The Pennsylvania State University
The Graduate School

MOTION ESTIMATION FROM RANGE IMAGE PAIRS

A Thesis in
Electrical Engineering
by
Donald J. Natale III

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

December 2008

The thesis of Donald J. Natale III was reviewed and approved* by the following:

Richard L. Tutwiler
Professor Electrical Engineering
Thesis Advisor

W. Kenneth Jenkins
Professor Electrical Engineering
Department Head

Kultegin Aydin
Professor of Electrical Engineering
Graduate Program Coordinator

*Signatures are on file in the Graduate School.

Abstract

As rapidly as new sensor technologies become available to researchers, the usefulness of their integration into intelligent systems is explored. High resolution 3D video cameras are becoming available to researchers for the first time. These cameras give us the ability to measure 3D geometry at video frame rates.

The goal of this work is to estimate the 6 degrees of freedom motion transformation of a moving 3D camera between range image pairs. The intended application of this ability lies in autonomous navigation, where the motion of a vehicle could potentially be computed without the use of external sensors or baselines. This would give machines the ability to navigate much in the same way humans do, using visual cues to make running estimates of their motion.

Motion estimation techniques which use intensity images or stereo vision systems have not yet reached the levels of precision needed for generalized application in obstacle avoidance, crash prevention, or high speed navigation applications. Previous incarnations of 3D range cameras either did not operate at high enough frame rates, or were not of high enough resolution to be a practical solution to the problem of machine vision in autonomous navigation [2].

Starting with a formulation of a popular optic flow algorithm which had been modified by Horn and Harris for potential use with range imagery [5], a robust algorithm will be developed and tested with real data acquired with a novel high resolution 3D flash LADAR (LAser Detection And Ranging) video camera.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Machine Vision

As rapidly as new sensor technologies become available to researchers, the usefulness of their integration into intelligent systems is explored. In the fields of computer vision and image processing this is especially true, as accomplishing in hardware and software what people so easily do with their eyes and minds is a particularly daunting problem. While the magnitude of this problem is large, so would be the utility of its solution. To create machines that can see and navigate as adeptly as humans is an enticing prospect. Naturally, the notion of creating machines which can exceed human abilities is even more so.

1.2 Overview Motion Estimation

The major theme of this work is to determine the motion of an observer with respect to the world, using only what the observer sees. The intended application of this ability lies in autonomous navigation, where the motion of a vehicle could

potentially be computed without the use of external sensors or baselines. More specifically, an autonomous or semi-autonomous vehicle would be fitted with a forward-facing camera and as the vehicle moves the camera will take a sequence of images. Between each pair of images, a camera transformation will be computed. The ensemble of camera transformations could be filtered with an estimator to determine the vehicle's path. These images alone would then be used to compute the vehicle's motion [2], [5].

1.2.1 Classical Techniques

Motion estimation techniques which use intensity images or stereo vision systems have not yet reached the levels of precision or speed needed for generalized application in obstacle avoidance, crash prevention, or high speed navigation applications [2].

Motion estimation from sequences of intensity images is typically accomplished by computing the optic flow under the *brightness constancy assumption* and assuming that this approximates the underlying motion field. For a brief summary of how the problem is formulated, consider a sequence of images in time represented by intensity $\mathbf{I}(\mathbf{X}, \mathbf{Y}, t)$. The assumption we then make is that the brightness of a pixel on a moving object will remain constant in time. We know this assumption to be false in all but the most contrived of scenarios, but it must be made to get any solution at all. We will constrain the problem so that $\frac{d\mathbf{I}}{dt} = 0$. Using the chain rule to take the derivative with time gives rise to the *brightness constancy equation* [4],[13].

$$\frac{\partial \mathbf{I}}{\partial \mathbf{X}} \frac{d\mathbf{X}}{dt} + \frac{\partial \mathbf{I}}{\partial \mathbf{Y}} \frac{d\mathbf{Y}}{dt} + \frac{\partial \mathbf{I}}{\partial t} = 0 \tag{1.1}$$

Where $\frac{\partial \mathbf{I}}{\partial \mathbf{X}}, \frac{\partial \mathbf{I}}{\partial \mathbf{Y}}$ are the spatial gradients of a frame, $\frac{\partial \mathbf{I}}{\partial t}$ is the time derivative

between frames, and $\frac{d\mathbf{X}}{dt}, \frac{d\mathbf{Y}}{dt}$ are what we want to compute. We now have a linear equation in two unknowns. Representing this equation another way,

$$(\frac{\partial \mathbf{I}}{\partial \mathbf{X}}, \frac{\partial \mathbf{I}}{\partial \mathbf{Y}}) \cdot (\frac{d\mathbf{X}}{dt}, \frac{d\mathbf{Y}}{dt}) = -\frac{\partial \mathbf{I}}{\partial t} \tag{1.2}$$

we see that while it is not possible to determine motion in the direction of iso-brightness contours (perpendicular to the brightness gradient), but we can express the motion in the direction of the brightness gradient as

$$\frac{\frac{\partial \mathbf{I}}{\partial t}}{|\frac{\partial \mathbf{I}}{\partial \mathbf{X}}, \frac{\partial \mathbf{I}}{\partial \mathbf{Y}}|} \tag{1.3}$$

By using finite differences to approximate the partial derivative we can solve for $\frac{d\mathbf{X}}{dt}, \frac{d\mathbf{Y}}{dt}$ least-squares style or with a more robust estimator [6].

As Horn and others have shown since 1981, the results tend to be good in spite of the strength of the brightness constancy assumption [6], [7], [3], [12]. The mathematical formulation of this estimation requires that the image have generally smooth spatial first derivatives. The major flaw of this technique is not with the formulation or even it's ability to predict accurate optic flow, but that in all but the most controlled of environments, optic flow is not a very good approximation of the motion field. The motion field is what we would like to know. Consider the case in which a stationary object is illuminated by a moving light source. While the intensity gradients move in time, the actual motion field is zero. Consider the case of a rotating sphere with a purely specular surface under fixed illumination. The optic flow will appear to be zero, while the underlying motion field is non-zero [4]. An even more fundamental problem is present. This technique only estimates the movement of pixels on a 2D image plane. More work needs to be done to figure out what this motion represents in the world. Even if this is accomplished, in the absence of range information for the scene, the motion field can only be

determined up to a constant.

1.2.2 The Advantage of Using Range Data

While we as human observers successfully employ a simple stereo vision system for our navigation needs, we are also able to refine our motion estimations by incorporating a lot of other experiential data. For example, we know if a particular object in a scene is being occluded, then it is most likely behind the object which is occluding it. We also have a general intuition about the relative sizes of objects. If two objects of known size are present in a scene and one looks much smaller than the other then we can assume one is farther away than the other. It is presently too difficult to design a machine system which shares this ability, so a different mechanism must be exploited to achieve accurate range determination. Rather then computing range indirectly from intensity images, it is advantageous to measure it directly.

Range data refers to a complete geometric description of a scene from the point of view of the observer. Each pixel in a frame contains metric information about the cross-range, down-range and elevation of the point being represented. We can use a similar technique to that described previously in the summary of optic flow computation, except instead of using an intensity image $\mathbf{I}(\mathbf{X}, \mathbf{Y}, t_i)$ where intensity is a function of $\mathbf{X}$ and $\mathbf{Y}$ at time t_i, each frame of data is a *Range Image* represented by a function $\mathbf{Z}(\mathbf{X}, \mathbf{Y}, t_i)$. Since this direct measurement of range to pixels in a scene is not affected by changes in brightness, moving light sources, shadows, or changes in color, this is a much more reliable set of data from which to compute estimates of the motion field.

1.3 The Proposed Motion Estimation System

This work will focus on computing the 6 Degrees of Freedom (DoF) motion of a moving camera between two image frames. To do this an adapted version of the classical optic flow technique previously discussed will be used instead with range imagery. In the future, these camera transformations will be supplied to an estimator in order to generate a corrected overall motion description. This is an important feature to consider for a final system, because the motion transformations computed between frames will be noisy, and are only computed between pairs of frames. Estimators capable of tracking 6 DoF like those Welch uses in his SCAAT tracking work would be appropriate for this application [15]. Welch has demonstrated robust 6 DoF tracking using a twelve state filter which includes linear positions and velocities as well as angular displacements and angular velocities. Since absolute angle can not be represented as a linear combination of the state variables, absolute angle is kept track of externally using quaternions.

See Figure 1.1 for a top level flow diagram of the future proposed system. Here, range data is taken with a suitable range sensor or camera. The range data is processed and range maps (images) are generated. Two successive range maps are used to compute an apparent 6 degree of motion camera transformation. The camera transformations computed are then fed into an estimator which offers a refined motion estimate. It is with this estimator that all of the independent motion transformations are considered as an ensemble to generate a motion estimate and track the vehicle.

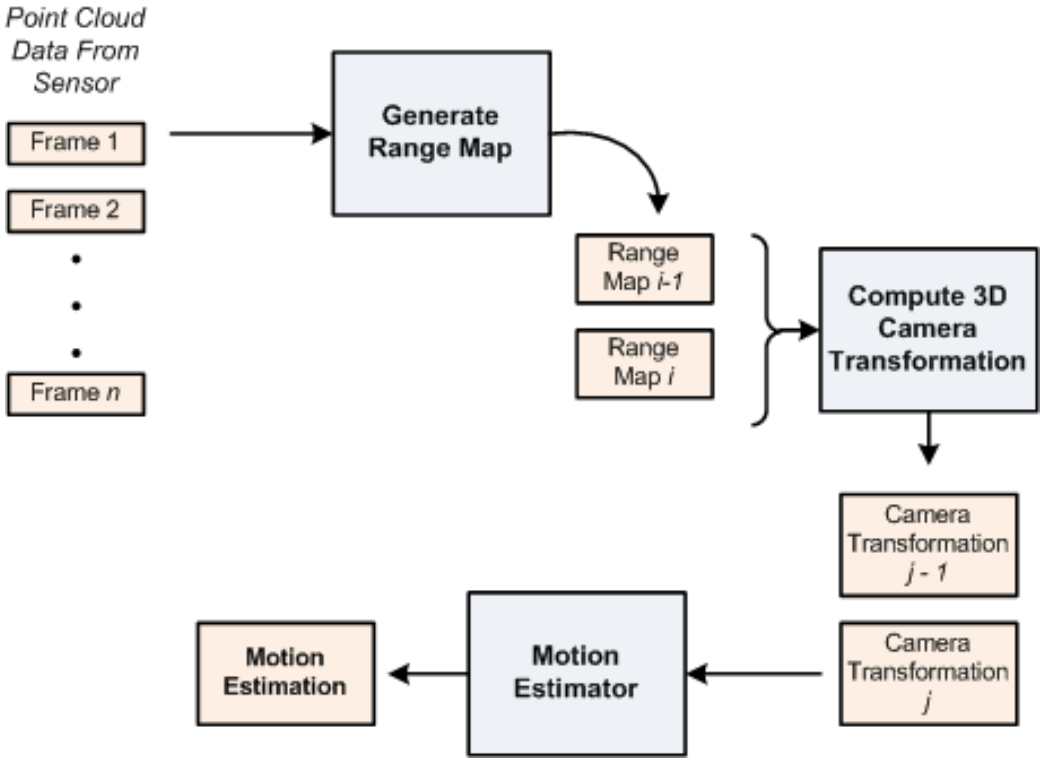

Figure 1.1. Top Level System Description

1.4 Thesis Outline

This chapter has given a brief overview of motion estimation by computer vision techniques and proposed a system in which the algorithms developed in the remainder of this work could be implemented. Chapter 2 will discuss the generation and preprocessing of range images that will be necessary to prepare the range data for use in the subsequent algorithms. Problems that could occur in the data will also be discussed. In Chapter 3, the derivation of Horn and Harris' range rate constraint equation will be performed and explained. This constraining set of equations will also be solved for camera transformation parameters in this chapter. Chapter 4 will discuss optimizations and other practical concerns which must be considered when solving the range rate constraint and implementing it as part of

a motion estimation algorithm. Most notably, Chapter 4 will explain why it is desirable to formulate the solution to the range rate constraint in a Random Sample Consensus construct. Chapter 5 will show results of using the algorithm on real range data taken of simple terrain.

Chapter 2

Generating Range Data

2.1 Generating Range Data

Passive Stereo is perhaps the most intuitive method for trying to generate range data, but has flaws which make it less suitable than other less intuitive solutions. Rapidly advancing ranging and range camera technology has afforded a number of options for generating accurate range imagery [11]. Range cameras currently available can be subdivided into two categories, single beam scanning systems (conventional LIDAR) and matrix rastering systems (structured light cameras, flash LADAR) [2].

2.1.1 Passive Stereo

The typical passive stereo system is comprised of two intensity cameras separated by a known baseline, T_x. In a simple stereo system, two cameras of identical focal length f are oriented parallel to each other and separated by a known baseline distance T_x. See Figure 2.1 The location in camera coordinates of the perspective

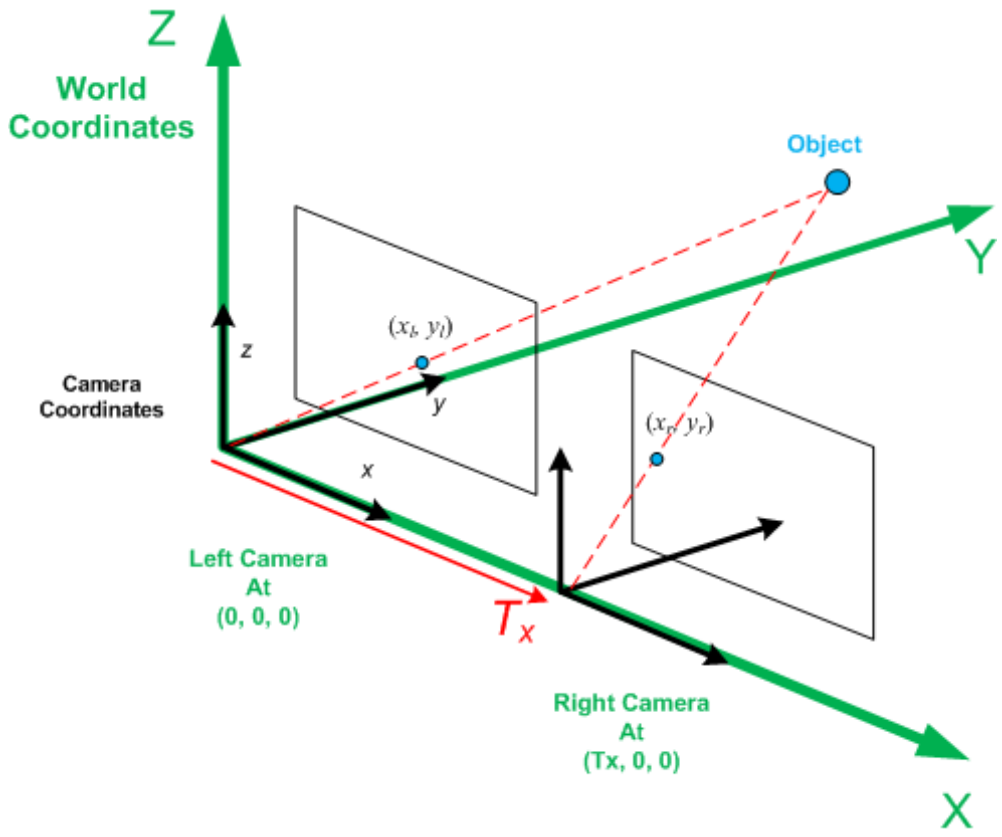

Figure 2.1. A Simple Passive Stereo System - Two cameras of identical focal length are separated by a known baseline T_x.

projection of the object on the image plane of the left camera is given by

$$x_l = f\frac{X}{Z} \quad \text{and} \quad y_l = f\frac{Y}{Z} \tag{2.1}$$

and the location in camera coordinates of the perspective projection of the object on the image plane of the right camera is given by

$$x_r = f\frac{X - T_x}{Z} \quad \text{and} \quad y_r = f\frac{Y}{Z} \tag{2.2}$$

We can compute the apparent stereo disparity d between the instance of the object in both image planes

$$d = x_l - x_r = f\frac{X}{Z} - f\frac{X - T_x}{Z} = f\frac{T_x}{Z} \tag{2.3}$$

Since we know f and T_x and can compute d from the images, we can solve for the objects distance in world coordinate Z.

For objects at close range, the stereo disparity d is large and Z can be computed accurately, but for more distant objects d is small and thus the range computation is less accurate. The accuracy of range estimates using stereo is also directly proportional to the baseline distance T_x which presents a practical problem. To make T_x large, one needs a physically larger platform on which to mount the cameras.

Thus far it has gone unmentioned, but a major difficulty with passive stereo is that the same object needs to be located in both images before any range calculations can take place. This correspondence matching adds another layer of complexity to motion estimation algorithms [9]. Often this is done using a feature matching technique. To achieve a robust result, the computation of a large number of cross-correlations or similar measures is required. The combinatorics of this technique scale rapidly and lead to large computational overhead times. Additionally, the accuracy of the range estimate decreases as does the baseline distance between cameras as well as with the square of distance from the camera system [13],[4].

2.1.2 Conventional Scanning LIDAR

Most LIDAR (LIght Detection And Ranging) systems use a single beam laser to make accurate range measurements to a single point in space. This beam is

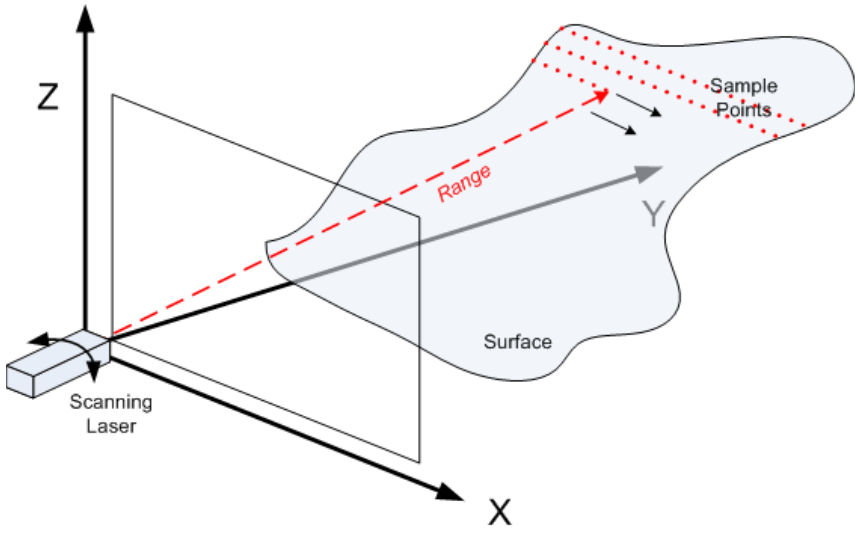

Figure 2.2. Conventional Scanning Lidar - As the laser is raster scanned across the scene, the range to each sample point is measured, one at a time.

mechanically scanned across the scene taking a dense range sampling of the scene. See Figure 2.2. While LIDAR systems function well as a range sensor, the scanning time inherent in the acquisition of an entire scene limits its use as a camera. Horn and Harris discuss this as a problem in using LIDAR data for motion estimation, as the scan times were such that their vehicle could move an average of 8 feet between image frames, a displacement too large for their algorithm to handle directly. Motion estimation with low frame rates necessitates the use of external sensors to pre-register the range images [5].

2.1.3 Structured Light Cameras

The most common of matrix rastering systems is the structured light camera. These systems are currently used with great precision in many close-range applications such as printed circuit board verification and botanical growth measurements [8], [10]. Structured light systems work by using a light source and a camera as an active stereo pair. See Figure 2.3. A light pattern is projected onto a scene and then

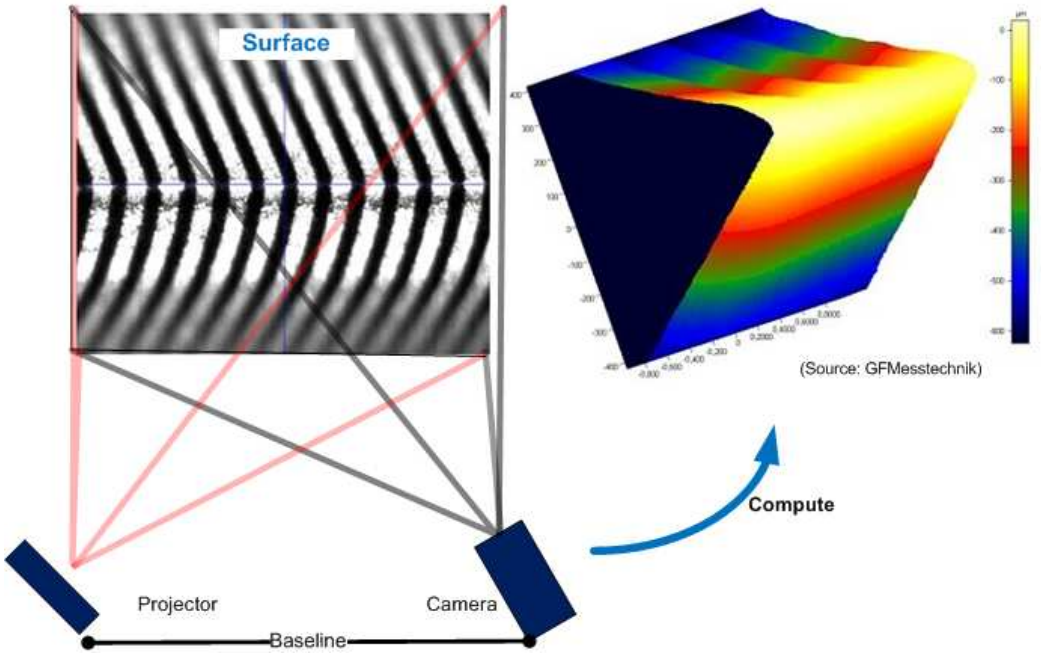

Figure 2.3. Structured Light System - A known light pattern is projected onto the surface. An image is taken with a passive camera and the distance to the object is triangulated by analyzing the distortion of the pattern

the scene is recorded with a passive intensity camera. The distortions of the light pattern are indicative of the 3D geometry of the scene. This is an improvement over passive stereo systems in that the problem of finding point correspondences between two images is avoided completely. Similarly to passive stereo systems, range is still computed indirectly from the data, presenting analytical challenges [14]. Also like passive stereo systems, if range is computed by triangulation the feature of accuracy decreasing with the square of distance is shared as well.

2.1.4 3D Flash LADAR

The most appealing technological option for application in motion estimation is the flash LADAR (LAser Detection and Ranging). A flash LADAR system uses a beam-spreader to illuminate an entire scene with a single laser pulse. A multi-

pixel array of receiver elements detects and measures the modulation shift of the returning laser light and computes range. What makes this technology appealing is that like a scanning LIDAR system, it is capable of directly outputting range information, known as a point cloud, but unlike a scanning LIDAR system, the range measurements for all pixels in a scene are returned simultaneously. From a systems engineering perspective, a flash LADAR is comparable in ease of use and data rate to a conventional digital video camera. Very recent developments in flash LADAR technology have produced range cameras with focal plane array sizes on the order of 128 x 128 pixels. These LADAR cameras also operate at video frame rates [11]. These cameras are ideal for measuring range data to be used in motion estimation systems. For an illustration of how flash LADAR systems work, see Figure 2.4.

Specifically, the flash LADAR used in this work is the Advanced Scientific Concepts Inc. Portable 3D Camera. This unit is designed to measure accurately within a range of between $10ft$ and $2000ft$ with measurement precision of ± 3". The camera was fitted with an 85 mm lens. The focal plane array has 128 x 128 pixels at $100\mu m$ pitch. The laser was air cooled and eye safe with a wavelength of $1.56\mu m$. The beam was spread to cover a 9^o field of view. The entire unit was 6" x 6" x 11" and weighed less than 12.5 lbs. Maximum power consumption was less than 400W and it was powered by a conventional 12V dry cell motorcycle battery.

2.2 Preprocessing Range Data

2.2.1 Cartesian Elevation Maps (CEM)

Recent algorithms for recovering the six degrees of motion of a vehicle use data in the form of a Cartesian elevation map [2]. This is a more convenient format for data processing than the spherical or psudo-spherical formats returned by both

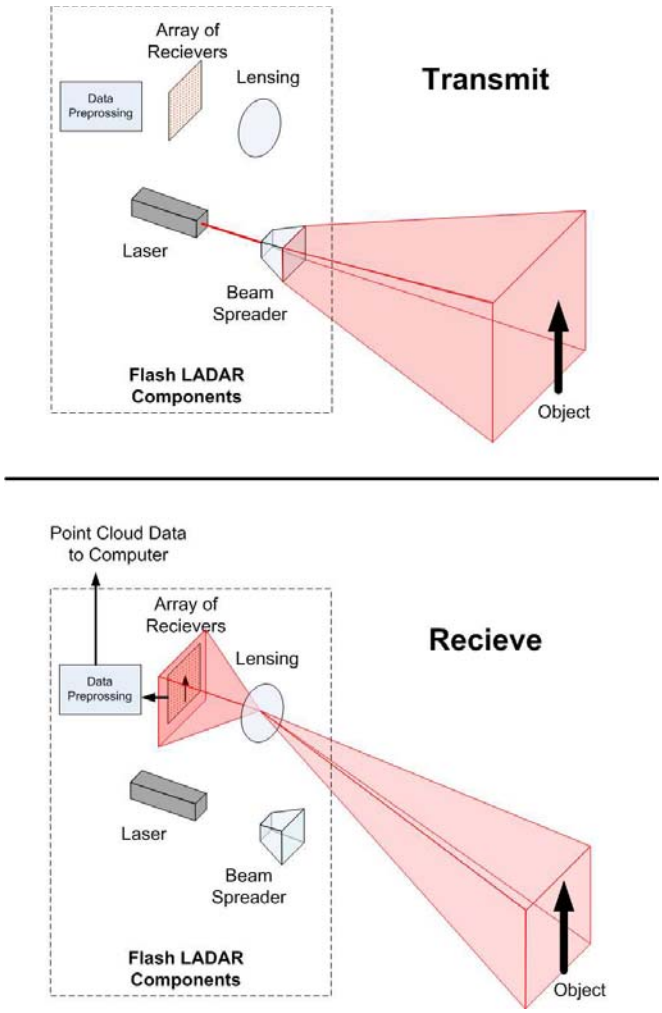

Figure 2.4. Illustration of a Flash LADAR System - Upon being triggered to take a range image, the laser is pulsed and the beam is spread out into a plane. The plane of laser light reflects off the objects in the scene and returns to the camera. The light is focused on a focal plane of receiver elements, each capable of independently detecting modulation shifts. The measurements are pre-processed into range data by an Application Specific Integrated Circuit (ASIC) and the data and the point cloud data outputs to a computer.

scanning LIDARs and flash LADARs [5]. We would also like to use image processing techniques on the data, and most of these techniques are developed in Cartesian coordinates.

The data acquired by the ASC flash LADAR was outputted in tabular format with receiver element (pixel) indices i, j forming the first two columns and *range*

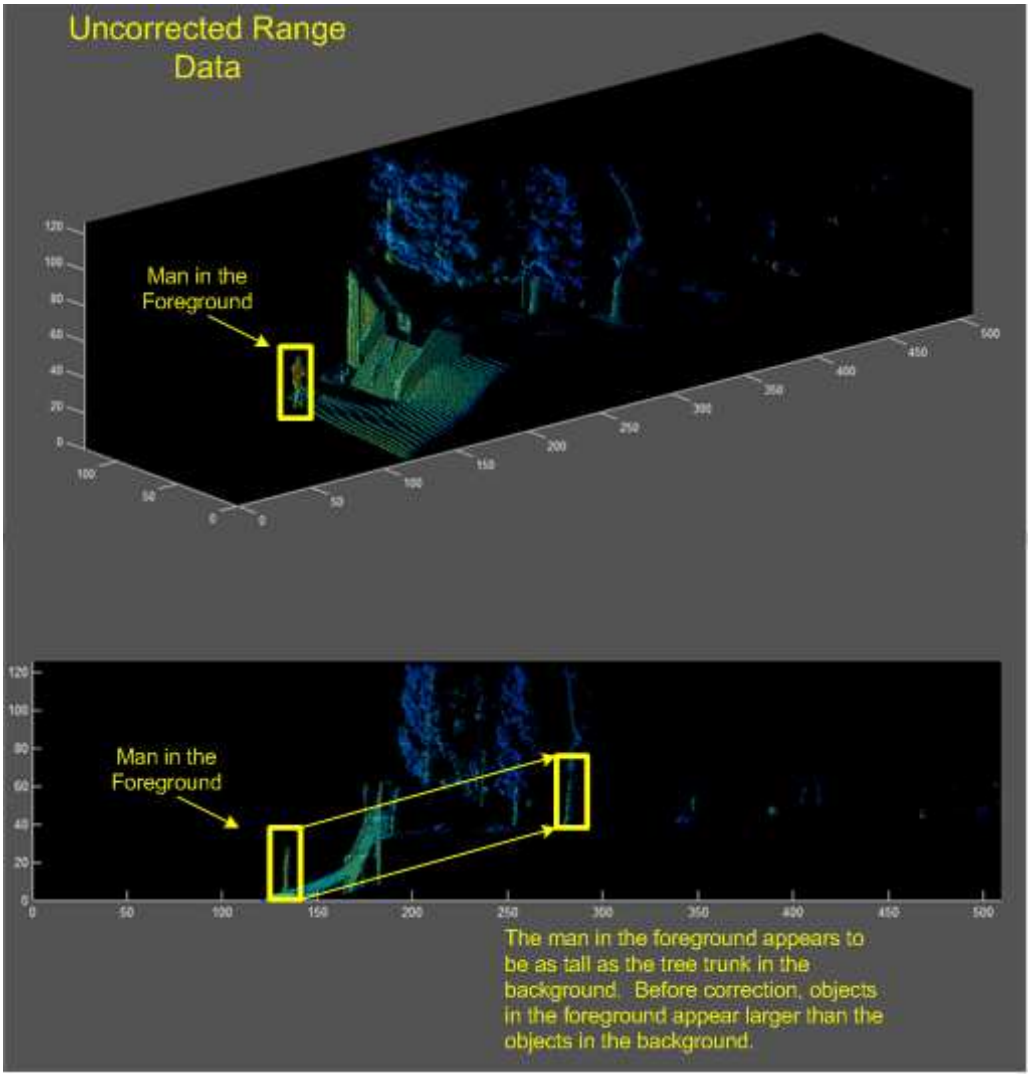

Figure 2.5. Raw Range Data Prior to Geometric Correction

and *intensity* forming the third and forth. Plotting this data directly gives an image like in Figure 2.5. The elevation and cross range axes are element indices on the 128 x 128 receiver array. The down-range axis is the distance in feet from the element to the point in the scene. As is plain to see, the sizes of the objects in the image are not correct. The man in the foreground looks to be as tall as the tree trunk in the background.

We must account for the optics and geometry of the receiver array in order to generate an accurate map of the scene where elevation, cross range, and down range axes are all in feet and objects have the proper proportion in the foreground and the background. A diagram of the receiver geometry is shown in Figure 2.6. We are given *i, j, Range* and would like to compute *X, Y, Z* distances in feet.

First we must locate the pixel distance *pl* from the center of the receiver. Where i and j are the pixel row indices and column indices respectively, and p_p is the pixel pitch represented in feet, so as to maintain correct dimensionality,

$$\begin{aligned} y &= (i - 64)p_p \\ x &= (j - 64)p_p \end{aligned} \tag{2.4}$$

and thus

$$\begin{aligned} pl &= \sqrt{x^2 + y^2} \\ \theta &= \arctan\left(\frac{pl}{fl}\right) \\ \phi &= \arctan\left(\frac{y}{x}\right) \end{aligned} \tag{2.5}$$

Care must be taken upon implementation to ensure that the sign of the angle is correct when computing the arctan functions. The MATLAB command *atan2* will compute these functions with the correct sign automatically. Also, note that half of row/column element number was subtracted from i and j. This was done to move the center of the new Cartesian coordinate axes to the center of the camera. In this way, all cross range locations to the right of the center of the camera will be positive, all elevation measurements above the center of the camera will be positive, and all down range measurements will be positive.

Next we can use θ and *Range* to compute Z and *Il* as so,

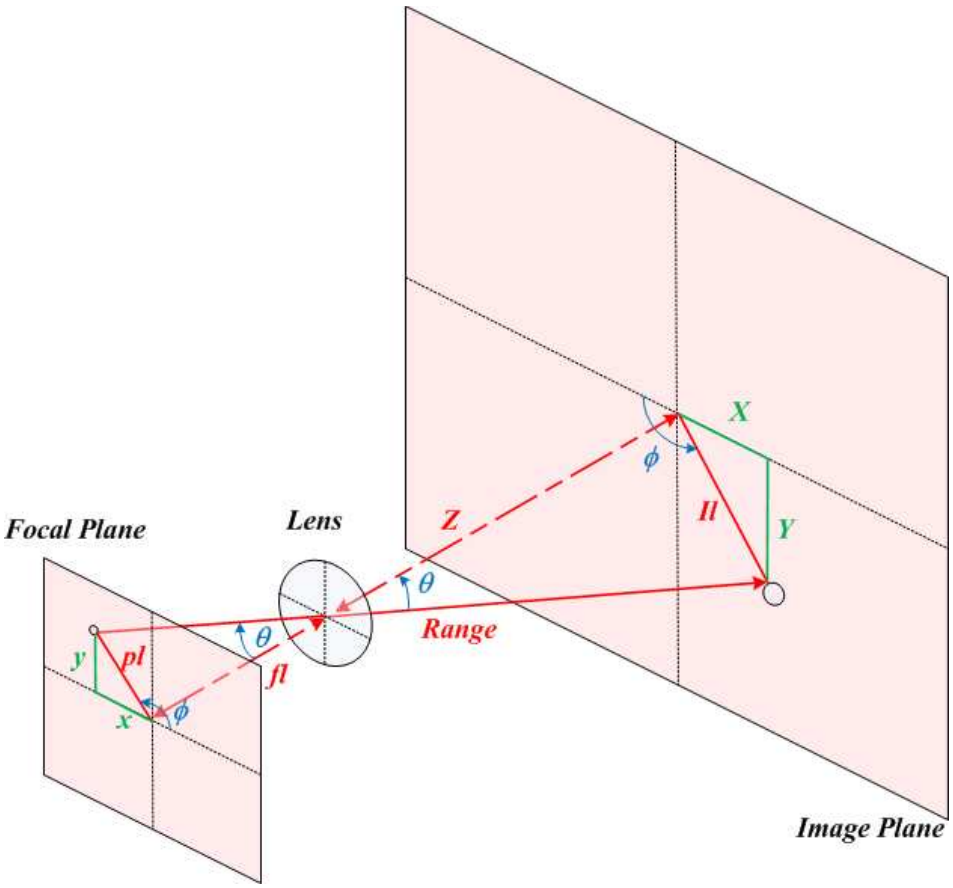

Figure 2.6. Range Correction - *fl* is the focal length of the lens, x and y are the components of *pl*, the distance from the center of the focal plane array to the center to element (i,j). Since x, y, and *Range* are known, real world distances X, Y, and Z are easily computed.

$$\begin{aligned} Z &= Range\sin(\theta) \\ Il &= Range\cos(\theta) \end{aligned} \tag{2.6}$$

and then using Il and ϕ we can compute X and Z,

$$\begin{aligned} X &= Il\cos(\phi) \\ Y &= Il\sin(\phi) \end{aligned} \tag{2.7}$$

This geometric correction was written as a MATLAB function *RangeCorrect* and the code can be found in the Appendix. The result of applying this geometric

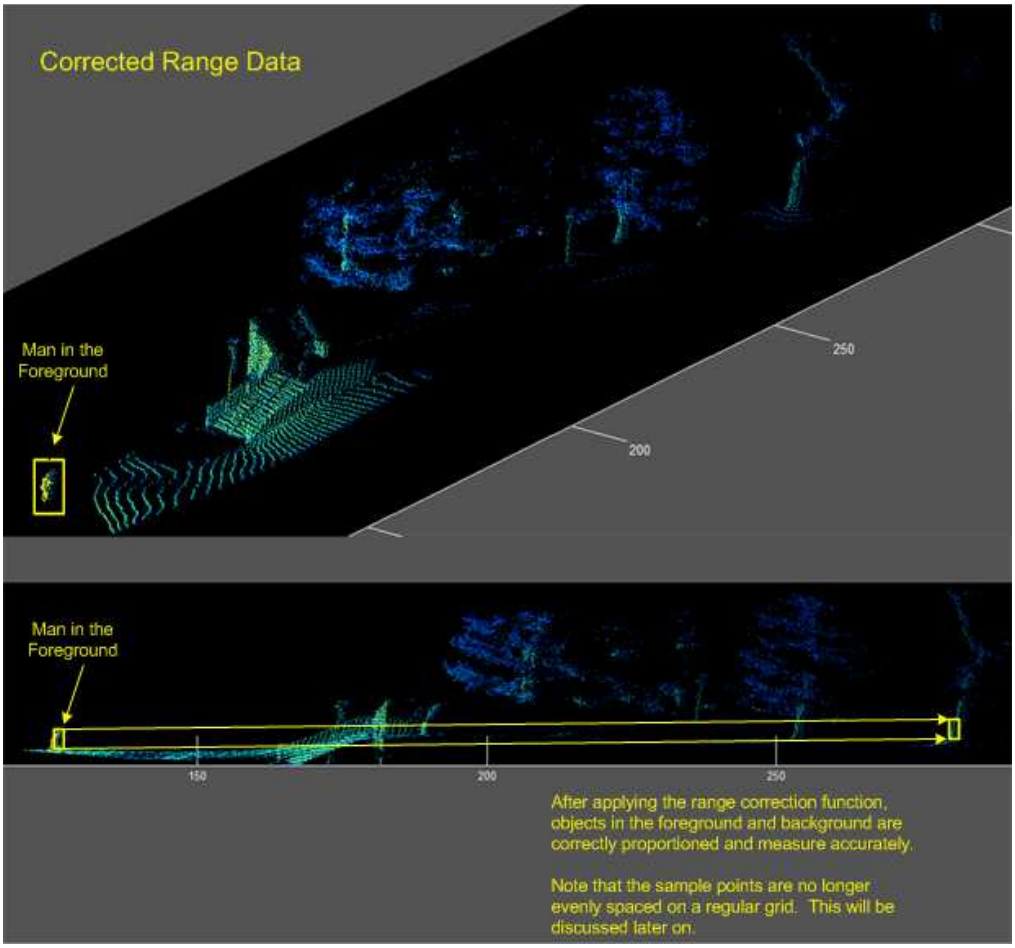

Figure 2.7. Range Map after Geometric Correction

correction is shown in Figure 2.7.

2.2.2 Interpolation and Re-sampling

As can be seen by comparing Figure 2.5 and 2.7, once the geometric correction is applied, the sampled points in the range image are no longer regularly spaced. Points farther down-range in the image are sampled more sparsely than points at close range. This creates difficulty when applying linear operators to the image by convolution or correlation, particularly if a finite difference kernel is applied to compute spatial gradients. There are two ways of solving this problem. The

most immediate is to interpolate and re-sample the range image on a regularized grid. The other solution is to derive all subsequent computations to work on a non-uniform grid. The Horn and Harris adaptation of optic flow to range flow has been modified by Spies to accommodate a non-regular grid [10]. This was done by Spies because measurement precision was the greatest priority in his work. For this work, modularity of each component of the motion estimation system, flexibility, and proof of concept are the top priorities, so interpolation and re-sampling to a regular grid is a more sensible solution at this stage.

Spies and Gharavi both explore a membrane fit interpolation to regularize the sample points. This interpolation model is based on computing an estimate e in a specified region A based on measurements m to minimize an energy function

$$\int_A h(e)dxdy \rightarrow \min \tag{2.8}$$

where

$$\begin{gathered} h(e) = \omega(e-m)^2 + \alpha(e_x^2 + e_y^2) \\ e_x = \frac{\partial e}{\partial x}; \quad e_y = \frac{\partial e}{\partial y} \end{gathered} \tag{2.9}$$

This function can be minimized by solving the Euler-Lagrange equations for each pixel, and in a discrete implementation the resulting Laplacian can be estimated by the difference between a local average and central value of e [2],[10].

Ultimately the most important consideration is that whatever interpolation is used, it offers a smooth fit such that first spatial derivatives are defined. Delaunay triangulation between sample points with cubic polynomial interpolation satisfies this requirement for continuity. For this work, the fitting of the non-regularly spaced data to a regular grid was done using the MATLAB function *griddata* with cubic interpolation. This method is based on Delaunay triangulation and does not create any discontinuities in the zeroth or first derivatives.

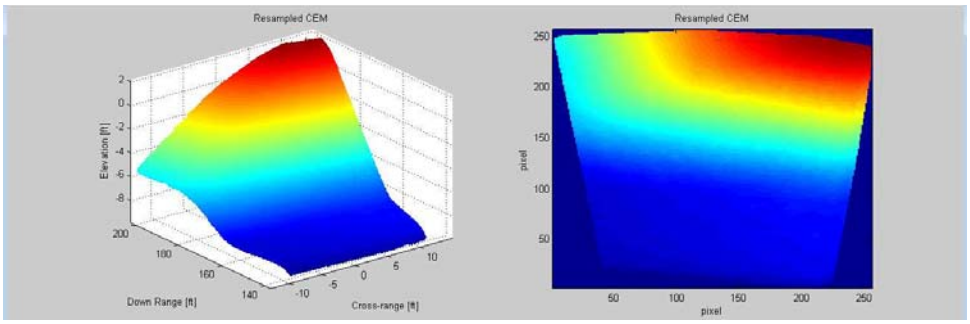

Figure 2.8. The Interpolated CEM - The dark blue areas around the borders of the data matrix are NaN. These areas should not be included in any calculations.

2.3 Getting Rid of Problem Regions

2.3.1 Non-Rectangular Maps

The result of gridding the range corrected point cloud data is shown in Figure 2.8. As mentioned before, objects farther down-range were sampled more sparsely compared to objects at close range, so we can expect greater interpolation error down-range. Also, note that the geometric correction has taken what was a rectangular image and turned it into an arc. The dark blue areas in Figure 2.8 are areas that were never sampled and hold NaN values. Rather than dealing with a non-rectangular CEM, it makes more sense at this point to crop a rectangular region out of the center since MATLAB's image processing functions are largely designed to operate on rectangular images. If one was to use a non-rectangular CEM, more sophisticated border handling methods would have to be used when convolving the data with various filter kernels. It is true that using as much of the sampled data as possible would make sense in a final system prototype, but working with rectangular CEM of uniform size offers more flexibility for research purposes. An example of the final CEMs used for motion estimation is shown in Figure 2.9.

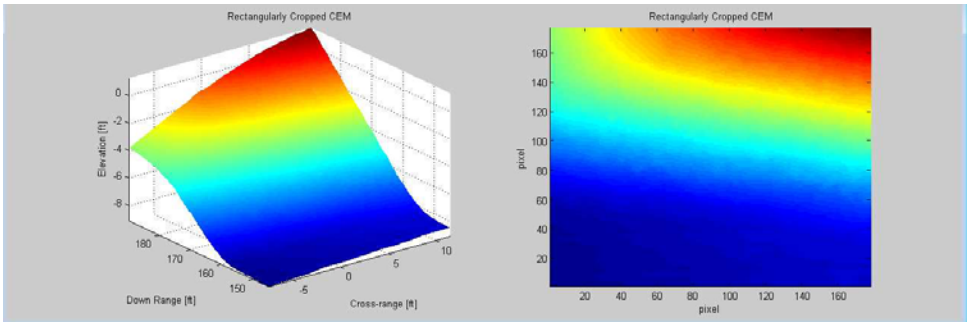

Figure 2.9. Final CEM - A rectangular region was cropped from the center of the non-rectangular CEM to avoid the NaN problem areas around the borders of the data matrix.

2.3.2 Gaussian Noise

We can assume that our measurements are corrupted by zero mean additive noise. In order to reduce this noise we could convolve the range image with a box filter to take an average over neighborhoods of predetermined size. The box filter however creates high-frequency artifacts in the image due to the side lobes present in its frequency-domain equivalent, the *sinc* function. For image applications the Gaussian is a preferable filter kernel due to the absence of these high frequency artifacts. 2D Gaussian kernels are also separable into two 1D Gaussian kernels, and filtering with two 1D Gaussian kernels is computationally more efficient than filtering with one 2D Gaussian kernel. A sketch of this proof is provided below

Let G_x and G_x be Gaussian kernels of equal variance σ^2 and let F be another function. The associative property of convolution states

$$G_x * (G_y * F) = (G_x * G_y) * F \tag{2.10}$$

If we take a closer look at $(G_x * G_y)$ a useful property is manifested. When $G_x = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2}{2\sigma^2}}$ and $G_y = \frac{1}{2\pi\sigma^2} e^{-\frac{y^2}{2\sigma^2}}$ then

$$\mathcal{F}_x[G_x]\mathcal{F}_y[G_y] = k_1 e^{-2\pi^2\sigma^2u^2} k_1 e^{-2\pi^2\sigma^2v^2} \Leftrightarrow k_2 e^{-\frac{(x^2+y^2)}{2\sigma^2}} \tag{2.11}$$

The 2D Gaussian kernel can be expressed as the convolution of two 1D Gaussian kernels, so we can efficiently smooth the range images by convolution with two 1D Gaussian kernels [13]. With regard to choosing a good window size for the kernel, it should be larger than the features in the image you wish to blur, and smaller than the features you do not want to blur. With regard to choosing a good value for σ^2, this value should be larger than the estimated variance of the Gaussian noise in your image.

2.3.3 Shot Noise

Pixel level shot noise can occur for a variety of reasons, most obviously due to the malfunction of a pixel in the receiver array. This kind of artifact is typically not zero mean and can't be averaged away without affecting other pixel values in the immediate neighborhood. The technique used to remove shot noise from the data was non-linear median filtering. This technique consists of computing the median value over a small neighborhood of pixels, and then assigning the center of that neighborhood this median value. The median filtering can be implemented using the function *medfilt2* in MATLAB. Median filtering removes shot noise while limiting distortions to edges in the range image [13]. The size of the median filter kernel should be chosen such that it is larger than the size of the shot noise, but smaller than the smallest features in the image you wish to maintain.

Chapter 3

Rigid Body Motion Estimation

3.1 Classes of Motion Estimation Algorithms

Motion estimation techniques using range data can be divided into two general classes. For applications such as botanical growth measurements [10], algorithms capable of estimating the motion of deforming surfaces are needed. In other applications where the objects in the scene are non-deformable, rigid body motion estimation algorithms are used [5], [2]. Our desired application is autonomous navigation and most of the objects which will comprise the range images will be terrain, vehicles, and building structures, all of which are rigid bodies. The rigid body assumption also allows for the imposition of important constraints which allow the motion equations to be solved more readily. It is thus that rigid body motion estimation algorithm are explored exclusively in this work.

3.2 Range Rate Constraint

We begin in a manner identical to Horn and Harris [5], by defining $\mathbf{R}$ as a vector from the camera to a point on the terrain. A right-handed coordinate system is

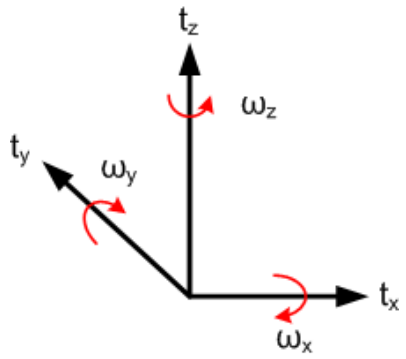

Figure 3.1. Coordinate System Definition

used where Z is elevation, X is cross-range, and Y is down range. See Figure 3.1 for a description of the coordinate system. Figure 3.2 illustrates the definitions of the parameters used in this derivation. $\mathbf{R}$ is expressed in sensor-centered Cartesian coordinates. The point on the terrain will appear to move with velocity

$$\dot{\mathbf{R}} = -\mathbf{t} - \omega \times \mathbf{R} \tag{3.1}$$

In this expression $\mathbf{t}$ is a vector comprised of the translational velocity components t_x, t_y, and t_z. Rotational velocity is denoted by ω, which also has three components.

$$\mathbf{t} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}, \quad \omega = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{3.2}$$

We are only interested in V_n, the component of $\dot{\mathbf{R}}$ in the normal direction of the tangent plane to the surface point. V_n is easily computed after determining unit normals $\hat{\mathbf{n}}$ to every point on the surface map. It has magnitude

$$V_n = -\dot{\mathbf{R}} \cdot \hat{\mathbf{n}} = -\mathbf{t} \cdot \hat{\mathbf{n}} - [\omega \mathbf{R} \hat{\mathbf{n}}] \tag{3.3}$$

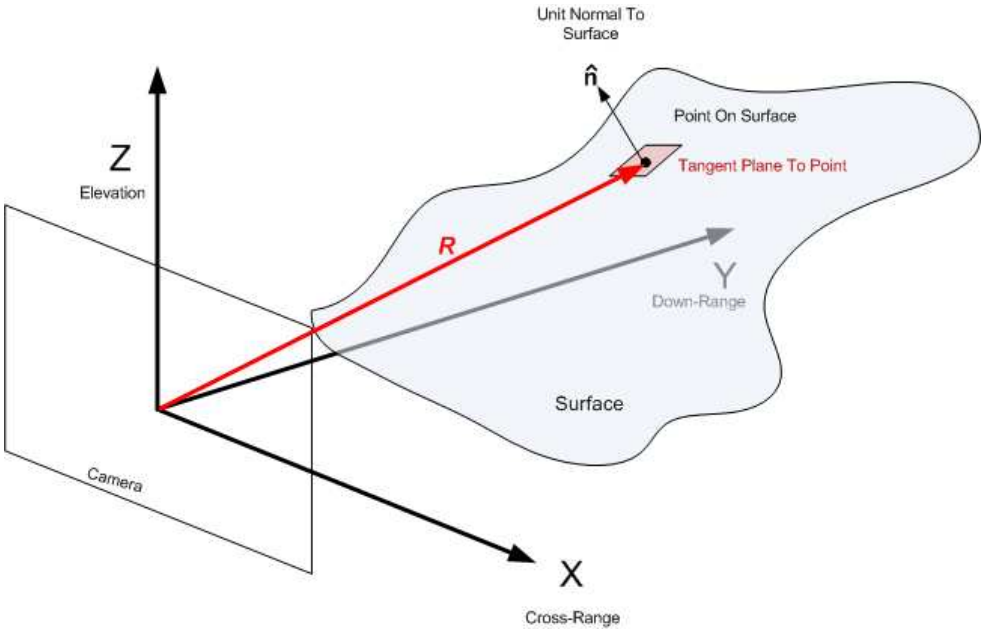

Figure 3.2. Illustration of the Parameters used in the Derivation

Note that [**abc**] is being used to denote the triple product $\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c})$ which is by identity equivalent to $(\mathbf{a} \times \mathbf{b}) \cdot \mathbf{c}$ [4].

Using a finite approximation of a derivative (for example, a finite backwards difference between the current image frame and the previous) we can estimate $\dot{\mathbf{R}}$, the rate at which the range of each pixel changes. Using this estimation, the velocity at which the image points appear to move can be constrained to the form

$$\dot{\mathbf{R}} = R_t \hat{\mathbf{r}} + \mathbf{s} \tag{3.4}$$

Where $\hat{\mathbf{r}}$ is a unit vector in the direction of $\mathbf{R}$ and $\mathbf{s}$ is any arbitrary vector in the tangent plane. This vector can be arbitrary since $\mathbf{s}$ is orthogonal to $\hat{\mathbf{n}}$ and we will next be computing the dot product of Equation 3.4 with $\hat{\mathbf{n}}$ next. The vector R_t is an estimate of the range derivative with respect to time. We will approximate this derivative by computing the finite difference between a frame of data with the

previous frame in time. The normal component of velocity is hence

$$V_n = \dot{\mathbf{R}} \cdot \hat{\mathbf{n}} = R_t(\hat{\mathbf{r}} \cdot \hat{\mathbf{n}}) \tag{3.5}$$

Equating this derivation of V_n with the previous, we get the following:

$$R_t(\hat{\mathbf{r}} \cdot \hat{\mathbf{n}}) + \mathbf{t} \cdot \hat{\mathbf{n}} + [\omega \mathbf{R} \hat{\mathbf{n}}] = 0 \tag{3.6}$$

Which Horn and Harris calls the *range rate constraint equation.* Horn and Harris also point out that this equation is similar to the *brightness change constraint equation* used in some optic flow algorithms. It is interesting to note that this problem has the same dimensionality as optic flow, except rather than using brightness as the third dimension, here range is used. This gives us an advantage in estimating the motion field. Optic flow is forced to make the assumption that the underlying motion field exhibited in a sequence of image frames is equivalent to the observed optic flow. This of course is not always the case, for example in the presence of moving light sources, moving brightness patterns and shadows will be observed. Non-zero optical flow will be exhibited even if the underlying motion field is static. Since the flash LADAR measures actual scene geometry, the observed range flow is equivalent to the underlying motion field, provided that the scene geometry is sufficiently complex that normals to its surface span the space of 3D motion in general, or at least the space of the expected camera transformation. Another way of thinking about this is that at least three, and ideally more than three surface normals are not coplanar.

3.3 Solution of the Range Rate Constraint Equation

We are interested in solving Equation 3.6 for the components of $\mathbf{t}$ and ω which make the observed range rate R_t equivalent at all image pixels. For various reasons it is likely that such a solution does not exist. In practice, the assumption that the entire scene moves as a rigid assemblage will frequently be false. Any object in the scene which moves with respect to world coordinates will invalidate the rigidity assumption. Also, measurements can be expected to be noisy and the finite approximation of R_t in Equation 3.6 may be inaccurate. We also have available many more image pixels available than are needed to compute the rigid transformation. Therefore, choosing a suitable norm and minimizing the error over the entire collection of image pixels makes sense. Minimizing the mean-squared error will be a sufficient example. The double integral here implies that the errors are being minimized over the area of the range image.

$$\iint (R_t(\hat{\mathbf{r}} \cdot \hat{\mathbf{n}}) + (\mathbf{t} \cdot \hat{\mathbf{n}} + [\omega \mathbf{R} \hat{\mathbf{n}}]))^2 d\alpha d\beta \tag{3.7}$$

Since the range rate constraint equation is linear in both of the unknown parameters $\mathbf{t}$ and ω, and the integrand of this expression is the square of a linear term, the minimized solution can be determined analytically by differentiating the integrand with respect to each parameter and setting the resulting partial derivatives equal to zero [5]:

$$\begin{aligned} \iint (R_t(\hat{\mathbf{r}} \cdot \hat{\mathbf{n}}) + (\mathbf{t} \cdot \hat{\mathbf{n}} + [\omega \mathbf{R} \hat{\mathbf{n}}]))\hat{\mathbf{n}} d\alpha d\beta = \mathbf{0} \\ \iint (R_t(\hat{\mathbf{r}} \cdot \hat{\mathbf{n}}) + (\mathbf{t} \cdot \hat{\mathbf{n}} + [\omega \mathbf{R} \hat{\mathbf{n}}]))(\mathbf{R} \times \hat{\mathbf{n}}) d\alpha d\beta = \mathbf{0} \end{aligned} \tag{3.8}$$

For the sake of a cleaner representation, let $\mathbf{c} = (\mathbf{R} \times \hat{\mathbf{n}})$. Employing the identities $(\mathbf{a} \cdot \mathbf{b}) = (\mathbf{b} \cdot \mathbf{a})$ and $(\mathbf{a} \cdot \mathbf{b}) = \mathbf{a}^T \mathbf{b}$ Equations 3.8 can be rearranged as

follows:

$$\begin{aligned}\left(\iint \hat{\mathbf{n}}\hat{\mathbf{n}}^T d\alpha d\beta\right)\mathbf{t} + \left(\iint \hat{\mathbf{n}}\mathbf{c}^T d\alpha d\beta\right)\omega &= -\iint R_t(\hat{\mathbf{r}}\cdot\hat{\mathbf{n}})\hat{\mathbf{n}} d\alpha d\beta, \\ \left(\iint \mathbf{c}\hat{\mathbf{n}}^T d\alpha d\beta\right)\mathbf{t} + \left(\iint \mathbf{c}\mathbf{c}^T d\alpha d\beta\right)\omega &= -\iint R_t(\hat{\mathbf{r}}\cdot\hat{\mathbf{n}})\mathbf{c} d\alpha d\beta\end{aligned} \tag{3.9}$$

Since we will be using digital images, the double integrals will be double summations over the all n_r rows and all n_c columns of the range image as so

$$\begin{aligned}\left(\sum_{i=1}^{n_r}\sum_{j=1}^{n_c}\hat{\mathbf{n}}_{ij}\hat{\mathbf{n}}_{ij}^T\right)\mathbf{t} + \left(\sum_{i=1}^{n_r}\sum_{j=1}^{n_c}\hat{\mathbf{n}}_{ij}\mathbf{c}_{ij}^T\right)\omega &= -\sum_{i=1}^{n_r}\sum_{j=1}^{n_c} R_{t_{ij}}(\hat{\mathbf{r}}_{ij}\cdot\hat{\mathbf{n}}_{ij})\hat{\mathbf{n}}_{ij}, \\ \left(\sum_{i=1}^{n_r}\sum_{j=1}^{n_c}\mathbf{c}_{ij}\hat{\mathbf{n}}_{ij}^T\right)\mathbf{t} + \left(\sum_{i=1}^{n_r}\sum_{j=1}^{n_c}\mathbf{c}_{ij}\mathbf{c}_{ij}^T\right)\omega &= -\sum_{i=1}^{n_r}\sum_{j=1}^{n_c} R_{t_{ij}}(\hat{\mathbf{r}}_{ij}\cdot\hat{\mathbf{n}}_{ij})\mathbf{c}_{ij}\end{aligned} \tag{3.10}$$

For a cleaner representation and more concise programming the range image pixels are lexicographically ordered column-wise by indice so that if the original image was of dimension $[n_r \times n_c]$ the new image is of dimension $[n'_r \times n'_c]$ where $n'_c = 1$ and $n'_r = n_r n_c = n_p$, which is the total number of pixels in the image.

$$\begin{aligned}\underbrace{\sum_{i=1}^{n_p}\hat{\mathbf{n}}_i\hat{\mathbf{n}}_i^T}_{3\times3}\ \mathbf{t} + \underbrace{\sum_{i=1}^{n_p}\hat{\mathbf{n}}_i\mathbf{c}_i^T}_{3\times3}\ \omega &= -\underbrace{\sum_{i=1}^{n_p} R_{t_i}(\hat{\mathbf{r}}_i\cdot\hat{\mathbf{n}}_i)\hat{\mathbf{n}}_i}_{3\times1}, \\ \underbrace{\sum_{i=1}^{n_p}\mathbf{c}_i\hat{\mathbf{n}}_i^T}_{3\times3}\ \mathbf{t} + \underbrace{\sum_{i=1}^{n_p}\mathbf{c}_i\mathbf{c}_i^T}_{3\times3}\ \omega &= -\underbrace{\sum_{i=1}^{n_p} R_{t_i}(\hat{\mathbf{r}}_i\cdot\hat{\mathbf{n}}_i)\mathbf{c}_i}_{3\times1}\end{aligned} \tag{3.11}$$

Each summation in under-brackets is a 3×3 matrix, as $\hat{\mathbf{n}}_i$ and $\mathbf{c}_i$ each have an x, y, and z component and are 3×1 vectors. The vectors $\mathbf{t}$, ω are 3×1 vectors. The arguments to the right of both equalities are 3×1 vectors since R_t is a scalar, as is the result of computing the dot product between $\hat{\mathbf{r}}_i$ and $\hat{\mathbf{n}}_i$. Expressed in matrix form this system of 6 equations and 6 unknowns becomes

$$\begin{bmatrix} \sum_{i=1}^{n_p} \hat{\mathbf{n}}_i \hat{\mathbf{n}}_i^T & \sum_{i=1}^{n_p} \hat{\mathbf{n}}_i \hat{\mathbf{c}}_i^T \\ \sum_{i=1}^{n_p} \mathbf{c}_i \hat{\mathbf{n}}_i^T & \sum_{i=1}^{n_p} \mathbf{c}_i \mathbf{c}_i^T \end{bmatrix} \begin{bmatrix} \mathbf{t} \\ \omega \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n_p} R_{t_i}(\hat{\mathbf{r}}_i \cdot \hat{\mathbf{n}}_i)\hat{\mathbf{n}}_i \\ \sum_{i=1}^{n_p} R_{t_i}(\hat{\mathbf{r}}_i \cdot \hat{\mathbf{n}}_i)\mathbf{c}_i \end{bmatrix} \tag{3.12}$$

Again, with $\mathbf{c} = (\mathbf{R} \times \hat{\mathbf{n}})$. For ease of reference in the future, let the above equation be represented as

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{3.13}$$

This concludes the derivation and solution of Horn and Harris' range rate constraint equation [5]. This will be used as a starting point to develop a robust version of a 6 DoF camera transformation estimator.

Chapter 4

Practical Considerations and Optimizations

4.1 Existence of a Solution

The minimization of the range rate constraint equation is computed over all of the pixels in the image. In most cases the complexity of the scene will give a highly overdetermined system of equations and **A** will be invertible. In some cases **A** will not be invertible, or the system will be poorly conditioned and the solution will be inaccurate. The insufficiency of the geometry in the scene to span 3D motion is a result of what is known as the Aperture Problem.

4.1.1 The Aperture Problem

As discussed earlier, when using the brightness constancy equation in classical optic flow estimation, it is the case that portions of the motion field can not be determined. Referring to Equation 1.1, consider the measurable part of the motional field, **v**.

$$\frac{\frac{\partial \mathbf{I}}{\partial t}}{|\frac{\partial \mathbf{I}}{\partial \mathbf{X}}, \frac{\partial \mathbf{I}}{\partial \mathbf{Y}}|}\mathbf{v} = \mathbf{v_n} \tag{4.1}$$

The measurable part of the motion field is the normal component to the brightness gradient. Components of the motion field orthogonal to the brightness gradient are not constrained, and can not be determined. This is often referred to as the Aperture problem [13].

Let us more closely examine Equation 3.6, the Range Rate Constraint equation, which is the 3D analog to the brightness constancy equation.

$$R_t(\hat{\mathbf{r}} \cdot \hat{\mathbf{n}}) + \mathbf{t} \cdot \hat{\mathbf{n}} + (\omega \times \mathbf{R}) \cdot \hat{\mathbf{n}} = 0 \tag{4.2}$$

The dot product of each quantity in the equation with $\hat{\mathbf{n}}$ means here that the observable motion is only that which is projected onto the spatial gradient. For this reason, motion of a point orthogonal to the surface normal will be unobservable. Another way of thinking about this is that the surface normals computed over a region of the range image may not contain a sufficient basis to describe a 3D motion transformation. This occurs when all of the surface normals in a region lie in a plane with each other. Refer to Figure 4.1. In Case 1 when the region is a single plane, we only have one surface normal available to view. Any in-plane motion will be undetectable, as its projection on the normal is zero. Any motion other than in-plane motion will appear to be in the direction of the normal to the plane, even if it has other component directions. This is called plane flow. In Case 2 there are two surface normals which allow us to locate two dimensional flow. Any motion that can be decomposed into components which are perpendicular to the edge can be determined, because this motion can be fully described by a linear combination of the two surface normals we can see, however any motion parallel to the line will have a zero projection onto the surface normals, and thus will not

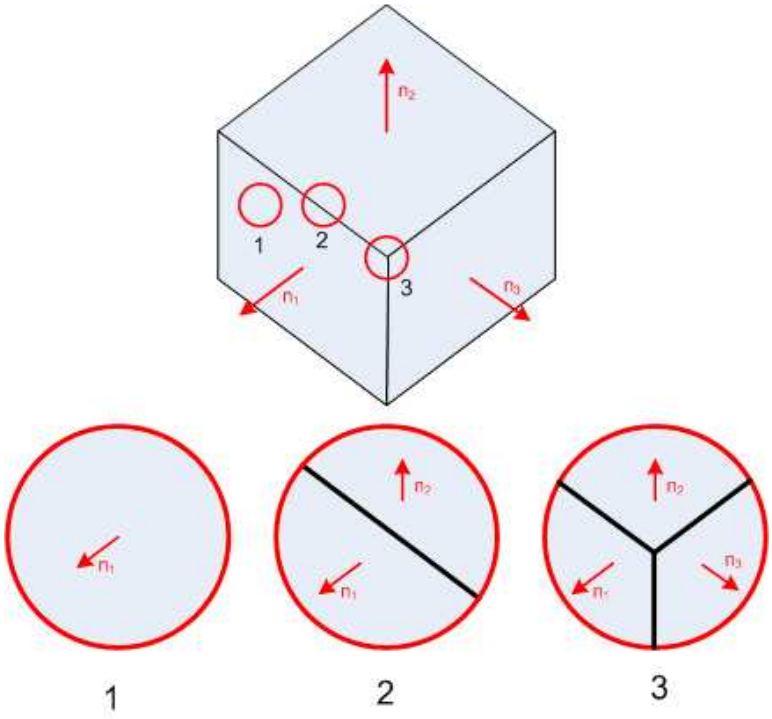

Figure 4.1. 3D Aperture Problem - Cases 1, 2, and 3 show 3 different order of geometry which can be present in the range image. Motion can only be determined if it can be decomposed into a linear combination of the observable surface normals.

be observable. Case 3 is the intersection of 3 planes. We have 3 surface normals available and any motion which can be described as a linear combination of these three surface normals is observable. There is no motion which can occur in 3D space that can not be described by a spanning set of components, hence we will have a complete description of the 3D flow of this neighborhood of points, however if these vectors are close to coplanar, the solution can become inaccurate [10], [13].

The effect of this 3D aperture problem on our camera transformation estimate is that the determination of the transformation parameters may be inaccurate or incomplete if there doesn't exist a set of at least 3 surface normals which are nearly orthogonal to each other. This also means that the smaller the field of view of the camera, the more likely we will suffer the affects of the aperture problem.

4.1.2 Determining if an Accurate Solution Exists Prior to Computing A

If it is the case that the solution will be inaccurate due to under-defined geometry, it would be useful to be able to determine so before going through the effort to construct the matrix $\mathbf{A}$. By definition, the inversion of $\mathbf{A}$ will be inaccurate if the ratio of its largest to smallest singular value is large. This ratio is referred to as the condition number of $\mathbf{A}$.

Consider $\mathbf{A}$ as a block-partitioned matrix with blocks $\mathbf{W}$, $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$ all of which are of dimension $[n \times n]$, it can be shown that the following is true,

$$\mathbf{A}^{-1} = \begin{bmatrix} \mathbf{W} & \mathbf{X} \\ \mathbf{Y} & \mathbf{Z} \end{bmatrix}^{-1} = \begin{bmatrix} (\mathbf{W} - \mathbf{X}\mathbf{Z}^{-1}\mathbf{Y})^{-1} & -\mathbf{W}^{-1}\mathbf{X}(\mathbf{Z} - \mathbf{Y}\mathbf{W}^{-1}\mathbf{X})^{-1} \\ -\mathbf{Z}^{-1}\mathbf{Y}(\mathbf{W} - \mathbf{X}\mathbf{Z}^{-1}\mathbf{Y})^{-1} & (\mathbf{Z} - \mathbf{Y}\mathbf{W}^{-1}\mathbf{X})^{-1} \end{bmatrix} \tag{4.3}$$

Noting that this inversion depends on the inverse of block $\mathbf{W}$ and $\mathbf{X}$, it follows that for $\mathbf{A}^{-1}$ to exist, it is a necessary condition that both $\mathbf{W}^{-1}$ and $\mathbf{Z}^{-1}$ exist. It also follows that if $\mathbf{W}$ and $\mathbf{Z}$ are poorly conditioned, the inversions will be inaccurate, and the inversion of $\mathbf{A}$ will also be inaccurate. In our case,

$$\begin{aligned} \mathbf{W} &= \sum_{i=1}^{n_p} \hat{\mathbf{n}}_i \hat{\mathbf{n}}_i^T \\ \mathbf{Z} &= \sum_{i=1}^{n_p} (\mathbf{R}_i \times \hat{\mathbf{n}}_i)(\mathbf{R}_i \times \hat{\mathbf{n}})_i^T \end{aligned} \tag{4.4}$$

Since we would have to construct the blocks $\mathbf{W}$ and $\mathbf{Z}$ anyway, it makes sense to construct these blocks first and then check their condition numbers. If the condition is larger than some predefined threshold, it may not be worth constructing and inverting the whole of the $\mathbf{A}$ matrix, and the next pair of image frames could be analyzed instead. In this way less computational effort will be spent on im-

ages that don't have sufficiently non-coplanar surface normals for producing good motion estimates. The definition of this threshold could be based on a minimum accuracy requirement for the solution, or could be determined through empirical trials. Also, if one desires to use a stochastic version of the algorithm in a Bayesian formulation, the conditions of the matrix blocks could perhaps be used as a way to assign precision values to the estimates made from the image pairs. These precision value would then be used to weight the prior versus the observation as estimates of probability density functions for the motion transformations are made.

4.2 Technique for Efficient Computation of A in MATLAB

The most straightforward method for computing **A** and **b** in Equation 3.13 is apparent when we rearrange the equation and pull the summations outside of the matrix blocks as so,

$$\sum_{i=1}^{n_p} \begin{bmatrix} \hat{\mathbf{n}}_i \hat{\mathbf{n}}_i^T & \hat{\mathbf{n}}_i (\mathbf{R}_i \times \hat{\mathbf{n}}_i)^T \\ (\mathbf{R}_i \times \hat{\mathbf{n}}_i)\hat{\mathbf{n}}_i^T & (\mathbf{R}_i \times \hat{\mathbf{n}}_i)(\mathbf{R}_i \times \hat{\mathbf{n}})_i^T \end{bmatrix} \begin{bmatrix} \mathbf{t} \\ \omega \end{bmatrix} = \sum_{i=1}^{n_p} \begin{bmatrix} R_{t_i}(\hat{\mathbf{r}}_i \cdot \hat{\mathbf{n}}_i)\hat{\mathbf{n}}_i \\ R_{t_i}(\hat{\mathbf{r}}_i \cdot \hat{\mathbf{n}}_i)(\mathbf{R}_i \times \hat{\mathbf{n}}_i) \end{bmatrix} \tag{4.5}$$

We can set up a FOR loop which iterates from 1 to n_p and at each iteration add the j^{th} instance to the $(j-1)^{th}$ instance of the matrix on the left and the vector on the right. Since n_p is equal to the number of pixels in the range image, and we are interested in dealing with high resolution range images, this for loop must execute for a long period of time. The computation of **A** and **b** alone took MATLAB an average of 0.2619 seconds on the computer used to implement the algorithm.

For this calculation $\hat{\mathbf{n}}$, $\hat{\mathbf{r}}$, and $(\mathbf{R} \times \hat{\mathbf{n}})$ for each pixel are stored as matrices of dimension $3 \times n_p$. The rows of the matrix correspond to the x, y, and z components of **R**. The columns are the **R** vectors for each pixel. The R_t are stored as a $1 \times n_p$

vector. To compute **A** and **b** without the use of FOR loops, all of the matrices of dimension $3 \times n_p$ were transformed into 3 dimensional arrays of dimension $3 \times 1 \times n_p$ and the vector of dimension $1 \times n_p$ was transformed into a 3D array of dimension $1 \times 1 \times 3$ All matrix multiplications and dot products were then computed between the 1^{st} and 2^{nd} dimension of the arrays, n_p times in parallel. Each of the resulting $a \times b \times n_p$ arrays were summed along the 3^{rd} dimension. See Figure 4.2 for a pictorial description of this programming approach. This implementation without FOR loops produces identical results to the FOR loop implementation, but executed an average of 22 times faster. The parallelism of this approach also lends itself to hardware implementation, which is better suited to the intended final application of this work. In the final application, these computations will be done in hardware where parallelism can be exploited to increase the speed of the computations.

4.3 Robustness of LS Solution

The least squares solution to an over-determined system is usually adequate when the data is relatively free of outliers. It is the case that in the presence of outliers, a bad data point exerts a stronger influence on the solution than a good data point [13]. If the nature of the noise is truly unbiased then the effect of all the outliers should sum to zero. In practice, the outliers tend to have a bias and will affect the solution. For example, a non-functioning receiver element of the focal plane will always create an outlier in the same place frame after frame. Another example; the range to a specular surface will measure as the distance to the objects in the reflection rather than the distance to the surface. This is because the laser light reflects from the specular surface to some out of scene object, reflects off of the object back to the specular surface, and then back to the camera. The time of flight of the photon reflects this much longer trip rather than the distance to the

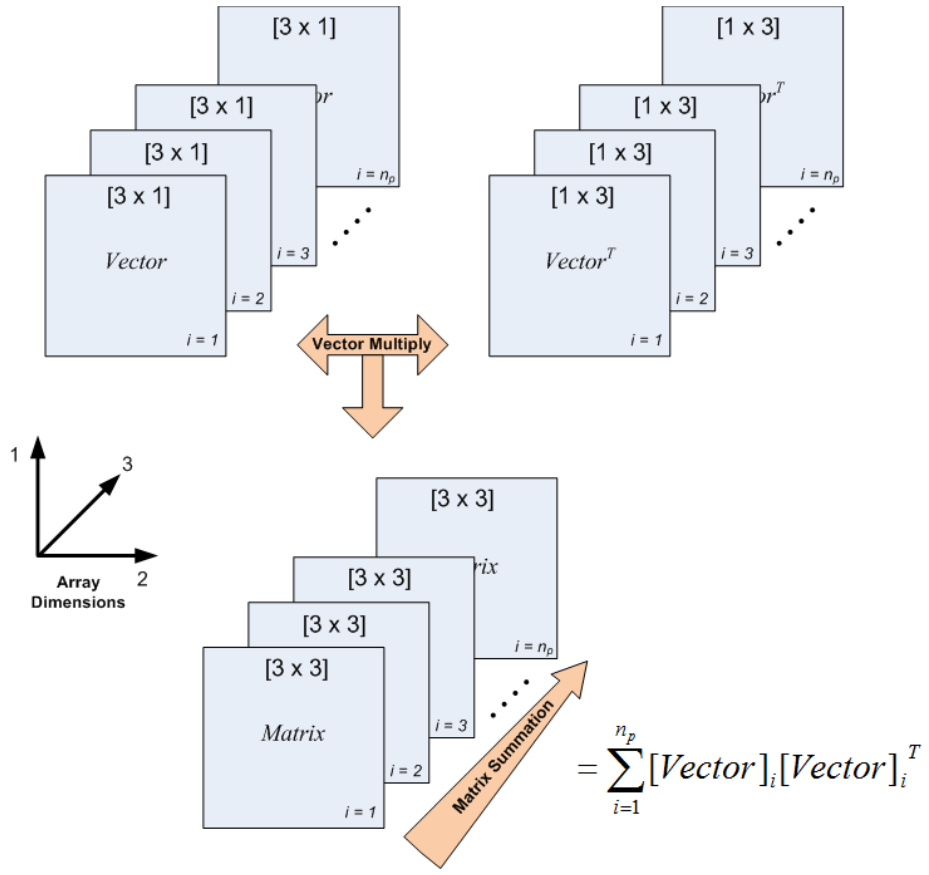

Figure 4.2. MATLAB Programming Approach for Eliminating FOR Loops

surface and back.

4.3.1 Breakdown of LS

Consider the data points in Figure 4.3 as a one dimensional example. We would like to fit a model to these data points. First we choose a model for parameterization $\mathbf{y} = m\mathbf{x}+b$ and then we choose an error function which will ultimately describe how well an instance of the chosen model fits the data. For this example we will choose the vertical distance between the model and a data point as the error function and compute the sum of the squared vertical distances:

$$\begin{aligned} d_i &= (mx_i + b) - y_i \\ E(m,b) &= \sum_{i=1}^{n} d_i^2 = \sum_{i=1}^{n} ((mx_i + b) - y_i)^2 \end{aligned} \tag{4.6}$$

Next we must minimize this summation. Since the argument is linear in each of the model parameters we can take partial derivatives with respect to m and b and set them equal to zero.

$$\begin{aligned} \frac{\partial}{\partial m}\left(E(m,b)\right) &= \sum_{i=1}^{n} \left(x_i^2 m + x_i b - y_i x_i\right) = 0 \\ \frac{\partial}{\partial b}\left(E(m,b)\right) &= \sum_{i=1}^{n} \left(x_i m + b - y_i\right) = 0 \end{aligned} \tag{4.7}$$

Expressing this as a matrix equation, we see that if the matrix on the left is invertible, then we can easily solve the system for parameters m and b. The resulting model for the dataset in Figure 4.3 is shown as trend line (b) in the accompanying chart contained in that Figure. We see that the presence of two outliers has resulted in a model that may not represent the underlying system. Intuitively the trend line (b) is closer to what we would expect.

$$\begin{bmatrix} \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i x_i \\ \sum_{i=1}^{n} y_i \end{bmatrix} \tag{4.8}$$

Because outlier points have a greater contribution than do inlier points in the summation over the squared distances between likely model instances and the data, they can greatly affect the model. The nature of these outliers is common in our application where occasional malfunctions of receiver elements or other optical interactions can create shot noise and other artifacts. It was observed that occasionally an insect flying by in the near field would trigger a receiver element and create a large spike in the data. A method robust to outliers is needed.

X	Y
1	7
2	90
3	16
4	22
5	30
6	34
7	35
8	45
9	51
10	53
11	59
12	61
13	3
14	73
15	80

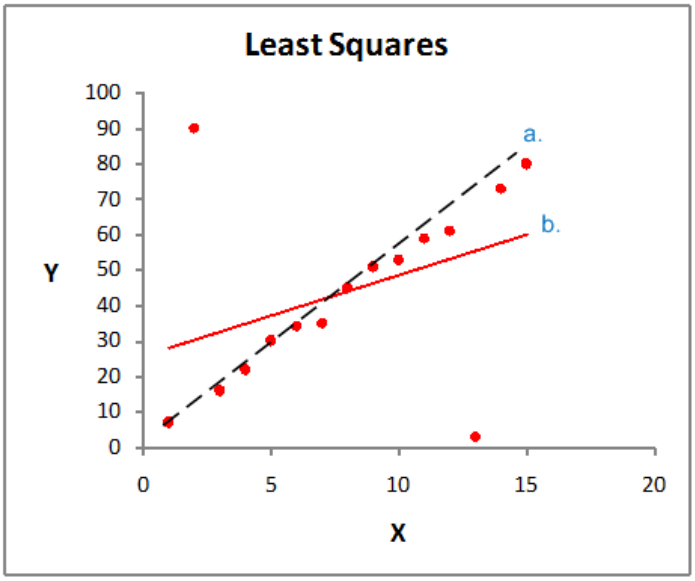

Figure 4.3. The Breakdown of Least Squares - In the case of extreme outliers, the model parameters computed by LS will be greatly skewed. Line (a.) reflects what we expect is the actual underlying trend, and line (b.) is the result of LS minimization.

4.3.2 Random Sample Consensus (RANSAC)

A possible solution to this problem is to use a robust estimator instead of, or in conjunction with Least Squares. One such robust estimation technique popular in Vision applications is Random Sample Consensus (RANSAC) which was first published in 1981 by Fischler and Bolles [1]. RANSAC is an iterative voting method by which a random sample of the data points are used to compute a least squares solution to the system of equations. In the case of our simple example, rather than computing the summations in Equation 4.8 over all points, the summations are computed over a randomly selected pair of points. The resulting model based on this random sampling is then computed. Next, the data points which lie within some tolerance of the model are counted. For this application the tolerance was chosen by qualitative iteration, but it could perhaps be based on measurement precision of the instruments involved in the future. If this count (or *vote*) is high,

then the model is considered accurate. If the random selection of points contained outliers and the vote will be low and the model is considered to be inaccurate. In Figure 4.4, this process is displayed for four iterations. For each iteration, the randomly-selected data points used to compute the model are circled. After four iterations, iteration four produces a model with the highest number of data points within the tolerance of the model. Comparing this model to the LS model computed using all of the data points in Figure 4.3, we can see that it is robust against the two major outlier points in the data. In a real application, the designer can select the number of iterations to be arbitrarily large so that a convergence occurs, or based on a probabilistic bound for a desired error tolerance. The number of iterations could also be determined by empirical trials.

The benefit of using RANSAC over other iterative methods is that the next iteration does not depend on the previous. This means that all of the iterations can potentially be done in parallel rather than sequentially. In a hardware implementation, this would result in a great reduction in computational time. RANSAC also can be optimized for the application by varying the number of iterations, the number of randomly selected points used to compute the model, and the tolerance from the model for which actual data points may contribute a vote. Moreover, these algorithm parameters can be adaptively changed using other information which may be available, such as information from other sensors or estimated measurement precisions.

In our case, the user first specifies the number of randomly selected points which the algorithm will use at each iteration to compute candidate rigid body camera transformations. Specifically, this number of points is inputted as a percentage of the total number of pixels in the image. So, instead of computing the summations in Equation 3.12 over all n_p points, the summations are computed over a randomly

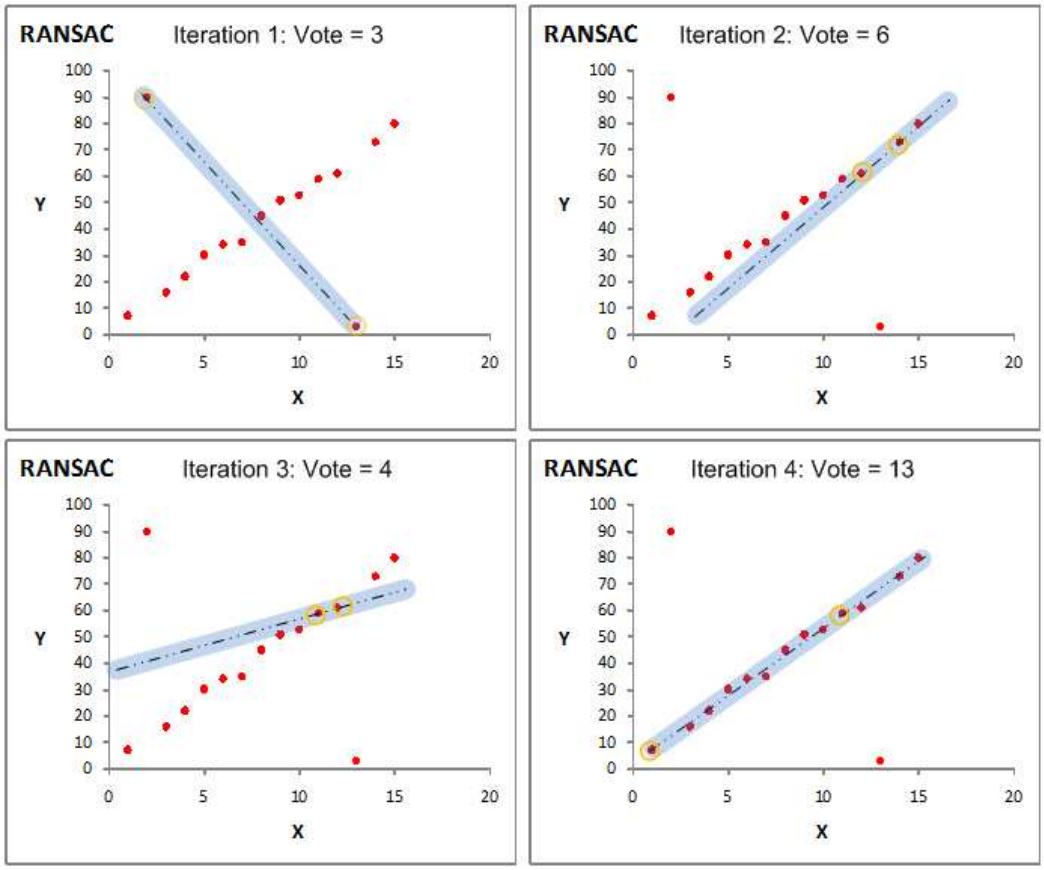

Figure 4.4. Operation of the RANSAC Algorithm - The user specifies the number of iterations for which the algorithm will be run, the number of randomly selected points to use to compute the candidate models, and the tolerance from the model to check for the presence of data points or *votes*. At the end of 4 iterations, iteration 4 produces a model which is more accurate to the underlying process than the LS model shown in Figure 4.3.

selected fraction of points. A new candidate transformation is generated at each iteration, each one computed from a different set of randomly selected points.

Each candidate transformation is then applied to frame *n-1* which generates an estimate of frame *n*. An error metric is then computed between frame *n* and the estimate of *n*. We have already described this metric as using a count of the number of points in frame *n* which lie within a specified tolerance of the estimated frame. One could also use the mean-squared algebraic distance between the observed frame *n* and the estimate, or similar measurements, but it is typical to use the

tolerance voting method. For this work, the tolerance voting method was used.

4.3.3 3D Perspective Projection

The camera transformations or 3D perspective projections which will be estimated define how an observed geometry appears to change under known rotations and translations in all 6 degrees of motion. If a point cloud is generated by a camera from some point in space we can reconstruct what the scene would look like from the new position by applying a mapping defined by the translation distances and rotation angles between the two camera positions. In the context of a RANSAC implementation, a 3D perspective projection transformation needs to be generated from the candidate solution to equation 3.12. Applying this transformation to frame 1 to generate an estimate of frame 2 is necessary in order to compute the vote described in the previous section.

The solution to 3.12 gives the vector

$$\mathbf{x} = \begin{bmatrix} t_x \\ t_y \\ t_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{4.9}$$

and the elements of this vector are used to generate rotation matrices about each of the three axes and a translation matrix. Non-zero 3D translation maps the null vector to a non-zero vector, so it is not linear in 3D space. Since this is the case a 4D homogeneous coordinate system is used to represent 3D perspective projection transformations. The forth coordinate is set to 1 so it doesn't affect the other coordinates when they are ultimately normalized by it to express the result in 3D

space.

The translation matrix $\mathbf{T}$ is given by

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.10}$$

The rotation matrices $\mathbf{\Omega_x}$, $\mathbf{\Omega_y}$, and $\mathbf{\Omega_z}$ are given by

$$\begin{aligned} \mathbf{\Omega_x} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-\omega_x) & \sin(-\omega_x) & 0 \\ 0 & -\sin(-\omega_x) & \cos(-\omega_x) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ \mathbf{\Omega_y} &= \begin{bmatrix} \cos(-\omega_y) & 0 & -\sin(-\omega_y) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(-\omega_y) & 0 & \cos(-\omega_y) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ \mathbf{\Omega_z} &= \begin{bmatrix} \cos(-\omega_z) & \sin(-\omega_z) & 0 & 0 \\ -\sin(-\omega_z) & \cos(-\omega_z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{4.11}$$

The 3D camera transformation taking point i from its original position $p_i = \begin{bmatrix} x_i & y_i & z_i & 1 \end{bmatrix}$ to its new location at position $p_i' = \begin{bmatrix} x_i' & y_i' & z_i' & 1 \end{bmatrix}$ is given by

$$p_i' = [\mathbf{\Omega_x \Omega_y \Omega_z T}]\ p_i \tag{4.12}$$

By applying this transformation to all of the points in frame 1, an estimate of

frame 2 is produced. A voting neighborhood around this estimate is what is used to compute the RANSAC vote. [13]

4.4 Segmenting the Image

While not thoroughly investigated it is worth mentioning that one doesn't necessarily have to use the entire volume, or a sampling over the entire volume of the range image to compute the camera transformation parameters. The range images could be segmented into different regions, and regions with better defined geometry could be used to compute the camera transformation. Regions with ill defined geometry or any other qualities which are known to negatively affect the precision of the estimation could be ignored. As a brief proof of concept, a range image pair was divided into equal tiles. As discussed in Section 4.1.2, the condition number of what would be the **W** and **Z** blocks of matrix **A** were computed in each region. The region which gave the lowest condition number was used to construct the complete **A** matrix and the camera transformation parameters were computed. Preliminary tests showed this method to be potentially promising. Regions with high condition numbers produced less accurate results, but more intelligent image segmentation must be experimented with than simply dividing the image into tiles.

4.5 Constraining the System

Thus far we have been computing the 6-degrees of motion camera transformations between range image pairs. In applications where one would be measuring the motion of air or underwater vehicles this would make sense, as the vehicles actually have the ability to translate in 3 dimensions as well as pitch roll and yaw. When measuring the motion of a land vehicle on the other hand, we can expect very

little movement in the cross-range and roll dimensions. If we constrain the camera transformation estimate to the remaining degrees of freedom of motion we could perhaps get a better result. For the sake of generality, the estimations in this work were not constrained, but could indeed be prudent to do so if such an algorithm is being used in a specific application.

Chapter 5

Testing

5.1 Method

5.1.1 Data Acquisition

Point clouds were acquired using the ASC flash LADAR system. The LADAR camera was powered by a standard 12 volt dry cell motorcycle battery. The camera was mounted with a large fluid pan video camera head affixed to a 2 inch diameter iron pipe. The pipe was mounted to the base of a large utility wagon to allow for mobility on rough terrain. A photograph of the experimental test rig can be seen in Figure 5.1. The point clouds were processed into range images. The camera was fitted with an 85 mm lens and had a 9 degree field of view. Since the field of view on this camera was so small, the nearest objects in the image were selected to be at least 100 feet away in order to get a larger amount of the scene into the image. Since the mean distance down-range is large, small unintentional pitch and yaw perturbations of the camera will look like large elevation and cross range movements, however the 6 degrees of freedom algorithm should be able to handle these small perturbations directly [5]. This would create more of a problem if the motion estimation was constrained to exclude pitch and yaw because the small

Figure 5.1. Experimental Test Rig

angle perturbations of the camera would be wrongly assigned to be movement in elevation and the cross range.

Several range images were taken of simple terrain. The particular terrain was selected due to the smoothness of its geometry and lack of foreground objects with abrupt edges or corners. Since R_t is being approximated by a finite difference, we expect range values at pixel indices which lie on sharp edges and corners to have large variations from frame to frame, even due to small movements of the camera. The terrain's 3D geometry also appeared to be sufficiently complex to ensure that large percentages of the surface normals wouldn't be coplanar. Between each range image that was taken, the camera was translated down-range approximately 1 foot.

Using one of the range images taken of simple terrain, a semi-artificial set of data was generated. This synthetic image pair functions as a control. A predefined camera transformation was applied to one of the range images taken of simple

terrain thus simulating the effect of moving the camera exactly 1 foot down range. The transformed set will be used as the second range image. Using this image pair, the best case result of the algorithm on this particular set of real terrain can be assessed.

One pair of synthetically translated CEMs was taken and shot noise was added to simulate the effect of outlier data points due to dead pixels or sharp edged foreground objects. This set of data was used to try to determine if the error in the results were due to derivative approximation methods used in the algorithm.

5.1.2 Generation of CEM

All of the point clouds were geometrically corrected based on the analysis in Section 2.2, resulting in non-uniformly spaced point clouds. The non-uniform samples were then interpolated and cropped using the method described in section 2.3. The coordinate system for the images of simple terrain was chosen to be identical to Horn and Harris [5], where *positive* x is cross-range, *positive* y is down range and *positive* z is elevation. The coordinate system for the complex scene was chosen differently so the foreground objects would be better represented by the CEM surface. In the case of this image sequence, *positive* x is cross range, *positive* y is elevation, and *negative* z is down range. The *negative* z axis was chosen to be positive down range in order to maintain a right-handed coordinate system.

Four total CEMs of simple terrain were generated from real data. The geometric correction was applied to the point cloud data from the camera. Bicubic interpolation onto a uniform grid was used to create the CEM surfaces from the corrected point cloud data. Each frame was cropped in the center to create uniformly rectangular CEMs. The single pair of synthetically translated range images were produced in the same manner.

5.2 Results

The algorithm developed by Horn and Harris [5] and discussed in Section 3.3 was applied using the four CEMs and the results are shown in Figure 5.2. Selections from the results in Figure 5.2 are shown in Figures 5.3, 5.4, 5.5. The Total Least Squares (TLS) solution was computed over all of the available image pixels. the parameters of the camera transformation are expressed as T0. The column T0c reflects the parameters of the camera transformation when approximated by translations only. Since the test data only reflects a down range translation plus some noise in other degrees of freedom, this was done so that fast comparisons of different techniques could be made simply by inspecting the resulting translational parameters. Again, T0 is the *actual* transformation, and T0c is an *approximation* of T0 by pure translation. The rotations ω_z and ω_x in combination with the mean down-range distance to the CEM, are used to compute arc-lengths around x and z. Since the mean distance to the CEM is large, on the order of 100 feet and the angles in the translation are very small, on the order of thousandths of radians, the arc lengths computed due to ω_z and ω_x were considered to approximate translations in x and z respectively. These translations were added to the existing translations in x and z. The geometry of T0c is easier to comprehend from simply looking at the numbers, but the actual transformation parameters T0 were used to generate the estimated frames.

The camera transformation from frame 1 to frame 1 was computed as a control. As expected, the transformation was all zeroes, since no camera movement had occurred. Between frames directly adjacent to each other in time the down-range translations all were on the order of 1 foot. Looking at T0c, there are small cross-range and elevation changes on the order of a few hundredths of a foot. Extensive testing on several synthetic and real datasets of widely varied terrain types will

be required to determine the sources of error conclusively. Possible sources of error which were investigated include small biases due to finite approximations of derivatives, imprecise inversion of the **A** matrix due to ill-defined surface geometry, or actual unintentional movements of the camera.

It is interesting however, to note that in the case of computing the translation between frame 1 and 3, and between frames 1 and 4, the cross-range and elevation errors increase. This suggests a decrease in accuracy with larger displacements between frames. This is in agreement with Horn and Harris who state in [5] that this is a known problem with this kind of algorithm. Unlike [5] however, the Flash LADAR allows us to acquire range images at a much higher rate, so this problem can be mitigated in a final system by simply avoiding large displacements between frames. These experiments suggest that displacements of under a foot are easily handled by the algorithm. Having a small field of view, as is the case with this Flash LADAR system, can also increase the ill effects due to the aperture problem. As the effective aperture is decreased, less of the scene geometry is captured in each frame and the possibility of having coplanar surface normals increases. In effect this means that shorter focal length lenses with larger fields of view will work better than longer focal length lenses with smaller fields of view.

The synthetically translated range image pair will provide a better assessment of how biases in the algorithm may cause error in the camera transformation parameter prediction, because with this dataset we are sure that the displacement is exactly one foot down-range and zero in all other degrees of freedom. On this data set, the total least squares (TLS) solution was used, as well as a RANSAC solution with varying numbers of iterations. For all of the RANSAC trials, a tolerance of 0.4 feet around the estimated surface was used as the voting neighborhood. Each RANSAC iteration uses a random sample of 1% of the available sample points to

Total Least Squares for Several Camera Transformations

Real Data, Simple Terrain, 1' Translation Down-range Between Successive Frames.

DoF	Frame 1 to 1		Frame 1 to 2		Frame 2 to 3	
	T0	*T0c*	*T0*	*T0c*	*T0*	*T0c*
tx	0	0	-0.8883	0.046	-1.3626	0.0818
ty	0	0	1.306	1.306	1.7225	1.7225
tz	0	0	-0.0479	-0.0445	0.0567	-0.0442
ωx	0		0		0.0006	
ωy	0		0.0005		0.0009	
ωz	0		0.0055		0.0087	

DoF	Frame 3 to 4		Frame 1 to 3		Frame 1 to 4	
	T0	*T0c*	*T0*	*T0c*	*T0*	*T0c*
tx	0.1452	-0.0144	-2.0934	0.1305	-2.3551	0.0901
ty	0.7376	0.7376	2.8196	2.8196	3.0752	3.0752
tz	0.154	-0.0104	0.0169	-0.0742	0.0583	-0.0652
ωx	0.001		0.0005		0.0007	
ωy	-0.0003		0.0017		0.0014	
ωz	-0.001		0.0133		0.0147	

Figure 5.2. Horn and Harris Algorithm -Implemented on real range images of simple terrain. Frames numbered in sequence have 1 foot displacements down-range between them. As you can see, the accuracy decreases as the displacements become larger.

compute the transformation parameters. Since theoretically only six sample points are needed to solve for the 6 transformation parameters, 1% of the 16384 available points is more than enough to use for each iteration. Additive shot noise was added to magnify problems that could be due to derivative approximations. Figure 5.6 shows the results. A weak trend is displayed, showing that the RANSAC implementation of the algorithm converges to an accurate solution as the number of iterations increases. The TLS solution in the presence of outliers exhibits more error than before. This shows the robustness of the RANSAC implementation to outliers. Additionally, the RANSAC solution does not converge to a down-range

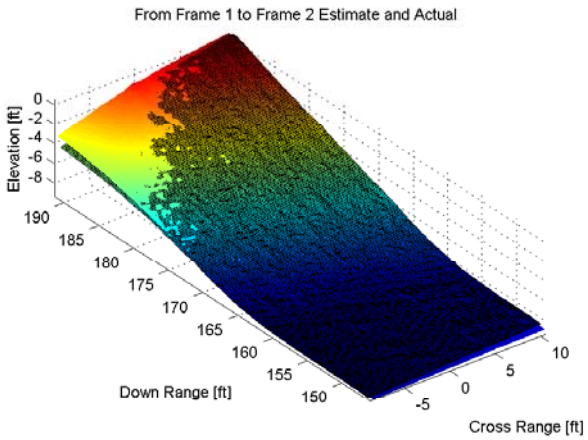

Figure 5.3. Estimating Frame 2 from Frame 1- The black hatched surface is the estimate of Frame 2 computed using the transformation parameters and Frame 1. The pseudo-colored surface is the actual Frame 2

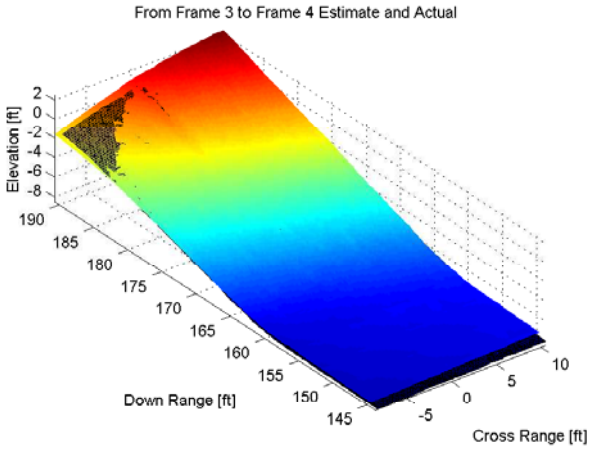

Figure 5.4. Estimating Frame 4 from Frame 3- The black hatched surface is the estimate of Frame 4 computed using the transformation parameters and Frame 3. The pseudo-colored surface is the actual Frame 4. As you can see, the algorithm is accurate for small displacements.

displacement of exactly one foot, suggesting that biases in the algorithm could be contributing to the non-zero cross-range and elevation changes shown in the example of real data reported in Figure 5.2. The fact that the small cross-range and elevation errors still exit in these results suggests that the algorithm itself causes some bias in the solution.

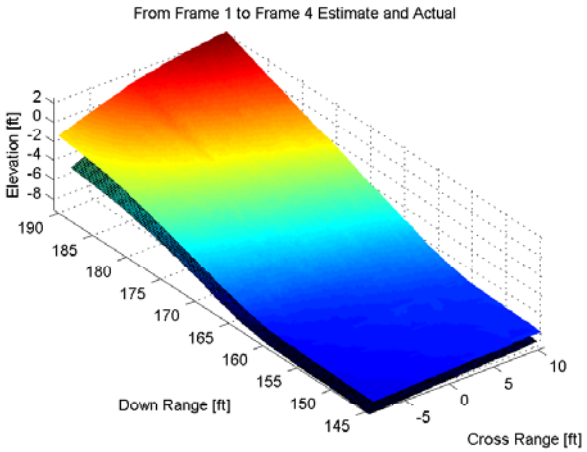

Figure 5.5. Estimating Frame 4 from Frame 1- The black hatched surface is the estimate of Frame 4 computed using the transformation parameters and Frame 1. The pseudo-colored surface is the actual Frame 4. In this instance the down-range displacement is 3 feet, which is estimated accurately, but there are large errors in the elevation dimension.

To access the effect of the voting neighborhood on the camera transformation parameter estimation, the number of iterations of the RANSAC algorithm was held fixed and the tolerance which defines the voting neighborhood was varied. The selection of tolerances were based somewhat arbitrarily on what would be practically useful for most autonomous navigation needs. The algorithm was performed on the synthetically translated image pair without any additional noise corruption. Figure 5.7 shows the results. An example of the results shown in Figure 5.7 are shown in Figure 5.8. The resulting transformation parameters vary minimally, suggesting that the tolerances within which to count votes and the number of iterations used were more than sufficient.

Figure 5.9 shows the effect of using different percentages of randomly selected points to compute the candidate transformations in the RANSAC implementation. These results suggest that for simple terrain and simple motion, the percentage of points used does not affect the solution greatly. This result is useful in terms of reducing computational time for the algorithm. The fewer the number of points

RANSAC with 0.4 ft Tolerance and Varying Iterations

Synthetically Translated Data, Additive Shot Noise 0.72% , 1% Random Sample Size

DoF	TLS		RANSAC 100		RANSAC 500	
	T0	*T0c*	*T0*	*T0c*	*T0*	*T0c*
tx	-1.8755	0.0775	-1.2615	0.0609	-1.8475	-0.0653
ty	1.1039	1.1039	1.0413	1.0413	0.7293	0.7293
tz	0.1164	-0.0458	0.1126	-0.0344	-0.0227	-0.058
ωx	0.001		0.0009		0.0002	
ωy	0.0005		0.0003		-0.0005	
ωz	0.0116		0.0078		0.0105	

DoF	RANSAC 1000		RANSAC 5000		RANSAC 10000	
	T0	*T0c*	*T0*	*T0c*	*T0*	*T0c*
tx	-1.4694	-0.023	-0.8034	0.0473	-0.8431	0.0425
ty	0.8898	0.8898	1.0732	1.0732	1.058	1.058
tz	0.0851	-0.0471	0.1327	-0.0334	0.1221	-0.0338
ωx	0.0008		0.001		0.0009	
ωy	0.0001		0.0005		0.0004	
ωz	0.0086		0.005		0.0052	

Figure 5.6. Results on Synthetically Translated Data with added Noise - A CEM ways synthetically translated one foot down-range. Additive shot noise was added to the CEM to simulate the effects of dead pixels and sharp-edged foreground objects. The number of RANSAC iterations was varied, but all other parameters were held constant. The existence of small error in the result indicates that the numerical approximates made in the algorithm can add a bias.

needed to accurately compute the transformation parameters, the faster the **A** matrix can be constructed. Figure 5.10 shows the same test tried with the noise corrupted, synthetically translated data. In this case the percentage of points used to compute the transformation parameters looks directly proportional to the accuracy of the result. In a more refined implementation, the number of points used to compute the result could be based on a required degree of accuracy, more sophisticated characterizations of noise in the system. With this functionality, additional computation time would only be incurred if the incoming data was

RANSAC, 1000 Iterations, Varying Tolerance

Synthetically Translated Simple Terrain

DoF	TLS		RANSAC 0.4 tol		RANSAC 0.2 tol		RANSAC 0.1 tol	
	T0	*T0c*	*T0*	*T0c*	*T0*	*T0c*	*T0*	*T0c*
tx	-0.971	0.047	-1.239	0.072	-0.6821	0.0168	-0.4422	0.027
ty	1.052	1.052	1.044	1.044	1.0494	1.0494	1.0549	1.055
tz	0.121	-0.034	0.112	-0.034	0.1259	-0.0348	0.1326	-0.034
ωx	9E-04		9E-04		0.001		0.001	
ωy	4E-04		3E-04		0.0005		0.0006	
ωz	0.006		0.008		0.0041		0.0028	

Figure 5.7. Results on Synthetically Translated Data - In this instance no noise was added to the CEM. The number of RANSAC iterations was held constant while the tolerance defining the voting neighborhood was varied.

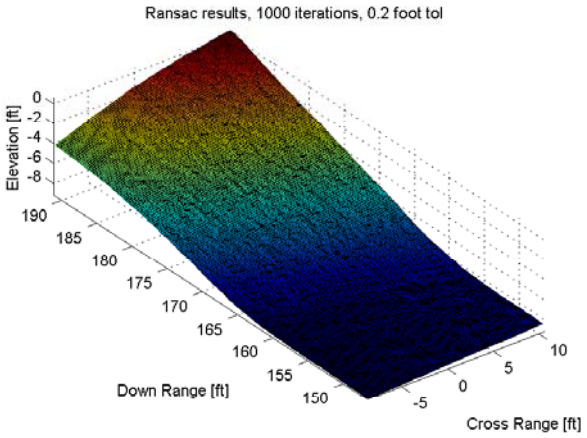

Figure 5.8. RANSAC Implementation- The black hatched surface is the estimate of Frame 2 computed using the transformation parameters and Frame 1. The pseudo-colored surface is the actual Frame 2. The two surfaces occupy the same space to within a few hundredths of a foot. This accuracy exceeds the measurement capabilities of the LADAR device.

thought to be noisy and the accuracy of the parameter estimation needed to be increased.

RANSAC, 500 Iterations, Varying Random Sample Size

Synthetically Translated Simple Terrain

DoF	TLS		RANSAC 1%		RANSAC 25%		RANSAC 75%	
	T0	*T0c*	*T0*	*T0c*	*T0*	*T0c*	*T0*	*T0c*
tx	-0.971	0.047	-1.32	0.059	-0.973	0.051	-0.968	0.049
ty	1.052	1.052	1.052	1.052	1.053	1.053	1.053	1.053
tz	0.121	-0.034	0.113	-0.035	0.121	-0.034	0.122	-0.034
ωx	9E-04		9E-04		9E-04		9E-04	
ωy	4E-04		3E-04		4E-04		4E-04	
ωz	0.006		0.008		0.006		0.006	

Figure 5.9. Effect of Varying the Number of RANSAC Random Sample Size

RANSAC, 500 Iterations, Varying Random Sample Size

Synthetically Translated Simple Terrain, Additive Shot Noise 0.72%

DoF	TLS		RANSAC 1%		RANSAC 25%		RANSAC 75%	
	T0	*T0c*	*T0*	*T0c*	*T0*	*T0c*	*T0*	*T0c*
tx	-1.876	0.078	-1.848	-0.065	-1.805	0.014	-1.6	-0.035
ty	1.104	1.104	0.729	0.729	1.11	1.11	1.088	1.088
tz	0.116	-0.046	-0.023	-0.058	0.124	-0.047	0.103	-0.048
ωx	0.001		2E-04		0.001		9E-04	
ωy	5E-04		-5E-04		9E-04		5E-04	
ωz	0.012		0.011		0.011		0.009	

Figure 5.10. Effect of Varying the RANSAC Voting Random Sample Size on Noisy Data

Chapter 6

Conclusion

6.1 Discussion of Results

The algorithm estimated the camera transformation parameters with a degree of accuracy which would be of use for autonomous navigation applications. Error biases due to the algorithm are on the order of hundreths of a foot for the scenes used. These error biases are well below the ± 3 inch measurement precision of the LADAR. More sophisticated methods of approximating derivatives could be used to reduce this error even farther if necessary. One approach would be to queue several frames of range imagery and use a finite central difference approximation, or even fit a low order polynomial to the data and take the derivative analytically. Other numerical methods for inverting the $\mathbf{A}$ matrix could also be explored.

The use of flash LADAR technology for estimating vehicle motion is proven to a degree that warrants farther research. Reformulating Horn and Harris' algorithm in an iterative RANSAC construct adds robustness to outliers in the data and lends itself to parallel computation, which can lead to faster calculations. The camera used in this work was designed for imaging within distances between 10 feet and 2000 feet. While it was much more convenient to work with than other available

technologies, was not optimal for this specific application. A flash LADAR camera with a greater field of view at distances between one and 75 feet would lend itself better to this application. Such cameras are currently becoming available and should be investigated.

6.2 Future Work

Future work on this project will entail working directly with the camera's hardware on a register level to produce streaming outputs of CEM which are ready for use with this algorithm. With access to a stream of CEM, ensembles of camera transformation parameters can be applied to 6 DoF motion estimators based on Kalman filtering can be used to increase accuracy of the predicted motion of the vehicle [9]. Estimators capable of tracking 6 DoF like those Welch uses in his SCAAT tracking work would be appropriate for this application [15]. Welch has demonstrated robust 6 DoF tracking using a twelve state filter which includes linear positions and velocities as well as angular displacements and angular velocities. Since absolute angle is can not be represented as a linear combination of the state variables, absolute angle is kept track of externally using quaternions.

Further benchmarking of the basic algorithm presented here should be conducted. As Horn and Harris state, this class of algorithm is too complicated for analytical sensitivity analysis [5]. It is thus necessary for more extensive testing on more datasets. Data which is synthetically generated could be used as a general baseline to test for biases in the algorithm, but testing on real data is necessary to ensure stability in the presence of non-ideal phenomenon [9]. The DIRSIG platform could be used to generate these kinds of datasets without needing to have access to actual flash LADAR hardware.

The scenes used in this work were largely of very simple terrain with minimal

existence of foreground objects, and absent of any moving objects. In the future it would be desirable to make accurate motion estimations from cluttered scenes which contain other moving objects. More sophisticated scene segmentation techniques can be used on the range images to avoid what for this implementation are problem areas. Trying to do this could present some interesting problems, such as the separation of object from background, when the camera's motion is unknown. Quantitatively determining the motion of these objects with respect to the camera is a hard problem with potentially meaningful applications such as determining a vehicles time to impact or time to intercept with another object.

Overall, this work will hopefully provide a small step toward the solution to many larger and more complicated problems in motion estimation for autonomous navigation.

Bibliography

[1] M.A. Fischler and R.C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, June 1981.

[2] H. Gharavi and S. Gao. 3-d motion estimation using range data. *IEEE Transactions on Intelligent Transportation Systems*, 8(1):113–143, March 2007.

[3] H. Gong, C. Pan, Q. Yang, H. Lu, and S. Ma. Generalized optical flow in the scale space. *Computer Vision and Image Understanding*, 105:86–92, 2007.

[4] B.K.P. Horn. *Robot Vision*. The MIT Press, 1986.

[5] B.K.P. Horn and J.G. Harris. Rigid body motion from range image sequences. *CVGIP: Image Understanding*, 53(1):1–13, January 1991.

[6] B.K.P. Horn and B.G. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–104, 1981.

[7] H. Madjidi and S. Negahdaripour. On robustness and localization accuracy of optical flow computation for underwater color images. *Computer Vision and Image Understanding*, 104:61–76, 2006.

[8] M. Moganti, F. Ercal, C. Dagli, and S. Tsukenawa. Automatic pcb inspection algorithms: A survey. *CVGIP: Image Understanding*, 63(2):287–313, March 1996.

[9] J. Oliensis. A critique of structure-from-motion algorithms. *Computer Vision and Image Understanding*, 80:172–214, June 2000.

[10] H. Spies. *Analyzing Dynamic Processes in Range Data Sequences*. PhD thesis, Rupertus Carola University of Heidelberg, Germany, 2001.

[11] R. Stettner, H. Bailey, and S. Silverman. Three dimensional flash ladar focal planes and time dependent imaging. Technical report, Advanced Scientific Concepts Inc., 2002. http://www.advancedscientificconcepts.com.

[12] C. Teng, S. Lai, Y. Chen, and W.Hsu. Accurate optical flow computation under non-uniform brightness variations. *Computer Vision and Image Understanding*, 97:315–346, 2005.

[13] E. Trucco and A. Verri. *Introductory Techniques for 3-D Computer Vision.* Prentice Hall, 1998.

[14] Z. Wei, F. Zhou, and G. Zhang. 3d coordinates measurement based on structured light sensor. *Sensors and Actuators A*, pages 527–535, January 2005.

[15] G.F. Welch. *SCAAT: Incremental Tracking with Incomplete Information.* PhD thesis, UNC-Chapel Hill, 1996.

Appendix A

MATLAB Code

A.1 Function: RangeCorrect.m

This function is used to process the raw point cloud data from the ASC Flash LADAR into geometrically correct point clouds which will be later turned into CEM.

```
function [N] = RangeCorrect(D)
%D is the data in tabular format from the
%ASC flash LADAR with columns i,
%j,Range, Intensity.  N is the data in tabular
%format with columns X, Y, Z,
%Intensity.  Distance measurements are in feet.
m2ft = 1/3.2808399; %[ft/m]

fl = 85e-3*m2ft;  %focal length in feet
                  %ex: 85mm / 25.4mm.in / 12in/ft
pitch = 100e-6*m2ft; %pixel pitch

    ri = D(:,1,:);  %row indice
    ci = D(:,2,:);  %column indice
    range = D(:,3,:); %reported range (hypotenuse)
    I = D(:,4,:); %intensity image

    %All variables are consistant with Optics_Diagram_labled.jpg
```

```
y = (ri-64)*pitch;
x = (ci-64)*pitch;
pl = (y.^2+x.^2).^.5;
theta = atan2(fl,pl);
phi = atan2(x,y);
Z = range.*sin(theta);
Il = range.*cos(theta);
X = Il.*cos(phi);
Y = Il.*sin(phi);

N = [X Y Z I];
```

A.2 Generation of CEM

The following MATLAB script takes the geometrically corrected point clouds and turns them into CEM for use with the 3D camera transformation estimation algorithms explored in this work.

```
%Donald J. Natale III
%Create CEM for Horn and Harris type algorithms
%November 2008



%This script converts corrected point cloud data into
%cartesian elevation maps
%(CEM) for use in range flow algorithms.
%
%Input p is a point cloud array structured as follows:
%               [X    Y      Z   Intensity]
%               [.    .      .       .    ]
%               [.    .      .       .    ]
%               [.    .      .       .    ]
%The output is a data structure containing three 2D
%arrays, X, Y, and Z
%corresponding to each pixel location in the CEM.
```

```
%X is cross range,
%Y is down range, and Z is elevation.
clear all
close all
clc

load D % Load the range corrected point clouds

[r c d] = size(D)

for i = 1:d
    p = D(:,:,i);

roi = find(p(:,3)>150);
p = p(roi,:);
```

%Dimension Assignments

```
%By changing the columns here you can choose which
%dimension is X Y and Z
% in the final CEM

xn = min(p(:,1));
xp = max(p(:,1));

yn = min(p(:,2));
yp = max(p(:,2));


x = p(:,1);
y = p(:,2);
z = p(:,3);

delx = dist(xn,xp)/255;
dely = dist(yn,yp)/255;
```

```
d1 = [xn:delx:xp];
d2 = [yn:dely:yp];

b = [xn xp delx; yn xp dely];

[XI YI] = meshgrid(d1,d2);
%YI = flipud(YI);


q_imp = flipud(flipud((griddata(x,y,-z,XI,YI,'cubic'))));
q_im = imcrop(q_imp,[10 10 176 176 ]);
XI_q = imcrop(XI,[10 10 176 176 ]);
YI_q = imcrop(YI,[10 10 176 176 ]);


figure()
surf(XI_q, YI_q, q_im)
xlabel 'Cross-range [ft]'
ylabel 'Elevation [ft]'
zlabel '-Down Range [ft]'
shading interp
axis equal

figure()
imagesc(q_im)
axis xy

%save the data to a datastructure
 frame(i).ZI = q_im;
 frame(i).XI = XI_q;
 frame(i).YI = YI_q;
end
```

A.3 Implementation of Horn and Harris 1991 Algorithm

The following is an implementation in MATLAB of the algorithm described in the appendix of Horn and Harris' 1991 paper [5].

```
% Donald J. Natale III
%Implementation of Horn and Harris 1991 Algorithm for estimating
%camera transformations between two range image frames.
%November 2008


clear all
close all
clc
```

%SET CEMs to use

```
load synHill.mat %Read in the preformatted CEM data structures
%choose frames within the structure to use.
i1 = 1;
i2 = 2;
```

%SET Smoothing Parameters

```
skip_smooth = 0;
sigma = .01;
win = 3;%
```

%Gaussian low pass filter the images

```
if skip_smooth ~= 1
tic
 gau = fspecial('gaussian',[win win],sigma);
 frame(i1).ZI = imfilter(frame(i1).ZI, gau,'replicate','same');
 frame(i1).XI = imfilter(frame(i1).XI, gau,'replicate','same');
 frame(i1).YI = imfilter(frame(i1).YI, gau,'replicate','same');

 frame(i2).ZI = imfilter(frame(i2).ZI, gau,'replicate','same');
```

```
 frame(i2).XI = imfilter(frame(i2).XI, gau,'replicate','same');
 frame(i2).YI = imfilter(frame(i2).YI, gau,'replicate','same');
 toc = toc
end

[rows cols] = size(frame(i2).ZI);
npix = rows*cols;

%Previous Frame
  Xp = reshape(frame(i1).XI,npix,1);
  Yp = reshape(frame(i1).YI,npix,1);
  Zp = reshape(frame(i1).ZI,npix,1);
  Rp = [Xp'; Yp'; Zp']; %vectors from the camera points on
                        %the surface

%Current frame
  X = reshape(frame(i2).XI,npix,1);
  Y = reshape(frame(i2).YI,npix,1);
  Z = reshape(frame(i2).ZI,npix,1);
  R = [X'; Y'; Z']; %vectors from the camera to each point
                    %on the surface


  %compute unit normals at each point on the surface
  [Nx Ny Nz] = surfnorm(frame(i2).ZI);
  nx = reshape(Nx,1,npix);
  ny = reshape(Ny,1,npix);
  nz = reshape(Nz,1,npix);
  n = [nx; ny; nz];  %unit normals at each point on
                     %the surface


  %compute unit vector in the direction of R to each point
  Rmag = sum(R.^2,1).^0.5;
```

```
  r = R./[Rmag;Rmag;Rmag]; %unit vectors in the direction of R

  %compute observed range rate
  Rtd = R - Rp;
  Rt = sum(Rtd.^2,1).^0.5 ;



  c = cross(R,n);
  d = dot(r,n);
```

%Compute the A matrix and the B vector,

then invert A to solve for the transformation.

```
si = [1:npix];

%Implementation without using FOR loops:
  nr  = reshape(n(:,si), 3,1,length(si));
  cr  = reshape(c(:,si), 3,1,length(si));
  Rtr = reshape(Rt(:,si),1,1,length(si));
  rr  = reshape(r(:,si), 3,1,length(si));

  A = [nr(:,:)*nr(:,:)'      nr(:,:)*cr(:,:)';...
       cr(:,:)*nr(:,:)'       cr(:,:)*cr(:,:)'];

  br1 = Rtr(:,:).*dot(rr(:,:),nr(:,:),1);
  br2 = [br1;br1;br1];
  br3 = sum(br2.*nr(:,:),2);
  br4 = Rtr(:,:).*dot(rr(:,:),nr(:,:),1);
  br5 = [br4;br4;br4];
  br6 = sum(br5.*cr(:,:),2);

  b = -[br3;br6];
```

```
  Ainv = inv(A);
  T_0 = Ainv*b;  %The solution to the transformation!
  T_0 =T_0.*[-1 -1 -1 1 1 1]'; % Correct from moving
                                %world to moving camera
```

%Computation of the 3D camera transformation, and application to frame 1

```
  %extract the parameters needed to build a 3D
  %Perspective projection that
  %will transform frame1 into an estimate of frame2.
  tx = T_0(1);
  ty = T_0(2);
  tz = T_0(3);
  ax = T_0(4);
  ay = T_0(5);
  az = T_0(6);

  %Define the rotation matricies
  Rotx = [ 1           0           0;...
           0           cos(-ax)    sin(-ax);...
           0          -sin(-ax)    cos(-ax)];

  Roty = [ cos(-ay)    0          -sin(-ay);...
           0           1           0;...
           sin(-ay)    0           cos(-ay)];

  Rotz = [ cos(-az)    sin(-az)    0;...
          -sin(-az)    cos(-az)    0;...
           0           0           1];

Rotation = Rotx*Roty*Rotz;

estimate = Rotation*([Xp'- tx; Yp'- ty; Zp'- tz]);
```

```
X_e = estimate(1,:)';
Y_e = estimate(2,:)';
Z_e = estimate(3,:)';

XeI = reshape(X_e,rows,cols);
YeI = reshape(Y_e,rows,cols);
ZeI = reshape(Z_e,rows,cols);

T_0
```

%Create the 3D Camera Projection

```
  %extract the parameters needed to build a 3D
  %Perspective projection that
  %will transform frame1 into an estimate of frame2.
  tx = T_0(1);
  ty = T_0(2);
  tz = T_0(3);
  ax = T_0(4);
  ay = T_0(5);
  az = T_0(6);



  %Define the rotation matricies
  Rotx = [ 1           0           0;...
           0           cos(-ax)    sin(-ax);...
           0           -sin(-ax)    cos(-ax)];

  Roty = [ cos(-ay)    0          -sin(-ay);...
           0           1           0;...
           sin(-ay)    0           cos(-ay)];
```

```
  Rotz = [ cos(-az)    sin(-az)    0;...
          -sin(-az)    cos(-az)    0;...
           0           0           1];


Rotation = Rotx*Roty*Rotz;


estimate = Rotation*([Xp' - tx; Yp'- ty; Zp'- tz]);


X_e = estimate(1,:)';
Y_e = estimate(2,:)';
Z_e = estimate(3,:)';

XeI = reshape(X_e,rows,cols);
YeI = reshape(Y_e,rows,cols);
ZeI = reshape(Z_e,rows,cols);


%Approximate transformation as 3D translational
%for ease of comparison
T_0c =...
 [tx + az*mean(mean(YeI)), ty, tz - ax*mean(mean(YeI)) 0 ay 0]'
```

%Display Settings

```
figure(1)
surf(frame(i2).XI,flipud(frame(i2).YI),frame(i2).ZI)
shading interp
hold on

surf(XeI,flipud(YeI),ZeI)
title 'Frame 2 and Frame 2 Estimate'
zlabel 'Elevation [ft]'
```

```
xlabel 'Cross Range [ft]'
ylabel 'Down Range [ft]'
axis equal

figure(2)
surf((frame(i2).XI-XeI),(flipud(frame(i2).YI-YeI)),...
 (frame(i2).ZI-ZeI))
shading interp
title 'Errors'
zlabel 'Elevation Error [ft]'
xlabel 'Cross Range Error [ft]'
ylabel 'Down Range Error [ft]'
```

A.4 RANSAC Implementation

The following is a RANSAC implementation in MATLAB of the Horn and Harris algorithm.

Contents

```
% Donald J. Natale III
%RANSAC Implementation of Horn and Harris 1991 Algorithm for
%estimating camera transformations between two range image frames.
%November 2008

clear all
close all
clc
```

%SET CEMs to use

```
load synHill.mat %Read in the preformatted CEM data structures
%choose frames within the structure to use.
i1 = 1;
i2 = 2;
```

%SET Smoothing Parameters

```
skip_smooth = 0;
sigma = .01;
win = 3;%
```

%SET RANSAC Parameters

```
tolx = .4;
toly = .4;
tolz = .4;

%use fraction = 0, use points = 1;
frac_or_points = 0;
%fraction of points to use
fraction = .25;
%number of points to use
points_to_use = 6;

%Specify number of
Ransac_Iterations = 500;
```

%Compute RANSAC Points

```
[rows cols] = size(frame(i1).ZI);
npix = rows*cols;
 %number of randomly  selected pixels used to solve LS equations
nsum = ceil(npix*fraction);
```

%Gaussian low pass filter the images

```
if skip_smooth ~= 1
tic
 gau = fspecial('gaussian',[win win],sigma);
 frame(i1).ZI = imfilter(frame(i1).ZI, gau,'replicate','same');
 frame(i1).XI = imfilter(frame(i1).XI, gau,'replicate','same');
 frame(i1).YI = imfilter(frame(i1).YI, gau,'replicate','same');
```

```
 frame(i2).ZI = imfilter(frame(i2).ZI, gau,'replicate','same');
 frame(i2).XI = imfilter(frame(i2).XI, gau,'replicate','same');
 frame(i2).YI = imfilter(frame(i2).YI, gau,'replicate','same');
 toc = toc
end

%Previous Frame
  Xp = reshape(frame(i1).XI,npix,1);
  Yp = reshape(frame(i1).YI,npix,1);
  Zp = reshape(frame(i1).ZI,npix,1);
%vectors from the camera points on the surface
  Rp = [Xp'; Yp'; Zp'];

%Current frame
  X = reshape(frame(i2).XI,npix,1);
  Y = reshape(frame(i2).YI,npix,1);
  Z = reshape(frame(i2).ZI,npix,1);
  %vectors from the camera to each point on the surface
  R = [X'; Y'; Z'];


  %compute unit normals at each point on the surface
  [Nx Ny Nz] = surfnorm(frame(i2).ZI);
  nx = reshape(Nx,1,npix);
  ny = reshape(Ny,1,npix);
  nz = reshape(Nz,1,npix);
  n = [nx; ny; nz];  %unit normals at each point on the surface


  %compute unit vector in the direction of R to each point
  Rmag = sum(R.^2,1).^0.5;
  r = R./[Rmag;Rmag;Rmag]; %unit vectors in the direction of R
```

```
  %compute observed range rate
  Rtd = R - Rp;
  Rt = sum(Rtd.^2,1).^0.5 ;



  c = cross(R,n);
  d = dot(r,n);
```

%Begin RANSACing

```
    vote_previous = 0;
    err_previous = inf;
for it = 1:Ransac_Iterations
    clc
    it
```

%RANSAC Random Point Selection

```
  %Choose random sampling of points to
  % compute the rigid transformation from.
  %'fraction' is defined at the beginning of the file.

  if frac_or_points ~= 1
  si = randsample(npix,ceil(npix*fraction));
  else
  si = randsample(npix,points_to_use);
  end
```

%Compute the A matrix and the B vector,

then invert A to solve for the transformation.

```
%Implementation without using FOR loops:
  nr  = reshape(n(:,si), 3,1,length(si));
  cr  = reshape(c(:,si), 3,1,length(si));
```

```
  Rtr = reshape(Rt(:,si),1,1,length(si));
  rr  = reshape(r(:,si), 3,1,length(si));

  A = [nr(:,:)*nr(:,:)'      nr(:,:)*cr(:,:)';...
       cr(:,:)*nr(:,:)'      cr(:,:)*cr(:,:)'];

  br1 = Rtr(:,:).*dot(rr(:,:),nr(:,:),1);
  br2 = [br1;br1;br1];
  br3 = sum(br2.*nr(:,:),2);
  br4 = Rtr(:,:).*dot(rr(:,:),nr(:,:),1);
  br5 = [br4;br4;br4];
  br6 = sum(br5.*cr(:,:),2);

  b = -[br3;br6];


  Ainv = inv(A);
  T_0 = Ainv*b;  %The solution to the transformation!
% Correct from moving world to moving camera
  T_0 =T_0.*[-1 -1 -1 1 1 1]';
```

%Computation of the 3D camera transformation, and application to frame 1

```
  %extract the parameters needed to build a 3D Perspective
  %projection that will transform frame1 into an estimate
  %of frame2.
  tx = T_0(1);
  ty = T_0(2);
  tz = T_0(3);
  ax = T_0(4);
  ay = T_0(5);
  az = T_0(6);

  %Define the rotation matricies
```

```
  Rotx = [ 1           0           0;...
           0           cos(-ax)    sin(-ax);...
           0          -sin(-ax)    cos(-ax)];

  Roty = [ cos(-ay)    0          -sin(-ay);...
           0           1           0;...
           sin(-ay)    0           cos(-ay)];

  Rotz = [ cos(-az)    sin(-az)    0;...
          -sin(-az)    cos(-az)    0;...
           0           0           1];

Rotation = Rotx*Roty*Rotz;

estimate = Rotation*([Xp'- tx; Yp'- ty; Zp'- tz]);


X_e = estimate(1,:)';
Y_e = estimate(2,:)';
Z_e = estimate(3,:)';

XeI = reshape(X_e,rows,cols);
YeI = reshape(Y_e,rows,cols);
ZeI = reshape(Z_e,rows,cols);
```

%Compute RANSAC Score:

```
%'tol' is the Tolerance from the estimate to look for points.
%'tol' is defined at the beginning of this M-file.

overlap =...
find(  frame(i2).XI >= XeI - tolx & frame(i2).XI <= XeI + tolx...
     & frame(i2).YI >= YeI - toly & frame(i2).YI <= YeI + toly...
     & frame(i2).ZI >= ZeI - tolz & frame(i2).ZI <= ZeI + tolz);
```

```
vote = length(overlap);

    if vote >= vote_previous
        vote_previous = vote;
        T_0_Best = T_0;
    end
```

%END RANSACING

```
end
```

%Show Best Match

```
if vote ~= 0
T_0 = T_0_Best
vote
else
T_0
end
```

%Create the 3D Camera Projection

```
  %extract the parameters needed to build a 3D Perspective
  %projection that will transform frame1 into
  %an estimate of frame2.
  tx = T_0(1);
  ty = T_0(2);
  tz = T_0(3);
  ax = T_0(4);
  ay = T_0(5);
  az = T_0(6);



  %Define the rotation matricies
  Rotx = [ 1           0           0;...
```

```
          0           cos(-ax)    sin(-ax);...
          0           -sin(-ax)    cos(-ax)];

  Roty = [ cos(-ay)    0          -sin(-ay);...
           0           1           0;...
           sin(-ay)    0           cos(-ay)];

  Rotz = [ cos(-az)    sin(-az)    0;...
          -sin(-az)    cos(-az)    0;...
           0           0           1];


Rotation = Rotx*Roty*Rotz;


estimate = Rotation*([Xp' - tx; Yp'- ty; Zp'- tz]);


X_e = estimate(1,:)';
Y_e = estimate(2,:)';
Z_e = estimate(3,:)';

XeI = reshape(X_e,rows,cols);
YeI = reshape(Y_e,rows,cols);
ZeI = reshape(Z_e,rows,cols);


%Approximate transformation as 3D translational
%for ease of comparison.
T_0c = ...
[tx + az*mean(mean(YeI)), ty, tz - ax*mean(mean(YeI)) 0 ay 0]'
```

%Display Settings

```
figure(1)
surf(frame(i2).XI,flipud(frame(i2).YI),frame(i2).ZI)
shading interp
hold on

surf(XeI,flipud(YeI),ZeI)
title 'Frame 2 and Frame 2 Estimate'
zlabel 'Elevation [ft]'
xlabel 'Cross Range [ft]'
ylabel 'Down Range [ft]'
axis equal

figure(2)
surf((frame(i2).XI-XeI),(flipud(frame(i2).YI-YeI)),...
(frame(i2).ZI-ZeI))
shading interp
title 'Errors'
zlabel 'Elevation Error [ft]'
xlabel 'Cross Range Error [ft]'
ylabel 'Down Range Error [ft]'
```