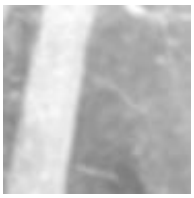
Exercise 7: Edge detection and non-maximum suppression

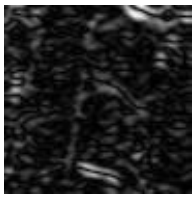
M.Thaler/J. Rosset, 11/2023, ZHAW

1 Introduction

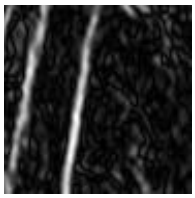
When looking for edges within an image, in a first step often the magnitude of the gradient is computed, which is then thresholded to generate a binary image. This procedure is shown by the following sequence of images:



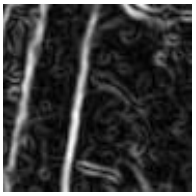
a) original



b) Gx



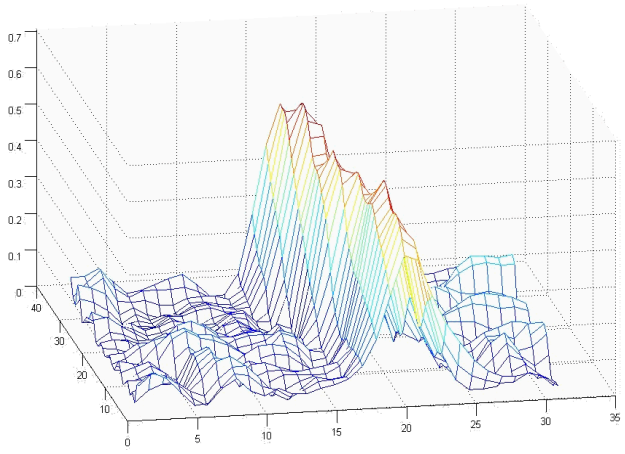
c) Gy



d) magnitude of G

e) thresholded magnitude

One of the problems is that this procedure yields edges that are several pixels wide and thus do not represent the exact location of these edges. A detailed analysis of the magnitude image shows that edges are a kind of *mountain ridges* as is illustrated by the following 3-d image:



The direction of the gradient is perpendicular with respect to the edge and has its maximum in the middle of the edge. Choosing only pixels along the path with maximum gradient magnitude yields a one pixel wide edge (the ridge pixels).

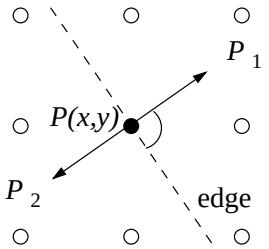
2 Non-maximum Suppression

To find the exact edge, pixels on the *magnitude ridge* must be selected as edge pixels, all other pixels must be removed (set to 0). This procedure is called *non-maximum suppression*. In the following we show a simple version to find the ridge pixels.

2.1 Approach

For each pixel in the image:

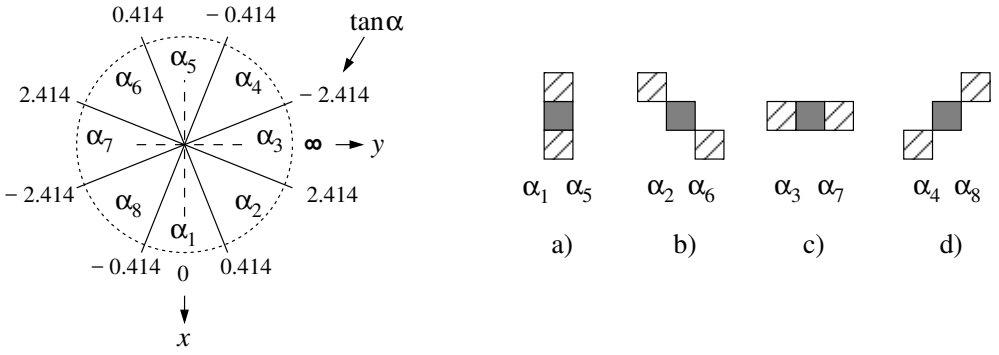
1. Find two pixels P_1 and P_2 in the 8-neighborhood of the actual pixel $P(x, y)$ such that they are located in the direction of the gradient (perpendicular to the edge):



2. Compute the magnitude G_1 and G_2 of the gradient at pixel position P_1 and P_2.

3. If the magnitude of the gradient at pixel P_1 or P_2 is larger than the magnitude of the gradient at the actual pixel $P(x, y)$, pixel $P(x, y)$ is not a ridge pixel and must be set to 0 in the resulting image, otherwise the magnitude value $G(x, y)$ is copied into the resulting image.

The gradient at pixel P_1 and P_2 can be computed in several ways, e.g. by linear interpolation or through appropriate neighborhoods. In the last case only the direction of the gradient has to be determined. For an 8-neighborhood the following angle regions must be taken into account (the shaded neighbors correspond to pixels P_1 and P_2):



The tangent of angle α can be computed from the gradient at pixel $P(x, y)$ as follows:

$$\tan\alpha = \frac{G_y(x, y)}{G_x(x, y)}$$

keep in mind that G_x might be zero.
Based on the value of α the appropriate neighborhood (a) - d)) is then used to find the location of pixels P_1 and P_2 (Step 2) where the magnitude can be selected to compute step 3.

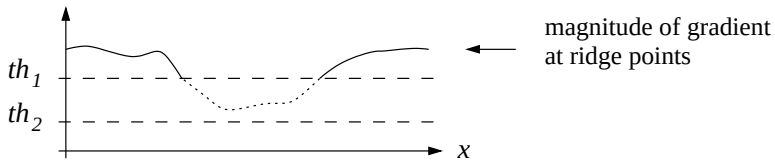
2.2 Tasks

Given the image `arterie.jpg`. Find the thinned edges of this image with the help of the non-maximum suppression algorithm. Proceed as follows:

1. Read the image with Matlab, cast it to double and scale it to range $\{0, 1\}$.
2. Smooth the image with a Gaussian low pass filter of size 7x7 and $\sigma = 4.0$ to reduce noise.
3. Find the gradient components G_x and G_y as well as the magnitude of the gradient G based on the sobel operators:
$$G = \sqrt{G_x^2 + G_y^2} \text{ resp. } G = |G_x| + |G_y|$$
4. Find the gradient image with the help of non-maximum suppression, compare the gradient magnitude image: with and without non-maximum suppression.

3 Post-processing: hysteresis thresholding

Non-maximum suppression yields images with intensity values proportional to the strength of the gradient at ridge points. A binary image can be obtained by applying hysteresis thresholding to the non-maximum suppressed image. First all ridge points that have magnitude $G(x, y) \geq th_1$ are added to the binary image. Then all points which are neighbours of already chosen points and whose magnitude is $G(x, y) \geq th_2$ are added to the binary image: iterating this step adds the doted points.



3.1 Tasks for hysteresis thresholding

Start from the non-maximum suppressed image (non-zero magnitude of gradient at ridge locations).

1. Scale this image to range $\{0, 1\}$, then threshold the image with $th_1 = 0.3$.
2. From upper left to lower right find for each pixel of the thresholded image pixels within it's neighborhood whose intensity level exceeds threshold $th_2 = 0.01$ ($th_2 < th_1$). Use as neighborhood pixels $P(x + 1, y)$, $P(x + 1, y + 1)$, $P(x, y + 1)$ and $P(x + 1, y - 1)$. Choose an appropriate value for th_2.
3. The same as above, but now from lower right to upper left and Use as as neighborhood pixels $P(x, y - 1)$, $P(x - 1, y - 1)$, $P(x - 1, y)$ and $P(x - 1, y + 1)$.
4. Compare your result with the result of the Canny edge detector in Matlab (`edge(f,'canny')`). Vary the thresholds th_1 and th_2 and discuss the results. How could your algorithm be improved?

Comment: You have implemented a simplified version of a Canny edge detector. How do you assess the performance of your detector?