Assignment 2 - DD2423 - Computer Vision and Image Processing

Alexandra Hotti

November 30, 2018

1 Difference operators

Here, two difference operators delta x and delta y that approximate the first order partial derivatives in two orthogonal directions are applied to an image. These are created using simple difference operators, central difference operators, Robert's diagonal operators and the Sobel operators.

1.1 Results

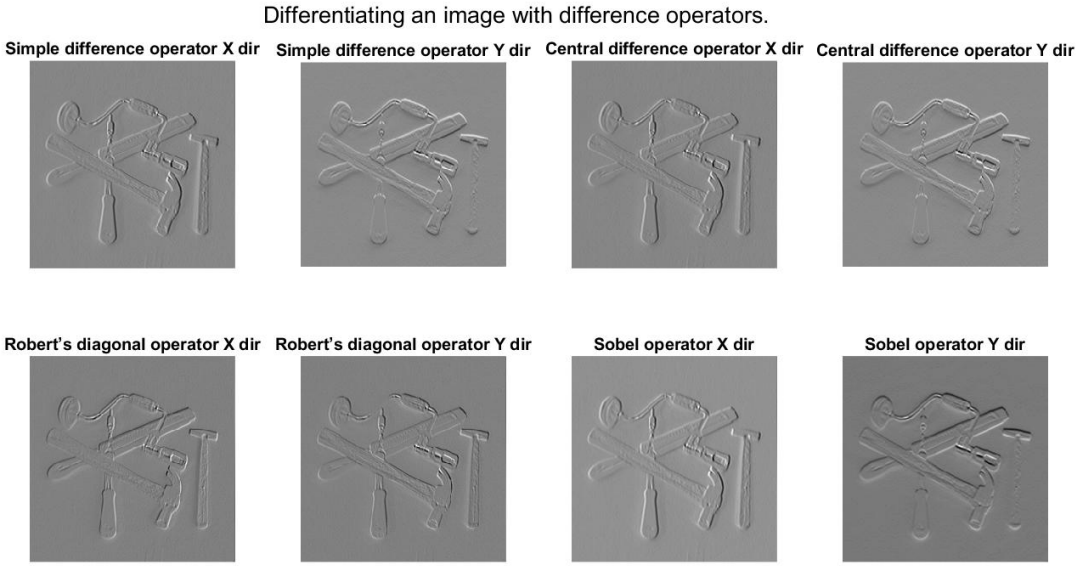


Figure 1: The first row contains the result from sub sampling an image to lower resolutions. On the second row the sub sampled images on the first row have been smoothed via a Gaussian Filter and on the third row via an Ideal Low Pass Filter.

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?

What we expect the result to look like:
We compute discrete derivative approximations in the x and y direction. Thus, when we use the x-wise difference operators we get the changes in the x-direction between pixels. Which means that we see changes and thus max and mins in this direction in the image in the x-direction when the x-wise operator is used. Then when they-wise operator is used we see the minimas and maximas in the y-direction, i.e. were edges occur in the image. See Figure 1.

Compare the size of dxtools with the size of tools. Why are these sizes different?

The sizes of the differentiated image when several different filters are used differ in size because the filters differ in size. When padding is not used the filters of different sizes will "fit" onto the image a varying amount of times. The Sobel oerator for instance is a 3x3 filter with its center pixel placed in the upper left corner. Thus, if padding is not used, the filter will have to skip the two last rows as well as the two last columns.

Table 1: Sizes of differentiation filters.

Operator	size dx filter	sixe dy filter
Simple difference operator	1x3	3x1
Sobel operator	3x3	3x3
Robert's operator	2x2	2x2
Central difference operator	1x3	3x1

Table 2: Size of differentiation in X-direction of image using several differences operators on an image of size 256x256.

Operator	X size	Y size
Simple difference operator	256	254
Sobel operator	254	254
Robert's operator	255	255
Central difference operator	256	254

Table 3: Size of differentiation in Y-direction of image using several differences operators on an image of size 256x256.

Operator	X size	Y size
Simple difference operator	254	256
Sobel operator	254	254
Robert's operator	255	255
Central difference operator	254	256

2 Point-wise thresholding of gradient magnitudes

Below the histograms of an image are computed and used to guess a threshold that yields reasonably thin edges when applied to a gradient magnitude approximation.

In Figure 2 these histograms can be found. On the x-axises we have the pixel intensity magnitudes values and on the y-axises we have the number of pixels for the different magnitudes. Thus, by inspecting the histograms we can find an approximate range for were a good threshold for thin edges should lie. We want to keep high magnitudes (i.e. large intensity changes = edges), thus we remove allot of the pixels with low magnitudes that are just part of the background. Therefore, the sobel operator for instance gets a higher threshold compared to the central difference operator which has lower pixel intensity magnitudes.

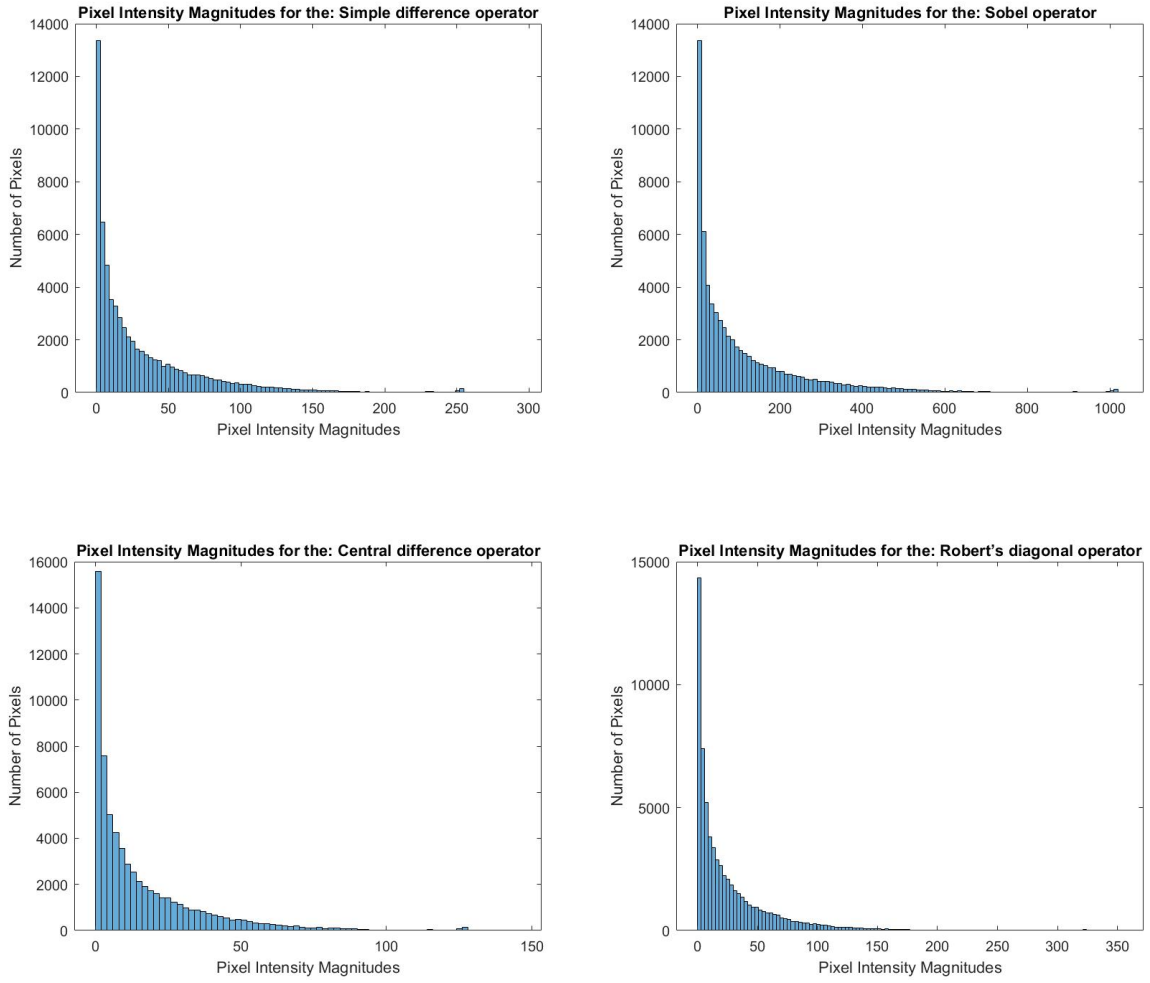


Figure 2: Histograms for the pixel intensity magnitudes for the gradients.

Here the thresholds found above are used to threshold the gradient magnitude computations of an image. The images below are the binary output of gradient magnitude pixel intensities

over the given threshold, i.e. were edges should occur.

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

Edges are not concise. They do not have the same concise steepness/magnitude throughout the entire edge. Therefore, if we use a too low threshold we get too thick edges since we allow too low magnitudes to pass. If we instead increase the threshold edges that do not have a concisely strong magnitude throughout the edge get patches. Also, low magnitude edges disappear. For instance look at the picture of the house in Figure 3 in many of the images the ceiling edge is almost gone. Therefore, it is hard to get thin edges with high magnitude thresholds that do not have patchy edges.

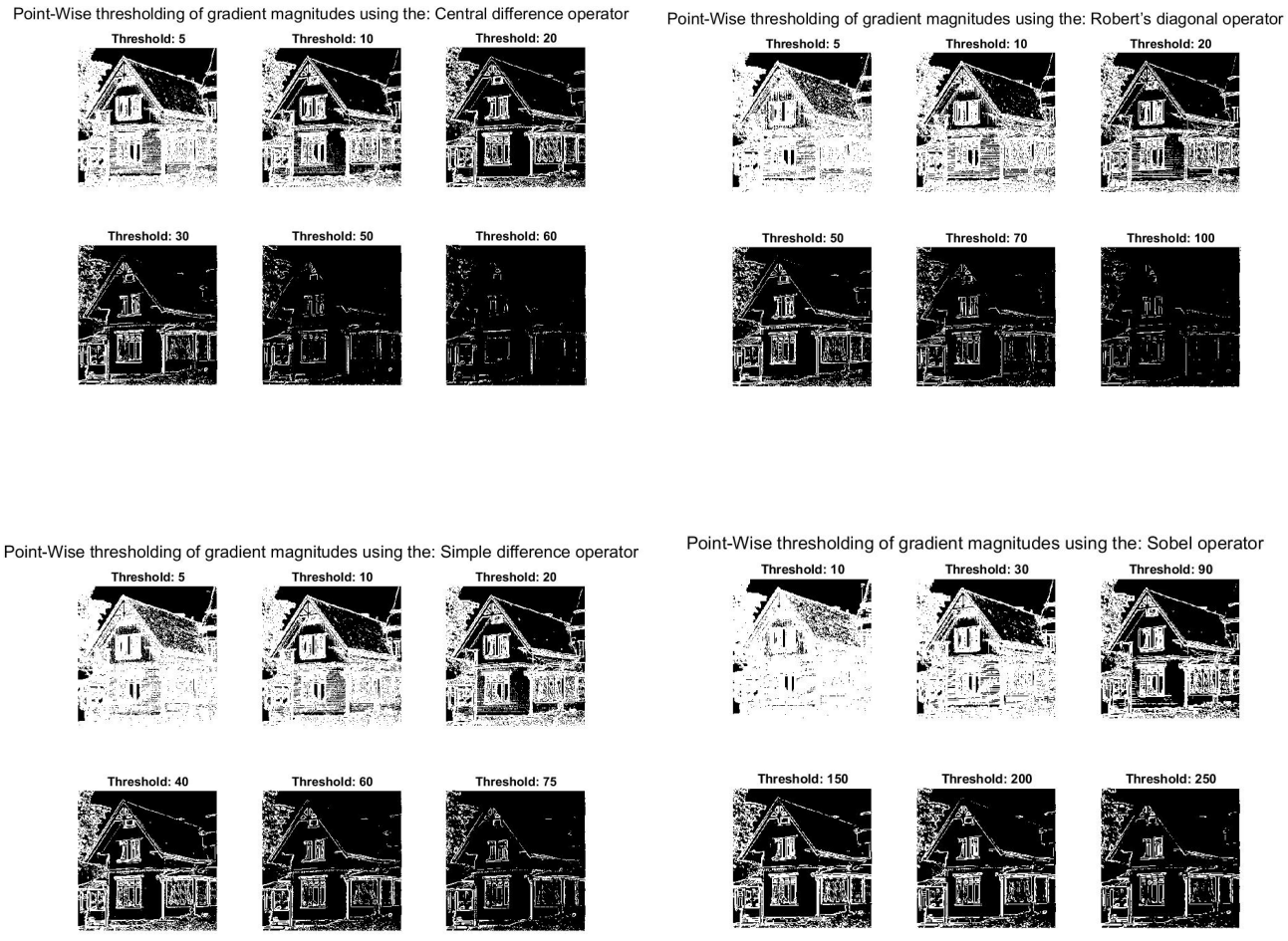


Figure 3: Thresholded gradient magnitudes of an images.

2.1 Differentiation and noise

Differentiation of an image that contains noise is ill posed as an arbitrary small perturbation in the input can lead to an arbitrarily large perturbation in the output. Below is an example.

$$\begin{aligned} f(x) &= tan^{-1}(x) + \epsilon sin(\omega x) \\ f'(x) &= \frac{1}{1+x^2} + \epsilon\omega cos(\omega x) \end{aligned} \tag{1}$$

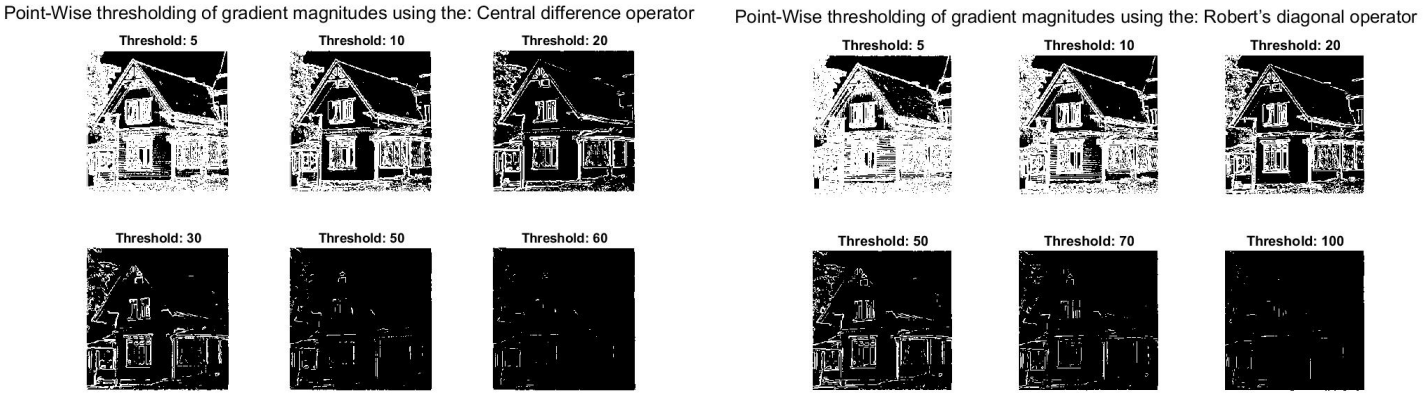
The difference $\epsilon\omega cos(\omega x)$ can be arbitrarily large if $\omega >> \frac{1}{\epsilon}$. Thus, differentiation results in an amplification of the noise.

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

In Figure 4 are the results from smoothing the images with a Gaussian filter with variance 0.5 before differentiating the image.

When applying smoothing there is a trade off between the effects of using it and not using it. When the amount of smoothing is increased more noise is suppressed which results in less false positives. However, increasing the smoothing also disrupts true edges thus the number of true positives decrease. Thus, we perverse thick edges since sharp edges with a large magnitude has a slower increase. All in all, increasing the smoothing suppresses high frequency edges and increases lower frequencies.

Decreasing the amount of smoothing preserves more edges and thus increases the amount of correct feature detection and thus we get more true positives. However, since less noise is suppressed we also get more false positives.



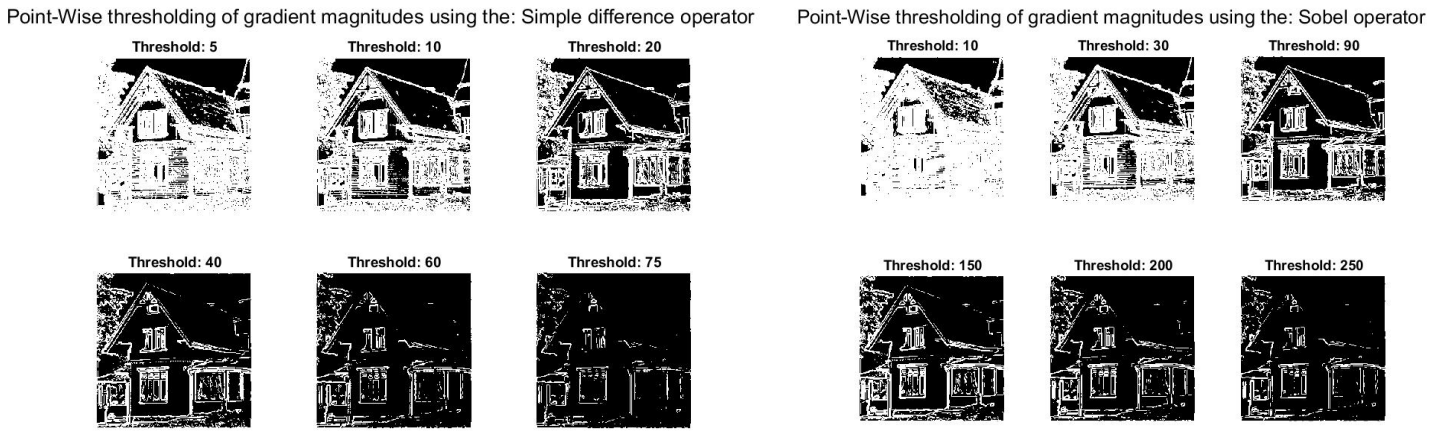


Figure 4: Thresholded gradient magnitudes of an image with Gaussian smoothing with variance 0.5.

3 Differential geometry based edge detection

One way of extracting thin edges is by considering points for which the gradient magnitude reaches local maxima in gradient direction. As it has been mentioned during the lectures, this edge definition can be expressed in differential geometry terms as the second order derivative being equal to zero and the third order derivative being negative. Introduce in each point a local coordinate system (u; v) such that the v direction is parallel to the gradient direction and the u direction is perpendicular to this. Then an edge can be defined as:

$$\begin{cases} L_{vv} = 0 \\ L_{vv} < 0 \end{cases}$$

where L_{vv} and L_{vvv} denote the second and third order derivatives of the smoothened intensity function L in the v direction. We get L from the original image function f by convolving it with a Gaussian kernel.

The expressions for L_{vv} and L_{vvv} can be approximated as:

$$\begin{cases} L_{vv} = 0 \\ L_{vv} < 0 \end{cases}$$

To express the edge definition above in partial derivatives of the smoothened intensity function L with respect to the Cartesian coordinate directions, we write the gradient vector at an arbitrary point (x_0, y_0) as:

$$\bigtriangledown L = \begin{pmatrix} L_x \\ L_y \end{pmatrix} \Bigg|_{(x_0,y_0)} = |\bigtriangledown L| = \begin{pmatrix} cos\varphi \\ sin\varphi \end{pmatrix} \Bigg|_{(x_0,y_0)}$$

Where:

$$cos\varphi = \frac{L_x}{\sqrt{L_x^2 + L_y^2}}$$
$$sin\varphi = \frac{L_y}{\sqrt{L_x^2 + L_y^2}}$$

Using these expressions the derivatives of L can be expressed as:

$$\begin{cases} \tilde{L}_{vv} = L_v^2 L_{vv} = L_x^2 L_{xx} + 2L_x L_y L_{xy} + L_y^2 L_{yy} = 0 \\ \tilde{L}_{vvv} = L_v^3 L_{vvv} = L_x^3 L_{xxx} + 3L_x^2 L_y L_{xxy} + 3L_x L_y^2 L_{xyy} + L_y^3 L_{yyy} < 0 \end{cases}$$

For a more extensive explanation see the Assignment description.

4 Computing differential geometry descriptors

Here two Matlab procedures for $\tilde{L}_{vv}$ and $\tilde{L}_{vvv}$ are written. These are approximated using central difference masks corresponding to discrete derivative approximations. Then, the zero crossings of $\tilde{L}_{vv}$ is computed on an image with the scales 0.0001, 1.0, 4.0, 16.0 and 64.0.

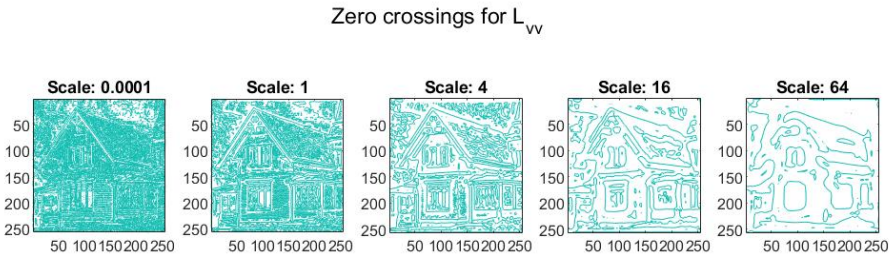


Figure 5: Zero Crossings for L_{vv}.

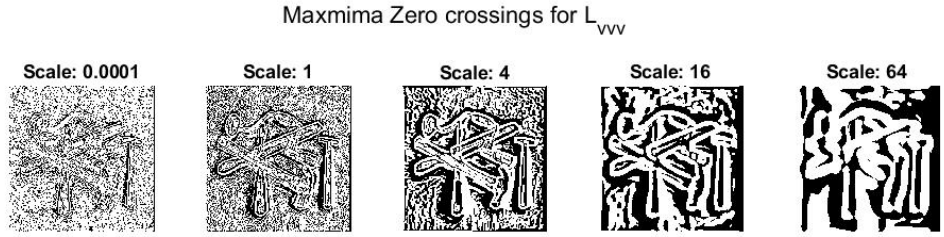


Figure 6: Maxima Zero Crossings for L_{vvv}.

Question 4: What can you observe? Provide explanation based on the generated images. Study the sign of the third order derivative in the gradient direction.

- In Figure 7 and 8 we can see that a lower scale gives a more noisy result do to less smoothing.
- However, increasing the smoothing too much destroys true edges as well. This can for instance be seen in Figure 7 when the scale 64 is used.

Also, if we only look at Figure 7 where $L_{vv} = 0$ we get a Laplacian detector which suffers from 2 problems:

- Zero-crossings of the Laplacian also respond to "false edges" (corresponding to minima in the derivative response for 1D structures). This can for instance be seen in the sky in Figure 7 where the detector finds false edges in the sky.
- It is poor at localizing curved edges. Instead it creates an offset that increases with the curvature of the edge.

Question 5: Which are your observations and conclusions from Figure 8?

- The sign of the gradient is given by L_{vvv} in Figure 8. Evidently there exists max points in the image when the third order derivative is less than 0. These areas become white when the third order differential filter is applied, and the purpose is that these max points should coincide with edges.
- In non maximum areas, when $L_{vvv} >= 0$ the filtered image becomes black, i.e. these areas are non max points and thus non edges.
- Note that the maximum zero crossings of L_{vvv} has also identified non edge areas such as the background of the image. This is solved by combining both the third and second order derivatives from both Figure 7 and Figure 8 such that: $L_{vvv} < 0$ and $L_{vv} = 0$
- Lastly, from the experiments above it appears that increasing the amount of smoothing gives wider, softer edges.

Question 6: How can you use the response from L_{vv} to detect edges, and how can you improve the result by using L_{vvv}?

When L_{vv} equals zero we have found a zero crossing, however this crossing could be either a maxima or a minima. Thus, we can combine this result with the sign of L_{vvv}. If the sign is negative we have a max point, i.e. an edge.

5 Extraction of edge segments

So far we have only considered the result of computing dierential operators at dierent scales and looked at their zero crossings and sign variations. In this exercise we will collect these components to a Matlab function called edgecurves that computes the edges of an image at arbitrary scale and returns these lists of edge points. If the parameter threshold is given, this value shall be used as threshold for the gradient magnitude computed at the same scale. Otherwise, no thresholding shall be used and all edge segments will be returned.

The exercise consists of computing suitable differential quantities that can be used as input data for the functions zerocrosscurves and thresholdcurves, so as to determine edges thresholded with respect to the gradient magnitude at arbitrary scale.

Question 7: When your function works satisfactory, use it to extract edges from the images house and tools. First try different scales e.g. 0.0001, 1.0, 4.0, 16.0 and 64.0. For each image find the best scale and adjust the threshold accordingly. Then overlay the extracted edge curves on top of the original image. Then present your best results obtained with extractedge for house and tools.

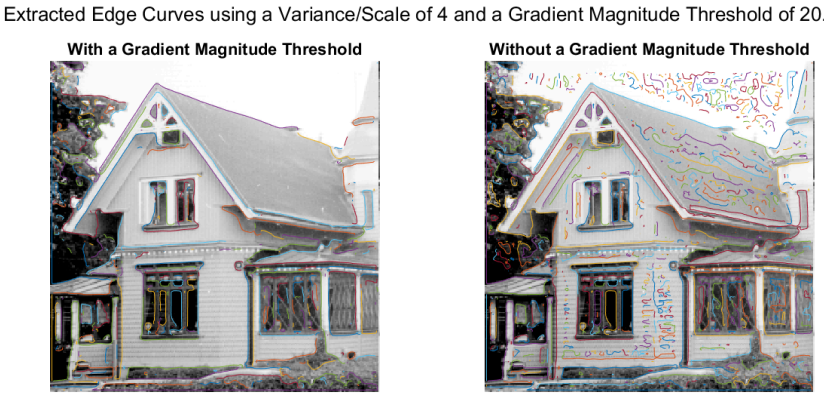


Figure 7: Extracted edges with and without thresholding.

Figure 8: Extracted edges with and without thresholding.

6 Hough transformations

In this exercise a you a Matlab function called houghline is implemented. This function uses the Hough transform to determine linear approximations of a given number of curves, based on parameterizations of the type:

$$x \cdot cos\theta + y \cdot sin\theta = \rho \tag{2}$$

where x and y are suitably defined (e.g. centered) image coordinates, and ρ and θ have appropriately selected definition areas with respect to the choices above. For a more thorough explanation see the assignment description.

Based on this function and extractedge a function called ihoughedgeline is written. This function performs an edge detection step and then applies a Hough transform to the result.

In Figure 9 are some results from hough transformations of a few images.

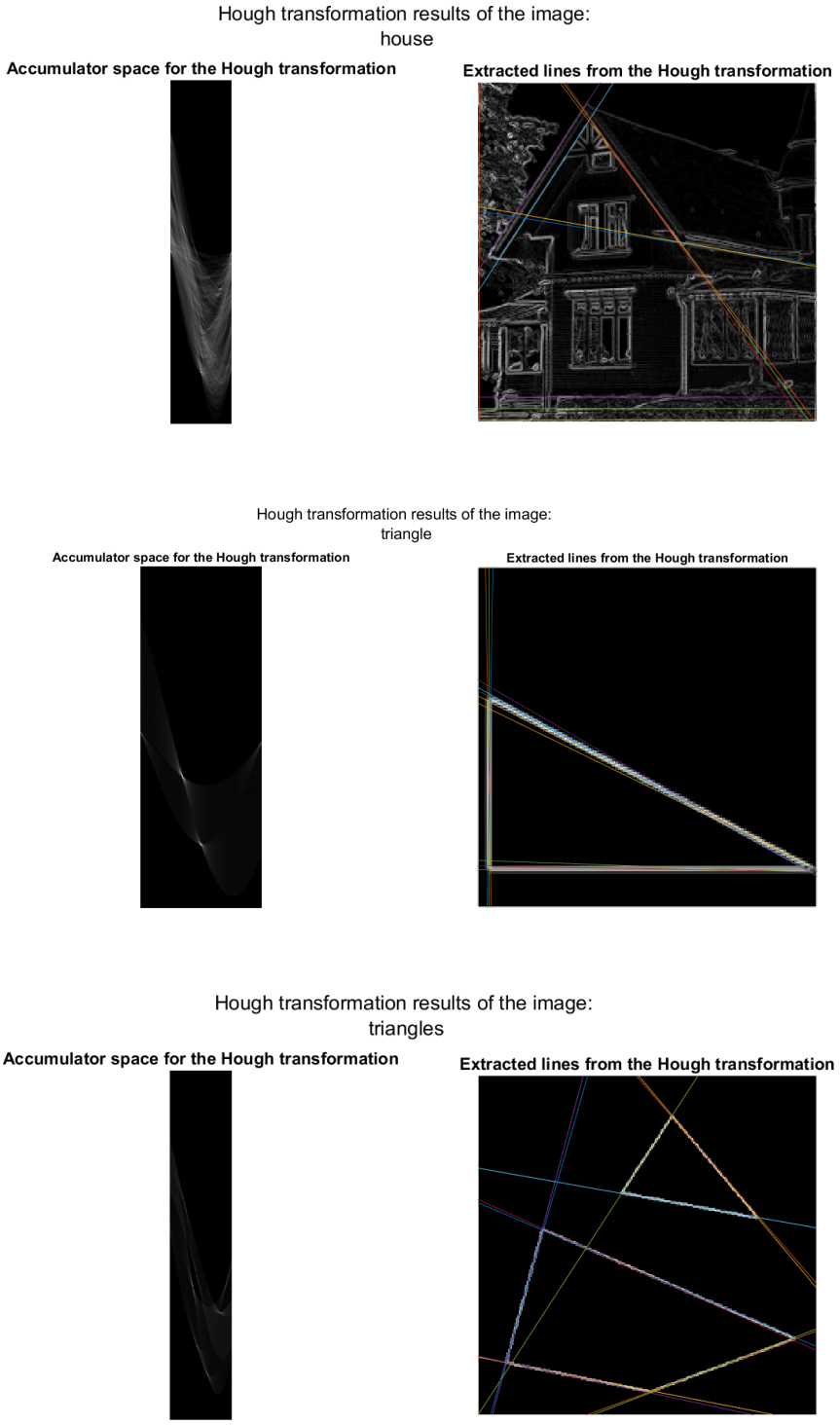


Figure 9: Lines detected in images via hough transformations and their accumulator spaces..

Question 8: Identify the correspondences between the strongest peaks in the accu-mulator and line segments in the output image. Doing so convince yourself that the implementation is correct. Summarize the results of in one or more figures.

Here the relationship between a few of the strongest lines in an image and the accumulator space is displayed in Figure 10. The marked angle is the θ angle of the line. Where θ is the angle formed by the perpendicular distance ρ from the origin to the line and the horizontal axis, measured counter-clockwise.

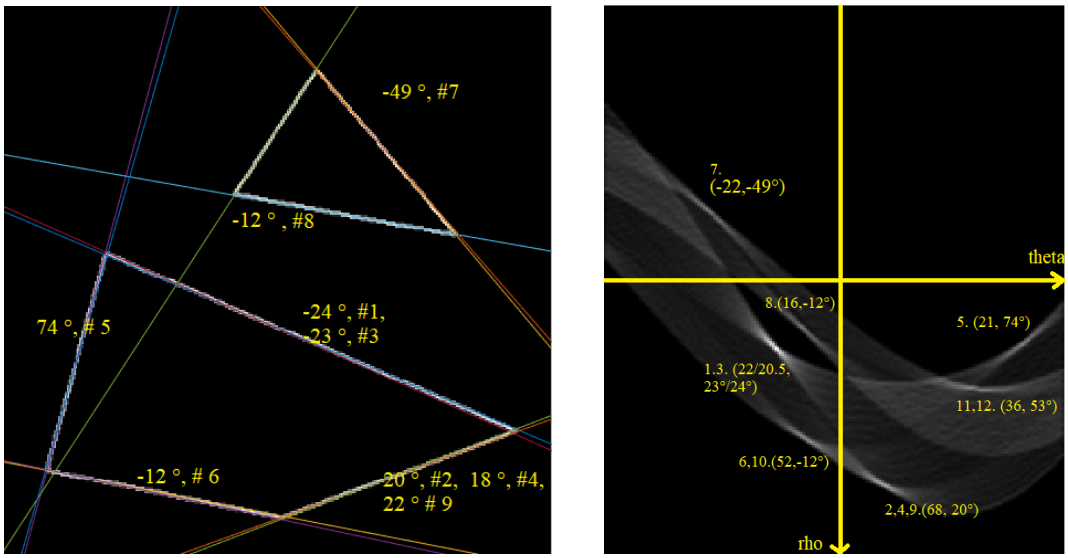


Figure 10: Relationship between θ and in the accumulator space and the corresponding image for a few of the strongest found lines.

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

6.1 Quality

The results improves as the number of cells in the accumulator space increases since we will find a closer match between the calculated value and the accumulator cells value.

However, increasing the number of cells could also result in multiple local maximas per actual image edge/line, where there otherwise would be a single pair of theta and rho values. This gives rise to multiple lines per every single line that we would wish to approximate and thus requires a higher value for lines in order to locate more lines.

Table 4: Sizes of differentiation filters.

	many ρ cells	few ρ cells
many θ cells	We can represent all angels and we can represent many distances to the origin in the image. However, we will get multiple responses for the same line but high accuracy with true edges being detected.	We can represent many angels, but we cannot represent all distances to the origin in the image. Therefore, when we compute the closest possible ρ and lines will have correct angles, but they will have an offset to the closest rho in the accumulator space grid.
few θ cells	We can represent few angels, but we can represent a lot of distances to the origin in the image. Therefore, points will have the correct distance to the origin, but they will have the wrong theta angle which will result in an incorrect position with respect to the line.	We can represent both few angels and few distances to the origin in the image. Therefore things will be placed at wrong distances with respect to the origin and they can also get a considerable offset to the closest ρ to their actual coordinates.

6.2 Time complexity

As the number of cells in the accumulator space increases the computational time will also increase.

First, as we iterate over all possible theta values, the complexity becomes:

$$\mathcal{O}(noPixels(noCurves) \cdot noThetas) \tag{3}$$

Increasing rho will also increase the time complexity as we access elements in a larger matrix:

$$accumulator(rhoIndex, thetaIndex) = accumulator(rhoIndex, thetaIndex) \tag{4}$$
$$+accumulatorIncrement;$$

Question 10: A natural choice of accumulator increment is letting the increment always be equal to one. Alternatively, let the accumulator increment be proportional to a monotonically increasing function of the gradient magnitude. How do you propose to do this? Try out a function that you would suggest and see if it improves the results. Does it?

We propose to use the logarithm of the gradient as the voting factor for each point in the accumulator space. In this approach stronger edges gain multiple lines, while weaker ones as well ass falls edges lose their lines.

Another approach could be to use the gradient direction which has already been computed when we compute L_v since $L_v = (L_x, L_y)|_{(x_0,y_0)}$ we know that the θ angle corresponds to the angle to the gradients direction. Thus, we do not need to iterate over all possible theta values in the function houghline. Instead we could compute the ρ value using the gradient directions angle, i.e.:

$$\theta = arctan(L_y/L_x) \tag{5}$$

However, problems arise when the gradient angle is uncertain. For instance, imagine that all the points of a line do not have gradients that perfectly point in the same direction. This will thus result in different theta values.