COSC470 2013 Assignment 2

Due: Friday 4th October, 4pm

Marks: 30% of your final grade

Overview

In this assignment you will compute the relationships between a set of images in order to construct a larger mosaic image. The relationships in this case are homographies, and can be computed from feature matches using the Direct Linear Transform (DLT) algorithm. However, there are generally outliers in the feature matches, so a Random Sample and Consensus (RANSAC) approach is usually required. These algorithms are outlined in Lecture 6, and the mosaicing process summarised in Lecture 7.

There are three main tasks associated with the assignment, which are detailed below. For each of these there is a 'sufficient' level of work which can earn up to 80% of the available marks and a bit extra for the last 20%. You are encouraged to get the 'sufficient' portions working first.

Supporting Code

Supporting code is available in `~steven/Public/COSC470/assignment2.tar` on the departmental servers. This code is a basic mosaicing program that takes as input (command line parameters) a set of images to mosaic and an output filename. For example:

```
assignment2 image1.png image2.png image3.png mosaic.png
```

will create a mosaic from `image1.png`, `image2.png`, and `image3.png` and write the result to `mosaic.png`.

The part of the code that you need to replace is the function `computeHomography`. In the sample code this uses OpenCV's `findHomography` function. You should replace this with function(s) that compute the homography with the following techniques:

- The basic DLT algorithm
- A RANSAC version of the algorithm

Note that the RANSAC algorithm will need to use the DLT algorithm for each sample, and for the final solution.

Sample data is available in the `~steven/Public/COSC470/MosaicData/` directory, which contains 9 images named `IMG_01.JPG` to `IMG_09.JPG`. You can also generate your own data with any digital camera – just remember to restrict the camera motion to rotation, or to use a planar scene.

Implement the DLT

You should first implement the direct linear transform as described in lectures. This should be used as a replacement for the implementation of `computeHomography` provided. A basic implementation is sufficient, but for full marks you should consider normalising the input data in order to make the method more robust.

1. Implement RANSAC

Once you have a DLT implementation you should implement a RANSAC homography estimation technique. Again a basic implementation with a fixed number of iterations is sufficient, but for full marks you should determine the number of iterations from the current largest inlier set and evaluate different inlier thresholds.

2. Compare the Two Methods

Finally you should conduct a comparison of mosaicing with and without RANSAC. If you have implemented normalisation or automatic RANSAC termination then these should be evaluated as well. A visual comparison of the mosaicing results is sufficient, but for full marks some sort of quantitative (numeric) evaluation of accuracy is expected. There are a number of ways that you could do this, but here are a few possibilities:

- You could compare the overlapping pixels of the aligned images, and examine their differences. A good homography should align the images well, leading to a small image difference.
- You could evaluate the alignment of the feature points when matched with the homography. A good homography should align the features well and so give small errors.
- You could compare different orderings of images and see how consistent the homographies are. Is the homography computed from images 1 and 3 the same as the combination of the homographies from 1 to 2 and 2 to 3?

Note that OpenCV provides the ability to find homographies with and without RANSAC, so you can conduct the comparison of methods using those implementations, even if you do not complete all of the implementation for parts 1 and 2.

Use of Libraries

As with assignment 1, you should make use of the OpenCV and Eigen libraries as appropriate. However, you need to implement the DLT and RANSAC algorithms yourself – 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. The Eigen library provides routines for the singular value decomposition, which is required for the DLT. To use these you will need to include the header

```
#include <Eigen/SVD>
```

And to compute the SVD of double precision matrix, A, you would use

```
Eigen::JacobiSVD<Eigen::MatrixXd> svd(A,
    Eigen::ComputeFullV | Eigen::ComputeFullU);
```

The flags at the end tell Eigen that you want to get the matrices U and V out, not just the singular values. You can then extract the matrices U and V as

```
svd.matrixU()
```

and

```
svd.matrixV()
```

and get a vector of singular values as

```
svd.singularValues()
```

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. 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 tests and your evaluation of the 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 methods that you have implemented. You should outline how your implementation works, and any special parameters or instructions for running your code. The report should explain clearly the tests that you conducted, and how you evaluated them. 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. Note that emailing ZIP files from external accounts can run afoul of the University's spam filters, so you may wish to change the extension (the spam filters aren't very smart) and let me know what sort of archive it is in the email. If in doubt, send me a separate

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.