Step 1: Install OpenCV

```
!pip install opencv-python-headless
```

```
Requirement already satisfied: opencv-python-headless in /usr/local/lib/python3.
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-p
```

Step 2: Import Necessary Libraries

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Function to display an image using matplotlib
def display_image(img, title="Image"):
  plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
  plt.title(title)
  plt.axis('off')
  plt.show()

# Function to display two images side by side
def display_images(img1, img2, title1="Image 1", title2="Image 2"):
  plt.subplot(1, 2, 1)
  plt.imshow(cv2.cvtColor(img1, cv2.COLOR_BGR2RGB))
  plt.title(title1)
  plt.axis('off')
  plt.subplot(1, 2, 2)
  plt.imshow(cv2.cvtColor(img2, cv2.COLOR_BGR2RGB))
  plt.title(title2)
  plt.axis('off')
  plt.show()
```

cv2: This imports OpenCV, which provides functions for image processing. numpy (np): This library is used for handling arrays and matrices, which images are represented as. matplotlib.pyplot (plt): This is used to display images in a Jupyter notebook or Google Colab environment.

Step 3: Load an Image

```
from google.colab import files
from io import BytesIO
from PIL import Image
# Upload an image
```

```
# Upload an image
uploaded = files.upload()
# Convert to OpenCV format
image_path = next(iter(uploaded)) # Get the image file name
image = Image.open(BytesIO(uploaded[image_path]))
image = cv2.cvtColor(np.array(image), cv2.COLOR_RGB2BGR)
```

Choose Files piksurkopogi.jpeg

- **piksurkopogi.jpeg**(image/jpeg) - 43882 bytes, last modified: 9/16/2024 - 100% done

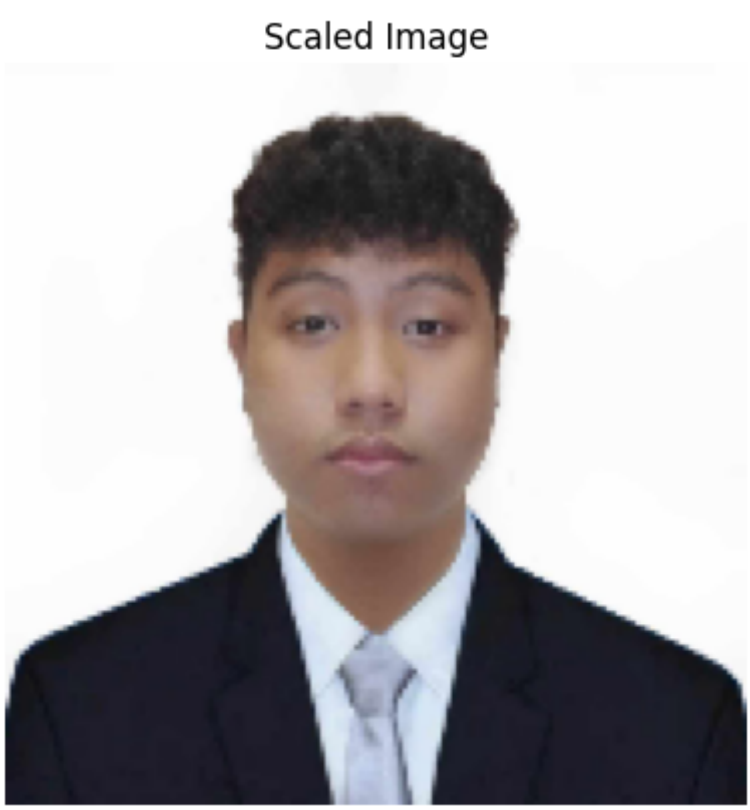
```
Saving piksurkopogi.jpeg to piksurkopogi (4).jpeg
```

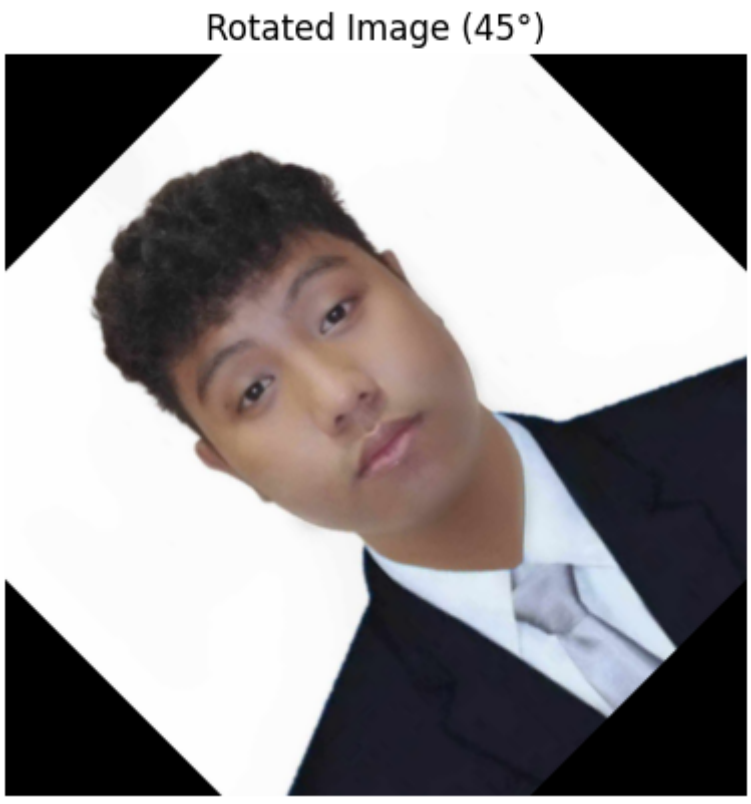
display_image(): Converts the image from BGR (OpenCV's default color format) to RGB (the format expected by matplotlib) and displays it using imshow(). display_images(): This function allows two images to be displayed side by side for comparison. We use subplot to create a grid of plots (here, 1 row and 2 columns).

Exercise 1: Scaling and Rotation

```
# Scaling
def scale_image(img, scale_factor):
  height, width = img.shape[:2]
  scaled_img = cv2.resize(img,
  (int(width * scale_factor), int(height * scale_factor)), interpolation=cv2.INTE

  return scaled_img
"""
scale_image(): This function scales the image by a given factor.
The cv2.resize() function takes the original dimensions of the image, multiplies
"""
# Rotate
def rotate_image(img, angle):
  height, width = img.shape[:2]
  center = (width // 2, height // 2)
  matrix = cv2.getRotationMatrix2D(center, angle, 1.0)
  rotated_img = cv2.warpAffine(img, matrix, (width, height))
  return rotated_img
"""
rotate_image(): Rotates the image around its center. cv2.getRotationMatrix2D() cr
"""
# Scale image by 0.5
scaled_image = scale_image(image, 0.1)
display_image(scaled_image, "Scaled Image")
# Rotate image by 45 degrees
rotated_image = rotate_image(image, 45)
display_image(rotated_image, "Rotated Image (45°)")

```

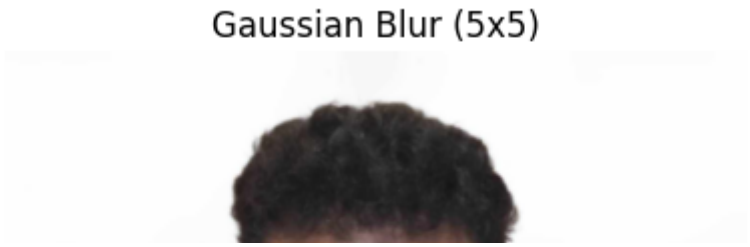
```
"""
These lines apply the scaling and rotation functions to the uploaded image and di
```





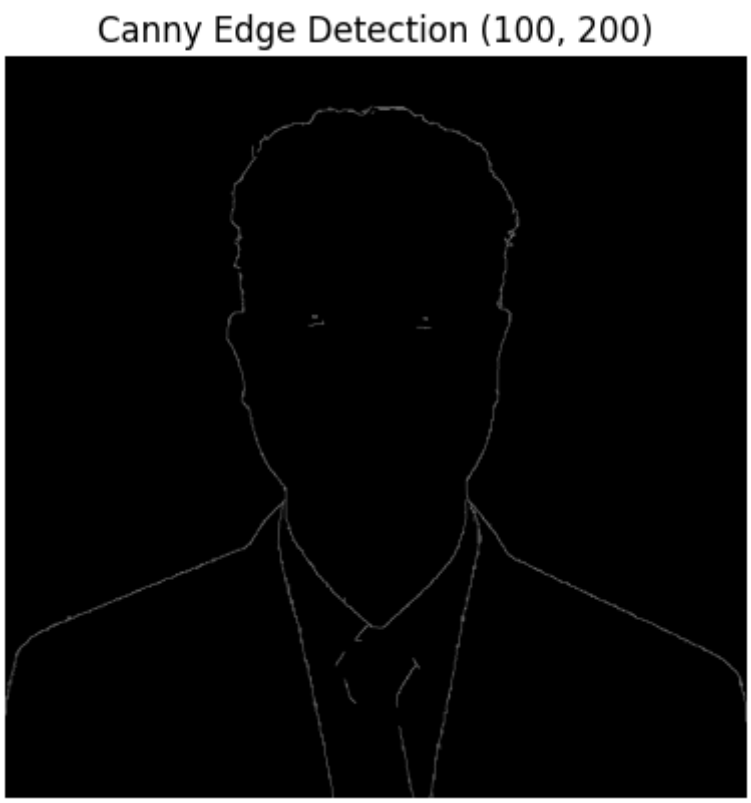
Exercise 2: Blurring Techniques

```
# Gaussian Blur
gaussian_blur = cv2.GaussianBlur(image, (5, 5), 0)
display_image(gaussian_blur, "Gaussian Blur (5x5)")
"""
cv2.GaussianBlur(): Applies a Gaussian blur to the image, which smooths it by ave
the pixel values in a 5x5 kernel (a small matrix). This is useful for reducing no
# Median Blur
median_blur = cv2.medianBlur(image, 5)
display_image(median_blur, "Median Blur (5x5)")
"""
cv2.medianBlur(): Applies a median blur, which replaces each pixel's value with t
"""
```



Exercise 3: Edge Detection using Canny

```
# Canny Edge Detection
edges = cv2.Canny(image, 100, 200)
display_image(edges, "Canny Edge Detection (100, 200)")
"""
cv2.Canny(): Detects edges in the image by calculating the gradient (rate of inte
sensitivity. Lower thresholds detect more edges, while higher thresholds detect o
most prominent edges.
"""

```

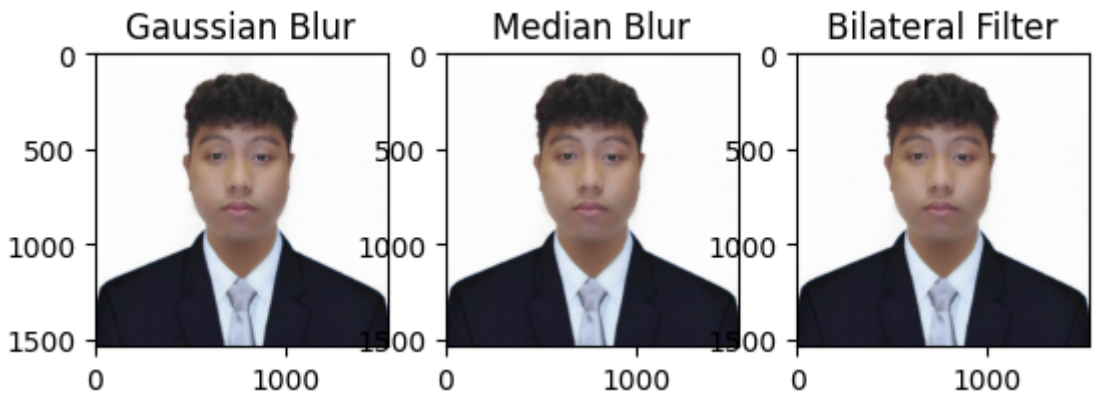



```
'\ncv2.Canny(): Detects edges in the image by calculating the gradient (rate of
intensity change) between pixels. The two threshold values (100 and 200) define
the edges'\nsensitivity. Lower thresholds detect more edges, while higher thresh
y effective in removing salt-and-pepper noise.\n'
```

Exercise 4: Basic Image Processor (Interactive)

Exercise 5: Comparison of Filtering Techniques

```
# Applying Gaussian, Median, and Bilateral filters gaussian_blur = cv2.GaussianBl
bilateral_filter = cv2.bilateralFilter(image, 9, 75, 75)
"""
cv2.bilateralFilter(): This filter smooths the image while keeping edges sharp, u
# Display the results for comparison plt.figure(figsize=(10, 5))
plt.subplot(1, 3, 1)
plt.imshow(cv2.cvtColor(gaussian_blur, cv2.COLOR_BGR2RGB))
plt.title("Gaussian Blur")
plt.subplot(1, 3, 2)
plt.imshow(cv2.cvtColor(median_blur, cv2.COLOR_BGR2RGB))
plt.title("Median Blur")
plt.subplot(1, 3, 3)
plt.imshow(cv2.cvtColor(bilateral_filter, cv2.COLOR_BGR2RGB))
plt.title("Bilateral Filter")
plt.show()
```

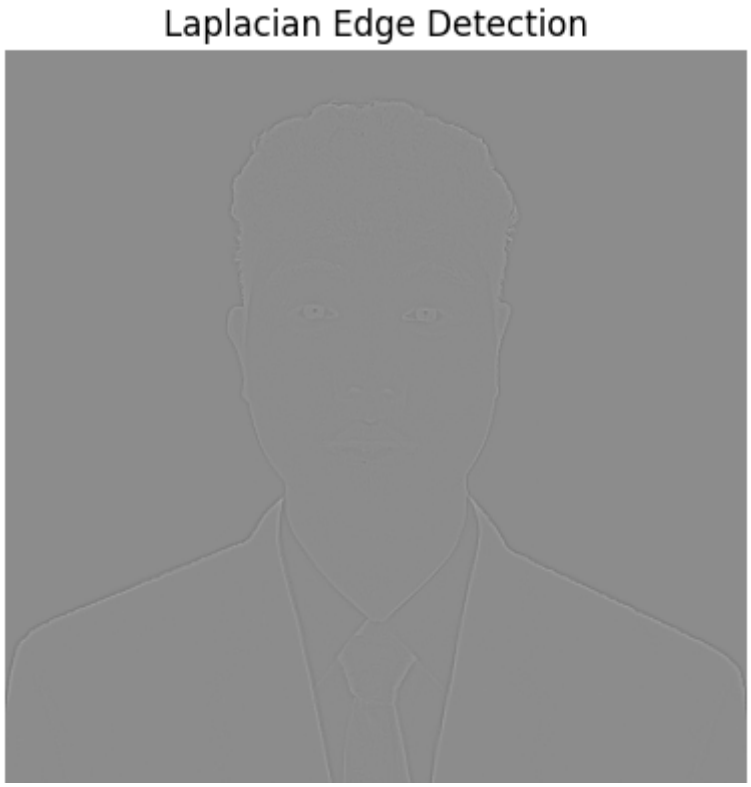


Exercise 6: Using Sobel Detection

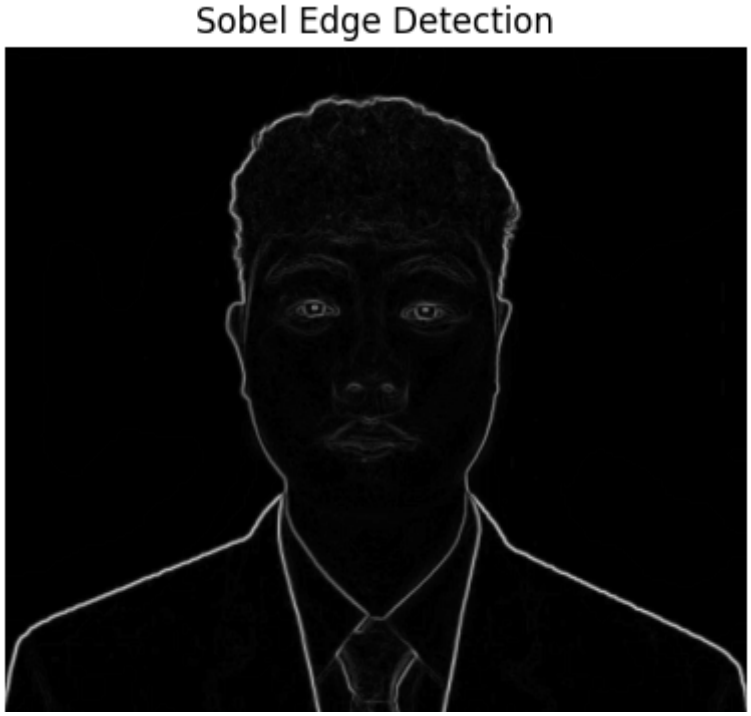
Laplacian Edge Detection

```
# Laplacian Edge Detection
def laplacian_edge_detection(img):
# Convert to grayscale
  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Apply Laplacian operator
  laplacian = cv2.Laplacian(gray, cv2.CV_64F)
  return laplacian
# Apply Laplacian edge detection to the uploaded image
laplacian_edges = laplacian_edge_detection(image)
plt.imshow(laplacian_edges, cmap='gray')
plt.title("Laplacian Edge Detection")
```

```
plt.axis('off')
plt.show()
```

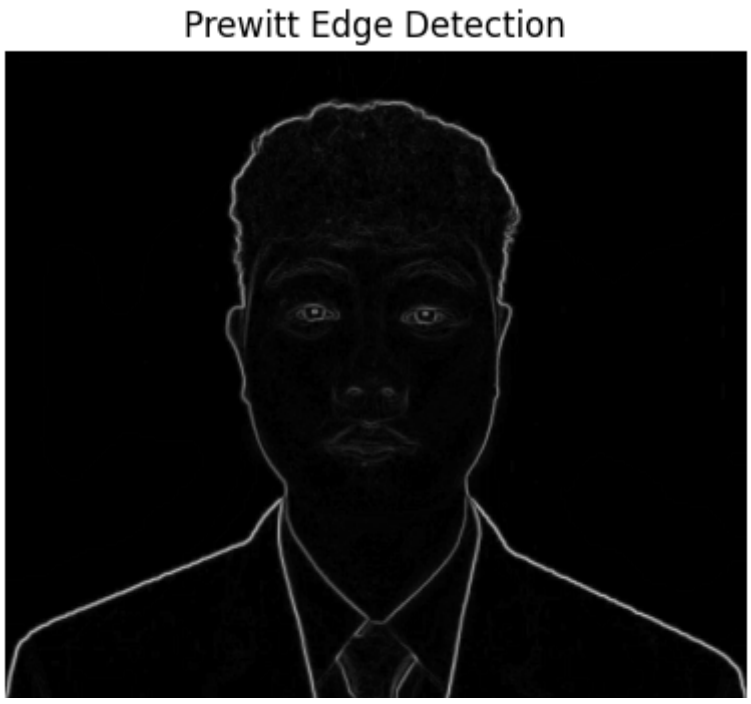



```
# Sobel Edge Detection
def sobel_edge_detection(img):
    # Convert to grayscale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Sobel edge detection in the x direction
    sobelx = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=5)

    # Sobel edge detection in the y direction
    sobely = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=5)

    # Combine the two gradients
    sobel_combined = cv2.magnitude(sobelx, sobely)

    return sobel_combined

# Apply Sobel edge detection to the uploaded image
sobel_edges = sobel_edge_detection(image)
plt.imshow(sobel_edges, cmap='gray')
plt.title("Sobel Edge Detection")
plt.axis('off')
plt.show()

```



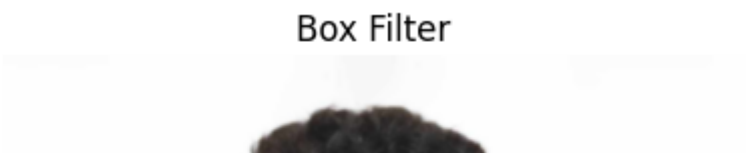
Prewitt Edge Detection

```
# Prewitt Edge Detection
def prewitt_edge_detection(img):
    # Convert to grayscale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Prewitt operator kernels for x and y directions
    kernelx = np.array([[1, 0, -1], [1, 0, -1], [1, 0, -1]], dtype=int)
    kernely = np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]], dtype=int)

    # Applying the Prewitt operator
    prewittx = cv2.filter2D(gray, cv2.CV_64F, kernelx)
    prewitty = cv2.filter2D(gray, cv2.CV_64F, kernely)

    # Combine the x and y gradients by converting to floating point
    prewitt_combined = cv2.magnitude(prewittx, prewitty)

    return prewitt_combined

# Apply Prewitt edge detection to the uploaded image
prewitt_edges = prewitt_edge_detection(image)
plt.imshow(prewitt_edges, cmap='gray')
plt.title("Prewitt Edge Detection")
plt.axis('off')
plt.show()
```



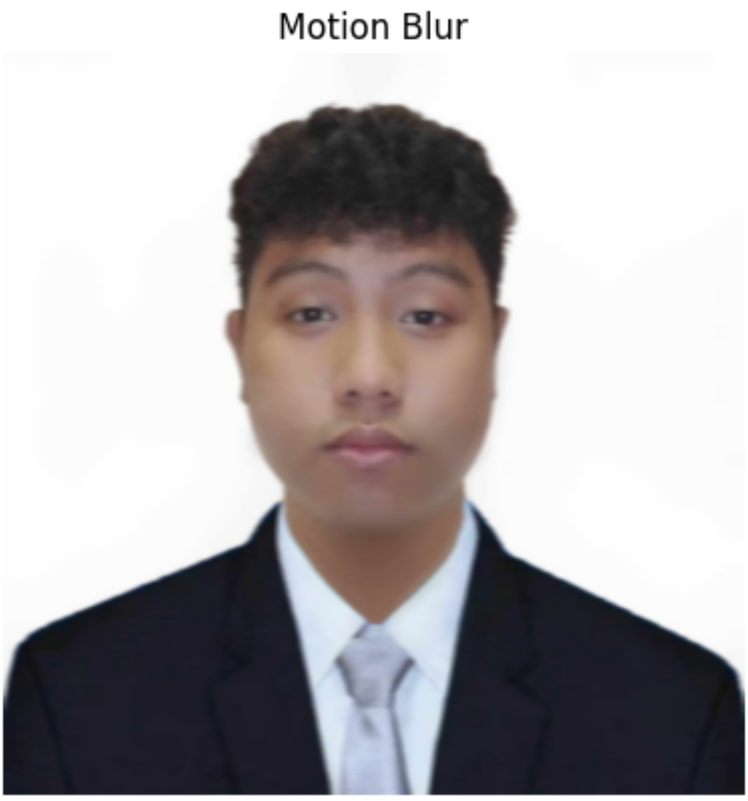
box filter

```
# Box Filter
def box_blur(img):
  box = cv2.boxFilter(img, -1, (5, 5))
  return box
# Apply Box filter to the uploaded image
box_blurred = box_blur(image)
plt.imshow(cv2.cvtColor(box_blurred, cv2.COLOR_BGR2RGB))
plt.title("Box Filter")
plt.axis('off')
plt.show()
```



Motion Blur

```
# Motion Blur
def motion_blur(img):
# Create motion blur kernel (size 15x15)
  kernel_size = 15
  kernel = np.zeros((kernel_size, kernel_size))
  kernel[int((kernel_size - 1) / 2), :] = np.ones(kernel_size)
  kernel = kernel / kernel_size
# Apply motion blur
  motion_blurred = cv2.filter2D(img, -1, kernel)
  return motion_blurred
# Apply Motion blur to the uploaded image
motion_blurred = motion_blur(image)
plt.imshow(cv2.cvtColor(motion_blurred, cv2.COLOR_BGR2RGB))
plt.title("Motion Blur")
plt.axis('off')
plt.show()
```



Unsharp Masking (Sharpening)

```
# Unsharp Masking (Sharpening)

def unsharp_mask(img):
# Create a Gaussian blur version of the image
  blurred = cv2.GaussianBlur(img, (9, 9), 10.0)
# Sharpen by adding the difference between the original and the blurred image
  sharpened = cv2.addWeighted(img, 1.5, blurred, -0.5, 0)
  return sharpened
# Apply Unsharp Masking to the uploaded image
sharpened_image = unsharp_mask(image)
plt.imshow(cv2.cvtColor(sharpened_image, cv2.COLOR_BGR2RGB))
plt.title("Unsharp Mask (Sharpening)")
plt.axis('off')
plt.show()
```

Generate | output all images in one plot and save it in pdf also put the original | Close

1 of 1 | Use code with caution

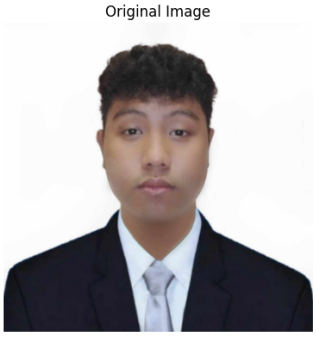
Suggested code may be subject to a license | akshay25vyas/demo-hub

```
# prompt: output all images in one plot and save it in pdf also put the original

import matplotlib.pyplot as plt
import matplotlib.backends.backend_pdf

# Create a PDF file to save the plots
pdf = matplotlib.backends.backend_pdf.PdfPages("image_processing_results.pdf")
```

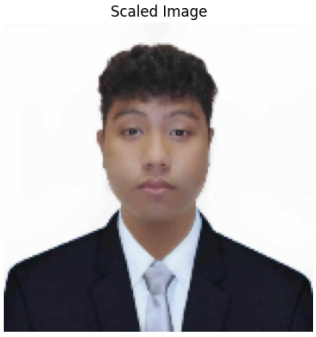
```
 8
 9 # Create a figure with subplots for all images
10 fig, axs = plt.subplots(nrows=4, ncols=4, figsize=(16, 16))
11 axs = axs.flatten()
12
13 # Display the original image
14 axs[0].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
15 axs[0].set_title("Original Image")
16 axs[0].axis('off')
17
18 # Display the scaled image
19 axs[1].imshow(cv2.cvtColor(scaled_image, cv2.COLOR_BGR2RGB))
20 axs[1].set_title("Scaled Image")
21 axs[1].axis('off')
22
23 # Display the rotated image
24 axs[2].imshow(cv2.cvtColor(rotated_image, cv2.COLOR_BGR2RGB))
25 axs[2].set_title("Rotated Image")
26 axs[2].axis('off')
27
28 # Display the Gaussian blur
29 axs[3].imshow(cv2.cvtColor(gaussian_blur, cv2.COLOR_BGR2RGB))
30 axs[3].set_title("Gaussian Blur")
31 axs[3].axis('off')
32
33 # Display the median blur
34 axs[4].imshow(cv2.cvtColor(median_blur, cv2.COLOR_BGR2RGB))
35 axs[4].set_title("Median Blur")
36 axs[4].axis('off')
37
38 # Display the Canny edge detection
39 axs[5].imshow(edges, cmap='gray')
40 axs[5].set_title("Canny Edge Detection")
41 axs[5].axis('off')
42
43 # Display the bilateral filter
44 axs[6].imshow(cv2.cvtColor(bilateral_filter, cv2.COLOR_BGR2RGB))
45 axs[6].set_title("Bilateral Filter")
46 axs[6].axis('off')
47
48 # Display the Laplacian edge detection
49 axs[7].imshow(laplacian_edges, cmap='gray')
50 axs[7].set_title("Laplacian Edge Detection")
51 axs[7].axis('off')
52
53 # Display the Sobel edge detection
54 axs[8].imshow(sobel_edges, cmap='gray')
55 axs[8].set_title("Sobel Edge Detection")
56 axs[8].axis('off')
57
58 # Display the Prewitt edge detection
59 axs[9].imshow(prewitt_edges, cmap='gray')
```

```
axs[9].set_title("Prewitt Edge Detection")
axs[9].axis('off')

# Display the Box filter
axs[10].imshow(cv2.cvtColor(box_blurred, cv2.COLOR_BGR2RGB))
axs[10].set_title("Box Filter")
axs[10].axis('off')

# Display the Motion blur
axs[11].imshow(cv2.cvtColor(motion_blurred, cv2.COLOR_BGR2RGB))
axs[11].set_title("Motion Blur")
axs[11].axis('off')

# Display the Unsharp Masking
axs[12].imshow(cv2.cvtColor(sharpened_image, cv2.COLOR_BGR2RGB))
axs[12].set_title("Unsharp Masking")
axs[12].axis('off')

# Remove any unused subplots
for i in range(13, 16):
  fig.delaxes(axs[i])

# Adjust the layout and save the plot to the PDF
plt.tight_layout()
pdf.savefig(fig)
plt.show()

# Close the PDF file
pdf.close()

```

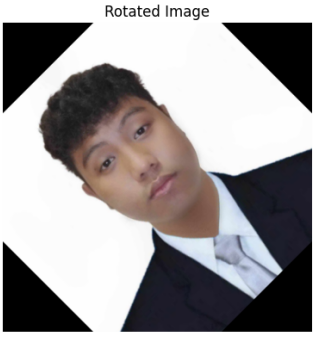
Original Image



Scaled Image



Rotated Image



Gaussian Blur

Median Blur

Canny Edge Detection

Bilateral Filter

Laplacian Edge Detection

Sobel Edge Detection

Prewitt Edge Detection

Box Filter

Motion Blur

Unsharp Masking