Answers to questions in

Lab 2: Edge detection & Hough transform

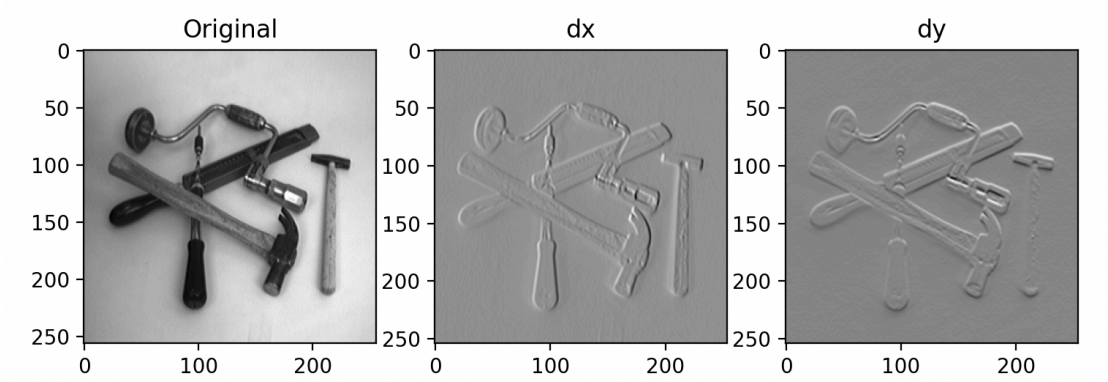
Name: _____ Program: _____

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?

I used the Sobel operator in order to computer the image δx and δy.

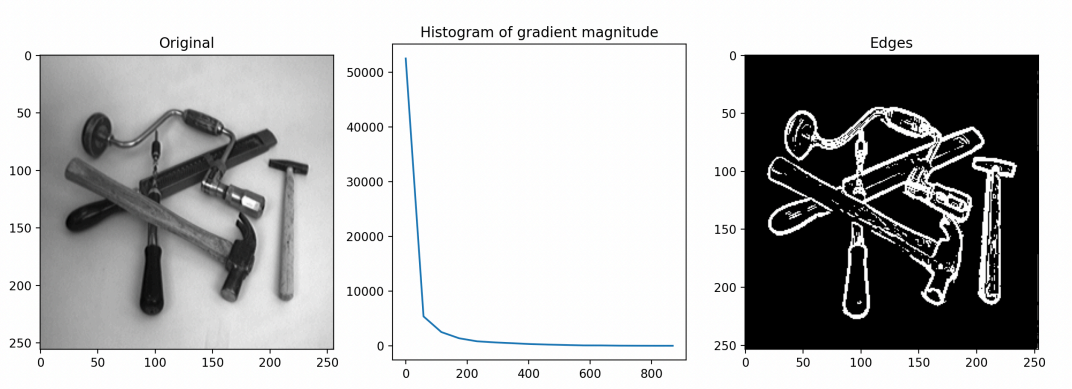


Answers:

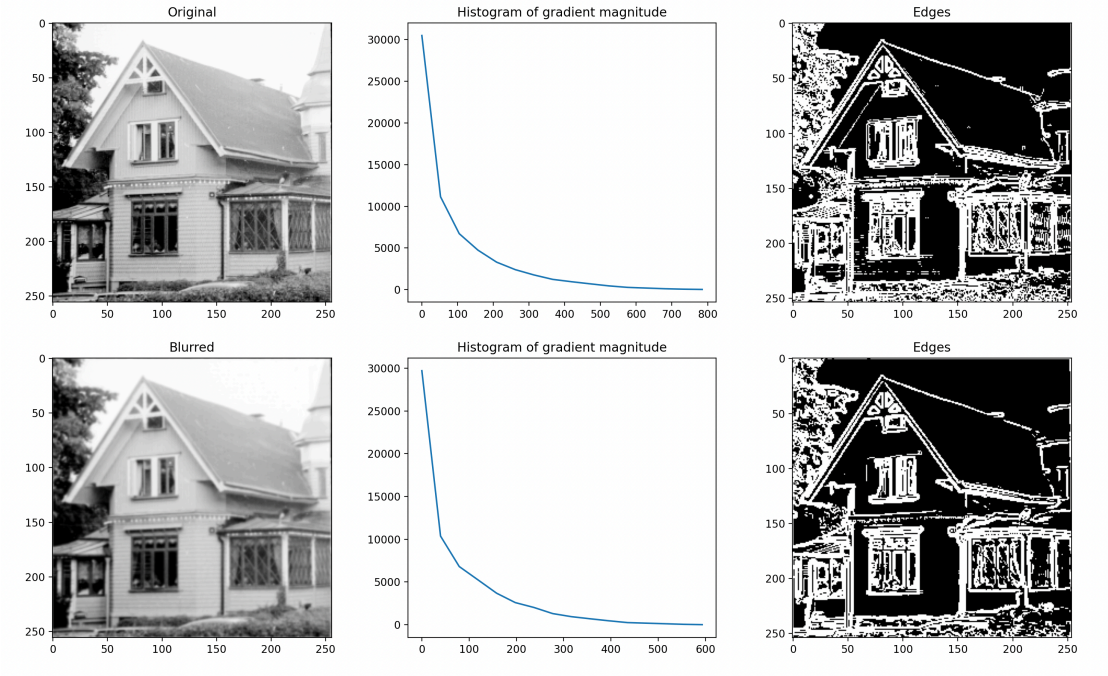
The image represent the edges in the X or Y direction depending on the direction of the Sobel filter used. The images are smaller because you cannot slide the filter past the edges of the image and thus a size reduction occurs. You can avoid this by padding the image with either the last pixel of the image or zero values but each methods have their pros and cons.

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

For the first image I used a 16 bin histogram the determine the best threshold. In the end I picked a threshold of 100 and got the result displayed on the right.



For the second image, I also used 16 bins but I applied a gaussian blur with a variance of 0.5 first. The final threshold for this image happens to also be of 100.

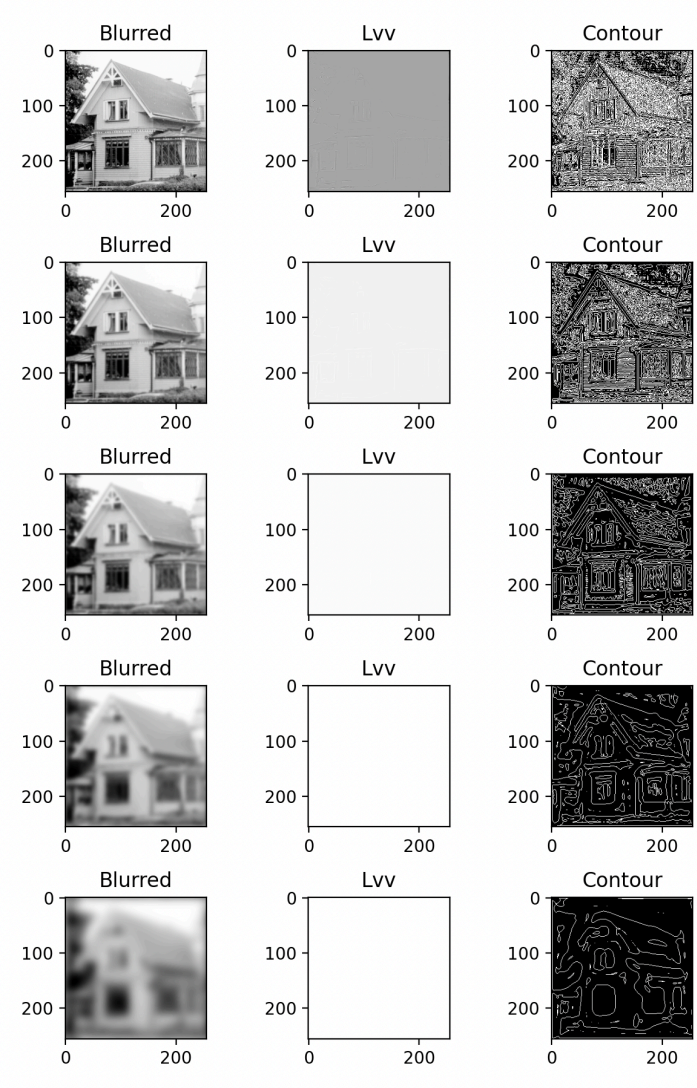


Answers:
It is pretty hard to find a good threshold by only looking at the histogram, a good starting point is to pick a point that is right after the hump where most of the low value pixels are located (the black pixels that represent a low gradient and thus not edges). This way you only keep the part of the image that have a high gradient and are most likely caused by edges in the image. The problem with this method is that a if some noise of a certain gradient level exist and you want to filter it out, you increase your threshold. Doing so on the other hand will also remove all the edges that were at that same gradient level and thus there is no way to pick what you want to keep explicitly.

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

Answers:
Blurring the image does help. This is due to the blurring getting rid of the noise. Thus, the edges in the output image are only coming from the real edges and no longer from the noise of the image (eg. We are now getting edge that look more like the real boundaries in the image and not from the noise anymore).

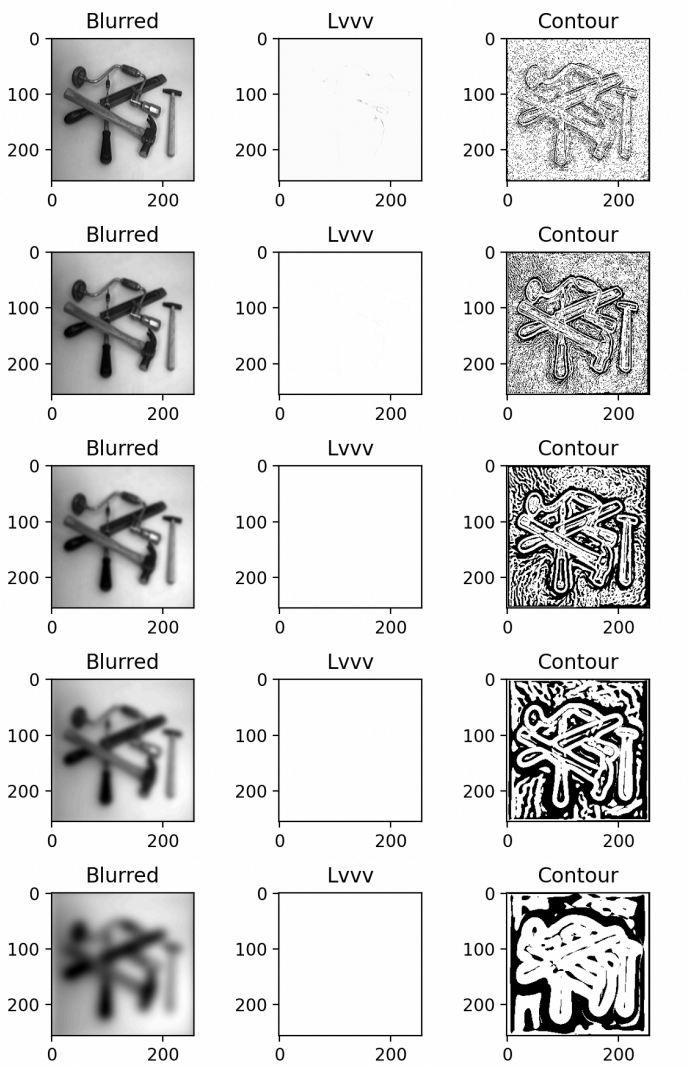
Question 4: What can you observe? Provide explanation based on the generated images.



Answers:
The edges that are detected depend on the scale of the input image (based on the blur applied). The less blur, the more high frequency pattern can be picked up (eg. The wooden

board that make up the house). Vice versa for higher amount of blur the edges that are picked up are very strong singular edges (eg. The rooftop and the general shape of the house).

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



Answers:
The third derivative indicates to us that there is an edge starting when it's value is negative.

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

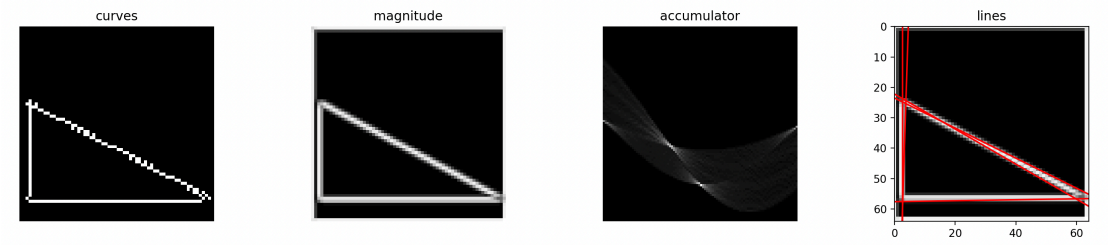
Answers:
Lv represents the gradient of the image, a local maxima in Lv represents an edge in the image. To find edges you need to detect the maxima on the first derivative of the image (eg. Lv). In order to do so, you can check when the second derivative is equal to zero. This is not sufficient to know wether you have a local maxima or a local minima. In order to only detect local maxima, you need to add an extra condition. Namely, that the third derivative is less

than zero at that point. If you meet all these conditions, you can be certain that you have found a local maxima of the first derivative.

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

Answers:

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:
Looking at the accumulator space, I found that the highest value is at pixel x=50 y=81. Knowing that the x axis represents ρ and the y axis θ we can convert back to the correct value of ρ and θ using the mapping defined by our linspace. Asking our program to print out the position of each maximum we get that our highest value converts back to this:

```
Pos: (81, 50), rho: 57.597, theta: 0.016
```

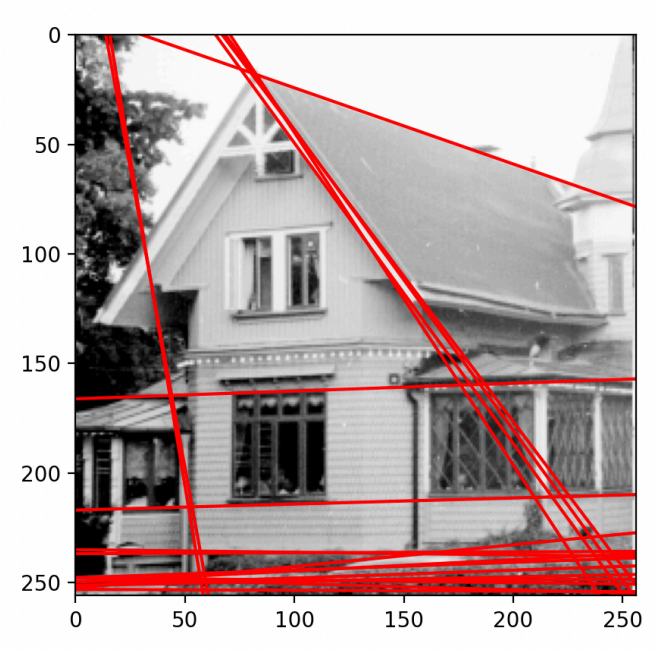
Even without going trough any math, we can see that this corresponds to a very low steep line located at ~58 pixels from the top of the image (the fact that it is almost horizontal allows us to know that it will indeed be at ~58 pixels from the origin since its middle point will be on the left of the image, same as the origin) and this corresponds to the line forming the base of the triangle. This shows that the algorithm is running properly.

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

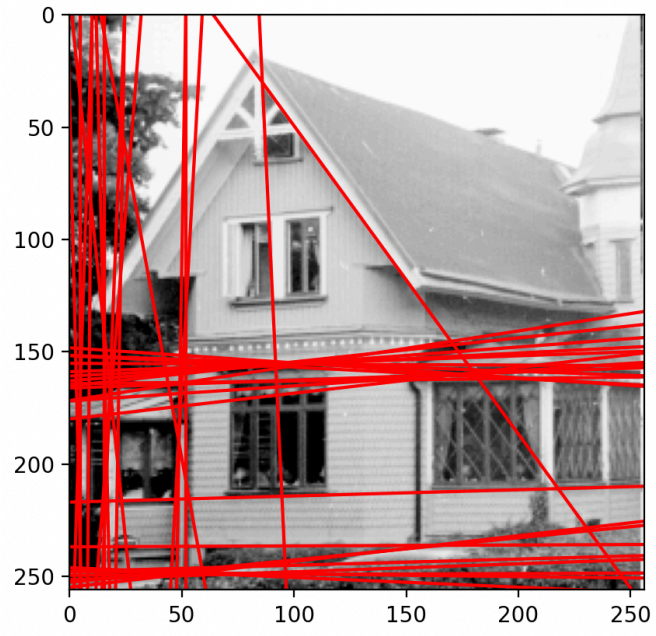
Answers:
If the amount of cells in the accumulator is too low, the precision is greatly reduced. On the other hand, if it is too high, the value in the accumulator are not very varied since their is little overlap of line and thus a peak detector does not help finding the maxima. The higher the amount of cells, the longer the computation time. The relation between both is pretty complex since it involves the number of pixel considered as curve, the number of theta cells, running a filter across the entire hough space to get local maximas and then sorting $\frac{N^2}{9}$ values. Suffice it to say that it is fairly slow for a large 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:
By only incrementing the accumulator by one we get this:



Taking the simplest function (adding the gradient magnitude instead of 1) we get something like this:



This is not really better but it is picking up on different edges in the image that the previous function was not picking up.
