OpenCV Practice

GitHub Repository:
https://github.com/Emmanuelcsam/OpenCV-Practice

Libraries & Initial Setup

Required Libraries

- OpenCV (cv2): Core computer vision functionalities
- NumPy: Numerical operations and array manipulation
- Matplotlib: Plotting images and histograms
- OS: File system operations

Initial Setup Code

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
import os

# Set base path and read image
base_path = 'C:/Users//Documents/GitHub/OpenCV-Practice/'
image_path = base_path + 'circle.png'
assert os.path.exists(image_path), f"File could not be read.
Check if '{image_path}' exists."
```

Image Processing Techniques

1. Hough Circle Transform

A method used to detect circles in an image.

Documentation: OpenCV HoughCircles Tutorial

Code Example

```
import cv2 as cv
import numpy as np
```

```
# Reads image in grayscale
img = cv.imread(image_path, cv.IMREAD_GRAYSCALE)

# Adds blur to reduce noise, prevents detection of false circles
img_blur = cv.medianBlur(img, 5)

# Creates a colored version of the image for drawing circles
cimg = cv.cvtColor(img_blur, cv.COLOR_GRAY2BGR)

# Applies Hough Transformation
circles = cv.HoughCircles(
    image=img_blur,
    method=cv.HOUGH_GRADIENT,
    dp=1,
    minDist=20,
    param1=50,
    param2=30,
    minRadius=0,
    maxRadius=0
)

# Draw detected circles
if circles is not None:
    # Convert the circle parameters (x, y, radius) to integers
    circles = np.uint16(np.around(circles))
    # Loop through all detected circles
    for i in circles[0, :]:
        center = (i[0], i[1])
        radius = i[2]

        # Draw the outer circle in green
        cv.circle(cimg, center, radius, (0, 255, 0), 2)
        # Draw the center of the circle in red
        cv.circle(cimg, center, 2, (0, 0, 255), 3)
    # Show Image
    cv.imshow('Detected Circles', cimg)
    cv.waitKey(0)
    cv.destroyAllWindows()
else:
    print("No circles were detected in the image.")
```

Parameters for cv.HoughCircles

- image: The input 8-bit grayscale image
- method: The detection method (must be cv.HOUGH_GRADIENT)
- dp: Inverse ratio of accumulator resolution to image resolution (determines sampling density)
- minDist: Minimum distance between detected circle centers
- param1: Upper threshold for the internal Canny edge detector
- param2: Accumulator threshold for circle centers (lower value = more circles detected, including false positives)
- minRadius: Minimum circle radius (0 = no minimum)
- maxRadius: Maximum circle radius (0 = no maximum)

2. Thresholding

The simplest method of image segmentation, where pixel values are changed based on comparison to a threshold value.

Documentation: [OpenCV Thresholding Tutorial]

2.1 Simple Thresholding

Applies a fixed threshold value to every pixel in the image.

```
import cv2 as cv
from matplotlib import pyplot as plt

# Converts image to grayscale
img = cv.imread('gradient.png', cv.IMREAD_GRAYSCALE)
assert img is not None, "file could not be read, check with
os.path.exists()"

# Apply different thresholding types
ret, thresh1 = cv.threshold(img, 127, 255, cv.THRESH_BINARY)
# pixels > 127 become 255, others 0
ret, thresh2 = cv.threshold(img, 127, 255, cv.THRESH_BINARY_INV)
# pixels > 127 become 0, others 255
ret, thresh3 = cv.threshold(img, 127, 255, cv.THRESH_TRUNC)
# pixels > 127 become 127, others unchanged
ret, thresh4 = cv.threshold(img, 127, 255, cv.THRESH_TOZERO)
# pixels > 127 unchanged, others 0
ret, thresh5 = cv.threshold(img, 127, 255, cv.THRESH_TOZERO_INV)
```

```
# pixels > 127 become 0, others unchanged

# Display the results
titles = ['Original Image', 'BINARY', 'BINARY_INV', 'TRUNC',
'TOZERO', 'TOZERO_INV']
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]
for i in range(6):
    plt.subplot(2, 3, i + 1)
    plt.imshow(images[i], 'gray', vmin=0, vmax=255)
    plt.title(titles[i])
    plt.xticks([])
    plt.yticks([])
plt.show()
```

2.2 Adaptive Thresholding

Calculates threshold values for smaller regions based on surrounding pixel intensities. Useful for images with varying illumination.

```
# Assuming 'img' is a grayscale image
# Global thresholding for comparison
ret, th1 = cv.threshold(img, 127, 255, cv.THRESH_BINARY)

# Adaptive thresholding using the mean method
th2 = cv.adaptiveThreshold(img, 255, cv.ADAPTIVE_THRESH_MEAN_C,
                           cv.THRESH_BINARY, 11, 2)

# Adaptive thresholding using the Gaussian method
th3 = cv.adaptiveThreshold(img, 255,
cv.ADAPTIVE_THRESH_GAUSSIAN_C,
                           cv.THRESH_BINARY, 11, 2)

# Parameters:
# img: source image
# 255: maximum value
# cv.ADAPTIVE_THRESH_MEAN_C: adaptive method (mean of
neighborhood)
# cv.ADAPTIVE_THRESH_GAUSSIAN_C: adaptive method (weighted sum of
neighborhood)
```

```
# cv.THRESH_BINARY: threshold type
# 11: blockSize (neighborhood size)
# 2: C (the constant to subtract)
```

2.3 Otsu's Binarization

Automatically determines the optimal global threshold value from the image histogram. Effective for images with bimodal histograms (two distinct peaks).

```
# Assuming 'img' is a grayscale image
# Global thresholding (for reference)
ret1, th1 = cv.threshold(img, 127, 255, cv.THRESH_BINARY)

# Otsu's thresholding
ret2, th2 = cv.threshold(img, 0, 255, cv.THRESH_BINARY +
cv.THRESH_OTSU)

# Otsu's thresholding after Gaussian filtering
blur = cv.GaussianBlur(img, (5, 5), 0)
ret3, th3 = cv.threshold(blur, 0, 255, cv.THRESH_BINARY +
cv.THRESH_OTSU)
```

3. Histogram Analysis

A histogram represents the distribution of pixel intensities in an image.

Documentation: [OpenCV Histogram Equalization]

```
img = cv.imread('home.jpg', cv.IMREAD_GRAYSCALE)
assert img is not None, "file could not be read, check with
os.path.exists()"

# Calculate histogram for the full image
hist_full = cv.calcHist([img], [0], None, [256], [0, 256])

# Create a mask
mask = np.zeros(img.shape[:2], np.uint8)
```

```
mask[100:300, 100:400] = 255
masked_img = cv.bitwise_and(img, img, mask=mask)

# Calculate histogram for the masked area
hist_mask = cv.calcHist([img], [0], mask, [256], [0, 256])

# Display results
plt.subplot(221), plt.imshow(img, 'gray'), plt.title('Original')
plt.subplot(222), plt.imshow(mask, 'gray'), plt.title('Mask')
plt.subplot(223), plt.imshow(masked_img, 'gray'),
plt.title('Masked Image')
plt.subplot(224), plt.plot(hist_full), plt.plot(hist_mask),
plt.title('Histograms')
plt.xlim([0, 256])
plt.show()
```

4. Foreground Extraction (GrabCut)

```
An interactive algorithm for extracting the foreground from an
image.

import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

img = cv.imread('messi5.jpg')
assert img is not None, "file could not be read, check with
os.path.exists()"

mask = np.zeros(img.shape[:2], np.uint8)
bgdModel = np.zeros((1, 65), np.float64)
fgdModel = np.zeros((1, 65), np.float64)

# 1. Initialize with a rectangle
rect = (50, 50, 450, 290)
cv.grabCut(img, mask, rect, bgdModel, fgdModel, 5,
cv.GC_INIT_WITH_RECT)
mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')
img_fg = img * mask2[:, :, np.newaxis]
plt.imshow(img_fg), plt.colorbar(), plt.show()
```

```
# 2. Refine with a mask (optional)
newmask = cv.imread('newmask.png', cv.IMREAD_GRAYSCALE)
assert newmask is not None, "file could not be read, check with
os.path.exists()"

# wherever it is marked white (sure foreground), change mask=1
# wherever it is marked black (sure background), change mask=0
mask[newmask == 0] = 0
mask[newmask == 255] = 1
mask, bgdModel, fgdModel = cv.grabCut(img, mask, None, bgdModel,
fgdModel, 5, cv.GC_INIT_WITH_MASK)
mask = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')
img_fg_refined = img * mask[:, :, np.newaxis]
plt.imshow(img_fg_refined), plt.colorbar(), plt.show()
```

5. Canny Edge Detection

```
A multi-stage algorithm to detect a wide range of edges in
images.

import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

img = cv.imread('messi5.jpg', cv.IMREAD_GRAYSCALE)
assert img is not None, "file could not be read, check with
os.path.exists()"

edges = cv.Canny(img, 100, 200)

plt.subplot(121), plt.imshow(img, cmap='gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(edges, cmap='gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
plt.show()
```

6. Image Gradients

Image gradients are used to find directional changes in intensity or color, which helps in edge detection.

Documentation: [OpenCV Gradients Tutorial]

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

img = cv.imread('dave.jpg', cv.IMREAD_GRAYSCALE)
assert img is not None, "file could not be read, check with
os.path.exists()"

# Laplacian Gradient
laplacian = cv.Laplacian(img, cv.CV_64F)

# Sobel Gradients
sobelx = cv.Sobel(img, cv.CV_64F, 1, 0, ksize=5) # X-direction
sobely = cv.Sobel(img, cv.CV_64F, 0, 1, ksize=5) # Y-direction

plt.subplot(2, 2, 1), plt.imshow(img, cmap='gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 2), plt.imshow(laplacian, cmap='gray')
plt.title('Laplacian'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 3), plt.imshow(sobelx, cmap='gray')
plt.title('Sobel X'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 4), plt.imshow(sobely, cmap='gray')
plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])
plt.show()
```

7. Video Denoising

The fastNlMeansDenoisingMulti function denoises a frame using information from preceding and succeeding frames in a video sequence.

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

cap = cv.VideoCapture('vtest.avi')
```

```
# Create a list of first 5 frames
img = [cap.read()[1] for i in range(5)]

# Convert all to grayscale and then to float64
gray = [cv.cvtColor(i, cv.COLOR_BGR2GRAY) for i in img]
gray = [np.float64(i) for i in gray]

# Create noise
noise = np.random.randn(*gray[1].shape) * 10

# Add noise to images
noisy = [i + noise for i in gray]
noisy = [np.uint8(np.clip(i, 0, 255)) for i in noisy]

# Denoise 3rd frame using the 5-frame sequence
dst = cv.fastNlMeansDenoisingMulti(noisy, 2, 5, None, 4, 7, 35)

plt.subplot(131), plt.imshow(gray[2], 'gray'),
plt.title('Original')
plt.subplot(132), plt.imshow(noisy[2], 'gray'),
plt.title('Noisy')
plt.subplot(133), plt.imshow(dst, 'gray'), plt.title('Denoised')
plt.show()
```

Additional Topics & Documentation

- **Fourier Analysis:** OpenCV Fourier Transform Tutorial
- **Morphological Transformations:** Methods for processing images based on shapes. Tutorial
- **Object Detection (ArUco):**
  - ArUco Detection
  - ArUco Board Detection
  - ChArUco Diamond Detection
- **Feature Detection:** OpenCV Feature Detection

External Libraries, Datasets, and

Repositories

Libraries & Tools

- ~~Mahotas: Computer Vision library with morphology and texture features. Documentation~~
- ~~Pymorph: Image morphology library for Python. Documentation~~
- circle-fit: A library for fitting circles to 2D data. PyPI
- anomalib: Anomaly detection library with state-of-the-art algorithms. GitHub | PyPI
- napari: A fast, interactive, multi-dimensional image viewer for Python. GitHub
- rules: A durable rules engine. GitHub
- CellProfiler: Interactive software for cell image analysis. Wikipedia

Scratch & Defect Detection Repositories

- High-quality-ellipse-detection: Ellipse detector based on arc-support line segments. GitHub
- ScratchDetection (love6tao): Weak Scratch Detection. GitHub
- Scracth-Detection-for-Automated-Optical-Inpection: AOI system for scratch detection on metal surfaces. GitHub
- segdec-net-jim2019: Surface Defect Detection with Segmentation-Decision Network. GitHub
- Surface-Defect-Detection (Charmve): A large collection of industrial defect detection datasets and papers. GitHub
- DEye: A system for defect inspection. GitHub
- AIPSDSS: Automated Image Processing for Scratch Detection on Specular Surfaces. GitHub

Anomaly Detection Repositories

- PaDiM-Anomaly-Detection-Localization-master: Unofficial implementation of the PaDiM paper. GitHub
- patchcore-ad: Unofficial implementation of PatchCore. GitHub

Datasets & Resources

- MVTec Anomaly Detection Dataset (MVTec-AD): A benchmark

dataset for anomaly detection. Website
- Detection of Scratch Defects on Metal Surfaces Based on MSDD-UNet: Research paper. MDPI
- 7 Best Python Rule Engines: Blog post on rule engines. Nected Blog
- Data-Pixel D-Scope: High-quality microscope for 2D measurement. Website
- Sensors Special Issue: Special issue on Sensors Signal Processing and Visual Computing. MDPI

Additional Links & Resources

Fiber Optic & End-Face Inspection

- Automated Inspection of Defects in Optical Fiber Connector End Face Using Novel Morphology Approaches
- Optical inspection methods for assessing fiber endface workmanship | Lightwave
- Easier fiber end-face inspection (Changes to IEC 61300-3-35) | Fluke Networks (German)

Hardware, Cameras, and Microscopy

- Acquiring images from Basler Cameras | Python For The Lab
- Basler a2A2048-37gcBAS Camera
- Camera Field Of View | Teledyne Vision Solutions
- Ring Illumination | KEYENCE America
- Illumination Systems | Nikon Instruments Inc.
- Measure Specimens & Add Scale Bars in Microscopy | Motic Microscopes

OpenCV Tutorials & Computer Vision Papers

- OpenCV: Morphological Transformations
- OpenCV: Histogram Equalization
- Sensors | Special Issue : Sensors Signal Processing and Visual Computing

Defect & Anomaly Detection Repositories

- GitHub - defect-detection-opencv-python
- GitHub - anomalib (Anomaly detection library)
- GitHub - DEye (Defects Inspection)

Load Image

`cv2.imread()`

Convert to Grayscale

`cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)`

Denoise: To minimize false detections caused by noise introduced during image acquisition, apply a smoothing filter

```
cv2.GaussianBlur(gray_image, (kernel_size, kernel_size), 0)

#kernel_size should be a tuple of odd numbers, e.g., (3, 3) or (5, 5)
```

Automatic Circle Detection

`cv2.HoughCircles()`

```
# fine-tune the minDist, param1, and param2
```

Define and Display Core/Cladding Regions

`cv2.circle()`

Display Diameters: Calculate the diameter (radius * 2) in pixels.

`cv2.putText()`

Region-Based Defects (DO2MR Method)

Maximum Filtering: Find the brightest pixel value within a local neighborhood. This operation is equivalent to morphological dilation and expands bright regions.

```
I_max = cv2.dilate(denoised_gray_image, kernel)
```

Minimum Filtering: Find the darkest pixel value within a local neighborhood. This is equivalent to morphological erosion and expands dark regions.

```
I_min = cv2.erode(denoised_gray_image, kernel)
```

Generate Residual Map: $I_r(x,y)=I_{max}(x,y)-I_{min}(x,y)$ The paper defines the residual map as the difference between the maximum and minimum filtered images This subtraction effectively highlights areas with high local contrast, which correspond to defects

```
I_residual = cv2.subtract(I_max, I_min)
```

Threshold Segmentation: Convert the residual map into a binary image to isolate defects. The paper uses a sigma-based method, where the threshold is set based on the image's mean (μ) and standard deviation (σ)

```
cv2.threshold()
```

Noise Removal: To eliminate small, isolated islands that are likely false positives, the paper suggests a final morphological opening operation.

```
cv2.morphologyEx(binary_image, cv2.MORPH_OPEN, kernel)
```

Scratch Defects (LEI Method)

Image Enhancement: To make low-contrast scratches more visible, the first step is to enhance the image using histogram equalization

```
cv2.equalizeHist(denoised_gray_image)
```

Scratch Searching: This process involves applying specially designed linear filters at multiple orientations (e.g., every 15 degrees) to detect scratches at any angle. Each filter application produces a "response map" where the response is high if the filter aligns with a scratch

`np.array()` to create the custom linear kernels for each

orientation

cv2.filter2D(): Apply each kernel to the enhanced image to generate the corresponding response map

Scratch Segmentation a threshold to each individual response map to create a binary image highlighting potential scratch segments at that specific orientation
cv2.threshold()

Result Synthesization Combine all the individual binary scratch maps into a single, comprehensive map. The paper specifies using a logical OR operation for this synthesis
cv2.bitwise_or(map1, map2)

Region-Specific Analysis

cv2.bitwise_and() to isolate the defects that fall within each zone.

cv2.findContours() and **cv2.contourArea()** on the resulting images to count the number of defects and measure their features