Tutorial 5.1: Image processing using openCV

Q1. Write a Python function that takes an RGB image and a specific color range as input and returns a grayscale image where only the pixels within that color range are highlighted. All other pixels should be set to black.

Steps to Implement:

1. **Isolate the RGB Channels:**
    - Extract the Red, Green, and Blue channels from the image separately.
2. **Apply a Color Mask:**
    - Create a mask that identifies the pixels falling within the specified color range. This should be done by checking if each pixel's R, G, and B values fall within the given bounds.
3. **Convert to Grayscale:**
    - For the pixels that fall within the specified range, convert their RGB values to grayscale using a standard grayscale conversion formula (e.g., Y=0.299R+0.587G+0.1B). Set all other pixels to black.
4. **Return the Grayscale Image:**
    - The output should be a grayscale image highlighting the specific color shades from the original image.

Example Input:

- An RGB image of a your choice.
- A color range of green shades using MSPaint or GIMP (e.g., R: 50-100, G: 100-255, B: 50-100).

Expected Output:

- A grayscale image where only the green shades from the original image are visible, and all other areas are black.

Question:

Explain why simple grayscale may not always work in image processing. Explain various scenarios and conjure an example image using MSPaint or GIMP to highlight your point.

Hints:

- You can extract individual color channels using image[:, :, 0] for red, image[:, :, 1] for green, and image[:, :, 2] for blue, assuming that you have converted the image from the BGR to the RGB format.
- The mask can be created using logical conditions in numpy, e.g., mask = (R > 50) & (R < 100) & (G > 100) & (G < 255) & (B > 50) & (B < 100).
- Use the mask to selectively convert only certain pixels to grayscale while setting the rest to black.

Q2. Create a directory and add a few sample JPEG images of your choice to it.

Write a Python program that will do the following actions:

1. Traverses through all JPEG files in the directory.
2. For each image, extracts the Red component.
3. Saves the Red component as a new JPEG file in the same directory with the filename format <original_file>_redch.jpeg.

Steps to Implement:

1. **File Traversal:**
    - Use Python's os module to iterate through all JPEG files in the specified directory.
    - Use the following code to isolate only jpeg files for processing:

```
for filename in os.listdir(directory):
      if filename.endswith(".jpeg") or filename.endswith(".jpg"):
```

2. **Extracting the Red Channel:**
    - For each JPEG image, read the image using OpenCV.
    - Extract the Red channel (i.e., the first channel in the BGR image format used by OpenCV).
3. **Saving the Red Component:**
    - Save the extracted Red channel as a new JPEG file using Matplotlib or OpenCV.
    - The new filename should follow the format <original_file>_redch.jpeg.

Hints:

- Use os.listdir() to get the list of files in the directory.
- Use cv2.imread() to read the images.
- The Red channel can be accessed using image[:, :, 2].
- Use cv2.imwrite() or matplotlib.pyplot.imsave() to save the Red channel as a JPEG file. In case you are having issues with imwrite, just use matplotlib instead.

Q3. An image of a SS crack initiation is provided. Your task is to processes the image of a material sample with visible cracks. The goal is to isolate the major crack from smaller ones using morphological operations. Once isolated, calculate the area of the major crack in square microns, given that each pixel represents an area of 100 microns by 100 microns.

Steps to Implement:

1. **Read the Image:**
    - Load the binary image where cracks appear as black regions on a mostly white background. Apply a thresholding upon looking at the historgram of the image. This one is fairly easy as the image in color itself looks quite black-and-white.
2. **Apply Morphological Closing:**
    - Use the closing operation to fill in small gaps within the cracks, making the cracks more contiguous.
3. **Apply Morphological Opening:**
    - Use the opening operation to remove small cracks and noise, leaving only the major crack.
4. **Calculate the Area of the Major Crack:**
    - Count the number of pixels in the isolated major crack.
    - Multiply the number of pixels by the area each pixel represents (given as 100 microns by 100 microns).
5. **Display Results:**
    - Display the original image, the image after morphological operations, and the image with the isolated major crack highlighted.
    - Print the calculated area of the major crack as the title of the image in matplotlib. Save the image as major_crack_SS.jpg

Hints:

- Use cv2.morphologyEx() with cv2.MORPH_CLOSE to fill small gaps in the cracks.
- Use cv2.morphologyEx() with cv2.MORPH_OPEN to remove smaller cracks and noise.

Q4. Write a program that processes an image of a spanner taken in a lighting gradient environment; figure is provided. The goal is to use adaptive thresholding to account for the varying lighting conditions and then detect and highlight the outline of the spanner.

Steps to Implement:

1. **Read the Image:**
    - Load the image of the spanner, which has varying lighting conditions across different parts of the image.
2. **Convert to Grayscale:**
    - Convert the image to grayscale, as adaptive thresholding works on single-channel images.
3. **Apply Adaptive Thresholding:**
    - Use adaptive thresholding to binarize the image. This technique adjusts the threshold value dynamically across the image, making it suitable for images with uneven lighting.
    - Choose appropriate parameters for cv2.adaptiveThreshold() to ensure the spanner is well highlighted.
4. **Detect Edges (optional):**
    - Use an edge detection method like Canny to find the edges of the spanner.
5. **Find and Draw the Contours:**
    - Use contour detection to find the outline of the spanner.
    - Draw the detected contour on the original image to highlight the outline of the spanner.
6. **Display Results:**
    - Display the original image, the binary image after adaptive thresholding, and the image with the detected spanner outline.
    - Save the final output file to disk.

Hints:

- Use cv2.cvtColor() to convert the image to grayscale.
- Use cv2.adaptiveThreshold() with parameters such as cv2.ADAPTIVE_THRESH_GAUSSIAN_C or cv2.ADAPTIVE_THRESH_MEAN_C to handle the lighting gradient.
- Use cv2.Canny() for edge detection to enhance the contours, if required.
- Use cv2.findContours() to detect the contours of the spanner and cv2.drawContours() to highlight it.