CSE 152 – Introduction to Computer Vision – Assignment 3 (SP15)
Instructor: Ben Ochoa
Maximum Points : 50
Deadline : 11:59 p.m., Wednesday, 18-May-2016

Instructions:

- This assignment must be solved, and written up individually
- There is no physical hand-in for this assignment.
- Coding for this assignment must be done in MATLAB. In general, code does not have to be efficient. Focus on clarity, correctness and functionality here, we can worry about speed in another course.
- All code developed for this assignment must be included in the appendix of the report. If the appendix does not contain your code, points may be deducted.
- Submit your assignment electronically by email to Akshat Dave `[akdave@ucsd.edu]` **and** Mihir Patankar `[mpatanka@eng.ucsd.edu]` with the subject line *CSE152-Assignment-3*. The email should contain one attached file named [`CSE_152_HW3_<student-pid>.zip`]. This zip file must contain the following two artifacts:
  1. A pdf file named [`CSE_152_HW3_<student-pid>.pdf`] containing your writeup. Please clearly state the author's full name and student identity in the report. The report must contain all the result images. Images not embedded in the report will not be considered for evaluation. In the case of figures containing text and labels, it is your responsibility to ensure that the text is readable; points may be deducted if these are not readable.
  2. A folder named [`CSE_152_HW3_<student-pid>_code`] containing all your matlab code files

1 Corner Detection [20 points]

In this section, we will implement a detector that can find corner features in our images. To detect corners, we rely on the matrix C defined as:

$$C = \begin{pmatrix} \sum_w I_x^2 & \sum_w I_x I_y \\ \sum_w I_x I_y & \sum_w I_y^2 \end{pmatrix}$$

where the sum occurs over a $w \times w$ patch of neighboring pixels around a point p in the image. The point p is a corner if both eigenvalues of C are large.

(a) (10 points) Implement a procedure that filters an image using a 2D Gaussian kernel and computes the horizontal and vertical gradients I_x and I_y. You cannot use built in routines for smoothing. Your code must create your own Gaussian kernel by sampling a Gaussian function. You can use built-in functions to perform convolution. The width of the kernel should be $\pm 3\sigma$. Include images of the two components of your gradient I_x and I_y on the image `dino01` (found in `dino2.mat`) for $\sigma = 1$, $\sigma = 3$, and $\sigma = 5$.

(b) (10 points) Implement a procedure to detect corner points in an image. The corner detection algorithm should compute the minor (i.e., smallest) eigenvalue $\lambda_{\min}$ of the matrix C at each pixel location in the image and use that as a measure of its cornerness score. Run non-maximal suppression, such that a pixel location is only selected as a corner if its minor eigenvalue $\lambda_{\min}$ is greater than that of its 8 neighboring pixels. Have your corner detection procedure return the

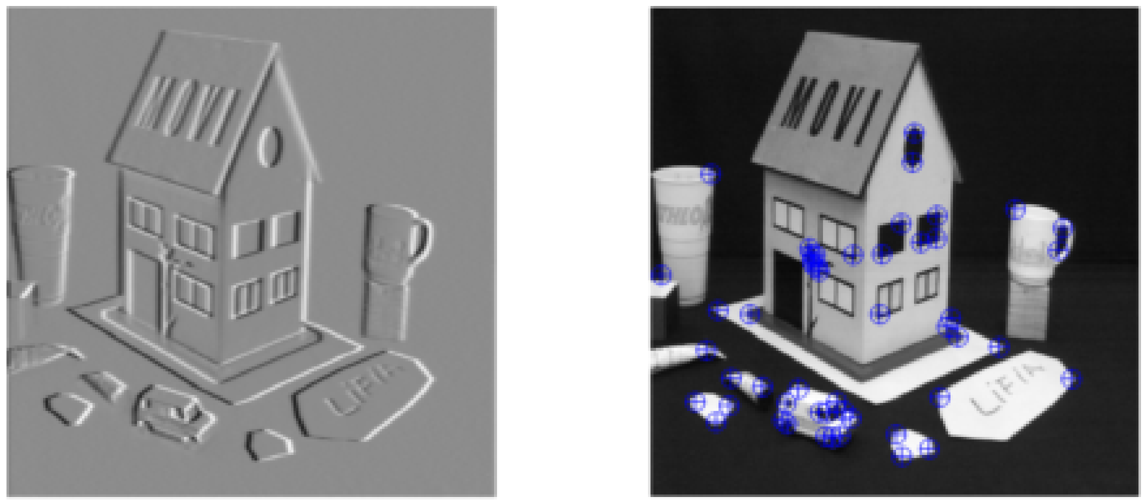

Figure 1: x-component of the gradient I_x (left) and the top 50 detected corner points (right) on the image `house.pgm`, both with $\sigma = 1$. Your report should also include the y-component of the gradient I_y and results for different values of σ on the dinosaur images.

top n non-maximally suppressed corners. Test your algorithm on `dino01` with $n = 50$ corners, and plot the resulting detected corners using `drawPoint.m`. Show corner detection results and report the parameters used.

Notes:

- An example of what your plots should look like on a different image pair are shown in Figure 1.
- You can use the MATLAB function `conv2.m` to perform 2D convolution for computing gradient images with the `'same'` option.
- The patch width w for detecting corners should be the same as the width of your Gaussian kernel. You can use the function `conv2.m` to help compute $\sum I_x^2$, $\sum I_x I_y$, and $\sum I_y^2$ efficiently for all pixel locations in the image.

2 Sparse Stereo Correspondences [10 points]

For this part of this assignment, we will try to find correspondences between detected corners on stereo image pairs. Extract a 9×9 patch of neighboring pixels around each corner point x in `dino01`. Consider matching x to a corner point x' in `dino02` if the Normalized Sum of Squared Differences (NSSD) distance between the two patches is less than some threshold τ_a. If multiple corner points are within this threshold, choose the one with the smallest NSSD distance (i.e., the best match) such that there only one-to-one matches. Display the images of detected matches, and report the number of correct matches and the number of incorrect matches (by visual inspection). Specify which value of τ_a you used.

Notes:

- An example of what your plots should look like on a different image pair are shown in Figure 2.
- You can discard corner points that are close to the boundary of the image, such that extracting a patch around that corner point would go outside the boundary of the image.

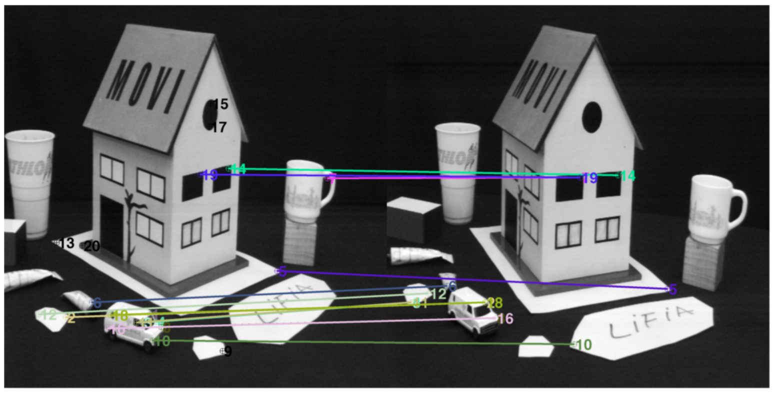

Figure 2: Predicted corner point matches on `house1.pgm` and `house2.pgm`. In this matching, the following matches are correct: [14,19, 5, 6, 3], and the following incorrect [7, 12, 2, 18, 16, 11, 3, 8, 10, 1]. The remaining 5 points were not matched. Your report should include matching results on the dino images when using NSSD.

- In order to create a figure like Figure 2, you can do imshow([img1 img2]). This makes the plot treat them as one image, and then you can add the dots using `drawPoint.m` and `line`. Do *not* use `drawLine.m` - that is only for drawing epipolar lines. Remember to add an offset (= number of columns in the first image) to points you want to plot in the second image.
- The NSSD distance between two image patches P_1 and P_2 is the sum of squared differences between each pixel in the patch after normalizing each patch by their mean and variance.

$$NSSD(P_1, P_2) = \sum_{i,j} \left(\tilde{P}_1(i,j) - \tilde{P}_2(i,j) \right)^2 \quad \text{where,} \quad \tilde{P}_k(i,j) = \frac{P_k(i,j) - \mu_k}{\sigma_k}, \tag{1a}$$

$$\mu_k = \frac{1}{n} \sum_{i,j} P_k(i,j), \quad \text{and,} \quad \sigma_k = \frac{1}{n} \sum_{i,j} \left(P_k(i,j) - \mu_k \right)^2 \tag{1b}$$

3 Estimating the Fundamental Matrix [20 pts]

The geometry of two camera views are related by the fundamental matrix **F**, which maps a point in one image to the corresponding epipolar line in the other image. Given eight or more image point correspondences $x \leftrightarrow x'$ between the two views, a linear estimate of the fundamental matrix can be computed using the direct linear transformation (DLT) algorithm. Typically before estimating the fundamental matrix from an image pair, one must perform outlier rejection (e.g., using RANSAC) in an attempt to remove false matches. However, for this problem we will assume that we have a set of relatively "clean" correspondences (i.e., no false matches) provided in `cor1` and `cor2`.

(a) (15 points) Implement linear estimation of the fundamental matrix using the DLT algorithm (with data normalization) by writing a function `estimateFundamental`(x, x') to estimate the fundamental matrix **F**. Do not forget to enforce the rank-2 constraint on the fundamental matrix.

(b) (5 points) Using your developed `estimateFundamental`, estimate the fundamental matrix **F** from the point correspondences provided in `cor1` and `cor2`. Similar to the previous problem, plot the point correspondences used to estimate the fundamental matrix using `drawPoint.m` and `line`. Additionally, in a separate figure, use any 3 points x from the `dino01` image and plot the epipolar lines $l' = \mathbf{F}x$ corresponding to these point on `dino02` using `drawLine.m`. Each epipolar line should pass through the corresponding point x'.