OpenCV를 이용한 Contour 검출 과정

뮤텍코리아 지원자 이지혁
illu140411@gmail.com

CONTENTS

개요

반도체 Wafer의 불량 검출 프로세스

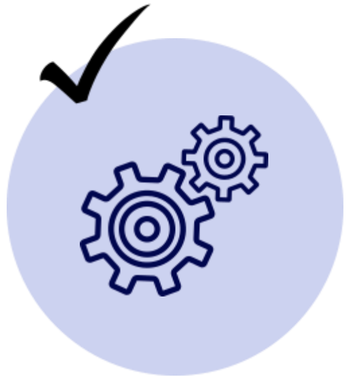
OpenCV를 활용한 Contour 검출

이미지 품질 개선 방안

Wafer의 불량 검출 프로세스

이미지 획득

광학 혹은 스캐닝을 통한
이미지 획득 과정



이미지 전처리

노이즈 제거
이미지 보정

특징 추출

윤곽선 검출
특징 추출

불량 검출

패턴 매칭
결함 분류

이미지 전처리 및 특징 추출 프로세스

Step 1

이미지 읽기 및
GrayScale 이미지 생성

Step 2

Binary 이미지 생성

Step 3

Contour 검출

Step 4

결과 이미지 출력 및 저장

이미지 품질 개선

이미지 필터링

- **가우시안 필터**

  가우시안 분포를 기반으로 이미지에 부드러운 블러 처리가 적용되지만 경계가 어느정도 흐려지는 특성이 있다.

- **미디언 필터**

  커널 내부의 픽셀들의 중간값으로 대체하여 흰점, 검은점에 매우 효과적으로 노이즈가 제거되지만 가는 선이나 작은 구조가 손실될 수 있다.

경계 강화 기법

- **언샤프 마스킹**

  원본 이미지에서 블러된 이미지를 제거하고, 고주파 성분을 추출한 후 강화하여 사용한다. 이에 따라 경계부분을 강화시킬 수 있다.

- **히스토그램 평활화**

  이미지를 작은 타일 형태로 구분하여 히스토그램을 평활화를 진행한다. 명암 대비를 높이는 기법으로 이미지를 선명하게 할 수 있지만 이미 밝은 부분이 더 밝아져 품질이 낮아지는 경우도 있다.

구현 내용

이미지 읽기

```python
# openCV 컨투어 검출
# 이지혁
# 최초 코드 작성일 : 2025.06.19
# 수정 코드 작성일 : 2025.06.20
# other_thresholds 기능 추가 : 2025.06.19
import numpy as np···


def main(image_path):
    print("=== main() 시작 ===")
    print("\n1단계 : 이미지 읽기")
    try:
        # 원본 이미지 읽기
        image_origin = cv2.imread(image_path)

        # 없다면 에러처리
        if image_origin is None:···

        # 그레이 스케일 이미지 생성
        image_gray = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        print(f"이미지 파일을 성공적으로 읽었습니다 -> 파일경로 : {image_path}")
        print(f"원본 이미지 크기 -> 세로: {image_origin.shape[0]} 가로: {image_origin.shape[1
        print(f"그레이스케일 이미지 크기 -> 세로: {image_gray.shape[0]} 가로: {image_gray.sha

    except FileNotFoundError as e:
        print(f"error : {e}")
        return
```

1. c2.imread() 를 통한 이미지 읽기

⇩

[ERROR] 파일을 찾지 못하면 Error 처리 및 프로그램 종료

⇩

2. Binary 이미지 생성을 위한GrayScale 이미지 생성

Binary 이미지 생성

1. 이진 이미지 형성 함수 set_image_binary 함수 호출

⇩

2. np.where()를 통해 Threshold 값 100 이상이면 0(검정색), 100 미만이면 255(흰색) 처리

⇩

Error 처리 : 0과 255가 아닌 값이 있다면 ERROR, 프로그램 종료

⇩

3. 이진 이미지 형성 완료

```
print("\n2단계 : 이진 이미지 생성 시작")
image_binary = set_image_binary(image_gray) # 이진 이미지 생성 함수 호출
if image_binary is None:···
print(f"이진 이미지 생성 완료\nThreshold 값 -> 100")
print(f"이진 이미지 크기 -> 세로: {image_binary.shape[0]} 가로: {image_binary.shap
```

```
def set_image_binary(image_gray): # 2단계; 이진 이미지 생성 함수

    # Threshold을 이용한 이진 이미지, Threshold 값 100
    # 여기서 흑백 반전을 하는 이유 = 검은 배경의 흰점 처리가 더 유리하다
    # openCV 의 findContours() 함수는 흰색 픽셀(255)를 컨투어(윤곽선)으로 인식하고 검은픽셀(0)을

    #region np, cv2 두 가지 방법으로 이진 이미지 생성(장,단점 비교) 택 1
    # np.where
    ### 장점 : 조건의 다양화(ex gray< 80, gray > 200 등) 가능, ### 단점 : 파이썬 루프 기반(상대적
    # cv2.threshold
    ### 장점 : C++ 기반(상대적 빠름), ### 단점 : 조건의 다양화 불가(단순 임계값만 가능)

    # cv2.threshold를 이용한다면 Thresh_BINARY_INV 옵션을 사용하여 흰색 픽셀을 0(검정)으로
    # 검은색 픽셀을 255(흰색)로 반전해주는 것이 countours 검출에 유리하다
    #endregion

    image_binary = np.where(image_gray < 100, 255, 0).astype(np.uint8)
    # _, image_binary = cv2.threshold(image_gray, 100, 255, cv2.THRESH_BINARY_INV)

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

    return image_binary
```

Contour 검출

```python
print("\n3단계 : 컨투어 검출 시작")
result_contour = detect_contours(image_binary)
if result_contour is None:
    print("컨투어가 검출되지 않았습니다.")
else:
    print(f"컨투어 검출 완료, 총 {len(result_contour)}개의 컨투어가 검출되었습니다.")
```

```python
def detect_contours(image_binary): # 3단계; 컨투어 검출 함수
    #region 컨투어 검출
    # cv2.RETR_EXTERNAL: 가장 외곽 컨투어(255)만 검출
    # cv2.CHAIN_APPROX_SIMPLE: 컨투어 포인트를 압축하여 저장
    #endregion
    contours, _ = cv2.findContours(image_binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    print(f"검출된 컨투어 개수: {len(contours)}")

    if len(contours) == 0:
        return None
    else:
        for i, contour in enumerate(contours):
            # 컨투어의 면적 계산
            area = cv2.contourArea(contour)
            perimeter = cv2.arcLength(contour, True)
            print(f"컨투어 {i+1}: 면적 = {area:.2f}, 둘레 = {perimeter:.2f}")
        return contours
```

1. Contour 검출 함수 detect_contours() 호출

⇩

2. findContours() 를 통한 contour 검출

⇩

Error 처리 : contours 배열길이가 0이라면 None 반환

⇩

3. 컨투어의 값을 통해 면접 및 둘레 계산 로그 출력

⇩

4. 컨투어 검출 완료

결과 이미지 출력 및 저장

1. 원본 이미지에 drawContours()를 이용하여 contour 부분 Green으로 변경

⇩

2. matplotlib.pyplot 을 이용하여 이미지 display

⇩

3. pyplot 종료시, 저장 함수 save_result() 실행

⇩

Error 처리 : 이미지 저장 실패시 NONE 반환

⇩

4. 저장 완료

```
print("\n4단계 : 결과 이미지 출력")
image_contour = cv2.cvtColor(image_origin.copy(), cv2.COLOR_BGR2RGB)
# 컨투어 green으로 그림
cv2.drawContours(image_contour, result_contour, -1, (0, 255, 0), 2)
display_result(image_origin, image_gray, image_binary, image_contour)

print("\n5단계 : 결과 이미지 저장 시작")
if (save_results(image_origin, image_gray, image_binary, image_contour) == None):···
else:
    print("결과 이미지 저장 완료")
print("=== main() 종료 ===")
```

```
def display_result(image_origin, image_gray, image_binary, image_contour): # 4단계 결과 이

def save_results(image_origin, image_gray, image_binary, image_contour): # 5단계 결과 이미

    print("결과 이미지 저장중")
    folder_path = "./result_img"
    images = {
        "image_origin": image_origin,
        "image_gray": image_gray,
        "image_binary": image_binary,
        "image_contour": image_contour
    }
    try:
        for key,value in images.items():
            cv2.imwrite(f"{folder_path}/{key}.png", value)
            print(f"{key}.png 저장 완료")
    except Exception as e:
        print(f"결과 이미지 저장 실패 {e}")
        return None
    print("결과 이미지 저장 완료")
```

결과

프로그램 로그

```
=== 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 ===
```

Pyplot 결과 화면

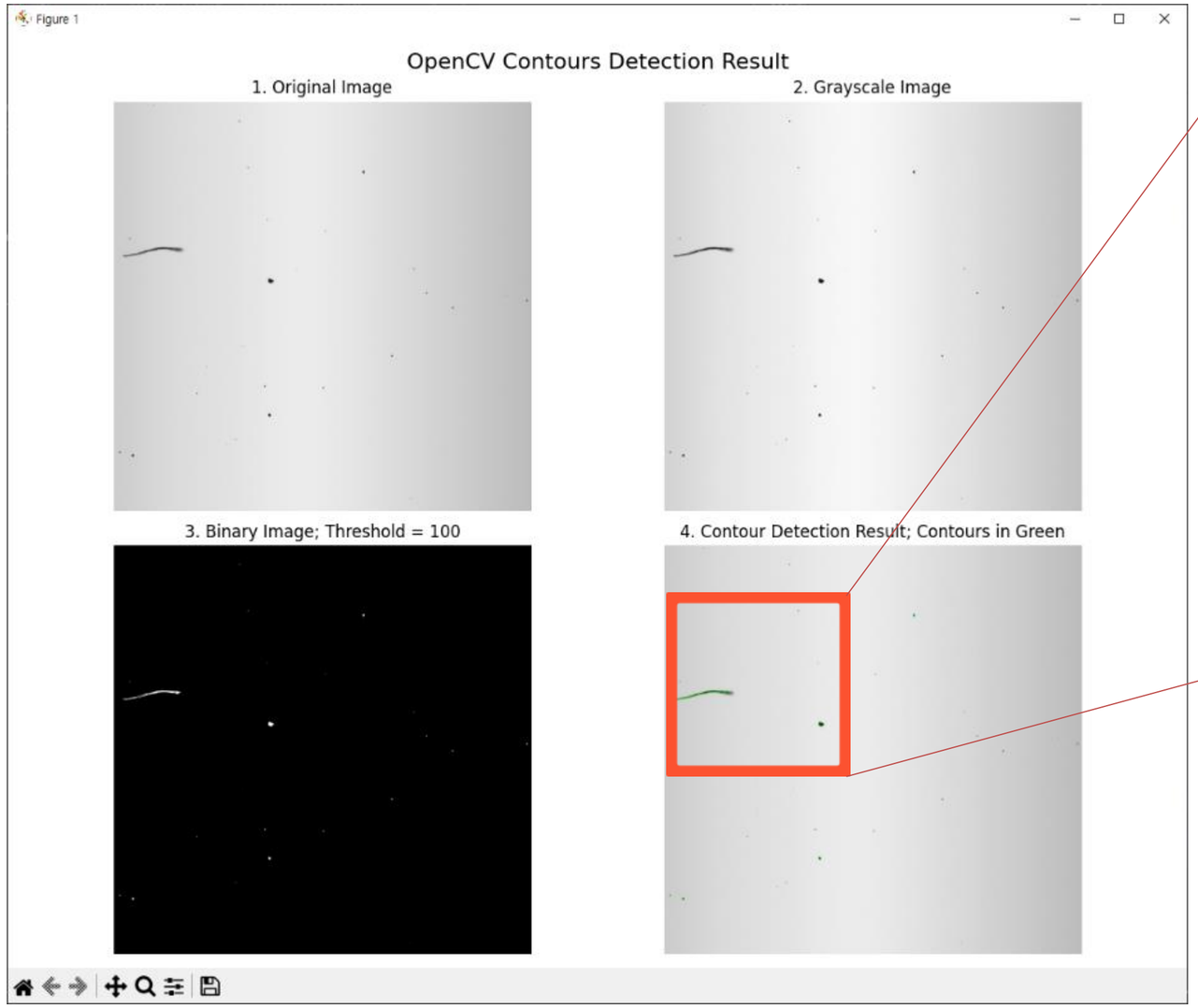
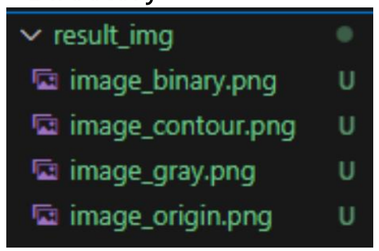


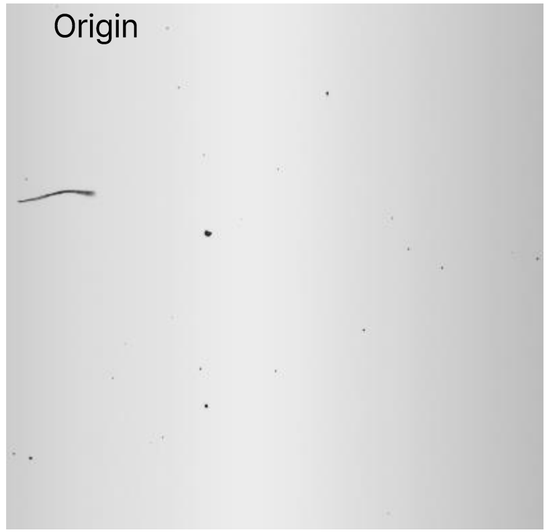
Figure 1
OpenCV Contours Detection Result
1. Original Image
2. Grayscale Image
3. Binary Image; Threshold = 100
4. Contour Detection Result; Contours in Green

이미지 저장 결과

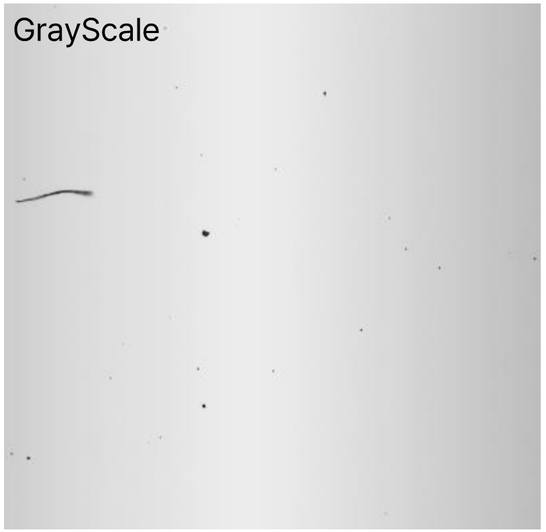
Directory



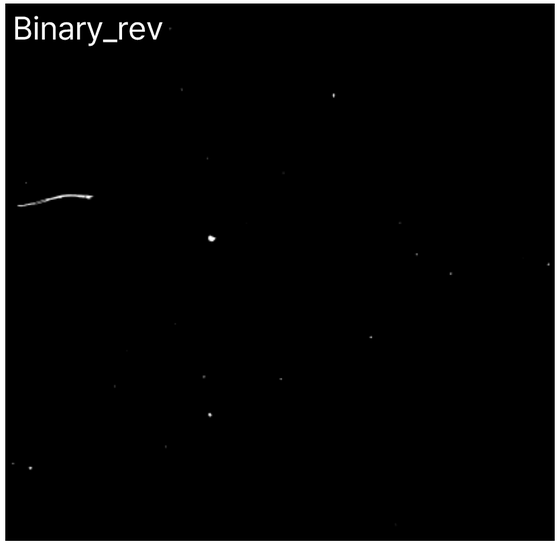
Origin



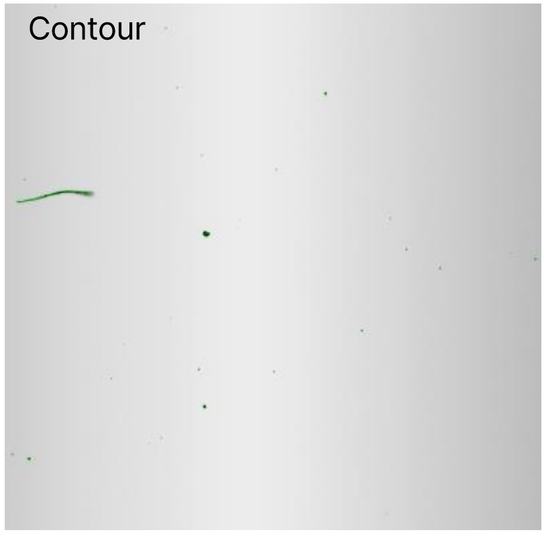
GrayScale



Binary_rev



Contour



추가 기능

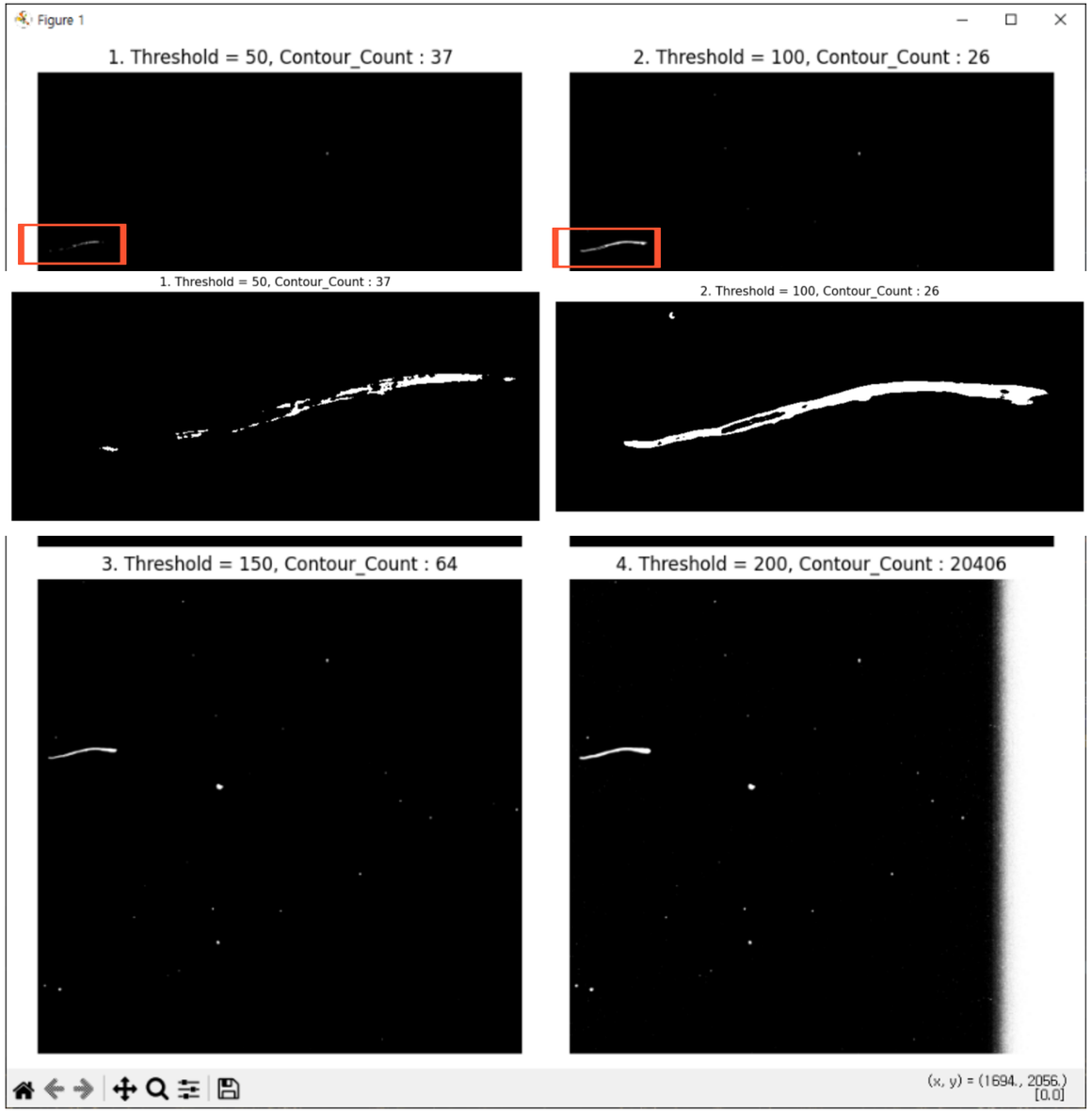
추가 기능1: 임계값에 따른 결과값

1. 목적
임계값 설정이 컨투어 검출 결과에 미치는 영향을 분석하여
최적의 임계값을 찾는 것을 목표로 합니다.

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

3. 결과 분석
50, 100을 비교하여 50이 더 많은 Contour를 검출하는데,
이는 Contour가 배경과 구분되어 잘린것도 개수에 포함됩니다.
반대로 200의 경우 조명의 명암까지도 배경으로 인식하여
원본 사진이 잘리는 경우도 존재합니다.
일반적으로 임계값이 커질수록 컨투어 검출이 많아집니다.

이에 따라 Contour 검출에 사용될 임계값의 적정값을 찾는 것이 필요로 합니다.



추가 기능2 : Contour 모양에 따른 분류

1. 목적

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

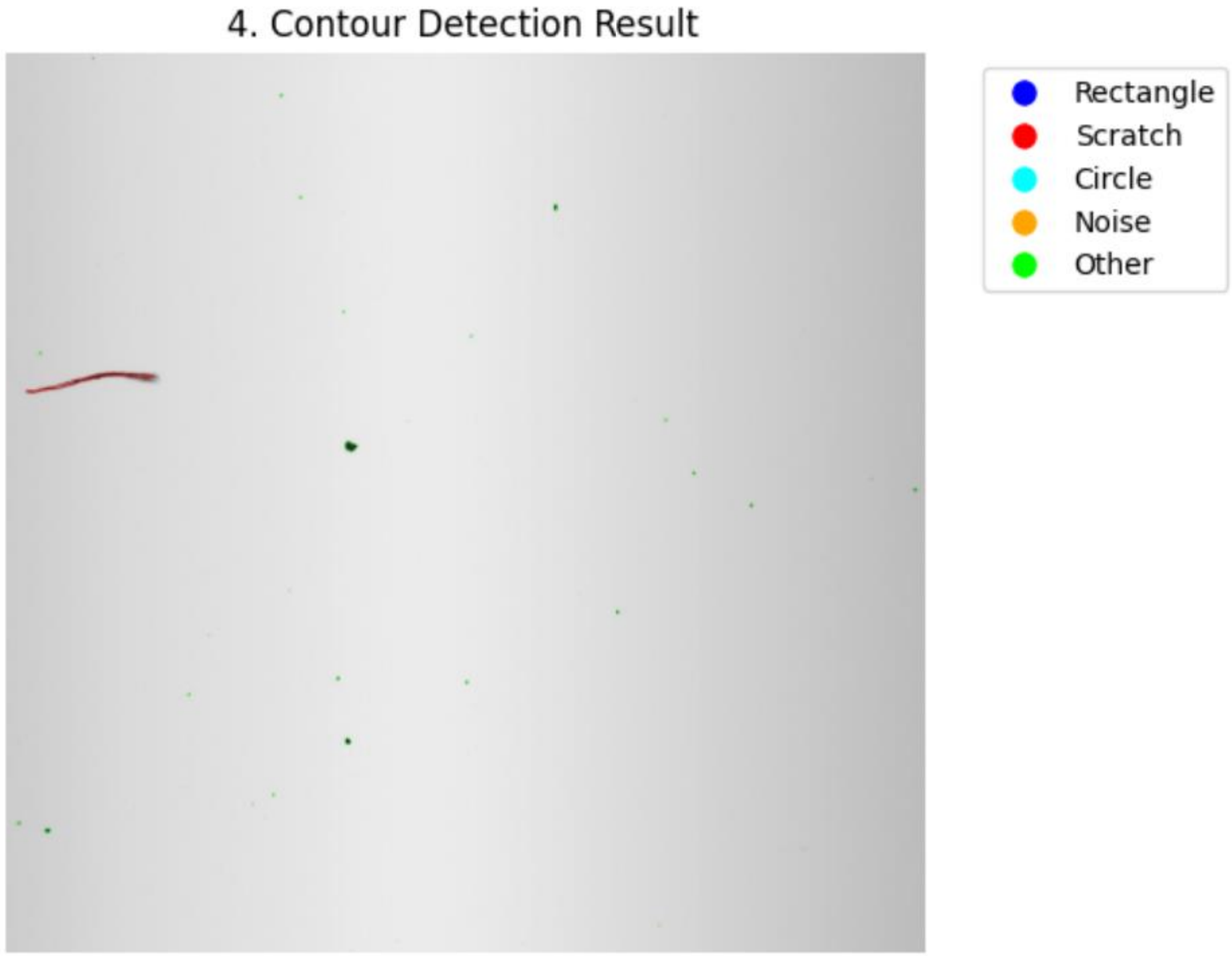
2. 조건

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

3. 결과 분석

각 Contour를 5가지 카테고리로 분류했습니다.
Rectangle(파란색)은 정사각형에 가까운 형태, Scratch(빨간색)는 길쭉한 선형 형태, Circle(청록색)은 원형에 가까운 형태로 구분되었습니다. 이를 통해 특정 모양의 객체를 선별적으로 처리할 수 있게 되었습니다.

추후 불량 검출하는 모델링에 사용할 수 있도록 분류의 조건을 불량 조건에 맞게 정확성을 보장할 수 있도록 설계해야 합니다.



추가 기능3 : 가우시안 블러를 통한 이미지 개선

1. 목적

이미지 개선을 위한 가우시안 블러의 바이너리 전 후에 적용하여
차이를 확인하고, 컨투어 검출의 품질 향상을 목표로 합니다.

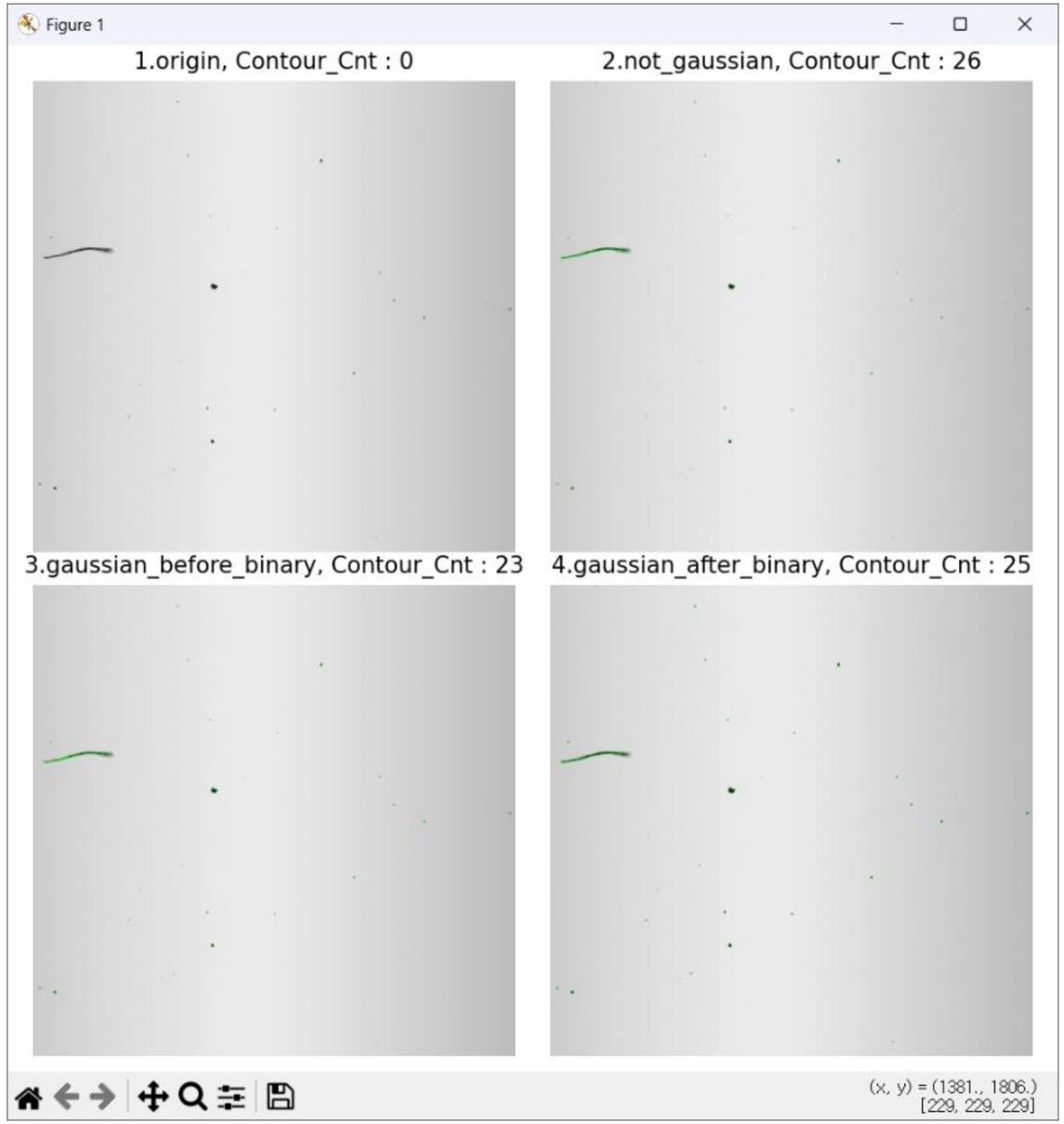
2. 조건

가우시안 블러를 바이너리 이미지 생성 전후로 적용하고,
가우시안 블러를 적용하지 않은 이미지와 Contour 개수 비교합니다.
가우시안 블러 : 커널 크기 7 x 7, 표준 편차 1

3. 결과 분석

가우시안 미적용: 26개 컨투어 검출
바이너리 전 가우시안 블러 적용: 23개 컨투어 검출
바이너리 후 가우시안 블러 작용: 25개 컨투어 검출

가우시안 블러 적용하면 노이즈가 제거되어 컨투어 개수가 적어지는 것으로
추정되며, 적용 순서에 따라 검출되는 컨투어 개수에 차이가 나타났습니다.
무엇이 더 효율적인지는 알 수 없었습니다.



회고

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.

추후 적응형 임계값을 위한 심화 처리나, 윤곽선의 필터링을 통하여 Contour 검출의 품질을 향상하여 이미지 처리를 개선할 수 있도록 하겠습니다.

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