Intro to Image Understanding (CSC420)

Available: November 9^{th}, 2020
Due Date: November 23^{rd}, 10:59:00 pm, 2020
Total: 150 marks

Notes: You are encouraged to become familiar with the **Python** environments to use it for the experimental parts of the assignment. We EXPECT EVERYONE TO SUBMIT **ORIGINAL WORK** FOR ASSIGNMENTS. This means that if you have consulted anyone or any sources (including source code), you must disclose this explicitly. Anything you submit must reflect your work.

Submission Instructions:

- Your submission should be in the form of an electronic report (PDF), with the answers to the specific questions (each question separately), and a presentation and discussion of your results. For this, please submit a file called "**report.pdf**" to MarkUs directly.
- Submit documented codes that you have written to generate your results separately. Please store all of those files in a folder called Assignment2, zip the folder, and then submit the file "**Assignment2.zip**" to MarkUs. You should also include a **"README.txt"** file (inside the Assignment 2 folder) which details how to run the submitted codes.
- Don't worry if you realize you made a mistake after submitting your zip file: you can submit multiple times on MarkUs. We always only mark the last submission.

Q1) (20 marks)

The Laplacian of Gaussian operator is defined as:

$$\nabla^2 G(x, y, \sigma) = \frac{\partial^2 G(x, y, \sigma)}{\partial x^2} + \frac{\partial^2 G(x, y, \sigma)}{\partial y^2} = \frac{1}{\pi\sigma^4}\left(\frac{x^2 + y^2}{2\sigma^2} - 1\right) e^{-\frac{x^2+y^2}{2\sigma^2}},$$

where the Gaussian filter G is defined as

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

The characteristic scale is defined as the scale that produces peak (minimum or maximum) of the Laplacian response.

1. **(10 marks)** What scale (i.e. what value of σ) maximises the magnitude of the response of the Laplacian filter to an image of a black circle with diameter D on a white background? Justify your answer.
2. **(5 marks)** What scale should we use if we want to instead detect a white circle of the same size on a black background?

3. **(5 marks)** Experimentally find the value of σ that maximises the magnitude of the response for a square of side length s.

Q2) (20 marks) RANSAC

In class we saw an example of aligning a planar query image (e.g. a book cover) with an image of it. In the example used in class we used `cv2.findHomography`, which finds a 3×3 perspective projection matrix and can be called with cv2.RANSAC to allow for RANSAC-based robust estimation.

1. **(10 marks)** Implement your own version of RANSAC together with `cv2.getAffineTransform`, which estimates an affine transformation given 3 point matches and via least squares. Try your RANSAC-based affine warp estimation on the example given in class (Book_cover & Book_pic) and visualize it using `cv2.warpAffine`.

   - In your RANSAC implementation assume that at least half of the matches found via SIFT (or SURF or ...) descriptor matching are inliers, and that we want to find the correct affine with higher than 99.5% chance. Set your alignment threshold to 5 pixels; i.e. if a matched pair of descriptors are aligned within 5 pixels you can consider them as voting for a particular alignment.
   - compare the visualization of the boundary found by the affine warp and by the homography.
   - compare the 2×3 M affine matrix found via RANSAC with the 3×3 homography found via RANSAC

2. **(10 marks)** Update your RANSC code such that it dynamically updates the required number of iterations if a given RANSAN iteration aligns a higher percentage of point matches than the initial estimate.

   - show your derivations
   - compare the number of iterations between the original implementation and this updated version when you try to find an affine warp between the book cover the picture of the book.

Q3

Q4) (50 marks) Optical flow

In the Lucas-Kanade construction, the motion field is obtained by finding, for each pixel in an image, a corresponding pixel in the next frame of the sequence of images, in such a way that it minimizes the sum of squared intensity differences computed over a window. We saw in class in our discussion that this leads directly to the use of the second moment matrix. In this part of the assignment we will explore the use of this strategy for frame to frame optical flow (motion field estimation). Using essentially the same notation as in the notes, but treating I^n and I^{n+1} as successive frames, the motion field (v_x, v_y) for frame I^n satisfies:

$$\begin{bmatrix} \sum(\frac{\partial I^n}{\partial x})^2 & \sum(\frac{\partial I^n}{\partial x})(\frac{\partial I^n}{\partial y}) \\ \sum(\frac{\partial I^n}{\partial x})(\frac{\partial I^n}{\partial y}) & \sum(\frac{\partial I^n}{\partial y})^2 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix} = - \begin{bmatrix} \sum(I^n(x,y) - I^{n+1}(x,y))\frac{\partial I^n}{\partial x} \\ \sum(I^n(x,y) - I^{n+1}(x,y))\frac{\partial I^n}{\partial y} \end{bmatrix}$$

This motion field can be solved either in one shot, or via an iterative process, as discussed in class. However, there are a few parameters that you will need to play with for the actual implementation. The first has to do with the σ of the 2D Gaussian you will use to smooth each frame spatially. This is required because otherwise the image gradients (I_x, I_y) that will be used to construct the second moment matrix will not be well defined. The second has to do with the window size in which one averages (computes sums) for the second moment matrix parameters. The third has to do with the fact that derivatives in time on the right hand side of the equation might not be well defined, so for those you might also want to "smooth", but with a 1D Gaussian.

At the end of the day, once you have settled on those parameters, you can compute frame to frame motion field estimates for v_x, v_y. Attempt to do this for the image sequence(s) we have given and to overlay the results as a quiver plot for a particular frame. For simplicity you can start by overlaying a flow vector only for the central pixel in a window and you can assume that the windows do not overlap spatially. Once that is working, you can then compute flow vectors more densely (by allowing the windows to overlap spatially). Try to make some sense of your results once you have them and provide a discussion of them. For example, you would expect that at homogeneous regions the flow vectors would be somewhat incoherent geometrically (but would be small) and at background regions the flow vectors would also be small, so you can threshold on flow vector length to get a more useful (qualitatively) quiver plot.

Q4) (60 marks) Local Descriptor: Histogram of oriented gradients

The histogram of oriented gradients (HOG) is a feature descriptor used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in localized portions of an image. This method is similar to that of edge orientation histograms, scale-invariant feature transform (SIFT) descriptors, and shape contexts (a similar technique we haven't seen in class), but differs in that it is computed on a dense grid of uniformly spaced cells and uses overlapping local contrast normalization for improved accuracy. Until deep learning, HOG and another method we will later see in class (deformable part model) were long standing top representations for object detection.

In this assignment, you will implement a variant of HOG. Given an input image, your algorithm will compute the HOG feature and visualize as shown in Figure below (the line directions are perpendicular to the gradient to show edge alignment).

The orientation and magnitude of the red lines represents the gradient components in a local cell. A HOG descriptor is formed at a specified image location as follows:

1. Compute image gradient magnitudes and directions over the whole image, thresholding small gradient magnitudes to zero.

2. Center a cell grid $(m \times n)$ on an image location. To create this grid cell, assume we have a fixed size length for each of cell in this grid, let us call that size τ. For example, if your image is size of 1021×975 and your $\tau = 8$, then you will have a grid size of $(m = 127) \times (n = 121)$. You can ignore the boundary of the image that can not be fit into a grid (on either end), i. e., just consider the crop of the image that fits to the grid perfectly 1016×968.

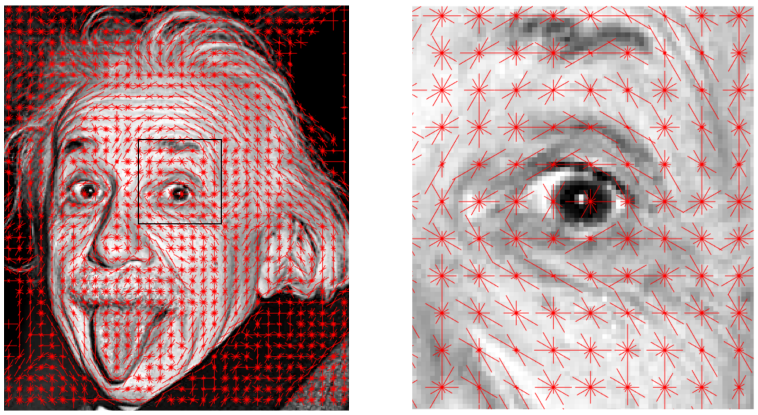

Figure 1: HoG features plotted on an example image.

3. For each cell, form an orientation histogram by quantizing the gradient directions and, for each such orientation bin, add the (thresholded) gradient magnitudes. This process can be done in two steps: Imagine gradient orientations are discretized by 6 bins:

$$[-15°, 15°), [15°, 45°), [45°, 75°), [75°, 105°), [105°, 135°), [135°, 165°)$$

Remember $165° = -15°$ where the orientation is not directed. Now create a 3D array $(m \times n \times 6)$ where in each element of this 3D array you will store the corresponding gradient magnitude of that bin and the number of occurrences. Visualize this array on the given images for this Question similar to the quiver plot of Figure 1.

4. To account for changes in illumination and contrast, the gradient strengths must be locally normalized, which requires grouping the cells together into larger, spa- tially connected blocks (adjacent cells). Given the histogram of oriented gradients, you apply L2 normalization as follow:

   - Build a descriptor of the first block by concatenating the HOG within the block. You can use block_size = 2, i.e., 2×2 block will contain $2 \times 2 \times 6$ entries that will be concatenated to form one long vector
   - Normalize the descriptor as follow:

$$\hat{h}_i = \frac{h_i}{\sqrt{\sum_i h_i^2 + e^2}}$$

     where h_i is the i^{th} element of the histogram and $\hat{h}_i$ is the normalized histogram. e is the normalization constant to prevent division by zero (e.g., $e = 0.001$).
   - Assign the normalized histogram to first cell of histogram array (1,1)
   - Move to the next block of histogram array (1,2) with the stride 1 and iterate 1-3 steps above.

The resulting histogram array will have the size of $(m-1) \times (n-1) \times 24$. Compute histogram arrays for the images given for this question, and store them in a single line text file where

the data is stored row by row (first row then second row etc.). Submit both codes and the files that are generated by your code. Please note that the file should have the same name as the image (e.g. 'image.jpg' $\rightarrow$ 'image.txt').