Leonid Sigal

Associate Professor, University of British Columbia

Menu

Home | About | Biography | CV Students and Collaborators | Research | Teaching | CPSC 425 Winter 1, 2018 CPSC 532L Winter 2, 2017 CMU 15-869 Fall 2012 CMU 16-824 Spring 2012 CSCD18 Fall 2008 CSCD18 Fall 2007 Publications | Code and Data |

Contact

Dep. of Computer Science
University of British Columbia
ICCS 119
2366 Main Mall
Vancouver, B.C. V6T 1Z4
CANADA

Phone: 1-604-822-4368
Email: lsigal at cs.ubc.ca

Assignment 4: Local Invariant Features and RANSAC

Due: At the **start of class**, Wednesday, November 14, 2018.

The purpose of this assignment is to understand and to implement image matching using local invariant features.

The assignment

1. This assignment will make use of the SIFT features that are described in the paper Distinctive Image Features from Scale-Invariant Keypoints, by David Lowe. You don't need to read the whole paper, but you may wish to refer to parts of it to understand the approach.

2. The zip file containing sample images and code is hw4.zip. The unzipped directory, `assign`, contains a skeleton Python program `SIFTmatch.py` (which you can run by typing

```
python SIFTmatch.py
```

   at the command prompt). It also contains image files and their associated SIFT features that have been precomputed for some sample images. See the `README` file for further information, including how to run the sample program `keypoints` to compute SIFT features on images of your own choosing.

3. `(11 points)`

   The sample program, `SIFTmatch.py`, loads two images and their invariant keypoints and then draws 5 lines between randomly selected keypoints to show how matches can be displayed. Your task is to improve this program so that it identifies and displays correct matches by comparing the keypoint descriptor vectors. Note: Your program should find **all possible matches**, not just 5 as shown in this sample.

   The function `match` in the file `SIFTmatch.py` loads the invariant descriptor vectors into array *descriptors1* for the first image and *descriptors2* for the second. Each row corresponds to a descriptor vector. To select the best match for a vector from the first image, you should measure its angle to each vector from the second matrix. As the descriptor vectors are already normalized to have unit length, the angle between them is the inverse cosine (*math.acos(x)* function in Python) of the dot product of the vectors. The vector with the smallest angle is the nearest neighbor (i.e., the best match).

   However, many keypoints from one image will have no good match to the second image. To eliminate false matches, the most effective method is to compare the smallest (best) match angle to the second-best angle. A match should be selected only if this ratio is below a threshold. Hints: The Python function *sorted* can be used to find the two smallest values in a list. The list method *index* can then be used to determine the (original) indices of these sorted elements.

   Select a threshold that gives mostly good matches for the images ``scene'' and ``book.'' Try different thresholds so that only a small number of outliers are found (less than about 10). You can judge this by eye, as the outliers will usually produce matching lines at clearly different angles from the others. Print the box image showing the set of matches to the scene image for your suggested threshold value. Write a short description of the particular threshold value used, why you chose it and how important it was to get the value correct.

4. `(14 points)`

The second part of this assignment is to reduce the number of false matches by using RANSAC. For each RANSAC iteration you will select just one match at random, and then check all the other matches for consistency with it. Repeat the random selection 10 times and then select the largest consistent subset that was found.

To check other matches for consistency with the first match, you need to use the *keypoints1* and *keypoints2* arrays that are provided for each image. Each row provides 4 numbers for a keypoint specifying its location, scale, and orientation in the original image (see the function `ReadKeys` in the file `SIFTmatch.py` for details).

To check that one match is consistent with another, you should check that the change of orientation between the two keypoints of each match agrees within, say, 30 degrees. In other words, if the two keypoints for one match have a difference in orientation of 130 degrees, the second match should have an orientation difference between 100 and 160 degrees. Note that orientation is measured in radians, and that orientations are equal modulo 2 pi. Also, check that the change of scale agrees within plus or minus, say, 50%. For this assignment, we won't check consistency of location, as that is more difficult.

Try different values for the orientation and scale agreement (instead of using 30 degrees and 50% as mentioned above), and raise the matching threshold to get as many correct matches as possible while having only a few false matches. Try getting the best possible results on matching the difficult UBC ``library'' images.

Include resulting library images showing your best set of matches in a PDF. Aslo include in a PDF a paragraph summarizing the effects of consistency checking and the degree to which it allowed you to raise the matching threshold.

Deliverables

Hand in your functions (i.e., `*.py` files) or a Jupyter Notebook and PDF file showing results and answering questions. These must have sufficient comments for others to easily understand the code. Note you will have to hand in,

1. the box image showing the set of matches to the scene image for your suggested threshold value
2. the library images showing your best set of matches.

as part of the PDF. Both Python files and PDF should be submitted through Canvas

© 2018 Leonid Sigal | Design by Andreas Viklund