A NEW FEATURE DESCRIPTOR FOR LIDAR STRIP MATCHING

by

MYTHREYA JAYENDRA LAKSHMAN

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2014

Acknowledgements

I would like to acknowledge Professor Venkat Devarajan for his kind support throughout my doctoral studies. He groomed me not only as a scientist, but also as an individual and a professional in a continual manner during my eventful and exciting studies. I would also like to thank Professor Farhad Kamangar for helping me greatly in my dissertation work by adding multiple dimensions with his constant help and availability. I would also like to thank my doctoral committee members, Dr. Jonathan Bredow, Dr. Michael Manry, Dr. Samir Iqbal and Dr. Vassilis Athitsos who gave me valuable inputs in the form of coursework or discussions during my studies and helped me in achieving my goals for my dissertation.

I would like to thank the Department of Electrical Engineering at UT Arlington for firstly admitting me into their competitive doctoral program. The faculty and staff were very friendly and they made my stay here a pleasure filled experience. Working at Virtual Environment Laboratory was the best part of each day and my group members helped me with enlightening discussions on computer vision and other scientific topics. I would like to thank the National Resources Conservation Service (NRCS) a part of the United States Department of Agriculture (USDA) for giving me the funding and opportunity to work on the LiDAR data matching project. They conducting frequent meetings and acknowledged that we were on the right track at various crucial junctures which helped the direction of research constantly.

Finally, I would like to conclude with a big word of thanks for the encouragement and great deal of patience shown by my wife and my parents during the entire period of my studies. Without their support I wouldn't have been in a position to write these words.

April 11th, 2014

Abstract

A NEW FEATURE DESCRIPTOR FOR LIDAR IMAGE MATCHING

Mythreya Jayendra Lakshman, PhD

The University of Texas at Arlington, 2014

Supervising Professor: Venkat Devarajan

Light Detection and Ranging (LiDAR) data has been getting a lot of attention these days because of its applicability in various fields - city planning, disaster response & precaution, soil conservation, infrastructure or forestry. Raw LiDAR data is not directly usable in the above-mentioned applications. Therefore, an important aspect of the aerial LIDAR system is to process raw LiDAR data accurately. The processed data is then made available to us in the form of point clouds or digital elevation maps (DEM's) or digital surface maps (DSM's). Each swath of raw data represents a single flight path over a region of interest. Multiple swaths or rectangular strips must be adjusted with respect to each other so that calibration errors and random errors that occur during the acquisition process are corrected.

Raw LiDAR data is available in point clouds of overlapping rectangular strips with 10%-30% overlap between the strips. Traditionally, LiDAR point clouds are matched and strip-adjusted (registered with respect to each other) using techniques based on iterative closest point (ICP) method or modifications of the same, which as the name suggests, runs over multiple iterations. The goal of the research presented in this dissertation is to introduce a novel and alternative method for point matching on LiDAR data based on finding point-to-point correspondences on interpolated LiDAR strips.

Publically or commercially viable LiDAR data is obtained by rescanning an irregular grid by interpolating the data onto a regular raster grid. This representation is known as 2.5D representation. The novel algorithm presented in this dissertation combines the power of LiDAR elevation data with a keypoint detector and descriptor - all obtained on the 2.5D LiDAR data. The keypoint detector finds interesting keypoints in the LiDAR strip by using standard keypoint detection techniques such as the scale-space blob-detection and corner detection techniques. Elevation statistics obtained from subdivisions of the neighborhood patch surrounding a keypoint are defined as the keypoint descriptor used in the approach presented in this dissertation. The elements of such descriptors are referred to as *features*. Once all the keypoints and descriptors are obtained for two overlapping patches, correspondences are found using the nearest neighbors of each point in the feature space. These matches can be used to find the homography that is used to transform one swath onto another. Scale Invariant Feature Transformation (SIFT) is a well-known and understood algorithm that uses a similar approach.

The novelty of the presented work called the Aerial Range Feature Descriptor (ARFD) is in the adaptive development of unique signatures of each keypoint obtained by examining a patch surrounding that keypoint in *the elevation and/or LiDAR intensity data*. Such hybrid descriptor development has been made possible only by the availability of pre-registered intensity and elevation data from modern LIDAR platforms.

Table of Contents

Acknowledgements iii

Abstract iv

List of Illustrations xiii

List of Tables xiv

Chapter 1 Introduction 1

1.1 Background 1

1.2 Details of LiDAR System 3

1.2.1 LiDAR Mission Planning 3

1.2.1.1 External Factors 3

1.2.1.2 Internal Factors 4

1.2.2 LiDAR on Board System 5

1.2.3 Post Processing System 6

1.3 LiDAR Surface Interpolation 7

1.4 LiDAR Strip Adjustment 8

1.4.1 Iterative Closest Point (ICP) Algorithm for Strip Adjustment 9

1.4.2 Tasks Involved in Strip Adjustment 9

1.5 Image Point Matching 10

1.5.1 Area Based Matching Techniques 10

1.5.2 Feature Based Matching Techniques 11

1.6 Dissertation Problem Statement and Goal 13

Chapter 2 Image Matching – Previous Work 14

2.1 Background 14

2.2 Background on Image Matching Techniques 14

2.2.1 Area Based Matching Techniques 14

2.2.1.1 Pixel Intensity Based Matching Techniques 14
2.2.1.2 Frequency Based Matching Techniques 17
2.2.1.3 Mutual Information Based Matching Techniques 18
2.2.2 Feature Based Matching Techniques 18
2.3 Details of Selected Feature Based Image Matching Techniques 19
2.3.1 ICP type approaches: IC-Patch Strip Adjustment 19
2.3.1.1 Linear and Aerial Feature Strip Adjustment 19
2.3.1.2 IC-Patch Strip Adjustment 20
2.3.2 Feature Based Matching 21
2.3.2.1 Neighborhood Based Matching Techniques 21
2.3.2.2 Intensity Based Matching Techniques 22
2.3.2.2.1 Scale-invariant Interest Point Indexing 22
2.3.2.2.2 Scale-invariant Feature Transform (SIFT) 23
2.3.2.2.3 Speeded Up Robust Features (SURF) 24
2.3.2.2.4 Gradient Location and Orientation Histogram (GLOH) 24
2.3.2.2.5 A Summary of Other Feature Descriptor Algorithms 25
2.3.3 Feature Based Matching Techniques in LiDAR 26
2.4 Conclusion 28
Chapter 3 Elevation Based Feature Descriptor 29
3.1 Background 29
3.2 Steps in LiDAR Strip Matching When Global Coordinates Are Available 29
3.2.1 Interpolation 31
3.2.2 Keypoints 33
3.2.2.1 Corner Detector 33

3.2.2.2 Scale-space Blob Detection 35
3.2.3 Feature descriptor 35
3.2.4 Distance Score for Keypoints in Two Images 36
3.2.5 Model Estimation (Homography) Using RANSAC 37
3.2.6 Quality of Matching 40
3.3 Steps in LiDAR Strip Matching When Geographic Information Is Unavailable 41
3.3.1 Distance Score for Keypoints in Two Images – Geographic Data Unavailable 41
3.3.2 Possible Matches – Geographic Data Unavailable 41
3.3.2.1 Nearest Neighbor Distance Ratio (NNDR) Based Pruning 43
3.3.3 Homography Calculation Following RANSAC 43
3.3.4 Quality of Matching 43
3.3.5 Recalculation of Homography after Obtaining True Positives and False Negatives 44
3.4 Feature Descriptors Used in Our Experiments 44
3.4.1 The Scale Invariant Feature Transform Algorithm 44
3.4.2 The Aerial Range Feature Descriptor (ARFD) 47
3.5 Variants of the Feature Descriptors Used in Our Experiments 49
3.5.1 Variants of ARFD Sub-patch Descriptors 49
3.5.2 Variants of Experiments Common Both to SIFT and ARFD 51
3.5.2.1 Use of SIFT and ARFD on Standard LiDAR Elevation Data 51
3.5.2.2 Use of SIFT and ARFD on Standard LiDAR Intensity Data 51
3.5.2.3 Modified Versions of SIFT and ARFD 51
3.6 Conclusion 52

Chapter 4 Details of Implementation of ARFD and Modified SIFT 53
4.1 Implementation Platform 53
4.2 LiDAR Dataset 53
4.3 Approach to Workflow Validation and Comparison of ARFD and SIFT 54
4.4 Implementation Details 54
4.4.1 Obtaining the LiDAR Data in Text Format 55
4.4.2 Interpolation of LiDAR Strip Data 55
4.4.3 Overlap Detector 56
4.4.4 Keypoint Detector 58
4.4.4.1 Shi-Tomasi Detector 58
4.4.4.2 Scale-space (LoG) Blob Detector 59
4.4.5 Feature Descriptor 59
4.4.5.1 ARFD 59
4.4.5.2 SIFT 60
4.4.6 Distance Evaluation 60
4.4.6.1 Geographic Distance Evaluation 60
4.4.6.2 Feature Distance Score 61
4.4.7 Nearest Neighbor Matching and Nearest Neighbor Distance Ratio (NNDR) Based Outlier Elimination 61
4.4.8 RANSAC Based Outlier Elimination 61
4.4.9 Performance Metrics 61
4.5 Details of Performance Metrics 62
4.5.1 True Positives (TP) 62
4.5.2 False Positives (FP) 62
4.5.3 True Negatives (TN) 63

4.5.4 False Negatives (FN) 63
4.6 Additional Quality Measurement Parameters 63
4.6.1 Precision 63
4.6.2 True Positive Rate (TPR) 64
4.6.3 False Positive Rate (FPR) 64
4.6.4 Accuracy 64
4.6.5 Mean Square Elevation Error (MSEE) 64
4.7 Conclusion 65
Chapter 5 Performance of the Feature Descriptors for LiDAR Matching for Unknown Search Area 66
5.1 Background 66
5.2 Performance Comparison of SIFT and ARFD Using LiDAR Elevation Data 67
5.2.1 Parameters of the Keypoint Detector 67
5.2.1.1 Performance of the SIFT Descriptor for Matching of Elevation Data 67
5.2.1.2 Performance of the ARFD Descriptor for Matching of Elevation Data 67
5.2.1.3 Performance Comparison of the ARFD and SIFT Descriptors for Matching of Elevation Data 68
5.2.2 Performance of Elevation Data When Keypoints Were Obtained from Shi-Tomasi Corner Points 69
5.3 Performance Comparison of SIFT and ARFD Using LiDAR Intensity Data 70

5.3.1 Performance of Intensity Data When Keypoints Were Obtained from Scale-space Blobs 70
5.3.2 Performance of Intensity Data When Keypoints Were Obtained from Shi-Tomasi Corner Points 71
5.4 Performance Comparison Between Elevation and Intensity Based Approaches 72
5.5 Performance of the Adaptive Approaches 72
5.5.1 Prior Attempts at Establishing Adaptive Approaches 72
5.5.2 Adaptive Technique for ARFD Based Matching 73
5.5.3 Adaptive Technique for SIFT Based Matching 75
5.5.4 Comparison Between Adaptive SIFT and Adaptive ARFD 76
5.6 Conclusion 77
Chapter 6 Performance of the SIFT and ARFD Algorithms for LiDAR Matching for Predetermined Search Area 78
6.1 Background 78
6.2 Requirement for LiDAR Strip Adjustment Under Common Coordinate System 79
6.3 Matching with ARFD and SIFT Descriptors With Elevation Data as Input 80
6.3.1 Matching Using Scale-space Blobs 80
6.3.2 Matching Using Shi-Tomasi Corner Points 81
6.4 Matching With ARFD and SIFT Descriptors With Intensity Data As Input 83
6.4.1 Matching Using Scale-space Blobs 83
6.4.2 Matching Using Shi-Tomasi Corner Points 83

6.5 Conclusion 84
Chapter 7 Conclusion 85
Appendix A MATLAB Codes 87
References 117
Biographical Information 123

List of Illustrations

Figure 1-1 Relationships among coordinate systems for a LIDAR mission..... 5

Figure 2-1 IC Patch Transformation Illustration 20

Figure 2-2 Feature Descriptor for Li and Olson Paper..... 26

Figure 3-1 LiDAR Image Matching Block Diagram – Geographic information available .. 30

Figure 3-2 Adaptation of inverse distance weighting 32

Figure 3-3 LiDAR Image Matching Block Diagram – Geographic information unavailable 42

Figure 3-4 SIFT Pyramid Formation [20] 45

Figure 3-5 SIFT Descriptor [20] 47

Figure 3-6 Sampling the keypoints 48

Figure 3-7 Extracting Sub-patch Information 49

Figure 4-1 Patch extraction from individual strips 54

Figure 4-2 Strip Overlap Configurations Horizontal Overlap 57

Figure 4-3 Strip Overlap Configurations Vertical Overlap..... 57

Figure 6-1 Two matching strips on which elevation based matching was performed 82

Figure 6-2 Alignment of the strips shown in Figure 6-2Figure 6-1 82

List of Tables

Table 5-1 Performance comparison of SIFT and ARFD for elevation data with Scale-Space blob keypoints 68

Table 5-2 Performance comparison of ARFD on elevation data using Scale-Space blob vs. Shi-Tomasi corner keypoints 70

Table 5-3 Performance comparison of SIFT and ARFD for intensity data with Scale-Space blob keypoints 71

Table 5-4 Performance comparison of ARFD on intensity data using Scale-Space blob vs. Shi-Tomasi corner keypoints 71

Table 5-5 Performance comparison – elevation and intensity based approaches 72

Table 5-6 Performance comparison of the ARFD adaptive algorithm versus intensity and elevation based approaches for the scale-space blob detector. 74

Table 5-7 Performance comparison of the adaptive algorithm versus intensity and elevation based approaches for the Shi-Tomasi detector 75

Table 5-8 Performance comparison of the SIFT adaptive algorithm versus intensity and elevation based approaches for the scale-space blob detector. 76

Table 5-9 Performance comparison between adaptive versions of SIFT and ARFD descriptors 76

Chapter 1

Introduction

1.1 Background

LIght Detection And Ranging (LiDAR) is a technique that is used to obtain depth information for objects by shining a light pulse at them. Aerial LiDAR is collected from airplanes and it is used to obtain elevations of points on the ground. These LiDAR datasets are collected on rectangular strips from aircraft equipped with a scanning laser emitter-receiver unit, Differential Global Positioning System (DGPS) and an Inertial Measurement Unit (IMU). The laser pulses that are fired from a mounted laser unit return after reflecting off the target area on earth. The time taken to return back to a sensor is measured and converted into a distance of the point on ground from the airplane. The *X*, *Y* & *Z* coordinates of the point on earth are obtained by further processing. In some cases, the LiDAR intensity *I* is obtained from the amplitude of the returned light pulse. LiDAR elevations (*Z* data) are quite accurate with a tolerance of 10 to 20 centimeters for current systems [1]. Common LiDAR systems work in the near-infrared band in the electromagnetic spectrum and are limited by their inability to penetrate water.

Despite using laser for firing light pulses to the ground, there is some divergence in the beam of light, causing a significant footprint to form on the ground. The area of the LiDAR footprint is given by [2]:

$$A_{L_{inst}} = \frac{h}{\cos^2 \theta_{inst}} \gamma$$

where,

γ – divergence of the laser beam

h – height of the plane from the ground

θ_{inst} – instantaneous scan angle

The footprint of LiDAR varies according to applications; there are small-footprint LiDAR systems with footprints of 10-30 centimeters. Typically occurring footprint diameters and footprint spacing are less than 0.5 meters and 1.5 meters respectively. This entire footprint is returned to the scanning sensor, which processes the data and assigns the average value in the footprint as the elevation (and intensity if applicable) of the point in question. It is important that the footprint diameter is considerably lesser than the footprint spacing for the purpose of accuracy of point measurements.

The rectangular strip of data, also called *the swath* has a width dependent on the maximum scan angle and the height of the airplane sensor from the ground. It is given by [2]:

$$SW = 2h \tan\frac{\theta}{2}$$

where,

h – height of the plane from the ground

θ – scan angle

With these equations taken into account, LiDAR data is collected by aircrafts and processed for further use by providing with the coordinates *X*, *Y & Z*, and intensity *I*. These coordinates and intensities can be used for further processing in the manner that the end application requires. Detailed explanations of the LiDAR data collecting system are provided in Section 1.2.

Another fact to note about LiDAR is that the laser pulses that are shot to a spot on the ground can have multiple returns depending on nature of the spot. For instance, a tree can have multiple returns because a portion of the light pulse is reflected back from the top of the tree, while some of the light passes through the tree's leaves or branches

and reflects from a part of the tree or bare earth under the tree. Therefore, care must be taken to consider the correct return based on the end application.

1.2 Details of LiDAR System

The LiDAR data collection process can be broadly classified into three major subdivisions, viz.: the LiDAR mission planning, the onboard system and post-processing.

1.2.1 LiDAR Mission Planning

LiDAR mission planning is dictated by external and internal factors. The external factors are the environmental factors while the internal factors are the equipment and calibration methodologies.

1.2.1.1 External Factors

The LiDAR mission planning refers to the set of steps taken before the aircraft is flown over the target area for data collection. Successful collection of LiDAR data depends on a lot of parameters mainly related to weather and atmospheric conditions. It is necessary that there is no cloud cover or fog on the day of the LiDAR mission. The light beams are unable to overcome these conditions to give successful measurements for intensity and elevation data. Therefore the mission must be planned days ahead of the actual flight date. Constant monitoring of weather must be done to make sure that there is no bad weather at the time of LiDAR data collection. Examples of bad weather patterns that can affect the LIDAR collection mission are cloud cover, rain, thunderstorms, mist, fog or smog.

LiDAR data can be collected either in the night or in the daytime without much loss of information. This is because of the fact that LiDAR is an active sensing technique

which doesn't require any aid from sunlight. This is an advantage in urban areas where the clutter in the scene due to traffic goes down at night resulting in higher accuracy of DEM's.

LiDAR missions need to take into account the type of terrain the aircraft would be flying over in order to plan flight paths correctly. In heavily forested areas, the system might consider the last returns to accurately ascertain digital terrain models (DTMs). However, for the purposes of the matching algorithm presented in this dissertation, it is necessary that the first returns are obtained from the reflected light pulses. Another factor to be considered is the weather in the past few weeks; this is not only crucial in predicting the weather on the flight mission date, but also informative in how much moisture is contained on an otherwise dry land. It is preferable to avoid wet-ground for the reason that near-infrared lasers do not penetrate water [3].

1.2.1.2 Internal Factors

There are various internal factors influencing data collection of LiDAR. The scanning rate required for good point cloud density must be carefully considered during the mission planning stage. The laser pulses are repeated typically up to 200 kHz and the scanning of the field of view (FOV) is performed at a rate of about 100 Hz. Depending on the amount of variation on the target area, correct values for these rates are chosen in order to achieve proper sampling of ground points.

Proper calibration of the devices used, viz. the IMU, DGPS and the Laser and scanning unit, – all of these need to be accomplished before the LiDAR mission.

1.2.2 LiDAR on Board System

The on board system, as mentioned in Section 1.2.1.2, contains the IMU, the DGPS and the laser scanning system. The DGPS system measures the current coordinates of the aircraft at all times and finds the coordinates in the reference world coordinate system in which the point cloud data is to be obtained. The IMU measures the roll, pitch and the yaw of the aircraft which is crucial in post-processing to accurately obtain the point cloud data. Tilts in any direction have to be accounted for when the time is measured and it is converted accordingly to the *X*, *Y* & *Z* coordinates of the point target on the ground. The IMU and DGPS are closely related in measuring of the point cloud data. The laser scanning system mainly has the laser unit and a mirror to reflect the beam to the point on the ground. The relationships among the coordinate systems within a LIDAR system are shown in Figure 1-1 [4].

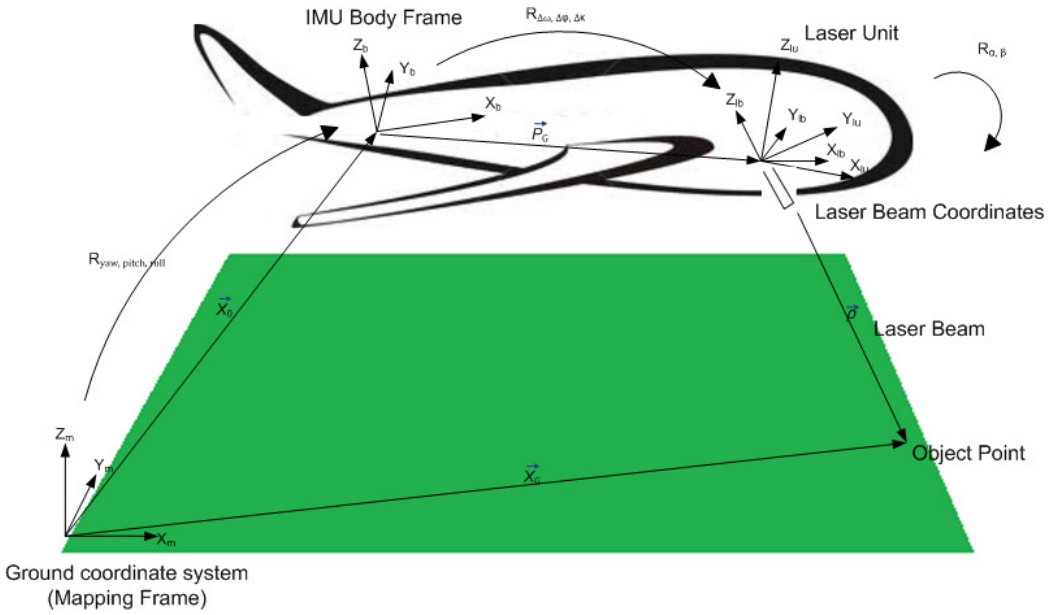

Figure 1-1 Relationships among coordinate systems for a LIDAR mission

1.2.3 Post Processing System

The point on the ground is evaluated by the post-processing system. The data from the DGPS is required by the post-processing system at the timestamp when the scanner picks up the returned beam. The roll, pitch, yaw parameters are supplied by the IMU and with proper calibration, the exact coordinates of the target point on the ground are evaluated using the following equation [4]:

$$\vec{X}_G = \vec{X}_0 + R_{yaw,pitch,roll}\vec{P}_G + R_{yaw,pitch,roll}R_{\Delta\omega,\Delta\phi,\Delta\kappa}R_{\alpha,\beta}\begin{bmatrix}0\\0\\-\rho\end{bmatrix}$$

where,

- $\vec{X}_G$ is the position of the point on the ground
- $\vec{X}_0$ is the vector between the origins of the ground and the IMU coordinate system
- $\vec{P}_G$ is the offset between the laser unit and the IMU coordinate system
- $\vec{\rho}$ is the laser range vector whose magnitude is the distance from the laser firing point to the footprint
- $R_{yaw,pitch,roll}$ is the rotation matrix relating the ground and IMU coordinate systems
- $R_{\Delta\omega,\Delta\phi,\Delta\kappa}$ is the rotation matrix relating the IMU and laser unit
- $R_{\alpha,\beta}$ is the rotation matrix relating the laser unit and the laser beam coordinate system.
- α, β are the mirror scan angles.

Post processing is performed offline to obtain the ground points using the above equation. There are multiple chances for the above setup to falter. Firstly, the calibration has to be up to date to have the correct parameters for the laser scanning unit and the

offsets and rotations. The IMU needs to have been calibrated properly to supply the correct roll, pitch and yaw parameters. The above set of steps constitutes quality assurance. However, during flight mission, random errors can occur in properly calibrated systems too. Therefore, once the points have been obtained, quality control must be performed in order to ascertain the accuracy of the data in hand. There is a likelihood that the system might need to be recalibrated based on the quality control steps. The relevance of the work in this dissertation related to quality control lies in providing a method for LiDAR strip matching, enabling quality control by fitting a projective transformation to the points in a query LiDAR strip to a reference strip or control points on the ground.

1.3 LiDAR Surface Interpolation

Interpolation is required in order to perform quality control of LiDAR data using strip adjustment. The reason for using interpolated data is that the probability that one point on one strip has a point-to-point correspondence in another strip is very low because of the differences in the scan directions and the inherent resolutions of the swath data. In order to find point-to-point correspondence for strip adjustment, interpolation must be performed to find the exact correspondence in the reference strip for each point in the query strip.

Another outcome of interpolation is the use of 2.5D techniques where the entire elevation data is set into a raster grid and point-to-point correspondences can then be obtained quite easily for the points in the raster grid for both point clouds.

Some of the LiDAR interpolation techniques that exist in literature are:

1. Inverse Distance Weighted (IDW) [5] interpolation estimates the unknown pixel values as a weighted average of the points that lie close to it. The weights

attached to each point are inversely proportional to the distance from the pixel center and they are normalized so that all the weights add up to unity. This is the algorithm that was considered for the purposes of LiDAR swath matching in this dissertation because of the simplicity of the underlying idea and ease of coding.

2. Natural Neighbor Interpolation [6] is a technique that uses area-based weighting to determine the unknown elevations.
3. Spline Interpolation [7] fits mathematical functions of piecewise polynomials of varying degrees to interpolate missing information.
4. Kriging [8] calculates unknown values from the weighted average of sample values where the weights are based both on the distance and the correlation among the samples.

1.4 LiDAR Strip Adjustment

LiDAR strip adjustment is a technique used for LiDAR data quality control purposes. The idea behind strip adjustment is to match a cloud of points in 3D - $P\{(x_i^p, y_i^p, z_i^p),\ i = 0,1, \dots, n\}$ and $Q\{(x_j^p, y_j^p, z_j^p),\ j = 0,1, \dots, n\}$. These points belong in an *irregular sampling* of the ground. It is difficult to set up the laser scanner so that a regular sampling grid is obtained during actual flight. Also, as mentioned earlier, obtaining point-to-point correspondences in two overlapping swaths of differing scanning directions is nearly impossible. The task involved in strip adjustment therefore is finding a transformation *T* that minimizes the distance between the two point clouds. This is described by the following equation [9]:

$$\min_T \|P - T(Q)\|$$

The transformation matrix that is required is given by the form shown below:

$$\begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix} = \begin{bmatrix} t_0 & t_1 & t_2 & t_3 \\ t_4 & t_5 & t_6 & t_7 \\ t_8 & t_9 & t_{10} & t_{11} \end{bmatrix} \begin{bmatrix} x_q \\ y_q \\ z_q \\ 1 \end{bmatrix}$$

where,

x_p , y_p, z_p- are the coordinates of the transformed points

x_q , y_q, z_q- are the coordinates of the points to be transformed

$t_0 - t_{11}$ – are the transformation parameters

1.4.1 Iterative Closest Point (ICP) Algorithm for Strip Adjustment

The ICP algorithm [10] aims to find the transformation matrix between by resolving it into two rigid body transformations, a rotation, *R*, and a translation, *T* [3]:

$$Min_{(R,T)} \sum_i \|P_i - (RQ_i + T)\|^2$$

The closest point in model *P* for each point in model *Q* is computed by ICP and at each step the model *Q* is updated by transforming using the current *R* and *T* values. This is continued until the relative changes in *R* and *T* are lesser than a threshold when compared to the previous iteration. At the termination point, the final *R* and *T* are found which is used to transform one point cloud dataset to another.

1.4.2 Tasks Involved in Strip Adjustment

Summarized below are the steps involved in most traditional strip adjustment techniques for LiDAR data:

- Within the strip overlap area sub-regions are selected in both strips for the saliency of the points present in them. Only one of the returns is chosen for the adjustment (either the first or the last).

- Strip discrepancies are evaluated based on the type of matching that is done: point-to-point, TIN-to-TIN (Triangulated Irregular Network), surface-to-surface or line-to-line.
- Transformation from one point cloud to another is first evaluated and later applied to the query image with respect to the reference image or control points.

1.5 Image Point Matching

It is therefore possible to match LiDAR data with feature detector-descriptor pairs. This enables either complete matching or at least an initial match for further point correction by ICP techniques. In this dissertation, techniques similar to image point matching are used for matching LiDAR swath data.

The image matching problem has been addressed from various directions in literature. Some of the common types of image matching are area based approaches and feature based approaches.

1.5.1 Area Based Matching Techniques

In the simplest form of these techniques, normalized cross correlations are performed between a template and a query image and a peak score is sought. The time complexity for this technique is pretty high. It is equal to $O(n_r n_c m_r m_c)$ where n_r is the number of rows in the query image, n_c is the number of columns in the query image, m_r is the number of rows in the template image, and m_c is the number of columns in the template image. In other words, the template image is searched for in all the possible sub-patches of the size of the template in the query image.

This technique, as defined in the preceding paragraph doesn't deal with transformations like rotation, scaling or perspective change. In order to address these transformations, the scaling factor of the template image must be varied and likely matches must be searched for across all of those scales. The same rule applies for rotation and perspective distortions. This technique doesn't even deal with illumination changes as it has been described so far and it requires normalization to function in uncontrolled environments.

Another drawback arises if the template and query images of comparable sizes are required to be matched. It would then be necessary that multiple sub-windows are found to be matched between the images based on a keypoint detection technique which picks out saliencies in the images. One or many of these keypoints have to be matched with each other.

While these are typically used in aligning different exposures of the same scene more finely, the drawbacks of area-based techniques encourage the use of feature based matching for LiDAR applications.

1.5.2 Feature Based Matching Techniques

Feature based matching techniques reduce the time taken for image matching and registration. The crux of these techniques is that certain landmarks that seem salient enough are chosen in both the images. Nearest neighbors are found for each keypoint in a reference image to each keypoint in the image that it is being matched with in order to transform and align the two images.

Feature based methods are intended to reduce the cost of matching images. To that end, interesting features are found by these methods which are repeatable in multiple views of the same image. Depending on the image transformations between the

two images, different types of features can be used for matching purposes. For instance, in the case of stereo matching for narrow baseline cameras, it can be assumed that lines do not undergo much transformation and linear features could be matched as a result of this assumption. In the case of applications like panorama stitching of images, it is quite possible that the viewpoints have changed appreciably between multiple views of the scene causing lines to skew to the extent of dissimilarity. In such cases, using invariants like corner points or blobs (affine-invariant in some cases) is a better idea to find matches across views.

Matching a few points selected from each image to another set in another image is not much different from the strip adjustment task usually undertaken in LiDAR. This is often performed in image matching for rigid transforms or when there is little or no occlusion in the image sets and both the images share a considerable portion of points or features. However, in tasks like matching elevation point clouds could be hindered by the lack of direct point-to-point correspondences and hence, a direct feature-to-feature correspondence is desired.

To obtain point-to-point or in general, feature-to-feature correspondences, there must be a unique signature for each interest feature in every image. These unique signatures are also called feature descriptors. It is within this context that the scope of the research presented in this dissertation lies and these techniques are discussed in detail in the following chapters.

1.6 Dissertation Problem Statement and Goal

In the algorithm presented in this dissertation called the Aerial Range Feature Descriptor (ARFD), the aim is to match overlapping LiDAR strips by feature detection and matching techniques. Firstly, the LiDAR data is interpolated and set on a raster grid, keypoints are detected in the resulting data using known techniques and the ARFD feature descriptor is created which is used to match detected keypoints in the overlapping image by a nearest neighbor matching algorithm.

Techniques that match the LiDAR data with its standalone elevation and standalone intensity data are developed. Following this, an adaptive technique is developed that chooses the descriptor from every keypoint's elevation or intensity information based on how much more 'interesting' one of these modalities are with respect to the other. The novelty of the algorithm lies in the fact that both the LiDAR intensity and elevation data are used in an adaptive fashion in creating a feature descriptor. The dissertation also modifies the Scale Invariant Feature Transform [11] algorithm by performing the adaptive descriptor choice. The performance of multiple versions of ARFD are compared with the multiple modified versions of the SIFT and comprehensive results are presented.

Two cases are considered: a) where matching is performed with a common coordinate system for the two LiDAR strips being unavailable (like matching two images with unknown camera locations), and b) where the matching is performed with a known common coordinate system

The performance of the multiple versions of ARFD is compared to the multiple versions of the Scale Invariant Feature Transform (SIFT) algorithm and detailed results are presented for both cases of matching.

Chapter 2

Image Matching – Previous Work

2.1 Background

Image matching is the general task of matching two possibly overlapping images of the same or of different imaging modalities. Given the close alignment between the problems of image matching and LiDAR data quality control, it is important that the existing image matching techniques that could aid the end goal of matching LiDAR data are introduced. Image matching, as mentioned in Section 1.5 can be performed using area-based methods and feature based methods. While area based approaches are applicable to some problems where a template needs to be found in a query image, the speed-up offered by feature based techniques in matching and finding correspondences [11-15] in overlapping images makes it more suitable in LiDAR strip matching.

2.2 Background on Image Matching Techniques

2.2.1 Area Based Matching Techniques

The emphasis in area based techniques is to find a high similarity between the template image and the query image. Therefore, the feature detection and feature description stages are merged into one [16]. Because of this reason, salient points or features are not recognized in area based methods and no pruning of the search space in the query image to find matches between the two images.

2.2.1.1 Pixel Intensity Based Matching Techniques

Intensity based matching techniques have been used for matching and registration of images. A similarity score like the normalized cross-correlation maximum

indicates the presence of the template window in the query image. The normalized cross-correlation score is given by:

$$C(i,j) = \frac{\sum_{u=-W}^{W}\sum_{v=-W}^{W}(Q(i+u,j+v)-\bar{Q})(T(W+u,W+v)-\bar{T})}{(2W+1)\sigma_q\sigma_t}$$

where,

- $C(i,j)$ is the normalized cross-correlation score at position (i,j) of the query image.
- Q is the query image that is to be match the template image T with.
- $2W+1$ is the size of the template window selected.
- $\bar{Q}$ and $\bar{T}$ are the means of the selected window of the query and template images respectively.
- σ_q and σ_t are the standard deviations of the selected window of the query and template images respectively.

The match for the template in the query image is obtained where the $C(i,j)$ function peaks. Some of the challenges with using these techniques are listed below:

1. In the case of matching two images of similar size, an appropriate template significantly smaller than either image must be selected in one of the two images.
2. Invariance of the matching approach to changes in any one of the following: scale, rotation or perspective changes in the images, is not automatic. Therefore adaptation must be made in order to account for these changes – the template window must be changed for multiple scales, 360 degree rotations or tilts based on the expected transformation between the images. In case of a complex transform that involves multiple rigid body transformations, all of the changing parameters must be accounted for. This is a high cost operation and it doesn't even deal with non-rigid transformations of images.

3. Where there is a great degree of self-similarity between parts of images, the cross-correlation score peaks in multiple locations in the query image. It makes it hard to find a unique transformation between the images.
4. When the objects in the scene are occluded choice of a wrong template window is possible. Also, the sub-window of a template with the query image doesn't guarantee that the rest of the overlap area will match.

Despite these drawbacks, the simplicity of the underlying concept prompted use in situations where time taken to match is not of utmost importance. Normalized cross-correlation was used with adaption to scale and rotation since the 1970s.

Anuta [17] tackled the problem of multi-image registration with cross-correlation. The paper also tackled multi-temporal registration of aerial images where the images undergo changes in time. They investigated multi-image registration with the use of Fast-Fourier Transforms (FFT) for registration of the images and concluded that the algorithm performed better in terms of computation time.

Barnea and Silverman [18] developed a method of registration for translated images that speeded up the normalized cross-correlation by introducing the Sequential Similarity Detection Algorithm (SSDA). Their algorithm took into account the fact that it is wasteful to evaluate a similarity measure or a normalized cross correlation score for all windows, especially for mismatching windows. In order to counter this problem, they took both the windows and evaluate a L_1-norm difference score for random orderings of the pixels within the window. They continued this evaluation until the difference score exceeded a threshold. For dissimilar windows this algorithm terminated quickly, and for similar windows the algorithm might not terminate till all the pixels in the window are evaluated. The number of pixel required for termination of the algorithm was noted as the matching score. If the threshold was chosen properly, the algorithm picked out the best

match in the query image. The authors claimed that in typical matching cases, it was not unusual to see speedups of 50 times the speed for normalized cross correlation. They also suggested a coarse-to-fine matching algorithm by initially sampling uniformly one in every *m* points. For windows with good similarity measures, the algorithm could be fine-tuned by reducing the sampling distance progressively until only one maximum score is obtained. Of course, they made the assumption that there exists only one match for the template in the query image.

Pratt [19] was able to register two translated images not only by applying the normalized cross correlation, but also by taking into account the statistical properties of each of the windows that were being correlated. The paper recommended increasing the dimensions of the correlation match function in order to account for rotations and other image transformations. It also addressed the problem of high computation for gross mis-registrations and suggested improvements for Sequential Similarity Detection Algorithms (SSDA) [18].

2.2.1.2 Frequency Based Matching Techniques

Frequency based matching techniques aim to match the images in the frequency domain [17]. The concept used here is the Fourier Shift Theorem [20, 21]. Fourier matching is highly resilient to frequency noise and changes in image illumination. The correlation function is given by:

$$\frac{F(t)F(q)^*}{|F(t)F(q)^*|} = e^{2\pi i(ux_0+vy_0)}$$

where,

t, q are the template and query images and x_0, y_0 are the shifts.

Application of the inverse Fourier transform yields the translation parameters for registering the two images.

This approach was extended further to account for rotation and translation parameters by De Castro and Morandi [22]. They were able to extract the three parameters - two for translation, and one for rotation – to register two images.

2.2.1.3 Mutual Information Based Matching Techniques

It is often the case in medical images and remote sensing images that registration has to be performed on multimodal images [20]. It is difficult to register a pair of images of different imaging modalities; even different lighting conditions (day vs. night). In such cases, the statistical properties of the images must be leveraged to find a match and eventually register the images. Mutual information between two datasets/images Q and T is given by:

$$MI(Q,T) = H(T) - H(T|Q) = H(T) + H(Q) - H(T,Q)$$

Where $H_X = -E_X(\log \mathrm{P(X)})$ is the entropy of a random variable, X. This method is not applicable to our feature based matching objective.

2.2.2 Feature Based Matching Techniques

Feature based matching techniques, unlike area based methods, find interest points or features in both the query and the template images and tries to match them based on geometric proximity or proximity in the feature space with an appropriate distance measure [23-25]. Features can range from being very simple to extremely complex; even drawing from characteristics of human vision [11, 15]. Other features can be as simple as points, blobs, line segments, curves or basic shapes.

Feature based matching finds use in the areas of wide baseline stereo matching [26-29], Simultaneous Localization and Mapping (SLAM) [30-33], 3D reconstruction [14, 34, 35], image mosaicking [36-38], texture recognition/matching [39-41], image retrieval [42-44], object classification [45-47], object recognition [48-51], face recognition [52-55] just to name a few**.**

Feature based matching techniques came to the forefront in the 1980s with the advent of the Canny edge detection algorithm [56] and the Harris-Stephens corner detection algorithm [57], which helped automatic selection of keypoints in both images. Thereafter, matching point clouds of data was made possible by matching these keypoints - which were not restricted only to the above two algorithms – by using iterative techniques to register 3D point clouds or 3D shapes [10, 58]. These same techniques are similar to the ones used to match LiDAR data by strip adjustment with the difference that LiDAR point clouds are denser.

Later, feature-based techniques were developed that used these points and assigned unique signatures or descriptors to them. Having these unique signatures for the keypoints greatly improved matching by obtaining direct point-to-point correspondences. [11, 14]

2.3 Details of Selected Feature Based Image Matching Techniques

2.3.1 ICP type approaches: IC-Patch Strip Adjustment

2.3.1.1 Linear and Aerial Feature Strip Adjustment

Habib *et al.* [4] introduced the IC-Patch strip adjustment technique. They also presented a method of matching and quality control of LiDAR data by using linear and aerial features for strip adjustment and finding the transformation between two sets of point clouds. For strip adjustment, they assumed that for registration of aerial features, it

is enough that the features are represented by their centroid together with its surface normal. Linear features were represented using their end points. The linear features were obtained by selecting corresponding areas in the overlapping strips after segmentation.

The transformation parameters are estimated from these roughly corresponding features. The parameters of the transformation are obtained by making slight modifications to the ICP algorithm – instead of minimizing regular distances between point clouds, they minimized the normal distances between corresponding features in the two point sets which they obtained by segmentation. They suggested a weight restriction system based on variance-covariance properties of elevation data to compensate for the fact that the correspondences are only approximate. Applying the restriction ensured that the geometric distances between the matching planar features were minimum.

2.3.1.2 IC-Patch Strip Adjustment

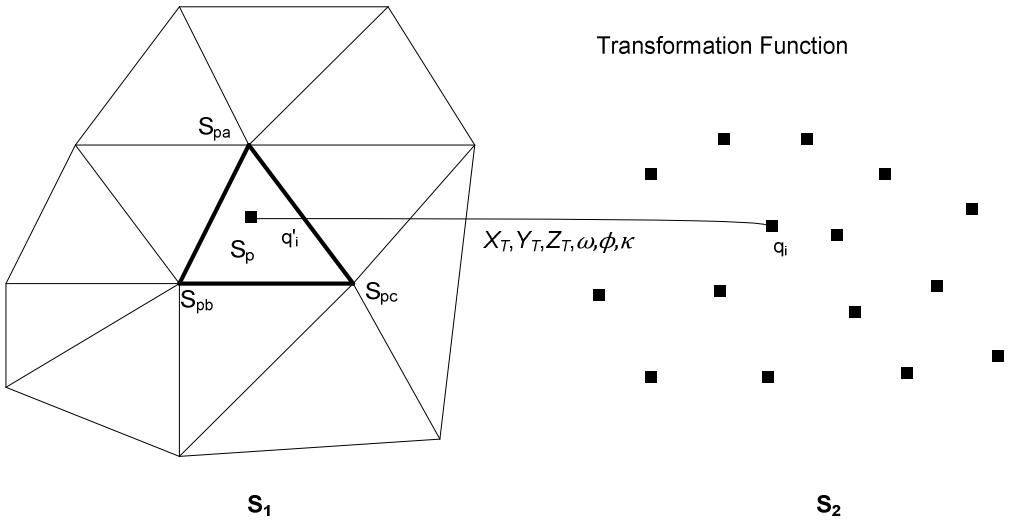

Figure 2-1 IC Patch Transformation Illustration

Figure 2-1 illustrates the IC-Patch transformation function. In essence, a TIN is matched with a group of points instead of matching points to points, or aerial and linear features to corresponding features. These TIN's may or may not exactly represent the physical structure of the ground, but for every point in the matching dataset, a rough correspondence is approximated within the TINs. A modification of the ICP algorithm is applied to the tins and patches to perform strip-adjustment.

2.3.2 Feature Based Matching

There are many feature based matching techniques in image processing, some of the algorithms use point cloud based matching, while some use feature descriptors. Zitova and Flusser [16] reviewed existing methods of image registration of multimodal images and images acquired from various viewpoints. The methods included area-based techniques – correlation based, Fourier methods and mutual information; and feature based techniques – methods involving spatial relations, descriptors, relaxation and pyramids & wavelets. It also provided surveys of the transform model estimation for linear and non-linear cases. L.G. Brown [59] contains a detailed survey of image registration techniques. In the following subsections and in Section 2.3.3, some techniques relevant to the work in this dissertation will be presented.

2.3.2.1 Neighborhood Based Matching Techniques

Point based matching techniques is best suited for matching points clouds with each other. The descriptors are based on the neighboring points of each one of the detected keypoints (features).

Johnson and Hebert [51] used this technique to match objects of varying poses. A feature point, p is defined by the orientations and distances of the neighborhood points x, to the surface normal n to the plane P. For any nearby point, o, the spin-map is the relationship between itself and the feature point, x.

$$S_o(x) \rightarrow (\alpha, \beta) = \left(\sqrt{\|x-p\|^2 - (n.(x-p))^2}, (n.(x-p)\right)$$

where, α and β are the horizontal and vertical distances of the point, x to the based point, p. The spin-image is defined as the collection of such 2D points applied to all the surface points about a feature point. Since this is relative to the plane, P and the normal, n, the spin-image is invariant to object pose. Since this dissertation focuses on raster imagery, this technique, though useful for matching point clouds, is not closely aligned to the line of work pursued in this dissertation.

2.3.2.2 Intensity Based Matching Techniques

Intensity based feature descriptors are among the most common areas of research in image matching. There have been numerous algorithms developed in this regard and the most relevant ones to this dissertation are explained briefly.

2.3.2.2.1 Scale-invariant Interest Point Indexing

Mikolajczyk and Schmid [60] introduced a new scale invariant keypoint detection technique by adapting interest points to changes in scale [61]. The interest points they detected were local extrema over scale of normalized derivatives [62]. They selected points where the Laplacian attained maximum over scales (Harris-Laplacian). Their local feature descriptors were Gaussian derivatives computed at the characteristic scale and they achieved invariance to rotation by steering the derivatives in the direction of the

gradient at the interest point [63]. This technique is extremely useful to image matching due to the fact that the descriptor can be scale-adapted and hence offer a more robust matching technique. The Laplacian blob detector is a keypoint detector in our algorithm.

2.3.2.2.2 Scale-invariant Feature Transform (SIFT)

The SIFT algorithm [11] is one of the seminal pieces of work in the field of image matching using feature descriptors. A brief stepwise explanation of the algorithm is provided here:

1. A pyramid of images is first created for multiple octaves, starting with 2 times the initial size of the image and shrinking by a factor of 2 each time.
2. Various blurring levels are applied to the octaves and subsequently a Difference of Gaussian image is obtained for each pair of scales.
3. 3D non-maxima suppression is performed on the DoG images and the strong points are selected from the surviving points that exceed a threshold.
4. The dominant orientation is obtained for the keypoint by observing the histogram of local gradients about the keypoint.
5. Finally, the feature descriptor is obtained by finding 8-bin histograms of local gradients in 16 sub-patches each having 16 pixels. The 16 8-bin histograms are concatenated after subtracting the dominant orientation to form the rotation and scale invariant descriptor.

Considering the relevance of this algorithm to the scope of work in this dissertation a deeper explanation is provided in Chapter 3.

2.3.2.2.3 Speeded Up Robust Features (SURF)

Bay, Tuytelaars and Van Gool [14] introduced Speeded Up Robust Features (SURF), an image matching technique based on integral images given by the summation of all the pixel values from the origin [point $(1,1)$] up to (x, y) for any point $P(x, y)$. They based the interest point detector on Hessian of a patch in the neighborhood on which they performed 3D non-maxima suppression. Box filters of varying sizes (a kernel used to extract a portion of an image) were applied to the same integral image in order to obtain responses at various scales for 3D non-maxima suppression. They were able to obtain the orientation of the keypoints firstly by applying Haar wavelets at the scale of the keypoint along with a Gaussian weighting window. The 2D Haar wavelet responses were then summed up with a sliding window of angle $\pi/3$ and the highest response gave the orientation.

The descriptor was created as follows: a square region about the interest point of size $20s$ was then split up into 4×4 patches of size $5s \times 5s$ each. The Haar wavelet response about the x and y directions were calculated for each patch (dx, dy) in which 5×5 points were sampled from the $5s \times 5s$ patch. A four dimensional vector was created for each patch which is the sum of dx, the sum of dy, the absolute sum of dx, and the absolute sum of dy. Therefore, the total dimensionality of this was $16 \times 4 = 64$.

2.3.2.2.4 Gradient Location and Orientation Histogram (GLOH)

Gradient Location and Orientation Histogram (GLOH) [64], an extension of the SIFT descriptor was designed to increase SIFT's robustness and distinctiveness. The SIFT-like descriptor was computed for a log-polar location grid with three bins in radial direction with the radius set to 6, 11, and 15 and 8 in angular direction, which resulted in

17 location bins. The gradient orientations were quantized in 16 bins giving a 272-bin histogram. The size of this descriptor was reduced by PCA to 128 with the covariance matrix estimated on 47,000 image patches collected from various images. The authors compared the performance of GLOH to SIFT, PCA-SIFT and other previously-introduced techniques. They generated $recall\ vs. 1 - precision$ graphs to compare performances, where $recall = \frac{\#correct\ matches}{\#correspondences}$ and $1 - precision = \frac{\#false\ matches}{\#false\ matches + \#correct\ matches}$. Their inference was that SIFT and GLOH predominantly performed better than other descriptors detector for various matching techniques, viewpoint changes, scale-changes, blur, JPEG compression and illumination changes. GLOH has an interesting binning strategy, which could be explored in the future.

2.3.2.2.5 A Summary of Other Feature Descriptor Algorithms

Apart from the aforementioned descriptors, there still exist many descriptors in literature. It is difficult to cover all of them in detail. In this subsection, an attempt is made to capture some of the popular feature descriptors.

Histogram of Oriented Gradients (HOG) descriptor [65] is a technique that is used extensively in detection of humans with pose variations. It used a descriptor based on gradient orientations much like SIFT and a Support Vector Machine (SVM) classifier to find if the subject is human or not. This idea can be extended to classes other than humans too.

The Maximally Stable Extremal Regions (MSER) algorithm [27] applied thresholds from minimum intensity to maximum intensity to an image. The intensity values corresponding to the local minima of the rate of change of the area function were selected as the final threshold which when applied to the image produced the so-called maximally stable extremal regions.

There are feature descriptors that are invariant to affine and perspective (projective) transformations. The Affine-SIFT (ASIFT) [66] algorithm simulated all possible viewing angles by sweeping the two camera axis orientation parameters namely the latitude and the longitude. For each view of the image, the SIFT descriptors were calculated. This is similar to the way rigid-body transformations are dealt with in area based techniques. Mikolajczyk and Schmid [67] introduced the Harris-Affine detector and compared it to other affine-invariant detectors in detail.

2.3.3 Feature Based Matching Techniques in LiDAR

Li and Olson [68] introduced a general-purpose feature detector based on image processing techniques. It employed Kanade-Tomasi (Shi-Tomasi) corner detector [69] and detected keypoints. The angle descriptor contained four angles, the first three were the angles formed by the extracted corner and the three point sets that are *d*, *2d* and *3d* meters away from the extracted corner, and the fourth descriptor element was the heading of the corner; i.e., the dominant orientation of the corner. The three point sets were generated by finding the intersection of circles of radii *d*, *2d* and *3d* meters. Joining actual LiDAR points then formed the line segments. Figure 2-2 shows an illustration of the formation of the descriptor.

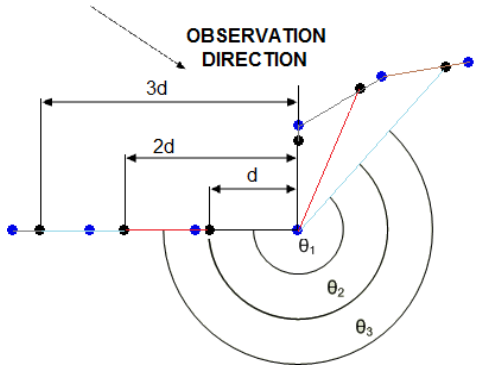

Figure 2-2 Feature Descriptor for Li and Olson Paper

Their descriptor had very good positive detection rates and lower false detection rates. They were also able to show that their descriptor had lower differences in Euclidean distances of features for correspondences and higher differences in the case of non-correspondences. They showed that their algorithm performed better than SIFT in relation to the aforementioned metrics.

Moment grid was a descriptor introduced by Zolt and Bosse [70]. The key point detector was the same as SIFT, while the descriptor was formed by evaluating eight descriptor elements that were computed from the weighted moments of oriented points (x_i, y_i, θ_i) for a total of 13 grids with the detected key point in the center. Thus the total size of the descriptor was $13 \times 8 = 104$. The aforementioned grids were of size 2×2 and 3×3 covering the same $9m \times 9m$ region.

From each of the grids, the 8-dimensional histogram that was obtained contained the following weighted moments:

$$\begin{bmatrix} \sqrt{M_{00}} \\ \bar{x} \\ \bar{y} \\ 2\mu_{11}/\sqrt{\mu_{20}+\mu_{02}} \\ \mu_{20}-\mu_{02}/\sqrt{\mu_{20}+\mu_{02}} \\ \sqrt{\mu_{20}+\mu_{02}} \\ \overline{n_x} \\ \overline{n_y} \end{bmatrix}$$

where,

$$M_{pq} = \sum_i w_i x_i^p y_i^p$$
$$\bar{x} = M_{10}/M_{00}$$
$$\bar{y} = M_{10}/M_{00}$$
$$\mu_{20} = M_{20}/M_{00} - \bar{x}^2$$

$$\mu_{11} = M_{11}/M_{00} - \bar{x}\bar{y}$$
$$\mu_{02} = M_{02}/M_{00} - \bar{y}^2$$
$$\overline{n_x} = \sum_i w_i \cos\theta_i / M_{00}$$
$$\overline{n_y} = \sum_i w_i \sin\theta_i / M_{00}$$

Once point to point correspondences were established, a transformation was obtained that registered the two datasets/strips and corrected errors in an incorrect dataset. They concluded that their feature descriptor outperformed other feature descriptors which included a modification of SIFT when it came finding a correspondence from a point's *k*-nearest neighbors. They also had a better *Receiver Operating Characteristics* (*ROC*) [71] response than the other descriptors.

While the aforementioned techniques introduced feature descriptors for terrestrial LiDAR matching, this dissertation focuses its attention on Aerial LiDAR matching. Although these techniques have advantages in matching, our approach, which creates a feature descriptor based on the statistics of the keypoints and intended for use primarily in aerial LiDAR doesn't benefit much from these techniques.

2.4 Conclusion

It should be noted that while many point cloud matching techniques exist that register overlapping LiDAR data, there are few techniques that employ a descriptor. To our knowledge there isn't any technique that uses the perfectly co-registered intensity data to match points. The aim of this dissertation is to detect keypoints in the overlapping LiDAR strips by using a combination of well-known detector schemes on both the intensity and elevation data. A new feature descriptor is presented in this dissertation; one that consists of data from the elevation and intensity information, thus leveraging as much information as possible from the LiDAR data.

Chapter 3

Elevation Based Feature Descriptor

3.1 Background

Using a feature descriptor to match data requires much preprocessing. It includes steps to resample data and detect keypoints. In this dissertation, our objective is to work with co-registered intensity and elevation from a LIDAR platform. Since the raw LiDAR data is a cloud of irregularly spaced points, a key and somewhat complicated first step is the re-scanning of the swath data to create a regularly spaced raster grid. This is followed by the actual matching algorithms.

Sections 3.2 and 3.3 contain the description of our two main matching algorithms: one that considers known relative 3D coordinates of points for matching and another that assumes that such information is not available and treats the data essentially as two images. The two algorithms have potential applications in the LiDAR and non-LiDAR domains for which each of these approaches could be valid.

3.2 Steps in LiDAR Strip Matching When Global Coordinates Are Available

In the case that geographic information about the chosen keypoints is available, it is possible to match the overlapping datasets quite efficiently by conducting a search for a keypoint's matching keypoint in the overlapping image only in the vicinity of its geographic neighborhood. The explanation for this technique and the block diagram is provided in the subsections that follow. The explanation that follows after Figure 3-1 is a technical overview while Chapter 4 contains the implementation details.

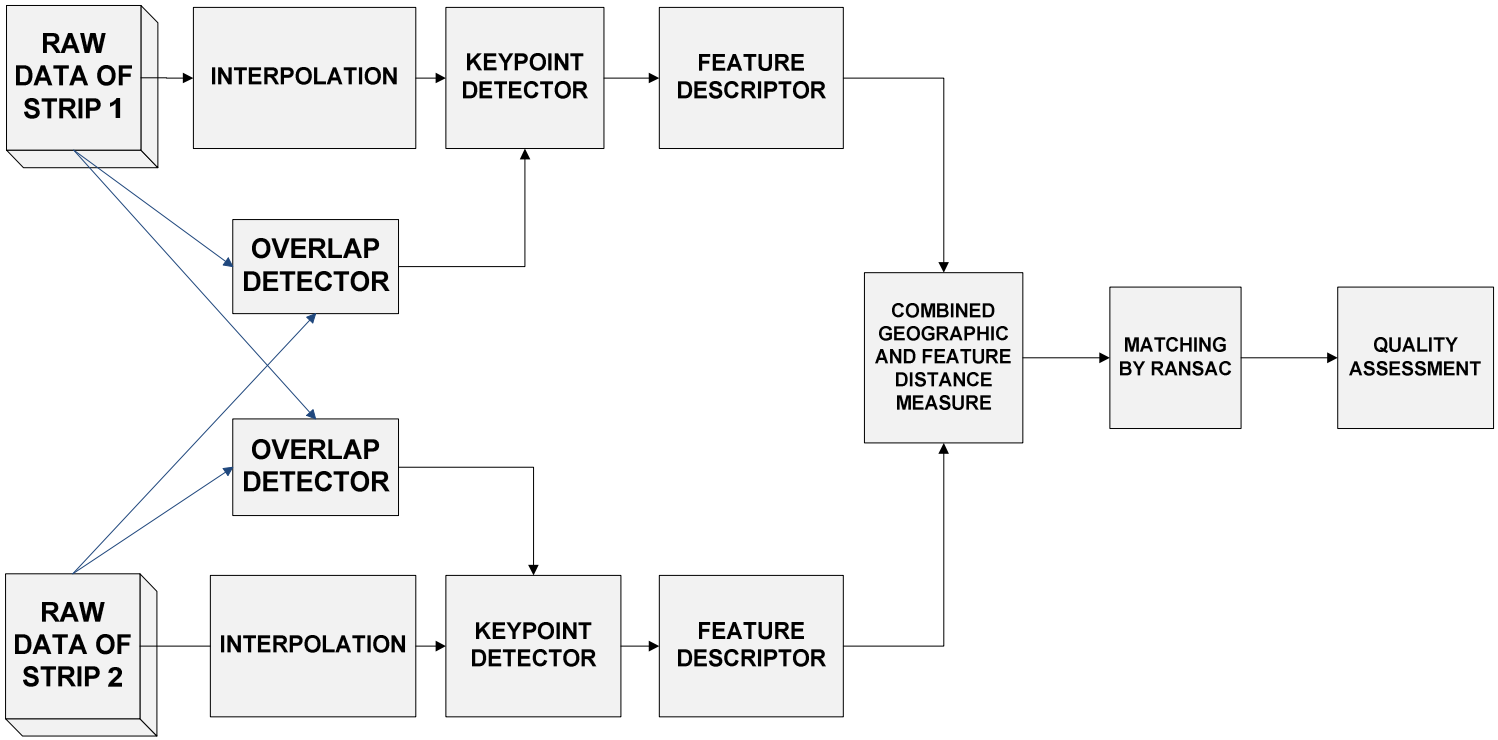

Figure 3-1 LiDAR Image Matching Block Diagram – Geographic information available

The block diagram in Figure 3-1 implements the LiDAR data matching algorithm as explained in the following steps:

1. The two strips of LiDAR data are interpolated separately and the data set to a raster grid (2.5D data).
2. The overlap regions from the perspective of both the LiDAR datasets are calculated so that keypoint are searched for only in the region of overlap.
3. Following the interpolation, the keypoints are detected in both the strips using the overlap region information.
4. The feature descriptors are calculated for each keypoint obtained.
5. For each keypoint in the first image, likely matches are searched for in its geographic vicinity and pruning is done with the feature distance for each likely correspondence.
6. Likely matching pairs are obtained and outliers eliminated using RANSAC.
7. The homography/projective transformation between the overlapping swaths is found and quality of matches is evaluated based on true and false positive and negative matches.

3.2.1 Interpolation

The LiDAR data is typically acquired on an irregularly sampled. Therefore, LiDAR data on parallel and overlapping flight paths do not contain the same points with a very high probability (close to unity). In order to match the strips, it is necessary to interpolate this data; the matches occur only in interpolated regions. Therefore, the irregularly sampled data needs to be organized on a raster grid thereby making the data ‘regular’. The importance of a good interpolation technique is crucial in obtaining good keypoint matching.

The interpolation method employed in our algorithms is the inverse distance weighting (IDW) interpolation algorithm [5]. When interpolation is performed with a distance weighting scheme, many of matrix entries are empty because of the irregular sampling grid. In order to tackle this, the interpolation algorithm was implemented using the following steps:

1. The intensity and elevation data are initialized and the centers of the matrix points are identified.
2. Each point in the LiDAR dataset is assigned to the location it occupies and the 8 neighboring locations. (Shown in Figure 3-2 below)

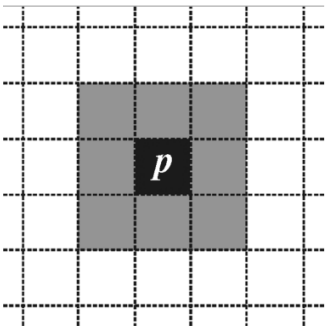

Figure 3-2 Adaptation of inverse distance weighting

3. Once each point has been assigned to its locations, the empty locations are identified for further processing.
4. For each one of these empty locations, data is “lent” from their 8-neighborhood.
5. The lending/borrowing process is repeated thrice to account for empty locations due to setting irregularly sampled LiDAR data to a regular grid. The reason why a maximum of 3 iterations is performed to avoid over smoothing. At the same time the need to interpolate points that were missed due to irregular sampling is present.

6. Once all the locations have been assigned some points, the elevation and intensity level at each location is calculated using the following formula:

$$E_{x,y} = \frac{\sum_{k=1}^{n} \frac{e_k}{d_k}}{\sum_{k=1}^{n} \frac{1}{d_k}} \quad and \; I_{x,y} = \frac{\sum_{k=1}^{n} \frac{i_k}{d_k}}{\sum_{k=1}^{n} \frac{1}{d_k}}$$

where,

$E_{x,y}$ and $I_{x,y}$ are the elevation and intensity at location (x, y).

e_k and i_k are the elevation and intensity of the individual data points that have been assigned to the location.

It should be noted that steps 1, 2 and 6 belong to the standard inverse distance weighting interpolation method. The algorithm was modified in our experiments so as to not have many "zero" entries for intensity and elevation. The proposed technique fills up empty entries by borrowing points from the neighborhood.

3.2.2 Keypoints

Keypoints are detected using two major types of algorithms. One is the corner detector and another is detectors based on scale-space blobs.

3.2.2.1 Corner Detector

Corner detector algorithms aim to find features in an image (LiDAR data in the case of this dissertation) that have a good amount of variation of the gradient in more directions than just one. A flat feature has very little or no variation in all directions, an edge has variation along one direction (perpendicular to the edge), while a corner has variation in more than just one direction. Corner features are desired for the fact that they have a great deal of robustness to viewpoint changes.

The corner score, E for each point of an image is evaluated by shifting a local window centered at the point by a small margin and the score is approximated after a Taylor series expansion by $[u \quad v]M\begin{bmatrix}u\\v\end{bmatrix}$. Here u and v are the shifts M and is the moment matrix. The moment matrix is given by:

$$M = \sum_{x,y} w(x,y)\begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}$$

where, $w(x,y)$ is a windowing function and the expression in the brackets are the products of the gradients, I_x and I_y.

Whether a point is a corner or not can be easily evaluated by finding the eigenvalues of the *M* function. If both the eigenvalues are small, it is likely to be a flat surface; if only one of the eigenvalues is large, then it is likely to be an edge in the direction of the larger of the eigenvalues; finally, for a corner, both the eigenvalues are large indicating variations in two perpendicular directions.

The Harris corner score is given by $R = \lambda_1\lambda_2 - \kappa(\lambda_1 + \lambda_2)$ which can be evaluated by $R = \det(M) - \kappa * trace(M)$, where κ, is an experimentally evaluated constant fixed at values between 0.04 and 0.15.

The Shi-Tomasi or the Kanade-Tomasi corner detector computes the minimum of the two eigenvalues and decides if a point is worthy of being a corner. This function is given by $R = \min(\lambda_1, \lambda_2)$.

There are many other ways of detecting corners in an image; however, this dissertation uses the Shi-Tomasi detector instead of the Harris corner detector because the "cornerness" criterion according to the Shi-Tomasi detector is not as stringent as the latter.

3.2.2.2 Scale-space Blob Detection

Lindeberg [62] showed that Gaussian blurring of an image provides a scale-space for the image. The Laplacian of the image provides excellent keypoints that are invariant to viewing conditions. It is possible to obtain an approximation of the Laplacian by performing the difference of Gaussian (DoG) [72] operation.

Scale-space blob detection is a technique to find blobs in an image at their characteristic scale. A blob is an artifact in an image which has fairly smooth intensity values and has high contrast with respect to its surroundings. Mikolajczyk and C. Schmid [60] introduced a technique to find the characteristic scale at which a blob attains the maxima for the Laplacian of Gaussian (LoG) operator. By attaching the characteristic scale, it is possible to identify the proper sampling scale to attach to a feature descriptor in the next steps.

The DoG [11], is very similar to the LoG operator apart from the fact that DoG creates pyramids of Gaussian blurred images at each image scale and subtracts one from its neighbor to create the DoG image as opposed to the LoG which applies the Laplacian filter to the images; then non-maxima suppression is applied in 3 dimensions to obtain the strong keypoints.

3.2.3 Feature descriptor

A feature descriptor is a unique signature of the keypoint detected. Often, in area based matching methods, the feature detection and description steps are merged into one; in certain feature based methods, salient points are extracted out of images and points clouds are matched with each other eliminating the feature description step. However, a feature descriptor is created so that it is easy to identify similarity between keypoints of two different images so that point-to-point correspondences can be obtained.

In our experiments, we have used both the SIFT and a new descriptor we have introduced, viz. ARFD, the detailed explanations for which are present in Section 3.4. Our implementation of the SIFT algorithm is a modified implementation of the original SIFT algorithm. The original SIFT algorithm matches keypoints in *two images* that have overlapping content. In our implementation of the SIFT algorithm, we consider the geographic distances for two keypoints to match, and only then we match the feature descriptors to find a nearest neighbor both in terms of physical proximity and in terms of proximity in feature space.

3.2.4 Distance Score for Keypoints in Two Images

For each keypoint detected in the first image, the neighborhood coordinates are calculated from the bounding box information for the entire strip. Using this information it is possible to look for keypoints in the overlapping image in the exact same neighborhood in a neighborhood with a radius of 0.5 pixels. Although the original LiDAR data might not have contained exact point-to-point correspondences, the rasterized version of the same is likely to contain *pseudo-correspondences* [4]. Following this, the *pseudo-correspondences* are pruned using the feature descriptor to find the best possible match for each keypoint. This process is continued and all possible matches are accumulated as *positives*.

The features are to be matched in the feature descriptor space following the neighborhood based pruning. This follows that the physical distance on the ground is not taken into consideration any longer. The Euclidean distance for any two keypoints from the two strips is given by the equation:

$$D_{i,j} = \sqrt{\sum_{k=1}^{n} (x_{i,k} - y_{j,k})^2}$$

where,

$x_{i,k}$ and $y_{j,k}$ are the entries of the k^{th} the dimension of the keypoint X_i detected in the first strip and Y_j detected in the second strip.

$D_{i,j}$ is the Euclidean distance between the two keypoints.

n is the number of dimensions of the feature descriptor.

3.2.5 Model Estimation (Homography) Using RANSAC

The homography/projective transformation is the transform matrix which when applied one strip transforms it to be stitched with the corresponding overlapping strip. In the case of the data used in this dissertation, the homography transformation is obtained by first taking the putative matches and then applying the RANdom SAmple Consensus (RANSAC) [73] algorithm to it. The RANSAC algorithm is a very useful tool in outlier elimination in any type of model fitting to data. For the case of the workflow presented in this dissertation, the following steps are undertaken to eliminate false matches:

Goal:

To fit data into a model into a dataset S that contains multiple outliers.

Algorithm:

1. A random choice of s points in the set of points in the data S is made.
2. Obtain the model for fitting the data from the s points.
3. The model/transform is applied to all the data and the number of points that fall within a threshold t is accepted as inliers to the transform.

4. This procedure is repeated N times and the points from the model that has the maximum acceptable number of points are chosen for further consideration.
5. Absolute least squares algorithm is applied to all the inliers and the robust fitting to the points is obtained by eliminating the outliers.

It should be noted that the choice of t and N determines the accuracy of the model that is obtained finally.

3.2.6.1 Homography

The homography between two view images of a scene is the transformation matrix required to align one dataset with the other perfectly. In order to obtain the homography, one requires solving for 8 unknown parameters. Having 4 point-to-point correspondences is enough to create the homography model between two views. In the case of LiDAR strip matching, more than 4 point-to-point correspondences are obtained to begin with. After applying RANSAC, there are still more than 4 point-to-point correspondences in most cases.

After the RANSAC step, the inliers must be used to obtain final homography matrix for the transformation of the strips with respect to each other. The homography between two keypoints is given by:

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\lambda X' = HX$$

Where X' is the coordinates of the point in the transformed view and X is the coordinates of the point in the original view. The H matrix is the transformation required to go from view X to view X'.

The above set of equations holds good for one set of point to point correspondence. The homography equation seemingly has 9 parameters. But the parameter h_{33} determines the scale of the transformation and making the whole matrix normalized so that $h_{33} = 1$ fixes the scale parameter. Therefore, there are just 8 parameters that must be obtained, which concurs with the necessity of having atleast 4 point-to-point correspondences.

However, in most cases of image matching, more than 4 correspondences are likely to be obtained. Therefore, after RANSAC, the least squares equation must be solved to find the best homography transform to fit the inliers. Below is a detailed explanation of solving for the least squares criteria for multiple point correspondences:

For the i^{th} point correspondence, the homography relation is given by:

$$\lambda \begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$\lambda X_i' = H X_i$$

Taking a cross product of the above term with X_i' gives a final result of 0. This can be expressed as:

$$X_i' \times H X_i = 0$$

$$\begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} \times \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} X_i = \begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} \times \begin{bmatrix} h_1^T X_i \\ h_2^T X_i \\ h_3^T X_i \end{bmatrix}$$

The cross product in the above equation can be rewritten as:

$$\begin{bmatrix} y_i' h_3^T X_i - h_2^T X_i \\ h_1^T X_i - x_i' h_3^T X_i \\ x_i' h_2^T X_i - y_i' h_1^T X_i \end{bmatrix} = 0$$

This can further be rewritten as:

$$\begin{bmatrix} 0^T & -X_i^T & y_i' X_i^T \\ X_i^T & 0^T & -x_i' X_i^T \\ -y_i' X_i^T & x_i' X_i^T & 0^T \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = 0$$

Therefore, there are 3 equations with 9 unknown parameters in the H matrix. Of these 3 equations, there are only two linearly independent equations. Thus, two equations out of three can be picked for each point-to-point correspondence and homography can be solved for. The equation for solving the $y_i' X_i^T$ matrix parameters using the least squares criteria is given below:

$$\begin{bmatrix} 0^T & X_1^T & -y_1' X_1^T \\ X_1^T & 0^T & -x_i' X_i^T \\ \cdots & \cdots & \cdots \\ 0^T & X_n^T & -y_n' X_n^T \\ X_n^T & 0^T & -x_n' X_n^T \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = 0 \; or \; Ah = 0$$

The above least squares equation can be solved by finding the h matrix that minimizes $\|Ah\|^2$.

3.2.6 Quality of Matching

Since we use the geographic distances of points along with the feature distance, we only consider points that are matches, all other pairs of keypoints between images are considered as mismatches. Therefore, the performance metric here is number of point-correspondences that were obtained for overlapping images.

3.3 Steps in LiDAR Strip Matching When Geographic Information Is Unavailable

In the case of LiDAR strip matching algorithm when geographic information is unavailable, the workflow has to be modified slightly to perform the matching tasks. Figure 3-3 contains the LiDAR image matching process for such a case. Also following the block diagram is an explanation of the blocks that are not part of the case covered in Section 3.2. These are the steps that are different from Section 3.2:

1. Following creation of the feature descriptors for each of the keypoints in either image, the feature distance scores are obtained for pairs of keypoints from both images.
2. Likely matches are obtained using a nearest neighbor distance ratio and outliers eliminated using RANSAC.
3. The homography/projective transformation between the overlapping swaths is found and quality of matches is evaluated based on true and false positive and negative matches.

Each of these steps is explained in greater detail in the following subsections.

3.3.1 Distance Score for Keypoints in Two Images – Geographic Data Unavailable

In this case, the feature distance is only calculated using the formula already demonstrated in Section 3.2.4. Here, the geographic proximity is not considered.

3.3.2 Possible Matches – Geographic Data Unavailable

Possible matches for each one of the points in the first strip are found with respect to each of the points in the second strip. These matches are found by examining the smallest distance score between the chosen point in the first strip and all the keypoints in the second strip. It is described by the following equation:

$$j^* = \min_D D_{ij}$$

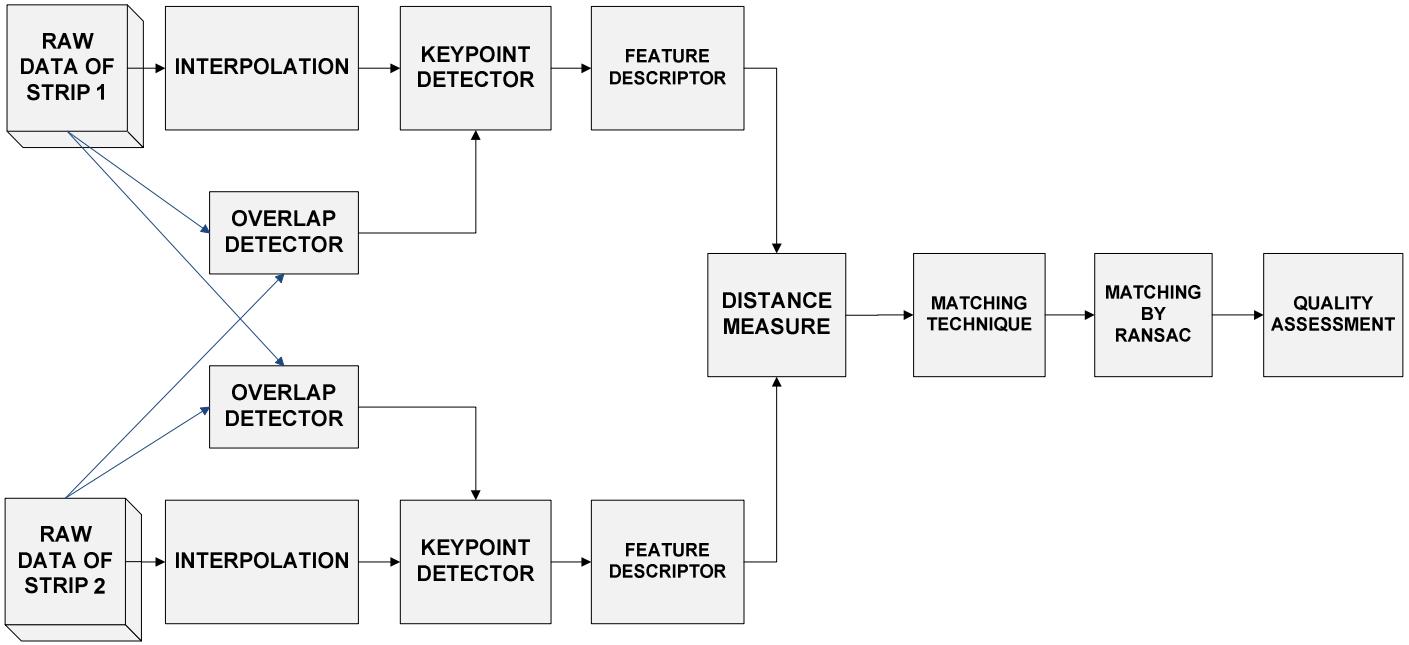

Figure 3-3 LiDAR Image Matching Block Diagram – Geographic information unavailable

where, i is the keypoint in the first strip and j is any keypoint in the second strip. The minimum distance D is obtained for j^*, the matching keypoint in the second strip.

3.3.2.1 Nearest Neighbor Distance Ratio (NNDR) Based Pruning

The NNDR is the ratio of the distance between the best match to the second best match. This eliminates the possibility of occluded keypoints in either strip matching with another keypoint in its corresponding strip. Despite the fact that the nearest Euclidean distance method should be able to pick out the best match; this is not necessarily true in the case that the keypoint in the first strip has no matching keypoints in the second strip. Therefore it is necessary to be sure about the match obtained by the nearest Euclidean distance. Therefore, the NNDR method is applied to eliminate false matches in the case of partial or significant occlusions between two datasets.

3.3.3 Homography Calculation Following RANSAC

RANSAC is applied to the positive matches from the previous step. Following this, the inliers and outliers to the model are obtained and also a homography is obtained.

3.3.4 Quality of Matching

In order to measure the performance of the elevation based descriptor and to compare it with SIFT; the natural choice considering how ubiquitous it is in the field of image matching and object recognition. The metrics that were important in this measurement were the true positives (*TP*), true negatives (*TN*), false positives (*FP*) and false negatives (*FN*). They are discussed in further detail in Chapter 4.

3.3.5 Recalculation of Homography after Obtaining True Positives and False Negatives

In the algorithm presented in this dissertation, the homography is estimated in a slightly different manner from the traditional homography estimation. After having obtained the *TP* and *FN* matches; homography estimation is run again for the *TP* and *FN* to find the exact homography with absolute certainty. Following this it is possible to estimate the mean square elevation error that gives a measure of how much alignment error exists between the two swaths of data after re-estimating the homography.

Therefore, the importance of the algorithm by itself (without recalculating homography) lies in the hope that it is able to generate a lot of top quality matches to begin with which helps in converging to an accurate homography to begin with. The more number of *TP* are with respect to the *FP*, the more accurate are the measures: *TP*, *FP*, *TN* and *FN*. The more accurate the *TP* and *FP* are, the more accurate is the final homography that is sought for the final realignment.

After having aligned the two images in the right manner, the mean square elevation error is calculated by finding the mean square error for the overlapping pixels that are both non-zero (zero indicates lack of a LiDAR scanning point or the presence of a water body).

3.4 Feature Descriptors Used in Our Experiments

3.4.1 The Scale Invariant Feature Transform Algorithm

The SIFT algorithm revolutionized invariant point matching. This algorithm is not only invariant to scale as the name suggests. It is also invariant to rotation, illumination (with slight limitations in dark lighting) and it is robust to a degree of viewpoint change. Because the SIFT algorithm was one of the first comprehensive techniques/workflows for matching images invariant to various transformations, it is considered most important in this regard. It has been adapted in multiple instances: PCA-SIFT, Affine-SIFT, GLOH and SURF, which either treat SIFT as a starting point or as an inspiration for their respective algorithms.

Given below is a detailed explanation of the SIFT algorithm in steps:

1. A pyramid of images is first created for multiple octaves, starting with 2 times the initial size of the image and shrinking by a factor of 2 each time. Typically 4 such octaves are created for the purposes of covering multiple scaling variations between images that are being matched.
2. Various blurring levels are applied to the octaves and subsequently a Difference of Gaussian (DoG) image is obtained by subtracting two neighboring images in the same octave. The DoG is an approximation of the Laplacian of Gaussian (LoG) which is useful in picking out scale-space blobs. This technique is used as one of the keypoint detector steps in this dissertation. The illustrations for points 1 and 2 are given in Figure 3-4.

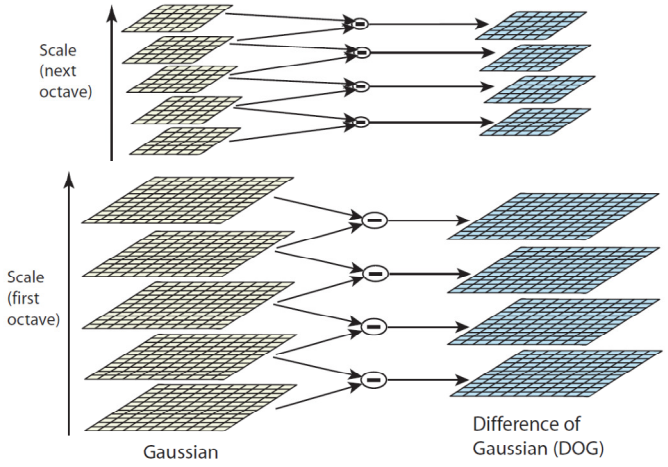

Figure 3-4 SIFT Pyramid Formation [20]

3. 3D non-maxima suppression is performed by considering the 8-neighborhood in the current scale and the two neighboring scales. The strong points are selected from the surviving points that exceeded a threshold from the 3D non-maxima suppression step. There exists an alternative implementation that includes an additional step of accurately finding the location of a keypoint by a Taylor expansion of the 3D scale-space. Finally, some of these points are eliminated on the basis of an intensity threshold. The end

result of this step is that a set of good keypoints are generated mostly in an automated fashion.

4. The dominant orientation(s) of each keypoint is (are) obtained by observing the histogram of local gradients about the keypoint. In order to obtain the keypoint orientations, the neighborhood about the keypoint is first extracted (16×16 patch). Then, the gradient magnitudes and orientations are obtained for the keypoint locations by resampling the image based on the scale obtained for the keypoint. Finally, the orientations are set into a 36-bin histogram and peaks within these are considered as keypoint orientations. The peaks are obtained by examining the bin belonging to the peak and the two neighboring bins for the peaks and by fitting a parabola to the three bin values to find the actual peak orientation. This step mitigates the effect of binning the 360 degrees into just 36 bins.
5. Finally, a feature descriptor (shown in Figure 3-5) is obtained by finding 8-bin histograms of local gradients in 16 sub-patches each having 16 pixels. The gradient magnitude of the patch is weighted by a Gaussian function with a standard deviation σ equal to one half the width of the descriptor window. Following this, an 8-bin histogram is created from the gradient orientations by using the weighted magnitudes as a further weighting function. These 8-bin sub-patch histograms are concatenated after subtracting the dominant orientation to form the rotation and scale invariant descriptor. The result is the final SIFT descriptor which is invariant to scaling, rotation, illumination and robust to a degree of viewpoint change.

The SIFT algorithm went in to much details to capture the uniqueness of a given key point in generating a corresponding descriptor for it primarily for optical image data. However, it could be argued that the SIFT algorithm can be applied to any raster data. This possibility has allowed us to keep some aspects of the overall structure of SIFT such as the scale-space blob

based detector yet, entirely change the descriptor design. This flexibility has also allowed us to incorporate and even integrate multiple image modalities – specifically, the integration of registered intensity and elevation from a modern LiDAR sensor. Additional subtler changes in details of implementation have resulted in nuanced but significant alteration to the original algorithm and improvements in the performance of the corresponding matching algorithm. We will restate this point later with greater clarity when we describe our results in Chapter 5 and Chapter 6.

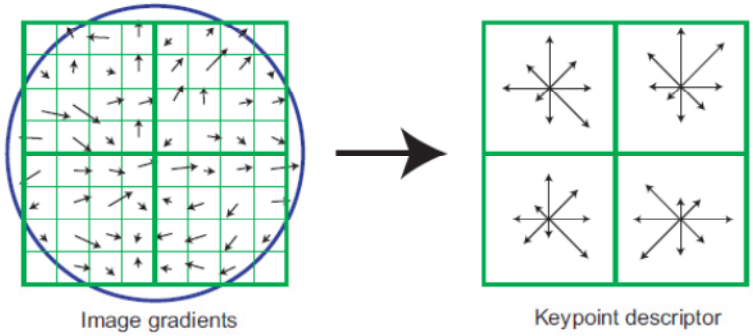

Figure 3-5 SIFT Descriptor [20]

3.4.2 The Aerial Range Feature Descriptor (ARFD)

The ARFD evaluates the statistics elevation data in the neighborhood of the detected key point location. These statistics are the mean, variance, median and the range of data. Since there are no actual point-to-point correspondences in the original LiDAR data, the statistics of the sub-patches are slightly different even for matching keypoints. Despite this, due to the robustness of the matching algorithm, the slight differences don't offset the performance of the algorithm in selecting strong correspondences. Compressing the sub-patch data into a 4 dimensional descriptor reduces the exactness required for match and also achieves speed-up

compared to direct patch data. It is an adaptation of the well-understood technique employed in SIFT [11], GLOH [71] and PCA-SIFT [74].

Elevations of points on earth are invariant to changes in viewing conditions or viewpoints. A strong feature descriptor must use the elevation values or a compressed version of the same in the neighborhood of the keypoint. Therefore, elevation points on a 16×16 grid about the key point are extracted and the sub-patches are used to create local signatures based on invariant statistics of elevation data.

The following are the steps applied to evaluate the invariant descriptor:

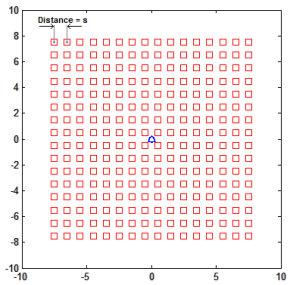

Figure 3-6 Sampling the keypoints

1. For every key point location detected by the detector algorithm, in the corresponding elevation data, a 16×16 patch about the key point location is extracted based on the scaling factor as shown in Figure 3-6. This patch is subdivided into 16 sub-patches of size 4×4 each. These are extracted starting at the top left, going all the way to the bottom right. The process is shown in Figure 3-7.
2. A 4 dimensional (4D) descriptor is created using the 16 samples in each of the sub-patches. The descriptor of choice from the above three types is chosen to be created from the sub-patch information. The details of the variants are presented in Section 3.5.1 titled *Variants of Sub-patch Descriptors*.

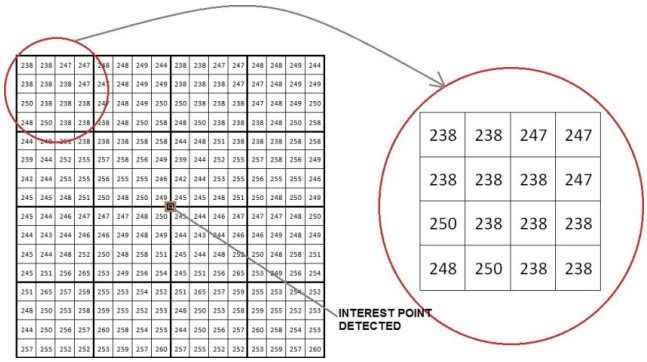

Figure 3-7 Extracting Sub-patch Information

3. The data from the 16 sub-patches are concatenated together to form the 64 dimensional descriptor. This is the unique signature of each keypoint that can be used next in the matching stage.

Given the probabilistic nature of the problem, the best patch size and the number of dimensions needed in each patch, have been experimentally determined. Since the process is highly non-linear, a combination of local statistics and adaptive parameters is desired (number of samples per patch & number of patches) for achieving the goals.

3.5 Variants of the Feature Descriptors Used in Our Experiments

3.5.1 Variants of ARFD Sub-patch Descriptors

In order to create the feature descriptor, a few types of descriptors were considered. The feature descriptor must be designed in a fashion that allows for robustness to irregular sampling differences between the two flight swath datasets, at the same time it must be efficient in ignoring false matches that are slightly similar to each other but in reality are mismatches. Therefore, the following types of descriptors were considered for use in the ARFD:

1. Histogram of sub-patch data: The histogram of the sub-patch puts into 4-bin the values read from the 16 samples. This is very useful as a descriptor in the context of SIFT wherein the gradient orientations (after correcting for keypoint orientations) are used as a descriptor invariant to rotations and illumination changes, however in

the context of elevations or simply LiDAR illuminations it may lead to misleading values descriptor elements. This can be shown with an example for a scaled-down version of the descriptor. Consider a 2-bin descriptor built out of 4 sub-patch elements. If the 4 sub-patch entries are {0, 15, 15, 2000}, they form the same histogram as {199,199,199,401}. This is potentially harmful for matching identical keypoints. Therefore, after some experiments, we decided not to use this feature descriptor.

2. Lower order statistics of sub-patch elements: A consideration was made for using lower order statistics like the mean, variance, median and range of data (difference between maximum entry and minimum entry) as the statistics. This method addresses the problem faced by using just a histogram to create feature description. The mean, median and range are robust to changes in sampling while the variance is a good description of the sub-patch statistics. The lower order statistics, although they capture some information about a sub-patch, don't contain enough depth to form a unique enough descriptor of a sub-patch. Although it worked satisfactorily for the set of images we used in our experiments, we decided not to use this in the standard form of our algorithm.
3. A mix of lower order and higher order statistics: A mix of the lower order statistics, the mean and variance and higher order statistics the *skewness* and *kurtosis* were considered for an alternate descriptor. The mean is quite robust to sampling variations and the variance, as mentioned in the previous descriptor type, is a good descriptor for the amount of variation in the data. The *skewness* and *kurtosis* capture the skew and the variation of variation or the "peakedness" in the sub-patch distribution. This was used as the feature descriptor in all our experiments and it performed very well because of the multiple levels of moments it contains.

3.5.2 Variants of Experiments Common Both to SIFT and ARFD

The SIFT and the ARFD descriptor based matching algorithms were implemented by varying them in order to maximize their utility. The variations included choice of keypoint detectors as mentioned earlier. The following subsections contain details about these variant that were implemented in the work related to this dissertation.

3.5.2.1 Use of SIFT and ARFD on Standard LiDAR Elevation Data

The SIFT and ARFD algorithms were implemented by using the elevation data (the *Z*-values) from the LiDAR points. In these implementations, the keypoints and their descriptors that were generated were based on the LiDAR elevation data. Having done the aforementioned step, a comparison with LiDAR intensity was made possible

3.5.2.2 Use of SIFT and ARFD on Standard LiDAR Intensity Data

The SIFT and ARFD algorithms were implemented by using the intensity data (the *I*-values) from the LiDAR points. In these implementations, the keypoints and their descriptors that were generated were based on the LiDAR intensity data. The idea behind considering the intensity data despite the fact that it is possible that it can be rendered inaccurate by specular reflections is that it is possible to get some strong keypoints from the visual information present in the data. Elevations do supply some interesting points, but it is likely that variations in intensity values are more common in nature than variations in elevation values. . Therefore, to test this theory, some experiments were performed on these lines and the results presented in Chapter 5 and Chapter 6.

3.5.2.3 Modified Versions of SIFT and ARFD

In this case, the emphasis was to choose as many keypoints as possible from a combination of the elevation and intensity data. Following this, the standards SIFT or ARFD

descriptors were created for every keypoint strictly on the basis of whether the elevation or the intensity had a better *kurtosis* measure. The *kurtosis* is the normalized version of the fourth central moment of a distribution and the measure finds the variation of variation. Therefore, if a portion of land on which the analysis were being performed had a lot of variation in intensity whereas its elevation remains constant, in such a case the intensity data is chosen to create the descriptor. In the other case, the elevation data is chosen to create the feature descriptor. This allowed the algorithm to describe each keypoint in the most unique way possible and hence the matching was stronger.

3.6 Conclusion

In this chapter, the ARFD algorithm was presented for matching of raw LiDAR data strips with one another. The workflow included conversion of raw LiDAR data into a rasterized matrix form followed by keypoint extraction, feature descriptor creation and the matching paradigm. In order to evaluate the quality of the algorithm, the inliers and outliers in both the possible matches and the mismatches discarded by the algorithm was analyzed. In Chapter 4, the details of the implementation are presented for each of the blocks and the algorithms that the ARFD method is being compared with.

Chapter 4

Details of Implementation of ARFD and Modified SIFT

4.1 Implementation Platform

The algorithms were implemented on MATLAB 2013a mainly with the aid of the Image Processing Toolbox. The computer on which the algorithm was implemented had an Intel i7 processor with 12 GB RAM and a Windows 7 64-bit operating system.

4.2 LiDAR Dataset

The dataset used for the experiment was obtained from the National Resource Conservation Service (NRCS), a part of the US Department of Agriculture. The LiDAR data was obtained from Saginaw Bay, Michigan. A set of 9 flight swaths was made available in raw flight strip data. Since each flight swath covered a large geographic area, small sub-patches of rectangular shapes were extracted from each of the overlapping patches. This data was extracted using commercial GIS software from ESRI inc. and GeoCue Corporation and it was converted into an ASCII file of the X, Y, Z and I data. The swaths were in the east-west directions and there were 8 overlapping areas in the 9 swaths. Seven such verticals with a total of 56 patches were obtained. Certain patches were mostly flat with no trees or discernible height differences. This, coupled with the fact that these patches were smooth even in the intensity data did not allow us to obtain matches for those patches using the algorithm proposed in this paper or the SIFT algorithm. Therefore, not all the patches were well suited for obtaining keypoints or for image matching. However, in Chapter 5 and Chapter 6, we have presented results for all possible overlapping patches, including the ones that were predominantly flat.

Figure 4-1 shows the strip extraction process in detail. Each vertical has 9 overlapping strips and therefore 8 pairs of overlapping strips in the ideal case (some swaths were smaller than the others thereby offering lesser number of strips). From each of these strips, the

bounding box that corresponds to the range of the verticals was applied to the LiDAR data and points extracted from them for further processing.

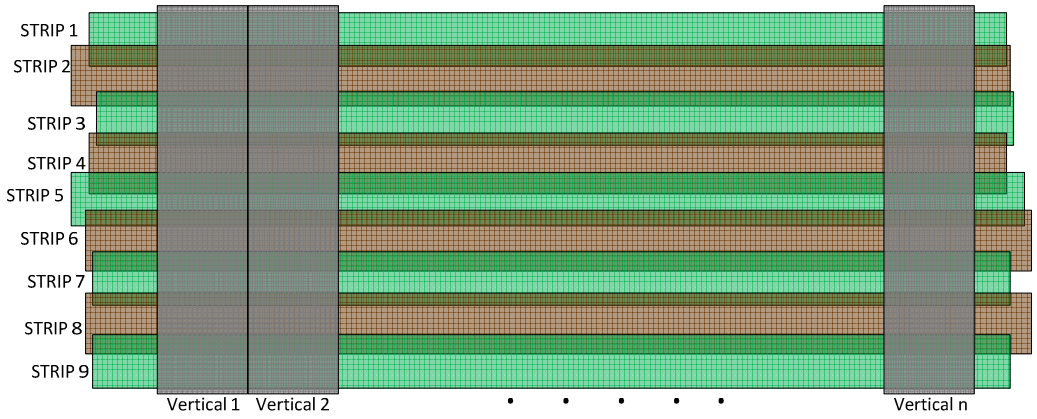

Figure 4-1 Patch extraction from individual strips

4.3 Approach to Workflow Validation and Comparison of ARFD and SIFT

The data for each individual patch was interpolated and the matching was performed on sets of two overlapping patches. As mentioned before, each set used two keypoint detection schemes. The two detectors were also combined as a third option in a set of experiments. Thereafter, both the ARFD and SIFT were applied to the datasets to compare the performance of the descriptors. SIFT was used for comparison because: 1) SIFT is presently considered the benchmark feature descriptor, 2) the scale-space blob detector used in ARFD is based on detecting maximums of LoG, which is similar to the DoG filter used in SIFT.

4.4 Implementation Details

This subsection contains the implementation details of the ARFD and the modified SIFT algorithm. The entire algorithm is discussed block by block in detail barring the actual code. In order to talk about the implementation, it is necessary to take another look at the block diagram of the algorithm.

The feature matching algorithms are shown in Figure 3-1 and Figure 3-3. Firstly, the raw LiDAR strip data is simultaneously fed into the interpolation block and the overlap detector blocks. Following this, the keypoints are detected in both the strips and the ARFD is calculated for each one of those keypoints. For each one of the keypoints detected in the first strip, the distance score for all the keypoints in the second strip is obtained. These distances are used to find the first and the second nearest neighbors. Then, the NNDR method is applied to eliminate possible false matches arising from self-similarity within the images. The final homography matrix is obtained for the points by applying the RANSAC algorithm. Following this step, the quality parameters are obtained for further comparison with SIFT. The implementation details of each of the steps in the block diagrams are provided here in the following subsections:

4.4.1 Obtaining the LiDAR Data in Text Format

The raw LiDAR strip data is available in a '.LAS' format. All the strips from the Saginaw Bay, MI data were loaded onto the ArcGIS platform with the LP360 extension. The loaded data was available for a very wide strip of ground. The polygon cutting tool was used to extract patches out of each LiDAR strip for a reasonable aspect ratio of the raster grid. Once the polygons were overlaid on each individual strip, the '.LAS' data was converted into ASCII format and the result were written onto a '.txt' file with only the *X*, *Y*, *Z* & *I* data. This raw LiDAR data in the text format was supplied to both the Interpolation block as well as the overlap detector block.

4.4.2 Interpolation of LiDAR Strip Data

The interpolation of the LiDAR strip data immediately follows the extraction step from the .LAS format to the .txt format. In order to interpolate the data, the bounding box of the *X* and *Y* fields are calculated. Once this is done, the extent of the raster grid is calculated as follows: the floor value of the leftmost entry is used to find the coordinates of the leftmost matrix entries,

the ceiling value of the rightmost is used to find the rightmost pixel entries and a similar procedure is used for the top and bottom matrix entries. Once the above operations are done, the points in the LiDAR sequence are assigned to the nearest matrix coordinate centers along with the inverse of the distance.

As mentioned before in the algorithm. The empty locations are assigned values by "borrowing" from its neighbors and this is repeated for three iterations to take care of bad sampling. This process is not repeated more than thrice because water bodies have no return values and repeating this process till each entry of the matrix fills up with non-zero values incorporates errors in datasets that contain water bodies.

Once all the points have been assigned to the relevant matrix location centers along with the distances, a weighted average is applied to each of the matrix positions based on the inverse distance to the actual LiDAR point location. Therefore, the interpolated intensity and elevation data is obtained by the slight modification to the IDW algorithm.

4.4.3 Overlap Detector

The overlap detector is essentially one block that takes in the extent information from both the LiDAR strip data and converts them into finding the region in strip 1 which has an overlap with strip 2 and vice versa. The various configurations in which the strips can overlap are shown in Figure 4-2:

The configurations in Figure 4-2 have an overlap region in the horizontal (east-west) direction and in Figure 4-3 have an overlap in the vertical (north-south) direction. To find the overlap boundaries for each of the strips involves, the exact area of the overlap region in the coordinate system of the other strip must be considered.

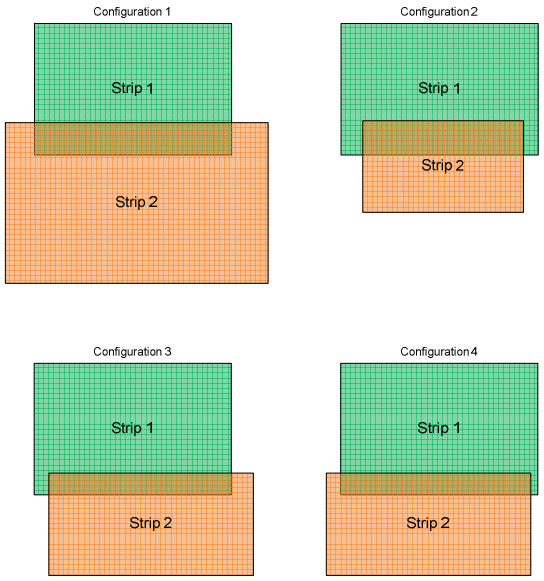

Figure 4-2 Strip Overlap Configurations Horizontal Overlap

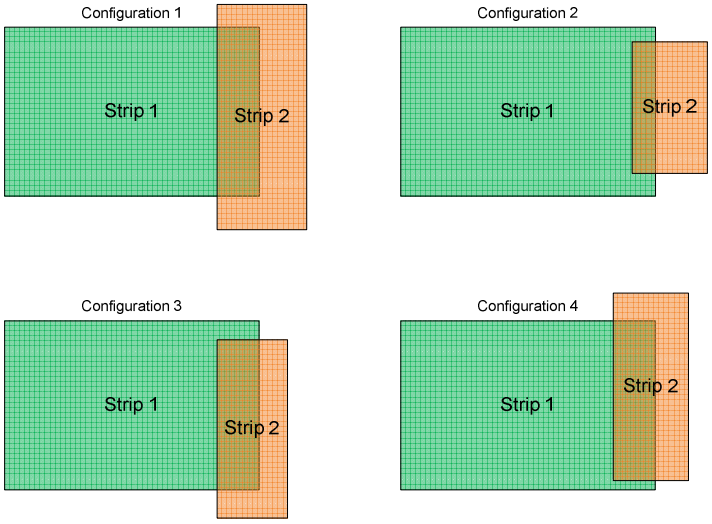

Figure 4-3 Strip Overlap Configurations Vertical Overlap

1. Left Boundary: In configurations 1 and 4 shown in Figure 4-2, the left boundary of strip 2 is outside the left boundary of strip 1. In configurations 2 and 3, the left boundary of strip 1 is outside of the left boundary of strip 2. For Figure 4-3, the left boundary of overlap in strip 1 is simply the left boundary of strip 2 and in the case of strip 2 it is the

left boundary of strip 2 itself. In the general case, the left boundary is calculated as the maximum values of the left boundaries of both strips in their respective coordinate system.

2. Right Boundary: The right boundaries are calculated similar to the left boundaries except that the minimum right boundaries of both strip 1 and strip 2 are taken as the right boundaries for the strips.
3. Top Boundary: The top boundary is calculated as the maximum value of both the strips' top boundaries in their respective matrix locations.
4. Bottom Boundary: The bottom boundary is similarly the minimum value of both the strips' bottom boundaries in their respective coordinate systems.

Since the strips considered for the experiments conducted were aligned either in the east-west or north-south directions, the possible overlaps in diagonal directions were not considered. In a general case, the calculation of the exact overlapping region is harder than the configurations considered above.

4.4.4 Keypoint Detector

The keypoint detectors considered in this dissertation were the Shi-Tomasi corner detector and the Scale-Space (LoG) blob detector. These were implemented as follows.

4.4.4.1 Shi-Tomasi Detector

The Shi-Tomasi detector was implemented using a modification of the Harris corner detector. The Harris corner detector used a mask to first calculate LiDAR elevation data's derivatives in the *X* and *Y* directions, I_x and I_y respectively. The masks used are shown below:

$$dx = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \text{ and } dy = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

After finding derivatives I_x and I_y, a Gaussian filter was applied to these derivatives based on the blurring standard deviation, σ. Then, for every matrix location in the resulting gradient function, the eigenvalues of the local neighborhood were obtained and the criterion to qualify as a corner was determined by the Shi-Tomasi function which requires the minimum of the two eigenvalues to be over a threshold. Since the Shi-Tomasi detector's implementation is not scale adaptive, a scaling factor was chosen as 4 after experimentation to be passed on to the descriptor stage.

4.4.4.2 Scale-space (LoG) Blob Detector

The scale-space blob detector was implemented by applying the LoG filter to the LiDAR intensity data at various filter scaling factors. One thing to consider was the fact that in a LoG detector, the higher the standard deviation of the blurring filter, the weaker the response. Therefore, each filter kernel was magnified by the square of the standard deviation to normalize the responses at any level. The collection of the responses of the intensity data to the filters formed the scale space. Once the scale-space was obtained, 2D and 3D non-maximum suppression was carried out to find the locations of the blobs at various levels. The level at which the blob was detected was the scaling level of that particular keypoint detected. This procedure is approximated very well by the SIFT keypoint detector. In order to visualize the blobs, a further scaling factor of 1.5 was applied to the circle display routine.

4.4.5 Feature Descriptor

4.4.5.1 ARFD

The ARFD descriptor was implemented by taking into account the scale passed on from the keypoint detection stage. Based on the scale, a sampling grid was created in the neighborhood of each keypoint detected. A 16×16 grid was sampled about the keypoint to create the final descriptor. Since the LiDAR elevation data was discreet and set on a regular

sampling grid, based on the scaling factor, it is required to interpolate the data points if the sampling locations are non-integer. The MATLAB function 'qinterp2' was used to sample it as mentioned. The final descriptor was created by recoding the statistics for each 4×4 sub-patch about the detected keypoint and concatenating them in row-order.

4.4.5.2 SIFT

The SIFT descriptor used in this dissertation was downloaded from Professor Svetlana Lazebnik's webpage [75]. This was the implementation for the stand-alone descriptor without the pre-processing steps of scale-space creation, keypoint detection or orientation assignment. The reason for skipping these steps is because the workflow in this dissertation already generates keypoints before the descriptors are created and the LiDAR elevation and image data is "upright" meaning the orientations are the same allowing for an easier implementation of feature descriptors.

4.4.6 Distance Evaluation

4.4.6.1 Geographic Distance Evaluation

The matching algorithm which has access to geographic information (world coordinate reference) can be used to find the approximate geographic location of every pixel in every image. All that is required for this step is the bounding box information for the strip being processed so that the offsets can be used to calculate the geographic location based on the reference system. Matching is performed by checking for a geographic proximity of 0.5 pixels or lesser to choose a set of points as *pseudo-correspondences*.

4.4.6.2 Feature Distance Score

The Euclidean distance scores for pairs of keypoints in both the LiDAR strips were calculated in the space of the feature descriptor into a $M \times N$ matrix where M was the number of keypoints in the first strip and N was the number of keypoints in the second strip.

4.4.7 Nearest Neighbor Matching and Nearest Neighbor Distance Ratio (NNDR) Based Outlier Elimination

The distance scores matrix was used to find the nearest neighbor for each keypoint in the first LiDAR strip within all the points in the second strip. The outliers were eliminated by using a NNDR value of 1.5.

4.4.8 RANSAC Based Outlier Elimination

RANSAC was applied to this function using the off-the-shelf implementation of Peter Kovesi [76]. The implementation took all the correspondences available after the initial outlier elimination step performed by the NNDR criterion and used randomized sets of 4 points to create a projective transformation. Once the transformation was created, the number of inliers was counted and this process was repeated until the projective model (homography) with the maximum number of inliers was obtained. Finally, with the points that fit this projective model, a final homography matrix was obtained using the least squares technique.

4.4.9 Performance Metrics

The performance metrics were obtained as mentioned earlier. After the inliers were obtained from RANSAC, the homography matrix was applied to both the negative samples and positive sample. Negative samples being the points that were classified as mismatches and positives the ones that were classified as true matches. Following this step, the false positives were found by applying a pixel distance to the transformed positive points and rejected a

positive as a false positive if it was beyond 2 pixels away from its correspondence. As for the negative matches, the same homography was applied and the pixel distance checker was used to see if a negative was a true negative. If a negative was less than 1 pixel away from its correspondence, it was taken to be a false negative.

From the parameters: *TP*, *FP*, *TN* and *FN*. The *precision*, *accuracy*, *TPR* and *FPR* were obtained too.

4.5 Details of Performance Metrics

As discussed before in Chapter 3, the performance metrics that were introduced were the true positives, false positives, true negatives and false negatives. Given below is a detailed explanation of how these metrics were obtained from the experimental dataset.

4.5.1 True Positives (TP)

These are the points picked out by the matching scheme that are real correspondences. The feature descriptors were obtained for each of the points on both strips. From a reference strip, a nearest neighbor was obtained in the matching strip. After the matches were eliminated based on a nearest neighbor distance ratio, a set of putative matches (correspondences) were obtained. These putative matches were pruned by the RANSAC algorithm to obtain the homography transformation required to transform the matching strip to the reference strip for registration. The points that transformed to fit the homography were designated as the true positive matches.

4.5.2 False Positives (FP)

These are the points picked out by the matching scheme, which are in reality mismatches. They are called type I errors in the field of statistics. The putative matches are

picked out by the matching algorithm based on the feature descriptor. The matches which did not fit the homography calculated in the above step were the false positives.

4.5.3 True Negatives (TN)

The points that are eliminated by the matching scheme which are actual mismatches are the true negatives. The true negatives were found by fitting the homography transformation to all the negative points. The points that don't fit the transformation were labeled true negatives.

4.5.4 False Negatives (FN)

These are the correspondences that were missed by the matching scheme but were, in reality matches. These correspondences were obtained by fitting a homography to all the negatives and obtaining the points that fit the homography.

4.6 Additional Quality Measurement Parameters

From the above data, a few more measurement parameters can be obtained to illustrate the strength or weakness of an algorithm. The parameters that were obtained for each run of the algorithm were as follows:

4.6.1 Precision

This is the fraction of the putative matches, which are the relevant matches.

$$precision = \frac{TP}{TP + FP}$$

An ideal matching algorithm has $precision = 1$.

4.6.2 True Positive Rate (TPR)

The TPR, which is also known as sensitivity or recall is the ratio of the number of true positives to the total number of actual correspondences.

$$TPR = \frac{TP}{TP + FN}$$

An ideal matching algorithm has $TPR = 1$.

4.6.3 False Positive Rate (FPR)

The FPR is the ratio of the number of false positives matches to the total number of actual non-correspondences.

$$FPR = \frac{FP}{FP + TN}$$

An ideal matching algorithm has $FPR = 1$.

4.6.4 Accuracy

This is the proportion of true results reported by the algorithm, be it positive or negative. It is given by the equation:

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

An ideal algorithm has $accuracy = 1$. The accuracy is a rough measure of how good the algorithm performs in eliminating false positives and negatives.

4.6.5 Mean Square Elevation Error (MSEE)

The mean square error for the elevation is calculated by applying the homography transformation to the second rasterized data and calculating the difference in the values of elevation in the pixels that undergo an overlap. In the case of pixels in either dataset not having a valid value (for instance, a zero value due to lack of LiDAR returns), the mean square error didn't take those pixels into account for the final calculation of MSEE. Elevation was chosen

instead of intensity for MSEE calculation because the intensity data, as mentioned before, contains much spurious information such as specular reflection either on land or in water.

It can be said further that because of the nature of the error in the intensity data, some of the intensity maps show drastic changes within the overlapping areas. Also, the error is of a non-linear and non-deterministic nature. Taking all of this into account, it can be postulated that applying standard feature descriptor techniques to the intensity data alone for matching can lead to significant false matches.

4.7 Conclusion

In this chapter, detailed explanation has been provided about the implementation details of the algorithm. Appendix A contains the MATLAB code for each of the blocks mentioned in the implementation details. In Chapter 5 and Chapter 6, discussions of results obtained from the implementations of the techniques are presented for the following cases: a) one that assumes that a common coordinate system is not present b) and one that assumes that there is a common coordinate system.

Chapter 5

Performance of the Feature Descriptors for LiDAR Matching for Unknown Search Area

5.1 Background

In this chapter, the results presented are for the case where we do not take into account the geographic positions of the keypoints for matching. By this we mean that when a keypoint is picked in one image, we do not assume to know even the rough location (X, Y or Latitude and Longitude) of that keypoint in the second image. This is the general situation in many cases, when the intensity and the elevation data are known, but the *x* and *y* locations of the matching points are unknown ahead of time. For instance, the *x* and *y* locations are unknown when a digital elevation map is created from the overlapping intensity data. In a well run LIDAR mission, generally the two sets of (*X*, Y) are fairly close. However, even in this case, calibration errors could preclude the assumption that the two sets of (X, Y) are identical. Therefore we believe that the results presented in this chapter are relevant to many applications in real life.

The experiments were conducted for three different scenarios

1. the elevation data alone was used for a descriptor for each keypoint
2. the intensity data alone was used for a descriptor for each keypoint and
3. an adaptive version of the algorithms, which used the kurtosis of a patch to determine which dataset to choose, between the elevation and intensity data for a descriptor for each keypoint.

All of these experiments were conducted for 42 overlapping strip pairs using the Shi-Tomasi detector and the scale-space blob detector as keypoint detection techniques.

Two different feature-matching algorithms were used:

1) The SIFT and
2) The ARFD

5.2 Performance Comparison of SIFT and ARFD Using LiDAR Elevation Data

5.2.1 Parameters of the Keypoint Detector

In this set of experiments the SIFT and ARFD algorithms were implemented using keypoints from scale-space blobs. In the scale-space blob detector, the minimum and maximum standard deviations σ for the scale pyramids were set between 0.01 and 30 and there were 20 levels between the ranges specified. The threshold for accepting a keypoint was set to 0.5 in order to detect blobs in all possible sizes since aerial data could possibly have blobs covering that entire range and beyond.

5.2.1.1 Performance of the SIFT Descriptor for Matching of Elevation Data

The SIFT with elevation data alone with a scale-space blob detector had the following performance: it had an average *precision,* $\frac{TP}{TP+FP} = 0.4930$, an average *TPR,* $\frac{TP}{TP+FN} = 0.4637$, an average *FPR,* $\frac{FP}{FP+TN} = 0.0571$ and an *accuracy,* $\frac{TP+TN}{TP+FP+TN+FN} = 0.8901$. After having obtained these metrics, the homography was recalculated for the datasets from the *TP* and *FN* matches. The Mean Square Error (*MSE*) was calculated for the resampled and overlaid data. The elevation *MSE* was found to be 0.0848 and the intensity *MSE* was found to be 0.1664. The SIFT algorithm took an average time of 36.0915 seconds to match.

5.2.1.2 Performance of the ARFD Descriptor for Matching of Elevation Data

The ARFD descriptor when used with elevation data alone with a scale-space blob detector had the following performance: it had an average *precision,* $\frac{TP}{TP+FP} = 0.8139$, an average *TPR,* $\frac{TP}{TP+FN} = 0.4162$, an average *FPR,* $\frac{FP}{FP+TN} = 0.0238$ and an *accuracy,* $\frac{TP+TN}{TP+FP+TN+FN} = 0.8887$. After having obtained these metrics, the homography was recalculated for the datasets from the *TP* and *FN* matches. The Mean Square Error (*MSE*) was calculated for the resampled and overlaid data; the elevation *MSE* was found to be 0.0663 and the intensity

MSE was found to be 0.1032. The ARFD algorithm took an average time of 35.9232 seconds to match.

5.2.1.3 Performance Comparison of the ARFD and SIFT Descriptors for Matching of Elevation Data

Table 5-1provides a comparison between the SIFT and the ARFD algorithms for matching data using the scale-space blob detector. It can be seen from the table that the rate of *False Positives* (*FP*) in the ARFD is much lower on average in comparison with the SIFT algorithm. This therefore increases the *precision* of the ARFD descriptor by 65% and lowers the *FPR* by 58% over the SIFT descriptor. The *TPR* is lower by 10% in ARFD over SIFT and the *accuracy is lower* by just 0.1%. The slightly worse performance of ARFD in the aforementioned is more than offset by the much better performance of ARFD over SIFT in *precision* and *FPR* and by the excellent performance in the *MSE* for both the intensity and elevation data alignment.

Table 5-1 Performance comparison of SIFT and ARFD for elevation data with Scale-Space blob keypoints

Average Performance Metrics	SIFT Descriptor	ARFD Descriptor	Percentage improvement of ARFD over SIFT
Precision	0.4930	0.8139	+65
TPR	0.4637	0.4162	-10
Accuracy	0.8901	0.8887	-0.1
MSE Elevation	0.0848	0.0663	+22
MSE Intensity	0.1664	0.1032	+38

5.2.2 Performance of Elevation Data When Keypoints Were Obtained from Shi-Tomasi Corner Points

For these experiments we used the Shi-Tomasi detector with parameters set as follows: the blurring standard deviation $\sigma = 1$, the threshold for detection $th = 1$ and the *radius* of detection for the non-maximal suppression as 2.

When the keypoints were obtained from the Shi-Tomasi corner detector, the SIFT algorithm didn't perform well because SIFT traditionally uses a scale-space technique to detect blobs. Hence a comparison with SIFT is not being made here. However, the ARFD performed well when used with this technique. The performance of the ARFD detector is as follows: it had an average *precision,* $\frac{TP}{TP+FP} = 0.7388$, an average *TPR,* $\frac{TP}{TP+FN} = 0.5036$, an average *FPR,* $\frac{FP}{FP+TN} = 0.0335$ and an *accuracy,* $\frac{TP+TN}{TP+FP+TN+FN} = 0.8997$. After having obtained these metrics, the homography was recalculated for the datasets from the *TP* and *FN* matches. The Mean Square Error (*MSE*) was calculated for the resampled and overlaid data; the elevation *MSE* was found to be 0.0883 and the intensity *MSE* was found to be 0.1351. The ARFD algorithm took an average time of 8.9151 seconds to match.

The Shi-Tomasi based ARFD matching algorithm is quite fast with comparable performance in terms of matching and alignment. Table 5-2 has the performance comparison between the versions of the ARFD algorithms using the scale-space blobs and the Shi-Tomasi corner detectors on elevation data.

We can see that the *precision* for the Shi-Tomasi is lower and the *MSE Elevation* and *MSE Intensity* are significantly higher. Although the *TPR* for the Shi-Tomasi based version is higher by 20% (because of lower number of *False Negatives*) and the *accuracy* is higher by a mere 1%, the worse performance in the other metrics indicates that the scale-space blob detector based approach works better for ARFD matching and alignment.

Table 5-2 Performance comparison of ARFD on elevation data using Scale-Space blob vs. Shi-Tomasi corner keypoints

Average Performance Metrics	ARFD with Shi-Tomasi Corner Detector	ARFD with Scale-Space Blob Detector	Percentage improvement – Scale-Space over Shi-Tomasi
Precision	0.7388	0.8139	+9
TPR	0.5036	0.4162	-20
Accuracy	0.8997	0.8887	-1
MSE Elevation	0.0883	0.0663	+12
MSE Intensity	0.1351	0.1032	+31

5.3 Performance Comparison of SIFT and ARFD Using LiDAR Intensity Data

For the SIFT and ARFD algorithms based on intensity data alone, the experiments were performed using both keypoint detection techniques and the results presented in the same manner as the elevation-only case.

5.3.1 Performance of Intensity Data When Keypoints Were Obtained from Scale-space Blobs

In these experiments the parameters that were set were the same as the elevation case, but the threshold for accepting a keypoint was set at a higher level of 5 for accepting a blob. The reason for this choice is the fact that for the dataset we used, the intensity was more varying overall and the higher threshold was useful in culling the number of keypoints chosen from each image.

Table 5-3 compares the performance of the SIFT and ARFD descriptors for matching using the intensity information. It can be seen that the *precision* is improved by using the ARFD algorithm. However, the *TPR* reduced by a fair bit. The increase in *MSE* for elevation shows that the alignment quality by using the SIFT algorithm is better than the ARFD algorithm.

Table 5-3 Performance comparison of SIFT and ARFD for intensity data with Scale-Space blob keypoints

Average Performance Metrics	SIFT Descriptor	ARFD Descriptor	Percentage improvement of ARFD over SIFT
Precision	0.2179	0.3896	+79
TPR	0.5001	0.4057	-19
Accuracy	0.9548	0.9746	+2
MSE Elevation	0.1007	0.1160	-15
MSE Intensity	0.2361	0.1662	+30

5.3.2 Performance of Intensity Data When Keypoints Were Obtained from Shi-Tomasi Corner Points

In this set of experiments, we used the intensity data to obtain corner points and we used the ARFD algorithm to match the strips of data. Again, SIFT algorithm wasn't used here because of the mismatch of the detector type used with SIFT.

Table 5-4 shows the comparison of the performance between the ARFD used with the scale-space blob detector and the Shi-Tomasi corner detector as the keypoint selection methodology.

Table 5-4 Performance comparison of ARFD on intensity data using Scale-Space blob vs. Shi-Tomasi corner keypoints

Average Performance Metrics	ARFD with Shi-Tomasi Corner Detector	ARFD with Scale-Space Blob Detector	Percentage improvement – Scale-Space over Shi-Tomasi
Precision	0.3245	0.3896	+17
TPR	0.3945	0.4057	+3
Accuracy	0.9751	0.9746	-0.05
MSE Elevation	0.1138	0.1160	-2
MSE Intensity	0.2964	0.1662	+78

It can be seen from Table 5-4 that the *precision* and *TPR* decrease and the *accuracies* is very comparable. The *MSE* for the elevation data is comparable while the *MSE* for intensity data has increased significantly.

5.4 Performance Comparison Between Elevation and Intensity Based Approaches

Table 5-5 compares the performance of ARFD for elevation based data against the performance of ARFD for intensity based data for the scale-space blob detector.

Table 5-5 Performance comparison – elevation and intensity based approaches

Average Performance Metrics	ARFD Intensity-based Descriptor	ARFD Elevation-based Descriptor	Percentage improvement of elevation over intensity
Precision	0.3896	0.8139	+109
TPR	0.4057	0.4162	+2
Accuracy	0.9746	0.8887	-9
MSE Elevation	0.1160	0.0663	+43
MSE Intensity	0.1662	0.1032	+37.91

While Table 5-5 compares different versions of only the ARFD algorithms, a similar comparison can be made for the SIFT descriptor too. It can generally be inferred from the table that the elevation based descriptor performs better than the intensity based approach. This is likely due to the fact that the LiDAR intensity data is more susceptible to different types of noise.

5.5 Performance of the Adaptive Approaches

5.5.1 Prior Attempts at Establishing Adaptive Approaches

We attempted to establish an adaptive approach at solving the matching problem. In our attempts, we tried the following techniques listed in chronological order of our attempts:

1. Concatenating the elevation and intensity descriptors in order to form a 128 dimensional descriptor.

2. Concatenating the element wise sum and element wise product of the 64 dimensional elevation and intensity descriptors to form a 128 dimensional descriptor.
3. Concatenating the normalized versions of the elevation and intensity data with an additional tuning factor to enhance or decrease the magnitude of either of the descriptors.
4. Choosing keypoints from both the elevation and intensity data from the overlapping images and culling these keypoints to obtain only those that are keypoints both in intensity and elevation data.

While these techniques were able to match some keypoints in the two overlapping images, the level of success that one would expect from a quality algorithm was missing.

Finally, the technique that worked significantly better than elevation-only data was one that considered the kurtosis of the elevation and intensity to choose a more informative descriptor. The details of our experiments are provided in Sections 5.5.2, 5.5.3 and 5.5.4.

5.5.2 Adaptive Technique for ARFD Based Matching

In this set of experiments, the descriptor was selected for each keypoint using an adaptive fashion by choosing to use the elevation or intensity data based on a higher kurtosis value of either of the modalities. What this enabled was a choice of the more “interesting” or informative neighborhood to describe a keypoint in a more unique manner. For instance, a flat piece of land that has a prominent street intersection would have a flat elevation response but a very unique intensity response. Similarly, for a complex shaped building, which otherwise blends in with the background from an intensity point of view, the elevation information is more likely to be prominent, interesting or informative. In the first case an intensity based descriptor would be chosen and in the second case an elevation based descriptor would be chosen. Our postulate was that the right choice of feature descriptors could lead to better matching than a non-adaptive descriptor based matching technique.

Table 5-6 shows the performance of the adaptive ARFD compared to the elevation based and intensity based ARFD approaches.

Table 5-6 Performance comparison of the ARFD adaptive algorithm versus intensity and elevation based approaches for the scale-space blob detector.

Average Performance Metrics	ARFD Intensity-based Descriptor	ARFD Elevation-based Descriptor	ARFD – adaptive algorithm	Percentage improvement of Adaptive over intensity based	Percentage improvement of Adaptive over elevation based
Precision	0.3896	0.8139	0.8978	+130	+10
TPR	0.4057	0.4162	0.5285	+30	+27
Accuracy	0.9746	0.8887	0.9021	-7	+1.5
MSE Elevation	0.1160	0.0663	0.0914	+21.2	-38
MSE Intensity	0.1662	0.1032	0.1270	+24	-23

From Table 5-6 it can be seen that the performance of the Adaptive ARFD algorithm is significantly better the elevation-only case with respect to the measures *precision* and *TPR.* While the increase in *accuracy* is not that significant, it can be argued that the overwhelming number of *True Negatives* (mismatches truly classified as non-correspondences) can saturate the values of *accuracies.* When a comparison of the adaptive ARFD is made with the intensity based approach the improvements in *precision* and *TPR* are also very significant. It was also inferred that the *precision*, *accuracy* and *TPR* of the adaptive ARFD approach were better than that of the elevation-only approach.

Table 5-7 Performance comparison of the adaptive algorithm versus intensity and elevation based approaches for the Shi-Tomasi detector

Average Performance Metrics	ARFD Intensity-based Descriptor	ARFD Elevation-based Descriptor	ARFD – adaptive algorithm
Precision	0.3245	0.7388	0.6681
TPR	0.3945	0.5036	0.4958
Accuracy	0.9751	0.8997	0.9140
MSE Elevation	0.1138	0.0883	0.0734
MSE Intensity	0.2964	0.1351	0.1476

From Table 5-7, it can be seen even without a percentage based comparison that the adaptive algorithm doesn't offer a better performance than the elevation based algorithm. This is perhaps because the corner points detected are not the best keypoints for extracting an "interesting" descriptor for matching. Therefore, the conclusion that can be drawn from this is that space blob detectors may be the best keypoint extracting methodologies for matching LiDAR data.

5.5.3 Adaptive Technique for SIFT Based Matching

It occurred to us that we could also create an adaptive version of SIFT much like we did for ARFD. Thus SIFT was modified to use an elevation based or an intensity based descriptor for each keypoint. A comparison similar to that described in Section 5.5.1 was performed for a new *adaptive* SIFT which too considers the kurtosis of the nearby neighborhood for the selection of the appropriate descriptor. The results obtained from this experiment are presented in Table 5-8. The adaptive SIFT did improve over the original SIFT descriptor in terms of *precision* values but its *TPR* didn't improve at all. The reason that we postulate for the mediocre performance in *TPR* is that SIFT *considers the gradients of the data (elevation or intensity)* in creating its descriptor. The kurtosis may offer information about how interesting a patch is either in its elevation or intensity data. However, this doesn't necessarily translate directly into

interesting *gradient* information and hence the change in performance is likely very little compared to the stand alone elevation based matching.

Table 5-8 Performance comparison of the SIFT adaptive algorithm versus intensity and elevation based approaches for the scale-space blob detector.

Average Performance Metrics	SIFT Intensity-based Descriptor	SIFT Elevation-based Descriptor	SIFT – adaptive algorithm	Percentage change of Adaptive over intensity based matching	Percentage change of Adaptive over intensity based matching
Precision	0.2179	0.4930	0.5360	+145.98	+9
TPR	0.5001	0.4637	0.4600	-8	-0.8
Accuracy	0.9548	0.8901	0.9174	-4	+3
MSE Elevation	0.1007	0.0848	0.0578	+43	+32
MSE Intensity	0.2361	0.1664	0.1010	+57	+39.30

5.5.4 Comparison Between Adaptive SIFT and Adaptive ARFD

A comparison between the adaptive SIFT and the adaptive ARFD algorithms is performed and the results shown in Table 5-9. We can see that the improvement in *precision* and *TPR* for adaptive ARFD is very high. Percentage numbers for these two criteria are generally valid since range of precision and TPR is 0 -1. However, the *MSE* in both elevation and intensity data has worsened in terms of percentages. These are not as significant because the actual MSE numbers for all cases are quite small.

Table 5-9 Performance comparison between adaptive versions of SIFT and ARFD descriptors

Average Performance Metrics	SIFT – adaptive algorithm	ARFD – adaptive algorithm	Percentage improvement of ARFD over SIFT
Precision	0.5360	0.8978	67.5
TPR	0.4600	0.5285	15
Accuracy	0.9174	0.9021	-1
MSE Elevation	0.0578	0.0914	-58
MSE Intensity	0.1010	0.1270	-257

5.6 Conclusion

We performed several experiments using the overlapping strip pairs for matching. In these experiments, we did not consider the fact that the data was obtained based on a common coordinate system.

It must be noted that, given the non-linear nature of all of these algorithms, the performance is clearly sensitive to the nature of the actual data – the variations in the intensity and elevations in specific geographic locations. It must be mentioned that the 49 strips that we used had only sparing amount of rugged elevation.

From the experiments that we conducted, it can generally be concluded that scale-space blobs are best suited for extracting the most interesting features for LiDAR data matching. Also, the elevation based data is more reliable for obtaining a good matching between the strips. Furthermore, we introduced an adaptive technique to choose the feature descriptor for a keypoint based on whether the elevation or intensity data in the neighborhood of a keypoint is more “interesting”. We noted that this approach performed better than even the elevation only based approach.

Finally, comparisons of the ARFD descriptor with the SIFT descriptor were made in great detail and we can conclude that the ARFD algorithm performs significantly better in comparison with the SIFT descriptor for LiDAR matching under the condition that the geographic proximity of keypoints were not considered for selecting correspondences.

Chapter 6

Performance of the SIFT and ARFD Algorithms for LiDAR Matching for Predetermined Search Area

6.1 Background

In this chapter, the results presented are for the case where we know the approximate geographic locations of the keypoints. In other words, for every keypoint in the first image, we know the rough location (*X*, *Y* or Latitude and Longitude) of a possible keypoint match in its overlapping second image. This is true for LIDAR acquired from an aerial platform, when the intensity and the elevation data are known, but the *precise* x and y locations of the matching points are unknown ahead of time. The differences in the *X*, *Y* locations for the same ground locations but visible in the overlapping areas of two strips are caused by calibration errors.

The matching strategy described in this chapter is as follow: matching is performed by seeking a correspondence for every keypoint in the first image within a permissible radius surrounding its exact location, unlike the more general situation where a matching point can be sought anywhere in the overlapping area.. In case of gross calibration errors, a relaxed radius allows us to find an exact match while in the case of nearly perfect LiDAR data; a stricter threshold reduces the computational time to match.

In our experiments, we used LiDAR data acquired from a system that is reasonably well calibrated. We were able to determine this by matching and visual examination of many overlapping strip pairs. Therefore, we used a search radius of only 0.5 pixels, which translated for the given scale of this particular data set, to a radius of 0.5 meters in terms of physical distance on the ground.

The experiments were conducted for two different scenarios:

1. the elevation data alone was used for a descriptor for each keypoint
2. and the intensity data alone was used for a descriptor for each keypoint.

All of these experiments were conducted for 47 overlapping strip pairs using the Shi-Tomasi detector and the scale-space blob detector as keypoint detection techniques.

Two different feature-matching algorithms were used:

1. The SIFT and
2. The ARFD

6.2 Requirement for LiDAR Strip Adjustment Under Common Coordinate System

LiDAR data, although acquired from calibrated systems, cannot be taken as flawless before being made available for further processing in its end applications. There is a requirement for Quality Control before it can be made publically available. In quality control procedures, “the relative and absolute accuracies” of the LiDAR strip data are verified [4]. Habib et al. [4] introduced the possible errors that can occur in LiDAR systems and broke it down into two categories:

1. Random errors
2. and systematic errors.

Random errors were classified as position noise – creating an equivalent noise in final point cloud, orientation noise – creating noise in the horizontal scanning direction and range noise – creating noise in the vertical scanning direction.

The systematic errors are a result of biases present in the LiDAR system components and measurements. Habib et al. focused on the fact that some of these errors can cause discrepancies in points in overlapping strips too. They even showed the differences between bias contaminated and true coordinates in overlapping strips which are typically collected by flying the collecting airplane in opposite directions. They provided the effect of each of the following biases in the LiDAR system:

1. Lever-arm offset bias
2. bore-sighting pitch bias

3. bore-sighting roll bias
4. bore-sighting yaw bias
5. range bias
6. and mirror angle scan bias.

They even concluded that three translation parameters and a rotation angle were sufficient to model the discrepancies in two parallel strips of data. They provided mathematical analysis for such a model in [77]. Therefore, we are attempting to match the LiDAR data even under a common coordinate system for Quality Control purposes of the LiDAR mission without blindly assuming that the LiDAR data collecting vendor did their calibration tasks perfectly that too in regular intervals of time to follow proper Quality Assurance practices.

6.3 Matching with ARFD and SIFT Descriptors With Elevation Data as Input

6.3.1 Matching Using Scale-space Blobs

In these experiments, the scale space blobs were used to obtain keypoints from both the SIFT and the ARFD descriptors using only their elevation data. The parameters for the scale-space blob detector were set as 20 levels in between 0.01 for the minimum scale and 30 for the maximum scale. The detection threshold was set as 0.5 for the elevation data because of the more regular nature of the elevation information in the experimental dataset.

Between the SIFT and ARFD descriptors, the matching was similar because of the constraint applied to both algorithms that the match must be sought within a radius of 0.5 meters about the keypoint in the first image. From visual inspection, the alignment was very good for most of the patches. One such case is shown in Figure 6-1 and Figure 6-2. It was also noteworthy that the ARFD, despite having the same alignment quality in terms of *MSE* and visual inspection, was able to pick up on an average 10% more matches that the SIFT algorithm. For ARFD algorithm, the number of matches was either equal to or more than that of the SIFT algorithm.

6.3.2 Matching Using Shi-Tomasi Corner Points

In this case, we applied the Shi-Tomasi algorithm to pick out corner points from each of the matching images. When we performed this, we set the parameters for the Shi-Tomasi corner detector as follows:

1. the blurring standard deviation $\sigma = 1$,
2. the threshold for detection $th = 1$ and
3. the *radius* of detection for the non-maximal suppression = 2.

Even in these cases, the performance of the SIFT and ARFD algorithms were comparable in terms of alignment *MSE* values and by visual inspection. The improvement in the number of matches in the ARFD algorithm was 2.75%. Thus the performances of the algorithms with this descriptor are very comparable.

A comparison between the keypoint detection techniques is not entirely possible because of the identical homography transforms obtained from both the approaches and the fact that the geographic proximity condition ruled out any wildly negative matches at the beginning stages.

Figure 6-1 Two matching strips on which elevation based matching was performed

Figure 6-2 Alignment of the strips shown in Figure 6-2Figure 6-1

6.4 Matching With ARFD and SIFT Descriptors With Intensity Data As Input

6.4.1 Matching Using Scale-space Blobs

In these experiments, the scale space blobs were used to obtain keypoints from both the SIFT and the ARFD descriptors using only their elevation data. The parameters for the scale-space blob detector were:

1. 20 levels in between 0.01 for the minimum scale and 30 for the maximum scale.
2. The detection threshold was set as 5 for the intensity data, higher than the elevation data for reasons explained in Section 6.3.1.

Here too, the alignment *MSE* and visual inspection suggested identical homography matrices for both the SIFT and ARFD descriptor algorithms. However, the ARFD was able to pick up on an average 8.5% more matches than the SIFT algorithm. However, there were instances in which the SIFT algorithm performed significantly better than the ARFD algorithm possibly because SIFT might is better suited for intensity images.

6.4.2 Matching Using Shi-Tomasi Corner Points

We applied the Shi-Tomasi algorithm to pick out corner points from each of the matching images. The parameters for the Shi-Tomasi corner detector as follows:

1. the blurring standard deviation $\sigma = 3$,
2. the threshold for detection $th = 1$ and
3. the radius of detection for the non-maximal suppression as 2.

The reason for the increase in the blurring standard deviation is that the intensity information has more contrast and hence there are more potential corner points and there is a necessity to cull the number of points used.

Even in these cases, the performance of the SIFT and ARFD algorithms were identical in all respects: *MSE*, number of matching points and by visual inspection.

6.5 Conclusion

In this chapter, we performed experiments with several methodologies on matching LiDAR data. This set of experiments were performed taking into account our knowledge of the geographic proximity of points

It was evident from our experiments that in a well calibrated LiDAR set up, the performance of the ARFD and SIFT algorithms are identical and can be used interchangeably with the desired effect. Despite that we found that in terms of number of matching points, the ARFD does marginally better than SIFT on most occasions. We can conclude that the elevation data is better represented by their statistics as in the ARFD algorithm than their gradients as in the SIFT algorithm. However, for intensity data based descriptors, the performance of ARFD and SIFT algorithms were almost exactly the same.

Chapter 7

Conclusion

In this dissertation we have introduced a technique to match co-registered LiDAR elevation and intensity data using a feature descriptor called the Aerial Range Feature Descriptor (ARFD). We investigated matching LiDAR data using standalone elevation and intensity responses. Within the task of matching LiDAR data, we considered two cases: a) when a reference coordinate system for both the LiDAR strips is available, b) and when a reference coordinate system is unavailable. For the latter task, we also presented an adaptive technique that chooses either of the data modalities to create the most informative feature descriptor. We used the Scale Invariant Feature Transform (SIFT) algorithm to perform the same matching tasks and came to the conclusion that the ARFD technique is more suited for matching LiDAR data strips. Among the keypoint detection techniques used for matching these LiDAR images, we used the scale-space blob detector and the Shi-Tomasi corner detector.

We came to an overall conclusion that for the case when a reference coordinate system is known beforehand, either the SIFT or ARFD techniques can be used with the same effect. The keypoint detection techniques were identical in their performance too. When the reference coordinate system is unknown, the best approach was to use the scale-space blob detector with an adaptive ARFD algorithm to choose better feature descriptors among intensity or elevation in order to obtain a better quality of matches.

In the future, techniques can be introduced that create feature descriptors from point clouds of data. Although they are in the same family of descriptors as the spin-images [14], these techniques must take into account the highly irregular sampling of LiDAR into account and adapt their working to this limitation. Also, binning techniques that are similar to the Gradient Location and Orientation Histogram (GLOH) [71] could be applied to the feature descriptor.

More important than these is the necessity to introduce techniques that use affine-invariant blobs [67] and create affine-invariant neighborhoods to sample points and hence create affine invariant feature descriptors. This could extend the uses of the ARFD to terrestrial LiDAR or ranging data, which mainly uses ICP to perform tasks like 3D from motion or structure-from-motion.

It would also be an interesting study to add various types of noises to the intensity and the elevation values of the raw LIDAR data to simulate noise and calibration errors and investigate how each of the algorithms cope with it. Another experiment worth trying is to apply changes to the data that offset the *X* or *Y* values to see how errors in horizontal values affect the matching.

Another issue that we recognized towards the end of our work is how to quantify the diversity of terrain captured by the 42 strips of LiDAR data used in our experiment. We simply picked a large sample of terrain data in consultation with NRCS engineers, hoping that the quantitative numbers obtained for the matching techniques from such a choice was reliable. It would be useful to get statistical measures to define the general elevation variations in the terrain as one of the independent variables.

Appendix A

MATLAB Codes

I. Interpolation of a LiDAR strip

Main script:

```
%% CLEARING WORKSPACE
clear all
close all
warning off all
clc

[int1, ele1, bb1] = readtextinterpolate('C:\Users\veladmin\Google
Drive\Mythreya VA Extraction\Set2\patch1.txt');
[int2, ele2, bb2] = readtextinterpolate('C:\Users\veladmin\Google
Drive\Mythreya VA Extraction\Set2\patch2.txt');
[int3, ele3, bb3] = readtextinterpolate('C:\Users\veladmin\Google
Drive\Mythreya VA Extraction\Set2\patch3.txt');
save set2mountain
figure
subplot(221); imshow(int1,[]);
subplot(222); imshow(int2,[]);
subplot(223); imshow(int3,[]);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Function 1: readtextinterpolate.m

Input: path to location of file.

Outputs: interpolated intensity, elevation images and bounding box of points.

```
function [int, ele, bounding_box] = readtextinterpolate(path)

    dlm = ',';
    numl =  numel(textread(path,'%1c%*[^\n]')); %read 1st char of each
line
    range = [0 0 numl-1 3];
    points = dlmread(path,dlm,range); %delimited read into a vector

    %% CREATING THE INTENSITY AND ELEVATION PLOTS
    [int, ele, bounding_box] = interpolate_LiDAR_trilinear(points);

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Function 2: interpolate_LiDAR_trilinear.m

Input: points obtained from the path supplied to readtextinterpolate function.

Outputs: interpolated intensity, elevation images and bounding box of points.

```
function [int, ele, bounding_box] =
interpolate_LiDAR_trilinear(points)

    %% SETTING UP OF PIXEL BOUNDARIES
    min_points = min(points);
    max_points = max(points);
    left = floor(min_points(1));
    right = ceil(max_points(1));
    up = floor(min_points(2));
    down = ceil(max_points(2));
    range_x = right-left;
    range_y = down-up;
    del = range_x/round(range_x);
    x_array = del/2:del:range_x;
    y_array = del/2:del:range_y;
    pel_x = length(x_array);
    pel_y = length(y_array);
    int = zeros(pel_y,pel_x);
    ele = int;
    relevant_points = cell(pel_x,pel_y);
    bounding_box = [min_points(1) min_points(2) max_points(1)
max_points(2)];

    %% DO LOOP ON THE POINTS ONLY TO EXTRACT PIXELS THAT ARE RELEVANT
TO THE POINTS
    for index = 1:size(points,1)
        i = ceil((points(index,1)-left+eps)/del);
        j = ceil((points(index,2)-up+eps)/del);
        block_x = max(i-1,1):min(pel_x,i+1);
        block_y = max(j-1,1):min(pel_y,j+1);
        [valid_x, valid_y] = meshgrid(block_x,block_y);
        for pts_1 = 1:size(valid_x,1)
            for pts_2 = 1:size(valid_x,2)
                temp_arr =
[relevant_points{valid_x(pts_1,pts_2),valid_y(pts_1,pts_2)} 0];
                temp_arr(end) = index;

relevant_points{valid_x(pts_1,pts_2),valid_y(pts_1,pts_2)} = temp_arr;
            end
        end
    end

    %% ITERATIVE ASSIGNMENT TO EMPTY CELLS
    ct = 3;
    while(ct)
        empty_cells = cellfun('isempty',relevant_points);
        [e_r, e_c] = find(empty_cells == 1);
        ct = ct-1;
        temp_cells = cell(1,length(e_r));
        for ind=1:length(e_r)
            block = relevant_points(max(e_r(ind)-
1,1):min(pel_x,e_r(ind)+1),max(e_c(ind)-1,1):min(pel_y,e_c(ind)+1));
```

```
            block(cellfun(@isempty,block)) = {0};
            tp = unique(cell2mat((block(:))'));
            tp(1) = [];
            temp_cells{1,ind} = tp;
        end
            % UPDATING THE POINTS LIST
        for ind=1:length(e_r)
            relevant_points{e_r(ind),e_c(ind)} = temp_cells{1,ind};
        end
    end

    %% INTERPOLATIONG USING INVERSE DISTANCE MEASURE
    for i=1:pel_x
        for j=1:pel_y
            sum_i = 0;
            sum_e = 0;
            dr = 0;
            current_points = relevant_points{i,j};
            loc_x = min_points(1)+del/2+(i-1)*del;
            loc_y = min_points(2)+del/2+(j-1)*del;
            for ind=1:length(current_points)
                if current_points(ind) ~= 0
                    dr_temp = sqrt((points(current_points(ind),1)-
loc_x)^2+(points(current_points(ind),2)-loc_y)^2);
                    sum_i =
sum_i+points(current_points(ind),4)/dr_temp;
                    sum_e =
sum_e+points(current_points(ind),3)/dr_temp;
                    dr = dr + 1/dr_temp;
                end
            end
            int(j,i) = sum_i/dr;
            ele(j,i) = sum_e/dr;
        end
    end
    int = flipud(int);
    ele = flipud(ele);

end
```

II. Main script for matching two overlapping LiDAR strips

Code#1: Matching under unknown geographic proximity

Note: This code contains matching with scale-space blobs on intensity data for the ARFD descriptor. Modifications to this code are made to use the Shi-Tomasi detector or the SIFT descriptor or even elevation images to generate the keypoints in the place of intensity images.

```
%% CLEARING WORKSPACE
clear all
close all
warning off all
clc
tstart = tic;

%% PARSING DATA AND RESHAPING THE MATRIX (done offline)
%% LOADING THE PARSED DATA
load vertical1
intensity1 = int19;
intensity2 = int20;
elevation1 = ele19;
elevation2 = ele20;
bb1 = bb19;
bb2 = bb20;
clear int16 int17 int18 int19 int20 int21 int22 int23 int24
clear ele16 ele17 ele18 ele19 ele20 ele21 ele22 ele23 ele24
clear bb16 bb17 bb18 bb19 bb20 bb21 bb22 bb23 bb24

%% FINDING BOUNDING PIXELS
bounds1 = boundPixels(bb1,bb2,1);
bounds2 = boundPixels(bb1,bb2,0);

%% SIFT IMAGE PYRAMIDS AND KEYPOINT SELECTION, WITH ORIENTATION AND
SCALE
s=1;
[rowse1,colse1,scalese1,responsee1] =
keypoint_scales(intensity1,1,bounds1,20,30,0.01,5);
[rowse2,colse2,scalese2,responsee2] =
keypoint_scales(intensity2,0,bounds2,20,30,0.01,5);
scalese1 = s*scalese1; scalese2 = s*scalese2;

%% ELIMINATING POINTS IN THE BORDERS
[rowse1,colse1,scalese1,responsee1] =
prunefeatures(rowse1,colse1,scalese1,responsee1,intensity1);
[rowse2,colse2,scalese2,responsee2] =
prunefeatures(rowse2,colse2,scalese2,responsee2,intensity2);

%% ARFD
```

```
Descriptorse1 =
descriptor_std(elevation1,intensity1,rowse1,colse1,scalese1);
Descriptorse2 =
descriptor_std(elevation2,intensity2,rowse2,colse2,scalese2);

%% ELEMINATION OF KEYPOINTS AT THE CORNERS
vec = sum(Descriptorse1,2);
eliminate = find(isnan(vec));
Descriptorse1(eliminate,:) = []; rowse1(eliminate) = [];
colse1(eliminate) = []; responsee1(eliminate) = [];
scalese1(eliminate) = [];
vec = sum(Descriptorse2,2);
eliminate = find(isnan(vec));
Descriptorse2(eliminate,:) = []; rowse2(eliminate) = [];
colse2(eliminate) = []; responsee2(eliminate) = [];
scalese2(eliminate) = [];
clear vec eliminate

%% MATCHING
len1 = size(Descriptorse1,1);
len2 = size(Descriptorse2,1);
Distances = zeros(len1,len2);
diste = Distances;
for i=1:len1
    for j=1:len2
        diste(i,j) = sqrt(sum((Descriptorse1(i,:) -
Descriptorse2(j,:)).^2));
    end
end
%% FORWARD MATCH: MINIMUM FOR QUERY
positives_e = [];
negatives_e = [];
for i=1:size(Descriptorse1,1);
    vector = diste(i,:);
    minimum = min(vector);
    candidates = find(vector == minimum );
    vector(candidates(1)) = [];
    if min(vector)>=1.5*minimum
        for j=1:length(candidates)
            positives_e = [positives_e;[i candidates(j)]];
        end
    else
        for j=1:length(candidates)
            negatives_e = [negatives_e;[i candidates(j)]];
        end
    end
end

%% RANSAC AND OUTLIER ESTIMATION
x1 = zeros(size(positives_e,1),2);
x2 = x1;
for i=1:size(positives_e,1)
```

```
    x1(i,:) = [colse1(positives_e(i,1)) rowse1(positives_e(i,1))];
    x2(i,:) = [colse2(positives_e(i,2)) rowse2(positives_e(i,2))];
end
maxi = 0; TP_list = []; FP_list = []; Homography_matrix = [];

for i=1:100
    [H1, inliers] = ransacfithomography(x1',x2',0.001);
    outliers = setdiff(1:size(positives_e,1),inliers);
    if length(inliers) > maxi
        maxi = length(inliers);
        TP_list = inliers;
        FP_list = outliers;
        Homography_matrix = H1;
    end
end
H1 = Homography_matrix;
H1 = H1'/H1(3,3)
TP = length(TP_list);
FP = length(FP_list);
%% TRUE AND FALSE NEGATIVES
[TN_list, FN_list] =
find_tnfn_eleonly(negatives_e,rowse1,rowse2,colse1,colse2,H1);
TN = length(TN_list);
FN = length(FN_list);
%% RECALCULATE HOMOGRAPHY - make use of TP and FN points
x1 = zeros(TP+FN,2);
x2 = x1;
for i=1:TP
    x1(i,:) = [colse1(positives_e(TP_list(i),1))
rowse1(positives_e(TP_list(i),1))];
    x2(i,:) = [colse2(positives_e(TP_list(i),2))
rowse2(positives_e(TP_list(i),2))];
end
for i=1:FN
    x1(i+TP,:) = [colse1(negatives_e(FN_list(i),1))
rowse1(negatives_e(FN_list(i),1))];
    x2(i+TP,:) = [colse2(negatives_e(FN_list(i),2))
rowse2(negatives_e(FN_list(i),2))];
end
maxi = 0; Homography_matrix = [];
[H2, inliers] = ransacfithomography(x1',x2',0.0001);
outliers = setdiff(1:(length(TP_list)+length(FN_list)),inliers);
H2 = H2'/H2(3,3)

[TP FP TN FN]
x1(:,3) = 1; x2(:,3) = 1;

%% FIND THE RESAMPLED IMAGE
i_offset = max(-round(H2(3,2)),0);
j_offset = max(-round(H2(3,1)),0);
error = 0;
count = 0;
```

```
T = maketform('projective',H2);
transformedI = imtransform(intensity2,T);
transformedI(isnan(transformedI))=0;
I_stitched = intensity1;
I_stitched(isnan(I_stitched))=0;
for i=1:size(transformedI,1)
    for j=1:size(transformedI,2)
        if(i+i_offset <= size(intensity1,1) && j+j_offset <=
size(intensity1,2))
            if(intensity1(i+i_offset,j+j_offset) ~= 0 &&
transformedI(i,j) ~= 0 )
                error = error+abs(I_stitched(i+i_offset,j+j_offset) -
transformedI(i,j))^2;
                count = count+1;
            end
        end
        if transformedI(i,j)
            I_stitched(i+i_offset,j+j_offset) = transformedI(i,j);
        else
            continue
        end
    end
end
figure
imshow(I_stitched,[]);

%% FIND THE MSE AND PSNR
MSE = sqrt(error)/count
PSNR = 10*log10(255^2/MSE)
toc(tstart)
%% STITCHING THE IMAGES AND DISPLAYING MATCHES ONLY FROM THE LIST
output_image = intensity1;
output_image(size(intensity1,1)+1+10:size(intensity1,1)+size(intensity
2,1)+10,1:size(intensity2,2)) = intensity2(:,:);
% ctt = 0;
figure
imshow(output_image,[])
hold on
% TRUE POSITIVES
for i=1:TP
    p1 = [rowse1(positives_e(TP_list(i),1))
colse1(positives_e(TP_list(i),1))];
    p2 = [rowse2(positives_e(TP_list(i),2))+10+size(intensity1,1)
colse2(positives_e(TP_list(i),2))];
    plot([p1(2),p2(2)],[p1(1),p2(1)],'Color','g','LineWidth',1);
end

% FALSE POSITIVES
output_image = intensity1;
output_image(size(intensity1,1)+1+10:size(intensity1,1)+size(intensity
2,1)+10,1:size(intensity2,2)) = intensity2(:,:);
% ctt = 0;
```

```
figure
imshow(output_image,[])
hold on
for i=1:FP
    p1 = [rowse1(positives_e(FP_list(i),1))
colse1(positives_e(FP_list(i),1))];
    p2 = [rowse2(positives_e(FP_list(i),2))+10+size(intensity1,1)
colse2(positives_e(FP_list(i),2))];
    plot([p1(2),p2(2)],[p1(1),p2(1)],'Color','r','LineWidth',1);
end

output_image = intensity1;
output_image(size(intensity1,1)+1+10:size(intensity1,1)+size(intensity
2,1)+10,1:size(intensity2,2)) = intensity2(:,:);
% ctt = 0;
figure
imshow(output_image,[])
hold on

% TRUE NEGATIVES
for i=1:TN
    p1 = [rowse1(negatives_e(TN_list(i),1))
colse1(negatives_e(TN_list(i),1))];
    p2 = [rowse2(negatives_e(TN_list(i),2))+10+size(intensity1,1)
colse2(negatives_e(TN_list(i),2))];
    plot([p1(2),p2(2)],[p1(1),p2(1)],'Color','b','LineWidth',1);
end

% FALSE NEGATIVES
output_image = intensity1;
output_image(size(intensity1,1)+1+10:size(intensity1,1)+size(intensity
2,1)+10,1:size(intensity2,2)) = intensity2(:,:);
% ctt = 0;
figure
imshow(output_image,[])
hold on
for i=1:FN
    p1 = [rowse1(negatives_e(FN_list(i),1))
colse1(negatives_e(FN_list(i),1))];
    p2 = [rowse2(negatives_e(FN_list(i),2))+10+size(intensity1,1)
colse2(negatives_e(FN_list(i),2))];
    plot([p1(2),p2(2)],[p1(1),p2(1)],'Color','y','LineWidth',1);
end
```

Code#2: Matching under known geographic proximity

```
%% CLEARING WORKSPACE
clear all
close all
warning off all
clc

%% PARSING DATA AND RESHAPING THE MATRIX (done offline)
%% LOADING THE PARSED DATA
load vertical1
intensity1 = int16;
intensity2 = int17;
elevation1 = ele16;
elevation2 = ele17;
bb1 = bb16;
bb2 = bb17;
clear int16 int17 int18 int19 int20 int21 int22 int23 int24
clear ele16 ele17 ele18 ele19 ele20 ele21 ele22 ele23 ele24
clear bb16 bb17 bb18 bb19 bb20 bb21 bb22 bb23 bb24

%% FINDING BOUNDING PIXELS
tstart = tic;
bounds1 = boundPixelsNew(bb1,bb2);
bounds2 = boundPixelsNew(bb2,bb1);

%% SIFT IMAGE PYRAMIDS AND KEYPOINT SELECTION, WITH ORIENTATION AND
SCALE
s=1;
[rows1,columns1,scales1,response1] =
keypoint_scales(elevation1,1,bounds1,20,30,0.01,5);
[rows2,columns2,scales2,response2] =
keypoint_scales(elevation2,0,bounds2,20,30,0.01,5);
scales1 = s*scales1; scales2 = s*scales2;

%% RUNNING CODE
[rows1,columns1,scales1,response1] =
prunefeatures(rows1,columns1,scales1,response1,intensity1);
[rows2,columns2,scales2,response2] =
prunefeatures(rows2,columns2,scales2,response2,intensity2);

%% OBTAIN DESCRIPTORS
Descriptors1 =
descriptor_std(elevation1,intensity1,rows1,columns1,s*ones(length(colu
mns1)));
Descriptors2 =
descriptor_std(elevation2,intensity2,rows2,columns2,s*ones(length(colu
mns2)));

% MATCHING
distTh = 2;
positives = [];
```

```
negatives = [];
for i=1:length(rows1)
    Descriptor_currenti = Descriptors1(i,:);
    feature_distance = inf;
    match = -1;
    for j=1:length(rows2)
        % INITIAL WORLD POSITIONS
        iwp1 = [bb1(4),bb1(1)];
        iwp2 = [bb2(4),bb2(1)];
        % POSITIONS AFTER OFFSET
        off1 = iwp1+[-rows1(i) columns1(i)];
        off2 = iwp2+[-rows2(j) columns2(j)];
        % DIFFERENCES
        geographic_distance = abs(off2-off1);
        list = [];
        if norm(geographic_distance) < distTh
            Descriptor_currentj = Descriptors2(j,:);
            current_feature_distance = sqrt(sum((Descriptor_currentj -
Descriptor_currenti).^2));
            if current_feature_distance < feature_distance
                feature_distance = current_feature_distance;
                match = j;
            end
            list = [list j];
        end
    end
    if ~isempty(list)
        list
    end
    if match~=-1
        positives = [positives; [i match]];
    else
        continue
    end
end

%% RANSAC
x1 = zeros(size(positives,1),2);
x2 = x1;
for i=1:size(positives,1)
    x1(i,:) = [columns1(positives(i,1)) rows1(positives(i,1))];
    x2(i,:) = [columns2(positives(i,2)) rows2(positives(i,2))];
end
[H1, inliers] = ransacfithomography(x1',x2',0.1);
outliers = setdiff(1:size(positives,1),inliers);
TP_list = inliers;
FP_list = outliers;
H1 = H1'/H1(3,3)
TP = length(TP_list);
FP = length(FP_list);
telapsed = toc(tstart)
%% STITCHING IMAGES
[I_stitched,MSE_i] = stitching(H1,intensity1,intensity2);
```

```
[E_stitched,MSE_e] = stitching(H1,elevation1,elevation2);

%% POSITIVE MATCHES
output_image = intensity1;
output_image(size(intensity1,1)+1:size(intensity1,1)+size(intensity2,1
),1:size(intensity2,2)) = intensity2(:,:);
figure
imshow(output_image,[])
hold on
% POSITIVES
for i=1:size(positives,1)
    p1 = [rows1(positives(i,1)) columns1(positives(i,1))];
    p2 = [rows2(positives(i,2))+size(intensity1,1)
columns2(positives(i,2))];
    plot([p1(2),p2(2)],[p1(1),p2(1)],'Color','g','LineWidth',1);
end

%% STITCHED IMAGES
figure
imshow(I_stitched,[]);
% figure
% imshow(E_stitched,[]);

%% DATA
% load SIFT_geo_SS_int
% load ARFD_geo_SS_int
% ARFD_geo_SS_int = [ARFD_geo_SS_int;[TP MSE_e MSE_i telapsed]];
% save ARFD_geo_SS_int ARFD_geo_SS_int
% clear
% load SIFT_geo_SS_int
% load ARFD_geo_SS_int

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Function 1: boundPixels.m

Inputs: bounding boxes for upper and lower images, flag to tell if images is upper or lower.

Outputs: overlapping area for that image in pixel coordinates.

```
function bounds = boundPixels(bb1,bb2,tblr)

    if tblr == 1
        l = max(bb1(1),bb2(1));
        r = min(bb1(3),bb2(3));
        t = bb2(4);
        b = bb1(2);
        pix_l = floor(l) - floor(bb1(1))-1;
        pix_r = floor(r) - floor(bb1(1))-1;
        pix_t = abs(floor(t) - floor(bb1(4))-1);
        pix_b = abs(floor(b) - floor(bb1(4))-1);
        bounds = [pix_l pix_r pix_t pix_b];
```

```
    elseif tblr == 0
        l = max(bb1(1),bb2(1));
        r = min(bb1(3),bb2(3));
        t = bb2(4);
        b = bb1(2);
        pix_l = floor(l) - floor(bb2(1))-1;
        pix_r = floor(r) - floor(bb2(1))-1;
        pix_t = abs(floor(t) - floor(bb2(4))-1);
        pix_b = abs(floor(b) - floor(bb2(4))-1);
        bounds = [pix_l pix_r pix_t pix_b];
    end

end


            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Function 2: keypoint_scales.m

Inputs: input image and input parameters for running the scale-space blob detector.

Outputs: row, column, key scale and detector response for each detected keypoint.

```
function [row, column, key_scale, response] =
keypoint_scales(image,top,bounds,levels,sigma0,sigma_min,th)

    %% READING DATA
    if length(size(image))>2
        I = double(rgb2gray(image));
    else
        I = double(image);
    end
    r = size(I,1);
    c = size(I,2);

    %% INITIALIZATIONS
    scale_space = zeros(r,c,levels);
    maxima_levels = scale_space;

    %% CREATING THE SCALE SPACE AND FILTERING THE IMAGES
    for i=1:levels
        sigma = sigma0-(sigma0-sigma_min)*(i-1)/(levels-1);
        kern_size = max(1,floor(6*sigma));
        kern = fspecial('log', kern_size, sigma );
        norm_kern = sigma.^2 * kern;
        filtered_image = (imfilter(I, norm_kern, 'replicate')).^2;
        scale_space(:,:,i) = imresize(filtered_image,[r
c],'bilinear');
    end

    %% NON-MAXIMA SUPRESSION
    %% a) 2D NON-MAXIMA SUPRESSION
```

```
    locations = [];
    for i=1:levels
        size_filt = ceil(sigma0 - (sigma0-sigma_min)*(i-1)/(levels-
1));
        maximas =
ordfilt2(scale_space(:,:,i),size_filt^2,ones(size_filt));
        maxima_levels(:,:,i) = (scale_space(:,:,i) == maximas) &
(scale_space(:,:,i)> th);
        [rows,columns]  = find(maxima_levels(:,:,i));
        locations = [locations, {[rows,columns]}];
    end

    %% b) 3D NON-MAXIMA SUPRESSION
    op_points = [];
    for i=2:levels-1
        current_points = locations{i};
        sigma = sigma0 - (sigma0-sigma_min)*(i-1)/(levels-1);
        filter_size = floor(sigma);
        for j=1:size(current_points,1)
            x = current_points(j,1);
            y = current_points(j,2);
            data = scale_space(x,y,i);

            if (x <= filter_size || x >= r - filter_size)
                continue;
            end
            if (y <= filter_size || y >= c - filter_size)
                continue;
            end
            block = scale_space(x-filter_size:x+filter_size,y-
filter_size:y+filter_size,1:levels);
            maxx = max(max(max(block)));

            if (data==maxx)
                if top && x >= bounds(3) && y>=bounds(1) &&
y<=bounds(2)
                    op_points = [op_points; [x y 1.5*sigma data]];
                elseif ~top && x <= bounds(4) && y>=bounds(1) &&
y<=bounds(2)
                    op_points = [op_points; [x y 1.5*sigma data]];
                end
            end
        end
    end


show_all_circles(uint8(image),op_points(:,2),op_points(:,1),op_points(
:,3));
    row = op_points(:,1);
    column = op_points(:,2);
    key_scale = op_points(:,3);
    response = op_points(:,4);
```

```
end
```

%%

Function 3: corner.m [76]

Note: This is a modified version of the code written by Peter Kovesi for the Harris corner detector. This code finds the Shi-Tomasi corner points instead.

Inputs: input image and input parameters for the corner detector.

Output: corner detector response, row and column for each keypoint.

```
function [cim, r, c] = corner(im, sigma, thresh, radius, disp, method,
top, bounds)

    error(nargchk(2,8,nargin));

    dx = [-1 0 1; -1 0 1; -1 0 1]; % Derivative masks
    dy = dx';

    Ix = conv2(im, dx, 'same');    % Image derivatives
    Iy = conv2(im, dy, 'same');

    % Generate Gaussian filter of size 6*sigma (+/- 3sigma) and of
    % minimum size 1x1.
    g = fspecial('gaussian',max(1,fix(6*sigma)), sigma);

    Ix2 = conv2(Ix.^2, g, 'same'); % Smoothed squared image
derivatives
    Iy2 = conv2(Iy.^2, g, 'same');
    Ixy = conv2(Ix.*Iy, g, 'same');

    if method == 0
        cim = (Ix2.*Iy2 - Ixy.^2)./(Ix2 + Iy2 + eps); % Harris corner
measure
    elseif method == 1
        T = Ix2 + Iy2;
        D = Ix2.*Iy2 - Ixy.^2;
        lambda1 = (T - sqrt(T.^2-4*D))/2;
        lambda2 = (T + sqrt(T.^2-4*D))/2;
        cim = min(lambda1,lambda2);
    end
    % Alternate Harris corner measure used by some.  Suggested that
    % k=0.04 - I find this a bit arbitrary and unsatisfactory.
%   cim = (Ix2.*Iy2 - Ixy.^2) - k*(Ix2 + Iy2).^2;

%     if nargin > 2   % We should perform nonmaximal suppression and
threshold
```

```
    % Extract local maxima by performing a grey scale morphological
    % dilation and then finding points in the corner strength image
that
    % match the dilated image and are also greater than the threshold.
    sze = 2*radius+1;                   % Size of mask.
    mx = ordfilt2(cim,sze^2,ones(sze)); % Grey-scale dilate.
    cim = (cim==mx)&(cim>thresh);       % Find maxima.

    [r,c] = find(cim);                  % Find row,col coords.

    if top
        delete = find(r<bounds(3));
        r(delete) = [];
        c(delete) = [];
        delete = find(c<bounds(1));
        r(delete) = [];
        c(delete) = [];
        delete = find(c>bounds(2));
        r(delete) = [];
        c(delete) = [];
    else
        delete = find(r>bounds(4));
        r(delete) = [];
        c(delete) = [];
        delete = find(c<bounds(1));
        r(delete) = [];
        c(delete) = [];
        delete = find(c>bounds(2));
        r(delete) = [];
        c(delete) = [];
    end

    if disp      % overlay corners on original image
        figure, imagesc(im), axis image, colormap(gray), hold on
        plot(c,r,'ys'), title('corners detected');
    end

end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Function 4: prunefeatures.m

Inputs: obtained locations and parameters for all keypoints and input image.

Outputs: keypoint locations and parameters for keypoints retained.

```
function [rows,cols,scales,response] =
prunefeatures(rows,cols,scales,response,image)
```

```
    rt=[]; ct=[]; scalest = [];
    scales = 0.5*scales;
    for i=1:length(rows)
         if rows(i)<=16*scales(i) || cols(i)<=16*scales(i) ||
rows(i)>size(image,1)-16*scales(i) || cols(i)>size(image,2)-
16*scales(i)
            continue;
         else
             rt = [rt rows(i)];
             ct = [ct cols(i)];
             scalest = [scalest scales(i)];
         end
    end
    rows = rt'; cols = ct'; scales = scalest';

end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Function 5: find_sift.m [75]

Inputs: input image, locations of keypoints and enlargement factor.

Outputs: SIFT feature descriptors.

```
function sift_arr = find_sift(I, circles, enlarge_factor)

if ndims(I) == 3
    I = im2double(rgb2gray(I));
else
    I = im2double(I);
end


% fprintf('Running find_sift\n');

% parameters (default SIFT size)
num_angles = 8;
num_bins = 4;
num_samples = num_bins * num_bins;
alpha = 9; % smoothing for orientation histogram

if nargin < 3
    enlarge_factor = 1.5;
end

angle_step = 2 * pi / num_angles;
angles = 0:angle_step:2*pi;
angles(num_angles+1) = []; % bin centers

[hgt wid] = size(I);
```

```
num_pts = size(circles,1);

sift_arr = zeros(num_pts, num_samples * num_angles);

% edge image
sigma_edge = 1;


[G_X,G_Y]=gen_dgauss(sigma_edge);
I_X = filter2(G_X, I, 'same'); % vertical edges
I_Y = filter2(G_Y, I, 'same'); % horizontal edges
I_mag = sqrt(I_X.^2 + I_Y.^2); % gradient magnitude
I_theta = atan2(I_Y,I_X);
I_theta(isnan(I_theta)) = 0; % necessary????

% make default grid of samples (centered at zero, width 2)
interval = 2/num_bins:2/num_bins:2;
interval = interval - (1/num_bins + 1);
[grid_x grid_y] = meshgrid(interval, interval);
grid_x = reshape(grid_x, [1 num_samples]);
grid_y = reshape(grid_y, [1 num_samples]);

% make orientation images
I_orientation = zeros(hgt, wid, num_angles);
% for each histogram angle
for a=1:num_angles
    % compute each orientation channel
    tmp = cos(I_theta - angles(a)).^alpha;
    tmp = tmp .* (tmp > 0);

    % weight by magnitude
    I_orientation(:,:,a) = tmp .* I_mag;
end

% for all circles
for i=1:num_pts
    cx = circles(i,1);
    cy = circles(i,2);
    r = circles(i,3) * enlarge_factor;

    % find coordinates of sample points (bin centers)
    grid_x_t = grid_x * r + cx;
    grid_y_t = grid_y * r + cy;
    grid_res = grid_y_t(2) - grid_y_t(1);

    % find window of pixels that contributes to this descriptor
    x_lo = floor(max(cx - r - grid_res/2, 1));
    x_hi = ceil(min(cx + r + grid_res/2, wid));
    y_lo = floor(max(cy - r - grid_res/2, 1));
    y_hi = ceil(min(cy + r + grid_res/2, hgt));
```

```
    % find coordinates of pixels
    [grid_px, grid_py] = meshgrid(x_lo:x_hi,y_lo:y_hi);
    num_pix = numel(grid_px);
    grid_px = reshape(grid_px, [num_pix 1]);
    grid_py = reshape(grid_py, [num_pix 1]);

    % find (horiz, vert) distance between each pixel and each grid
sample
    dist_px = abs(repmat(grid_px, [1 num_samples]) - repmat(grid_x_t,
[num_pix 1]));
    dist_py = abs(repmat(grid_py, [1 num_samples]) - repmat(grid_y_t,
[num_pix 1]));

    % find weight of contribution of each pixel to each bin
    weights_x = dist_px/grid_res;
    weights_x = (1 - weights_x) .* (weights_x <= 1);
    weights_y = dist_py/grid_res;
    weights_y = (1 - weights_y) .* (weights_y <= 1);
    weights = weights_x .* weights_y;

    % make sift descriptor
    curr_sift = zeros(num_angles, num_samples);
    for a = 1:num_angles
        tmp = reshape(I_orientation(y_lo:y_hi,x_lo:x_hi,a),[num_pix
1]);
        tmp = repmat(tmp, [1 num_samples]);
        curr_sift(a,:) = sum(tmp .* weights);
    end
    sift_arr(i,:) = reshape(curr_sift, [1 num_samples * num_angles]);

end

%%
%% normalize the SIFT descriptors more or less as described in Lowe
(2004)
%%
tmp = sqrt(sum(sift_arr.^2, 2));
normalize_ind = find(tmp > 1);

sift_arr_norm = sift_arr(normalize_ind,:);
sift_arr_norm = sift_arr_norm ./ repmat(tmp(normalize_ind,:), [1
size(sift_arr,2)]);

% suppress large gradients
sift_arr_norm(find(sift_arr_norm > 0.2)) = 0.2;

% finally, renormalize to unit length
tmp = sqrt(sum(sift_arr_norm.^2, 2));
sift_arr_norm = sift_arr_norm ./ repmat(tmp, [1 size(sift_arr,2)]);

sift_arr(normalize_ind,:) = sift_arr_norm;
```

```
function [GX,GY]=gen_dgauss(sigma)

f_wid = 4 * floor(sigma);
G = normpdf(-f_wid:f_wid,0,sigma);
G = G' * G;
[GX,GY] = gradient(G);

GX = GX * 2 ./ sum(sum(abs(GX)));
GY = GY * 2 ./ sum(sum(abs(GY)));

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Function 6: descriptor_std.m

Inputs: input image, locations of keypoints and enlargement factor.

Outputs: ARFD feature descriptors and flag to tell whether its corresponding descriptor was elevation or intensity based.

```
function varargout =
descriptor_std(elevation,intensity,rows,columns,scales)

    [AA,BB] = meshgrid(1:size(elevation,2),1:size(elevation,1));
    scales = scales/3;
    output = zeros(length(rows),64);
    eleorint = zeros(1,length(rows));
    for i=1:length(rows)
            desc1 = zeros(1,64);
            desc2 = zeros(1,64);
            u = scales(i)*((1:16) - 8.5);
            v = scales(i)*((1:16) - 8.5);
            [pX, pY] = meshgrid(u,v);
            off = [pX(:)';pY(:)'];
            pts(1,:) = off(1,:)+rows(i);
            pts(2,:) = off(2,:)+columns(i);
            siz = 16;
            block = elevation(rows(i)-siz:rows(i)+siz, columns(i)-
siz:columns(i)+siz);
            measure1 = kurtosis(block(:));
            block = intensity(rows(i)-siz:rows(i)+siz, columns(i)-
siz:columns(i)+siz);
            measure2 = kurtosis(block(:));
            for j=1:4
                for k=1:4
                    p = (((j-1)*4+1:(j-1)*4+4)-8.5)*scales(i);
                    q = (((k-1)*4+1:(k-1)*4+4)-8.5)*scales(i);
                    [X, Y] = meshgrid(p,q);
                    offsets = [X(:)';Y(:)'];
                    new_points(1,:) = offsets(1,:)+rows(i);
```

```
                    new_points(2,:) = offsets(2,:)+columns(i);
                    if measure1>measure2
                        temp1 =
qinterp2(AA,BB,elevation,new_points(2,:),new_points(1,:));
                        temp1 = [mean(temp1) var(temp1) (max(temp1)-
min(temp1)) median(temp1)];
                        index = (j-1)*16+(k-1)*4;
                        desc1(index+1:index+4) = temp1;
                        eleorint(i) = 1;
                    else
                        temp1 =
qinterp2(AA,BB,intensity,new_points(2,:),new_points(1,:));
                        temp1 = [mean(temp1) var(temp1) (max(temp1)-
min(temp1)) median(temp1)];
                        index = (j-1)*16+(k-1)*4;
                        desc1(index+1:index+4) = temp1;
                        eleorint(i) = 0;
                    end
                    p = (((j-1)*4+1:(j-1)*4+4)-8.5)*scales(i);
                    q = (((k-1)*4+1:(k-1)*4+4)-8.5)*scales(i);
                    [X, Y] = meshgrid(p,q);
                    offsets = [X(:)';Y(:)'];
                    new_points(1,:) = offsets(1,:)+rows(i);
                    new_points(2,:) = offsets(2,:)+columns(i);
                    temp1 =
qinterp2(AA,BB,intensity,new_points(2,:),new_points(1,:));
                    temp1 = [mean(temp1) skewness(temp1) median(temp1)
kurtosis(temp1)];
                    index = (j-1)*16+(k-1)*4;
                    desc2(index+1:index+4) = temp1;
                end
            end
            output(i,:) = desc1(:);
    end
    varargout{1} = output;
    varargout{2} = eleorint;

end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Function 7: ransacfithomography.m [76]

Inputs: locations of correspondences and tolerance factor for fitting.

Outputs: homography and list of inliers.

```
function [H, inliers] = ransacfithomography(x1, x2, t)

    if ~all(size(x1)==size(x2))
        error('Data sets x1 and x2 must have the same dimension');
    end
```

```
    [rows,npts] = size(x1);
    if rows~=2 & rows~=3
        error('x1 and x2 must have 2 or 3 rows');
    end

    if npts < 4
        error('Must have at least 4 points to fit homography');
    end

    if rows == 2    % Pad data with homogeneous scale factor of 1
        x1 = [x1; ones(1,npts)];
        x2 = [x2; ones(1,npts)];
    end

    % Normalise each set of points so that the origin is at centroid
and
    % mean distance from origin is sqrt(2).  normalise2dpts also
ensures the
    % scale parameter is 1.  Note that 'homography2d' will also call
    % 'normalise2dpts' but the code in 'ransac' that calls the
distance
    % function will not - so it is best that we normalise beforehand.
    [x1, T1] = normalise2dpts(x1);
    [x2, T2] = normalise2dpts(x2);

    s = 4;  % Minimum No of points needed to fit a homography.

    fittingfn = @homography2d;
    distfn    = @homogdist2d;
    degenfn   = @isdegenerate;
    % x1 and x2 are 'stacked' to create a 6xN array for ransac
    [H, inliers] = ransac([x1; x2], fittingfn, distfn, degenfn, s, t);

    % Now do a final least squares fit on the data points considered
to
    % be inliers.
    H = homography2d(x1(:,inliers), x2(:,inliers));

    % Denormalise
    H = T2\H*T1;

%---------------------------------------------------------------------
-
% Function to evaluate the symmetric transfer error of a homography
with
% respect to a set of matched points as needed by RANSAC.

function [inliers, H] = homogdist2d(H, x, t);

    x1 = x(1:3,:);   % Extract x1 and x2 from x
```

```
    x2 = x(4:6,:);

    % Calculate, in both directions, the transfered points
    Hx1    = H*x1;
    invHx2 = H\x2;

    % Normalise so that the homogeneous scale parameter for all
coordinates
    % is 1.

    x1     = hnormalise(x1);
    x2     = hnormalise(x2);
    Hx1    = hnormalise(Hx1);
    invHx2 = hnormalise(invHx2);

    d2 = sum((x1-invHx2).^2)  + sum((x2-Hx1).^2);
    inliers = find(abs(d2) < t);
    outliers = find(abs(d2) > t);


%-------------------------------------------------------------------------
-
% Function to determine if a set of 4 pairs of matched  points give
rise
% to a degeneracy in the calculation of a homography as needed by
RANSAC.
% This involves testing whether any 3 of the 4 points in each set is
% colinear.

function r = isdegenerate(x)

    x1 = x(1:3,:);    % Extract x1 and x2 from x
    x2 = x(4:6,:);

    r = ...
    iscolinear(x1(:,1),x1(:,2),x1(:,3)) | ...
    iscolinear(x1(:,1),x1(:,2),x1(:,4)) | ...
    iscolinear(x1(:,1),x1(:,3),x1(:,4)) | ...
    iscolinear(x1(:,2),x1(:,3),x1(:,4)) | ...
    iscolinear(x2(:,1),x2(:,2),x2(:,3)) | ...
    iscolinear(x2(:,1),x2(:,2),x2(:,4)) | ...
    iscolinear(x2(:,1),x2(:,3),x2(:,4)) | ...
    iscolinear(x2(:,2),x2(:,3),x2(:,4));

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Function 8: ransac.m [76]

Inputs: functions and parameters to obtain a fit.

Outputs: current model and set of inliers for the same.

```
function [M, inliers] = ransac(x, fittingfn, distfn, degenfn, s, t,
feedback, ...
                               maxDataTrials, maxTrials)

    % Test number of parameters
    error ( nargchk ( 6, 9, nargin ) );

    if nargin < 9; maxTrials = 10000;    end;
    if nargin < 8; maxDataTrials = 100; end;
    if nargin < 7; feedback = 0;        end;

    [rows, npts] = size(x);

    p = 0.999;          % Desired probability of choosing at least one
sample
                        % free from outliers (probably should be a
parameter)

    bestM = NaN;       % Sentinel value allowing detection of solution
failure.
    trialcount = 0;
    bestscore =  0;
    N = 1;             % Dummy initialisation for number of trials.

    while N > trialcount

        % Select at random s datapoints to form a trial model, M.
        % In selecting these points we have to check that they are not
in
        % a degenerate configuration.
        degenerate = 1;
        count = 1;
        while degenerate
            % Generate s random indicies in the range 1..npts
            % (If you do not have the statistics toolbox with
randsample(),
            % use the function RANDOMSAMPLE from my webpage)
            if ~exist('randsample', 'file')
                ind = randomsample(npts, s);
            else
                ind = randsample(npts, s);
            end

            % Test that these points are not a degenerate
configuration.
            degenerate = feval(degenfn, x(:,ind));

            if ~degenerate
                % Fit model to this random selection of data points.
```

```
                % Note that M may represent a set of models that fit
the data in
                % this case M will be a cell array of models
                M = feval(fittingfn, x(:,ind));

                % Depending on your problem it might be that the only
way you
                % can determine whether a data set is degenerate or
not is to
                % try to fit a model and see if it succeeds.  If it
fails we
                % reset degenerate to true.
                if isempty(M)
                    degenerate = 1;
                end
            end

            % Safeguard against being stuck in this loop forever
            count = count + 1;
            if count > maxDataTrials
                warning('Unable to select a nondegenerate data set');
                break
            end
        end

        % Once we are out here we should have some kind of model...
        % Evaluate distances between points and model returning the
indices
        % of elements in x that are inliers.  Additionally, if M is a
cell
        % array of possible models 'distfn' will return the model that
has
        % the most inliers.  After this call M will be a non-cell
object
        % representing only one model.
        [inliers, M] = feval(distfn, M, x, t);

        % Find the number of inliers to this model.
        ninliers = length(inliers);

        if ninliers > bestscore    % Largest set of inliers so far...
            bestscore = ninliers;  % Record data for this model
            bestinliers = inliers;
            bestM = M;

            % Update estimate of N, the number of trials to ensure we
pick,
            % with probability p, a data set with no outliers.
            fracinliers =  ninliers/npts;
            pNoOutliers = 1 -  fracinliers^s;
            pNoOutliers = max(eps, pNoOutliers);  % Avoid division by
-Inf
```

```
            pNoOutliers = min(1-eps, pNoOutliers);% Avoid division by
0.
            N = log(1-p)/log(pNoOutliers);
        end

        trialcount = trialcount+1;
        if feedback
            fprintf('trial %d out of %d         \r',trialcount,
ceil(N));
        end

        % Safeguard against being stuck in this loop forever
        if trialcount > maxTrials
            warning( ...
            sprintf('ransac reached the maximum number of %d
trials',...
                    maxTrials));
            break
        end
    end

    if feedback, fprintf('\n'); end

    if ~isnan(bestM)   % We got a solution
        M = bestM;
        inliers = bestinliers;
    else
        M = [];
        inliers = [];
        error('ransac was unable to find a useful solution');
    end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Function 9: homography2d.m [76]

Inputs: locations of correspondences.

Outputs: homography.

```
function H = homography2d(varargin)

    [x1, x2] = checkargs(varargin(:));

    % Attempt to normalise each set of points so that the origin
    % is at centroid and mean distance from origin is sqrt(2).
    [x1, T1] = normalise2dpts(x1);
    [x2, T2] = normalise2dpts(x2);

    % Note that it may have not been possible to normalise
    % the points if one was at infinity so the following does not
```

```
    % assume that scale parameter w = 1.

    Npts = length(x1);
    A = zeros(3*Npts,9);

    O = [0 0 0];
    for n = 1:Npts
    X = x1(:,n)';
    x = x2(1,n); y = x2(2,n); w = x2(3,n);
    A(3*n-2,:) = [  O  -w*X  y*X];
    A(3*n-1,:) = [ w*X   O  -x*X];
    A(3*n  ,:) = [-y*X  x*X   O ];
    end

    [U,D,V] = svd(A,0); % 'Economy' decomposition for speed

    % Extract homography
    H = reshape(V(:,9),3,3)';

    % Denormalise
    H = T2\H*T1;


%-------------------------------------------------------------------------
-----
% Function to check argument values and set defaults

function [x1, x2] = checkargs(arg);

    if length(arg) == 2
    x1 = arg{1};
    x2 = arg{2};
    if ~all(size(x1)==size(x2))
        error('x1 and x2 must have the same size');
    elseif size(x1,1) ~= 3
        error('x1 and x2 must be 3xN');
    end

    elseif length(arg) == 1
    if size(arg{1},1) ~= 6
        error('Single argument x must be 6xN');
    else
        x1 = arg{1}(1:3,:);
        x2 = arg{1}(4:6,:);
    end
    else
    error('Wrong number of arguments supplied');
    end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Function 10: normalise2dpts.m [76]

Inputs: 2D points.

Outputs: Normalized points and transformation.

```
function [newpts, T] = normalise2dpts(pts)

    if size(pts,1) ~= 3
        error('pts must be 3xN');
    end

    % Find the indices of the points that are not at infinity
    finiteind = find(abs(pts(3,:)) > eps);

    if length(finiteind) ~= size(pts,2)
        warning('Some points are at infinity');
    end

    % For the finite points ensure homogeneous coords have scale of 1
    pts(1,finiteind) = pts(1,finiteind)./pts(3,finiteind);
    pts(2,finiteind) = pts(2,finiteind)./pts(3,finiteind);
    pts(3,finiteind) = 1;

    c = mean(pts(1:2,finiteind)')';            % Centroid of finite
points
    newp(1,finiteind) = pts(1,finiteind)-c(1); % Shift origin to
centroid.
    newp(2,finiteind) = pts(2,finiteind)-c(2);

    dist = sqrt(newp(1,finiteind).^2 + newp(2,finiteind).^2);
    meandist = mean(dist(:));  % Ensure dist is a column vector for
Octave 3.0.1

    scale = sqrt(2)/meandist;

    T = [scale   0   -scale*c(1)
         0     scale -scale*c(2)
         0       0      1      ];

    newpts = T*pts;

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Function 11: hnormalise.m [76]

Input: vector to be normalized

Output: normalized output with a *norm* value of 1.

```
function nx = hnormalise(x)
```

```
    [rows,npts] = size(x);
    nx = x;

    % Find the indices of the points that are not at infinity
    finiteind = find(abs(x(rows,:)) > eps);

    if length(finiteind) ~= npts
        warning('Some points are at infinity');
    end

    % Normalise points not at infinity
    for r = 1:rows-1
    nx(r,finiteind) = x(r,finiteind)./x(rows,finiteind);
    end
    nx(rows,finiteind) = 1;

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Function 12: iscolinear.m [76]

Inputs: points and input flag.

Outputs: flag to show if collinear or otherwise.

```
function r = iscolinear(p1, p2, p3, flag)

    if nargin == 3   % Assume inhomogeneous coords
    flag = 'inhomog';
    end

    if ~all(size(p1)==size(p2)) | ~all(size(p1)==size(p3)) | ...
        ~(length(p1)==2 | length(p1)==3)
        error('points must have the same dimension of 2 or 3');
    end

    % If data is 2D, assume they are 2D inhomogeneous coords. Make
them
    % homogeneous with scale 1.
    if length(p1) == 2
        p1(3) = 1; p2(3) = 1; p3(3) = 1;
    end

    if flag(1) == 'h'
    % Apply test that allows for homogeneous coords with arbitrary
        % scale.  p1 X p2 generates a normal vector to plane defined
by
        % origin, p1 and p2.  If the dot product of this normal with
p3
        % is zero then p3 also lies in the plane, hence co-linear.
    r =  abs(dot(cross(p1, p2),p3)) < eps;
```

```
    else
    % Assume inhomogeneous coords, or homogeneous coords with equal
        % scale.
    r =  norm(cross(p2-p1, p3-p1)) < eps;
    end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Function 13: find_tnfn_eleonly.m

Inputs: list of negative points & locations in pixel coordinates and the homography

Outputs: Lists of true and false negatives.

```
function [TN_list, FN_list] =
find_tnfn_eleonly(negatives_e,rowse1,rowse2,colse1,colse2,H)

    x1 = zeros(size(negatives_e,1),2);
    x2 = x1;
    for i=1:size(negatives_e,1)
        x1(i,:) = [colse1(negatives_e(i,1)) rowse1(negatives_e(i,1))];
        x2(i,:) = [colse2(negatives_e(i,2)) rowse2(negatives_e(i,2))];
    end
    x1(:,3) = 1; x2(:,3) = 1;
    x1_tform = x1*H;
    x1_tform(:,1) = x1_tform(:,1)./x1_tform(:,3);
    x1_tform(:,2) = x1_tform(:,2)./x1_tform(:,3);
    x1_tform(:,3) = 1;
    FN = 0;
    TN = 0;
    TN_list = []; FN_list = [];
    for i=1:size(x1_tform,1)
        x_dist = abs(x1_tform(i,1)-x2(i,1));
        y_dist = abs(x1_tform(i,2)-x2(i,2));
        dist = sqrt(x_dist^2+y_dist^2);
        if x_dist<1 && y_dist<1 && dist<sqrt(1.5)
            FN = FN+1;
            FN_list = [FN_list i];
        else
            TN = TN+1;
            TN_list = [TN_list i];
        end
    end

end
```

References

[1] K. Schmid, K. Waters, L. Dingerson, B. Hadley, R. Mataosky, J. Carter and J. Dare. Lidar 101: An introduction to lidar technology, data, and applications. *NOAA Coastal Services Center* 2008.

[2] A. Wehr and U. Lohr. Airborne laser scanning—an introduction and overview. *ISPRS Journal of Photogrammetry and Remote Sensing 54(2),* pp. 68-82. 1999.

[3] R. N. Faux, J. M. Buffington, M. G. Whitley, S. H. Lanigan and B. B. Roper, "Chapter 6.—Use of airborne near-infrared LiDAR for determining channel cross-section characteristics and monitoring aquatic habitat in pacific northwest rivers: A preliminary analysis," in ***PNAMP Special Publication: Remote Sensing Applications for Aquatic Resource Monitoring***, J. M. Bayer and J. L. Schei, Eds. 2009, pp. 43.

[4] A. Habib, A. P. Kersting, K. I. Bang and D. Lee. Alternative methodologies for the internal quality control of parallel LiDAR strips. *Geoscience and Remote Sensing, IEEE Transactions On 48(1),* pp. 221-236. 2010.

[5] D. Shepard. A two-dimensional interpolation function for irregularly-spaced data. Presented at Proceedings of the 1968 23rd ACM National Conference. 1968, .

[6] A. Okabe, B. Boots, K. Sugihara and S. N. Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams* 2009*501*.

[7] C. F. VanLoan. *Introduction to Scientific Computing* 1997.

[8] M. A. Oliver and R. Webster. Kriging: A method of interpolation for geographical information systems. *International Journal of Geographical Information System 4(3),* pp. 313-332. 1990.

[9] J. Shan and C. K. Toth. *Topographic Laser Ranging and Scanning: Principles and Processing* 2008.

[10] P. J. Besl and N. D. McKay. Method for registration of 3-D shapes. Presented at Robotics-DL Tentative. 1992, .

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

[12] E. Rublee, V. Rabaud, K. Konolige and G. Bradski. ORB: An efficient alternative to SIFT or SURF. Presented at Computer Vision (ICCV), 2011 IEEE International Conference On. 2011, .

[13] S. Leutenegger, M. Chli and R. Y. Siegwart. BRISK: Binary robust invariant scalable keypoints. Presented at Computer Vision (ICCV), 2011 IEEE International Conference On. 2011, .

[14] H. Bay, A. Ess, T. Tuytelaars and L. Van Gool. Speeded-up robust features (SURF). *Comput. Vision Image Understanding 110(3),* pp. 346-359. 2008. . DOI: http://dx.doi.org/10.1016/j.cviu.2007.09.014.

[15] A. Alahi, R. Ortiz and P. Vandergheynst. Freak: Fast retina keypoint. Presented at Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference On. 2012, .

[16] B. Zitova and J. Flusser. Image registration methods: A survey. *Image Vision Comput. 21(11),* pp. 977-1000. 2003.

[17] P. E. Anuta. Spatial registration of multispectral and multitemporal digital imagery using fast fourier transform techniques. *Geoscience Electronics, IEEE Transactions On 8(4),* pp. 353-368. 1970.

[18] D. I. Barnea and H. F. Silverman. A class of algorithms for fast digital image registration. *Computers, IEEE Transactions On C-21(2),* pp. 179-186. 1972. . DOI: 10.1109/TC.1972.5008923.

[19] W. Pratt. Correlation techniques of image registration. *Aerospace and Electronic Systems, IEEE Transactions On AES-10(3),* pp. 353-358. 1974. . DOI: 10.1109/TAES.1974.307828.

[20] B. Zitova and J. Flusser. Image registration methods: A survey. *Image Vision Comput. 21(11),* pp. 977-1000. 2003.

[21] R. N. Bracewell and R. Bracewell. *The Fourier Transform and its Applications* 1986*31999*.

[22] E. De Castro and C. Morandi. Registration of translated and rotated images using finite fourier transforms. *Pattern Analysis and Machine Intelligence, IEEE Transactions On PAMI-9(5),* pp. 700-703. 1987. . DOI: 10.1109/TPAMI.1987.4767966.

[23] M. -. Dubuisson and A. K. Jain. A modified hausdorff distance for object matching. Presented at Pattern Recognition, 1994. Vol. 1 - Conference A: Computer Vision & Image Processing., Proceedings of the 12th IAPR International Conference On. 1994, . DOI: 10.1109/ICPR.1994.576361.

[24] P. C. Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta) 2*pp. 49-55. 1936.

[25] R. Short and K. Fukunaga. The optimal distance measure for nearest neighbor classification. *Information Theory, IEEE Transactions On 27(5),* pp. 622-627. 1981.

[26] A. Baumberg. Reliable feature matching across widely separated views. Presented at Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference On. 2000, . DOI: 10.1109/CVPR.2000.855899.

[27] J. Matas, O. Chum, M. Urban and T. Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image Vision Comput. 22(10),* pp. 761-767. 2004.

[28] E. Tola, V. Lepetit and P. Fua. DAISY: An efficient dense descriptor applied to wide-baseline stereo. *Pattern Analysis and Machine Intelligence, IEEE Transactions On 32(5),* pp. 815-830. 2010. . DOI: 10.1109/TPAMI.2009.77.

[29] F. Schaffalitzky and A. Zisserman. Viewpoint invariant texture matching and wide baseline stereo. Presented at Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference On. 2001, . DOI: 10.1109/ICCV.2001.937686.

[30] J. Artieda, J. M. Sebastian, P. Campoy, J. F. Correa, I. F. Mondragón, C. Martínez and M. Olivares. Visual 3-d slam from uavs. *Journal of Intelligent and Robotic Systems 55(4-5),* pp. 299-321. 2009.

[31] W. Y. Jeong and K. M. Lee. Visual SLAM with line and corner features. Presented at Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference On. 2006, .

[32] P. Jensfelt, D. Kragic, J. Folkesson and M. Bjorkman. A framework for vision based bearing only 3D SLAM. Presented at Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference On. 2006, .

[33] P. Newman and K. Ho. SLAM-loop closing with visually salient features. Presented at Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference On. 2005, .

[34] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman and A. Davison. KinectFusion: Real-time 3D reconstruction and interaction using a moving depth camera. Presented at Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology. 2011, .

[35] M. Pollefeys, D. Nistér, J. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S. Kim and P. Merrell. Detailed real-time urban 3d reconstruction from video. *International Journal of Computer Vision 78(2-3),* pp. 143-167. 2008.

[36] D. Capel. *Image Mosaicing and Super-Resolution* 2004.

[37] R. Szeliski. Image alignment and stitching: A tutorial. *Foundations and Trends® in Computer Graphics and Vision 2(1),* pp. 1-104. 2006.

[38] P. F. McLauchlan and A. Jaenicke. Image mosaicing using sequential bundle adjustment. *Image Vision Comput. 20(9–10),* pp. 751-759. 2002. . DOI: http://dx.doi.org/10.1016/S0262-8856(02)00064-1.

[39] J. Zhang, M. Marszałek, S. Lazebnik and C. Schmid. Local features and kernels for classification of texture and object categories: A comprehensive study. *International Journal of Computer Vision 73(2),* pp. 213-238. 2007.

[40] B. S. Manjunath and W. Ma. Texture features for browsing and retrieval of image data. *Pattern Analysis and Machine Intelligence, IEEE Transactions On 18(8),* pp. 837-842. 1996.

[41] S. Lazebnik, C. Schmid and J. Ponce. A sparse texture representation using local affine regions. *Pattern Analysis and Machine Intelligence, IEEE Transactions On 27(8),* pp. 1265-1278. 2005.

[42] T. Gevers and A. W. Smeulders. Pictoseek: Combining color and shape invariant features for image retrieval. *Image Processing, IEEE Transactions On 9(1),* pp. 102-119. 2000.

[43] D. L. Swets and J. J. Weng. Using discriminant eigenfeatures for image retrieval. *Pattern Analysis and Machine Intelligence, IEEE Transactions On 18(8),* pp. 831-836. 1996.

[44] A. W. Smeulders, M. Worring, S. Santini, A. Gupta and R. Jain. Content-based image retrieval at the end of the early years. *Pattern Analysis and Machine Intelligence, IEEE Transactions On 22(12),* pp. 1349-1380. 2000.

[45] A. Vailaya, M. A. Figueiredo, A. K. Jain and H. Zhang. Image classification for content-based indexing. *Image Processing, IEEE Transactions On 10(1),* pp. 117-130. 2001.

[46] E. Nowak, F. Jurie and B. Triggs. "Sampling strategies for bag-of-features image classification," in *Computer Vision–ECCV 2006*Anonymous 2006, .

[47] O. Chapelle, P. Haffner and V. N. Vapnik. Support vector machines for histogram-based image classification. *Neural Networks, IEEE Transactions On 10(5),* pp. 1055-1064. 1999.

[48] S. Belongie, J. Malik and J. Puzicha. Shape matching and object recognition using shape contexts. *Pattern Analysis and Machine Intelligence, IEEE Transactions On 24(4),* pp. 509-522. 2002.

[49] J. Mutch and D. G. Lowe. Multiclass object recognition with sparse, localized features. Presented at Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference On. 2006, .

[50] D. G. Lowe. Object recognition from local scale-invariant features. Presented at Computer Vision, 1999. the Proceedings of the Seventh IEEE International Conference On. 1999, .

[51] A. E. Johnson and M. Hebert. Recognizing objects by matching oriented points. Presented at Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference On. 1997, . DOI: 10.1109/CVPR.1997.609400.

[52] M. A. Turk and A. P. Pentland. Face recognition using eigenfaces. Presented at Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference On. 1991, .

[53] J. W. Tanaka and J. A. Sengco. Features and their configuration in face recognition. *Mem. Cognit. 25(5),* pp. 583-592. 1997.

[54] R. Brunelli and T. Poggio. Face recognition: Features versus templates. *Pattern Analysis and Machine Intelligence, IEEE Transactions On 15(10),* pp. 1042-1052. 1993. . DOI: 10.1109/34.254061.

[55] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry and Y. Ma. Robust face recognition via sparse representation. *Pattern Analysis and Machine Intelligence, IEEE Transactions On 31(2),* pp. 210-227. 2009.

[56] J. F. Canny. Finding edges and lines in images. *Massachusetts Inst.of Tech.Report 1*1983.

[57] C. Harris and M. Stephens. A combined corner and edge detector. Presented at Alvey Vision Conference. 1988, .

[58] A. Makadia, A. Patterson and K. Daniilidis. Fully automatic registration of 3D point clouds. Presented at Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference On. 2006, . DOI: 10.1109/CVPR.2006.122.

[59] L. G. Brown. A survey of image registration techniques. *ACM Computing Surveys (CSUR) 24(4),* pp. 325-376. 1992.

[60] K. Mikolajczyk and C. Schmid. Indexing based on scale invariant interest points. Presented at Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference On. 2001, .

[61] Y. Dufournaud, C. Schmid and R. Horaud. Matching images with different resolutions. Presented at Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference On. 2000, .

[62] T. Lindeberg. Feature detection with automatic scale selection. *International Journal of Computer Vision 30(2),* pp. 79-116. 1998.

[63] W. T. Freeman and E. H. Adelson. The design and use of steerable filters. *IEEE Trans. Pattern Anal. Mach. Intell. 13(9),* pp. 891-906. 1991.

[64] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions On 27(10),* pp. 1615-1630. 2005.

[65] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. Presented at Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference On. 2005, .

[66] G. Yu and J. Morel. ASIFT: An algorithm for fully affine invariant comparison.

[67] K. Mikolajczyk and C. Schmid. Scale & affine invariant interest point detectors. *International Journal of Computer Vision 60(1),* pp. 63-86. 2004.

[68] Y. Li and E. B. Olson. A general purpose feature extractor for light detection and ranging data. *Sensors 10(11),* pp. 10356-10375. 2010.

[69] J. Shi and C. Tomasi. Good features to track. Presented at Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference On. 1994, .

[70] M. Bosse and R. Zlot. Keypoint design and evaluation for place recognition in 2D lidar maps. *Robotics and Autonomous Systems 57(12),* pp. 1211-1224. 2009.

[71] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions On 27(10),* pp. 1615-1630. 2005. . DOI: 10.1109/TPAMI.2005.188.

[72] T. Lindeberg. Scale-space theory: A basic tool for analysing structures at different scales. Presented at Journal of Applied Statistics. 1994, .

[73] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun ACM 24(6),* pp. 381-395. 1981.

[74] Y. Ke and R. Sukthankar. PCA-SIFT: A more distinctive representation for local image descriptors. Presented at Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference On.

[75] S. Lazebnik, "Computer Vision CS543/ECE549 Spring 2013," 2013.

[76] P. D. Kovesi. MATLAB and octave functions for computer vision and image processing. 2000. Available: http://www.csse.uwa.edu.au/~pk/research/matlabfns/.

[77] A. Habib, K. Bang, A. P. Kersting and D. Lee. Error budget of LiDAR systems and quality control of the derived data. *Photogramm. Eng. Remote Sensing 75(9),* pp. 1093-1108. 2009.

Biographical Information

Mythreya Jayendra Lakshman received the B.E. degree in electronics and communication engineering from SSN College of Engineering (affiliated to the Anna University in Chennai, India), in 2006; and the M.S. degree in electrical engineering with a focus on digital circuits and systems from the University of Texas at Dallas in 2008. He is currently working towards the doctoral degree at the University of Texas at Arlington, Arlington, TX, with a focus on computer vision and image processing applications.