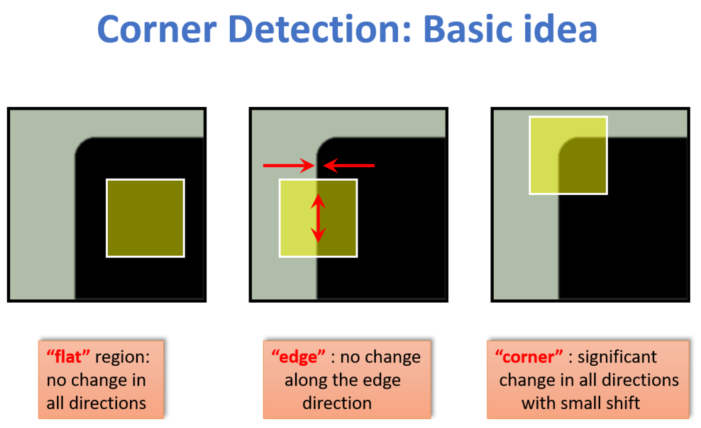
BACHELARD Maéva
GALLIEN Anthony

SIA 2023-2024

TP1 Image Processing
Basic Image Processing

1. Harris Corner Detector

The Harris corner detector is used to locate corners in an image. Intuitively, if one moves a window over an image patch, and a corner is present, then a strong intensity change can be observed regardless of the moving direction.


Corner Detection: Basic idea
"flat" region: no change in all directions
"edge" : no change along the edge direction
"corner" : significant change in all directions with small shift

We have chosen to work on 2 images, one simple and the other more complex. The first one will allow us to verify that our algorithms work effectively since it's easy to identify shapes, corners, and estimate the image gradient (image number 1). With the second image, we will be able to rigorously test our algorithms and also assess the limits of the methods we will use (image number 2).

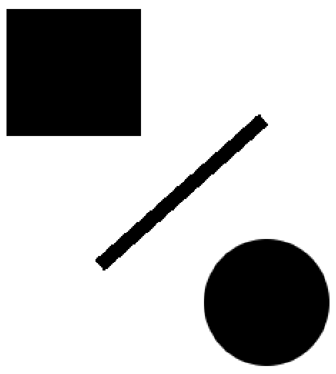


image number 1

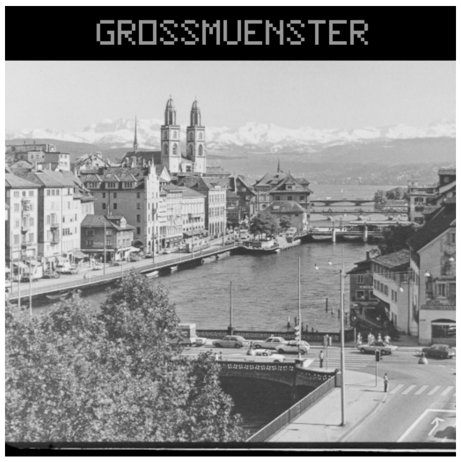
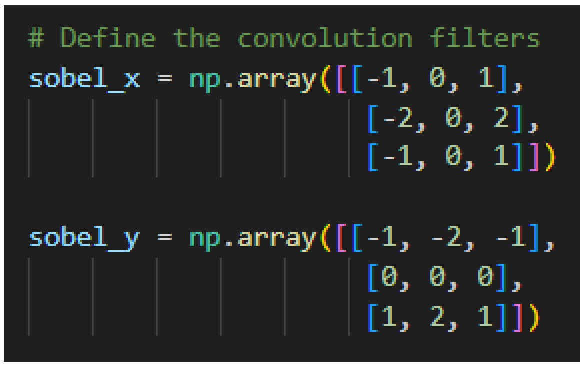


image number 2

BACHELARD Maéva
GALLIEN Anthony

SIA 2023-2024

STEP 1 :



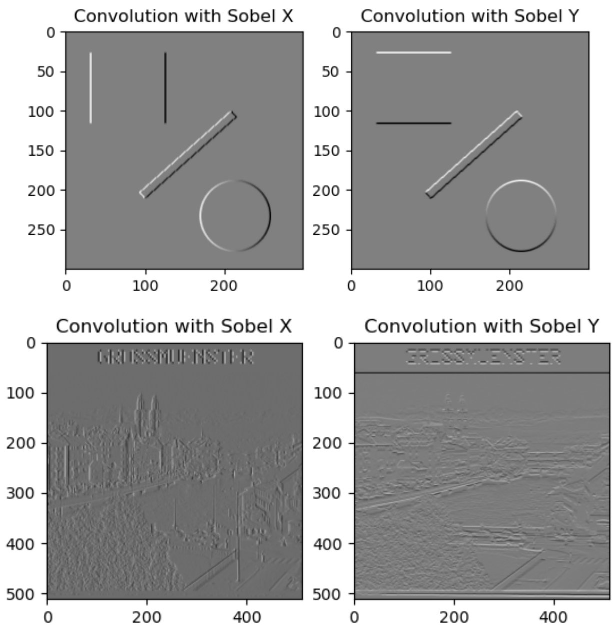
```
# Define the convolution filters
sobel_x = np.array([[-1, 0, 1],
                    [-2, 0, 2],
                    [-1, 0, 1]])

sobel_y = np.array([[-1, -2, -1],
                    [0, 0, 0],
                    [1, 2, 1]])
```

We apply Sobel filters to the images (numpy matrices) through convolution. We obtain the approximations of Ix and Iy through this calculation.

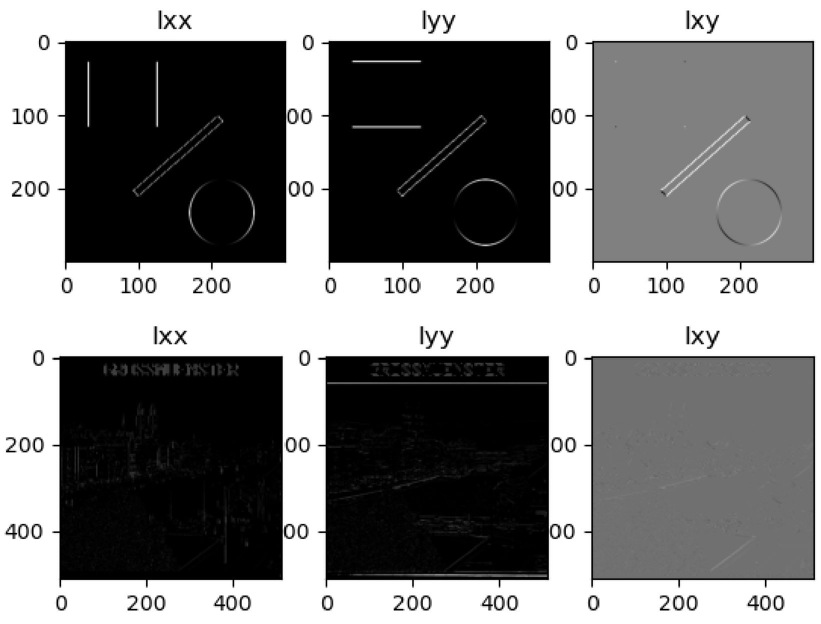
The Sobel filter is an operator used in image processing for edge detection (refer to the course). The operator computes the gradient of the intensity of each pixel in both the x and y directions in our case. This indicates the direction of the biggest change from light to dark, as well as the rate of change in that direction in RGB (even though our images are in black and white here). This allows us to identify **sudden changes** in brightness, likely corresponding to edges, as well as the orientation of these edges. This will be highly useful for locating corners, as we explained earlier.

In the first image, the vertical and horizontal edges are clearly visible. This validates that our Sobel filters are acting as we intended. On the second image, it seems less clear, although we can still see the edges.



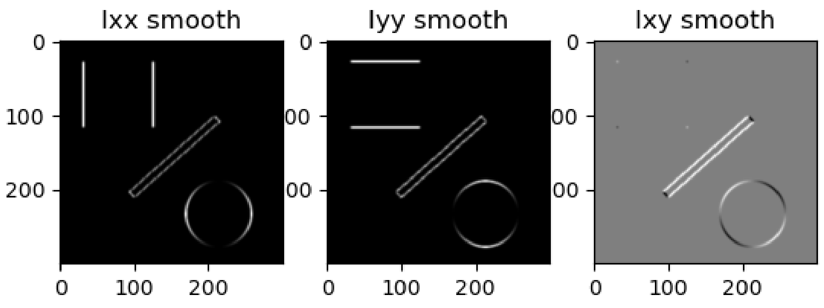
STEP 2 :

We proceed to the next step in order to obtain the intermediate results Ixx, Iyy and Ixy, useful for the construction of matrices A and H. In this step we simply multiply each coefficient of the matrices by the other one (ex : Ixy[1][1] = Ix[1][1] * Iy[1][1]).

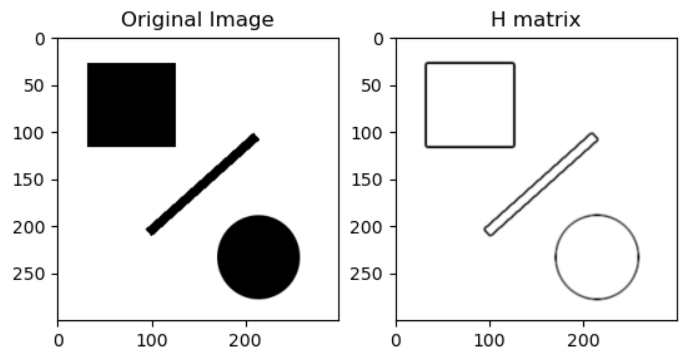


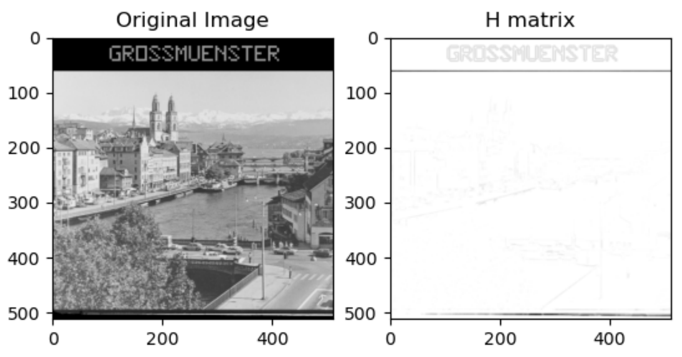
BACHELARD Maéva
GALLIEN Anthony

STEP 3 :
To obtain a smooth version, we apply a Gaussian filter to our matrices. The filter parameter can be chosen as the input to our Harrison Corner function, but if it is not specified then it is automatically set to 1. We chose the value 1 because after multiple tests it seemed the most interesting.

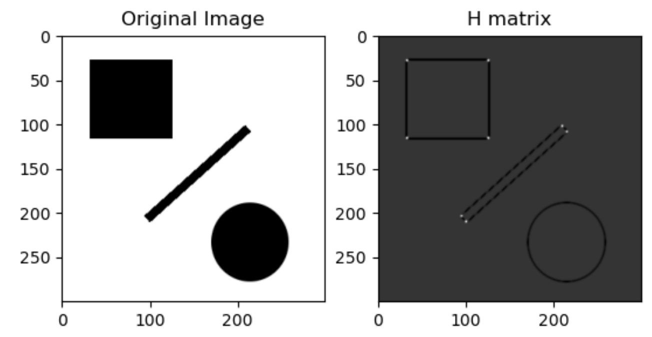


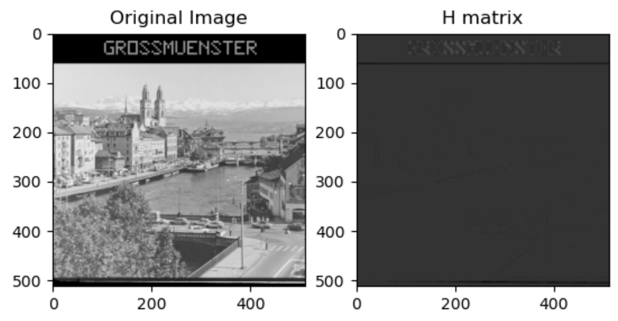
STEP 4 :
In step 4 we construct the matrix H using our last 3 matrices Ixx, Ixy and Iyy arranged in a matrix A. We start by setting the simple value of k=1 to see the result (the parameter). Then we start testing other values of H in order to get something more interesting. This is our results with k=1. Note that the H matrix gives us all the remarkable contours of the image thanks to the gradients.





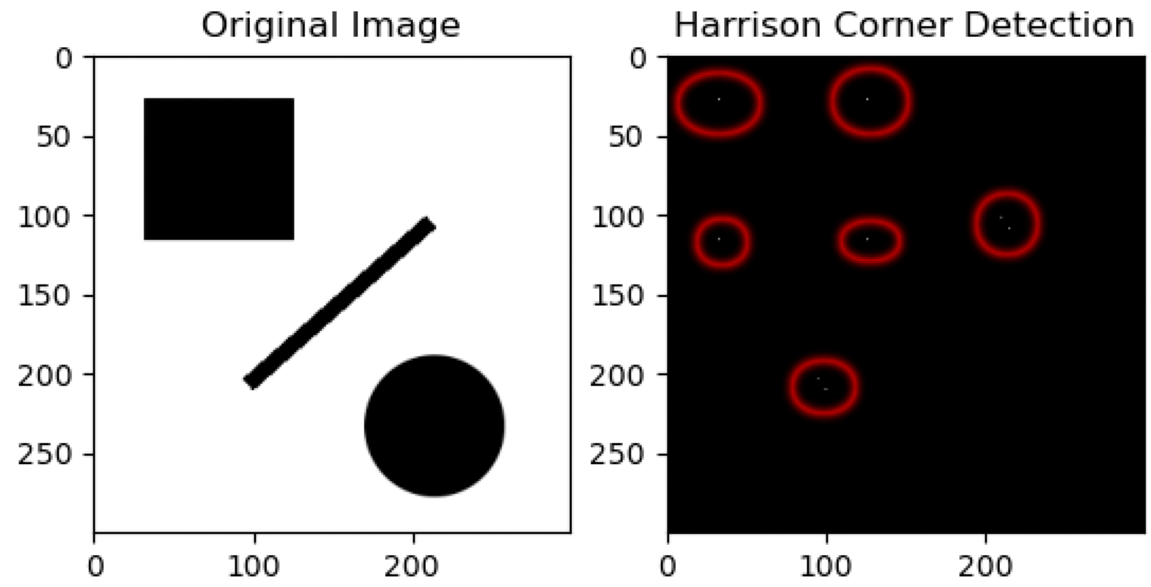
 The value of k=0.05 seems to us the most interesting and here are the results obtained with this value:

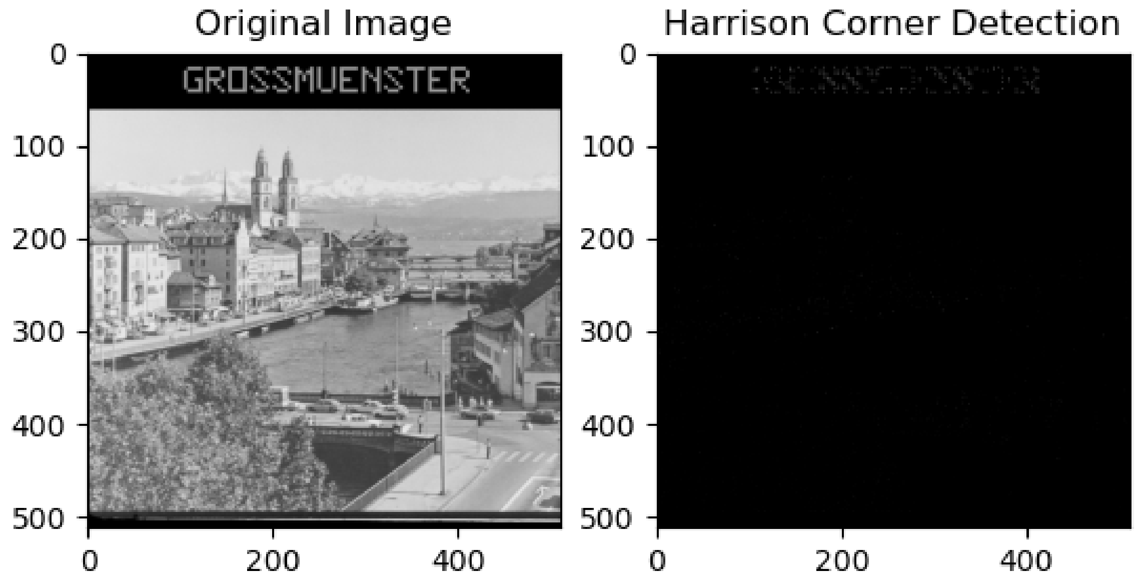




STEP 5:
Thanks to the k value, we can make the corners stand out. All that remains now is to extract them using the non-maximum suppression method. This will allow us to keep only the

'maximum' values of their neighbors and when we look at our current image we can see that the maximum value of a neighborhood is always a corner.





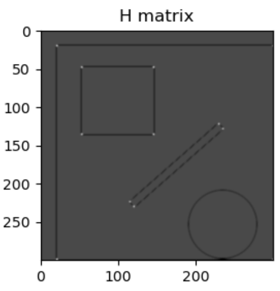
In the first image, we can finally see the corners very clearly (circled in red). On the second image it doesn't seem to stand out, the image having been very difficult to study from the outset due to its complexity.

- Is the Harris corner detector robust with respect to intensity shifts and intensity scalings ?

The Harris corner detector is not robust with intensity shifts because it relies on local intensity gradients. Hence, if there is an intensity shift, the gradient of the image will shift accordingly resulting in a shift of the corner locations. This means that the corner detector is not invariant to intensity shifts.

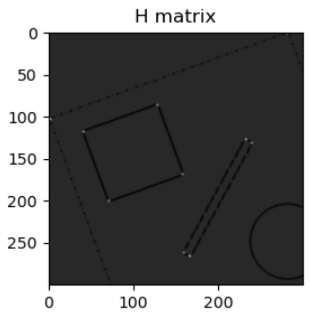
Furthermore, the detector is not robust to intensity scaling either. Indeed, intensity scaling means multiplying the pixel value of an image by a constant. Because of that, the magnitude of the local gradients will be different and can change the corner local values and thus location.

- Is the Harris corner detector robust with respect to translation ?



The Harris corner detector is robust to translation as we can see on the plot on the left. It works because the local intensity patterns and the gradients around the corners stay the same. Hence, a translation of the image results in the same translation of the corners.

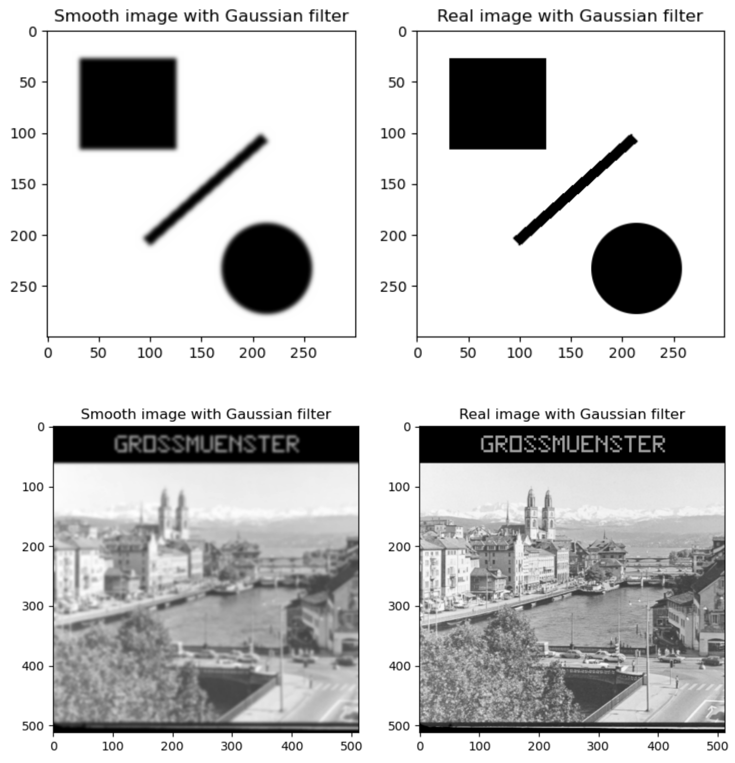
- Is the Harris corner detector robust with respect to rotation ?



The Harris corner detector is robust to rotation as we can see on the image on the left because the values of gradients stay the same. Hence, a rotation of the image results in a rotation of the corners.

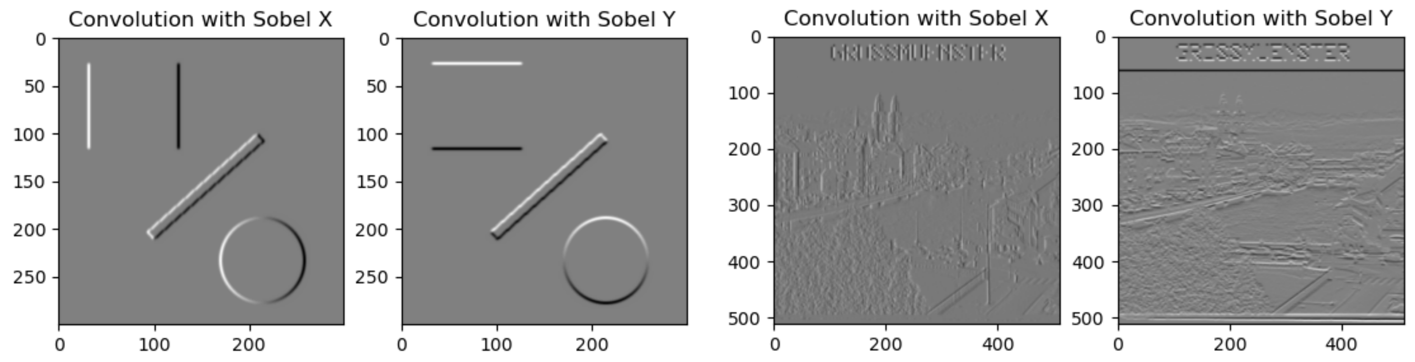
2. Canny Edge Detection

STEP 1 :


Smooth image with Gaussian filter
Real image with Gaussian filter
Smooth image with Gaussian filter
Real image with Gaussian filter
GROSSMUENSTER

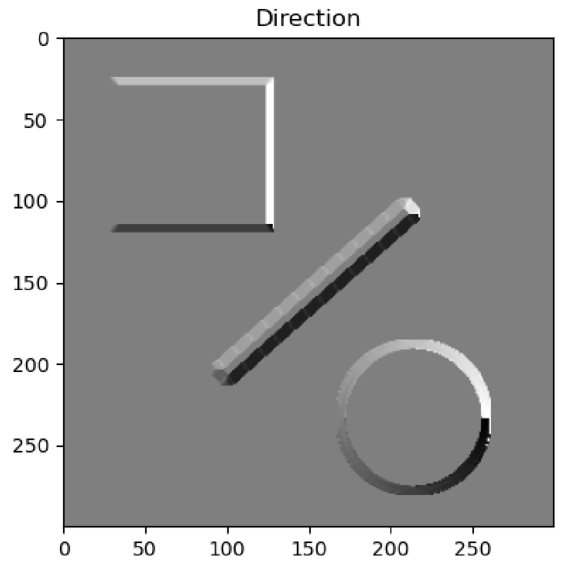
The first step is to smooth the image using a gaussian filter. To do so, we use the gaussian_filter function.
The idea behind this step is to reduce the noise of the image. Indeed, because the final algorithm implies gradient computation, it is highly sensitive to image noise. One way of getting rid of this noise is by blurring the image and applying the gaussian filter.

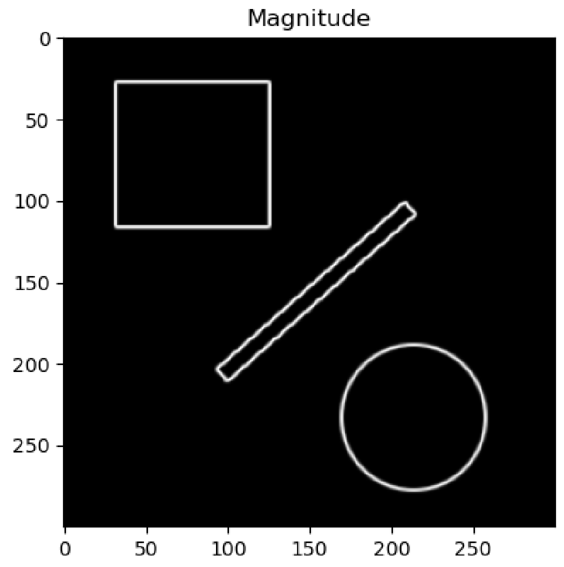
STEP 2 :
Then, we need to apply Sobel filters along the x and y axis to get the derivative of our smooth image. That way we computed the gradient of the image. As in the first exercise, his indicates the direction of the biggest change from light to dark, it will be useful to detect edges.

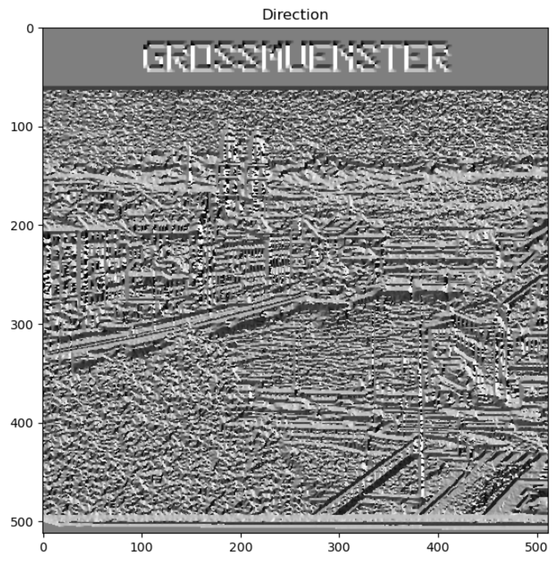


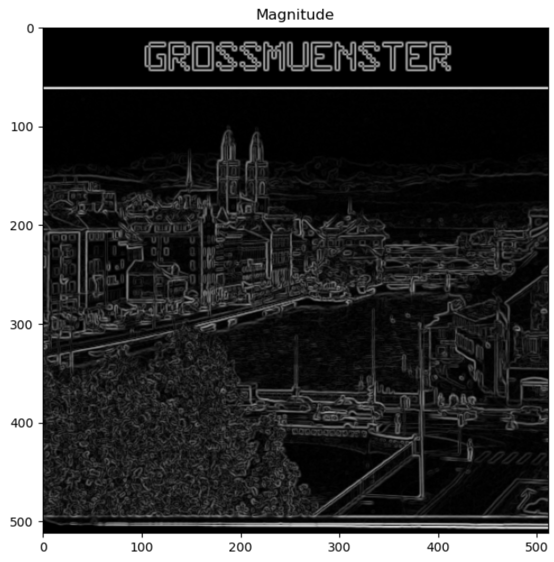
STEP 3 :

In this step, we compute the magnitude of the gradient for each pixel and determine its orientation. By doing so, we now have access to the intensity of the change in brightness and the direction of the most significant change for each pixel. To compute the direction, we use the arctan2 function because it handles divisions by 0.

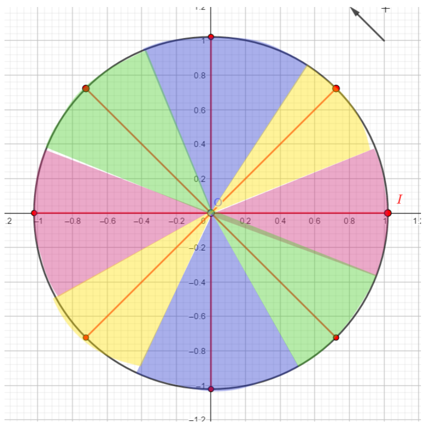




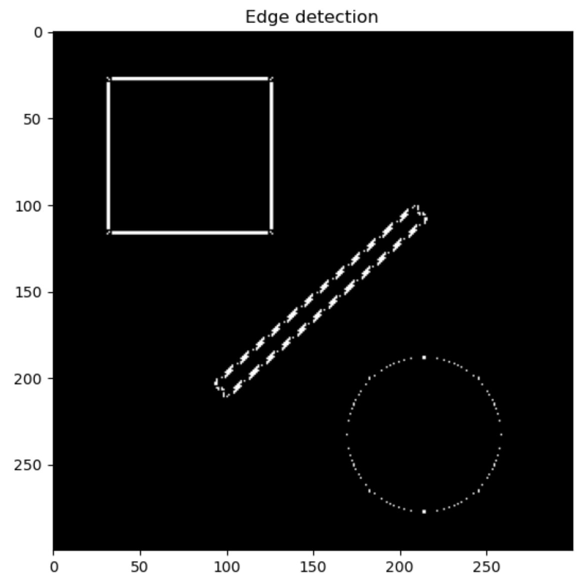


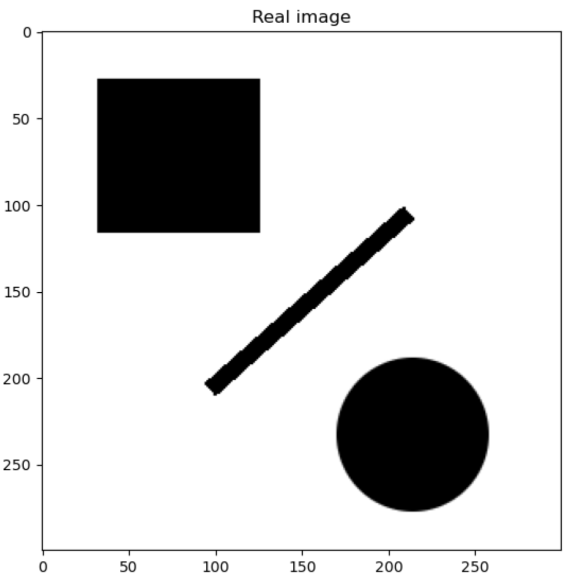


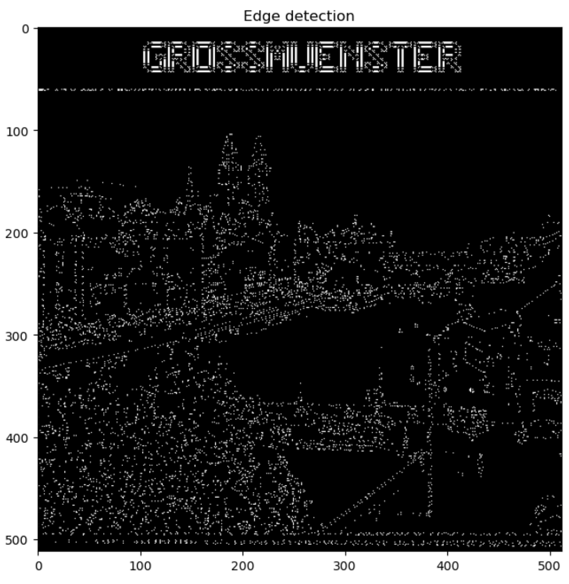
Direction is the angle between -Pi and Pi of the edge for each pixel. So we divided the unit circle into 4 parts (8 directions grouped by 2 because they follow the same line but with different directions). We take the two neighboring pixels that are orthogonal to our direction and compare the magnitude value of our pixel and these two neighbors. That's how we can tell whether we are on an image edge or not.

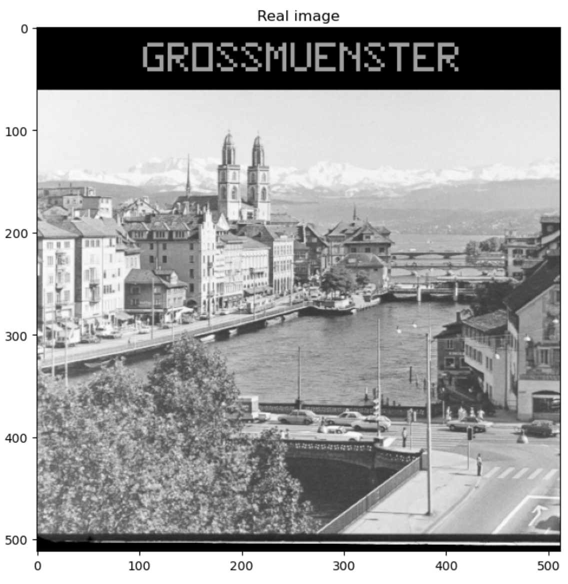


STEP 4 :
In the final step, we perform a non-maximum suppression to refine the edges. To do so, we compare the magnitude of the gradient for each pixel to its two neighbors (orthogonal to the direction we computed earlier). We set the edges to 255 and the rest to 0 so that we can highlight the edges of our image.



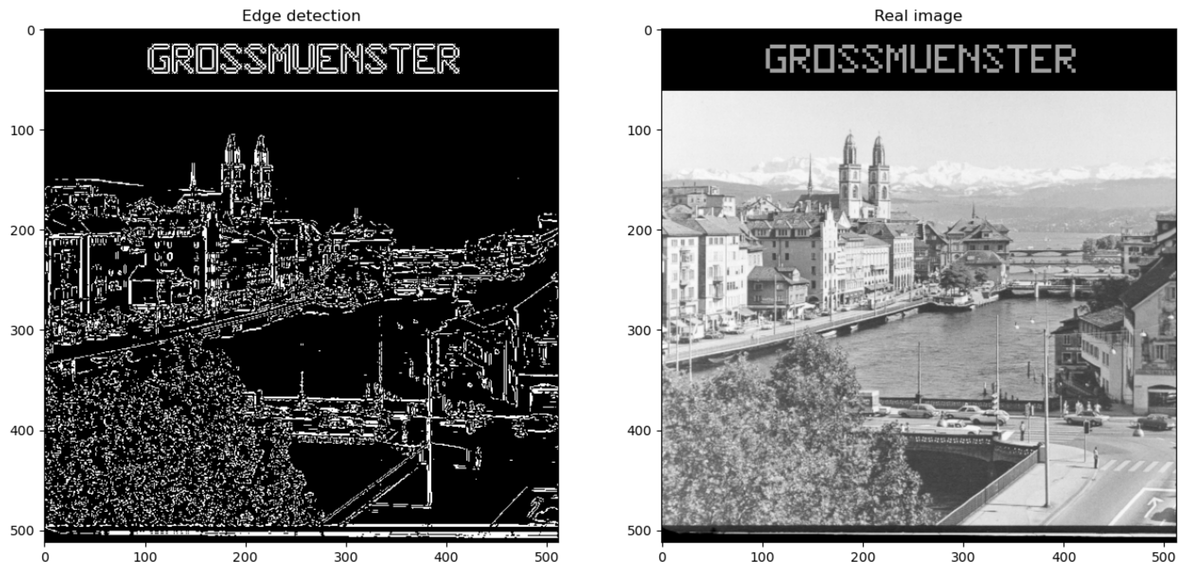


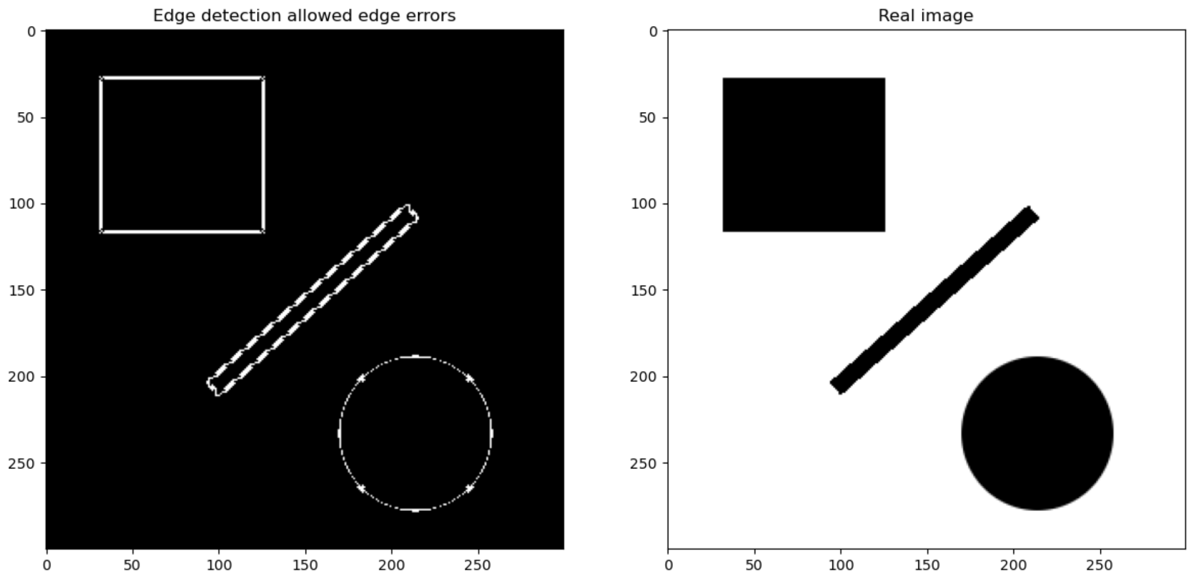


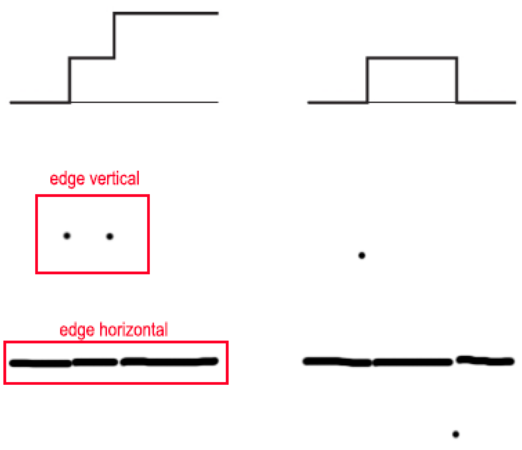


Once all the steps have been completed, we can see the edges of the images, more or less precisely depending on the parameters. The overall result is quite satisfactory, especially if the aim is to recognise an image by its outline or to separate elements. However, some edges seem to need to be 'completed'. This is because there are gaps or spaces in the lines that we want to fill in.

We've decided to modify our function slightly so that it doesn't compare strictly with the neighbors, but leaves a margin of error. In fact, you might think that the error comes from the magnitude when you look at the image precisely on the errors. In this way, we obtain larger and more visible contours. But as we might expect, the image is beginning to fill in. Depending on how the contours are used, it may be worth incorporating the margin of error into the algorithm to improve the result.







Questions :

We draw the derivative according to the y axis. Now we can see the vertical edges. By computing the x derivative, we would have the other edges. That way, we can draw the edges of the image.