Labwork 3:

Author: Duong Tran Khanh (BI9-078)

Tools being used

1. Python 3
2. Pycharm IDE
3. Opencv library

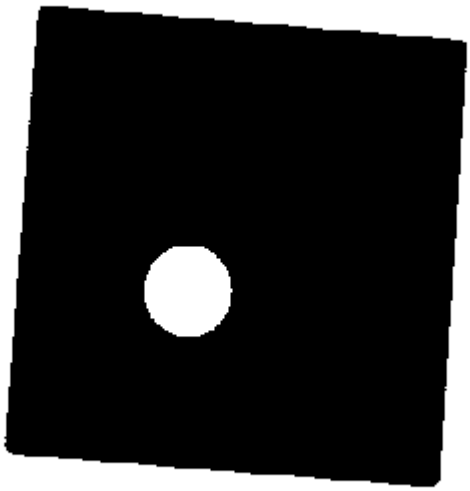
Binarization

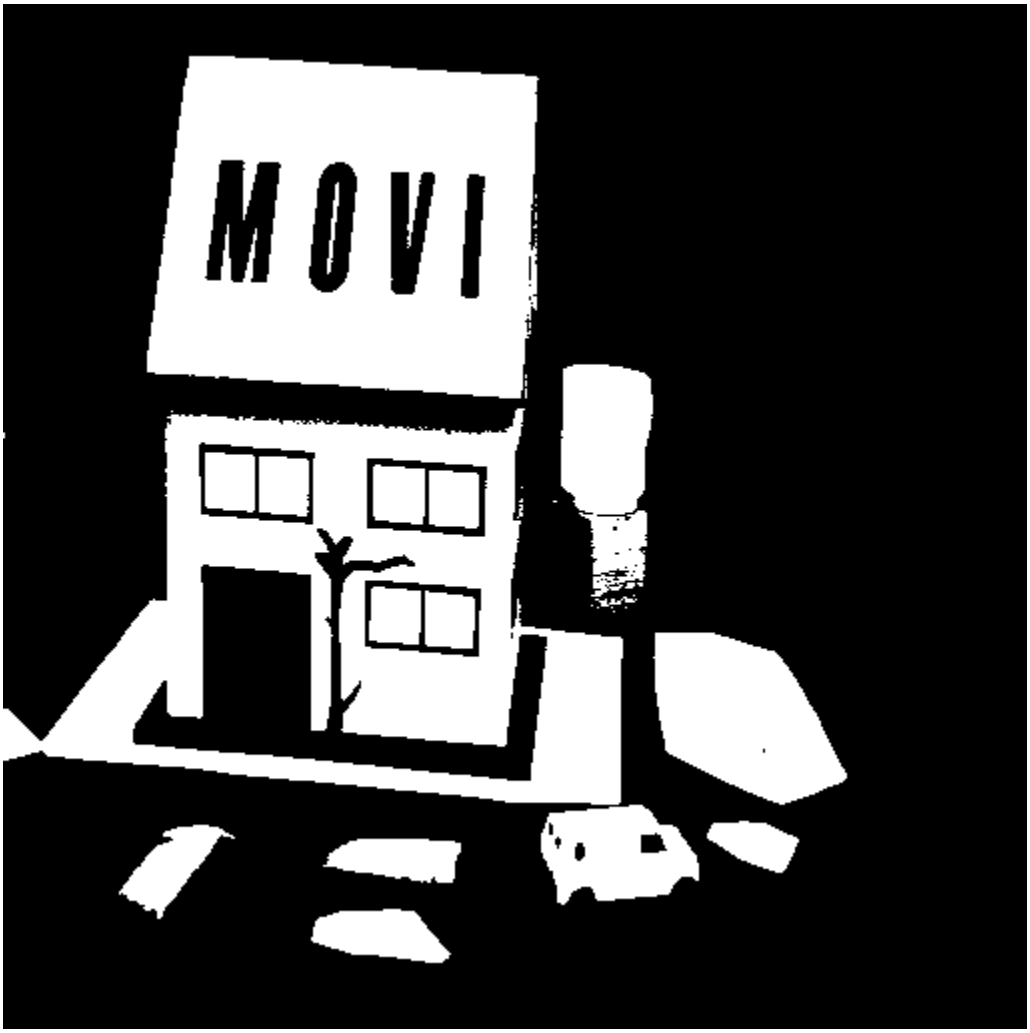
Binarization consists in transforming a grayscale image into a binary image.

Arbitrary threshold

For this method, we apply a threshold operation into the images by looking at the image pixels. First we load the image by using build-in function in Opencv library `cv.imread()` and transform it into gray scale. Then, with function `cv.threshold()`, choose option *cv.THRESH_BINARY*. This means if the chosen pixel value is greater than the given max value *(which in this case we choose 255 as maximum value)*, program will set the destination image with the max value, otherwise, it will be 0.

The outcome of this exercise will look like this:



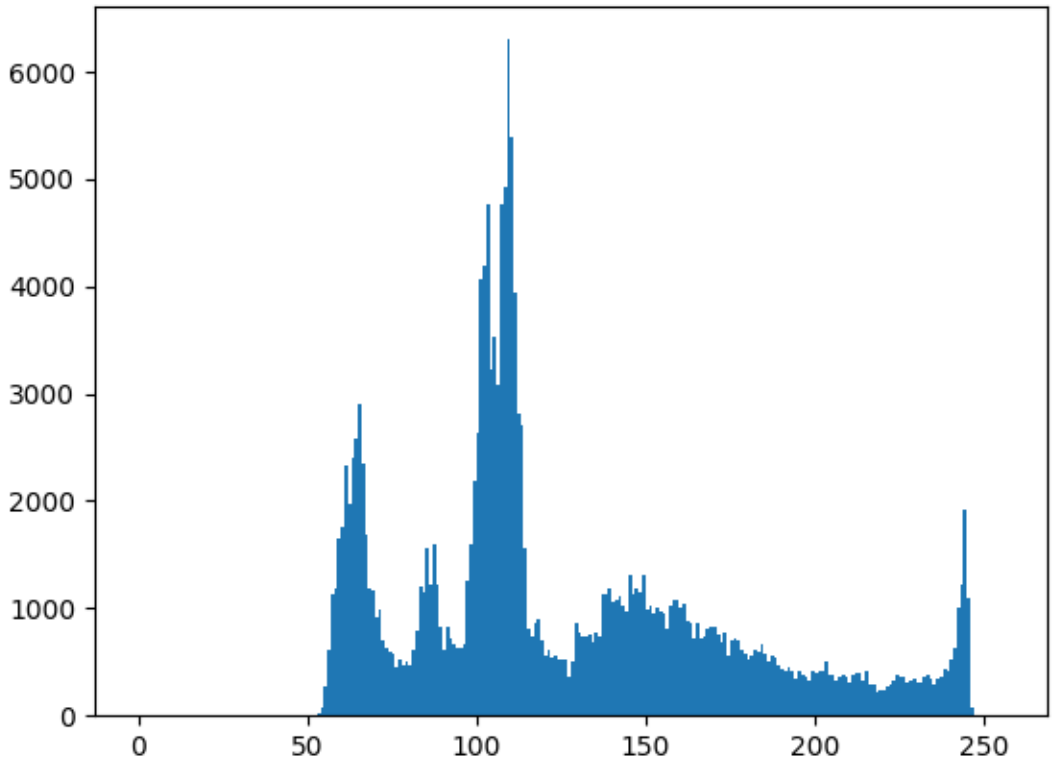


We will use the function `cv.imshow()` to show all three images: *forme1.png*, *house8.png* and *femme.png*.

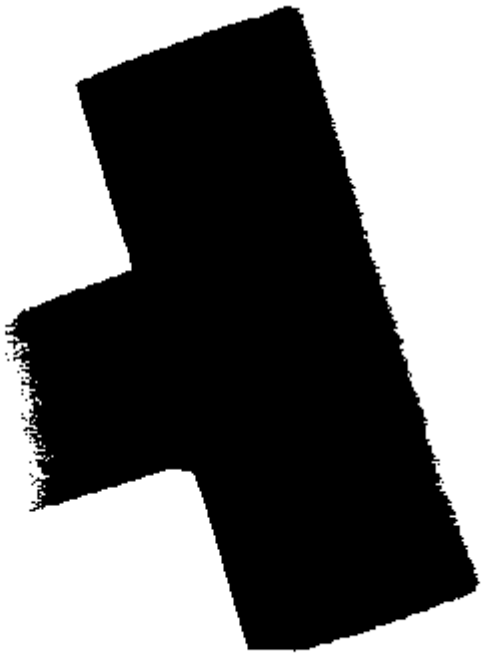
Noted: After various time of testing pixel values of three images, I determine that to get a near perfect greyscale image of the original ones, we can use the value of 120. Other than that value, the recreated images are not very good in quality.

Threshold with Histogram

Histogram represents the pixel intensities with a graph. In this particular exercise, to plot a histogram, we will need to import a library in Python called **matplotlib** in order to execute. After loading the images and transform into grayscale, we shall plot the image with function `plt.hist()` and show it by `plt.show()` The histogram will look as follow:



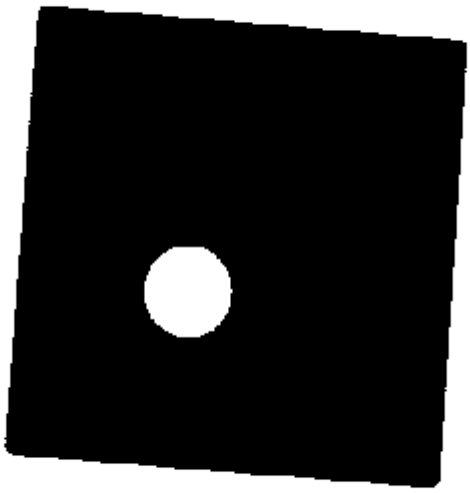
With the graph, I decided to use a couple of number between 80 to 120. The result shows that with each number, the outcomed images are vary and the best possible one is with the value of 90.

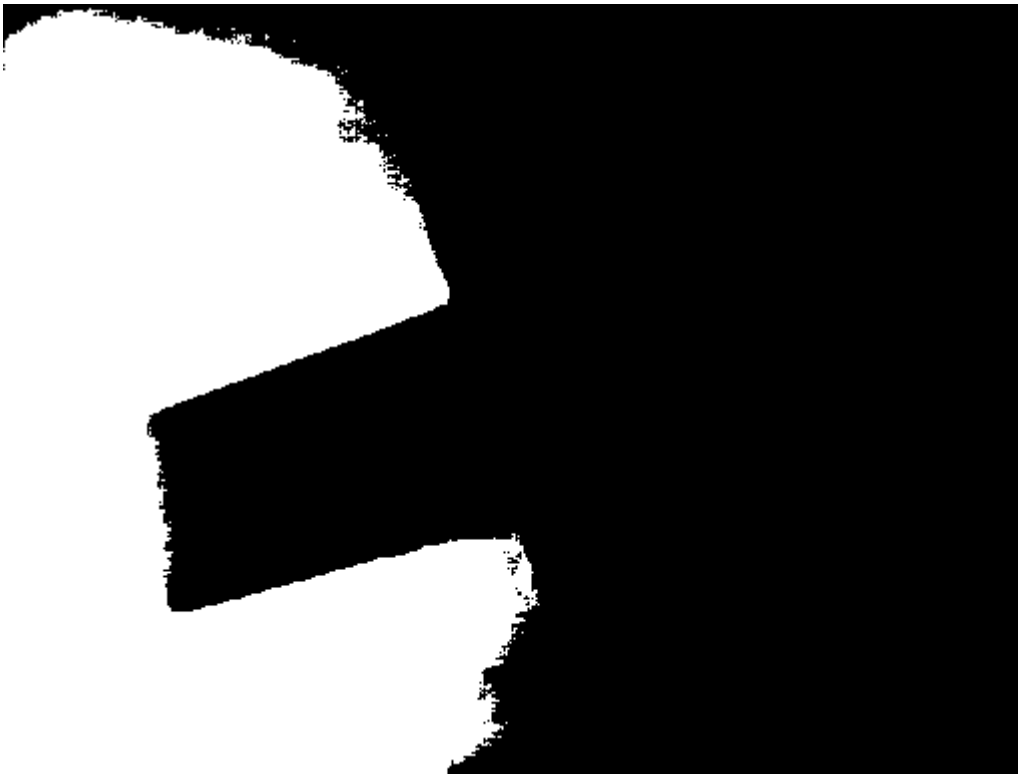


With this method, the difficult is that we cannot determine an exact best value to extract the image from its original form as we estimate by eyes. Despite that the quality of the image is improved, it is not most optimized method to extract from the background.

Threshold with Otsu method

Once again, load the image, transform to grayscale and apply a threshold with function `cv.threshold`. However, at this point, we add an extra option *cv.THRESH_OTSU*. This tells the program to process the image with Otsu method. The result will be:





As we can clearly see, *form1.png* and *lena.png* both give two new thresh image with well quality compared to arbitrary threshold. However, *forme3.png* look worse in this method compared to other method. To explain, it seems because of the Otsu method does not work well when the histogram show many fluctation in pixel indensities.

Aplication exercises

Text extraction

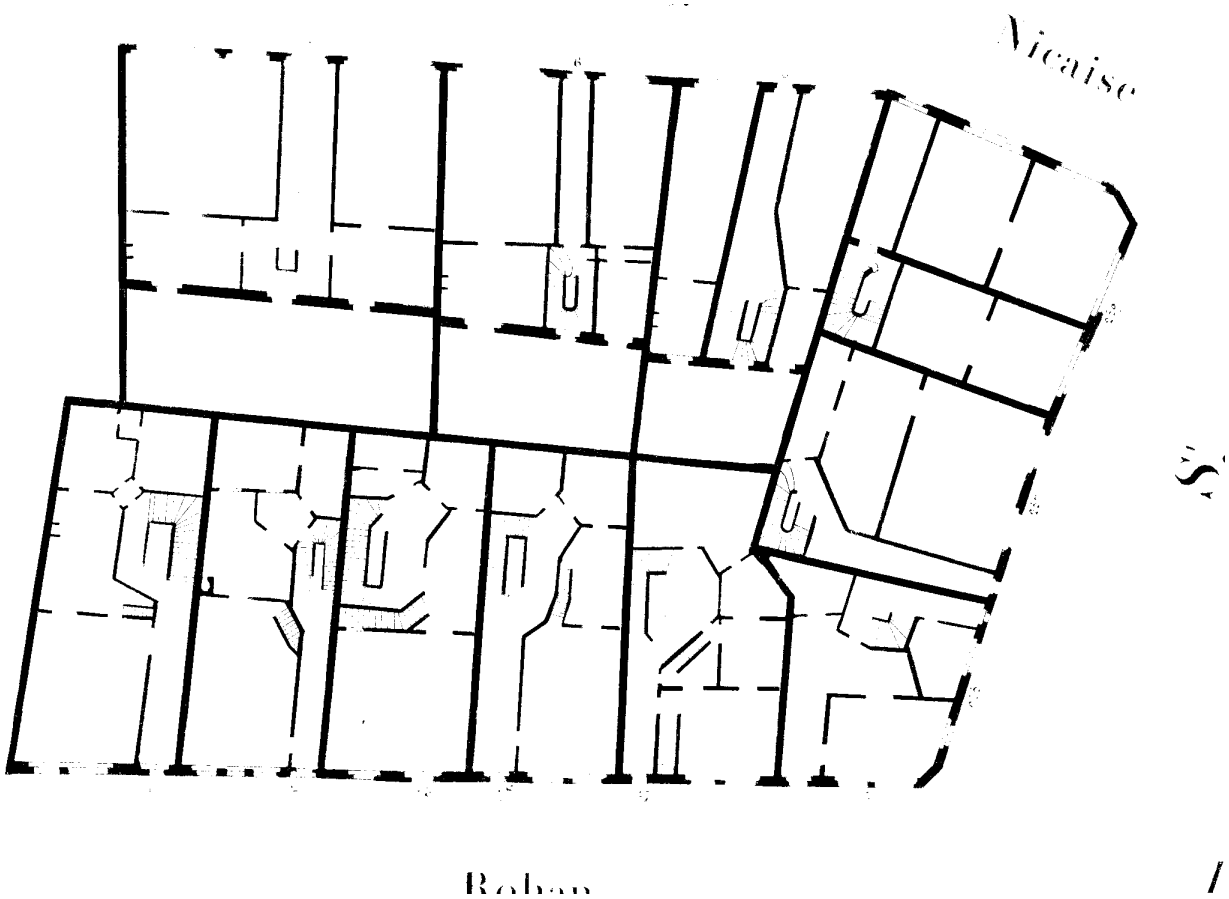
As mention above, *Arbitrary threshold* has a drawback of determining the best threshold value and *Otsu method* will create a better gray scale image if the original image has a good histogram. With the case of *cadastre1.png*, after plotting the histogram, we can see that it has few fluctuation in pixel indensities, hence, using *Otsu method* should be simple and can obtain the best result possible. To make this happen, load the image then transform to grayscale, use the Otsu technique in Opencv python with code `cv.THRESH_BINARY+cv.THRESH_OTSU`

The result should look like this:

Extraction of objects

In this exercise, we want to obtain only the thickest wall, therefore, it is best to use *Arbitrary method* as this should give us the option to modify the value of thresh. Utilize the code `cv.THRESH_BINARY` as flag in function `cv.threshold()`

The result will look like this:



Code

All the code can be found on github, link to the repository: https://github.com/Diseemm/Digital-Forensics