Assignment2

Jingyuan Wang

Question1

Codes and comments in file: <u>'Question1.m'</u>

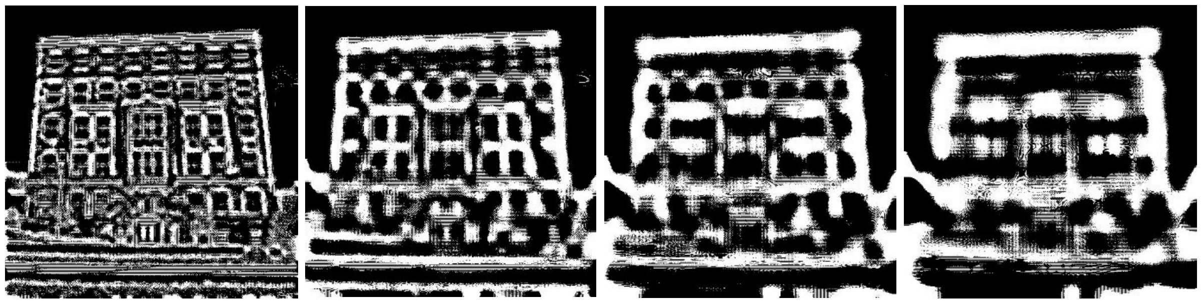
In this question, I firstly created gaussian smoothed James building images using 'imgaussfilt' function in Matlab, with a series of σ = 4,8,12,16. Later, create a heat equation function to iterate t times of computing second order partial derivatives of x and y of the image, summing up which means Laplacian operation, and adding them onto the input image. Also create 4 images by running heat equation for t = ½* σ^2 for each σ.

Gaussian smoothing with σ = 4

Heat equation diffusion for t = 8

As seen in the two images above, we can barely see any differences between them, so is the same with other three pair of images. If we compute the differences between images in pair and show it in figure, it is still full of dark, which indicates the differences are really small. But I think this is mainly because I used 'imgaussfilt' to smooth the image. If I use 'conv2' with my own creation of a Gaussian filter, the difference will be quite obvious due to <u>the different size of gaussian kernels</u>. For my results, I made a binary image to show all the pixels that these two image do not match perfectly, and below are what I got.

Although quite much pixels are white, indicating there is a difference between the output of two methods, the difference magnitude is rather small. This discrepancy occurs might be due to <u>the way I compute second order partial derivative</u> leading to some calculation errors as well as we used heat equation function to compute discrete matrices instead of the continuous values as it should be.



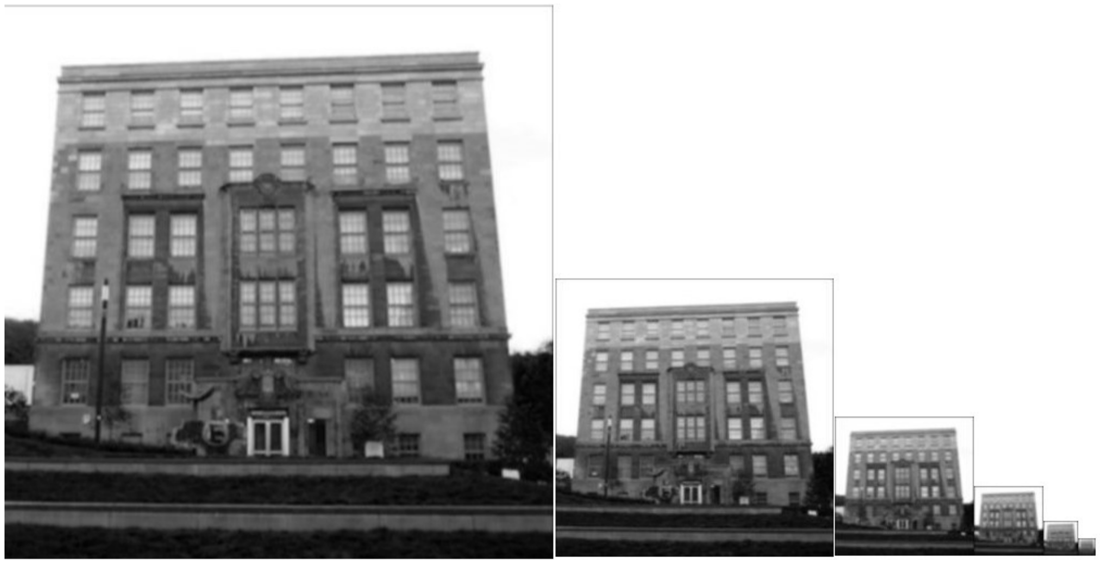
$\sigma = 4$ $\sigma = 8$ $\sigma = 12$ $\sigma = 16$

Question2

Codes and comments in file: ‘Question2.m’

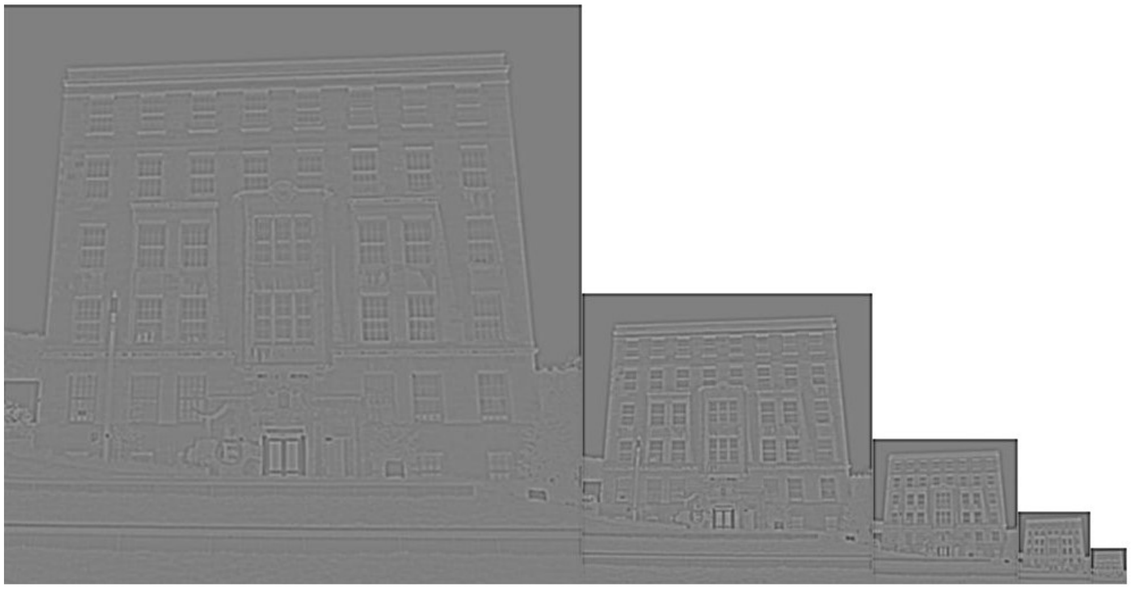
Part 1

Create a gaussian pyramid, in each level, smooth the image with a gaussian filter and then resize it.



Part 2

Create a laplacian pyramid, in each level, upsample the image at higher level and calculate differences.



Part 3

In heat equation, the partial derivative of t equals to the Laplacian operation, which is:

$$\frac{\partial I(x,y,t)}{\partial t} = \Delta I(x,y,t) = \frac{\partial^2 I(x,y,t)}{\partial x^2} + \frac{\partial^2 I(x,y,t)}{\partial y^2}$$

Due to the consistency of heat equation and gaussian smoothing, similarly we get:

$$\frac{\partial G(x,y,\sigma)}{\partial \sigma} = \sigma \nabla^2 G(x,y,\sigma)$$

And the partial derivative of G(x,y, σ) wrt. σ approximately equals to the difference between Gaussian function with σ and a slightly larger Gaussian with k σ:

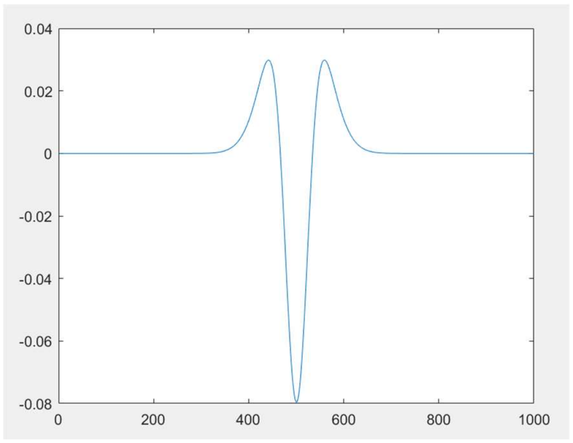
$$\frac{\partial G}{\partial \sigma} \approx \frac{G(x,y,k\sigma) - G(x,y,\sigma)}{k\sigma - \sigma}$$

Therefore,

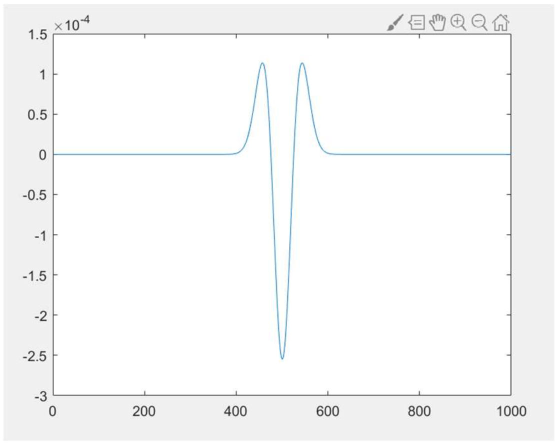
$$G(x,y,k\sigma) - G(x,y,\sigma) \approx (k-1)\sigma^2 \nabla^2 G$$

As a result, differences of Gaussian approximates Laplacian of Gaussian as long as the factor $(k-1)\sigma^2$ approximates constant.

Next, create difference of 1D Gaussian functions and compare to Laplacian of gaussian with σ=1 and k=2. As shown below, these two plots resembles whereas the scales and values are not exactly the same.



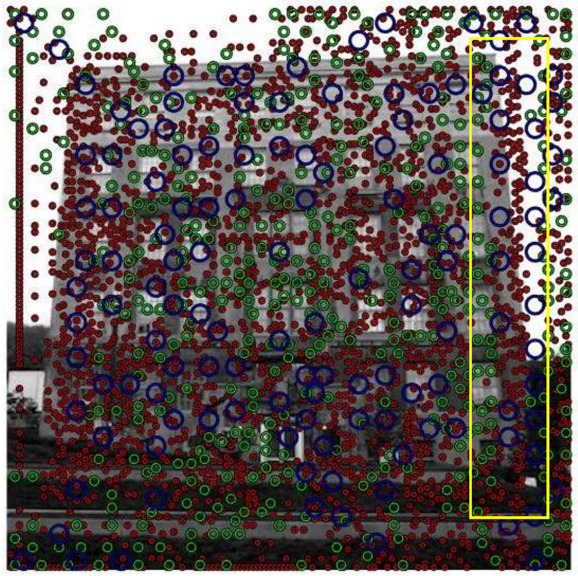
difference of gaussian

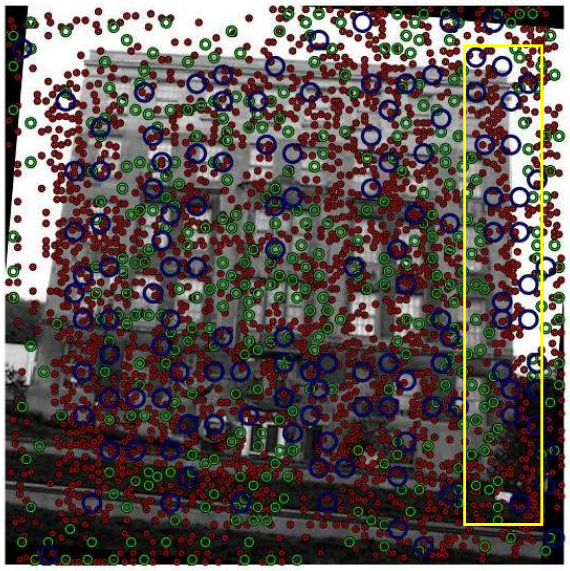


Laplacian of gaussian

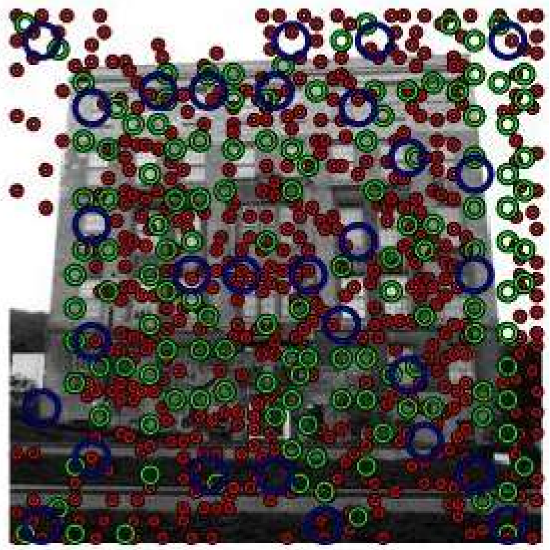
Part 4

In the first part of SIFT algorithm, we need to find key-points in the Laplacian pyramid which are extrema over location as well as scale. So the strategy is to go through the middle levels of the pyramid. In our case, the Laplacian pyramid has level 1 to 5 and we have to check for each pixel in level 2,3,4. First step is to resize the upper level h+1 and lower level image h-1 to the current level h, then compare each pixel in level h with the other 26 neighbours of it, save as extrema if current pixel has the largest or smallest value. At last, use 'viscircles' function to plot all the (σ,x,y) tuples I saved.





In these figures, dots in red, green and blue represent extrema in level 2,3,4 respectively. Later, use 'imrotate' to rotate the original image for 5 degrees and find keypoints in this set of Laplacian pyramid, the result of which is shown on the right. The quantities of keypoints are approximately the same and if we check for example the vertical series of dots in yellow frames, it's obvious that the line those dots forms has rotated a little bit in the right figure.

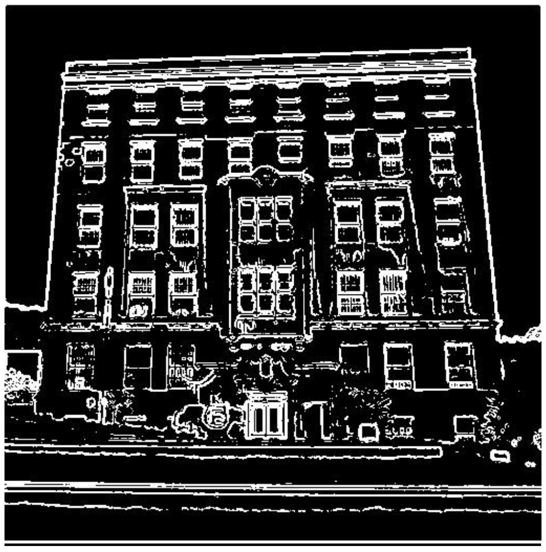


Here is the resized image with keypoints drawn.

Question3

Codes and comments in file: 'Question3.m'

First, apply codes from assignment1 to compute gradient magnitude and orientation of each pixel, binary the pixel magnitudes with threshold = 14. Then build 3 vectors x, y, theta by using find() function to save indexes of non-zero magnitude elements in vectors 'row' and 'col' and searching orientations by those indexes with matlab function 'sub2ind'.



Edge map

After that, implement a RANSAC-like function as wrote in the requirements . The way I used to define the line model is using line formula $(x-x_0)$*slope + $(y-y_0) = 0$, where 'slope' is the gradient of the direction perpendicular to gradient orientation. If we transfer this formula into form of

Ax+By+C=0, then we get A = slope, B = 1, C= -x^0*slope-y_0, so distance from each point(x,y) to the current selected pixel(x_0,y_0) is :

$$d = \frac{|Ax + By + C|}{\sqrt{A^2 + B^2}} = \frac{|slope * x + 1 * y - x_0 * slope - y_0|}{\sqrt{slope^2 + 1^2}}$$

The way I decided if one point is an inlier or outlier is to choose each point in the edge map, compute this distance and choose those has a distance smaller than 2(flexible) and has approximately same orientations(difference of these radians is smaller than 1/(2*pi) as inliers. Save as the best model if the count of inliers are larger than current inlier counts.

After randomly choose 1000 pixels and repeat these operations, plot the current inlier and outliers. Here are some of the output I got. Most likely it will choose the crowded horizontal set of lines in either top or bottom of this image :