School of Computer Science and Communication

DD2423 Image Analysis and Computer Vision:

Lab 1: Edge detection & Hough transform

1. Difference operators

Question 1: **What do you expect the results to look like and why? Compare the size of dxtools with the size of tools. Why are these sizes different?**

All of these operators are Edge Detectors operators that use difference operators to 'find' these edges. These operators can be applied in the x direction or in the y direction, and they compute the difference between the intensity in the pixels in one of these directions; the gradient. Then the output will show this variation or gradient in the image. In consequence, we will see a grey scale image where the edges (high variation of the intensity of the pixels) will have a stronger intensity and therefore will be more visible than the 'continuous' parts of the image. The next step would be to add a threshold so only the edges will be shown in the output, but this has not be done for this question.

In the figure 1 we can see both the outputs after the applications of the different operators and the size of these images.

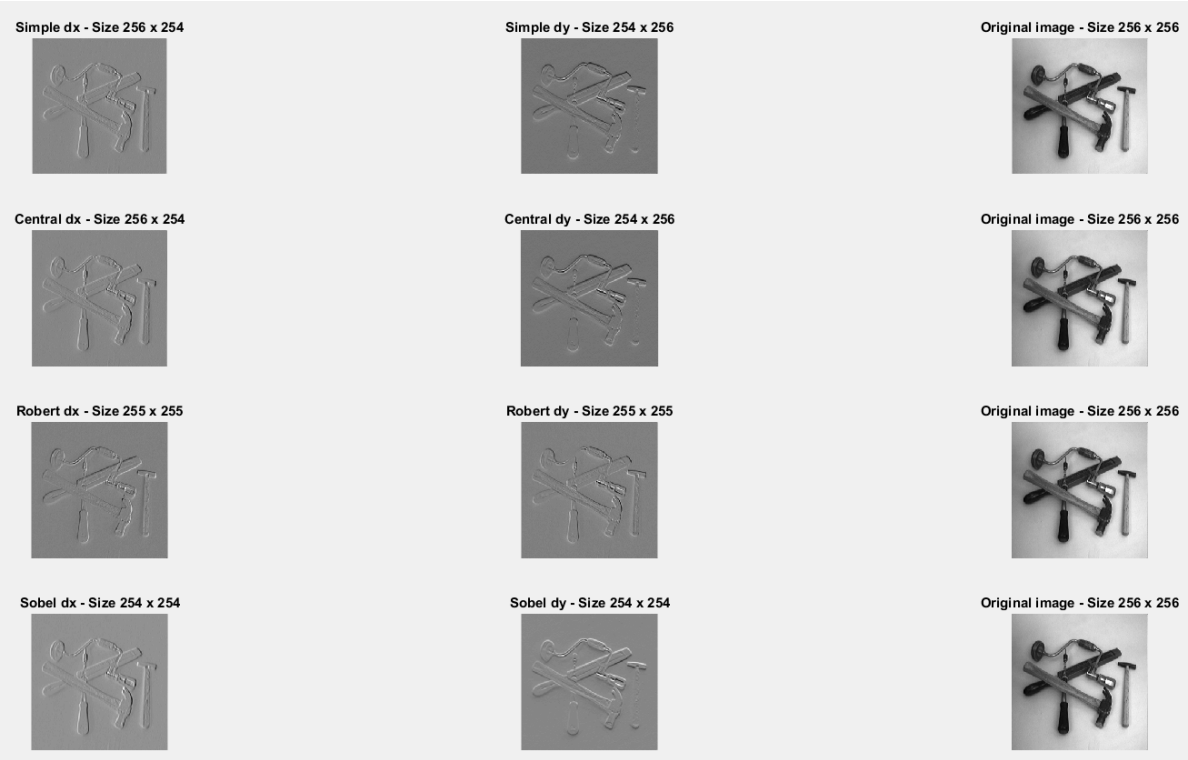

Figure 1: output images after applying the different operators.

We can observe that the output shows something similar as we described before. Also, we can see that the images sizes change. This is because we are using the option 'valid' when using the `conv2()` function. If we take a look to this function description, we can read that this is because with this option the function 'returns only those parts of the convolution that are computed without the zero–padded edges'. The problem is when we apply the convolution to the edge pixels of an image, the convolution

`g(x-y)f(y)` – g being the kernel and f the image/function

cannot be defined as y can be beyond the boundaries of the picture. What zero padding does is to assign a value to these undefined pixels. Instead, if we use the 'valid' options, we just don't use these undefined pixels.
In the figure 2 we can see the consequences of this, as the image now shows a border with a black line of pixels.

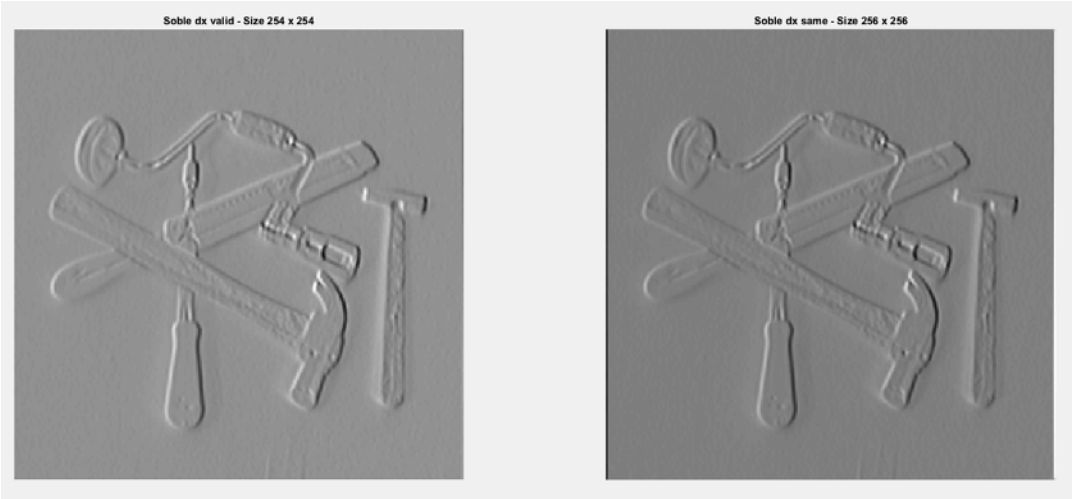

Figure 2: Comparison between using 'same' and 'valid'

2. Point–wise thresholding of gradient magnitudes

Question 2: **Is it easy to find a threshold that result in thin edges? Explain why or why not!**

It is not easy. First, due to the fact that if we increase the threshold, the lines of course will get thinner but a lot of discontinuities will appear. Also, the smoothing makes these lines to be thicker (Figure 3), but without the smoothing (Figure 4), a lot of not–edges lines will be detected as edges wrongly.

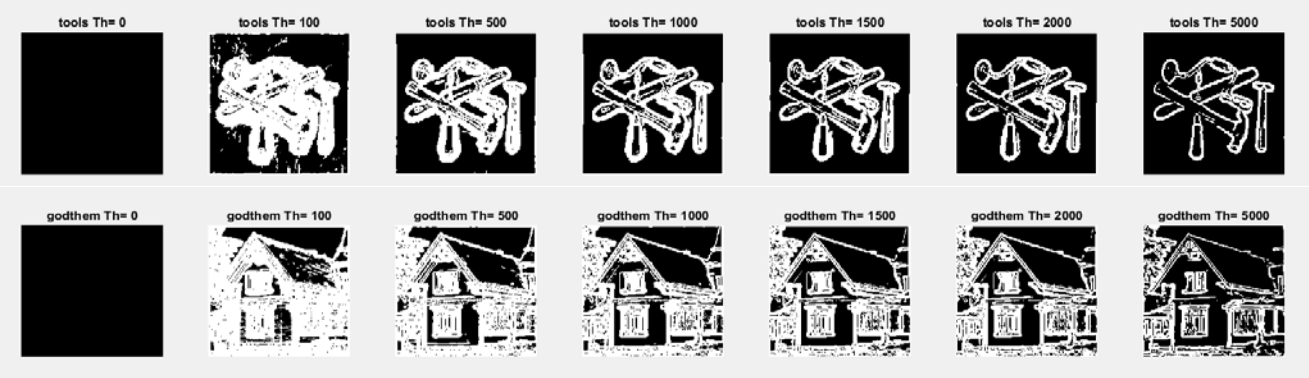

Figure 3: Output after edge detecting and thresholding with previous Gaussian smoothing

Question 3: **Does smoothing the image help to find edges?**

As it was said before, smoothing helps to find edges better as fewer lines will be detected wrongly. This is caused by the blurring effect that the smoothing gives to the image. Due to this, the edges will still be visible in the smoothed image but the pixels that are not part of the edges will look much alike to each other being easy then to differentiate them from the one that are part of the edges.
The comparison is shown in the figure 4.

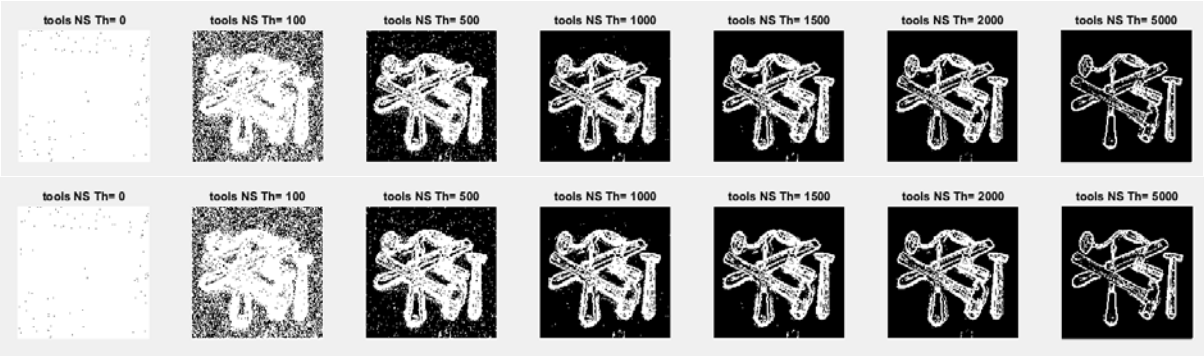

Figure 4: Output after edge detecting and thresholding without previous smoothing

3. Computing differential geometry descriptors

Output of the operation

$$\delta_x(x^n) = nx^{n-1} + (\text{lower order terms}),$$
$$\delta_{x^n}(x^n) = n!$$
$$\delta_{x^{n+k}}(x^n) = 0$$
$$\delta_{x^n}(y^k) = 0$$

We can observe that we get the expected outputs.

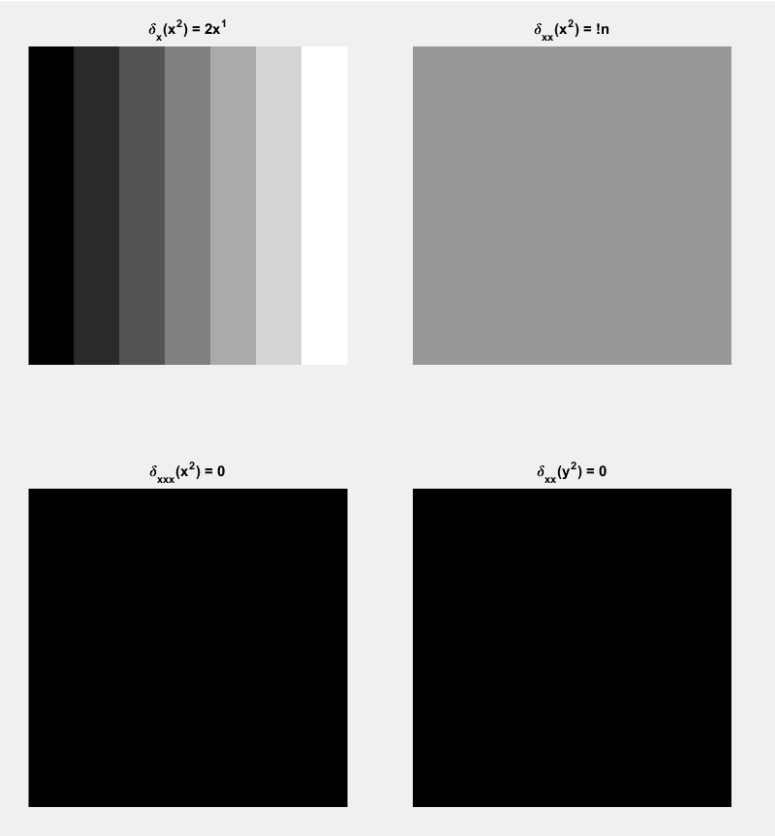

Figure 5: Output after applying first, second and third order operators to different functions

Question 4: **What can you observe? Provide explanation based on the generated images.**

In the figure 6 we can observe the different outputs after applying our second order edge detector. Each image has been pre smoothed with a different value of the variance of the Gaussian filter (Scale): We can observe that for higher values of the scale more lines are filtered and not wrongly detected as edges. Also the sharp shapes (e.g.: corners) get more rounded as this Scale value increases. For higher values of the scale factor the image edges are too distorted and is impossible to identify the main edges of the original image.

Question 5: **Assemble the results of the experiment above into an illustrative collage with the subplot command. Which are yours observations and conclusions?**

Using the third order operators wecan observe that we get thicker lines than in with the second order ones, but the edges remains clearer for higher values of the scale than when we use the second order operator. As in the second order operator, for low scale values the edges are not correctly detected as many lines are mistaken as edges.

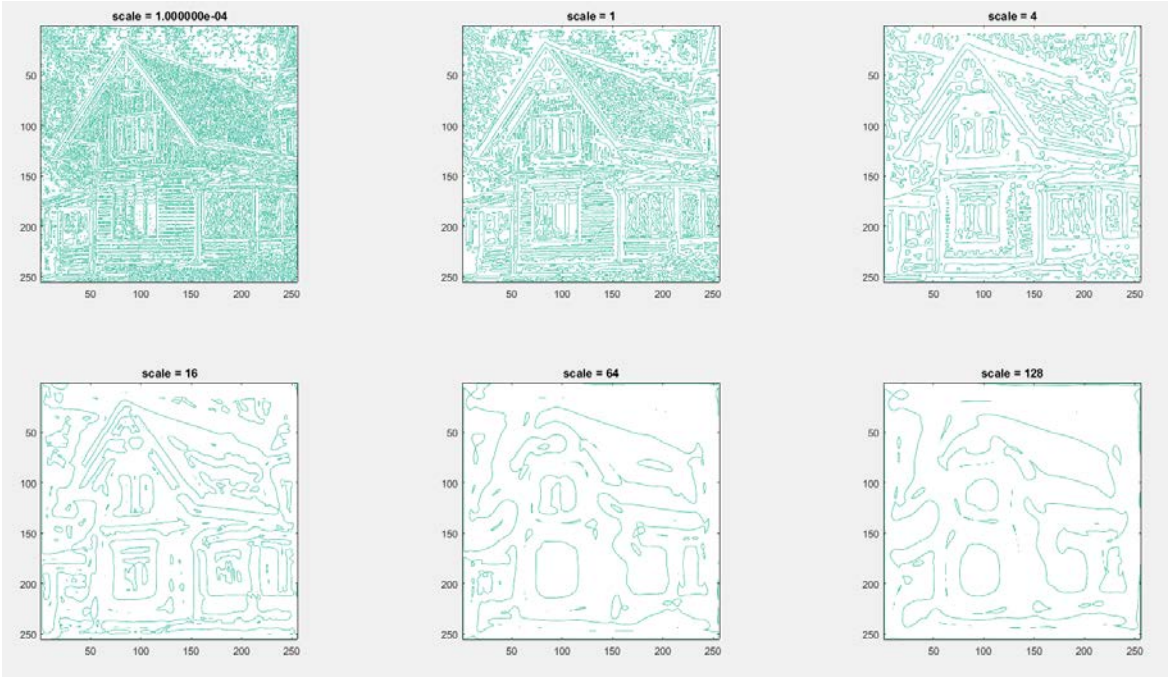

Figure 6: contours output after applying smoothing and a second order edge detector. The scale is the variance of the Gaussian filter.

Question 6: How can you use the response from Lvv to detect edges, and how can you improve the result by using Lvvv?

The problem with the Lvv is that its zero–crossing can mean both a max or min value of the gradient magnitude. The problem with this is that our edges are represented by local max values of this gradient. Using Lvvv we can know if these zero–crossings are a local maxima or minima, and therefore, discard min value zero–crossings and get the actual edges of the image.
This is why in the code we use the < 0, to only show in the output image the max value zero–crossing, that represent the actual edges.

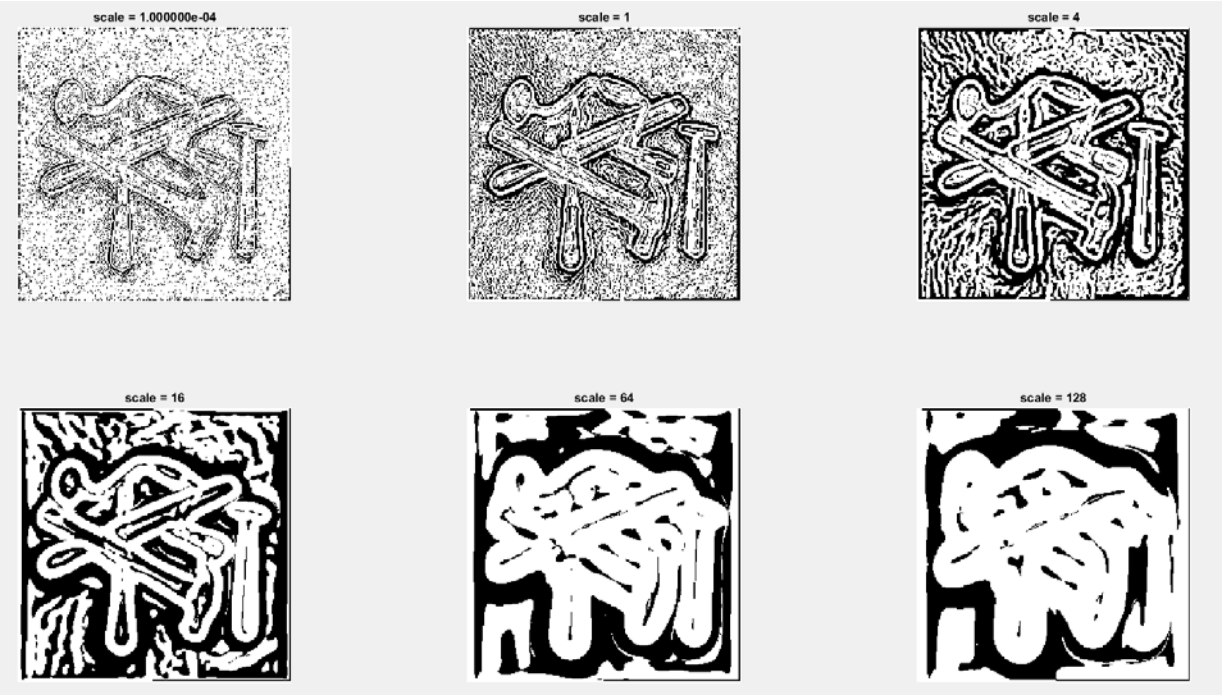

Figure 7: grey image output after applying smoothing and a third order edge detector. The scale is the variance of the Gaussian filter.

4. Extraction of image edges

Question 7: **Output after applying the function njetedge for scales values of 0.0001 1.0 2.0 6.0 10.0 12.0 Threshold is set to 80.**

Figure 8: Output after applying the function `njetedge()` for scales values of 0.0001 1.0 2.0 6.0 10.0 12.0. Threshold is set to 80.

5. Hough Transforms

Question 8: Identify the correspondences between the strongest peaks in the accumulator and line segments in the output image. Doing so convince yourself that the implementation is correct. Summarize the results of this study and print out on paper.

The figure 9 was built in order to explain the system of coordinates of the accumulator (Hough space) and the Hough lines in the real space in order to identify which line is which local maxima in the accumulator.

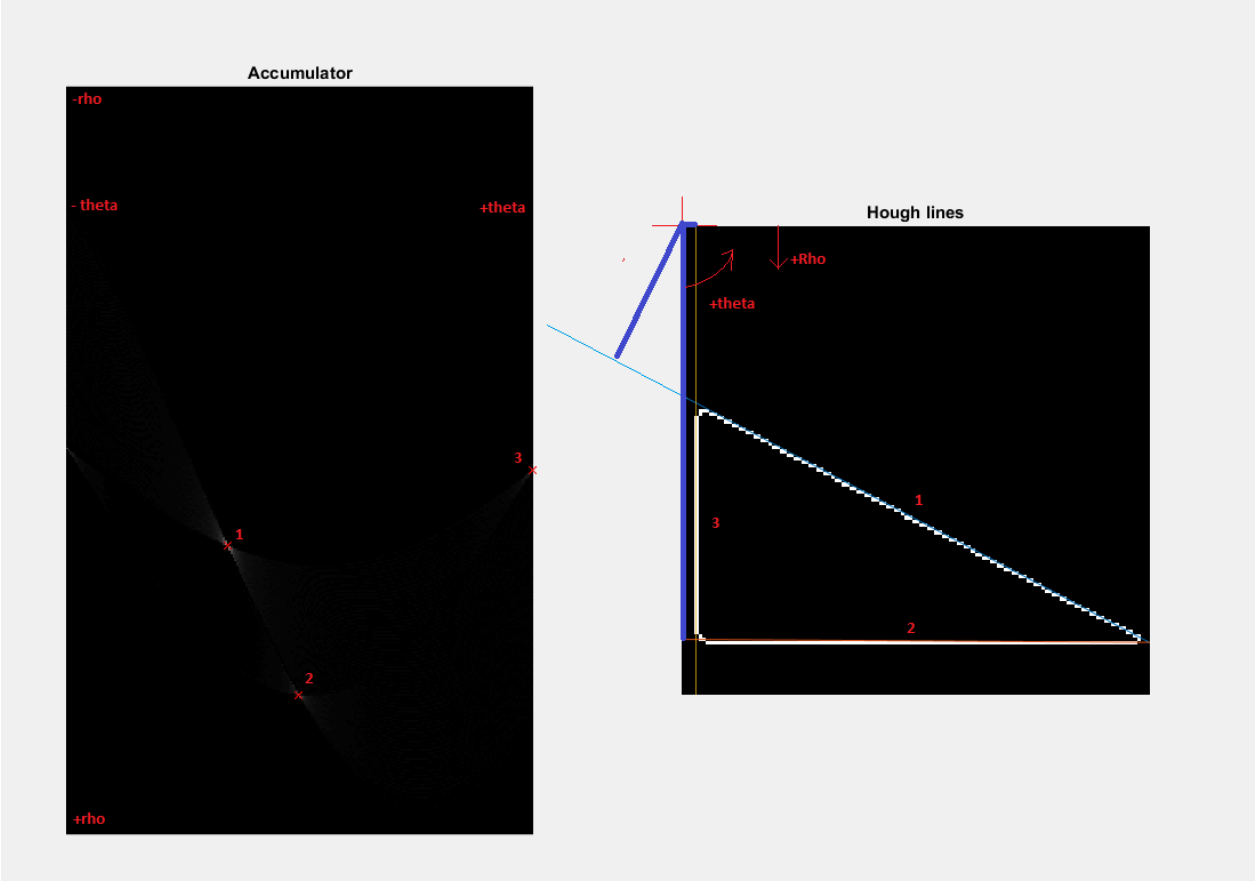

Figure 9: Coordinates representation and correspondence between accumulator points in the Hough space and the final Hough lines.

Analyzing the picture we can easily see that the biggest positive theta is the one for the line 3, and we also have that it is the smallest positive value of rho (shortest blue line). It has to be then the line number 3. The biggest rho is obtained for the line number 2 (longest blue line), and also we have a theta value close to cero. This is the same in the accumulator; point 2, as theta is closer to zero (it is in the middle of the x axis) and the rho is the biggest one. Point number 1 is the only one with negative theta, as line number 1 is defined by a negative theta.

Question 9: How do the results and computational time depend on the number of cells in the accumulator?

In the figure 10 it is shown the output after applying the algorithm in the same image with the same parameters and only changing the size of the accumulator (choosing different values of number of rhos and thetas)

We can observe that if we increase the size of the accumulator, we get more precise lines, as now the accumulator is bigger we have more ponts and the local maxima are easier to identify. But on the contrary, it is also easier to mistake a point in the accumulator grid as an edge line as more 'details' of the image are included now in the hough space gird.

Also, and obviously, the computation time increases notably when we increase the number of thetas as the accumulator is computed using every point of each curve with every theta to obtain the possible rhos. So more thetas implies more computational time. Increasing the number of rhos does not increase the computational time a lot as the increased due to rho is related to the output but not the computations of the algorithm (Figure 11)

Figure 10: output after applying the algorithm in the same image with the same parameters and only changing the size of the accumulator.

Figure 11: Comparison of computational time between changing only the number of rhos and changing only the number of thetas

5.2 Choice of the accumulator incrementation function

Question 10*:* **How do you propose to do this? Relate your answer to the different parts of the images.**

For the implementation of this part I had to change the line that I used to update the accumulator:

```
accumulator(rho_bin,i) = accumulator(rho_bin,i) + 1;
```

By the following one:

```
accumulator(rho_bin,i) = accumulator(rho_bin,i) +
                    h*magnitude(roundx,roundy);
```

I chose the line `f(x) = h*y` as my monotically increasing function as it was the simplest I could think of.

I then designed two experiments to help me understand what is happening. First, I wanted to know how different values of h, the proportional parameter, affect the line edge detection and Hough lines identification. The output of this experiment is shown in the figure 12.

Then, after I checked that there were no changes when the proportional parameter is changed (the reason will be explained later), I wanted to see if there were any

differences between adding the magnitude to the accumulator value and just adding one. For h = 1, the results of this experiments are shown in the figure 14.

As we said, the output does not change when we change the proportional value that we use to update the accumulator using the magnitude. This makes sense as the magnitude matrix is the same and we only change the intensity of the whole accumulator.

But, as the shape of the accumulator is exactly the same, the detected Hough lines will be the same too. This can be shown in the figure 13, where we can see the accumulator for `h = 0.001` and `h = 1000`. The intensity difference is not shown as the `showgrey()` function takes the higher value of the input matrix as white and the lower as black. This is why both images are exactly the same.

Figure 12: Outputs for different proportional rates (h) for magnitude–proportional update of the accumulator

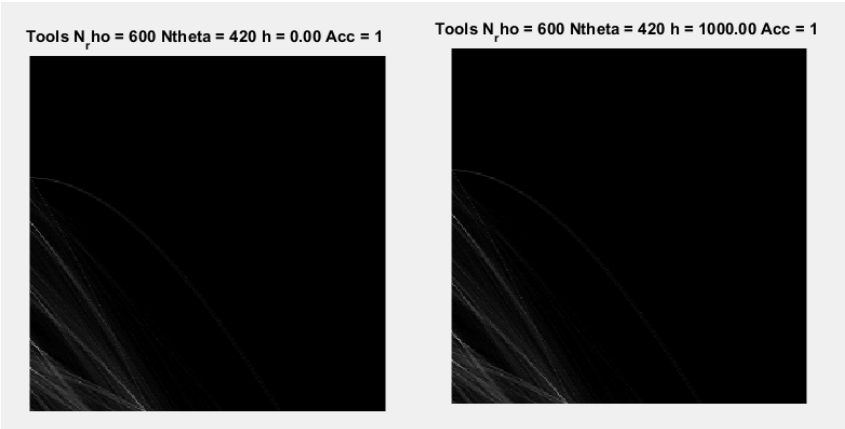

Figure 13: Hough space accumulator representation for h = 0.001 and h = 1000

However, in the figure 14 we can see that the lines detected are not the same when we use the unitary update or the proportional update. While in the unitary update better lines are detected, in the proportional some of this lines are missed. If we take a look to the figure 15 we can see that the accumulator look the same, but some lines have less intensity in the proportional update. This makes sense, as the intensity of the lines will be proportional to the magnitude of the detected lines, while in the unitary upload, the 'weight'given to every update is the same.

We can observed a lot of lines detected on the left side of the images that used the magnitude update. As the magnitude represents the first derivate, it will be bigger when the difference of the intensity between pixels is higher. This is why, using this kind of update, a lot of lines are detected arround this left side,as the difference of intensity is bigger than for instance, between the roof and the sky. (figure 16). This is due to the decreasing orderig that we apply to the local maxima values of the accumulator. As a consequence, the line that represents the roof is not detected using this method, but is perfectly detected using the unitary method.

We should then figure out how to filter this lines, and detect the lower magnitude ones. One way would be to take into account the gradient direction. This is, edges with similar magnitudes and directions of their gradient should be consider only as one. This gradient direction can be expressed by theta. So points with similar values of the gradient and theta should be ignored. This can be done by defining a circle around the higher values of the accumulator so that the values inside this circles would ignored.

The main difference between both methods is that the unitary method gives preference to long straig edges and the magnitude methode to the intensity gradient of the straight edges.

Figure 14: Unitary update of the accumulator (top) versus Magnitude–based update for h = 1 (down).

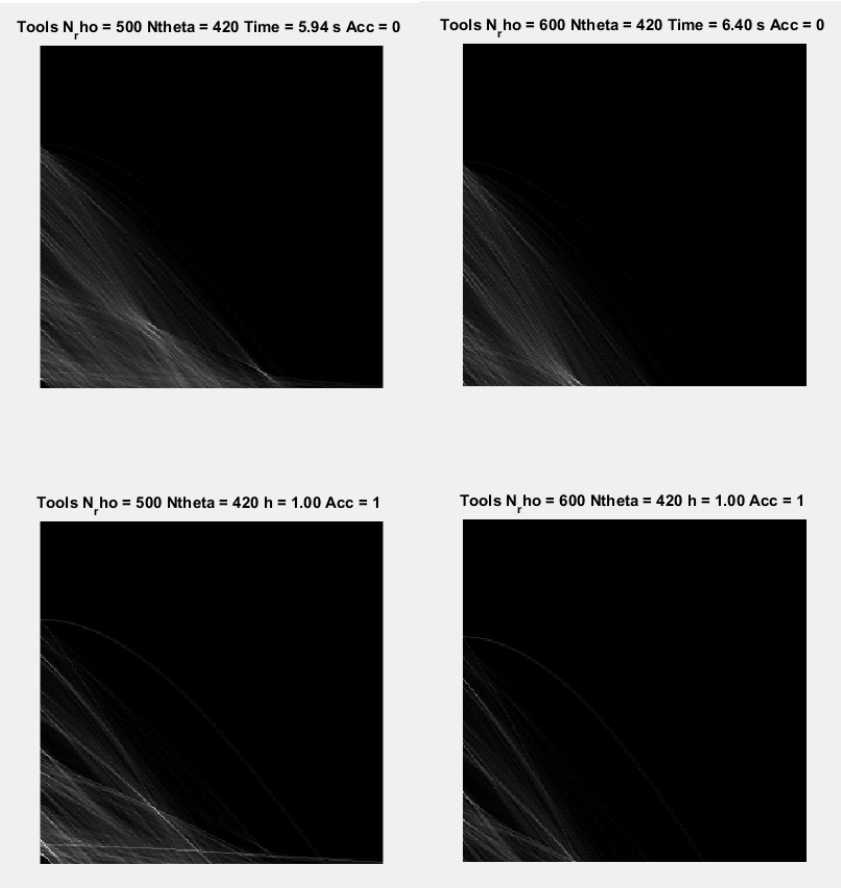

Figure 15: Accumulator representation for unitary update of the accumulator (top) versus Magnitude–based update for h = 1 (down).

Figure 16: Gradient of the 'house' picture.

Laboratorion 1 in DD2423 Image Analysis and Computer Vision

. . . . . .
Student's personal number and name (filled in by student)

. . . . . .
Approved on (date) Course assistant