CS5290: Computer Vision

Department of Computer Science and Engineering, IIT-Hyderabad

Assignment 3

Max Marks: 75
This assignment is to be done individually.

Submission Due: 30 Sep 2015 11:59 pm

Instructions

- Your submission, named Your_Roll_No_Assign2 should comprise of a ZIP file with a DOC/DOCX/PDF file with your theory/descriptive answers, and a .PY file for each programming exercise. Please name the .PY file with the problem number (e.g. Prob2a.py). Make sure your code actually works (in an empty workspace) and that all your results are displayed properly.
- For late submissions, 10% is deducted for each day (including weekend) late after an assignment is due. Late submissions will automatically use your grace days balance, if you have any left. You can see your balance on the "Grace Days" section of the course portal.
- Your evaluation will of course depend on the correctness of your answer, but also on a clear presentation of your results and good writing style. Note that you will get marks for the solution, not for the result. If you get stuck, try to explain why and describe the problems you encountered – you can get partial credit even if you did not complete the task. So please hand in enough information for us to understand what you did, what things you tried, and how it worked!
- Please read the department plagiarism policy available under the course site. Do not engage in any form of cheating – please talk to instructor or TA if you have concerns.

THEORY

1. Steerable Filters *(3+6+4 = 13 marks)*: Let $G^0(x, y)$ be some 2-d filter, a function of the Cartesian coordinates x and y. Let $G^\theta(x, y)$ be a rotation of $G^0(x, y)$ by q radians about the origin in the counterclockwise direction, i.e.,

$$G^\theta(x, y) = G^\theta(r\cos\Phi, r\sin\Phi) = G^0(r\cos\Phi - \Theta, r\sin\Phi - \Theta)$$

where $r = \text{sqrt}(x^2 + y^2)$ and $\tan\Theta = y/x$. In this problem, you may give your answers in Cartesian or polar coordinates, whichever is more convenient.

(a) Suppose $G^0(x, y) = -2xe^{-(x^2 + y^2)}$. Find $G^\theta(x, y)$.

(b) Show that:

$$G^\theta(x, y) = \cos(\Theta)\, G^0(x, y) + \sin(\Theta) G^{\pi/2}(x, y)$$

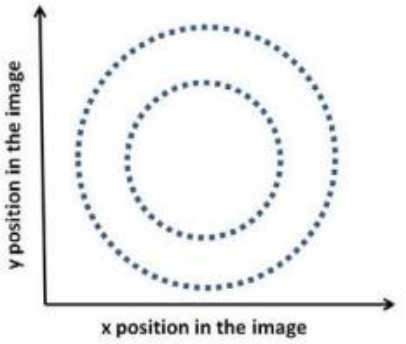
and that the output image $F(x, y) = I(x, y) * G^\theta(x, y)$ is equal to:

$$\cos(\Theta)\{ I(x, y) * G^0(x, y)\} + \sin(\Theta)\, \{I(x, y) * G^{\pi/2}(x, y)\}$$

where * denotes convolution.

(c) Find the direction and magnitude of maximum response at a point (x, y) of the image I(x, y) to the steerable filter $G^\theta(x, y)$. The direction of maximum response is the q, such that $F^\theta(x, y)$ has the biggest magnitude. Give your answer in terms of the image I(x, y) and the responses $F^0(x, y)$, $F^{\pi/2}(x, y)$ to the two steerable basis filters $G^0(x, y)$, $G^{\pi/2}(x, y)$. Note that other steerable filters could require more basis filters than two.

2. K-means Clustering *(2+3 = 5 marks)*: Consider the adjoining figure. Each data point denotes an edge point extracted from an image. Say we are going to use k-means to cluster these points' positions into k=2 groups. That is, we will run k-means where the feature inputs are the (x,y) coordinates of all the small square points. What is a likely clustering assignment that would result? Briefly explain your answer with proper diagram. Can you provide a method to improve the clustering assignment?



3. SIFT *(2+5 = 7 marks)*: (i) Given that the dimensions of the SIFT keypoint descriptor is 128, what exactly does the value recorded in a single dimension of a SIFT keypoint descriptor signify? (Don't just describe how it is obtained, think and explain what aspect of the keypoint it is capturing.)

(ii) Give any one idea to extend SIFT to find spatio-temporal keypoints in videos (3-dimensions, considering space and time). Provide the pseudocode of your idea, with details of every step involved.

PROGRAMMING

1. Harris Corner Detection *(10+3+3+8 = 24 marks)*:

(a) The Harris Corner Detector operates on the auto-correlation matrix, which can be described as:

$$\mathbf{A} = w * \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Using this matrix, the feature points are found by computing the value:

$$H = det(A) - \alpha trace(A)^2$$

Write your own code to implement the Harris corner detector. Run your corner detector on the given 'grid1.jpg' image and try to tweak the parameters so that all (or as many as you can manage) corners of the squares are detected and the number of spurious corners detected is minimized.

An OpenCV implementation of Harris corner detector is here:
http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html#harris-corners

Compare the output of your program with this OpenCV implementation (both performance and speed) on any of the images provided in parts (b)-(c) below.

(b) For fixed parameter values, run the detector on 'grid1.jpg' and 'grid_rotated.jpg'. The latter image is a rotated copy of the former. Is this corner detector rotation-invariant? That is, if we rotate the input image, do the detected corner positions rotate by the same amount? Justify your answer based on your observations.

(c) Is the corner detector scale invariant? That is, if we scale down the input image, are all the detected corner positions scaled accordingly? You can test this experimentally by comparing the corner detection on the images 'grid1.jpg', 'grid2.jpg', 'grid3.jpg'. Each image in this sequence is half the size of its predecessor. Justify your answer based on your observations.

(d) Extend your implementation of the Harris corner detector to a Harris-Laplace scale-invariant interest point detector (which uses Harris for detecting extrema in space, and the Laplacian of Gaussian for detecting extrema in scale). Compare your interest point detection with the interest point detection of SIFT (`cv2.sift`) on images of your choice, and explain your observations. Include figures in your report, and explain why one does better than the other in cases of specific interest points from your figures.

2. Introduction to Feature Matching *(4 + 2 + 2 + 4 + 4 = 16 marks)*:

(a) Suppose we are given a reference template image of an object, and a second image in which we want to find the same object; an example pair of images is shown below.

(a) Template image


(b) Search image

To do so, we need to find pairs of features in the two images that correspond to the same point on the object; these should be features with similar descriptors. Thus, feature matching involves finding features with similar descriptors. We'll use SIFT and nearest neighbor matching for this task: for each feature in image 1 (the template image), we'll find the most similar feature in image 2 (the search image), that is, the feature with the most similar descriptor. We'll define the distance between two descriptors a and b (two 128-dimensional vectors) using the standard Euclidean distance:

$$\texttt{distance}(a, b) = \sqrt{\sum_{i=1}^{128} (a_i - b_i)^2}$$

Assuming a and b are column vectors, this formula can be written using vector operations as:

$$\texttt{distance}(a, b) = \sqrt{(a - b)^T (a - b)}$$

where T denotes the transpose operator. Your first task is to write a function to find nearest neighbors. This function will take in two sets of descriptors, `descs1` and `descs2` (with `n` and `m` descriptors, respectively), and will return a `3×n` matrix, i.e., a column for each descriptor in `descs1`. The first two rows of this matrix will be pairs of indices of matching features, and the third row will be the distances between the matching feature descriptors. For instance, the first column of this matrix might be [1; 27; 1.5]; this means that feature 1 in image 1 is closest (in terms of its descriptor) to feature 27 in image 2, and the distance between the descriptors is 1.5. (There are many ways to write this function; we'll reserve a bit of extra credit for implementations that are much faster than the simplest approach.)

(b) The above matching function will undoubtedly return many false matches. One way to reduce the number of false matches is to threshold the matches by distance between descriptors. You will do this by writing a function to threshold matches, which takes in a set of matches (the output of the previous function) and a threshold, and returns a new matrix of matches whose distances are less than the threshold.

(c) It turns out that the even this thresholding doesn't work that well. Consider the example image pair shown in the figure below.

There are a lot of repetitive features in these images, and all of their descriptors will look similar. To find unique, distinctive feature matches, consider the following strategy: for each descriptor `a` in image 1, find the two nearest neighbors in image 2. Call these `b` and `c`, and let their distances from a be `distance(a,b)` and `distance(a,c)`. If `distance(a,b)` is much smaller than `distance(a,c)`, then `b` is a much better match than even the second closest

feature. Thresholding using this test will tend to get rid of features with multiple possible matches. To make this concrete, we'll define our new distance function between `a` and `b` as the ratio of the distances to the two nearest neighbors.

$$\frac{\texttt{distance}(a, b)}{\texttt{distance}(a, c)}$$

We'll call this the ratio distance. You'll now write a function that returns the ratio distance in the third row of the output matrix.

(d) You'll now need to find the top two nearest neighbors in this function. Your function must spend at most O(n) time finding the top two neighbors for each feature in the first image, where n is the number of features in the second image. You can now use your threshold matches function on the output of this function. Note that the distances are now ratios rather than pure distances, so the "units" are different (for instance, the ratio distance will be always be between 0 and 1). For the ratio distance, a threshold of 0.6 usually works pretty well.

(d) To visualize a set of matches, write a function that takes in two images, two sets of frames, and your matches, and draws a figure where you can see the matches. Here's an example figure below. (Images have been provided with the assignment for you to test your submission).

3. Textons *(10 marks)*: This problem consists in computing textons (building blocks of image textures) given an image. Take any image from the provided 'Textons' folder.

1. Apply Sobel operators (*x* and *y* directions) to the image. Let's call the outputs, M_x and M_y. Plot the results and show them in your submission.
2. Compute orientation based on M_x and M_y (i.e. $\theta = \tan^{-1}(M_y/M_x)$). Plot the result and show them in your submission as arrows emanating from the points.
3. Discretize the orientation into 8 different angles. For each pixel, compute a descriptor that will be the histogram of orientations in a neighborhood of 11 × 11 pixels. Orientations should only contribute to the histogram if the magnitude M at each pixel is above some threshold (you have to handpick this threshold).
4. Using K-means clustering (you can use OpenCV's function for this), cluster the vectors into 5, 10 and 50 textons. For each setting, assign textons to each pixel, plot the result and include it in your submission.

Please submit all the figures and codes in addition to any of your comments.