Computer Vision Exercises

Tutorial 1: Image Filtering and Edge Detection

26th October 2017

The exercises proposed include contents related to the image filtering in spatial and frequency domain, and the edge detection.

To perform this tutorial, use the images provided. Please select different grey scale images (e.g., high contrast, and low contrast) and apply the procedures to each.

Tutorial 1 for assessment: A solution for the proposed exercises and the respective report should be delivered until 3rd November.

➢ Design a program to smooth images

Create a script called "smoothfilters" to call a function "main_smoothfilters.m" that execute the functions outlined in "2. Algorithm". In the script, you should create the grey-scale image before running the function "main_smoothfilters.m".

1. Inputs:

- **a grey-scale image**

The filename of the input image will be given as an argument to your program. Document what command one should type to run your program.

- **type of noise**

Your program should ask to the user the type of noise to be added separately to the input image: salt-an-pepper or gaussian noise.

- **noise parameters**

Provide the required parameters to add noise to the image (e.g., variance, % occurrence)

- **filtering domain**

Your program should ask the user the desired filtering domain: spatial domain or frequency domain

- **type of smoothing**

In accordance to the selected domain, your program should ask the user whether he/she wants to use average, Gaussian and median filters or Gaussian and Butterworth filters.

- **Filter parameters**

In accordance to the selected filter, your program should ask the user to enter

- width of the filter that will be used for noise filtering.
- standard deviation (Gaussian filtering)

- filter order (Butterworth)

2. Algorithm

- **create a function to introduce noise in the image**

The noise should be added separately to the input image: salt-an-pepper or gaussian noise.

- **Smoothing using average, Gaussian or median filters in spatial domain**

Smooth the image re-using the code of the last exercise according to the parameters above. For the filtering, you can use correlation or convolution methods.
Repeat the procedure for different kernel sizes (5, 10, 20) and variances.
The report should detail the results observed for each condition, the differences between each filtering output and justify which filter is more indicated for each type of noise.
Note 1: use grayscale levels $[0, 255]$.

- **Compute the Discrete Fast Fourier Transform (DFT)**

Compute the DFT of the original, noisy and filtered images compare them. What is the difference, and why is it caused? Identify the location of the noise. If necessary, compute DFT to select the most appropriated.

- **Smoothing using Butterworth and Gaussian filters in frequency domain**

Perform the filtering only in the frequency domain. If necessary, perform an image spectrum analysis to select the most appropriated.
Repeat the procedure for different Butterworth's orders and variances.
Check the image borders before and after filtering. Analyse the presence of "ringing effect".
Compare the filtering performance in spatial and frequency domain for similar conditions. What is more efficient? What is more accurate? Use quantitative metrics for this assessment.
Compare the processing time of frequency in spatial and frequency domain for similar conditions (note: indicate the features of the used processor).

3. Output

- **noisy image**

The resulting of noise introduction. I should be saved as follows:
OriginalName_noisetype_noiseparameter.png

- **smoothed image**

The resulting smoothed image. It should be saved as follows:
OriginalName_smooth_filtertype_filterparameter.png

➢ Design a program to detect edges using Canny Detector

Create a script called "CannyDetector" to call the main function of Canny Detector "main_CannyDetector", which in turns execute the functions outlined in "2. Algorithm".

1. Input:

- **Noisy image**

Introduce the noisy image with Gaussian noise.

2. Algorithm

- "**Gaussian_smoothing.m**" (.m = Matlab® function)

Smooth the image re-using the code of the last exercise for Gaussian filter in spatial domain. This function, should allow to select the kernel size and σ. Choose a suitable σ. What is the trade-off of using bigger σ?

- "**gradient.m**"

Compute the horizontal and vertical gradients of the input image using suitable operator of Sobel's kernels. Compute the magnitude (i.e., edge strength $E_s(i,j)$) and orientation/diretion ($E_o(i,j)$) of the gradient.
For gradient direction computation, you can use the function *atan2*().

- "**nonmax.m**"

Find the best fitting orientation of the edge d_k in $d_1 \ldots . d_4$, which correspond to the angles 0º, 45º, 90º, 135º using gradient orientation ($E_o(i,j)$). Then, enhance the edges by computing an improved image (E_m) – thinned image.

$$E_m(i,j) = \begin{cases} 0 & \text{if } E_s(i,j) \text{ is smaller than at least one of its two neighbors on } d_k, \\ E_s(i,j) & \text{else.} \end{cases}$$

- **"double_threshold"**

Note: The edge pixels remaining after the non-maximum suppression step above are still marked with their strength pixel-by-pixel. Many of these will probably be true edges in the image, but some may be caused by noise or color variations. The simplest way to discern between these would be to use a threshold, so that only edges stronger than a certain value would be preserved. The Canny edge detection algorithm uses double thresholding for this step. Edge pixels stronger than the high threshold are marked as strong; edge pixels weaker than the low threshold are suppressed and edge pixels between the two thresholds are marked as weak.

Implement a function called *double_threshold* to implement the above functionality. The input to the function should be the thinned image weighted with the gradient magnitude and two thresholds. The output of the function should be the set of strong and weak edges.

- "**hysteresis_thresholding**"

Note: Strong edges can immediately be included in the final edge image. However, weak edges are included if and only if they are connected to strong edges. The idea is of course that noise and other small variations are unlikely to result in a strong edge (with proper adjustment of the threshold levels). Thus, strong edges will (almost) only be due to true edges in the original image. The weak edges can either be due to true edges or noise/color variations. Usually a weak edge pixel caused from true edges will be connected to a strong edge pixel while noise responses are unconnected. To track the edge connection blob analysis is applied. The edge pixels are divided into connected blobs using 8-connected neighborhood. Blobs containing at least one strong edge pixel are preserved, while blobs containing only weak edges are suppressed.

Implement a function called *hysteresis_thresholding* to implement blob analysis on weak and strong edges. The input to the function should be the set of strong and weak edges of the previous step. The output should be the final edge map of the original image.

3. Output

- **edge strength image BEFORE nonmax suppression**

The image that results after the canny enhancer and before the nonmax suppression should be saved as follows:

OriginalName_edge_canny_filtersize_variance.png

- **edge strength image AFTER "nonmax suppression"**

The image that results after the nonmax suppression is applied should be saved as:

OriginalName_edge_canny_nonmax_filtersize_variance.png

- **edge strength image AFTER "hysteresis thresholding"**

The image that results after the hysteresis thresholding is applied should be saved as.

OriginalName_edge_canny_hysteresis_filtersize_variance.png