DD2423 - Image Analysis and Computer Vision
Lab 2

Gabriela Zarzar Gandler
gzrsm@kth.se

Huijie Wang
huijiew@kth.se

November 2016

1 Difference operators

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?**

   The result is shown as Figure 1:

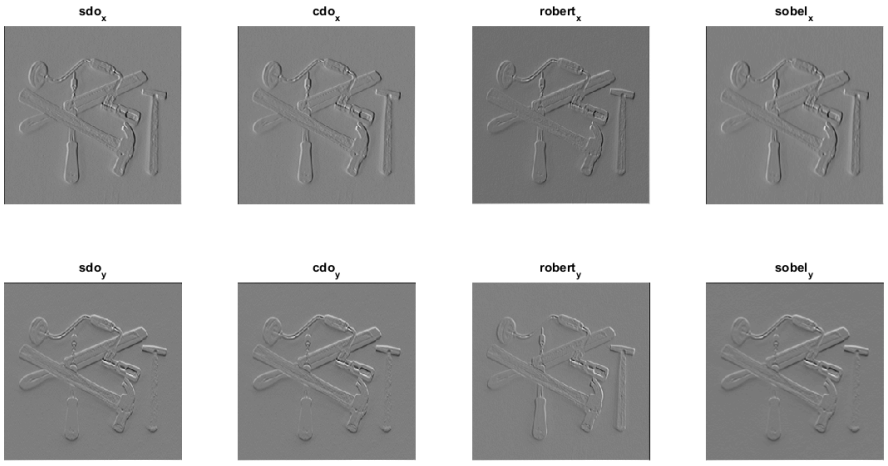

Figure 1: Discrete derivations using difference operators

We expect the result to highlight places where there are sharp transitions - such as edges - in the x direction and in the y direction. For the Robert operator, particularly, we expect these transitions to be highlighted in the diagonals (45 degrees). These expectations are justified by the filters, which represent derivatives. The size of the image is 256. After applying different filters, the size of **dxtools** turns out to be 256×254 for simple and central difference operator, 255×255 for Robert's diagonal operator and 254×254 for Sobel opeartor. That is because we use 'valid' as parameter when applying convolution in Matlab, which doesn't add padding to initial image. So that the size after convolution should be $size_x - (size\_filter_x - 1) \times size_y - (size\_filter_y - 1)$.

2 Point–wise thresholding of gradient magnitudes

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

   No, it is not easy to find a threshold that results in thin edges, because we are using first-derivatives. This is the problem of solving edge detection by thresholding on the edge strength (gradient magnitude): the resulting edges are often several pixels wide. The first-derivative is usually a ramp-like transition and thresholding may remove too much of certain edges and too little of others.

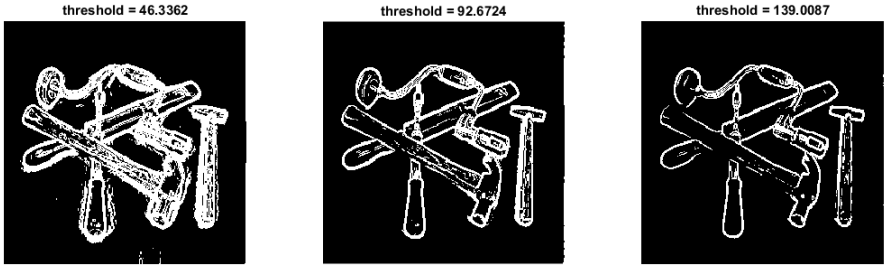

Figure 2: Point–wise thresholding of gradient magnitudes

3. **Does smoothing the image help to find edges?**

   Smoothing reduces the noise, which is very good. The gradient of the original image is much affected by noises. So smoothing the image makes it possible to "trust" more on the sharp transitions that the gradient detects. However there is a trade-off problem: if the image is smoothed too much, the actual edges are not detected, while if the image is not smoothed enough, the edges are still highlighted, but there is also a lot of "false positives" (noise, for example). Smoothing the image also makes the edges thicker, because the intensity "ramps" get longer.

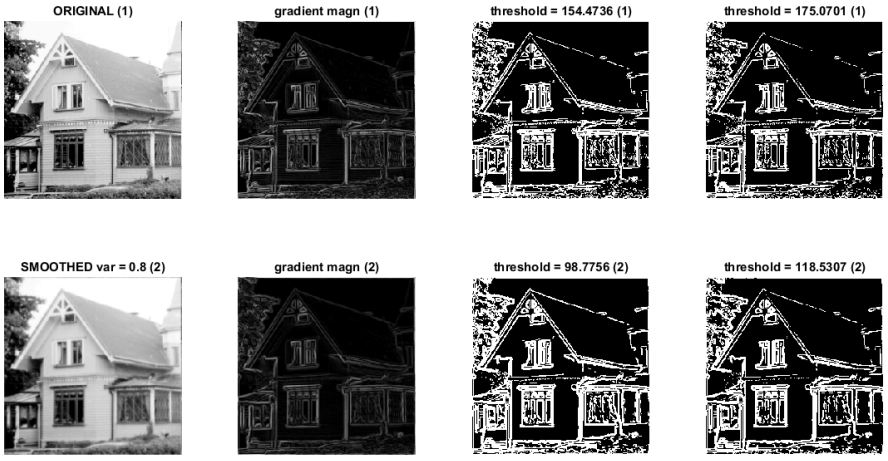

Figure 3: Effect of smoothing

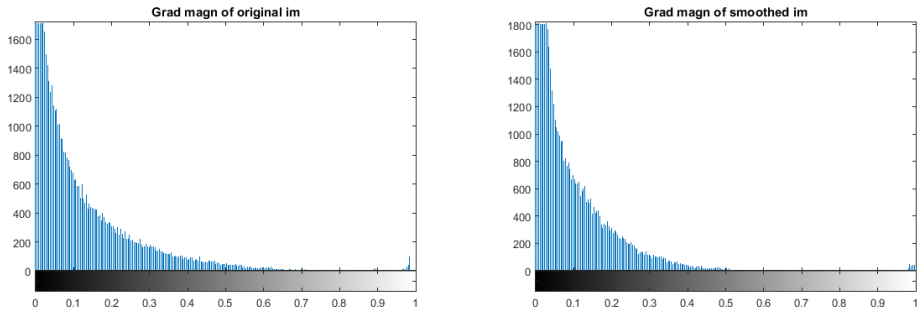

Figure 4: Histogram of gradient magnitude before and after smoothing

3 Differential geometry based edge detection: Theory

4 Computing differential geometry descriptors

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

   We can observe in Figure 5 that, the larger the scale (variance of the Gaussian filter), the less contour there is. This means that the noises get suppressed - the first image is corrupted by so much contour, due to inconvenient non-smoothed variations in the intensity level. However if one increases the scale too much, there is a loss of valuable information, as the edges get distorted - they tend to get curved, especially on areas close to corners.

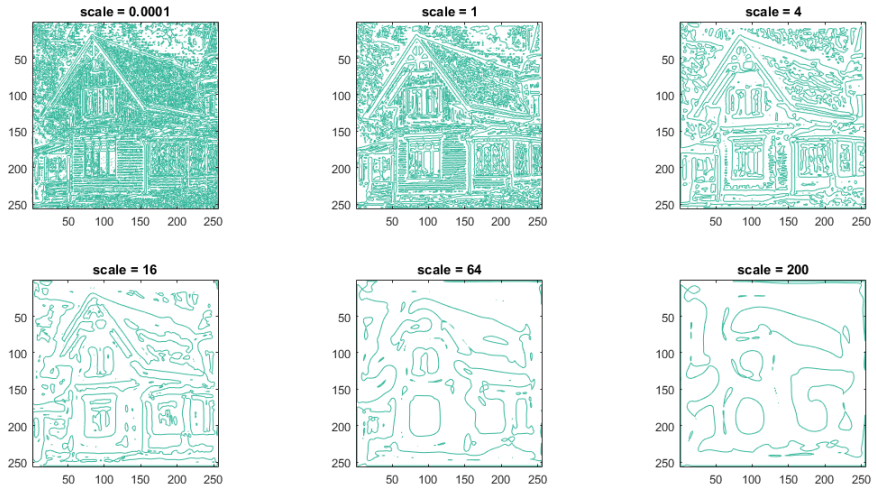

Figure 5: Contour of zero-valued second derivative

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

   We conclude that the more intense the smoothing is, the less varying the sign of the third derivative is. This makes the areas with negative third derivative thicker. The white areas

are followed by a black "contour" on the outside, due to the shape of the third derivative in the presence of an edge.

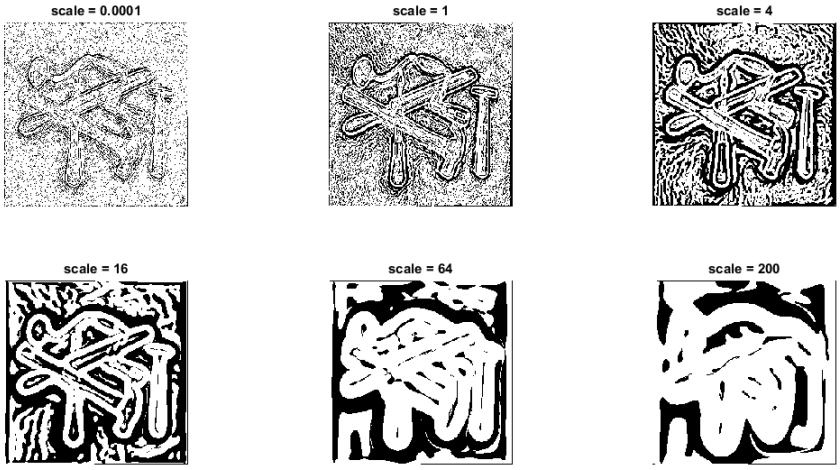

Figure 6: Highlighting negative third derivative

6. **How can you use the response from $\tilde{L}_{vv}$ to detect edges, and how can you improve the result by using $\tilde{L}_{vvv}$?**

   Firstly we look to the L_{vv} representation of the image. Then we consider only the points of this representation where the third derivative is negative. Finally we plot the contour corresponding to zero-valued pixels.

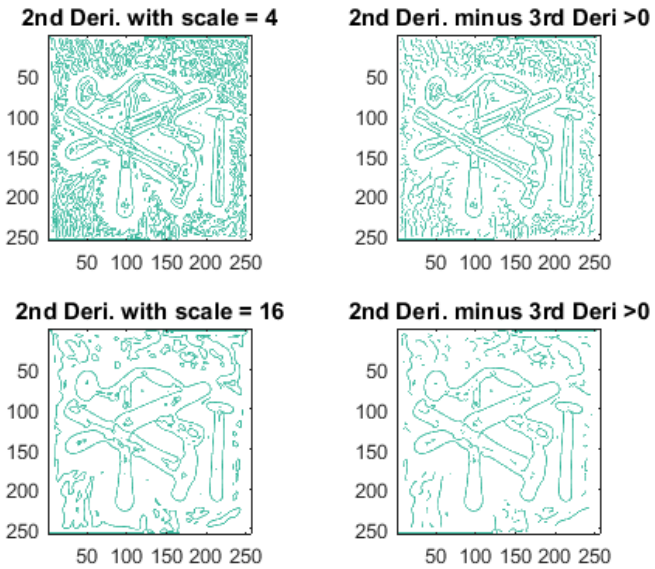

Figure 7: Improving $\tilde{L}_{vv}$ with $\tilde{L}_{vvv}$

5 Extraction of edge segments

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

   In Figure 8, we can find $scale = 4$ and $threshold = 8$ is a decent value for *Tools* and in Figure 9, $scale = 4$ and $threshold = 4$ is a decent value for *House*.

   Through these images, we can find that a large scale gives curved edges and a too small scale would do poor in eliminating noises.

Figure 8: Extraction of edge segments on *Tools*

Figure 9: Extraction of edge segments on *House*

6 Hough transform

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 results are shown as Figure 10 and Figure 11. In simple shapes, it is obvious that the detected lines correspond with the maximum points in Hough space.

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

   The more cells in the accumulator end up with more computational time. That is because in Hough Transform we use loops to go through every θ for every edge point.

   On the other hand. It is important to find appropriate number of $n\rho$ and $n\theta$. Too large number, especially for $n\rho$, will lead to multiple response to a single line. On the other hand, too small number will lead to low accuracy.

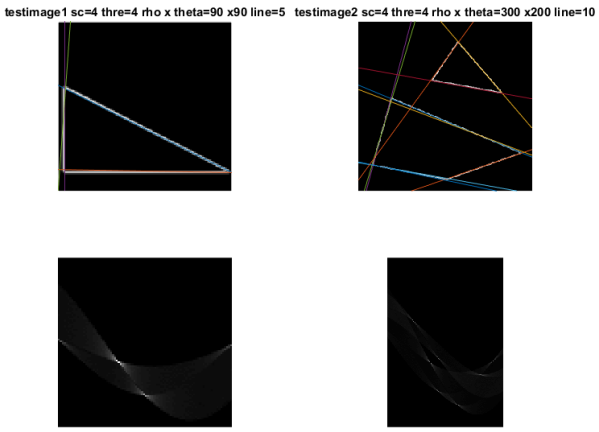

Figure 10: Hough transform on simple images

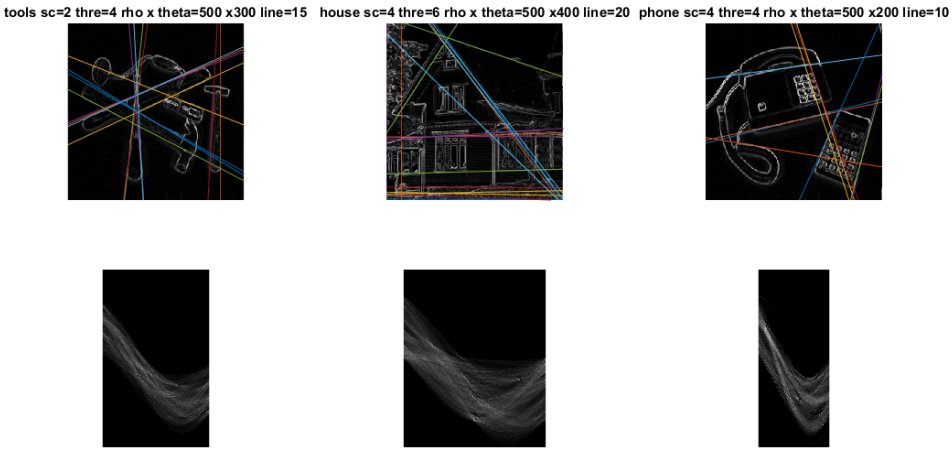

Figure 11: Hough transform on complex images

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

    As a larger value of magnitude is more likely to be an edge pixel, using magnitude value or a monotonically increasing function of magnitude can accelerate the increment in Hough space. We tried 4 ways:
    1. $increment = 1$.
    2. $increment = magnitude\ of\ the\ pixel$.
    3. $increment = log(1 + magnitude\ of\ the\ pixel)$.
    4. $increment = \sqrt{magnitude\ \ of\ \ the\ \ pixel}$.

    As a result, seems like the first and third work fine.