OpenCV 를 활용한 Contour 검출 보고서

뮤텍코리아 지원자 이지혁

Illu140411@gmail.com

010-8480-9400

뮤텍코리아

목차

1. 개요 3

2. 개발 환경 설정 3

3. 이미지 처리 구현 3

4. 단계별 구현 설명 4

5. 실행 결과 6

6. 추가 기능 8

7. 결론 11

8. 추후 개선 방안 11

1. 개요

본 보고서는 Python과 OpenCV 라이브러리를 활용하여 이미지 처리 및 윤곽선 검출을 수행하는 과정을 상세히 기술합니다. RAW_02-66_0309 이미지를 대상으로 Grayscale 변환, Binary 변환, Contour 검출 및 시각화 작업을 단계별로 진행하였습니다. 또한 추가기능1) 임계점 변화에 따른 Contour 검출의 차이, 추가기능2) 검출된 Contour의 기하학 모양에 따른 분류, 추가기능3) Binary 전후 가우시안 필터를 통한 이미지처리에 따른 Contour 검출 변화를 서술하고 있습니다.

2. 개발 환경 설정

2.1 개발 환경

1. VScode
2. Python 3.13.5
3. opencv-python 4.11.0.86

2.2 실행 절차

1. 프로젝트 다운로드
2. 파이썬 가상환경 생성

```
>> bash
python -m venv venv
```

3. 필요 모듈 설치

```
>> bash
pip install -r requirements.txt
```

4. 파이썬 인터프리터 설정
5. 프로젝트 실행

3. 이미지 처리 구현

3.1 전체 소스 코드(프로젝트 폴더 내부)

1. **readme.md 참조 – 프로젝트 개요 및 실행 방법
2. **opencv.py 파일 코드 및 주석 참조 – 메인 구현 코드
3. **opencv_method.md 참조 – 사용된 라이브러리의 기능 정리

4. 단계별 구현 설명

4.1 이미지 읽기

- 원본 이미지 읽기

  cv2.imread() 함수를 사용하여 이미지를 로드

```
# 원본 이미지 읽기
image_origin = cv2.imread(image_path)
```

  ➢ 이미지 경로 오류 발생시 프로그램 종료하도록 예외처리

```
# 없다면 에러처리
if image_origin is None:
    raise FileNotFoundError(f'이미지를 찾을 수 없습니다')
```

- Grayscale 이미지 생성

  두번째 매개변수가 0이거나 cv2.IMREAD_GRAYSCALE 을 통하여 흑백이미지 형성

```
# 그레이 스케일 이미지 생성
image_gray = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
```

4.2 Binary 이미지 생성

- 함수 호출 : set_image_binary()

```
image_binary = set_image_binary(image_gray) # 이진 이미지 생성 함수 호출
```

- 처리 방식

  np.where() 를 이용하여 임계값 100을 기준으로 binary_image 생성

```
image_binary = np.where(image_gray < 100, 255, 0).astype(np.uint8)
```

  ➢ 이진 이미지의 생성 오류시 None 반환

```
# 이진 이미지 검증 0 과 255 가 아닌 값이 있을 경우 NONE
if not np.all(np.isin(image_binary, [0, 255])):
    print("이진 이미지 생성 실패: 0 과 255 이외의 값이 존재합니다.")
    return None
```

4.3 Contour 검출

- 함수 호출 : detect_contours()

```
result_contour = detect_contours(image_binary)
```

- 처리 방식

  cv2.findContours() 함수를 통해 contours 검출

  2번째, 3번째 매개변수인 mode와 method를 다양하게 지원하며,

  사용한 cv2.RETR_EXTERNAL mode는 외곽선만 검출하며, cv2.CHAIN_APPROX_SIMPLE method는 윤곽선의 점들을 근사화하여 저장한다.

```
contours, _ = cv2.findContours(image_binary, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
```

  - contour가 검출되지 않았을 경우 None 반환

4.4 윤곽선 그리기

- 원본 이미지에 검출된 Contour들을 그림

  cv2.drawContours() 함수는 매개변수로 contour들과 contourIdx, 색상(BGR), 두께 등을 받아 이미지를 그린다.

```
cv2.drawContours(image_contour, result_contour, -1, (0, 255, 0), 2)
```

- 함수 호출 : display_result()

```
display_result(mode, image_origin, image_gray, image_binary,
image_contour)
```

- 처리 방식

  matplotlib.pyplot을 이용하여 Figure를 그리며, 각각의 사진들을 보여 준다.

4.4 완성된 이미지 저장

- 함수 호출 : save_results()

```
save_results(mode, image_origin, image_gray, image_binary,
image_contour)
```

- 처리 방식

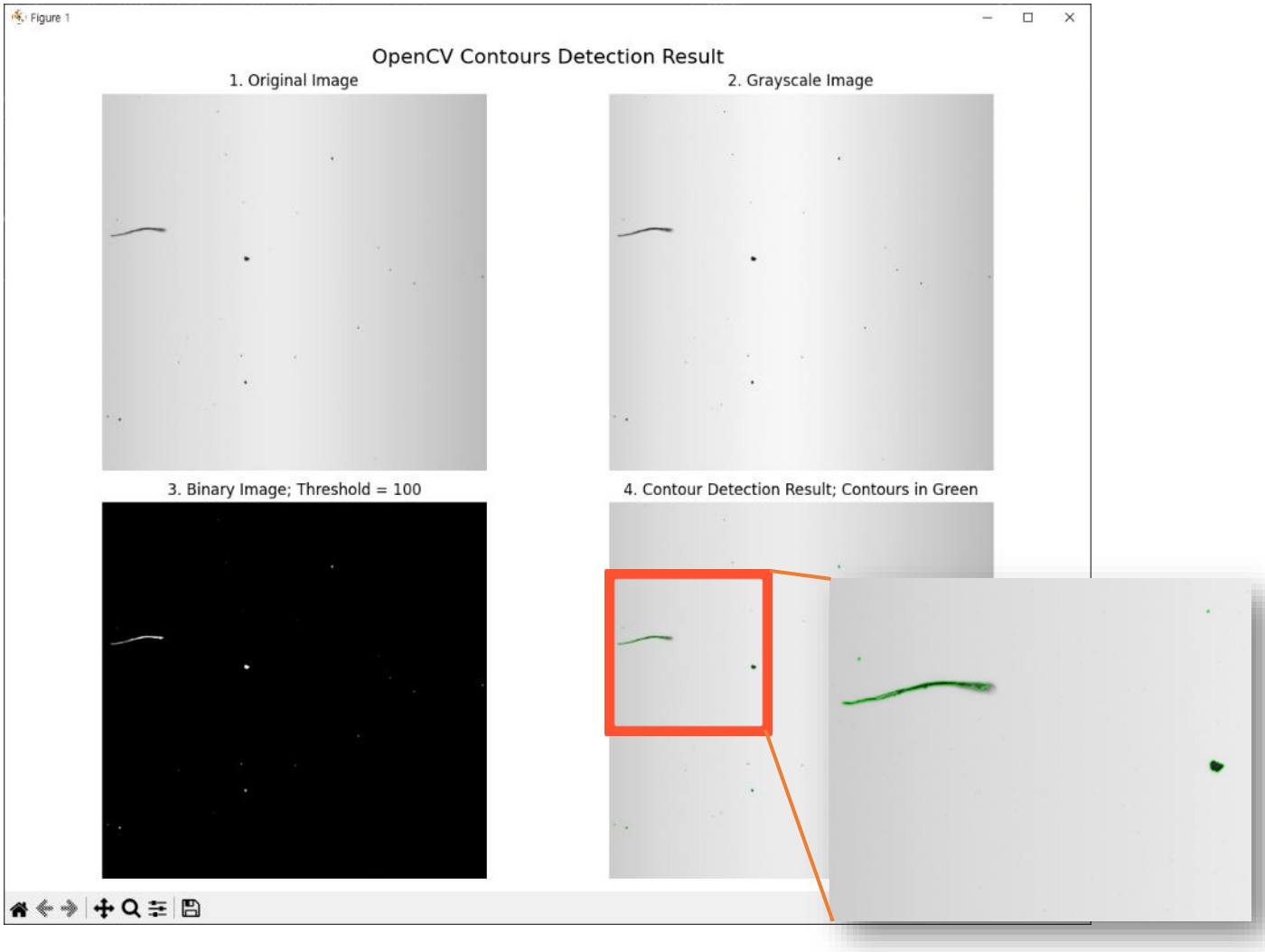
  파일 경로와 매칭되는 이미지들을 저장한다.

5. 실행 결과

5.1 로그 출력

```
=== main() 시작 ===

1단계 : 이미지 읽기
이미지 파일을 성공적으로 읽었습니다 -> 파일경로 : RAW_02-66_0309.bmp
원본 이미지 크기 -> 세로: 3915 가로: 4000
그레이스케일 이미지 크기 -> 세로: 3915 가로: 4000

2단계 : 이진 이미지 생성 시작
이진 이미지 생성 완료
Threshold 값 -> 100
이진 이미지 크기 -> 세로: 3915 가로: 4000

3단계 : 컨투어 검출 시작
검출된 컨투어 개수: 26
컨투어 1: 면적 = 11.50, 둘레 = 17.07
컨투어 2: 면적 = 264.00, 둘레 = 62.77
컨투어 검출 완료, 총 26개의 컨투어가 검출되었습니다.

4단계 : 결과 이미지 출력
결과 이미지 생성중
결과 이미지 생성 완료
plt 시작
plt 종료

5단계 : 결과 이미지 저장 시작
결과 이미지 저장중
image_origin.png 저장 완료
image_gray.png 저장 완료
image_binary.png 저장 완료
image_contour.png 저장 완료
결과 이미지 저장 완료
=== main() 종료 ===
```

5.2 결과 이미지



처리 결과는 2x2 격자로 구성된 시각화 창에 표시됩니다:

1. **원본 이미지 (좌상단)**: RAW_02-66_0309의 원본 RGB 컬러 이미지
2. **Grayscale 이미지 (우상단)**: 색상 정보를 제거한 흑백 이미지
3. **Binary 이미지 (좌하단)**: 임계값 100을 기준으로 이진화된 이미지
4. **Contour 검출 결과 (우하단)**: 원본 이미지에 녹색 윤곽선이 그려진 결과

6. 추가 기능

6.1 임계점 변화에 따른 Contour 검출 변화

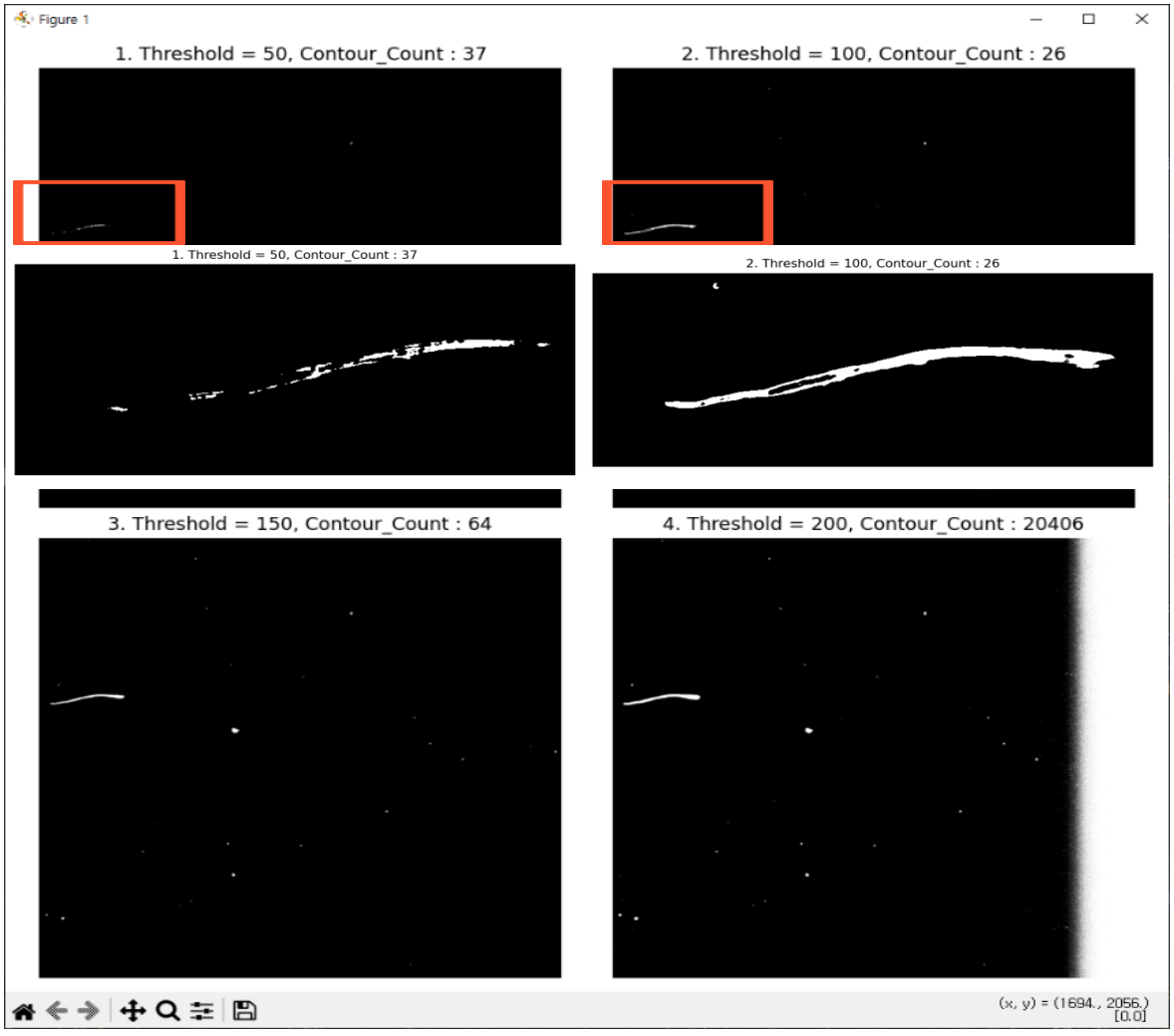
6.1.1 목적

임계값 설정이 Contour 검출 결과에 미치는 영향을 분석하기 위해 설정하였습니다.

6.1.2 조건

임계값을 50, 100, 150, 200으로 설정하여 비교 분석합니다.

6.1.3 결과 분석



50, 100을 비교하여 50이 더많은 Contour를 검출하는데, 이는 배경과 구분되어 하나의 객체가 잘린 것도 개수에 포함됩니다. 반대로 200의 경우 조명의 명암까지도 배경으로 인식하여, 원본 사진이 잘리는 경우도 존재합니다. 다만 일반적으로는 임계값이 커질수록 컨투어 검출이 많아지는 것으로 추정됩니다.

이에 따라 Contour 검출에 사용될 임계값의 적정값을 찾는 것이 필요합니다.

6.2 Contour 모양에 따른 분류

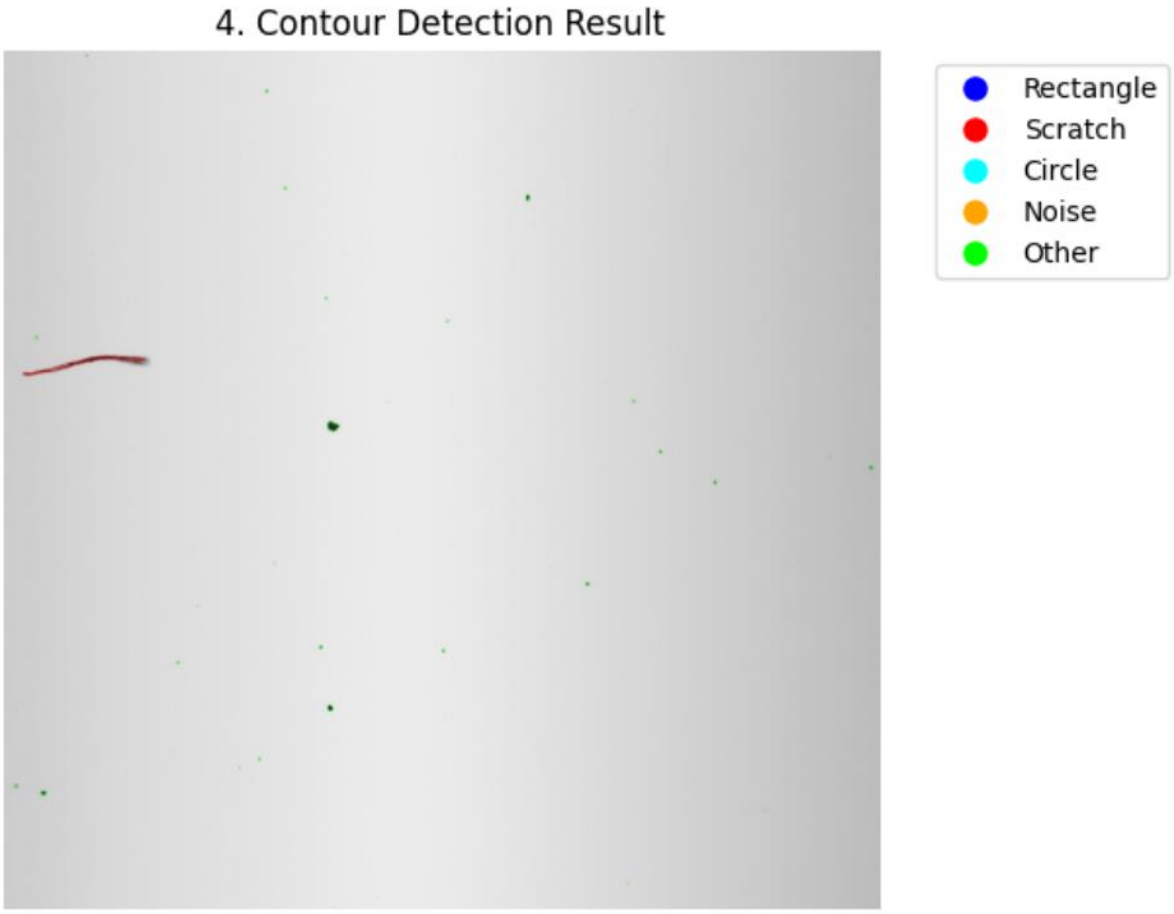
6.2.1 목적

Wafer의 불량 검출에 있어 이미지에서 검출된 Contour를 기하학적 특성에 따라 자동으로 분류하여 객체의 종류를 구분하는 것을 목표로 설정하였습니다.

6.2.2 조건

다양한 형태의 Contour를 면적, 둘레, 원형도, 꼭지점 등의 특성으로 분석 및 분류합니다.

6.2. 3 결과 분석



각각의 Contour를 5가지 카테고리로 분류하였습니다. 사각형(파란색), 스크래치(빨간색), 원형(청록색), 노이즈(주황색), 기타(초록색)으로 구분하였고, 이를 통해 특정 모양의 객체를 선별적으로 후처리 할 수 있습니다.
추후 불량 검출을 위한 모델링을 진행한다면 분류조건을 불량검출 정확성을 보장할 수 있도록 설계해야 합니다.

6.2 가우시안 블러를 통한 이미지 개선

6.2.1 목적

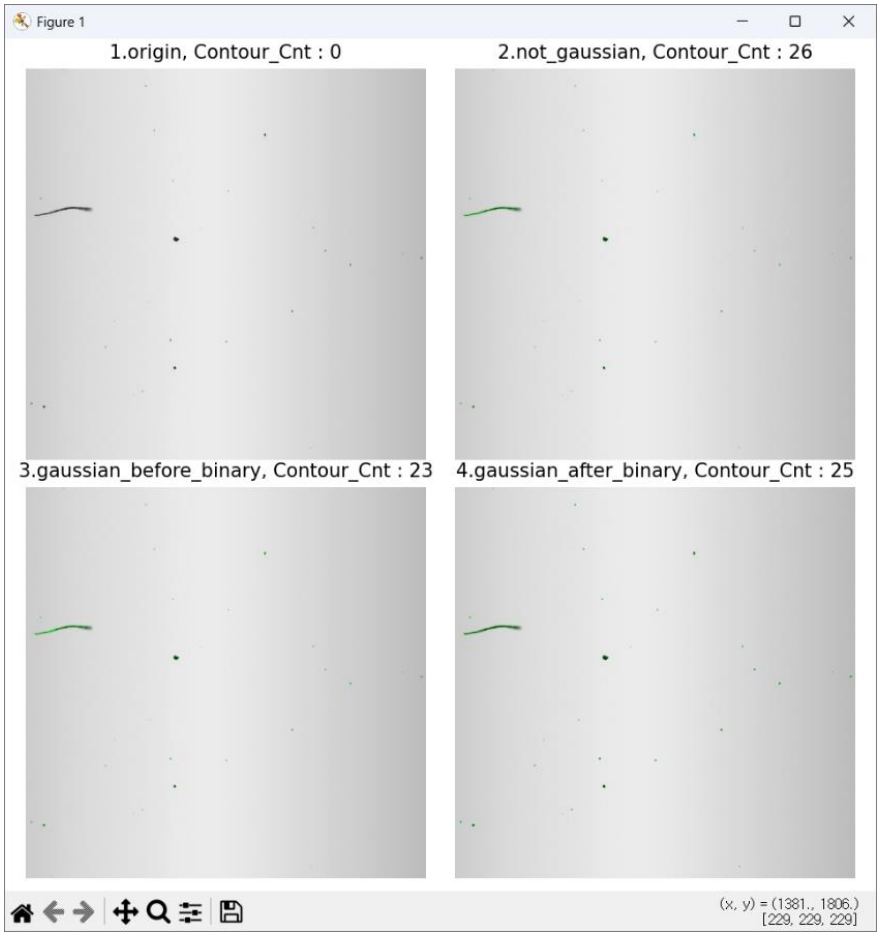
이미지 개선에 많이 사용되는 가우시안 블러처리를 바이너리 전 후에 적용하여 Contour의 차이를 확인하고, Contour 검출의 품질 향상을 위해 진행하였습니다.

6.2.2 조건

가우시안 블러를 바이너리 이미지 생성 전 후로 적용하고, 가우시안 블러를 적용하지 않은 이미지와 Contour 개수를 비교합니다.

가우시안 블러 : 커널크기 7x7, 표준편차 1

6.2. 3 결과 분석



가우시안 미적용 : 26개

바이너리 전 적용 : 23개

바이너리 후 적용 : 25개

가우시안 블러를 적용하면 노이즈가 제거되어 컨투어 개수가 적어지는 것으로 추정됩니다.
적용 순서에 따라 검출되는 컨투어 개수가 차이가 나타났지만, 무엇이 더 효과적인 이미지 처리인지는 알 수 없었습니다.

7. 결론

본 프로젝트에서는 Python과 OpenCV를 활용하여 이미지 처리 방식의 기초 플로우를 습득하였고, 이론적 지식을 실제 코드로 구현하는 경험을 쌓았습니다. 기본 구현 완료 후 추가적으로 다양한 기능들을 하나씩 구현해보며, 각 처리 단계별 차이점들을 분석하고 비교하는 과정을 통해 이미지 처리가 다양한 조건과 매개변수에 따라 다른 결과를 도출할 수 있다는 것을 확인하였습니다.

특히 임계값 변경, 윤곽선 검출 방식 조정 등의 실험을 통해 각 단계가 최종 결과에 미치는 영향을 간접적으로 체험할 수 있었습니다. 다만, 일부 처리 결과의 명확한 원인 분석은 현재의 지식 수준으로는 한계가 있었으며, 이는 향후 더 깊이 있는 학습이 필요한 부분으로 판단됩니다.

위 경험을 통해 컴퓨터 비전 분야의 기초를 다졌으며, 향후 더 고급 이미지 처리 기법 및 등을 다루며 뮤텍코리아에서 개발자로 성장하고 기여하고 싶습니다.

8. 추후 개선 방안

1. **적응형 임계값 처리** 심화
2. **윤곽선 필터링** 기능 추가 (면적, 둘레 기준)