CS4495 Fall 2011 — Computer Vision
Project 4: Alignment - Supplemental Document

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 MATLAB and OpenCV.

MATLAB

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`. 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.

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.

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)
```

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

There are a few different ways of using SIFT in OpenCV, we will be giving examples on how to use one of those interfaces. You can try other ones, but be warned that you may get different results, which in turn may affect your grade.

Installation

If you have OpenCV 2.3 installed then you will be fine. You will need to import the header `<opencv2/features2d/features2d.hpp>`.

Feature Extraction

The main classes you will be using are the SIFT class documented at `http://opencv.itseez.com/modules/features2d/doc/feature_detection_and_description.html#SIFT`. You will be using the `cv::KeyPoint` class `http://opencv.itseez.com/modules/features2d/doc/common_interfaces_of_feature_detectors.html#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` with parameters as follows:

```
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.

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::BruteForceMatcher`: `http://opencv.itseez.com/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://opencv.itseez.com/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`.