COSC470 2013 Assignment 1

Due: Friday 23rd August, 4pm

Marks: 30% of your final grade

Overview

The first assignment is based on material from the first 5 weeks of lectures and covers feature detection, description, and matching. Experimental evaluation is also an important part of this assignment. There are three main components to the assignment:

1. Implement a feature detector – the Harris-Stephens or Shi-Tomasi detector is the default choice, but if you want to implement some other detector then talk to me about that.
2. Implement a feature descriptor – normalised image patches or colour histograms would be appropriate, but again other choices are possible by agreement.
3. Compare your detector and descriptor to the SIFT implementation in OpenCV. This part of the assignment is focussed on the experimental design and evaluation, rather than the performance of your chosen methodss – there is no expectation that your implementations will out-perform SIFT.

Supporting Code

A basic framework for feature detection and matching is available on the departmental machines in the file `~steven/Public/COSC470/assignment1.tar` – you need to fill in the details of the specified functions in the `MyDetector` and `MyDescriptor` classes to complete parts 1 and 2. These classes provide a simplified version of OpenCV's feature detection and description interfaces. They inherit from the `SimpleDetector` and `SimpleDescriptor` abstract base classes, and wrappers around OpenCV's SIFT detector/descriptor are provided in this framework as well as `SIFTDetector` and `SIFTDescriptor`. You will also need to modify other parts of the code, depending on your experimental design for part 3.

1. Implement a Feature Detector

For your feature detector you need to provide an implementation for `MyDetector::detect`. This method takes in an `image` and a vector of `keypoints`. The `keypoints` are passed as a non-const reference, and so can be modified. You need to provide code that processes the `image` and fills the vector of `keypoints`.

A sample implementation is provided as guidance, which implements a naïve and not very useful 'feature' detector. Features are produced on a regular 10-pixel grid over the image. This should be replaced with a more realistic feature detector, such as that of Harris and Stephens or Shi and Tomasi, as discussed in lectures. OpenCV `KeyPoints` have a number of different properties, but the main one that you will need to provide is the location. You may want to provide other information depending on your choice of descriptor – for example, SIFT descriptors use the size and orientation of the feature. More information about OpenCV's `KeyPoints` can be found in the online documentation.

2. Implement a Feature Descriptor

For your feature descriptor you need to provide an implementation of `MyDescriptor.compute`. This method takes in an `image`, a list of `keypoints`, and a `descriptor`. The `descriptor` is a cv::Mat object and should have type `CV_32F` (which corresponds to the `float` datatype on most architectures). The `descriptor` should be modified by your code.

The descriptors are stored in a `cv::Mat` with one row per keypoint, and a number of columns equal to the descriptor length. For SIFT descriptor the descriptor length is 128. Once again a sample (but not useful) descriptor is provided. This simply uses the position of the feature as its descriptor, so has length 2. This should be replaced with a more meaningful descriptor such as a normalised image patch or a greyscale, colour, or gradient histogram.

3. Compare Your Detector/Descriptor with SIFT

Finally you should design, conduct, and evaluate an experiment to compare your feature detector/descriptor to OpenCV's SIFT implementation. You should consider what images to use for this experiment – the commonly use data set of Mikolajczyk et al. is provided in ~steven/Public/COSC470/Data. This data set contains ground truth homographies which can be used to determine which matches are correct. The framework code provided shows how this can be done by identifying and drawing matches which are consistent with the provided homography to within some threshold. The sample program takes as input two images and the homography between them, specified as command line arguments:

```
assignment1 ~steven/Public/COSC470/Data/Graffiti/img1.ppm
            ~steven/Public/COSC470/Data/Graffiti/img2.ppm
            ~steven/Public/COSC470/Graffiti/H1to2p
```

Use of Libraries

You should make use of the OpenCV and Eigen libraries as appropriate. For your feature detector/descriptor implementations you should **not** use the libraries to solve the core problems – part of the purpose of the assignment is to give you experience in implementing computer vision algorithms, not just applying them to a set of images. As a general guideline the following uses are appropriate:

- Using the Eigen or OpenCV libraries to carry out matrix operations, including high-level tasks such as eigenvalue computations or matrix decompositions
- Using fundamental OpenCV data structures and operators. Generic functions such as `cv::filter2D` are perfectly fine, as are specific filters such as `cv::GaussianBlur` or `cv::Sobel`.
- Using OpenCV's display routines such as `cv::drawMatches` to visualise your results.

It would not, however, be appropriate to use `cv::goodFeaturesToTrack` as your feature detector or `cv::OrbDescriptorExtractor` as your descriptor

Marking Scheme

10 marks will be assigned to each of parts 1, 2, and 3. You will be marked on the quality of your code, and correctness and clarity are the main concerns. While more efficient methods are to be preferred, you should worry primarily about making sure that your code does what it should, and

ensuring that this clear to someone inspecting the code. Marks will also be given according to the complexity and validity of your chosen methods. Your code should compile with `make` on the lab Macs, and any additional libraries that are needed should be included in your submission.

For part 3 you will be marked on the design of your experiment and your evaluation of the experimental results. Considerations for this part of the assignment include the choice of test cases, the metrics used to evaluate the methods, and the evaluation and discussion of the result.

Deliverables

The deliverables for this assignment are your code and a written report.

Your code should include all of the files needed for your program to run, other than the libraries provided in `~steven/Public/COSC470`. It should also include a makefile to build an executable on the laboratory machines.

The report should briefly describe the detector and descriptor that you have implemented, with reference to the literature when appropriate. You should outline how your implementation works, and any special parameters or instructions for running your code. The report should explain clearly the experimental method you used to evaluate your detector and descriptor, the results of that evaluation, and an analysis of your methods' performance compared to SIFT features. It is expected that your report should be approximately 3-5 pages long, with the majority of the space dedicated to part 3 of the assignment.

You should email your code (preferably in a common archive format such as ZIP or a compressed tar) and a PDF copy of the report to steven@cs.otago.ac.nz before the due date. Late submissions will be penalised at the rate of 10% of the total available marks per working day. Extensions to the deadline may be granted where appropriate. The usual university regulations relating to plagiarism (www.otago.ac.nz/study/plagiarism) apply, and any work you submit for assessment must be your own.