EE 492/592 Introduction to Computer Vision - Fall 2012

Project 1 - Feature Detection and Matching[1]

Due: Friday, October 26

You need to provide a LaTeX document detailing your entire approach and illustrating your results on the provided benchmark images.

Professor:
Dr. Randy C. Hoover, Assistant Professor Dept. Electrical and Computer Engineering
EP 320, Email: randy.hoover@sdsmt.edu

Please read each step very carefully before proceeding. You are allowed to use a limited set of built-in Matlab functions for your project. Some functions you may find helpful are:

- `imread`
- `imshow`
- `imagesc`
- `fspecial`
- `conv2`

Note: you can always access Matlab's help system by typing `help conv2` at the command prompt.
As for simply finding a solution to these problems on-line and copying someone else's source code, this is **STRICTLY FORBIDDEN**.

Project Goals: Develop your own (MATLAB/C++) functions to compute image pyramids, extract features and edges, and correspond said features between successive image frames. The features that you detect and descriptors you develop should be at least minimally invariant to rotation, scale, and illumination changes. The project has been broken into different steps for your convenience.

Step 1: Feature extraction.

To help you display each feature in an image, you should first write a MATLAB function (named `ShowFeatures`) to display a red square depicting the dominant orientation. Your function should take three input parameters, the (x, y) location in the image where a feature has been detected, the scale that the feature was detected at, and a dominant orientation as determined by your feature descriptor. Your function will not return anything but rather overlay your feature locations on top of the image. An example of the output of such a function in action is illustrated in Fig. 1.

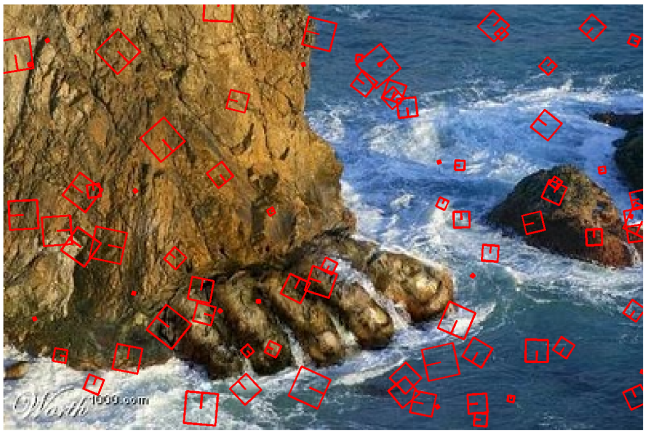

Figure 1: Example image showing features extracted at different scales and orientations.

[1] Adapted in part from Richard Szeliski

The first step in this project (after you have a proper display function working) is to detect interest points in an image. For this purpose, we will use the Harris corner detection method. We will proceed as follows: For each point $p = (x, y)$ in the image we consider a window of pixels around this point and compute the Harris matrix H for the point defined as

$$H = w(x, y) * \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix},$$

where the weighting function w should be chosen to be circularly symmetric for rotation invariance. A common choice is a 3×3 or 5×5 Gaussian mask similar to

$$w = \frac{1}{273} \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}.$$

Note that for each point $p = (x, y)$, the matrix H is only 2×2. Therefore, to find the interest points, we first need to compute the corner strength function

$$c(H) = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2} = \frac{\det(H)}{\mathrm{trace}(H)}$$

Once you've computed $c(H) \ \forall p \in I$, choose points where $c(H)$ is above a user defined threshold. You also want $c(H)$ to be a local maximum in at least a 3x3 neighborhood.

Step 2: Feature description.

Now that you've identified points of interest, the next step is to come up with a descriptor for the feature centered at each interest point. This descriptor will be the representation you'll use to compare features in different images to see if they match.

For starters, try using a small square window (say 5x5) as the feature descriptor. This should be very easy to implement and should work well when the images you're comparing are related by a translation.

Next, try implementing a better feature descriptor. You can define it however you want, but you should design it to be robust to changes in position, orientation, and illumination. You are welcome to use techniques described in lecture (e.g., detecting dominant orientations, using image pyramids), or come up with your own ideas.

Step 3: Feature matching.

Now that you've detected and described your features, the next step is to write code to match them, i.e., given a feature in one image, find the best matching feature in one or more other images. This part of the feature detection and matching component is mainly designed to help you test out your feature descriptor. You will implement a more sophisticated feature matching mechanism in the second project when you do image alignment for the mosaicing.

The simplest approach is the following: write a MATLAB function that compares two features and outputs a distance between them. For example, you could simply sum the absolute value of differences between the descriptor elements. You could then use this distance to compute the best match between a feature in one image and the set of features in another image by finding the one with the smallest distance. **You are required to implement the following two distance metrics and compare their performance:**

(a) Use a threshold on the match score. This is called the SSD distance.
(b) Compute (score of the best feature match)/(score of the second best feature match). This is called the "ratio test".

Deliverables:
First and foremost, you need to develop a web-page to display your solution. Your main page should simply have an index.html file that references all additional links. I will then link your pages to the course page so others can view your results. Your page should provide a brief overview with an illustration of your results. In addition, you will be required to write a document detailing the entire project, discussing your proposed approach to description, matching, and correspondence, and illustrating the success and failures in your approach.

As for the output, I will provide a set of "benchmark" images for you to test your algorithm on. At a minimum: For step 1 you are required to show images with your features being overlaid on them using your `ShowFeatures` function you wrote. For step 2 you are required to describe the feature descriptor you came up with and justify why you believe it is invariant to image transformations. Further, if your descriptor uses a dominant orientation direction (and it should), you should "zoom" in on a few of your feature locations to illustrate your orientation direction. Finally, you are required to display two (or more) images side-by-side with blue lines drawn between the features in each image showing that they do indeed correspond.

Extra Credit:
Here are some possible extra credit extensions (feel free to come up with your own variants to make your program more robust and efficient):

- Implement a version of adaptive non-maximum suppression.
- Implement automatic scale selection.
- Implement a scale invariant feature detector.
- Implement a better test for determining if two features match.
- Implement a better search method to speed up the matching process.

Some suggestions:
This is a big project. Therefore, I suggest that you divide and conquer. You should write modular code so you can reuse it for future projects (i.e., functionalize everything if possible). I would also recommend you write some helper functions to compute image gradients, derivatives of Gaussians, Laplacians, image pyramids, etc.