Image Alignment in Matlab

Julian Anthony Brackins

Abstract—Introduction to Computer Vision project studying the process of image stitching and alignment in order to stitch images of similar views.

Index Terms—Computer Science, Computer Vision, Feature Matching, Image Alignment, Panorama, Matlab

1 Introduction

THE goal of this exercise is to construct a functional panorama stitcher. The program is separated into two stitching methods. The first iteration of the program will perform image alignment and stitching in a user-driven interface. Given an interface, a user can select 4 or more features that correspond to one another in similar scenes. In the second iteration of the program, the interest points will be automatically generated using the *Scale-invariant feature transform* algorithm (SIFT) for detecting and matching features and the *RANdom SAmple Consensus* algorithm (RANSAC) to remove outliers from the automatically matched key feature sets.

2 Process

As stated previously, this project has been divided into two main approaches. The first method for image alignment is the user-driven implementation.

2.1 Manual Mosaicking

The first step is to design a manual, user-based design for image stitching. A lot of the functionality present in this implementation, particularly the Homography calculations, Image warping, and mosaic display are identical in both the manual and automated implementations for image alignment. Because of this, most of these topics will be explained in depth only in the manual mosaicking section.

2.1.1 Feature Selection

Using the *ginput()* function in Matlab, a user can click on corresponding points in two images to select distinctive features visible in both scenes or images. The *ginput()* functionality has been modified slightly in this implementation to allow the user to see each point in the scene immediately after clicking. Because of this, the traditional functionality of *ginput()* that allows the user to cease selecting points by pressing the space bar has been disabled; instead, the user specifies how many interest points she/he will select in a graphical user interface before selecting the features.

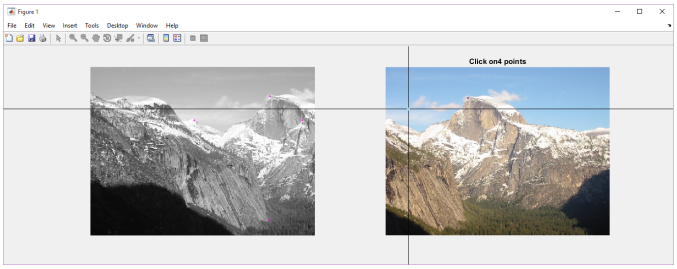


Fig. 1: User interface for selecting matching features in two images.

• *J. Anthony Brackins is a Computational Sciences and Robotics graduate student at The South Dakota School of Mines and Technology, 501 E St Joseph St, Rapid City, SD, 57701.*
E-mail: julian.brackins@mines.sdsmt.edu

Fig. 2: Resultant image from selected corresponding points in Figure 1.

2.1.2 Computing Homography

Computation of the Homography Matrix is done by using the Direct Linear Transformation algorithm. After calculating the H matrix, the images in one point can be warped onto a second image reference frame. With these warpings computed, two images can then be stitched together.

2.1.3 Image Warping

Image warping is performed by interpolating points from one image reference frame to lie in the same reference frame as a second image. Given two images p and p', the points in image p are inverse warped onto the same plane as p' using a computed homography matrix.

For some reason, using $interp2()$ seemed to be fine at first when performing image warping and mosaicking. However, the bounding box would not work properly with the $meshgrid()$ function that creates the 2 dimensional canvas which the points are interpolated onto. The $WarpImage.m$ shows the unsuccessful implementation of image warping that does not compute the appropriate bounding box.

The correct implementation of image warping and mosaic displaying can be found in the $Overlay.m$ function. This method uses an $Interpolation()$ function that uses this equation for linear interpolation:

2.1.4 Mosaic Display

Due to the nature of the previously described image warping implementation, the mosaic is generated at the same time that the second image is being warped. Therefore, all that is needed is a simple *imshow()* call to display the mosaic containing the stitched images on the same plane.

2.2 Automatic Mosaicking

With all of the functionality for image stitching properly implemented, the next step is to automate the feature selection process. In order to accomplish this, two additional features are added to the program, the automated feature selection for returning a list of corresponding points, and a RANSAC algorithm to reduce the number of invalid correspondences generated from the automated feature selection points list.

2.2.1 Automated Feature Selection

The automated feature selection method used in this program is SIFT. As per permission from instructor, the SIFT implementation available on David Lowe's website, http://www.cs.ubc.ca/~lowe/keypoints/.

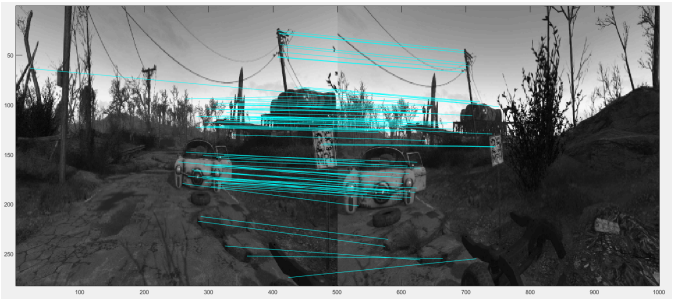


Fig. 3: SIFT feature matching for a scene in the video game Fallout 4.

2.2.2 Removing Poor Correspondences

Automated feature selection is not perfect. Due to the potential of returning false positive correspondences in a purely automatic feature selection algorithm, an additional verification step must be implemented in order to ensure the homography transform is computed only using valid corresponding points.

3 Benchmarks

In order to validate the implementation of image stitching in the Matlab program, the following benchmark image sets have been tested.

Fig. 4: Image Stitching of Figure 3 without RANSAC.

Fig. 5: Image Stitching of Figure 3 with RANSAC.

Three different validation categories have been created for the purposes of evaluating the image stitching implementation for a course grade.

3.1 Validation on Course Website Benchmark Images

Several benchmark images have been uploaded to the course web page, http://webpages.sdsmt.edu/~rhoover/Dr._Randy_C._Hoover/CENG414_F15.html for the purpose of validating the image alignment code written. For brevity, these results have been added to the appendix of this document.

3.2 Validation on Personal Images

To validate the capability of the image stitching implementations, scenes from the computer game Fallout 4 have been used to create panoramic views of the game world.

Figures 6 demonstrates the functionality of the manual image stitching implementation on a skyline view in the game world.

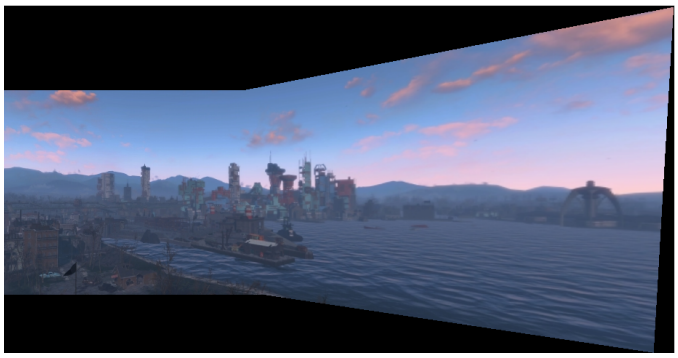
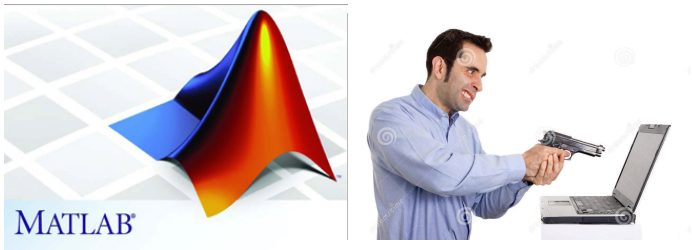


Fig. 6: Image stitched view of post-apocalyptic Boston in Fallout 4.

Figures 3, 4, and 5 are another image set taken from Fallout, demonstrating the functionality of RANSAC.

3.3 Warping Images Together

A rather amusing feature of image alignment is the ability to take advantage of Homographic transformation in order to impose images onto the features of other, dissimilar images. The particular example illustrated in Figure 8 depicts a logo being warped onto the screen of a laptop in another image.



(a) Matlab logo (b) Man Holding a firearm.

Fig. 7: Two Dissimilar Images

3.4 Other Features

3.4.1 User Interface

The program contains a simple user interface when running the image alignment. Simply select two image files using the Image Select buttons, and choose the method for performing image alignment. For Manual image alignment, the number of corresponding points must be input by the user, with the default being 4. After running the program, a canvas pops up,

Fig. 8: Images Warped Together.

allowing the user to click features on two images. For Automated image alignment, the user is given the choice to run the program with the RANSAC algorithm, increasing computation time but reducing the number of erroneous correspondences.

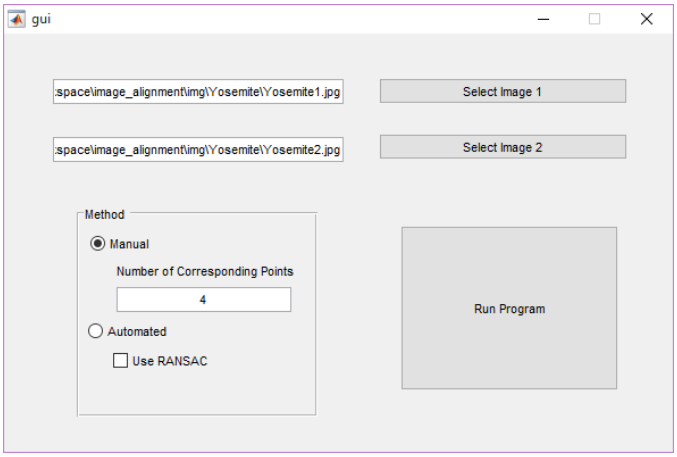


Fig. 9: User Interface

3.4.2 *Image Rectification*

Image Homography can be used to perform image rectification. Based on the angle in which an object is viewed, its proportions can be skewed. Homography transformation can be used to modify the skew of an image. In 10 a Rubix cube is being displayed with 3 sides visible, the two sides, and the top of the cube. The cube also appearst to be rotated approximately 45 degrees in reference to the viewing angle. In 11, the projections of a Rubix cube have been modified so that the squares on the surface do not appear warped, as if one were looking at the squares from a top-down perspective (although this is technically inaccurate, since the two sides would not be visible at all if the cube were being viewed from above.)

Fig. 10: Rubix Cube Image. Top squares skewed due to angle of image captured

Fig. 11: Rubix Cube Image Warped to project top squares without any skewing.

4 Conclusion

This project was an in depth analysis on image alignment and stitching. Stitching multiple images into a panoramic view is a very important computer vision task, and while feature matching algorithms such as SIFT have come a very long way in regards to feature detection, additional algorithms must be implemented to validate matched features. In this program, RANSAC was implemented. Despite being a useful algorithm, RANSAC is computationally slow, and greatly increased the amount of time spent aligning two images. Despite this, the RANSAC algorithm still proved to be useful in removing false positives generated in an automated feature list.

The repository containing the code discussed in this writeup can be found at https://github.com/jbrackins/image_alignment

Appendix A
Image Results

Fig. 12: SDSM&T Quad (Manual Stitching)

Fig. 13: Dinosaur Park in Rapid City, South Dakota (Automated Stitching)

Fig. 14: Fire Tower Image Set (Manual Stitching)

Fig. 15: Sylvan Lake in South Dakota (Automated Stitching)

References

[1] David G. Lowe. "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*. 60, 2 (2004), pp. 91-110.

[2] Konstantinos G. Derpanis *Overview of the RANSAC Algorithm*. May 13, 2010.

[3] Linear Interpolation
`https://en.wikipedia.org/wiki/Linear_interpolation`