CSE 152 – Introduction to Computer Vision – Homework 4
Instructor: Ben Ochoa
Due : Wednesday, May 24, 2017, 11:59 PM

Instructions:

- Review the academic integrity and collaboration policies on the course website.
- This assignment must be completed individually.
- Submit your PDF report on gradescope. You must prepare a report describing your solutions and showing your results. The report must contain enough information for a reader to understand your methodology for solving the problem. Make sure you include the source code with comments under Appendix listing at the end of your report.
- Programming aspects of the assignments must be completed using MATLAB or Python.
- Submit your zipped code folder electronically by email to lmelvix@ucsd.edu, jsc078@ucsd.edu, and dperoni@ucsd.edu with the subject line **CSE152 Homework 4**. Make sure to include the **exact** subject line. The email should have one file attached. Name this file: CSE152_hw4_lastname_studentid.zip.
- All your source code should be in code folder with README.txt file explaining your code. For example, if you have **rotate_img.m**, your README.txt should have a detailed description: rotate_img.m: A function that takes an image matrix and degree as input and outputs the image rotated by the degree.
- The code is thus attached *both* as text in the appendix of the report and as m-files/py-files in the compressed archive.
- No physical hand-in for this assignment.

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:

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

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 $\mathbf{C}$ are large.

1. (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 $\sigma = 5$.

2. (10 points) Implement a procedure to detect corner points in an image. The corner detection algorithm should compute the minor (i.e., smallest) eigenvalue λ_{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 λ_{min} is greater than that of its 8 neighboring pixels. Have your corner detection procedure return

the top n non-maximally suppressed corners. Test your algorithm on dino01 with $n = 50$ corners, and plot the resulting detected corners using drawPoint.m or draw_points() function found in util.py. Show corner detection results and report the parameters used.

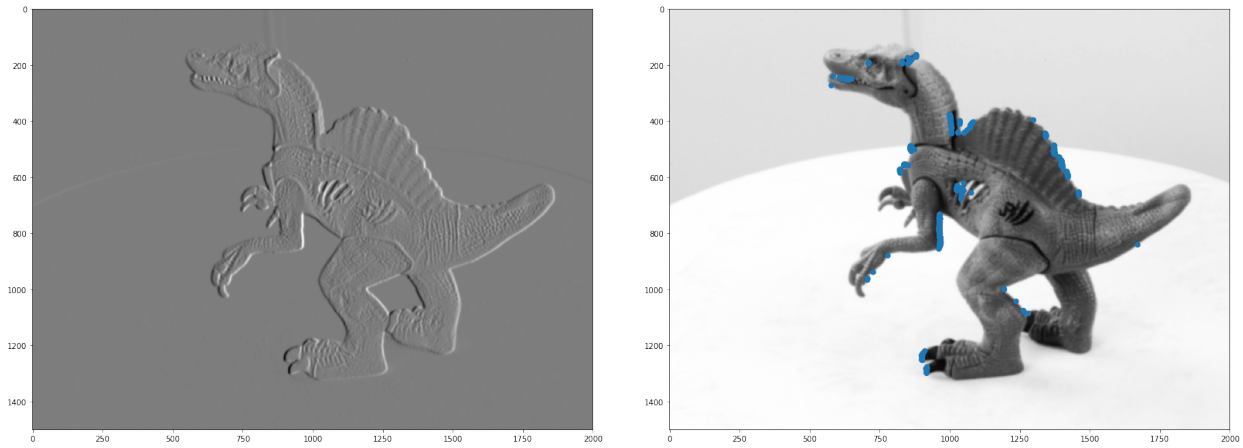

Figure 1: x-component of the gradient (left) and the top 200 detected corner points (right) on dino image

Notes:

- An example of what your plots should look like on different image pair are shown in Figure 2
- You can use the MATLAB function conv2.m to perform 2D convolution for computing gradient images with the "same" option or scipy.ndimage.filters.correlate for Python
- 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 $\sum_w I_x^2$, $\sum_w I_x I_y$ and $\sum_w I_y^2$ 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 and the ratio of distance between closest match and next best match for x with other points is less than matching ratio r_{match} (also check the same for x') . This qualifies if corner pair (x,x') is unique enough to be matched. 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 and r_{match} you used.

Notes:

- An example of what your plots should look like on image pair are shown in Figure 2

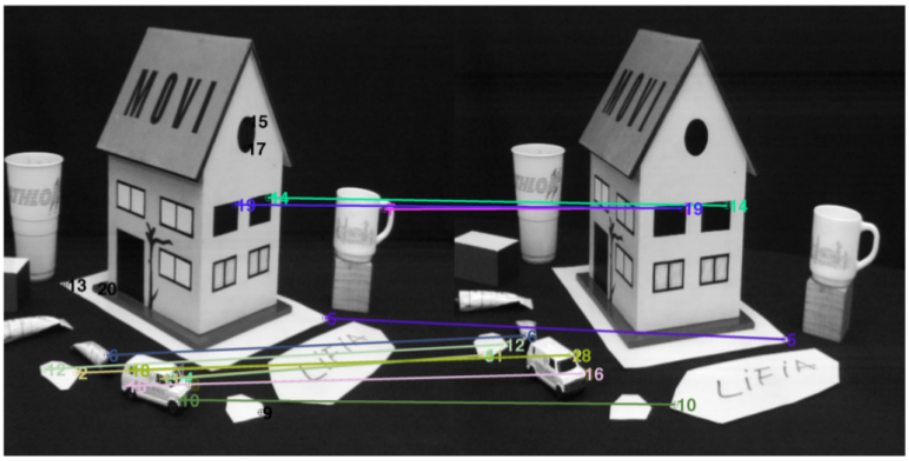

Figure 2: Predicted corner point matches on sample images. In this matching, the following matches are correct: [14,19, 5, 6, 3], and the following incorrect [7, 12, 2, 18, 16, 11, 3, 8, 10, 1]. The remaining 5 points were not matched. Your report should include matching results on the dino images when using NSSD. You don't have to label the corners

- 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
- In order to create a figure like Figure 2: In Matlab, 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. In Python, use draw_correspondences() function from util.py. Refer to IPython notebook utilities_helper.ipynb for examples on how to use this function.
- The NSSD distance between two image patches P1 and P2 is the sum of squared differences between each pixel in the patch after normalizing each patch by their mean and variance.

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

$$\mu_k = \frac{1}{n} \sum_{i,j} P_k(i,j) \text{ and } \sigma_k = \sqrt{\frac{1}{n} \sum_{i,j} (P_k(i,j) - \mu_k)^2}$$

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.

1. (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.

2. (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 (for Matlab) or draw_lines() from util.py . Additionally, in a separate figure, select any 3 points x in the dino01 image *not* contained in cor1 and plot the corresponding epipolar lines l' = Fx on dino02 using drawLine.m or draw_lines() for Python. Don't forget to include dino01 image with the selected points marked using draw_points() in Python. Each epipolar line should pass through the corresponding point in dino02.