Answers to questions in Lab 2: Edge detection & Hough transform

Name: _____ *de Gibert, Xavi* AND *Laso, Pablo* _____
Program: _____ *DD2423 CVHT21 HT21-1 Image Analysis and Computer Vision* _____

Instructions: Complete the lab according to the instructions in the notes, and respond to the questions stated below. Keep the answers short and focus on what is essential. Illustrate with figures only when explicitly requested.

Good luck!

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?

Answers:

After creating two difference operators *deltax* and *deltay*, we expect *deltax* to capture the variations in the **x-axis**, and *deltay*, for the **y-axis**, that is, the **vertical (|) and horizontal (-) edges**, respectively. We can observe results in Figure 1 that this is actually fulfilled.
By the definition of the *Sobel filter*, we also expect some smoothing and therefore robustness to noise. On the other hand, *Roberts filter* might be more susceptible to noise -but also defines thinner edges.

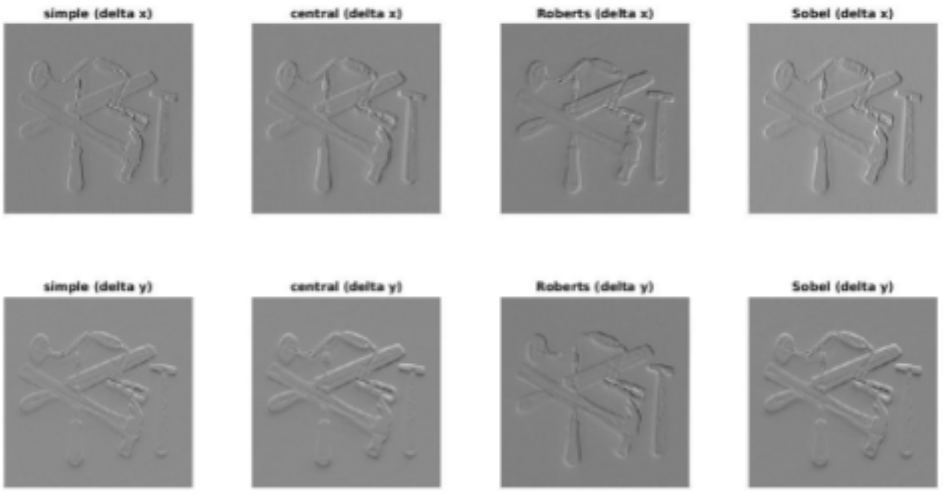


Figure 1. Discrete derivation approximations after applying each of the different filters.

The size of *dxtools* and the size of *tools* are different because we are specifying the parameter 'valid', and so, the "conv" function returns only those parts of the convolution that are computed without the zero-padded edges, i.e., LENGTH(C) is:

MAX(LENGTH(A)-MAX(0,LENGTH(B)-1),0).

In other words, since we are not using *padding* (i.e., "making up" pixels at the edges of the image such that our filters can actually perform convolution in the pixels at the edges), filters cannot mathematically compute the value of the edge pixels, since part of their matrix is "outside" the image. The new size of the image is: 254 x 254 because we use a 3x3 filter. Thus, the returned image must be slightly smaller. Alternatively, we could perform "padding", and the resulting image would be the same size.

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

Answers:

In Figure 2 is depicted the **histogram** of the image, after computing an approximation of the gradient magnitude. **Thresholding** is a fairly straightforward method to segment an image into a foreground and background object. However, this is not always possible (e.g. the image might be more complex -or even if it is simple, the *lighting* throughout the picture might vary).

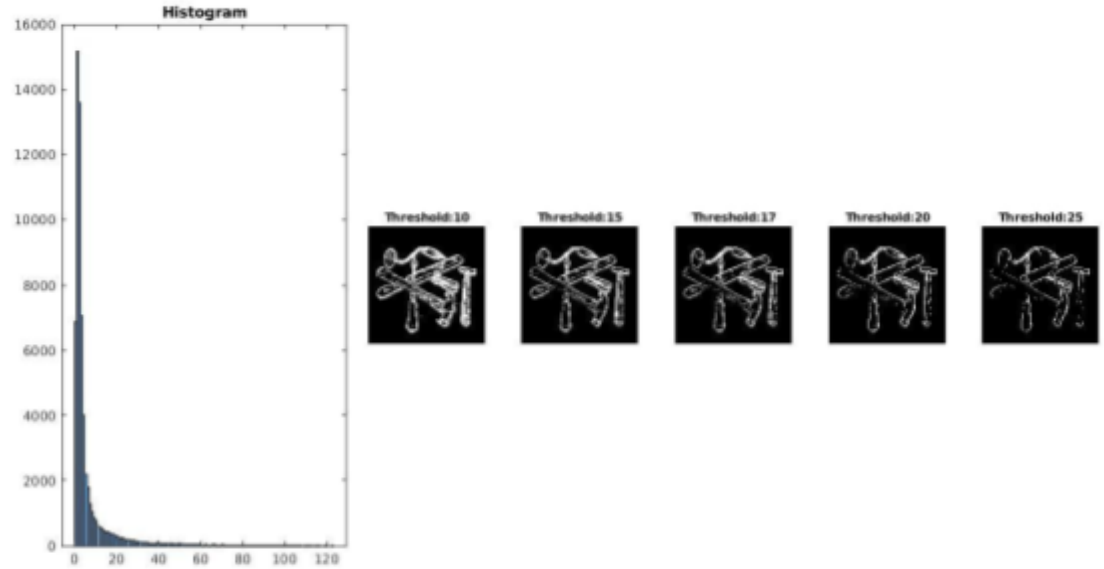


Figure 2. Histogram (left) and thresholding technique with different values (right).

Figure 3 illustrates the effect of **smoothing prior to thresholding**. We can observe how noise is reduced when we smooth prior to thresholding. As when in the first case, when no smoothing is applied, high threshold values will make the image eventually fade away. Within this limit (image and edges still "conserved"), it is flagrant that smoothing is able to do a better job at separating two objects (house and tress from background) than no smoothing

does, since it also takes in too much noise. Smoothing removes this noise, yielding a better segmentation before reaching a limit where noise -but also some edges!- are lost.

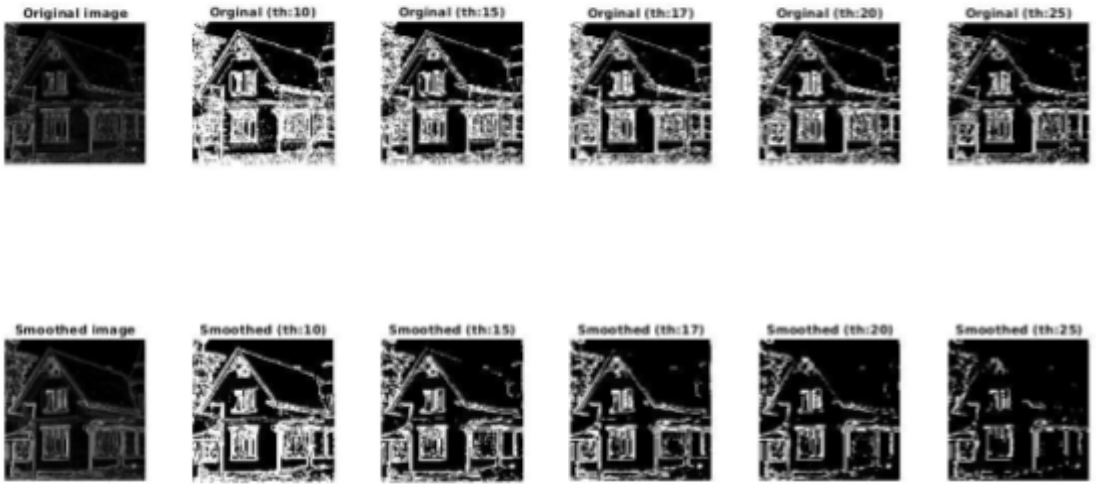


Figure 3. Thresholding only (top) and thresholding after smoothing (bottom).

Unsurprisingly, there is a **trade-off** between thin edges and detection. We can indeed increase the threshold to only capture large variations in the image. However, whereas this will make some edges thinner, others (whose variations are respectively smaller) will be erased.
That is, it is easy to find a threshold that results in thin edges… up to the point where other relevant edges are removed (at which point we should stop increasing our threshold).

NB: Appropriate threshold values could be "valleys" in the histogram pixels because it might mean a clear separation between intensities and a sharp transition will be formed there. However, this might not always be the case, and other techniques may be needed, such as *Otsu's method* (which maximizes "between class variance" of the segmented classes), or *adaptive thresholding* (especially useful, for instance, when the image has different lightning in different regions).

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

Answers:

When segmenting, we try to find the edges, that is, areas in the image in which there are high frequency variations. However, not all variations are due to the transitioning between different objects. It might happen that the image is too noisy. **Smoothing** the image might therefore help us to first get rid of these "unhelpful" variations, so that we are only left with the relevant edges.
To sum, we get rid of some high frequencies in the image (noise... and edges, too) so that only the edges remain (i.e., try to remove noise only). We must be nonetheless careful not to exceed the limit, where we may very well remove edges, too.

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

Answers:

We can observe, from Figure 4, a decrease in lines/edges detected as the **scale value** grows. However, image distortion may ensue from an excessively high scale value. That is, **smoothing** images will make it easier to detect edges, but there will be a point which will, when surpassed, lead to a distorted image.

NB: Second derivative is a fairly robust method to detect edges. However, it is also susceptible to noise. This is why it is commonly combined with smoothing.

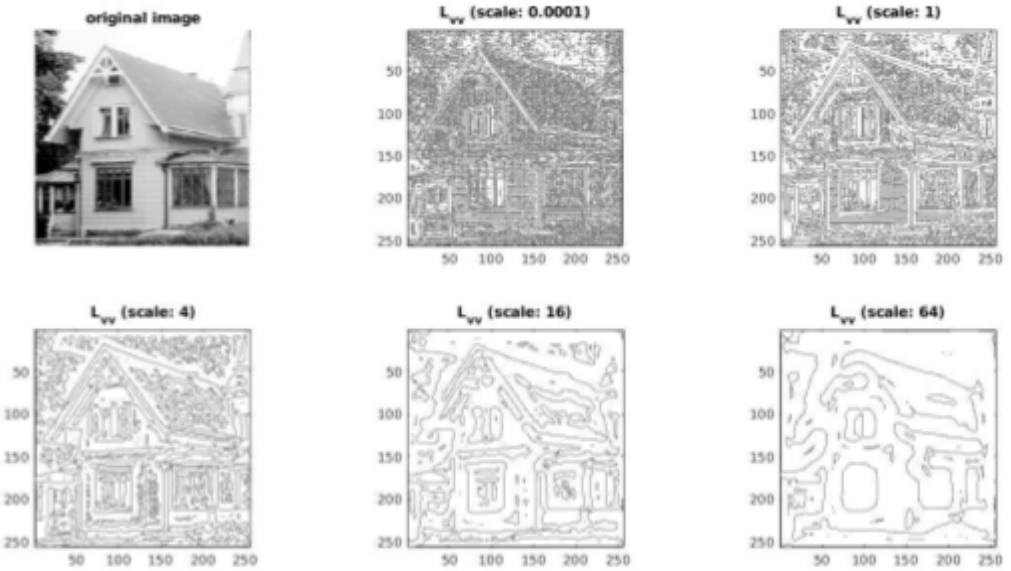


Figure 4. Second derivative-based edge extraction (with different scales).

Choosing the **second derivative** to be zero[1] (that is, a maximum in the first derivative; which is, in turn, a variation in the original image) as a condition to detect edges will result in a high number of edges observed (every time the second derivative shifted between negative and positive values, making $L_{vv}=0$). Smoothing will reduce high frequencies and make it easier for the algorithm to recognize the most important edges only (possibly discarding noise, too).

Choosing a **third derivative** to be negative (that is, a change in the second derivative from positive (when "climbing up" the maximum in the first derivative) to negative ("climbing down" the maximum in the first derivative)) as a condition for edge detection has similar results as the previous condition, giving rise to white areas whenever $L_{vvv}<0$. However, we are able to distinguish some aspects:

- Zero crossing produces thinner edges.
- Noise reduction.
- Zero crossing creates closed loops (spaghetti effect).
- Sophisticated computation (third derivative implies second and first derivative, too).

[1] aka zero-crossing

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

Answers:

As depicted in Figure 4, thin dark curves correspond to edges in L_{vv}. Now, as Figure 5 illustrates, white thick lines correspond to the edges in L_{vvv}. L_{vv} yields noisier images, since there are objects detected that do not correspond to edges. Increasing variance might ensue in detail loss in the image. However, edges are thinner and, hence, more precise. L_{vvv} yields clearer edges (gets rid of noise), although they are also thicker (as variance is increased), i.e., less precise than L_{vv}.
To sum, each algorithm has its own advantages and drawbacks -and their separate, individual results are not optimal.

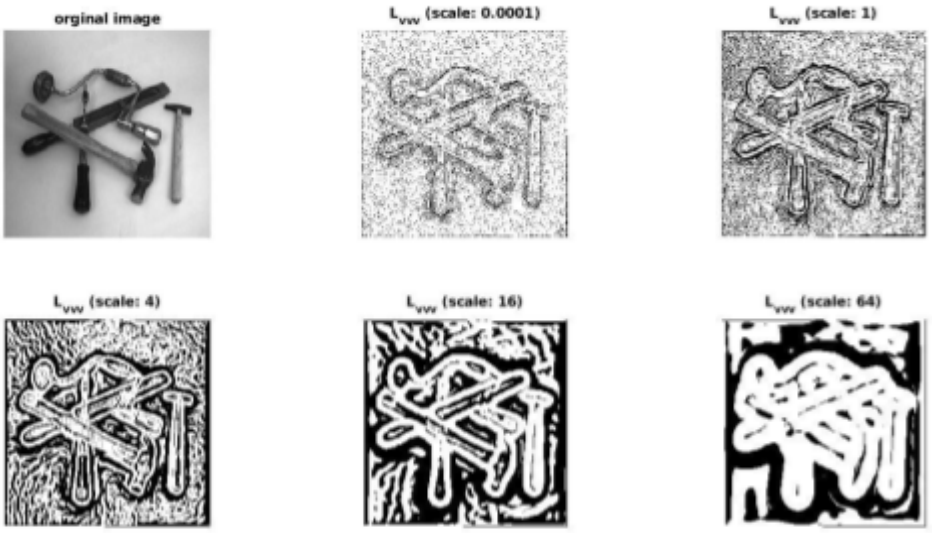


Figure 5. Third derivative-based edge extraction (for different scale values).

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

Answers:

As aforementioned, each method has its own pros and cons. Mixing both ideas might however reinforce the definition of edge in the image, i.e., taking a local maximum of the gradient magnitude (L_v) where L_{vvv} is also negative.
Thus, we would get thin, precise lines from L_{vv} in the thick, white areas (edges only) given by L_{vvv}.

Question 7: Present your best results obtained with *extractedge* for *house* and *tools*.

Answers:

We first tested some scales (1.0 2.0 4.0 6.0 8.0 10.0 12.0 16.0 32.0 64.0), and then chose the best. Likewise, we choose the optimal threshold from a list ([1 5 10 15 20 30). From all the resulting images[2], the best options (summarized in the table below) are depicted in Figure 6 (left: tools, right: house).

NB: When choosing an appropriate scale, do not choose just the one that seems the best, but maybe some scale below it, since then, after thresholding, the result will actually be better.

image	optimal scale	optimal threshold
tools	8	10
house	8	7

Figure 6. Best scale and threshold option for tools (left) and house (right).

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 in one or more figures.

Answers:

The strongest peaks in the accumulator correspond to the largest lines in the output image, as it has been shown in the following images (Figures 7, 8, 9, and 10).

[2] All tested images are saved in the "/img" folder attached. We could not display all of them here.

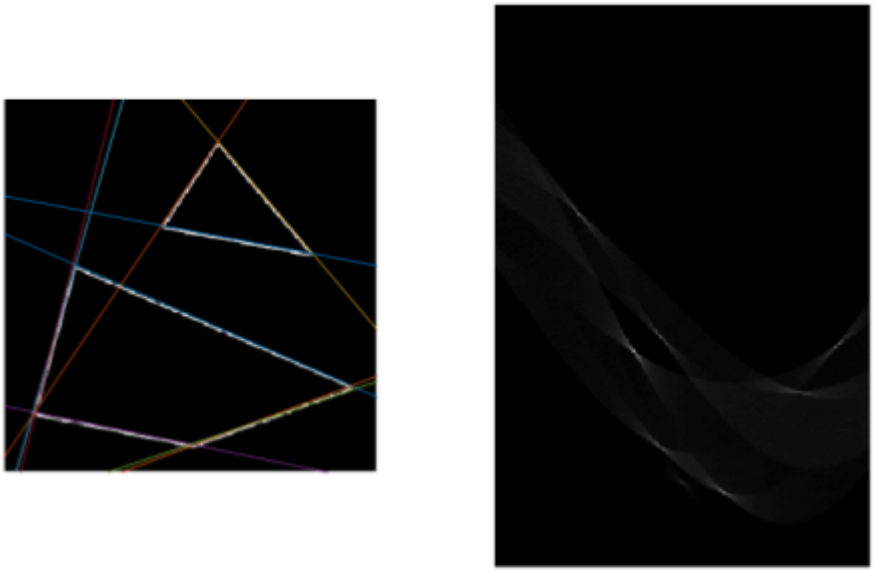


Figure 7. Houghtest256 (nlines=9; Threshold = 100)

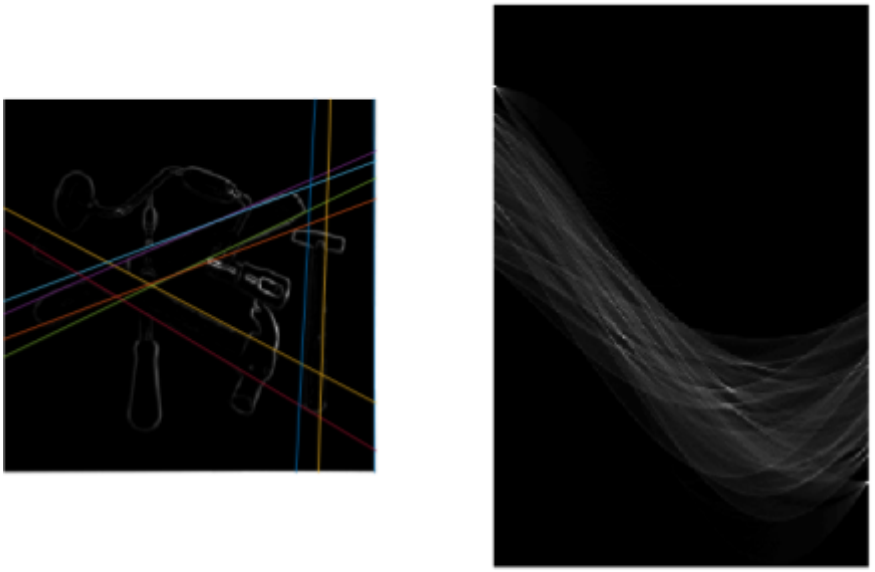


Figure 8. Few256 (nlines=10; Threshold = 1000)

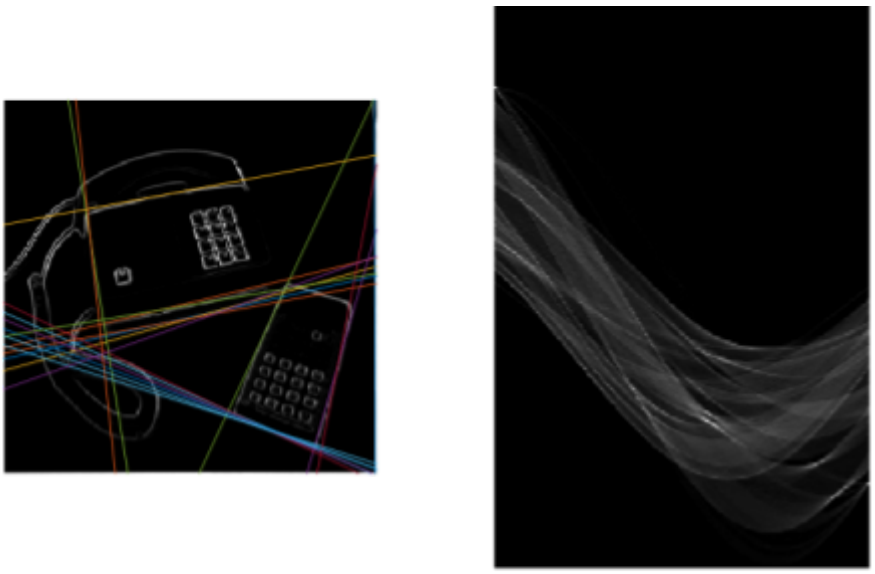


Figure 9. Phonecalc256 (nlines=20; Threshold = 2200)

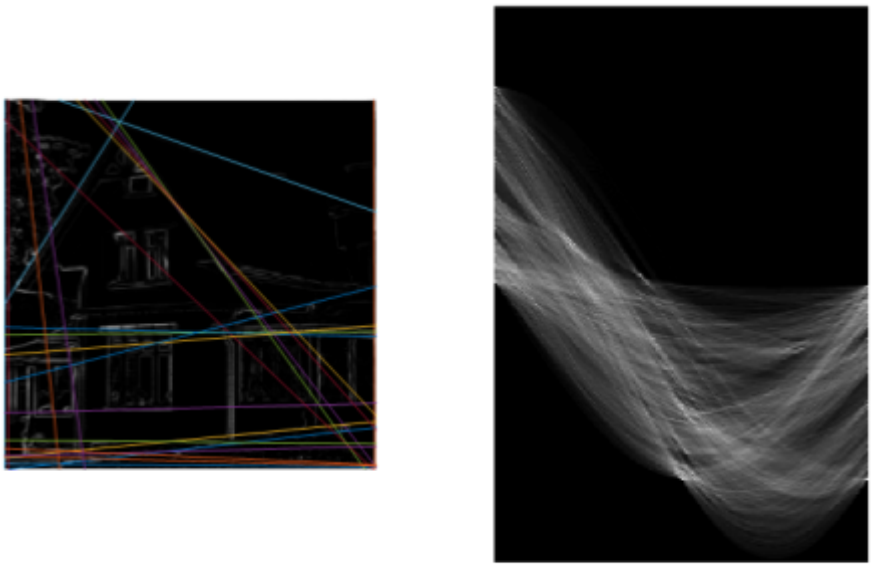


Figure 10. Godthem256 (nlines = 25; Threshold = 2500)

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

Answers:

We can think of the *Hough Transform* as an algorithm that will search for an edge point (top to bottom, left to right) and start iterating for each possible of rho and theta[3], and define the

[3] Rho is the perpendicular distance from origin to the line, and theta is the angle formed by this perpendicular line and horizontal axis, measured in counter-clockwise.

occurrences of these points. Increasing theta and rho values will result in a higher computational cost and longer time.
Low number of rho values introduce some errors, the lines do not correspond totally to their edges, they are shifted. On the other hand, if we have a low number of theta values, we can observe some angular variations. If we increase the number of rho or theta values, the computational time will increase due to the extra parameters. Additionally, if we increase the number of rho and theta values, we will achieve much more precision with the points and curves due to it will be easier for the accumulator to define the position in spatial domain. Examples are depicted in the following Figures 11-14.

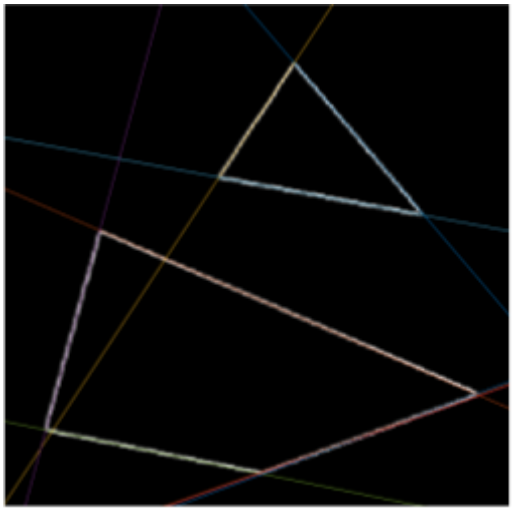


Figure 11. High number of rho values.

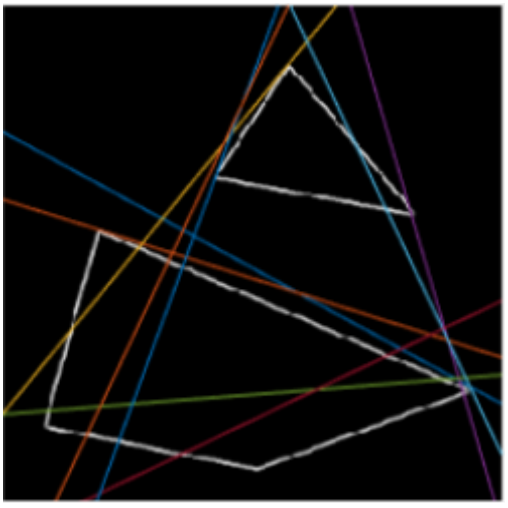


Figure 12. Low number of rho values.

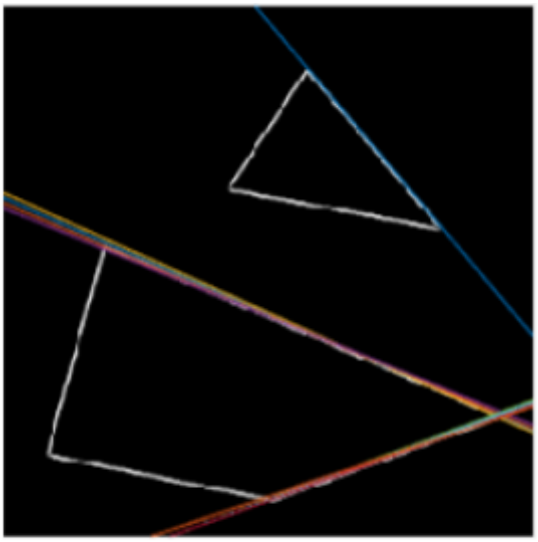


Figure 13. High number of theta values.

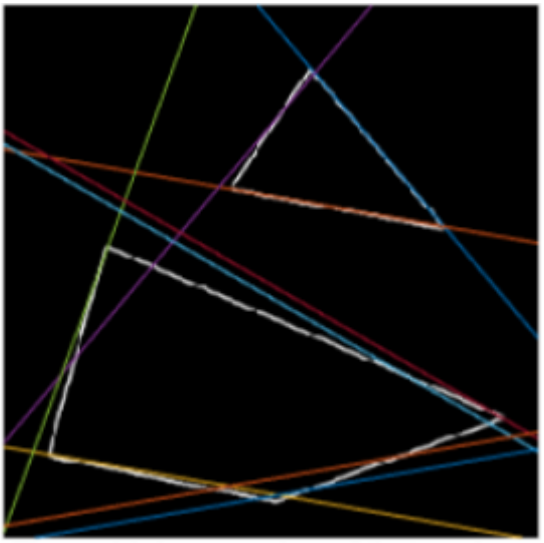
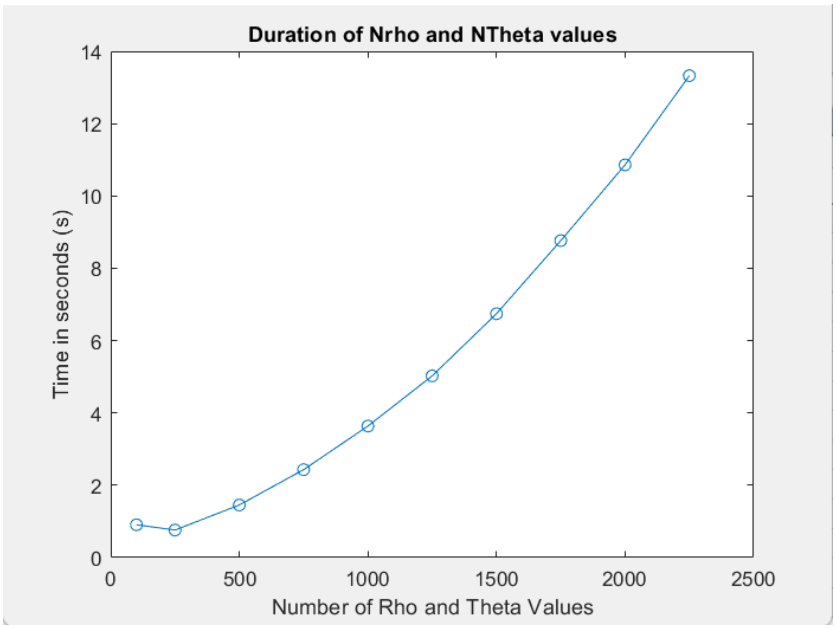


Figure 14. Low number of theta values

The following chart represents the computational time needed to compute the different transformation depending on the number of rho and theta values.



Question 10: How do you propose to do this? Try out a function that you would suggest and see if it improves the results. Does it?

Answers:

The **accumulative** is updated by default with an increment of one unit, i.e.:

$$acc(j, i) = acc(j, i) + 1$$

Alternatively, we can set such increment (∇L) to some monotonically increasing function proportional to the gradient magnitude (∇L), i.e., $\Delta S = h(|\nabla L|)$. "h" is such function as the **logarithm** of the magnitude[4], or the **magnitude** itself (linear, or raised to some power), i.e.:

$$acc(j, i) = acc(j, i) + \log(mag)$$
$$acc(j, i) = acc(j, i) + (mag)^{\{power\}}$$

---

tq, yo +

[4] magnitude = `sqrt(Lv(img)`