CS482
COMPUTER VISION

Project Report

Kevin St.Andrie
(kstandri@masonlive.gmu.edu)
Travis Hile
Martin Taheri
Jeremiah Howdeshell
Chris Sahno

May 3, 2012

1. Objective

Our stated objective was to investigate and understand the methods used to develop panoramic and composite images and to adapt these techniques into a portable code library which could conceivably be run on a smartphone (such as an Android device).

1.1. **Relevance to cs482.** We are learning how to use SURF, Affine Transforms and RANSAC methods to determine keypoint matches between images and to position overlapping images to stitch together.

2. Methodology

2.1. **Dataset Acquisition.** The first step was to acquire our dataset. Using a 10 megapixel digital camera and a tripod, we captured 96 images. By using the same camera and focusing on one object for each set, we maximized the number of matching features from which to build a composite image.

- 40 images were taken at 10° increments from the same fixed position using the tripod along the path between the Research building and the Arts building.
- 27 images were taken of the Engineering building from different vantage points, in order to generate parallax errors issues.
- 16 images were taken of the Research building from many different vantage points, in order to generate parallax errors issues.
- 13 images were taken of one the many sidewalks on campus, for parallax issues and to acquire human 'ghosts'.

Image resolution at capture is 3648x2736, and the size of the images range from 2.00 MB to 5.84 MB. Images are in JPEG format

FIGURE 1. A series of images from the control dataset. $10^\circ, 20^\circ, 30^\circ, 40^\circ$ from the origin frame, respectively.

FIGURE 2. A series of images from the control dataset. Note the humans in the scenes, and the difference in camera angles.

(A) Image A (B) Image B

FIGURE 3. Our Inputs

Our initial research showed that the effort required to develop a library of our own design was beyond the time and scope of the course. Simply implementing some of the algorithms by ourselves (such as RANSAC) would have been a large undertaking.

We settled on focusing on our primary objective, which was to investigate and understand the methods for stitching images. To this end, we used two computer vision libraries, Boofcv and OpenCV.

We built two programs, one for Android end users using the Boofcv library, and a proof of concept using the OpenCV library. Output images used in this report will be from the Python-OpenCV set.

3. STITCHING PROCESS

We use the SURF [1] algorithm to extract local features in the images. We then match keypoints in each image by comparing the similarity of their 'descriptor' vectors (Euclidean distance in this case). We then use Random Sample Consesus (RANSAC) to determine the homography matrix, which is the perspective transformation from the plane in image B to the plane in image A.

3.1. **Overview.** The Image Stitching Process is conceptually simple, but as discussed earlier, the individual pieces are somewhat more complex. Here it is laid out as pseudocode:

$A, B \leftarrow$ FirstImage, SecondImage
$Keys_A \leftarrow KeyPoints(A)$ ▷ Find keypoints using SURF
$Keys_B \leftarrow KeyPoints(B)$
$PointPairs_{AB} \leftarrow Match(A, B)$ ▷ Match those keypoints
$H \leftarrow FindHomography(PointPairs_{AB})$ ▷ Build the homography matrix
$Image_{New} \leftarrow Stitch(A, B, H)$ ▷ Stitch A, B using homography matrix H

[1] Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool, "SURF: Speeded Up Robust Features", Computer Vision and Image Understanding (CVIU), Vol. 110, No. 3, pp. 346–359, 200

(A) Image A (B) Image B

FIGURE 4. Keypoints Found using SURF

3.2. **Find Keypoints.** Keypoints in each image are found using Speed Up Robust Feature (SURF), an image feature detector and descriptor. It detects similar features or points of interest in an image and creates a descriptor vector for it. SURF is based on the algorithm Scale-invariant feature transform (SIFT). SURF is faster than SIFT and less sensitive to image transformations.

Feature detection in SURF is done using the determinant of the Hessian (DoH) approximation. The interest points are located in blobs with a maximum determinant. The Hessian matrix is computed by the 2nd order Gaussian convolution. This is done using large box filters, which can be applied efficiently using the integral images algorithm. The integral images algorithm (also know as summed area table) is an efficient way to sum an area in a grid. It runs in almost real time.

Feature description in SURF is done using descriptor vectors. A circular Haar wavelet is used to determine the interest point orientation. Then a square Haar wavelet is used to generate the descriptor vector around the orientation. Haar wavelets are calculated using the integral images algorithm.

3.3. **Matching Keypoints.** For each point of interest a descriptor vector is generated. Due to transformations of the same objects between two images the corresponding features might give similar but not exactly the same descriptor vectors.

Point correspondences between features in different images is done by finding pairs of vectors that have the smallest distance (are closest to each other). This can be done using a greedy search or k nearest neighbor algorithm.

Our implementation finds the nearest neighbor from one image to another by comparing the Euclidean distance between the descriptor vectors.

$$d(\overrightarrow{x}, \overrightarrow{y}) = \sqrt{\Big(\sum_{i=0}^{n}(x_i - y_i)^2\Big)}$$

3.4. **Transformation Matrix.** The homography, a matrix describing the perspective transformation from the plane in image B to the plane in image A, is found using the matched pair of keypoints

(A) Image A (B) Image B

FIGURE 5. Matching Keypoints using descriptor Euclidean distance

$$s \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \sim H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$H \approx \begin{bmatrix} 1.2277 & -0.0232 & -273.0147 \\ 0.0978 & 1.1543 & -41.5342 \\ 0.0004 & -3.1083e-06 & 1.0 \end{bmatrix}$$

FIGURE 6. Resulting Homography Matrix

from images A and B. A library function uses RANSAC (RANdom SAmple Consensus) to statistically remove outliers, which are incorrect point correspondences. This final, revised, set of feature matches is used for image alignment.

RANSAC is an iterative, statistical algorithm used to remove outliers from a data set. The algorithm assumes:

(1) most points are inliers
(2) there are some outliers
(3) there is possible noise

RANSAC fits the data to a data model and removes points that don't fit. The algorithm selects a random sample, creates a model for it, and fits all other data points to that model. Points that do not fit are removed. A new model is created for the points left over. It checks to see if the newly created model included a decent amount of points. If it does, it is considered to be a model for the entire data set. The more iterations RANSAC goes through the closer it gets to the actual model. In image stitching RANSAC is used to remove mismatched points. It is given the keypoint pairs from SURF and it removes the incorrect correspondences. The result is a much smaller, more accurate keypoint pair list.

FIGURE 7. Result : a composite image

4. PERFORMANCE EVALUATION

Our results showed that the SURF feature extraction works very well in identifying the keypoints. Our two biggest areas for improvement were the feature matching (which is very slow) and the lack of image blending.

Because of the speed issues, we used a very large Hessian threshold (> 1000) for the SURF extractor. Dropping this threshold would probably yield a larger set of matching keypoints, which should result in a more exact homography matrix. However, increasing the number of extracted keypoints would result in an increased matching time. Image blending would require breaking the image transformation and pasting into its constitute parts and implementing them ourselves. If so, we would need to leave the Python implementation behind.

Using three proprietary applications (Hugin[2], Microsoft Image Composite Editor[3], and Adobe Photoshop[4]), we established a benchmark using our dataset. The proprietary software blended the overlap region far better, and although our feature matches were good overall, there were a few obvious errors.

All of the three image stitching programs we used clearly performed better feature matching and blending than our program (see fig 8 on page 7). Of the three, only Adobe photoshop was not a free program. However, both the Hugin and Microsoft Image Composite Editor (ICE) handled the different exposures of the images stitched together better than Photoshop. None of the images handled the sky well; all three chose a different way to stitch the overlapping sky sections. Hugin arguably seemed to handle it best by blending a section rather than the two solid blue sections that Photoshop and ICE produced to contrast against the white sky adjacent to it.

There seems to be no perfect way to stitch images together, but our software does the job fairly well. With more time and a proper blending routine, we could conceivably match the performance of the applications we used for comparison. Nevertheless, we feel that we do understand the process enough to implement it for this assignment.

[2]http://hugin.sourceforge.net/

[3]http://research.microsoft.com/en-us/um/redmond/groups/ivm/ice/

[4]http://www.adobe.com/products/photoshop.html

FIGURE 8. A composite image created using Hugin (top) and our composite of the same scene (bottom).