IMAGE STITCHING OF AERIAL FOOTAGE

NG WEI HAEN

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering (Honours) Electrical and Electronic Engineering

Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

April 2021

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature	:	[illegible]
Name	:	Ng Wei Haen
ID No.	:	1602039
Date	:	16/4/2021

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"IMAGE STITCHING OF AERIAL FOOTAGE"** was prepared by **NG WEI HAEN** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Honours) Electrical and Electronic at Universiti Tunku Abdul Rahman.

Approved by,

Signature	:	[illegible]
Supervisor	:	Ng Oon-Ee
Date	:	16 April 2021

Signature	:	[illegible]
Co-Supervisor	:	See Yuen Chark
Date	:	17 April 2021

ABSTRACT

With the advent of modern drones or unmanned aerial vehicles (UAVs), it is used in the application of infrastructure, agriculture monitoring, disaster assessment, etc. It has simplified and automated the site assessment and monitoring procedure. A lot of well-known image stitching software or applications, including Image Composite Editor (ICE), Adobe Photoshop and AutoStitch have been developed to allow users to stitch the images for monitoring or assessment purposes. However, the problems arise when the input data is aerial footage as these software are only taking images as input data. In this project, an image stitching framework is proposed to take aerial footage as input data. The proposed algorithm extracts the frames of the aerial footage and undistorts the bird-eye-effect of the images to remove the noises. Scale-Invariant Feature Transform (SIFT) approach is used to detect and describe the feature points of the extracted frames. The randomized k-d tree of FLANN matcher is utilized to match the feature point pairs between the images. The Lowe’s ratio test is applied to discard the mismatched point pairs. RANSAC is exploited in the homograhy estimation to calculate the corresponding homography matrix and remove the outliers. The images are warped to the key frame of the footage to generate a stitched image by using the computed homography. The algorithm performance is evaluated using the Orchard datasets, consisting of L-shape flight pattern and lawnmower flight pattern. The implemented method successfully stitched the frames extracted from the aerial footage to generate a large scene image beyond the normal resolution.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

APAP	As-Projective-As-Possible
BRIEF	Binary Robust Independent Elementary Features
DBM	Depth Estimation Method
DLT	Direct Linear Transformation
DoG	Difference of Gaussian
FAST	Features from Accelerated Segment Test
FLANN	Fast Approximate Nearest Neighbor Searches
GMS	Grid-based Motion Statistics
GPS	Global Positioning System
ICE	Image Composite Editor
IMU	Inertial Measurement Unit
k-d	k-dimensional
LoG	Laplacian of Gaussian
LR	Lowe’s Ratio
LSH	Locality Sensitive Hashing
NIR	Near-Infrared
ORB	Oriented FAST and rotated BRIEF
RANSAC	Random Sample Consensus
RGB	Red, Green, Blue
ROD	Region of Difference
ROI	Region of Interest
SIFT	Scale-Invariant Feature Transform
SURF	Speeded Up Robust Features
UAV	Unmanned Aerial Vehicle
VFC	Vector Field Consensus
2D	2-Dimensional
3D	3-Dimensional

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 General Introduction

In recent years, unmanned aerial vehicles (UAVs) or drones are no longer exclusive to the military domain. They have been commercialized for leisure and industrial domain, causing skyrocketing usage of the commercial and domestic drone. With the advantages of the capability to fly at low altitude and convenience, UAVs have been widely employed in various fields for remote sensing purposes. Nonetheless, the advancements of GPS, IMU, RGB, NIR and video camera unit installed in small drones have made it becomes the primary device to the aerial applications that require high resolution, low-cost solutions and high portability.

During the flight of the UAV, the camera's location is constantly varying and producing images with various view angles. Generally, the UAV image acquisition modes are divided into three types, including manual acquisition mode (manually triggers to capture aerial images), fixed-point mode (stops at its location to capture aerial images along the predefined flight route) and cruise acquisition mode (takes the aerial images without stopping flying) (Eisenbeiss and Sauerbier, 2011). All three modes mentioned can obtain images over large area coverage and are often applied in the application of environment, infrastructure and agriculture monitoring as well as disaster assessment. Hence, an image stitching technique is needed to stitch the aerial images or footage. Image stitching, also known as image mosaicing is a process that combines numerous images with the overlapped areas to generate a large scene image beyond the normal aspect ratio and resolution. It has been utilized in the daily lives of people, such as artistic photography, medical imaging, etc. Furthermore, it has simplified the assessment and monitoring procedure. Many well-known applications, such as AutoStitch, Image Composite Editor (ICE) and Adobe Photoshop have the functionality to stitch overlapped images to produce a stitched image with wide-angle view.

There are two types of image stitching approaches. The two commonly employed image stitching approaches are pixel-based (direct) approaches and feature-based approaches. These methods will be further discussed in Chapter 2.

1.2 Importance of the Study

Nowadays, image stitching software mainly aims to process images to produce a large scene image. Before utilizing these software or application to perform image stitching with the input of aerial footage, manual keyframe selection is a necessary prerequisite to image stitching. Generally, this always brings inconvenience to the users. Therefore, this project shows the implementation of automated image stitching algorithm for aerial footage.

1.3 Problem Statement

Most image stitching software or application can stitch multiple aerial images with adequate overlapping field of view to generate a panoramic image or stitched image. However, those image stitching software are meant to take only images as input data rather than the footage to generate stitched image. Manual keyframe selection of the aerial footage before using the software is essential and cause the inconvenience to the end-users.

Furthermore, UAVs used in the monitoring process usually perform at low flight altitudes, and the neigboring frames of aerial footage often have a high degree of overlap. Zhao *et al.* (2019) mentioned that the quality of stitched image is profoundly affected by the texture features, overlap, and structure content of the aerial images. These aspects significantly affect the number of matched feature points in image stitching algorithm. Meanwhile, many mismatched point pairs are generated when the images' structure contains high degrees of similarity (ie. orchard or farm), causing the failure of aerial image stitching.

Afterward, Moussa and El-Sheimy (2016) stated that large number of input images is one of the factors that cause the failure of image stitching. Meanwhile, aerial footage usually consists of large number of continuous frames, and the image stitching algorithm has difficulty stitching the frames of the footage. Therefore, aerial image stitching is not straightforward and challenging.

1.4 Aim and Objectives

The main objective of this project is to develop an image stitching program for aerial footage. The details of the objectives are:

(i) To implement an automated image stitching algorithm for aerial footage

(ii) To automate the keyframe selection process for image stitching

1.5 Scope and Limitation of the Study

This project focuses on stitch multiple aerial input images from aerial footage to produce a large scene image. Therefore, the resolution of the stitched image will not be taken into consideration.

1.6 Contribution of the Study

There are many publications and research work on image stitching. This project intends to apply an automated image stitching algorithm for aerial footage to address the inconvenience of manual keyframe selection and stitch multiple frames extracted from aerial footage to produce a large scene image.

1.7 Outline of Report

Chapter 1 provides an overview of the importance of the image stitching technique and the problems of conventional image stitching software. The aim and objectives of the project are described in Chapter 1 as well.

The literature review in Chapter 2 highlights the types of image stitching techniques, including pixel-based (direct) approaches and feature-based approaches. The methodology in Chapter 3 explains the framework proposed for stitching the aerial footage.

The results and discussion in Chapter 4 shows the result generated and the discussions involved. Afterward, the results among various algorithms are compared and discussed. Graphs and tables were generated to visualize the data.

Chapter 5 concludes the results of image stitching of aerial footage and provides suggestions on future methodology improvement and future research directions.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction of Image Stitching

Image stitching is the process of merging multiple images with overlapping regions to generate a large scene image or panoramic image. In recent years, a lot of research works have been conducted to produce a large view for applications in the realm of surveillance, reconstruction, monitoring, medical imaging, etc. Image stitching approaches are generally categorized into two classes, namely pixel-based(direct) approaches and feature-based approaches.

In the early days, the effectiveness and efficiency of the direct methods were sufficient and widely being implemented in professional applications (Lyu *et al.*, 2019). Existing direct approaches are primarily focused on tackling the problems caused by the image properties, including the brightness difference between the overlapping images. Many useful research works have been presented in the topic of image stitching, using the pixel information of the image, including depth, color, geometry and gradient. Deforming and aligning the overlapped images using global estimated image transformation. However, direct-based matching is inefficient and limited in addressing images with multiple planes. It is also inadequate to be implemented in stitching images with complex properties, such as motion change, parallax change and non-planar scene.

Due to the limitation of the pixel-based methods, a lot of research papers introduce feature-based methods to stitch images. Generally, feature-based image stitching pipeline is divided into several algorithmic stages:

(i) Feature Representation
(ii) Image Matching
(iii) Outliers Removal and Robust Estimation
(iv) Image Transform Estimation

The feature representation is used to identify and describe the image patches with high repeatability and distinctiveness. Normally, in the framework and algorithm of image stitching, the process of feature detection and feature descriptor is executed consecutively. The feature detector is to detect the repeatable feature points, also known as interest point, salient point or keypoint based on some criterion, such as the local maximum of some functions in the image (Li et al., 2014). The feature descriptor is usually a vector of values, which describes the image patches around the feature point detected by the detector. It would be interpreted as simple as the raw pixel values, yet it can be as complex as a histogram of gradient orientation. The invariant feature-based approach presented by Brown and Lowe is the most popular method, which is robust and reliable on stitching single planar model and some properties difference among the images such as illumination changes and zoom in. The typical invariant feature-based approaches that are used include SURF, SIFT, ORB and BRIEF, etc. In term of robustness and distinctiveness in solving photometric transformations, a performance evaluation (Hossain and Alsharif, 2007) shows that the SIFT feature outperform others. Yet, the major drawback of this feature is that it imposes high computational burden caused by the scale-invariant feature point detection and the spatial histogram feature description. Other than the complex SIFT float descriptors, the binary feature descriptors, including ORB and BRIEF have become the first selection for the fast processing application due to its fast computations and less storage space needed (Li et al., 2014). However, binary feature descriptors are slightly weaker compared to SIFT-like descriptor in terms of distinctiveness and robustness.

Image matching or feature matching is to match the registered feature points between the images. Two commonly utilized methods, namely brute-force searching methods and tree-based methods. Tree-based methods are formulated to get the k nearest neighbors in the indexing tree efficiently. But it is much more time-consuming as indexing tree set up is necessary before the feature searching process. On the other hand, brute-force searching is much more simple strategy to search the most identical feature correspondence.

Removing outliers from initial feature correspondence is a crucial step in image stitching. Currently, Random Sample Consensus (RANSAC) is the most popular and widely employed robust approach to remove the outlier from the feature correspondence (Li et al., 2014). It is a robust estimation method. It utilizes a minimal set of randomly sampled data to produce image transformation parameters and determine a candidate solution of homography that has the best consensus with the entire feature dataset.

Image transform estimation is usually divided into two classes of method, which are global estimated image transformation and local estimated image transformation. Global estimated image transformation is to deform the image, identify the best-estimated transformation matrix from a particular frame to the reference image, and align the overlapped images globally. Yet, it is limited on a single planar model only. Hence, local estimated image transformation is introduced to solve the limitation. Local estimated image transformation is generally deforming the image into uniform grids, warp the grid, and aligned the image with the estimated transformation matrix (Lyu *et al.*, 2019).

2.2 Pixel-Based (Direct) Approaches

Pixel-based approaches, also known as direct approaches, register multiple images by minimizing pixel-to-pixel dissimilarities in image stitching. Multiple researchers proposed their method by applying the information of the image such as color, gradient, geometry and depth to stitch the images to obtain large stitched image. In this section, multiple pixel-based approaches are discussed.

2.2.1 Gradient domain-based Method

Gradient information is responsive to high-level features, such as edges, lines and contours in the images and conducive to the understanding of image scenes. Levin et al. (2004) proposed an image stitching method in gradient domain (GIST) to perform seamless image stitching, each technique corresponding to a cost function. The outcome and quality of various formal cost function were valued and compared by authors. They aimed to mitigate a cost function based on dissimilarity to each of the input images to overcome the geometric misalignments and photometric inconsistencies between the input images. From the performance evaluation (Levin et al., 2004), the methods under a feathered cost function L1 optimization on the original image gradients (GIST1) was recommended as the standard stitching algorithm in this paper. The utilization of L1 norm is crucial in solving geometrical misalignments of the input images. GIST1 is denoted as the minimum of E_p corresponding to $\hat{I}$:

$$E_p(\hat{I}; I_1, I_2, W) = d_p(\nabla\hat{I}; \nabla I_1, \tau_1 \cup \omega, W) + d_p(\nabla\hat{I}; \nabla I_2, \tau_2 \cup \omega, U - W) \quad (2.1)$$

where

I_1, I_2 = two aligned original images, τ_1 corresponding to a region viewed in image I_i

ω = overlapping region

U = uniform image and set to 1

W = weighting mask

d_p = weighted distance between two regions

E_p = cost function

Furthermore, Levin et al. (2004) optimized the L1 norm by iterating the algorithm to converge the cost functions and mitigate the edge duplication along the seam and seam artifacts between two input images. The image stitching is shown in Figure 2.1. The *ω* indicates the overlap region. Moveover, the top-right image shows a image is simply being pasted onto another. Whereas, the bottom-right image is the stitching result of GIST1 framework.

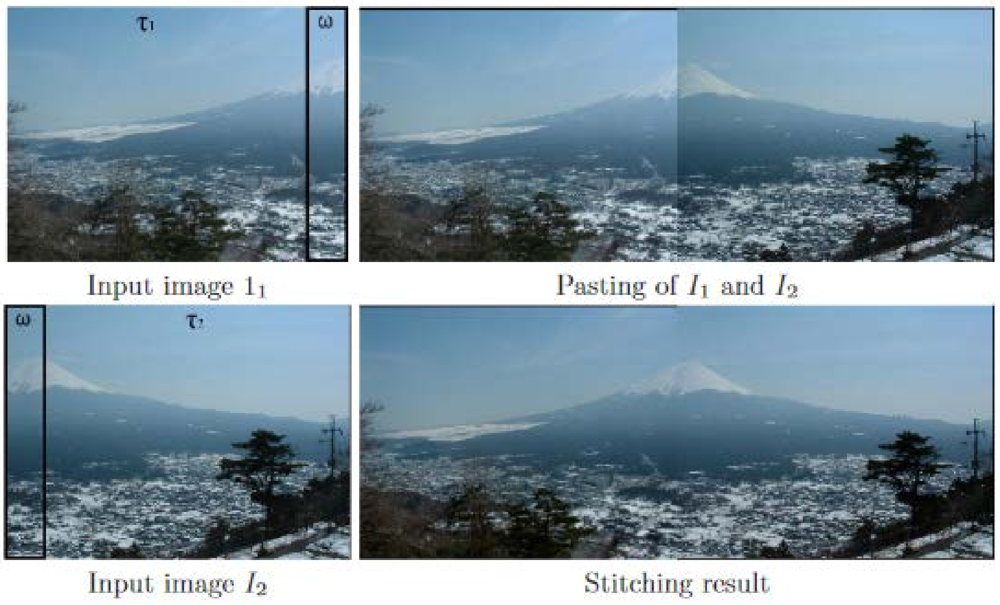


Figure 2.1: Image Stitching (Levin *et al.*, 2004)

Based on the approach proposed, gradient domain-based method can successfully reduce the global inconsistencies in the overlapping region caused by illumination differences between images. The seam artifacts and edge duplication of the stitched image can be reduced by optimization over the image gradient. But, this method has difficulty coping with the image structure with huge misalignment (Zhi and Cooperstock, 2012). The input images must be well-aligned, and it is sensitive to the orientation of the images, which makes this method not suitable for practical application.

2.2.2 Graph-based Method

Ghosting effect is observed when movement occurred in the overlapping region of the stitched image. The method proposed by Levin et al. (2004) may not be able to find a solution when the ghosting caused by object motion in the overlapping region. Hence, Uyttendaele, Eden and Szeliski (2001) proposed a method of constructing a graph to stitch the original images when moving objects presents in the overlapping area among the images. Region of Difference (RODs) assumed to be vertices in a graph and the edges linked the corresponding RODs, as shown in Figure 2.2. Higher weight is appointed to have more central and larger RODs, to discard the low weight vertices, avoiding the moving object discontinuities arising from selected either of the side image. Nevertheless, to further remove the exposure artifacts, they divide each image into blocks, each block corresponding to a quadratic transfer function. They were averaging the functions in each patch with those of their neighbors, further blending the resulting pixel with corresponding transfer function to eliminate the exposure artifacts.

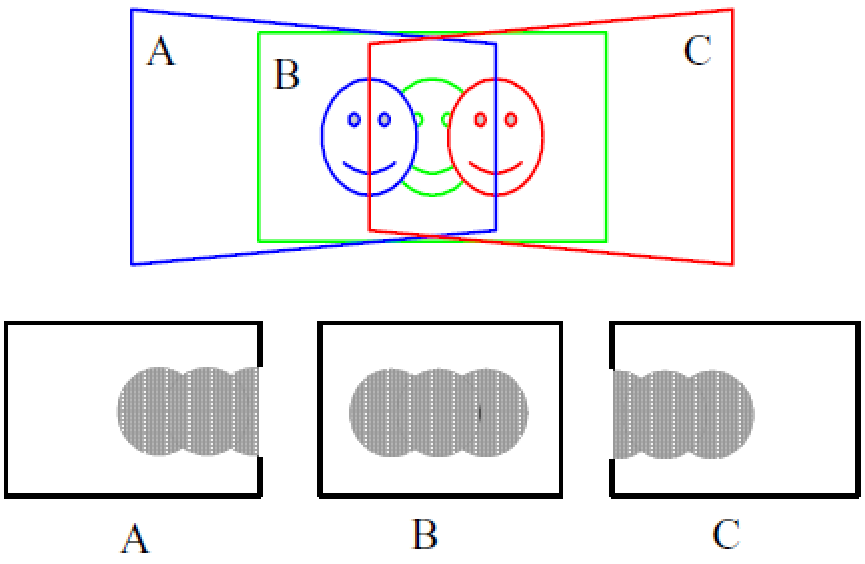


Figure 2.2: (Upper) Image stitching of moving-face in three images. (Bottom) Corresponding RODs affected by motion in the upper image. (Uyttendaele, Eden and Szeliski, 2001)

According to the method proposed by Uyttendaele, Eden and Szeliski (2001), it can eliminate the ghosting effect caused by the moving objects in the overlapping region with further dealing exposure artifacts. However, the complicated calculation caused by pixel blending with the corresponding transfer function is a significant drawback.

2.2.3 Depth-based Method

Other than the graph-based method, Zhi and Cooperstock (2012) took advantage of a smooth transition criterion and depth cues to achieve image stitching. They proposed a Depth Estimation Method (DBM) to solve parallax-related issues and object motion between the images. The static background scenes are divided into overlapping region and non-overlapping region with the use of multiple input cameras. In synthesizing overlapping regions, plane sweep algorithm is used to divide space into layered depth levels. Figure 2.3 illustrates the plane sweep algorithm. The two images (a) and (b) are wrapped onto the parallel sampling planes, where the planes are stacked at different depth levels (d, e and f). Then, a virtual image (c) located at a spot between the input images is synthesized.

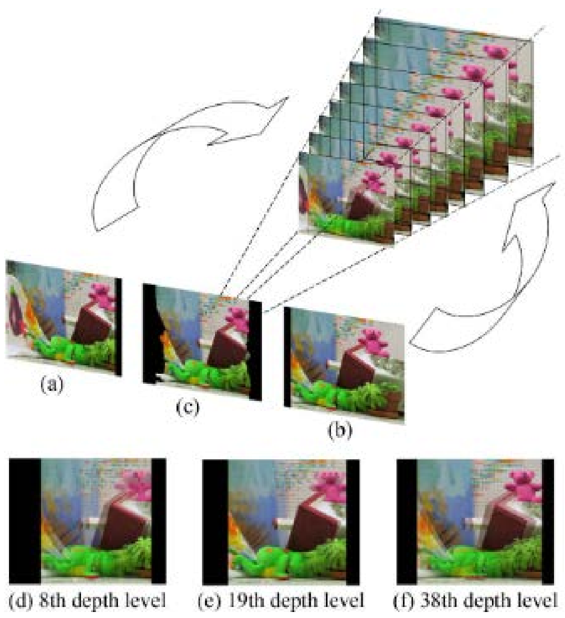


Figure 2.3: Demonstration of plane sweep algorithm (Zhi and Cooperstock, 2012)

Furthermore, to synthesize non-overlapping region, the color segmentation of the input images is to be done first, followed by the depth to be transmitted to neighboring color segments, preserving smooth-appearance connection among them in a stitched result. The demonstration of foreground-background segmentation is shown in Figure 2.4.

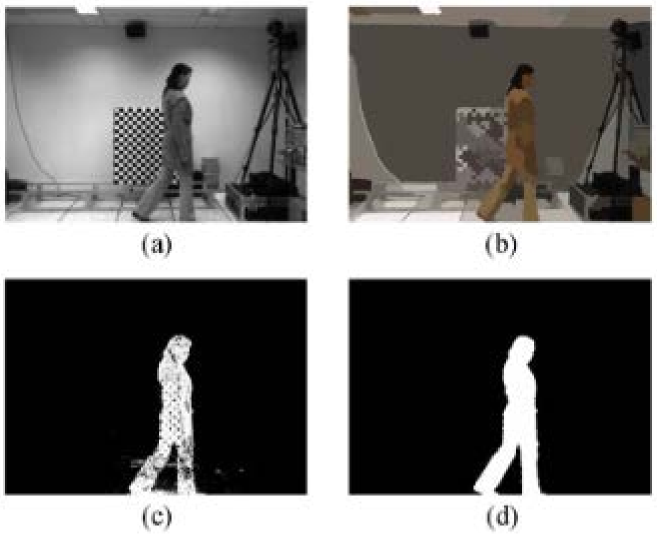


Figure 2.4: Demonstration of foreground-background segmentation. Original input image (a). Color segmented image (b). Foreground layer's raw mask (c). Final mask (d). (Zhi and Cooperstock, 2012)

Nevertheless, their method also targeted on solving the dynamic scene caused by the moving object, such as human walking or running, by introducing optimization of various energy functions for corresponding scenes.

In this paper, the DBM technique can dramatically overcome the parallax problem. On the other hand, the authors mentioned that when the position and orientation of the virtual stitching camera are different from the source camera will results in holes due to occlusion in the non-overlapping region and cause parallax effect (Zhi and Cooperstock, 2012). Their method requires complicated calculations to distinguish non-overlapping and overlapping regions, which leads to high computational burden.

2.2.4 Summary and Comparison

From the direct methods discussed previously, they were mainly targeted on solving the problems affected by the properties of the input images itself, for example, illuminance difference mentioned in Levin et al. (2004). However, these methods perform computation for each pixel on input images, becoming computational intensive if the algorithm is complex or the input dataset is huge, such as frames of aerial footage (Pravenaa and Menaka, 2016). Hence, these methods are not adopted by commercial image stitching software and not an effective and suitable approach to stitch aerial images. Moreover, the existing direct approaches are constrained to stitch the image with single planar and parallax-free scene. These limitations make direct matching is generally inefficient on stitching images with complex properties, such as aerial images. The comparison between different direct methods is shown in Table 2.1.

Table 2.1: The comparison of direct methods

Approach	Principal	Strength	Drawback
Levin et al. (2004)	Gradient Weighting	• Seamless	• Only aligned input image
Uyttendaele, Eden and Szeliski (2001)	Graph Structure	• Remove ghosting caused by motion	• Complicated Calculation
Zhi and Cooperstock (2012)	Depth and Color	• Solve certain degree of depth discontinuity	• Limited orientation of input images • Complicated calculation

2.3 Feature-based Approaches

Unlike Pixel-based (Direct) approaches, feature-based approaches are able to evaluate the 2D motion model by adopting the sparse feature points. Several researchers proposed different optimization strategies on stitching the images to obtain huge stitched image. As mentioned early in this chapter, feature-based method is generally divided into several processes, including feature representation, image matching, outliers removal and robust estimation, and image transform estimation. Feature-based methods are discussed in this section.

2.3.1 Sparse Feature-based Method

In the early years, sparse feature-based method had been nominated image stitching. A traditional approach was first introduced by Brown and Lowe (2007), using local invariant feature approach to produce a panoramic image. A well-known image stitching software tool, AutoStitch was designed based on their approach. Despite the rotation, illumination change and zoom in the input images, their algorithm can provide a reliable matching of the panoramic image in an unordered dataset. Scale-Invariant Feature Transform (SIFT), a notable feature detector and descriptor was developed by Brown and Lowe and presented in this paper. SIFT algorithm is one of the state-of-the-art algorithms. It utilizes the difference of Gaussians (DoG) to build the difference of Gaussian scale-space. The feature points are extracted when different spatial scales are detected on images, and the unstable feature points are removed. Then, the principal direction of the feature points is identified. Once the principle direction is determined, the SIFT feature descriptor is generated to prevent the mismatch caused by noise, rotation, scale and illumination. In terms of feature matching, Brown and Lowe matched the sparse feature between the input images and employed probabilistic model for image match verification. It can estimate the global homography transformation using RANSAC to deform and align the overlapped images. Brown and Lowe further introduced a robust bundle adjustment and performed automatic panorama straightening on stitched view. Nevertheless, multi-band blending and gain compensation were applied to generate a seamless stitched image.

The approach proposed by Brown and Lowe (2007) was limited to single homography transformation. Thus, Brown presented an improved model, dual-homography model to solve the problem by aligning the images involving numerous planes in the overlapping area (Gao, Kim and Brown, 2011). Two dominant planes, namely background plane and foreground plane were obtained by dividing the input image, and homography transformation estimation was performed to each plane.

In aerial application, Zhao et al. (2019) exploited the scale-invariant feature transform (SIFT) to perform fast aerial image stitching for crop growth monitoring. To reduce the high computational burden in image stitching when using the standard SIFT algorithm, they proposed a simple and improved sparse-feature based method to increase the accuracy and efficiency of aerial image stitching in crop growth monitoring. Their framework is to optimize the dynamic setting of the contrast threshold in the DoG scale-space in SIFT to enhance the efficiency of the algorithm. The optimization is done by evaluating the image contrast that can express the difference in image details and calculating the new contrast threshold in the DoG scale-space. Meanwhile, the local features of the aerial image are retained. Furthermore, the mismatched point pairs in the non-overlapping region are deleted to enhance the stitching accuracy and reduce the processing period according to the relative positional relationships of the aerial image. The algorithm eliminates the mismatch point pairs by evaluating the longitudinal overlap and transverse overlap between the images. Figure 2.5 shows the schematic diagram for UAV taking images in crop growth monitoring.

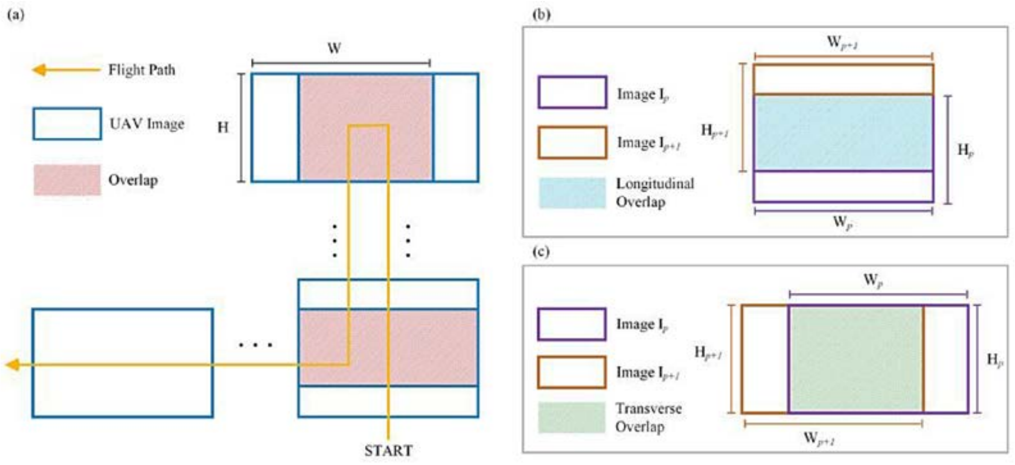


Figure 2.5: Schematic diagram for UAV taking images in crop growth monitoring. In (a), the flight trajectory is represented with the orange; the captured image is represented with the blue; and the overlap of images is represented with the shaded-blue rectangles. (b) denotes the transverse overlap with 70%, whereas (c) indicates the longitudinal overlap with 75%. (Zhao *et al.*, 2019)

In the method presented by Brown and Lowe (2007), it assumes that the camera motion is purely rotational and the group of transformations that the input images may undergo is a special group of homographies. In reality, it is very rare to obtain such an ideal dataset (Chen et al., 2019). Usually, aerial images captured by UAV are hard to fulfill such a perfect situation, where the images might not be on a single plane, except the aerial input data used by Zhao et al. (2019). Because the aerial crop field images are generally planar compared to other aerial images, such as buildings. Thus, it might be difficult to produce good stitched results when nonrigid input images are applied. Nevertheless, SIFT-liked feature detector and descriptor imposes a large computational burden and not suitable for real-time systems (Rublee et al., 2011).

2.3.2 Binary Descriptor-based Method

Apart of SIFT-liked feature detector and feature descriptor, Rublee et al. (2011) proposed a scheme of binary-based feature detector and descriptor, known as

Oriented Fast and Rotated BRIEF (ORB), where it is built on the FAST feature point detector and BRIEF feature descriptor. It is much more robust against the rotational problem compared with purely BRIEF algorithm (Li et al., 2014). This scheme was built to address the limitations imposed in SIFT, such as large computational burden. The feature points identified in the images is described in binary string instead of spatial histogram. For the requirement of the high-quality image stitching system with low computational time, Adel, Elmogy and Elbakry (2015) made an in-depth comparative study on the feature detector and descriptor. It concluded that ORB algorithm is the best among others, including SIFT, SURF, FAST and HARRIS.

Furthermore, Li et al. (2014) developed a high-speed aerial video stitching scheme. They employed the ORB algorithm to find the feature point and describe it. Some modification had been made on the algorithm by rescaling a scaling factor to $\sqrt{2}$ on the extracted feature points in five scale images separately to produce multi-scale features. Nonetheless, the algorithm can generate the keyframe dynamically to mitigate accumulation errors. In terms of outlier removal, RANSAC is often being used, yet, the major drawback it imposed is high processing time in relation to the additional outlier. Thus, an motion- and appearance-based spatial and temporal filter was implemented to eliminate most of the outliers before RANSAC to improve the processing time efficiency. They used the scale of the feature point in calculating the appearance coherence and motion coherence in the filter:

$$\phi_{appearance}\left(K_t, K'_{ref}\right) = \begin{Bmatrix} 1 & if\ min\left(S_t - \lambda^1_{t-1} S'_{ref}, S_t - \lambda^2_{t-1} S'_{ref}\right) \leq T_s \\ 0 & otherwise \end{Bmatrix} \tag{2.2}$$

$$\phi_{motion}\left(K_t, K'_{ref}\right) = \begin{Bmatrix} 1 & if\ \left\|X_t - H^{-1}_{t-1,ref} X'_{ref}\right\|_2 \leq T_{dis} \\ 0 & otherwise \end{Bmatrix} \tag{2.3}$$

$$\phi\left(K_t, K'_{ref}\right) = \phi_{apperance}\left(K_t, K'_{ref}\right) \cdot \phi_{motion}\left(K_t, K'_{ref}\right) \tag{2.4}$$

where

T_s = difference of maximal scale between the K'_{ref} and keypoint K

λ_{t-1}^{1} and λ_{t-1}^{2} = scale of inliers

$\mathrm{T_{dis}}$ = distance threshold between two adjacent frames

$\mathrm{X_t}$, $\mathrm{H_{t-1,ref}^{-1}}$, $\mathrm{X'_{ref}}$ = transform matrix of image from the initial image at time $\mathrm{t} - 1$ to the reference image, ref

De Lima and Martinez-Carranza (2017) proposed a real-time based aerial image stitching method by using the characteristics of ORB binary descriptor along with a feature matching approach based on Locality-Sensitive Hashing (LSH) technique. ORB descriptor is the most widely employed binary feature as it offers some advantages, including fast comparison, ease to compute, robustness to affine transformations and low memory footprint (De Lima and Martinez-Carranza, 2017). The characteristics of ORB, including invariant to in-plane rotation and binary string-based descriptor, help the ORB vectors good to be organized through a hash table. The low memory footprint used by ORB vector can prevent the increase of memory resources. Nonetheless, it is processing time efficient in storing and fetching data from memory as the feature correspondence is found in the hamming space. Hashing technique is divided into two main algorithmic steps, which are using hashing function to fill and search the tables consecutively. For the filling step, hashing keys is chosen by using the consecutive subset of bits, which allow the descriptors can be rapidly stored into different tables. The process proceeds for searching the tables to match the feature between the current frame and reference frame once the tables were filled. For searching the tables, it is divided into four algorithmic steps:

(i) Search the hashing keys against different tables

(ii) Determine the hamming distance for all the descriptors in the bucket for each table

(iii) Determine the minimum distance given for each table

(iv) It considered match if the minimum distance does not exceed a preset threshold

Next, they employed RANSAC to remove the additional outlier and compute the best homography matrix to perform global homography transformation on the images to produce a stitched image.

According to the methods proposed by Li et al. (2014) and De Lima and Martinez-Carranza (2017), both can significantly reduce the processing time of image stitching. Nevertheless, Li et al. (2014) method can reduce the accumulation error by generating the keyframe dynamically to prevent ghosting effect. However, it was only targeted on the high-altitude input images. Thus, it might be challenging to produce a good stitched result when input aerial images are taken in low altitude or huge parallax shown in the images. For De Lima and Martinez-Carranza method, although the real-time performance was accomplished, yet, the overlapping region between the input images cannot be aligned accurately.

2.3.3 Mesh-based Alignment Method

Mesh-based alignment method produced an outstanding result in image stitching in the early days. Due to excellent results, the mesh-based alignment method is deemed as one of the state-of-the-art methods until today (Lyu *et al.*, 2019). The images are split into uniform meshes, and each mesh corresponding to a transformation. This method was first proposed by Zaragoza et al. in 2014. Then, more and more research works adopted this method in their optimization method on various prior problems.

To solve the misalignment artifacts or ghosting produced during the single planar wrap and transformation of image stitching as well as expensive postprocessing algorithm, Zaragoza et al. (2014) proposed a method, known as as-projective-as-possible (APAP) image alignment to dramatically reduced the ghosting without jeopardizing the geometric realism of the stitched image. The input images were divided into uniform meshes, and each mesh undergoes a local homography estimation with a Moving Direct Linear Transformation (Moving DLT) and RANSAC. Moreover, a simultaneous refinement of bundle adjustment proposed in the research paper (Brown and Lowe, 2007) was employed to align multiple images accurately to obtain large panoramic scene. Figure 2.6 demonstrates the image stitching using APAP image alignment strategy.

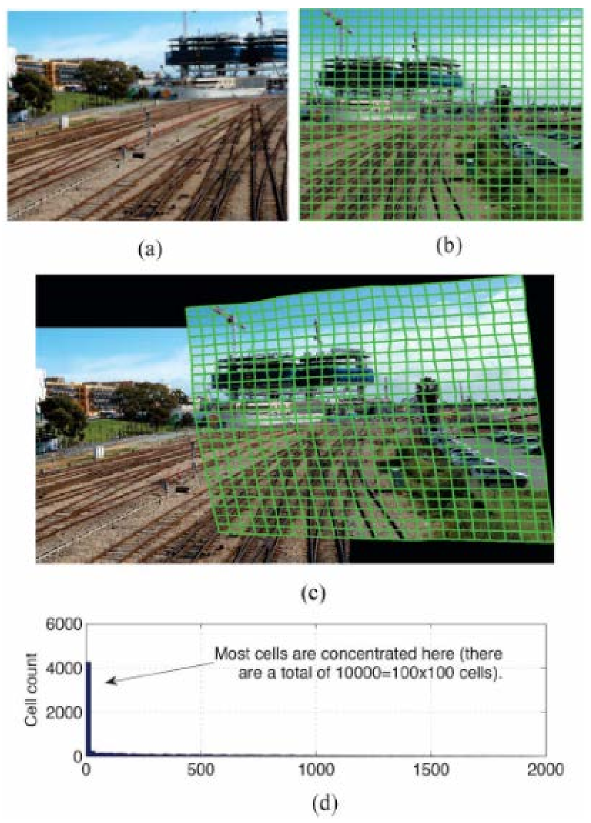

(a)
(b)
(c)
6000
4000
2000
0
Cell count
Most cells are concentrated here (there are a total of 10000=100x100 cells).
0
500
1000
1500
2000
(d)

Figure 2.6: Mesh-based image alignment. (a) Input image (b) Meshed input image (c) Warped and aligned images. (d) Histogram of number of weights for each cell. (Zaragoza et al., 2014)

Then, Liu and Chin (2016) developed a mesh-based alignment optimization algorithm to automatically identify the misaligned overlapping regions and add correct point correspondences to improve the alignment of the composition image and enhance the flexibility of the warp. They encapsulated their technique in a scheme called data-driven warp adaption scheme, which repeatedly inserts new appropriate points in the overlap area. In this paper, the comparison of the result for APAP and their method (APAP + Correspondence Insertion) has been shown, and it can be concluded that their method produces better results compare to pure APAP method in terms of the alignment of overlap regions.

2.3.3.1 Image Differentiating Technique for Aerial Image Stitching

In the traditional image stitching method, single planar perspective transformation with bundle adjustment is leading to ghosting error. Inspired by the mesh-based alignment method introduced by Zaragoza et al. (2014), Chen et al. (2019) employed the image differentiating technique with Moving DLT to diminish the error and tolerate the parallax in the image. Moreover, they introduced a nonrigid matching algorithm to enhance the accuracy of feature matching and improve the bundle adjustment proposed by Brown and Lowe to make the system applicable for aerial image stitching effect. At the feature matching stage, a SIFT-based nonrigid matching algorithm using Vector Field Consensus (VFC) is employed to obtain the corresponding relationship of feature point between images with further filtering of outliers from the inliers. Chen et al. (2019) mentioned that global homography could not efficiently match the two aerial images. Thus, they introduced a location-dependent homography with moving DLT to simultaneously improve the projection functions and mitigate the cumulative error when stitching numerous input images.

2.3.3.2 Summary of Mesh-based Alignment Method

By evaluating the results based on the strategies proposed by Chen et al. (2019), the ghosting effect is efficiently reduced when only two aerial images are involved in the stitching process. In the paper, the algorithm that robust against parallax change, noise and blurriness has been proved. However, when stitching multiple images, the stiffness of the stitched image has to be sacrificed (Chen et al., 2019). The distortion and the misalignment on the overlapping region are observed when the parallax of the input images is particularly huge, such as the buildings in low-altitude images.

2.3.4 Summary and Comparison

Sparse feature-based approaches have dominated the stitching technique. It performs well on standard datasets and numerous challenging datasets. These works are primarily focussed on producing better results than time efficiency, resulting in that they are not suitable for real-time applications. They deform and align the images with global homography estimation. Unlike the mesh-based alignment method, they do not categorize non-overlapping and overlapping regions, resulting in ghosting and distortion. Furthermore, binary descriptor-based methods are concerned with designing a processing time-efficient system to stitch the images. However, it is often relatively weak in terms of distinctiveness and robustness during the image stitching process. Mesh-based alignment methods can eliminate some prior limitations to improve the alignment, while local distortion is still inevitable due to the coupling relationships between different constraints. Table 2.2 shows the comparison of feature-based methods.

Table 2.2: The comparison of feature-based methods

Approach	**Principal**	**Strength**	**Drawback**
(Brown and Lowe, 2007)	Simple Sparse feature matching	• Automatic	• Limited to single plane model
(Zhao et al., 2019)	Modified Sparse feature matching	• Fast	• Limited to single plane model and rigid input images
(Li et al., 2014)	Binary descriptor	• Real-time performance	• Limited to high altitude image
(De Lima and Martinez-Carranza, 2017)	Hashing-based matching	• Real-time performance	• Poor alignment accuracy
(Zaragoza et al., 2014)	Mesh-based	• Multiple transformations	• Local distortion
(Chen et al., 2019)	location-dependent homography with moving DLT	• Reduce cumulative error • Multiple transformations	• Limited to small parallax images

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Overview of Project Work Plan

Figure 3.1 shows the general flow of the project work plan. One aerial footage dataset (UMN Horticulture Field Station) that is publicly available is chosen for this project. The algorithms of the feature representation methods (SIFT and ORB), feature point matchers (FLANN and Brute-Force Searching) and feature match filtering technique (LR and GMS) are used for the evaluation. The details of the work plan are discussed in the subsequent chapters.

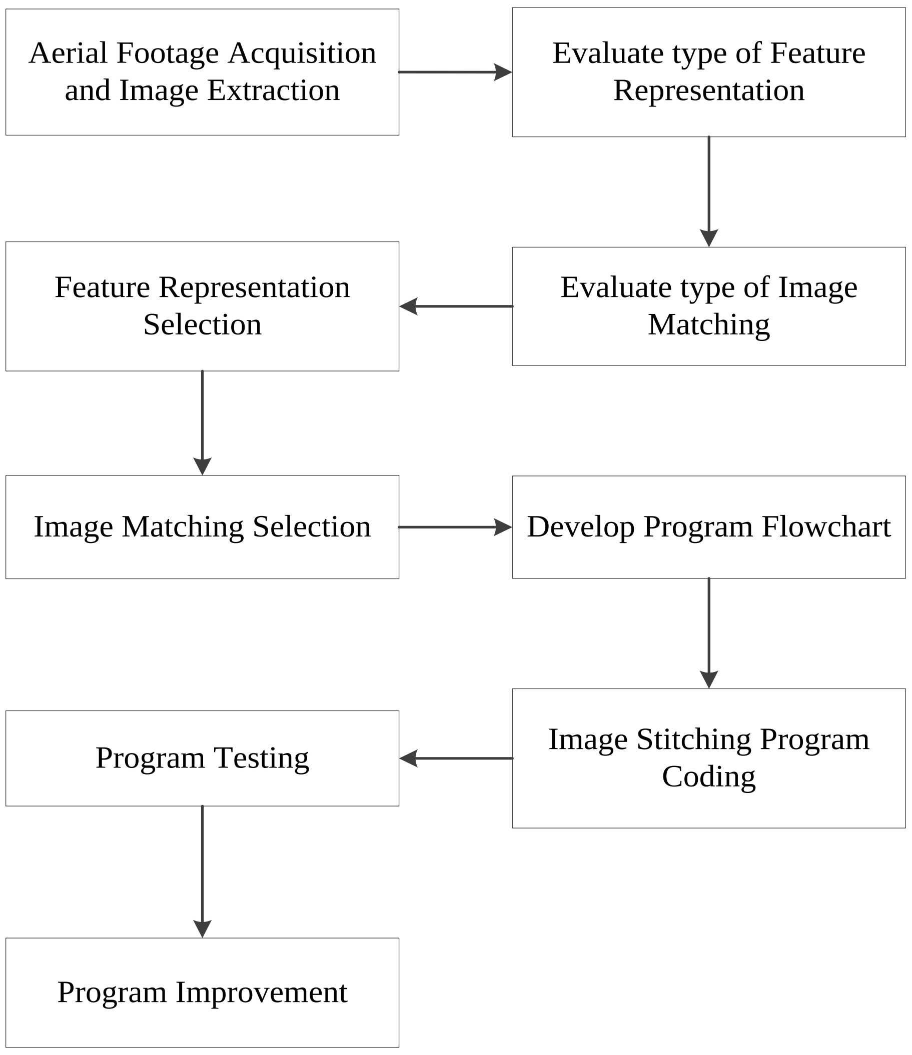

Aerial Footage Acquisition and Image Extraction
Evaluate type of Feature Representation
Evaluate type of Image Matching
Feature Representation Selection
Image Matching Selection
Develop Program Flowchart
Image Stitching Program Coding
Program Testing
Program Improvement

Figure 3.1: Flowchart for Project Workflow

3.2 Introduction

The proposed framework's flow chart is shown in Figure 3.2.

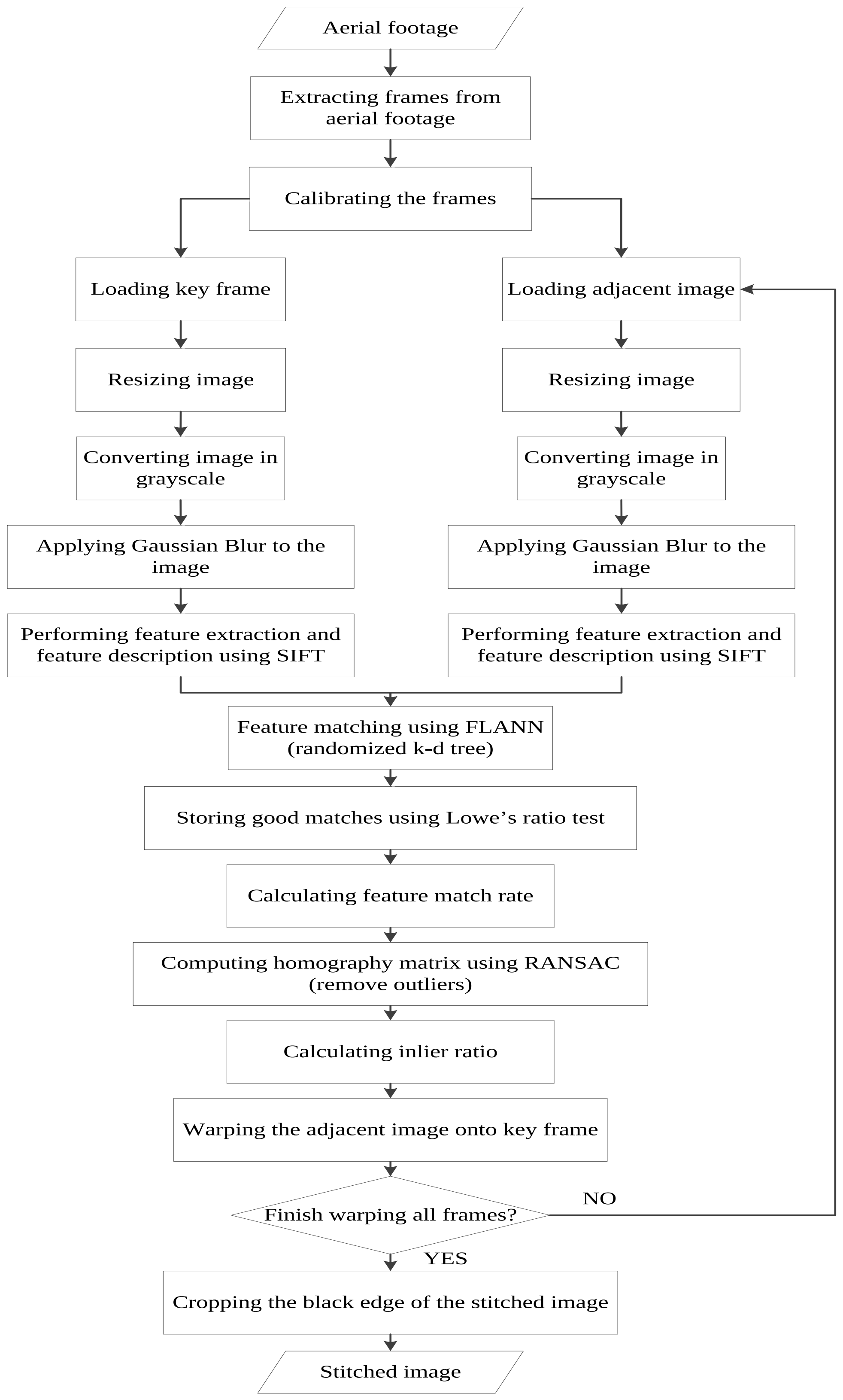

Aerial footage
Extracting frames from aerial footage
Calibrating the frames
Loading key frame
Loading adjacent image
Resizing image
Resizing image
Converting image in grayscale
Converting image in grayscale
Applying Gaussian Blur to the image
Applying Gaussian Blur to the image
Performing feature extraction and feature description using SIFT
Performing feature extraction and feature description using SIFT
Feature matching using FLANN (randomized k-d tree)
Storing good matches using Lowe's ratio test
Calculating feature match rate
Computing homography matrix using RANSAC (remove outliers)
Calculating inlier ratio
Warping the adjacent image onto key frame
Finish warping all frames?
NO
YES
Cropping the black edge of the stitched image
Stitched image

Figure 3.2: Flowchart of the proposed framework

The method's performance in every subchapters, such as feature representation, feature matching and feature match filtering is evaluated by analyzing the feature match rate and inlier ratio. Moreover, the capability of stitching images is the primary concern of the proposed framework. The observation on the alignment of the frames in the stitched images is recorded in this project. Further information about the performance evaluation will be discussed in Chapter 4.

3.3 Dataset

Table 3.1 shows the details of aerial footage from the dataset. And Figure 3.3 illustrates the examples of aerial footage of orchard in the dataset. The aerial footage provided in the dataset was shot by a downward-pointing camera, GoPro Hero 3. Meanwhile, the dataset contains the GoPro Hero 3 camera information, including focal length, pixel error, skew, distortion coefficient, and principal point.

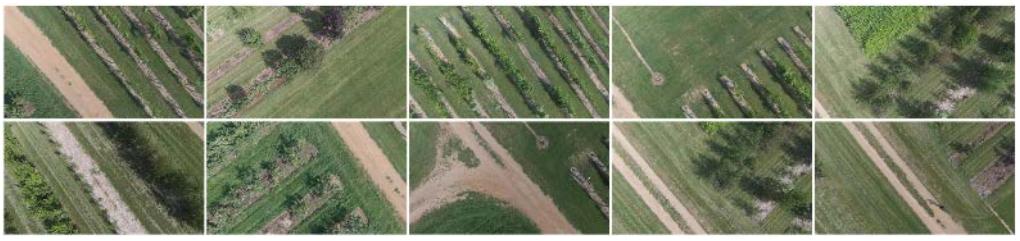


Figure 3.3: Examples of aerial footage of orchard

Table 3.1: Details of aerial footage from the dataset

Location	UMN Horticulture Field Station (Orchard)
Area Covered	800 m^2
Camera	GoPro Hero 3
Flying Height (altitude)	13 m
Flying Speed	1 m/s
Flight Pattern	L-shape Flight Pattern
Video Type	MP4
Length	00:00:40
Frame Width	1920
Frame Height	1080
Frame rate	30 fps

3.4 Frame Extraction from Aerial Footage

The aerial footage undergoes frame extraction in the first place to extract the frames from the video. The drone used in the dataset moved at one meter per second and generated thirty frames per second recorded footage. Since the video was shot at low altitude, extracting one image every fifty frames is sufficient to obtain at least sixty percent of the overlapping region.

3.5 Camera Calibration

Images captured or recorded from the camera are required to be calibrated. This is because the captured images or videos are inevitably being distorted by the camera's lens itself. Ju and Kang (2010) mentioned that it is challenging to find the correspondence between images directly or linearly during image stitching as the lens distortion is impossible to be linearly represented. The authors also stated that the lens distortions is the main cause that induce the mismatches of image stitching (Ju and Kang, 2014).

In this project, Zhang's arithmetic of camera calibration in OpenCV is applied. The approach of Zhang (2000) can remove two major distortions in images, including radial distortion and tangential distortion. These distortions are caused by the limitation of artificiality in camera production. The radial distortion and tangential distortion can be solved by using the following formulas (Yuan, Zhu and Su, 2011):

$$x_{corrected,\ radial} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.1}$$

$$y_{corrected,\ radial} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.2}$$

$$x_{corrected,\ tangential} = x + [2p_1 xy + p_2(r^2 + 2x^2)] \tag{3.3}$$

$$y_{corrected,\ tangential} = y + [p_1(r^2 + 2y^2) + 2p_2 xy] \tag{3.4}$$

Where,

$r = (x - c_x)^2 + \left(y - c_y\right)^2$

$\left(c_x, c_y\right)$ is the optical center

(x, y) is the real plane offset coordinate caused by lens distortion

k_1, k_2, k_3 are radial distortion parameters

p_1, p_2 are tangential distortion parameters

The combination of radial distortion and tangential distortion parameters is known as distortion coefficients (k_1, k_2, p_1, p_2, k_3).

Apart from the distortion coefficient, the camera's intrinsic and extrinsic parameters are crucial parameters to obtain the transformation of certain points from 3D space into image coordinates. The intrinsic parameters contain information such as focal length and optical center to form the camera intrinsic matrix. It is denoted as a 3x3 matrix:

$$camera\ intrinsic\ matrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

Where,

$\left(c_x, c_y\right)$ is the optical center of the camera

$\left(f_x, f_y\right)$ is the focal length of the camera

The extrinsic parameters contain information such as rotation and translation vectors to form camera’s extrinsic matrix. It is denoted as a 2x2 matrix:

$$camera\ extrinsic\ matrix = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \tag{3.6}$$

where R is rotation vector and t is translation vector.

3.6 Image Pre-processing

In the image pre-processing stage, the size of input images with Red, Green and Blue (RGB) scale are scaled-down by three size ratios, which means scaling down 1920x1080 images into 640x360 images.

After that, the images are converted into grayscale to speed up the computational process. Converting images to grayscale is one of the frequently used image enhancement techniques. RGB image represents the intensity levels of image in 24 bits as it contains three channels (red, green and blue). Whereas, the grayscale image has one channel and represents the intensity levels in 8 bits. For the application in this project, feature extraction in Chapter 3.7 does not require the RGB color information to extract the features of image. Thus, the complex RGB color information is the noise to the image and will decelerate the algorithm's computation period. Hence, converting the image to grayscale can reduce the image to one channel to obtain a quicker computation process.

Upon completing the conversion process, Gaussian blur is then applied to denoise the grayscale image. Gaussian blur, also known as Gaussian smoothing, is used to blur the image, reducing image’s Gaussian noise by a Gaussian filter. Figure 3.4 illustrates the image pre-processing flow diagram.

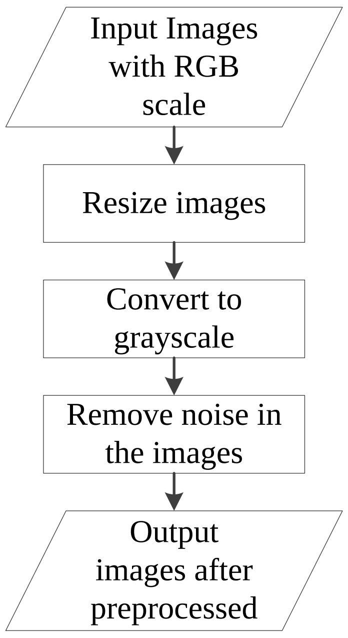

Input Images with RGB scale
Resize images
Convert to grayscale
Remove noise in the images
Output images after preprocessed

Figure 3.4: Image Pre-processing Flow Diagram

3.7 Feature Representation

The details about the sparse feature-based method and binary descriptor-based method have been explained in Chapter 2. In this project, Scale Invariant Feature Transform (SIFT) is used in feature representation due to its robustness to the geometric changes and properties difference of the image.

3.7.1 Scale-Invariant Feature Transform (SIFT)

In this step, the feature detection and the feature description on the image are executed consecutively. The image undergoes feature representation using SIFT algorithm framework developed by Lowe (2004). SIFT possesses the capability of invariance against image rotation, scaling transformations and image translation. It exhibits excellent robustness to light changes, affines transformation and noise. The framework is generally divided into four major steps, namely scale-space extrema detection, keypoints localization, orientation assignment and descriptor computation. Figure 3.5 shows the flow of four main steps of SIFT framework.

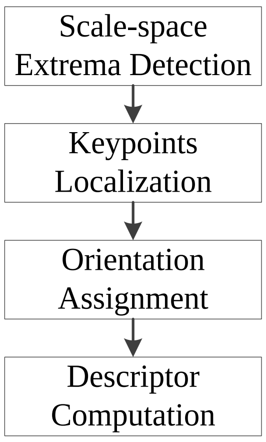


Figure 3.5: The major steps of SIFT framework

3.7.1.1 Scale-space Extrema Detection

In order to detect the candidate feature points that are scale invariance, the image is resized with uniformly increasing scale factors in the first place to construct an image pyramid. Figure 3.6 illustrated the combination of images at different scales to form octaves.

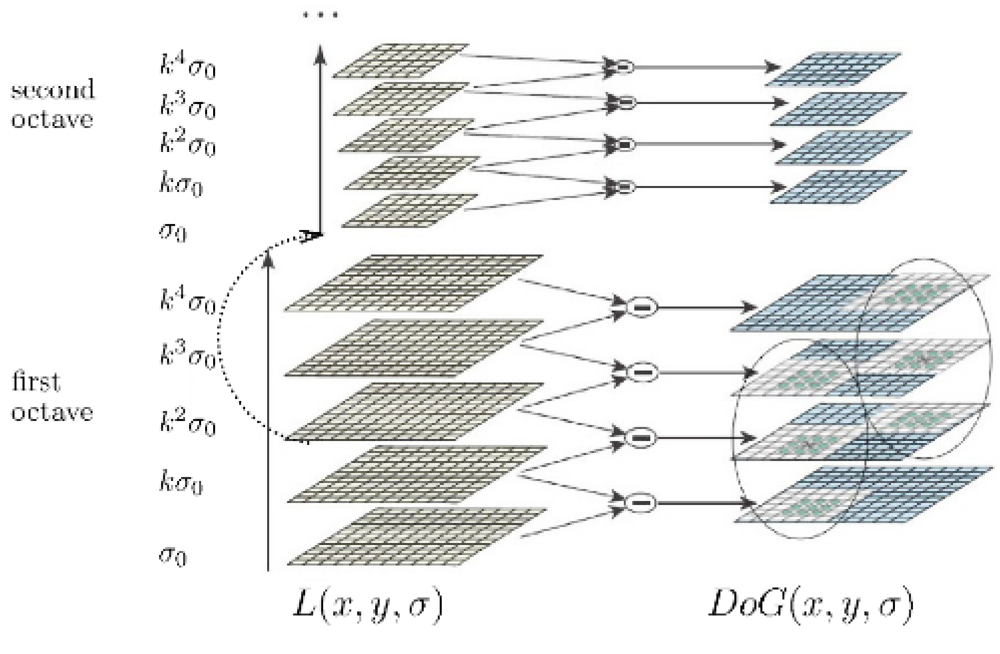


Figure 3.6: Octaves of image pyramid (Younes, Romaniuk and Bittar, 2012)

Each octave is further applied with Gaussian smoothing by carrying out convolution operation in x and y, which formulated as the equation at the following below:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}} \tag{3.7}$$

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{3.8}$$

Where,

σ = Scaling factor. The bigger the value, the greater the blur

$L(x, y, \sigma)$ = Smoothed image of its octave

$G(x, y, \sigma)$ = Gaussian smoothing

$I(x, y)$ = Image at different scale factor, σ

Lowe (2004) suggested that σ_0 = 1.6, where the value is obtained from the equation below:

$$\sigma_0 = \sqrt{\sigma_1^2 + \sigma_2^2} \tag{3.9}$$

Where σ_1 equal to 1 while σ_2 equal to 1.26. This equation is applied consecutively to the images to build the octaves. In Lowe (2004) paper, he suggested that the $k = \sqrt{2}$ and the number of scale levels = 5 as optimal values. In short, each octave contains five scale levels which contain $\sigma_0, k\sigma_0, k^2\sigma_0, k^3\sigma_0$ and $k^4\sigma_0$. Then, the image is rescaled to its half for the next octave. He also advised that three-octave is the optimal number for each image.

The SIFT algorithm utilizes the Difference of Gaussians (DoG), which is an approximation of Laplacian of Gaussian (LoG) to detect the scale-space extrema. The pyramid of DoG is calculated from the image pyramid, $L(x, y, k^n\sigma)$. The result of each DoG is obtained as the subtraction of adjacent Gaussian smoothed images at different scales $(k^n\sigma_0 \; and \; k^{n+1}\sigma_0)$ in the pyramid. It is formulated as:

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma) \tag{3.10}$$

Upon the DoG are found, the corresponding local extrema are identified over three consecutive DoG images within one octave. For instance, a pixel in a DoG image is compared with its eight neighbors pixel as well as the nine neighbors at inferior and superior scales, as shown in Figure 3.6 (circles).

3.7.1.2 Keypoints Localization

Once the candidate feature points' locations are found, refinement of the feature points is needed to accurately localize the points by rejecting those low contrasted keypoints and edge keypoints. In SIFT framework, a second-order Taylor approximation is utilized to calculate the local extremum of the scale-space function $D(x, y, \sigma)$ in the sample keypoint:

$$D(x) = D + \frac{\partial D^T}{\partial x} x + \frac{1}{2} x^T \frac{\partial^2 D}{\partial x^2} x \tag{3.11}$$

where $x = (x, y, \sigma)^T$ is the offset from the sample keypoint, while the D and its derivatives are computed at the sample keypoint. The location of the extremum, $\hat{x}$ is equal to zero, where it is determined by taking the derivative of the function $D(x)$:

$$\hat{x} = \begin{pmatrix} \hat{x} \\ \hat{y} \\ \hat{\sigma} \end{pmatrix} = -\frac{\partial^2 D^{-1}}{\partial x^2} \frac{\partial D}{\partial x} \tag{3.12}$$

As suggested by Brown and Lowe (2002), the derivative of D and the Hessian are approximated by the difference of Gaussian (DoG) in the neighborhood of the sample keypoint. When the offset, x is larger than the threshold in any one of the three dimensions, it indicates that the extremum lies closer to a neighbor of sample keypoint than sample keypoint itself. In this case, the sample keypoint is changed and the computation is performed again from the new point. It then computes the value of DoG at the extremum, $\hat{x}$ after the sample keypoint is changed. The keypoint is discarded to reject the unstable extrema with low contrast if the value of $|D(\hat{x})|$ less than contrast threshold.

Since the DoG has a strong response to the edges, the candidate keypoint localized along the edges of small curvature needs to be rejected as these edges act as the noises to the DoG function. Thus, the only point on huge-curved edges, such as corners, will be selected as the final keypoint. To only keep the DoG of obvious-curved edges, the principal curvatures of D is computed. It is computed from a 2x2 Hessian matrix, H at localized keypoint. These are proportional to the eigenvalues α and β of Hessian matrix, H:

$$H = \begin{pmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{pmatrix} \tag{3.13}$$

To avoid the explicit computation of the eigenvalues, the determinant and trace values of the matrix are used. Let r be the ratio between the largest and smallest eigenvalue, $\alpha = r\beta$. Afterward, the ratio between the square of sum and the product of the eigenvalues is computed:

$$\frac{Tr(H)^2}{Det(H)} = \frac{\left(D_{xx} + D_{xy}\right)^2}{D_{xx}D_{yy} - (D_{xy})^2} = \frac{(\alpha + \beta)^2}{\alpha\beta} = \frac{(r\alpha + \beta)^2}{r\alpha\beta} = \frac{(r + 1)^2}{r} \tag{3.14}$$

Lowe (2004) suggested that the edge threshold is set to 10 as the optimal value to discard any candidate keypoints’ ratio that is greater than the edge threshold.

3.7.1.3 Orientation Assignment

Subsequently, consistent orientation is assigned to every detected keypoints to achieve invariance against the image rotation. The magnitude, $m(x, y)$ and the orientation, $\theta(x, y)$ of the gradient are computed using the selected closest scale factor, σ for each keypoint, (x, y) and the Gaussian-smoothed image, $L(x, y, \sigma)$ at this scale:

$$m(x, y) = \sqrt{L_x^2 + L_y^2} \tag{3.15}$$

$$\theta(x, y) = tan^{-1}\frac{L_y}{L_x} \tag{3.16}$$

Where,

$L_x = L(x+1,y) - L(x-1,y)$

$L_y = L(x,y+1) - L(x,y-1)$

Based on the calculated gradient magnitude and the direction in the region around the keypoint, a histogram of orientation is formed from the image, $L(x,y,\sigma)$. The established orientation histogram consists of thirty-six bins covering a three hundred and sixty degrees range of orientations. It is weighted by its magnitude of gradient and a Gaussian-weighted circular window with a scale factor, σ that is 1.5 times bigger than the scale of feature point:

$$\omega m(x,y) = m(x,y)\frac{1}{2\pi(1.5\sigma)^2}e^{-\frac{dx^2+dy^2}{2(1.5\sigma)^2}} \tag{3.17}$$

Where,

$\omega m(x,y)$ = weighted magnitude of the gradient at location (x,y)

$m(x,y)$ = magnitude of the gradient at location (x,y)

dx = distance in x to the feature point

dy = distance in y to the feature point

σ = scale factor

The keypoint's dominant direction is characterized by finding the maximal value of bins in the orientation histogram. Any bins greater than eighty percent of the maximal value is also associated with its computed orientation. Therefore, additional keypoints will be created at the same location and scale but with different orientations. Identifying the orientation among the feature points provides stability in matching and invariance to the image rotation.

3.7.1.4 Descriptor Computation

The numerical descriptor is expressed in vector and it is computed from the orientations and magnitudes of the gradients around the location of the localized keypoint. The gradient magnitudes in each keypoint are weighted by a Gaussian weighting function of standard deviation, one-half the width of descriptor window. This would give less emphasis to gradients that are far from the descriptor's center. Thus, the descriptor window's gradient orientations can rotate to its dominant orientation to provide rotational invariance capability to the descriptor, as shown in Figure 3.7.

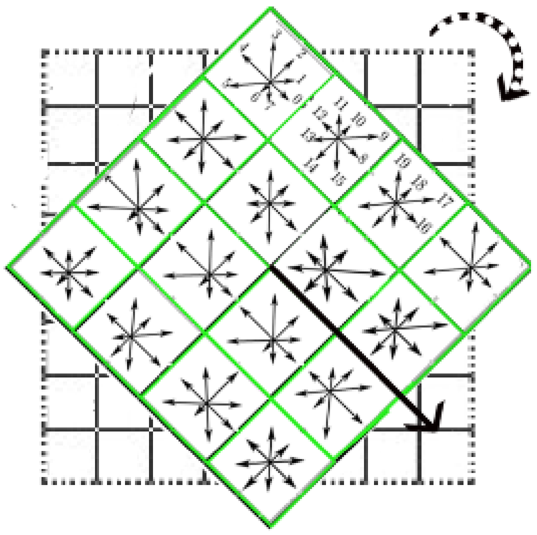


Figure 3.7: Sixteen 8-direction histogram concatenation of descriptor (Younes, Romaniuk and Bittar, 2012)

Eight bin histogram of orientations is computed for the 4x4 sample regions of the descriptor window. The product of the weighted magnitude of the keypoint multiplied with an additional coefficient contributes to the bin of the histogram corresponding to its gradient orientation:

$$p(x, y) = \omega m(x, y)(1 - d) \tag{3.18}$$

where, d is the distance between the central orientation of the histogram bin and gradient orientation. The 8-direction histograms containing the product value of the 16 sample regions are concatenated in a 4 x 4 x 8 = 128-dimensional vector, producing feature points with their unique identification.

3.8 Feature Matching

Upon accomplishing the feature detection and feature description of the base image and query image, the best candidate matches for every keypoints between two images are computed in this phase. In this project, the image matching method named Fast Library for Approximate Nearest Neighbors (FLANN) proposed by Muja and Lowe (2014) is applied. FLANN is preferred over the brute-force searching method as it provides the ability and efficacy of stitching large datasets.

3.8.1 Randomized K-D Tree Algorithm of FLANN

The randomized k-d tree algorithm of the FLANN is used to find the best matches between the base image and query image. The randomized k-d tree framework is worked by building numerous randomized k-d trees and searching the nearest neighbor candidates in parallel. The split dimension of the k-d trees is found from the top N_d dimensions with the highest variance in random manner, where the $N_d = 5$ is suggested by Muja and Lowe. During the randomized k-d forest searching, a single priority queue is preserved across the randomized k-d tree. The priority queue is organized based on the increase of the distance to the decision boundary of each branch in the queue. Hence, the search process will start to explore at the closest leaves from all the trees. During the process, the data points that have been compared to the query points inside a tree are marked to avoid the data points from being re-examined in other trees. The nearest neighbors approximation level is defined based on the maximum leaves number to be explored across all the k-d trees, resulting in the best nearest candidates.

3.8.2 Feature Match Filtering

During image matching between two images, the finest candidate matches for every point are found by searching their nearest neighbor candidates using the randomized k-d tree in the database of feature points from the base image. The nearest neighbor feature correspondence is identified when the feature point with minimum Euclidean distance for the descriptor vector is computed in Subchapter 3.7.1.4. However, a lot of false matches exist among the feature points. This is due to background clutter or the feature points not being detected in database of the base image during the matching. Therefore, a ratio test suggested by Lowe (2004), Lowe’s ratio test is used to identify and remove the features that do not have any good match to the database of the base image. The test works by comparing the distance of the closest invariant descriptor between the two images to that of the second-closest invariant descriptor between the two images. The second-closest candidate match can estimate the density of the feature space's false matches and find the specific feature ambiguity. Figure 3.8 illustrate the example of the match of the descriptor, f_a in query image to the closest descriptor, f_b^1 and second-closest descriptor, f_b^2 in the database of base image.

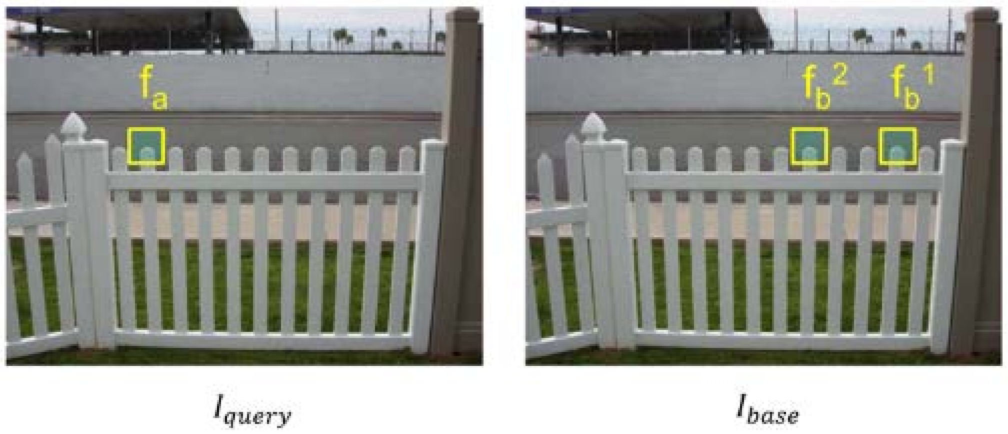


Figure 3.8: Matches between a base image and query image

If the Euclidean distance between the descriptor in the query image and the closest descriptor in the base image is smaller than the product of that to the second closest descriptor in base image with distance ratio factor, R the match will be kept and vice versa. The values of R used to investigate the quality of the match are 0.5, 0.6, 0.7 and 0.8 (to be discussed in Chapter 4). The quality of feature point matches is defined by evaluating the relationship at the following below:

$$d(f_a, f_b^1) < d(f_a, f_b^2) * R \tag{3.19}$$

Where,

$d(f_a, f_b^1)$ = Euclidean distance between the descriptor, f_a in the query image and closest descriptor, f_b^1 in base image

$d(f_a, f_b^2)$ = Euclidean distance between the descriptor, f_a in the query image and second closest descriptor, f_b^1 in base image

R = Constant distance ratio factor

3.9 Robust Homography Estimation

The extracted aerial images are always not well-aligned due to the aerial motion drift and several geometric transformations, including translation, rotation and scaling. Thus, calculating a robust homography transformation matrix is necessary to deal with these transformations to warp the query image to the base image. The transformation is formulated as follows:

$$X' = HX \tag{3.20}$$

where X' represents the warped current image, H is the transformation matrix and X is the current image.

To accomplishing image stitching of the continuous images, a keyframe, $F = X_0$ need to be selected. In this project, the first extracted frame of the footage is initialized as the keyframe. The transformation is achieved by applying the cumulative homography of the current frame with respect to the keyframe. The transformation is calculated by:

$$\begin{cases} X_1' = H_1^0 X_0 \\ X_2' = H_2^0 X_0 \\ \quad\dots \\ X_n' = H_n^0 X_0 \end{cases} \tag{3.21}$$

where,

$$\begin{cases} H_2^0 = H_2^1 * H_1^0 \\ H_3^0 = H_3^2 * H_2^1 * H_1^0 \\ \quad\dots \\ H_n^0 = H_n^{n-1} * H_{n-1}^{n-2} * \dots * H_1^0 \end{cases} \tag{3.22}$$

hence, it can be deduced as:

$$X'_n = \prod_{1}^{n} H_n^{n-1} X_0 = \prod_{1}^{n} H_n^{n-1} F \tag{3.23}$$

The estimated homography transformation is a linear transformation on homogeneous 3-vectors represented by a non-singular 3x3 matrix, $X' = HX$:

$$\begin{bmatrix} wx'_i \\ wy'_i \\ w \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{3.24}$$

The transformation can be expressed in inhomogenous form as:

$$\begin{aligned} x'_i &= \frac{x_i}{w} = \frac{h_{00}x_i + h_{01}y_i + h_{02}}{h_{20}x_i + h_{21}y_i + h_{22}} \\ y'_i &= \frac{y_i}{w} = \frac{h_{10}x_i + h_{11}y_i + h_{12}}{h_{20}x_i + h_{21}y_i + h_{22}} \end{aligned} \tag{3.25}$$

Each feature correspondence produces two equations for the H elements:

$$\begin{aligned} x'_i(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{00}x_i + h_{01}y_i + h_{02} \\ y'_i(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{10}x_i + h_{11}y_i + h_{12} \end{aligned} \tag{3.26}$$

During the homography estimation, only four feature correspondences out of all the feature correspondences are required. However, a question arises for which four feature correspondence should be selected. Hence, a robust estimation algorithm, named Random Sample Consensus (RANSAC) in OpenCV, is utilized to find the best four feature correspondence from the database and remove the matches' outliers. The computed homography matrix is further refined by using the Levenberg-Marquardt technique on computed inliers to mitigate the reprojection error.

3.9.1 Random Sample Consensus (RANSAC)

Random Sample Consensus (RANSAC) algorithm is robust to discard large proportion of outliers in the database. Figure 3.9 illustrates the way of categorizing inliers by the algorithm. In this framework, two points are randomly selected and a line is virtually drawn between them (red point with blue line in Figure 3.9). The distance of each point to the line is calculated and a distance threshold is defined. The points that lie within the distance threshold are constituted as inliers (green point in Figure 3.9). Besides, the black points in Figure 3.9 are defined as outliers. The random selection of two points is repeated until the line with most of the points is deemed the robust fit.

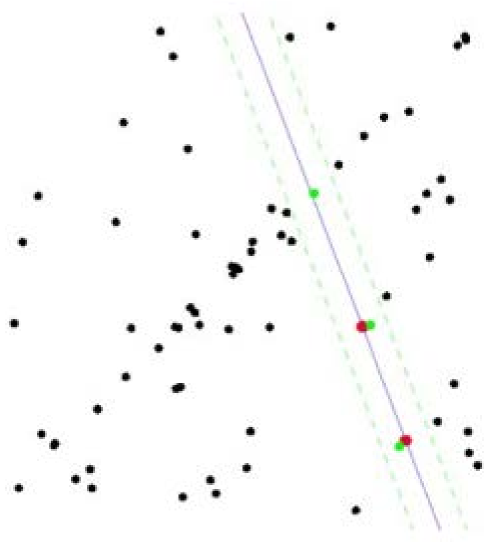


Figure 3.9: Example of categorizing inliers (Baid, 2015)

The algorithm works in this way to compute the homography:

1. Any four points required to calculate the homography matrix are randomly selected.
2. The homography matrix is computed by using Direct Linear Transformation (DLT).
3. The number of inliers is calculated.
4. The ratio of inliers to the total number of points in the dataset is calculated.

5. If the ratio is greater than the predefined reprojection threshold, T, the homography is re-computed using all the determined inliers and terminate.
6. Else, step 1 to 5 is repeated (for N iterations)

3.10 Image Warping and Black Edges Removal

Upon the robust homography estimation, all query images in the dataset are warped in sequential order with respect to the keyframe by utilizing their calculated homography matrix to generate a stitched image. Nonetheless, the result usually contains black edges around it. The black edges of the image are necessary to remove in order to obtain the image with stitched view only. So, the region of interest (ROI) of the stitched image needs to be determined in the first place. Then, the black edges are removed by cropping the black region of the stitched image automatically.

3.11 Project Planning and Resource Allocation

Figure 3.10 shows the Gantt chart planned for the project based on the project workflow in Figure 3.1.

Project Planning			DURATION (Days)
START DATE	END DATE	DESCRIPTION	
1/18/21	1/22/21	Aerial footage acquisition and image extraction	4
1/22/21	1/29/21	Evaluate type of feature representation	7
1/29/21	2/5/21	Evaluate type of image matching	6
2/5/21	2/6/21	Feature Representation Selection	1
2/6/21	2/7/21	Image Matching Selection	1
2/7/21	2/14/21	Develop Program Flowchart	7
2/14/21	3/22/21	Image Stitching Program Coding	38
3/4/21	3/29/21	Program Testing	25
3/22/21	3/29/21	Program Improvement	7
2/1/21	4/19/21	Report Writing	78

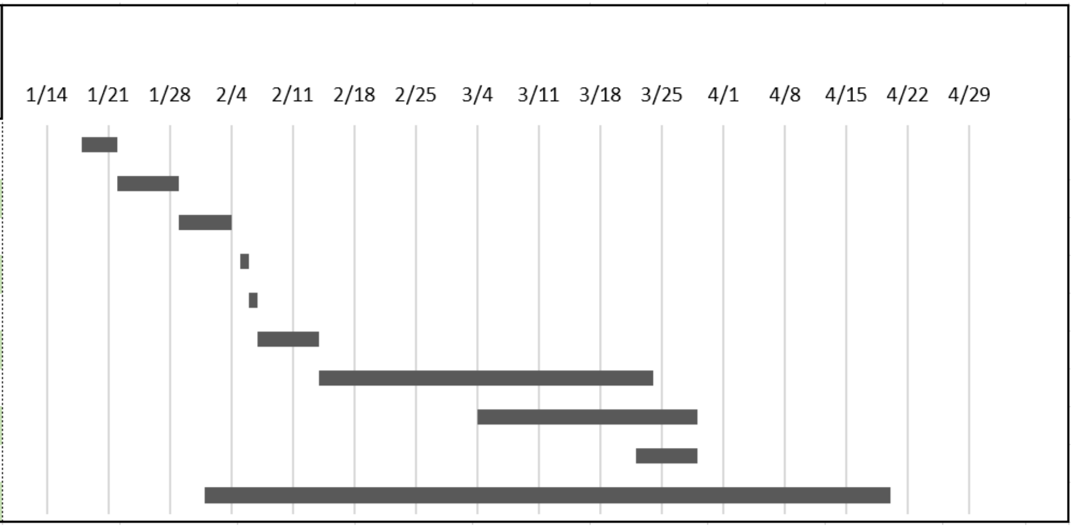


Figure 3.10: Gantt Chart of Task List Project Planning

The image stitching program was scripted in Anaconda Spyder (version 4.1.4) using Python (3.7.6) and ran on Central Processing Unit (CPU). In this project, the python binding of the OpenCV library is used. It contains numerous powerful computer vision algorithms such as the SIFT feature detector and feature descriptor, FLANN feature macher, RANSAC, numpy, etc.

3.12 Summary of Methodology

This chapter has talked about the techniques used in the image stitching program. Frame extraction from aerial footage is executed first, followed by camera calibration and image pre-processing. Then, SIFT is used as feature representation to detect and describe the feature points in the images. Afterward, the randomized k-d tree of FLANN matcher is employed to find the feature point pairs between the images. The Lowe’s ratio test is implemented to remove the mismatch point pairs. To calculate the homography matrix, RANSAC is utilized. Finally, the image warping and black edges removal is executed consecutively to produce a stitched image. The results are presented in Chapter 4.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Introduction

As mentioned in Chapter 3, feature match rate and inlier ratio are calculated to measure the performance of the methods. The feature match rate of the system is defined as follows:

$$Feature\ match\ rate = \frac{M}{(Fp1\ + Fp2)/2} \tag{4.1}$$

where, M is the number of feature matches, Fp1 is the number of feature points extracted from the base image and Fp2 is the number of feature points extracted from the query image.

The greater the feature match rate, the greater the feature points between two images being matched.

The inlier ratio of the system is denoted as follows:

$$inlier\ ratio = \frac{number\ of\ inlier}{number\ of\ feature\ matches} \tag{4.2}$$

The greater the inlier ratio, the greater the inliers found in the matches.

4.2 Frame Extraction and Camera Calibration

For this dataset, one frame of every fifty frames is extracted to obtain adequate overlapped regions with each other. Therefore, twenty-four images are extracted from the Orchard aerial footage dataset with L-shape flight pattern.

Figure 4.1 shows the example of the extracted frame with lens distortion (fish-eye effect) and the calibrated frame.

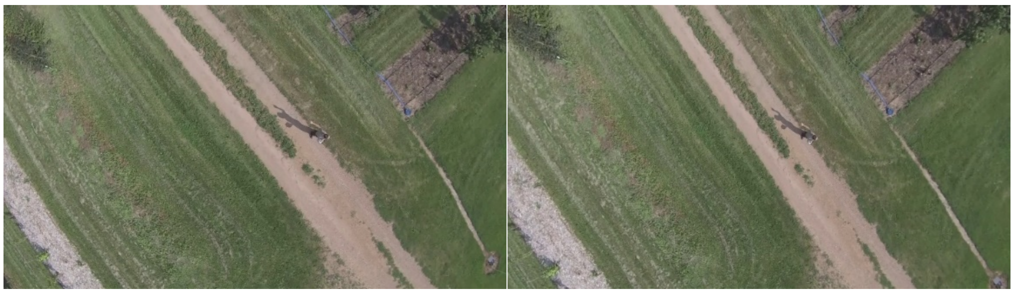


Figure 4.1: Example of an extracted frame with fish-eye effect (left) and calibrated frame (right)

In this project, the radial distortion parameters and tangential distortion parameters are $(-0.30409, 0.10001, 0)$ and $(-0.00065, -0.00156)$ respectively. Their combination, which is distortion coefficients $(-0.30409, 0.10001, -0.00065, -0.00156, 0)$ is used. Meanwhile, the camera intrinsic matrix is $\begin{bmatrix} 1237.36828 & 0 & 954.79824 \\ 0 & 1275.40537 & 560.46217 \\ 0 & 0 & 1 \end{bmatrix}$. These two crucial calibration parameters successfully optimize the lens distortion of the images.

4.3 The Pre-processing Phase

In the pre-processing stage, the input images are resized and converted to grayscale format. Then, the Gaussian blur is applied to the grayscaled images to discard the Gaussian noises.

4.3.1 Image Resize and Grayscale Conversion

Every input images are converted into grayscaled format from the RGB complex format to reduce the dimensionality of the color information of images. Therefore, the computation burden of the algorithm is reduced. Figure 4.2 shows an example of the conversion of the RGB image into grayscaled image.

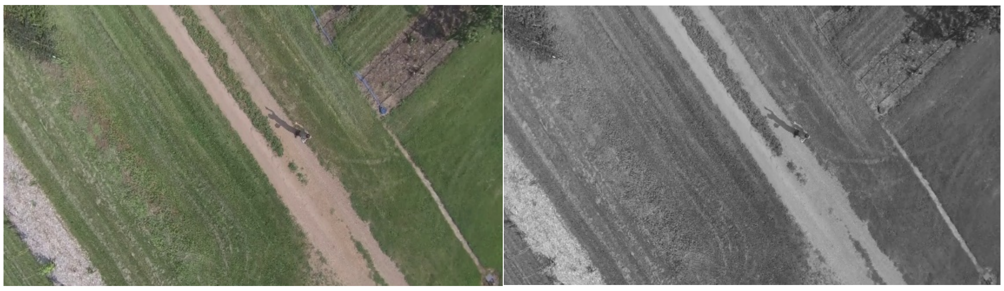


Figure 4.2: Grayscale conversion of aerial image

4.3.2 Gaussian Noise Removal

As stated in Chapter 3, the Gaussian blur act as the “low-pass filter” to remove detail and noise, leaving the majority of the image intact. Thus, the image features information are still preserved. The OpenCV’s built-in Gaussian smoothing function (Bradski and Kaehler, 2009) is utilized to remove the Gaussian noise. The example of the effect of Gaussian smoothing on the aerial image is shown in Figure 4.3.

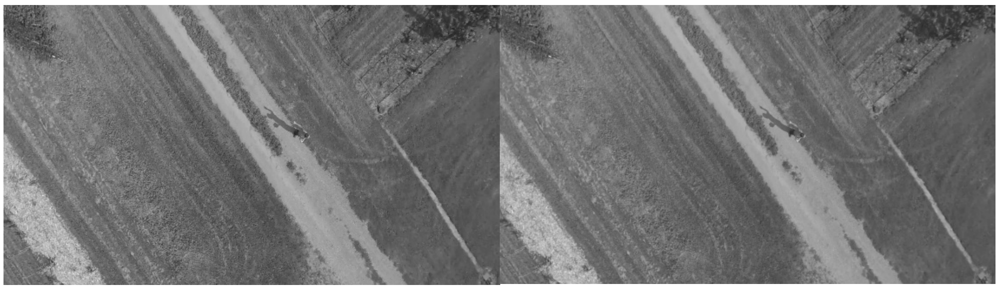


Figure 4.3: Effect of gaussian blur (left = before, right = after)

Table 4.1 shows the study of the effect of gaussian blur on average feature match rate, average inlier ratio and time taken. Figure 4.4 shows the feature match rate graph between image stitching with Gaussian blur and without Gaussian blur. Meanwhile, the graph of the inlier ratio between image stitching with Gaussian blur and without Gaussian blur is shown in Figure 4.5.

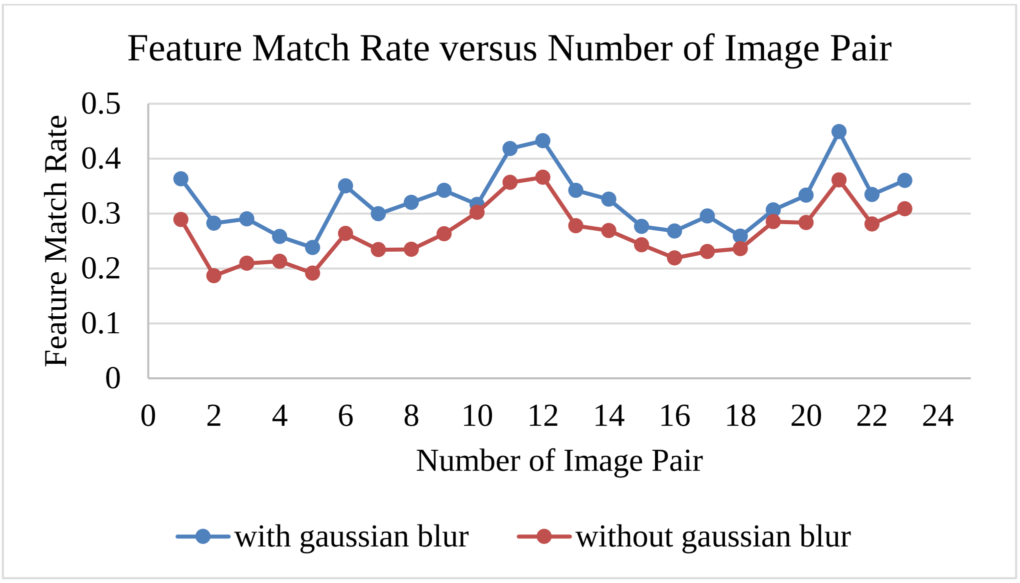


Figure 4.4: Graph of feature match rate between image stitching with Gaussian blur and without Gaussian blur

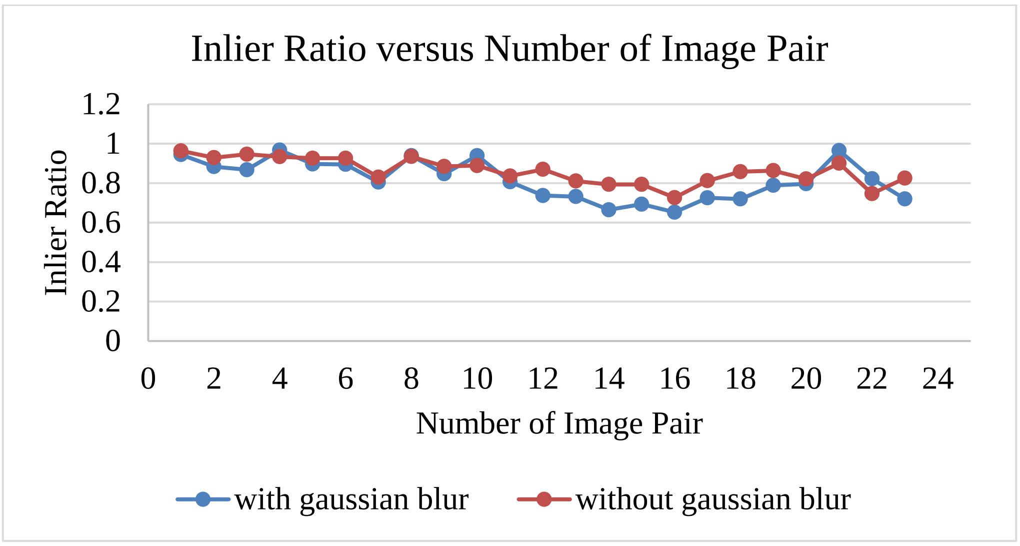


Figure 4.5: Graph of inlier ratio between image stitching with Gaussian blur and without Gaussian blur

Table 4.1: Study of the effect of Gaussian blur on average feature match rate, average inlier ratio and time taken

Database	Orchard (without Gaussian blur)	Orchard (with Gaussian blur)
Feature representation	SIFT	SIFT
Feature matcher	FLANN (randomized k-d tree)	FLANN (randomized k-d tree)
Lowe's ratio test	Yes (ratio = 0.8)	Yes (ratio = 0.8)
Outlier removal	RANSAC	RANSAC
Average feature match rate (23 image pair)	0.265436	0.32445
Average inlier ratio (23 image pair)	0.861916	0.817921
Time taken	64.109s	60.875s

Figure 4.6 shows the stitched image of orchard without Gaussian blur and with Gaussian blur.

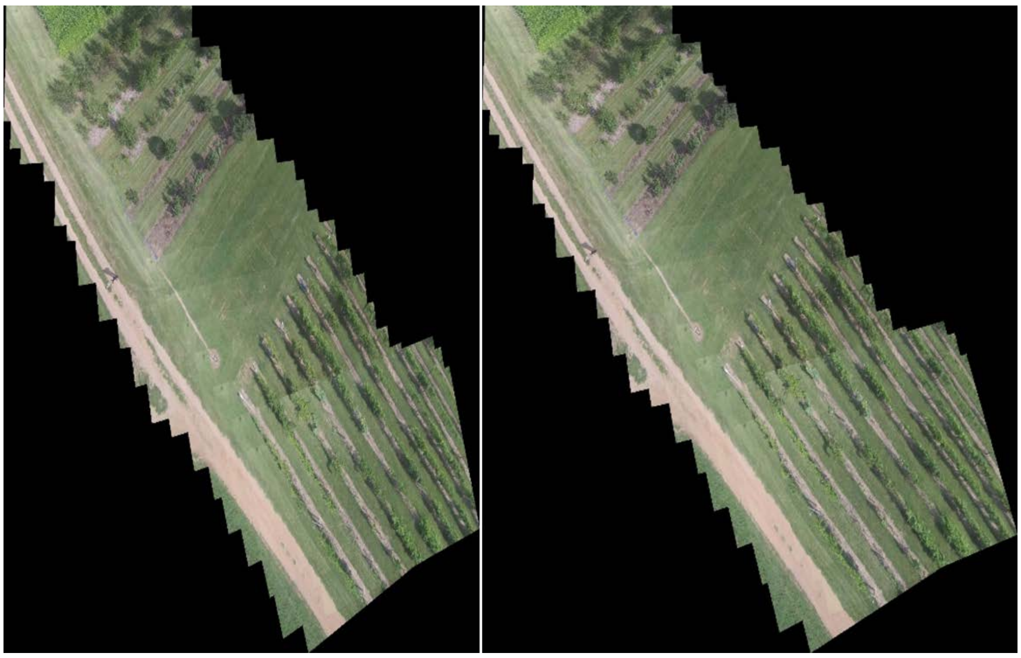


Figure 4.6: Stitched image of orchard without Gaussian blur (left) and with Gaussian blur (right)

The average feature match rate and average inlier ratio in Table 4.1 are calculated based on Figure 4.4 and Figure 4.5 respectively. Based on the empirical result shown in Table 4.1, when the Gaussian blur is used, the average feature rate rises from 0.265436 to 0.32445, the average inlier ratio reduces from 0.861916 to 0.817921, time taken reduces from 74.109s to 60.875s. Although the average inlier ratio is reduced when the gaussian blur is applied, the alignment between the stitched image frames remains excellent with respect to the test without Gaussian blur as shown in Figure 4.6. Therefore, the Gaussian blur is adopted in the subsequent benchmark tests to investigate the performance of image stitching method.

4.4 Feature Representation

For the feature detection, 3-octave layers with five scale each layer are constructed to compute the scale-space local extrema in this project. The sigma of Gaussian smoothing, σ_0 is set as 1.6, applying to the octaves pyramid.

Afterward, the contrast threshold is set to 0.04 in localizing the keypoints of the image. The low-contrasted keypoints are removed to avoid the existence of low contrast unstable extrema if the local extremum value of the scale-space function is less than the predefined contrast threshold, 0.04.

The edge threshold is set to 10 to reject the keypoints that being detected along the edges of the small curvature.

4.4.1 Comparison Results of Sparse Feature-based and Binary-based Feature Representation

For feature representation, the comparison between sparse feature-based feature representation and binary-based feature representation is obtained. Since the ORB binary-based method and SIFT sparse feature-based method both are invariant against rotation and scaling changes made by the aerial motion. The binary-based feature representation method proposed by De Lima and Martinez-Carranza is used to compare with the proposed method. The authors implemented an aerial image stitching method using ORB feature representation and locality-sensitive hashing (LSH) of FLANN matcher. Figure 4.7 and Figure 4.8 show the graph of the feature match rate and the comparison of the inlier ratio between the method proposed in this project and the method proposed by De Lima and Martinez-Carranza.

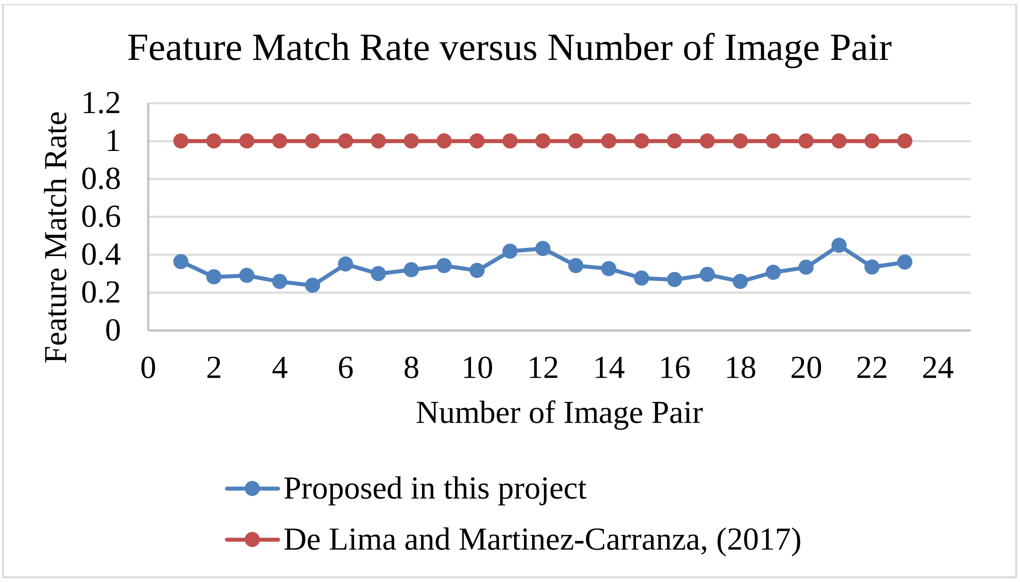


Figure 4.7: Graph of feature match rate between SIFT, randomized k-d tree and ORB, locality-sensitive hashing

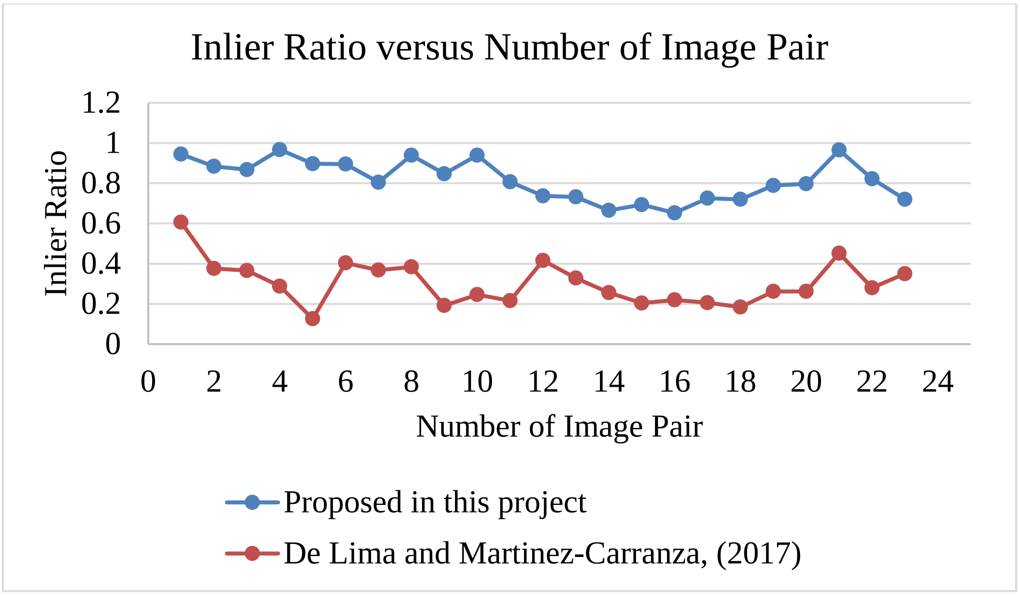


Figure 4.8: Graph of inlier ratio between SIFT, randomized k-d tree and ORB, locality-sensitive hashing

Table 4.2 shows the comparison results for feature representation with FLANN matching method on the Orchard dataset. Moreover, Figure 4.9 shows the generated stitched image using SIFT, FLANN (randomized k-d tree) and

ORB, FLANN (locality-sensitive hashing). Furthermore, Figure 4.9 shows the stitched image using SIFT feature representation and ORB feature representation respectively.

Table 4.2: Comparison results of feature representation with FLANN matching method on the orchard dataset

Column	1 (Proposed in this project)	2 (De Lima and Martinez-Carranza, 2017)
Feature representation	SIFT	ORB
Feature matcher	FLANN (randomized k-d tree)	FLANN (locality-sensitive hashing)
Lowe's ratio test	Yes (ratio, 0.8)	No
Outlier Removal	RANSAC	RANSAC
Average feature match rate (23 image pair)	0.324554	1.0
Average inlier ratio (23 image pair)	0.805587	0.304

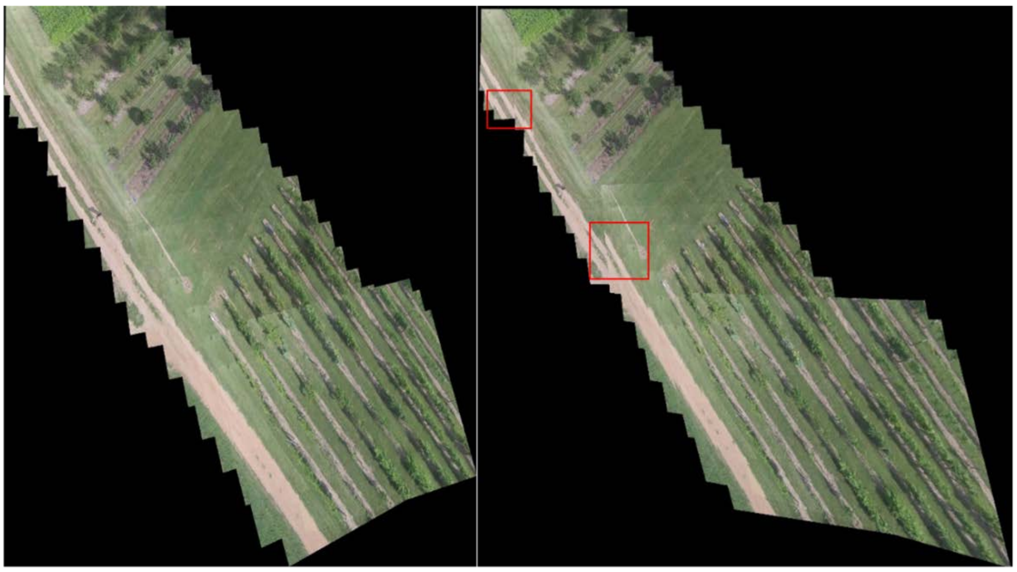


Figure 4.9: Stitched image using SIFT, FLANN (randomized k-d tree) (left) and ORB, FLANN (locality-sensitive hashing) (right). The red bounding box shows the misaligned frame.

By referring the Table 4.2, the average feature match rate using SIFT-based feature representation is weaker than that using ORB-based feature representation. Whereas the average inlier ratio using SIFT-based feature representation is more significant than that using ORB-based feature representation. Even though the average feature match of ORB-based feature representation is outperforming compare to that of SIFT-based feature representation, some stitched frames do not align well during the image warping process as shown in Figure 4.9. This is due to the average amount of inlier generated per match of the ORB-based method, which is relatively poor, 152 compared to 171 produced by SIFT-based feature representation. Hence, poor homography estimation is obtained to warp the frame onto the base image, resulting in poorly aligned frames in the stitched image. It can deduce that SIFT performs better than ORB in this project.

4.5 Feature Matching

The OpenCV's randomized k-d tree in FLANN matcher (Bradski and Kaehler, 2009) is used to pair the extracted feature points between the query image and base image. In the randomized k-d tree, the number of trees is set as 5 in constructing the forest. Each tree is established independently, and each tree chooses randomly among the five split dimensions at each level.

In OpenCV, there is another widely used feature matcher, brute-force searching matcher. Therefore, the performance of the brute-force searching matcher and the randomized k-d tree of FLANN matcher is investigated and discussed.

4.5.1 Comparison Results of FLANN Matcher and Brute-Force Matcher in Feature Matching

The feature match rate and inlier ratio between the FLANN matcher and brute-force matcher are compared. Figure 4.10 and Figure 4.11 show the graph of feature match rate and inlier ratio between FLANN (randomized k-d tree) and brute-force matcher.

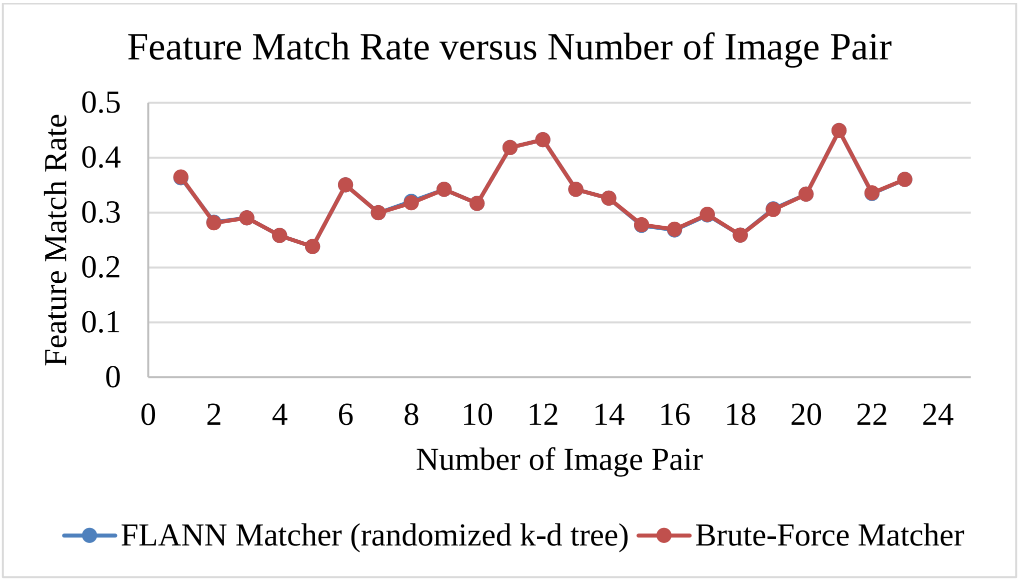


Figure 4.10: Graph of feature match rate between FLANN (randomized k-d tree) and brute-force matcher

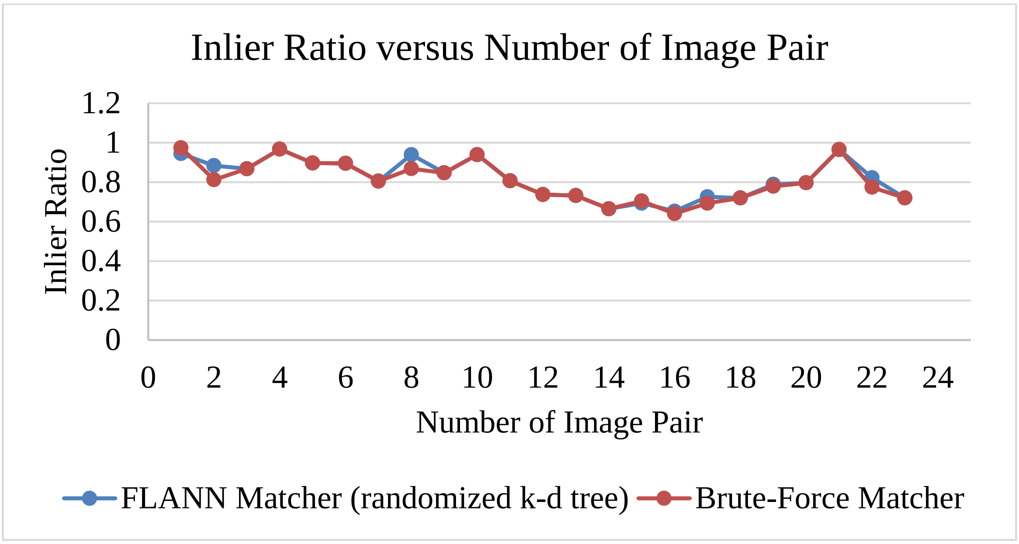


Figure 4.11: Graph of inlier ratio between FLANN (randomized k-d tree) and brute-force matcher

Table 4.3 shows the comparison result of SIFT with FLANN and Brute-Force matcher. Figure 4.12 shows the stitched image using SIFT with FLANN and Brute-Force matcher respectively.

Table 4.3: Comparison results of SIFT with FLANN and Brute-Force matcher

Column	1	2
Feature representation	SIFT	SIFT
Feature matcher	FLANN (randomized k-d tree)	Brute-Force
Lowe’s ratio test	Yes (ratio, 0.8)	Yes (ratio, 0.8)
Outlier Removal	RANSAC	RANSAC
Average feature match rate (23 image pair)	0.324554	0.324475
Average inlier ratio (23 image pair)	0.805587	0.809046

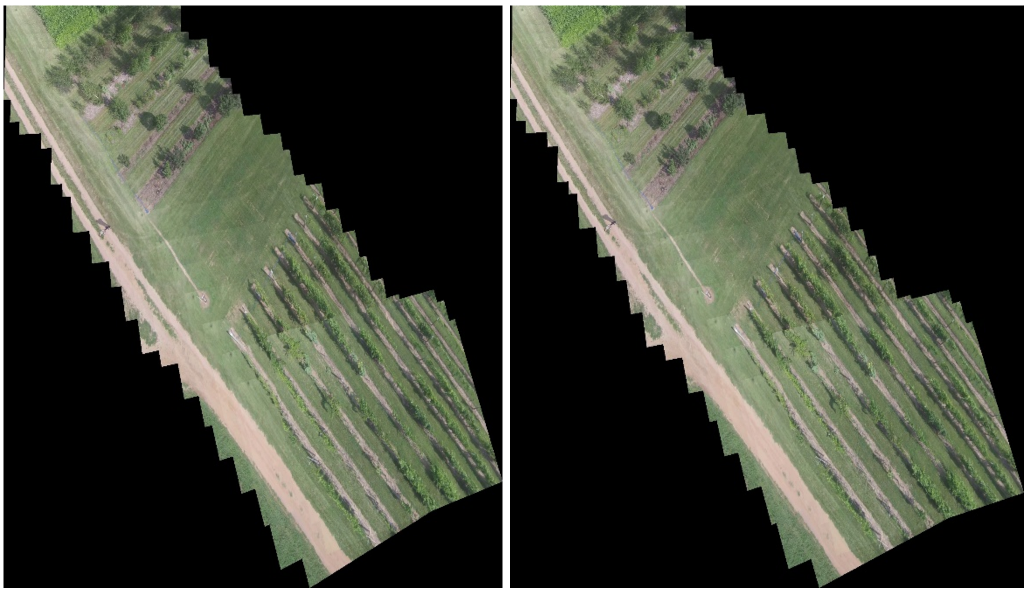


Figure 4.12: Stitched image using SIFT, FLANN (randomized k-d tree) (left) and SIFT, Brute-Force (right)

By comparing the average feature match rate and inlier ratio in Table 4.3, the average feature match rate and the inlier ratio of FLANN matcher are slightly more significant than those of the brute-force matcher. Moreover, the stitched frames' alignment in the stitched image obtained by using brute-force matcher is equally great compared to that by using FLANN matcher as shown in Figure 4.12. According to Li *et al.*, (2014), the authors stated that tree-based methods, such as k-d tree, are relatively more efficient in the application involving large image dataset retrieval (aerial images) than brute-force searching method to stitch images. Hence, the FLANN matcher is preferred as it may provide repeatability and efficacy on stitching large aerial image dataset (>1000) in future work.

4.5.2 Comparison Results of LR, GMS and LR-GMS in Feature Match Filtering

The generated feature match pairs usually contain a lot of false matches. Thus, match pairs filtering is necessarily applied to mitigate the number of false matches and reduce the chance of resulting poor homography. In this project, the effect of using Lowe's ratio test (LR), Grid-Based Motion Statistic (GMS) (Bian *et al.*, 2020), and the combination of LR and GMS on feature match rate, inlier ratio is studied as well as the efficacy of stitching aerial image is observed. The feature match rate graph and the inlier ratio among LR, GMS, and LR-GMS are shown in Figure 4.13 and Figure 4.14 respectively.

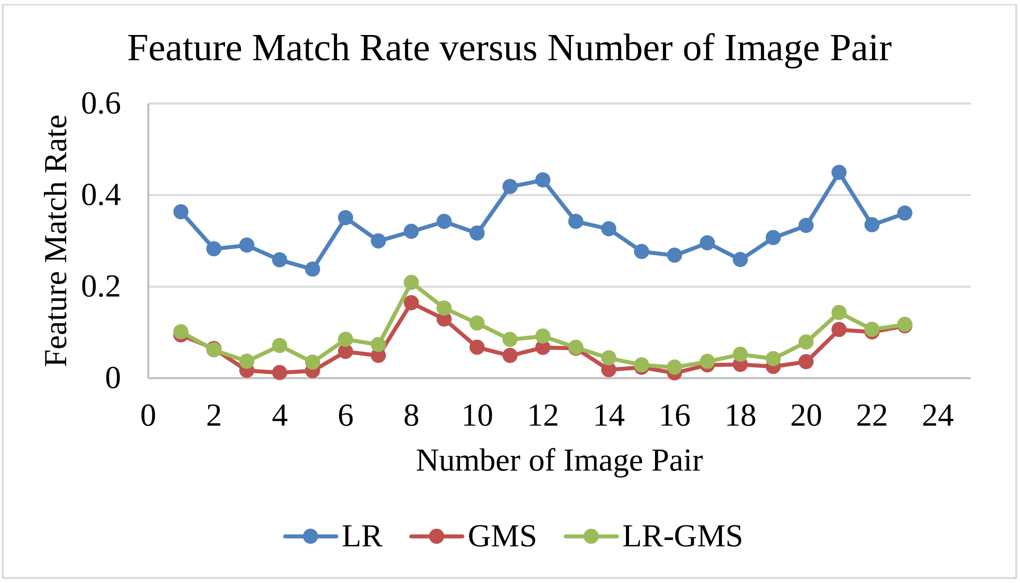


Figure 4.13: Graph of feature match rate among LR, GMS and LR-GMS

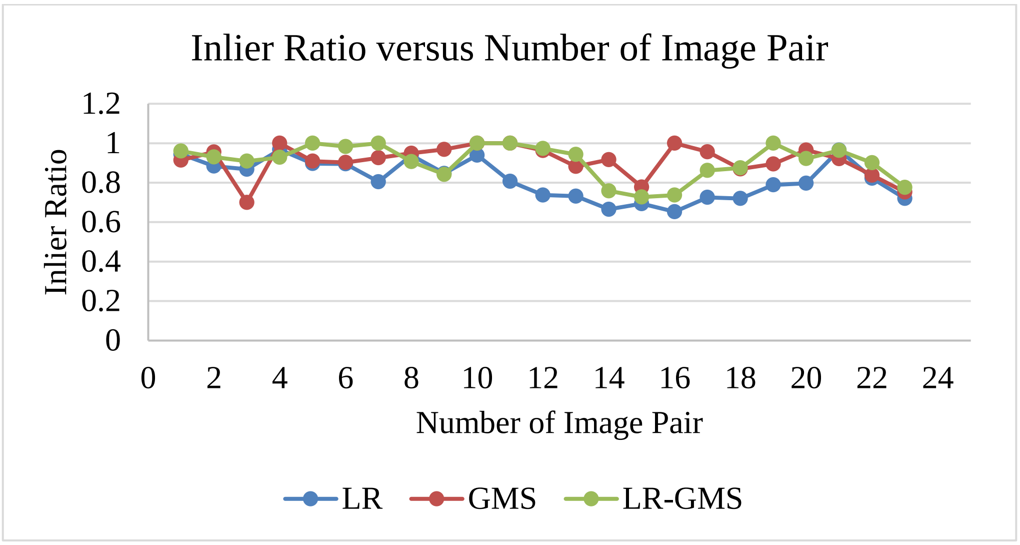


Figure 4.14: Graph of inlier ratio among LR, GMS and LR-GMS

Table 4.4 shows the comparison results of LR, GMS and LR-GMS. Figure 4.15 shows the stitching image using LR, GMS and LR-GMS. The graph of the average inlier count among LR, GMS and LR-GMS is shown in Figure 4.16.

Table 4.4: Comparison results of LR, GMS and LR-GMS

Column	1	2	3
Feature representation	SIFT	SIFT	SIFT
Feature matcher	FLANN (randomized k-d tree)	FLANN (randomized k-d tree)	FLANN (randomized k-d tree)
Lowe's ratio test	Yes (ratio, 0.8)	No	Yes (ratio, 0.8)
GMS	No	Yes	Yes
Outlier Removal	RANSAC	RANSAC	RANSAC
Average feature match rate (23 image pair)	0.32445	0.080801	0.058421
Average inlier ratio (23 image pair)	0.817921	0.90861	0.911442

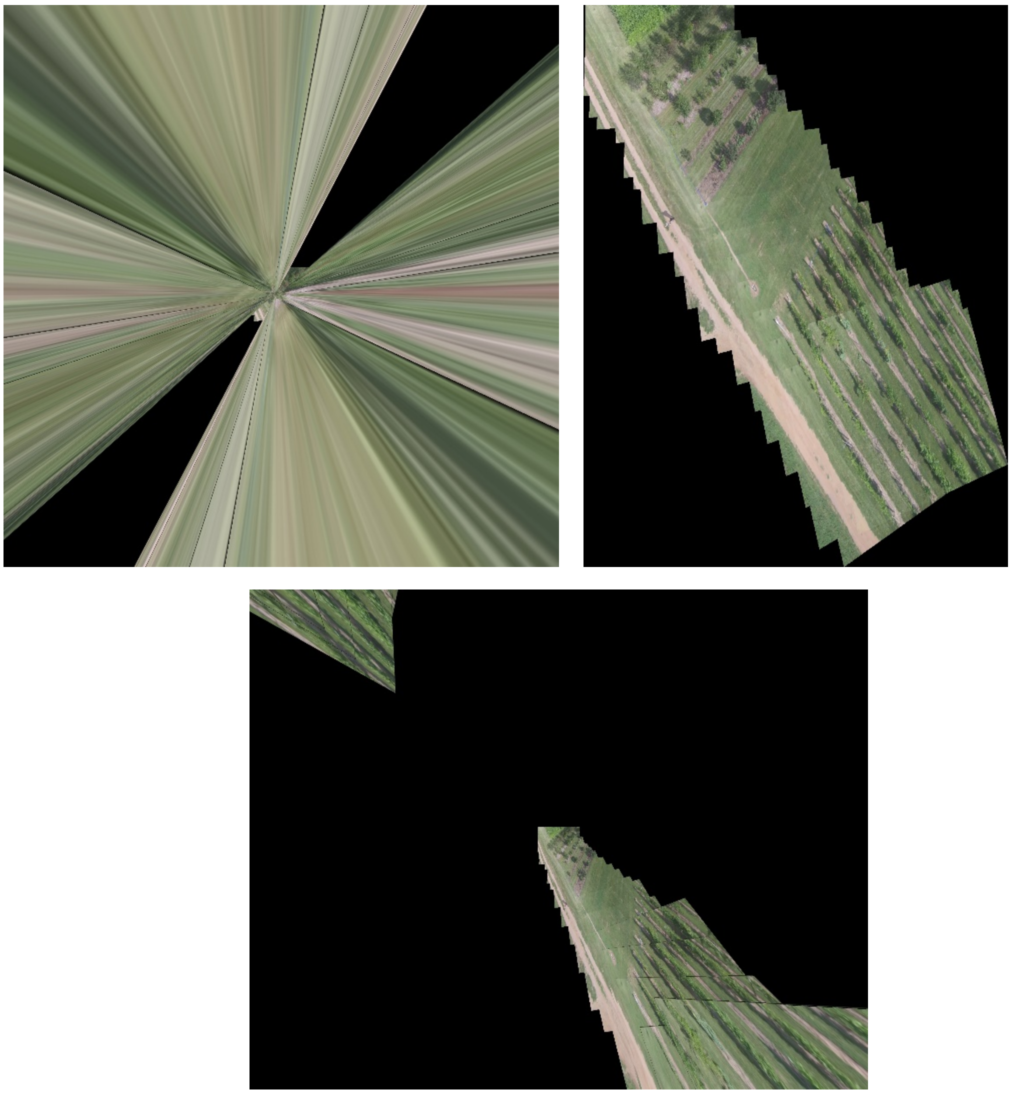


Figure 4.15: Stitched image using GMS (top-left), LR (top-right) and LR-GMS (bottom)

By comparing the average feature match rate in Table 4.4, it is found that the average feature match rate of GMS is the highest among them (0.080801), followed by LR-GMS (0.58421) and then LR (0.32445). Nonetheless, by comparing the average inlier ratio, the average inlier ratio of LR-GMS is the highest (0.911442), followed by GMS (0.90861) and then LR (0.817921). Despite the fact that the average inlier ratio of LR-GMS and GMS are more significant than that of LR, both methods are failed to stitch the aerial images, as shown in Figure 4.15. This is because the GMS failed to strike an optimal balance between keeping good matches and discarding bad matches, resulting in overly removing the good matches. Moreover, the RANSAC further removes the outlier in the matches and induces a low count of inlier, where the average inlier count per image pair is less than 100, as shown in Figure 4.16. The inlier number is crucial in estimating a good homography to warp the query image onto the base image. Based on Figure 4.16, it is shown that the inlier count using GMS and LR-GMS are much lower than that of the LR, which cause the failure in stitching aerial images. Therefore, GMS is not suitable to be implemented in this orchard dataset.

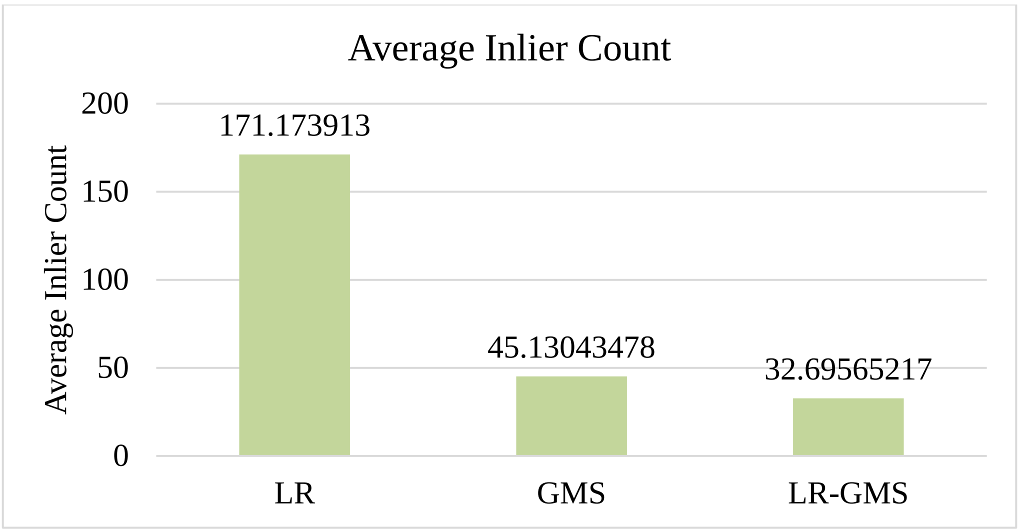


Figure 4.16: Graph of average inlier count among LR, GMS and LR-GMS

4.5.2.1 The Effect of Lowe's Ratio on Feature Match Rate and Inlier Ratio

Figure 4.17 and Figure 4.18 show the graph of feature match rate and inlier ratio over the Lowe's Ratio. The study of the effect of various ratios on the feature match rate and inlier ratio is shown in Table 4.5. Figure 4.19 shows the average inlier count over Lowe's ratios.

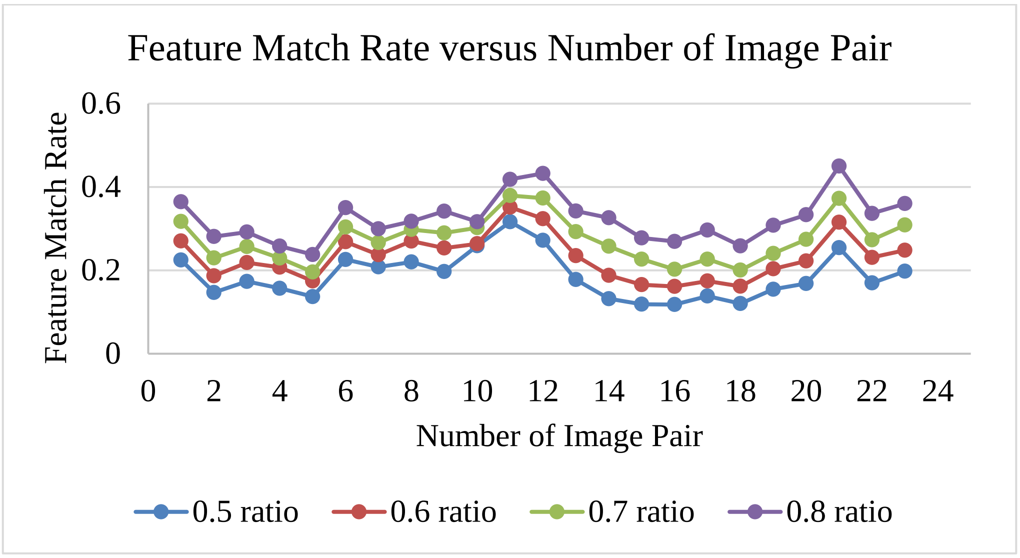


Figure 4.17: Graph of feature match rate over the Lowe's ratios

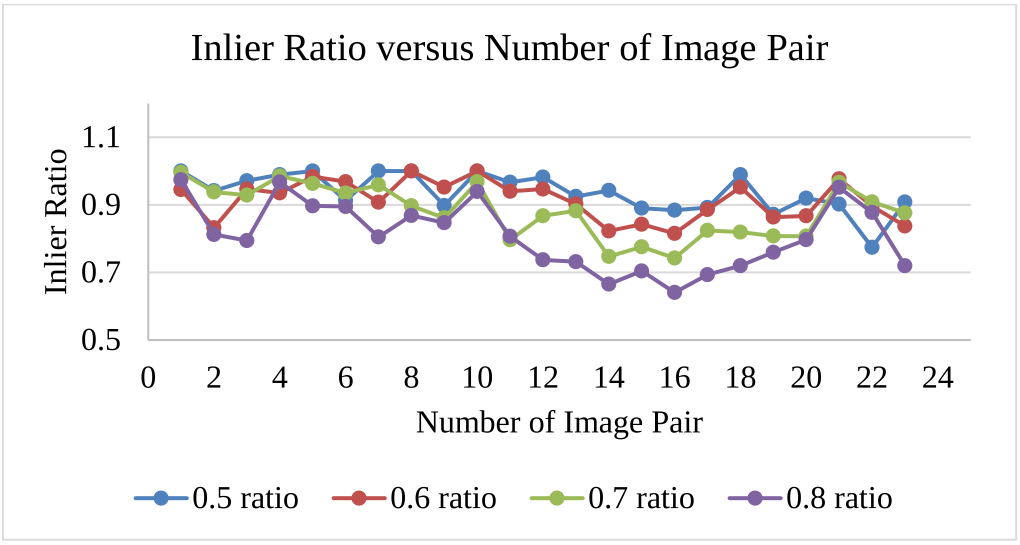


Figure 4.18: Graph of inlier ratio over the Lowe's ratios

Table 4.5: Study of the effect of various ratios on the feature match rate and inlier ratio

Column	1	2	3	4
Feature representation	SIFT	SIFT	SIFT	SIFT
Feature matcher	FLANN (randomized k-d tree)	FLANN (randomized k-d tree)	FLANN (randomized k-d tree)	FLANN (randomized k-d tree)
Lowe's ratio test	Yes (ratio, 0.5)	Yes (ratio, 0.6)	Yes (ratio, 0.7)	Yes (ratio, 0.8)
GMS	No	No	No	No
Outlier Removal	RANSAC	RANSAC	RANSAC	RANSAC
Average feature match rate (23 image pair)	0.186405	0.23189	0.274729	0.32445
Average inlier ratio (23 image pair)	0.937371	0.9139	0.880527	0.817921

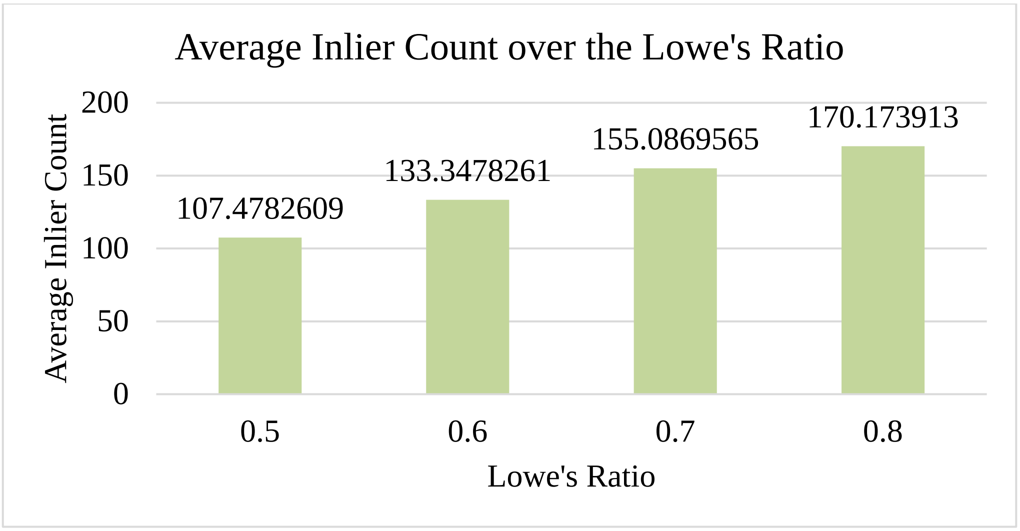


Figure 4.19: Graph of average inlier count over the Lowe's ratio

Based on Figure 4.17 and Table 4.5, it is found that the greater the Lowe’s ratio, the greater the feature match rate. This is because fewer nearest neighbor descriptor is discarded when the Lowe’s ratio is high. By comparing the average inlier ratio recorded in Table 4.5 and Figure 4.18, the inlier ratio is the highest when the Lowe’s ratio is set to 0.5, followed by 0.6, 0.7 and 0.8. However, a better-stitched view is observed when the average inlier count is more than 160 empirically. The stitched images generated by using 0.5, 0.6 and 0.7 Lowe’s ratio induce slight skewness as shown in Figure 4.20. Hence, this proves that Lowe’s ratio at 0.8 is optimal to generate a well-aligned stitched image.

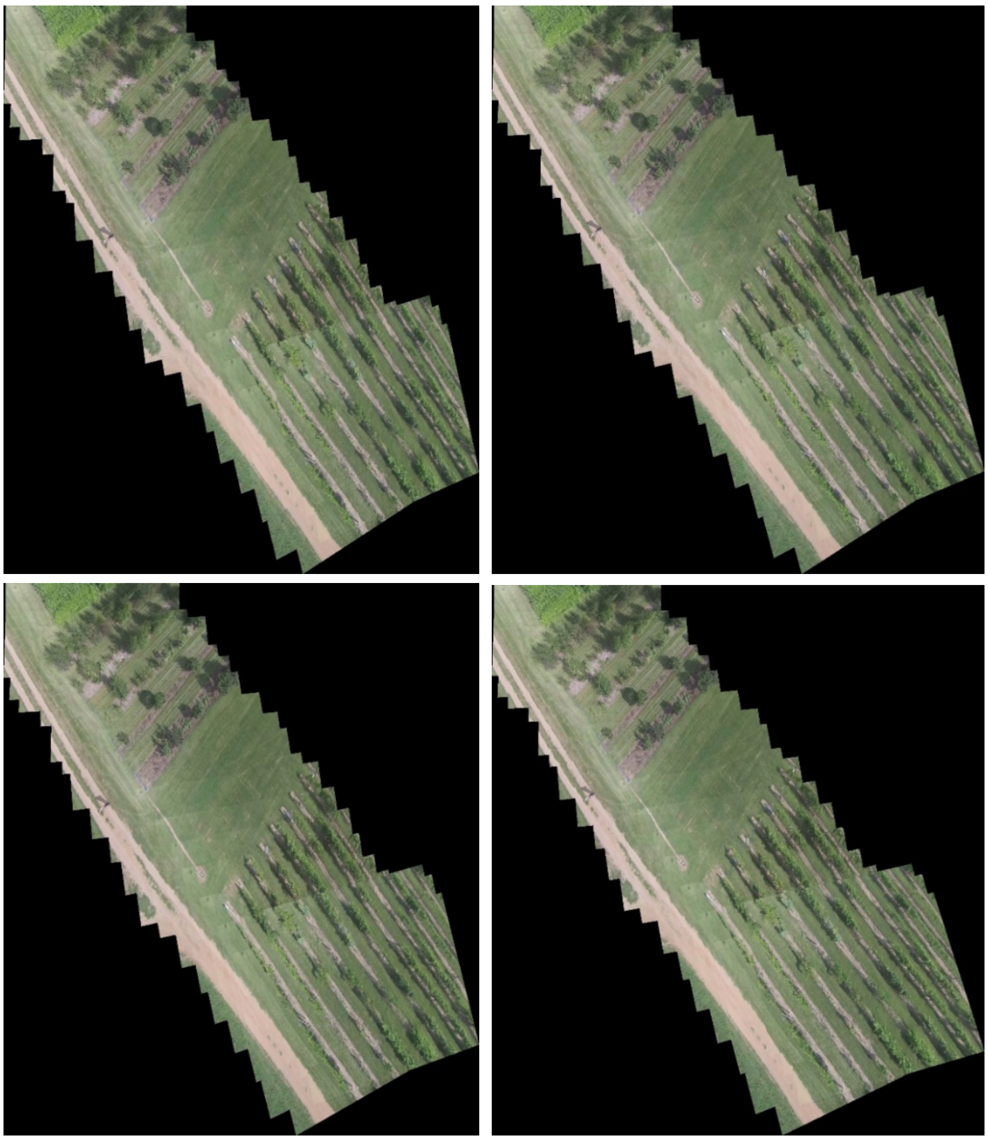


Figure 4.20: Stitched image using 0.5 ratio (top-left), 0.6 ratio (top-right), 0.7 (bottom-left) and 0.8 (bottom-right)

4.6 Robust Homography Estimation using RANSAC

During the homography calculation using RANSAC, the RANSAC reprojection threshold is set to 5. It is a maximum allowed reprojection error to treat the match as an inlier. Any computed values greater than five are treated as outliers. The relationship is express as following below:

$$C(dst) - C[H * C(ori)] < T \tag{4.3}$$

Where,
$C(dst)$ = The coordinates of the points in the target plane
$C[H * C(ori)]$ = The converted homogenous points from the points of the original plane
T = RANSAC reprojection error.

Furthermore, the iteration of the RANSAC algorithm, N is set to 2000, which is the maximum value that the OpenCV library allows. The RANSAC algorithm iterates 2000 times to discard any match exceeding the RANSAC reprojection error and use the inliers to estimate the homography matrix. The greater the iteration, the greater the number of outliers or noises being removed.

Figure 4.21 shows the stitched image without using RANSAC algorithm and using RANSAC algorithm.

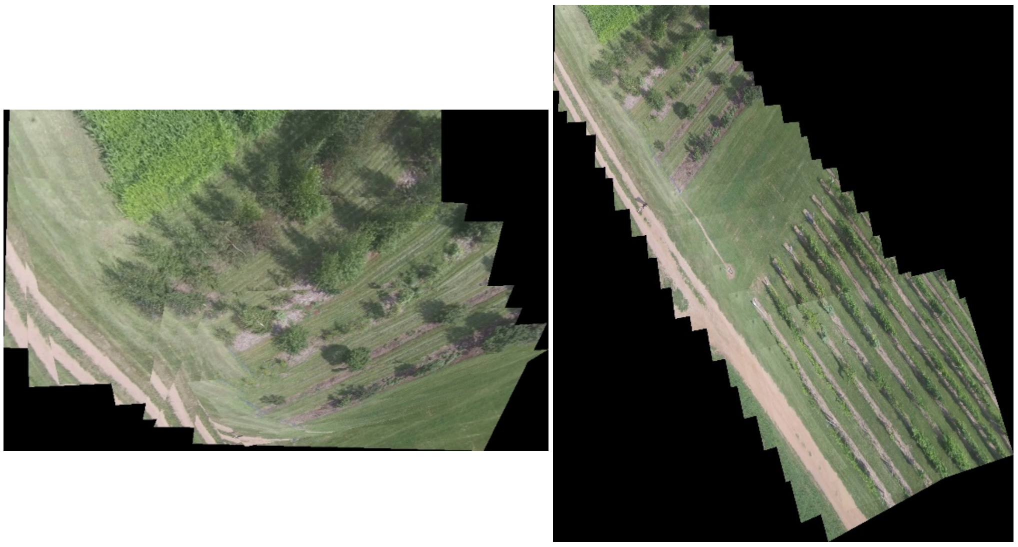


Figure 4.21: Stitched image without using RANSAC algorithm(left) and using RANSAC algorithm(right)

According to Figure 4.21, it is found that the images failed to stitch when bypassing the RANSAC algorithm. This is due to the noises present in the images. The noises are taken as correct matches to estimate the homography matrix, resulting in bad homography estimation and causing the misalignment between the frames. Therefore, RANSAC is necessary to be implemented to remove the outliers embedded in the matches.

4.7 Image Warping and Black Edges Removal

Figure 4.22 shows the example of sequential order image wrapping. The keyframe is warped in the middle of the image with a black background. Then, the adjacent images are warped onto the keyframe using the computed accumulative homography matrix in sequential order

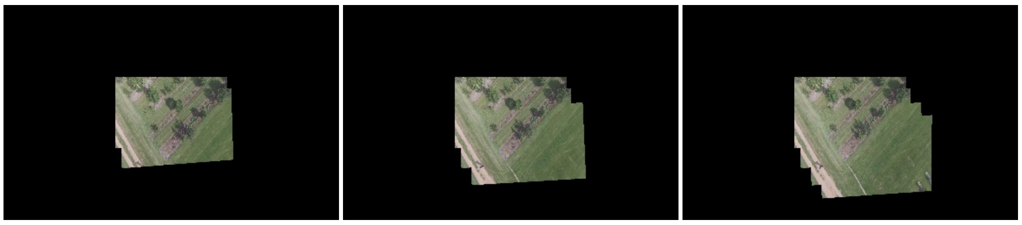


Figure 4.22: Example of sequential order image wrapping

Figure 4.23 shows the black edge removal of the stitched image. The algorithm detects the region of interest (ROI) of the image and the black region is automatically being cropped out.

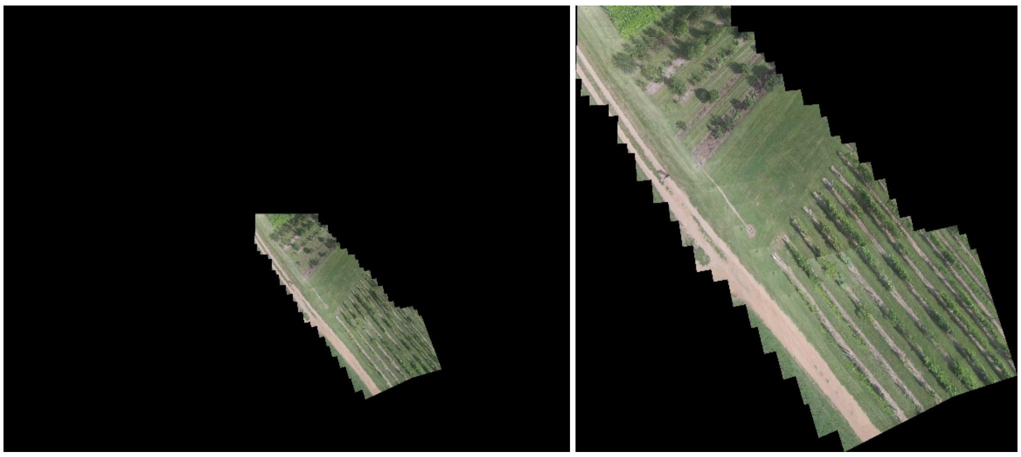


Figure 4.23: Black edge removal of the stitched image

4.8 Dataset with Lawnmower Flight Pattern

Figure 4.24 shows the stitched image of dataset with lawnmover flight pattern. Based on the observation in Figure 4.24, the proposed framework is able to stitched the aerial frames of the footage. However, there is visible skewness and several frames that are misaligned with each other in the stitched image. This is because the homography estimation is very sensitive to the cumulative homography error during the homography matrix computation. The cumulative homoprahy error is caused by the interframe homography error, where the interframe homography error is made up of the camera reprojection error between the frames. One of the alignment optimization approaches, named Bundle Adjustnment is widely used by researchers to optimize the cumulative homography error. However, it is a costly and high complexity algorithm to develop.

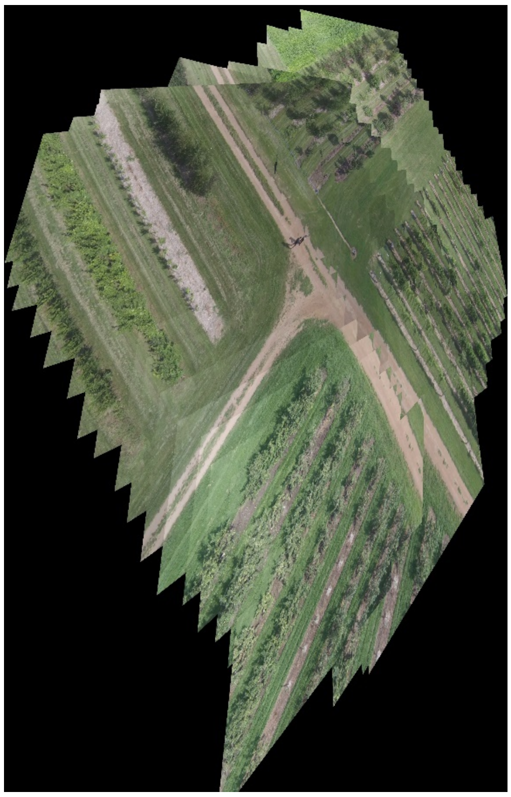


Figure 4.24: Stitched image of the dataset with lawnmower flight pattern

4.9 Finalization of Parameters

The summarized parameters are set in Table 4.6.

Table 4.6: Summarized parameter settings of orchard dataset

Dataset	UMN Horticulture Field Station (Orchard)
Resolution after resize (pixels)	640 x 360
Convert to grayscale	Yes
Gaussian blur	Yes
Feature representation	SIFT
Octave layer	3.0
Scale of each layer	5.0
Sigma of Gaussian, σ_0	1.6
Contrast threshold	0.04
Edge threshold	10.0
Feature matcher	FLANN (Randomized k-d tree)
Number of trees	5
Lowe's ratio test	Yes
Lowe's ratio	0.8
GMS	No
Outlier Removal	RANSAC
Reprojection threshold	5.0
Iteration, N	2000

With the use of parameters shown in Table 4.6, final stitched result is shown in Figure 4.25. The interframe alignment of the stitched image is great. The Orchard is visualized clearly.

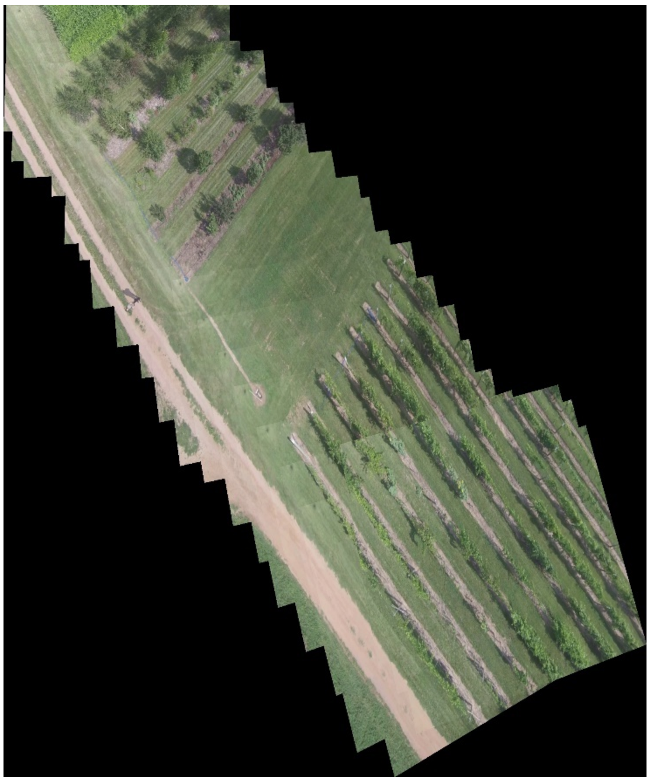


Figure 4.25: Final stitched result

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

This project has managed to stitch the aerial footage's frames to generate a well-aligned stitched image with a large scene view. It has successfully dealt with the concern of manual keyframe selection. The framework has managed to undistort the image to remove the bird-eye effect and prevent image stitching failure.

This project utilized the sparse feature-based feature representation method, SIFT for feature point detection and description. The generated results between the SIFT and binary-based feature representation, ORB is evaluated. The result of using SIFT performs better than ORB. For the feature matching, the randomized k-d tree of FLANN matcher is implemented in the proposed project and successfully stitches the images of aerial footage. Its performance is compared with that of brute-force searching method and found that their results are equally great. Yet, FLANN matcher is better in terms of efficiency in processing large dataset. Besides, the effect of feature match filtering methods is investigated. It is demonstrated that the use of Lowe's ratio test is the appropriate method to be employed in this project. The proposed framework has managed to stitch the aerial footage of the Orchard dataset with an L-shape flight pattern and high degree of similarity in term of structure content of the images.

The proposed framework successfully stitches the aerial footage of orchard dataset with the lawnmower flight pattern. However, the misalignment of the frames in the stitched image is observed when the Orchard dataset with the lawnmower flight pattern is inserted. Therefore, the alignment optimization method shall be implemented in future work.

5.2 Recommendations for Future Work

The future work of this project can be expanded with the following recommendations:

1. It is suggested to implement bundle adjustment to mitigate the camera reprojection error and reduce the cumulative homography error. Bundle

adjustment is a state-of-the-art approach on optimizing the alignment of the frames in the stitched image.

2. It is recommended to enable the image stitching program to execute in graphical processing unit (GPU) instead of the conventional central processing unit (CPU). Parallel computing can be done by using GPU and it has optimized memory bandwidth. Hence, fast image processing speed is provided in large memory operation.
3. It is suggested to adopt 3D reconstruction method, such as Simultaneous Localization and Mapping (SLAM) or Structure from Motion (SfM) to register images. It would allow the reconstruction of the ground plane of the flight trajectory, improve the alignment, and prevent the visible skewness of the stitched image.
4. It is recommended to employ deep learning based semantic image matching. The trained CNN features can provide invariant capability against geometric deformations and illumination change of the images. Therefore, it can accurately detect the feature points of the images.

REFERENCES

Adel, E., Elmogy, M. and Elbakry, H. (2015) ‘Image Stitching System Based on ORB Feature-Based Technique and Compensation Blending’, *International Journal of Advanced Computer Science and Applications*, 6(9), pp. 55–62. doi: 10.14569/ijacsa.2015.060907.

Baid, U. R. (2015) *Image Registration and Homography Estimation*. doi: 10.13140/RG.2.2.19709.67043.

Bian, J. W. *et al.* (2020) ‘GMS: Grid-Based Motion Statistics for Fast, Ultra-robust Feature Correspondence’, *International Journal of Computer Vision*. Springer US, 128(6), pp. 1580–1593. doi: 10.1007/s11263-019-01280-3.

Bradski, G. and Kaehler, A. (2009) *Learning OpenCV—Computer Vision with the OpenCV Library*, *IEEE Robotics and Automation Magazine*. doi: 10.1109/MRA.2009.933612.

Brown, M. and Lowe, D. (2002) ‘Invariant Features from Interest Point Groups’, pp. 23.1-23.10. doi: 10.5244/c.16.23.

Brown, M. and Lowe, D. G. (2007) ‘Automatic Panoramic Image Stitching Automatic 2D Stitching’, *International Journal of Computer Vision*, 74(1), pp. 59--73. Available at: http://dx.doi.org/10.1007/s11263-006-0002-3.

Chen, J. *et al.* (2019) ‘A robust method for automatic panoramic UAV image mosaic’, *Sensors (Switzerland)*, 19(8), pp. 1–17. doi: 10.3390/s19081898.

Eisenbeiss, H. and Sauerbier, M. (2011) ‘Investigation of Uav Systems and Flight Modes for Photogrammetric Applications’, *Photogramm*.

Gao, J., Kim, S. J. and Brown, M. S. (2011) ‘Constructing image panoramas using dual-homography warping’, *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 49–56. doi: 10.1109/CVPR.2011.5995433.

Hossain, M. F. and Alsharif, M. R. (2007) ‘A Performance Evaluation of Local Descriptors’, *2007 International Conference on Convergence Information Technology, ICCIT 2007*, 27(10), pp. 1439–1444. doi: 10.1109/ICCIT.2007.4420457.

Ju, M. H. and Kang, H. B. (2010) ‘A new simple method to stitch images with lens distortion’, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6454 LNCS(PART 2), pp. 273–282. doi: 10.1007/978-3-642-17274-8_27.

Ju, M. H. and Kang, H. B. (2014) ‘Stitching images with arbitrary lens distortions’, *International Journal of Advanced Robotic Systems*, 11(1), pp. 1–11. doi: 10.5772/57160.

Levin, A. *et al.* (2004) ‘Seamless image stitching in the gradient domain’, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3024, pp. 377–389. doi: 10.1007/978-3-540-24673-2_31.

Li, J. *et al.* (2014) ‘Fast aerial video stitching’, *International Journal of Advanced Robotic Systems*, 11. doi: 10.5772/59029.

De Lima, R. and Martinez-Carranza, J. (2017) ‘Real-time aerial image mosaicing using hashing-based matching’, *2017 Workshop on Research, Education and Development of Unmanned Aerial Systems, RED-UAS 2017*, pp. 144–149. doi: 10.1109/RED-UAS.2017.8101658.

Liu, W. X. and Chin, T.-J. (2016) ‘Correspondence Insertion for As-Projective-As-Possible Image Stitching’. Available at: http://arxiv.org/abs/1608.07997.

Lowe, D. G. (2004) ‘Distinctive Image Features from Scale-Invariant Keypoints’, *International Journal of Computer Vision*.

Lyu, W. *et al.* (2019) ‘A Survey of image and video stitching’, *Virtual Reality & Intelligent Hardware*, pp. 55-83.

Moussa, A. and El-Sheimy, N. (2016) ‘A fast approach for stitching of aerial images’, *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, 41(July), pp. 769–774. doi: 10.5194/isprsarchives-XLI-B3-769-2016.

Muja, M. and Lowe, D. G. (2014) ‘Scalable Nearest Neighbour Methods for High Dimensional Data’, *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36(April), pp. 1–14. Available at: http://elk.library.ubc.ca/handle/2429/44402.

Pravenaa, S. and Menaka, R. (2016) ‘A methodical review on image stitching and video stitching techniques’, *International Journal of Applied Engineering Research*, 11(5), pp. 3442–3448.

Rublee, E. *et al.* (2011) ‘ORB: An efficient alternative to SIFT or SURF’, *Proceedings of the IEEE International Conference on Computer Vision*. IEEE, pp. 2564–2571. doi: 10.1109/ICCV.2011.6126544.

Uyttendaele, M., Eden, A. and Szeliski, R. (2001) ‘Eliminating ghosting and exposure artifacts in image mosaics’, *Proceedings of the IEEE Computer*

Society Conference on Computer Vision and Pattern Recognition, 2, pp. II509–II516. doi: 10.1109/CVPR.2001.991005.

Younes, L., Romaniuk, B. and Bittar, E. (2012) ‘A Comprehensive and Comparative Survey of the Sift Algorithm (feature detection, description, and characterization)’, *2012 7th Internaltional Conference on Computer Vision Theory and Application, VISAPP 2012 - Proceeding,* pp. 467-474.

Yuan, X., Zhu, R. and Su, L. (2011) ‘A calibration method based on OpenCV’, *2011 3rd International Workshop on Intelligent Systems and Applications, ISA 2011 - Proceedings*, pp. 1–4. doi: 10.1109/ISA.2011.5873428.

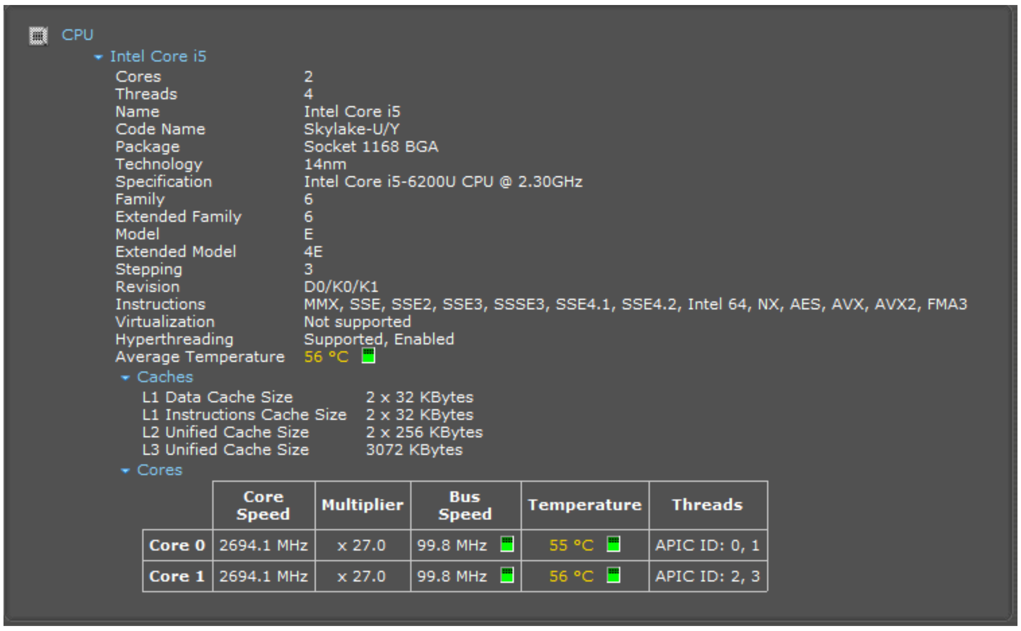
Zaragoza, J. *et al.* (2014) ‘As-projective-as-possible image stitching with moving DLT’, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7), pp. 1285–1298. doi: 10.1109/TPAMI.2013.247.

Zhao, J. *et al.* (2019) ‘Rapid mosaicking of unmanned aerial vehicle (UAV) images for crop growth monitoring using the SIFT algorithm’, *Remote Sensing*, 11(10). doi: 10.3390/rs11101226.

Zhi, Q. and Cooperstock, J. R. (2012) ‘Toward dynamic image mosaic generation with robustness to parallax’, *IEEE Transactions on Image Processing*, 21(1), pp. 366–378. doi: 10.1109/TIP.2011.2162743.

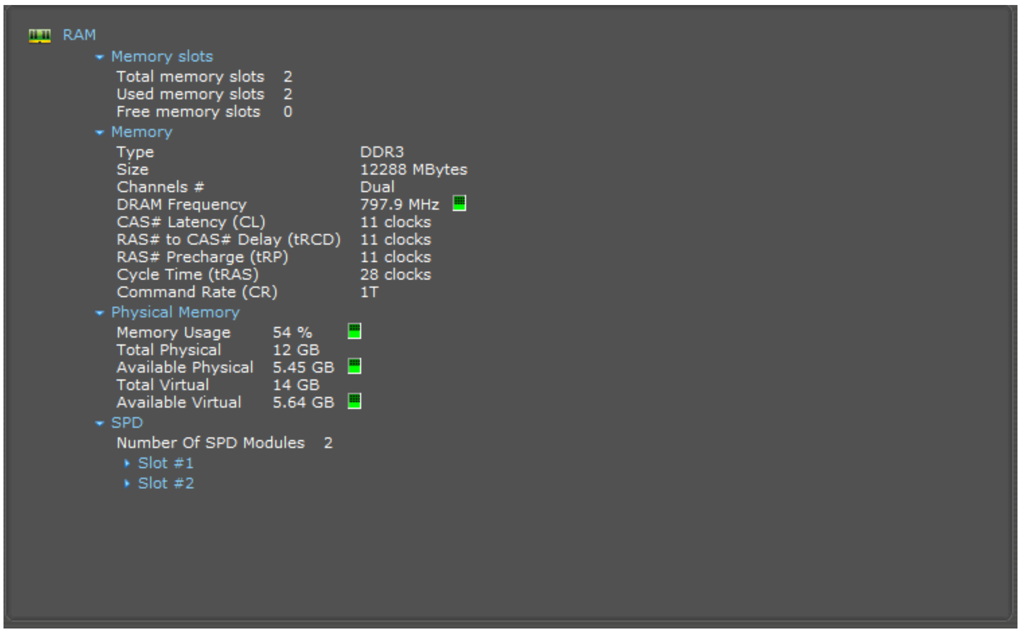
APPENDICES

APPENDIX A: Computer Specification



```
CPU
  Intel Core i5
    Cores                    2
    Threads                  4
    Name                     Intel Core i5
    Code Name                Skylake-U/Y
    Package                  Socket 1168 BGA
    Technology               14nm
    Specification            Intel Core i5-6200U CPU @ 2.30GHz
    Family                   6
    Extended Family          6
    Model                    E
    Extended Model           4E
    Stepping                 3
    Revision                 D0/K0/K1
    Instructions             MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, Intel 64, NX, AES, AVX, AVX2, FMA3
    Virtualization           Not supported
    Hyperthreading           Supported, Enabled
    Average Temperature      56 °C 🟩
    Caches
      L1 Data Cache Size           2 x 32 KBytes
      L1 Instructions Cache Size   2 x 32 KBytes
      L2 Unified Cache Size        2 x 256 KBytes
      L3 Unified Cache Size        3072 KBytes
    Cores
```

	Core Speed	Multiplier	Bus Speed	Temperature	Threads
Core 0	2694.1 MHz	x 27.0	99.8 MHz 🟩	55 °C 🟩	APIC ID: 0, 1
Core 1	2694.1 MHz	x 27.0	99.8 MHz 🟩	56 °C 🟩	APIC ID: 2, 3



```
RAM
  Memory slots
    Total memory slots   2
    Used memory slots    2
    Free memory slots    0
  Memory
    Type                              DDR3
    Size                              12288 MBytes
    Channels #                        Dual
    DRAM Frequency                    797.9 MHz 🟩
    CAS# Latency (CL)                 11 clocks
    RAS# to CAS# Delay (tRCD)         11 clocks
    RAS# Precharge (tRP)              11 clocks
    Cycle Time (tRAS)                 28 clocks
    Command Rate (CR)                 1T
  Physical Memory
    Memory Usage         54 %     🟩
    Total Physical       12 GB
    Available Physical   5.45 GB  🟩
    Total Virtual        14 GB
    Available Virtual    5.64 GB  🟩
  SPD
    Number Of SPD Modules   2
      Slot #1
      Slot #2
```

APPENDIX B: Python Codes

footageProcessing.py

```
# -*- coding: utf-8 -*-
"""
```

```
Frame Extraction and Saving from Video
"""
'''
~ MATLAB ~
Focal Length:        fc = [ 1237.36828   1275.40537 ] ± [ 5.56050   5.78613 ]
Principal point:     cc = [ 954.79824   560.46217 ] ± [ 3.16465   3.22632 ]
Skew:            alpha_c = [ -0.00099 ] ± [ 0.00039  ]   => angle of pixel axes =
90.05701 ± 0.02233 degrees
Distortion:          kc = [ -0.30409   0.10001   -0.00065   -0.00156  0.00000 ] ±
[ 0.00525   0.00635   0.00062   0.00059  0.00000 ]
Pixel error:         err = [ 0.20609   0.21830 ]

~ openCV ~
Camera Matrix:                     K = [1237.36828, 0, 954.79824],[0,
1275.40537, 560.46217],[0, 0, 1]
Distortion Coefficient d = [-0.30409, 0.10001, -0.00065, -0.00156, 0]

(Reference: https://www.graceunderthesea.com/thesis/camera-calibration-to-
undistort-images)
'''
import os
import cv2
import argparse
import numpy as np
import glob

print ("[PACKAGE] openCV version: " + cv2.__version__)
pathIN = "F:/CourseSubject/FYP/FYP 2
Progress/imstCode/ImageStitch/video/GOPROHERO3.mp4"
pathOUT = "F:/CourseSubject/FYP/FYP 2
Progress/imstCode/ImageStitch/image/ExtImg"
pathOUT2 = "F:/CourseSubject/FYP/FYP 2
Progress/imstCode/ImageStitch/image/ExtImg/ProcessedImg"
```

```
# copy parameters to arrays
K = np.array([[1237.36828, 0, 954.79824],[0, 1275.40537, 560.46217],[0, 0, 1]])
d = np.array([-0.30409, 0.10001, -0.00065, -0.00156, 0]) # just use first two
terms

def extImages(pathIN, pathOUT):
    # grab the paths to the input video and starts extracting
    print("[INFO] loading video...")
    vidObj = cv2.VideoCapture(pathIN)

    # measure fps of the video
    fps = int(vidObj.get(cv2.CAP_PROP_FPS))
    print("[INFO] video FPS...", fps)



    # Used as couter variable
    cnt = 0
    IMGcounter = 1

    # checks whether frames were extracted
    success = 1

    while success:
        # videoObj object calls read
        # function extract frames
        success, image = vidObj.read()
        # save 1 frame out of 50 frame
        if cnt%50 == 0:
            print("[INFO] extracting frame %d ..." % IMGcounter)
            cv2.imwrite(os.path.join(pathOUT, "frame%d.jpg" % IMGcounter),
image)
            #saves the frames with fram-count
            h, w = image.shape[:2]
            # undistort the images
            newCamera, roi = cv2.getOptimalNewCameraMatrix(K, d, (w,h), 0)
            undistort_img = cv2.undistort(image, K, d, None, newCamera)

            # crop the image
            x, y, w, h = roi
            undistort_img = undistort_img[y:y+h, x:x+w]
            cv2.imwrite(os.path.join(pathOUT2, "frame%d.jpg" % IMGcounter),
undistort_img)
            IMGcounter += 1
        cnt += 1
    print("[INFO] Done extracting...")

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""        MAIN        FUNCTION
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
if __name__=="__main__":
    '''
    # construct the argument parser and parse the arguments
    ap = argparse.ArgumentParser()
    ap.add_argument("-v", "--video", type=str, required=True,
        help="Path to input directory of video to extract")
    ap.add_argument("-o", "--output", type=str, required=True,
        help="Path to the extracted images")
    args = vars(ap.parse_args())
    extImages(args["video"], args["output"])
    '''
    extImages(pathIN, pathOUT)
```

imageStitcher.py

```
# -*- coding: utf-8 -*-
"""
Author: Ng Wei Haen
```

```
Topic: Image Stitching of Aerial Footage
Start Date: 17 Dec 2020
"""

import cv2
import numpy as np
import glob
import os
import time
#import math
from colorama import Style, Back
import xlsxwriter as xls
"""
Important Parameter
-------------------
detector_type (string): type of determine, "sift" or "orb"
                    Defaults to "sift".
matcher_type (string): type of determine, "flann" or "bf"
                   Defaults to "flann".
resize_ratio (int) = number needed to decrease the input images size
output_height_times (int): determines the output height based on input image
height.
                      Defaults to 2.
output_width_times (int): determines the output width based on input image
width.
                      Defaults to 4.

"""
detector_type = "sift"
matcher_type = "flann"
resize_ratio = 3
output_height_times = 20
output_width_times = 15
```

```
gms = False
visualize = True

image_dir = "image/testsetORI"
key_frame = "image/testsetORI/frame1.jpg"
output_dir = "image/TestSIFT"

class ImageStitching:
    def __init__(self, first_image,
                 output_height_times = output_height_times,
                 output_width_times = output_width_times,
                 detector_type = detector_type,
                 matcher_type = matcher_type):
        """This class processes every frame and generates the panorama

        Args:
            first_image (image for the first frame): first image to initialize the output
size
            output_height_times (int, optional): determines the output height based
on input image height. Defaults to 2.
            output_width_times (int, optional): determines the output width based
on input image width. Defaults to 4.
            detector_type (str, optional): the detector for feature detection. It can be
"sift" or "orb". Defaults to "sift".
        """
        self.detector_type = detector_type
        self.matcher_type = matcher_type
        if detector_type == "sift":
            # SIFT feature detector
            self.detector = cv2.xfeatures2d.SIFT_create(nOctaveLayers = 3,
                                                        contrastThreshold = 0.04,
                                                        edgeThreshold = 10,
                                                        sigma = 1.6)
            if matcher_type == "flann":
                # FLANN: the randomized kd trees algorithm
                FLANN_INDEX_KDTREE = 1
                flann_params = dict(algorithm = FLANN_INDEX_KDTREE, trees =
5)
                search_params = dict (checks=200)
                self.matcher = cv2.FlannBasedMatcher(flann_params,search_params)

            else:
                # Brute-Force matcher
                self.matcher = cv2.BFMatcher()
        elif detector_type == "orb":
            # ORB feature detector
            self.detector = cv2.ORB_create()
            self.detector.setFastThreshold(0)
            if matcher_type == "flann":
                FLANN_INDEX_LSH = 6
                flann_params= dict(algorithm = FLANN_INDEX_LSH,
                                   table_number = 6, # 12
                                   key_size = 12,     # 20
                                   multi_probe_level = 1) #2
                search_params = dict (checks=200)
                self.matcher = cv2.FlannBasedMatcher(flann_params,search_params)
            else:
                # Brute-Force-Hamming matcher
                self.matcher = cv2.BFMatcher(cv2.NORM_HAMMING,
crossCheck=True)

        self.record = []
        self.visualize = visualize
        self.output_img = np.zeros(shape=(int(output_height_times *
first_image.shape[0]),
                                          int(output_width_times*first_image.shape[1]),
                                 first_image.shape[2]))

        self.process_first_frame(first_image)

        # output image offset
        self.w_offset = int(self.output_img.shape[0]/2 - first_image.shape[0]/2)
        self.h_offset = int(self.output_img.shape[1]/2 - first_image.shape[1]/2)

        self.output_img[self.w_offset:self.w_offset+first_image.shape[0],
                        self.h_offset:self.h_offset+first_image.shape[1], :] = first_image
        a = self.output_img
        heightM, widthM = a.shape[:2]
        a = cv2.resize(a, (int(widthM / 4),
                           int(heightM / 4)),
                       interpolation=cv2.INTER_AREA)
        # cv2.imshow('output', a)
        self.H_old = np.eye(3)
        self.H_old[0, 2] = self.h_offset
        self.H_old[1, 2] = self.w_offset

    def process_first_frame(self, first_image):
        """processes the first frame for feature detection and description

        Args:
            first_image (cv2 image/np array): first image for feature detection
        """
        self.base_frame_rgb = first_image
        base_frame_gray = cv2.cvtColor(first_image, cv2.COLOR_BGR2GRAY)
        base_frame = cv2.GaussianBlur(base_frame_gray, (5,5), 0)
        self.base_features,                    self.base_desc                    =
self.detector.detectAndCompute(base_frame, None)

    def process_adj_frame(self, next_frame_rgb):
        """gets an image and processes that image for mosaicing

        Args:
            next_frame_rgb (np array): input of current frame for the mosaicing
        """
        self.next_frame_rgb = next_frame_rgb
        next_frame_gray = cv2.cvtColor(next_frame_rgb,
cv2.COLOR_BGR2GRAY)
        next_frame = cv2.GaussianBlur(next_frame_gray, (5,5), 0)
        self.next_features, self.next_desc =
self.detector.detectAndCompute(next_frame, None)


        self.matchingNhomography(self.next_desc, self.base_desc)

        if len(self.matches) < 4:
            return

        print ("\n")
        self.warp(self.next_frame_rgb, self.H)

        # For record purpose: save into csv file later
        self.record.append([len(self.base_features), len(self.next_features),
                    self.no_match_lr, self.no_GMSmatches, self.inlier,
self.inlierRatio, self.reproError])

        # loop preparation
        self.H_old = self.H
        self.base_features = self.next_features
        self.base_desc = self.next_desc
        self.base_frame_rgb = self.next_frame_rgb

    def matchingNhomography(self, next_desc, base_desc):
        """matches the descriptors

        Args:
            next_desc (np array): current frame descriptor
            base_desc (np array): previous frame descriptor

        Returns:
            array: and array of matches between descriptors
        """
        # matching
        if self.detector_type == "sift":
            pair_matches = self.matcher.knnMatch(next_desc, trainDescriptors = 
base_desc,
                                              k = 2)

            """
                Store all the good matches as per Lowe's ratio test'
                The Lowe's ratio is refer to the journal "Distinctive
                Image Features from Scale-Invariant Keypoints" by
                David G. Lowe.
            """
            lowe_ratio = 0.8
            matches = []
            for m, n in pair_matches:
                if m.distance < n.distance * lowe_ratio:
                    matches.append(m)
            self.no_match_lr = len(matches)
            # Rate of matches (Lowe's ratio test)
            rate = float(len(matches) / ((len(self.base_features) + 
len(self.next_features))/2))
            print (f"Rate of matches (Lowe's ratio test): 
{Back.RED}%f{Style.RESET_ALL}" % rate)
```

```
        elif self.detector_type == "orb":
            if self.matcher_type == "flann":
                matches = self.matcher.match(next_desc, base_desc)
                '''
                lowe_ratio = 0.8
                matches = []
                for m, n in pair_matches:
                    if m.distance < n.distance * lowe_ratio:
                        matches.append(m)
                '''
                self.no_match_lr = len(matches)
                # Rate of matches (Lowe's ratio test)
                rate = float(len(matches) / (len(base_desc) + len(next_desc)))
                print (f"Rate of matches (Lowe's ratio test): {Back.RED}%f{Style.RESET_ALL}" % rate)
            else:
                pair_matches = self.matcher.match(next_desc, base_desc)
                # Rate of matches (before Lowe's ratio test)
                self.no_match_lr = len(pair_matches)
                rate = float(len(pair_matches) / (len(base_desc) + len(next_desc)))
                print (f"Rate of matches: {Back.RED}%f{Style.RESET_ALL}" % rate)


        # Sort them in the order of their distance.
        matches = sorted(matches, key=lambda x: x.distance)

        # OPTIONAL: used to remove the unmatch pair match
        matches = cv2.xfeatures2d.matchGMS(self.next_frame_rgb.shape[:2],
                                           self.base_frame_rgb.shape[:2],
                                           self.next_features,
                                           self.base_features, matches,
                                withScale = False, withRotation = False,
                                thresholdFactor = 6.0) if gms else matches
        self.no_GMSmatches = len(matches) if gms else 0
        # Rate of matches (GMS)
        rate = float(self.no_GMSmatches / (len(base_desc) + len(next_desc)))
        print (f"Rate of matches (GMS): {Back.CYAN}%f{Style.RESET_ALL}" %
rate)

        # OPTIONAL: Obtain the maximum of 20 best matches
        # matches = matches[:min(len(matches), 20)]

        # Visualize the matches.
        if self.visualize:
            match_img = cv2.drawMatches(self.next_frame_rgb, self.next_features,
self.base_frame_rgb,
                              self.base_features, matches, None,

flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
            cv2.imshow('matches', match_img)

        self.H,                  self.status,                  self.reproError                  =
self.findHomography(self.next_features, self.base_features, matches)
        print ('inlier/matched = %d / %d' % (np.sum(self.status), len(self.status)))
        self.inlier = np.sum(self.status)
        self.inlierRatio = float(np.sum(self.status)) / float(len(self.status))
        print ('inlierRatio = ', self.inlierRatio)
        # len(status) - np.sum(status) = number of detected outliers

        '''
            TODO -
                To minimize or get rid of cumulative homography error is use block
bundle adjustnment
```

```
            Suggested from "Multi View Image Stitching of Planar Surfaces on
Mobile Devices"
            Using 3-dimentional multiplication to find cumulative homography is
very sensitive
            to homography error.
        '''
        # 3-dimensional multiplication to find cumulative homography to the
reference keyframe
        self.H = np.matmul(self.H_old, self.H)
        self.H = self.H/self.H[2,2]
        self.matches = matches
        return matches

    @ staticmethod
    def findHomography(base_features, next_features, matches):
        """gets two matches and calculate the homography between two images

        Args:
            base_features (np array): keypoints of image 1
            next_features (np_array): keypoints of image 2
            matches (np array): matches between keypoints in image 1 and image 2

        Returns:
            np arrat of shape [3,3]: Homography matrix
        """

        kp1 = []
        kp2 = []
        for match in matches:
            kp1.append(base_features[match.queryIdx])
            kp2.append(next_features[match.trainIdx])
        p1_array = np.array([k.pt for k in kp1])
        p2_array = np.array([k.pt for k in kp2])
        homography, status = cv2.findHomography(p1_array, p2_array, method =
cv2.RANSAC,
                                    ransacReprojThreshold = 5.0,
                                    mask = None,
                                    maxIters = 2000,
                                    confidence = 0.995)

        #### Finding the euclidean distance error ####
        list1 = np.array(p2_array)
        list2 = np.array(p1_array)
        list2 = np.reshape(list2, (len(list2), 2))
        ones = np.ones(len(list1))
        TestPoints = np.transpose(np.reshape(list1, (len(list1), 2)))
        print ("Length:", np.shape(TestPoints), np.shape(ones))
        TestPointsHom = np.vstack((TestPoints, ones))
        print ("Homogenous Points:", np.shape(TestPointsHom))

        projectedPointsH = np.matmul(homography, TestPointsHom) #
projecting the points in test image to collage image using homography matrix
        projectedPointsNH =
np.transpose(np.array([np.true_divide(projectedPointsH[0,:],
projectedPointsH[2,:]), np.true_divide(projectedPointsH[1,:],
projectedPointsH[2,:])]))

        print ("list2 shape:", np.shape(list2))
        print ("NH Points shape:", np.shape(projectedPointsNH))
        print ("Raw Error Vector:", np.shape(np.linalg.norm(projectedPointsNH-
list2, axis=1)))
        Error = int(np.sum(np.linalg.norm(projectedPointsNH-list2, axis=1)))
        print ("Total Error:", Error)
        AvgError = np.divide(np.array(Error), np.array(len(list1)))
        print ("Average Error:", AvgError)
```

```
        ##################
        return homography, status, AvgError

    def warp(self, next_frame_rgb, H):
        """ warps the current frame based of calculated homography H

        Args:
            next_frame_rgb (np array): current frame
            H (np array of shape [3,3]): homography matrix

        Returns:
            np array: image output of mosaicing
        """
        warped_img = cv2.warpPerspective(
            next_frame_rgb, H, (self.output_img.shape[1],
self.output_img.shape[0]),
            flags=cv2.INTER_LINEAR)

        transformed_corners = self.get_transformed_corners(next_frame_rgb, H)
        warped_img = self.draw_border(warped_img, transformed_corners)

        self.output_img[warped_img > 0] = warped_img[warped_img > 0]
        output_temp = np.copy(self.output_img)
        output_temp = self.draw_border(output_temp, transformed_corners,
color=(0, 0, 255))

        # Visualize the stitched result
        if self.visualize:
            output_temp_copy = output_temp/255.
            output_temp_copy = cv2.normalize(output_temp_copy, None, 0, 255,
cv2.NORM_MINMAX, cv2.CV_8U)  # convert float64 to unit8
            size = 720
            heightM, widthM = output_temp_copy.shape[:2]
            ratio = size / float(heightM)
            output_temp_copy = cv2.resize(output_temp_copy, (int(ratio * widthM),
size), interpolation=cv2.INTER_AREA)
            cv2.imshow('output',  output_temp_copy)

        return self.output_img

    @ staticmethod
    def get_transformed_corners(next_frame_rgb, H):
        """finds the corner of the current frame after warp

        Args:
            next_frame_rgb (np array): current frame
            H (np array of shape [3,3]): Homography matrix

        Returns:
            [np array]: a list of 4 corner points after warping
        """
        corner_0 = np.array([0, 0])
        corner_1 = np.array([next_frame_rgb.shape[1], 0])
        corner_2 = np.array([next_frame_rgb.shape[1], next_frame_rgb.shape[0]])
        corner_3 = np.array([0, next_frame_rgb.shape[0]])

        corners = np.array([[corner_0, corner_1, corner_2, corner_3]],
dtype=np.float32)
        transformed_corners = cv2.perspectiveTransform(corners, H)

        transformed_corners = np.array(transformed_corners, dtype=np.int32)
        # output_temp = np.copy(output_img)
        # mask = np.zeros(shape=(output_temp.shape[0], output_temp.shape[1],
1))
        # cv2.fillPoly(mask, transformed_corners, color=(1, 0, 0))
        # cv2.imshow('mask', mask)

        return transformed_corners

    def draw_border(self, image, corners, color=(0, 0, 0)):
        """This functions draw rectancle border

        Args:
            image ([type]): current mosaiced output
            corners (np array): list of corner points
            color (tuple, optional): color of the border lines. Defaults to (0, 0, 0).

        Returns:
            np array: the output image with border
        """
        for i in range(corners.shape[1]-1, -1, -1):
            cv2.line(image, tuple(corners[0, i, :]), tuple(
                corners[0, i-1, :]), thickness=5, color=color)
        return image

    @staticmethod
    def stitchedimg_crop(stitched_img):
        """This functions crop the black edge

        Args:
            stitched_img (np array): stitched image with black edge

        Returns:
            np array: the output image with no black edge
        """
        stitched_img = cv2.normalize(stitched_img, None, 0, 255,
cv2.NORM_MINMAX, cv2.CV_8U)  # convert float64 to unit8
        # Crop black edges
        stitched_img_gray = cv2.cvtColor(stitched_img,
cv2.COLOR_BGR2GRAY)
        _, thresh = cv2.threshold(stitched_img_gray, 1, 255,
cv2.THRESH_BINARY)
        dino, contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
        print ("Cropping black edge of stitched image ...")
        print ("Found %d contours...\n" % (len(contours)))

        max_area = 0
        best_rect = (0,0,0,0)

        for cnt in contours:
            x,y,w,h = cv2.boundingRect(cnt)

            deltaHeight = h-y
            deltaWidth = w-x
            if deltaHeight < 0 or deltaWidth < 0:
                deltaHeight = h+y
                deltaWidth = w+x

            area = deltaHeight * deltaWidth

            if ( area > max_area and deltaHeight > 0 and deltaWidth > 0):
                max_area = area
                best_rect = (x,y,w,h)

        if ( max_area > 0 ):
            final_img_crop = stitched_img[best_rect[1]:best_rect[1]+best_rect[3],
                    best_rect[0]:best_rect[0]+best_rect[2]]

        return final_img_crop
```

```
def main():
    images = sorted(glob.glob(image_dir + "/*.jpg"),
                    key=lambda x: int(os.path.splitext(os.path.basename(x))[0][5:]))
    # read the first frame
    first_frame = cv2.imread(key_frame)
    heightM, widthM = first_frame.shape[:2]
    first_frame = cv2.resize(first_frame, (int(widthM / resize_ratio),
                                          int(heightM / resize_ratio)),
                              interpolation=cv2.INTER_AREA)

    image_stitching = ImageStitching(first_frame)
    round = 2
    for next_img_path in images[1:]:
        print    (f'Reading    {Back.YELLOW}%s{Style.RESET_ALL}...'    %
next_img_path)
        next_frame_rgb = cv2.imread(next_img_path)
        heightM, widthM = next_frame_rgb.shape[:2]
        next_frame_rgb = cv2.resize(next_frame_rgb, (int(widthM / resize_ratio),
                                          int(heightM / resize_ratio)),
                              interpolation=cv2.INTER_AREA)

        print ("Stitching %d / %d of image ..." % (round,len(images)))
        # process each frame
        image_stitching.process_adj_frame(next_frame_rgb)

        round += 1
        if round > len(images):
            print ("Please press 'q' to continue the process ...")
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cv2.waitKey(0)
    cv2.destroyAllWindows()
    # cv2.imwrite('mosaic.jpg', image_stitching.output_img)
    final_img_crop                                                          =
image_stitching.stitchedimg_crop(image_stitching.output_img)

    print ("Image stitching done ...")
    cv2.imwrite("%s/Normal.JPG" % output_dir, final_img_crop)

    # Save important results into csv file
    tuplelist = tuple(image_stitching.record)
    workbook = xls.Workbook('Normal.xlsx')
    worksheet = workbook.add_worksheet("Normal")
    row = 0
    col = 0
    worksheet.write(row, col, 'number_pairs')
    worksheet.write(row, col + 1, 'basefeature')
    worksheet.write(row, col + 2, 'nextfeature')
    worksheet.write(row, col + 3, 'no_match_lr')
    worksheet.write(row, col + 4, 'match_rate')
    worksheet.write(row, col + 5, 'no_GMSmatches (OFF)')
    worksheet.write(row, col + 6, 'gms_match_rate')
    worksheet.write(row, col + 7, 'inlier')
    worksheet.write(row, col + 8, 'inlierratio')
    worksheet.write(row, col + 9, 'reproerror')
    row += 1
    number = 1
    # Iterate over the data and write it out row by row.
    for basefeature, nextfeature, no_match_lr, no_GMSmatches, inlier, inlierratio,
reproerror in (tuplelist):
        worksheet.write(row, col, number)
        worksheet.write(row, col + 1, basefeature)
        worksheet.write(row, col + 2, nextfeature)
        worksheet.write(row, col + 3, no_match_lr)
        match_rate = no_match_lr / ((basefeature+nextfeature)/2)
        worksheet.write(row, col + 4, match_rate)
        worksheet.write(row, col + 5, no_GMSmatches)
        gms_match_rate = no_GMSmatches / ((basefeature+nextfeature)/2)
        worksheet.write(row, col + 6, gms_match_rate)
        worksheet.write(row, col + 7, inlier)
        worksheet.write(row, col + 8, inlierratio)
        worksheet.write(row, col + 9, reproerror)
        number += 1
        row += 1

    workbook.close()

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""    Main
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
if __name__ == "__main__":
    program_start = time.process_time()
    main()
    program_end = time.process_time()
    print (f'Program elapsed time: {Back.GREEN}%s s{Style.RESET_ALL}\n' %
str(program_end-program_start))
```