Summer 2023

Real Time Panoramic Image Processing

Matthew Gerlits
San Jose State University

Real Time Panoramic Image Processing

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Matthew Gerlits

May 2023

The Designated Project Committee Approves the Project Titled

Real Time Panoramic Image Processing

by

Matthew Gerlits

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2023

Dr. Teng Moh	Department of Computer Science
Dr. Melody Moh	Department of Computer Science
Dr. Mark Stamp	Department of Computer Science

ABSTRACT

Real Time Panoramic Image Processing

by Matthew Gerlits

Image stitching algorithms are able to join sets of images together and provide a wider field of a vision when compared with an image from a single standard camera. Traditional techniques for accomplishing this are able to adequately produce a stitch for a static set of images, but suffer when differing lighting conditions exist between the two images. Additionally, traditional techniques suffer from processing times that are too slow for real time use cases. We propose a solution which resolves the issues encountered by traditional image stitching techniques. To resolve the issues with lighting difference, two blending schemes have been implemented, a standard approach and a superpixel approach. To verify the integrity of the cached solution, a validation scheme has been implemented. Using this scheme, invalid solutions can be detected, and the cache regenerated. Finally, these components are packaged together in a parallel processing architecture to ensure that frame processing is never interrupted.

ACKNOWLEDGMENTS

I would like to thank Dr. Teng Moh for his help and guidance over this course of this project. He was always ready with advice to point me in the right direction. His suggestions were instrumental for the formulation of this paper. I would like to thank my wife, Carina Gerlits, for her patience and support over the course of my return to school. She provided me with the love I needed to push through the tough and stressful times over these past few years. Finally, I would like to thank my cat Kiki. She has been with me for nearly every class during my journey to obtain my master's degree.

TABLE OF CONTENTS

CHAPTER

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

The average human has a field of view of approximately 170°-180°, while standard cameras only have a field of view ranging from 6°-94°. Humans can achieve this wide field of view by having access to two adjacent optical sensors, their eyes. The combined image, provided by their eyes, allows humans to collect a large amount of information. By mimicking human biology, the field of view of standard cameras can be extended so that a single image that contains all the information from each camera can be produced. The information contained in this image allows humans to more easily digest the information collected by the cameras.

Currently, a procedure know as image stitching can accomplish the feat of joining images together. This is accomplished by collected a set of matching features between a pair of images, and using those matching features to align the images so that their overlapping areas match. This procedure is usually adequate for stitching together a static pair of images, but suffers from a few limitations. The first limitation is that the processing speed which, while not cumbersome for casual use cases, is insufficient for use in a real time application. Next, the stitched images produced by this approach do not factor in differences in lighting. This can cause a very obvious seam to appear, even if the image are well aligned. The seam can be hidden by utilizing image blending, but many of the currently available blending schemes have processing times which are too slow for use in a real time application and perform inconsistently. Finally, the alignment of the images are dependant on the matched features, which are susceptible to inaccurate matches. Each inaccurate match reduces that accuracy of the alignment operation.

To resolve these limitations, we propose a system which assumes that the camera positions do not change relative to each other and that the cameras are horizontally

aligned. Using these assumptions, a wide range of operations, including the most costly portions, can be cached. By caching the results of these operations, the processing of each frame can be drastically simplified. Next, we propose a technique for detecting the occasional misalignment of the cameras, at which point the cached information will be recalculated. Finally, we detail a parallel pipeline that allows for continuous stitching, alongside the detection and correction of camera misalignment.

CHAPTER 2

Existing Solutions

To understand the need for a robust real time solution to extending a cameras field of view, we must first explore the limitations of existing camera systems. Feature detection is the first step in extending a set of images field of vision, and this paper reviews the inner working of feature detection, specifically for the SIFT algorithm. Image stitches are provided to show the results of using SIFT, along with benchmarks of other feature detectors.

2.1 Background

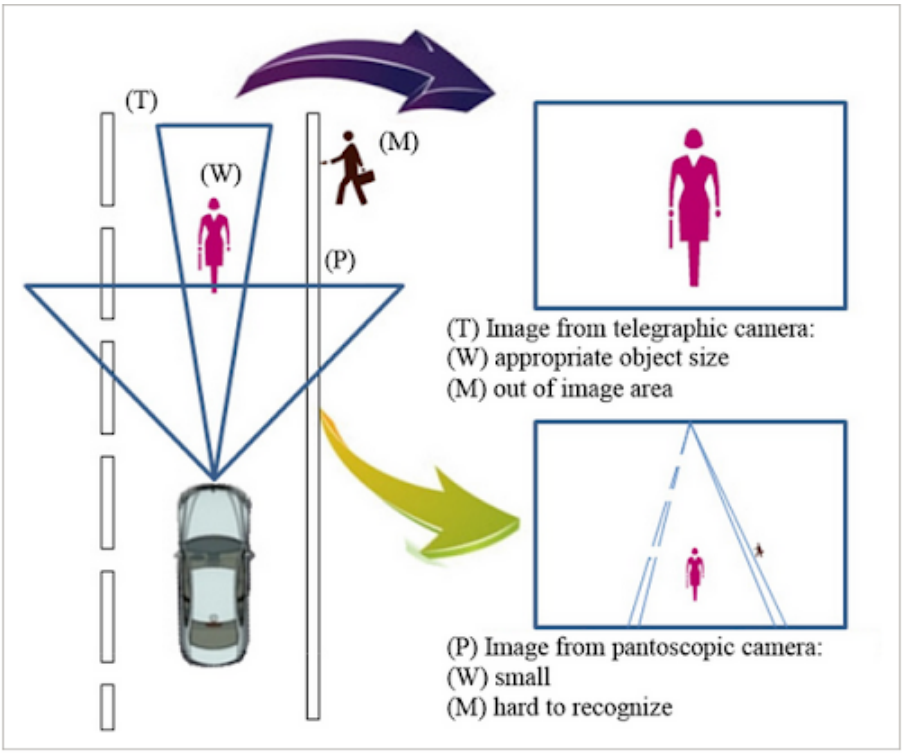

Figure 1: Representation of standard cameras vs panoramic camera [4]

The limited field of view of standard cameras limits the amount of information that can be collected by an individual camera, while the distortions create by using a fisheye lens mitigates their usefulness, despite the larger field of view that they

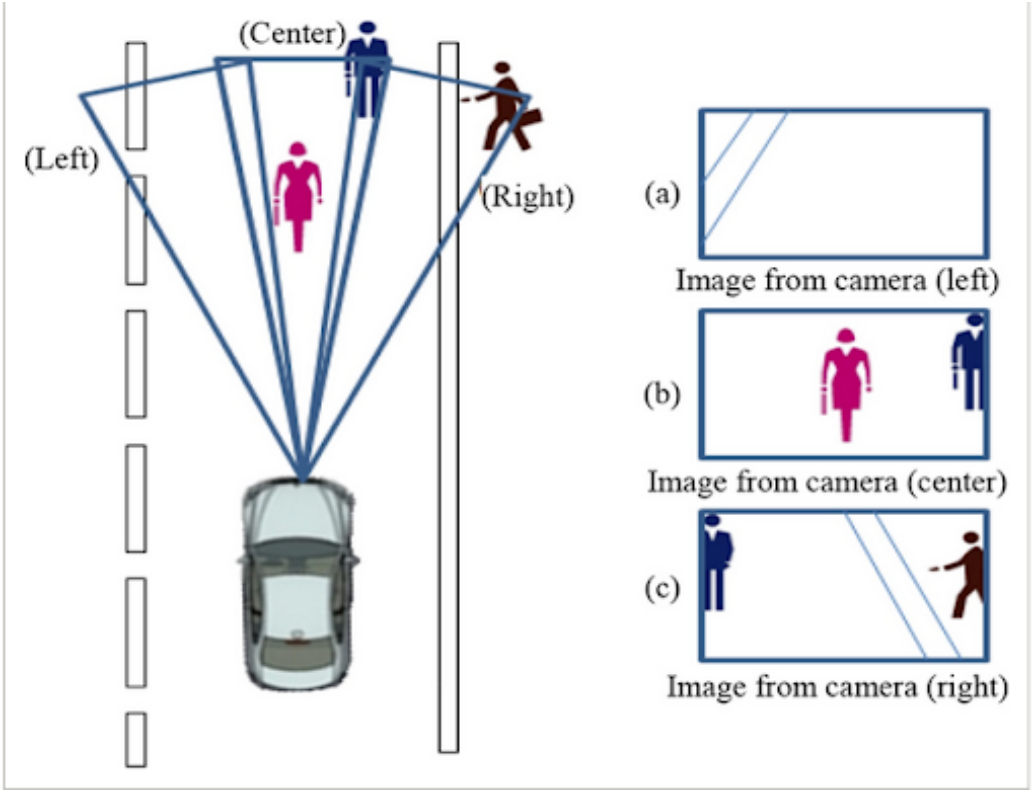

Figure 2: Array of standard cameras [4]

provide. The first scenario in Figure 1 depicts a situation in which a standard camera is able to view a pedestrian on the road, but does not see the pedestrian about to cross the road, due to the cameras limited field of view. To circumvent this issue, one might be tempted to utilize a wide angle lens, such as a panoramic lens. However, objects which appear in the edges of images collected by this lens have their distances distorted, which can also be seen in Figure 1. Alternatively, using an array of standard cameras can provide a wide field of view, capturing more information, while avoiding the image distortions caused by a panoramic lens. By joining the images produced by this camera array together in arrangement like Figure 2, we can increase the amount of information available to images processing systems.

2.2 Feature Detection

In order join images together, features from each image must identified and matched. This provides the required reference points for aligning the images together. A number of feature detection algorithms have been developed, and this paper explores several of them in order to determine the ideal feature detector for this solution.

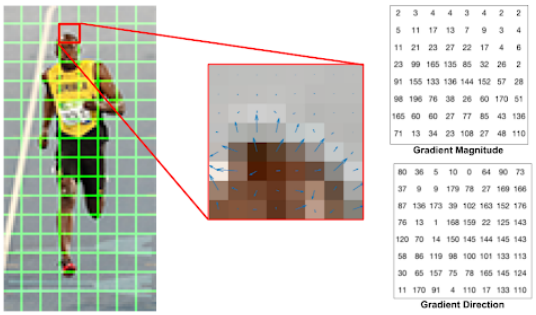

Figure 3: Illustration of gradients from [2]

We can generate a histogram of gradient descents, which serves as a baseline for distinguishing different features in an images. The major advantage of this process is that the magnitude and direction of the gradients in a given neighborhood are reduced into a 9 bin area, with each bin representing a rotational area, which allows for simpler processing when attempting to find matching features between images [2].

SIFT is a scale invariant feature detection algorithm proposed by D. Lowe, which utilizes Laplacian of Guassians to normalize the scale between features. Figure 6 depicts a high level overview of the stages required by SIFT to determine keypoints. Extrema detection utilizes the difference of gaussian function depicted in Figure 8 to determine keypoint candidates. To make this scale invariant, feature candidates are scaled down over a number of octaves, till they have a normalized scale. The keypoints stage removes the keypoints that are determined to be less useful. First, it removes keypoint which have low contrast with their surrounding areas. Next, it removes keypoints whose values fall below the threshold determined by the equation

in Figure 9. Finally it removes keypoints which are sensitive to noise, which are defined as those that have a large principal curvature along an edge, and a small one in the perpendicular direction [1]. Finally, orientation is determined for each surviving keypoint. An orientation histogram is generated by sampling neighborhoods within the image, and keypoints are assigned the orientation of the nearest peak that is within 80% of the highest local peak. This assignment has been proven to be 95% accurate [6], and makes the keypoints orientation invariant.

In addition to generating keypoints, descriptors are generated to make the keypoints more useful. These descriptors allow for comparison with other keypoints, and allow for to the detection of matching keypoints. This utilizes a process similar to the generation of a histogram of gradient descent described earlier. In this process, a neighborhood in the image is sampled, and its magnitude and direction are shrunk down into a 2x2 area. This process has the added benefit of making the descriptors to be orientation invariant. Additionally, the descriptors are modified by a constant to mitigate the effect of differences in lighting.

2.3 SIFT Results

In order to test SIFT's ability to generate image stitches, parameter sweeps were performed on the edgeThreshold, contrastThreshold, and distance in order to find the best combination. The end result of this testing showed that the default parameters proposed by Lowe [1] performed the best. SIFT produced good quality images most of the time. In Figure 11, we can see that SIFT finds a very large number of matching features, and these can be filtered to only include the n best features, as can be seen in Figure 12. Features are filtered out if they fall below a set threshold, determined by multiplying their distance with a threshold constant. By increasing the threshold value, lower quality features can be filtered out. Figure 13 is the result

of this operation, and while there is a visible seam in the resulting image, the overall stitch is good quality. Figure 14 shows a better stitch, which is likely attributed to a much more simple, but still feature rich environment. Further tests revealed that SIFT struggled to find many features on sections of images, an example of which can be seen in Figure 15. In this image, while plenty of features can be found on the building behind the truck, the truck itself is a low contrast area, so no matching features were found on the truck itself.

This resulted in a good stitch on the building in the region containing the building, but a poor stitch on the truck, which can be seen in Figure 16. Further experimentation revealed that the cause of this poor stitch was due to a misalignment between the top half of the image and the bottom half, which is evident in the manually alignment image shown in Figure 17. This issue likely also stems from the lack of keypoints in the bottom half of the image, since those additional keypoints may have helped find a stitch which correctly combined the images. This finding also revealed an issue with the dataset, as there is no guarantee that objects are in the same position relative to the camera between images. These stitches were performed without blending, which attributes to some of the visibles seams. OpenCV includes a stitcher class, which performs all the required stitching operations, including feature detection, matching, alignment, and blending. Using the stitcher class, Figure 18 was generated, which looks better than the normal stitch, but the lack of feature points still causes clearly visible distortions. Additionally, the stitcher class is very high level, taking in only the images, feature detector, and feature matcher, making it a poor resource for this project.

Table 1: Tool benchmarks

Feature Detectors	**Benchmark Timings**		
	Detection Time(s)	***Merge Time(s)***	***Total Times***
AKAZE	0.47	0.008	0.48
KAZE	2.54	0.009	2.55
ORB	0.48	0.007	0.49
SIFT	1.011	0.008	1.02

2.4 Benchmarks and Performance

Since this project's intention is to create a real time image stitching solution, it was imperative that processing time benchmarks of current feature detection and image stitching algorithms be determined, in order to choose the optimal feature detector for this project. The benchmarks from Table 1 were computed from generating batches of stitched images using each feature detector, followed by the use of a nearest neighbor algorithm to find feature points, generation of the homography matrix, and application of that matrix to stitch the images together. The average processing time for detection, merging, and total time were then computed and used to determine the processing time for these algorithms. These results show a few things. The first is that detection time takes the vast bulk of the processing time, with merge time being nearly negligible in comparison. Of these, AKAZE appears to be the fastest followed by ORB. In addition to processing time, quality of the stitch is vitally important to this application, as poor stitches can cause loss of the information provided by the stitches images.

Figure 4: Visualization of image gradients

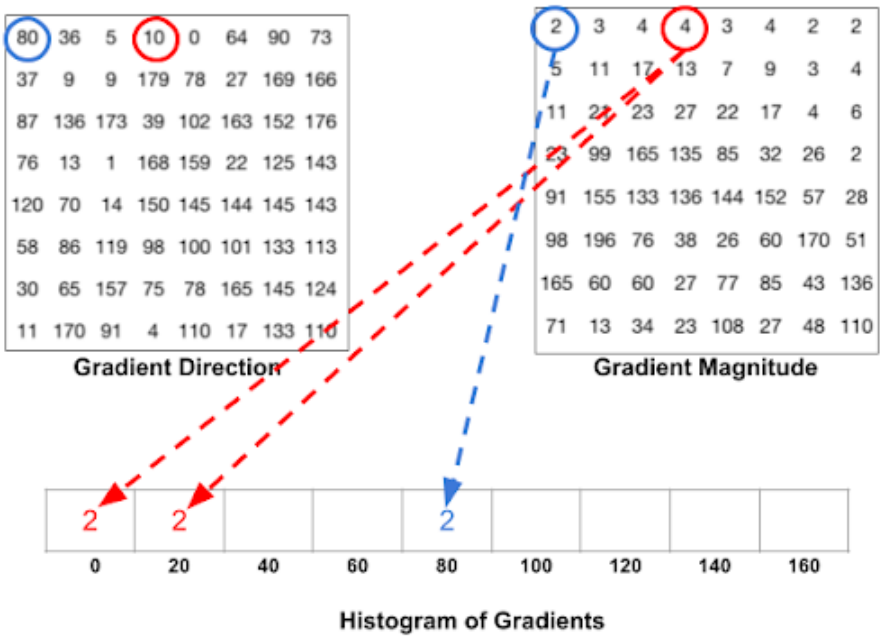

Figure 5: Visualization of histogram of gradient descents [2]

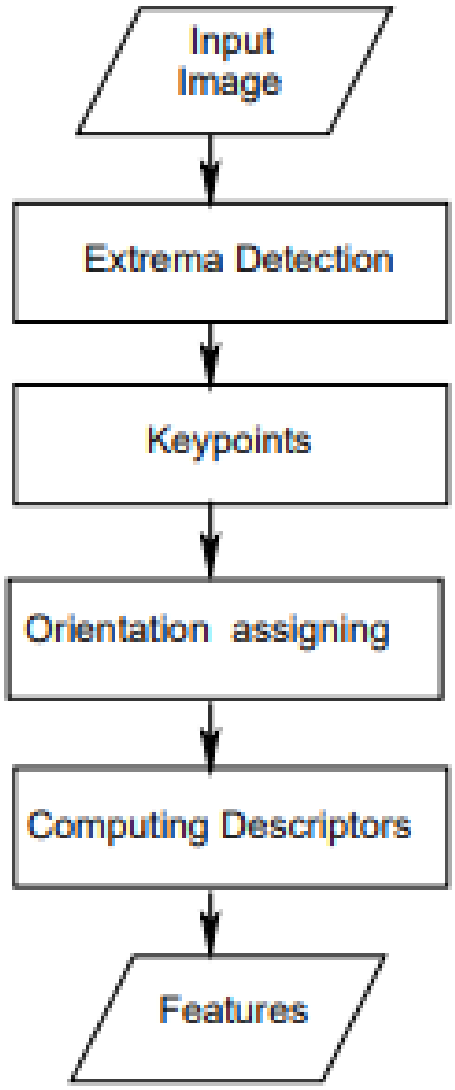

Figure 6: High level overview of SIFT [1]

$$\begin{aligned} D(x,y,\sigma) &= (G(x,y,k\sigma) - G(x,y,\sigma)) * I(x,y) \\ &= L(x,y,k\sigma) - L(x,y,\sigma). \end{aligned}$$

Figure 7: Difference of Gaussian function [6]

$$\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} < \frac{(r+1)^2}{r}.$$

Figure 8: Threshold equation for keypoints [1]

$$\mathrm{Tr}(\mathbf{H}) = D_{xx} + D_{yy} = \alpha + \beta,$$
$$\mathrm{Det}(\mathbf{H}) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta.$$

Figure 9: Sub-equation for Figure 8 [1]

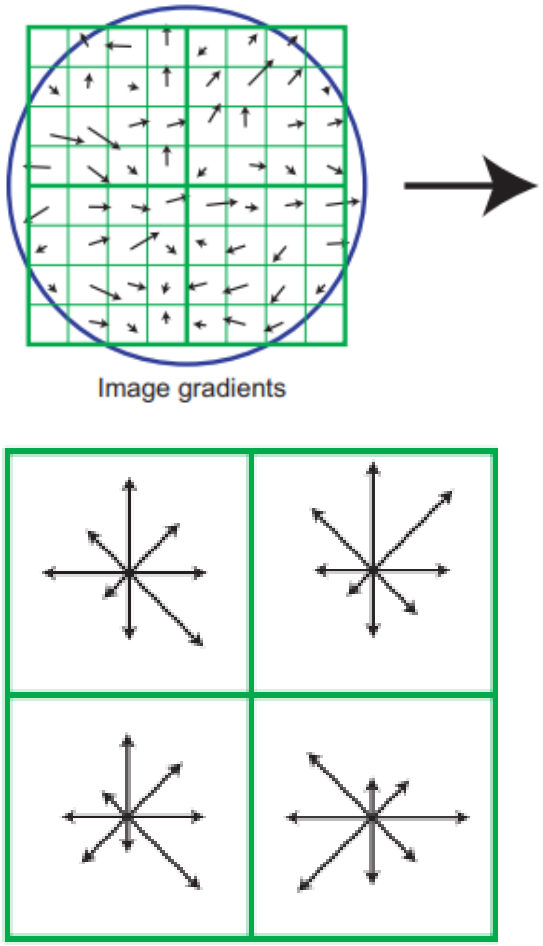

Figure 10: Illustration of descriptor generation [6]

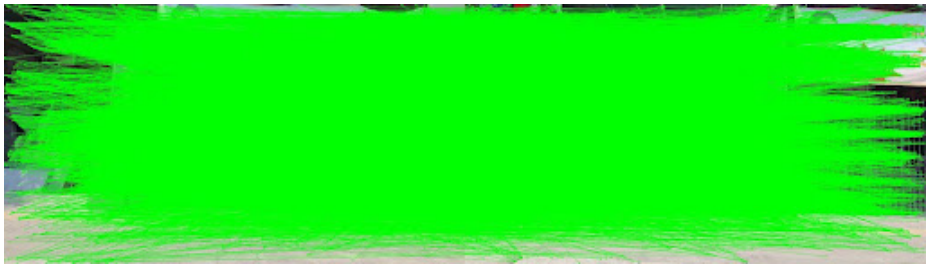

Figure 11: Unfiltered matching features

Figure 12: Filtered matching features

Figure 13: Stitched image

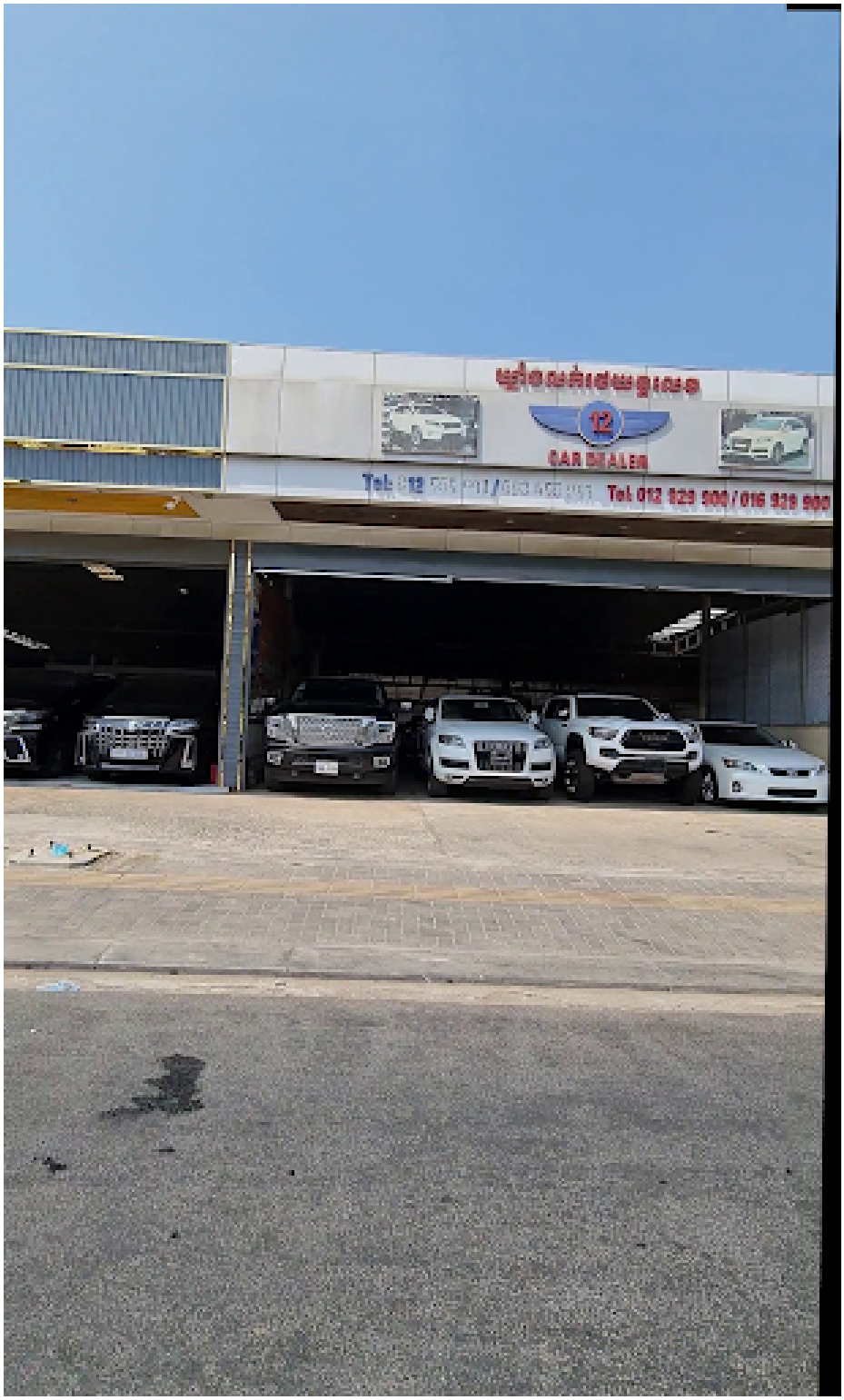

Figure 14: Stitched image

Figure 15: Matching feature pairs

Figure 16: Stitch on image of the truck

Figure 17: A manually stitched image

Figure 18: Image Stitch performed with stitcher class

CHAPTER 3

Related Works

Suk et al. [4] have proposed an architecture for real time image stitching, which utilizes both a software and hardware component in order to reduce the computational complexity required to match, stitch, and blend images together. Their approach has several improvements over traditional image stitching pipelines. The first of these improvements is the need to only generate the homography matrix one time, as generating the homography matrix is time consuming and computationally expensive. Additionally, Suk et al. found that generating the homography matrix for every frame created inconsistent results, as it relies on the RANSAC algorithm, which has a degree of randomness, and resulted in visible seams on some iterations of the same image pair. They purport 3 advantages to a fixed homography matrix: decreased computation complexity, seamless image viewing without blurring, and preventing mismatch errors in the pixel position between frames [4]. As can be seen in Figure

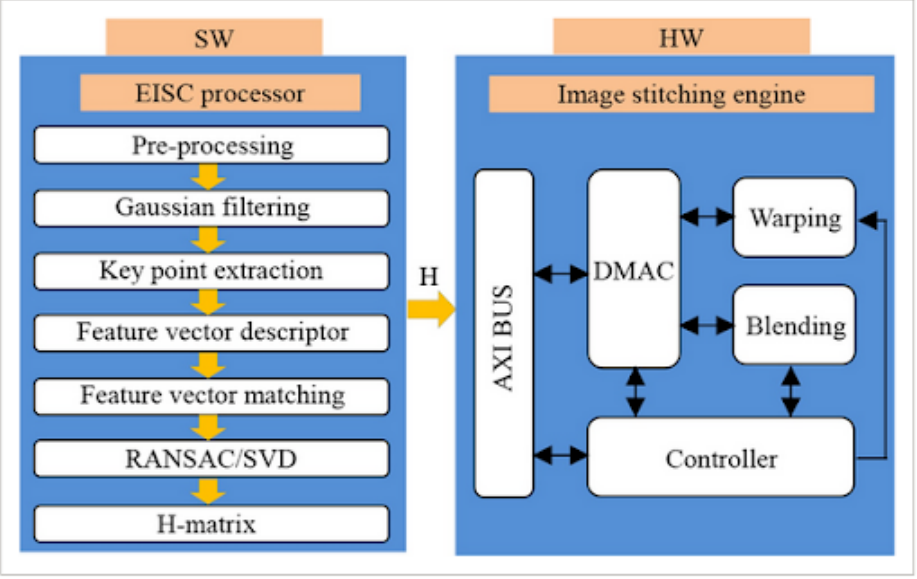

Figure 19: Software/Hardware architecture [4]

19, Suk et al's solution involves processing the homography matrix(H-Matrix) from

within the software component of their architecture.This follows a standard pipeline for producing the homography matrix. They then introduce a hardware component which utilizes the homography matrix to perform the blending and warping operation, and produce a seamless stitched image. In order to determine the features in the images, Suk et al. used SIFT as the feature detector because SIFT is good at finding a high number of robust features, with the trade-off of a long processing time. For this application, this processing time is acceptable since the homography matrix is only produced once, and therefore will have limited impact on the real time processing requirement. Taking the features generated by SIFT, they then utilize a nearest neighbor matching schema in order to find the corresponding features between the 2 images, generating matching pairs. To reduce the number of errors in the transformed image, RANSAC is applied to estimate the best homography matrix. RANSAC accomplishes this by removing outliers from the corresponding features through an iterative process which checks for consistency between those corresponding features

Task	Time	Features
Feature extraction for left image	10 s	764 features
Feature extraction for right image	9 s	484 features
Correspondence matching	2.3 s	196 pairs
RANSAC/SVD	217 ms	188 inliers
Total	21.517 s	N/A

Figure 20: Benchmarks using a 32 bit processor for SIFT [4]

Figure 22 illustrates the necessity of pre-computing the homography matrix. Without doing so, a real time solution is infeasible. Through the use of their hardware component, Suk et al. have been able to reduce the computation complexity of the

warping and blending portion of their application by 90% [4]. In order to blend the images, they utilize a cut graph, followed by alpha blending, which normalizes the pixel values on each side of the graph cut line, in order to smooth any existing seams.

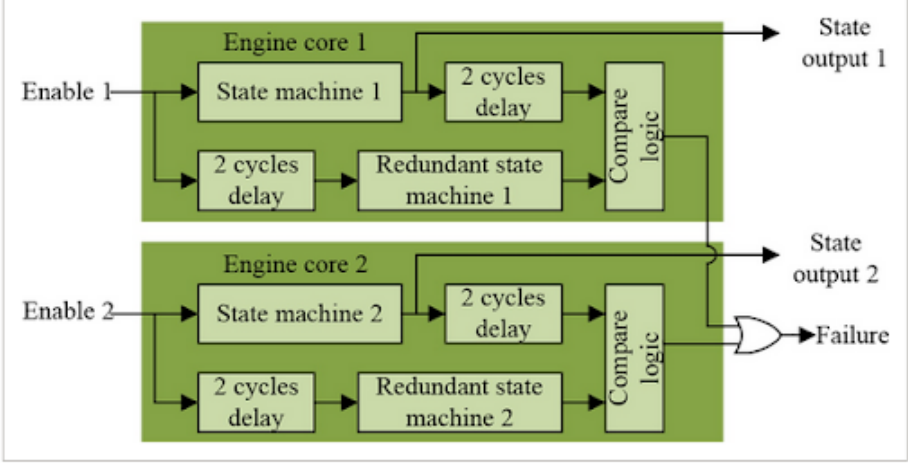

Figure 21: Redundant state machine for failure detection [4]

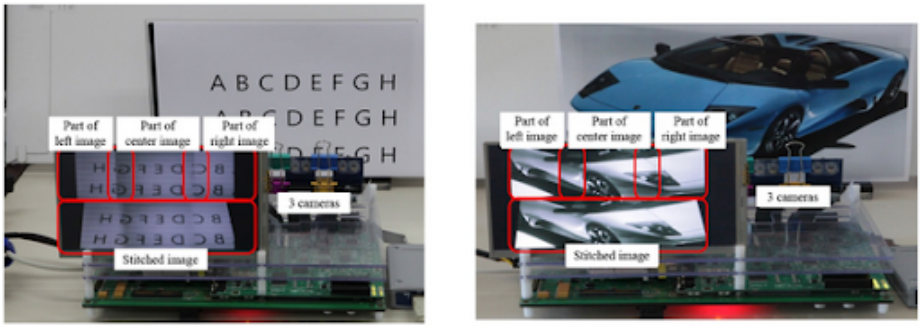

Figure 22: Experiment results from [4]

Looking at the experimental results in Figure 22, we can see that Suk et al's algorithm produces seamless stitches, while Figure 23 shows us that they were able to achieve real time results, at fps, using a low powered system. These results are useful since it allows for real time image stitching for such systems. The trade off here is that the input camera resolution is very low, and specialized hardware is required in order to achieve this real time solution While Suk et al. were able to achieve real time

Category	Features	
Tool	Synopsys Design Compiler TM	
Process	65 nm (GF)	
Operating clock	Max 333 MHz	
Engine size	Core 1	245,220 μm^2(127,514 gates)
	Core 2	245,104 μm^2(127,454 gates)
Performance analysis	VGA 44×3 fps @ 200 MHz (single core)	

Figure 23: Experiment results from [4]

image processing, the low resolution could potentially lead to a loss of information, as details within the image are more difficult to distinguish. Additionally, the hardware component of their architecture makes usage of their solution more restrictive. Due to this, their solution is incompatible with this thesis project. Fortunately, some ideas can be gleaned from Suk et al's project, which may prove useful for the development of a purely software approach to real time panoramic image stitching. The first and foremost inspiration is the need to only produce the homography matrix once. As will be discussed later in this paper, feature detection is the main bottleneck which limits image stitching for use in real time applications. This project is also currently not focused on a low computational cost solution, simply one that be perform in real time

CHAPTER 4

Proposed Solution

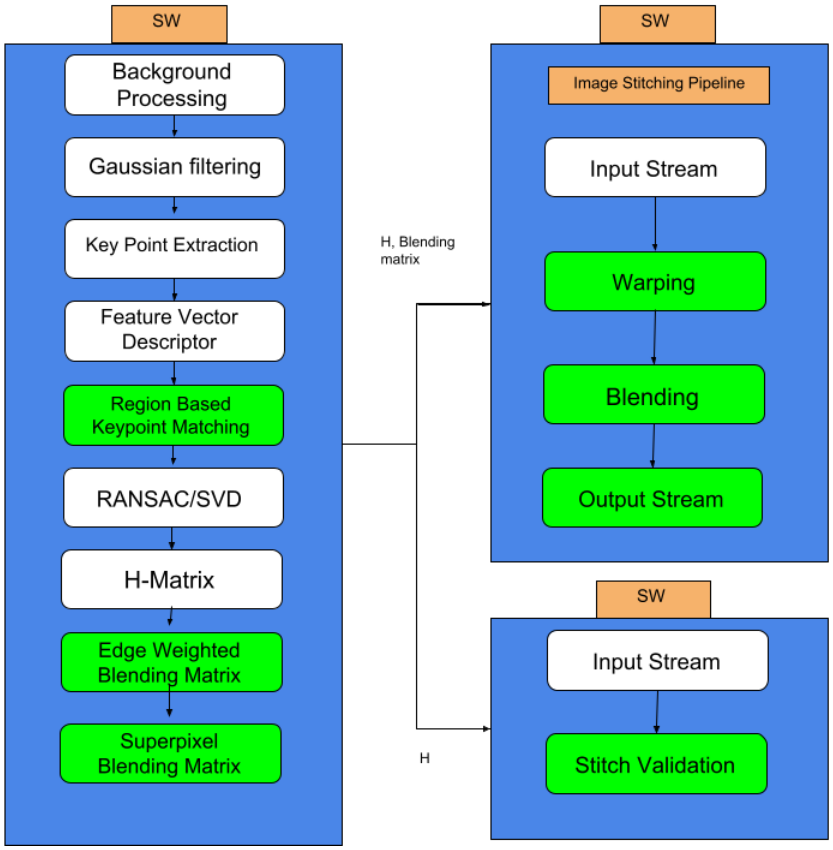

Figure 24: High level proposed architecture

In order to generate high quality image stitches in real time, a homography matrix must be generated from a set of keypoints. Then, image blending is applied to the resulting images to achieve a seamless result. The resulting aligned images are validated to ensure that the image stitch is high quality. Each of these components is packaged into a parallel processing pipeline, which utilizes a background processing stage to improve processing time. The proposed solution was tested and validated on the newly collected datasets.

Figure 24 depicts the high level architecture of the proposed solution. In this

diagram, the entries highlighted in green show where the proposed solution has deviated from the implementation of Suk et al. Their solution can be seen in Figure 19. The proposed solution further simplifies the processing by adding the generation of blending matrices to the background processing stage, as well as introducing two complimentary blending schemes. The proposed solution also eliminates the need for specialized hardware by proposed software components in its place.

The proposed solution makes the following assumptions:

- The camera positions generally do not change
- At least 4 matching keypoints exist between images
- The cameras are horizontally adjacent

Figure 25: Sample from new dataset

The collected datasets were generated from dash mounted webcams, which recorded at a 15fps at 1980x1080 resolution. OpenCV was leveraged in order to synchronize and save the recording. This setup allows for a large amount of overlap, and the cameras had a fixed position in relation to each other. One issue that impacted this dataset collection was that the webcams were unable to record at a consistent 30fps, which is what they are rated for. This caused the videos to be very sped up, as OpenCV would assume 30fps, regardless of what was actually recording. This

limitation forced the FPS to be limited to 15 for consistency. One dataset was recording during the day on a mountain road, while the other was recording on a road in rainy conditions. An additional dataset as recorder with the same settings, but in a different environment while it was raining.

4.1 Keypoint Selection for Homography Generation

A set of matching keypoints needs to be calculated to generate a homography matrix, but not all of these matches are accurate. To reduce the number of inaccurate matches, we propose a method of selecting only optimal keypoints through the use of horizontal subsets.

Generating an accurate homography matrix is essential in order for a fixed homography approach to provide a seamless image stitch. Traditionally, keypoints are detected and matched for the entirety of both images, then each set of keypoints is checked for matches with the keypoints from the other images. Of the matches found, only a subset of the found matches are required in order to achieve an accurate homography matrix. Additionally, for cameras that are horizontally aligned, it can generally be assumed that the corresponding keypoint match for any keypoint lies withing a similar location on the vertical axis. When comparing 2 images, only the overlapping regions have valid matching keypoints, and as a result, keypoints which lay outside of the overlapping region are not used, or can even introduce additional noise. Figure 26 illustrates different possible sets of overlapping regions, and this idea serves as the basis the region based approach to keypoint proposed by Qu et al. [10] selection. There are a few benefits which may come from reducing the number of keypoints. The first being processing time, as the a smaller set of keypoints will need to be searched for matching elements. Another is reducing the amount of bad matches, as it reduces that possibility of matches originate from non-overlapping areas.

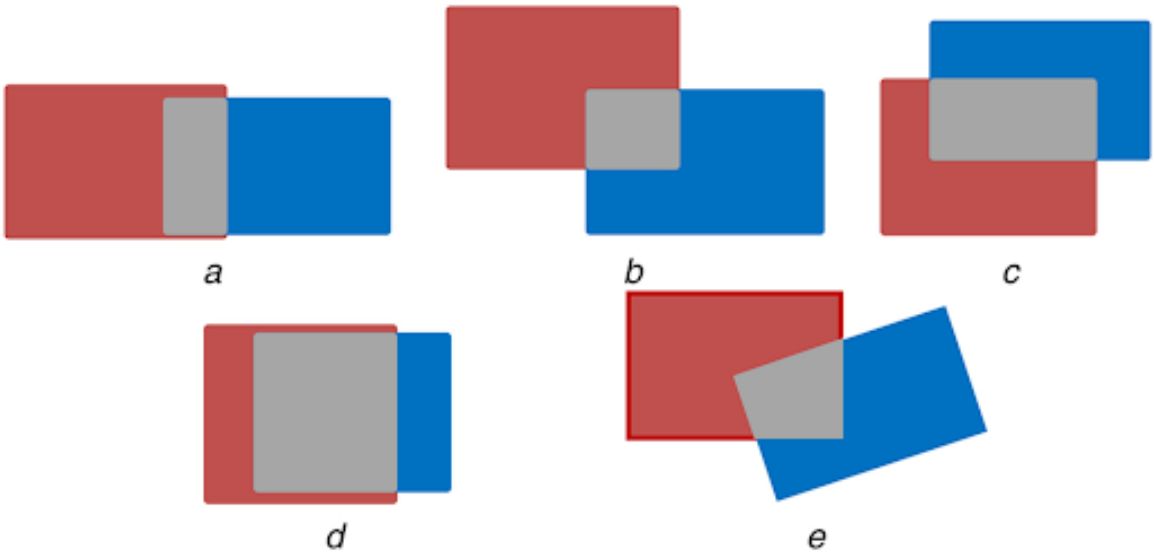

Figure 26: Examples of image overlap from [10]

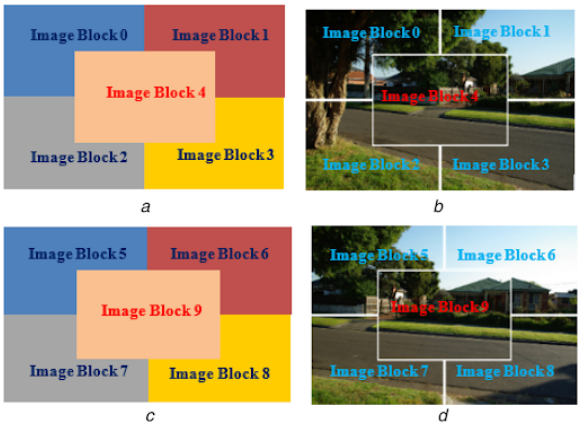

Figure 27: Image blocking scheme [10]

Their proposed blocking scheme can be seen in Figure 27, in which each image is divided into 5 equally sized blocks. Keypoints from each block are detected using the FAST feature detector. The block from each image with the lowest number of keypoints is dropped, in order to improve performance. Then feature matching is performed between each block, and the blocks from the second image, with the exception of blocks which share the same space, such as as Image Block 0 and Image block 5 from Figure 27. Using these sets of matches, the best pair of regions are the

ones with the most matching keypoints.

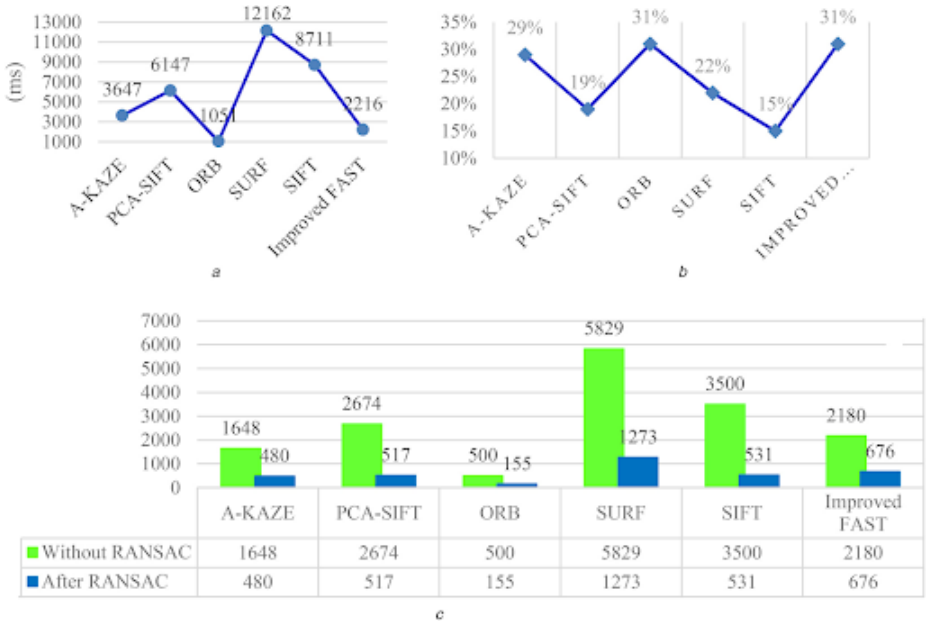

Figure 28: Experimental results from [10]

Qu et al.'s experimental results, shown in Figure 28, show that there is a significant performance increase associated with this region based keypoint selection approach, both in terms if processing speed, and in terms of correctness. Based on Figure 28, improved FAST, Qu et al.'s approach provides the second fastest processing time, with ORB being the fastest. Despite ORB being faster, improved FAST provides many more correct keypoints, with minimal processing time impact, making it a much more desirable approach.

Despite their excellent results, it is difficult to determine the effectiveness of their approach due to a number of issues. The first is that they have provided no experimental results for standard FAST. This leads to the question of whether or not the performance increases are the result of FAST, or if they are a result of the region based keypoint selection. Additionally, an approach like this is susceptible to pairs of images with sparse keypoints, as this scenario could lead to a situation where not

enough matching keypoints exist in a given region to produce a homography matrix. Finally, despite the significant speed improvements, this approach is still too slow for real time use.

While the technique proposed by Qu et al [10] still proves to be slow for the needs of this project, the region based approach provides a way to gather correct keypoints more effectively. Using the concept of paper [10] by Qu et al., a standard version of their approach was first implemented in order to see if qualitative improvements could be acquired.

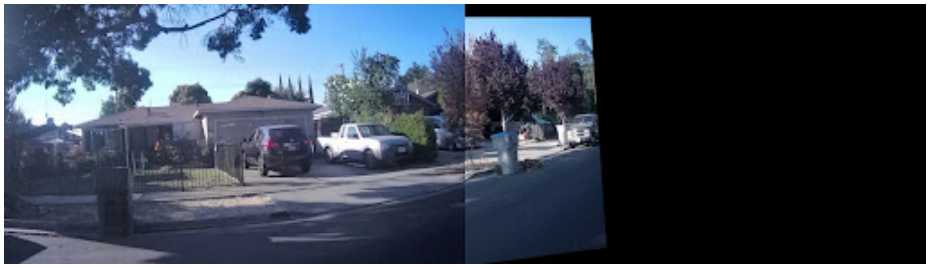

Figure 29: Image stitch from a homogprahy matrix generated from all keypoint in a pair of images

Figure 30: Image stitch from a homography matrix generated from regional keypoints

From figures 29 and 30, it can be seen that a slight qualitative improvement can be obtained by using a region based approach. This indicates that the homography being generated by the region based approach is more accurate than that of the standard approach. Figure 31 indicates that this region based homography generation is slower than the standard approach, due to the additional processing introduced

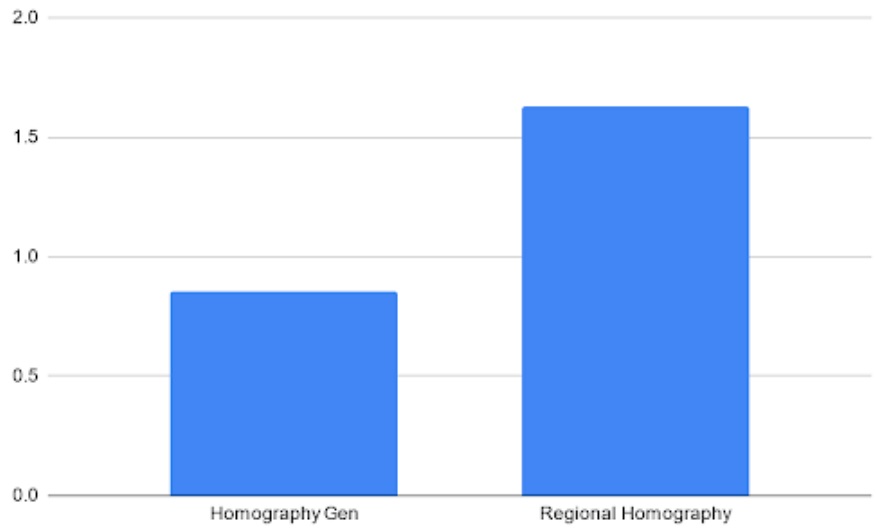

Figure 31: Processing speed comparison between standard and region based homography generation

by partitioning keypoints into regions. While this contradicts part of the result from Qu et al, [10] this is likely due to inefficiencies in the region partitioning of this implementation. Despite this slowdown, the qualitative improvements make this approach better than the standard homography generation. Since this portion of the processing is not needed to be run for every frame, the slower processing speeds is not a major issue.

To further develop this idea, alternative region schemes were experimented with, namely vertical and horizontal slices. Figure 32 depicts the new proposed schemes, with there being 50% overlap between adjacent regions. This overlap was implemented in order to improve the chances that an optimal region is selected, with a tradeoff of processing time. Each vertical slice contains keypoints from 20% of the image, while each horizontal slice contains keypoints from 25% of the image.

Experimenting with Vertical Slices, Horizontal Slices, and the Box regions from Qu et al [10] reveal that horizontal slices provide the best qualitative results. This

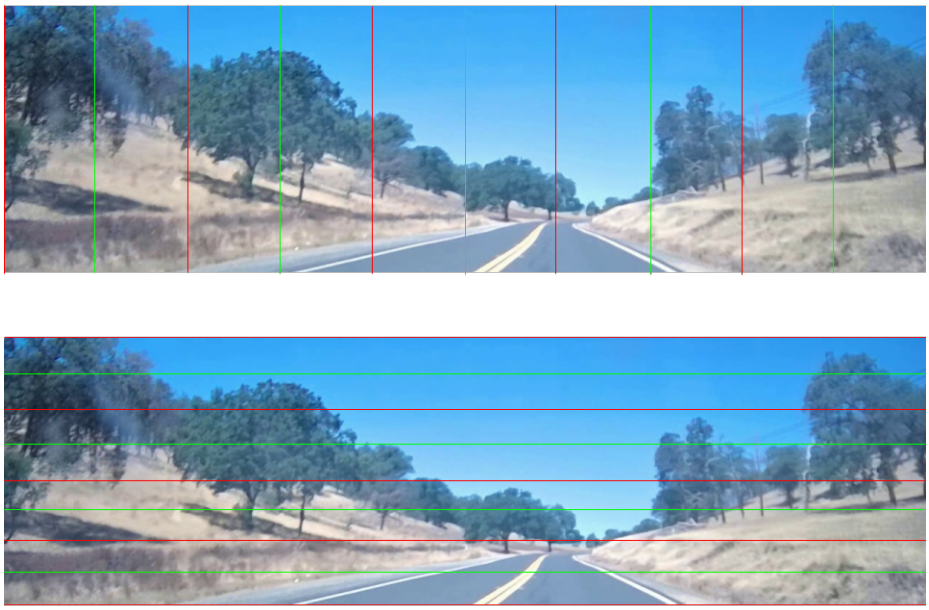

Figure 32: Vertical and horizontal regioning schemes

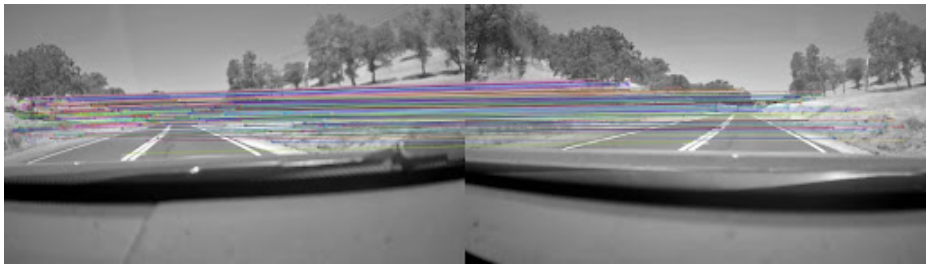

Figure 33: Matching keypoints for best horizontal slice regions

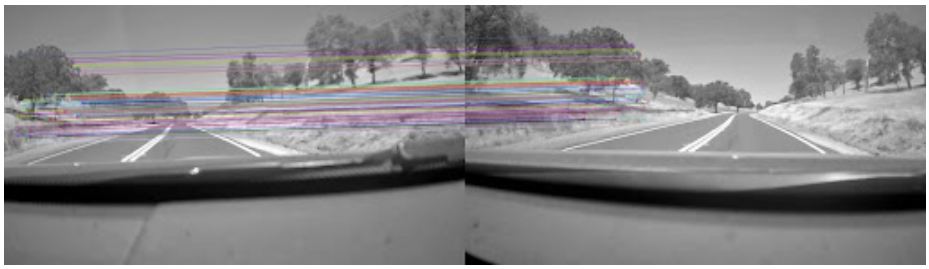

Figure 34: Matching keypoints for best vertical slice regions

makes sense since the cameras are arranged horizontally, so each horizontal slice is guaranteed to contain part of the overlapping area and keypoints should match with

Figure 35: Image stitch result from Vertical Slice Homopgraphy Matrix

Figure 36: Image stitch result from Horizontal Slice Homopgraphy Matrix

Figure 37: Image stitch result from horizontal slice, vertical slice, and boxes homography matrix

another keypoint on a similar location on the vertical axis. This way, the densest keypoint region of the overlapping are gets selected. This idea is illustrated in Figure 33 where the matching keypoints are densely packed and Figure 34 where the keypoints are sparsely arranged. It is hypothesized that for a vertical camera arrangement, that vertical slices would perform best, though further experimentation needs to done to verify this hypothesis.

Through the use of this region based approach to keypoint selection, the number of inaccurate keypoint matches is reduces, which in turn improves the accuracy of

the resulting homography matrix. The process ultimately generates a higher accuracy image alignment than previous approach, and supplies the high quality needed for use in a real time application

4.2 Edge Weighted Blending

Even with the improved homography matrices created by the regional based approach to keypoint selection, a visible seam still occurs within the resulting images. To remove this seam, we propose a new blending scheme, Edge Weighted Blending, that can remove this seam in real time through the use of a blending mask.

$$X = (columns - i)/columns$$
$$P(X) = L * X + R * (X-1)$$

Figure 38: Equation used for Edge Weighted Blending

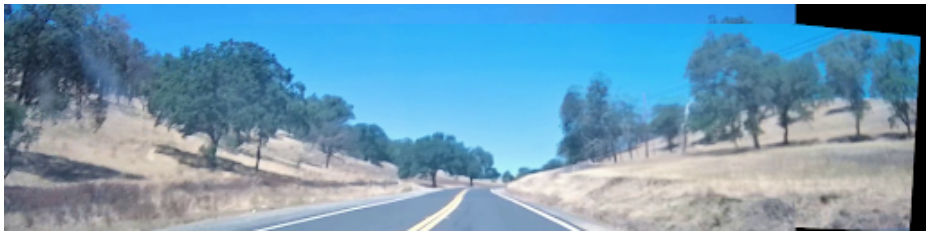

Figure 39: Result of Edge Weighted Blending

Edge Weighted Blending is inspired by the addWeighted function, but instead weighs the pixels based on which edge they are closest to. This helps eliminate the visible seam on the side of the image that was not favored, as it supplies a gradual transition between each image the stitch. By only blending the area directly around the seam, processing time is reduced and the output image is of higher quality. It was found that approximately 50 columns are needed in order to hide the seams around the blended area. An additional benefit of this approach is that it make the seams on the top and bottom of the blended region smaller, and thus less visible. From Figure

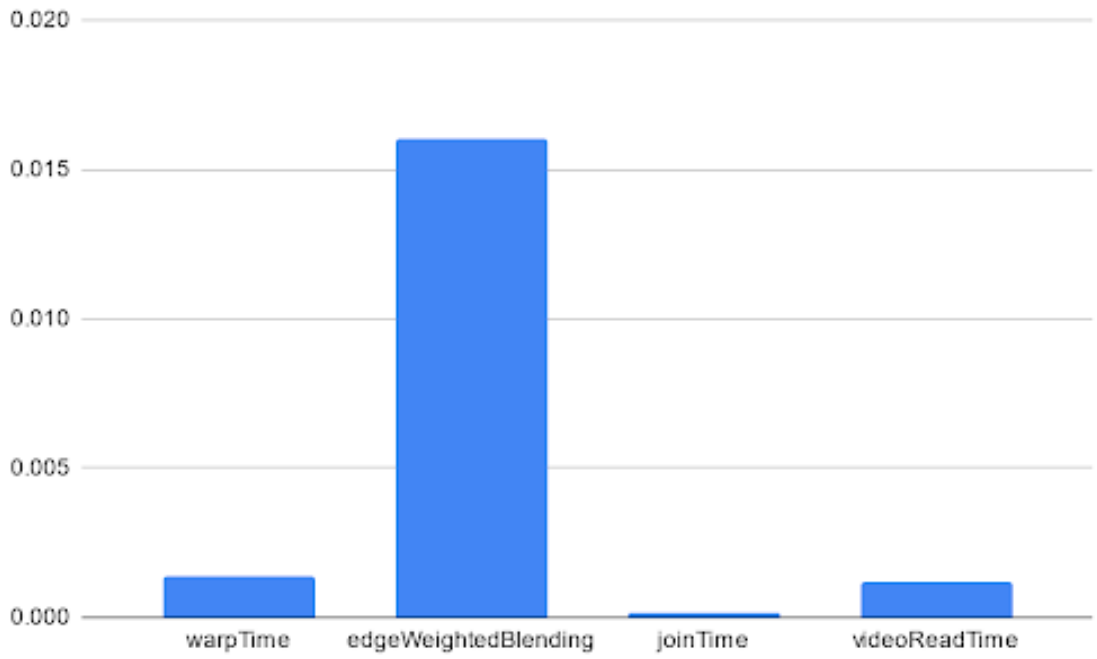

Figure 40: Performance of Edge Weighted Blending

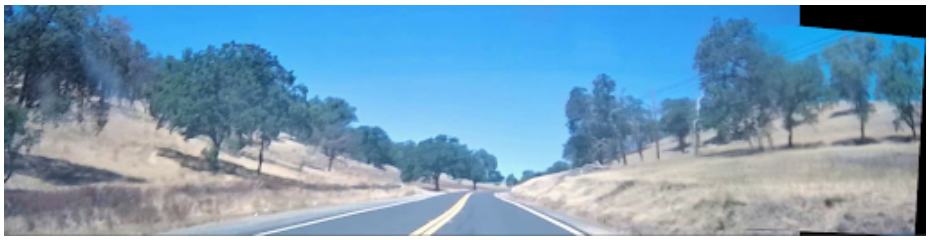

Figure 41: Stitch result from subset Edge Weighted Blending

41, we can see that the subset provides a good quality stitch, while Figure 42 shows that drastic performance increase provided by blending only a subset. Looking at Figure 43, it can be seen that this performance increase bring the blending operation almost perfectly in line with the rest of the pipeline.

By using this Edge Weighted Blending scheme, a nearly seamless result can be produced for any set of well aligned images. This can be done in real time by generating a mask of the Edge Weighted Blending result, providing a real time solution

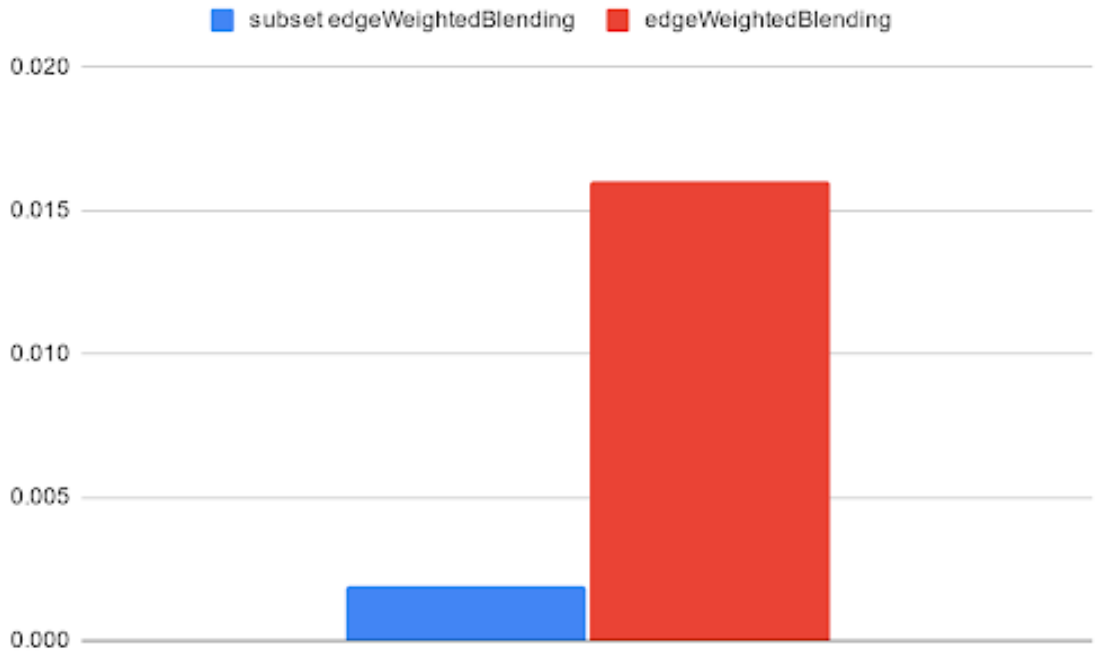

Figure 42: Performance comparison for Subset Edge Weighted Blending

that can provide seamless image stitches.

4.3 Superpixel Seam Detection and Blending

Edge Weighted Blending is able to smooth out the seam so that is is nearly nonexistent, but ghosting can occur if the blended area is not aligned well. To reduce the possibility of ghosting, we propose a method utilizing superpixels to determine the optimal position and shape for the seam.

In order to find this best path, we must use superpixels, using a technique proposed by Miao et al. [13]. In their paper, Mial et al. [13] describe the use of superpixels, a collection of similar neighboring pixels, in order to generate a less notable seam when stitching images together. This is accomplished by finding the a path of connected superpixels that are most similar.

In order to generate superpixels, openCV's Superpixel library was leveraged.

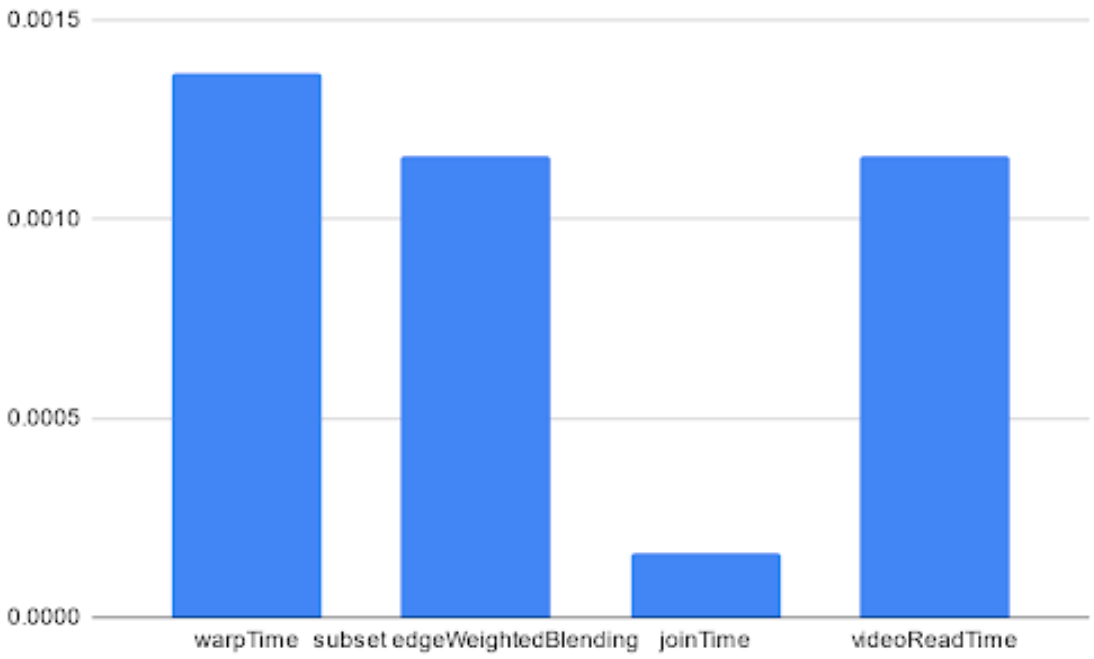

Figure 43: Pipeline benchmarks with optimized blending

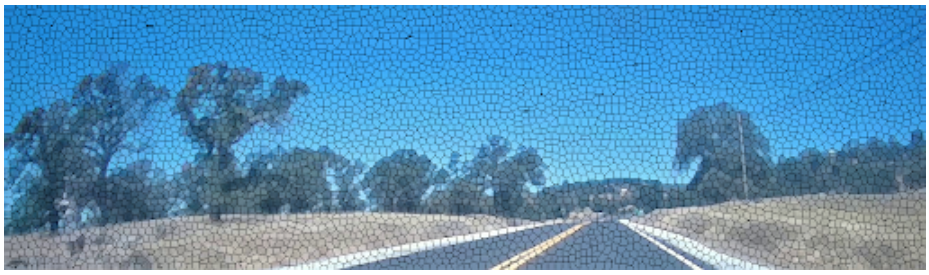

Figure 44: Superpixel example

$$W_s = \sum_{i=0}^{n} S1_i - \sum_{j=0}^{n} S2_j$$

Figure 45: Superpixel weight equation

Using the SLIC superpixel generation method, superpixels for one of the overlapping images are generated. These superpixels are then applied to the other overlapping image, with the idea that the superpixels should be identical for an image generated

from a perfect homography. From here, the weight of each superpixel can be generated. The weight is defined as the differences between the summation of the pixels values of a superpixel on an image, subtracted the summation of of the pixel values of the same superpixel on the corresponding image, depicted in Figure 45.

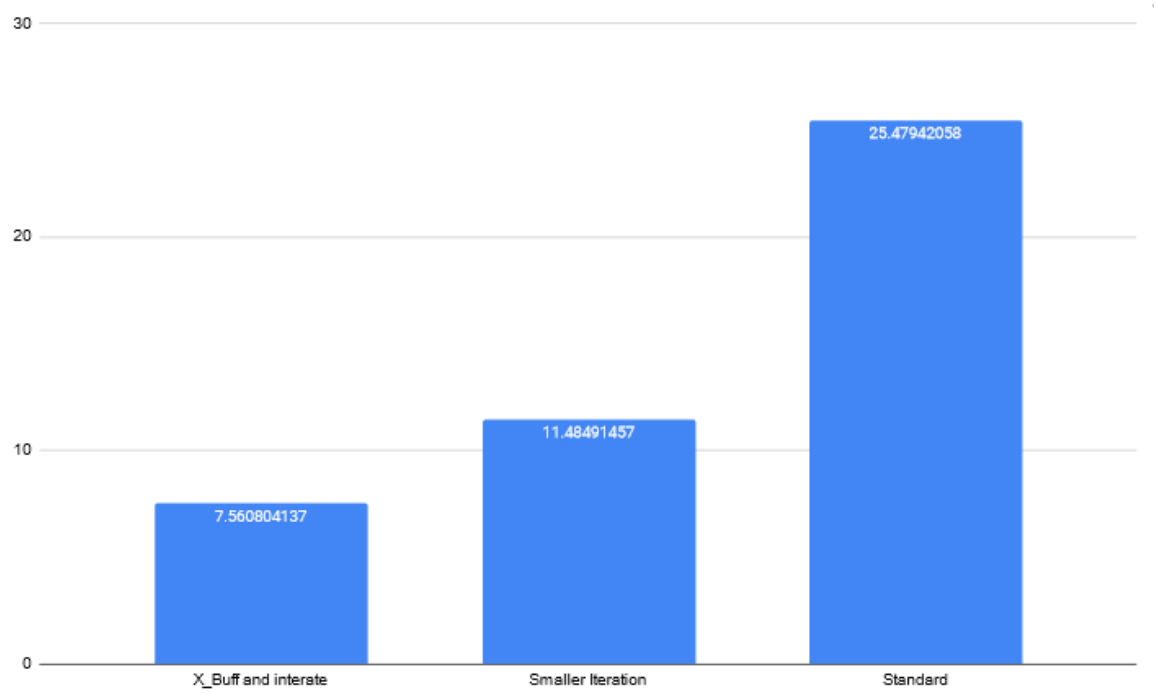

Figure 46: Runtimes for different iterations of the superpixel lowest cost path generation

The next step in the procedure is to generate the connections between superpixels. This is required to ensure that only viable paths are explored when searching for the lowest cost solution, and serves as the base framework for the search algorithm. Initially, valid candidates for the next step of a superpixel's path were any other superpixel below it. This allowed for the lowest cost algorithm to consistently calculate the best path, but it would occasionally cause the lowest cost algorithm to update for upwards of 50 seconds, with an average runtime of 25 seconds. In order to reduce the runtime, different approaches were attempted. The first successful approach was

to reduce the number of iterations when generating superpixels, which causes each superpixel to become less accurate. For the purposes of this application, that was determined to be an acceptable tradeoff, and resulted in a reduction in preocessing time of 11.5 seconds. Taking it further, it was hypothosized that superpixels that were mostly horizontally adjacent to another could be detected as a viable candidate for the next step on the path if part of the adjacent superpixel was below the one being evaluated. To remedy this situation, a horizontal buffer was introduced, which would remove any candidates that were only within the buffer region. The result of this was a processing time of 7.6 seconds.

$$C_s = \sum_{i=0}^{i} W_i$$

Figure 47: Equation of cost of path to each superpixel

To generate the lowest cost path, the cost of the path to each superpixel is calculated based on the sum of the weights of all superpixels on the path leading to it. The path starts at a superpixel at the top of the image. Additionally, only the lowest cost path to each superpixel is kept. Any path to a superpixel that is greater than the current lowest cost path for that superpixel stops processing and is discarded. The final result of this is a graph of superpixels with their lowest cost path and its associated cost. Finally, by finding the best seam can be found be selecting the lowest cost superpixel in the last row of the image. Using this path, a mask can be generated, show in Figure 48. This mask has the same blending described previously to smooth any remaining lightning differences from the stitch operation. The final result can be seen in Figure 49, and is nearly seamless. While the previously discussed Edge Weighted Blending is able to provide seamless stitches most of the time, the

Figure 48: Image masks for superpixel lowest cost path

Figure 49: Result of stitching using the superpixel mask

superpixel blending mask is able to ensure that a seamless result is found, and the result can be applied in real time.

4.4 Stitch Validation

We need to be able to determine that the condition that the cameras positions generally do not change holds true. To accomplish this, the proposed method generates a difference score by comparing the gradients of the aligned images.

If two images are aligned perfectly, the sum of their absolute difference would be close to 0. Differing lighting causes a greater difference score, regardless of how well aligned the images are. To remove the increased noise caused by lighting differences, gradients for each aligned image are calculated. These gradients contain only the strong edges that exist within the images, an example of which can be seen in Figure 50. We can then determine how well aligned the images are by comparing the edges in each image. This is done by calculating the sum of absolute differences of the aligned images. The score represents how different the images are, with scores closer to 0

Figure 50: Gradient of image pair

being more similar. This score is then compared against a set threshold. The camera position condition is considered to be invalid if this threshold is exceeded, and action is taken to reestablish the condition.

This technique proved to be the most effective at detecting camera shifts, and was the only one sensitive enough to detect small differences in the overlapping area. Due to his increased sensitivity, this gradient based comparison proved to be the most reliable way to validate the image alignment. Through the use of this software component for validating image stitches, the need for the redundant state machine used in the solution of Suk et al. is completely eliminated. This makes it so that the stitch validation component is usable on a much wider variety of systems, as it is not reliant on specialized hardware.

4.5 Background Processing

Certain steps of the image processing procedure are too slow for use in a real time application. To reduce the impact of these steps, we propose a method which reuses the results of these steps.

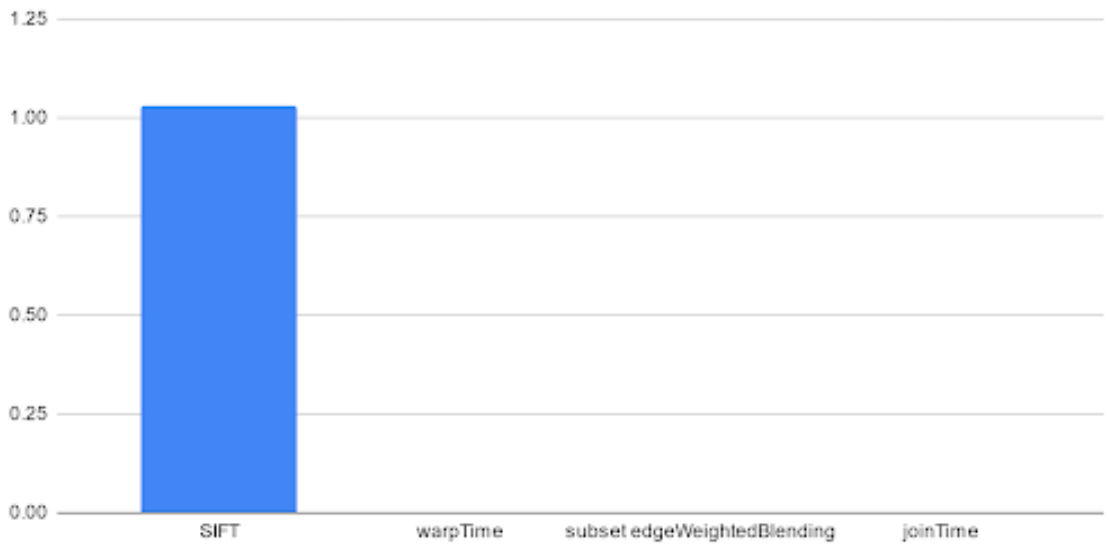

Figure 51: Comparison of processing times for image stitching

The background processing procedure, depicted in in Figure 53, consists of any processing which performs work that only need to be completed once per solution. This solution was inspired by the fixed homography approach used by Suk Et Al. [4]. In most situations, this procedure will not need to be run. On startup, the previous valid homography and blending matrices can be read from a stored location in order to ensure there is no delay in operation at startup.

For background processing, a set of frames from the input stream is taken and used to generate a homography matrix. If the homography matrix generation fails, the Validate Homography step triggers for the procedure to restart with a new set of frames. If the homography matrix is successfully generated, a blending matrix is created based on a supplied seam size. The homography and blending matrix are saved so that they can be accessed by the the blend and stitch operation. This is considered the Primary Processing step, as it is required for the stitching operation and provides a solution that is often good enough. The Secondary Processing step is

separated due to its longer processing time. These steps take on average 7.56 seconds to complete, though will occasionally take up to 20 seconds to complete. To avoid a deadlock in case the background processing needs to be restarted while this step is being performed, checks have been implemented which will end Secondary Processing so that Primary Processing can be rerun. The Secondary Processing consists of the procedure to generate the superpixel blending mask.

4.6 Parallel Pipeline

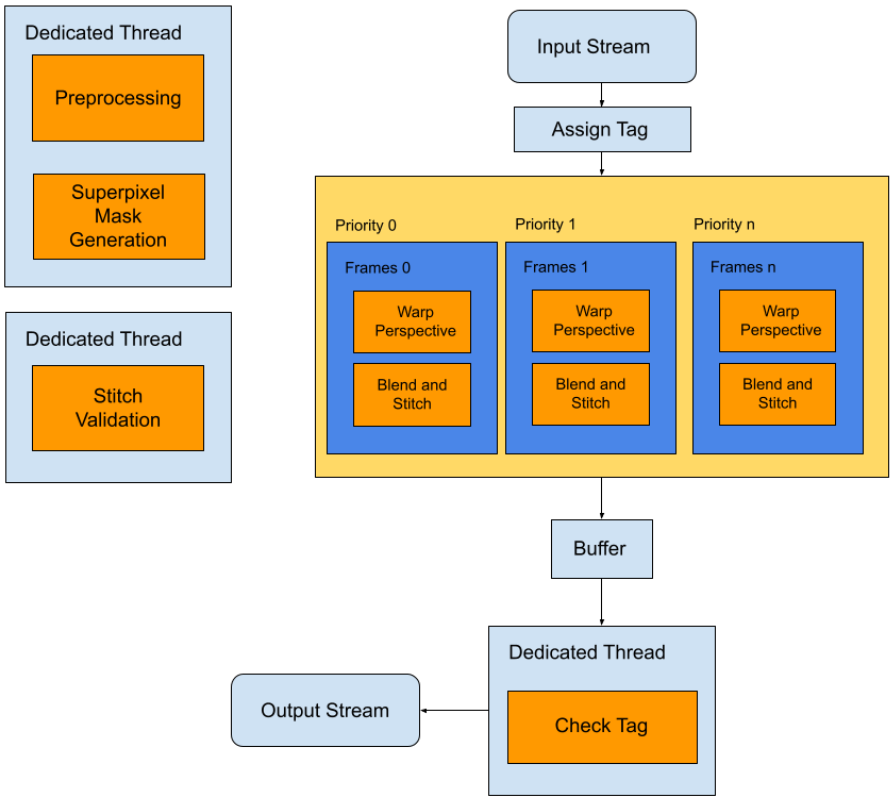

Figure 52: Finalized parallel pipeline

To allow for the proposed application to work in real time, the previously discussed components are run in parallel. This parallel processing pipeline is able to generates

the homography and blending matrices in the background processing step, and is able to apply them in real time. Alongside these processes, the realigned images are validated to ensure stitch quality.

In order to produce real time, high quality image stitches, the architecture in Figure 52 is proposed. This architecture is able to process multiple video streams, and stitch them together in real time. In order to accomplish this, certain tasks must be relegated to dedicated thread, and must be run only when needed. These are tasks which had too long of a processing time for use in a real time application, but whose function was not needed for every frame, allowing for this real time application to be feasible. Additionally, another separate thread is dedicated to validating the stitch result, in order to ensure that the current homography solution remains valid.

Using the results from the background processing stage, images from the input stream are stitched and blended together to achieve a single larger image. This is accomplished by applying the generated homography matrix to the adjacent image from the input stream, modifying it so that it its perspective matches that of the first image. After, the blending matrices are applied to each corresponding image to smooth the seam and create a higher quality stitch. This result is tagged with a frame number and sent to the buffer's ordered queue. The buffer checks to see if the image of the image at the top of the queue is the next expected image. If it is, then it is output to the output stream, otherwise it waits for the correct images. If the expected image does not appear after a timeout duration, the tag of the image at the top of the queue becomes the the next expected tag and processing continues as normal.

Our background processing procedure is inspired by the work of Suk et al. [4], specifically the software component of their architecture from Figure 19. Our solution deviates from the software implementation of Qu et al. in two key areas. The first of these areas is Keypoint matching. While Qu et al. obtains features from the entirety

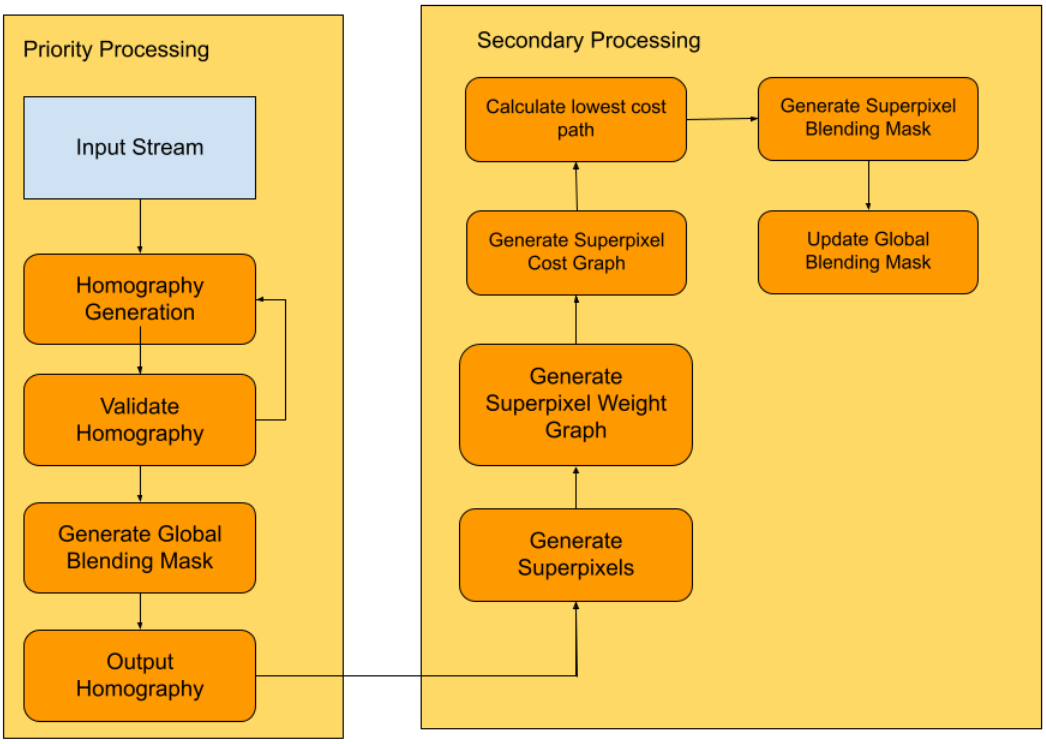

Figure 53: Background processing procedure

of both images, our approach narrows the search down to the horizontal regions with the most matching key points. through this process, we are able to improve the quality for the resulting homography matrix. The second area is through the addition of the generation of the blending matrix within the background processing stage. By including the creation of the blending matrix in the background processing stage, the processing time of the image stitching operation for each frame is significantly reduced.

In order for a real time system to be considered reliable, it must be able to handle failure scenarios accurately and quickly. For the system presented by this paper, this means that the system must be able to detect when the current homography and

blending solution is not longer valid, and generate a new solution in a timely manner.

Its impossible to ensure that the cameras never shift during operation, though it isn't expected to happen often. To validate the integrity of the stitch, a gradient based comparison is proposed. By calculating the sum of the absolute difference of the matrices, a difference score can be obtained. Through experimentation, it was found that a difference score of 0.1 serves as a good threshold for consistently detecting mismatch homography, while avoiding false positives. If a mismatch it detected, a flag it set to re-trigger reprocessing.

Qu et al. accomplishes fault detection via the use of a redundant state machine, which can be seen in Figure 21. By creating a software based fault detection system, we have eliminated the need to specialized hardware for fault detection. Since image stitching is also achieved at a software level, the need for specialized hardware is completely eliminated. This allows for this architecture to be usable on a wider variety of systems.

CHAPTER 5

Experiments

In the process of creating the proposed solution, a number of experiments were performed to determine the optimal techniques for generating high quality image stitches in real time. Each of these experiments led to the creation of the components of the proposed solution, but had disadvantages that made them poor candidates for the solution.

5.1 Randomized Subset Key point Selection

Not all keypoint matches detected are correct, and incorrect matching reduce the quality of the homography matrix. Due to the high number of keypoints generated, a randomly selected region should have enough keypoints to generate a homography, and should contain more valid matches.

Table 2: Randomized approach benchmarks

	Benchmark Data		
α	***1***	***0.5***	***0.3***
Img1 detect(seconds)	0.29	0.32	0.26
Img1 compute(seconds)	0.26	0.26	0.25
Img1 keypoints	14880	14880	14880
Img2 detect	0.32	0.11	0.02
Img2 compute	0.004	0.002	0
Img2 keypoints	13207	5568	385
Keypoint shift(seconds)	0.04	0.02	0
Match time (seconds)	0.47	0.21	0.02
Number of Matches	507	360	7
Filter Time	0.005	0.004	0.002
Stitch Time	0.007	0.007	0.007
Total Time	1.63	1.12	0.76

Image stitching algorithms require a minimum of 4 matching keypoints and feature detection algorithms, such as SIFT, often find far more matching features

than needed. This can be seen in the $\alpha = 1$ results in Table 2, which represents a standard stitching operation, where 507 matching features were found. Since only 4 matching features are needed, a valid stitching solution can be generated using only a subset of the matching features from a set of images. By selecting a subset of the input images, the number of bad matches can be reduced, improving the resulting homography matrix.

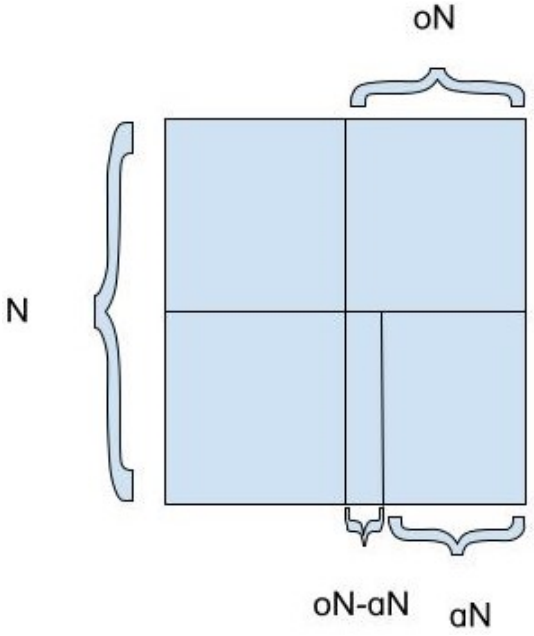

Figure 54: Illustration of subset area selection

$$Pr(X = 1) = o - (o - a)^2$$

Figure 55: Probability equation for a valid subset

To select the subset image, we randomly choose an area within the image with a predefined shape. This sub area can be defined as having a size of N where is a constant value less than or equal to 1. Additionally, we define the overlapping region as oN, with o representing the percentage of overlap within the image. We can

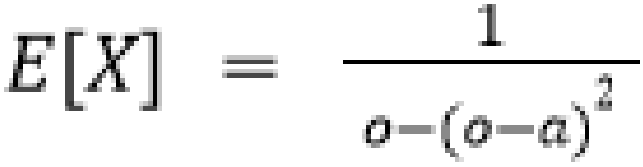

Figure 56: Expected number of runs before a solution is found

then calculate the probability that the selected sub area will reside entirely inside of the overlapping area using the equation in Figure 55. Batches of 100 samples were produced using this random approach, using values for α of 0.7,0.5, and 0.3 and with an estimated overlapping area of 90%. Each of these tests found that only 1 run was needed on average in order for enough keypoints for a stitch to be found. Using the equation in Figure 56, we can find the expected number of runs to be: 1.16 for α 0.7, 1.35 for α 0.5, and 1.85 for α 0.3. While the expectations α 0.7 and 0.5 appear to be in lane with the experimental results, the expectation of α 0.3 is exceeded by experimental results. This may indicate either an incorrect assumption was made when determining the probability equation, or that the overlapping area was incorrectly estimated.

The results in table 2 indicate that there is a processing time benefit as α becomes smaller. These results also highlight the fact that feature detection takes a significant portion of the processing time, and by limiting the area being sampled for feature detection, processing time can be significantly reduced. One interesting observation that can be taken from the results in table 2 is that the descriptor generation is not impacted, but matching time is improved. Matching time is likely improved due to the reduced number of total matching features that are found. From Figure 59, we can see that many matching features can be found, even when only selecting them

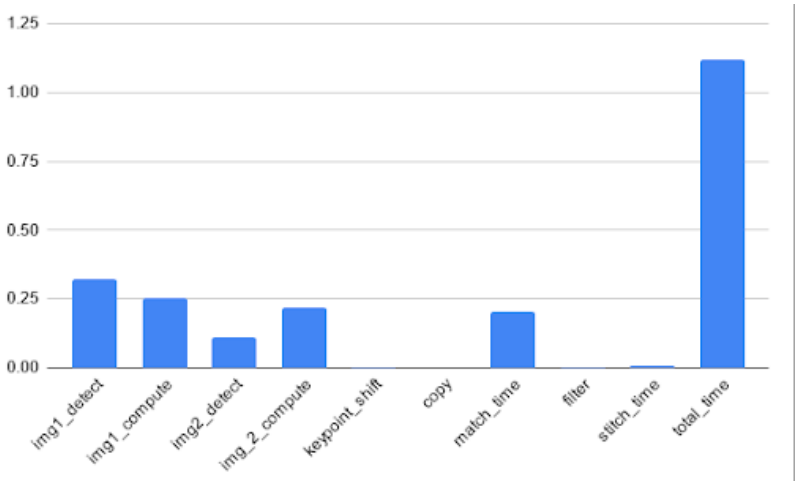

Figure 57: Processing time analysis for $\alpha = 1$

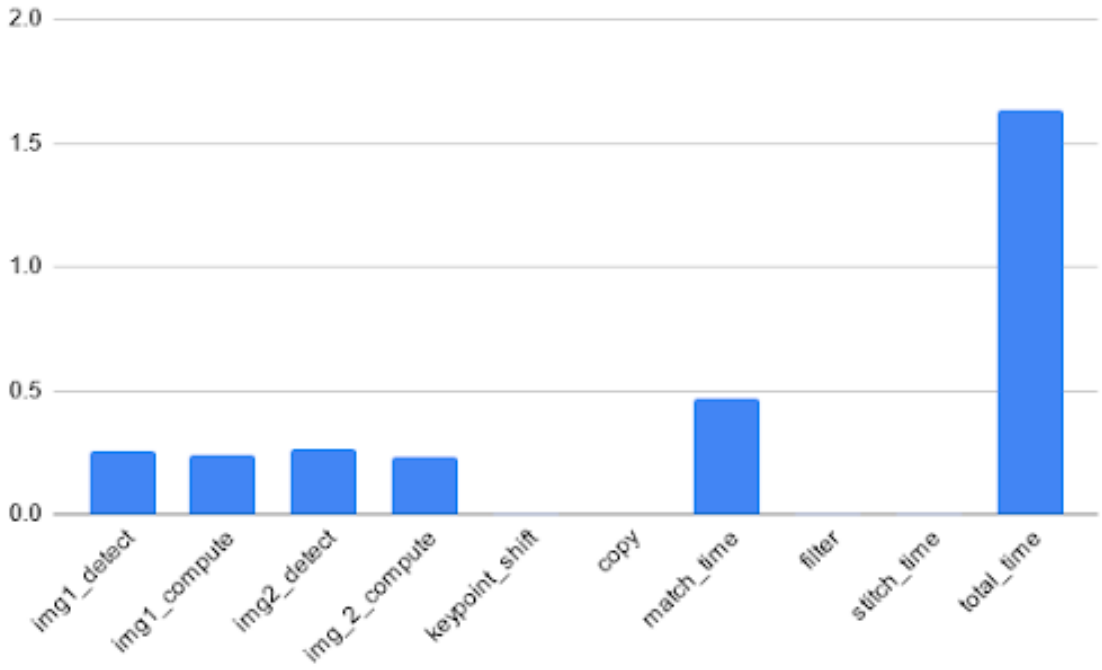

Figure 58: Processing time analysis for $\alpha = 1$

from a subset from one of the images. In addition to this, Figure 60 shows a completed stitch image, which has a similar result to that of a standard stitching operation seen in Figure 61. One additional observation that was made is that the randomized approach is more heavily impacted by a low overlapping area. In the dataset used,

Figure 59: Matching feature points using a subset of the image

Figure 60: Stitched image using randomly sampled subset

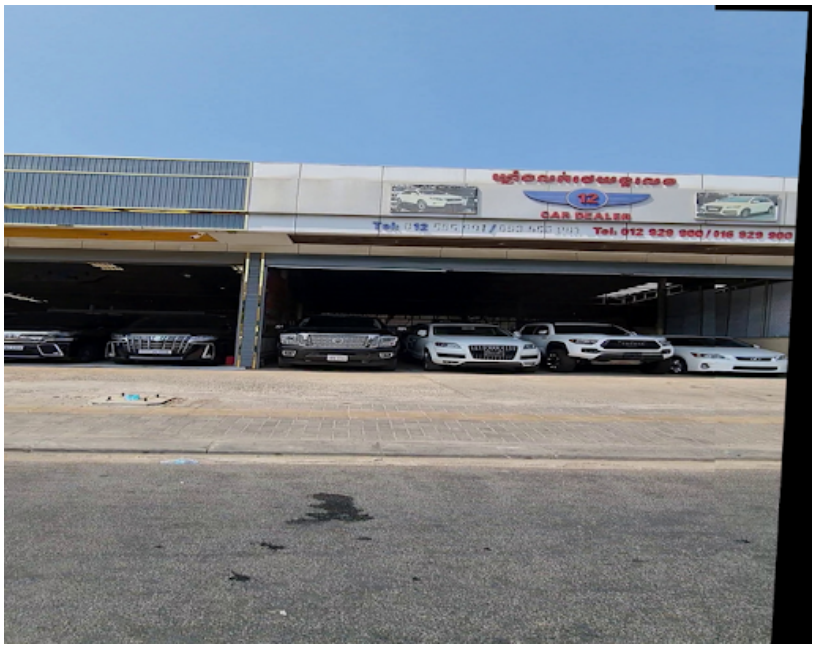

Figure 61: Stitch produced from a standard stitch operation

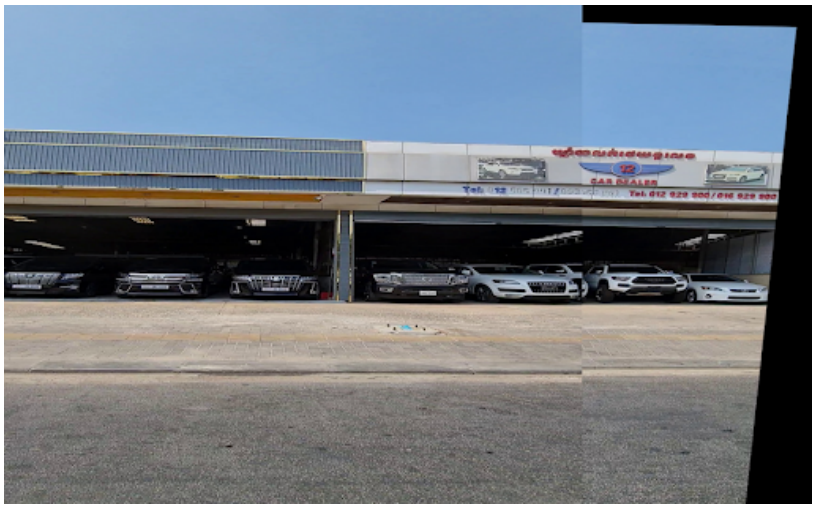

Figure 62: Standard stitch with a small overlap

images have less overlap, the further apart they are. Figures 38 and 39 represent image stitches where 1 image of separation was used.Figures 40 and 41 have 3 images of separation and this large separation leads to a smaller overlap, and creates a lower quality stitch. This is likely a symptom of having enough keypoints to achieve a

Figure 63: Randomized stitch with $\alpha = 0.5$ and low overlap

stitch, but not enough to compensate for mismatched keypoints

Unfortunately, despite the improvements to stitch quality seen in Figure 60, the inconsistently experienced by the randomized approach make a poor candidate for a real time system. Instead, the regional based approach used in the proposed solution was able to obtain higher quality image stitches, without the incosistant performance of the randomized approach.

5.2 Image Blending

When stitching images together, a visible seam is often still present. To eliminate this seam, we experimented with openCV's image blending functions.

Blending images helps reduce the visible seam in order to create one cohesive image. This is accomplished by taking the average pixel value for each pixel in the overlapping region. In Figure 64, it can be seen that the diagonal lines of the seam cause very visible seams and tears where the images overlap, but the right side of image has a smoothed over seam. To mitigate this issue and achieve a better blend, straight lines needed to be enforced, and the resulting image can be seen in Figure 65. This is a much better result then the one show in Figure 64, though some of the seam

Figure 64: Blended image

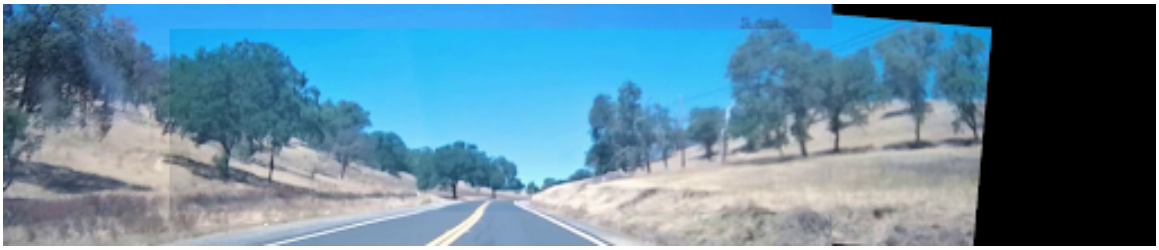

Figure 65: Blended image with straight lines enforced

is still visible.

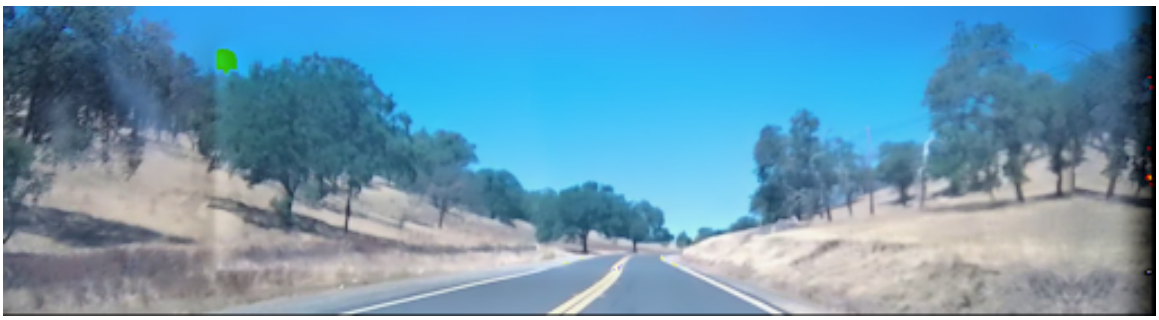

Figure 66: Multiband blended images

Other blending schemes are available in OpenCV, for example MultiBand blending. The resulting image seen in Figure 66 has some artifacting and a visible seem on the

left side of the overlap. Additionally, in Figure 67 we can see that multiband blending

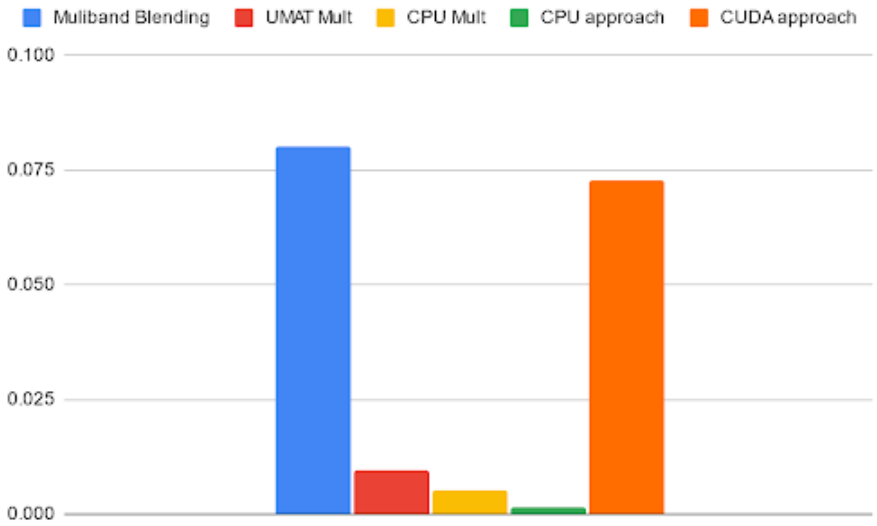

Figure 67: Benchmarks for blending operations

is very slow, with the CPU approach actually yielding the fastest result.

Ultimately openCV's native functions failed to be adequate for producing high quality image stitches in real time. Multiband blending produced artifacts in the output image. Using the add weighted functions was able to stitch the images together quickly, but experienced heavily darkened regions where one of the input images had black regions. These limitations made the creation of a custom blending scheme necessary, and led to the creation of Edge Weighted blending. Edge weighted blending was able to overcome these limitations to produce a seamless image stitch in real time.

5.3 Machine Learning Based Stitch Validation

To ensure that the condition that the cameras do not change position is valid, we need to be able to detect when the condition has failed. In an attempt to find a solution for this, experiments were performed using BRISQUE and machine learning.

Brisque is a technique which leverages SVC and SVR in order to provide a quantitative score for input images. This approach aims to determine whether or

Figure 68: ORB with score of 61.19, worst score

Figure 69: Score of 6.89, best score

not images contain "Natural scenes", which are defined as any image taken with an optical camera [9]. The theoretical basis for this approach comes from the idea that the Mean Subtracted Contrast Normalized Coefficients have statistical properties that are affected by distortions in the image [9]. Mital Et al. supplied a petrained SVC and SVR model for us with their technique. The closer the score is to 0, the better the score is. All generated stitched images were tested using this model,and returned

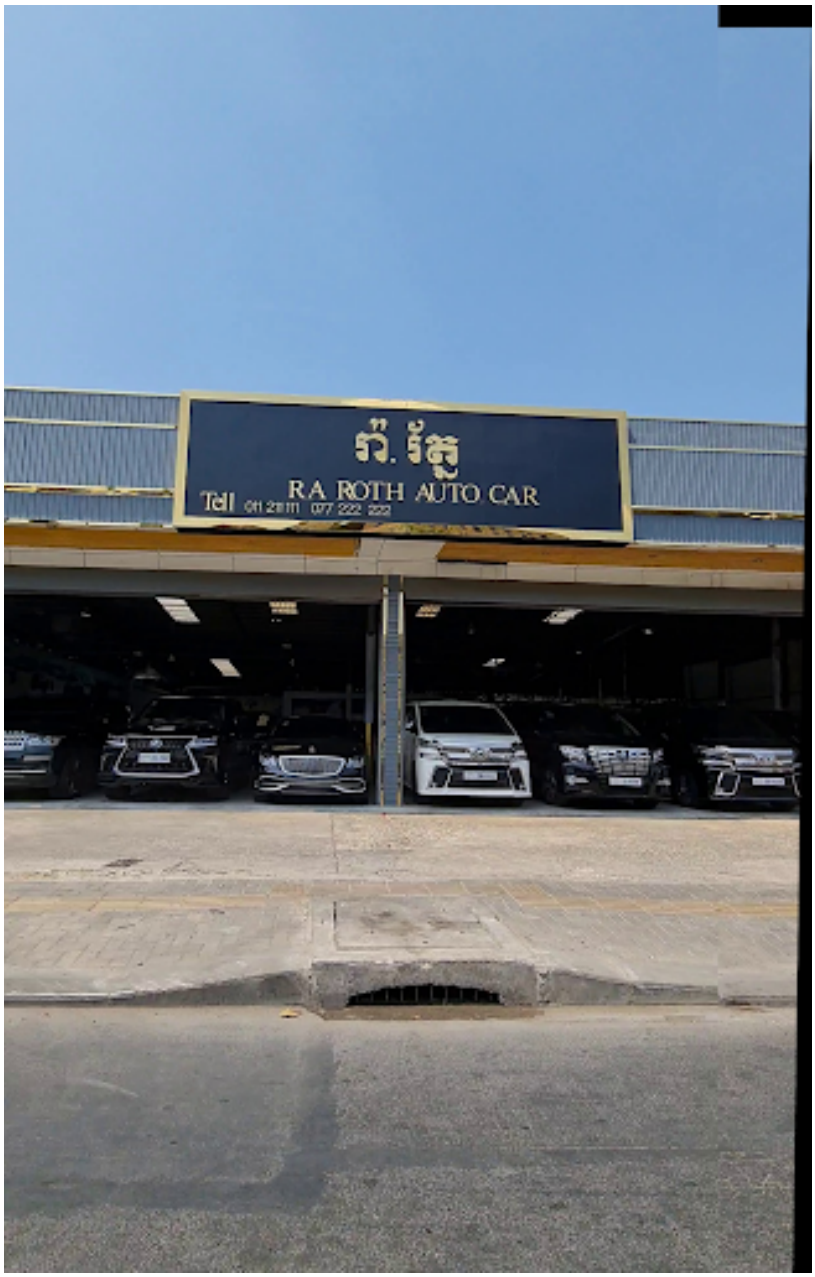

Figure 70: Score of 24.65

with the scores for figures 69,70, and 71. A qualitative assessment shows that this tool's scoring is unreliable to determining the quality of the image stitches, as can be seen when comparing Figure 70 with Figure 69.

Due to the inaccuracy of the stitch validation produced by BRISQUE, a SVM

was created in order to determine the presence of bad stitches, via the detection of seams. This SVM was trained on a manually labeled dataset consisting of stitched images produced by these feature detection algorithms. Using 2000 samples and a 80:20 train/test split, an accuracy of 88% was achieved.There were 3000 samples in total, but the full sample set was unable to be utilized due to hardware limitations. Figure 71 shows the results of several feature detectors, with the score being the percentage of images classified as "good", and scored by the SVM.

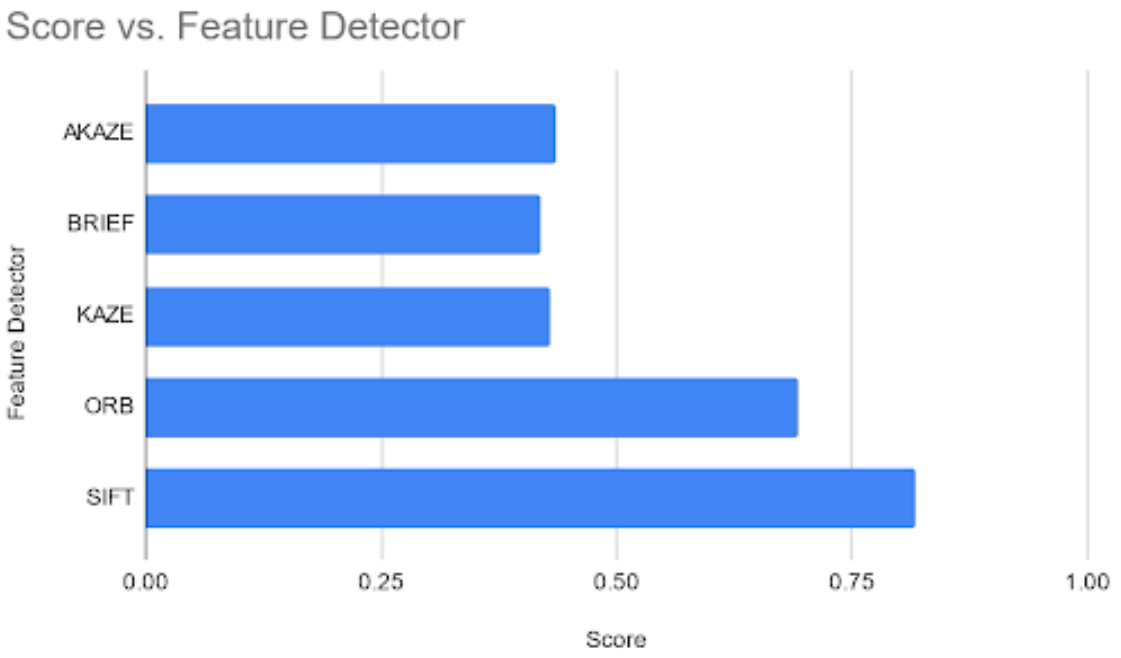

Figure 71: Accuracy of feature detectors scored by an SVM

From Figure 71, we can see that SIFT outperforms the other algorithms by a wide margin, but from table 1 we can see that it is also slow. ORB is significantly accurate, while being much faster, potentially making it more applicable for use in a real time application. If a fixed homography is decided upon for this project, SIFT is the best candidate for a feature detector. If features must be generated for each frame, then ORB provides much faster processing time than SIFT, while being comparatively

accurate.

When testing the SVM on a new dataset, it as found that the SVM scored every image as "good". This is likely due to the fact that the environment of the new dataset was significantly different than the dataset the SVM was trained on. Due to this issue, it was determine that a machine learning approach was not reliable enough for stitch validation, as creating a reliable SVM for this purpose would likely require a much larger and varied training dataset. Instead, the gradient comparison used in the proposed soluton proved to be much more reliable method of validating the stitches

5.4 Direct Comparison

BRISQUE and the SVM failed to provide a reliable method of validating the image stitch. To provide a way to validate image stitches, a direct comparison the images is able to provide validtion, without the need for training data.

The first new solution attempted was a direct comparison of the pixel values within the overlapping region. The theory behind this approach was that a good overlapping area would have more identical pixel values. To calculate this score, the sum of the absolute difference of each overlapping image matrix was taken, then normalized by dividing by the area of the overlapping region. Using this score, an threshold could be determined experimentally, with a higher score being an indicated of a greater difference between the images in the overlapping region. Ideally, a high score would serve as an indication of a homography mismatch. Unfortunately, while this technique proved to be able to detect poor quality stitches, it did so inconsistently, and failed to detect the manual camera shift from the rain dataset. Additionally, this technique was unable to deter minor errors produced by the stitch.

The lack of sensitivity of the direct comparison approach was suspected to be related to the additional information provided by the color values of each pixel. These

values cause a number of issues which likely impacted the accuracy of the validation technique. The first of these issues was the unnecessary information of the color values. When checking for similarity, color provides little additional value, and can actually introduce additional noise the the process. This is due to the fact the colors in the images are very impacted by the brightness of the image.

Figure 72: Images taken from adjacent cameras

As can be seen in Figure 72, lighting differences can be substantial, even for image takes simulaniously and in the same environment, This changes the color values on the image, and could lead to false positives when determining the quality of the image stitch. In order to circumvent this issue, grayscaling was introduced, as this should reduce the impact of differing brightness, as well as simplify the processing. This has the added benefit of improving the processing time of the validation operation.

By grayscalng the images, we can reduce the impact of differing brightness, while preserving the necessary information required to determine whether or not the overlapping area is being correctly generated. In Figure 73, the same images from Figure 72 has been grayscaled. It is clear that these images appear much more similar after being grayscaled, indicating that their pixel values are more similar. Using this, a more accurate comparison using the sum of absolute difference technique previously described can be calculated. Experimental testing revealed that while this technique

Figure 73: Grayscaled images

was more sensitive to camera changes than the full color approach, it was unable to detect smaller difference between the overlapping images, and as a result would have false negatives on bad homography matrices. After normalizing each score, the difference scores generated by Figure 72 was 0.131, while the score generated by the gray scale image of Figure 73 had a score of 0.123, with a higher score indicating that a greater difference. Based on this, it can be seen that the color image results in a greater difference, despite the fact that the images in figures 72 and 73 are from the same images. While grayscale images are able to perform a more accurate comparison than when using a full color image, image where lots of non distinct features exists, such as tree foliage, could result in false negatives. In order to mitigate this possibility, the extraction of only the most prominent features is required. Image thresholding can be used to accomplish this. Adaptive threshold techniques are able to identify local means within regions of the image, which results in an image where only the most prominent features appear features appear.

Adaptive threshold and Otsu's Threshold techniques were experimented with in order to determine their viability for stitch validation. A sample of their results can be seen in figures 74 and 75. From these samples, a number of problematic areas can be seen. For Adaptive threshold, the resulting image has many small black lines,

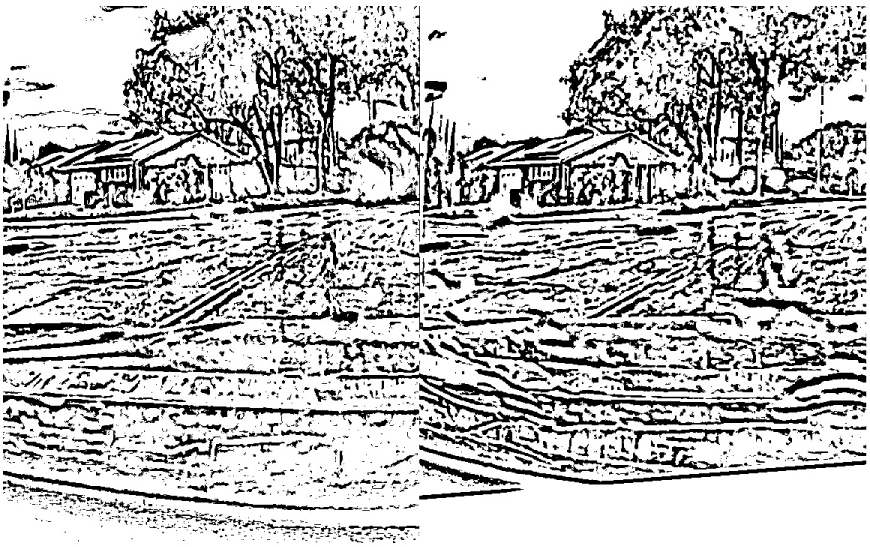

Figure 74: Adaptive threshold Image

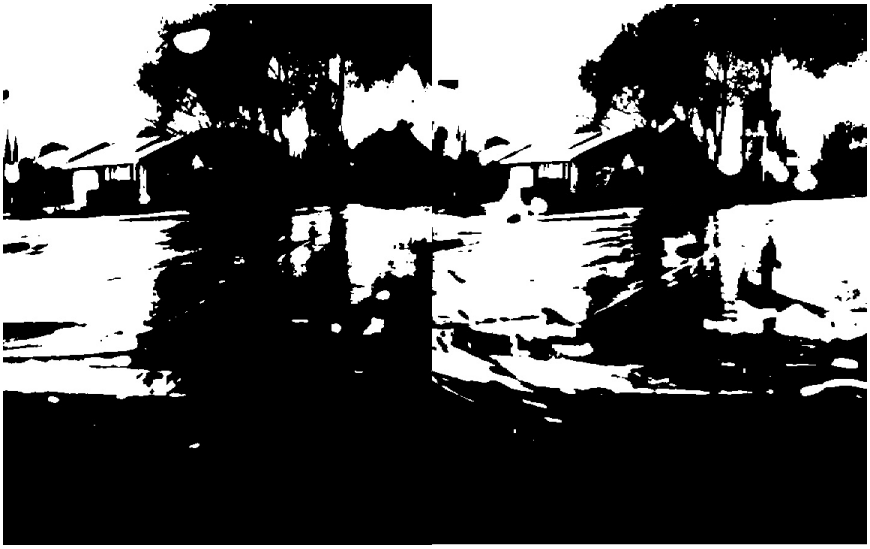

Figure 75: Otsu threshold Image

which vary widely between the two images shown in Figure 74. Additionally, the visibility of certain features, such as the trunks of the trees vary. Both of these issues cause a loss of accuracy when performing the validation operation.

For Otsu's Threshold, there is an overall improvement with the threshold areas, but there are a few regions which pol ute the final result. The most obvious problem

area comes from the tree canopy, in which one photo has a large area that passed the threshold, causing it to appear white. Additionally, the certain features in the center of each image vary on whether or not they surpass the threshold. Both of these issues, like with the adaptive threshold, cause the accuracy of validation using this technique to be less accurate. Unfortunately, due do these issues, neither of these techniques are good candidates for validating the image stitch. The gradient based approach used in the proposed solution was able to circumvent the issues experiences by these methods, as it is much less sensitive to lighting difference, and was able to better detect smaller mistakes in the resulting stitch.

5.5 GPU Experiments

The parallel pipeline is able to achieve image stitches in real time, but utilizes the CPU. This section explores the use of the GPU to improve the processing speed of the solution. The OpenCV and CUDA GPU libraries were leveraged for these experiments.

The first library researched was CUDA, a vastly populatar GPU pipeline for image processing on the GPU. In Figure 76, it is evident that GPU processing can vastly improve the processing time of the operations involved in the image stitching pipeline, with performance gains of approximately 2.5x-13x [11]. The process of using CUDA with the GPU involves uploading the Mat object into a GPU_mat, where additional procedures can then be performed on the image matrix using the GPU.

The OpenCV library has a "transparent" way to implement GPU processing via the use of openCL. OpenCL is a "hardware aware" library which is called automatically when it the library determines that processing can be performed on the a given piece of hardware. In the context of this project, this is accomplished by using the UMat data structure.

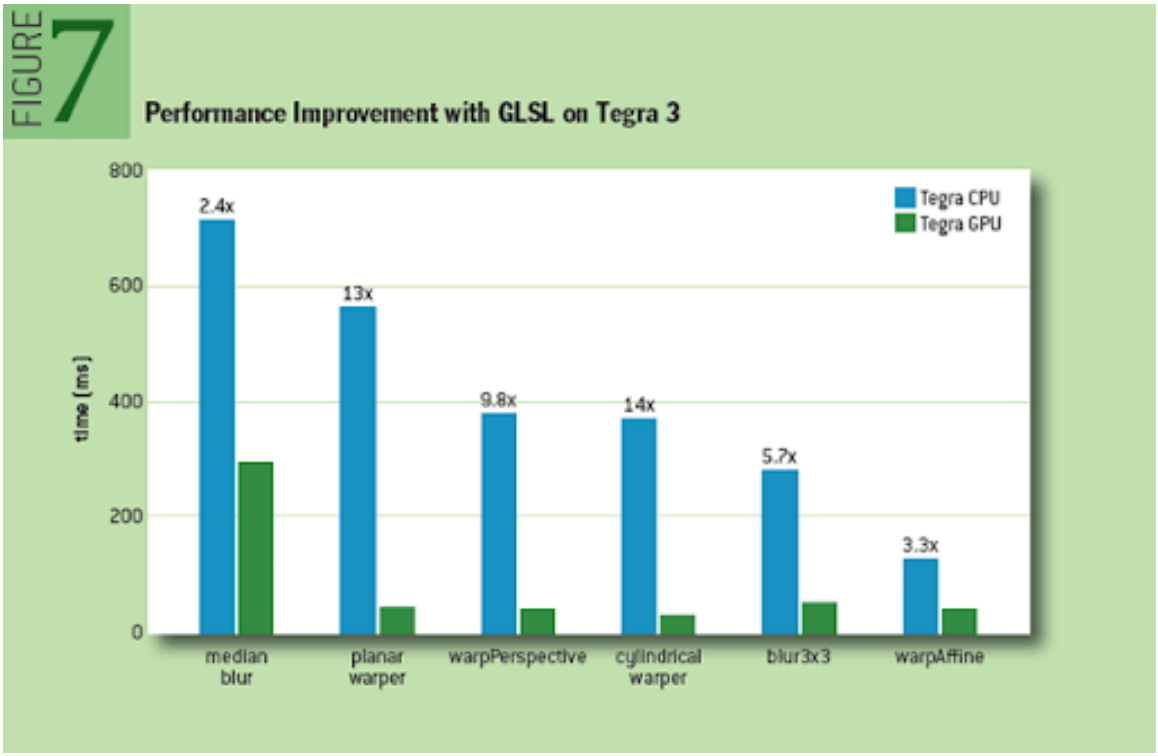

Figure 76: CUDA processing benchmarks [11]

Though both CUDA and OpenCL provide significant performance increases for image processing, there are some important trade offs to consider. The first of these trade offs is the loss of flexibility with the processing of the image matrix. Once an object in upload to a GPU_Mat, it must be processed using CUDA functions, and the same goes for UMAT objects, which must be processed using openCV functions. This limitation makes debugging more difficult, and makes it so that not all image processing problems can easily leverage the GPU. In this instance, since OpenCL leverages OpenCV functions, it has more functions available when it comes to working with the image matrix, and thus is a more attractive prospect.

In order to validate the viability of GPU processing for this project, benchmarks were gathered for the processing of the relevant functions. Figure 56 illustrates that GPU processing has the potential to significantly improve the image stitching

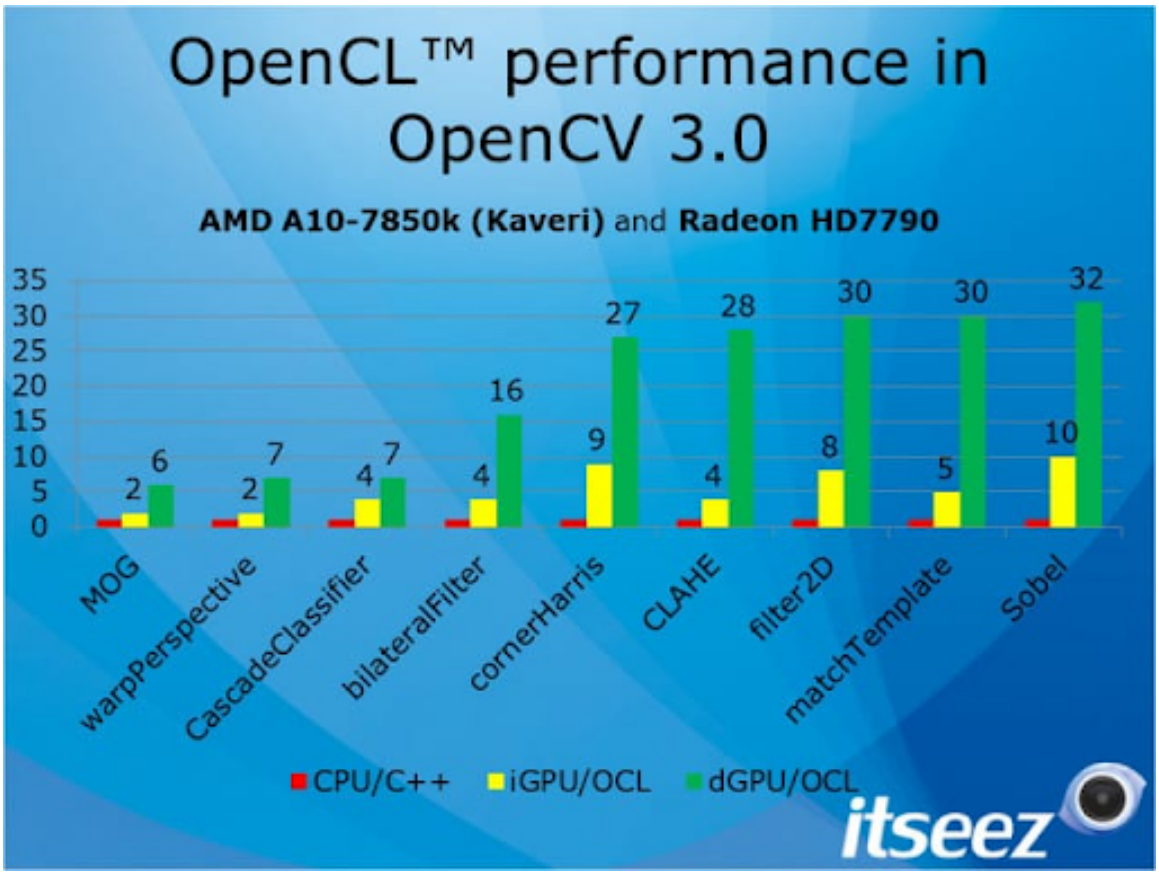

Figure 77: OpenCL processing benchmarks [12]

operations of the application, with CUDA outperforming the standard approach, and OpenCL performing the best, if overhead costs are ignored. Unfortunately, that overhead cost is very significant. With the overhead cost, both CUDA and OpenCL no longer outperform the standard CPU approach. These overhead cost come from uploading the image to a GPU_mat and converting them to UMat.

In order to reduce the overhead cost associated with sending an image to the GPU, the number of uploads and downloads to and from the GPU must be minimized. As a result, the original CPU implementation was no longer viable and had to be rewritten. Additionally, the limitation of only being able to use OpenCV or CUDA functions for GPU objects increased the complexity of solving this problem. Since

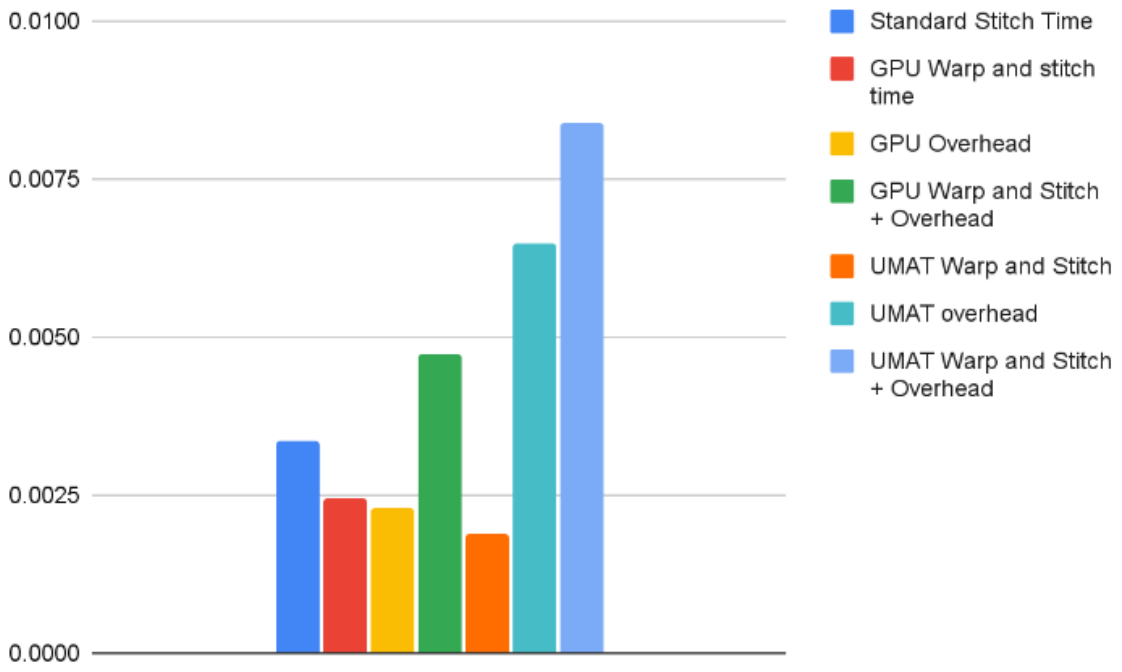

Figure 78: GPU benchmarks

openCL performed that best, ignoring overhead cost, and can be triggered for most OpenCV functions, it was selected as the focus for this conversion. The approach to this is as follows: Generate a homography matrix, perform the warpPerspective operation on one of the images, and finally stitch and blend the the images. The first 2 operations are trivial, since they are generated from OpenCV functions. The complication arises when we can try to stitch and blend the images, since the original implementation directly modifies the image matrix. UMat, the data type needed to work on the GPU, is not directly accessible, so an alternative approach which uses OpenCV functions was required. The following relevant matrix operations are available in OpenCV: bitwise_or, bitwise_and, bitwise_not,bitwise_xor, add, and addWeighted. The add function works by adding every element in one matrix to its corresponding element in another. Doing so can join 2 images together, but results in a highly brightened region, the overlapping area, since the pixel values at those

Figure 79: Add operation on two images

Figure 80: addWeighted operation on two images

points getting added together. addWeighted alternatively preforms the add operation, but takes in weight values to cause the operation to have a bias towards one of the input matrices. This is potentially useful as it also performs a blending operation simultaneously with the stitching. Unfortunately areas of black space cause the pixel values in those regions to become darkened, as can be seen in the left side of Figure 80.

In order to get around the darkened region issue, the overlapping area needs to be

Figure 81: Result of multiplying by the mask

Figure 82: Result of multiplying by the inverse mask

isolated so that only that region needs to have the addWeighted operation performed on it. To accomplish that, the following procedure is followed: First a constant is added to each pixel value to eliminate the possibility of any pixels with a value of 0 to exist in each image. Then the matrices are padded with black space to that

Figure 83: addWeighted operation favoring the left image

Figure 84: addWeighted operation favoring the right image

they have the same size. Next the matrices are multiplied together, resulting in an image where only the overlapping area is exposed. Using this output matrix, a binary mask can be generated by using OpenCV's findNonZero function, which returns the location of all non zero pixel values in a matrix. This mask allows for the isolation the overlapping area, and its inverse allows for the isolation of the non overlapping areas.

Figure 85: addWeighted operation with alpha channel

Once the overlapping area in each image is isolated by using matrix multiplication on the images and the overlap mask, the addWeigthed operation can be performed to blend the images together, and the resulting blended image can be added to the isolated non-overlapping areas of the image. Unfortunately, visible seams appear in the resulting image, regardless of which image of favored, or if they are favored equally, as can be see in figures 61 and 62. In an attempt to remediate this issue, addWeighted with an alpha channel was attempted. The alpha channel determines the opacity of a pixel, and by applying 0 to the alpha channel of the black pixel, ideally the addWeighted function would produce an image which ignored the black space. As shown in Figure 223, this approach produced a result identical to the 3 channel addWeighted, most likely because addWeighted does not take the alpha channel into account. While addWeighted does perform the blending operation, figures 83, 84, and 85 show that blending is this way does not achieve good results. Additionally, from Figure 86, it can be seen that the GPU approaches, segmented addWeighted and addWeighted(alpha) take longer than the CPU approach to achieve the same

Figure 86: Run time comparison between GPU and CPU addWeighted approaches

result. This is due to the large number of matrix multiplications required in order to isolate the overlapping and non overlapping areas, and the additions to join them. Ultimately, due to the limitations on customized code and overhead cost associated with translating GPU objects to CPU objects, it was determined that using the GPU was not a viable option for this solution.

5.6 Evaluating Feature Detectors

Table 3: Quality evaluation of feature detectors

	SIFT	***ORB***	***KAZE***	***AKAZE***
Quality score	0.05557	0.068	0.055517	0.05795
Failure Rate	0.005	0.98	0.005	0.024

To make this application for use in real time use cases, a reliable feature detector that also produces a high quality features must be selected.To accomplish this, the

stitch validation procedure described in the proposed solution was leveraged to provide a quantitative score for the quality of the produced images. To generate these scores, 6800 image stitches were generated from the rain dataset. Each set of images from the stitching procedure are then scored using the stitch validation technique. The resulting scores are averaged to provide the quality score seen in Table 3. Failure rate was calculated taking the number of frames that failed to produce a valid homography matrix and dividing them by the total number of sets of images processed.

Based on the results seen in Table 3, SIFT is the best keypoint detector due to its good images quality score and low failure rate. This result is followed closely KAZE, which has a similar quality score and the same failure rate. Due to the close quality scores, additional analysis was needed to determine whether SIFT or KAZE was a better fit for this application.

To perform this analysis, videos from the rain dataset were generated from the image stitches provided by the matches from SIFT and KAZE. Using these videos, a qualitative assessment was performed.

Figure 87: KAZE Result

Figures 87 and 88 show the results of the stitching and blending operation using

Figure 88: SIFT Result

KAZE and SIFT as feature detectors. The resulting images are similar in quality, though SIFT provides a higher quality image, especially in the region on the dashboard of the car. Based on this information and the results in Table 1, which shows that SIFT is faster than KAZE, SIFT was chosen as the feature detector for the proposed solution.

CHAPTER 6

Conclusion

Table 4: Tasks processing average run times

Background processing	***Superpixel Mask Generation***	***Blend and Stitch***
1.3s	7.6s	0.0034s

Image stitching allows for the creating of a wider field of view, and can utilize multiple cameras to provide more information on the surrounding information than a single camera image could. In order to make this technology usable for real time application, the processing for each frame must be completed fast enough so that there is little to no perceptible delay. Due to the nature of the problem this paper is trying address, it can be assumed that the position of the cameras generally do not change, and therefore certain features such as the homography and blending matrices would only need to be run as needed. The results shown in Table 4 were generated by taking the average runtime of 6800 frames and they show that blend and stitch operation needs to be performed on each frame, with a processing time of 0.0034s, well within real time. Removing the Background processing and Superpixel Mask Generation from the processing of each frame was essential to make this application work in real time, as the processing time of each of them would be much to long otherwise. Due to the an assumption is made that the cameras do not shift, validation must be performed to detect if this assumption is no longer valid. If this occurs, then the background processing and Superpixel Generation steps are rerun in order to make the assumption valid again. These tasks are run in parallel with the Blend and Stitch operation, so that the video output is never interrupted. The need for specialized hardware required by the proposed solution of Suk et al. has been completely removed

in this proposed solution. Instead, all functionality is able to be performed in real time using purely software solutions. This greatly expands the numbers of systems that this architure can be used in. Ultimately each of these tasks working in tandem allow for a system that can perform high quality image stitches in real time, while remaining robust towards camera misalignment and without the need for specialized hardware.

LIST OF REFERENCES

[1] P. Vinukonda, "A study of the scale-invariant feature transform on a parallel pipeline," LSU Master's Thesis, 2011

[2] S. Mallick, "Histogram of oriented gradients explained using opencv," LearnOpenCV, 30-Nov-2021. [Online]. Available:https://learnopencv.com/histogram-of-oriented-gradients/. [Accessed: 06-Apr-2022].

[3] "Feature detection and description," OpenCV. [Online]. Available: https://docs.opencv.org/3.4/db/d27/tutorial_py_table_of_contents_feature2d.html. [Accessed: 17-Feb-2022].

[4] J.-H. Suk, C.-G. Lyuh, S. Yoon, and T. M. Roh, "Fixed homography-based real-time SW/HW image stitching engine for Motor Vehicles," ETRI Journal, vol. 37, no. 6, pp. 1143–1153, 2015.

[5] "How Fast Is Realtime? Human Perception and Technology," PubNub. https://www.pubnub.com/blog/how-fast-is-realtime-human-perception-and-technology

[6] D. Lowe, "Distinctive image features from scale-invariant keypoints" International Journal of Computer Vision, 2004

[7] G. Hollows and N. James, "Understanding focal length and field of view: Edmund Optics," Edmund Optics Worldwide. [Online]. Available:https://www.edmundoptics.com/knowledge-center/application-notes/imaging/understanding-focal-length-and-field-of-view/. [Accessed: 08-Aug-2022].

[8] A. Vazquez, B. Wang, G. Yang, and J. Saniie, "A single-camera 3D microscope scanner with image stitching and stereo matching," 2019 IEEE International Conference on Electro Information Technology (EIT), 2019.

[9] A. Mittal, A. K. Moorthy, and A. C. Bovik, "Blind/referenceless image spatial quality evaluator," 2011 Conference Record of the Forty Fifth Asilomar Conference on Signals, Systems and Computers (ASILOMAR), 2011.

[10] Qu, Z., Wang, T., An, S. and Liu, L. (2018), Image seamless stitching and straightening based on the image block. IET Image Processing, 12: 1361-1369. https://doi.org/10.1049/iet-ipr.2017.1064

[11] "Realtime Computer Vision with opencv," Realtime Computer Vision with OpenCV - ACM Queue. [Online]. Available: https://queue.acm.org/detail.cfm?id=2206309. [Accessed: 31-Aug-2022].

[12] "OpenCL," OpenCV, 18-Apr-2019. [Online]. Available: https://opencv.org/opencl/. [Accessed: 31-Aug-2022].

[13] X. Miao, T. Qu, X. Chen, and C. He, "Superpixel-based foreground-preserving image stitching," Machine Vision and Applications, vol. 34, no. 1, Dec. 2022, doi: 10.1007/s00138-022-01363-1.