Apply image processing techniques(Scaling, Rotation, Blurring, Edge Detection) OpenCV

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


def display_image(img,title='Image'):
  plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
  plt.title(title)
  plt.axis('off')
  plt.show()

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.

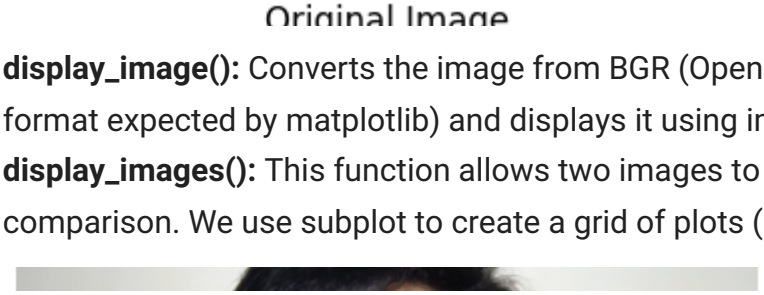
Step 3: Load an Image

```
from google.colab import files
from io import BytesIO
from PIL import Image
# Upload an image
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 Messenger_…E1AA.jpeg

- **Messenger_creation_1BC25005-A74E-47F3-BED0-32E8D97CE1AA.jpeg**(image/jpeg) - 426718 bytes, last modified: 9/16/2024 - 100% done

```
display_image(image,"Original Image")
```



- **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 Rotating

```
# 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, 0.5)
display_image(scaled_image, "Scaled Image (50%)")
# 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
```

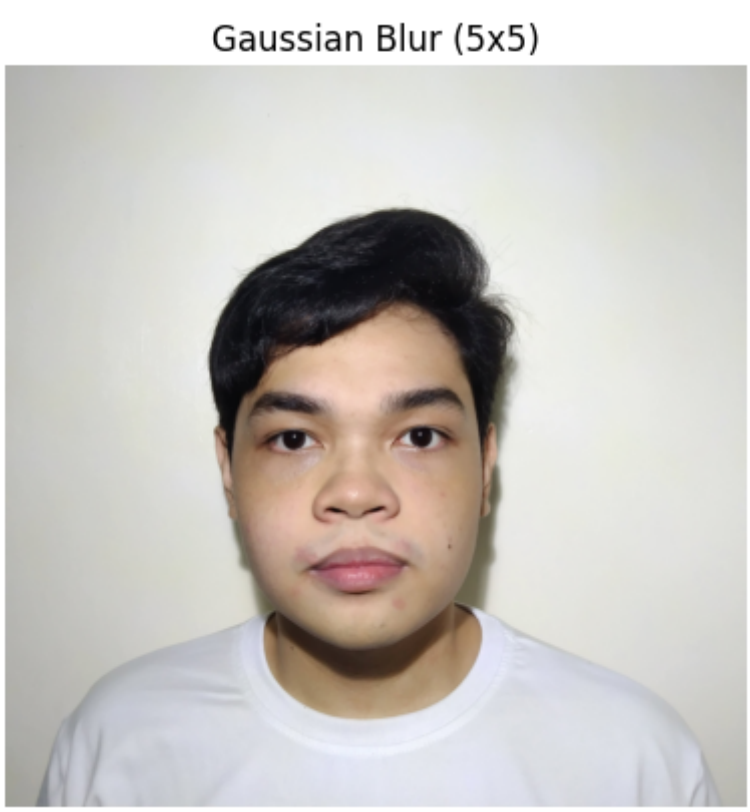
Scaled Image (50%)

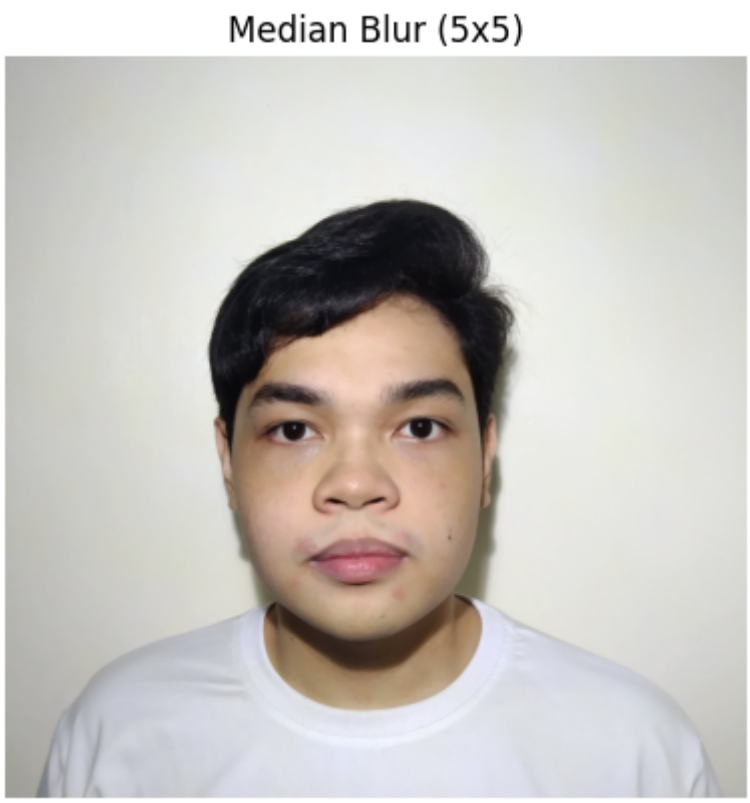
Rotated Image (45°)

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)")
"""
```

```
11 cv2.medianBlur(): Applies a median blur, which replaces each pixel's value with t
12 """
```

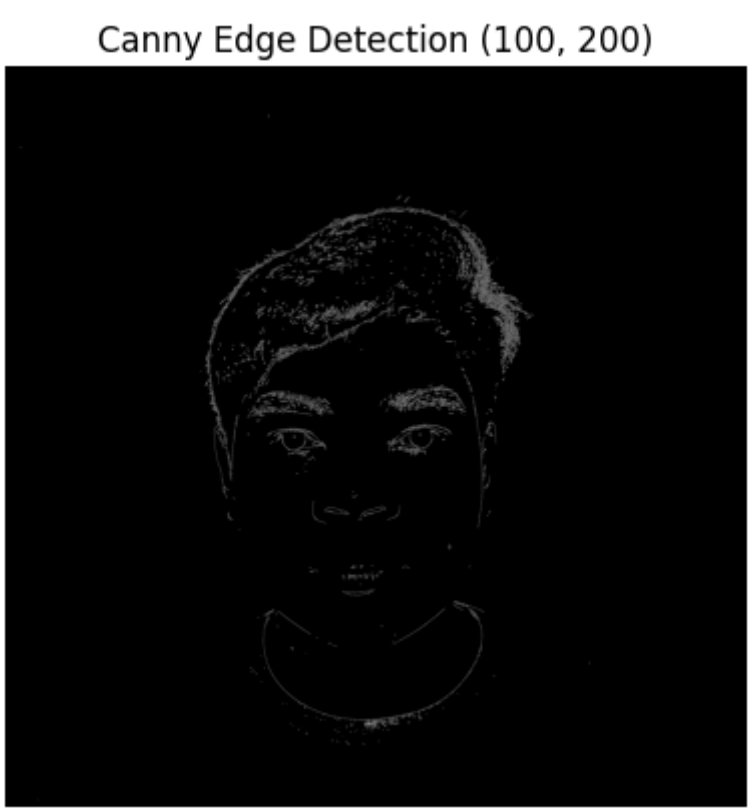





```
'\ncv2.medianBlur(): Applies a median blur, which replaces each pixel's value wi
th the median value of its neighbors in a 5x5 kernel. This method is particularl
```

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.
"""
```



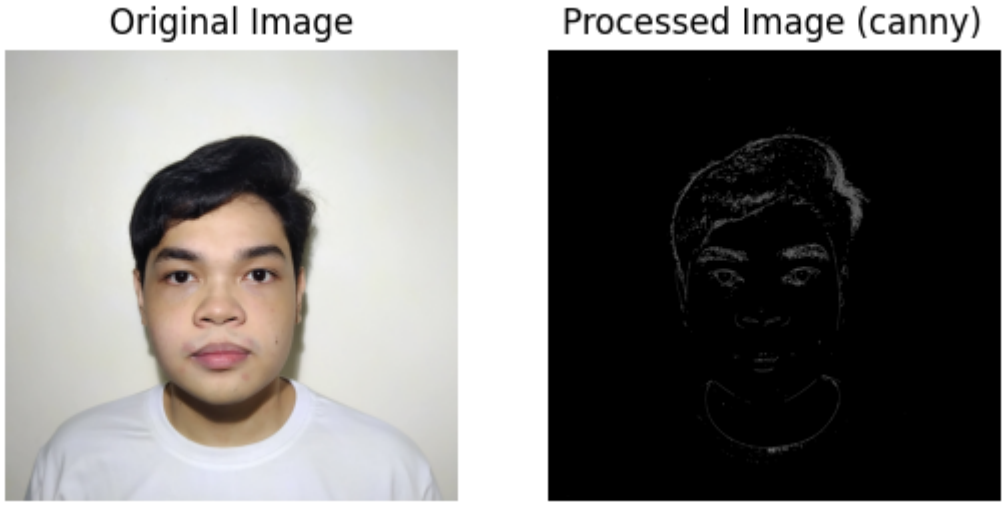

```
'\ncv2.Canny(): Detects edges in the image by calculating the gradient (rate of
intensity change) between pixels. The two threshold values (100 and 200) define
the edges'\nsensitivity. Lower thresholds detect more edges, while higher thresh
```

Exercise 4: Basic Image Processor (Interactive)

```
def process_image(img, action):
  if action == 'scale':
    return scale_image(img, 0.5)
  elif action == 'rotate':
    return rotate_image(img, 45)
  elif action == 'gaussian_blur':
    return cv2.GaussianBlur(img, (5, 5), 0)
  elif action == 'median_blur':
    return cv2.medianBlur(img, 5)
  elif action == 'canny':
    return cv2.Canny(img, 100, 200)
  else:
```

```
    else:
        return img
"""
process_image(): This function allows users to specify an image transformation (sca
"""
action = input("Enter action (scale, rotate, gaussian_blur, median_blur, canny): ")
processed_image = process_image(image, action)
display_images(image, processed_image, "Original Image", f"Processed Image ({actior
"""
This allows users to enter their desired transformation interactively (via the
input() function). It processes the image and displays both the original and transf
"""

```

```
Enter action (scale, rotate, gaussian_blur, median_blur, canny): canny
```

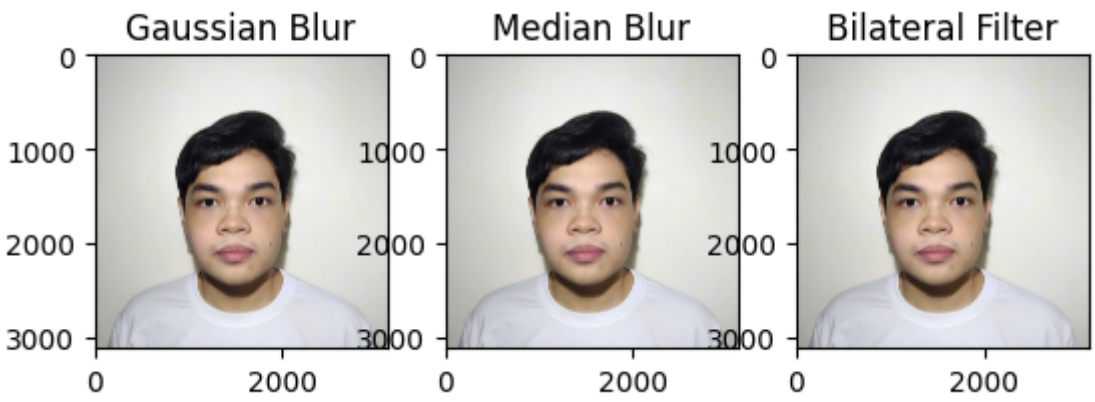



```
'\nThis allows users to enter their desired transformation interactively (via th
e\ninput() function). It processes the image and displays both the original and
```

Exercise 5: Comparison of Filtering Techniques

```
# Applying Gaussian, Median, and Bilateral filters
gaussian_blur = cv2.GaussianBlur(image, (5, 5), 0)
median_blur = cv2.medianBlur(image, 5)
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))
```

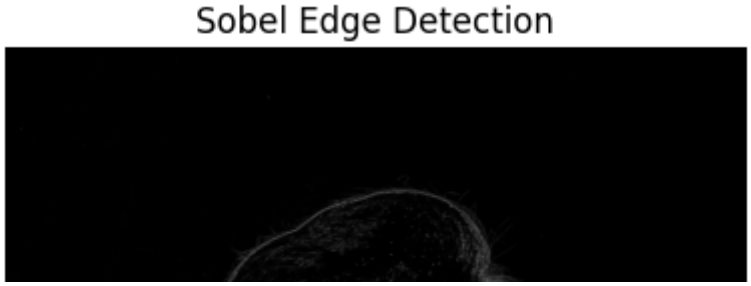
```
plt.title("Bilateral Filter")
plt.show()
```



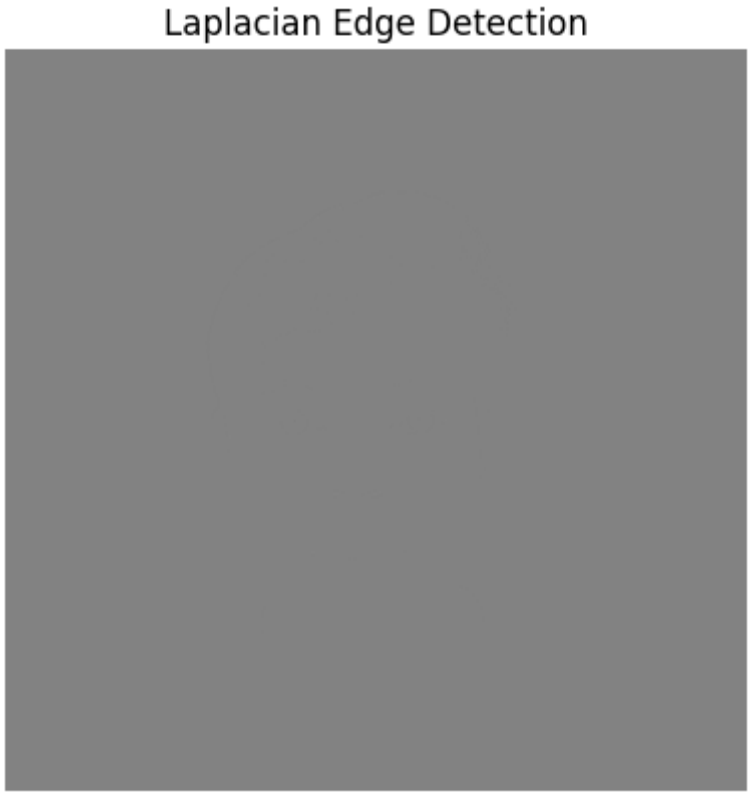
Exercise 6: Sobel Edge Detection

```
# 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
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()
```

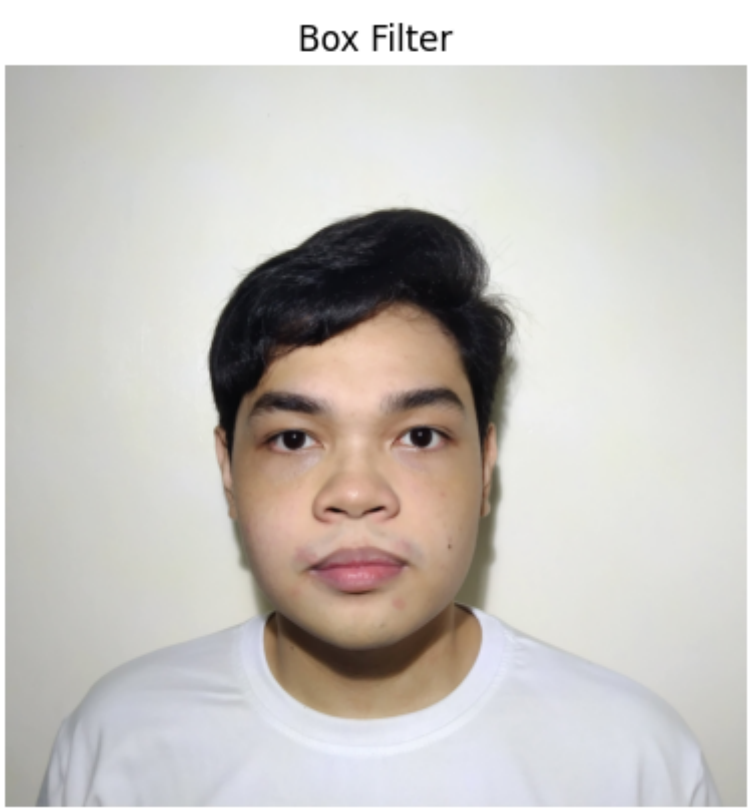



```
def laplacian_edge_detection(img):
# Convert to grayscale
  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  # Apply Laplacian operator
  laplacian = cv2.Laplacian(gray, cv2.CV_64F)
  return laplacian
# Apply Laplacian edge detection to the uploaded image
laplacian_edges = laplacian_edge_detection(image)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

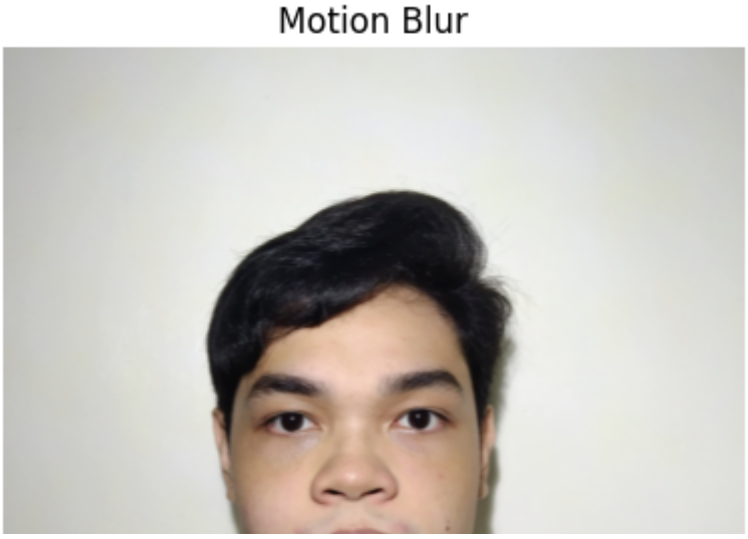



```
def box_blur(img):
  box = cv2.boxFilter(img, -1, (5, 5))
  return box
# Apply Box filter to the uploaded image
box_blurred = box_blur(image)
plt.imshow(cv2.cvtColor(box_blurred, cv2.COLOR_BGR2RGB))
```

```
plt.title("Box Filter")
plt.axis('off')
plt.show()
```

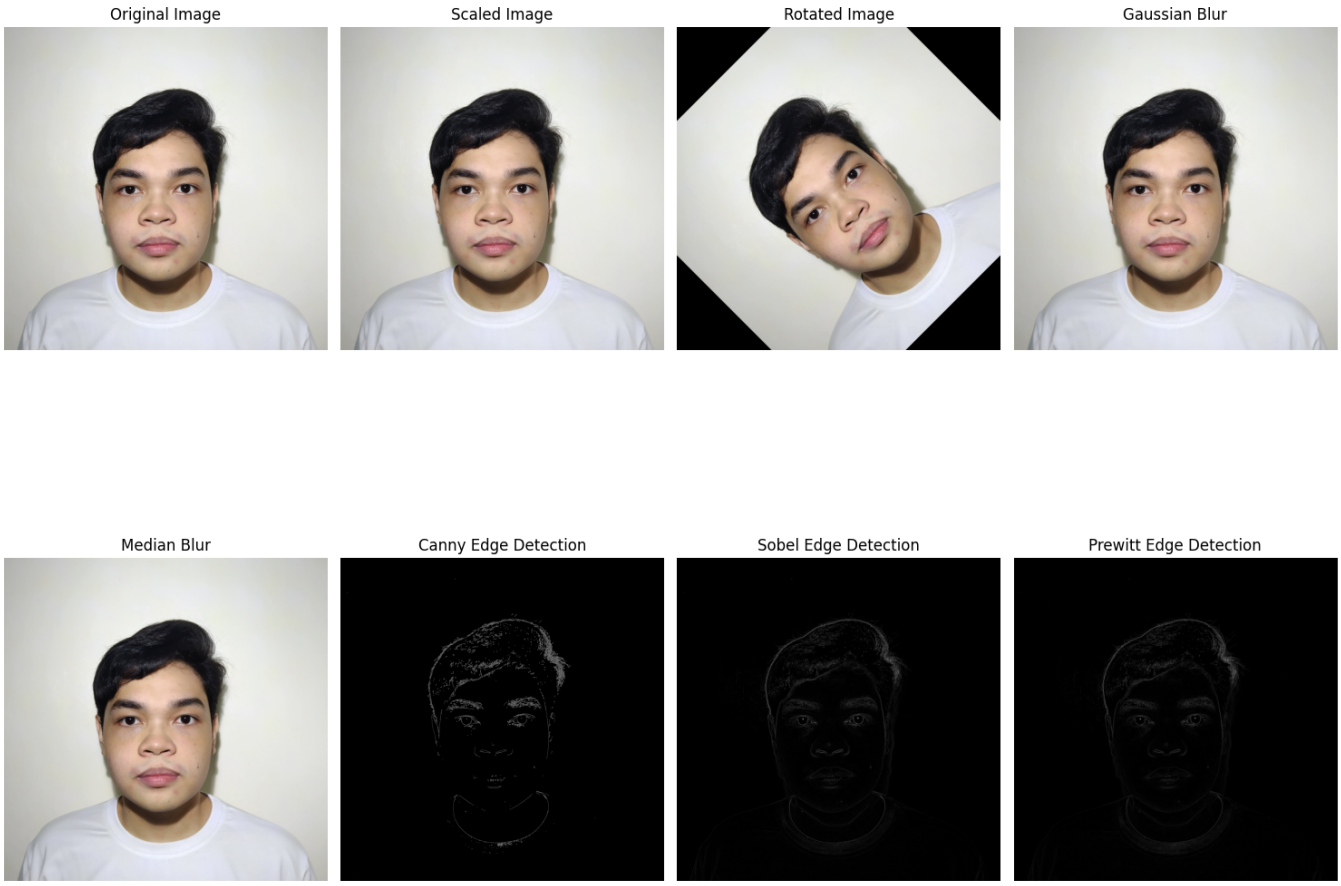



```
def motion_blur(img):
# Create motion blur kernel (size 15x15)
  kernel_size = 15
  kernel = np.zeros((kernel_size, kernel_size))
  kernel[int((kernel_size - 1) / 2), :] = np.ones(kernel_size)
  kernel = kernel / kernel_size
  # Apply motion blur
  motion_blurred = cv2.filter2D(img, -1, kernel)
  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()
```




```
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()
```

```
# prompt: output all iamges in one plot

plt.figure(figsize=(15, 14))

plt.subplot(2, 4, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title("Original Image")
plt.axis('off')

plt.subplot(2, 4, 2)
plt.imshow(cv2.cvtColor(scaled_image, cv2.COLOR_BGR2RGB))
plt.title("Scaled Image")
plt.axis('off')

plt.subplot(2, 4, 3)
plt.imshow(cv2.cvtColor(rotated_image, cv2.COLOR_BGR2RGB))
plt.title("Rotated Image")
plt.axis('off')

plt.subplot(2, 4, 4)
plt.imshow(cv2.cvtColor(gaussian_blur, cv2.COLOR_BGR2RGB))
plt.title("Gaussian Blur")
plt.axis('off')

plt.subplot(2, 4, 5)
plt.imshow(cv2.cvtColor(median_blur, cv2.COLOR_BGR2RGB))
plt.title("Median Blur")
plt.axis('off')

plt.subplot(2, 4, 6)
plt.imshow(edges, cmap='gray')
plt.title("Canny Edge Detection")
plt.axis('off')

plt.subplot(2, 4, 7)
plt.imshow(sobel_edges, cmap='gray')
plt.title("Sobel Edge Detection")
plt.axis('off')

plt.subplot(2, 4, 8)
plt.imshow(prewitt_edges, cmap='gray')
plt.title("Prewitt Edge Detection")
plt.axis('off')


plt.tight_layout()
plt.show()

```


Original Image
Scaled Image
Rotated Image
Gaussian Blur
Median Blur
Canny Edge Detection
Sobel Edge Detection
Prewitt Edge Detection

Generate | output all images in one plot | Close

1 of 1 | Undo Changes | Use code with caution

```
plt.figure(figsize=(15, 14))

plt.subplot(4, 4, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title("Original Image")
plt.axis('off')

plt.subplot(4, 4, 2)
plt.imshow(cv2.cvtColor(scaled_image, cv2.COLOR_BGR2RGB))
plt.title("Scaled Image")
plt.axis('off')

plt.subplot(4, 4, 3)
plt.imshow(cv2.cvtColor(rotated_image, cv2.COLOR_BGR2RGB))
plt.title("Rotated Image")
plt.axis('off')

plt.subplot(4, 4, 4)
plt.imshow(cv2.cvtColor(gaussian_blur, cv2.COLOR_BGR2RGB))
plt.title("Gaussian Blur")
plt.axis('off')

plt.subplot(4, 4, 5)
plt.imshow(cv2.cvtColor(median_blur, cv2.COLOR_BGR2RGB))
plt.title("Median Blur")
plt.axis('off')

plt.subplot(4, 4, 6)
plt.imshow(edges, cmap='gray')
plt.title("Canny Edge Detection")
plt.axis('off')

plt.subplot(4, 4, 7)
plt.imshow(sobel_edges, cmap='gray')
plt.title("Sobel Edge Detection")
plt.axis('off')

plt.subplot(4, 4, 8)
plt.imshow(prewitt_edges, cmap='gray')
plt.title("Prewitt Edge Detection")
plt.axis('off')

plt.subplot(4, 4, 9)
```

```
plt.imshow(laplacian_edges, cmap='gray')
plt.title("Laplacian Edge Detection")
plt.axis('off')

plt.subplot(4, 4, 10)
plt.imshow(cv2.cvtColor(box_blurred, cv2.COLOR_BGR2RGB))
plt.title("Box Blur")
plt.axis('off')

plt.subplot(4, 4, 11)
plt.imshow(cv2.cvtColor(motion_blurred, cv2.COLOR_BGR2RGB))
plt.title("Motion Blur")
plt.axis('off')

plt.subplot(4, 4, 12)
plt.imshow(cv2.cvtColor(sharpened_image, cv2.COLOR_BGR2RGB))
plt.title("Unsharp Mask")
plt.axis('off')


plt.subplot(4, 4, 13)
plt.imshow(cv2.cvtColor(bilateral_filter, cv2.COLOR_BGR2RGB))
plt.title("Bilateral FIlter")
plt.axis('off')



plt.tight_layout()
plt.show()

```

Original Image

Scaled Image

Rotated Image

Gaussian Blur

Median Blur

Canny Edge Detection

Sobel Edge Detection

Prewitt Edge Detection

Laplacian Edge Detection

Box Blur

Motion Blur

Unsharp Mask

Bilateral FIlter