CS 766 HW3

Steve Jackson (sjackso@cs)

2 November 2006

Methods

My implementation is broken into several steps, each step implemented by its own C++ or Matlab program. Source code for each program is available in my handin directory. The breakdown is as follows.

- **tracker.cpp**: A program that accepts n images and uses the KLT feature point tracker to determine the motion between consecutive pairs of images. The output is $n-1$ binary "feature table" files that contain the pixel locations of tracked feature points between images.
- **homography.cpp**: A program that accepts the output of the tracker program and determines which points from each image pair should be used to compute homographies. In particular, the homography program uses the RANSAC algorithm to randomly choose sets of 4 tracked points, then computes the quality of the resulting planar transformation based on how many points are correctly mapped by the transformation. The output is one text file for each pair of image. Each text file contains the set of best point mappings determined by the RANSAC algorithm.
- **leastsq_transform.m**: A Matlab program that accepts the output of the homography program and generates a 3x3 projective matrix that maps the first image into the coordinate frame of the second. Since the homography program outputs all the point correspondences that are found to be "good," the computation of this projective matrix is an overconstrained problem, so the projection is computed via least squares.
- **homography_convert.m**: Given the $n-1$ image-to-image projections computed by the leastsq_transform function, this Matlab function generates n projection matrices that directly map the coordinates of each of the n images into the coordinate from of one of those images. This is used to create a single matrix for each image that will map that image into the coordinates of an image in the middle of the image sequence.
- **make_mosaic.m**: The final step. This program accepts the n images and the n homographies computed by the homography_convert program and creates a mosaic of the images.

Image Sequences

I made mosaics of two video image sets. The first is the Monona Terrace set provided in the CS766 public directory. The second is a home video that pans slowly over a long line of aluminum cans. In this latter image set, the optical center of the camera is moving, but its motion is (approximately) only along a line. I chose this set as an experiment to see how well the algorithm would stand up to one-dimensional camera motion.

Monona Terrace

There are 27 images in this sequence, each of size 1024 x 768. The parameters used were as follows.

- Number of KLT features tracked between each image pair: 1000
- Modifications to KLT default parameters: The `KLT_TrackingContext` structures was modified with the following code.

```
tc->smoothBeforeSelecting = TRUE;
tc->window_width = 13;
tc->window_height = 13;
KLTUpdateTCBorder(tc);
```

- Number of RANSAC trials for each image pair: 40000
- Distance tolerance for a point mapping to be considered "good" by the RANSAC algorithm: 0.5

Parameters were chosen based on experimentation– using fewer features or fewer RANSAC trials tended to produce poorer results.

Figure 1: Monona terrace sequence, third frame

Figure 2: Monona terrace sequence, nineteenth frame

Figure 3: Monona terrace sequence, final mosaic (2789 x 1000 pixels)

Soda Cans

There are 348 images in this sequence, each of size 320 x 240. The parameters used were as follows.

- Number of KLT features tracked between each image pair: 100
- Modifications to KLT default parameters: The `KLT_TrackingContext` structures was modified with the following code.

  ```
  tc->smoothBeforeSelecting = TRUE;
  ```

- Number of RANSAC trials for each image pair: 40000
- Distance tolerance for a point mapping to be considered "good" by the RANSAC algorithm: 0.08
- In the final step, the mosaic was made with data from every third image in the sequence.

Parameters were chosen based on experimentation. Note that the resulting mosaic image is fairly poor, for reasons discussed below. I will continue experimenting with different parameters to the algorithm to see if I can produce a better output– if I manage to create an especially good one, I will send it along at a later date.

Figure 4: Soda sequence, tenth frame

The results for the cans sequence are clearly poor. Because there is some significant motion parallax in the image sequence, I expected the regions beneath the tops of the cans to be blurred; however, the tops of the cans themselves ought to line up more crisply. Part of the problem may be the unsteadiness of the camera in the image sequence. Floating point error could be another issue, particularly since there are so many images in the sequence.

Figure 5: Soda sequence, 120th frame

Figure 6: Soda sequence, 280th frame

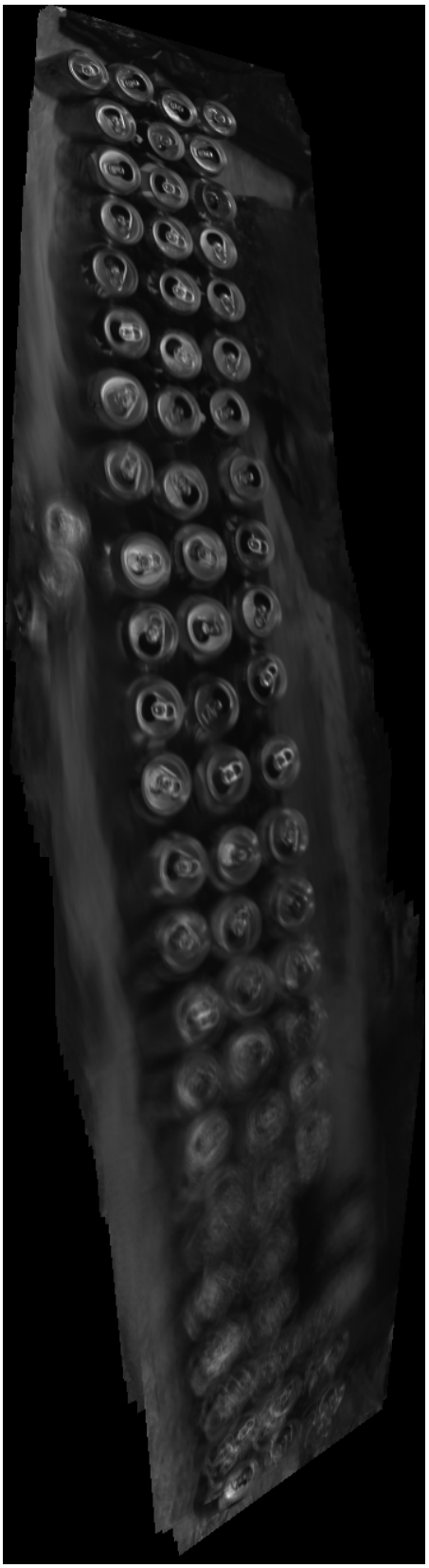

Figure 7: Soda sequence, final mosaic (1312x355 pixels)