CS4495 Fall 2014 — Computer Vision
Problem Set 4: Harris, SIFT, RANSAC - Supplemental Document
UPDATED: October 21

This short document is to help you understand the relevant API for using SIFT libraries for this project. This is in no way the definitive guide, such documentation already exists, instead it is designed to help you quickly identify which functions you will need and which ones you can likely ignore. We give relevant pointers for both VLFEAT-MATLAB, VLFEAT-Python, VLFEAT-C, and OpenCV (both C and Python).

VLFEAT-MATLAB (Recommended)

VLFeat is an open source computer vision feature library that is actively maintained and has MATLAB interfaces that work on Linux, Windows, and OS X `http://www.vlfeat.org`.

Installation

Installation of the MATLAB toolbox binaries is straightforward and should be done following the instructions at: `http://www.vlfeat.org/install-matlab.html`.

Once installed you will need to run the following command in MATLAB:

```
run('VLFEATROOT/toolbox/vl_setup');
```

Here `VLFEATROOT` is the path to the directory where you installed the VLFeat library. This will need to be done each time you launch MATLAB, or you can embed it at the top of your MATLAB script.

Feature Extraction

A brief tutorial covering the usage of the VLFeat SIFT descriptor extractor is available at: `http://www.vlfeat.org/overview/sift.html`. This is also a good review of how SIFT works.

The most relevant section for this work is `http://www.vlfeat.org/overview/sift.html#tut.sift.custom` which describes extracting features at specified keypoints locations. Below we describe using the function `vl_sift`. You could also try `vl_siftdescriptor` which is described at `http://www.vlfeat.org/matlab/vl_siftdescriptor.html`.

You will be required to specify the matrix of keypoints, F_in, used as input to the function `vl_sift()`. Given your k detected keypoints, `F_i` is $4 \times k$ matrix, where each column specifies a keypoint's location, scale, and orientation. For keypoint j at location (x, y) and orientation θ its column vector would be:

$$f_j = \begin{bmatrix} x \\ y \\ 1.0 \\ \theta \end{bmatrix}$$

Since you are not doing any scale space extraction of your Harris corners, we will extract the SIFT features at scale equal to 1.0. But see the note below about smoothing the image first.

Once the matrix F_in is defined you can extract the SIFT descriptors located at the points in F_in on image I with:

```
[F_out, D_out] = vl_sift(I, 'frames', F_in)
```

Note: While we are not explicitly playing with scale, we are of course computing the Harris corners over a window and you are also using a smoothed image to compute the gradient. You might use the smoothed version of tyour image — smoothed by the same amount you smoothed the image to compute the gradient — as the input to `vl_sift` or `vl_siftdescriptor` function.

Feature Matching

In order to match points between two images you will use the function `vl_ubcmatch()`. This function takes as input two sets of SIFT descriptors `D_a` and `D_b`. It gives as output a $2 \times k$ matrix `M` containing a list of indexes for corresponding descriptors from `D_a` and `D_b`. Use as follows:

```
M = vl_ubcmatch(D_a, D_b)
```

You can then access the keypoints corresponding to the ith match with something such as `ka1 = F_a(:,M(1,i))` and `kb1 = F_b(:,M(2,i))`.

OpenCV - Python (Recommended)

OpenCV has a non-free SIFT implementation in Python. You can find a brief tutorial here: `http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html`.

Installation

The installation of OpenCV - Python has been covered in Lecture 2. Windows users check out: `http://docs.opencv.org/trunk/doc/py_tutorials/py_setup/py_setup_in_windows/py_setup_in_windows.html#install-opencv-python-in-windows`. Linux users try

```
sudo apt-get install python-opencv.
```

Mac users, the easy way is to use Homebrew to install OpenCV and manually setup the path.

Feature Matching

The main classes you will be using are the SIFT class documented above. You will be using the `cv2.KeyPoint` class `http://docs.opencv.org/modules/features2d/doc/common_interfaces_of_feature_detectors.html#keypoint-keypoint` to define the keypoint location of your Harris corners for input to the SIFT feature extractor.

For each detected Harris keypoint you will create a `cv2.KeyPoint` instance, where you set the values of `x` and `y` appropriately to the corner location, `_size` to the size of the local region, and the value of `_angle` to the dominant orientation computed for the corner. Additionally you will set the value of `_octave` to 0, since all points were located at the full scale version of the image. Here is an example.

```
point = cv2.KeyPoint(x=10, y=10, _size = 3, _angle = 90, _octave=0)
```

You will need to put the `cv2.KeyPoint` instances for all of your data into a list.

You will need to create an instance of the class `cv2.SIFT`. This is simply done by:

```
sift = cv2.SIFT().
```

Extracting the SIFT descriptors then requires you to call the `SIFT.compute()` function:

```
points,descriptors = sift.compute(I,points)
```

Here `I` is the image to compute the descriptors of, `points` is the list of keypoints and you are returned the descriptors for the points in `descriptors`. `descriptors` is a NumPy array with the jth row corresponds to the 128 element SIFT feature extracted at the location of `points[j]`.

Feature Extraction

To find the putative matches you will use the class `cv2.BFMatcher`: `http://docs.opencv.org/modules/features2d/doc/common_interfaces_of_descriptor_matchers.html?highlight=match#BruteForceMatcher`.

You will need to create an instance of this class by

```
bfm = cv2.BFMatcher()
```

If you have an instance named `bfm` you can compute matches for descriptors of `d_a` and `d_b` with

```
matches = bfm.match(d_a, d_b)
```

This returns by reference `matches` which is a list of `cv2.DMatch`. The class `cv2.DMatch` is defined at `http://docs.opencv.org/modules/features2d/doc/common_interfaces_of_descriptor_matchers.html#DMatch`.

For a given match, the keypoint index from set A will be the value in `dmatch.queryIdx` and that from set B will be at `dmatch.trainIdx`. If you are interested, it is also easy to implement the ratio test in the original SIFT paper. Here is an example.

`http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html`.

VLFEAT-PYTHON

New this year we're also mentioning VLFeat in Python. Besides the VLFeat web site mentioned above, see: `//https://pypi.python.org/pypi/pyvlfeat`. The only difficulty you might have is that according to the documentation, the matching function `vl_ubcmatch()` is not provided. So you may need to either write your own (pretty easy), find another on one the web, or wrap the C version (see below).

As we get more experience using Python on VLFeat (and you let us know what works) we'll update this supplemental document. Currently, we recommend OpenCV - Python combination for all Python users.

VLFEAT-C

All the significant functions in the VLFeat library are C callable because that's how MATLAB calls them. The web site has pretty good documentation. This might be an easier way for C folks to go than the OpenCV route below only because OpenCV has moved away from SIFT because of legal reasons.

OpenCV - C

There are a few different ways of using SIFT in OpenCV, we will be giving examples on how to use one of those interfaces.

Installation

If you have OpenCV 2.3 or 2.4 installed then you should be fine. For 2.3 you will need to import the header `<opencv2/features2d/features2d.hpp>`. If you have 2.4, it has been moved to a module called "non-free" since it has patent limitations. Check out: `http://docs.opencv.org/modules/nonfree/doc/nonfree.html`.

Feature Extraction

The main classes you will be using are the SIFT class documented above. You will be using the `cv::KeyPoint` class `http://docs.opencv.org/modules/features2d/doc/common_interfaces_of_feature_detectors.html?highlight=keypoint#KeyPoint` to define the keypoint location of your Harris corners for input to the SIFT feature extractor.

For each detected Harris keypoint you will create a `cv::KeyPoint` instance, where you set the values of `pt.x` and `pt.y` appropriately to the corner location and the value of `angle` to the dominant orientation computed for the corner. Additionally you will set the value of `octave` to 0, since all points were located at the full scale version of the image. You will need to put the `cv::KeyPoint` instances for all of your data into a `std::vector<cv::KeyPoint>` instance.

You will need to create an instance of the class `SIFT`. In 2.3 the following parameters worked:

```
SIFT sift(1.0, true, false).
```

This will allow you to extract SIFT features at the default scale without recomputing the orientations you find above. The 2.4 constructor documentation seems to imply that this has changed - see `http://docs.opencv.org/modules/nonfree/doc/feature_detection.html?highlight=sift#sift`

Extracting the SIFT descriptors then requires you to run the operator() as:

```
sift(I, cv::Mat(), points, descriptors, true)
```

Here `I` is the image to compute the descriptors of, points is the vector of keypoints and you are returned the descriptors for the points in `descriptors`. The jth row of `descriptors` corresponds to the 128 element SIFT feature extracted at the location of `points[j]`.

NOTE: if you do not set the last parameter to `true`, then new keypoints will be detected and the descriptors will not be extracted at your corners.

Feature Matching

To find the putative matches you will use the class `cv::BFMatcher`: `http://docs.opencv.org/modules/features2d/doc/common_interfaces_of_descriptor_matchers.html?highlight=match#BruteForceMatcher`.

You will need to create an instance of this class and then call the inherited method `match()`. If you have an instance named `bfm` you can compute matches for descriptors of `d_a` and `d_b` with

```
bfm.match(d_a, d_b, matches)
```

This returns by reference `matches` which is instance of the form `std::vector<cv::DMatch>`. The class `cv::DMatch` is defined at `http://docs.opencv.org/modules/features2d/doc/common_interfaces_of_descriptor_matchers.html#dmatch`.

For a given match, the keypoint index from set A will be the value in `dmatch.queryIdx` and that from set B will be at `dmatch.trainIdx`.