Programming Assignment #2

Announcement:	22 January 2019
Submission Deadline:	**05 February 2019**

DESCRIPTION

In this assignment, you will write code to detect discriminating features in an image and find the best matching features in other images. Because features should be reasonably invariant to translation, rotation (plus illumination and scale if you do the extra credit), you'll use a feature descriptor discussed during lecture and you'll evaluate its performance on a suite of benchmark images. As part of the extra credit you'll have the option of creating your own feature descriptors.

In the Project, you will apply your features to automatically stitch images into a panorama.

DESIGN A FEATURE DETECTOR

This involves three steps: feature detection, feature description, and feature matching.

Feature detection

In this step, you will identify points of interest in the image using the Harris corner detection method. The steps are as follows (see the lecture slides/readings for more details):

1. For each point in the image, consider a window of pixels around that point.
2. Compute the Harris matrix H for that point, defined as:

$$\sum_u \sum_v w(u,v) \begin{bmatrix} I_x(u,v)^2 & I_x(u,v)I_y(u,v) \\ I_x(u,v)I_y(u,v) & I_y(u,v)^2 \end{bmatrix} = \begin{bmatrix} \langle I_x^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I_y^2 \rangle \end{bmatrix}$$

where the summation is over all pixels *(u,v)* in the window. The weights *w(.,.)* should be chosen to be circularly symmetric (for rotation invariance). A common choice is to use a 3x3 or 5x5 Gaussian mask. Note that H is a 2x2 matrix. To find interest points, first compute the corner strength function (the "Harris operator").

$$c(H) = \frac{\text{determinant(H)}}{\text{trace(H)}}$$

Once you've computed ***c*** for every point in the image, choose only the points for which their ***c*** is above a user-defined threshold. You also want ***c*** to be a local maximum in at least a 3x3 neighborhood.

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 [rotational invariance should be taken into account in this step]. This descriptor will be the representation you'll use to compare features in different images to see if they match.

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. The simplest approach is the following: write a procedure 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.

Two distance measures you should implement are:

1. A threshold on the match score. This is called the SSD distance.
2. (score of the best feature match)/(score of the second best feature match). This is called the "ratio test".

Testing

Using the OpenCV API you can load in a set of images, view the detected features using cv::drawKeypoints(...), and visualize the feature matches that your algorithm computes using cv::drawMatches(...).

We are providing a set of benchmark images to be used to test the performance of your algorithm as a function of different types of controlled variation (i.e., rotation, scale, illumination, perspective, blurring).

Extra Credit [Undergraduate].

For Graduate students 1-2 are compulsory. [3-5 are extra credit]

Here is a list of suggestions for extending the program for extra credit. You are encouraged to come up with your own extensions as well. Feature detection and matching is an active area in computer vision - there are many interesting techniques you can attempt here based on techniques on the literature and on your own ideas.

1. Make your feature more contrast invariant. This was discussed in lecture.
2. Implement adaptive non-maximum suppression (MOPS paper).
3. Make your feature detector scale invariant.
4. Implement a method that outperforms the SSD ratio test for deciding if a feature is a valid match.

5. Use a fast search algorithm to speed up the matching process. You can use code from the web or write your own (with extra credit proportional to effort). Some possibilities in, rough order of difficulty: k-d trees (code available here), wavelet indexing, locality-sensitive hashing.

Submission (electronic submission through EAS only)

Please create a zip file containing your C/C++ code, a report and a readme text file (.txt). In the readme file document the features and functionality you have implemented, and anything else you want the grader to know i.e. control keys, keyboard/mouse shortcuts, etc. The report should contain results and a brief description for each of the steps you have implemented.

Additional Information

- Image sets provided for testing and experimentation here.

Credits

Adopted from an assignment developed by James Hays which was based on a similar project by Derek Hoiem.