CSE152a – Computer Vision – Assignment 3 WI14
Instructor: Prof. David Kriegman.
Revision 1

Instructions:

- This assignment should be solved, *and written up* in groups of 2. Work alone *only* if you can not find a partner. No groups of 3 are allowed.
- Submit your assignment electronically by email to obeijbom@cs.ucsd.edu with the subject line *CSE152 Assignment 3*. The email should have two files attached.
  1. A pdf file with your writeup. This should have all code attached in the appendix. Name this file: CSE_152_hw3_writeup_lastname1_lastname2.pdf.
  2. A compressed archive with all your matlab code files. Name this file: CSE_152_hw3_code_lastname1_lastname2.zip.

  The code is thus attached *both* as text in the writeup appendix and as m-files in the compressed archive.
- Please make this a proper report, with methods, thoughts, comments and discussions. All code should be tucked away in an appendix.
- No physical hand-in for this assignment.
- You may do problems on pen an paper, just scan and include in the writeup pdf file.
- In general, MATLAB code does not have to be efficient. Focus on clarity, correctness and function here, and we can worry about speed in another course.

1 Homework Overview

In this homework, we will play around with stereo images. **You will work with two images: desk1.gif & desk2.gif**. In addition, two other images are provided: house1 & house2. We provide sample outputs on the house images pair. This is for you to be able to debug and compare. However, note that since there are several parameters of choice in these method, don't expect to get the same exact results as shown in the assignment. The important thing is to get reasonable outputs (e.g. the detected corners are on actual corners in the image). But before we start coding, let's warm up with a nice pen and paper problem. Good luck!

2 Epipolar Geometry Theory [10 pts]

Suppose a camera calibration gives a transformation (R, T) such that a point in the world maps to the camera by ${}^{C}P = R\ {}^{W}P + T$.

1. Given calibrations of two cameras (a stereo pair) to a common external coordinate system, represented by R_1, T_1, R_2, T_2, provide an expression that will map points expressed in the coordinate system of the right camera to that of the left. (4 points)
2. What is the length of the baseline of the stereo pair. (2 points)
3. Give an expression for the Essential Matrix in terms of R_1, T_1, R_2, T_2. (4 points)

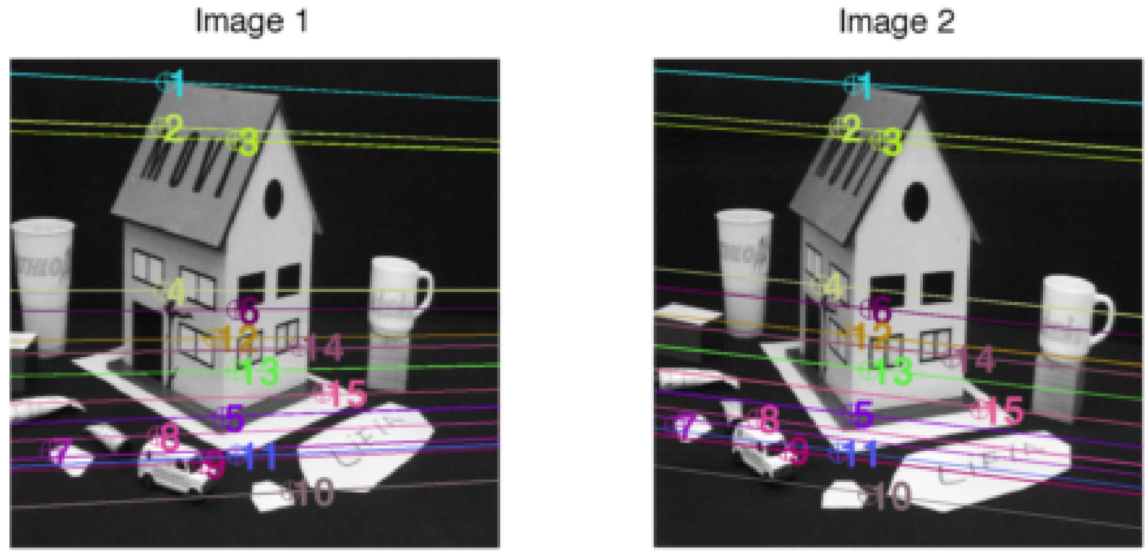
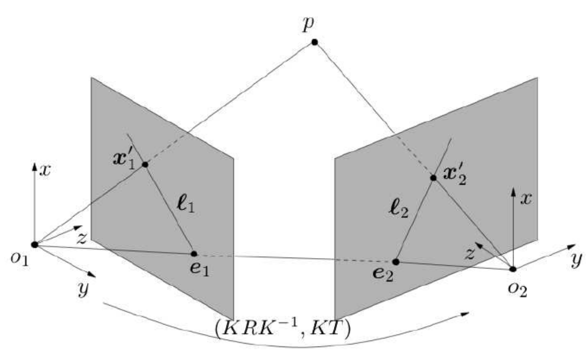


Figure 1: 15 hand-clicked points on the stereo pair *house1.gif* and *house2.gif* were used to compute the fundamental matrix F. Plot shows the locations of these 15 points in both images and their corresponding epipolar lines.



3 Epipolar Geometry Practice [20 pts]

In class, we learned that the geometry of two camera views are related by a transform called the “fundamental matrix.“ We can solve for this matrix using an algorithm called the ”eight-point algorithm“ given at least eight correspondences between the two views. In this assignment, the eight-point algorithm has already been implemented for you in *fund.m*. The goal in this section is to gain some familiarity with how the fundamental matrix relates the two views.

(a) Find the fundamental matrix relating the stereo pair of *desk1.gif* and *desk2.gif* with $n = 15$ hand-clicked correspondences. Plot the epipolar lines l_1 and l_2 in both images for at least three points in the first view, and verify that they pass approximately through the corresponding points in the second view. Include plots of hand-clicked points and epipolar lines and the values of the estimated fundamental matrix. (10 points)

(b) The eight-point algorithm requires only 8 points to solve for the fundamental matrix F, whereas in part a) you selected 15 points. Using more than 8 points makes the estimated F more stable to errors in hand-clicked locations. A consequence though is that clicked points won't in general lie exactly on their epipolar lines. As an estimate of the error in hand-clicked correspondences, compute the average distance between each clicked point in *desk2.gif* and its corresponding epipolar line and include its value in your report. (10 points)

Notes:

- You can use the built-in matlab function *cpselect.m* to select and save hand-clicked correspondences
- Use the provided functions *drawLine.m* and *drawPoint.m* to plot points and epipolar lines. An example of what your plots should look like on a different image pair are shown in Figure 1.

4 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:

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

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 C are large.

(a) 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 $\pm 3\sigma$. That is, if $\sigma = 2$, then the kernel should be 13 pixels wide. Include images of the two components of your gradient I_x and I_y on the image *desk1.gif* for $\sigma = 1$, $\sigma = 2$, and $\sigma = 4$. (10 points)

(b) Implement a procedure to detect corner points in an image. The corner detection algorithm should compute the smallest eigenvalue λ_2 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 eigenvalue λ_2 is greater than that of its 8 neighboring pixels. Have your corner detection procedure return the top n non-maximally suppressed corners. (10 points)

    Test your algorithm on *desk1.gif* with $n = 50$ corners, and plot the resulting detected corners using *drawPoint.m*. Show corner detection results for $\sigma = 1$, $\sigma = 2$, and $\sigma = 4$.

Notes:

- An example of what your plots should look like on a different image pair are shown in Figure 2.
- You can use the Matlab function *conv2.m* to perform 2D convolution for computing gradient images. Note that I2=conv2(I, k) will result in an output $I2$ that is bigger than the original input I, while I2=conv2(I, k, 'same') will result in an output that is the same size.
- Display brightness normalized images of the two components of your gradient using imshow(Ix,[])
- 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.

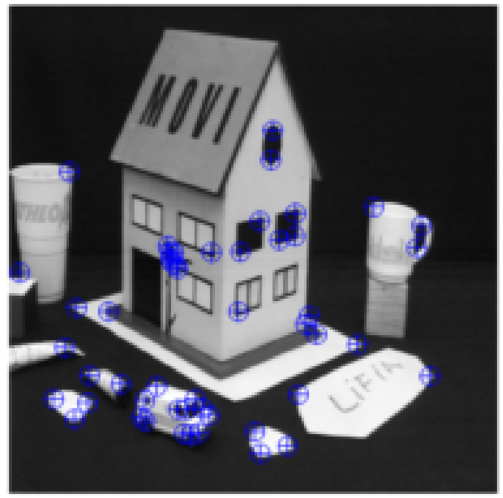


Figure 2: x-component of the gradient I_x (left) and the top 50 detected corner points (right) on the image *house.pgm*, both with $\sigma = 1$. Your report should also include the y-component of the gradient I_y and results for different values of σ on the desk images.

5 Sparse Stereo Correspondences [24 points]

Recall from class that triangulation can be used to estimate the 3D depth for stereo image pairs if 1) we have computed the epipolar geometry (as in problem 1), and 2) we can establish point correspondences between images. For the last part of this assignment, we will try to find correspondences between detected corners on stereo image pairs.

(a) We will test our correspondence algorithm on the desk stereo pair. For each image, detect $n \geq 20$ corners. For each detected corner location p in *desk1.gif* use your fundamental matrix F as computed in Problem 1 to compute its epipolar line in *desk2.gif*. Consider matching p to a corner point in *disk2.gif* if the distance to the epipolar line is less than some threshold τ_e. If multiple corner points are within this threshold, choose the one with the smallest distance to the epipolar line. Display images of detected matches using *drawPoint.m*, and count the number of correct matches and the number of incorrect matches. Specify which value of τ_e that you used. (8 points)

(b) In part a), we chose matches based on the epipolar geometry relating the two images. In this part, we consider matching corner points based on their appearance instead. Extract a 9×9 patch of neighboring pixels around each corner point p in *desk1.gif*. Consider matching p to a corner point in *desk2.gif* if the 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. As before, display images of detected matches, and report the number of correct matches and the number of incorrect matches. Specify which value of τ_a you used. (8 points)

(c) In this part, we consider matching corner points based on both their epipolar geometry and their appearance. Consider matching p to a corner point in *desk2.gif* if the distance to its epipolar line is less than τ_e, and the NSSD distance between the two patches is less than τ_a. If multiple corner points are within these thresholds, choose the one with the smallest NSSD distance. As before, display images of detected matches, and report the number of correct matches and the number of incorrect matches. (8 points)

Notes:

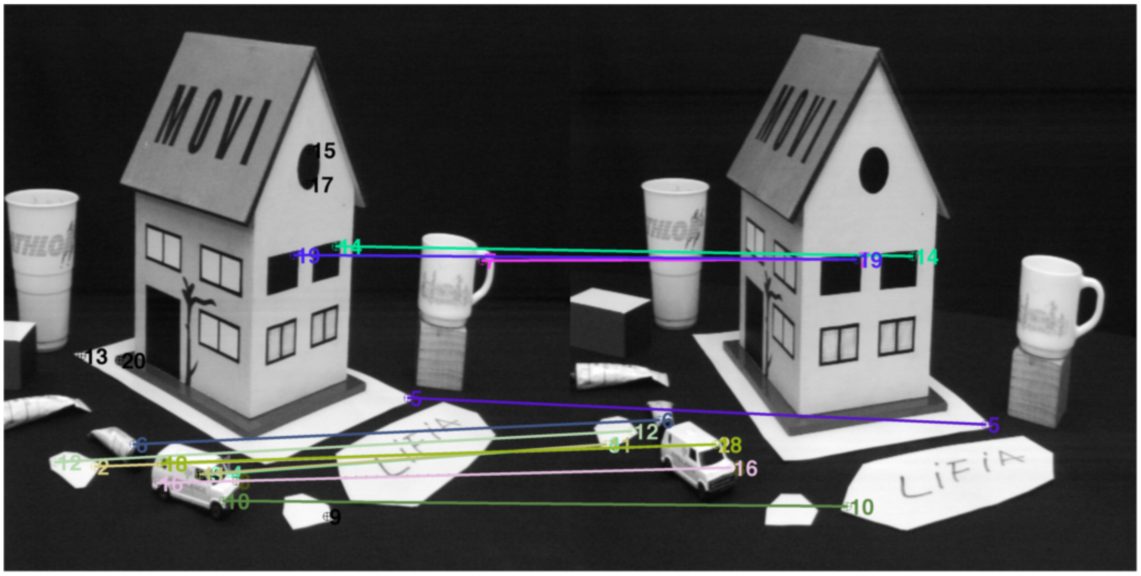


Figure 3: Predicted corner point matches on *house1.pgm* and *house2.pgm*, using just epipolar geometry. In this case, the following are correct: [14,19, 5, 6, 3], and the following incorrect [7, 12, 2, 18, 16, 11, 3, 8, 10, 1]. The remaining 5 points were not matched. Your report should include matching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

- An example of what your plots should look like on a different image pair are shown in Figure 3.
- 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.

$$NSSD(P_1, P_2) = \sum_{i,j} \left(\tilde{P}_1(i,j) - \tilde{P}_2(i,j) \right)^2$$

$$\tilde{P}_k(i,j) = \frac{P_k(i,j) - \mu_k}{\sigma_k}, \qquad \mu_k = \frac{1}{n} \sum_{i,j} P_k(i,j), \qquad \sigma_k = \frac{1}{n} \sum_{i,j} \left(P_k(i,j) - \mu_k \right)^2$$

- 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.
- In order to create a figure like Figure 3, you can do imshow([img1 img2]). This makes the plot treat them as one image, and then you can add the dots using drawPoint and line.m. Do *not* drawLine - that is only for drawing epipolar lines. Remember to add an offset (= nbr columns in the first image) to points you want to plot in the second image.
- Note that you will have to inspect manually to know if the matches are correct. Since the corner points are extracted automatically, you have *no guarantee* that the list of corners will have the same order (That e.g. corner 1 in the first image should match corner 1 in the second image). Instead, just plot it like in Figure 3, and look manually to see how well it looks.

Good luck!