Creating a Web Application for Image Stitching Using Various Image Augmentations, Feature Detectors, Feature Descriptors, and Feature Matchers

Bajkowski, Trevor - CMP_SCI 7650
Hogg, Matthew - CMP_SC 4650

11 December 2019

Contents

1 Introduction

Note: All code can be found at https://github.com/trevorBajkowski/miz_image_stitcher/tree/web-app/. Any references to datasets are in the sample folder and are named accordingly. Any datasets not cited were created by the authors.

1.1 Roles

Trevor Bajkowski

- Feature Detection
- Feature Matching
- Image Augmentation
- Homography Calculation

Matthew Hogg

- Web Application
- Image Augmentation

1.2 Image Stitching

Before discussing the overall project, it is important to know what it means to perform Image Stitching, or Image Mosaicing. We shall define the process of "Image Stitching" as discovering the mapping between two overlapping images from a shared scene. That is, given two images with some overlap of contents, can we create one seamless image with the contents of both images?

Image Stitching receives a great deal of attention from those interested in Remote Sensing[5], as this problem is inherently tied to others such as object detection/recognition, map building, terrain analysis and land usage. The process of Image Stitching can be said to be composed of 3 main parts: Key-Point Detection and Description, Key-Point Matching and the Mapping Calculation. These topics will be discussed respectively in Sections 4, 5 and 6.

1.3 Project Goals

Now, with a high-level understanding of Image Stitching, the reader has necessary context for understanding the project proposed and developed by the authors.

The authors seek to create a web-based application that allows users to upload a series of sequential, shared scene images and to receive a single mosaic of these images in return. The development of this website and the architecture will be discussed in Section 2. The users will be allowed to choose from a series of augmentations to perform on their images prior to the stitching process; these will be discussed in Section 3. The users will also have choices for different algorithms for Key-Point Detection and Matching; these algorithms are discussed in Sections 4 and 5.

In Section 7, the results of running various image sets through the website will be compared and discussed. As this application is meant to be multipurpose, we will compare results on 4 datasets. The first two will be Remote Sensing focused while the second two will be target to panoramic stitching.

1.4 Tools

- Python 3.7.0
- OpenCV 4.1.2.30
- Flask 1.1.1

2 Web Application - server.py, templates/home.html

2.1 Flask

Flask is a lightweight web application framework. It was used in this project to create a simple web app that allows the user to upload shared scene images and stitch them together. The user may select from any of the feature detection/matching algorithms mentioned in sections 4-5 to stitch their images, as well as apply any of the image augmentations mentioned in section 3. Once the user has uploaded their images and selected their algorithms and augmentations, the app will stitch the images together and return the stitched image.

2.2 Development

Development required the creation of a Flask app with a frontend template. Once that was done, the app had to be configured so that it could effectively run the image stitching functions. The largest hurdle in the development of the web app was managing dependencies, a problem solved using Conda.

2.3 Architecture

The architecture of the app is relatively straightforward. It uses a HTML/CSS/JS frontend in templates/home.html and a Python backend consisting of a variety of files that implement image stitching.

3 Image Augmentations - image_augment.py

3.1 Color Balance

One of the four optional image augmentations is RGB-Histogram Equalization. This is done by plotting 3 histograms of the red, green and blue pixel intensities for both images. We choose one image to be the reference image (the one that is stitched onto by implementation) and use its cumulative intensity histograms for histogram matching. The thought here is that balancing the RGB intensities between images will allow for smoother boundaries between sequential images. An example correction can be seen in Figure 1.

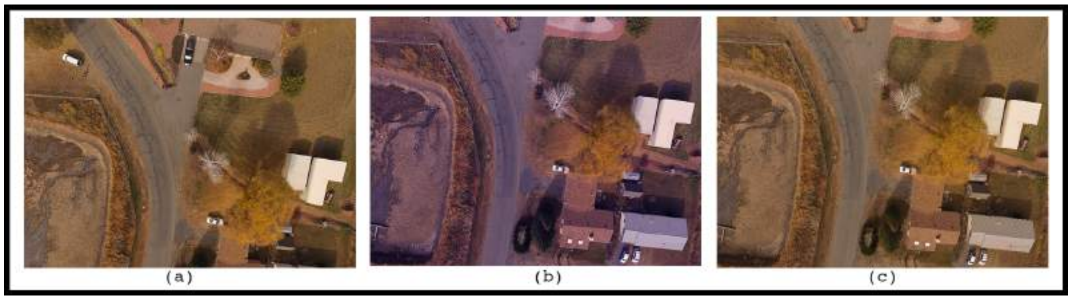

Figure 1: (a) Ref. Image (b) Source Image (c) Corrected Source Image
Dataset 7 [6]

3.2 Value Balance

This is the same process as Color Balancing, but the RGB image is first converted to Hue, Saturation, Value (HSV). Then histogram matching is performed just on the Value channel of the image. Value is the lightness of a pixel, so how black or how white it is. This image is then converted back into RGB color for processing.

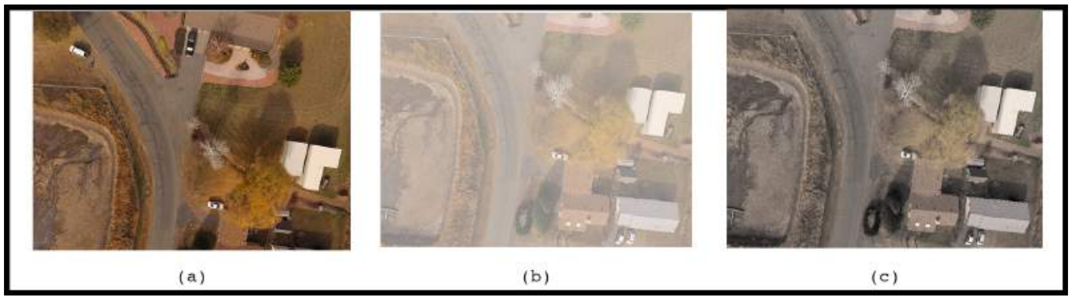

Figure 2: (a) Ref. Image (b) Source Image (c) Corrected Source Image
Dataset 7 [6]

3.3 Image Smoothing

Image Smoothing can be done in three different ways in this application. First, a 5x5 median filter can be passed over the image. The second option is Gaussian Blurring where a 5x5 Gaussian Filter with $\sigma = 1$ is passed over the image. The final and default option is to use Bilateral Smoothing, where pixels are replaced by weighted averages of their neighbors based on radiometric and Euclidean distance.

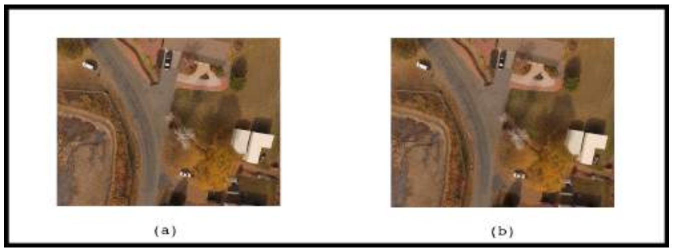

Figure 3: (a) Source Image (b) Smoothed Image
Dataset 7 [6]

3.4 Image Sharpening

A simple 3x3 sharpening filter is used. The center value of the 3x3 matrix is 9, while the rest are -1. The output of sharpening a graffiti wall[7] image is seen in Figure 4.

4 Feature Detection - **detection.py**

Feature Detection is the process by which "key-points" are found in the image. These are classically corners because corners are preserved for many changes in perspective [8]. Further, many key-points

Figure 4: (a) Source Image (b) Sharpened Image
Dataset 6 [7]

these days have descriptors which give them an orientation [3, 9, 10] or some notion of individuality that will help match it to the same point in another image. This can also be used in tracking objects from frame to frame in videos, sequential images, etc.

4.1 ORB[1]

ORB was created by researchers at the OpenCV team in response to SURF and SIFT becoming proprietary algorithms. It outperforms the aforementioned algorithms in speed and is comparable in performance. Oriented FAST and Rotated BRIEF (ORB[1]) uses the FAST-9-16[11] algorithm to nominate key-points. For every pixel, this looks for a 9-pixel arc in the 16-pixel circle around the pixel. This arc has to have in intensity "significant" enough to be nominated as a key-point. The key-point uses the BRIEF[12] feature descriptor, and calculates an orientation with 12-degree accuracy. The N-Best key-points will be nominated based on how corner-like they are, as calculated by the Harris Score[8].

4.2 A-KAZE[2]

Accelerated KAZE (A-KAZE[2]) is a follow up and improvement on the original KAZE[15]. It uses Fast Explicit Diffusion to search for key-points on various scales, while maintaining strong edges throughout the process. Disappearing edges can be problematic when it comes to scaling images, so FED can have better results in finding important key-points. A-KAZE also achieves scale-invariance by using the determinant of the Hessian for key-point nomination. Further, in the paper that introduces A-KAZE, the Modified-Local Difference Binary (M-LDB) descriptor that is introduced. The authors claim this descriptor is "highly efficient, exploits gradient information from the nonlinear scale space, is scale and rotation invariant and has low storage requirements[2]." This technique finds an orientation for a given point by using circular sliding windows which rotate in $\frac{\pi}{3}$ radian intervals. The authors boast an improved compromise in efficiency and performance over the algorithms contemporaries.

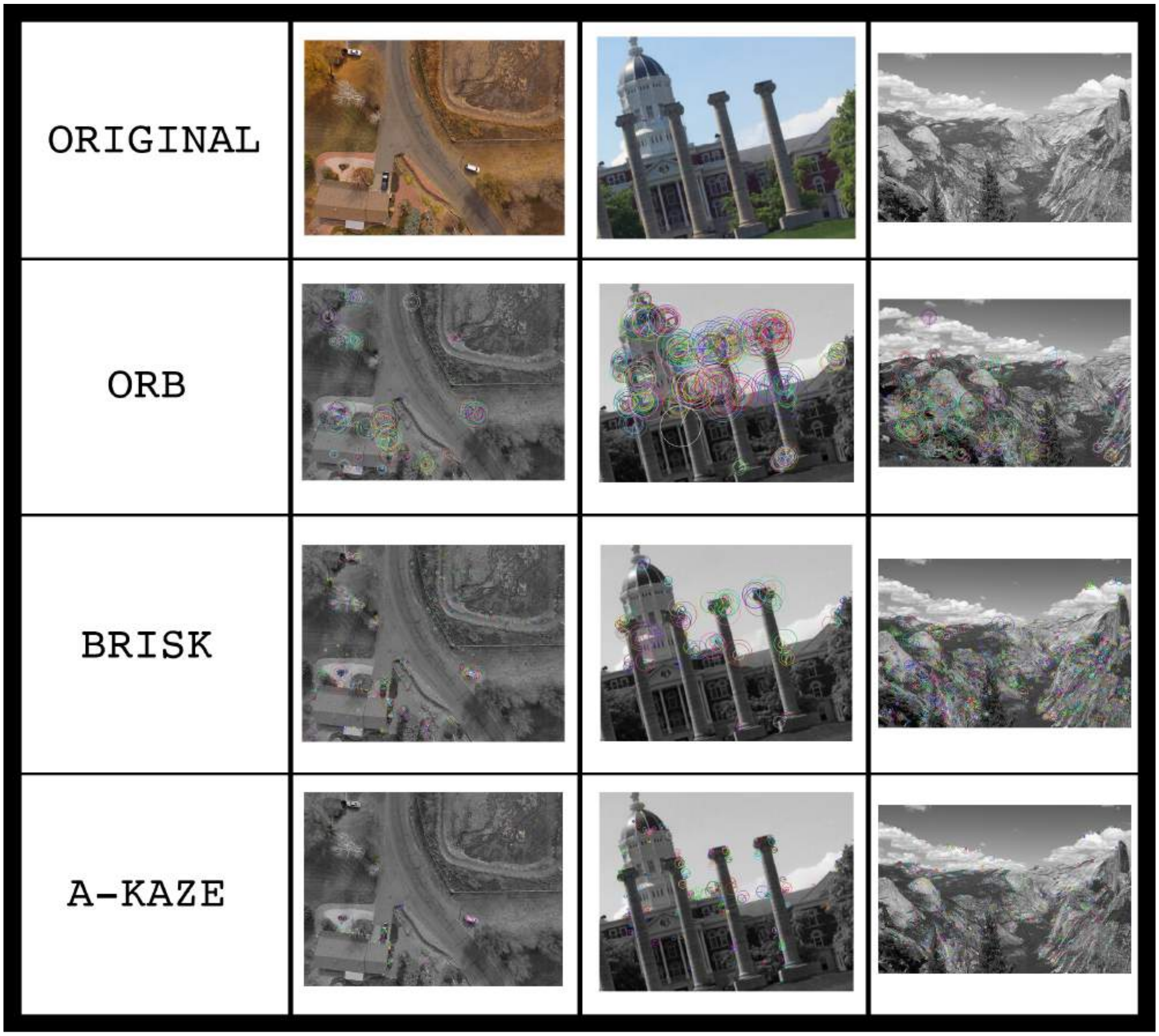

Figure 5: Left to Right: Set 7 [13],
Set 1, Set 10[14]

4.3 BRISK[3]

The Binary Robust Invariant Scalable Keypoints (BRISK[3]) algorithm was introduced by ETH Zürich researchers in 2011. Like ORB, BRISK uses the FAST-9-16[11] feature detector for key-point nomination. Additionally, BRISK uses scaled octaves of the images to look for features on multiple scales. This process helps to pick key-points that should be easier to find in more contexts and helps achieve scale-invariance. BRISK key-points are oriented using rings of circles, equally spaced around the key-point. Using these points and the edges connecting them, the orientation of the key-point is calculated.

5 Feature Matching - matching.py

Feature Matching is the method by which the key-points of two images are compared so that the "most similar" key-point pairs can be used to find the mapping between two images. Two methods will be examined: Brute Force Matching and the Fast Library for Approximate Nearest Neighbors[4].

5.1 Brute Force Matching

Brute Force Matching is named aptly. Let i be the number of key-points nominated in Image I_1, with key-points k_{1i} and let j be the number of key-points nominated in Image I_2, with key-points k_{2j}. For every pair of nominated key-points, (k_{1i}, k_{2j}), the Hamming distance is calculated between their descriptor vectors, d_{1i} and d_{2j}. The Hamming distance for bit vectors is $Hamming(b_1, b_2) = \sum_{0 \leq m \leq i} \sum_{0 \leq n \leq j} b_1[m] \bigoplus b_2[n]$. Then, the matches are sorted based on their distances and the closest N matches are chosen for calculating the 3x3 homography between the images.

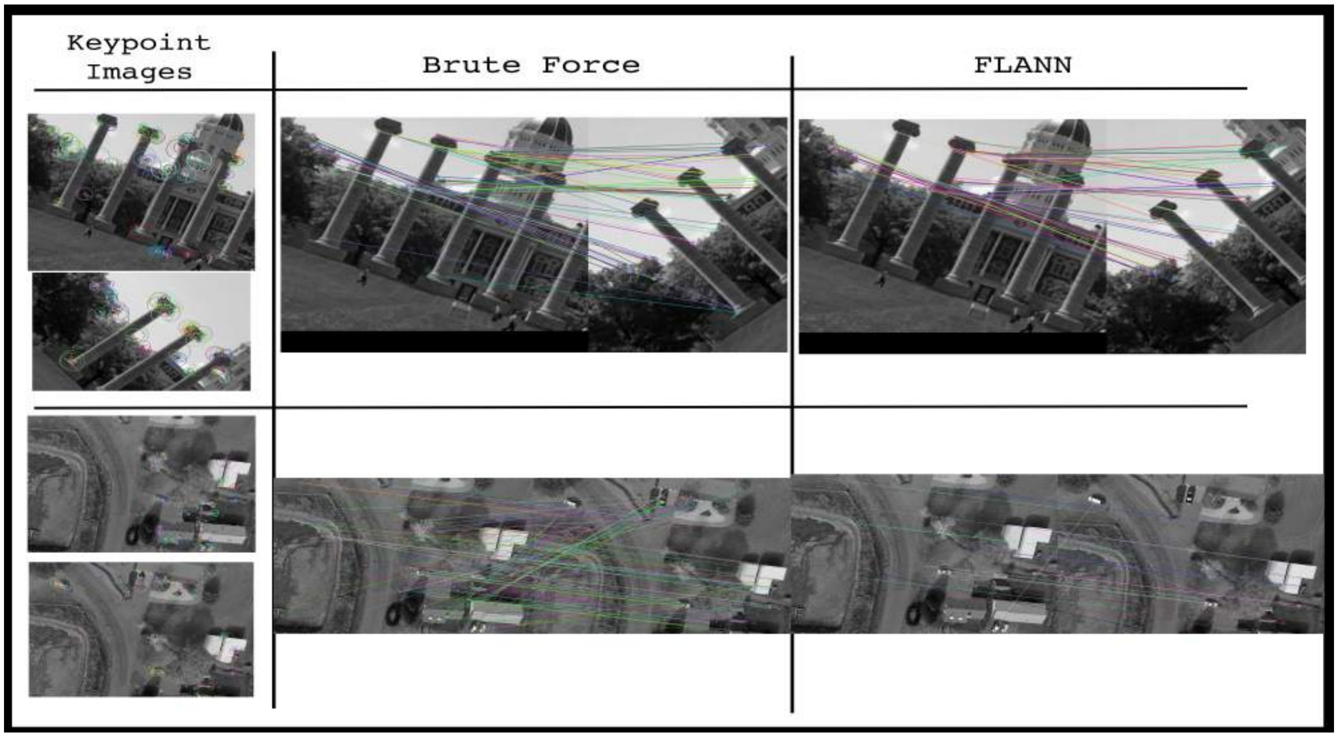

Figure 6: Top to Bottom: Set 1, Set 7 [13]

5.2 FLANN[4]

The Fast Library for Approximate Nearest Neighbors (FLANN[4]) is a suite of libraries which contain optimizations for performing K-Nearest-Neighbor matching. Using FLANN is the approximately the same as using a traditional K-Nearest-Neighbors algorithms, but it requires significantly less time, which is important when building an application that is intended for web-usage. This library finds matches using a K value of two, and then using the distance test presented by Lowe[16]. That test iterates through every sequential pair, $m[i]$ and $m[i+1]$ in the list of matches m, and removes $m[i]$ if $distance(m[i]) > 0.7 * distance(m[i+1])$. This is done to trim down the match population to speed up the homography calculation and to choose "better" matches.

6 Calculating the Transform - stitching.py

Once the key-points and their corresponding matches have been attained, one must generate a 3x3 homography, or transformation matrix, which describes how to translate one image to "stitch" it onto the other. We use the RANdom SAmpling Consensus (RANSAC[17]) algorithm to decrease the effects of incorrect matches. In brief, this algorithm randomly samples the matches in samples of increasing size, all while maintaining a "consistent" model. More samples are added until a sufficient number of samples is reached, resampling if the consistency constraints are broken. Using

7 Results

7.1 Panorama

7.1.1 Sample Set 1

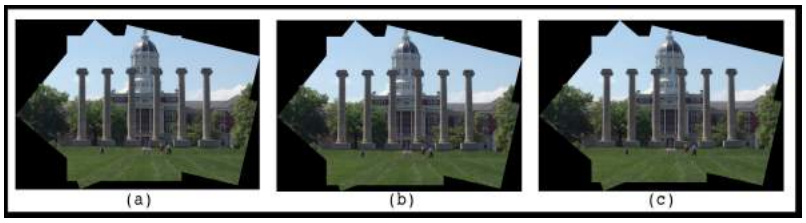

Figure 7: Set 1 experiments

Seen in Figure 7. Variation in Methods:

- a) A-KAZE, BRUTE, SCALE=1, NO AUGMENTATIONS
- b) BRISK, BRUTE, SCALE=1, NO AUGMENTATIONS
- c) ORB, BRUTE, SCALE=1, NO AUGMENTATIONS

7.1.2 Sample Set 5 [14]

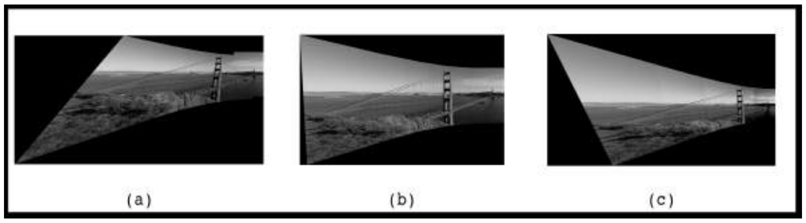

Figure 8: Set 5 [14] experiments

Seen in Figure 8. Variation in Methods:

- a) A-KAZE, FLANN, SCALE=1, NO AUGMENTATIONS
- b) A-KAZE, FLANN, SCALE=1, SHARPEN
- c) A-KAZE, FLANN, SCALE=1, VALUE BALANCE

7.1.3 Sample Set 6 [7]

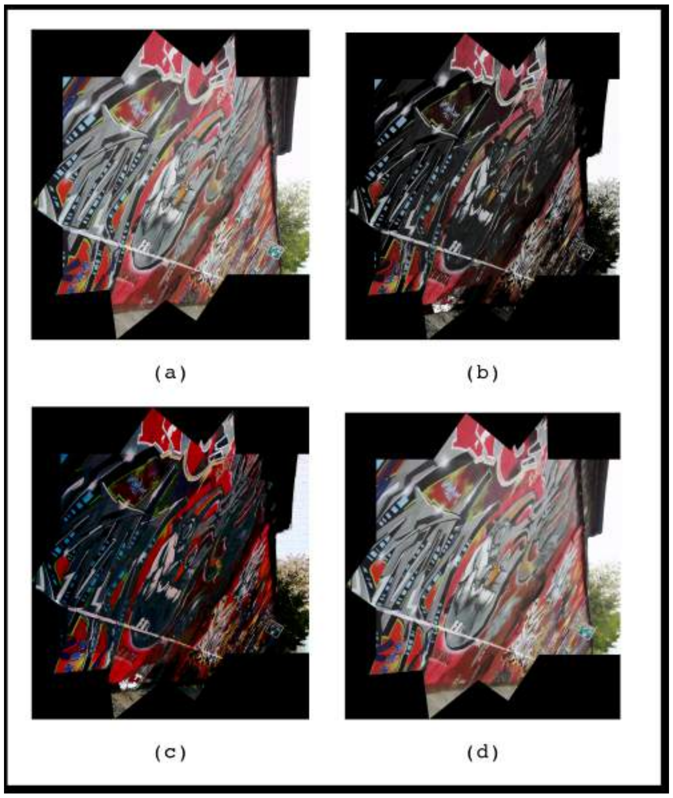

Figure 9: Set 6 [7] experiments

Seen in Figure 9. Variation in Methods:

- a) A-KAZE, FLANN, SCALE=1, NO AUGMENTATIONS
- b) A-KAZE, FLANN, SCALE=1, VALUE BALANCE
- c) A-KAZE, FLANN, SCALE=1, RGB BALANCE
- d) A-KAZE, FLANN, SCALE=1, SMOOTH

7.1.4 Sample Set 10 [14]

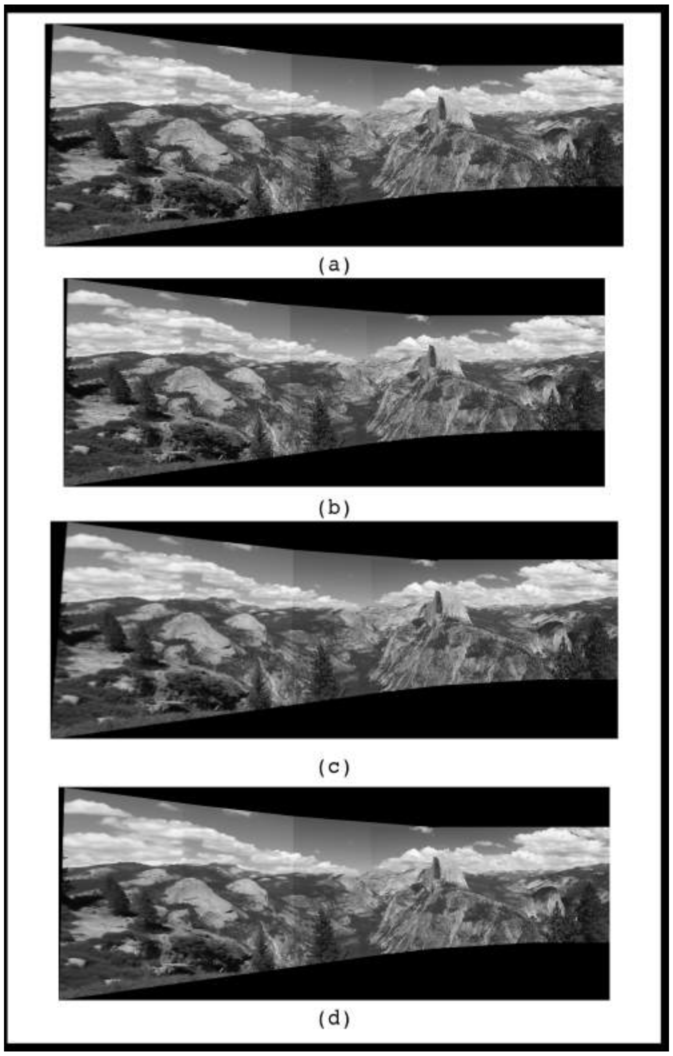

Figure 10: Set 10 [14] experiments

Seen in Figure 10. Variation in Methods:

- a) A-KAZE, FLANN, SCALE=1, NO AUGMENTATIONS
- b) BRISK, FLANN, SCALE=1, NO AUGMENTATIONS

- c) BRISK, FLANN, SCALE=0.5, NO AUGMENTATIONS
- e) BRISK, FLANN, SCALE=1, SMOOTH

7.2 Overhead Imagery

7.2.1 Sample Set 2

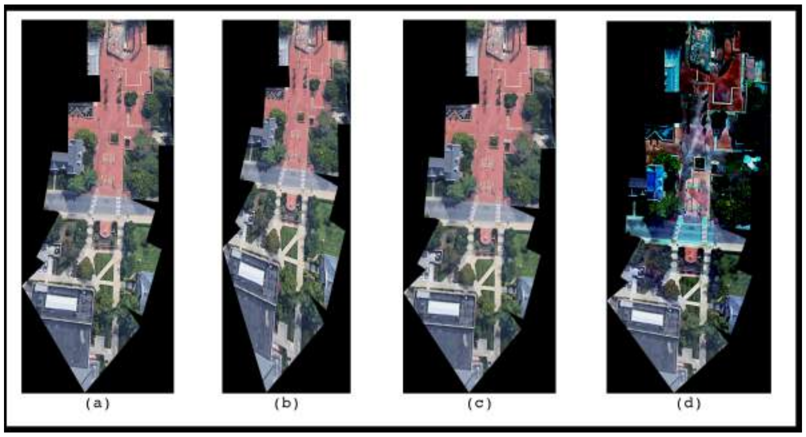

Figure 11: Set 2 experiments

Seen in Figure 11. Variation in Methods:

- a) A-KAZE, FLANN, SCALE=1, NO AUGMENTATIONS
- b) BRISK, FLANN, SCALE=1, NO AUGMENTATIONS
- c) A-KAZE, FLANN, SCALE=1, SMOOTH
- f) BRISK, FLANN, SCALE=1, SMOOTH, RGB-BALANCE

7.2.2 Sample Set 7 [13]

Seen in Figure 12. Variation in Methods:

- a) A-KAZE, FLANN, SCALE=0.25, NO AUGMENTATIONS
- b) BRISK, FLANN, SCALE=0.25, NO AUGMENTATIONS
- e) A-KAZE, FLANN, SCALE=0.25, RGB-BALANCE, VALUE-BALANCE
- f) BRISK, FLANN, SCALE=0.25, RGB-BALANCE, VALUE-BALANCE

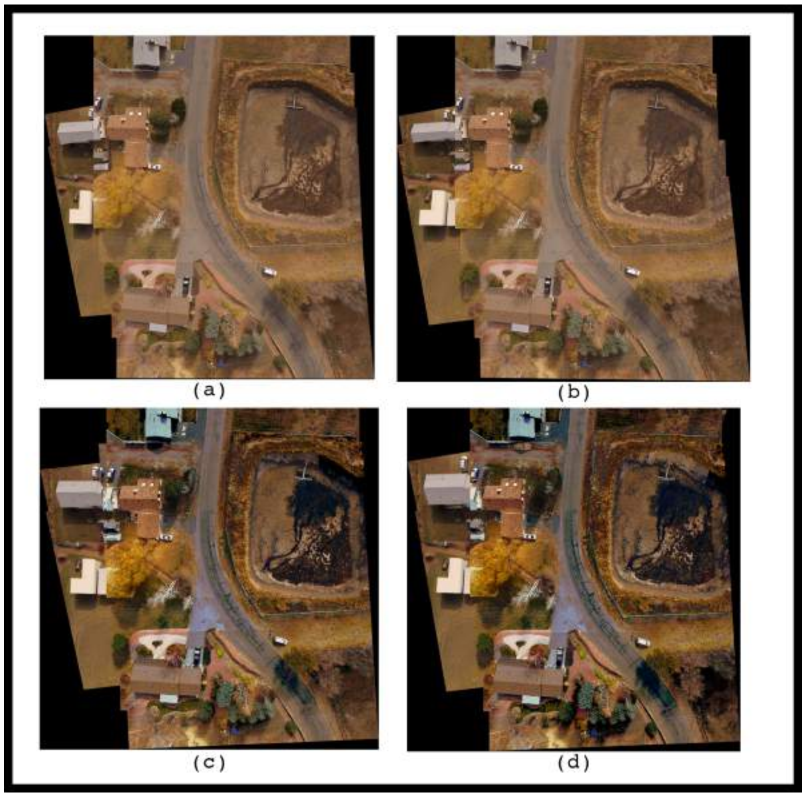

Figure 12: Set 7 [13] experiments

7.3 Application Screenshots

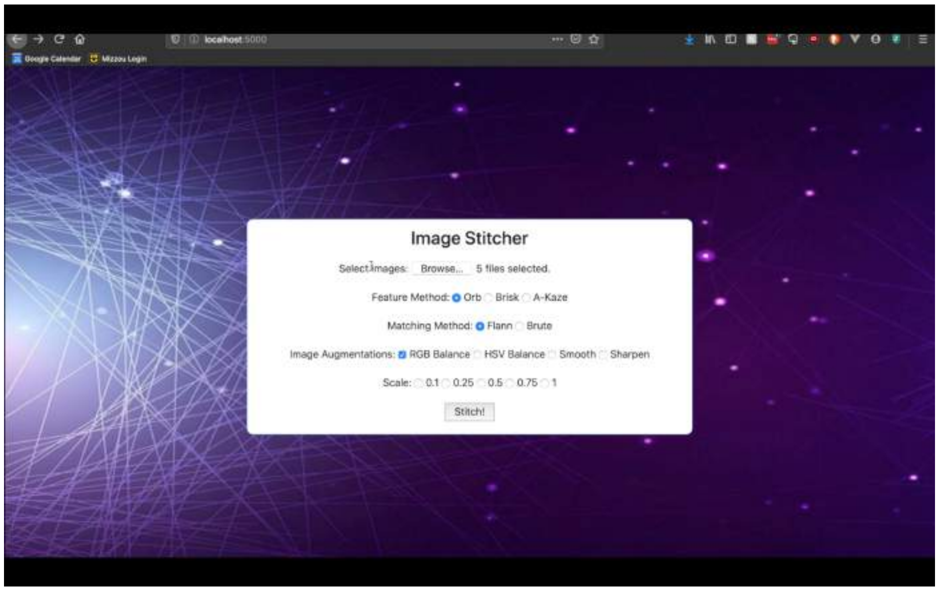

Image Stitcher

Select Images: 5 files selected.

Feature Method: [✓ Orb] Brisk A-Kaze

Matching Method: [✓ Flann] Brute

Image Augmentations: [✓ RGB Balance] HSV Balance Smooth Sharpen

Scale: 0.1 0.25 0.5 0.75 1

Figure 13: Application screenshot

8 Future Work

There are many areas in which this application could be improved. To begin, there are many optimizations to be made in terms of how key-points are calculated. They are not stored from stitching to stitching, so doing so would save a lot of computation time. Further, images must be presented such that sequential images have overlap. An improvement would be to allow images to be uploaded in no particular order, and to create a mosaic from that. There are more feature detectors and descriptors such as CENSURE and SIFT that would be interesting to test with. Additionally, the web-application and stitching code could use error handling, as the system is fragile to incorrect inputs in its current state. The code currently produces many intermediate images, we would like to incorporate these matches and key-point images into a visualization while the stitching is being completed. It would be nice to also offload the computation to an AWS server to improve computation times. Finally, the website could be more aesthetically pleasing.

References

[1] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “Orb: An efficient alternative to sift or surf,” in *2011 International Conference on Computer Vision*, pp. 2564–2571, Nov 2011.

[2] P. Fernández Alcantarilla, “Fast explicit diffusion for accelerated features in nonlinear scale spaces,” 09 2013.

[3] S. Leutenegger, M. Chli, and R. Y. Siegwart, “Brisk: Binary robust invariant scalable keypoints,” in *2011 International Conference on Computer Vision*, pp. 2548–2555, Nov 2011.

[4] M. Muja and D. Lowe, *FLANN - Fast Library for Approximate Nearest Neighbors User Manual.* UBC.

[5] S. Ait-Aoudia, R. Mahiou, H. Djebli, and E.-H. Guerrout, “Satellite and aerial image mosaicing - a comparative insight,” pp. 652–657, 07 2012.

[6] DroneMapper, “4th ave reservoir capacity map – cedaredge, colorado.”

[7] Visual Geometry Group, K. Leuven, I. Rhone-Alpes, and The Center for Machine Perception, “Affine covariant features image set.”

[8] C. Harris and M. Stephens, “A combined corner and edge detector,” in *Proceedings of the 4th Alvey Vision Conference*, pp. 147–151, 1988.

[9] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05) - Volume 1 - Volume 01*, CVPR ’05, (Washington, DC, USA), pp. 886–893, IEEE Computer Society, 2005.

[10] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, “Speeded-up robust features (surf),” *Comput. Vis. Image Underst.*, vol. 110, pp. 346–359, June 2008.

[11] E. Rosten and T. Drummond, “Machine learning for high-speed corner detection,” in *ECCV*, 2006.

[12] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, “Brief: Binary robust independent elementary features,” in *Proceedings of the 11th European Conference on Computer Vision: Part IV*, ECCV’10, (Berlin, Heidelberg), pp. 778–792, Springer-Verlag, 2010.

[13] DroneMapper, “Red rocks, colorado – oblique.”

[14] J. Brandt, “Transform coding for fast approximate nearest neighbor search in high dimensions,” 2010.

[15] P. F. Alcantarilla, A. Bartoli, and A. J. Davison, “Kaze features,” in *Proceedings of the 12th European Conference on Computer Vision - Volume Part VI*, ECCV’12, (Berlin, Heidelberg), pp. 214–227, Springer-Verlag, 2012.

[16] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International Journal of Computer Vision*, vol. 60, pp. 91–110, Nov 2004.

[17] M. A. Fischler and R. C. Bolles, “Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography,” *Commun. ACM*, vol. 24, pp. 381–395, June 1981.