Stavanger, August 30, 2024

University of Stavanger

Faculty of Science and Technology

ELE610 Applied Robot Technology, H-2024

Image Acquisition assignment 1

The purpose of this assignment is to install Python and packages and prepare your Python programming environment and refresh Python skills. The IDS camera is not used in this assignment. You should capture an image and transfer it to the computer. Then you install and check the Python software, in particular the OpenCV package, and you do some basic image processing. You should refresh `numpy` and `OpenCV` as good as possible within the time limit.

For this assignment each group should write a brief report (pdf-file). Answer the questions in the alphabet-numbered lists in each subsection and include figures and images as appropriate. You may answer in Norwegian or English, or a mix.

Note that to give exhaustive answers to all questions and do all tasks below within reasonable time is impossible for most students. The intention here is that you should do as much as you are able to within the time limit for each student for each assignment, which is 15-20 hours. A report containing a table showing time used, for each student of the group, will normally be accepted even if all tasks are not done. If all tasks are done, the time report does not need to be included.

1 Image acquisition using smart phone

The first task is to look up technical information on the smart phone camera, and to understand what the different properties means. Note that not all listed properties are available or relevant for the smart phone camera. Then you should capture an image using the camera on your smart phone and transfer it to your laptop.

Finally some simple image processing should be done. The goal is to be able to use Python and OpenCV for some basic image processing tasks.

Karl Skretting, Department of Electrical Engineering and Computer Science (IDE), University of Stavanger (UiS), N-4036 Stavanger. E-mail: karl.skretting@uis.no.

1.1 Camera properties

Many years ago, when cameras were mechanical, there was some few common properties to adjust when an image was captured, mainly:

- exposure time ↗ as fractions of second (lukketid),
- the physical shutter ↗ mechanism (lukker),
- aperture ↗ size as f-number (blender).

When digital cameras were introduced more properties (both for camera and image capture) was added, but the old properties related to the lens and objective are still relevant.

The task is simply to look up and read the technical information for the cameras relevant for you in this course. They are

a. The small IDS uEye XS ↗ camera.

b. One of the other IDS camera, either the camera model IDS `UI-3140CP-C-HQ R2` ↗ or the camera model IDS `UI-336xCP-C` ↗. They are mounted on each of our two camera rigs.

c. You should also check the properties of the camera on your own cell phone.

The report should only contain what you consider to be the five most important, most relevant, properties with the corresponding values for each camera.

1.2 Capture a test image

The task here is to capture an image using the camera on your smart phone. The scene should be a simple chessboard pattern as in figure 1.

Transfer the image to your laptop, or another computer, and view it in an image viewer program, or MATLAB, to explore the image. Answer the points below, a chessboard pattern is assumed.

a. Store the image on your computer, as a jpg- or png-file, and include it in your report.

b. What meta information is available for the image.

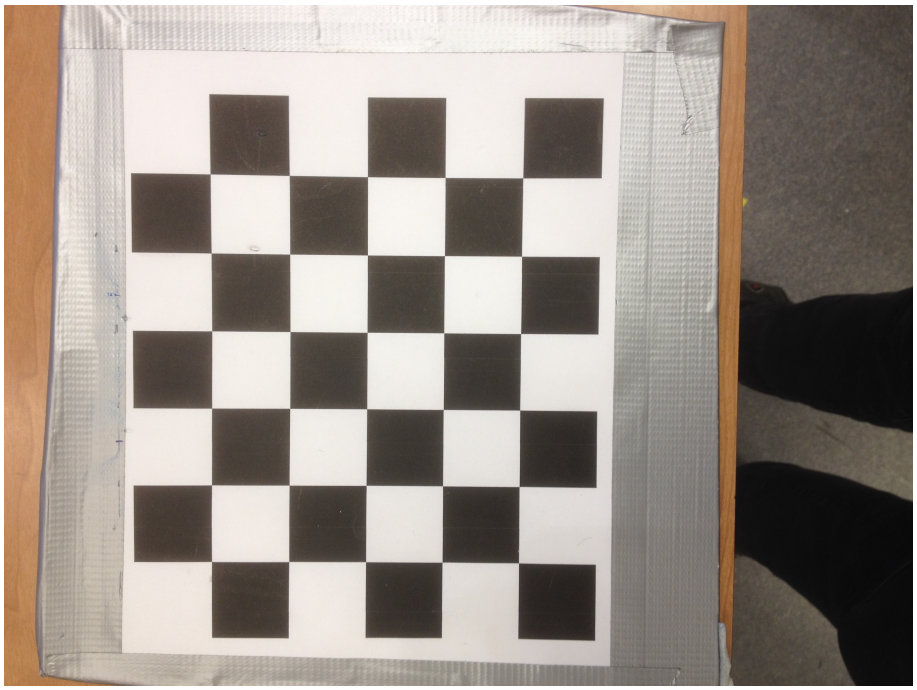

Figure 1: Image from Iphone 4, width 3264 and height 2448 pixels. The size of a square is 30x30 mm.

c. See the image in figure 1. I don't remember the actual distance from the camera to the chessboard pattern, but let us here assume it is $d_1 = 400$ mm. One square in figure 1 is measured to be $p_1 = 285$ pixels wide, this was found by using `appImageViewer1.py` to display the image, here the location, x- and y-coordinates, for the pixel that the arrow (cursor) points on is shown.

   Assume I now capture a new image of this chessboard pattern, using the same camera setup but another distance from camera to the pattern, and measure the side length of a square in the image as $p_2 = 406$ pixels. What is the distance d_2 now from the camera to the chessboard pattern? The report should include the calculated distance and the formula used.

1.3 Install and check Python

The first thing to do here, if it is not already done, is to install Python, including all the needed packages on your laptop computer. Python should be installed on the office computers in E462 and E464. To do this, carefully read the instructions in the Fragments of Python stuff ↗ document. Make the decisions appropriate for you, do the necessary installations, and check that everything is as it should be. To confirm the installation answer the following questions.

a. What kind of computer do you use?

b. What kind of OS do you use?

c. Which version of Python (`sys version`) do you use?

d. Which editor, or IDE, do you use?

e. Which version of `numpy` do you use?

f. Which version of OpenCV do you use?

g. Which version of Qt do you use?

h. Have you installed and checked `pyueye`?

i. Have you installed and checked `qimage2ndarray`?

1.4 Image Processing using Python and OpenCV

This part is divided into several steps: First, it is important to understand how an image is represented in numpy and OpenCV. The purpose here is mainly to learn to use `numpy`, and some basic OpenCV stuff. Then, some basic image processing is explained, mainly edge detection and line detection (Hough Transform). Finally, edges and corners should be located in the example image. You should be familiar to `numpy` and OpenCV documentation, you may search the web and also some links are included in fragments of Python stuff ↗ document.

1.4.1 Test image in OpenCV

Use OpenCV to read the test image of a chessboard pattern captured and stored on your computer, task 1.2.a above. The Python code may be as below.

```
img = cv2.imread( filename )  # filename is a string with the name of the file
print( f"{img.dtype = },  {img.size = }, {img.ndim = },  {img.shape = }" )
```

You should now crop this (probably) large image and keep a central part containing at least 3 black squares. That is, you should select only a part of the image. Then you store (write to disk) this resulting image, use the same format as the original file. Use a print-statement as above to print the properties for the cropped, and now stored, image.

a. Include the result of the print-statement above in your report.

b. Also, include the result of the print-statement for the cropped image in your report.

c. Give the size (in bytes) for the original large image and for the cropped image as stored on your computer.

1.4.2 Image as a numpy array

This subsection is to make you more familiar with OpenCV and `numpy`. Nothing in this part needs to be included in the report.

OpenCV use `numpy` array for images. A gray scale image with 400 pixel in horizontal direction (w or width) and 300 pixels in vertical direction (h or height) where all pixels are zeros (black) can be done by the following Python code. You should try the code below and hopefully a black rectangle will be shown.

```
import numpy as np
import cv2
from bf_tools import mpl_plot, cv_plot
#
A = np.zeros((300,400),dtype=np.uint8)
cv_plot(A, 'All black image')
mpl_plot(A, 'All black image')
```

The code above plot the image `A` using two different functions. You may look in the file `bf_tools.py`, included in ELE610py3files.zip ↗, to see exactly how this is done.

Well, an all black image is dull. The next step is to draw some simple shapes into image `A`. To do this you need to know the reference system. The upper left pixel is referred as `A[0,0]`, row 10 as `A[9,:]`. The axis system for the image has origin in upper left pixel and x-axis to the right and y-axis downward. A point is represented as `(x,y)` where $0 \leq x < w$ and $0 \leq y < h$ where `(w,h)` are width and height. This point is in pixel `A[int(y), int(x)]`. Note order of indexes and that `int()` round towards zero.

You can use OpenCV functions to draw some simple shapes into image `A`, see OpenCV Drawing Functions ↗ or the more general OpenCV tutorial ↗. Some examples you may try are:

```
A = np.zeros((300,400),dtype=np.uint8)
A = cv2.line(A, (10,10), (10,50), 255)
A = cv2.rectangle(A, (20,10), (50,60), 255, thickness=-1)
A = cv2.ellipse(A, (120,50), (40,40), angle=360, \
    startAngle=0, endAngle=360, color=255, thickness=2)
mpl_plot(A, 'Black image with some white shapes')
```

You may also set individual pixels, or a range of pixels, directly into the array using `numpy`. See `numpy` documentation for details on array slicing, indexing

and manipulation, there are a lot of possibilities. The code below makes a white disk on the black image, i.e pixels inside the circle are white (value 255) but pixels on the edge are gray. This is denoted as anti-aliasing and makes the disk edge look smoother. The `numpy`-code is below.

```
(cx,cy,r) = (100,200,40)
yRange = np.arange( np.floor(cy-r-1.0), np.ceil(cy+r+1.0001), 1.0)
xRange = np.arange( np.floor(cx-r-1.0), np.ceil(cx+r+1.0001), 1.0)
yD2 = np.power(abs(yRange + 0.5 - cy), 2)
xD2 = np.power(abs(xRange + 0.5 - cx), 2)
xyD = np.sqrt(np.dot(yD2.reshape(len(yD2),1), np.ones((1,len(xD2)))) +
              np.dot(np.ones((len(yD2),1)), xD2.reshape(1,len(xD2))))
#
for y in range(len(yD2)):
    for x in range(len(xD2)):
        b = xyD[y,x]
        if (b < (r-0.7)):
            xyD[y,x] = 255
        elif (b > (r+0.7)):
            xyD[y,x] = 0
        else:     # r-0.7 < b < r+0.7,   0 < r+0.7-b < 1.4
            xyD[y,x] = int(floor(182.1*(r+0.7-b)))
#
A[int(yRange[0]):int(yRange[-1]+1),int(xRange[0]):int(xRange[-1]+1)] = xyD
mpl_plot(A, 'Gray scale image with some more shapes')
```

1.4.3 Make an image with some simple shapes

Now you should start with a new black image `A` and add the following shapes to image `A`. When done it should look something like the image in figure 2.

a. Make a circle in upper left quadrant of the image A, center at x=140 and y=90 and radius=40.

b. Add a filled white rectangle where upper corner is in (x,y) = (250,50) and size is (w,h) = (100,50), i.e. in upper right quadrant of the image.

c. Add a line from (100,200) to (200,250), i.e. in lower left quadrant of the image.

d. Add a (filled) triangle with corners at points (x,y): (250,250), (350,250) and (300,163.4), i.e in lower right quadrant of image. You don't need to do "anti-aliasing" as in the `numpy` circle example. But, if you want some extra challenges, you may do this too.

e. Save/store (write to disk) the resulting image (as a png-, bmp- or jpg-file).

f. Include the resulting image in the report.

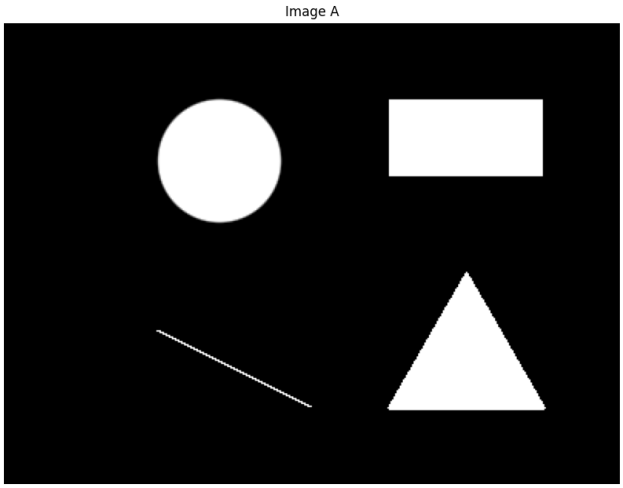

Figure 2: Synthetic image as made in part 1.4.1. The image is gray scale and most pixels have value 0 (black) or 255 (white), so it is almost an binary image.

1.4.4 Image edges and image lines

To solve the problems below you should use the OpenCV documentation ↗, and perhaps also the tutorials, whenever needed. Especially the `imread, cvtColor, Sobel, sepFilter2D, threshold, Canny, HoughLines, HoughLinesP, line, ...` documentation will be useful here. You may also use the program `appImageViewer1.py` ↗ to see how `HoughLines` and `HoughLinesP` functions works.

The appImageViewer-programs will be useful and needed to solve the following tasks and tasks in following assignments as well. To understand the appImageViewer-programs, **carefully read** the overview in the Fragments of Python stuff ↗ document. For this, the most important section is section 5.3 and figure 1. There are some video lectures available on *canvas*, in Modules on ELE610 the course page. I strongly recommend to **watch these videos** because they should be very helpful in understanding the appImageViewer-programs. Also, I encourage you to look at the implementation details in the Python code, all py-files should be included in `ELE610py3files.zip` ↗.

The tasks listed below should be tested on the cropped image from section 1.4.1. You could make a Python function that takes the task letter and an image filename as the input, and then does the selected task. An even better solution could be to integrate the tasks in the menu of your own image viewer program, ex. `appImageViewer1X.py`, where `X` is replaced by your group letter.

For this latter approach many of the tasks below are already done, and for those tasks you only need to try each of them in the `appImageViewer1.py` program, understand what is done and how the code works in the program.

If you reach the limit for the assignment time budget (20 hours) working on these tasks, you may skip the remaining tasks from the list below. Remember to include an hour list in the report.

a. Open the image file, and print the properties of the resulting numpy array.

b. Display the image on screen using OpenCV function(s). Now you should try `cv2.imshow(..)` and `cv2.waitKey(..)`, but if this does not work you may use `matplotlib.pyplot` as in section 1.4.2. You should include a screen-dump (or window-dump) in the report.

c. Observe what color space the image representation is, RGB or BGR? If it is wrong, you should swap Red and Blue color components and redisplay the image. This option is actually implemented in the `appImageViewer3.py` program, and the swap is fairly easy. If color seems to be ok, or if the gray scale image in next task is ok you don't need to do more than observe color space in this task. You may extend the - File - print Info - menu point in `appImageViewer1.py` program to also print this information for the self.npImage object.

d. Convert the image into a gray scale image using OpenCV function. Documentation on `Matplotlib: colormaps` ↗ and OpenCV: Color Space Conversions ↗ may be helpful.

e. Display the gray scale image on screen and include it in the report.

f. Make and display a histogram of the pixel values in the gray scale image. Include the resulting histogram plot in the report. You may see histogram tutorial ↗.

g. Explain the difference between image edges and image lines.

h. Emphasize image edges using the `Sobel` filter. Display the resulting image on screen and include it in the report.

i. Threshold the gray scale image (not the filtered one from previous task) into a binary image. OpenCV thresholding ↗ tutorial may be helpful.

j. Locate lines using OpenCV `HoughLines` function. Plot the lines into the image and include the resulting image in the report. The `draw_lines` function in `HoughLines` ↗ tutorial may be useful.
Or look in the `draw_lines` function in `clsHoughLinesDialog.py`, included in `ELE610py3files.zip` ↗.

The "find Lines" menu point in `appImageViewer1.py` also includes the `HoughLinesP` function as an option, you should try this too.

k. This task should only be done by students that want an extra challenge. The challenge is to locate corners in the image. You may add a new option, i.e. menu point, in your `appImageViewer1X.py` program. The easiest way is probably to use the Harris corner detector in OpenCV, see the tutorial ↗. An alternative is to find points where lines, approximate 90 degrees apart, cross each other. Yet another alternative when the edges are horizontal and vertical is to use the "Filter image" option in the `appImageViewer1X.py` program. You should try this, filtering doesn't need a binary image but this implemetation doesn't works on a color image.

l. If you still want more challenges you may try to rotate, by 30, 45 or 60 degrees, the original large image from section 1.2, using OpenCV as in this rotate tutorial ↗. Some image editors (and Matlab `imrotate`) have this feature. You may add an `Image rotate` option in your `appImageViewer1X.py` program.