JORGE SILVA / LUÍS TEIXEIRA

Classic Computer Vision using OpenCV

1. Images – read, write and display; ROIs

a) Read the name of a file containing an image in 'jpg' format and show it in a window, whose name is the name of the file. Test whether the image was successfully read. Display the height and width of the image, on the console.

b) Read a color image from a file in 'jpg' format and save it in 'bmp' format.

c) Read a color image from a file, show the mouse cursor over the image, and the coordinates and RGB components of the pixel under the cursor. When the user clicks on the mouse, let him modify the RGB components of the selected pixel.

d) Read an image from a file, allow the user to select a region of interest (ROI) in the image, by clicking on two points that identify two opposite corners of the selected ROI, and save the ROI into another file.

2. Images – representation, grayscale and color, color spaces

a) Create a grayscale image, having 100(lines)x200(columns) pixels with constant intensity, 100; draw the two diagonals of the image with intensity 255. Display the image.

b) Create a color image, having 100(lines)x200(columns) pixels with yellow color; draw the two diagonals of the image, one in red color, the other in blue color. Display the image.

c) Read a color image, display it in one window, convert it to grayscale, display the grayscale image in another window and save the grayscale image to a different file.

d) Read an image (color or grayscale) and add "salt and pepper" noise to it. The number of noisy points must be 10% of the total number of image points. Suggestion: start by determining the number of image channels.

e) Read a color image, in RGB format, split the 3 channels and show each channel in a separate window. Add a constant value to one of the channels, merge the channels into a new color image and show the resulting image.

f) Read a color image, in RGB format, convert it to HSV, split the 3 HSV channels and show each channel in a separate window. Add a constant value to saturation channel, merge the channels into a new color image and show the resulting image.

3. Video – acquisition and simple processing

a) Display a video acquired from the webcam (in color) in one window and acquire and save a frame when the user presses the keyboard. Show the acquired frame in another window.

b) Display the video acquired from the webcam (in color) in one window and the result of the conversion of each frame to grayscale in another window.

c) Modify the program developed in b) so that the resulting frames are in binary format (intensity of each pixel is 0 or 255); use a threshold value of 128.

d) Implement a simple tracking algorithm for colored objects, using the following steps: 1) take each frame of the video; 2) convert from BGR to HSV color-space; 3) threshold the HSV image for a range of color values (creating a binary image); 4) extract the objects of the selected range (use a bitwise AND operation, using as operands the original and the binary image) .

SOME USEFUL LINKS:

1. https://docs.opencv.org/4.x/d0/de3/tutorial_py_intro.html
2. https://docs.opencv.org/4.x/d3/df2/tutorial_py_basic_ops.html
3. https://matplotlib.org/stable/tutorials/introductory/images.html
4. https://numpy.org/devdocs/user/quickstart.html
5. https://opencv-tutorial.readthedocs.io/en/latest/intro/intro.html
6. https://pythonexamples.org/python-opencv/
7. https://learnopencv.com/getting-started-with-opencv/
8. https://learnopencv.com/how-to-select-a-bounding-box-roi-in-opencv-cpp-python/
9. https://learnopencv.com/mouse-and-trackbar-in-opencv-gui/#mouse-annotation
10. https://docs.opencv.org/4.x/d7/dfc/group__highgui.html
11. https://learnopencv.com/color-spaces-in-opencv-cpp-python/
12. https://docs.opencv.org/4.x/df/d9d/tutorial_py_colorspaces.html
13. https://docs.opencv.org/4.x/dd/d43/tutorial_py_video_display.html
14. https://learnopencv.com/reading-and-writing-videos-using-opencv/
15. https://learnopencv.com/opencv-threshold-python-cpp/
16. https://pyimagesearch.com/category/opencv/

SOME USEFUL FUNCTIONS / ATTRIBUTES:

OpenCV
cv2.imread()
cv2.imwrite()
cv2.cvtColor()
cv2.namedWindow()
cv2.imshow()
cv2.waitKey()
cv2.destroyAllWindows()
cv2.selectROI()
cv2.rectangle()
cv2.line()
cv2.randu()
cv2.threshold()
blue,green,red = cv2.split(imgBGR) #note: BGR
hue,sat,value = cv2.split(imgHSV)
img_merged= cv2.merge((hue,sat,value)) #note: convert to BGR before display
cap = cv2.VideoCapture(0); while(...): ret, frame = cap.read(); ... ; cap.release();
NumPy
nLins, nCols, nChannels = img.shape
imgZeros = np.zeros(img.shape,np.uint8)
imgGrayConstValue = np.full((nLins,nCols), value, dtype= np.uint8)
imgColorConstValue = np.full((nLins,nCols,3), (Rvalue,Gvalue,Bvalue), dtype=np.uint8) # note: RGB
Matplotlib
plt.imshow(img); plt.title('Image XXX'); plt.xticks([]), plt.yticks([]); plt.show()
img1 = cv2.cvtColor(img, cv2.COLOR_BGR2RGB); plt.imshow(img1); plt.show(); # Matplotlib expects RGB format

4. Image enhancement – filtering

Take a noisy image (see problem **2.d**) and filter it (try different filter sizes), using:

(for **a-d** see https://docs.opencv.org/4.x/dc/dd3/tutorial_gausian_median_blur_bilateral_filter.html)

a) a mean filter;

b) a Gaussian filter;

c) a median filter;

d) a bilateral filter.

e) a filter defined by you, adapting the following code (note: remove the (1/16) factor in the assignment of kernel_3x3 and explain what happens):

```
import os
import numpy as np
from matplotlib import pyplot as plt
from cv2 import cv2
from scipy import ndimage

# Read image
datadir = 'C:/.../...' # to complete
img = cv2.imread(os.path.join(datadir,'image1.jpg')) # to be changed

# Convert to grayscale if needed
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Smooth using OpenCV GaussianBlur()
gaussianBlurred = cv2.GaussianBlur(img, (3,3), 0)

# Smooth using convolution operation coded below
kernel_3x3 = (1/16)*np.array([[1, 2, 1],[2, 4, 2],[1, 2, 1]])
print(kernel_3x3)
myConvolutionResult = ndimage.convolve(img, kernel_3x3)

# Show results
cv2.imshow("Original", img)
cv2.imshow("OpenCV Gaussian Blur", gaussianBlurred)
cv2.imshow("My 3x3 convolution w/Gaussian mask", myConvolutionResult)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

5. Image enhancement – histogram equalization

(see: https://docs.opencv.org/4.x/d8/dbc/tutorial_histogram_calculation.html
https://docs.opencv.org/4.x/d1/db7/tutorial_py_histogram_begins.html
https://docs.opencv.org/master/d5/daf/tutorial_py_histogram_equalization.html
https://www.cambridgeincolour.com/tutorials/histograms1.htm)

a) Take a low contrast grayscale image and plot its histogram.

b) Enhance the image constrast using:
b1) simple histogram equalization, or
b2) CLAHE,
and show the resulting enhanced images and their histograms.

c) Repeat the previous operations on a color image.

6. Edge detection – Sobel filter

(see: https://docs.opencv.org/4.x/d2/d2c/tutorial_sobel_derivatives.html
https://docs.opencv.org/4.x/da/d6a/tutorial_trackbar.html
https://docs.opencv.org/4.x/db/d8e/tutorial_threshold.html)

Detect the edges of an image using the Sobel filter, by implementing the following steps:

a) calculate the first derivatives of the image in **x** and **y** directions, using the **Sobel()** function;

b) calculate the approximate value of the gradient by combining the directional derivatives;

c) show the "gradient image";

d) show the result of thresholding the "gradient image"; use a trackbar to select the threshold value;

e) try different kernel sizes (see: Sobel documentation page, in OpenCV site);

f) test the effect of applying a Gaussin blur filter before applying the Sobel filter; use Gaussian filters with increasing sizes (ex: 3x3, 7x7, 11x11, 31x31).

7. Edge detection – Canny filter

(see: https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html
https://learnopencv.com/mouse-and-trackbar-in-opencv-gui/#mouse-annotation)

a) Detect the edges of an image using the **Canny()** OpenCV function. Suggestion: use trackbars to select different low and high threshold for the hysteresis procedure and different aperture size for the **Sobel()** function.

b) Compare the results of applying the following two filters to the same image:
- Sobel filter, with threshold **t**, after smoothing the image with a Gaussian blur filter with size **s**;
- Canny filter, with "low threshold" = "high threshold" = **t** and "aperture" = **s**,
using the same **t** and **s** values. Try also with a "low threshold" different from the "high threshold".

8. Hough transform – line and circle detection

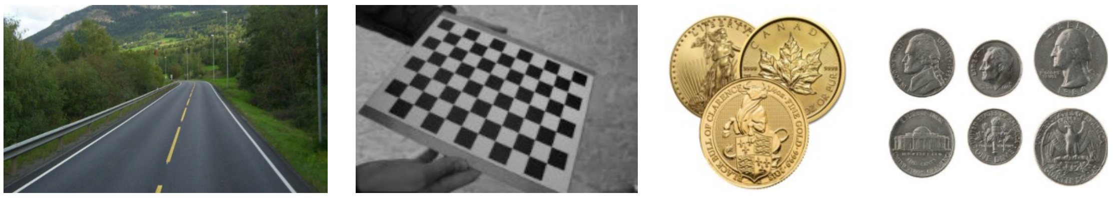
a) Compare the functionality of **HoughLines()** and **HoughLinesP()** OpenCV functions for line detection.

(see: https://docs.opencv.org/4.x/d3/de6/tutorial_js_houghlines.html
https://docs.opencv.org/4.x/d9/db0/tutorial_hough_lines.html
https://learnopencv.com/hough-transform-with-opencv-c-python/)

b) Use **HoughLines()** to detect lines in images like those in figure 1.a and 1.b; try different parameter values; draw the detected lines on the image.

c) Use **HoughLinesP()** to detect line segments in the same images that you used in the previous problem; try different parameter values; draw the detected line segments on the image.

d) Use **HoughCircles()** to detect the coins present in images like those in figure 1.c and 1.d (without or with superposition among the coins).



a) b) c) d)

Figure 1

9. Local features – corner detectors

a) Use the **Harris** corner detector to detect the corners in one or more images of your choice. Try different parameters, namely, neighborhood size, aperture for the Sobel operator and Harris detector free parameter.
(see: https://docs.opencv.org/4.x/dd/d1a/group__imgproc__feature.html#gac1fc3598018010880e370e2f709b4345)

b) Use the **Shi-Tomasi** corner detector to detect the corners in the same images. Try different parameters, namely, block size, and aperture for the Sobel operator; compare the results with those of Harris detector for the equivalent parameters; notice that the Shi-Tomasi implementation has more parameters than Harris implementation.
(see: https://docs.opencv.org/4.x/d8/dd8/tutorial_good_features_to_track.html
https://docs.opencv.org/4.x/d4/d8c/tutorial_py_shi_tomasi.html)

c) Use the **FAST** corner detector to detect the corners in the same images.
(see: https://docs.opencv.org/4.x/df/d0c/tutorial_py_fast.html
https://docs.opencv.org/4.x/df/d74/classcv_1_1FastFeatureDetector.html)

d) Resize the images, downsizing them to 1/4 of the original size and reapply all the corner detectors. Compare the results with each other and with those that you have obtained previously.

10. Local features – blob detectors

a) Use the **SIFT** blob detector to detect the keypoints in an image and show their position over the image (note: to visualize the circle associated with each keypoint with a different color and a different radius - corresponding to the scale of the keypoint – as well as the associated orientation, use the following call: cv.drawKeypoints(img,keypoints,img,(-1,-1,-1), cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)).
(see: https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html
https://docs.opencv.org/4.x/d7/d60/classcv_1_1SIFT.html)

b) Use the **ORB** blob detector to detect the keypoints in an image and show their position over the image. (note: if your program breaks with a "Python has stopped working" message, include the following statement before ORB execution: **cv.ocl.setUseOpenCL(False)**)
(see: https://docs.opencv.org/4.x/db/d95/classcv_1_1ORB.html
https://docs.opencv.org/4.x/d1/d89/tutorial_py_orb.html)

11. Local features – matching

Detect the keypoints in two different images (ex: taken from different viewpoints) and draw the found matches. Try different sets of keypoint detectors, matchers and visualization of the matches:

- **SIFT** or **ORB** for keypoint detection;
- **Brute Force** or **FLANN** for matching, with the alternatives **match()** or **KnnMatch()**; choose an appropriate distance measure;
- **drawMatches()** (⇒match()) or **drawMatchesKnn()** (⇒KnnMatch()) for displaying the results.

(see: https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html)

12. Local features + Homography – localizing objects

Develop a program that tries to detect a given object in an image, delineating the countour of the object in the image (figure 2). Use one of the methods used in the previous exercises to extract and match keypoints, and the functions **cv::findHomography()**, to find the transform between matched keypoints, and **cv::perspectiveTransform()**, to map the points of the contour of the object to the image (note: in the case of figure 2, it is enough to map the 4 corners of the object, then draw the lines that connect the mapped corners).
(see: https://docs.opencv.org/4.x/d7/dff/tutorial_feature_homography.html)

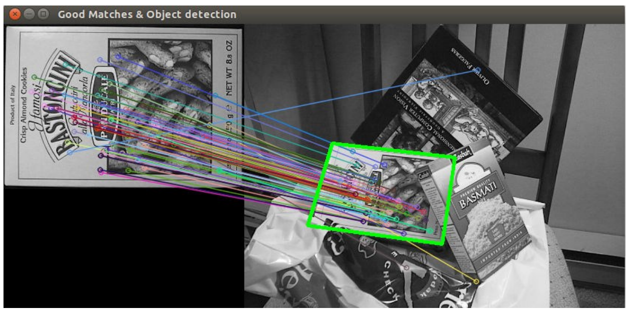


Figure 2