24-678: Computer Vision for Engineers
Ryan Wu
ID: weihuanw
PS3 Report
Due: Sep 29 2023

This file contains the following:
PS3-1 Image improvement via area-to-pixel filters

- pcb-improved.png
- golf-improved.png
- pots-improved.png
- rainbow-improved.png
- readme.txt
- source code file(s) (attached to the end)

PS3-1 Edge detection

- cheerios-sobel.png, cheerios-canny.png
- professor-sobel.png, professor-canny.png
- gear-sobel.png, gear-canny.png
- circuit-sobel.png, circuit-canny.png
- readme.txt
- source code file(s) (attached to the end)

Using 1 late day for this assignment

PS3-1 Information on filter combinations used

Median filter:

- kernel size: 5

```
smoothed_image = cv2.medianBlur(input_image, 5)
```

Figure 1. Code used for median filtering.

Bilateral filter:

- pixel value: 9 by 9 neighborhood
- $\sigma_1 = \sigma_2$: 75

```
smoothed_image = cv2.bilateralFilter(input_image, d=9, sigmaColor=75, sigmaSpace=75)
```

Figure 2. Code used for bilateral filtering.

Sharpening filter:

- kernel size 1
- $matrix \ = kernal\ size\ (1) \times \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$

```
kernel = 1
sharpening_kernel = kernel * np.array([[-1, -1, -1], [-1,  9, -1], [-1, -1, -1]])
sharpened_image = cv2.filter2D(smoothed_image, -1, sharpening_kernel)
```

Figure 3. Code used for sharpening filtering.

PS3-1 PCB image (filter combination in order: median & sharpening)

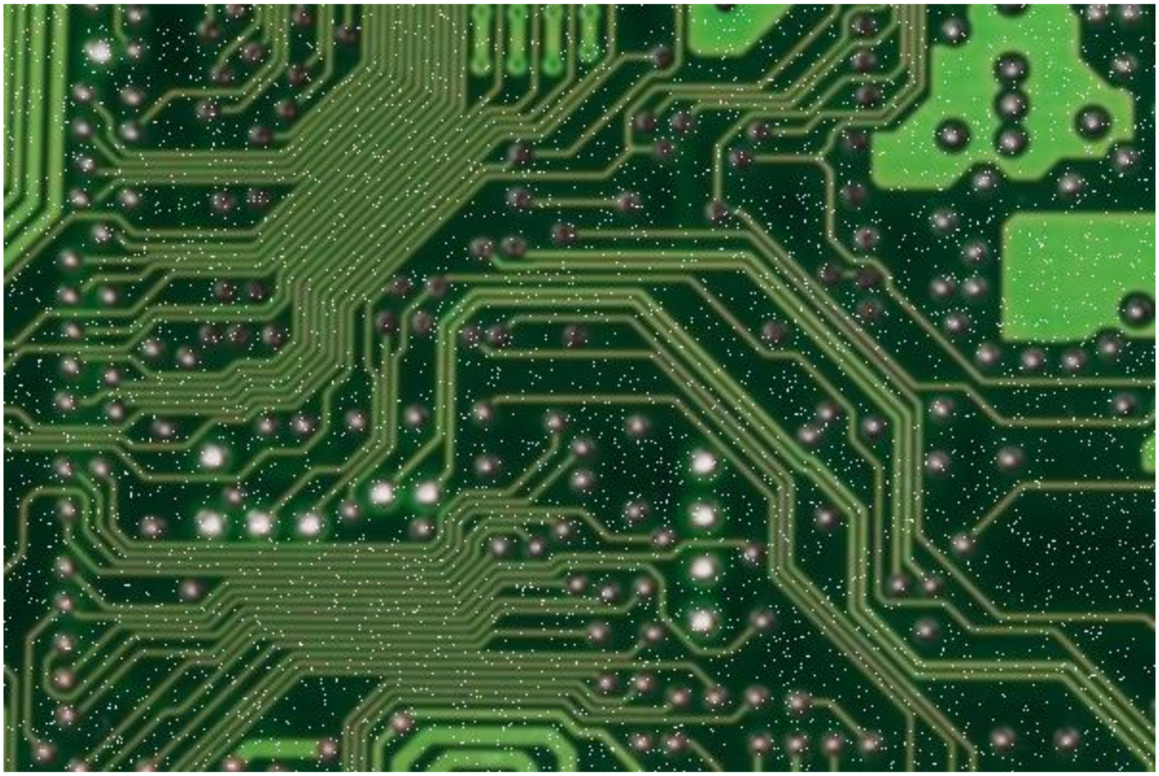

Figure 4. The given PCB image without filtering.

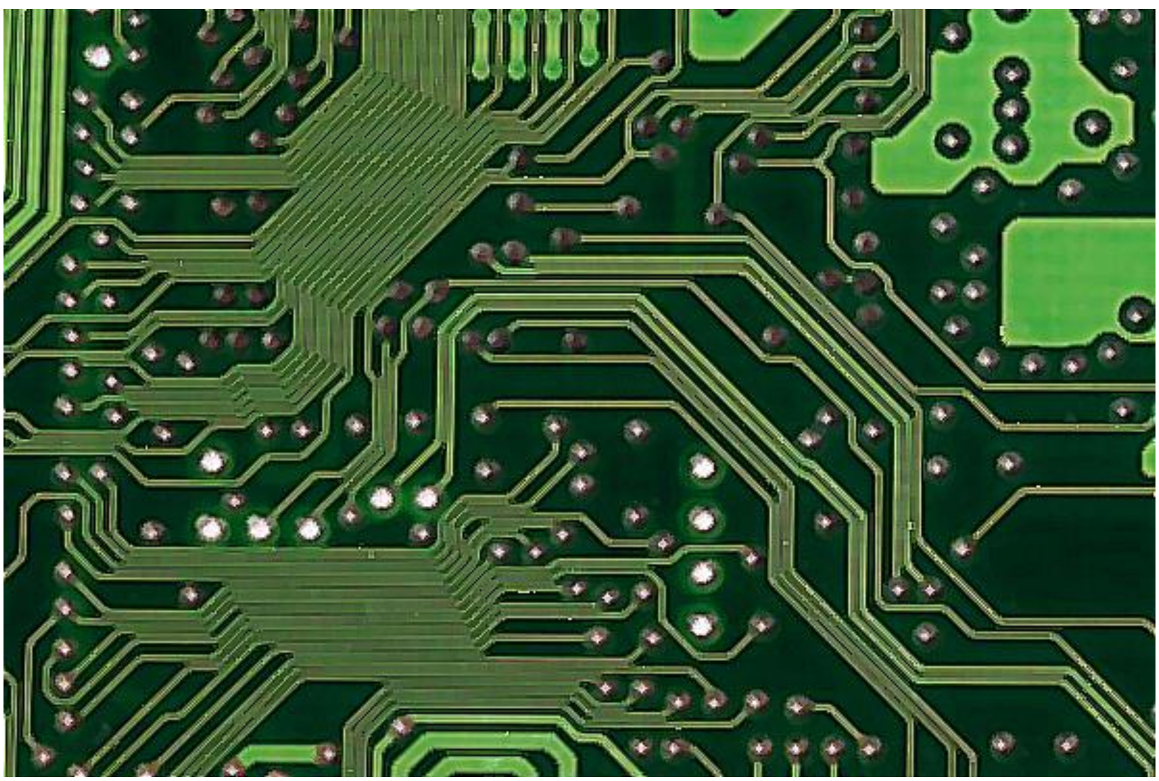

Figure 5. The improved PCB image with filters applied.

PS3-1 Golf image (filter combination in order: median & sharpening)

Figure 6. The given golf image without filtering.

Figure 7. The improved golf image with filters applied.

PS3-1 Pots image (Filter combination in order: median & sharpening)

Figure 8. The given pots image without filtering.

Figure 9. The improved pots image with filters applied.

PS3-1 Rainbow image (filter combination in order: bilateral & sharpening)

Figure 10. The given rainbow image without filtering.

Figure 11. The improved rainbow image with filters applied.

PS3-1 readme.txt

24-678: Computer Vision for Engineers
Ryan Wu
ID: weihuanw
PS3-1 Image improvement via area-to-pixel filters

Operating system: macOS Ventura 13.5.2
IDE you used to write and run your code: PyCharm 2023.1.4 (Community Edition)
The number of hours you spent to finish this problem: 6 hours.

PS3-2 Information on edge detection method used

Sobel:

- Horizontal Sobel matrix: $\frac{1}{16} \times \begin{bmatrix} -1 & -2 & 0 & 2 & 1 \\ -2 & -4 & 0 & 4 & 2 \\ -3 & -6 & 0 & 6 & 3 \\ -2 & -4 & 0 & 4 & 2 \\ -1 & -2 & 0 & 2 & 1 \end{bmatrix}$
- Vertical Sobel matrix: $\frac{1}{16} \times \begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 4 & 6 & 4 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ -2 & -4 & -6 & -4 & -2 \\ -1 & -2 & -3 & -2 & -1 \end{bmatrix}$

```python
def sobel_filter(grayscale_image_sobel):
    sobel_horizontal = 1 / 16 * np.array([[-1, -2, 0, 2, 1], [-2, -4, 0, 4, 2], [-3, -6, 0, 6, 3], [-2, -4, 0, 4, 2], [-1, -2, 0, 2, 1]])
    sobel_vertical = 1 / 16 * np.array([[1, 2, 3, 2, 1], [2, 4, 6, 4, 2], [0, 0, 0, 0, 0], [-2, -4, -6, -4, -2], [-1, -2, -3, -2, -1]])

    edge_horizontal = cv2.filter2D(grayscale_image_sobel, cv2.CV_64F, sobel_horizontal)
    edge_vertical = cv2.filter2D(grayscale_image_sobel, cv2.CV_64F, sobel_vertical)

    edge_magnitude = np.sqrt(edge_horizontal ** 2 + edge_vertical ** 2)
    edge_direction = np.arctan2(edge_vertical, edge_horizontal)

    return edge_magnitude, edge_direction
```

Figure 12. Code used for Sobel edge detection.

Canny edge:

```python
def canny_edge_filter(grayscale_image_canny, threshold1, threshold2, aperture_size, l2_gradient):
    aperture_size = max(3, min(aperture_size, 7))
    aperture_size = aperture_size if aperture_size % 2 != 0 else aperture_size - 1

    canny_edge = cv2.Canny(grayscale_image_canny, threshold1, threshold2, apertureSize=aperture_size, L2gradient=l2_gradient)
    negate_canny_edge_image = 255 - canny_edge

    cv2.imshow('Canny Edges', negate_canny_edge_image)

    return negate_canny_edge_image
```

Figure 13. Code used for Canny edge detection.

Findings and discussion:

In all the below comparisons, both Sobel and Canny edge detection method were used on each given image. The result shows that Sobel gives a slightly thicker edge border and also provides gradient information on both the horizontal and the vertical directions. However, Canny edge detection offers high-quality, well-localized edges and also reduces noise.

I would recommend using Sobel method when tasked with simpler image inputs that requires fast results. On the other hand, I would recommend using Canny edge method for more professional edge detection task.

PS3-2 Cheerios (Sobel and canny edge detection)

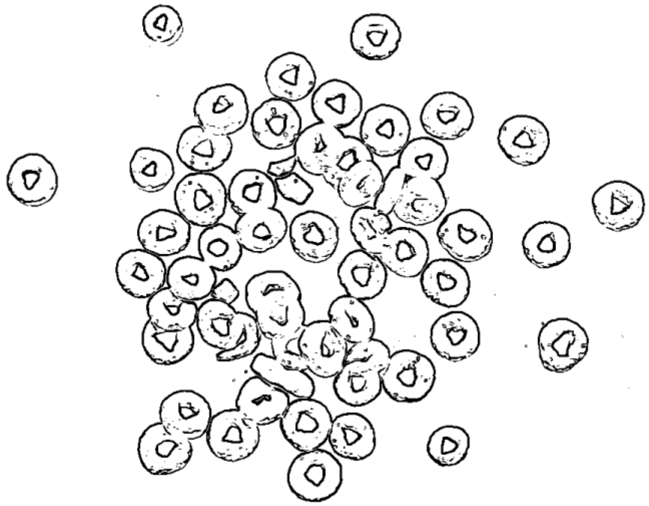

Figure 14. Cheerios binary image (Threshold: 195) with Sobel filter applied.

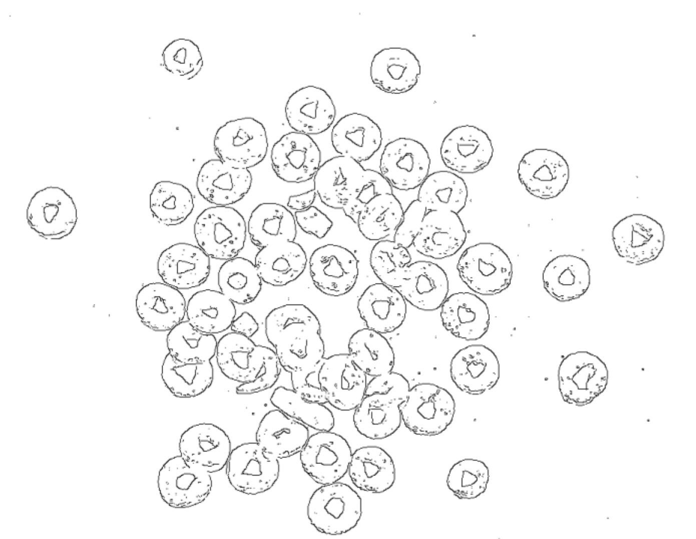

Figure 15. Cheerios image with canny edge detection applied. (Threshold 1: 255, Threshold2: 255, Aperture: 3, and using L2)

In Figure 14 and Figure 15, the given cheerios image is a mixed of simple and complex image structure with several edges. In this case, it will depend on your computational power and availability in using Sobel or Canny edge detection.

PS3-2 Professor image (Sobel and canny edge detection)

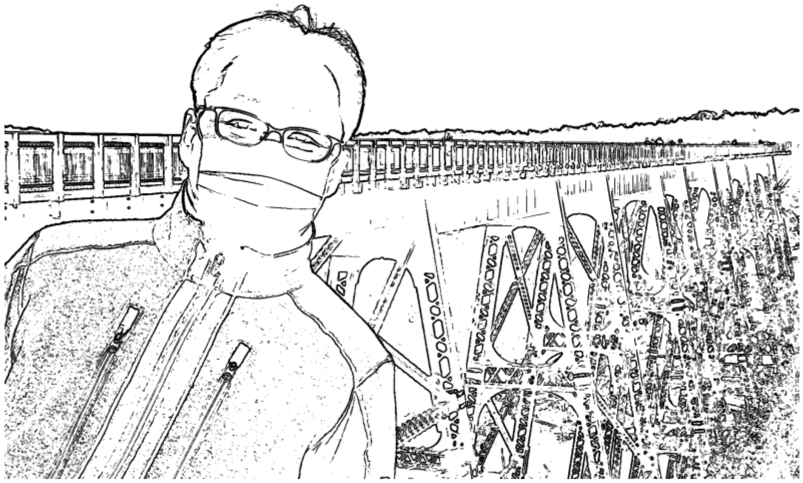

Figure 16. Professor binary image (Threshold: 220) with Sobel filter applied.

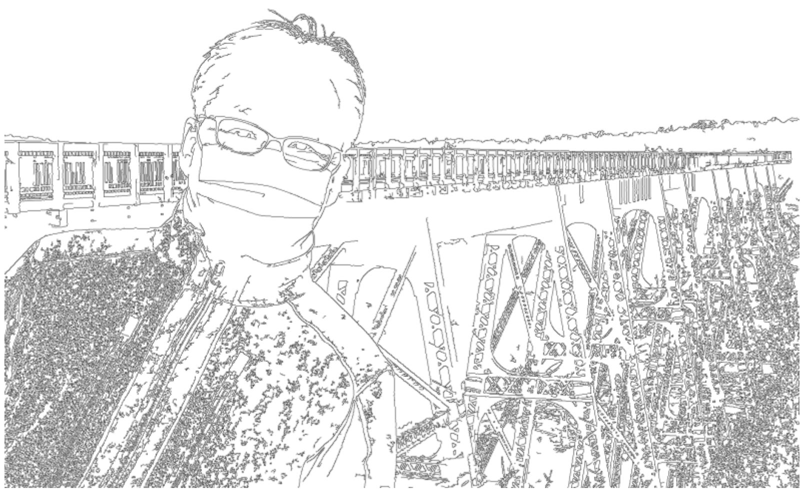

Figure 17. Professor image with canny edge detection applied. (Threshold 1: 150, Threshold 2: 50, Aperture: 3, and not using L2)

In Figure 16 and Figure 17, the given professor image can be considered a more complex image with many edges. In this case, the Canny edge detection would be the recommend method.

PS3-2 Gear image (Sobel and canny edge detection)

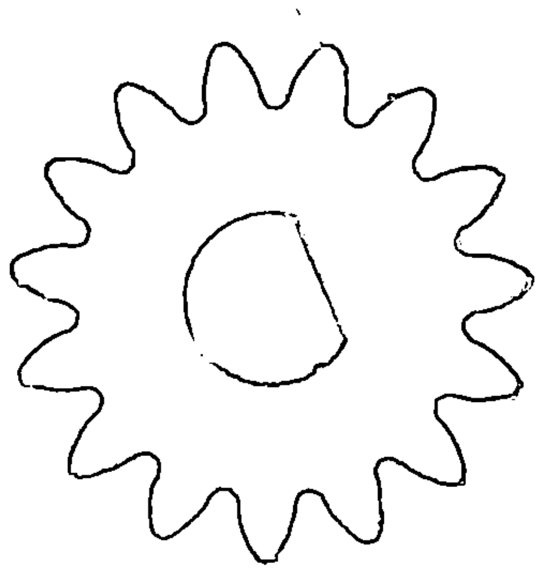

Figure 18. Gear binary image (Threshold: 155) with Sobel edge detection applied.

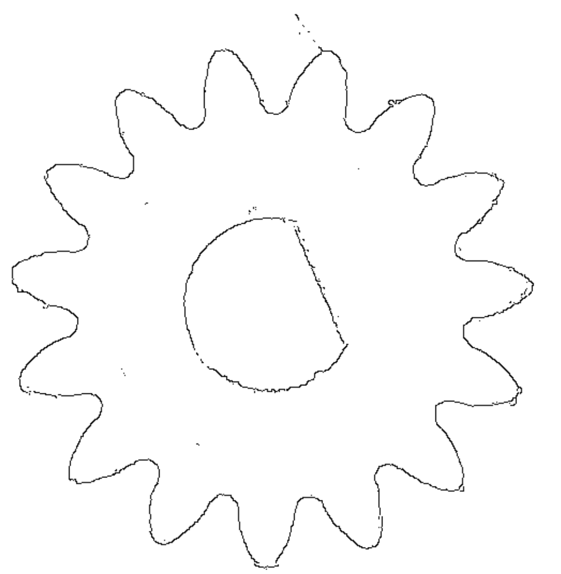

Figure 19. Gear image with canny edge detection applied. (Threshold 1: 255, Threshold 1: 255, Aperture: 3, and not using L2)

In Figure 18 and Figure 19, the given gear image can be considered a simpler image with less edges. In this case, the Sobel edge detection method would be sufficient.

PS3-2 Circuit image (Sobel and canny edge detection)

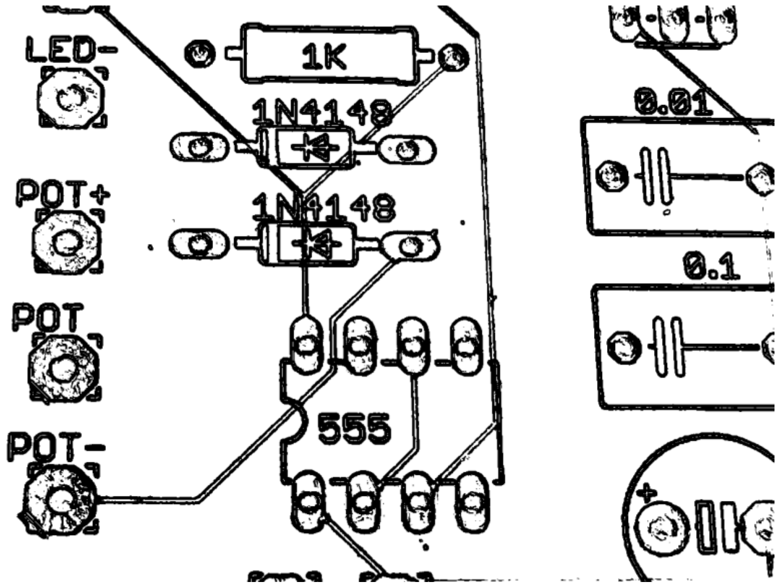

Figure 20. Circuit binary image (Threshold: 240) with Sobel filter applied.

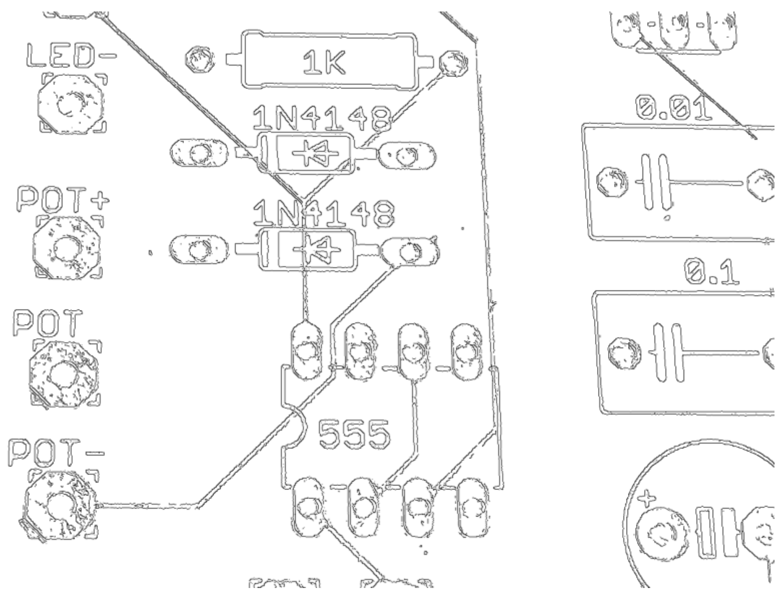

Figure 21. Circuit image with canny edge detection applied. (Threshold 1: 90, Threshold 2: 60, Aperture: 3, and not using L2)

In Figure 20 and Figure 21, the given circuit image can be considered a more complex image with many edges. In this case, the Canny edge detection would be the recommend method.

PS3-2 readme.txt

24-678: Computer Vision for Engineers
Ryan Wu
ID: weihuanw
PS3-2 Edge detection

Operating system: macOS Ventura 13.5.2
IDE you used to write and run your code: PyCharm 2023.1.4 (Community Edition)
The number of hours you spent to finish this problem: 6 hours.

```python
## PS3-1 Image Improvement via area-to-pixel filers
import cv2
import numpy as np
import os

# User input feature
user_input = input("Please name your input color file: ")
file_directory = os.getcwd()
image_location = os.path.join(file_directory, user_input)
if os.path.exists(image_location):
    print(f"Your '{user_input}' image loaded successfully.")
    input_image = cv2.imread(user_input)
    cv2.imshow(f"'{user_input}'", input_image)
    cv2.waitKey(0)
else:
    print(f"Error: unable to load your input image.\nPlease make sure '{user_input}' is in the correct directory.")
    exit()

# Filtering process
input_image = cv2.imread(user_input)
# Different filtering combination for rainbow (bilateral+sharpening)
if user_input == 'rainbow.png':
    # smoothed_image = cv2.GaussianBlur(input_image, (5, 5), 0)
    smoothed_image = cv2.bilateralFilter(input_image, d=9, sigmaColor=75, sigmaSpace=75)
    kernel = 1
    sharpening_kernel = kernel * np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])
    sharpened_image = cv2.filter2D(smoothed_image, -1, sharpening_kernel)

# Different filtering combination for all other images (median+sharpening)
else:
```

```
    smoothed_image = cv2.medianBlur(input_image, 5)
    kernel = 1
    sharpening_kernel = kernel * np.array([[-1, -1, -1], [-1,  9, -1], [-1, -1, -1]])
    sharpened_image = cv2.filter2D(smoothed_image, -1, sharpening_kernel)

# Saving the output image
cv2.imshow(f"'{user_input}'", sharpened_image)
output_image = user_input.split('.')[0] + '-improved.' + user_input.split('.')[-1]
cv2.waitKey(0)
cv2.imwrite(output_image, sharpened_image)

cv2.destroyAllWindows()
```

```python
# PS3-2 Edge detection
import cv2
import numpy as np
import os

# Sobel Filter Function
def sobel_filter(grayscale_image_sobel):
    sobel_horizontal = 1 / 16 * np.array([[-1, -2, 0, 2, 1], [-2, -4, 0, 4, 2], [-3, -6, 0, 6, 3], [-2, -4, 0, 4, 2], [-1, -2, 0, 2, 1]])
    sobel_vertical = 1 / 16 * np.array([[1, 2, 3, 2, 1], [2, 4, 6, 4, 2], [0, 0, 0, 0, 0], [-2, -4, -6, -4, -2], [-1, -2, -3, -2, -1]])

    edge_horizontal = cv2.filter2D(grayscale_image_sobel, cv2.CV_64F, sobel_horizontal)
    edge_vertical = cv2.filter2D(grayscale_image_sobel, cv2.CV_64F, sobel_vertical)

    edge_magnitude = np.sqrt(edge_horizontal ** 2 + edge_vertical ** 2)
    edge_direction = np.arctan2(edge_vertical, edge_horizontal)

    return edge_magnitude, edge_direction

# Canny Edge Filter Function
def canny_edge_filter(grayscale_image_canny, threshold1, threshold2, aperture_size, l2_gradient):
    aperture_size = max(3, min(aperture_size, 7))
    aperture_size = aperture_size if aperture_size % 2 != 0 else aperture_size - 1

    canny_edge = cv2.Canny(grayscale_image_canny, threshold1, threshold2, apertureSize=aperture_size, L2gradient=l2_gradient)
    negate_canny_edge_image = 255 - canny_edge

    cv2.imshow('Canny Edges',
```

```python
negate_canny_edge_image)

    return negate_canny_edge_image

# main script with user input feature
user_input = input("Please name your input color file: ")
file_directory = os.getcwd()
image_location = os.path.join(file_directory, user_input)
if os.path.exists(image_location):
    print(f"Your '{user_input}' image loaded successfully.")
    input_image = cv2.imread(user_input)
    cv2.imshow(f"'{user_input}'", input_image)
    cv2.waitKey(0)

    # Executing Sobel filter function
    grayscale_image_sobel = cv2.cvtColor(input_image, cv2.COLOR_BGR2GRAY)
    edge_magnitude, edge_direction = sobel_filter(grayscale_image_sobel)

    max_edge_magnitude = np.max(edge_magnitude)
    min_edge_magnitude = np.min(edge_magnitude)

    if max_edge_magnitude != min_edge_magnitude:
        edge_magnitude_normalized = 255 * (edge_magnitude - min_edge_magnitude) / (
                    max_edge_magnitude - min_edge_magnitude)
    else:
        edge_magnitude_normalized = edge_magnitude

    edge_magnitude_normalized = edge_magnitude_normalized.astype(np.uint8)
    negate_sobel_image = 255 - edge_magnitude_normalized

    # Converting grayscale into binary image
    if user_input == 'cheerios.png':
```

```
            threshold_value = 195
        elif user_input == 'professor.png':
            threshold_value = 220
        elif user_input == 'gear.png':
            threshold_value = 155
        else:
            threshold_value = 240

        _, binary_sobel_image = cv2.threshold(negate_sobel_image, threshold_value, 255, cv2.THRESH_BINARY)

        # Showing and saving Sobel filtered results
        cv2.imshow(f"'{user_input}'", binary_sobel_image)
        output_image_sobel = user_input.split('.')[0] + '-sobel.' + user_input.split('.')[-1]
        cv2.imwrite(output_image_sobel, binary_sobel_image)
        cv2.waitKey(0)

        # Executing Canny edge filter function
        grayscale_image_canny = cv2.cvtColor(input_image, cv2.COLOR_BGR2GRAY)

        # Canny edge GUI
        cv2.namedWindow('Canny Edge GUI')
        cv2.createTrackbar('Threshold1', 'Canny Edge GUI', 0, 255, lambda x: None)
        cv2.createTrackbar('Threshold2', 'Canny Edge GUI', 0, 255, lambda x: None)
        cv2.createTrackbar('Aperture Size', 'Canny Edge GUI',3, 7, lambda x: None)
        cv2.createTrackbar('L2 Gradient', 'Canny Edge GUI', 0, 1, lambda x: None)

        while True:
            threshold1 = cv2.getTrackbarPos('Threshold1', 'Canny Edge GUI')
            threshold2 = cv2.getTrackbarPos('Threshold2', 'Canny Edge GUI')
```

```python
        aperture_size = cv2.getTrackbarPos('Aperture Size', 'Canny Edge GUI')
        l2_gradient = cv2.getTrackbarPos('L2 Gradient', 'Canny Edge GUI')

        canny_edge_result = canny_edge_filter(grayscale_image_canny, threshold1, threshold2, aperture_size, l2_gradient)
        negate_canny_edge_image = 255 - canny_edge_result
        cv2.imshow('Canny Edges', canny_edge_result)

        key = cv2.waitKey(1) & 0xFF
        if key == ord(' '):  # If the space key is pressed
            output_image_canny = user_input.split('.')[0] + '-canny.' + user_input.split('.')[-1]
            cv2.imwrite(output_image_canny, canny_edge_result)
            print(f"Canny edge filter image saved as: {output_image_canny}")
            break
    cv2.destroyAllWindows()

else:
    print(f"Error: unable to load your input image.\nPlease make sure '{user_input}' is in the correct directory.")
    exit()
```