HW11 LS 근사

Git code

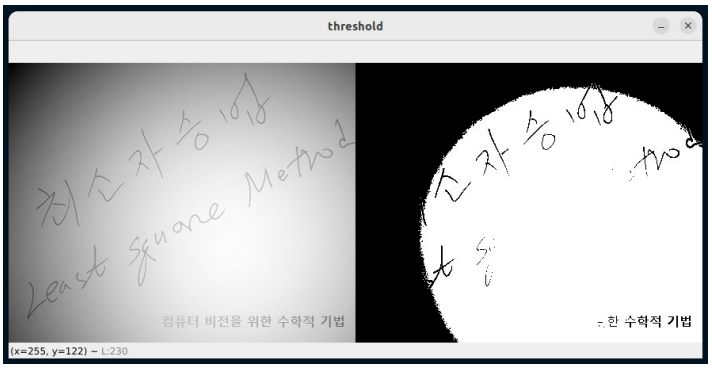
Mathvision22
02221081 황지현

Q1. Image Binarization Using Least Squares Method

1. Obtain the best binarized image of the provided sample image by a global thresholding (not by adaptive thresholding)

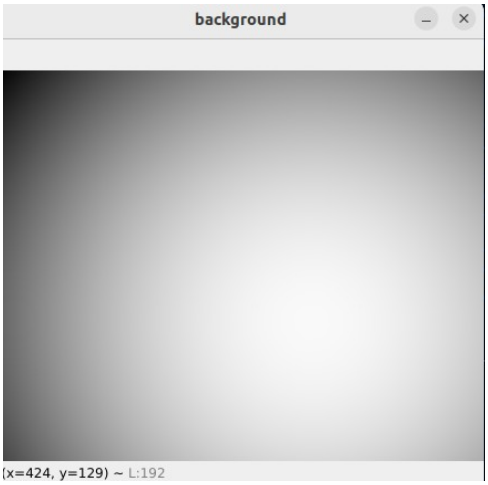
```python
def thres_finder(img, thres=20,delta_T=1.0):
    # Divide the images in two parts
    x_low, y_low = np.where(img<=thres)
    x_high, y_high = np.where(img>thres)
    # Find the mean of two parts
    mean_low = np.mean(img[x_low,y_low])
    mean_high = np.mean(img[x_high,y_high])
    # Calculate the new threshold
    new_thres = (mean_low + mean_high)/2
    # Stopping criteria, otherwise iterate
    if abs(new_thres-thres)< delta_T:
        return new_thres
    else:
        return thres_finder(img, thres=new_thres,delta_T=1.0)

# Load an image in the greyscale
img = cv2.imread('hw11_sample.png', cv2.IMREAD_GRAYSCALE)
# apply threshold finder
vv1 = thres_finder(img, thres=30,delta_T=1.0)
# threshold the image
ret, thresh = cv2.threshold(img,vv1,255,cv2.THRESH_BINARY)
# Display the image
out = cv2.hconcat([img,thresh])
cv2.imshow('threshold',out)
```

좌측 : 원본 이미지, 우측 : threshold



2. Approximate the background of the sample image by a 2nd order polynomial surface and then display it as an image

```python
# approximate the background
xs = np.arange(0, img.shape[1])
ys = np.arange(0, img.shape[0])
x, y = np.meshgrid(xs, ys)
pos = np.dstack((x, y))
pos = pos.reshape(-1, 2)
A = []
I = []
mean_x = pos[:, 0].mean()
std_x = pos[:, 0].std()
mean_y = pos[:, 1].mean()
std_y = pos[:, 1].std()
norm_x = (pos[:, 0] - mean_x) / std_x
norm_y = (pos[:, 1] - mean_y) / std_y
A = np.vstack((norm_x, norm_y, norm_x * norm_y, norm_x ** 2, norm_y ** 2, np.ones_
I = np.array(img.reshape(-1, 1))
A_plus = np.linalg.inv((A.T @ A)) @ A.T
P = A_plus @ I
background = A @ P
background = background.reshape(img.shape)
```

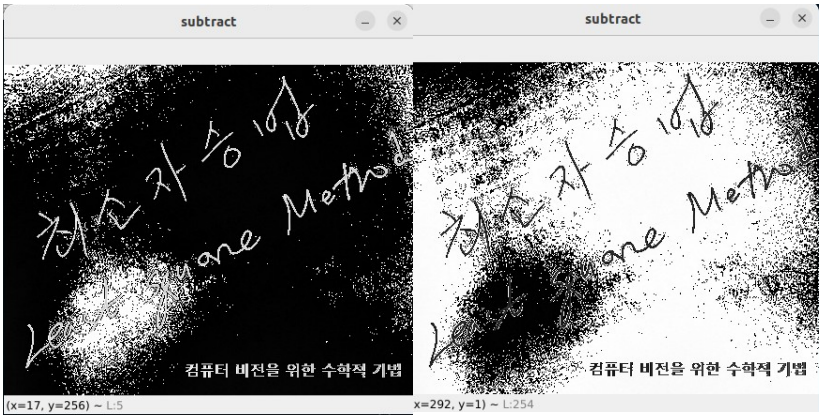


3. Subtract the approximated background image from the original and binarize the result (background-subtracted image) to obtain the final best binarized

error images

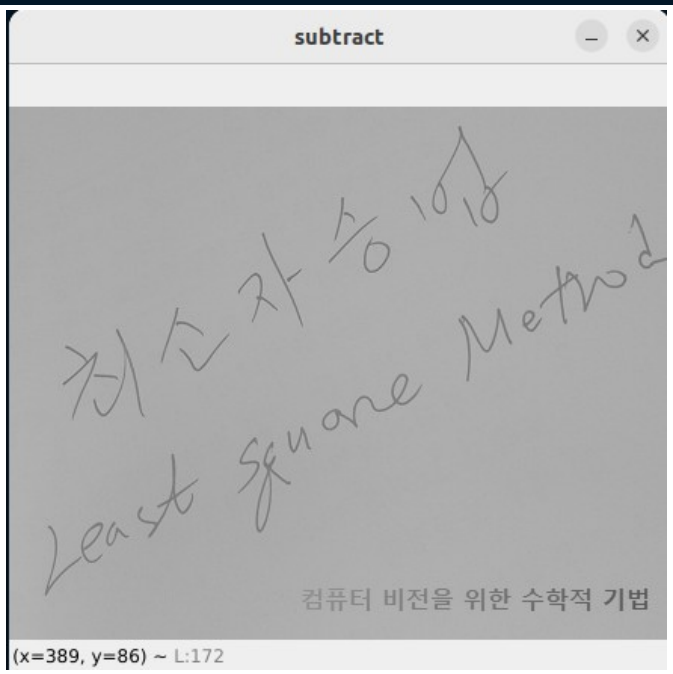
배경을 이미지에서 그대로 제거하였을 때 (원본, 색 반전)

```
# subtract the background
subtract = img - background

# Display the image
cv2.imshow('subtract', subtract.astype(np.uint8))
cv2.waitKey(0)
```



result

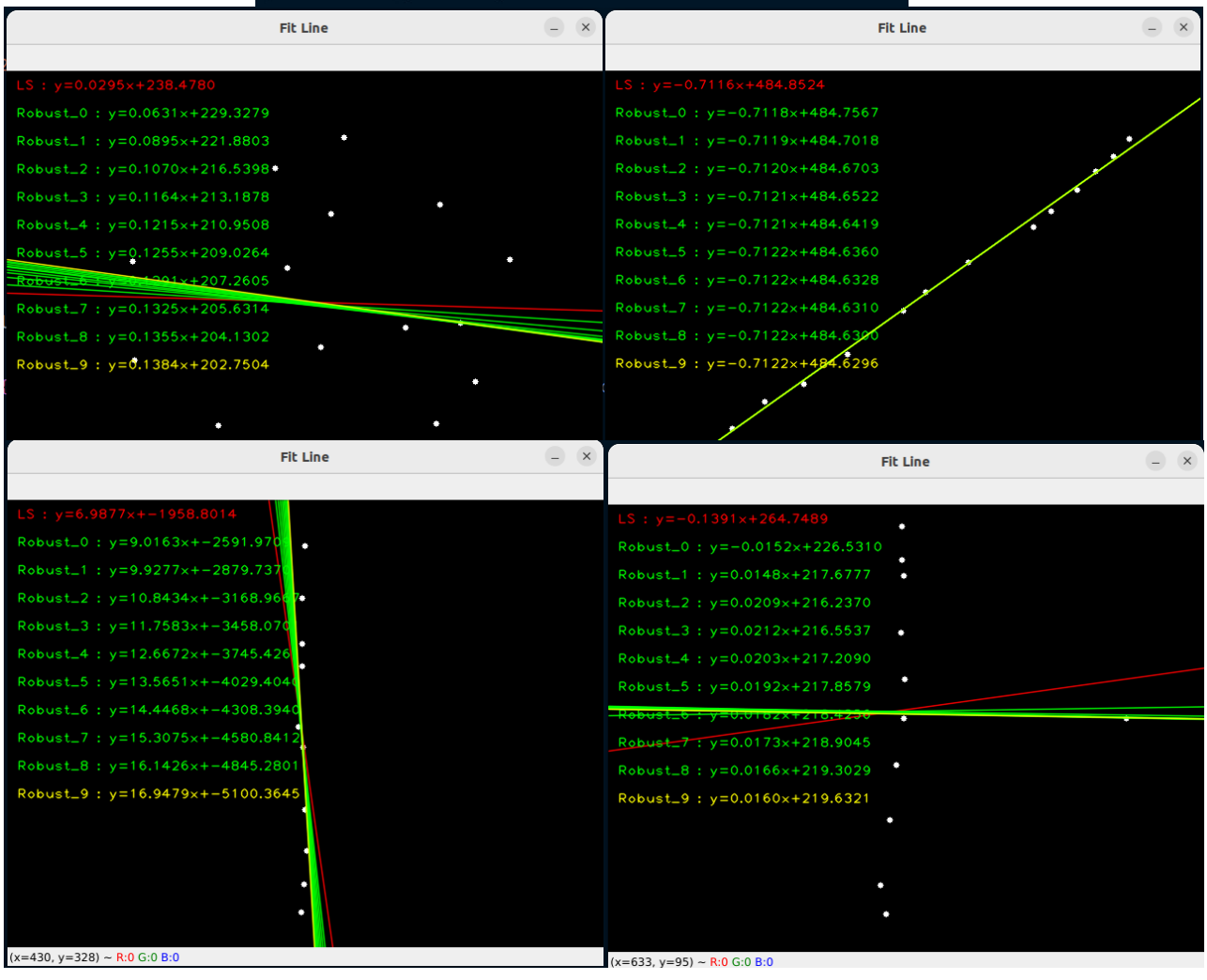
```
# subtract the background
subtract = img.astype(np.float32) - background.astype(np.float32)
subtract = subtract + subtract.min()
subtract = subtract.astype(np.uint8)
```

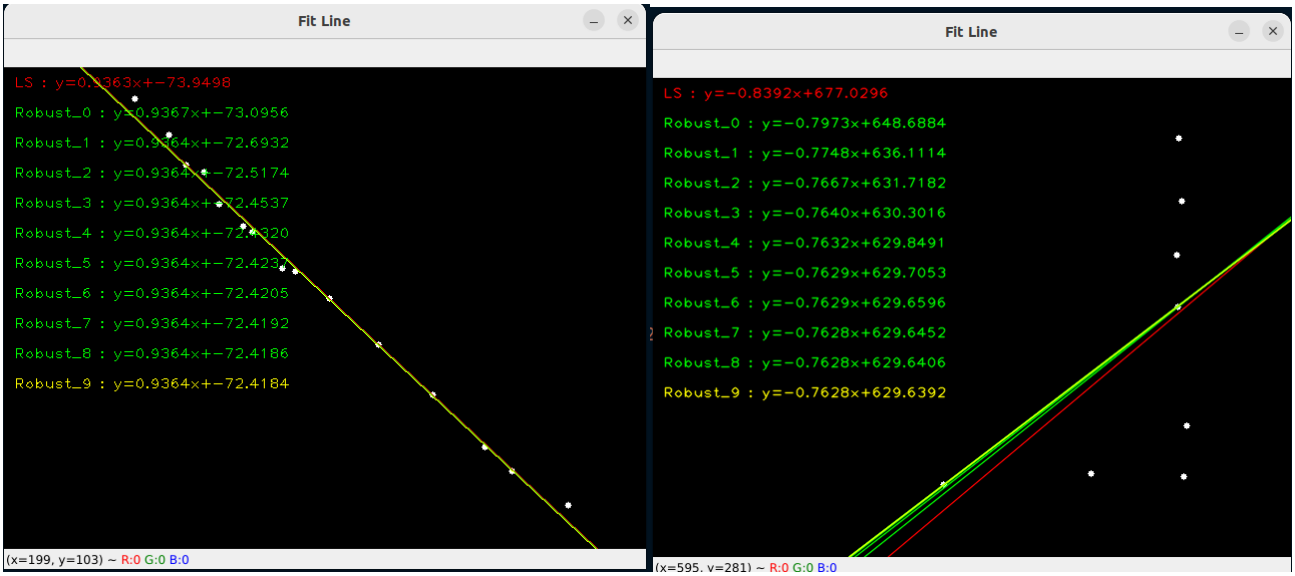


Q2. 사용자로부터 이상점(outlier)이 포함된 n(n≥2)개의 점을 입력받은 후 입력 점들의 근사 직선을 구하여 도시하는 프로그램을 작성하시오.

- 단, Cauchy weight function을 이용한 iterative weighted least square 방법을 이용하여 직선을 구하시오 (매 iteration에서 직선이 어떻게 변해가는지 확인하고 그 과정을 캡쳐하여 리포트에 포함)
- 일반적인 LS 방법을 이용한 직선 근사와 결과 비교 (색상을 다르게 표시)

```python
def robust_LS(A,y,iter=10) :
    A_plus = np.linalg.inv(A.T @ A) @ A.T
    p = A_plus @ y

    params =[]
    points = []
    for _ in range(iter) :
        r = y - A @ p
        W = np.diag(1 / (np.abs(r)/1.3998 + 1))
        p = np.linalg.inv(A.T @ W @ A) @ A.T @ W @ y
        a, b = p
        x1, x2 = 0, 640*2
        y1, y2 = int(a*x1+b), int(a*x2+b)
        params.append(p)
        points.append(((x1,y1), (x2, y2)))

    return params, points
```





결론

과제를 수행하며 image binary, global thresholding 등에 대해 찾아볼 수 있었다. 요즘 스마트폰에서도 그림자를 제거하는 기능이 많이 적용되어 있는데 생각보다 간단하게 구현해 볼 수 있음을 알게 되었다. 평소 포토샵 프로그램을 종종 사용하곤 하는데 좀 더 복잡한 작업이 필요했다. 앞으로 사용할 일이 또 생기게 된다면 위 방법을 활용할 생각이다.

이전 특이값 분해를 배워 적용하였던 과제와 동일하게 근사 직선을 구할 때 점들을 가로로 찍었을 때에는 결과가 잘 출력되는 것을 볼 수 있었지만, 세로로 분포하였을 경우에는 outlier 에 취약한 것이 보였다.
이 또한 3 차원 공간에서 수행의 필요성인지에 대해 궁금해졌다.