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

Name: Arturs Kurzemnieks Program: Computer Science

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:

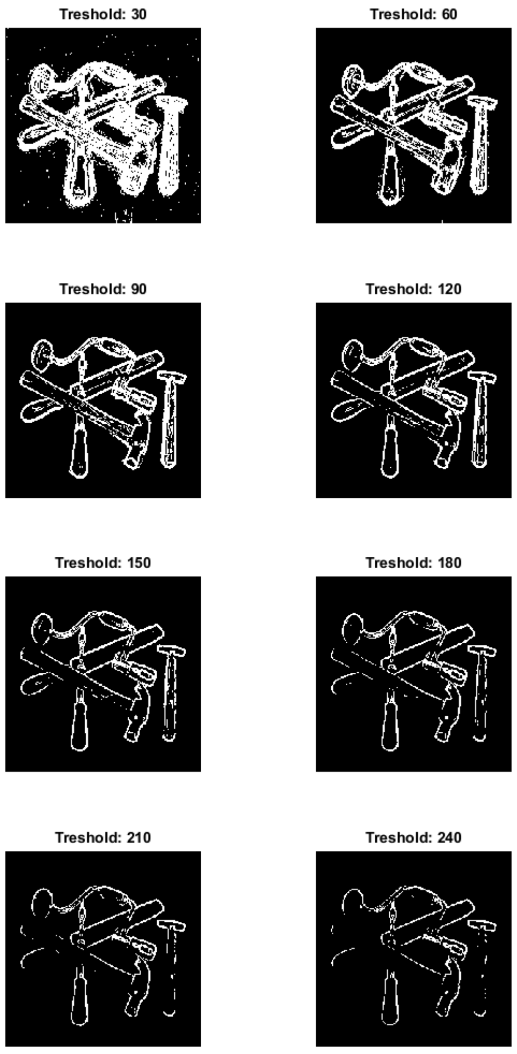
The task of the difference operators is to calculate first order partial derivatives. For this is chose Sobel operator, respectively for x and y direction. We treat the image as a function, i.e. in places where the pixel values change the fastest, the derivative is the highest. Naturally, this happens on the edges in the image.
If we take a look at, for example, partial derivative with respect to *x* (image as *f(x,y)*), when we walk horizontally from left to right and, coming from the lighter background, encounter the edge of the darker instrument handle, the pixel intensity values drop here. Similarly, when we walk off on the right side from the darker instrument handle onto the lighter background again, the pixel intensity values jump up. Now, as the filter kernel is flipped when doing the convolution, I made the x and y kernels preflipped so that the filter response corresponds to the value changes, i.e. so that the *left-side* edges of the objects appear blacker (as pixel values drop there) and *right-side* values appear whiter in *dxtools*. Similarly, in *dytools* the *top-side* edges are blacker and *bottom-side* are whiter. Also, naturally, the more orthogonal the edge is to the direction we're looking at, the more accented edge is produced in the filter response.

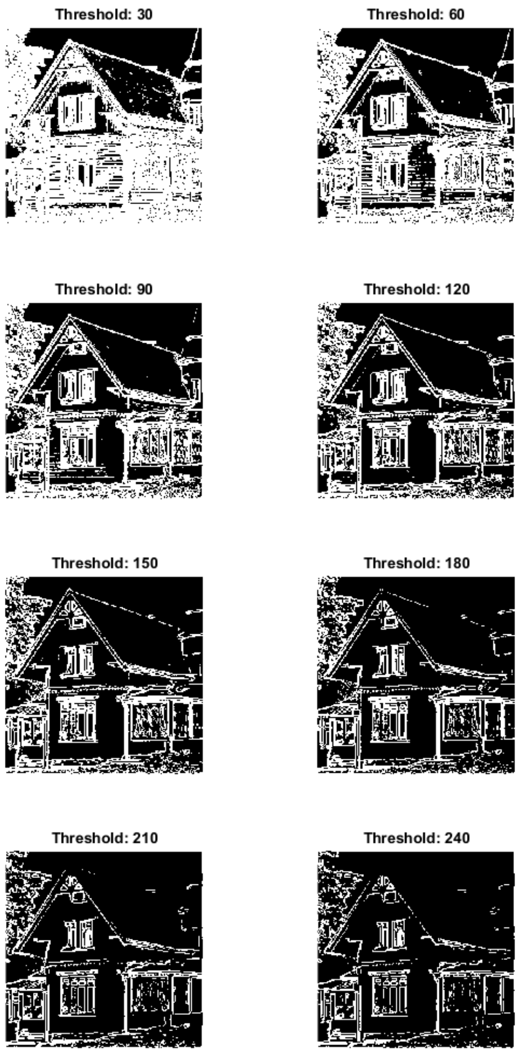
The filtered version of the image is slightly smaller, respectively 254x254 pixels compared to the original 256x256.
We perform the convolution with 'valid' shape argument, which is described to "return only parts of the convolution that are computed without zero-padded edges". As I used a Sobel operator with a 3x3 kernel, it means we can't place it on the edge pixels without zero padding on the outer side, therefore we lose one pixel on each side of the image as we only use the pixels on which the kernel can be placed on fully without using any extra zero-padding.

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

Answers:

Not particularly. The lower the threshold, the wider the edges are. The more we increase the threshold, the more we leave only the peaks of the curves that describe the edge points, resulting in thinner edges. Noise and a lot of small fine features are present in the images, which have lower magnitudes than the bigger, better defined structures, so an increased threshold also starts to filter these out, so technically we're getting more of the main features we're probably interested in. Unfortunately, the magnitudes are quite varying on the main edges, so eventually we start losing the weaker parts of the edges, resulting in discontinuities. Due to this we really have to look at each image separately to find the right threshold where we still have the main edges unbroken, while trying to minimize the noise.

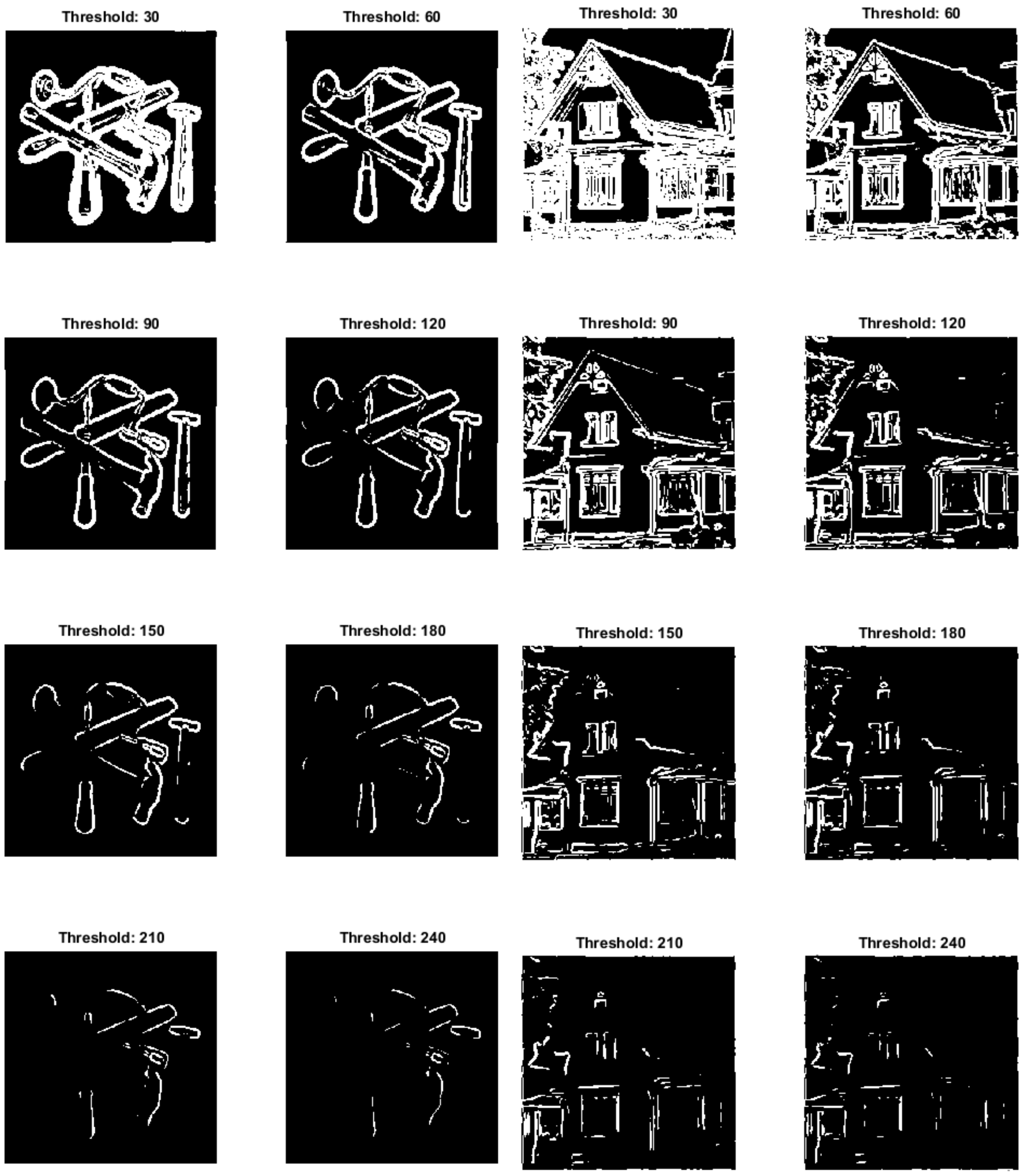




Question 3: Does smoothing the image help to find edges?

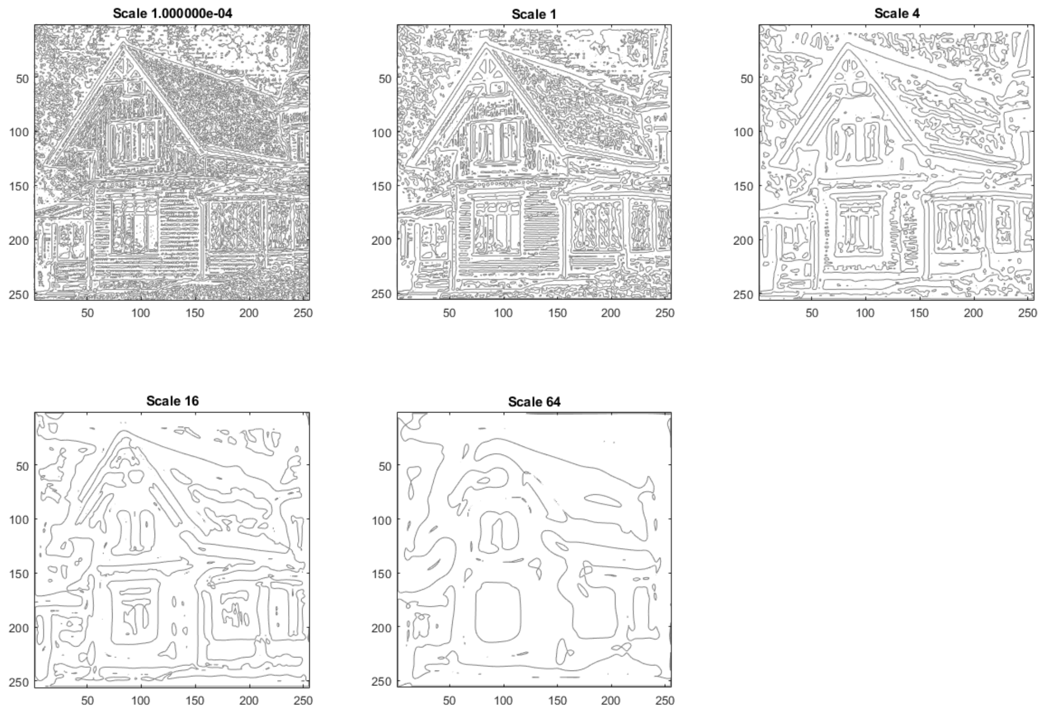
Answers:

Yes, it filters out noise and all the smaller, finer details in the image, while leaving the bigger features more or less intact. While it helps getting rid of the smaller features and a lot of false edges, and allows to use a lower threshold, it must be noted that the resulting edges for the bigger features are thicker, as the pixels are blurred out and the transitions are less sharp, and therefore might be less precise and distorted. As we're blurring, we also flatten out the magnitudes and get discontinuities on lower thresholds than for the non-filtered versions, but since the smaller features are blurred out, we can just use a lower threshold all together. It is definitely a good choice to pre-smooth the image if we're interested in only the main structures in the image.



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

Answers:



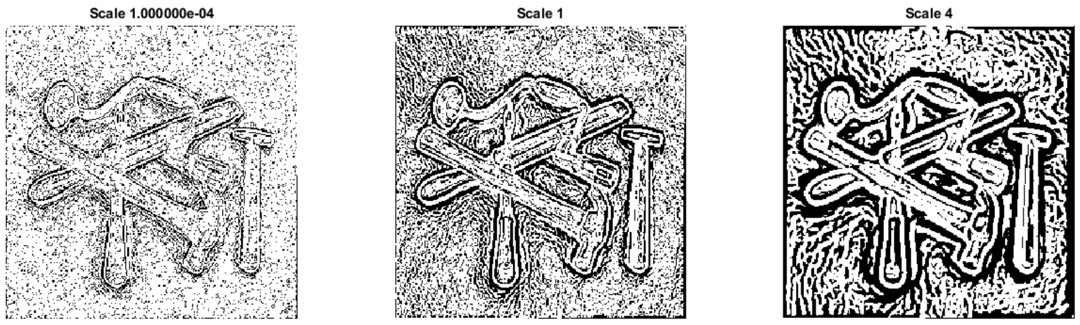
For this image we generate the second order derivative (technically not the full derivative, only the numerator part) at each point with respect to the gradient direction. If we look at the second order derivative near an edge point, we get a peak followed by a valley (since we get a peak in the first order derivative), so the middle point of the edge is represented at the zero-crossing point in the second order derivative, where the value comes down from the peak to proceed down to the valley. With this, we can retrieve an exact midpoints, resulting in thin well-defined edges, something which wasn't easy to do with first order derivatives, where maximum points of the peaks had different magnitudes and we could only filter out the bottom values by thresholding.

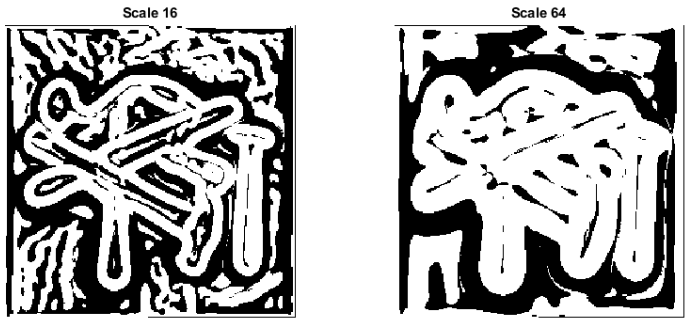
To obtain these images, Matlab's *contour* function is used to plot isolines between different value regions at zero-level, therefore the isolines follow the points where the second order derivative is at zero-crossing, which happens to be the edges we're interested in.

This is done at different scales, corresponding to different levels of Gaussian blur. For the larger scale images smaller features and noise are blurred out, therefore most of the local maximum and minimum points and the respective zero-crossings they produce at lower scale levels are removed. Unfortunately, the larger structures are blurred out as well, so sharp edges and straight lines can be heavily deformed, resulting in a rounded, inflated look for the final edges.

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

Answers:





For these pictures we compute the third order derivative at each point with respect to the gradient direction (again the numerator only, not full expression). The white areas correspond to the points where the derivative is negative, which in turn correspond to the drops in the second order derivative. These are the effect of the sign condition in the expression we use to render the images, i.e., we render white pixels where the third order derivative is negative, black otherwise.
Again, the blurring plays a big role here. The more we blur, the wider edges we get, as we're again using thresholding like for the first order derivative and the curves get wider. Blurring also introduces deformations in the main structures, and blurs out some noise, which gets more evenly spread out, but not entirely suppressed, so we just get wider regions of noise which can get hard to distinguish from the actual objects we're interested in on higher scales. Technically, this could be countered by using a threshold lower than zero, to threshold out only stronger edges, but that might have the same caveats as the thresholding for the first order derivative, e.g. some weaker parts of the main edges starting to disappear.

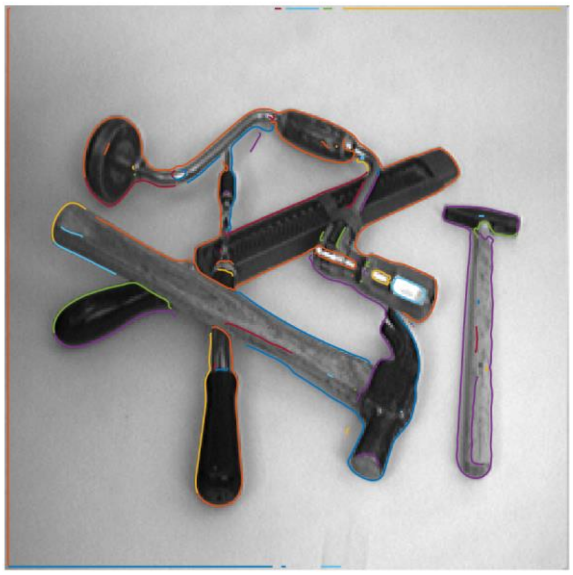
Question 6: How can you use the response from Lvv to detect edges, and how can you improve the result by using $Lvvv$?

Answers:

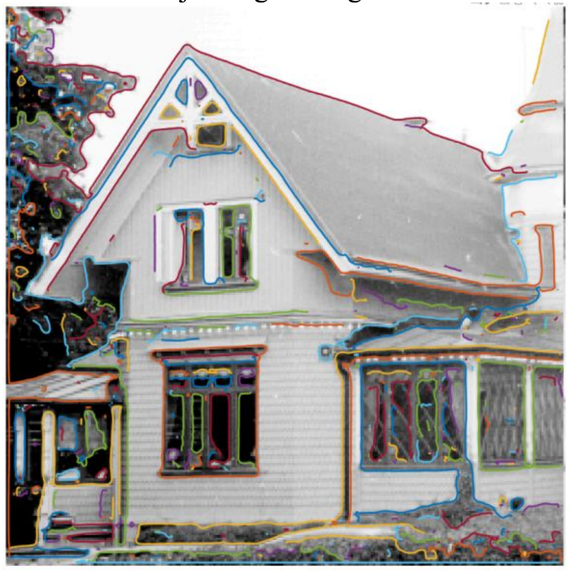
As mentioned, the zero-crossing points from the Lvv response correspond to the edge midpoints. Of course, the Lvv response can take zero value not only on edges, but on monochrome surfaces as well, where there's just no change. Therefore, if we use it together with $Lvvv < 0$, we can ensure that these points are indeed zero-crossings, as $Lvvv < 0$ is true only for regions where the value of Lvv is dropping and not monotone. Using these conditions together can ensure that we only get the points where the gradient is at a local maximum.

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

Answers:



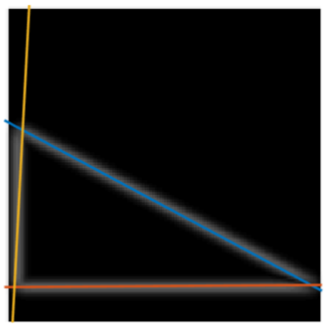
Scale = 4, threshold = 45
Needs a relatively large scale to get rid of the noise and avoid false edges on the minor features. Scale just high enough so there isn't major distortion on the tool shapes.

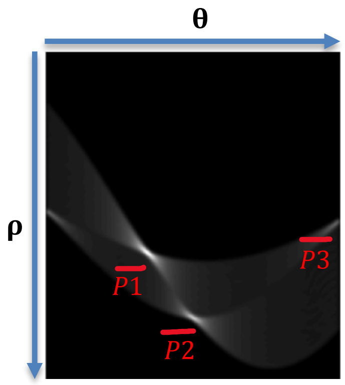


Scale = 3, threshold = 35
Finer features here, so less blur is used. Mainly focused on having the main outlines for the buildings being present, while trying to minimize noise induced edges.

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 in one or more figures.

Answers:

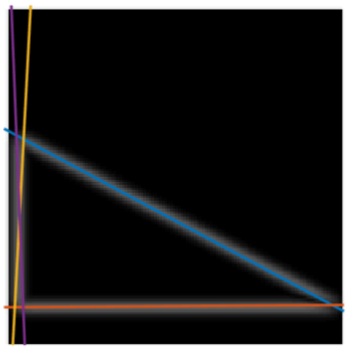




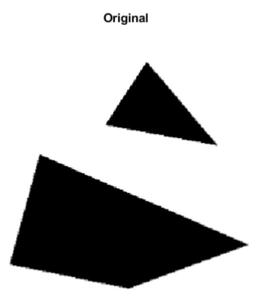
For the simple triangle test image three lines are correctly found, which correspond to the three visible hotspots in the (smoothed) Hough space representation. The origin (0,0) is located in the center of the image.
If we look at the parameters for these three lines

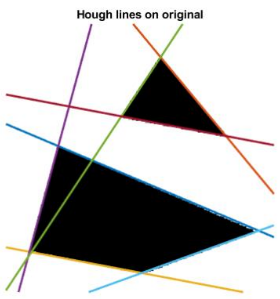
	ρ	**θ**
P1	41.2851	-0.4857
P2	113.8743	0.0044
P3	9.5273	1.5183

we can most easily see that the middle point P2 (ρ = 113.87, θ = 0.0044) corresponds to the horizontal orange line. The angle is almost zero against the x axis and it's the farthest away from the origin (the image is 128 pixels in height, so logically this line shows up almost at the bottom of it). In the Hough space it can be seen as the bottom-most hotspot. The space is defined with θ ∈ $\left[-\frac{\pi}{2}, \frac{pi}{2}\right]$ on the horizontal axis and ρ ∈ $[-d, d]$, where d is the diagonal length of the image (the farthest a line can be from the origin to still be visible in the image), on the vertical axis. As $\theta \approx 0$, it's centered on the horizontal axis.
The third point P3 corresponds to the (almost) vertical yellow line (ρ = 9.5273, θ = 1.5183). It's close to the origin and the angle is close to 90 degrees.
This leaves the first point P1, which maps to a line with θ = -0.4857 or about -27.5 degrees, corresponding to the blue line.
It must be noted that for P3 there's an opposite quite visible peak in the Hough space with roughly the same ρ value, but θ close to $-\frac{\pi}{2}$, which should give a quite similar line.
And indeed, if we plot the 4 highest scoring lines, we get also this second duplicate line.

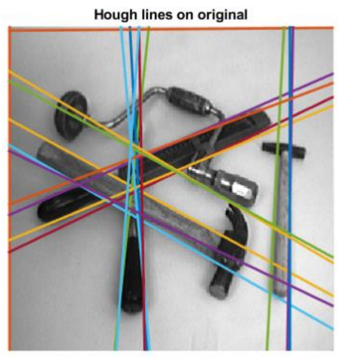


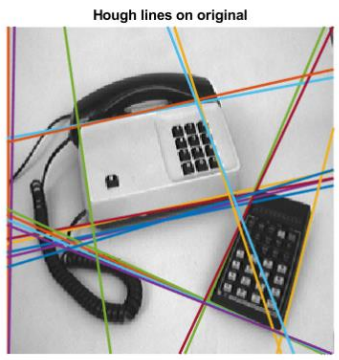
The lines produced for the second test image are accurate as well, although it takes more tweaking to find the right parameters.

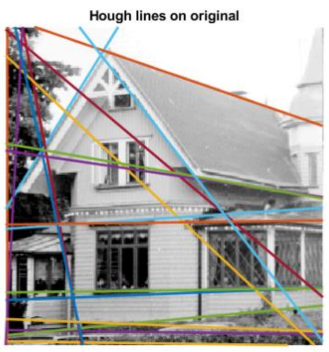




Results for the other images:



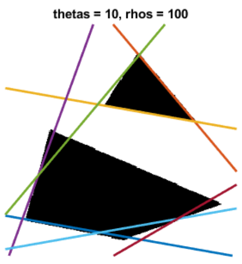


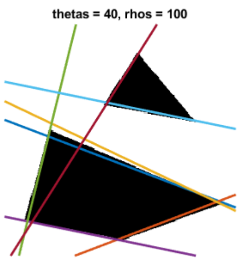


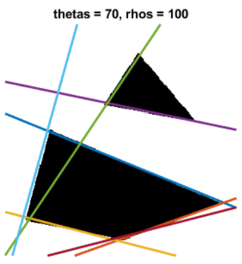
Question 9: How do the results and computational time depend on the number of cells in the accumulator?

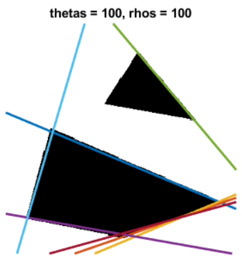
Answers:

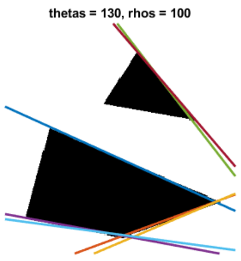
The higher the number of cells in the accumulator, the technically more precise lines we can get. In practice, the drawback is that, when the resolution of the accumulator gets higher and starts to near or exceed the resolution of the image itself, we start getting more multiple responses for each edge in the accumulator, as more neighbored cells start matching the edge.

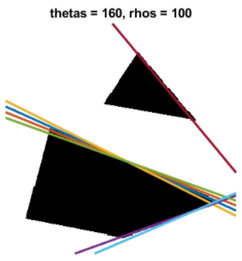




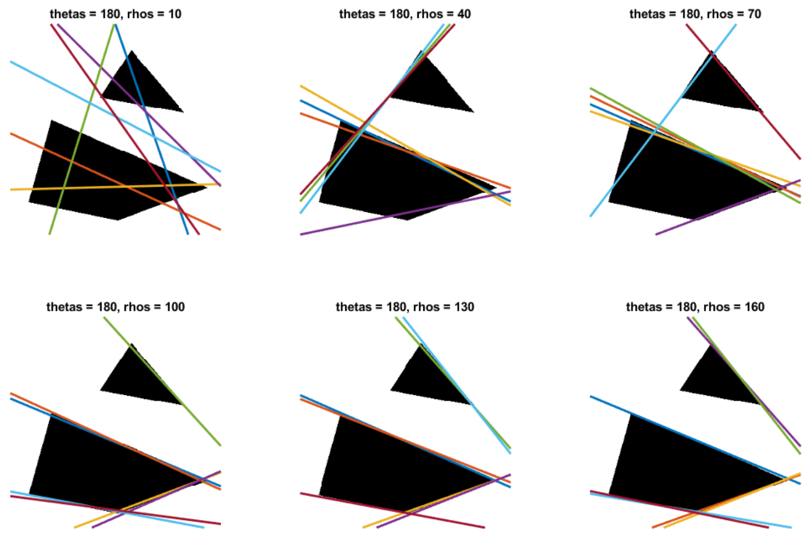




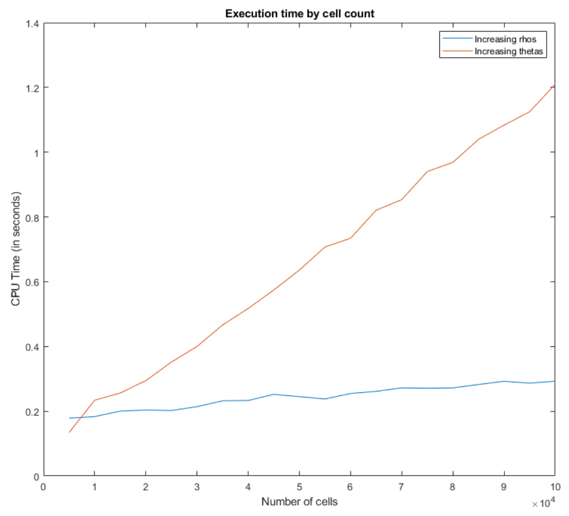




In this first experiment I used a constant number of possible ρ values, while increasing the number of possible θ values. As can be seen, in the beginning there's very little angular precision, so the lines don't fit as good, but there's also less duplication. In contrary, with a higher variety of θ, most of the top-scoring lines are just duplicates of some particular edges.



Similarly when checking the effect of different gradations for ρ, for a small value the resulting lines are matching poorly. As it increases, the more close knitted duplicate lines we're getting.

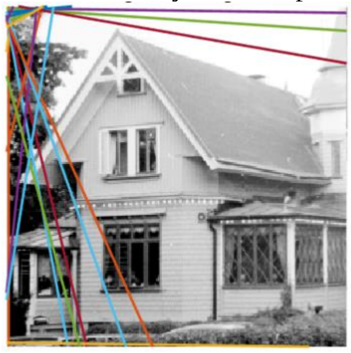


For effects on computational time I used an experiment where execution time of *houghedgeline* was measured on increasing number of cells from 500 to 100,000. This was done twice, first time increasing only the number of different possible ρ values, the second time increasing only the number of possible θ values.
As can be seen in the graph, for the same number of cells the computation time is much more sensitive to an increase in number of θ values, i.e. to a smaller $\Delta\theta$. This makes sense, because when finding suitable lines, all the θ values are looped for every edge point. Meanwhile, ρ value is computed directly and only lookup for the closest bin is made. Therefore it can be concluded that computational time mainly depends on the chosen $\Delta\theta$, not the total number of cells in the accumulator.

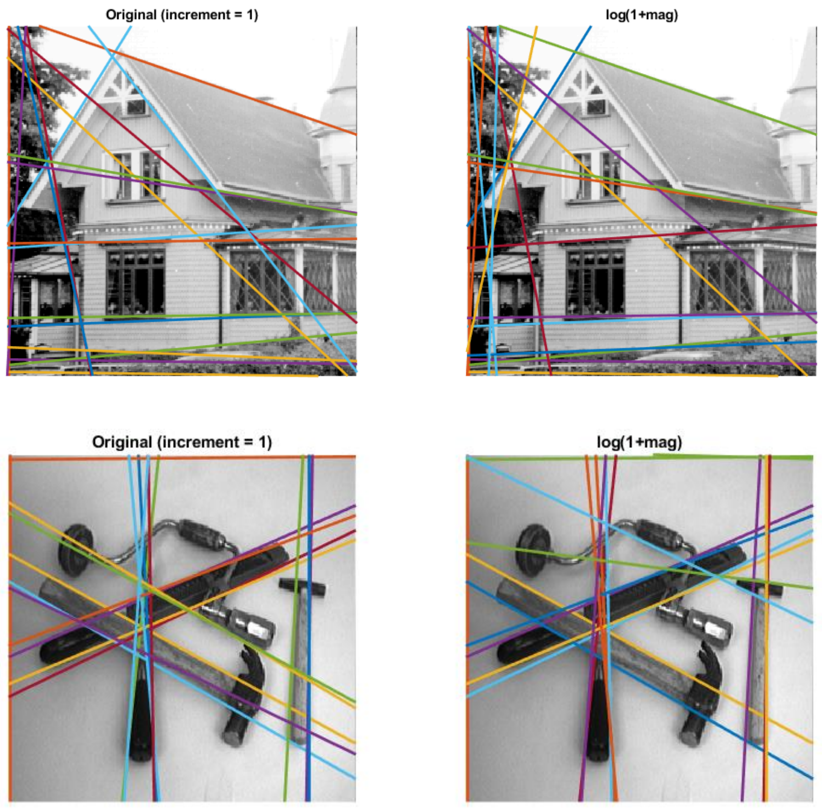
Question 10: How do you propose to do this? Try out a function that you would suggest and see if it improves the results. Does it?

Answers:

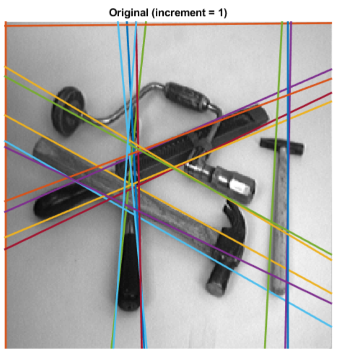
For the accumulator function, multiple choices were explored.
First approach was using $\Delta S = |\nabla L|^2$, but this yielded poor results, as it seems to enhance some few noisy extreme points, around which all the lines are then concentrated.

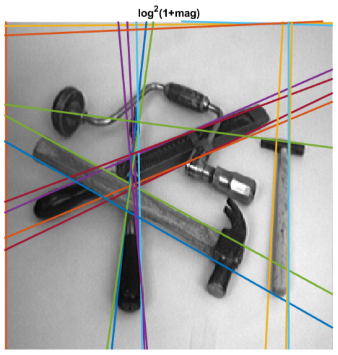


Second approach was using $\Delta S = log(1 + |\nabla L|)$. This gave more promising results, which might be usable in some specific cases, as the logarithm tends to enhance the lower values while flattening out the higher ones. In case of the test pictures, while the results were comparable to the default increment of 1, no major improvements were observed.

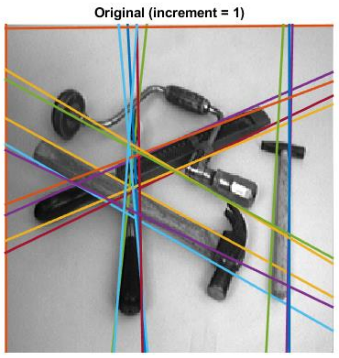


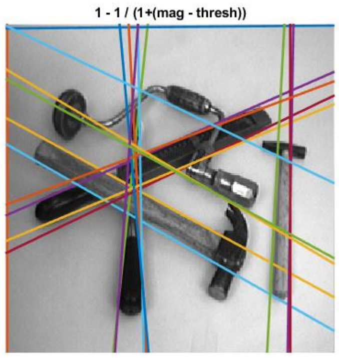
It's also possibly worth exploring $\Delta S = log^2(1 + |\nabla L|)$, which grows a bit more aggressively compared to the non-squared logarithm function. This brings out the more stronger edges, but doing so starts introducing more duplicates and noise induced lines.





The general intuition seems to be to try to bring out the "middle of the pack" values a bit more and more or less flattening the higher magnitudes, as they're going to be picked up anyway. This led me to try $\Delta S = 1 - \frac{1}{1+(|\nabla L|-threshold)}$, which flattens out faster than the logarithm based function.





In the case of the tools picture, this seemed to give a minor improvement, not losing any of the desired lines the constant version had, while adding at least one new desired line.

As for using the gradient direction, this might be used to increase the weighting of the line based on the orthogonality of the local gradient against the line at some point *(x,y)*, i.e. the closer the angle between the line and the gradient is to 90 degrees, the more weight it should have, as technically the line should match the edge better at that point.