Ecole Nationale
Supérieure
de l'Electronique
et de ses Applications

GHILENE RAYANE
LASSO JARAMILLO NICOLAS

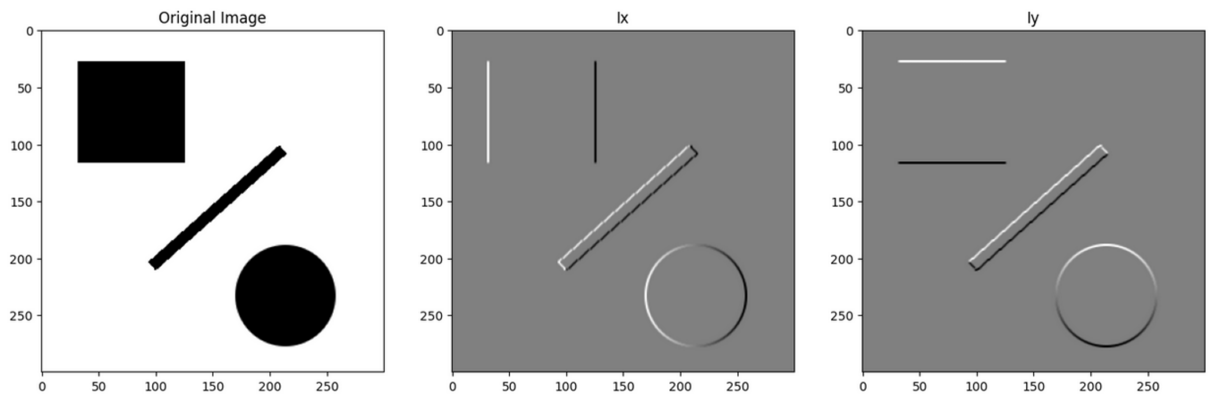
Image Processing

Lab1 Basic Image Processing

The goal of this lab: *is to experiment with different image processing techniques such as filtering and edge detection. We will start by implementing and testing a Harris Corner Detection algorithm, and then we will try a Canny Corner detection method.*

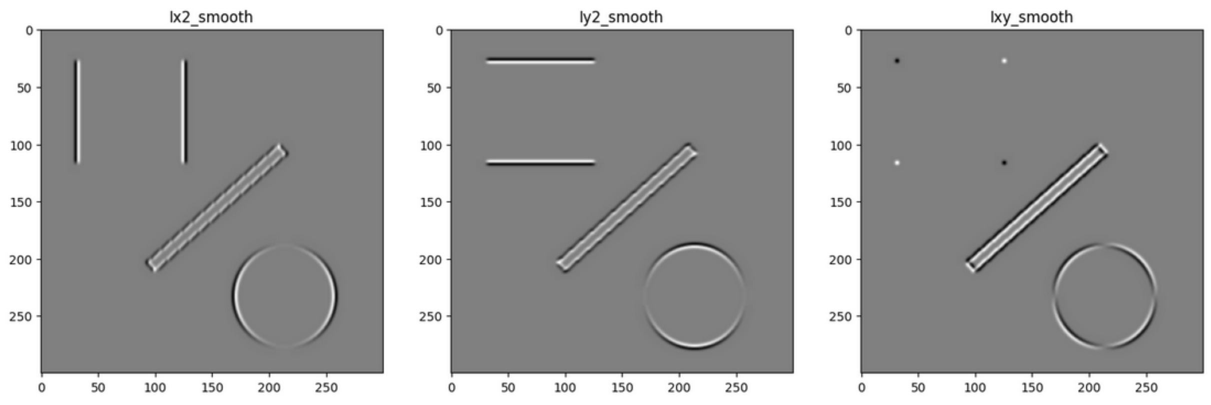
1. Harris Corner detection

We started by calculating Ix and Iy by applying Vertical and horizontal Sobel filters to our image:



plot 1: Ix and Iy

Then we calculated the second-degree derivatives of these two matrices, Ix2, Iy2, and Ixy:



plot 2: Ix2, Iy2, and Ixy

We then calculated the autocorrelation matrix Aij:

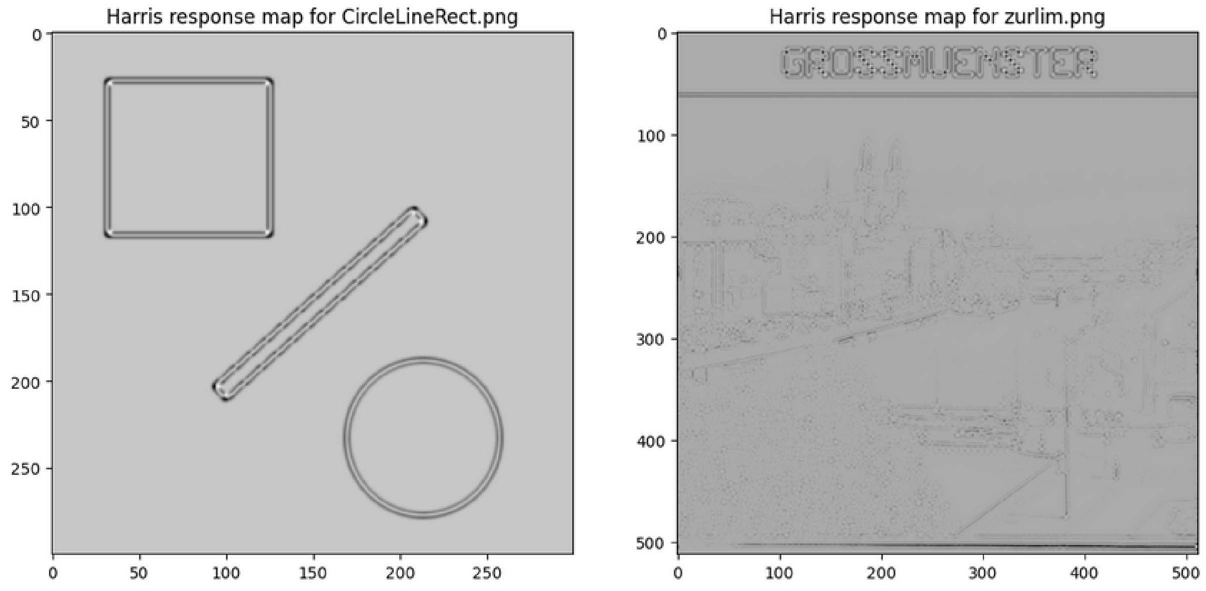
$$A_{i,j} = \begin{bmatrix} \bar{I}_x^2(i,j) & \bar{I}_x.\bar{I}_y(i,j) \\ \bar{I}_x.\bar{I}_y(i,j) & \bar{I}_y^2(i,j) \end{bmatrix}$$

And proceeded to calculate the Harris response for each Pixel (i,j):

$$H_{i,j} = det(A_{i,j}) - k(trace(A_{i,j}))^2$$

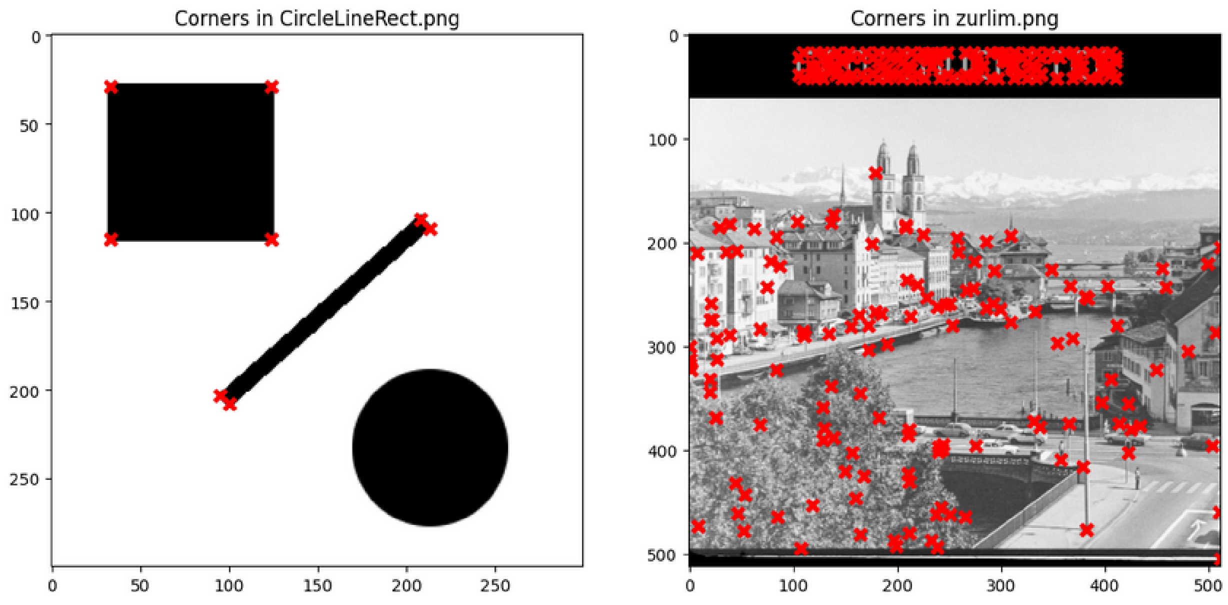
where k is a positive constant.

Here’s an application of the Harris response on two different images:



plot 3: Harris response test

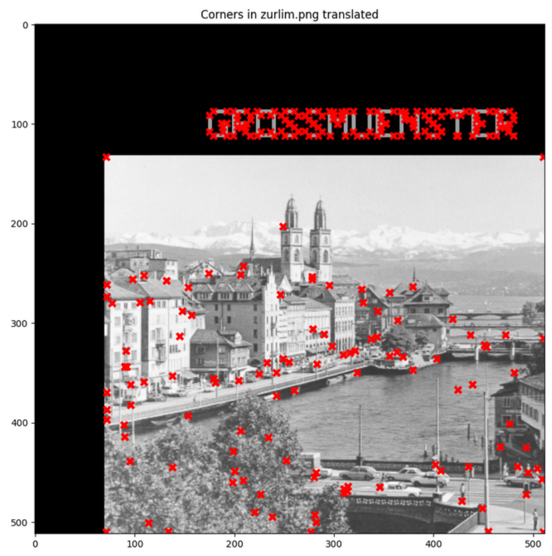
Now we will apply the non-maximum suppression to the Harris response that we just calculated to show the detected corners on top of the original images.



plot 4: Detected corners on top of the original image

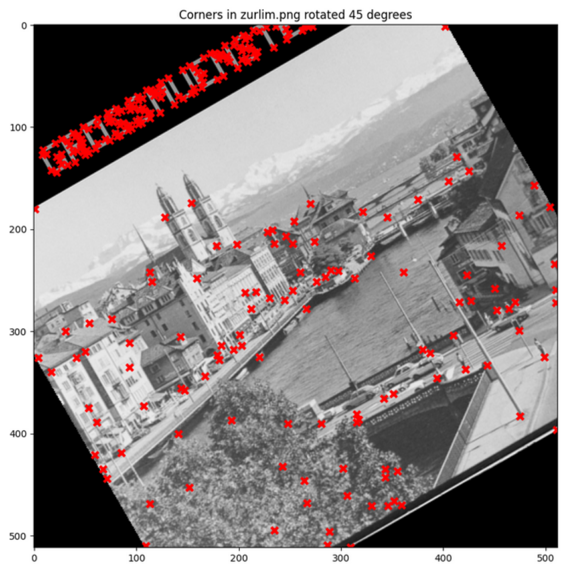
Experiments with image transformations

Now we want to check the robustness of the Harris Corner Detection Algorithm with three experiments. In the first experiment we translated the image, and by the results observed, and the calculations (which we cannot elaborate on this report due to time constraints) we can say that the algorithm is robust for translation transformations.



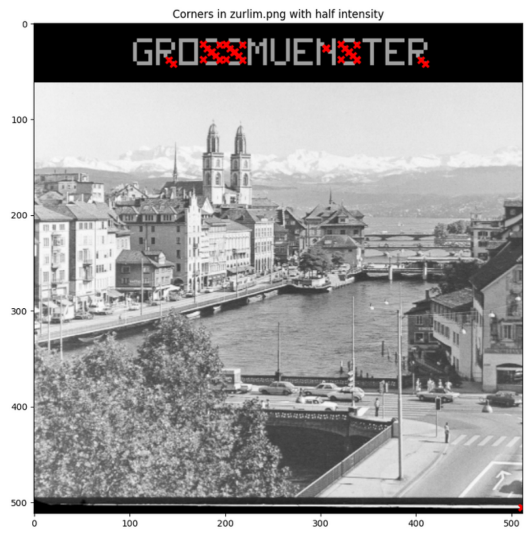
plot 5: Detected corners on a translated image

For rotation we observe different results, if we eyeball the original image corners and the corners after the rotation we can observe some of these markers change. This is no surprise considering the behavior of the algorithm with this type of transformation.



plot 6: Detected corners on a rotated image

For the final transformation, we change the intensity of the image by a factor of 0.5. Considering that the algorithm works with gradients it's safe to assume that this transformation would yield the biggest amount of difference on the corner markers. It is possible to correct this by tweaking the k and the theta parameters, but in this case we can conclude that the algorithm is not robust.



plot 7: Detected corners on a image with modified intensity

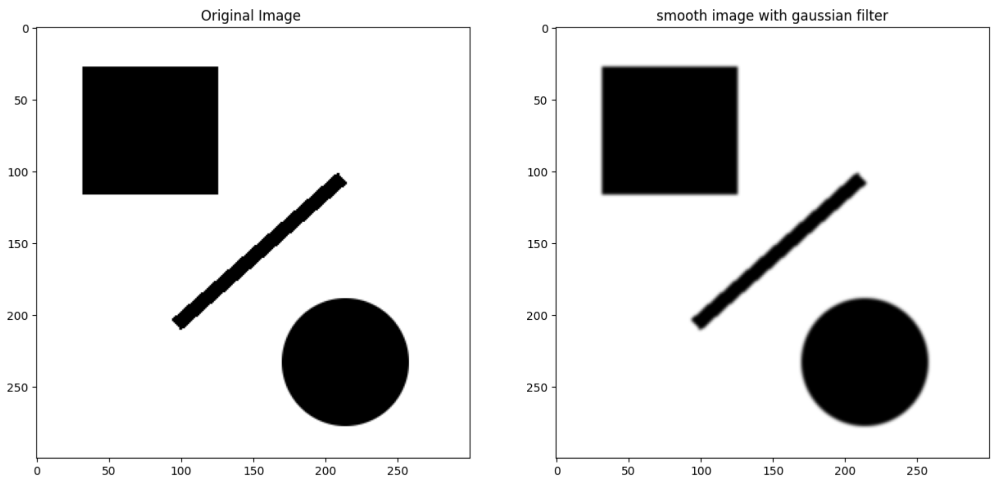
2. Canny Corner detection

The canny Filter is used to detect edges by the identification of local minima/maxima in the approximation of the first-order derivative of an Image

We started by applying a 3*3 Gaussian filter to smooth the image
kernel = 3

$$g(x) = \frac{1}{2\pi\sigma^2} \cdot exp(-\frac{(x-(\{\text{kernel\_size}\}-1)/2)^2+(y-(\{\text{kernel\_size}\}-1)/2)^2}{2\sigma^2})$$

here’s a comparison between the original and the filtered image :



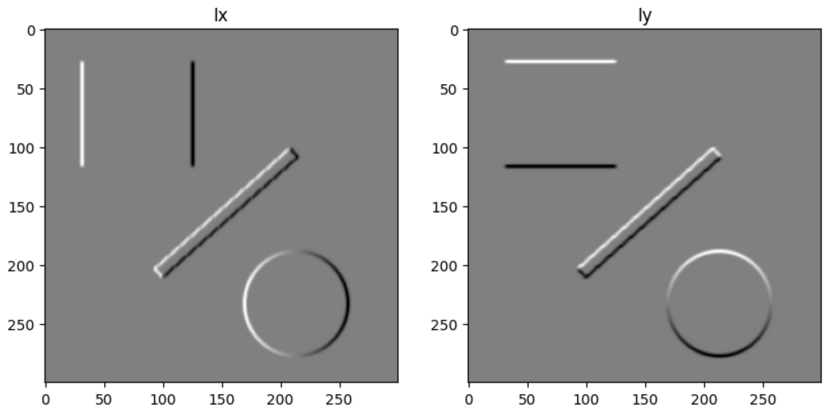
plot 8: Original image Vs Gaussian filtered Image

Then we approximated the Ix and the Iy at each point in the image using a Sobel filter

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -2 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Vertical Sobel *Horizontal Sobel*

Here's the plot of Ix and Iy

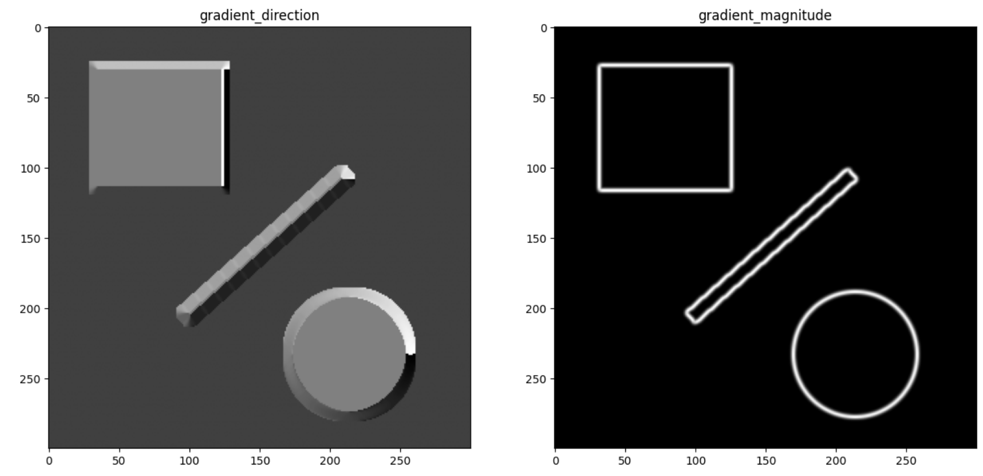


plot 9: Ix & Iy

Then we calculated the Gradient magnitude and direction using Ix and Iy, here are the formulas we used:

$$Magnitude = \sqrt{I_x^2 + I_y^2}\ ,\ Direction = arctan(I_x/I_y)$$

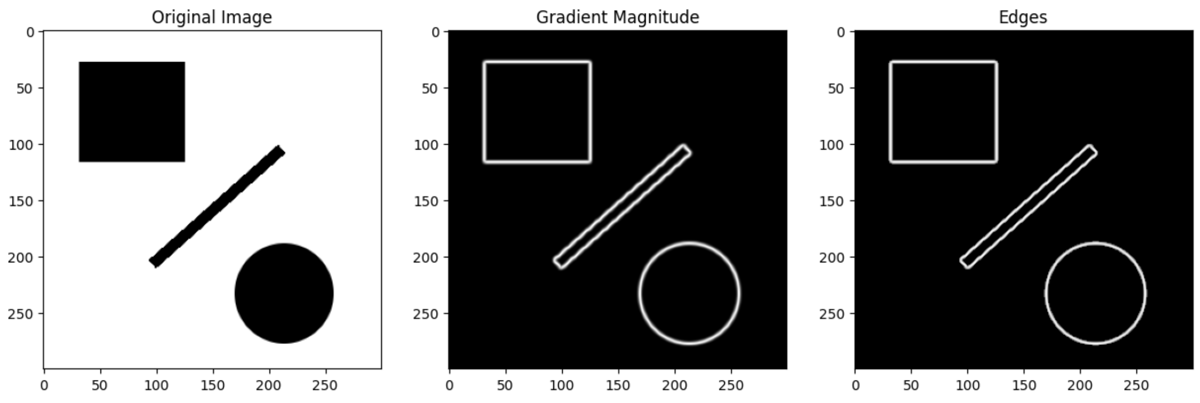
Here are the plots of the Magnitude and direction:



plot 10: Gradient direction and magnitude

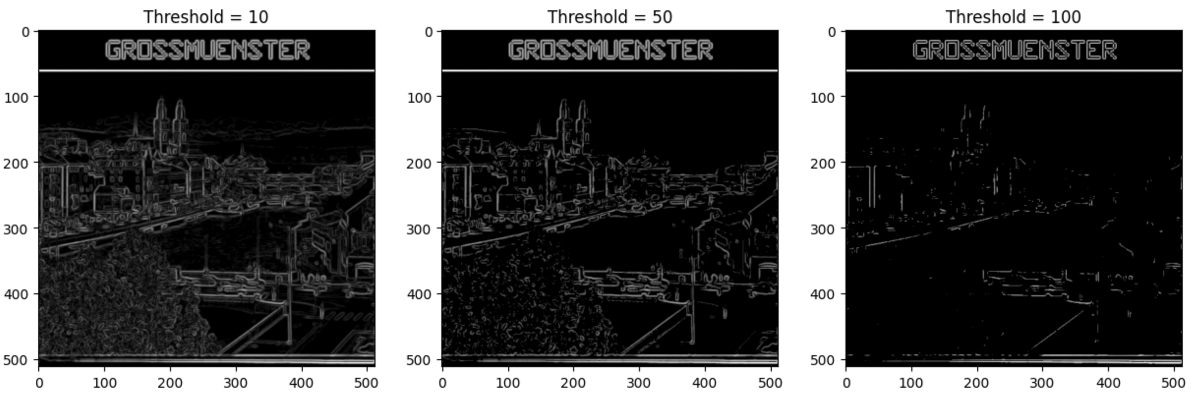
Edge detection using a threshold:

here we used a threshold to detect the edges, if the value of the gradient magnitude is smaller than the threshold it will be set to zero, which will in return emphasize the edges present in the image:



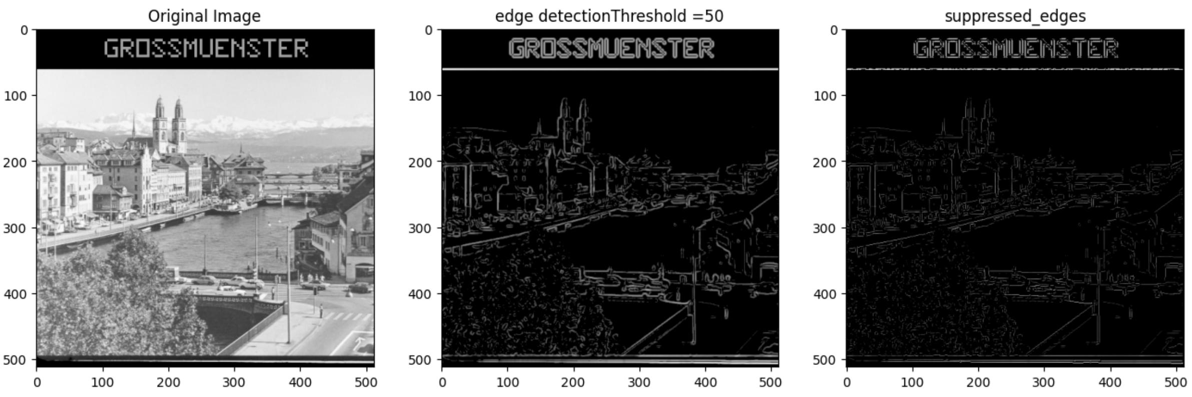
plot 11: Edge detection

Using this image the edges are quite obvious and their detection isn't challenging, here's another example to further explain the functioning of the function for different thresholds



plot 12: Edge detection for threshold = 10, 50 and 100

Now we will perform Non Maximum suppression to identify candidate edges. In this step, we looked for local maxima by comparing the neighbors on the orthogonal line to the gradient direction for each pixel.
here's the application of the non-maximum suppression on an image:



plot 13: Applying non-maximum suppression

end