Advanced Vision Practical 3
"Image Mosaic-ing"

Tim Lukins
School of Informatics

March 2008

Abstract

This describes the third, and last, assignment for assessment on Advanced Vision. The goal is to use a recent technique for identifying unique matching feature points within two images. You will then investigating the ways to establish the homographies between the images. Finally you then use this knowledge to "stitch" or "mosaic" the images together to form larger images.

Task Background

Finding robust and reliable feature points in an image that are invariant to scale, orientation, and lighting is a fundamental task in Computer Vision. Resolving which features are the same between images, or frames of video, allows us to define the geometric relationships that underpin the image formation process and motion of the camera. Exploiting such knowledge gives us the foundations for several higher-level vision tasks: such as structure-from-motion, SLAM, stereo reconstruction, and image mosaic-ing.

One of the most recent and advanced features descriptor is the Scale Invariant Feature Transform (SIFT), introduced by David Lowe (*D. Lowe, "Object recognition from local scale-invariant features". ICCV 1999.*). This provides a readily available fast solution to finding candidate points that are shared between two images. These can then be used to estimate a homography (as a 3×3 matrix) that describes the linear mapping between the set of points $p_i \leftrightarrow p'_i$ for each point i as $\mathtt{H}p_i = p'_i$[1].

This matrix $\mathtt{H}$ represents a 2D projective transformation between the established sets of matching points in each image. It can be estimated by employing a straight-forward linear algorithm - DLT - which is the approach introduced for homographies in the IVR course (see: `http://www.inf.ed.ac.uk/teaching/courses/ivr/` code for flat part recognition). Alternatively, for the case of outliers, a more robust iterative approach uses RANSAC (as presented in lectures) to estimate the homography:

1. Find 4 points randomly whose SIFT descriptors match.
2. Estimate the homography.
3. Verify the homography by counting the number of SIFT points paired by the homography and with matching descriptors.
4. Repeat 1-3 a given number of times until success or failure.
5. Perform the DLT with all matched points to improve accuracy.

[1] This is a projective relation using homogeneous co-ordinates - i.e. $p = [x, y, 1]^\mathsf{T}$.

Having established H it is possible to use it to *warp* one image into the co-ordinate frame of another. This information then allows us to re-project the images onto a single projective plane to recreate complete mosaics. If all the images share a planar surface (or indeed lie on the plane at infinity) and are taken from roughly the same position this should produce well-matched results, since we can ignore some of the more complex aspects involving camera properties. For a further explanation see: `http://pages.cs.wisc.edu/~dyer/ai-qual/szeliski-tr06.pdf`

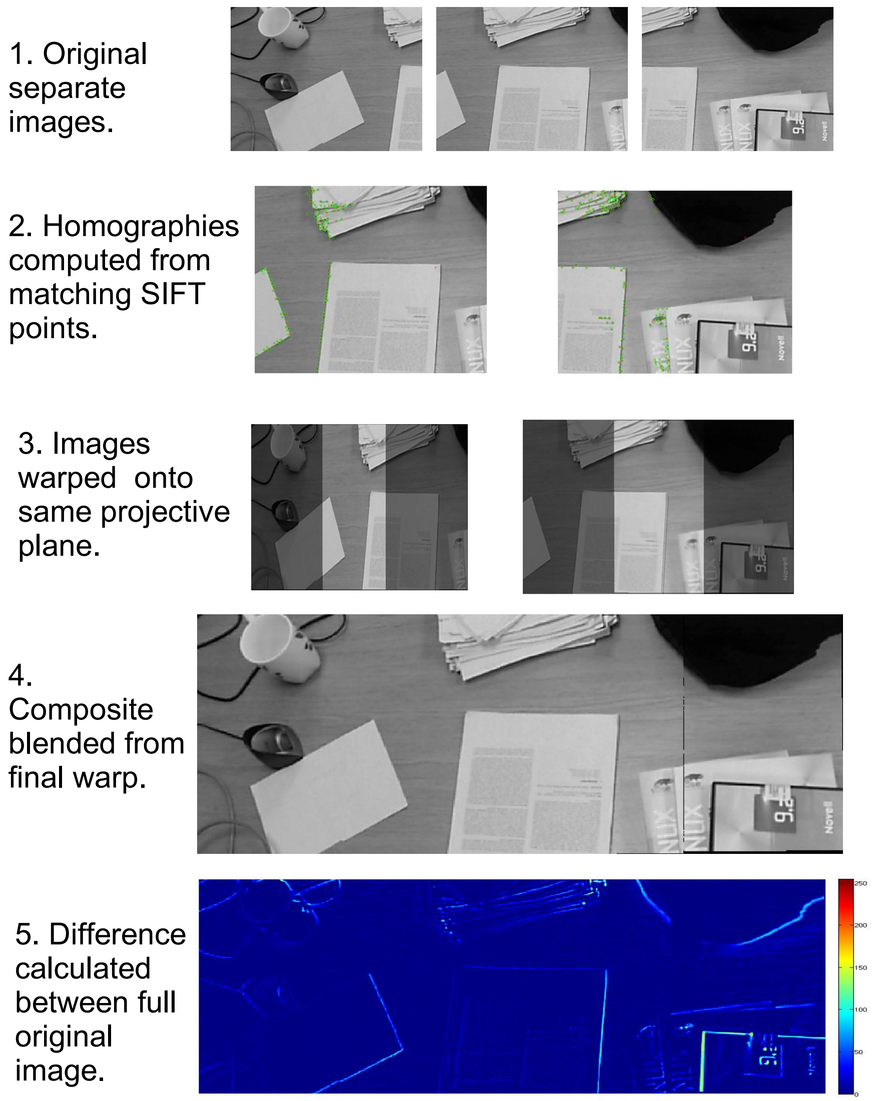
1. Original separate images.
2. Homographies computed from matching SIFT points.
3. Images warped onto same projective plane.
4. Composite blended from final warp.
5. Difference calculated between full original image.
250
200
150
100
50
0

Figure 1: The overall goal of the practical - from multiple images to a single panorama.

Once the pixel data has been projected into the same image co-ordinate frame, there is then the final task of deciding which values to keep or amalgamate. These are the issues of *compositing* - particularly with regard to the selection and/or blending the data together such that it has a seamless appearance. The final results should be indistinguishable from an original wide angle image. If we have a suitable test image - from which the individual single images are derived - then we can quantifiably measure the difference error of the reconstruction. For another example, see the following: `http://www.cs.ubc.ca/~lowe/papers/07brown.pdf`.

Specific Tasks

You will work in teams of 2 on these tasks (in a different pairing from the last practical). The practical is in two parts. In the first part you will familiarise yourself and experiment with established means of calculating homographies. In the second part you will combine this knowledge and use it to warp and blend separate images together. You will produce a report describing your experiences, including your final code listing along with example input and results showing evidence of empirical evaluation. You will also give a live demo of your solution.

Part I

1. Download the first (and official) implementation of SIFT from David Lowe's web site here:
`http://www.cs.ubc.ca/~lowe/keypoints/`
Unzip the siftDemoV4.zip file. In it you will find the pre-compiled implementation of the SIFT algorithm (for both Linux and Windows) - along with the MATLAB wrapper function *sift.m*. Have a look at the example **match.m** and run it to see the algorithm in operation. Understand how the matching is performed by comparing the feature vectors. Notice how some images have the obviously incorrectly matched outliers. In this phase of the practical we will test matching performance. We do this by computing the SIFT features for a known object and searching for it in a new test scene.
Use one of the web cameras in the lab to capture a "training" images (for example a book or CD cover, or something else flat). Then capture a set of "test" scene images with the object somewhere in view. If you have problems with lens distortion - try using the inner half of the image. Experiment with varying degrees of rotation and perspective effects/distance, plus different lighting conditions and levels of occlusions. What do you notice about the differences in matching between planar and non-planar surfaces? Is the matching perfect? Include figures of example output in your answer. Compute performance statistics on the basis of your varying examples (shown as a graph or table).
2. Combine the example code from the *match.m* example above with the other example code from Peter Kovesi's web-site here:
`http://www.csse.uwa.edu.au/~pk/research/matlabfns/`
First, download and use **homograpy2d.m** (you will also need to download *normalise2dpts.m* as well). This provides a standard implementation of the Direct Linear Transformation algorithm with enhanced estimation by first normalising the points to the centroid with mean distance $\sqrt{2}$. See:
`http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/FUSIELLO4/tutorial.html`
Take another set of test images (perhaps along the top of a nearby desk) using the web-cam. Use your combined code to find the points, calculate the homography, then apply the estimated matrix to map your "test" image points on top of the "training" image points. Plot both sets of points (remembering to correct the scaling of the homogenous transform) in the "training" image. Produce figures of your output and comment on the results. Does this simple estimation of the homography work in all cases? Are all the points matched? Try commenting out the normalisation code and observing the effects.
3. Next, download from Kovesi's site the function **ransacfithomography.m**. You will also need to download: *ransac.m, iscolinear.m* and *hnormalise.m.* These provide an alternative, and more robust solution to homography estimation by using the RANSAC algorithm to reject outliers. Compare the results of this approach - especially when you know you have a few mismatched points between the images. Does it perform better that just the DLT algorithm? Perhaps add a random amount of noise (using the *rand()*) function to your points. How does the RANSAC algorithm cope? Illustrate your results - perhaps with a graph of the number of inliers versus the amount of noise/motion.

Part II

By this stage you should have everything you need to be able to stitch **multiple** images together into a single mosaic. First try creating a simple test set by cutting up a larger image into smaller pieces. You can first safely assume the ordering - but think about how you would make it work for a random collection. Try varying degrees of overlap between images (33% – 50%) and establish the homographies (using your knowledge acquired in Part I).

You then need to produce a function that takes an image and applies the calculated homography to produce a warped version *in the image co-ordinates of the other image.* The standard way to accomplish this is to first project the four corner pixels of the image in question to establish the bounds of the warp, and then use the `interp2()` function to interpolate the values. Remember again to accommodate for the homogeneous scaling.

Following this, you will then have to create a new, larger, image array to copy the result into - along with the other "anchored" image. This is where you must consider how you are to blend the two sets of data together where they overlap. Again, you should write another function that takes the two images and outputs a seamless composite. In your report describe how you accomplish this - particularly if you attempt different approaches. You might have to consider a suitable approach to cropping out `null` values resulting from the transforms.

For verification of your final solution, you should consult the first reference given in the introduction (the paper by Szeliski) which discusses a variety of error metrics that work on images. You are not trying to directly estimate motion in this way, but should use the *Sum Square Difference* (SSD) between two images to quantifiably measure the similarity between the reconstruction and the original image. There may be some degree of global translation required to adjust the alignment between the two. Simply plotting the difference between the images can reveal a considerable amount of detail. Furthermore, looking at the pixel variance (i.e. distribution of matched pixel values) is another useful metric that can also be performed for local neighbourhoods.

Next, attempt to use the same code to stitch together a panorama of the lab captured using a web-cam. Does this work? How are the camera properties - especially the lens distortion - affecting the process? Produce some output images of your results.

Extra marks will be reserved for evidence of any of the following enhancements (you should include a section in your report for each one attempted outlining how you try to solve it):

- Dealing with colour images (especially blending).
- Handling un-ordered sets of images.
- Bundle adjustment.
- Coping with lens distortion effectively.
- Using a cylindrical or spherical re-projection for full panoramas.
- Novel and innovative uses of your final algorithm (nice results).

Credit will be given where quantifiable results show progressive improvements to the algorithm (i.e. measurable reductions in SSD).

Assignment Submission

Each team should present a **single** PDF file that includes your answers for Part I, along with a description of your solution to Part II (with suitable examples/graphs plus your source code listing at the end - you do not need to include print-outs of the downloaded functions unless you modify them).

This file should be submitted electronically by **9am Wednesday March 19th, 2008**. The on-line submission line is:

```
submit msc av-5 3 FILE
```

or

```
submit ai4 av-4 3 FILE
```

where FILE is the name of your file.

The assignment is estimated to take 10-15 hours work for the assignment, resulting in a $5-7$ page report. The assignment will be marked as follows:

Issue	Percentage
1. Part I	25
2. Part II	25
3. Demonstration	40
4. Clarity of code and Report	10

Assessment of parts I and II will include algorithm choices and use of evidence to show performance.

In addition, each team will also have to demonstrate their mosaic-ing algorithm on the afternoon of **Wednesday the 19th of March** (the same day you hand in) from 2-6pm. You will be allocated a time slot and will be asked to:

1. Take 3 overlapping images of a cluttered desktop using a web-cam, plus a 4th view that encompasses the first 3.
2. Draw the image borders of the first 3 images on the 4th.
3. Compute a mosaic of the first 3 images.
4. (Finally showcase any additional "neat" aspects to your solution.)