Image Processing Techniques | Documentation

Today, we were assigned to implement various image processing techniques using Google Colab for the exercises.

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

The `opencv-python-headless` library is installed to enable image processing and computer vision tasks in Python. This version is suitable for server environments or systems without a display, as it does not include GUI functionalities.

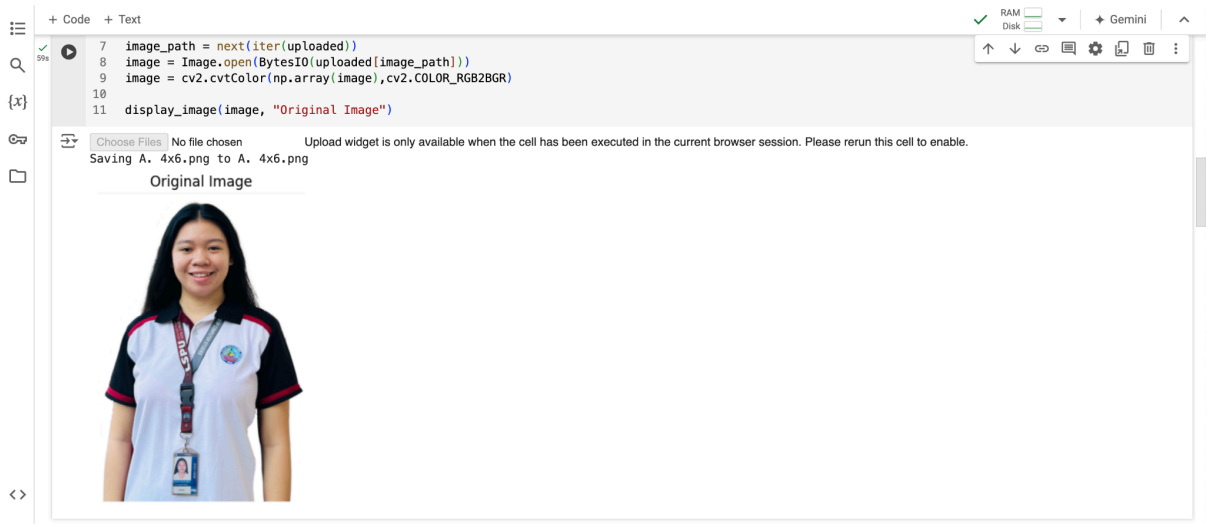
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

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="Image2"):
 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(img1,cv2.COLOR_BGR2RGB))
 plt.title(title2)
 plt.axis('off')

 plt.show()
```

This code defines two functions for displaying images with matplotlib and OpenCV. The display_image function shows a single image with a specified title, converting it from BGR to RGB color space for accurate color display. The display_images function displays two images side by side, also converting them from BGR to RGB, and allows setting titles for each image. Both functions utilize matplotlib to ensure proper image visualization and formatting.

```
from google.colab import files
from io import BytesIO
from PIL import Image

uploaded = files.upload()

image_path = next(iter(uploaded))
image = Image.open(BytesIO(uploaded[image_path]))
image = cv2.cvtColor(np.array(image),cv2.COLOR_RGB2BGR)

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

This code is intended for Google Colab to upload and display an image. It starts by using files.upload() to prompt the user to upload an image file. Once uploaded, the file reads the image into memory using BytesIO and PIL. The image is then converted from RGB to BGR color space using cv2.cvtColor for compatibility with OpenCV. Finally, it displays the image with the display_image function, labeled as "Original Image."

OUTPUT:



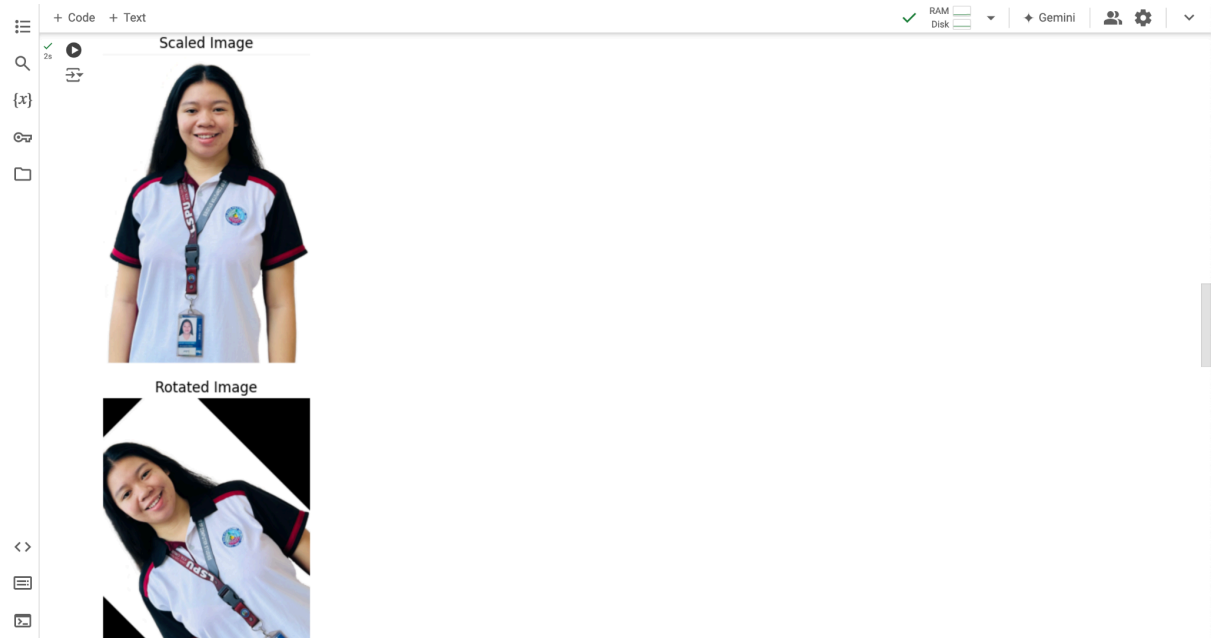
Exercise 1: Scaling and Rotation

```
def scale_image(img, scale_factor):
 height, width = img.shape[:2]
  scale_img = cv2.resize(img, (int(width * scale_factor), int(height *
scale_factor)), interpolation = cv2.INTER_LINEAR)
 return scale_img

def rotate_image(image, angle):
 height, width = image.shape[:2]
 center = (width//2,height//2)
```

```
 matrix = cv2.getRotationMatrix2D(center, angle, 1)
 rotated_image = cv2.warpAffine(image, matrix,(width, height))
 return rotated_image

scaled_image = scale_image(image, 0.5)
display_image(scaled_image, "Scaled Image")

rotated_image = rotate_image(image, 45)
display_image(rotated_image, "Rotated Image")
```

This includes two functions for image processing: scale_image, which resizes an image by a given scale factor, and rotate_image, which rotates an image by a specified angle. It then scales an image to half its size and rotates it by 45 degrees, displaying both results using the display_image function.

OUTPUT:

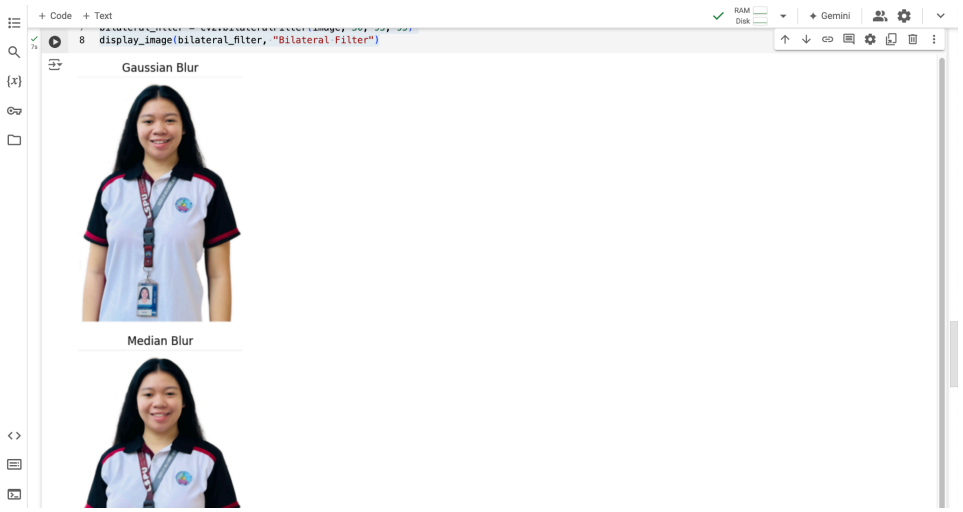


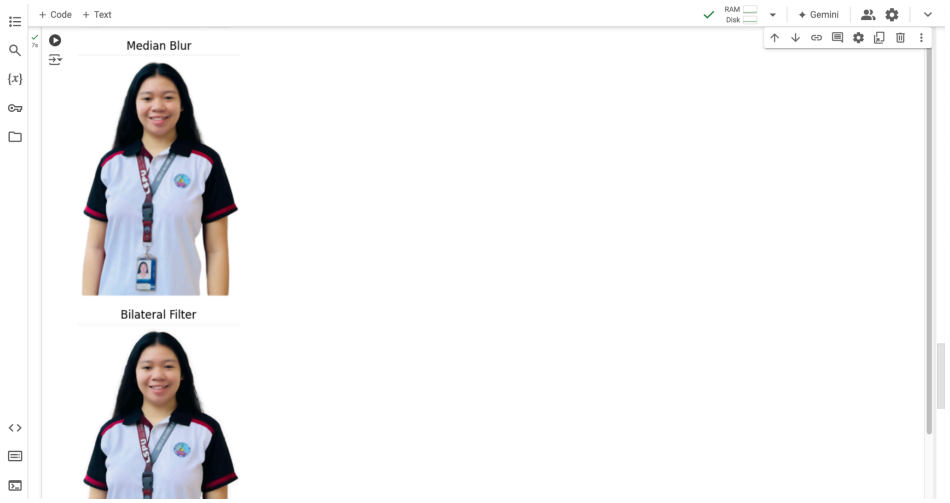
Exercise 2: Blurring Techniques

```
gaussian_blur = cv2.GaussianBlur(image, (11,11),0)
display_image(gaussian_blur, "Gaussian Blur")

median_blur = cv2.medianBlur(image, 15)
display_image(median_blur, "Median Blur")

bilateral_filter = cv2.bilateralFilter(image, 30, 95, 95)
display_image(bilateral_filter, "Bilateral Filter")
```

This uses OpenCV to apply three different types of image-blurring techniques. First, it applies Gaussian blur with a kernel size of 11x11, then median blur with a kernel size of 15, and finally, a bilateral filter with a diameter of 30 and sigma values of 95. Each processed image is displayed using the display_image function, labeled accordingly as "Gaussian Blur," "Median Blur," and "Bilateral Filter."

OUTPUT:



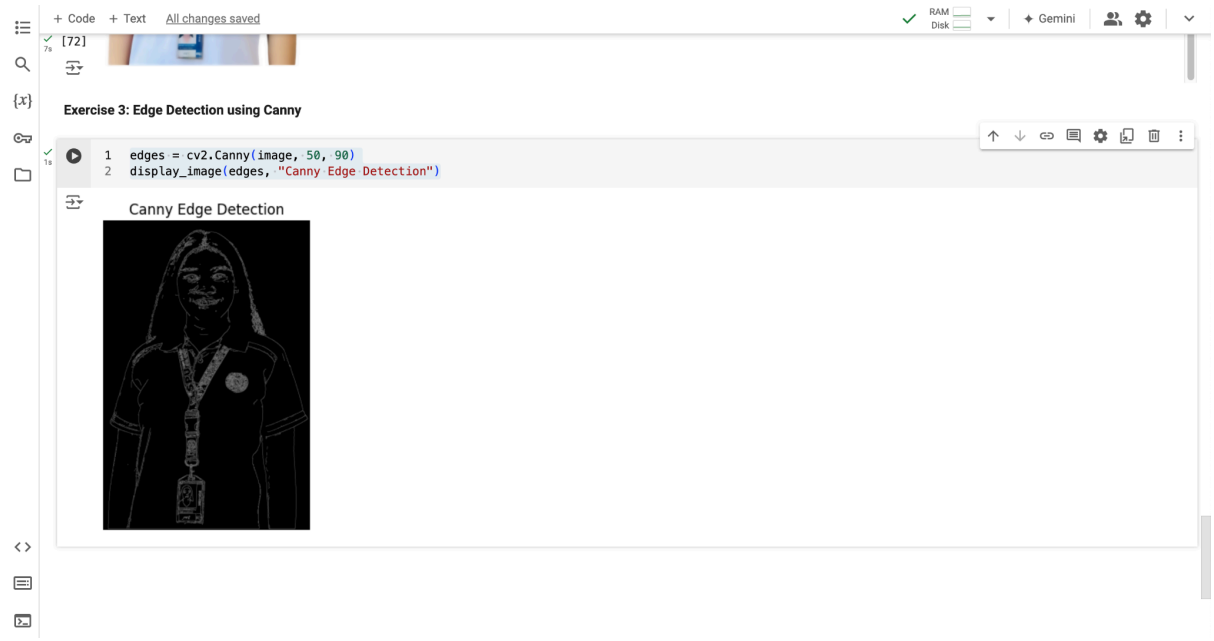


Exercise 3: Edge Detection using Canny

```
edges = cv2.Canny(image, 50, 90)
display_image(edges, "Canny Edge Detection")
```

The code performs Canny edge detection on an image using OpenCV. It detects edges by applying the Canny algorithm with threshold values of 50 and 90, then displays the result using the display_image function, labeled as "Canny Edge Detection."

OUTPUT:


+ Code
+ Text
All changes saved
RAM
Disk
Gemini
[72]
Exercise 3: Edge Detection using Canny
1 edges = cv2.Canny(image, 50, 90)
2 display_image(edges, "Canny Edge Detection")
Canny Edge Detection