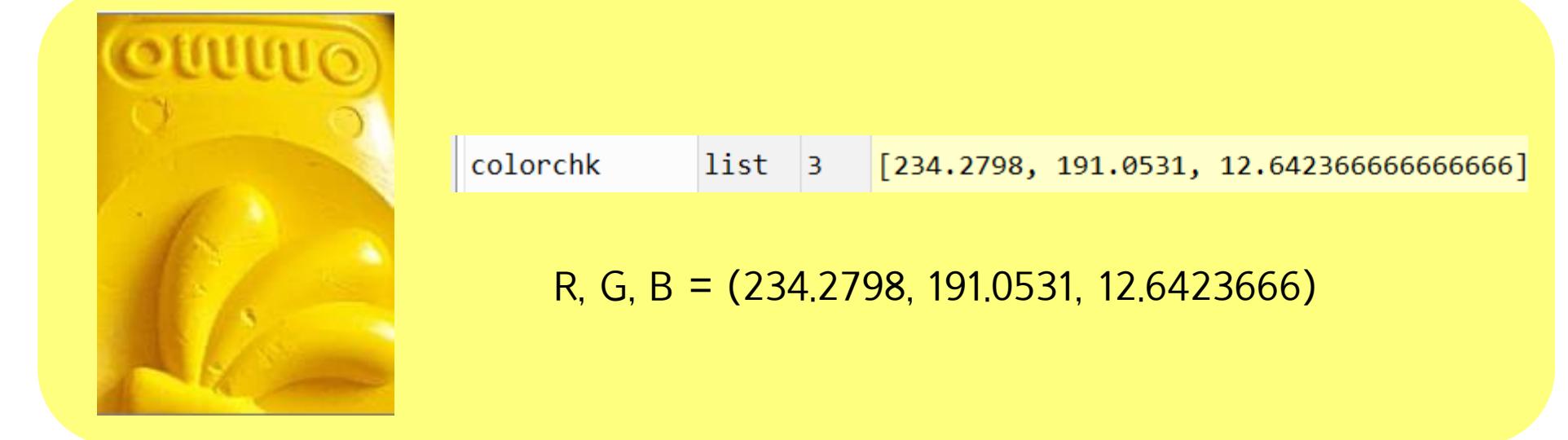
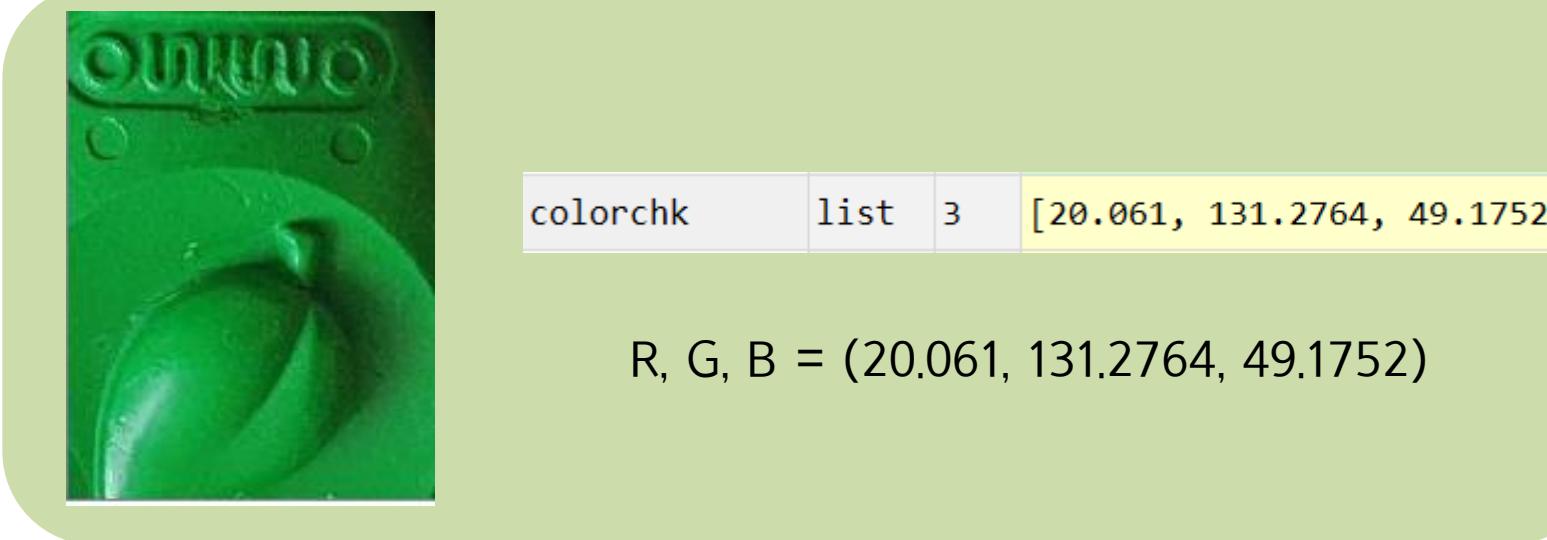
정상인 이미지의 12시 방향을 기준으로,
초록색을 1
파란색을 2
보라색을 3
빨간색을 4
주황색을 5
노란색을 6
으로 숫자를 부여한다.

R3 단계 별 알고리즘 설명

4) 크레용 부분만 crop 후 각각의 크레용의 픽셀 값(R,G,B) 추출

```
img_B = rotated[100:300 , 425:575,0] # B 이미지만 획득  
img_G = rotated[100:300 , 425:575,1] # G 이미지만 획득  
img_R = rotated[100:300 , 425:575,2] # R 이미지만 획득
```

```
colorchk[0] = np.mean(img_R)  
colorchk[1] = np.mean(img_G)  
colorchk[2] = np.mean(img_B)
```



R3 단계 별 알고리즘 설명

5) 추출한 크레용의 RGB 값과 기준 RGB 값과 비교

기준 RGB 값

Green: [0,160,50]

Blue: [0,120,220]

Purple: [150,30,150]

Red: [225,0,0]

Orange: [225,100,0]

Yellow: [225,200, 0]



colorchk list 3 [20.061, 131.2764, 49.1752]

R, G, B = (20.061, 131.2764, 49.1752)

Green과 가장 비슷



colorchk list 3 [234.2798, 191.0531, 12.642366666666666]

R, G, B = (234.2798, 191.0531, 12.6423666)

Yellow와 가장 비슷



colorchk list 3 [30.1942, 108.1949333333334, 190.52713333333333]

R, G, B = (30.1942, 108.19493333, 190.52713333)

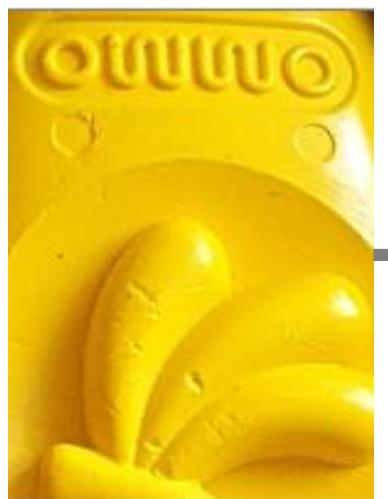
Blue와 가장 비슷

R3 단계 별 알고리즘 설명

6) 비교해 가장 가까운 RGB 값에 해당하는 색으로 판단 → 크레용의 색 검출



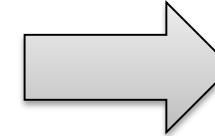
Green으로 판단: 숫자 1



Yellow로 판단: 숫자 6



Blue로 판단: 숫자 2



R3 검사 결과



정상일 경우 검사 결과

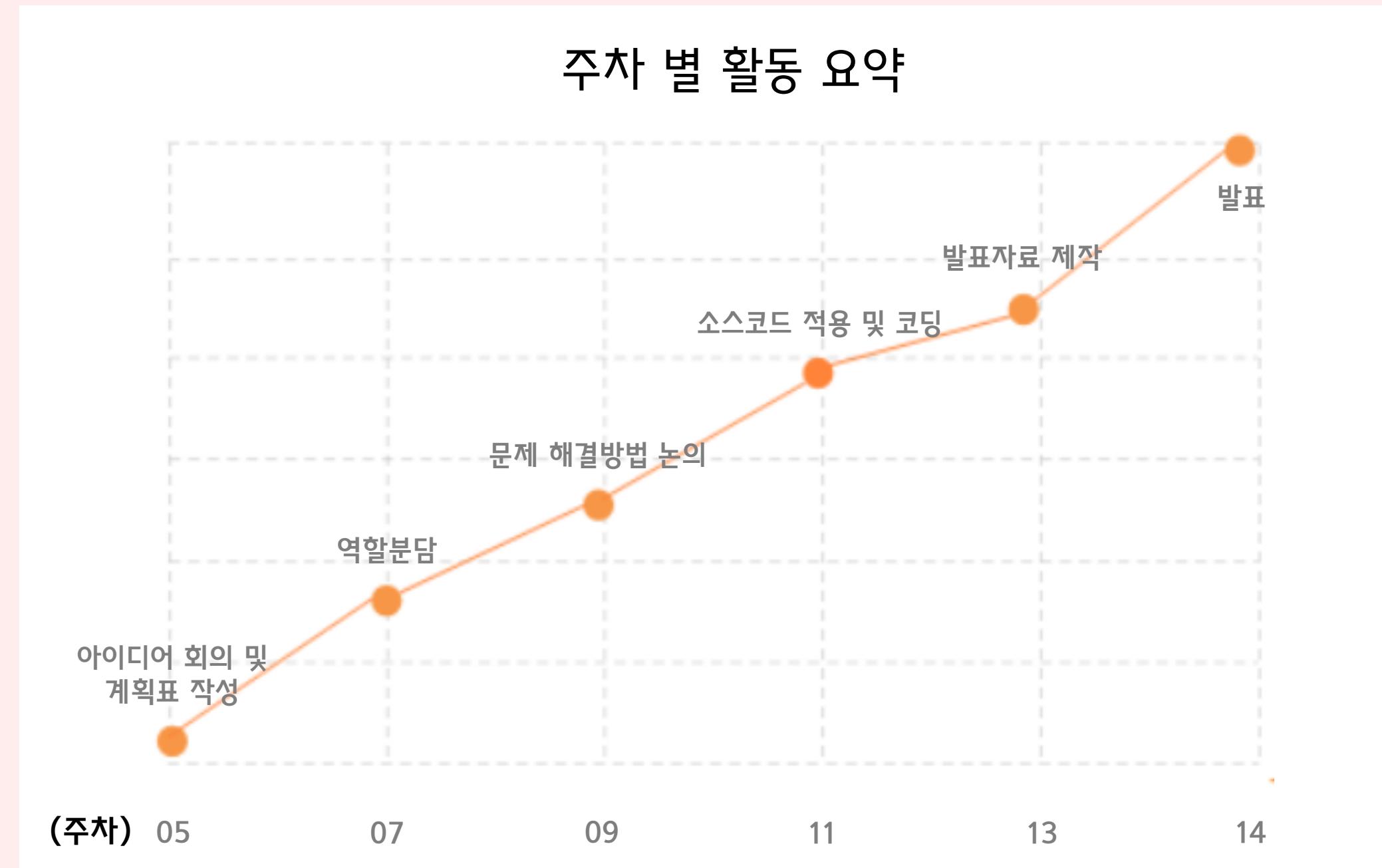


R2 정상 / R1, R3 비정상일 경우
검사 결과



R1, R2, R3 비정상일 경우
검사 결과

PBL 팀 활동 요약



PBL 팀 활동 요약

아이디어

› 크레용의 회전

크레용이 12시 위치에서부터 시계방향으로 60° 씩 회전되어 있다는 것을 발견했다.

› 크레용을 같은 방향으로 두고 검출하자는 의견에 따라, 팀미팅을 통해 제공받은 회전 소스코드를 활용하여 다음 사진과 같이 6개의 크레용을 모두 같은 방향으로 두고 R1, R2, R3를 검출 할 수 있었다.



PBL 팀 활동 요약

R1과 R2 알고리즘 작성에서

처음 아이디어: R1과 R2를 한번에 검출하기 위해 edge 검출로 전처리 후 픽셀 값의 평균으로 R1과 R2를 검출

최종 아이디어: edge 검출을 통해 R1과 R2를 한번에 검출 후 원 검출로 R1, R2 구분

문제 1: 처음 아이디어는 픽셀 값이 24~51의 범위에 고르게 분포되어 R1과 R2를 구분할 수 없었다.

해결방안: “크레용의 모양을 활용해 검출해보자”라는 의견이 나와 크레용에 있는 원을 활용하기로 하였다.

따라서 원 검출 소스코드를 이용해 원이 검출되지 않으면 R1과 원이 검출되면 R2로 구분할 수 있었다.

문제 2: 처음 원 검출 소스코드를 사용했을 때, 검출하고자 하는 부분이 제대로 검출되지 않았다.

해결 방안: 팀원 모두 원 검출에 대해 정보를 찾아보았고, 전처리가 필요하다는 것을 알게 되었다.

따라서 라플라시안 블러를 사용하여 원 검출 전 전처리를 행하였고, 원 검출을 해 원하는 결과를 얻을 수 있었다.

감사합니다.[®]

크레용 포장 불량 검사 알고리즘