Sequential Motion Estimation and Refinement for Applications of Real-time Reconstruction from Stereo Vision

Kevin V. Stefanik

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Engineering

Kevin Kochersberger

A. Lynn Abbott

Craig Woolsey

May 31, 2011

Blacksburg, Virginia

Keywords: stereo vision, feature points, bundle adjustment, matching, UAV, terrain mapping, IRLS, 3D reconstruction, SURF

Sequential Motion Estimation and Refinement for Applications of Real-time Reconstruction from Stereo Vision

Kevin V. Stefanik

ABSTRACT

This paper presents a new approach to the feature-matching problem for 3D reconstruction by taking advantage of GPS and IMU data, along with a prior calibrated stereo camera system. It is expected that pose estimates and calibration can be used to increase feature matching speed and accuracy. Given pose estimates of cameras and extracted features from images, the algorithm first enumerates feature matches based on stereo projection constraints in 2D and then backprojects them to 3D. Then, a grid search algorithm over potential camera poses is proposed to match the 3D features and find the largest group of 3D feature matches between pairs of stereo frames. This approach will provide pose accuracy to within the space that each grid region covers. Further refinement of relative camera poses is performed with an iteratively re-weighted least squares (IRLS) method in order to reject outliers in the 3D matches. The algorithm is shown to be capable of running in real-time correctly, where the majority of processing time is taken by feature extraction and description. The method is shown to outperform standard open source software for reconstruction from imagery.

Dedicated to my parents.

Acknowledgements

I would like to thank my committee members, Dr. Abbott, Dr. Woolsey and Dr. Kochersberger, along with the graduate students directly involved in the development for this project including Jason Gassaway, and Nathan Short. I'd also like to thank other graduate students at the Unmanned Systems Lab including Kenny Kroeger, Brian McCabe, Jerry Towler, Praither Lanier, Shajan Thomas, Mike Bromley, Eric Brewer, Eric Gustafson, Collin Lutz, Matt Torok, and Pete Fanto for their flight support, help, and comic releif. For supporting my persuit of a graduate degree, I thank my family, including my parents, sisters and nephews.

Contents

List of Algorithms

List of Figures

List of Tables

1. Introduction

1.1. Background

The Unmanned Systems Lab is developing an aerial nuclear materials detection and sampling system. As part of that system, a stereo camera system has been selected to rapidly provide 3D information about the terrain of the region that may contain nuclear materials. A radiation detector alongside the stereo imaging system is to provide radiation concentration data to be overlaid on top of the 3D terrain, and presented to a ground control station for human analysis. A real-time solution is preferred so that dynamic mission-level human intervention may occur.

Therefore, this thesis addresses the use of a calibrated stereo camera system for real-time terrain mapping. In the future, this technology could be extended to ground vehicle tracking and navigation, forest fire or post-disaster relief, beyond line-of-sight navigation and avoidance, and for VTOL aircraft it is likely to be used for autonomous landing zone assessment. Since a stereo camera system provides synchronized multi-view imagery, it can generate 3D information when the cameras are stationary, and even reconstruct dynamic changes in the environment.

In general, digital electro-optic (EO) imagers can greatly reduce hardware complexity and cost in 3D reconstruction. Reconstruction might be used for mapping and navigation for land, sea, and air vehicles–both manned and unmanned. EO sensors are relatively small and lightweight enough that they could even be installed on a micro air vehicle (MAV). Also, passive sensors inherently require much less power. Furthermore, they provide very dense data, which is dependent on the number of cells in the sensor array. Other 3D mapping sensor types, such as LIDAR (*Natale et al.*, 2010) and synthetic aper-

ture radar (SAR) (*Munoz et al.*, 2009) have their own strengths, but generally do not match the power, weight, size, cost, and data density of EO sensors. LIDAR generates laser pulses to record time-of-flight to and from an object with which the beam intersects. This active technique requires more power at greater distances, but is also highly accurate to just a few centimeters, with little or no outliers in results. SAR is also an active technology that benefits because it can penetrate most cloud cover and dust by using higher electromagnetic frequencies. However, it requires several passes to achieve comparable accuracy to LIDAR. So, for real-time mapping applications, SAR is infeasible and LIDAR is complex and expensive, and so we explore the use of EO imagery.

Many projects have already addressed 3D reconstruction from EO imagery. Specifically, Urbanscape at the University of North Carolina (*Mordohai et al.*, 2007), Bundler at the University of Washington (*Snavely et al.*, 2007), and AEROSYNTH at Rochester Institute of Technology (*Walli et al.*, 2009), each of which is discussed in section 2.8. Currently, the standard method of reconstruction is named bundle adjustment (BA), which is the maximum likelihood global estimator for camera pose, camera intrinsics, and radial lens distortion under zero-mean Gaussian noise (*Lourakis and Argyros*, 2009). Lourakis formulates BA as an optimization problem over variable camera calibration parameters and feature locations in 3D.

A feature in this text refers to any 3D location in the environment, which can be detected in an image as either the local minima or maxima some function over the image. Several image feature detectors have been designed that quickly extract information from an image, so that results can be used for classification and matching. In classification, one may wish to detect a ground vehicle from an air vehicle, for example. For matching, applications include optical flow, navigation, mapping and 3D reconstruction with BA. BA requires feature matches as an input and minimizes the squared norms of the residuals between the projection of the 3D feature locations on the image and the measured location in the image. Common features range from edge detectors (*Canny*, 1986), to corner detectors (*Harris and Stephens*, 1988), to more current scale and affine invariant point approaches (*Lowe*, 2004; *Bay et al.*, 2006; *Mikolajczyk and Schmid*, 2002), further

discussed in section 2.4.

1.2. Problem Statement

When refining camera parameters, robust and accurate feature matches between images must be provided to the BA algorithm. Robust in this context means maintaining performance in the presence of outliers and noise. Common feature detectors and descriptors can only assume that features undergo scaling and translational changes between different viewpoints. However, when mapping highly dynamic environments, such as a tree canopy or grass fields blowing in the wind, feature descriptors may vary greatly over time and viewpoint. Therefore, we consider environments that provide features which are difficult to match, because the feature descriptors are themselves dynamic. We also consider scenery with textures that may generate homogenous features such as gravel, asphalt, or concrete, that may also be difficult to match. These scenarios would result in many outlying feature matches.

In high speed video, features generally undergo small dynamic changes frame to frame. In this case, feature matching is not a problem because the baseline, or distance the camera travels, between frames is small. However, most applications would prefer low frame-rate wide baseline matching because 3D reconstruction results from a wide baseline are much more accurate. Also, high resolution and high frame-rate imagery would require heavy processing for real-time operation. Therefore, a low frame-rate (~2Hz), high resolution imaging (>1megapixel) system is the focus of problem.

The problem is then to develop an efficient and robust feature matching algorithm for homogeneous or dynamic features viewed with a low frame-rate, high resolution imaging system. In order to gather 3D information in the presence of dynamic scene changes, two synchronized and calibrated stereo cameras are assumed.

1.3. Contribution

This thesis presents an algorithm that first matches features between stereo image pairs using stereo calibration constraints. It then backprojects the matches to 3D and performs matching between sets of 3D features. Finally, it refines the relative pose of the cameras with the 3D feature matches.

A single frame in this paper will refer to two synchronized stereo images and 6DOF pose measurements *captured at a unique time instant*. Any derived data from any single frame will be considered part of that frame as well (i.e. a set of 3D backprojected points from the synchronized stereo pair). An inertial measurement unit (IMU) and GPS receiver provide the pose measurements in each frame. The presented algorithm performs alignment between frame pairs assuming field of view overlap between the cameras.

The algorithm begins by finding stereo correspondences between feature points within a single stereo frame and backprojecting them to 3D locations using prior stereo calibration parameters. It then attacks the homogenous and dynamic feature problem by matching two sets of 3D features from two different frames while imposing pose measurement constraints.

This method is expected to benefit for two reasons. Firstly, it will reduce the search space for feature matches by imposing 3D constraints. Secondly, it will be able to find more correct matches in a dynamic environment. The first benefit is rather straightforward since common kD-tree approaches target monocular applications with no pose measurements, so 3D constraints are not applied. The second benefit is more difficult to understand. Consider a car traveling down a road and a stationary camera viewing the car. If two images of the vehicle were captured from the same pose but at different times, standard feature matching techniques may find several features on the vehicle that match, but would likely throw them out when imposing affine constraints on the entire set of feature matches, resulting in correct matching. However, if relatively few distinct features are generated and matched from the rest of the environment, such as textures of homogeneous grass or sand, then the features on the vehicle may be accepted as the overall change. So, standard algorithms could find that the camera has moved, when it

was in fact stationary. However, by applying pose constraints when matching 3D features from stereo, correct feature matches might still be extracted from the rest of the scene.

The matching algorithm uses 3D space with 6DOF pose measurements. To overcome homogeneous features, an optimization problem is formulated to maximize the number of matches over the 6DOF pose variance. The objective function of this problem is not easily differentiable since it takes on discrete values. It is also computationally intensive since it depends on searching for matches in 3D. So, a grid-based pose search is used. Costs are calculated across a 6DOF grid of potential pose regions that cover the possible pose measurement variance. The grid region with the most high-certainty feature matches is considered the optimal value of the set.

Outliers are detected using a custom Iteratively Reweighted Least Squares (IRLS) algorithm, which isolates the group of matches with most coherence. This approach is expected to be more valuable than a RANSAC technique since pose measurements are available, and it is much more deterministic.

This approach will be analyzed in execution time and accuracy. Because reference datasets are not currently available, accuracy will only be able to be measured through manual inspection. Furthermore, it will be shown to produce effective results even in the GPS-denied case. Lastly, its processing requirements will be shown to be feasible for a real-time implementation.

This feature matching approach would be used in a higher level real-time mapping system. Coordinate transformations for geo-loation are emplyed, but correction for random-walk effects from multiple frame-to-frame alignment is not performed. Furthermore, the algorithm generates all 3D information from single pairs of calibrated stereo imagery, and so data reduction is not performed between multiple frames.

2. Literature Review

As this research required much investigation into the fields of image processing and estimation, this literature review will cover several mathematical standards that this research builds upon and attempts to exploit. This includes sections 2.1-2.6. These sections cover image filtering, multiple view geometry, estimation theory, feature extraction, feature matching and optimization methods for 3D reconstruction from imagery. Reviews of full reconstruction systems and projects are found in sections 2.7 and 2.8.

2.1. Image Filtering

2.1.1. Gaussian Smoothing

The Gaussian filter is a common method for image smoothing or down-sampling. This filter is implemented as the 2D convolution of the image, $I(x, y)$, with a Gaussian kernel:

$$g(x, y, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

$$G = g * I$$

For example, a 3×3 kernel could be approximated discretely as:

$$g_{3\times3} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

This can be considered a low-pass filter such that features of size σ or larger are retained. Therefore, it is commonly used as a filter to extract features of a specific size within an image.

2.1.2. Laplacian of Gaussian

The Laplacian of Gaussian (LoG) can be used for edge detection or noise detection. After running a Gaussian filter, the Laplacian is calculated. This can be reduced to a single kernel convolution that looks something like:

$$K_{5\times5} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 1 & 2 & -16 & 2 & 1 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{2.1.1}$$

2.1.3. Difference of Gaussians

The Difference of Gaussians is merely an approximation of the LoG, and is given by eqn. 2.1.2.

$$D_{\sigma,\Delta\sigma}(x,y) = \frac{1}{2\Delta\sigma}\left(g(x,y,\sigma+\Delta\sigma) * I(x,y) - g(x,y,\sigma-\Delta\sigma) * I(x,y)\right) \tag{2.1.2}$$

With $\Delta\sigma$ chosen appropriately, the resulting differences between the DoG and LoG are small enough to be negligible. Since it is merely the difference of two Gaussian-smoothed images, it can be quickly calculated at multiple scales. A resultant kernel would look similar to equation 2.1.1.

2.1.4. Determinant of the Hessian

The determinant of the 2D Hessian (equation 2.1.3) is commonly used to find scale-space maxima and minima. Its determinant is defined by equation 2.1.4. The maxima or

minima of this measure is commonly used for feature point detection (*Mikolajczyk and Schmid*, 2004). After a Gaussian smoothing operation, the determinant of the Hessian is used to find points, and sometimes edges, at that scale. This is found numerically using discretized second order partial derivatives. See *Bay et al.* (2006) for more information.

$$\mathcal{H}(I) = \begin{bmatrix} \frac{\partial^2 I}{\partial x^2} & \frac{\partial^2 I}{\partial x \partial y} \\ \frac{\partial^2 I}{\partial x \partial y} & \frac{\partial^2 I}{\partial y^2} \end{bmatrix} \tag{2.1.3}$$

$$|\mathcal{H}| = \left(\frac{\partial^2 I}{\partial x^2}\right)\left(\frac{\partial^2 I}{\partial y^2}\right) - \left(\frac{\partial^2 I}{\partial x \partial y}\right)^2 \tag{2.1.4}$$

2.2. Multiple View Geometry

Multiple view geometry theory enables 3D reconstruction of an environment from 2D images taken at different camera poses. All reconstruction techniques require the ability to align or calibrate imagery along epipolar lines, which represent the pixels in multiple images that project along the same plane. This calibration enables correlation and then 3D backprojection of points along corresponding epipolar lines. Projection is considered the process of projecting points in an environment onto a planar pixel sensor array. Backprojection is considered the reverse process of mapping 2D image points back into 3D coordinates.

Lens distortion must also be considered in epipolar alignment, because it will cause warped image projections. Also, relative camera poses are either unknown or have some measurement error which must be corrected. Therefore, this section summarizes common lens and camera models used in calibration of imagery for reconstruction. Generally, the calibration parameters are automatically extracted from features that can be automatically detected and matched between the images, later covered in sections 2.4 and 2.5.

2.2.1. Camera Calibration

Camera calibration is split between intrinsic and extrinsic parameters. The intrinsic parameters are unique for each sensor and lens combination and include radial lens distortion, focal length, and the center point of projection. The extrinsic parameters include the rotation and translation of the camera relative to some world-fixed coordinate system.

The following calibration models are summarized from the OpenCV library's C implementation, and previously documented by *Bradski and Kaehler* (2008) and *Bradski* (2011). If we let (u, v) be the coordinates of a 3D point (X, Y, Z) in an image after projection, the transformation is described by equations 2.2.1-2.2.3.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = [\mathbf{R}|t] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.2.1}$$

In equation 2.2.1, $\mathbf{R} \in \mathbb{R}^{3\times3}$ is the rotation matrix and $t \in \mathbb{R}^3$ is the location vector of the camera relative to the environment, also known as the extrinsics. The projection of the 3D point to the 2D image is given by 2.2.2.

$$\begin{bmatrix} \bar{x} \\ \bar{y} \end{bmatrix} = \begin{bmatrix} x/z \\ y/z \end{bmatrix} \tag{2.2.2}$$

Lens distortion effects, if present, are accounted for with equation 2.2.3.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & \alpha & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{K} \begin{bmatrix} \bar{x}\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) + 2p_1\bar{x}\bar{y} + 2p_2\left(r^2 + 2\bar{x}^2\right) \\ \bar{y}\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) + 2p_1\left(r^2 + 2\bar{x}^2\right) + 2p_2\bar{x}\bar{y} \\ 1 \end{bmatrix} \tag{2.2.3}$$

$$r^2 = \bar{x}^2 + \bar{y}^2$$

Here, the parameters f_x and f_y are the focal length scaled to a pixel count. This is calculated separately in the two dimensions, because the sensor may have different spacing in each dimension of the pixel array. Also, the center of projection (c_x, c_y), in pixels, is also where ($\bar{x}$, $\bar{y}$) = $\mathbf{0}$, and is the center point of the radial distortion. Therefore, it is the only point unaffected by the distortion. The parameter, α, is the skew of the camera, which should be near 0. However, due to shutter scanning from the top to bottom of the sensor, camera motion may result in skew. The camera matrix, K, includes the focal lengths, center of projection, and skew parameters.

The distortion coefficients, (k_1, k_2, p_1, p_2 [, k_3]) (with optional k_3), is a generic model for most optical lenses (*Heikkila and Silven*, 1996). The model which excludes k_3 has been found to be sufficient for this research.

Note that if the distortion coefficients can be assumed to have negligible effects, then the calibration reduces to equation 2.2.4 (*Lourakis and Argyros*, 2009).

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & \alpha & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} [\mathbf{R}|t]}_{\mathbf{M}} \underbrace{\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}}_{\mathbf{X}} \tag{2.2.4}$$

Here, $\mathbf{M}$ is the projection matrix and is an arbitrary homogeneous 3×4 matrix with rank 3 and depends on 11 parameters. It is also clearly possible to rectify an image to correct for radial distortion, and still quantify the remaining calibration with this 3×4 matrix. With known distortion coefficients, an undistorted point ($\bar{x}$, $\bar{y}$) can be found from raw coordinates (u, v) by solving for the inverse of a heavily non-linear function iteratively. This domain ($\bar{x}$, $\bar{y}$) is generally preferred when matching features, or calibrating extrinsics, primarily because all transformations of corresponding points between different camera views are affine.

Allowing the intrinsic parameters (camera matrix and distortion coefficients) to vary some about the prior calibration is important when performing mosaicking or matching between images. This is because they can actually change dynamically due to thermal or mechanical variations, including those induced by vibration. It is important to know the maximum variation of camera intrinsics, because with a smaller variation, a smaller search space can be imposed during refinement.

2.2.2. Stereo Imagery

In a calibrated two camera stereo system, not only can the cameras' intrinsics be calibrated a priori, but also their relative extrinsic parameters including rotation and translation. This enables projection of the two images onto the same plane, which aligns epipolar lines between the images. In this process, the images are also normalized relative to each other reducing the calibration data to a smaller set of intrinsics, $[f, c_{xl}, c_{xr}, c_y, T_x]$. Here, the focal length, f, is normalized between the cameras in the x and y directions. Also, the principal points of the two images are aligned in y, such that $c_{yl} = c_{yr} = c_y$, but the x positions are left independent as c_{xl} and c_{xr}.

Disparity is the difference in x-coordinates between a matched point the left and right images, $d = x_l - x_r$, which holds a linear relationship to the 3D coordinates of a point which has been matched between left and right images. The disparity must be normalized by the difference in x coordinates of principal points, $c_{xl} - c_{xr}$. The translation between cameras in the x direction is represented by T_x, in millimeters. From this domain, the disparity-to-depth mapping matrix in equation 2.2.5 will project a left/right point match to a 3D coordinate. Here, x is right in the image (horizontal), y is down in the image (vertical), and z is into the image (depth) in the direction of the stereo-rectified principal point of the left camera.

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & -c_{xl} \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f_x \\ 0 & 0 & \frac{1}{T_x} & \frac{c_{xl}-c_{xr}}{T_x} \end{bmatrix} \begin{bmatrix} x_l \\ y_l \\ d \\ 1 \end{bmatrix} \tag{2.2.5}$$
$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} X/W \\ Y/W \\ Z/W \end{bmatrix}$$

2.2.3. Rotations

2.2.3.1. Rotation Matrix Derivations

Let a 3D rotation be described by $\mathbf{a} = \theta\,[x, y, z]^T$, where $\hat{\mathbf{a}} = [x, y, z]^T$ is a unit vector for the axis of rotation, and $\|\mathbf{a}\| = \theta$ is the magnitude of that rotation. It is known that rotation of a point p_1 to p_2 can be described by a rotation matrix such that $p_2 = \mathbf{R}p_1$. The deriviation of this was found in *Hartley and Zisserman* (2004), and the results are in equation 2.2.6[1].

$$\begin{aligned} \mathbf{R} &= \cos\|\mathbf{a}\|\, I + \operatorname{sinc}\|\mathbf{a}\|\, [\mathbf{a}]_\times + \tfrac{1-\cos\|\mathbf{a}\|}{\|\mathbf{a}\|^2}\mathbf{a}\mathbf{a}^T \\ &= I + \sin\theta\,[\hat{\mathbf{a}}]_\times + \left(1 - \cos\theta\,[\hat{\mathbf{a}}]_\times^2\right) \end{aligned} \tag{2.2.6}$$

More explicitly, *Gruber* (2000) derived equation 2.2.7, which generates a rotation matrix quickly in software from the four parameters (x, y, z, θ).

$$\mathbf{R} = \begin{bmatrix} \gamma x^2 + c & \gamma xy - sz & \gamma xz + sy \\ \gamma xy + sz & \gamma y^2 + c & \gamma yz - sx \\ \gamma xz - sy & \gamma yz + sx & \gamma z^2 + c \end{bmatrix} \tag{2.2.7}$$
$$c = \cos\theta,\ s = \sin\theta,\ \gamma = 1 - \cos\theta$$

To convert from rotation matrix to axis/angle, the trace is considered (equation 2.2.8), and the angle is then derived by equation 2.2.9. This is to be used for matching 3D

[1] The notation $[\cdot]_\times$ denotes a cross-product with the expression or vector on the right-hand side.

rotations.

$$\begin{aligned} \operatorname{tr}\mathbf{R} &= \left(\gamma x^2 + c\right) + \left(\gamma y^2 + c\right) + \left(\gamma z^2 + c\right) \\ &= (1-c)\underbrace{\left(x^2 + y^2 + z^2\right)}_{=1} + 3c \\ &= 1 + 2cos\theta \end{aligned} \tag{2.2.8}$$

$$\theta = \cos^{-1}\left[\left(\operatorname{tr}\mathbf{R} - 1\right)/2\right] \tag{2.2.9}$$

2.2.3.2. Rotation Estimation between 3D Point Matches

The least-squares algorithm to directly estimate the rotation between two 3D point matches, $(p_i,\, p'_i)_{i=1,\ldots,N}$ was originally presented by *Arun et al.* (1987) is summarized below.

1. Calculate $p' \triangleq \frac{1}{N}\sum_{i=1}^{N} p'_i$ and $p = \frac{1}{N}\sum_{i=1}^{N} p_i$
2. Center the points, $q_i \triangleq p_i - p$ and $q'_i \triangleq p'_i - p'$
3. Calculate $H \in \mathbb{R}^{3\times 3}$, $H \triangleq \sum q_i q_i'^T$
4. Find the SVD, $H = U\Lambda V^T$, then calculate $X = VU^T$ and the determinant, $\det X = |X|$
   a) If $\det X = 1$, then $\mathbf{R} = X$
   b) If $\det X = -1$, then the algorithm fails

The algorithm may fail if the points are colinear, or the noise is too large. Arun recommends a "RANSAC-like technique" to attack the latter case. However, the Iteratively-Reweighted Least Squares (IRLS) algorithm (see section 2.3.4) has comparable performance to RANSAC, for its rejection of outliers, primarily when coupled with robust median estimators. IRLS does not operate in a random fashion, but isolates outliers iteratively (see section 2.3.4).

2.3. Estimation Theory

2.3.1. Median and Median Absolute Deviation

In many practical applications, modeling measurement errors with a Gaussian distribution is sufficient. However, outliers can easily throw off results. Therefore, a Gaussian estimator that is robust to outliers is desired. So, instead of the sample mean, $\mu = E[X]$, the sample median, $\hat{\mu} = \text{median}(X)$, can be used instead. Furthermore, to estimate the standard deviation, $\sigma = E\left[(X-\mu)^2\right]$, the median absolute deviation (MAD), $\hat{\sigma} = \text{MAD}(X) = 1.4826\,\text{median}(|X-\hat{\mu}|)$, can be used instead (*Mili*, 2006a). The scalar value in this estimator is a derived bias relative to the standard deviation of a Gaussian distribution. It is important to realize that the median is robust to less than 50% outliers biased on one side of the mean. If the outliers occur on both sides, then it is robust to even more than 50%.

2.3.2. Shortest Half

A more adaptive, robust estimator is the shortest half (*Mili*, 2006a). After sorting the samples, the shortest half is found by calculating the difference between $\frac{N}{2}$ samples, and choosing the location with the smallest difference. Consider the following example with $N = 10$, being the number of samples.

$$\underbrace{\mathbf{1}\ 3\ 6\ 7\ \mathbf{7}}_{7-1=6}\ 8\ 10\ 15\ 19\ 21$$

$$1\ \underbrace{\mathbf{3}\ 6\ 7\ 7\ \mathbf{8}}_{8-3=5}\ 10\ 15\ 19\ 21$$

$$1\ 3\ \underbrace{\mathbf{6}\ 7\ 7\ 8\ \mathbf{10}}_{10-6=4}\ 15\ 19\ 21$$

$$1 \quad 3 \quad 6 \quad \underbrace{\mathbf{7} \quad 7 \quad 8 \quad 10 \quad \mathbf{15}}_{15-7=8} \quad 19 \quad 21$$

$$1 \quad 3 \quad 6 \quad 7 \quad \underbrace{\mathbf{7} \quad 8 \quad 10 \quad 15 \quad \mathbf{19}}_{19-7=12} \quad 21$$

$$1 \quad 3 \quad 6 \quad 7 \quad 7 \quad \underbrace{\mathbf{8} \quad 10 \quad 15 \quad 19 \quad \mathbf{21}}_{21-8=13}$$

With $N = 10$, each consecutive set of 5 numbers is considered and the difference between the maximum and minimum is taken. The set $[6, 7, 7, 8, 10]$ is then the shortest half, and the mean and standard deviation can be estimated from this set.

2.3.3. Regression

The ordinary least-squares is the simplest regression algorithm, which has a closed-form solution for the estimate given by equation 2.3.1.

$$\mathbf{R} = \left(X^T X\right)^{-1} X^T y \tag{2.3.1}$$

In a true Guassian-noise case, this is unbiased and consistent. However, with non-Guassian noise containing outliers, this algorithm may become highly biased to an outlier. Instead, if the covariance of the noise is known, the regression becomes equation 2.3.2 (*Mili*, 2006c).

$$\mathbf{R} = \left(X^T \Sigma^{-1} X\right)^{-1} X^T \Sigma^{-1} y \tag{2.3.2}$$

2.3.4. Iteratively Reweighted Least Squares

The iteratively reweighted least squares (IRLS) algorithm can be used to perform outlier rejection and provide a robust estimate (*Mili*, 2006b). With IRLS, optimization is performed over a weighted least squares function as in equation 2.3.3.

Algorithm 2.1 Generic IRLS Algorithm

1. Initialize weights with any prior information:
   $w \Leftarrow w_0$
2. Initialize ϵ:
   $\epsilon \Leftarrow \infty$
3. Estimate initial β (i.e. regression estimator):
   $\beta \Leftarrow f_{est}^{-1}(y,\, w)$
4. while $\epsilon > \epsilon_{max}$
   a) Calculate the residuals:
      $r \Leftarrow y - f(\beta)$
   b) Re-calculate weights (i.e. Huber, equation 2.3.5):
      $w \Leftarrow g(r)$
   c) Estimate β:
      $\beta_{new} \Leftarrow f_{est}^{-1}(y,\, w)$
   d) $\epsilon \Leftarrow \max(|\beta - \beta_{new}|)$, $\beta \Leftarrow \beta_{new}$

$$\text{argmin}_\beta \sum_{i=1}^{n} w_i(\beta)\,|y_i - f_i(\beta)|^2 \tag{2.3.3}$$

The algorithm iteratively assigns the weights, w, based on the residuals, $r = y - f(\beta)$, using a cost function, such as the Huber cost function (equation 2.3.4). The weights are then calculated by equation 2.3.5. This benefits from the ability to give outliers an equally high cost, such that they do not affect the optimization result. The factor τ determines within how many standard deviations to give the index a weight of 1. Any normalized residual, $r_i/\hat{\sigma}$, outside τ standard deviations then has a proportionally decreasing weight.

$$\Psi(z,\tau) = \begin{cases} z, & |z| \le \tau \\ \tau\,\text{sign}(z), & |z| > \tau \end{cases} \tag{2.3.4}$$

$$w_i = \frac{\Psi(r_i/\hat{\sigma},\tau)}{r_i/\hat{\sigma}},\ \hat{\sigma} = \text{MAD}(r) \tag{2.3.5}$$

The generic IRLS algorithm is shown in algorithm 2.1.

2.4. Feature Types

For a 3D reconstruction application, features can be used to serve as anchor points when deriving camera calibration parameters. It is unimportant what these anchor points actually represent, but it is critical that they are projected from stationary surfaces in the environment and can be accurately localized. Features that are invariant to changes in camera pose and lighting conditions are also desireable. So, the image processing field has generated many types of features with repeatable, robust detection properties. This report will first discuss edge features, followed by point features encompassing KLT, MSER, SIFT and SURF. For more detailed discussions and exploration of other feature types, see *Tuytelaars and Mikolajczyk* (2008).

2.4.1. Edges

Edges are one type of feature which are generally detected by a high gradient response in a single direction. If a detected edge is linear, then it is relatively simple to fit a line to it, otherwise it is considered a more generic contour. Matching these simple features can attain sub-pixel accuracy with a high precision (*Pilz et al.*, 2009). They are very useful in indoor, structured environments where edge features are generally more prominent than point features.

A common edge detector is the Canny edge detector (*Canny*, 1986), or Canny-Deriche algorithm which is more suitable for real-time operation (*Deriche*, 1987). First, a LoG or DoG is run across the image. Then a Sobel operator is applied to approximate the horizontal and vertical derivatives to find edges, and then simply uses the arctangent to find the edge direction. Next, edges are traversed using the directions, and any nonmaximum value is set to 0 (if it's not an edge). Lastly, a two-fold thresholding algorithm is applied. First, any pixel above threshold T_1 is considered an edge. Second, any pixel connected to the found edges above threshold T_2 (where $T_2 < T_1$), is also considered an edge.

It is also important to note that while a straight line in an environment (i.e. intersection of a wall and ceiling or floor) are easily detectable and repeatable, a static contour

can change shape drastically from multiple perspectives when it has a 3D structure. However, algorithms like snakes (*Kass et al.*, 1988) has some potential for overcoming these problems and detecting the same contour between image sequences.

2.4.2. Points

Contrary to an edge, a point cannot exhibit a partial occlusion–it is either detected or not. This makes feature points much simpler and more straight-forward to process. However, the region around a point is typically used for classification, which would still undergo some affine change.

An image pixel is considered a feature point if it is located at the maxima or minima of an arbitrary measure, such as the determinant of Hessian, and is above some minimum threshold (*Mikolajczyk and Schmid*, 2004). By performing the operation at multiple scales, and finding the corresponding maxima and minima at each scale, features can also be generated at these different scales. This section summarizes some of the common feature detection algorithms and their strengths and weaknesses.

2.4.2.1. Kanade-Lucas-Tomasi

The Kanade-Lucas-Tomasi (KLT) feature tracker (*Tomasi and Kanade*, 1991) assumes a small translation between two consecutive images, and therefore, the pixel-to-pixel errors between the two images and the gradient (found from only one of them) can be used to approximate the translation. Simply put, derivation of a 2×2 system, G, enables formulation of the 2D translation, $\mathbf{d}$, as Equation 2.4.1, where I and J are successive images, $\mathbf{g}$ is the gradient, and w is an arbitrary weighting function (such as a Gaussian).

$$G\mathbf{d} = \int_W (I - J)\,\mathbf{g} w dA \tag{2.4.1}$$

So, G is then a weighted average of the gradients within the window, W, and is comprised of the estimated terms in Equation 2.4.2.

$$G = \int_W \mathbf{g}\mathbf{g}^T w dA = \begin{bmatrix} \left(\frac{dI}{dx}\right)^2 & \frac{dI}{dx}\frac{dI}{dy} \\ \frac{dI}{dx}\frac{dI}{dy} & \left(\frac{dI}{dy}\right)^2 \end{bmatrix} \tag{2.4.2}$$

This is a convenient result, and simple to calculate numerically. To select prominent features on which to base this result, the algorithm uses the eigenvalues of a given G matrix. The two eigenvalues must be above some threshold (to reject sensor noise over a surface without intensity variation), but also not differ by several orders of magnitude.

This tracker has been the standard for determining optical flow, generally from high frame rate (>10fps), and relatively low resolution (VGA quality, or 640×480 pixels) video. This can certainly be applied to higher resolution imagery too, but would possibly require down-sampling of the image, thus reducing accuracy, if a high frame rate cannot be achieved. Also, with high resolution imagery, only a few small, accurate, and robust features are needed for successful tracking. KLT is not suitable to match features from wide-baseline viewpoints. Wide-baseline implies a large change in perspective to better reconstruct a 3D scene, in which case the KLT small translation assumption does not hold.

2.4.2.2. Scale and Affine Invariant Features

Any viewpoint change can be characterized by an affine change in the projected image. Considering just scale differences (i.e. camera zoom), the imagery can be analyzed in scale-space using different sized Gaussian filters, or with pyramidal down-sampling.

An affine invariant detector generalizes this scale-invariant approach to include both scaling and skew. Simply put, a scale-invariant detector can fail with significant affine transformations (i.e. wide-baseline views). These methods include Edge-Based Regions (EBR), Intensity-Based Regions (IBR), and Maximally Stable Extremal Regions (MSER) (*Tuytelaars and Mikolajczyk*, 2008). EBR starts from Harris corners and two intersecting edges. The directions of those edges determine a parallelogram, and some measure of texture determines the length of the edges. With IBR, local intensity extrema are first extracted, then an ellipse is fitted to a region corresponding to intensity changes

around each extremum. MSER, however, begins with image segmentation using intensity thresholds. Then, if the pixels within a segment are either greater than or less than all the intensities in the neighboring segments, it is considered an extremal region. This image segment, or connected component, can then be fitted to an ellipse to generate an affine-invariant feature.

2.4.2.3. Scale-Invariant Feature Transform

The Scale-Invariant Feature Transform, or SIFT, is a widely used feature detector. It is not affine-invariant, but does have very good performance across most viewpoints. It begins by calculating Gaussian-smoothed images at multiple scales. Then, it calculates the DoG from these at each scale, thus creating a scale-space. To extract keypoints from the DoG, rather than just a high gradient response or intensity maximum, it uses the determinant of a 2×2 Hessian (*Lowe*, 2004). Then, orientation alignment is performed with a Histogram of Gradients (HOG) filter. Lastly, the area around the point, with size chosen according to its scale, is split into 4×4 sub-regions. Then, a 4 or 8-bin HOG is calculated for each sub-region to produce either a 64 or 128-element descriptor vector. The descriptor is normalized to have an ℓ_2-norm of 1, to accommodate lighting variances. Repeatability of this detector was originally demonstrated to be greater than 70% with 10% noise introduced on an image.

2.4.2.4. Speeded-Up Robust Features

Speeded-Up Robust Features (SURF), is a much faster approximation of the SIFT algorithm, which is more widely used because of its execution speed. It first calculates an integral image (equation 2.4.3). Then, the determinant of the Hessian is approximated using box filters[2], which run at an identical speed at each scale–the primary time-consuming factor with the original SIFT algorithm's detector.

[2] Any non-rotated rectangular, or box, sum of an image can be found with four accesses of the integral image: $\sum_{j=y_1}^{y_2} \sum_{k=x_1}^{x_2} I(k,\, j) = I_{\iint}(x_1,\, y_1) + I_{\iint}(x_2,\, y_2) - I_{\iint}(x_1,\, y_2) - I_{\iint}(x_2,\, y_2)$

$$I_{\iint}(j,\, k) = \sum_{m=0}^{j} \sum_{n=0}^{k} I(m,\, n) \tag{2.4.3}$$

For orientation alignment, SURF then uses Haar wavelets implemented with box filters on the integral image. The orientation is chosen at the angle where the response of the Haar wavelets is highest. It also generates an oriented descriptor by splitting the region around each feature into 4×4 sub-regions, and stores Haar wavelet responses within each region. It also generates either a 64 or 128-element descriptor, by optionally storing positive and negative Haar wavelet responses separately (*Bay et al.*, 2006). Like SIFT, it normalizes the descriptor to have an ℓ_2-norm of 1. However, it runs in approximately 1/3 the time as SIFT, and has similar repeatability characteristics (82% vs. 78% originally shown by Bay).

This particular algorithm was chosen for this research because it provides multi-scale features and can run relatively quickly for real-time wide-baseline matching.

2.5. Matching

2.5.1. Basic nearest-neighbor

Several matching algorithms are available for matching feature descriptor vectors. Ideally, a matching algorithm will minimize the 2-norm between the vectors. This is known as the "nearest neighbor". Say, features are generated from two images producing respective sets A and B, with sizes N_A and N_B. The simplest algorithm for finding $a_i \in A \,|\, \|a_i - b_k\|_2 \,,\, b_k \in B$ is to iterate over $i = 1, \ldots, N_A$. This produces an $O(N_A N_B)$ algorithm, which is exhaustive since it computes the norm for every possible combination. The square-root portion of the norm calculation is usually omitted to save computation time, which is then considered as the sum of squared differences or SSD (*Bay et al.*, 2006).

2.5.2. kD Tree

A k-dimensional (kD) tree structure can be used to sort one set of the features, and increase matching time. This structure, similar to a binary tree or quad tree is designed to accommodate an arbitrary number of dimensions. The tree structure begins by splitting the first dimension of data in two branches, based on the first feature that was added. As the algorithm moves down the tree, it cycles the comparisons through each dimension of the data. Often, for high dimensional descriptor vectors, just the first 5 or 10 elements are used in the kD tree. The complexity of building a tree is an $O(n \log n)$ operation. Searching the tree (if well-balanced) is an $O\left(n^{1-1/k} + m\right)$ operation, where m is the number of points within the requested hypersphere.

Bay et al. (2006) used an approach where a kD tree would store more than 100,000 points, and when searching for a nearest-neighbor, the tree would provide only the first 200 candidates.

The kD tree's strengths are for matching between a vast amount of features, where little is known about the feature's position or size, as feature matching algorithms are designed specifically for monocular algorithms. However, sorting, searching and balancing a tree over k dimensions is a complex task, which this research plans to avoid. Since 3D information can be found within a given stereo pair, this research will perform searches over 3D coordinates, and determine the nearest-neighbor from features within some distance of a target location.

2.5.3. Group Matching

Group matching has proven to increase robustness of match results (*Jung and Lacroix*, 2001). Group matching entails finding a small set of features–roughly 3 to 10–that undergo an affine change between frames. This follows from a planar assumption of the data. If, say, 3 features lying on a plane that is parallel to the image plane, then as the camera translates along the image plane, the features will translate together. However, if the features lie on a plane that isn't parallel to the image plane, which is much more likely, then their relative translation on the projective image plane will instead result in

an affine transformation. This method will reject false matches that do not experience the same affine transformation at other feature matches.

This research will use this concept, except with estimates of the transformation given by pose measurements. Each frame in the stereo system can generate 3D information, so features will first be projected to 3D before matching. Then, the algorithm will maximize the size of the group with the best feature matches in 3D that conforms to the pose measurement.

2.5.4. RANSAC

Random Sample Consensus (RANSAC) is a widely used algorithm in many practical signal processing algorithms that require real-time operation (*Hartley and Zisserman*, 2004). When used to match feature set A and B, random features are selected out of set A and matched across B either at random or with a kD tree. The critical assumption, however, is that only a few of the good matches are needed, and that there are many good unique feature matches between the two sets. This allows it to quickly find just a few matches that are sufficient for pose estimation. However, in a homogeneous environment, where feature descriptors are very similar, RANSAC will find itself spinning its wheels on searching for unique features. Also, the question arises as to "how many" comparisons is "enough" to accept a match as being unique, under the nearest-neighbor criterion in section 2.5.1.

RANSAC certainly proves well in practice, and it is convenient to abort a particular matching sequence if it takes too long. However, this report considers the case where RANSAC fails in real-time, because it will reduce to an exhaustive search in a homogeneous environment which provides non-unique features. In this case, the nearest-neighbor criterion will prevent correct matches from being accepted, and could potentially cause aliasing if thresholds are lowered.

2.6. Bundle Adjustment

Bundle adjustment (BA) is the general optimization problem for finding the calibration parameters of an arbitrary number of cameras with an arbitrary number of poses. The "bundles" referred to in this context are the bundles of light rays projecting into each camera, usually representing features that have been previously matched. Each bundle is rigid–the relative angles between each ray within a bundle is accurately known. However, the relative camera poses are not. For the purposes of this paper, we will assume that radial distortion coefficients are known and are already corrected.

The BA algorithm is to find the m camera projection matrices, $\mathbf{M}_i$ (see section 2.2.1), and the n 3D points $\mathbf{X}_i$, such that each matched image point $\mathbf{x}_{ij}$ is closely approximated by $\mathbf{M}_j\mathbf{X}_i$. For example, consider the optimization problem in equation 2.6.1. We can see that this is a very high degree-of-freedom problem with $11m + 3n$ variables. It is clear that having some approximation of each variable from pose measurements and prior calibration estimates, will reduce the solution space.

$$\min_{\forall \mathbf{M_j}, \mathbf{X_i}} \left\| \mathbf{M}_j\mathbf{X}_i - \lambda\mathbf{x}_{ij} \right\|, \; j = 1, \ldots, m, \; i = 1, \ldots, n \tag{2.6.1}$$

2.6.1. Levenberg-Marquardt Algorithm

Levenberg-Marquardt (LM) optimization is a widely-used algorithm for the nonlinear least squares problem in BA which is a blend of both vanilla gradient descent and Gauss-Newton iteration. The following is a summary of LM from *Ranganathan* (2004).

LM estimates the gradient (equation 2.6.2) and the Hessian matrix (equation 2.6.3) from the Jacobian.

$$\nabla f = \sum_{j=1}^{m} r\left(x\right) \nabla r_j\left(x\right) = J\left(x\right)^T r\left(x\right) \tag{2.6.2}$$

$$\nabla^2 f \cong J\left(x\right)^T J\left(x\right) \tag{2.6.3}$$

This assumes that the residuals (or errors from the estimation) are small, and therefore,

LM cannot be used in problems with large residuals. Simple vanilla gradient descent being the most intuitive technique to find a minima, suffers from the fact that it takes small steps when the gradient is small, and large steps when the gradient is large. Small steps are desired in regions where the gradient is large, because it would be easy to escape the minima region. Large steps are desired in regions where the gradient is small because it would take many steps otherwise.

We can use the gradient information to generate an update rule - also known as Newton's Method. Assuming f to be quadratic, high order terms of the Taylor expansion are set to 0, when solving$\nabla f = 0$:

$$x_{i+1} = x_i - \left(\nabla^2 f\left(x_i\right)\right)^{-1} \nabla f\left(x_i\right)$$

Using the above approximations, Levenberg proposed an algorithm based on the update rule:

$$x_{i+1} = x_i - \left(\mathcal{H} + \lambda I\right)^{-1} \nabla f\left(x_i\right)$$

Where $\mathcal{H}$ is the Hessian matrix at x_i. The algorithm is then:

1. Calculate x_{i+1}based on the update rule
2. Evaluate the error at the new parameter vector
3. If the error has increased, then retract the step, and increase λ by α
4. If the error has decreased, then accept the step and decrease λ by α

The adjustment factor, α, may be set to 10 or some large amount. It does, however, reduce the effect of the Hessian on the result for large λ.

Marquardt, however, observed that larger movements could occur where the gradient is smaller, so that the "error valley" problem no longer occurs. Therefore, he replaced the use of the identity matrix with the diagonal of the Hessian, producing the Levenberg-Marquardt update rule in equation 2.6.4.

$$x_{i+1} = x_i - (H + \lambda \operatorname{diag} \mathcal{H})^{-1} \nabla f(x_i) \tag{2.6.4}$$

The LM method will generate a locally optimal solution, which is not guaranted to generate a globally optimal solution. However, it works very well in many applications, and is, in general, much faster than any other algorithms. Since it adaptively moves across the search space using gradient and Hessian information, LM is considered a simpler approximation of trust-region methods (*Berghen*, 2004).

2.6.2. Sparse Bundle Adjustment

The sparse bundle adjustment (SBA) package provides a Levenberg-Marquardt implementation customized for feature matches. Most implementations of Levenberg-Marquardt had not taken advantage of the fact that the normal equations matrix has a sparse block structure, due to the lack of interaction among parameters for different 3D points and cameras. Therefore, the SBA package is a tailored an implementation to the BA problem (*Lourakis and Argyros*, 2009). The SBA package is an ANSI C implementation and enables the optimization of an arbitrary number of camera calibration parameters, including both intrinsic parameters and extrinsic pose parameters. The SBA package was tested over several test image sequences, and the pixel errors of features were refined to be as low as a quarter pixel.

2.6.3. Simple Sparse Bundle adjustment

The Simple Sparse Bundle Adjustment (SSBA) library is another newer implementation of sparse BA. Written by researchers at the University of North Carolina, it takes advantage of the SuiteSparse matrix library from the University of Florida. The SuiteSparse library provides fast routines for Cholesky and LDL factorization for sparse matrix applications. SSBA also implements the optimization as a Levenberg-Marquardt algorithm. It provides two primary optimization interfaces: one that assumes the same camera generated each image (also referred to as "common" in the library), and one that allows unique calibration parameters to be varied and optimized for each image (referred to as

"varying"). Like SBA, this library requires prior feature matching by the user, with little or no outliers, and estimates of their 3D coordinates. The latter optimizer was chosen, to account for use of two stereo cameras, and also to allow the intrinsics to be corrected for any possible lens vibration. SSBA claims to have processed over 1745 images in about 16 minutes. However, this does not include either feature extraction or matching.

2.7. Simultaneous Localization and Mapping

For a real-time 3D reconstruction or mapping task, simulatenous localization and mapping (SLAM) techniques are commonly employed. SLAM is a time sequential approach, which estimates both sensor location and locations of objects in the environment from sensor data such as LIDAR or cameras. When mounted to a vehicle or robot, the sensor location is also used to estimate the vehicle locatiom. The difficulty with this task is that with innacurate or no direct measurements on vehicle location, all localization errors are dependent on measurements of static object locations relative to the vehicle. The random walk in a particular SLAM method is generally used as a measure of accuracy. If the environment allows it, GPS may be used to reduce random walk errors.

Liu and Dai (2010) points out that airborne visual SLAM poses many challenges. Generally, UAVs have a large amount of dynamic motion. In our case, with a relatively small helicopter, wind disturbances can greatly affect the accuracy of position estimates or GPS measurements. Also, the brightness of terrain varies greatly due to cloud cover and position of the sun. For low-altitude mapping, environment dynamics may cause problems with tracking. Therefore, the SLAM community may not provide a drop-in solution for the UAV mapping problem.

2.7.1. Monocular SLAM

Newcombe and Davison (2010) developed a monocular vision-based SLAM approach, solely using structure-from-motion (SFM) from a single camera. This approach takes advantage of PTAM or Parallel Tracking and Mapping developed by *Klein and Murray*

(2007). It uses a ground-plane approximation to both track and map features. Mapping is based on choosing keyframes and performing BA, while tracking is run in a separate thread to determine camera jitter and determine matching points. Using this method to track and calibrate camera locations, dense reconstruction is then performed with a hierarchical approach by *Ohtake et al.* (2003). This method is efficient and robust. However, it assumes a static scene, and even a ground plane. In our application, we may not make these assumptions, because vegitation may be blowing in the wind. Also, the scenery may not have a prominent ground plane. For example, if a tree canopy covers the entire view, then it might fit an ellipsoid or sphere rather than a plane. However, from high-altitudes, a planar assumption may hold, as in the AEROSYNTH project (see section 2.8.3). Furthermore, relatively low-resolution, but high frame-rate (640×480, 15-30fps) cameras are used in these approaches. So these algorithms are not necessarily suitable for high resolution, low frame-rate sensors.

2.7.2. Low-Altitude Imagery

Jung et al. (2003) present a simultaneous localization and mapping (SLAM) approach to high-resolution terrain mapping with a stereo imaging system onboard a blimp. The algorithm first calculates stereo correspondences, then performs feature matching between frames. It then selects landmarks, estimates vehicle motion, and updates an extended Kalman filter (EKF) to generate a digital elevation map (DEM). For feature matching, they use a group-matching technique (*Jung and Lacroix*, 2001), which is helpful to reject outliers. Our approach plans intends to follow a similar approach, but extends this group-matching to finding the single largest group match between entire image frames through expoitation of prior stereo calibration.

2.8. Full Reconstruction

No freely available software exists to perform real-time full reconstruction, which includes feature extraction, matching, and BA on intrinsics and extrinsics. Part of the issue is

that the acquisition software must be customized to the particular hardware platform chosen. However, this section will cover projects which have demonstrated free batch processing to perform reconstruction (Bundler), and near-real-time projects, which have demonstrated the feasibility of reconstruction in real-time (Urbanscape), but have not published a working real-time system.

2.8.1. Bundler

Snavely et al. (2007) have developed a freely available software package, Bundler, for 3D modeling from uncalibrated image sets, specifically for the world's well-photographed sites. Bundler relies upon other free software packages such as SBA (see section 2.6.2), SIFT (see section 2.4.2.3), and ANN: A Library for Approximate Nearest Neighbor Searching (see section 2.5.2). This work focuses on reconstructing only sparse 3D models (specifically of features), since they are only concerned about smooth 3D transitions between photographs rather than visualizing the 3D environment. This software package also has a web interface for general use called Microsoft Photosynth.

This tool is highy relevant to this research, but must be highly-tailored to suit a real-time system. One approach would be to use Bundler to correct for errors in camera calibration over sets of four images, or two stereo pairs at a time. This could then be used as input to a SLAM algorithm.

2.8.2. Urbanscape

The Urbanscape project (*Mordohai et al.*, 2007) as a collaboration between University of North Carolina and University of Kentucky is a system to perform real-time 3D reconstruction with a stereo vision system on a ground vehicle. The system consists of a computer with a high-performance GPU, GPS, INS, and eight cameras - four on each side of a ground vehicle. Each set of four cameras have minimal field-of-view overlap, and the two sets are calibrated to perform stereo correlation with a plane sweeping method. The GPS and INS data provide accurate enough pose measurements that sparse feature matches are not needed for correction. The final step is fusion of all stereo measurements

to correct for errors and generate the most accurate and robust geo-registered map.

The selected aerial vehicle, however, does not have the power or weight capacity to take advantage of a high-performance GPU. Also, to reduce cost, power and complexity, a single high-resolution camera with a wide-angle lens can take the place of the four lower resolution cameras in the Urbanscape system. The Urbanscape software is not readily available, and because it is so customized to GPUs, without one available all software must be written from scratch. However, it is expected that it does make use of an SSBA variant, as it was written by the same research group (see section 2.6.3). The Urbanscape team was also rather large (~18 people), so replicating their work would be highly involved.

2.8.3. AEROSYNTH

The Airborne Synthetic Scene Generation or AEROSYNTH project at Rochester Institute of Technology is a monofocal SFM system for a fixed-wing aircraft (*Walli et al.*, 2009). It combines data from an arbitrary number of imagers to provide data of different light spectrums. It uses SIFT points for matching between frames from a single high-resolution camera to correct pose estimates using BA. Then, it generates dense 3D data from the high-resolution camera. Multi-spectral information from other imagers is then overlaid using prior calibration parameters. This system works well because at the altitude the aircraft flies, dynamic scene changes are virtually immaterial and the use of a synchronized stereo system is not needed. Multiple suppliers of high-altitude mapping and dense reconstruction systems similar to AEROSYNTH are readily available in industry, including Urban Robotics and AGI. Of course, all of these require post-processing of the image data and are not implemented in real-time.

3. Methodology

Since generic BA procedures have matured, and can be extended to stereo imagery because of their generic nature, this research primarily focuses on the feature matching process itself by taking advantage of prior stereo calibration. This is expected to improve matching results in applications where robust, unique features are hard to find. Also, it is expected to generate matches faster, because pose estimates are used to reduce the search space. Standard feature point matching algorithms such as a kD-tree do not currently exploit prior camera calibration information, because they are targeted at the generic image matching problem with no pose estimates and no prior intrinsic calibration. It is hypothesized that prior calibration and synchronized pose measurements can provide enough information to overcome problems when too many dynamic or homogeneous features are present. Still, the robustness and accuracy of existing approaches do provide a high standard for result comparison.

This work assumes that a secondary utility is used for dense reconstruction. An open source implementation of semi-global block matching (SGBM) from OpenCV (*Bradski*, 2011) was used throughout this research for dense data generation. Also, the accuracy of the pose refinement algorithms is manually inspected by the dense data alignment between frames.

3.1. Previous Methods

It was found that the work of *Short* (2009) projected 3D points with an estimate of angle derived from pixel location, rather than the projection methods presented in section 2.2.2, which are known to be more accurate. Furthermore, by only using prior stereo

calibration parameters, it was found that dense correlation of pixels with open-source algorithms would not consistently generate accurate results. This was later attributed to vertical offsets in the rectified images (along the y-axis in the image), which were found through manual inspection. A simple feature point matching algorithm was used to correct for these errors (see Appendix A). After correction using feature matches, dense correlation became much more consistent.

The offset has been attributed to delays in the software triggers for the cameras. While the vehicle is in motion, even the slightest offset in synchronization can result in several pixels of difference. Therefore, a hardware trigger is recommended in future designs. The correction is characterized by a vertical bias in the y-axis, and the algorithm does not correct for x-axis translation or rotations about the principal point. Any horizontal bias would cause depth-mapping errors after stereo re-projection, while rotations could cause epipolar misalignment. These further corrections are left to a more involved correction over several image frames, such as BA.

3.2. Hardware Design

To provide synchronized stereo images and pose measurements, a rigid stereo vision system was designed to include an IMU and GPS receiver. The stereo vision system consists of two black and white Sony XCD-U100 FireWire cameras installed on a carbon fiber-epoxy tube with a 1.5 m separation. The system mounted to a Yamaha RMAX helicopter is shown in Figure 3.2.1. The baseline was selected to maximize the accuracy of the stereo system without affecting helicopter stability. The cameras provide 1600×1200 resolution, and lenses were selected with focal length of 8 mm. A single XCD-U100CR color camera is mounted on the left side of the structure to provide a color overlay information. Since the effective resolution of color cameras is decreased by the Bayer pattern, the system instead uses only the black and white cameras in 3D backprojection. Each camera consumes 3 watts or 9 watts total. The spatial accuracy of a terrain map generated from stereo vision depends on camera resolution, baseline distance between the cameras, and the altitude above ground level (AGL). The depth accuracy, or possible

Figure 3.2.1.: Stereo camera system carried on a Yamaha RMAX.

error, for a stereo system's estimation of point, $(x,\ y,\ z)$, relative to a camera pinhole is given by Equation 3.2.1. The wider the baseline, B, and the greater the focal length, f, the lower the error and higher the accuracy. The disparity error, $\triangle d$, affects the accuracy linearly, and the altitude/distance, z, increases the error quadratically. Also, the position in the image affects resolution, where the center of the image has the highest accuracy.

$$\sqrt{\triangle x^2 + \triangle y^2 + \triangle z^2} = \frac{z \triangle d}{Bf} \sqrt{x^2 + y^2 + z^2} \tag{3.2.1}$$

From an altitude of 40 m, the accuracy of the system is shown by Figure 3.2.2. This plot displays the accuracy change across the image, which varies from 56 cm at the principal point to 65 cm at the corners per unit disparity. For other distances, accuracy at the principal point of the image changes according to the quadratic in Figure 3.2.3, while the accuracies of the rest of the image scale similarly. The horizontal resolution for both x and y axes are shown in Figure 3.2.4 as a function of altitude. At 40 m, the resolution is approximately 2.1 cm, which gives a point density of more than 2200 $points/m^2$.

Stereo calibration was performed with a large checkerboard, 1.22×2.44 m, with squares of 12×12 cm from a distance of approximately 7 m. This method was selected to

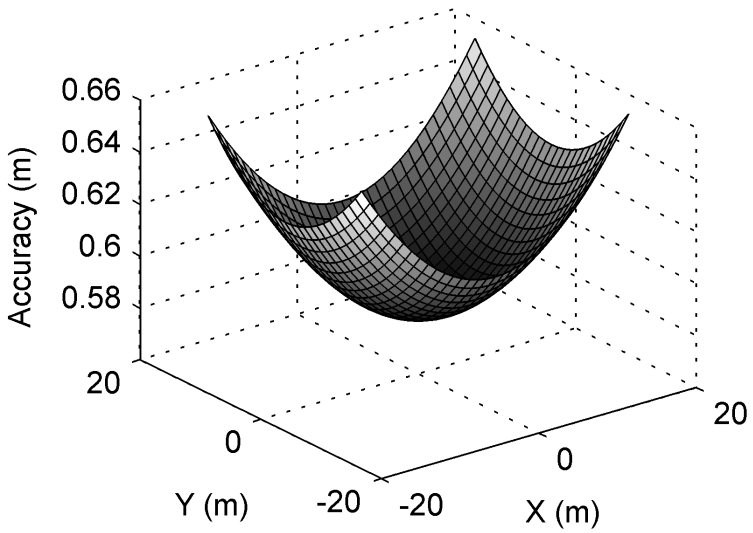


Figure 3.2.2.: Accuracy at 40m distance across stereo field of view per pixel error.

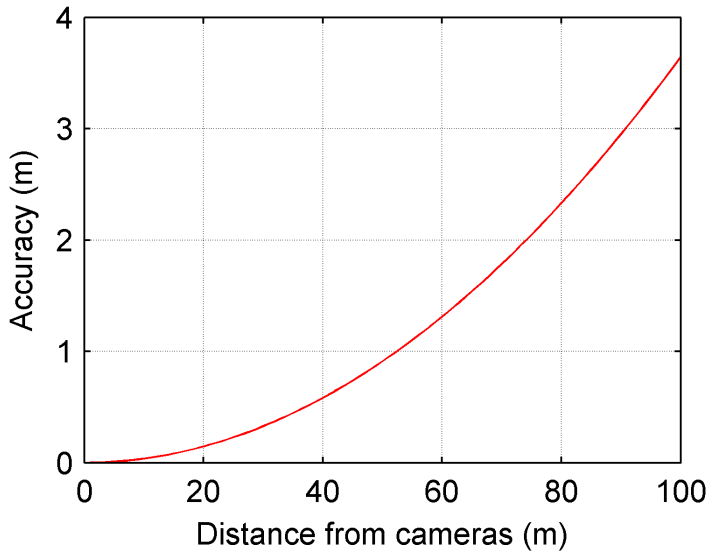


Figure 3.2.3.: Accuracy of principal point versus distance per pixel error.

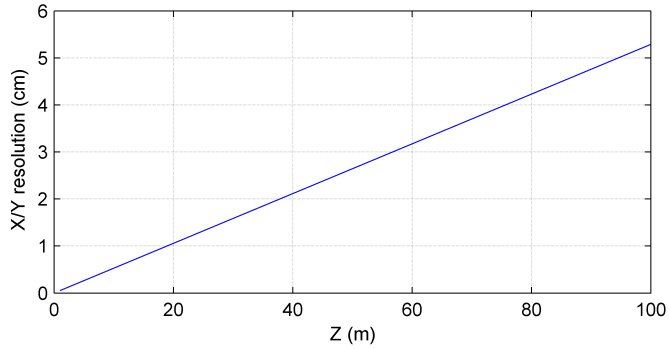


Figure 3.2.4.: Resolution, measured by pixel to pixel distance.

cover the whole field of view in the focal range of the cameras. In addition to the cameras, a Microstrain 3DM-GX3-35 is mounted on the right side of the stereo system, which includes a GPS receiver. The complete system weighs 3.1 kg. The cameras and IMU/GPS are connected to a PC/104 with an Atom processor for synchronization and data collection. Currently, all data is post-processed.

A new system will use two Zelos-655 GV color cameras from Kappa optronics GmbH. These have 5 megapixel sensors, which would provide equivalent resolution to the 1.9 megapixel grayscale cameras. Also, by using just two cameras instead of three, total cost is reduced significantly.

3.3. Coordinate Frames

The definition of the camera coordinate frame is as follows: $+X$ is right in the image, $+Y$ is down in the image, and $+Z$ is into the image. The IMU provides roll (ϕ), pitch (θ), and yaw (ψ) relative to a north-east-down (NED) coordinate frame, where zero yaw corresponds to magnetic north, and the right hand rule is used for each angle. The rotation matrix is defined by equations 3.3.1 and 3.3.2. The camera is physically aligned with the IMU such that equation 3.3.3 is a reasonable approximation of the transformation. We also convert from NED to east-north-up (ENU) coordinates, since this is more conceptual when viewing the output data. So, the final transformation equation from camera to ENU coordinates is summarized by equation 3.3.4.

$$R_{IMU} = \begin{bmatrix} \cos\psi\cos\theta & \sin\psi\cos\theta & -\sin\theta \\ \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \cos\theta\sin\phi \\ \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi & \cos\theta\cos\phi \end{bmatrix} \tag{3.3.1}$$

$$V_{IMU} = R_{IMU} V_{NED} \tag{3.3.2}$$

$$V_{IMU} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} V_{Cam} \tag{3.3.3}$$

$$V_{ENU} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \underbrace{R_{IMU}^{T} \underbrace{\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} V_{Cam}}_{V_{IMU}}}_{V_{NED}} \tag{3.3.4}$$

GPS data is generally provided in latitude, longitude and altitude. These coordinates are converted into a local ENU reference coordinate system, where the origin is chosen at the first GPS reading. GeoStarslib provides the functions for converting GPS coordinates to ENU. It also provides the magnetic declination at a given time and location for IMU yaw measurement correction relative to true north.

3.4. Feature Extraction

The algorithm begins by extracting SURF points while adapting the Hessian Threshold to ensure enough features are found. If too few features are extracted using the default Hessian threshold, the threshold is lowered, and the features are extracted again. This process repeats until at least 3000 features are extracted. This may force the Hessian threshold to be somewhat low, which is generally undesireable since it means the feature

is not very prominent and possibly not repeatable. Since this project targets conditions of homogeneous or dynamic features, we prefer to have many features rather than a handful of prominent features. So, a low Hessian threshold is acceptable. Also, by making the threshold adaptive, plenty of features can be extracted automatically and reduce any user tweaking.

3.5. Feature Matching

3.5.1. Stereo Feature Matching and 3D Projection

The feature point matching algorithm exploits the ability of pre-calibrated stereo cameras to backproject 3D points from two synchronized images. After extracting SURF features from both the left and right images, they are matched using the algorithm originally implemented for aligning the images vertically, found in Appendix A. These matches are then backprojected to 3D using equation 2.2.5, such that the 3D locations are relative to the left camera's center of projection. The scale of the feature is also converted from pixels to meters using equation 3.5.1.

$$\begin{aligned} \sigma_{meters} &= \tfrac{z}{f}\sigma_{pixels} \\ &= \left(\tfrac{T_x}{d+c_{xl}-c_{xr}}\right)\sigma_{pixels} \end{aligned} \tag{3.5.1}$$

Since SURF provides an angle of each feature relative to the original 2D image, the angle is converted to a 3D rotation in the ENU reference frame, and stored as 3 Euler angles extracted from a rotation matrix. The original angle in the image maps to yaw about the z axis in the camera coordinate frame (see section 2.2.2), and roll in the IMU coordinate frame. Rectification of this rotation to local ENU coordinates is then performed with equation 3.5.2. The feature descriptor for a 3D feature point is copied from only the left image.

$$R_{feature} = \underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}}_{NED \to ENU} \underbrace{R_{IMU}^{T}}_{IMU \to NED} \underbrace{\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}}_{Cam \to IMU} \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.5.2}$$

Additional acquisition measures were implemented to ensure that both the left and right cameras used the same exposure setting, and a software trigger was implemented to synchronize the images. The full data flow diagram is shown in Figure 3.5.1.

3.5.2. Spatial Sorting and Searching

This section presents an algorithm for matching two sets of 3D features in camera coordinates, A and B, given pose measurements for each camera from GPS and IMU data. Since this project assumes a nadir view of terrain, 3D features in set A are first sorted based on x and y, or the east-north plane, in a point-region quadtree structure. This enables fast search time complexity of $O(\log n)$, while generation of the structure is $O(n \log n)$. It also provides capability for enumerating potential matches within some radius of another point. The z coordinate relative to the camera is omitted from the quadtree sort because backprojection will generally result in unique x and y coordinates. The only time x and y may not be unique is if the cameras can both see underneath a surface from the side, which is uncommon for a normal angle ($< 40°$) lens.

The pose measurements are assumed to follow a multivariate Gaussian distribution, shown in equation 3.5.3. Each axis is assumed to be independent, so the covariance is defined as a diagonal matrix. This imposes a noise on the measured, geo-located points $a \in A$ and $b \in B$. We also approximate a maximum error in each dimension at 3σ, which is sufficient for 99% of the samples along one axis, under the Gaussian assumption. These errors will be represented by $\triangle x$, $\triangle y$, $\triangle z$, $\triangle \phi$, $\triangle \theta$, and $\triangle \psi$.

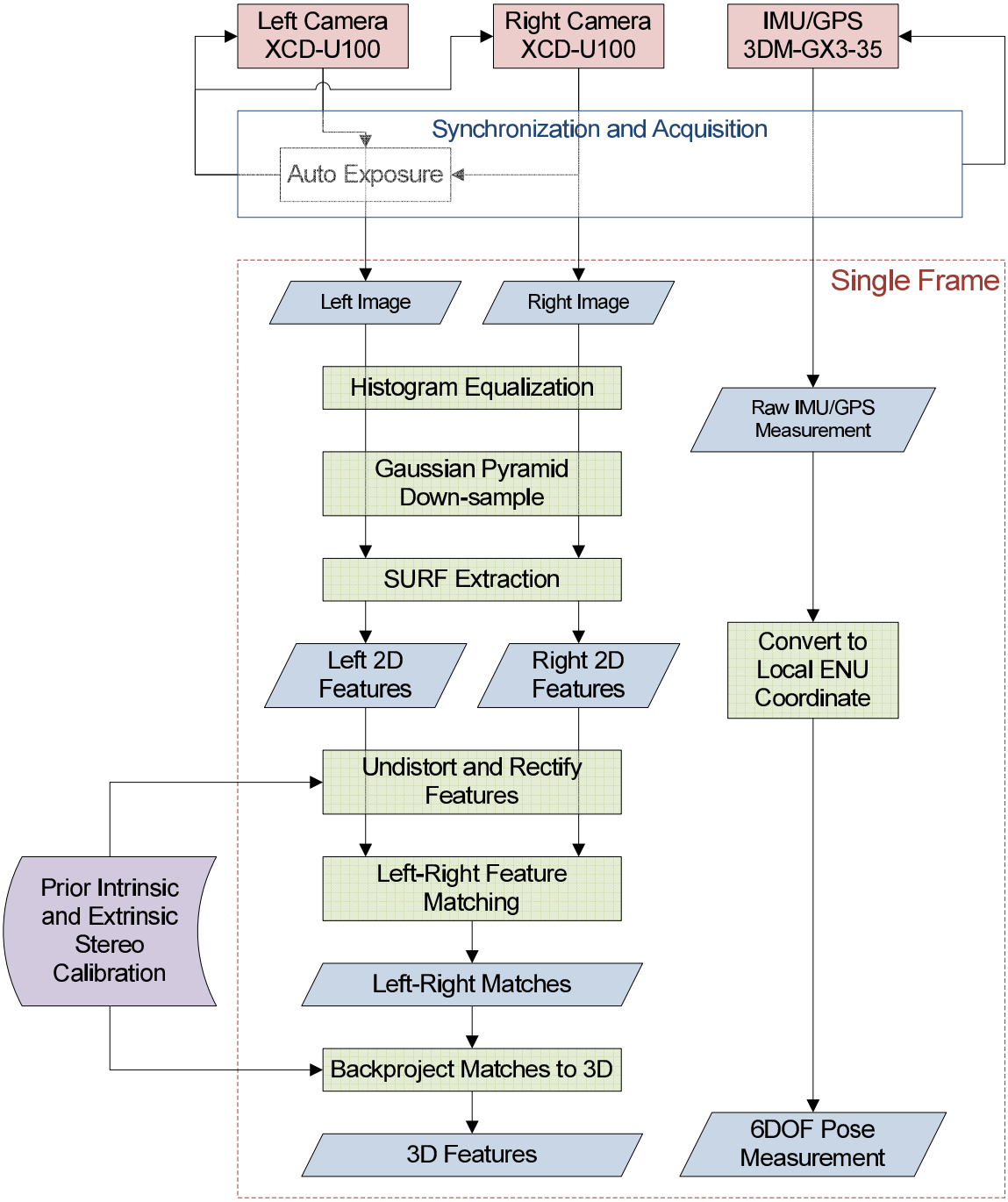


Figure 3.5.1.: Generation of 3D features and 6DOF camera pose measurement

$$\begin{bmatrix} X & Y & Z & \Phi & \Theta & \Psi \end{bmatrix}^T \sim \mathcal{N}\left(\mathbf{0}, \mathbf{\Sigma}^{6\times 6}\right)$$
$$\mathbf{\Sigma}^{6\times 6} = \text{diag}\begin{bmatrix} \sigma_X^2 & \sigma_Y^2 & \sigma_Z^2 & \sigma_\Phi^2 & \sigma_\Theta^2 & \sigma_\Psi^2 \end{bmatrix} \tag{3.5.3}$$

When searching the quadtree, a radial search is performed in x and y (east and north) using a maximum radius approximation. This approximation considers a generic 3D point in frame A (in left camera coordinates), $a = \begin{bmatrix} x_a & y_a & z_a \end{bmatrix}^T$, and the maximum distance it may have in x and y between its second measurement in B. The pitch and roll components are combined into a single distance assuming x_a and y_a to be zero because this provides the maximum error in the x and y dimensions for any x_a and y_a. The effect of yaw is estimated as the base of an isosceles triangle with two edges of length $\sqrt{x_a^2 + y_a^2}$ and the angle between them as the maximum yaw error. The maximum GPS translational error is simply the Euclidean distance of the errors in x and y. The sum of these components defines the total radius displayed in equation 3.5.4. This estimate also enables a single precomputation of coefficients, (α, β, ζ), which reduces calculations when searching for a match of each 3D point.

$$\begin{aligned} r &= z_a\sqrt{\sin^2\Delta\theta + \cos^2\Delta\theta\sin^2\Delta\phi} \\ &\quad +2\sin\tfrac{\Delta\psi}{2}\sqrt{x_a^2 + y_a^2} \\ &\quad +\sqrt{\Delta x^2 + \Delta y^2} \\ &= \alpha z_a + \beta\sqrt{x_a^2 + y_a^2} + \zeta \end{aligned} \tag{3.5.4}$$

3.5.3. Grid Pose Search

In the homogeneous feature case, it is possible for the single radial search method to still provide too many potential matches because features are too dense. Also, a long focal length or a high-resolution sensor would contribute to a highly dense set of features relative to the radius. Therefore it is desireable to be able to split the 6DOF search space into a grid whereby each grid region tests a particular pose using a significantly smaller search radius. The grid region with a midpoint that generates the highest number of good feature matches is then accepted to contain the correct pose. Effectively, this

acts as a correlation procedure over the 6DOF pose, while using the group-matching methods by *Jung and Lacroix* (2001). By choosing the midpoint which provides the largest group of feature matches, the algorithm can determine the region that contains the best correlation. This method also guarantees a higher coherence in the matching result as the grid spacing becomes smaller.

The grid search segments the search space of each axis which spans $[-3\sigma,\ 3\sigma]$. However, for a 6D search, where each axis is split into N segments, the size of the grid-space would be N^6. With further assumptions, the complexity of this grid-space can be reduced. Firstly, the z axis can be ignored for the same reason it was left out of the quadtree sort in section 3.5.2. Secondly, if we assume that pitch and roll errors are small enough ($< 15°$) that they do not cause significant scaling in x and y, then these two dimensions can be approximated by translations in the x and y dimensions. So the maximum translational error induced by pitch and roll error is added to $\triangle x$ and $\triangle y$, as defined in equation 3.5.5. In this case, the coefficient for roll and pitch is set to zero, $\alpha = 0$. The search space is reduced to (X, Y, Ψ) and is split into a $K \times M \times N$ grid where the midpoints of each grid region, $g\,(k,\ m,\ n)$, are defined by equation 3.5.6.

$$\Delta x = 3\sigma_X + z_{p,max} \sin \triangle\theta,\ \Delta y = 3\sigma_Y + z_{p,max} \sin \triangle\phi \tag{3.5.5}$$

$$g\,(k,\ m,\ n) = \begin{bmatrix} x_{g,k} \\ y_{g,m} \\ \psi_{g,n} \end{bmatrix} = \begin{bmatrix} \frac{\Delta X}{K}\,(2k + 1 - K) \\ \frac{\Delta Y}{M}\,(2m + 1 - M) \\ \frac{\Delta \Psi}{N}\,(2n + 1 - N) \end{bmatrix}, \quad \begin{matrix} k = 0, 1, \ldots, K-1 \\ m = 0, 1, \ldots, M-1 \\ n = 0, 1, \ldots, N-1 \end{matrix} \tag{3.5.6}$$

The cost at each midpoint is evaluated based on the number of feature matches it finds for the N_A features in set A and the descriptor distance, d_i, to the match $b \in B$. This cost function is defined by equation 3.5.7. This goal is to find the minimum value of this cost function over the grid. The descriptors of the features in a match, (a_i, b_i), are denoted by v_{a_i} and v_{b_i}. Note that the maximum distance two feature descriptors could have is 2, since the descriptors are normalized such that $\|v_{a_i}\|_2 = \|v_{b_i}\|_2 = 1$. A

nearest-nieghbor match (see section 2.5.1) is accepted if the descriptor distance is less than a threshold, κ. This threshold determines how the algorithm will perform among homogeneous or dynamic features. If selected to be low (~0.5), then descriptors that vary above the threshold will not be accepted as a match. So, κ was set to 2, allowing feature descriptors to vary dynamically and still be matched.

$$C\left(g\left(k,\, m,\, n\right)\right) = \sum_{i=1}^{N_A} \begin{cases} d_i, & d_i < \kappa \\ \kappa, & d_i \geq \kappa \text{ (or no match)} \end{cases}, \; d_i = \|v_{a_i} - v_{b_i}\|_2^2 \tag{3.5.7}$$

When calculating the cost for each grid region, a dynamic heap structure is used to minimize the number of unnecessary searches and comparisons. After each increment of i, the grid region is re-inserted to the heap, which then pushes the grid region with minimum cost to the top throughout the search. The search ends if the grid region on the top of the heap, having the minimum cost, has completed summing the cost for all $a \in A$. This method could be parallelized on a multi-core processor in the future by allocating the grid regions at the top of the heap to each core for processing.

If there is not much margin between the lowest cost grid region and the rest, this algorithm could incur many unnecessary quadtree lookups and feature comparisons. Also, if few matches are found across the whole grid, or the problem is infeasible, then this method would result in an exhaustive search. Still this grid search will guarantee that it finds the maximum group of matches possible for the two 3D sets, assuming that the pose measurement lies within the error bound.

3.6. Pose Refinement and Outlier Rejection with IRLS

Given a set of 3D point matches between two frames, the relative 6DOF pose measurement can be further refined using 3D constraints for outlier rejection. If at least 50% of the 3D matches have similar disparities (are inliers), it is statistically feasible to determine a refined relative pose. This refinement is similar to BA methods, but is robust to

outlier matches and only optimizes over 6DOF.

The chosen method for refinement and outlier rejection is IRLS (see section 2.3.4). This is expected to provide performance characteristics for real-time and be robust to outliers by using a match weighting scheme based on robust estimators such as the median (section 2.3.1) or shortest half (section 2.3.2). The custom IRLS algorithm begins with point sets $\mathbf{A} \in \mathbb{R}^{3\times N}$ and $\mathbf{B} \in \mathbb{R}^{3\times N}$, initial weights, $\mathbf{Q}_0 = \operatorname{diag} \mathbb{R}^N$, and the initial relative pose estimate, $\mathbf{M}_e \in \mathbb{R}^{4\times 3}$. The corresponding rows of $\mathbf{A}$ and $\mathbf{B}$ are 3D point matches. The initial weights are derived from the feature size and SSD of descriptor vectors. The pose estimate is comprised of a rotation matrix $\mathbf{R} \in \mathbb{R}^{3\times 3}$ and a translation $\mathbf{T} \in \mathbb{R}^3$, which comes from the grid search procedure. Alternatively, raw GPS and IMU data could be used instead.

The procedure for refining the measured pose estimate, $\mathbf{M}_e$, is shown in algorithm 3.1 and summarized as follows. In each iteration, the translation is estimated as the weighted median between the two sets of points. The Huber cost function (equation 2.3.4) is used to generate weights from the residuals. The cost of the Z residuals and X/Y residuals (steps 7e and 7f) use different thresholds because the error in Z is expected to be higher than the accuracy in X-Y. For example, the Z error might be up to 1-2 m, whereas the X-Y error may not be more than a few centimeters. After correcting for translation, the SVD decomposition technique from section 2.2.3.2 is used to estimate the rotation between the points. Note that both steps 3 and 7b use a weighted median (section 2.3.1) for outlier rejection, but the shortest half (section 2.3.2) could be used as well.

The entire matching and refinement process is shown in Figure 3.6.1.

Algorithm 3.1 IRLS for 6DOF Pose Refinement

1. Correct for the initial pose estimate: $\mathbf{B} \leftarrow \begin{bmatrix} \mathbf{B} & \mathbf{1} \end{bmatrix} \mathbf{M}_e$
2. Calculate translations: $\mathbf{T} \leftarrow \mathbf{A} - \mathbf{B}$
3. Find the weighted median of translations: $\mathbf{T}_{med} \leftarrow \operatorname{med}(\mathbf{T},\ \mathbf{Q}_0)$
4. Shift $\mathbf{B}$: $\mathbf{B} \leftarrow \mathbf{B} + \mathbf{T}_{med}$
5. Use regression estimator for rotation and correct with SVD:
   a) Generate SVD, $U\Sigma V^T = \left(\mathbf{B^T Q_0 B}\right)^{-1} \mathbf{B^T Q_0 A}$
   b) $\mathbf{R}_0 \leftarrow VU^T$
   c) Check that $\det \mathbf{R}_0 = 1$
6. $k \leftarrow 0$, $\epsilon \leftarrow 0.0001$
7. **do**
   a) Calculate translations: $\mathbf{T} \leftarrow \mathbf{A} - \mathbf{B}\mathbf{R}_k$
   b) Find the weighted median of translations: $\mathbf{T}_{med} \leftarrow \operatorname{med}(\mathbf{T},\ \mathbf{Q}_k)$
   c) Calculate the residuals: $\mathbf{E} \leftarrow \mathbf{T} - \mathbf{T}_{med}$
   d) Increment k: $k \leftarrow k + 1$
   e) Calculate the weights of the Z residuals, with a threshold of 0.35:
      i. $\vec{r} \leftarrow |\mathbf{E}_Z|$
      ii. $\hat{\sigma} \leftarrow 1.4826 \operatorname{med} \vec{r}$
      iii. $\mathbf{Q}_Z \leftarrow \operatorname{diag} \frac{\Psi\left(\vec{r}/\hat{\sigma},\ 0.35\right)}{\vec{r}/\hat{\sigma}}$
   f) Calculate the weights of the X-Y residuals, with a threshold of 0.15:
      i. $\vec{r} \leftarrow \|\mathbf{E}_{X,Y}\|_2$
      ii. $\hat{\sigma} \leftarrow 1.4826 \operatorname{med} \vec{r}$
      iii. $\mathbf{Q}_{X,Y} \leftarrow \operatorname{diag} \frac{\Psi\left(\vec{r}/\hat{\sigma},\ 0.15\right)}{\vec{r}/\hat{\sigma}}$
   g) Multiply the two weights together: $\mathbf{Q}_k \leftarrow \mathbf{Q}_Z \mathbf{Q}_{X,Y}$
   h) Use regression estimator for rotation and correct with SVD:
      i. $U\Sigma V^T = \left(\mathbf{B^T Q_k B}\right)^{-1} \mathbf{B^T Q_k A}$
      ii. $\mathbf{R}_k \leftarrow VU^T$
      iii. Check that $\det \mathbf{R}_k = 1$
8. **while** $\max |\mathbf{R}_k - \mathbf{R}_{k-1}| > \epsilon$
9. $\hat{\mathbf{M}} = \begin{bmatrix} \mathbf{R}_k \\ \mathbf{T}_k \end{bmatrix}$

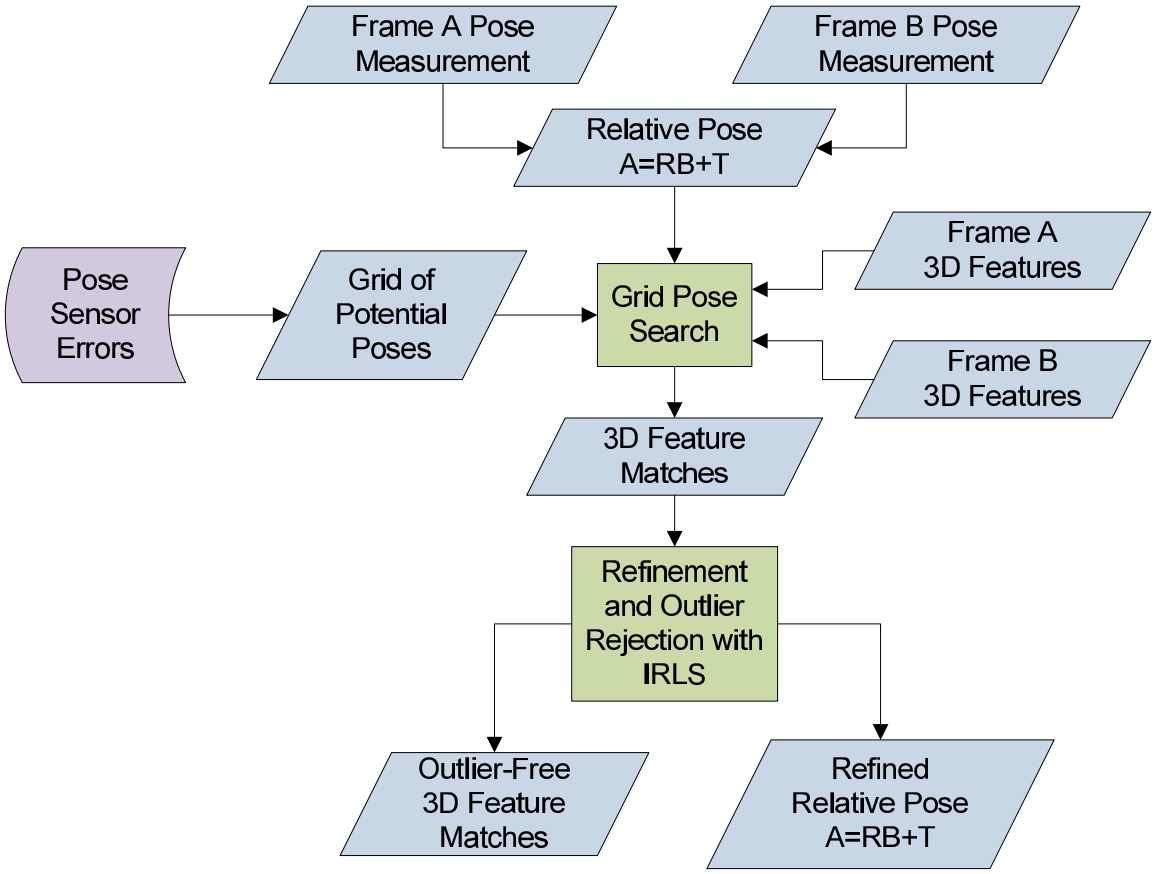


Figure 3.6.1.: 3D feature matching and refinement

4. Results

All presented stereo imagery was collected with the RMAX helicopter and stereo camera system over the terrain at Kentland Farm in Blacksburg, VA. A lawnmower path was designed to record stereo imagery with approximately 80% overlap, which was expected to provide adequate feature matches between frames. All testing was performed on a laptop with an Intel i5-430M processor running at 2.27GHz with 4GB of RAM. All software was written in C/C++ using the g++ compiler. Subroutines from OpenCV and the STL were employed where possible. No data is presented regarding image acquisition or file access time.

4.1. Feature Extraction

An example image displaying all extracted 2D features is shown in Figure 4.1.1. SURF feature extraction requires approximately 600ms on average for a single 800×600 image, given the specific results in Table 4.1. The grayscale images are first histogram-equalized to extract the highest image contrast without loss of information. Then, the images are down-sampled from 1600×1200 using a Gaussian pyramid, because extraction time is improved 4-fold by doing so. For a raw image, SURF would require over 2 seconds for extraction over the entire image. Although smaller, more precise features could be acquired by performing extraction at full resolution, this also removes any features generated from pixel noise. From the same images, the open-source Bundler package was also run over the same images, where SIFT feature extraction took approximately 7-8 seconds per image.

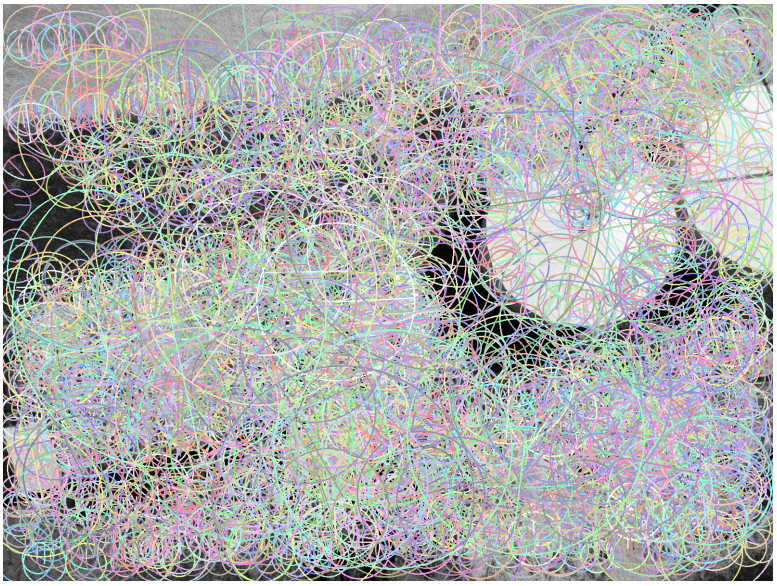
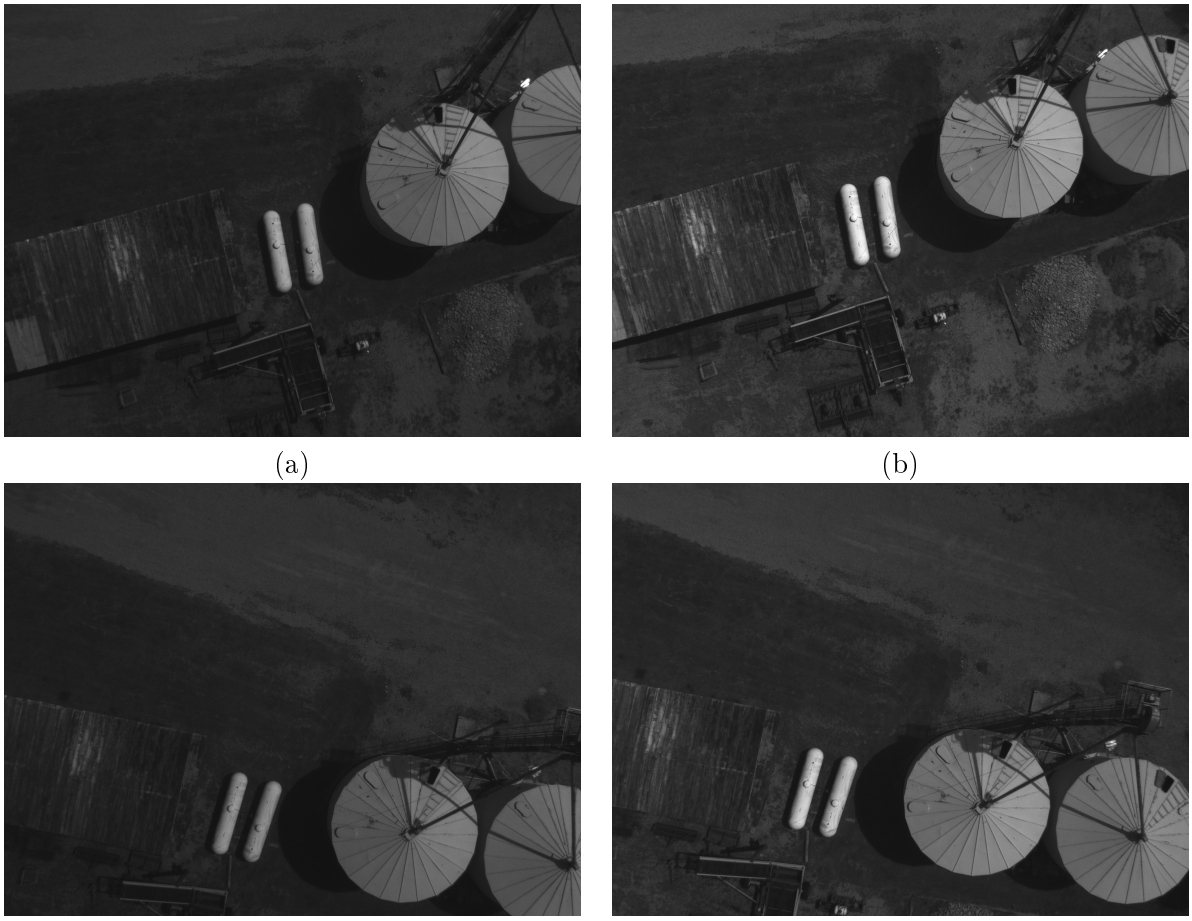


Figure 4.1.1.: Extracted SURF features (color-coded circles) overlaid on image

Table 4.1.: SURF extraction time for 800×600 (downsampled) images

Trial/Image	Time (ms)	# Features
1	552	3182
2	618	3308
3	583	3374
4	640	3454



(a) (b)

(c) (d)

Figure 4.2.1.: Two stereo image frames (a) frame 1 left image, (b) frame 1 right image, (c) frame 2 left image, (d) frame 2 right image

4.2. Two Standard Frames

Two frames were selected that appear to contain many unique objects that are expected to be easy to match. These were used for most initial algorithm testing and development. The original four images are shown in Figure 4.2.1. They contain several 3D man-made structures found on the terrain including a building, two small capped cylinders on their side, and two large white cylinders. There is also some grass, some dirt roads, and a patch of gravel. These are expected to provide adequately unique features.

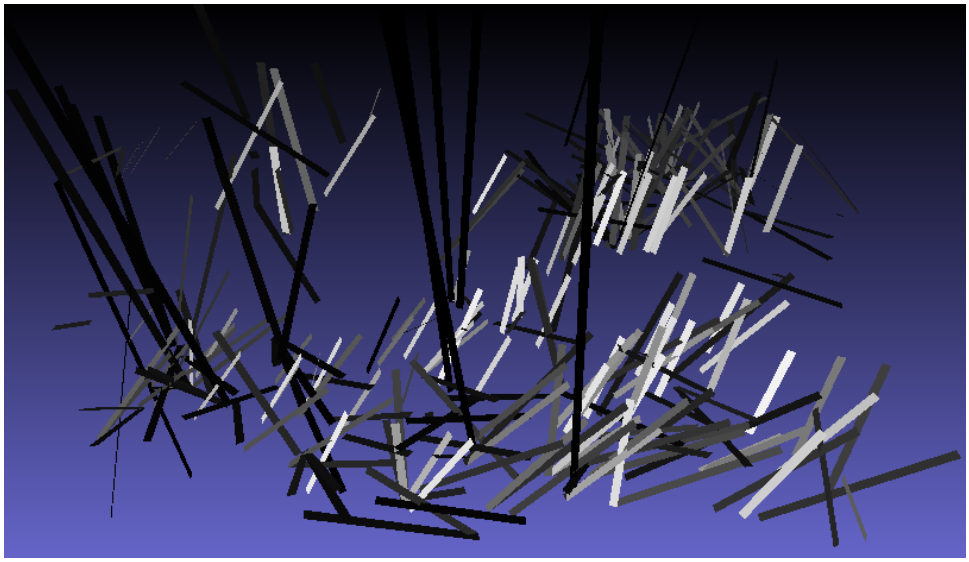


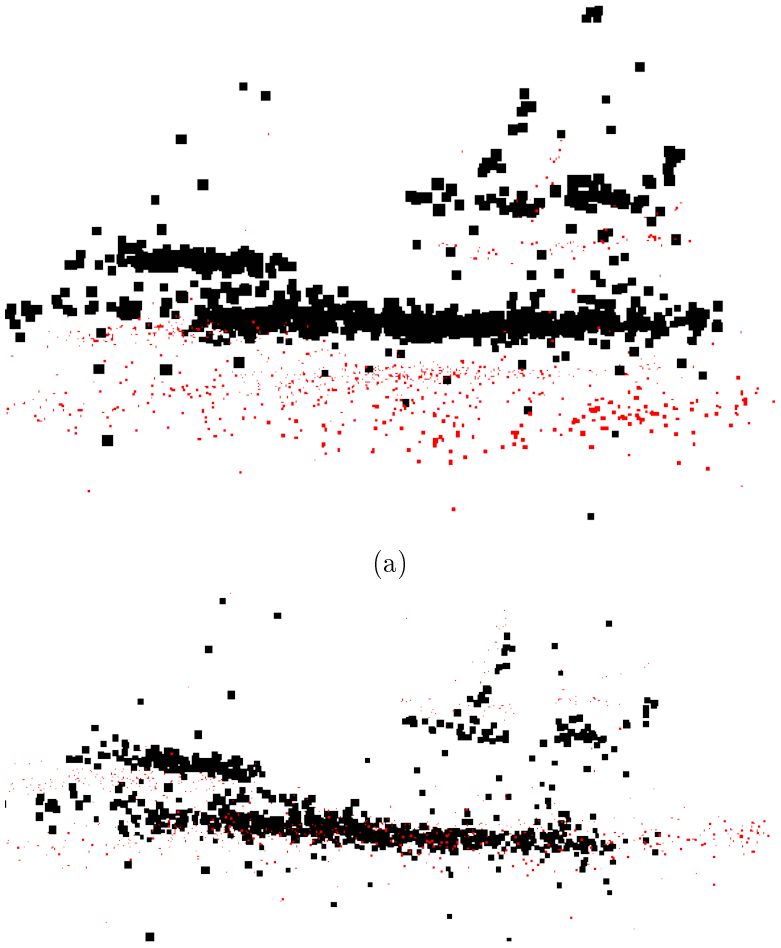
Figure 4.2.2.: Visualization of 3D feature matches with final IRLS weights are displayed as line intensity, where black lines indicate outliers and white lines indicate inliers

4.2.1. Sparse 3D Data

Feature points were first extracted from each image, matched within the stereo frame, and then backprojected to a 3D location. Then the features were geo-located with the synchronized GPS and IMU pose measurement data and plotted in Figure 4.2.3(a). The points were then matched in 3D using the method in section 3.5.2. The IRLS algorithm then corrected the result. The matches and their weights are displayed in Figure 4.2.2. Each match is represented by a line between the two points, and the line's intensity is proportional to the final weighting found with IRLS. This plot shows that the final result of IRLS is coherent since all the white lines are in consistent directions. The corrected 3D features are shown in Figure 4.2.3(b), where only a $1 \times 1 \times 1$ ($K \times M \times N$) grid was used.

4.2.2. Dense 3D Data

Dense correlation between stereo pairs was performed with the SGBM algorithm. The dense results are shown in Figure 4.2.4 using raw IMU and GPS data, and Figure 4.2.5



(a)

(b)

Figure 4.2.3.: 3D features from two stereo image frames. Red represents features from frame 1, while black shows features from frame 2. (a) points rectified only with IMU and GPS; (b) corrected results using 3D matching with $K = M = N = 1$ and IRLS.

using the corrected pose estimates. Higher grid sizes in the matching process resulted in such similar results that they are indistinguishable if presented on paper.

By manual inspection with Meshlab, the geo-located point clouds, corrected by raw GPS and IMU data, incurred a 3 meter offset, primarily in the vertical direction. However, after correction, less than 1 meter error is apparent between the two dense point clouds at the four corners of the intersection of the two clouds. This error is not in any one axis. In the X and Y axes this would be up to 50 pixels, and in the Z axis this is only one or two pixels from Figures 3.2.3 and 3.2.4. A histogram of the feature matches' sizes in meters is shown in Figure 4.2.6. The majority is less than 0.2m, which would about 10 pixels in the image at 40m distance. So, depth-mapping errors of a few pixels are expected.

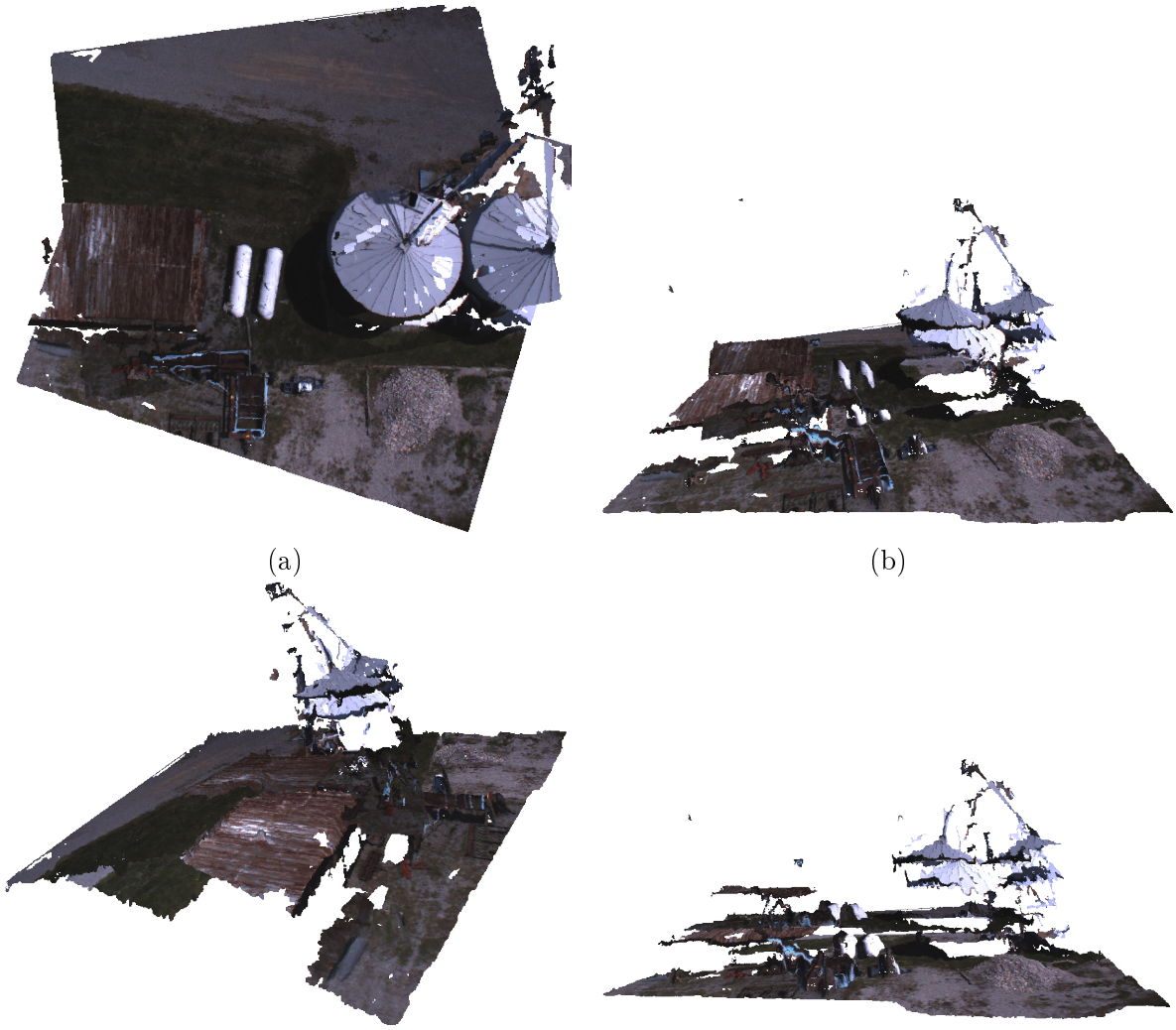
This method does not match the sub-pixel accuracy of existing BA techniques. This can partially be attributed to the fact that images are first down-sampled before feature extraction. However, IRLS refinement is much simpler and more practical for real-time alignment. Still, the feature matches found with this search method may be used as an input to a BA solver.

4.2.3. Grid Search Statistics

The grid search was run with $N = 1$ in each trial, while K and M (in the X and Y axes) were varied between 1, 3, 5, and 7. The results of which are shown in Table 4.2. As discussed above, improvement in final accuracy was negligible for larger grid sizes. Still, this table shows that matching time increases when more grid points are used. Also, it shows that once the grid becomes dense enough, the time required for IRLS no longer improves since the time was roughly the same for sizes of 3, 5 and 7.

4.3. Two Homogeneous Frames

Two more image frames were selected for their low uniqueness of texture, since they mostly contain grass. Also, when Bundler was run on them, it only found 48, 43, 89, and



(a) (b) (c) (d)

Figure 4.2.4.: Raw dense point cloud with maximum of 3 meters shift (a) nadir view, (b), (c), (d) side views showing vertical offsets.

Table 4.2.: Grid size vs. number of matches and search time for standard frame pair

Grid Size ($K \times M \times N$)	# 3D Matches	Matching Time (ms)	IRLS Time (ms)
$1 \times 1 \times 1$	276	87	11.15
$3 \times 3 \times 1$	146	260	6.46
$5 \times 5 \times 1$	117	514	7.39
$7 \times 7 \times 1$	97	869	6.87

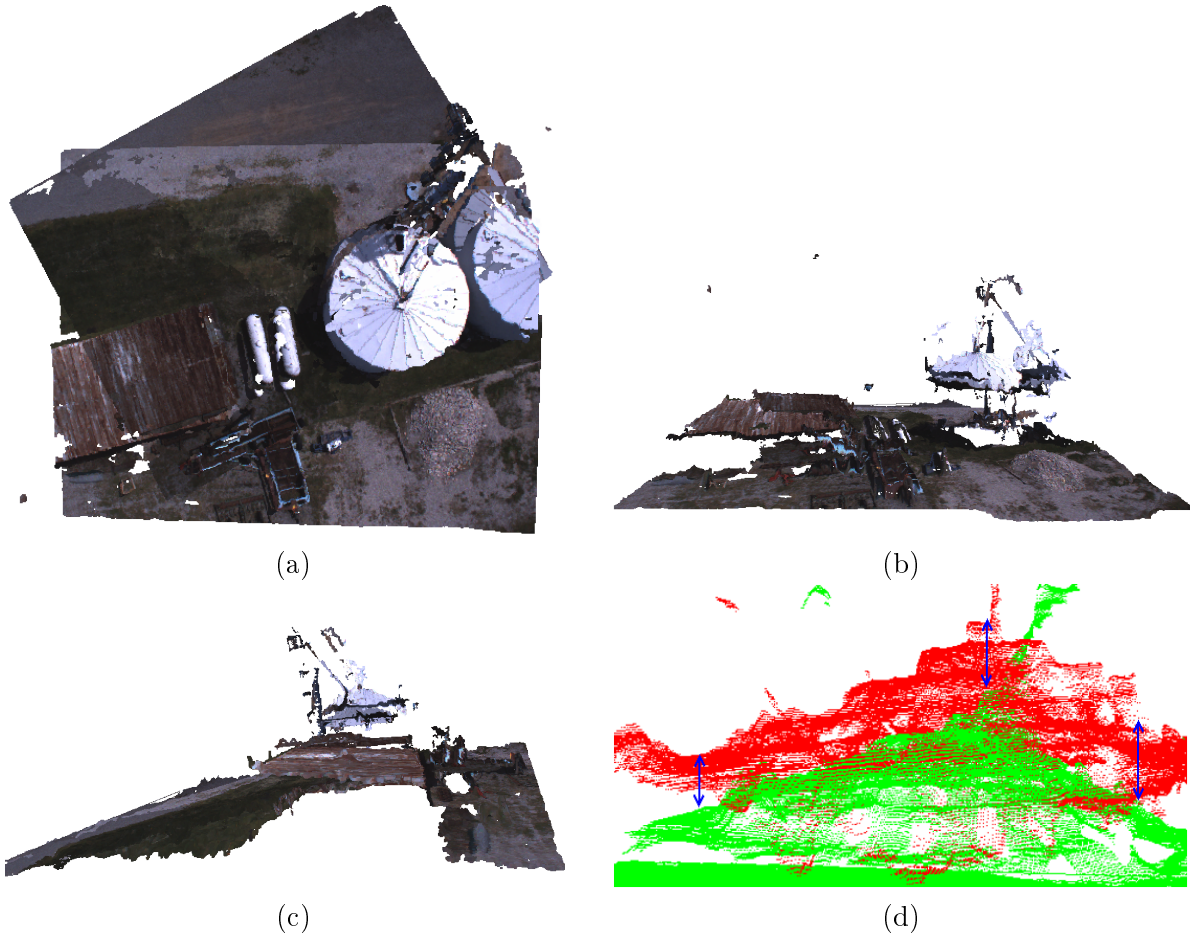


Figure 4.2.5.: Corrected dense point cloud (a) nadir view, (b) side view showing some vertical offsets, (c) second side view showing vertical gaps, (d) color-coded offsets of grain silo: red is frame 1, green is frame 2, blue indicates manually measured offsets less than 1 meter

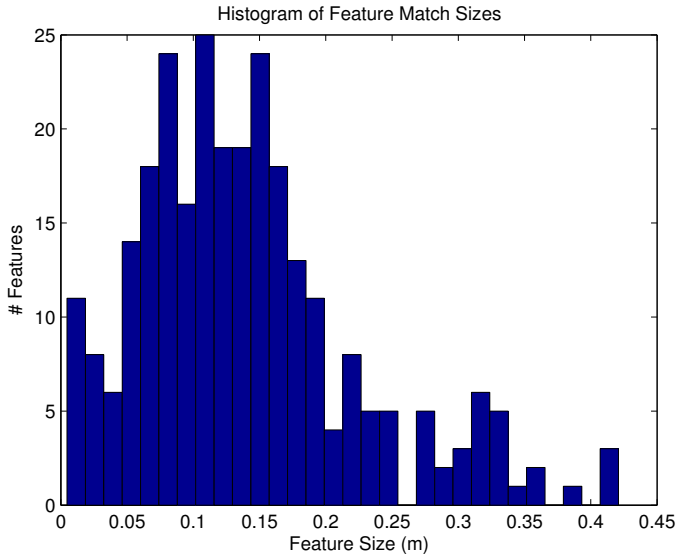


Figure 4.2.6.: Histogram of feature sizes

Table 4.3.: Grid size vs. number of matches and search time

Grid Size ($K \times M \times N$)	# 3D Matches	Matching Time (ms)	IRLS Time (ms)
$1 \times 1 \times 1$	156	43.4	29.7
$3 \times 3 \times 1$	96	131	6.95
$5 \times 5 \times 1$	79	264	5.02
$7 \times 7 \times 1$	70	505	2.85

150 features in each of the four images. Bundler was unable to match these few features successfully, so comparison of our algorithm with Bundler is infeasible. It is assumed that Bundler's SIFT extraction tool uses a static determinant of Hessian threshold, since it uses Lowe's (2004) demonstration software, which is available only in binary form. If this Hessian threshold could be reduced, more features might be extracted to avoid this problem.

These frames are, however, good examples of where our spatial sorting and searching algorithm do manage to still perform adequately well.

4.3.1. Sparse 3D Data

Considering that feature extraction requires between 1 to 1.5 seconds per stereo image pair, feature extraction appears to be more time consuming than the matching and refinement processes. With low frame-rate (0.5fps), high resolution imagery, this algorithm is considered feasible for real-time 3D feature matching applications. Dense image correlation is not accounted for in this, but if performed in parallel is expected to provide adequate real-time dense 3D data.

4.3.2. Dense 3D Data

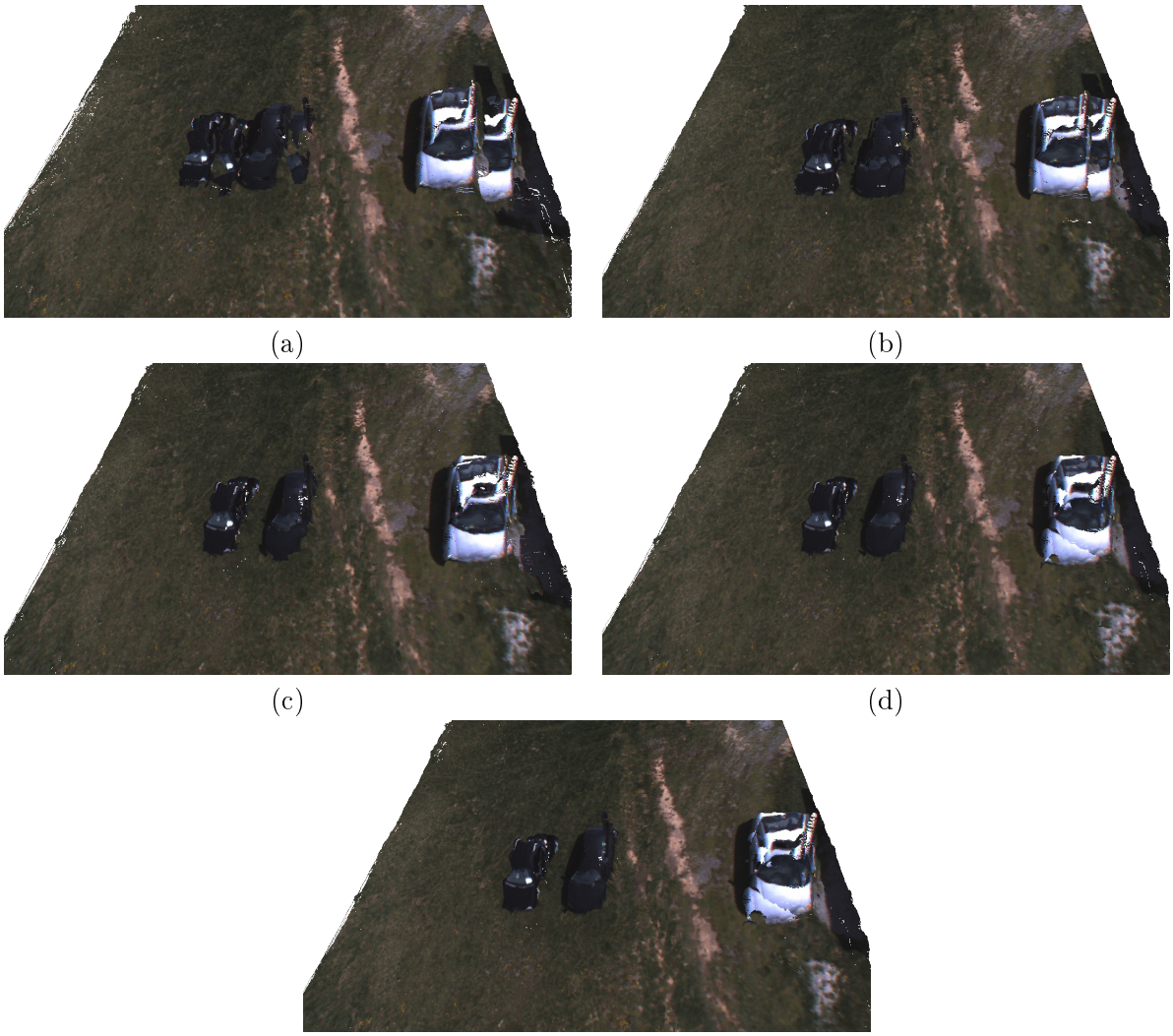
The dense point cloud results from the SGBM algorithm are shown in Figure 4.3.1. From manual inspection, the raw pose measurements impose an error of less than 1.4 meters across the overlapping data. The $1 \times 1 \times 1$ grid reduced the error to less than 0.7 meters. Still, the $3 \times 3 \times 1$, $5 \times 5 \times 1$, and $7 \times 7 \times 1$ reduced the error to below 0.4 meters. This demonstrates the improvements that the grid search provides. The smaller the grid, the fewer outliers had to be detected by the IRLS algorithm.

4.3.3. Grid Search Statistics

The grid search statistics for the two frames previously discussed are displayed in Table 4.4. For the matching process, the standard deviation was set to 1 meter in the X and Y axes, and 1.5° in yaw, based on the known accuracies of the GPS and IMU. The number of grid points in the X and Y dimensions were varied from 1 to 7. With increases of grid points, processing time increases and each grid point covers a much smaller region. So a smaller radius is used in the search and fewer potential matches are compared which causes fewer 3D matches to be found.

4.4. Data without GPS

The first data presented is a short pass comprised of 13 frames which does not contain GPS data. To visualize the input data, the SGBM results from each frame was plotted



(a) (b) (c) (d) (e)

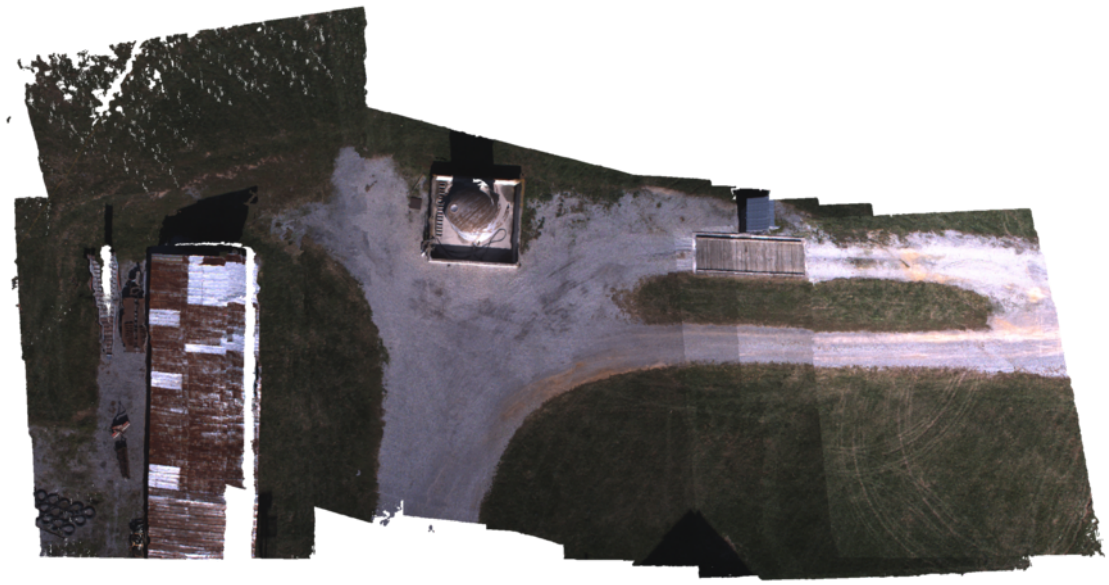
Figure 4.3.1.: Homogeneous stereo frame matching: (a) corrected with raw pose measurements, (b) corrected with grid method of $1 \times 1 \times 1$, (c) corrected with $3 \times 3 \times 1$ grid, (d) corrected with $5 \times 5 \times 1$ grid, (e) corrected with $7 \times 7 \times 1$ grid.

Table 4.4.: Grid search statistics. $\sigma_x = \sigma_y = 1$, $\sigma_\psi = 1.5°$

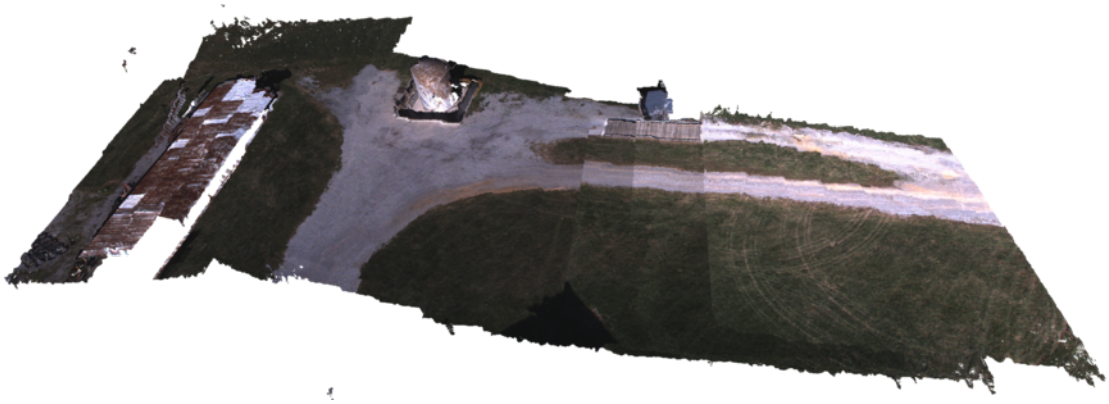
M (X)	N (Y)	K (Ψ)	# Grid Points	# 3D Matches	Time
1	1	1	1	276	519.946
3	3	1	9	146	1056.152
5	5	1	25	117	2004.889
7	7	1	49	97	3207.066

Figure 4.4.1.: All SGBM results overlayed with only IMU rotations applied

using the initial pose estimates from IMU data (Figure 4.4.1). As shown, each frame was assumed to be taken from the same position, with a standard deviation of 3 m in X and Y, and so the grid tests for errors of up to 9 m in each axis. The number of grid points in each dimension was 11, with which the matching algorithm mosaiced the frames correctly, shown in Figure 4.4.2. From this nadir view, it can be seen that the grid search and IRLS refinement correctly translate and rotate the frames relative to each other. However, upon closer inspection, it was found that some major offsets ocurred, mostly in the Z axis. Some minor misalignment does occur in X and Y. From manual inspection, the Z offsets range from 0.2 to 2 m between consecutive frames, while the X and Y errors range from 0.1 to 0.3 m. The Z offsets are best shown in Figures 4.4.3 and 4.4.4. These offsets are attributed to large features, which have more depth uncertainty.



(a)



(b)

(c)

Figure 4.4.2.: Point cloud mosaicked with grid-based method and no GPS data (a) nadir view, (b) angled view, (c) side view

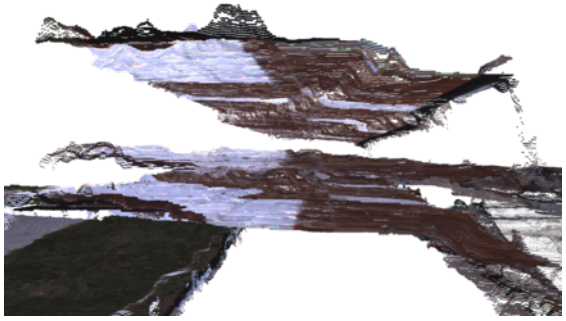


Figure 4.4.3.: Side view of roof height offsets

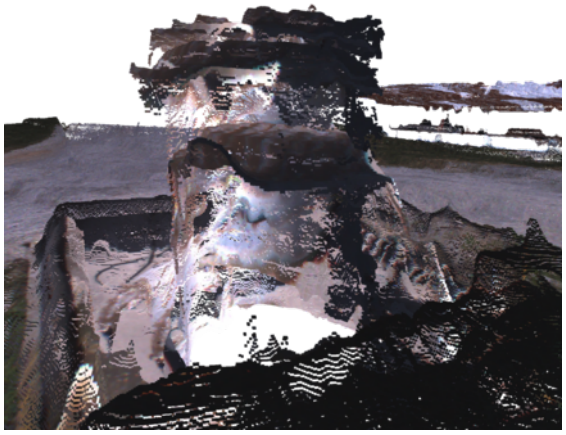


Figure 4.4.4.: Side view of tank height offsets

5. Conclusion

This paper has summarized the current technologies for image-based 3D reconstruction, presented a customized algorithm for matching and reconstruction of stereo image sequences, and demonstrated this algorithm. The relevant technologies included camera calibration techniques, the SURF feature detector/descriptor, stereo backprojection, pose sensors, grid-based search methods, various rotation matrix operations, and the Iteratively Re-weighted Least Squares algorithm. The customized algorithm requires prior calibration of the stereo cameras. It first extracts SURF features from the stereo images and matches them using stereo calibration constraints. It then backprojects feature matches within each stereo frame to 3D using the stereo calibration. Then, feature matching is carried out by imposing 3D constraints over a grid of relative camera poses. IRLS is then used to refine the rotation and translation of the stereo cameras between frames while determining outliers in the set of 3D feature matches.

The new matching algorithm experiences adequate timing requirements for use in a real-time application, and the feature extraction process is currently the most processor-intensive step. Sorting and searching spatially over only 2 dimensions proved to be competitive with standard kD-tree approaches over feature descriptors in time, accuracy, and robustness. The IRLS algorithm is relatively fast, and provides outlier detection in 3D. Its pose refinement accuracy of as low as 3 pixels does not match results of BA. However, this is expected since the images are downsampled, and camera intrinsics are not refined.

This method is dependent on the distance to the surfaces in the field of view of the stereo cameras. At far distances, accuracy of stereo backprojection decreases exponen-

tially. Since this algorithm is highly dependent on backprojection to 3D, the algorithm will breakdown at far distances relative to the stereo baseline and focal length.

5.1. Recommendations for Future Work

5.1.1. Further Analysis

The algorithm has been known to fail when matching long image sequences. However, it does succeed in matching most frames together. Frame pairs which fail the matching process should be analyzed. This may include generating more statistics from the matching process. Also, the case of two incoherent 3D feature sets should be studied. If no grid region in the grid pose search provides matches, then the maximum time for matching, which would result in the most exhaustive search, should be measured. This time is expected to vary based on distance and feature density. Also, statistics should be measured on the success of the algorithm relative to number of feature matches. From the presented data, only 70 matches generated successful refinement and outlier detection. However, if the number of 3D features matches found is very low (i.e. less than 20), then the IRLS algorithm might fail more frequently. Therefore, measuring this performance over existing and even future field data is recommended.

Testing of the accuracy of this algorithm with synthetic data is also recommended. Currently, there are no terrain databases with enough accuracy with which to compare the presented data. Therefore, generating image data sets from a synthetic enviroment is the only way to determine accuracy.

5.1.2. Improvements

Some of the frame-to-frame matching failures may be related to data synchronization. If the carrier vehicle is rotating or strafing during a frame capture, then the vertical alignment with feature points may not be adequate to correct the stereo calibration, since horizintal misalignment would occur causing depth-map errors. A software trigger was used to synchronize the two cameras, so they are effectively triggered sequentially rather

than instantaneously. Therefore, a hardware trigger on the cameras is recommended in future designs.

It is recommended to test results using feature extraction on the original images rather than down-sampled images. This is expected to increase accuracy, but the effects of pixel noise are currently unknown.

Smaller approximations of the search radius based on the pose variance could also be tested. This would speed up the algorithm by reducing the number of feature comparisons.

In the software implementation, the quadtree code used was not necessarily optimal. Using an off-the-shelf quadtree library, rather than an in-house one may provide better performance. Also, the heap from the C++ STL may not be the most efficient structure for this task, since a re-heap is performed very frequently. Evaluation of these data structure implementations should be performed.

The cost function used in the grid search could be changed to speed it up. For example, by weighting a "no match" with twice the value of the constant κ would allow the search to converge earlier. However, this would also not be able to guarantee that the grid point with the least cost is selected. Rather, this would be a more loosely estimating heuristic, and the heap structure would push an approximated minimum cost grid point to the top.

Use of a Kalman filter on the IMU and GPS data may be used to decrease pose measurement variance. The IMU data is already filtered, but its accelerometers could be used to interpolate between GPS readings more accurately.

Lastly, it is recommended to tailor the SURF algorithm to assign all features an orientation facing north. It is known that orientation alignment in both SIFT and SURF is the most processor-intensive task. This would greatly improve feature extraction speed. Furthermore, this would eliminate the case where a feature detected in two images is assigned different rotations because of the viewpoint change. Though it is unknown how common this case is.

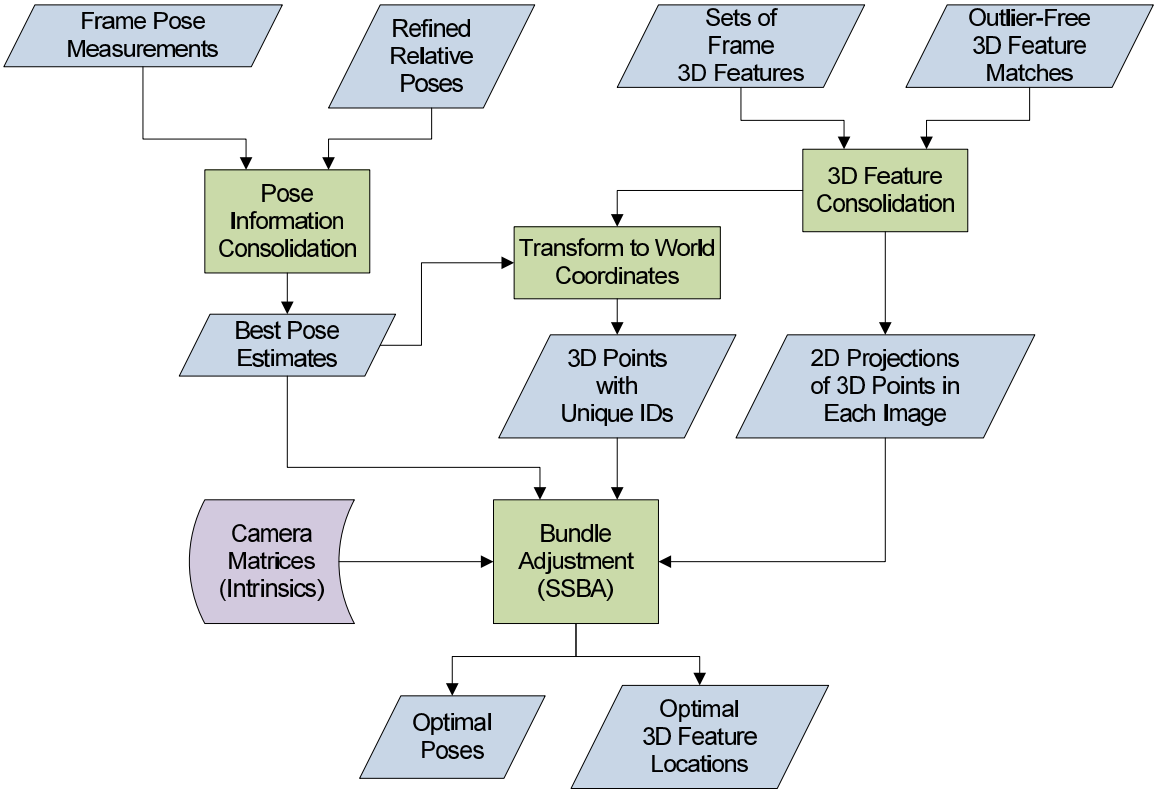


Figure 5.1.1.: Recommended bundle adjustment design

5.1.3. Bundle Adjustment Extensions

The match results from IRLS, after outlier removal, can be directly supplied to a BA solver. It is recommended to use the SSBA package (see section 2.6.3). Local BA techniques over several past image frames are expected to provide adequate results, if the matches are correct. This may or may not be necessary for every new frame. A proposed architecture is shown in Figure 5.1.1.

It is recommended to refactor the Bundler package to incorporate the adaptive Hessian SURF detector, in order to better compare the kD-tree matching algorithm in Bundler with the presented matching algorithm.

It is also recommended to integrate relative stereo pose constraints from prior calibration into a BA solver. These constraints are convex, and so imposing them on the optimization problem is possible.

5.1.4. IMU Integration

This approach requires some orientation/IMU measurement of the camera. The current state-of-the-art in IMU sensors are cabable of providing robust and accurate estimates of camera poses. The sensors have even become readily available in hobbyist OEM circuit boards for under just $200. It would be wise for camera manufacturers to begin including them directly next to an EO sensor, and provide filtered and synchronized IMU data along with each image, such as the Procerus Technologies' camera board (*Procerus*, 2011). This would enable considerable reduction in search space for any BA algorithm.

5.1.5. Real-time Viewing

The proposed algorithm is intended for real-time viewing and mapping, with a proposed architecture in Figure 5.1.2. Once the above improvements have been made to the matching and refinement algorithm, this is expected to outperform existing local BA techniques since BA is no longer required to optimize the location of every new frame. The idea is that BA would be run every N frames, optimizing the poses and camera calibration parameters, while the proposed frame-to-frame alignment procedure can be quickly run for every new frame.

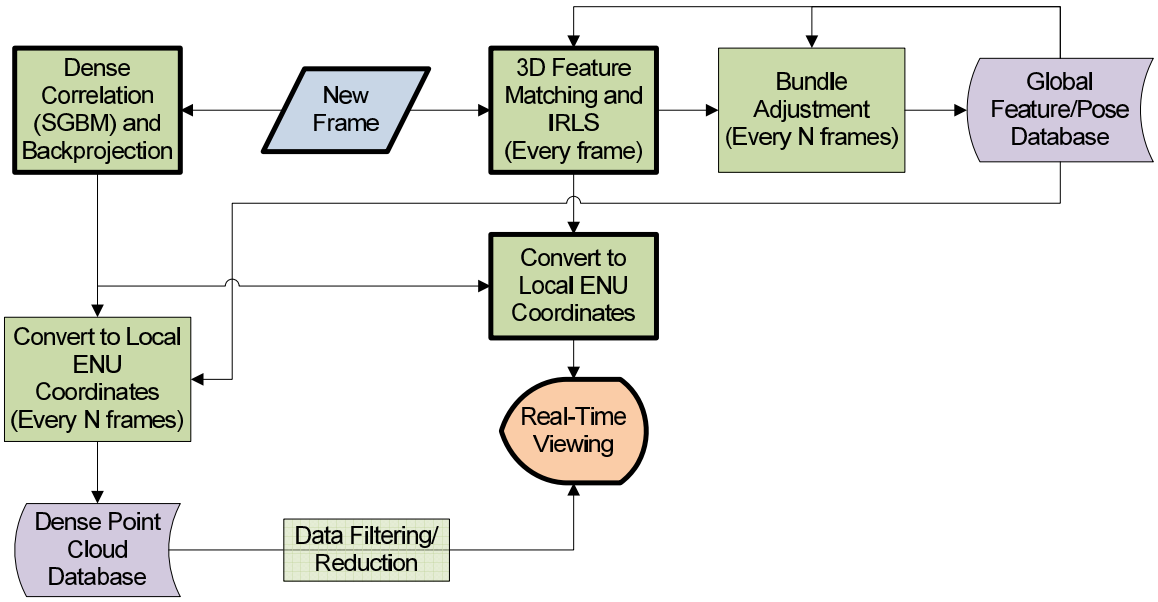


Figure 5.1.2.: Architecture for real-time mapping and viewing. Items with thick borders must run in real-time.

Bibliography

Arun, K. S., T. S. Huang, and S. D. Blostein, Least-Squares Fitting of Two 3-D Point Sets, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *PAMI-9*(5), 698–700, doi:10.1109/TPAMI.1987.4767965, 1987.

Bay, H., T. Tuytelaars, and L. V. Gool, SURF : Speeded Up Robust Features, in *European Conference on Computer Vision*, pp. 404–417, 2006.

Berghen, F. V., Levenberg-Marquardt algorithms vs Trust Region algorithms, 2004.

Bradski, G., OpenCV Wiki, 2011.

Bradski, G., and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*, 1st ed., O'Reilly Media, Sebastopol, CA, 2008.

Canny, J., A Computational Approach to Edge Detection, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *PAMI-8*(6), 679–698, doi:10.1109/TPAMI.1986.4767851, 1986.

Deriche, R., Using Canny's criteria to derive a recursively implemented optimal edge detector, *International Journal of Computer Vision*, *1*(2), 167–187, doi:10.1007/BF00123164, 1987.

Gruber, D., The Mathematics of the 3D Rotation Matrix, 2000.

Harris, C., and M. Stephens, A Combined Corner and Edge Detector, pp. 147–152, 1988.

Hartley, R., and A. Zisserman, *Multiple View Geometry in Computer Vision*, second ed., Cambridge University Press, Cambridge, UK, 2004.

Heikkila, J., and O. Silven, Calibration procedure for short focal length off-the-shelf CCD cameras, in *Proceedings of 13th International Conference on Pattern Recognition*, pp. 166–170, IEEE Comput. Soc. Press, doi:10.1109/ICPR.1996.546012, 1996.

Jung, I.-k., and S. Lacroix, A Robust Interest Points Matching Algorithm, in *8th International Conference on Computer Vision*, Vancouver, Canada, 2001.

Jung, I.-k., S. Lacroix, C. Roche, and T. C. France, High resolution terrain mapping using low attitude aerial stereo imagery, *Proceedings Ninth IEEE International Conference on Computer Vision*, pp. 946–951 vol.2, doi:10.1109/ICCV.2003.1238450, 2003.

Kass, M., A. Witkin, and D. Terzopoulos, Snakes–Active Contour Models, *International Journal of Computer Vision*, pp. 321—-331, 1988.

Klein, G., and D. Murray, Parallel Tracking and Mapping for Small AR Workspaces, *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pp. 1–10, doi:10.1109/ISMAR.2007.4538852, 2007.

Liu, Y.-c., and Q.-h. Dai, *Vision aided unmanned aerial vehicle autonomy: An overview*, 417–421 pp., IEEE, doi:10.1109/CISP.2010.5647995, 2010.

Lourakis, M. I. a., and A. a. Argyros, SBA: A Software Package for Generic Sparse Bundle Adjustment, *ACM Transactions on Mathematical Software*, *36*(1), 1–30, doi:10.1145/1486525.1486527, 2009.

Lowe, D. G., Distinctive Image Features from Scale-Invariant Keypoints, *International Journal of Computer Vision*, *60*(2), 91–110, doi:10.1023/B:VISI.0000029664.99615.94, 2004.

Mikolajczyk, K., and C. Schmid, An affine invariant interest point detector, in *Proceedings on the 7th European Conference of Computer Vision*, Copenhagen, Denmark, 2002.

Mikolajczyk, K., and C. Schmid, Scale & Affine Invariant Interest Point Detectors, *International Journal of Computer Vision*, *60*(1), 63–86, 2004.

Mili, L., Robust Estimation and Filtering: Lecture Slides (Ch. 1), 2006a.

Mili, L., Robust Estimation and Filtering: Lecture Slides (Ch. 3), 2006b.

Mili, L., Robust Estimation and Filtering: Lecture Slides (Ch. 6), 2006c.

Mordohai, P., J.-M. Frahm, A. Akbarzadeh, B. Clipp, C. Engels, D. Gallup, P. Merrell, C. Salmi, S. Sinha, B. Talton, L. Wang, Q. Yang, H. Stewenius, H. Towles, G. Welch, R. Yang, M. Pollefeys, and D. Nister, Real-time Video-based Reconstruction of Urban Environments, in *3D-ARCH'2007: 3D Virtual Reconstruction and Visualization of Complex Architectures*, Zurich, Switzerland, 2007.

Munoz, J. M. C., M. J. G. Bonilla, B. G. Miguel, J. Ramon, L. Sudupe, and M. G. Rodriguez, INTA's developments for UAS and small platforms: QUASAR, in *2009 IEEE International Geoscience and Remote Sensing Symposium*, pp. IV–999–IV–1002, IEEE, doi:10.1109/IGARSS.2009.5417548, 2009.

Natale, D. J., R. L. Tutwiler, M. S. Baran, and J. R. Durkin, Using full motion 3D Flash LIDAR video for target detection, segmentation, and tracking, *2010 IEEE Southwest Symposium on Image Analysis & Interpretation (SSIAI)*, pp. 21–24, doi: 10.1109/SSIAI.2010.5483929, 2010.

Newcombe, R. A., and A. J. Davison, Live Dense Reconstruction with a Single Moving Camera, 2010.

Ohtake, Y., A. Belyaev, and H. Seidel, A multi-scale approach to 3D scattered data interpolation with compactly supported basis functions, *Shape Modeling International*, pp. 153–161, doi:10.1109/SMI.2003.1199611, 2003.

Pilz, F., N. Pugeault, and N. Kr, *Comparison of Point and Line Features and Their Combination for Rigid Body Motion Estimation*, pp. 280—-304, Springer-Verlag, Berlin, Heidelberg, doi:10.1007/978-3-642-03061-1_14, 2009.

Procerus, OnPoint Targeting v1.3 Vision-Based, in

http://www.procerusuav.com/Downloads/DataSheets/Procerus_ OnPoint_ Targeting.pdf, Procerus Technologies, Vineyard, UT, 2011.

Ranganathan, A., The Levenberg-Marquardt Algorithm, 2004.

Short, N. J., 3-D Point Cloud Generation from Rigid and Flexible Stereo Vision Systems, Ph.D. thesis, Virginia Tech, 2009.

Snavely, N., S. M. Seitz, and R. Szeliski, Modeling the World from Internet Photo Collections, *International Journal of Computer Vision*, *80*(2), 189–210, doi:10.1007/s11263-007-0107-3, 2007.

Tomasi, C., and T. Kanade, Detection and Tracking of Point Features, *Tech. Rep. April*, Carnegie Mellon University, Pittsburgh, PA, 1991.

Tuytelaars, T., and K. Mikolajczyk, *Local Invariant Feature Detectors: A Survey*, now Publishers Inc., Hanover, MA, 2008.

Walli, K. C., D. R. Nilosek, and J. R. Schott, Airborne synthetic scene generation (aerosynth), in *ASPRS*, 2009.

Appendix

A. Stereo Pair Vertical Alignment Correction Procedure

The pseudo-code for this procedure, given stereo rectified feature point sets A and B, relative to the principal point, is as follows:

1. Let M be an empty set of matches
2. For each a in A
    a) Generate $\hat{B} \subseteq B$, such that $|a_y - b_y| \leq \triangle y_1$, $\forall\, b \in \hat{B}$
    b) Find the nearest-neighbor between a and $\hat{B}$
    c) If the nearest-neighbor exists, add the match to M
3. $\delta_{med} \leftarrow \text{med}_{(a,b)\in M}\,(a_y - b_y)$
4. Set M back to empty
5. For each a in A
    a) Generate $\hat{B} \subseteq B$, such that $|a_y - b_y - \delta_{med}| \leq \triangle y_2\, \forall\, b \in \hat{B}$
    b) Find the nearest-neighbor, b, between a and $\hat{B}$
    c) If the nearest-neighbor exists, add the match, (a, b), to M
6. $\delta_{med} \leftarrow \text{med}_{(a,b)\in M}\,(a_y - b_y)$
7. $\hat{\sigma} \leftarrow 1.4826\,\text{med}_{(a,b)\in M}\,|a_y - b_y - \delta_{med}|$
8. For each (a, b) in M

a) If $|a_y - b_y - \delta_{med}| > 2\hat{\sigma}$, remove (a, b) from M

9. $d_{med} \leftarrow \text{med}_{(a,b)\in M} \|a_{desc} - b_{desc}\|_2^2$

10. For each (a, b) in M

    a) If $\|a_{desc} - b_{desc}\|_2^2 > \alpha d_{med} + \beta$, remove (a, b) from M

After several trials with different image sets, final values for the constants in the algorithm were set at:

Name	Value
$\triangle y_1$	25
$\triangle y_2$	10
α	0
β	0.8

Source code is in Sample3D.cpp:

```
/*
 * Sample3D.cpp
 *
 *  Created on: May 18, 2010
 *      Author: kevin
 */

#include "Sample3D.h"
#include <algorithm>
#include <iostream>
#include <fstream>

using std::vector;
```

```
using std::sort;

//#define PRINT_REASONS

#include <iostream>
using namespace std;

#define RATIO   (0.7)

Sample3D::feature2Dwrap::feature2Dwrap(const feature2D* p, int ind)
{
    f = p;
    i = ind;
}
bool Sample3D::feature2Dwrap::operator>(const feature2Dwrap& p)
{
    return f->pt.y > p.f->pt.y;
}
bool Sample3D::feature2Dwrap::operator<(const feature2Dwrap& p)
{
    return f->pt.y < p.f->pt.y;
}
bool Sample3D::feature2Dwrap::operator==(const feature2Dwrap& p)
{
    return f->pt.y == p.f->pt.y;
}

Sample3D::Sample3D(const Sample2D& lft,
const Sample2D& rht,
```

```
const stereo_cal &cal,
double dy_max,
double dy_bias,
        const char* id)
{
    origin[0] = origin[1] = origin[2] = 0;
    origin[3] = 1;
    if (id)
        strcpy(ID, id);
    else
        strcpy(ID, lft.getID());
    AVL<feature2Dwrap> lftTree;

    for (unsigned int i = 0; i < lft.size(); i++)
    {
        lftTree.Insert(feature2Dwrap(&(lft[i]), i));
    }

    vector<match> matches;
    bias = dy_bias;
    getMatches(lftTree, rht, matches, dy_max, bias, cal.c1.c_xp - cal.c2.c_xp);
    matches2Features(matches, fs, cal, cal.f);
}

Sample3D::~Sample3D()
{
}

IplImage* Sample3D::getCorrespImg(IplImage* corr)
```

```
{
    CvRNG rnggstate = cvRNG(0xffffffff);
    for(unsigned int i = 0; i < fs.size(); i++ )
    {
        //generate random color
        CvScalar color = CV_RGB(cvRandInt(&rnggstate) % 255,
            cvRandInt(&rnggstate) % 255,
            cvRandInt(&rnggstate) % 255);

        //draw circle on left image
        CvPoint lcenter = cvPointFrom32f(fs[i].lftpnt);

        //draw circle on right image
        CvPoint rcenter = cvPointFrom32f(fs[i].rhtpnt);

        CvPoint rcenter2 = cvPoint(rcenter.x, rcenter.y+corr->height/2);

        cvCircle(corr, lcenter, 2, color, -1, 8, 0);
        cvCircle(corr, rcenter2, 2, color, -1, 8, 0);

        //draw line on correspondence image
        cvLine( corr, lcenter, rcenter2, color, 1);
    }
    return corr;
}

bool Sample3D::saveASCIIPLY(const char* fn)
{
    std::cout << "Saving ASCII PLY to: " << fn << std::endl;
```

```
    std::ofstream out(fn, std::ios::out);
    if (!out)
        return false;

    out << "ply\n" << "format ascii 1.0\n" << "element vertex " << fs.size()
            << "\n" << "property float x\n" << "property float y\n"
            << "property float z\n" << "end_header" << endl;
    char buff[256];
    for (unsigned int i = 0; i < fs.size(); i++)
    {
        sprintf(buff, "%.3f %.3f %.3f\n", fs[i].pt.x, fs[i].pt.y,
                fs[i].pt.z);
        out << buff;
        //out<<p.x<<", "<<p.y<<", "<<p.z<<", "<<p.r<<", "<<p.g<<", "<<p.b<<"\n";
        if (i % 100 == 0)
        {
            out.flush();
        }
        if (!out)
        {
            out.close();
            return false;
        }
    }
    out.flush();
    out.close();
    return true;
}
```

```
bool compareSSD(const feature3D &f1, const feature3D &f2)
{
    return f1.distance < f2.distance;
}

void Sample3D::sortBySSD()
{
    std::sort(fs.begin(), fs.end(), compareSSD);
}
void Sample3D::applyRandT(dMatrix3 m)
{
    applyRandT(this->fs, m);
    applyTR(origin, m);
}

void Sample3D::applyRandT(std::vector<feature3D> &ftrs, dMatrix3 m)
{
    dVector3 v1, v2;
    dMatrix3 R, R2;
    for (unsigned int i = 0; i < ftrs.size(); i++)
    {
        cvPointTodVector3(ftrs[i].pt, v1);
        applyTR(v2, m, v1); //v2 = m * v1
        dVector3TocvPoint(ftrs[i].pt, v2);

        eulerAnglesToMatrix3(R, ftrs[i].hpr);
        rm(R2, m, R);
        dMatrix3ToEulerAngles(R2, ftrs[i].hpr);
    }
```

```
}

void Sample3D::getOrigin(dVector3 ogn)
{
    memcpy(ogn, origin, sizeof(dVector3));
}

void Sample3D::matchPts(const AVL<feature2Dwrap> &lftTree,
        const Sample2D &rht,
        vector<match> &mtchs,
        double dy_max, double dy_bias,
        double scaleDiff,
        double angDiff,
        double mxsz, int cxl_cxr)
{
    dy_max = fabs(dy_max);
    int length = NUM_DESCS;
    double d, dist1 = 1e8, dist2 = 1e8;
    mtchs.clear();
    const feature2D* rhtFtr;
    const feature2D* neighbor;
    for (unsigned int i = 0; i < rht.size(); i++)
    {
        rhtFtr = &rht[i];
        if (rhtFtr->size > mxsz)
        {
#ifdef PRINT_REASONS
            cout<<"point is too big"<<endl;
#endif
```

```
            continue;
        }
        const float* vec = (const float*)rht[i].d;

        double miny = rhtFtr->pt.y + dy_bias - dy_max;
        double maxy = rhtFtr->pt.y + dy_bias + dy_max;

        feature2D dummy;
        dummy.pt.y = miny;
        feature2Dwrap p(&dummy, -1);
        AVL<feature2Dwrap>::iterator beg = lftTree.iterGT(p);
        dummy.pt.y = maxy;
        AVL<feature2Dwrap>::iterator end = lftTree.iterGT(p);

        neighbor = 0;
        dist1 = dist2 = 1e8;
        for (AVL<feature2Dwrap>::iterator it = beg; it != end; ++it)
        {
            const feature2D* lftFtr = (*it).f;
            //compare the laplacian
            if( lftFtr->laplacian != rhtFtr->laplacian )
            {
#ifdef PRINT_REASONS
                cout<<"laplacian"<<endl;
#endif
                continue;
            }

            //check the size
```

```
            if( lftFtr->size > mxsz )
            {
#ifdef PRINT_REASONS
                cout<<"point is too big"<<endl;
#endif
                continue;
            }

            //compare the x's
            if( lftFtr->pt.x - rhtFtr->pt.x  <= cxl_cxr)
            {
                continue;
            }

            //compare the angles
            if( fabs(lftFtr->dir - rhtFtr->dir) > angDiff )
            {
#ifdef PRINT_REASONS
                cout<<"angle"<<endl;
#endif
                continue;
            }

            //compare the sizes
            double avg = 0.5*(double)(lftFtr->size + rhtFtr->size);
            if( fabs(lftFtr->size - avg)/avg > scaleDiff/2 )
            {
#ifdef PRINT_REASONS
                cout<<"scale"<<endl;
```

```
#endif
                continue;
            }
            //get descriptor of the iterator point:
            const float* mvec = (const float*)lftFtr->d;

            //compare the descriptors
            d = compareSURFDescriptors(vec, mvec, dist2, length);
            if( d < dist1 )
            {
                dist2 = dist1;
                dist1 = d;
                neighbor = lftFtr;
            }
            else if ( d < dist2 )
                dist2 = d;
        }
        if ( dist1 < RATIO*dist2 )
        {
            match newMatch;
            newMatch.p1 = neighbor;
            newMatch.p2 = rhtFtr;
            newMatch.dist = dist1;
            mtchs.push_back(newMatch);
        }
        else
        {
#ifdef PRINT_REASONS
            cout<<"ratio"<<endl;
```

```
#endif
        }
    }
}

void Sample3D::getMatches(const AVL<feature2Dwrap> &lftTree,
        const Sample2D &rht,
        vector<match> &matches,
        double dy_max, double &dy_bias, int cxl_cxr)
{
    matchPts(lftTree, rht, matches,
            25, dy_bias,
            MAX_SCALE_DIFF,     //scale
            MAX_ANGLE_DIFF,     //angle
            150,                //size
            cxl_cxr);           //cxl - cxr

    //sort by dy
    sort(matches.begin(), matches.end(), dycmp);

    //get dy median
    match md = matches[matches.size()/2];
    dy_bias = md.p1->pt.y - md.p2->pt.y;

    /*********Re-match*************/
    matchPts(lftTree, rht, matches,
            dy_max, dy_bias,
            MAX_SCALE_DIFF,   //scale
            MAX_ANGLE_DIFF,     //angle
```

```
        800,                //size
        cxl_cxr);           //cxl - cxr

/***** get dy median again *****/
sort(matches.begin(), matches.end(), dycmp);
md = matches[matches.size()/2];
dy_bias = md.p1->pt.y - md.p2->pt.y;
for(unsigned int i = 0; i < matches.size(); i++)
{
    matches[i].med = dy_bias;
}

//sort by abs(dy - median)
sort(matches.begin(), matches.end(), dyAbsDiffCmp);

//get dy MAD & sigma
double mad = absdiff(matches[matches.size()/2]);
double sigma = 1.4826*mad;

//retain only elements within 2*sigma of dy
vector<match>::iterator it = matches.end()-1;
for (; it != matches.begin() &&
    absdiff(*it) > sigma*2; --it);
matches.erase(it+1, matches.end());

//sort matches by distance
sort(matches.begin(), matches.end(), distcmp);

//retain only elements with dist <= 0.8
```

```
    it = matches.end()-1;
    for (; it != matches.begin() &&
        (*it).dist > 0.8; --it);
    matches.erase(it+1, matches.end());

    //get median of descriptor SSD distance:
    double meddist = matches[matches.size()/2].dist;
    cout << "meddist=" << meddist << endl;
}

void Sample3D::matches2Features(const vector<match> &matches,
            vector<feature3D> &ftr,
            const stereo_cal &cal, double f)
{
    int n = matches.size();
    CvMat* src = cvCreateMat(1, n, CV_32FC3);
    CvMat* xyz = cvCreateMat(1, n, CV_32FC3);

    //double cxrmcxl = cal.c2.P->data.db[2] - cal.c1.P->data.db[2];

    for (int i = 0; i < n; i++)
    {
        cvSet2D(src, 0, i, cvScalar(matches[i].p1->pt.x, matches[i].p1->pt.y,
                matches[i].p2->pt.x - matches[i].p1->pt.x));// + cxrmcxl));
    }

    cvPerspectiveTransform(src, xyz, cal.Q);
    cvReleaseMat(&src);
```

```
ftr.resize(n);
CvScalar pnt;
for (int i = 0; i < n; i++)
{
    //get location
    pnt = cvGet2D(xyz, 0, i);
    //store points in camera coordinates:
    ftr[i].pt.x = pnt.val[0]/1000.;
    ftr[i].pt.y = pnt.val[1]/1000.;
    ftr[i].pt.z = pnt.val[2]/1000.;
    //ftr[i].pt.y = pnt.val[0]/1000.; //camera x is IMU y (right/east)
    //ftr[i].pt.z = pnt.val[1]/1000.; //camera y is IMU z (down)
    //ftr[i].pt.x = pnt.val[2]/1000.; //camera z is IMU x (forward/north)

    //copy hessian and laplacian
    ftr[i].hessian = matches[i].p1->hessian;
    ftr[i].laplacian = matches[i].p1->laplacian;

    //set size to the size*distance/focal length (converts to meters)
    ftr[i].size = matches[i].p1->size*fabs(ftr[i].pt.x)/f;

    //ftr[i].dir = (matches[i].p1->dir + matches[i].p2->dir)/2;
    //ftr[i].hpr[2] = matches[i].p1->dir; //feature dir corresponds to roll
    //ftr[i].hpr[1] = ftr[i].hpr[0] = 0;

    //feature direction is about z axis, so it corresponds to heading
    ftr[i].hpr[0] = matches[i].p1->dir;
    ftr[i].hpr[1] = ftr[i].hpr[2] = 0;
```

```
        //V_imu = Rot(hpr) * V_f

        memcpy(ftr[i].d, matches[i].p1->d, NUM_DESCS*sizeof(float));

        ftr[i].distance = compareSURFDescriptors(matches[i].p1->d,
                matches[i].p2->d, 1000, 64);

#ifdef RETAIN_COORDS
        ftr[i].lftpnt = matches[i].p1->pt;
        ftr[i].rhtpnt = matches[i].p2->pt;
#endif
    }
    cvReleaseMat(&xyz);
}

ostream& operator<<(ostream& strm, const Sample3D::feature2Dwrap& p)
{
    strm<<p.f->pt.y;
    return strm;
}
```

"The act of designing is more ephemeral; it is an intuitive process informed by external forces that direct the intuition. Whereas a solution can be explained, the process that created it can never adequately be understood."

-Paula Scher

"Always walk through life as if you have something new to learn and you will."

-Vernon Howard

"If at first you don't succeed, try, try again. Then quit. There's no use being a damn fool about it."

-W.C. Fields

From the GeoStarslib FAQs:

12. How do I get the sun position for any day when I only have a Julian date?
    - use **geoInitLocation()** to establish a base location
    - compute or obtain the Julian date/time
    - use **geoSunAzElJD()** to get the Azimuth and Elevation

13. How do I get a date with this Julian?
    - get a life!