Lab 5

Task: Implement all the topics/functions of OpenCV, in a notebook file (ipynb file) (main topics, can skip others)

Here's a Working of this Code:

1. Importing Libraries

- **`cv2` (OpenCV)**: A library used for image processing tasks such as reading, manipulating, and displaying images.
- **`numpy`**: A package for numerical operations, often used with OpenCV for image manipulation.
- **`matplotlib.pyplot`**: Used for displaying images in Jupyter Notebooks.

2. Helper Function: `display_image(image, title="Image")`

- **Purpose**: Displays an image using `matplotlib`.
- **Parameters**:
  - `image`: The image to be displayed.
  - `title`: The title for the displayed image (default is "Image").
- **Working**:
  - Converts the image from BGR (OpenCV default) to RGB for correct display using `cv2.cvtColor()`.
  - Uses `plt.imshow()` to display the image and `plt.title()` to set the title.
  - Hides axes using `plt.axis('off')`.

3. Basic Image Operations

Read and Display Image

- **`cv2.imread('Image.jpg')`**: Reads an image file from the specified path.
- **`display_image(img, 'Original Image')`**: Displays the original image using the helper function.

Save Image

- **`cv2.imwrite('output_image.jpg', img)`**: Saves the image as a new file (`output_image.jpg`).

Resize Image

- **`cv2.resize(img, (400, 400))`**: Resizes the image to a size of 400x400 pixels.
- **`display_image(resized_img, 'Resized Image')`**: Displays the resized image.

Crop Image

- `cropped_img = img[50:250, 100:300]`: Crops a section of the image (rows 50 to 250, columns 100 to 300).
- `display_image(cropped_img, 'Cropped Image')`: Displays the cropped image.

Rotate Image

- `rotated_img = cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE)`: Rotates the image 90 degrees clockwise.
- `display_image(rotated_img, 'Rotated Image')`: Displays the rotated image.

4. Image Processing

Convert to Grayscale

- `cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)`: Converts the image from BGR to grayscale.
- `display_image(gray_img, 'Grayscale Image')`: Displays the grayscale version of the image.

Thresholding (Binary)

- `cv2.threshold(gray_img, 127, 255, cv2.THRESH_BINARY)`: Applies a binary threshold to the grayscale image. All pixel values above 127 are set to 255 (white), and those below 127 are set to 0 (black).
- `display_image(threshold_img, 'Thresholded Image')`: Displays the thresholded image.

Gaussian Blur

- `cv2.GaussianBlur(img, (15, 15), 0)`: Applies a Gaussian blur with a kernel size of 15x15 to smooth the image.
- `display_image(blurred_img, 'Blurred Image')`: Displays the blurred image.

Edge Detection using Canny

- `cv2.Canny(gray_img, 100, 200)`: Applies the Canny edge detection algorithm to find edges in the grayscale image.
- `display_image(edges, 'Canny Edge Detection')`: Displays the image with detected edges.

5. Drawing on Images

Draw a Line

- `cv2.line(line_img, (50, 50), (300, 300), (0, 255, 0), 3)`: Draws a green line from (50, 50) to (300, 300) with a thickness of 3 pixels.
- `display_image(line_img, 'Line on Image')`: Displays the image with a line drawn.

Draw a Rectangle

- `cv2.rectangle(rect_img, (50, 50), (250, 250), (255, 0, 0), 2)`: Draws a blue rectangle with the top-left corner at (50, 50) and the bottom-right corner at (250, 250), with a thickness of 2 pixels.
- `display_image(rect_img, 'Rectangle on Image')`: Displays the image with a rectangle drawn.

Draw a Circle

- `cv2.circle(circle_img, (150, 150), 100, (0, 0, 255), 2)`: Draws a red circle with the center at (150, 150) and a radius of 100 pixels, with a thickness of 2 pixels.
- `display_image(circle_img, 'Circle on Image')`: Displays the image with a circle drawn.

Add Text to Image

- `cv2.putText(text_img, 'OpenCV!', (50, 350), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)`: Adds the text "OpenCV!" to the image at position (50, 350), with a font size of 1, color white, and thickness of 2.
- `display_image(text_img, 'Text on Image')`: Displays the image with text added.

6. Object Detection: Face Detection (Haar Cascades)

Load Pre-trained Face Classifier

- `cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')`: Loads a pre-trained Haar Cascade classifier for detecting faces.

Convert Image to Grayscale for Face Detection

- `cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)`: Converts the image to grayscale (necessary for Haar Cascade detection).

Detect Faces

- `face_cascade.detectMultiScale(gray_img_for_face, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))`: Detects faces in the image using the Haar Cascade classifier. The parameters control the sensitivity and accuracy of detection.
    - `scaleFactor=1.1`: Compensates for potential size variations of faces.

- `minNeighbors=5`: Minimum number of neighbors required for a region to be considered a face.
- `minSize=(30, 30)`: Minimum size of the detected face.

Draw Rectangles Around Faces

- `cv2.rectangle(face_img, (x, y), (x + w, y + h), (0, 255, 0), 2)`: Draws green rectangles around detected faces.
- `display_image(face_img, 'Face Detection')`: Displays the image with rectangles drawn around faces.

7. Close OpenCV Windows

- `cv2.destroyAllWindows()`: Closes all OpenCV

OUTPUT