Real-time Video Mosaicing for UAV Applications

Summary Report

Brendan P.W Adam[1]

University of New South Wales, Canberra

This summary report describes the methods involved in creating a panoramic image, called a mosaic, which can be adapted and designed for UAV applications. There is an increasing demand to expand the field of view while operating UAVs, both in the civilian sector, such as search and rescue, and for military purpose such as threat detection and object recognition. Image mosaics displayed at real time can overcome this inherent problem by expanding the field of view, and if used in conjunction with object detection software will increase the chance of detecting an object or person. Although this project did not complete the main objective of creating a real time video for UAV applications, this project outlines the theory, processes, and presents a solution, which could be further developed to achieve this. The processes involved in creating a two and ten image mosaic, simulated UAV trajectory and real time application are presented. The algorithms used to create a two and ten image mosaic perform quite well when compared to popular online software, with a time of 237ms and 4.2s respectively. This summary report will explore how the created algorithm performs on an embedded system called a Pandaboard. The designed real time application achieves a frame rate of 5-6 Frames per Second with a 400x400 output window size.

i. Acknowledgements

I would like to thank my supervisor, Dr Matt Garratt, for his advice and guidance for the duration of my project. A big thanks to my father in law Roy, who proof read many draft reports during my time at university. Thank you for your time and effort. I would like to thank my family, who supported and encouraged me through my time of study. To Quin, all those times I told you I couldn't play with you because I was too busy doing university work, thank you for trying to understand, and I promise I will play with you every day. Maggie, I am looking forward to getting to know you more and more each day. Most of all I would like to thank my wife Laura, without your support and love I would not have made it and you would not be reading this. I love you Laura. "I love you and that's what you, are getting yourself into…".

Contents

Nomenclature 2

I. Introduction 3

A. Project Aim 3

B. Approach 4

II. Related Work 4

A. Real time display using mosaics 4

III. Process flow and algorithm breakdown 5

A. Receive Frame 6

B. Detect keypoints and extract descriptors 6

[1]PLTOFF Brendan Adam, School of Engineering & Information Technology, ZEIT4500

C. Matching descriptors in related frames 7
D. Determine the best set of matches and estimate the transform matrix 7
E. Warp images to previous frame 7
F. Align frames and form the Panoramic mosaic 7
IV. Algorithm output implemented in OpenCV 7
A. Keypoints and descriptors 8
B. Matching output 8
C. RANSAC and estimation of the transform matrix 9
D. Warping frame 2 9
E. Two image mosaic 10
F. Ten Image mosaic 10
G. Reliability of algorithm 10
H. Simulated UAV trajectory 11
I. Real time application 11
V. Comparison against popular online software 12
A. Increase image size as a function of time 13
VI. Conclusion 14
VII. Future Work 14
References 15

Nomenclature

UAV- Unmanned Aerial Vehicle
SURF- Speeded Up Robust Features
OpenCV- OpenComputerVision
SIFT- Scale Invariant Feature Transform
GPU- Graphics Processing Unit
RANSAC- Random SAmple Consensus
PROSAC- Proactive SAmple Consensus
NM- Nearest Neighbour
OpenCV- Open Computer Vision
Hz- Hertz (cycle/s)
Frames Per Second- FPS

I. Introduction

Traditionally, mosaicing is the process of combining small equally sized rectangular patterns to produce a picture or panoramic photo (mosaic) and were historically made out of coloured glass. Image mosaicing works in a similar manner except that the process involved, in creating the mosaic, overlaps a sequence of images (or video frames) and aligns them spatially. There has been significant research in the field of image mosaicing with the main objective of producing high quality resolution images. As a result the algorithms do not have a real time dependency and therefore the complete process to create a mosaic may take a significant amount of time.

Using Corner detection, Sum of Squared Difference for matching, and tailor series for the transformation matrix Cho et al [1] showed that they could achieve a high quality panoramic image that takes 16s to stitch a 680x480 size image to create a two-image mosaic. Due to the time taken to stitch the images together, the process Cho et al used may not be suitable for real time. However, many of the ideas and concepts used to create a mosaic to date can be built upon and adapted to perform at real time. Real time systems respond to an input and display the output instantly. For example a weapons guided system. The real time application developed as part of this project is a 400x400 sized window that can achieve 5-6 Frames per Second (FPS).

Live video from a UAV has a limited field of view of the surrounding environment; as a result, using the UAV imagery for the purpose of object detection and recognition in a military or civilian context suffers from tunnel or narrow vision. This summary report presents a way to create panoramic mosaic that can be adapted to perform at real time for UAV applications. This could then be used to expand a UAV operator's field of view which will also increase the situational awareness when an object of interest is expanded. This software could also be used in conjunction with other applications such as object and automatic threat detection. The methods used in this project are based upon popular algorithms.

There are two popular methods used to align and warp video frames to create a real time mosaic: pixel alignment [2], where pixels are matched from video frames and then directly aligned, and feature extraction [3], where features and descriptors are identified and the most suitable matches found and used for the warping process. This project focused on the latter method: feature detection and extraction and a matching type algorithm. Figure 1 is the output of the designed algorithm, which illustrates a three-image mosaic. The test images were taken from Google maps.

Figure 1. Project output for a three-image mosaic

A. Project Aim

The aim of this project is to design software that will take a sequence of video frames, during post processing, or from a live video feed, and combine them into a panoramic mosaic for real time display. Other uses for which the software could be further developed or used in parallel, include object detection and recognition, and optic flow.

The software will also provide:

- a framework for a range of future ADFA projects
- image registration software available for future implementation into a UAV type platform supporting the activities of the UNSW UAV field or UAV club; and,
- software available for future development by other ADFA students.

B. Approach

Initially, coding was carried out in MATLAB and the algorithms were created based upon popular methods. MATLAB is a high-level language and different to many contemporary programming languages, such as C or C++, in the way solutions can be developed quicker. The inbuilt functions for Speeded up Robust Features (SURF) feature detection were used, however the code was manipulated to produce specific data used for determining the best set of matches. Firstly, a two image mosaic was created in MATLAB and coding then commenced in OpenComputerVision (OpenCV). OpenCV is a library of programming functions aimed at real time computer vision applications. Its primary interface is C++ and retains a less comprehensive C interface [4]. The main advantage of OpenCV is that it performs a lot faster than MATLAB since it is C++ based. However, programming in MATLAB was useful in terms of comparing results and data which assisted in coding the mosaic algorithm in OpenCV. Using OpenCV, again like MATLAB, the inbuilt functions for feature detection were used and the same algorithms were coded in OpenCV. A two and ten image panoramic mosaic, trajectory simulation and a real time application were developed. The reason for using the inbuilt functions for SURF was based upon time and complexity and scope of the project. If SURF or another feature detection algorithm were to be developed from the ground up, it would have been the focus and the goal of this project and not to create a panoramic mosaic. Due to the complexity of the SURF algorithm, and it being one of the core processes to create a panoramic mosaic, this was not developed. However, it required to be understood before proceeding into further processws that utilised SURF.

The mosaic algorithm was then ported across to an embedded system called a Pandaboard, where the mosaic algorithm efficiency and suitability were tested. The Pandaboard is an open development single board computer with an ARM Cortex-A9 MPCore 1.0 GHz processor, 1 GB DDR2 SDRAM and has a persistence storage via an SD card.

II. Related Work

A. Real time display using mosaics

There has been significant work conducted in the use of mosaics for real time display. Morse et al [5] utlised the Harris corner detection algorithm to identify features within each successive image, and Random Sample Consensus (RANSAC) for matching the features. This was achieved by estimating the Euclidean transformation between each pair of feature point matches. Morse et al used a deinterlacing type method for preprocessing each incoming frame to correct for radial distortion. This was done to minimize the number of artifacts produced for a high velocity UAV. Morse et al created a ‘temporal local mosaic’ using video frames captured from their UAV by firstly finding the Euclidean transformation from a current frame to the next frame, which is used as the basis to compose a cumulative transform to warp the frames together to form a panoramic mosaic. Morse et al do not comment on the performance of the image mosaic, but employ a user study using 16 people to quantify their display methods.

One particular method proposed by J de Villers[2] for real time mosaic display uses a Commercial off the Shelf (COTS) Nvidia Graphics Processing Unit (GPU), clocked at 280MHz, that implements a parallelised algorithm to stitch images together . A GPU was used because of the high costs associated with the processing equipment to carry out equivalent tasks. Although it would be impractical to mount this hardware to a MUAV, due to weight to lift restrictions, this type of setup could be used on a larger type UAV platform. Villers performs the stitching using photogrammetric parameters in the static staring array using 6 Degrees of Freedom (DOF) position of each camera (2 used) relative to the stitching reference.

It has been designed so that each pixel of the output stitched image is independent of all the others and so can be implemented as a fragment shader on a GPU and attached to a texture. This allows the GPU's horde of stream processors to evaluate the algorithm in parallel. deVillers [2] showed that the particular hardware could perform image stitching at 360 frames per second, and compared to other registration based stitching methods, such as GPU/CPU hybrid implementation, was seven times faster to stitch 6 images together for a256 by 256 region of interest. However, due to the cost and complexity of this type of implementation, this was not considered a practical approach to be employed for my project.

A new scheme developed by Botterill et al [6] called RT-AIM (real time aerial image mosaicing) uses the FAST corner detection algorithm; the descriptors are based on image patches that are centered on the detected corner features. Botterill et al use bag-of-words (BoW) algorithm model by representing each image as a set of descriptors that uses a hierarchical dictionary allowing fast indexing of images. BaySAC was used as an alternative algorithm to RANSAC, and it was found that this method, compared to RANSAC, used less iterations and was faster in finding the best suited inliers. Botterill et al further refined the inliers using top down outlier removal [6], thus allowing more inliers to be found while removing potential outliers. This result is the perspective transformation utilized to warp images together. Botterill et al results are as follows: Using 3220 images with size 800x532 captured at 2.8Hz, produced a video mosaic with size 1200x100 maintaining a frame rate of 6.2Hz, with each mosaic containing an average of 8 images. For images with size 320x212 Botterill achieved a frame rate at 20Hz. For a full literature review see the initial report.

Brzeszcz etal [7] directly used SURF for key point detection and descriptor assignment, then descriptors are matched by finding the Euclidean distance based on the NN algorithm, which is then compared to a threshold value. Once again, RANSAC was used to find the best possible inlier matches and remove the outliers from the set of data, creating the projective transform that maps one image to the other. The Brzeszcz et al method aligns and warps images to create a real time mosaic in two parts: the pairwise bundle adjustment and global bundle adjustment. Pairwise bundle adjustment occurs every input video frame and takes only two frames, adjusting the second to align it optimally to the first one. It starts from the last globally adjusted video frame, and ends on the last inputframe (the frame that has been most recently captured) [7]. Brzeszcz et al used this method due to the camera being non-linear represented, hence image alignment has also become non-linear. Global adjustment is performed periodically, adjusting all the images simultaneously taking into account the overall structure of the mosaic, and thus it corrects accumulated errors [7]. Brzeszczet al using their particular methods achieved a frame rate of 13 FPS in terms of processing the input video stream and 20 FPS in terms of visualisation.

III. Process flow and algorithm breakdown

Contemporary research in the use of image mosaicing uses four main methods [6] shown by two through four below. Processes one and six have been added for completeness and to explain the complete process as a whole.

1. Receive frame
2. Feature point detection and descriptor extraction used for registering images
3. Match descriptors in related frames
4. Determine the best set of matches and estimate the transform matrix
5. Warp image to previous frame
6. Align frames and form the Panoramic Mosaic

Figure 2 describes the process flow in creating a panoramic mosaic. This section of the summary will describe the operation of each algorithm to form a panoramic mosaic.

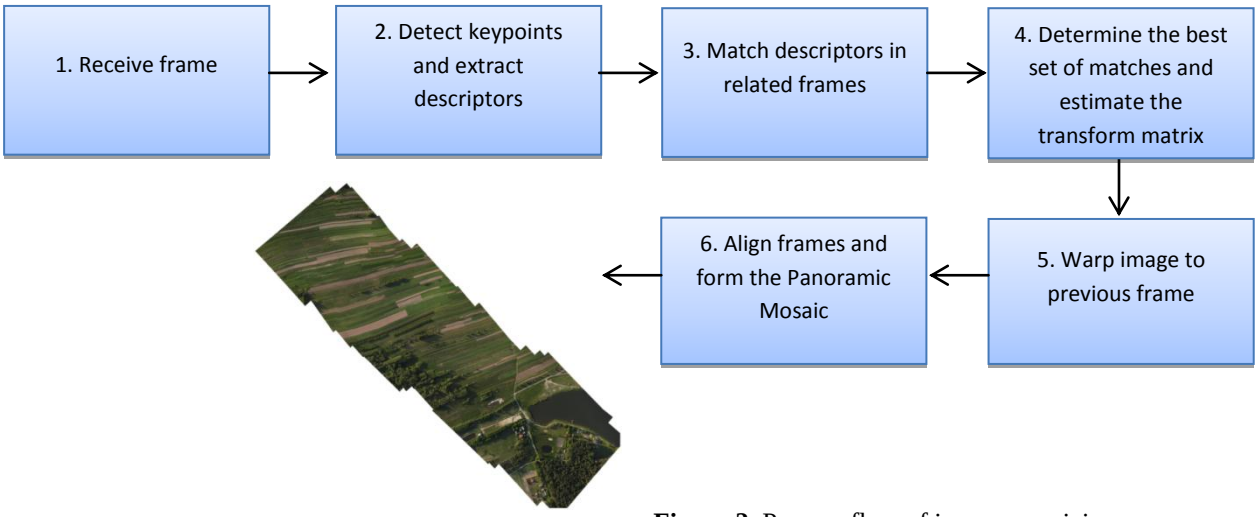

Figure 2. Process flow of image mosaicing

A. Receive Frame

Images are read in and converted to grey scale. This is done because the feature detection is based upon being able to locate feature points due to intensity changes within the image.

B. Detect keypoints and extract descriptors

There are many detection algorithms used for feature detection and descriptor extraction. The most widely used are SIFT (Scale Invariant Feature Transform) and SURF (Speeded Up Robust Features) [8]. SIFT was developed by David Lowe and published in 1999 [9]. However, there was a need for a more robust and speeded up version, and as a result Herbert Bay developed SURF in 2006 [10], which is the speeded up version of SIFT (hence the name). SURF is invariant to scale and rotation, this means that if an image has been rotated as a result of camera movement or rotation of a vehicle, SURF can still distinguish edges and features. Scale invariant means that if there is a change in perspective, for example the camera capturing a frame has moved closer or further away from an object, that object will still have the same descriptor and can then be successfully matched

SURF detects key points through the utilisation of a Hessian matrix, due to it s high computational efficiency (time and accuracy) and computes the determinant based on the Laplacian of Gaussian (LoG) with a box filter. The Laplacian measures the 2nd spatial derivative of an image (how much the image intensity values change per change in image position) and highlights regions of changing intensity, which makes it extremely useful in detecting edges in an image. Bay et al [10] increases the efficiency of this process by using a box filter which approximates the second order Laplacian derivative, by filtering at a position x on an image repeatedly, until a reasonable approximation is found. The box filter is used in conjunction with an integral image, which decreases the processing time, and as a result this makes it quicker than other contemporary feature detection algorithms. An integral image calculates the cumulative sum in the x(rows) and y(columns) direction of an image. The value of a pixel at a given location (x,y) is the sum of all values inside a rectangle defined by the origin and the pixel location (x,y).

Descriptors provide unique and robust description about each keypoint, which can then be later used for matching. The orientation for each keypoint is determined by convolving pixels in its neighborhood with the horizontal vertical Haar wavelet filters. The filters act as a sampling window over a keypoint, the window is broken into subsections and then the horizontal and vertical derivatives of the image's intensity are calculated. This allows the intensity gradient at each keypoint location to be characterized and enables the descriptors to be rotationally invariant. The absolute value of the derivatives of the subsections of the window are taken and then added together. This when represented as a vector and normalised, produces a descriptor with 64 dimensions. These dimensions represent the distinctness of a descriptor, which provide information of the descriptor for matching. Again, as for keypoint detection, this process is made faster through the use of integral images.

C. Matching descriptors in related frames

Matching descriptors in two frames is achieved through nearest neighbour based algorithm. Similarities between descriptors are found by calculating the Euclidean distance between each descriptor (feature vector) and then comparing this to a threshold value. If the Euclidean distance is less than the threshold value then the two feature vectors are considered a match. However, some mismatch may still exist (i.e. bad pair of matches).

D. Determine the best set of matches and estimate the transform matrix

RANdomSAmple Consensus (RANSAC)is usedto remove outliers and find the best possible set of inliers. RANSAC was first proposed by Fischler and Robert C. Bolles in 1981[11] and is frequently used in image mosaics, and also is used for image mosaicing for real time display [7]. RANSAC is an iterative algorithm used to estimate parameters of a mathematical model to find a good set of inlier correspondences. The set of data may be subject to noise (or false matches), referred to as outliers. The more iterations completed the greater the probability that the set of inliers are the best match for the set of data. The output of RANSAC is a projective transform called a Homography matrix. The Homography matrix maps the points of frame 2 to the corresponding points in frame 1. Below illustrates the variables in a 3x3 Homography matrix.

$$H = \begin{matrix} R11 & R12 & T1 \\ R21 & R22 & T2 \\ P31 & P32 & 1 \end{matrix} \quad where\ R - rotational, T - translation\ and\ P - perspective$$

Consider the simple Homography matrix with the following values: $H = \begin{matrix} 1 & 0 & 50 \\ 0 & 1 & 10 \\ 0 & 0 & 1 \end{matrix}$

This matrix would correspond to an image which requires it be shifted 50 pixels in the positive x direction and 10 pixels in the positive y direction (e.g. frame 2 is shifted downwards) to be in the same coordinate system as its predecessor image.

E. Warp images to previous frame

To warp the perspective of frame 2 to frame 1, the Homography needs to be applied to every single pixel within the source frame. However, doing this process to every single pixel is time consuming and the most efficient way, when manipulating a pixel grid, is to use a form of interpolation. One of the common methods used is bi-linear interpolation. Bi linear interpolation is not a linear process but rather the product of two linear functions and uses the closest 2x2 neighborhood pixels (forming a rectangle or square), whose location and values are known, surrounding the unknown pixel [12]. The unknown pixel value is calculated using a weighted average of the four surrounding pixels, where the weight of a pixel is based upon the pixels distance from each of the known x,y location.

F. Align frames and form the Panoramic mosaic

Now that the frames are in the same coordinate system, frame 2 can be aligned to frame 1 to form the panoramic mosaic.

IV. Algorithm output implemented in OpenCV

The previous section described the theoretical process in creating a panoramic mosaic; this section will illustrate the results produced at each stage of the algorithm, focusing on the OpenCV implementation of the mosaic algorithm. The two test images used to create a two-image panoramic mosaic are shown in figure 3. The images have been taken from Google maps. Frame 2 has a rightwards linear shift and a 70% overlap.

Figure3. Test images used for a two-image mosaic (frame 1 – left, frame 2 – right)

A. Keypoints and descriptors

Once the frames are read in they are converted to gray scale and the features identified and the descriptors extracted. As mentioned in the approach section of this report, SURF was used in both MATLAB and OpenCV to achieve this. Fig. 4 shows the keypoints found in frame 1 and the descriptors extracted from frame 1 are depicted in figure 5.

Figure 4. Keypoints identified

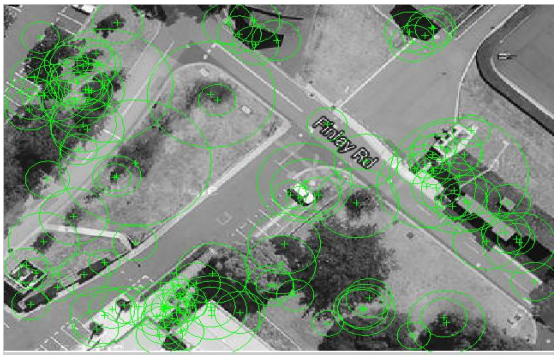

Figure 5. Descriptors extracted

B. Matching output

To determine a valid set of matches the Euclidean distance is calculated between each descriptor and compared to a distance threshold value by looping through the positional data produced by the extraction of the descriptors. This value must be large to enough to account for a large shift from frame 1 to frame 2, however not too small a value as some matches may be overlooked. The output is a vector with a list of valid matches and their x,y locations within frame 1 and frame 2. The x,y locations of the matches are useful in determining where the same feature occurred in frame 1, which will be used to calculate the translation part of the Homography matrix. Figure 6 shows the valid matches between frame 1 and frame 2. However, as mentioned in the algorithm process description, outliers exist within the set of data. For example, the black circles surrounding the red coloured descriptors represent a bad match. To remove invalid matches RANSAC is employed.

Figure 6. Matches found

C. RANSAC and estimation of the transform matrix

The RANSAC algorithm is an iterative loop that randomly selects from the set of valid matches with their corresponding x,y locations and set of data produced by extracting the descriptors . Each valid match corresponds to data within the descriptor data set used to determine the best set of inliers. Increasing the number of iterations produced a better approximation of the Homography matrix, however at the cost of processing time. The basic steps for the RANSAC algorithm:

1. While iterations is less than 10
2. Randomly select four pairs of matches with corresponding descriptor data
3. Calculate Homography with set of data
4. Store the set of inliers, if require more inliers (or set may contain outliers) go to step two.
5. Compute the Homography matrix with the best set of inlier correspondences:

$$H = \begin{matrix} R11 & R12 & T1 \\ R21 & R22 & T2 \\ P31 & P32 & 1 \end{matrix}$$ For frame 2 the Homgraphy matrix is: $$H = \begin{matrix} 1 & 0 & 101 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$$

The Homgraphy matrix produced for frame 2 shows that frame 2 requires to be shifted by 101 pixels in the positive x direction

D. Warping frame 2

As per the Homography matrix, frame 2 requires it be warped 101 pixels in the positive x direction, so that it will be in the same co ordinate system as frame 1. This is achieved by selecting four points; the Homography matrix is then applied to those four points and the pixels within those four points would be interpolated using bi linear interpolation. This process is carried out on each pixel, which is then copied into a new matrix with correct orientation and coordinates; this process would continue until the whole frame had been warped. Figure 7 illustrates the output after frame 2 has been warped.

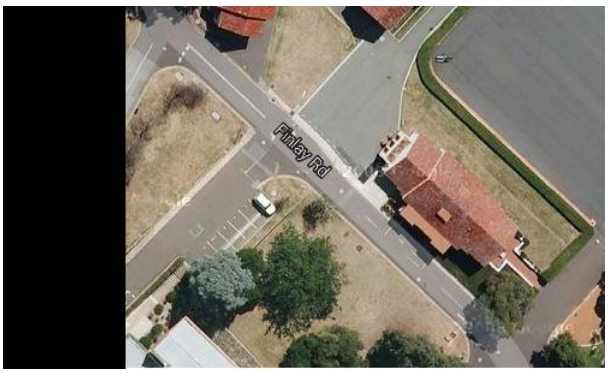

Figure 7. Frame 2 after warping

E. Two image mosaic

To align the frames it is a matter of copying frame 1 and placing it into the (0,0) x,y position in the resultant matrix produced after warping frame 2. Figure 8 shows a two image mosaic after the alignment process.

Figure 8. Two image mosaic

F. Ten Image mosaic

The process for creating a ten image mosaic is nearly identical to that for a two image mosaic, except that each frame is read into an array of matrices which is then looped through to form the ten image mosaic. The first iteration produces a two image mosaic which then becomes the new frame 1. The next iteration stiches frame 2 to the 2 image mosaic (which is now frame 1). This process is repeated until a ten image mosaic is created, as seen in figure 9. The test images were taken from Google maps.

Figure 9. Ten image mosaic where f represents the frame number

G. Reliability of algorithm

For the two and ten image mosaic each frame had an overlap of approximately 70%; many tests were executed to determine the point of failure. This occurred when images did not have an overlap of more than 35% (although this could slightly vary depending on the images used and the features within them). There needs to be at least four matches between corresponding frames; less than this and the algorithm will fail. There was no code written to handle these failures and this would need to be implemented to increase the reliability, and also if the algorithm is to be used for real time applications (UAV or not).

H. Simulated UAV trajectory

Five images were captured based on an exponential function as illustrated in figure 10 below. The purpose of this algorithm is to simulate a UAV trajectory over a piece of land. Figure 11 shows the output of the five test images stitched together. The process is similar to a ten-image mosaic, except that now the frames are not completely linear and each frame is captured based on an exponential function.

Figure 10. UAV captured frames from 1 to 5

Figure 11. Output of UAV trajectory

I. Real time application

A real time application was coded in OpenCV utilising two webcams which captured two frames: frame 1 and frame 2. The code consisted of a main loop which captured the 2 frames, which were then used as the arguments for the panoramic mosaic algorithm for stitching; the two image mosaic was returned as the result and displayed and with the output rendering at real time. Timing code was also placed inside the loop to calculate the amount of time taken to process a frame, stitch the frames and then display the frames. The purpose of this code was to determine the suitability and reliability for a real time application, which could then be further developed for use on a UAV ground station or on board UAV, utilising an embedded system for processing imagery. Figure 12 is a screenshot of the algorithm executing, when the subject is stationary and figure 13 is when the subject is walking past the camera at approximately 2.2 m/s (normal walking pace). The output window is 400x400 in size, with an overlap of 60 %. For this particular output window the main loop could process on average 5-6 Frames per Second (FPS), with a two second delay from when an object appeared on the output window and then left the field of view of the camera. As the window size increased, the FPS dropped as a result. When it was increased to 1080x720 the output was incredibly jittery with readings of less than 0.5 FPS on occasion and a delay of approximately 5 s.

Figure 12 Demonstrates very good alignment between frame 1 and frame 2 (the desks are aligned almost perfectly) and it is difficult to determine where the seam is, but can be seen if you follow an imaginary line from the desk to the subject's lower shoulder. Contrary to this in figure 13, the seam is more prominent due to the subject moving past the camera and the algorithm not being able to process the frames quick enough. At times during processing, frame 2 became misaligned to frame 1 but then re aligned. This is due to the nature of the RANSAC algorithm as it is selecting randomly from the given set of descriptor data and estimating a different Homography (with some Homography estimations better than others) each iteration of the for loop in the main part of the code. It was also noted that when objects were too close to the cameras, or a hand placed in front of the camera, the mosaic algorithm could not successfully match features in either frame, and the algorithm failed, which is completely understandable. As mentioned previously the mosaic algorithm requires at least 4 valid matches to complete the entire mosaic process successfully. Real time applications demand a high frame therefore the suitability of this algorithm is low. However, it could be further developed or integrated with a system with high end specifications.

Figure 12. Subject stationary (frame 1 left frame 2 right)

Figure 13. Subject walking past frame

V. Comparison against popular online software

Some popular online software for image mosaicing was downloaded for performance comparison. The software name, with their corresponding average (taken over five runs) running time is shown in table 1. One thing to note is, the online software was most likely designed for quality over computational speed and hence the reason why the time it takes to stitch multiple images is lengthy for most. Figures 14 and 15 also show timings for MATLAB and the algorithms executed on the Pandaboard. Enlarged versions of the images can be seen in the appendix. The online software and the mosaic algorithm were executed on an Intel i3 core 2 duo, clocked at 2.93 GHz with 4GB of DDR3 RAM. The mosaic algorithm, as shown in figures 14 and 15 performs quite well against the other software, with a time of 227ms for a two-image stitch and 4.1 s for 10 images. The other software performed reasonably well for two images, however the time taken to stitch 10 images together was quite long, with the Photo Stitcher software taking 163 s to stitch 10 images together.

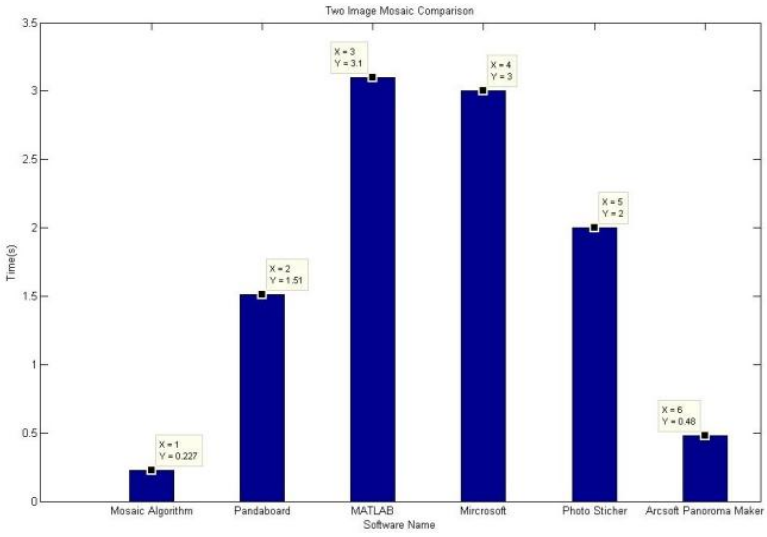

Figure 14. Two image mosaic timings

The Pandaboard, on average, could only stitch 2 images in only1.51 s and 10 images at 12.4s.The difference in time compared to the desktop PC and the Pandaboard is directly related to the processing power, the speed and size of the RAM. (1 GB DDR2 SDRAM compared to 4GB DDR3 RAM).Overall, the mosaic algorithm performed well against the various online software used to create a panoramic mosaic. There were also no visible quality differences detected between the mosaic algorithm and the online software packages.

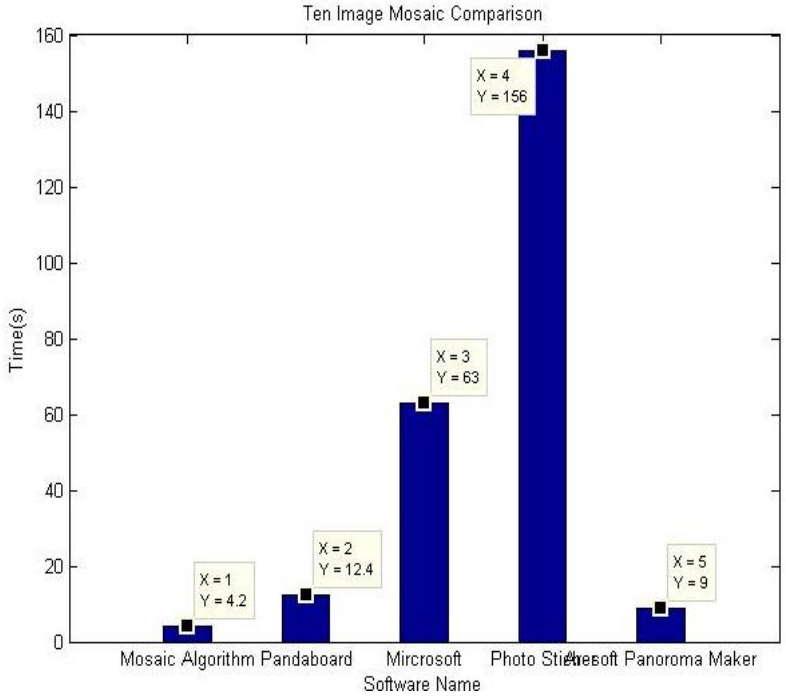

Figure 15. Ten image mosaic timings

Two Image Mosaic Elapsed Time (s)	**Ten Image Mosaic Elapsed Time (s)**
Mosaic Algorithm 0.227	4.2
Pandaboard 1.51	12.4
MATLAB 3.1	N/A
Microsoft Image Composite Editor 3	63
Photo Stitcher 2	156
Arcsoft Panorama Maker 0.480	9

Table1. Software elapsed time

A. Increase image size as a function of time

As the image size increases the time to stitch images increases. Figure 16 depicts this relationship. Three different sized test images were used: 400x500, 500x400 and 600x 500, and tested on a desktop and Pandaboard. Five test runs were conducted and the average calculated. The increase in time is directly related to the time taken to locate keypoints and extract descriptors, which increases as a function of image size. This is due to there being more pixels in the image, in turn increasing the overall time to stitch two images together. The same relationship is seen when the algorithm is executed on the Pandaboard. More test images could have been used, however this would be unnecessary as it would show a near linear relationship. For a breakdown on the timings of each individual process, refer to the table 1 in the appendix.

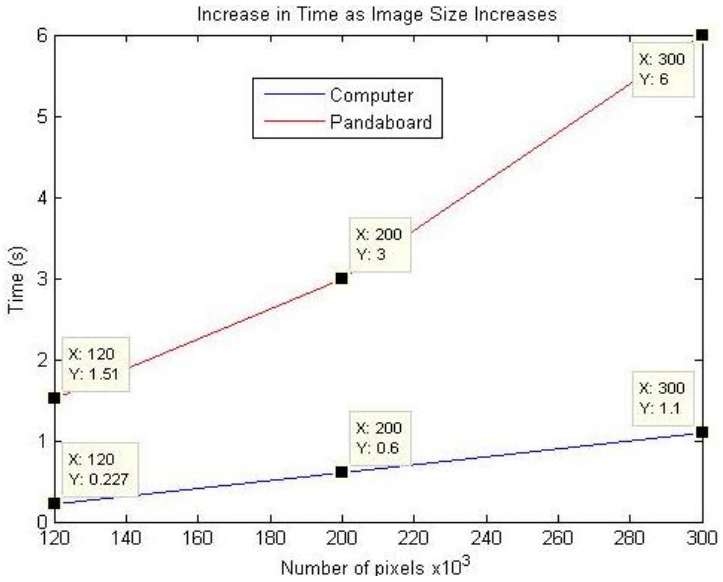

Figure 16. Increase image size vs. time

VI. Conclusion

This project has presented a way to create a panoramic image mosaic based on popular algorithms, which could be adapted and used for real time UAV applications, with the main focus to increase the field of view of the operator. The software further developed would be ideal for hobby enthusiasts or UAV systems that have a smaller budget. The project software could be evolved and used in conjunction with other software such as object detection and recognition and optic flow. This project summary report has shown the development of a mosaic algorithm, which initially was based in MATLAB where a two image mosaic was created. The algorithm was then developed in OpenCV, and a two and ten image mosaic algorithm, simulated UAV trajectory and real time application were created. The two and ten image mosaic was compared against popular online software and performed exceptionally well against the competition, taking on average 237ms and 4.2 s to create a two and ten image mosaic respectively. However, the online software was probably developed with low time constraint as its purpose is for quality over processing time. The output produced by mosaic algorithm compared to the online software had no discernable difference in terms of quality. The mosaic algorithm fails when there is not an overlap of greater than 35 %; if less than four matches are detected then the algorithm fails. This would need to be accounted for to increase the reliability of the algorithm. The algorithm was also tested on the Pandaboard, but due to the lower specifications did not perform well compared to when it was run on a lower end desktop PC. This demonstrated a low suitability of the algorithm when operated on the Pandaboard. A real time application was also developed using two webcams. For a 400x400 size window the algorithm achieved 5-6 FPS, increasing the window size, as expected, decreased the frame rate.

VII. Future Work

There are a few aspects in which the mosaic algorithm could be further developed to increase the efficiency. These are outlined below.

- Using multi threads for parallel processing. This will decrease the overall processing time for the algorithm, which would result in an increase in FPS. This would require further coding in OpenCV and a greater understanding of CPU and memory programming.
- Using a more capable embedded system. As more research and development goes into the use of embedded systems, there will be an increase in processing power and memory. The Odriod U3 is a more capable embedded system than the Pandaboard, and was the preferred choice for testing, however there were hardware and software issues with the system and no testing was conducted. If this board, or a more powerful board, could be used then this would increase the performance of the mosaic algorithm. The algorithm could also be tested directly on a high end GPU which could be connected to an embedded system.
- Capture a frame on the Pandaboard and send it via remote link (or Wi fi) to a ground station (laptop or desktop PC) and process the frames during post processing or real time. This could be used as the initial phase for sending and receiving video frames via UAV.
- Blending algorithm for a seamless stitch. Video frames stitched together from a UAV video capturing system would need blending to remove the seams (even Google maps cannot get this completely right). This would be done to achieve a smoother output, which would be ideal for a UAV operator.
- Try and catch statements. When the mosaic algorithm fails to find features or matches it fails and the program shuts down. Try and catch statements could be implemented to catch these errors and either capture a new frame, or start the capturing process from the beginning of the algorithm.

References

[1]S Cho, Y Chung,& J Lee,'Automatic Image Mosaic System Using Image FeatureDetection and Taylor Series',*Proceedings of the VIIth Digital Image Computing: Techniques and Applications Conference, 10-12 Dec. 2003, Sydney*, pp. 549-556.

[2] J de Villers, *Real-time photogrammetric stitching of high-resolution video on COTS hardware*, Council for Scientific and Industrial Research, Pretoria, 2009.

[3] K Mikolajczyk, T Tuytelaars, C Schmid, A Zisserman, J Matas, F Schaffalitzky, T Kadir, & L Van Gool,'A comparison of affine region detectors',*International Journal of Computer Vision*, Vol. 65, 1-2, November 2005, pp. 43–72.

[4] Code.opencv.org, (2014). *OpenCV - WikiStart - OpenCV DevZone*. [online] Available at: http://code.opencv.org/projects/opencv/wiki [Accessed 18 Oct. 2014].

[5] B S Morse, D Gerhardt, C Engh, M A Goodrich, N Rasmussen, D Thornton,& D Eggett, 'Application and Evaluation of Spatiotemporal Enhancement of Live Aerial Video using Temporally Local Mosaics', Brigham Young University, Provo, Utah, 2008.

[6]T Botterill, S Mills, &R Green, 'Real-time aerial image mosaicing', Geospatial Research Centre, University of Canterbury, Christchurch, NZ, 2010

[7] M Brzeszcz, T P Breckon, K Wahren, 'Real-time Mosaicing from Unconstrained Video Imagery for UAV Applications', In *26th International Conference on Unmanned Air Vehicle Systems,Bristol, UK, 11-12 April 2011*, Curran Associates, Red Hook, NY, pp. 359-374.

[8] O Temam, P Yew & B Zang, 'Advanced parallel processing technologies, Berlin: Springer,2011

[9]D G Lowe, 'Distinctive image features from Scale-Invariant Keypoints',International Journal of Computer Vision, Vol. 60, 2, November 2004, pp 91-110

[10] H Bay, T Tuytelaars & L Van Goo, 'SURF: Speeded Up Robust Features', *Lecture Notes in Computer Science* Vol. 3951, 2006, pp. 404-417.

[11]M A Fischler, & R C Bolles, 'Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography',*Communications of the ACM*, Vol. 24, 6, June 1981, pp. 381-395.

[12] Cambridgeincolour.com, (2014),'Understanding Digital Image Interpolation' [online] Available at: http://www.cambridgeincolour.com/tutorials/image-interpolation.htm [Accessed 11 Oct. 2014].