Image Acquisition assignment 1

Antón Maestre Gómez and Daniel Linfon Ye Liu

1. Image acquisition using smart phone

1.1 Camera properties

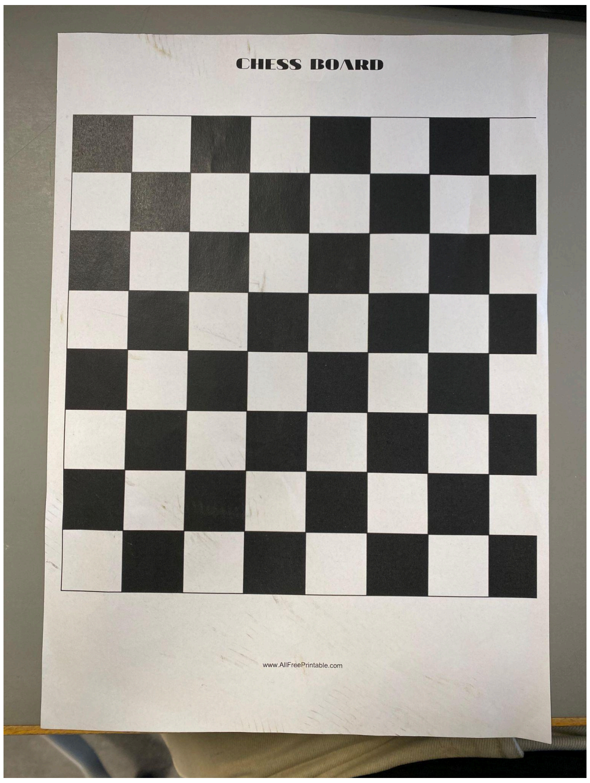
- **IDE uEye XS Rev.1.1**:
    - **Resolution**: 2592 x 1944 pixels
    - **Shutter**: Rolling Shutter
    - **Frame rate**: 15.0 fps
    - **Pixel size**: 1.4 µm
    - **Optical class**: 1/4"

- **IDS UI-3140CP-C-HQ R2**:
    - **Resolution**: 1280 x 1024 pixels
    - **Shutter**: Global Shutter
    - **Frame rate**: 169.0 fps
    - **Pixel size**: 4.8 µm
    - **Optical class**: 1/2"

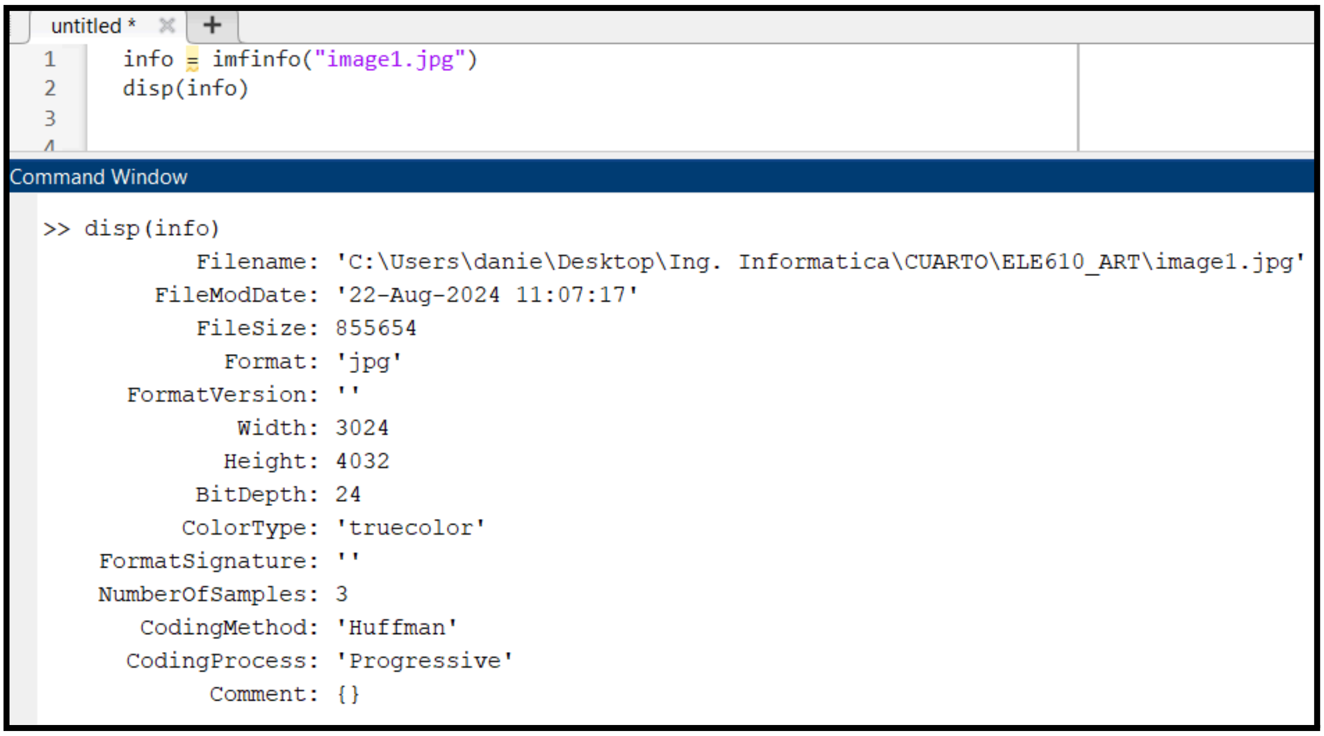
- **Iphone 11**:
    - **Resolution**: 4032 x 3024 pixels
    - **Shutter**: Electronic Shutter
    - **Frame rate**: Up to 60.0 fps for 4K video recording
    - **Pixel size**: 1.7 µm (wide) and 1.0 µm (ultra-wide)
    - **Optical class**: 1/2.55" sensor for wide and 1/3.6” sensor for ultra wide

1.2 Capture a test image

a)



b) We find the meta information In MATLAB:



```
info = imfinfo("image1.jpg")
disp(info)
```

```
>> disp(info)
           Filename: 'C:\Users\danie\Desktop\Ing. Informatica\CUARTO\ELE610_ART\image1.jpg'
        FileModDate: '22-Aug-2024 11:07:17'
           FileSize: 855654
             Format: 'jpg'
      FormatVersion: ''
              Width: 3024
             Height: 4032
           BitDepth: 24
          ColorType: 'truecolor'
    FormatSignature: ''
    NumberOfSamples: 3
       CodingMethod: 'Huffman'
      CodingProcess: 'Progressive'
            Comment: {}
```

c) The relationship between the distances and the pixel sizes can be derived from the fact that the size of the image of the chessboard squares on the sensor is inversely proportional to the distance from the camera. The formula to find the new distance $d2$ from the camera to the chessboard pattern is derived from the proportionality:

$$\boxed{p1/d1 \ = \ p2/d2}$$

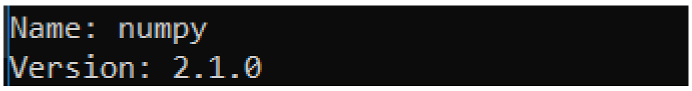
Where:

- ➔ d1 = 400 mm. Original distance from the camera to the chessboard pattern
- ➔ p1 = 285 pixels. Size of the square in pixels in the original image
- ➔ d2 = New distance from the camera to the chessboard pattern
- ➔ p2 = 406 pixels. Size of the square in pixels in the new image

$$d2 \ = \ d1 \ * \ p2/p1 \rightarrow d2 \ = \ 400 \ * \ 406/285 \rightarrow \boxed{d2 \ = \ 569.84\,mm}$$

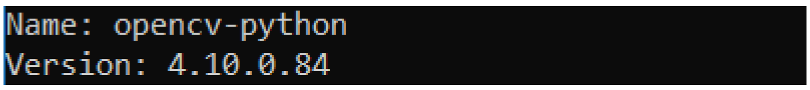
1.3 Install and check Python

- **a. What kind of computer do you use?**
  Lenovo ThinkPad (Daniel) & Lenovo ideapad 100-15IBD (Antón)
- **b. What kind of OS do you use?**
  Windows 10 Pro
- **c. Which version of Python (sys version) do you use?**
  Python 3.12
- **d. Which editor, or IDE, do you use?**
  Visual Studio Code
- **e. Which version of numpy do you use?**



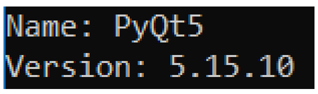
```
Name: numpy
Version: 2.1.0
```

- **f. Which version of OpenCV do you use?**



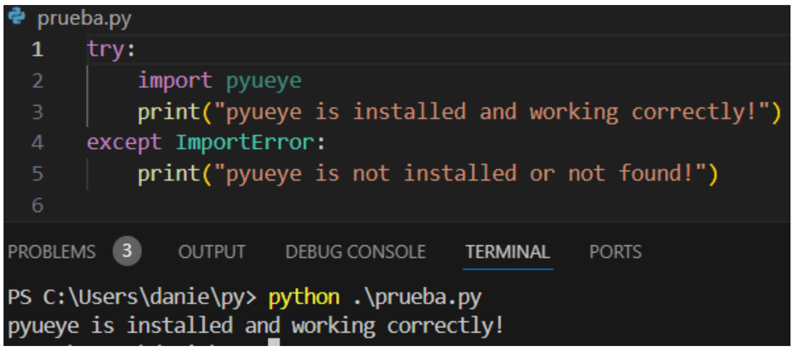
```
Name: opencv-python
Version: 4.10.0.84
```

- **g. Which version of Qt do you use?**



```
Name: PyQt5
Version: 5.15.10
```

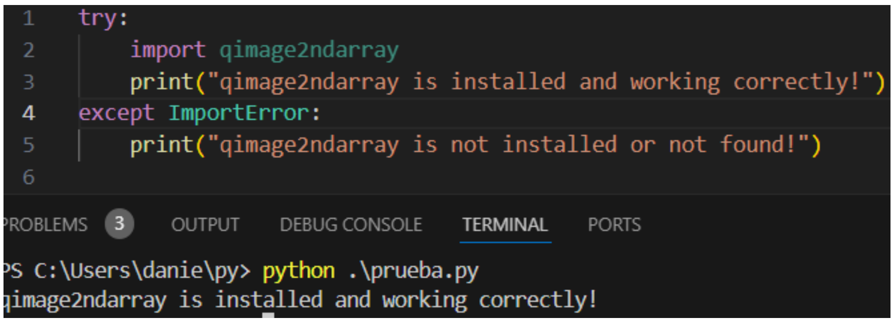
h. **Have you installed and checked pyueye?**



```
prueba.py
try:
    import pyueye
    print("pyueye is installed and working correctly!")
except ImportError:
    print("pyueye is not installed or not found!")
```

```
PS C:\Users\danie\py> python .\prueba.py
pyueye is installed and working correctly!
```

i. **Have you installed and checked qimage2ndarray?**



```
try:
    import qimage2ndarray
    print("qimage2ndarray is installed and working correctly!")
except ImportError:
    print("qimage2ndarray is not installed or not found!")
```

```
PS C:\Users\danie\py> python .\prueba.py
qimage2ndarray is installed and working correctly!
```

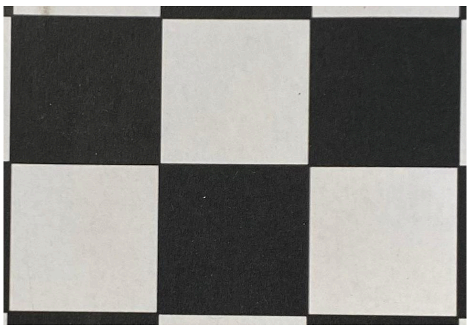
1.4 Image Processing using Python and OpenCV

1.4.1 Text image in OpenCV

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

img.dtype = dtype('uint8'), img.size = 36578304, img.ndim = 3, img.shape = (4032, 3024, 3)

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



img.dtype = dtype('uint8'), img.size = 1936968, img.ndim = 3, img.shape = (667, 968, 3)

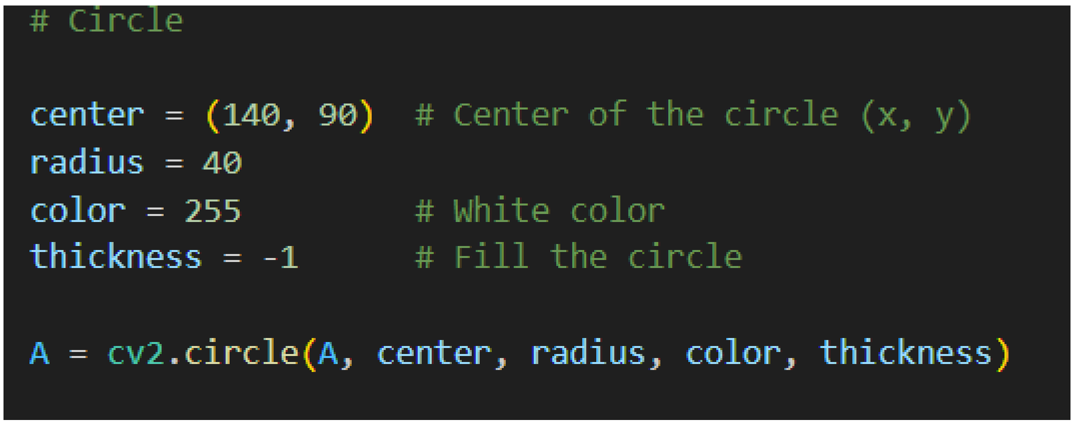
c. Give the size (in bytes) for the original large image and for the cropped image as stored on your computer.

Large image = 835 KB (855.654 bytes)

Cropped image = 56,5 KB (57.882 bytes)

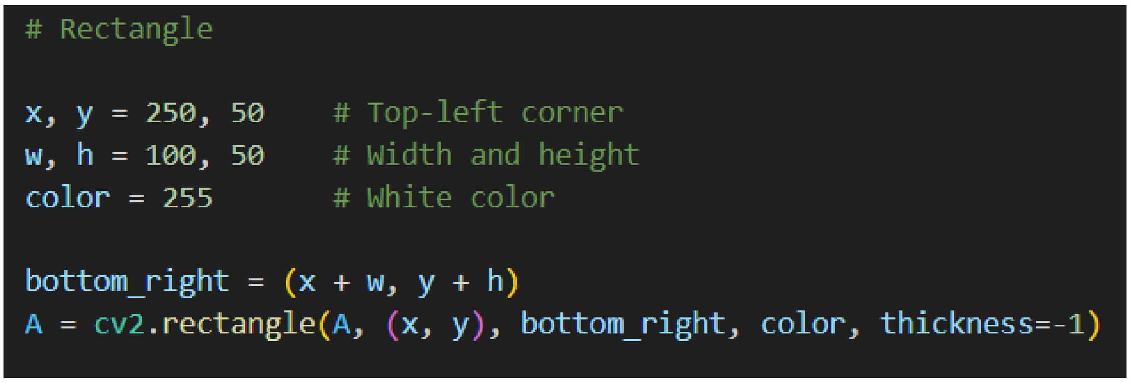
1.4.3 Make an image with some simple shapes

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



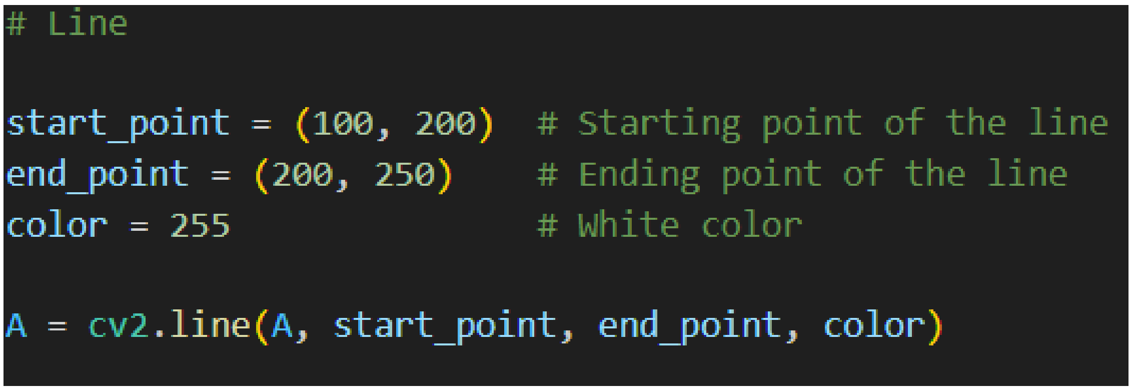
```python
# Circle

center = (140, 90)  # Center of the circle (x, y)
radius = 40
color = 255         # White color
thickness = -1      # Fill the circle

A = cv2.circle(A, center, radius, color, thickness)
```

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



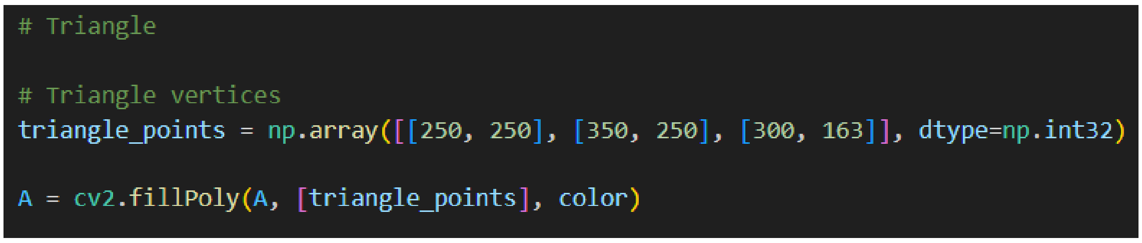
```python
# Rectangle

x, y = 250, 50     # Top-left corner
w, h = 100, 50     # Width and height
color = 255        # White color

bottom_right = (x + w, y + h)
A = cv2.rectangle(A, (x, y), bottom_right, color, thickness=-1)
```

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



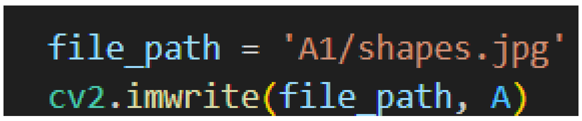
```python
# Line

start_point = (100, 200)  # Starting point of the line
end_point = (200, 250)    # Ending point of the line
color = 255               # White color

A = cv2.line(A, start_point, end_point, color)
```

D. Add a (filled) triangle with corners at points (x,y): (250,250), (350,250) and (300,163.4), i.e in the lower right quadrant of the 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.



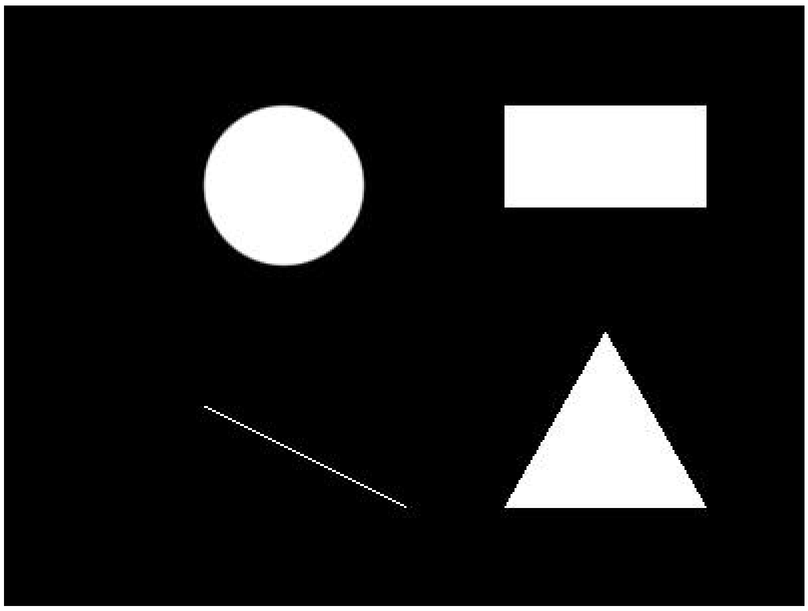
```
# Triangle

# Triangle vertices
triangle_points = np.array([[250, 250], [350, 250], [300, 163]], dtype=np.int32)

A = cv2.fillPoly(A, [triangle_points], color)
```

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



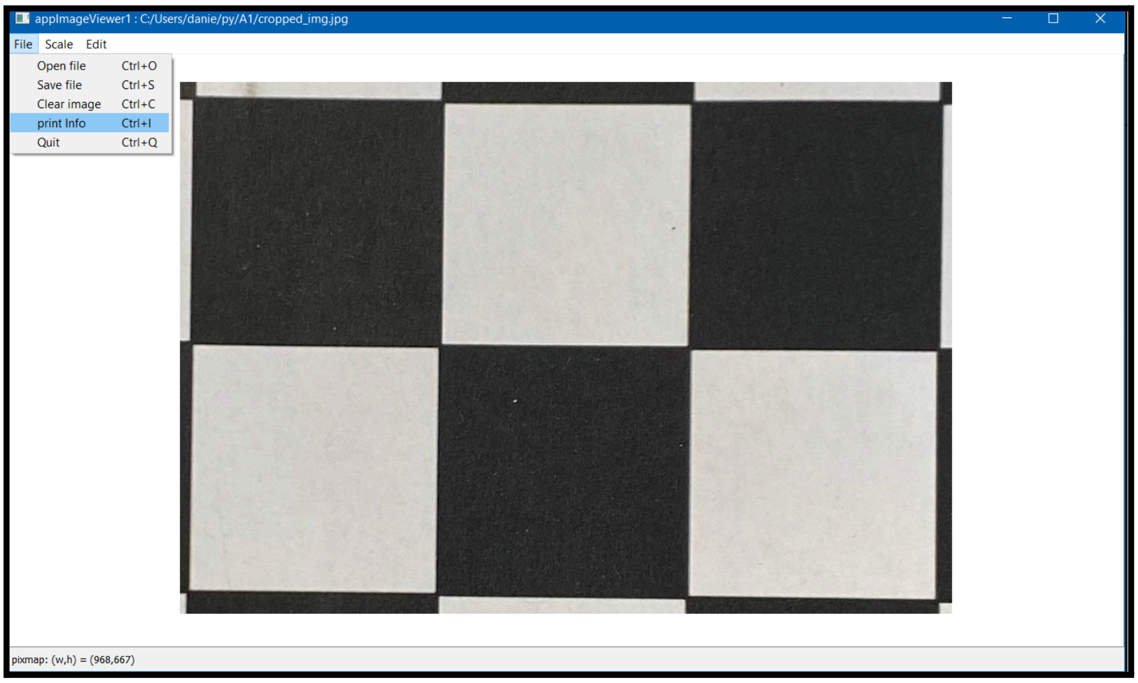
```
file_path = 'A1/shapes.jpg'
cv2.imwrite(file_path, A)
```

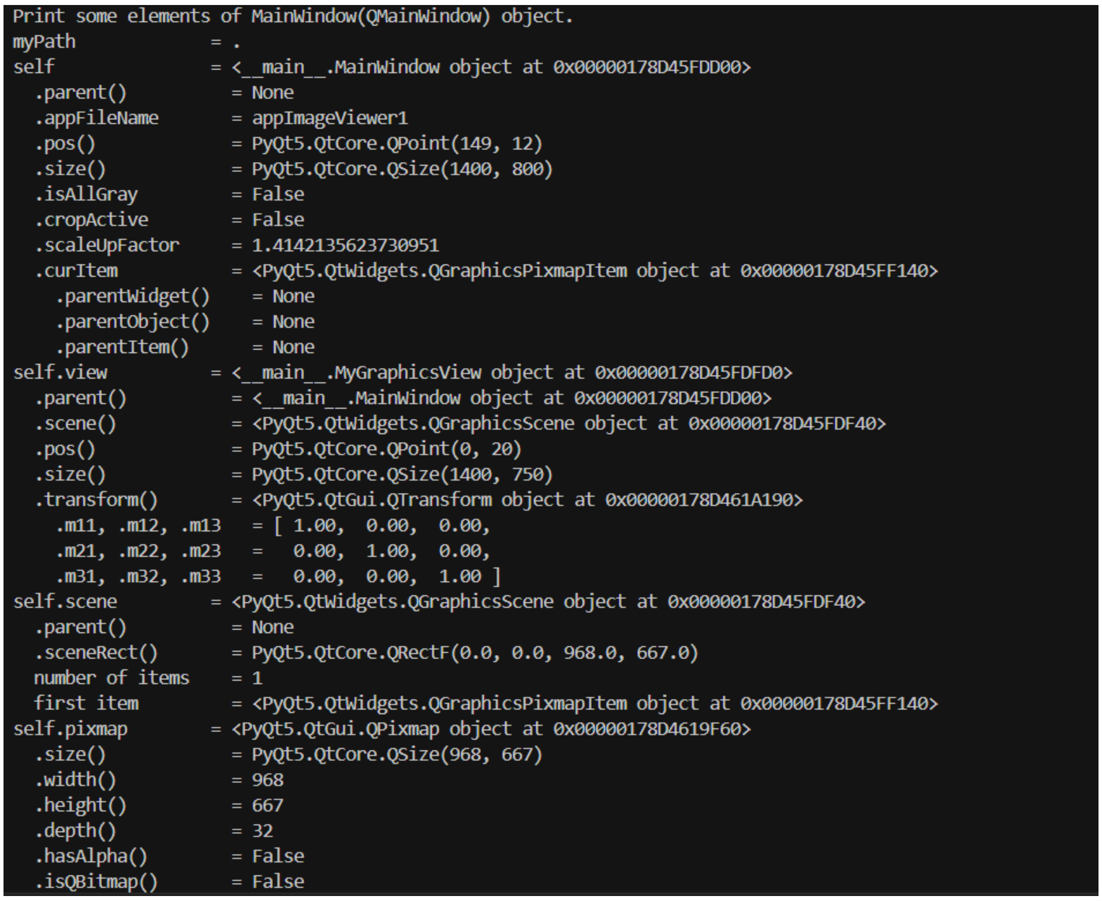
F. Include the resulting image in the report.



1.4.4 Image edges and image lines

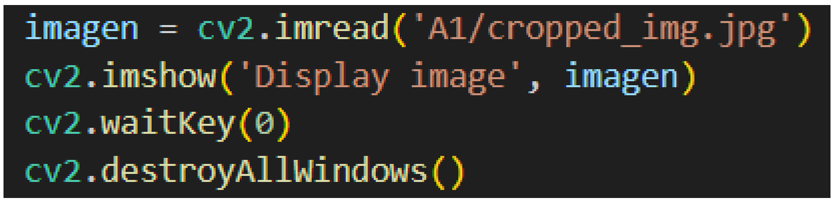
A. Open the image file, and print the properties of the resulting numpy array.



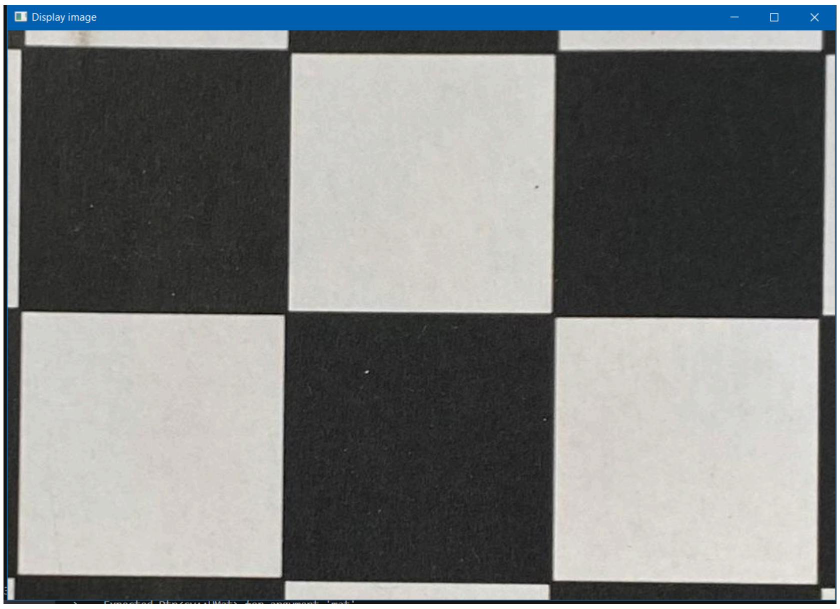


```
Print some elements of MainWindow(QMainWindow) object.
myPath              = .
self                = <__main__.MainWindow object at 0x00000178D45FDD00>
  .parent()           = None
  .appFileName        = appImageViewer1
  .pos()              = PyQt5.QtCore.QPoint(149, 12)
  .size()             = PyQt5.QtCore.QSize(1400, 800)
  .isAllGray          = False
  .cropActive         = False
  .scaleUpFactor      = 1.4142135623730951
  .curItem            = <PyQt5.QtWidgets.QGraphicsPixmapItem object at 0x00000178D45FF140>
    .parentWidget()     = None
    .parentObject()     = None
    .parentItem()       = None
self.view           = <__main__.MyGraphicsView object at 0x00000178D45FDFD0>
  .parent()           = <__main__.MainWindow object at 0x00000178D45FDD00>
  .scene()            = <PyQt5.QtWidgets.QGraphicsScene object at 0x00000178D45FDF40>
  .pos()              = PyQt5.QtCore.QPoint(0, 20)
  .size()             = PyQt5.QtCore.QSize(1400, 750)
  .transform()        = <PyQt5.QtGui.QTransform object at 0x00000178D461A190>
    .m11, .m12, .m13    = [ 1.00,  0.00,  0.00,
    .m21, .m22, .m23    =   0.00,  1.00,  0.00,
    .m31, .m32, .m33    =   0.00,  0.00,  1.00 ]
self.scene          = <PyQt5.QtWidgets.QGraphicsScene object at 0x00000178D45FDF40>
  .parent()           = None
  .sceneRect()        = PyQt5.QtCore.QRectF(0.0, 0.0, 968.0, 667.0)
  number of items     = 1
  first item          = <PyQt5.QtWidgets.QGraphicsPixmapItem object at 0x00000178D45FF140>
self.pixmap         = <PyQt5.QtGui.QPixmap object at 0x00000178D4619F60>
  .size()             = PyQt5.QtCore.QSize(968, 667)
  .width()            = 968
  .height()           = 667
  .depth()            = 32
  .hasAlpha()         = False
  .isQBitmap()        = False
```

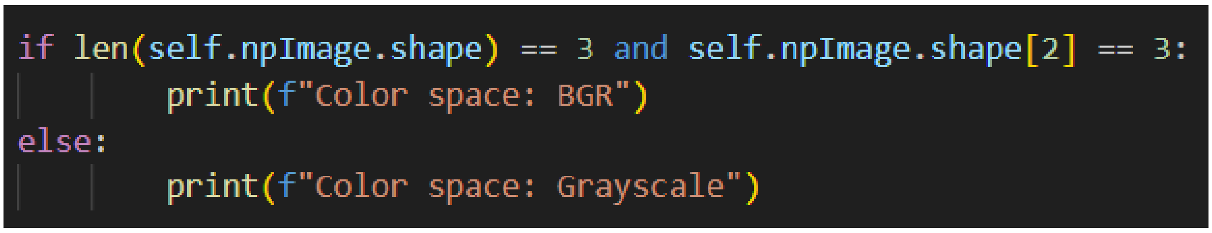
B. **Display the image on screen using OpenCV function(s). Now you should try cv2.imshow(..) and cv2.waitKey(..), but if this does not work you may use matplotlib.pyplot as in section 1.4.2. You should include a screen-dump (or window-dump) in the report.**



```
imagen = cv2.imread('A1/cropped_img.jpg')
cv2.imshow('Display image', imagen)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

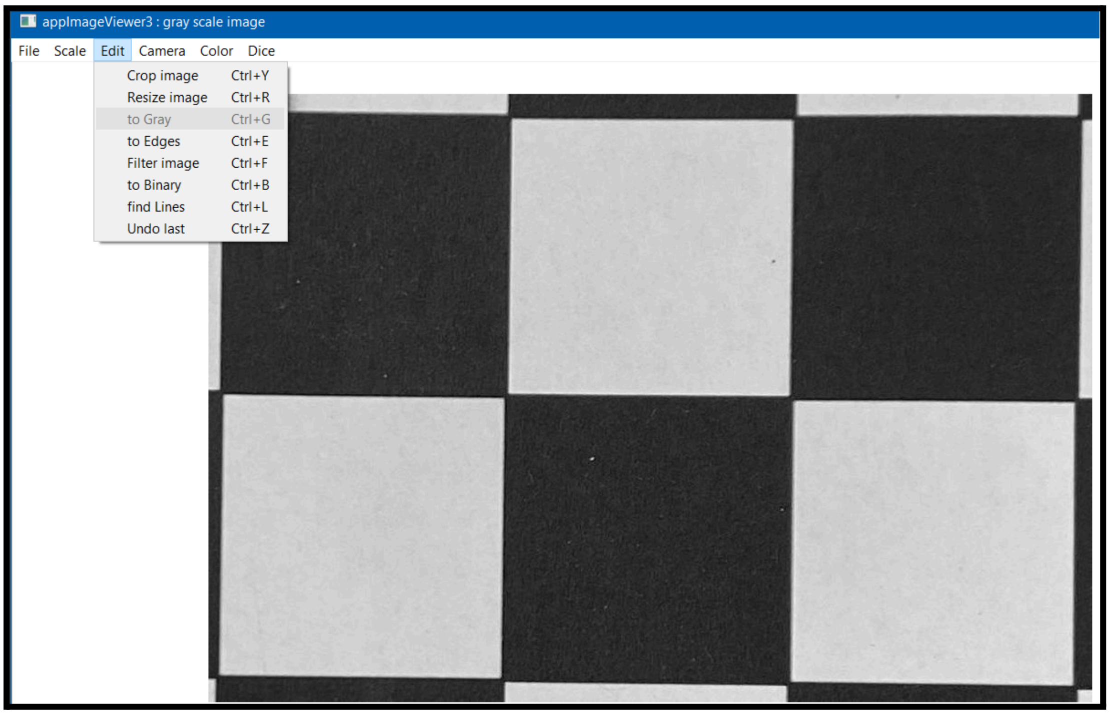


C. **Observe what color space the image representation is, RGB or BGR? If it is wrong, you should swap Red and Blue color components and redisplay the image. This option is actually implemented in the applmageViewer3.py program, and the swap is fairly easy. If color seems to be ok, or if the gray scale image in the 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 applmageViewer1.py program to also print this information for the self.npImage object.**



```
if len(self.npImage.shape) == 3 and self.npImage.shape[2] == 3:
        print(f"Color space: BGR")
else:
        print(f"Color space: Grayscale")
```

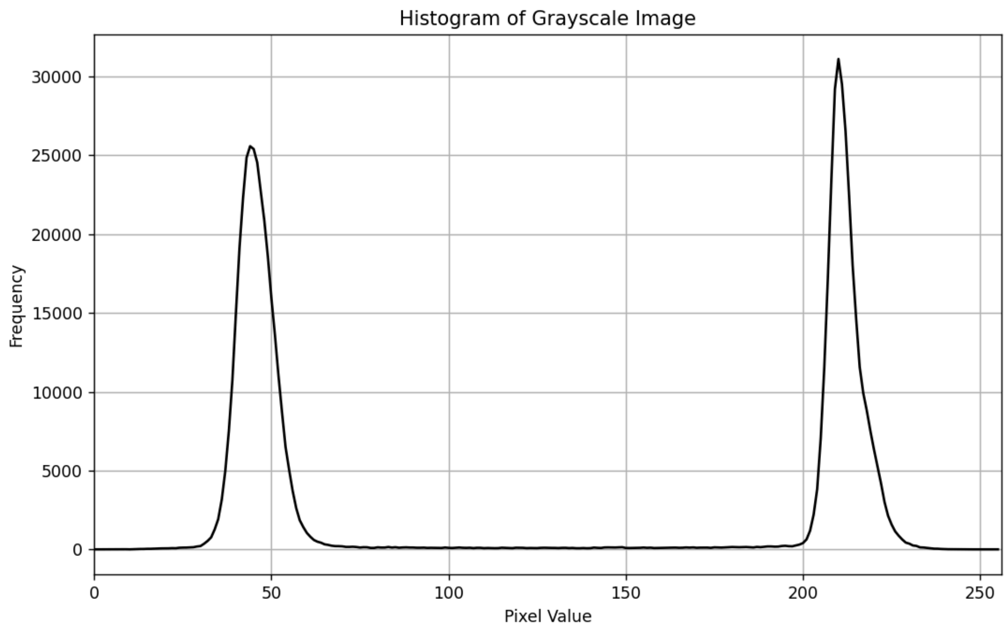
D. Convert the image into a gray scale image using OpenCV function. Documentation on Matplotlib: colormaps % and OpenCV: Color Space Conversions % may be helpful.



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

In section d).

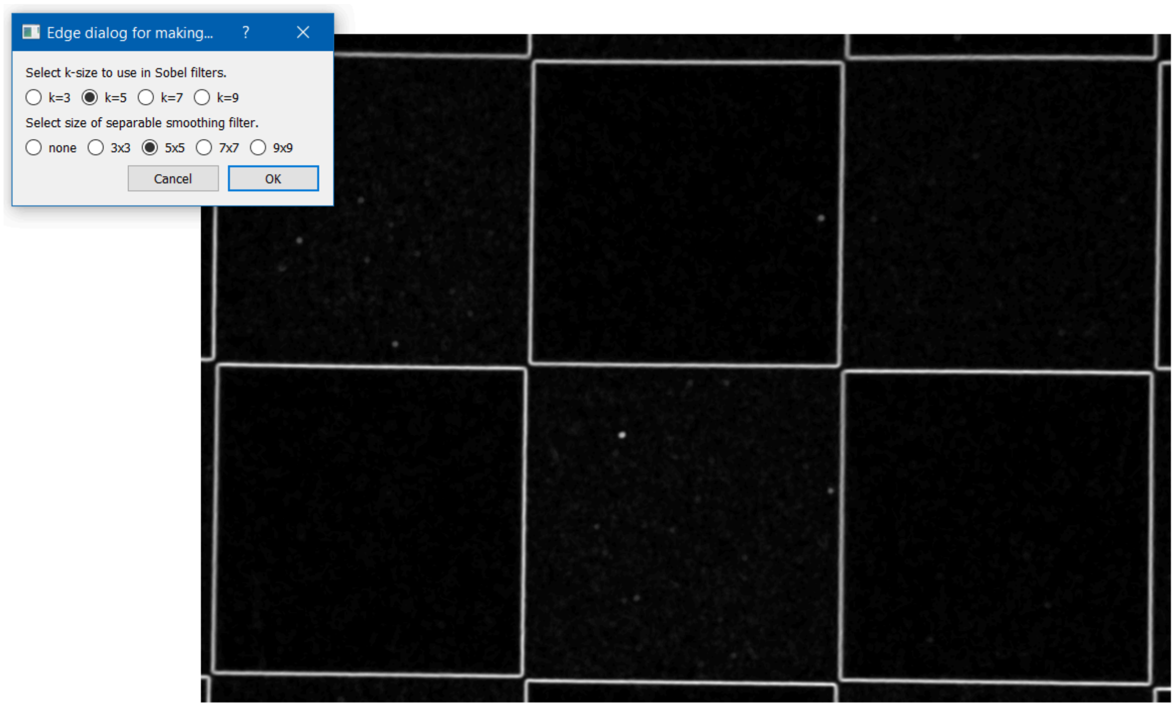
F. Make and display a histogram of the pixel values in the grayscale image. Include the resulting histogram plot in the report. You may see histogram tutorial %.



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

Edges in an image represent the boundaries between different regions, often where there is a significant change in intensity or color while lines are continuous geometric structures that can span across an image.

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


Edge dialog for making...

Select k-size to use in Sobel filters.

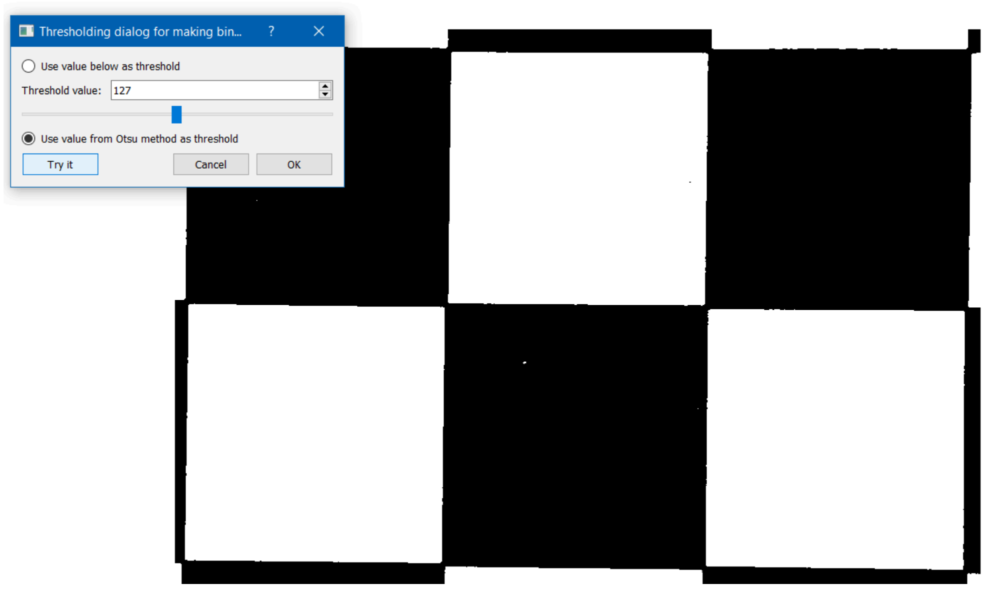
k=3 [✓ k=5] k=7 k=9

Select size of separable smoothing filter.

none 3x3 [✓ 5x5] 7x7 9x9

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

We used the Otsu method to choose the threshold.


Thresholding dialog for making bin...

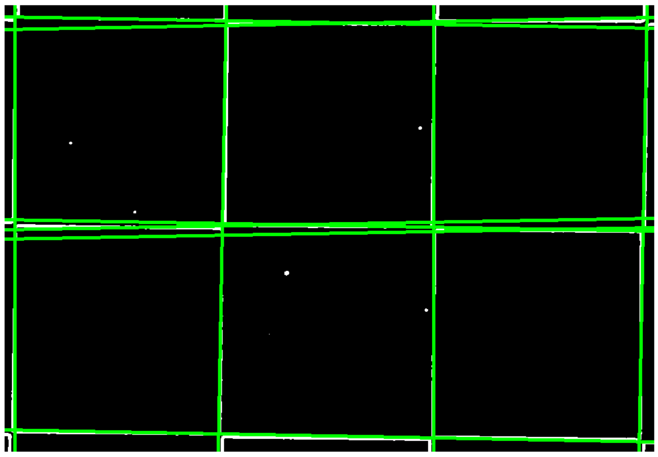
Use value below as threshold

Threshold value: 127

[✓ Use value from Otsu method as threshold]

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.

First, we converted the original image to grayscale to simplify the analysis. Next, we applied an edge detection algorithm to create an image with a black background and thin white lines, highlighting the important contours. Afterward, we binarized this image to emphasize the white lines against the black background, making it easier to identify the lines. Finally, we detected the lines in the binarized image and highlighted them in green for visualization and analysis.



Working hours

Date	Time	Student working
August 22	LAB: 10:15 - 14:00	Daniel & Anton
August 23	LAB: 10:05 - 13:00	Daniel & Anton