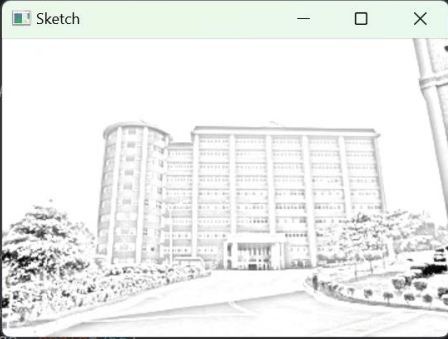


Midterm

Question 1

```
Midterm1.py x filter.py Midterm2.py mid3.py try.py
1 import cv2
2
3 # Load the image
4 image = cv2.imread('Lib2.jpg')
5
6 # Convert the image to grayscale
7 gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
8
9 # Invert the grayscale image
10 inverted_gray = 255 - gray_image
11
12 # Apply Gaussian blur
13 blurred_image = cv2.GaussianBlur(inverted_gray, (21, 21), 0)
14
15 # Blend the grayscale image with the blurred image
16 sketch = cv2.divide(gray_image, 255 - blurred_image, scale=255)
17
18 # Display the sketch
19 cv2.imshow('Sketch', sketch)
20 cv2.waitKey(0)
21 cv2.destroyAllWindows()
```



import cv2

Load the image

image = cv2.imread('Lib2.jpg')

Convert the image to grayscale

gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

Invert the grayscale image

inverted_gray = 255 - gray_image

Apply Gaussian blur

blurred_image = cv2.GaussianBlur(inverted_gray, (21, 21), 0)

Blend the grayscale image with the blurred image

```
sketch = cv2.divide(gray_image, 255 - blurred_image, scale=256)
```

```
# Display the sketch
```

```
cv2.imshow('Sketch', sketch)
```

```
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

brief discussion

Image Loading: The code starts by loading an image named 'Lib2.jpg' using the `cv2.imread()` function. This function reads the image as a NumPy array.

Preprocessing:

Grayscale Conversion: The loaded image is converted to grayscale using `cv2.cvtColor()` with the parameter `cv2.COLOR_BGR2GRAY`. This simplifies the image to a single channel, representing intensity.

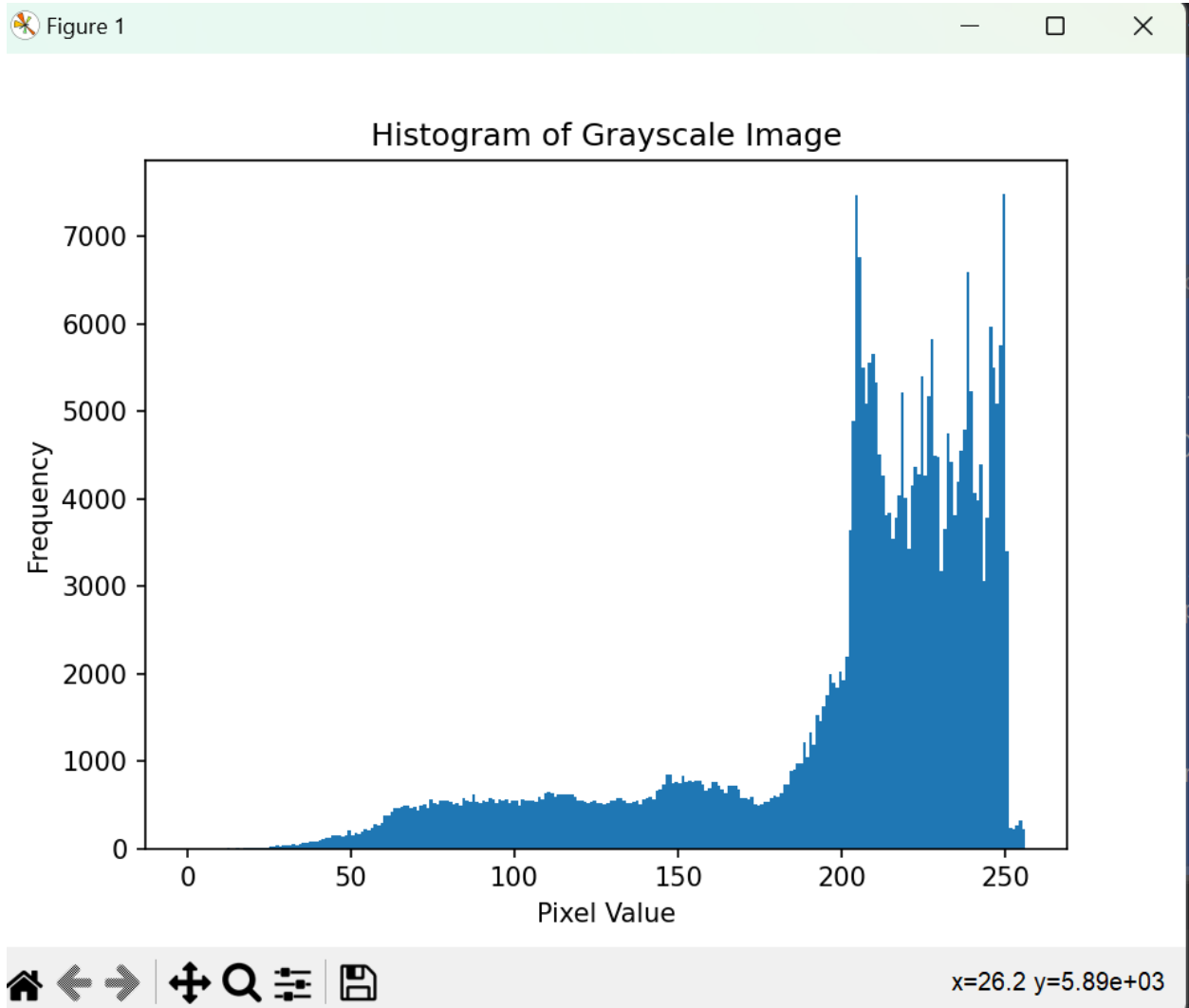
Inversion: The grayscale image is inverted by subtracting each pixel value from 255. This reversal enhances the sketch effect.

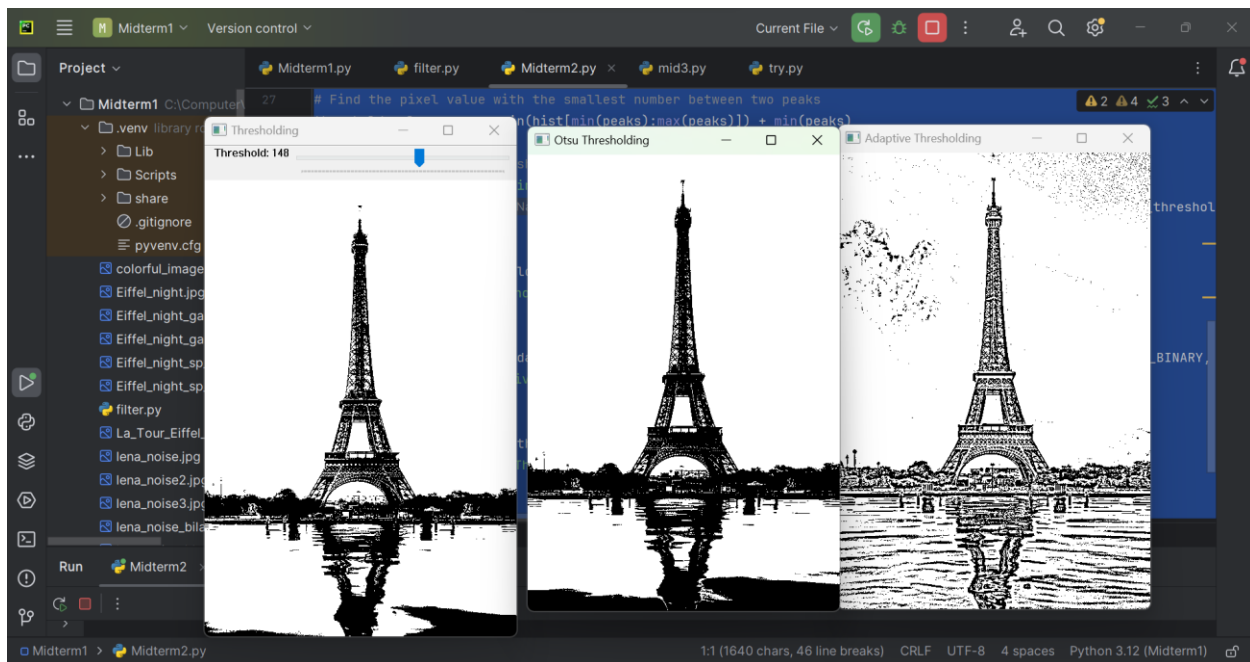
Gaussian Blur: A Gaussian blur is applied to the inverted grayscale image using `cv2.GaussianBlur()`. This smooths out the image and reduces noise. The kernel size (21, 21) specifies the amount of blurring.

Sketch Generation: The inverted grayscale image and the blurred image are blended together using `cv2.divide()`. This operation emphasizes the edges and creates the sketch effect. The scale parameter is set to 256 to prevent division by zero.

Display: The resulting sketch image is displayed using `cv2.imshow()`. The program waits for a key press with `cv2.waitKey(0)` and then closes all windows with `cv2.destroyAllWindows()`.

Question 2





```
import cv2

import numpy as np

import matplotlib.pyplot as plt

# Function to update threshold value using trackbar
def update_threshold_value(value):

    global threshold_value

    threshold_value = value

    binary_image = cv2.threshold(gray_image, threshold_value, 255, cv2.THRESH_BINARY)[1]

    cv2.imshow('Thresholding', binary_image)

# Load the image
image = cv2.imread('La_Tour_Eiffel_480x700.png')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Plot histogram
plt.hist(gray_image.ravel(), bins=256, range=[0, 256])
```

```
plt.title('Histogram of Grayscale Image')
plt.xlabel('Pixel Value')
plt.ylabel('Frequency')
plt.show()
```

```
# Find peaks in the histogram
hist, bins = np.histogram(gray_image.flatten(), 256, [0, 256])
peaks = np.where(np.diff(np.sign(np.diff(hist))) < 0)[0] + 1
```

```
# Find the pixel value with the smallest number between two peaks
threshold_value = np.argmin(hist[min(peaks):max(peaks)]) + min(peaks)
```

```
# Create trackbar for threshold value
cv2.namedWindow('Thresholding')
cv2.createTrackbar('Threshold', 'Thresholding', threshold_value, 255, update_threshold_value)
```

```
# Initial thresholding
binary_image = cv2.threshold(gray_image, threshold_value, 255, cv2.THRESH_BINARY)[1]
cv2.imshow('Thresholding', binary_image)
```

```
# Adaptive thresholding
adaptive_threshold = cv2.adaptiveThreshold(gray_image, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY, 11, 2)
cv2.imshow('Adaptive Thresholding', adaptive_threshold)
```

```
# Otsu's thresholding
ret, otsu_threshold = cv2.threshold(gray_image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
cv2.imshow('Otsu Thresholding', otsu_threshold)
```

```
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

brief discussion

Image Processing: The code loads an image and converts it to grayscale.

Histogram Analysis: It plots the histogram of the grayscale image to analyze pixel intensity distribution.

Thresholding: A trackbar allows dynamic adjustment of the threshold value for simple thresholding.

Initial thresholding highlights regions above the calculated threshold value.

Adaptive thresholding automatically determines the threshold value based on local regions.

Otsu's thresholding finds an optimal threshold value based on the image histogram.

Display: The resulting binary images from each thresholding method are displayed.

The code waits for a key press to keep the windows open, and closes them after a key press.

This main function demonstrates the application of thresholding techniques for enhancing features in the image, particularly focusing on making the Eiffel Tower prominent.



```
import cv2

import numpy as np

import matplotlib.pyplot as plt


# Function to process and display an image
def process_and_display_image(img_name):

    # Load the image
    img = cv2.imread(img_name, cv2.IMREAD_GRAYSCALE)


    # Histogram Equalization
    img_eq = cv2.equalizeHist(img)


    # Median Filter
    img_median = cv2.medianBlur(img_eq, 5)


    # Gaussian Blur
    img_gaussian = cv2.GaussianBlur(img_median, (5, 5), 0)


    # Canny Edge Detection
    img_edges = cv2.Canny(img_gaussian, 100, 200)


    # Display images
    fig, axs = plt.subplots(1, 5, figsize=(15, 3))


    # Display specific image name
    axs[0].imshow(img, cmap='gray')
    axs[0].axis('off')
    axs[0].set_title(img_name)
```



```
# Display processed images

axs[1].imshow(img_eq, cmap='gray')
axs[1].axis('off')
axs[1].set_title('Histogram Equalization')


axs[2].imshow(img_median, cmap='gray')
axs[2].axis('off')
axs[2].set_title('Median Filter')


axs[3].imshow(img_gaussian, cmap='gray')
axs[3].axis('off')
axs[3].set_title('Gaussian Blur')


axs[4].imshow(img_edges, cmap='gray')
axs[4].axis('off')
axs[4].set_title('Canny Edge Detection')


plt.tight_layout()
plt.show()
```

```
# Process and display each image
img_names = [
    "Eiffel_night.jpg",
    "Eiffel_night_sp_noise.jpg",
    "Eiffel_night_gaussian_noise.jpg",
    "Eiffel_night_gaussian_sp_noise.jpg"
]
```

```
for img_name in img_names:  
    process_and_display_image(img_name)
```

brief discussion

Image Processing: The function loads an image specified by `img_name` using OpenCV's `cv2.imread()` function with grayscale mode (`cv2.IMREAD_GRAYSCALE`).

It applies histogram equalization to enhance the contrast of the image using `cv2.equalizeHist()`.

A median filter is applied using `cv2.medianBlur()` to reduce noise and smooth the image.

Gaussian blur is performed using `cv2.GaussianBlur()` to further reduce noise and detail.

Canny edge detection is applied using `cv2.Canny()` to detect edges in the image.

Display: The original image and the images after each processing step are displayed side by side using matplotlib's `plt.imshow()` function.

Subplots are created for each image, and titles are added to describe each processing step.

Loop Through Image Names: The main function iterates through a list of image names specified by `img_names`. For each image name, the `process_and_display_image()` function is called to process and display the image.