Problem Set 2: Feature Matching

CS 4495/7495 Computer Vision

Prof. Jim Rehg
College of Computing, Georgia Institute of Technology
Wednesday, October 8, 2008

Due at the beginning of class, Monday October 20, 2008

In this problem set you will perform some experiments with feature matching, including extraction of
feature points, generation of feature descriptors, and the development of a matching algorithm. All students
enrolled in 4495 must submit answers for problems 1-4. Students enrolled in 7495 must submit problems
1-5. You do not need to typeset your answers, legible hand-written responses are fine.

For this problem set you will need to produce an executable that will take a pair of input images and
output a set of feature locations in each frame and a set of matches. This deliverable will be used to grade
all of the problems listed below. You must also submit the source code for your program. You are free to
use any standard programming language for this assignment. You will be provided with some sample code
for displaying images, feature points, and matches. You will also be provided with an output format which
your program must conform to. You will need to submit your executable. We will discuss the details of this
process in class.

It will be important that you write the inner loops of your code that perform the feature extraction and
matching in an efficient compiled language such as C++, java, etc. It goes without saying that attempting
to write these loops in Matlab or another scripting language is a bad idea. You may want to create Python
bindings for your routines and utilize the Python Imaging Library. We will discuss this approach in class.

The most useful resource for this project is David Lowe’s SIFT paper, which is available from the
class website. This problem set is based on an earlier problem set by Steve Seitz and Rick Szeliski. The
submission that produces the most impressive matching result will be demonstrated in class, so do your best
to wow us!

1. Finding Harris Corners

1. Write a function called ComputeHarrisMatrix that constructs the Harris matrix over a window $W$
defined by

$$H = \sum_{(x,y) \in W} m_{x,y} \nabla I(x, y) \nabla I(x, y)^T$$

In this expression, $m$ is a circularly symmetric weight mask. It is computed by sampling a 2D Gaussian
distribution at the template pixel coordinates in $W$. Use a 5 by 5 window. The output of this
function will be the 2 by 2 Harris matrix.

2. Write a function called ComputeCornerStrength which takes a Harris matrix as input and outputs the
score function:

$$S = \det(H) - \alpha \text{trace}(H)$$
where \( \text{det}(H) \) and \( \text{trace}(H) \) are the determinant and trace respectively. You should use \( \alpha = 0.06 \).

3. Write a function FindCorners which takes an input image and outputs a list of Harris corners. This function should call the previous two. A Harris corner must have a corner strength which is above threshold and is locally spatially maximal over a 3x3 neighborhood.

2. Computing Feature Descriptors
Your goal is to write a function which takes as input an image location (corresponding to a keypoint), and outputs a descriptor (feature vector). You will implement two versions of this function. The function ComputeDescriptorPixels simply outputs the pixel values themselves, unwrapped in raster scan order to form a vector. The second function, ComputeDescriptorGradients, should implement the gradient histogram descriptor defined in Lowe 2004 (but without the scale adaptation).

The deliverable for this part is a separately-compiled executable that takes an input image and a pixel location and outputs the two descriptor vectors corresponding to that image position.

3. Matching Points
Given a list of feature descriptors from a pair of images, this problem involves writing a function ComputeMatches that outputs a list of matching features. The most basic matching function computes the SSD between a pair of descriptors and compares it to a threshold to determine if there is a match. This function can be improved upon substantially (see below). The deliverable for this part is your overall point matching application.

4. Evaluation
Experiment with the performance of your matcher. Find a pair of images for each of the following situations (3 pairs of images in total):

- The basic pixel-based descriptor produces good matches
- The pixel-based descriptor produces poor matches, but the gradient-based descriptor performs well.
- Both descriptors produce poor matches

It should be obvious that we want cases where matches are possible, but the algorithm does not produce the desired output. You must submit your example images, showing the matches found by your code.
5. Improvement and Quantitative Evaluation (Optional for students in 4495)
You must answer one question, make two improvements to the basic feature matcher from problems 1-3 and quantify the benefit:

1. How can you justify the choice of corner strength measure in Equation 1? Derive an expression for $S$ in terms of the eigenvalues of the Harris matrix. How will this measure behave in the cases of corner, edge, and texture-less region?

2. Implement the ratio test described in Lowe 2004 as an alternative to thresholded SSD for feature matching. The resulting score function is computed by dividing the score of the best match by the score of the second best match. Why do you think this might work better?

3. Implement one other variation of your solution approach in an attempt to improve the performance. A nontrivial goal would be to add scale invariance by means of the difference of Gaussian image pyramid, following Lowe 2004. But simpler goals based on changing some aspect of the corner strength measure, keypoint selection criteria, or feature descriptor are also fine. You must clearly state the change you are making and why you believe it will result in an improvement. You must also submit any additional source code.

4. Write a function to quantify the performance of your matcher. Given a set of image pairs, associated with each pair is the list of point matches produced by your program above, and a list of known ground truth matches that I will provide you with. Your function will compute the number of incorrect matches and return a scalar measure of performance for a given set of images. Use this to quantify the benefit of the ratio test and the variation you implemented in part 3.

5. Extra Credit
There are many opportunities for extra credit. If you implement scale invariance for problem 4, part 3, then you will get extra credit. You can try additional improvements to the matching algorithm. You can change the evaluation method to take into account the effect of the threshold on the matching score, by plotting an ROC curve. I will discuss some of these options in class.