A Hardware Accelerated Computer Vision Library for 3D Reconstruction Onboard Small Satellites

Caleb Adams
NASA Ames Research Center
Intelligent Systems Division
Moffett Field
Mountain View, CA
caleb.a.adams@nasa.gov

Jackson Parker
NASA Ames Research Center
Engineering Systems Division
Moffett Field
Mountain View, CA
jackson.o.parker@nasa.gov

Dr. David Cotten
Small Satellite Research Laboratory
University of Georgia
210 Field St.
Athens, GA 30602
dcotte1@uga.edu

***Abstract*—Here we present benchmarks and the expected performance of the SSRLCV (Small Satellite Research Laboratory Computer Vision) library that will be tested in Low Earth Orbit onboard the 6U MOCI (Multiview Onboard Computer Vision) cube satellite. The SSRLCV software library, used on the MOCI cubesat, is written in CUDA and C++ for Nvidia GPU/SoCs and performs structure from motion to generate 3D terrain information from a series of locally generated orbital images. Scale and Rotation invariant features are extracted from images, matched between those images, then an initial 3D point cloud is estimated with feature triangulation. Noise is removed from the point cloud and a gradient descent method, known as bundle adjustment, is used to refine the estimated camera parameters and location information of the satellite. The research simulates satellite imagery from LEO with 3D rendering software to test image data. Tests are run on the Nvidia TX2 and TX2i with timing, state, and power usage tracking. Reconstruction accuracy is measured by volumetric comparison and an Iterative Closest Point algorithm to allow for comparison to ground truth 3D models. The results show accurate 3D reconstruction of the surface of Earth feasible within 15 to 100 meters, depending on the camera system and altitude, while maintaining favorable power usage and computation time.**

TABLE OF CONTENTS

1. INTRODUCTION

Overview

The University of Georgia's (UGA) Small Satellite Research Laboratory (SSRL) is utilizing a Nvidia TX2i Graphics Processing Unit (GPU) with a specially designed interface board for edge computing with the Multi-view Onboard Computational Imager (MOCI) Satellite Mission [1]. The MOCI satellite uses a custom computer vision pipeline to generate 3D point clouds from a series of images. The MOCI pipeline is a Structure from Motion (SfM) pipeline modified for use in a Low Earth Orbit (LEO) environment [2].

978-1-5386-6854-2/19/$31.00 ©2020 IEEE

The Multiview Onboard Computational Imager

The MOCI mission will acquire imagery of the Earth's surface from LEO and perform 3D surface reconstruction at a landscape scale using custom algorithms and modified off-the-shelf, high performance computational units. The MOCI mission will also identify and map or image surface objects, including but not limited to coastal environment phenomena, while training students in STEM-related fields. Efficient data compression, feature detection, feature matching, and SfM processing techniques of space-based imagery will be performed on board the spacecraft as a proof-of-concept of high performance, on-board processing capabilities. 3D models produced by the MOCI satellite will take the form of Point Clouds and Meshes as their end product for quick data downlink.

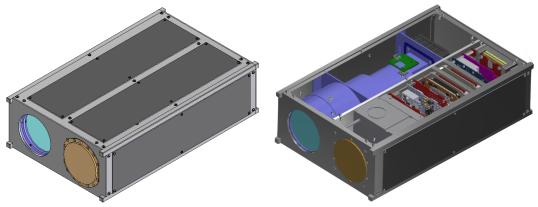


Figure 1. The 6U MOCI satellite, seen with and without the 6U cover panel.

The data collected from MOCI will be processed onboard the satellite while in LEO. This involves developing algorithms and technologies capable of detecting and matching features from multiple images, localizing features in 3D space, and computing accurate depth from point correspondence. By using high performance processors with these algorithms the amount of data and time needed to downlink completed data products will be greatly reduced.These goals align MOCI with the 2010-30 Air Force Science and Technology Horizon themes that can maximize capability superiority. Our system advances research by helping shift from Platforms to Capabilities, Control to Autonomy, Permissive to Contested domains, and Sensor to Information [3].

MOCI is a 6U cube satellite with 3U entirely devoted to an optical system. The optical tube, designed by the RUDA cardinal corporation, produces 4K imagery at a Ground Sample Distance (GSD) of approximately 6.5 meters per pixel. The optical system has a field of view of approximately ± 2.4 degrees, which means strict requirements have been placed on pointing and controls to ensure image overlap is possible for 3D reconstruction. If image overlap cannot be

guaranteed, then 3D reconstruction will not be possible. The MOCI satellite intends to perform slew maneuvers to achieve multiple views of ground targets but may also preform baseline stereo with minimal slewing.

Computer Vision Pipeline Overview

The overall goal of the MOCI's computer vision software is to take sets of 2D images and generate 3D models. One of the first steps the computer vision pipeline is the detection and description of features within the images. The pipeline then seeks to match identified points within these images. Next, some filtering removes matched points which are considered bad. After good matches have been obtained, rays are generated using camera parameters, the satellite's location, and match locations. Those rays are then "reprojected" into the real space using a triangulation. Thus, the goal with a reprojection is to take the pairs of matched points and move them from $\mathbb{R}^2$ into $\mathbb{R}^3$ so that equations of lines can be generated from the focal point of the camera into each matched point. Then, we want to find the minimum distance between those lines and choose the midpoint of that line segment at our reprojected point. Those sets of points are considered an unfiltered initial point cloud. Some additional filtering is done to this point cloud and then the bundle adjustment begins. The sets, or bundles, of lines which represent matched lines are adjusted to minimize the mismatch error of the lines. Techniques such as these are common in multiview geometry and 3D reconstruction pipelines [4][5].

2. Feature Detection, Extraction, and Matching

Two widely used concepts to begin to determine 3D information, which take in arbitrary input images and attempt to identify common features between the images, are feature extraction and feature matching [6][7]. While this is intuitive, identifying features and determining a consistent method for matching two features is computationally expensive. This is why we choose to parallelize computations on a Graphics Processing Unit (GPU) with Compute Unified Device Architecture (CUDA). In addition, knowing the camera's geometry and optical properties are vital. Solutions such as bundle adjustment [8] can optimize the knowledge of the full system; however, in regards to the application of satellite imagery, many assumptions can be made to improve the capabilities of imaging systems constrained in an orbital environment.

Effective and efficient feature matching is key to advancing on-orbit imaging capabilities and terrestrial data gathering techniques. The ultimate goal for these imaging systems is the ability to register features and decide how to decipher a solution for the objects detected [9][10]. The methods described here use a standard Scale-Invariant Feature Transform (SIFT) algorithm from Lowe's original implementation [11][12][13][14].

SIFT - the Scale Invariant Feature Transform

The SIFT algorithm can be broken up into many component stages. First, the SIFT algorithm attempts to identify extrema in scale space. These points are considered candidate points for feature description and are what make the algorithm scale invariant. Next, the algorithm achieves rotation invariance by assigning orientations to particular points from the scale space generation. Finally, the algorithm generates a feature descriptor in the form of a vector of 128 orientations, a histogram of oriented gradients. When this is performed across multiple images, nearly identical histograms are considered nearly identical points.

The Matching Problem

Subpixel dense matching is the most computationally expensive piece of the algorithm by orders of magnitude. We are given two input images I_1 and I_2 that are both n x m pixels in size. Each index pair (i, j), where $0 \leq i < n$ and $0 \leq j < m$, indicates a feature's location on the image in pixel coordinates. We denote a feature by $f_{image\ \#}(i, j)$, which is a 128-dimensional normalized feature descriptor. We then check two features for a match by calculating $||f_1(i_1, j_1) - f_2(i_2, j_2)||$. The goal is to pair each feature on the first image with the feature on the second that most closely matches its feature descriptor. The exact difference between the features need not be zero, just the closest possible match. The results do produce noise, so later filtering is required to remove features that were matched too liberally. Thus the precise goal mathematically, for every feature $f_1(i, j) \in I_1$ and $f_2(i, j) \in I_2$, is to compute:

$$\min_{(i',j')} ||f_1(i, j) - f_2(i', j')|| \tag{1}$$

The computational complexity of this process is $O(n^2m^2)$ because all features in image I_1 must be searched against all features in image I_2. This is computationally expensive, even in a GPU accelerated framework. luckly, epipolar geometry can be exploited in order to narrow this search space significantly.

Epipolar Geometry and the Fundamental Matrix

Epipolar geometry is the projective geometry between two views. When two cameras view a 3D scene from different positions, there are some geometric relations between the 3D points and their projections onto the images that define useful constraints between image points. Figure 2 depicts two cameras looking at some point X in world space.

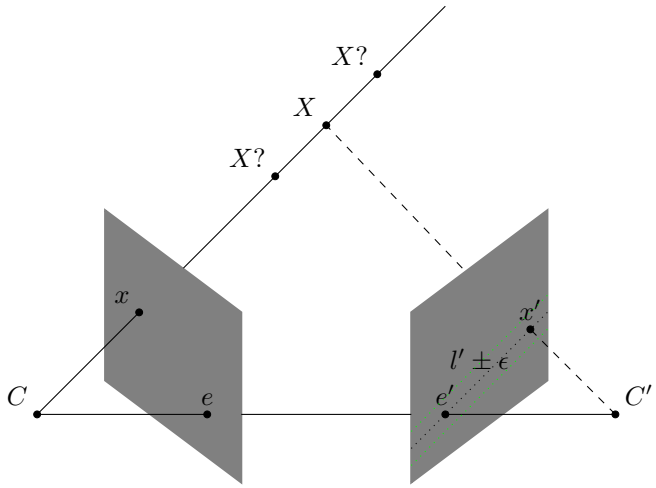


Figure 2. Epipolar geometry with epipolar line l' and constrained search path $l' \pm \epsilon$

Let C and C' be camera centers for our two views, and let x be some point on the first image. Figure 2 shows that x

alone is not enough to determine a unique location of the 3D world point X as any points on the line segment from x to X in the picture would project to the same image location on image 1; however, each one maps to a different location in image 2. If we were to project this entire line to image 2, then we would end up with a line segment which represents the valid potential locations for the corresponding matched point x, depending on the exact location of X in space. This line, referred to as the epipolar line, can be determined for every feature point. The fundamental matrix, denoted F, maps every point in image 1 to a corresponding epipolar line in image 2, and vice versa.

The fundamental matrix F is the algebraic representation of the epipolar geometry between two views. For a given point x on image 1, the matched point x' in image 2 must lie on the epipolar line $l' = Fx$. This constraint is summed up by the equation:

$$x'^T F x = 0 \tag{2}$$

which holds for all corresponding points (x, x'). In practice, this equation does not perfectly hold for all matched points, so point correspondence is not mathematically perfect. This is a combination of the accumulation of small floating point errors and errors in the initial estimation of keypoints during feature extraction. This error increases as the distance between x' and l' increases.

Restricted Search Region from Epipolar Geometry

An important issue with feature matching in general is dealing with mismatched points. That is, two points may be matched because their feature descriptors are similar even when the two features represent two entirely different points geometrically from the 3D scene. This mismatch of feature points can cause mathematical problems when using the matches later on, especially in applications such as 3D scene modeling that use a least squares algorithm and tend to be sensitive to outliers. Therefore, removing these outliers from the matched point data set is a vital post-processing step for matching algorithms.

One should recall from the previous section that outliers occur when matched features do not lie anywhere close to the epipolar lines of the corresponding feature. Since we already have this metric to determine error in matching, we can proceed to select some outlier tolerance threshold ϵ in pixels. Next, for each $f_1(i, j) \in I_i$, compute the epipolar line $l'_{i,j} = F\dot{f}_1(i, j)$. Define the set:

$$H_{i,j} \subset I_2 = \{h \in I_2 \mid d(h, l'_{i,j}) \leq \epsilon\} \tag{3}$$

where the function d is the distance in pixels between h and the epipolar line $l'_{i,j}$. We can now modify our original equation to only search in this region H:

$$\min_{(i',j') \in H_{i,j}} ||f_1(i, j) - f_2(i', j')|| \tag{4}$$

Note that our outlier tolerance ϵ represents an error bound which, if exceeded by any feature match, should be labeled a mismatch and thrown out. Since we are able to define this region geometrically, we save a huge amount of computation by only searching in this constrained region. The size of the constrained search space depends directly on our selection of ϵ, so we can compare results for different choices of the value, including $\epsilon = \infty$ (no search restriction).

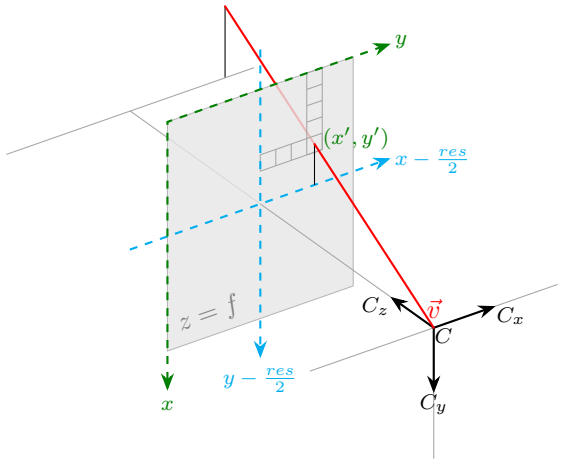


Figure 3. A visualization of line generation and reprojection from the image space to the world space

3. Triangulation and Reprojection

The goal of triangulation (also known as reprojection) is to take the pairs of matched points from the SIFT algorithm and move them from the camera's image plane in $\mathbb{R}^2$ into world coordinates in $\mathbb{R}^3$. After being translated into $\mathbb{R}^3$ equations of lines can be generated from the focal point of the camera into each matched point. Then, the minimum distance between those generated lines can be used to choose the midpoint as an estimated 3D coordinate.

Generating Equations of Lines

The generations of sets of lines, within the context of SSRL software, is known as bundle generation. This is because the sets of lines can be thought of as a bundle of matched lines. For a given 3D reconstruction there are potentially many hundreds of thousands of bundles. These bundles may contain 2 lines, but they may also contain many more lines.

The goal here is to generate a parametric equation of a line given camera position and orientation coordinates C, the camera focal length f, and the position of a coordinate in $\mathbb{R}^2$ on the image plane. We wish to generate vector v that can be used to make the parametric equation. The 2-view reprojection takes the matched points between 2 images and places them into $\mathbb{R}^3$. To place each set of points into $\mathbb{R}^3$, some trigonometry and and matrix transformations need to take place. The first step to moving a keypoint into $\mathbb{R}^3$ is to place it onto a plane in $\mathbb{R}^2$. the coordinates (x', y') in $\mathbb{R}^2$ require the size of a pixel $dpix$, the location of the keypoint (x, y), and the resolution of the image $(xres, yres)$ to yield:

$$x' = dpix\left(x - \frac{xres}{2}\right) \qquad y' = dpix\left(y - \frac{yres}{2}\right) \tag{5}$$

This is repeated for the other matching keypoint. The coordinate (x', y', z') in $\mathbb{R}^3$ of the keypoint (x', y') in $\mathbb{R}^2$ is given by three rotation matrices and one translation matrix. First we treat (x', y') in $\mathbb{R}^2$ as a homogenous vector in $\mathbb{R}^3$ to yield $(x', y', 1)$. Given a unit vector representing the camera orientation (r_x, r_y, r_z), in our case the spacecraft camera, we find the angle to rotate in each axis $(\theta_x, \theta_y, \theta_z)$. In our simple case we find the angle in the xy plane with:

$$\theta_z = \cos^{-1} \frac{\left(\begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} r_x & r_y & r_z \end{bmatrix} \right)}{\sqrt{\begin{bmatrix} r_x & r_y & r_z \end{bmatrix} \cdot \begin{bmatrix} r_x & r_y & r_z \end{bmatrix}}} \tag{6}$$

It is important to note that the software treats all planes in an identical way, rotating in several axis. Now, given a rotation in each plane $(\theta_x, \theta_y, \theta_z)$ (lets call this rotation matrix R_θ), we calculate the homogeneous coordinate $(r_x, r_y, r_z, 1)$ in $\mathbb{R}^4$ using linear transformations. The values (T_x, T_y, T_z) represent a translation in $\mathbb{R}^3$ and use camera position coordinates (C_x, C_y, C_z), the camera unit vectors representing orientation (u_x, u_y, u_z), and focal length f. Let the rotation matrix R in equation (7) represent a 3x3 rotation matrix generated from multiplying component rotation matrices of each axis.

$$R_\theta \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x_r \\ y_r \\ z_r \\ 1 \end{bmatrix} \tag{7}$$

$$\begin{bmatrix} C_x - (x_r + f \cdot u_x) \\ C_y - (y_r + f \cdot u_y) \\ C_z - (z_r + f \cdot u_z) \\ 1 \end{bmatrix} = \begin{bmatrix} T_x \\ T_y \\ T_z \\ 1 \end{bmatrix} \tag{8}$$

$$\begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r \\ y_r \\ z_r \\ 1 \end{bmatrix} = \begin{bmatrix} x'_r \\ y'_r \\ z'_r \\ 1 \end{bmatrix} \tag{9}$$

The above process should happen for all points that have been matched. This will result in 2 homogeneous points that we will call $\begin{bmatrix} x_0 & y_0 & z_0 & 1 \end{bmatrix}^T$ and $\begin{bmatrix} x_1 & y_1 & z_1 & 1 \end{bmatrix}^T$. Each point has a corresponding camera vector, that is already known thanks to the camera coordinates, $\begin{bmatrix} C_{x0} & C_{y0} & C_{z0} & 1 \end{bmatrix}^T$ and $\begin{bmatrix} C_{x1} & C_{y1} & C_{z1} & 1 \end{bmatrix}^T$. From this we can make parametric lines L_0 and L_1 with the parametric variables t_0 and t_1:

$$\begin{aligned} L_0 &= \begin{bmatrix} x_0 - C_{x0} \\ y_0 - C_{y0} \\ z_0 - C_{z0} \end{bmatrix} \begin{bmatrix} t_0 \\ t_0 \\ t_0 \end{bmatrix} + \begin{bmatrix} C_{x0} \\ C_{y0} \\ C_{z0} \end{bmatrix} \\ L_1 &= \begin{bmatrix} x_1 - C_{x1} \\ y_1 - C_{y1} \\ z_1 - C_{z1} \end{bmatrix} \begin{bmatrix} t_1 \\ t_1 \\ t_1 \end{bmatrix} + \begin{bmatrix} C_{x1} \\ C_{y1} \\ C_{z1} \end{bmatrix} \end{aligned} \tag{10}$$

Minimum Distance Between Skew Lines—Now that we have lines L_0 and L_1, the challenge is to find the points s_0 and s_1 of closest approach. First, we must test the assumption that our lines are skew, meaning they are not parallel and do not intersect. To frame this, we take the forms of L_0 and L_1 and simplify them by thinking of them as parametric vectors where C_0 and C_1 represent the camera position vectors v_0 and v_1 represent the vector was previously calculated from the subtraction of match coordinates with the camera vector. We make the simple equations:

$$L_0 = v_0 t_0 + C_0 \qquad L_1 = v_1 t_1 + C_1 \tag{11}$$

To make sure that the lines are not parallel, which is unlikely, we must verify that their cross product is not zero. If $v_0 \times v_1 = 0$, then we have a degenerate case with infinitely many solutions. As long as we know this is not the case we can proceed. We know that the cross product of the two vectors $c = v_0 \times v_1$ is perpendicular to the lines L_0 and L_1. We know that the plane P, formed by the translation of L_1 along c, contains C_1. We also know that the point C_1 is perpendicular to the vector $n_0 = v_1 \times (v_0 \times v_1)$. Thus, the intersection of L_0 with P is also the point, s_0, that is nearest to L_1, given by the equation:

$$s_0 = C_0 + \frac{(C_1 - C_0) \cdot n_0}{v_0 \cdot n_0} \cdot v_0 \tag{12}$$

This also holds for the second line L_1, the point s_1, and vector $n_1 = v_0 \times (v_1 \times v_0)$ with the equation:

$$s_1 = C_1 + \frac{(C_0 - C_1) \cdot n_1}{v_1 \cdot n_1} \cdot v_1 \tag{13}$$

Now, given two points that represent the closest points of approach, we simply find the midpoint m:

$$m = \begin{bmatrix} (s_0[x] + s_1[x])/2 \\ (s_0[y] + s_1[y])/2 \\ (s_0[z] + s_1[z])/2 \end{bmatrix} \tag{14}$$

N-View Reprojection

At this stage we are exclusively in $\mathbb{R}^3$. For the purpose of our software, we expect points in the form $P = (p_x, p_y, p_z)$ and their corresponding orientation as a unit vector $\hat{U} = (\hat{u}_x, \hat{u}_y, \hat{u}_z)$ we expect these together in a tuple $((p_x, p_y, p_z), (\hat{u}_x, \hat{u}_y, \hat{u}_z))_n$. This tuple comes with several (n many) tuples which all uniquely correspond with a matched set. So we have a $\mathbb{R}^3$ match set $M = \{(P, \hat{U})_0, (P, \hat{U})_1, ..., (P, \hat{U})_n\}$ where n is the number of tuple pairs and has a one-to-one correspondence with the number of views in which the $\mathbb{R}^2$ match was found. We also generate a set of $\mathbb{R}^3$ matches, M_i, resulting in a set which has a one-to-one correspondence with the total number of points we should have after reprojection.

In figure 4, assume that point C is the correct real world point and has no orientation. The goal is to make a best guess at the value of C given our imperfect information.

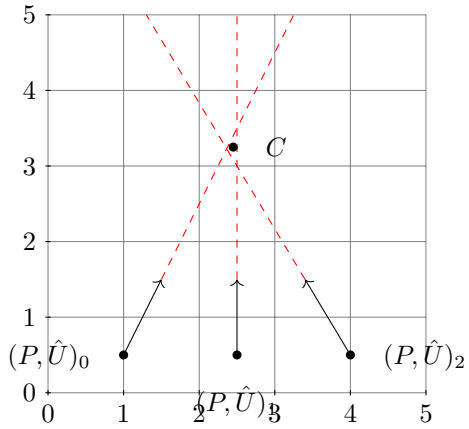


Figure 4. Nview triangulation / reprojection example within a single plane

3 by 3 Inversion Method—A method which finds a "mid-point" for n many views with minimal computational cost is described as follows: At this point *we do not actually know the coordinates* of the real point C, but we will derive how to find it. First, consider the identity:

$$(m \times n) \cdot (m \times n) = ||m||^2||n||^2 - (m \cdot n)^2 \tag{15}$$

Then, note we have the distance function, measure how much a given $((p_x, p_y, p_z), (\hat{u}_x, \hat{u}_y, \hat{u}_z))_n$ tuple (which represents a line) misses the real world target point C. Note this function is calculated for each tuple.

$$D_n = \frac{||(C - P_n) \times \hat{U}_n||}{||\hat{U}_n||} \tag{16}$$

We will want to use the square of the distance (as is common in many optimization problems) to insure convex optimization and positive distance values. We also use the identity mentioned above to get our primary distance equation. Taking the first derivative of the distance function will give us a local minimum value by finding a 0 solution.

$$\begin{aligned}
D_n &= \frac{||(C - P_n) \times \hat{U}_n||}{||\hat{U}_n||} \\
D_n^2 &= \Big(\frac{||(C - P_n) \times \hat{U}_n||}{||\hat{U}_n||}\Big)^2 \\
D_n^2 &= \frac{||(C - P_n) \times \hat{U}_n||^2}{||\hat{U}_n||^2} \\
D_n^2 &= \frac{||C - P_n||^2||\hat{U}_n||^2 - ||(C - P_n) \cdot \hat{U}_n||^2}{||\hat{U}_n||^2} \\
D_n^2 &= ||C - P_n||^2 - \frac{||(C - P_n) \cdot \hat{U}_n||^2}{||\hat{U}_n||^2} \\
\frac{dD_n^2}{dC} &= 2(C - P_n) - 2\hat{U}_n \frac{(C - P_n) \cdot \hat{U}_n}{||\hat{U}_n||^2}
\end{aligned} \tag{17}$$

We need to find a zero for the following (note that we are dealing with a vector in $\mathbb{R}^3$, so $0 = [0, 0, 0]^T$). The value m is the total number of $\mathbb{R}^3$ match points:

$$\begin{aligned}
0 &= \sum_{n=0}^{m} C - P_n - \hat{U}_n \frac{(C - P_n) \cdot \hat{U}_n}{||\hat{U}_n||^2} \\
0 &= \sum_{n=0}^{m} C - P_n - \frac{\hat{U}_n(C \cdot \hat{U}_n)}{||\hat{U}_n||^2} + \frac{\hat{U}_n(P_n \cdot \hat{U}_n)}{||\hat{U}_n||^2} \\
0 &= \sum_{n=0}^{m} C - P_n - \frac{\hat{U}_n\hat{U}_n^T C}{||\hat{U}_n||^2} + \frac{\hat{U}_n\hat{U}_n^T P_n}{||\hat{U}_n||^2} \\
0 &= \sum_{n=0}^{m} \Big(I - \frac{\hat{U}_n\hat{U}_n^T}{||\hat{U}_n||^2}\Big)C - \Big(P_n - \frac{\hat{U}_n\hat{U}_n^T P_n}{||\hat{U}_n||^2}\Big)
\end{aligned} \tag{18}$$

This is of the form $Ax = b$ because we now have $0 = Ax - b$. Thus, we can remove the summations and get a system that results in taking an inverse of a 3 by 3 matrix.

Notice the possible expansion:

$$\begin{aligned}
0 &= \sum_{n=0}^{m} \Big(I - \frac{\hat{U}_n\hat{U}_n^T}{||\hat{U}_n||^2}\Big)C - \Big(P_n - \frac{\hat{U}_n\hat{U}_n^T P_n}{||\hat{U}_n||^2}\Big) \\
0 &= \left(\Big(I - \frac{\hat{U}_0\hat{U}_0^T}{||\hat{U}_0||^2}\Big) + \Big(I - \frac{\hat{U}_1\hat{U}_1^T}{||\hat{U}_1||^2}\Big) + \Big(I - \frac{\hat{U}_2\hat{U}_2^T}{||\hat{U}_2||^2}\Big) + ...\right)C \\
&\quad - \left(\Big(P_0 - \frac{\hat{U}_0\hat{U}_0^T P_0}{||\hat{U}_0||^2}\Big) + \Big(P_1 - \frac{\hat{U}_1\hat{U}_1^T P_1}{||\hat{U}_1||^2}\Big) + ...\right) \\
0 &= (A)C - (b) \\
AC &= b \\
C &= A^{-1}b
\end{aligned} \tag{19}$$

So, the meat of this method is to calculate the A matrix's inverse and multiply it by vector b to find the estimated point C. Succinctly, these are calculated:

$$A = \sum_{n=0}^{m} \left(I - \frac{\hat{U}_n \hat{U}_n^T}{||\hat{U}_n||^2} \right) \tag{20}$$

$$b = \sum_{n=0}^{m} \left(P_n - \frac{\hat{U}_n \hat{U}_n^T P_n}{||\hat{U}_n||^2} \right) \tag{21}$$

Then, because this method takes the inverse of a 3x3 matrix, we can easily write (hardcode) a constant time inversion method. This makes the method an ideal N-view triangulation method. Additionally consider that this is run on GPU, so the point estimations can occur in parallel for each matched set of points. Similar benefits could be realized running the algorithm on a multithreaded system.

4. BUNDLE ADJUSTMENT

Bundle adjustment seeks to minimize the error in the point estimation methods mentioned above. It does this by iteratively adjusting the camera parameters, thus changing the match sets of lines, or "bundles", such that the total error of the system decreases to a local minimum. Usually it is necessary to consider that the system will converge to a local minimum only and not the global minimum. However, this concern is not necessary for this computer vision software as camera parameters are already relatively accurate.

Point Estimation Error

When points are estimated with the 2-view case, the minimal distance between skew lines is calculated by taking the Euclidean distance between the points. For that case the total error of the point cloud is simple to calculate and is just a summation of every point's individual error, or the average of that summation. The N-view case is similar; it can either be calculated with the Euclidean distance between projected estimated points onto the camera plane and their original match point or by calculating the average. These methods are an analog for the commonly used reprojection error, which is not directly calculated in this case. Thus there are four possible error functions, though only the last two are used practically:

1. **Linear Error:** Only calculated in the 2-view case, linear error is the shortest distance between matched lines.
2. **Average Linear Error:** Calculated in the N-view case, average linear error is the average shortest distance from the estimated point to its corresponding line.
3. **Squared Linear Error:** This is the practical 2-view error measurement, used so that the error function can be used in convex optimization.
4. **Squared Average Linear Error:** This is the practical N-view error measurement, used so that the error function can be used in convex optimization.

Their resultant functions of the practical error measurements are analyzed further in the following sections.

Noise Removal

The distribution of error (calculator of error is above) in 3D reconstructions is typically considered to be Gaussian. Thus, statistical filtering methods can be used to remove outlier error points. The methods used here start by calculating a sample variance, σ^2 , from a random set of points. Let a given sample point be s_i and the average of the total error be $\bar{s}$, then the sample variance can be calculated with equation (22)

$$\sigma^2 = \frac{1}{n} \sum_{i=0}^{n} \left(s_i - \bar{s} \right)^2 \tag{22}$$

Then, all points with some error outside of some $n \cdot \sigma$, where $n \in \mathbb{Z}^+$, can be discarded. In addition to the error function, the resultant point cloud can also be filtered on distance to k nearest neighbors. The nearest neighbors removal has the effect of removing bad points from regions of relatively low density and can still utilize the methods mentioned above.

Formulation as Gradient Descent

It is possible to formulate the bundle adjustment as an optimization to minimize one of the error functions listed above. Though many algorithms exist to solve this, Newtonian gradient descent is implemented here because of its simplicity. The most commonly used algorithm in bundle adjustment is the Levenberg-Marquardt (LM) algorithm [8], which differs from the standard Newtonian approach with its use of the second derivative, in this context the Hessian matrix, and contains a dampening parameter to slow the descent around a local minimum. It is certainly possible to implement the LM algorithm within the context of MOCI's bundle adjustment, this should be considered future work for SSRL lab members or graduate students. However, given that camera parameters are expected to be relatively accurate to start, the use of a second order Newtonian method with simple dampening might be considered sufficient for this application. If camera parameters are not known, but are instead estimated by some other computer vision procedure, then a more robust optimization algorithm is likely needed. Additionally, noise removal should not occur during the gradient descent as this can cause false minima and rapidly deteriorate into gratuitous point removal until no points are left.

First, consider the standard Newtonian gradient descent and the parameters we are using for this descent in equation (23). The position and orientation of the satellite are much more uncertain than the focal length and field of view of the imager. Thus, intrinsic camera parameters are considered, at least for now, to be close enough to their optimal configurations that they should not be modified. Instead, extrinsic camera parameters (position and orientation) are to be modified when searching for a minimum error. Let the chosen error function be $F(C_n)$ where C_n represents a vector of all input camera parameters at iteration n.

$$C_{n+1} = C_n - \alpha_n H^{\dagger} F(C_n) \nabla F(C_n) \tag{23}$$

∇ is the gradient for the error function F with respect to all camera parameters in the camera vector C. ∇ is a vector of partial derivatives that are calculated via a finite central difference. A given element of ∇, say $\frac{\partial F}{\partial C_n[i]}$, where $C_n[i]$ is the ith element in the vector C_n, is calculated with a set size h along a vector e which is all 0's other than a single 1 at the ith index. This has the effect of only stepping a distance of h along the ith element of C_n, thus estimating the partial derivative, as seen in (24).

$$\frac{\partial F}{\partial C_n[i]} = \frac{F(C_n + he) - F(C_n - he)}{2h}$$

$$\frac{\partial^2 F}{\partial C_n[i]^2} = \frac{\begin{array}{c} -F(C_n + 2he) \\ + 16F(C_n + he) - 30F(C_n) \\ + 16F(C_n - he) - F(C_n - 2he) \end{array}}{12h^2}$$

$$\frac{\partial^2 F}{\partial C_n[i]\partial C_n[j]} = \frac{\begin{array}{c} F(C_n + h_i e_i + h_j e_i) \\ - F(C_n + h_i e_i - h_j e_i) \\ -F(C_n - h_i e_i + h_j e_i) \\ + F(C_n - h_i e_i - h_j e_i) \end{array}}{4h_i h_j} \tag{24}$$

Additionally, the Hessian is calculated via a finite central difference seen above in equation (24), which is defined similarly to the gradient only with more than one change to consider (a change in different parameters $C_n[i]$ and $C_n[j]$). The Moore–Penrose pseudoinverse of the Hessian $H^\dagger$ is then calculated to produce a stepsize adjustment for each step of the gradient descent. The pseudoinverse is calculated, rather than the direct inverse, because the Hessian may not always be invertible. The process for calculating the inverse of the Hessian involves calculating a singular value decomposition (SVD), where $A = U\Sigma V^T$ for a given matrix A. To calculate the pseudoinverse $A^\dagger$, matrices U and V^T are transposed and the inverse Σ^{-1} is obtained by taking the reciprocal of each non-zero element within Σ. The equations cab be seen in (25).

$$\begin{aligned} A^\dagger &= (A^T A)^{-1} A^T \approx A^{-1} \\ A &= U\Sigma V^T \\ A^\dagger &= V\Sigma^{-1} U^T \end{aligned} \tag{25}$$

At each iteration of the descent, the dampening variable α, seen in the initial equation (23) is decreased by a ration of the previous error e_{n-1} and the currently computed error e_n such that $\alpha_n = \frac{e_{n-1}}{e_n}$. Here the assumption is that $e_{n-1} > e_n$; when this is no longer the case, the algorithm has found a local minima and exits.

Most bundle adjustment algorithms analyze how camera parameters are defined to help define derivatives. The camera models defined here special properties that are derived in other research the UGA SSRL has published [2].

5. Initial Results

SSRLCV Simulations with Blender

Initial tests relied on manually importing 3D meshes into Blender. These models were generated from the Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER) Global Digital Elevation Model version 3 (GDEMv3). ASTER GDEMv3 is released as a GeoTIFF, which is a Tagged Image File Format (TIFF) raster encoding for images with extra georeferencing information (defined by the Open Geospatial Consortium). At the time of writing, current versions of Blender poorly supported GeoTIFF importing. As a result, some simple scripts were written to convert the image into a 3D model. Thus, the height information was converted to a mesh encoded as a PLY filetype. Later, using the georeferencing information encoded in the GeoTIFF, an image from Google Earth Engine was applied to texture the mesh in Blender.

The resulting ASTER meshes are between 15 and 90 meters in resolution, which means a given polygon face is a square $15m^2$ to a square $90m^2$. The resolution used for the manual ASTER to Blender testing is 30 meters. The rationale here was that, despite the models only being accurate within 30 meters, because the PLY format linearly interpolates the height point locations in the raster it is possible to test for sub-30 meter accuracy. This means it is possible to generate a higher resolution model in the areas between raster height points. Additionally, the ASTER GeoTIFF and its resultant mesh can be viewed as a ground truth because the primary goal is to calculate height on a given surface. Different reconstruction methods can be compared by testing these same data sets with different methods.

In addition to testing ASTER models in Blender, other publicly available models can also be tested with the aid of the BlenderGIS addon. The addon is superb for isolating given regions of Earth and generating high quality textured meshes. The workflow for generating BlenderGIS models is much faster than generating models manually from ASTER GeoTIFFs; the model resolutions are often better, the textures used are often higher resolution, and they contain Earth curvature correction. Because of this ease, BlenderGIS is used for nearly all tests. The texture and model sources used in BlenderGIS vary, with elevation models coming from the NASA Shuttle Radar Topography Mission (SRTM) and textures coming from aerial satellite imagery (monetarily) acquired by Google.

An additional advantage of using the BlenderGIS addon, instead of manually adding 3D models and textures, is that the Google imagery used is very high quality and scales well to variable GSDs when modeling camera systems in orbit. Furthermore, the simulated images were generated with orientations from a slew maneuver where the camera was always looking at a fixed point on the ground. This was done to ensure image overlap and have fine control over viewing angles. These simulations assumed a circular orbit of 400km and the desire was to input a certain θ viewing angle and output the satellite's orbital position at that viewing angle. Slew times and rates for such maneuvers have been calculated by the UGA SSRL in other research [2].

Noise Removal

Here we consider noise to be any point which should not be within the pointcloud. Noise removal is a significant challenge when designing 3D reconstruction algorithms, and for the Small Satellite Research Lab Computer Vision (SSRLCV) it is necessary to remove erroneous 3D points after the initial 3D triangulation. Points are assigned particular errors, described in section 4.5.1, and can be filtered based on these errors. The noise is assumed to be Gaussian. A combination of linear cutoff filtering (removes all points past a certain error) and statistical filtering (removes all points past a certain multiple of the standard deviation σ in the error distribution) yields very clean point clouds. Additionally, filtering at the feature matching level should be used. In the context of the SSRLCV software, this is known as seeding,

where an image known to be unrelated to the satellite images is input as an example to counteract false positives. David Lowe, the creator of the SIFT algorithm, recommends this approach to improve matching [11] [12]. If no filtering is done at this stage, then the resultant point clouds are considerably noisier, thus reconstruction without a seed is never recommended.

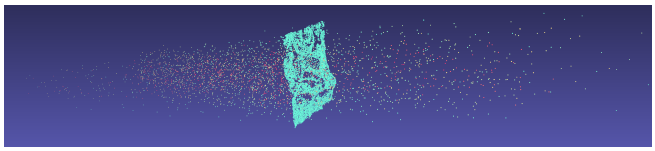


Figure 5. A visualization of the errors calculated for a 2-view case with filtering at the feature matching level. Red represents points with errors which are extrema and light teal represents errors which are close to zero.

When filtering 2-view reconstruction, the benefits of statistical filtering are most evident when viewing the distribution of errors. The intuition here is to remove the erroneous points which lay outside of the Gaussian. At first the distribution is quite large, but still densely packed around the optimal error of zero. It is important to observe that there are very few extremely erroneous points, but those points often contribute significantly to the total error of the point cloud. In order to perform a proper bundle adjustment, described in section 4, such points must be removed; otherwise their collective error may dominate the optimization resulting in a local minima of noise at the expense of valid points.

Filtering with the N-view case is less successful when only linear cutoff and statistical filtering are used. At this stage, it is necessary to perform regional density filtering because statistical filtering still leaves a significant number of noisy points and noise locations are considerably less dense in structure than the intended point cloud.

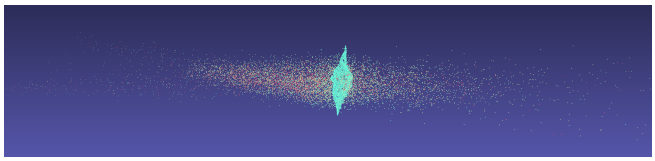


Figure 6. A visualization of the errors calculated for a 5-view case with filtering at the feature matching level. Red represents points with errors which are particularly bad and light teal represents errors which are close to zero.

There is considerably more noise, and thus more error in the cloud, for N-view cases. Overall the 2-view cases produce less noisy clouds that are about as accurate as the N-view clouds after filtering. N-view clouds require significantly more filtering to extract a viable model. This is because the N-view triangulation produces a wider spread of errors than the 2-view triangulation, seen in figures 7 and 8. This causes statistical filtering to time longer because it only removes a certain percentage of outliers at a time.

Reprojection and Triangulation

The tests with Blender and BlenderGIS seek to provide accuracy measurements which can be compared with current methods for the computer vision software developed for the MOCI mission. One of the most commonly cited existing

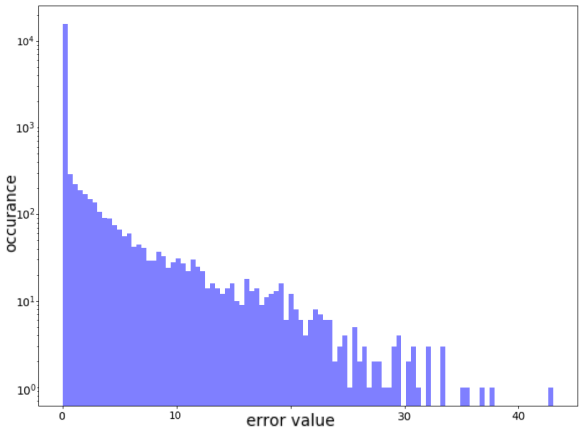


Figure 7. The distribution of linear errors in a 3-view triangulation.

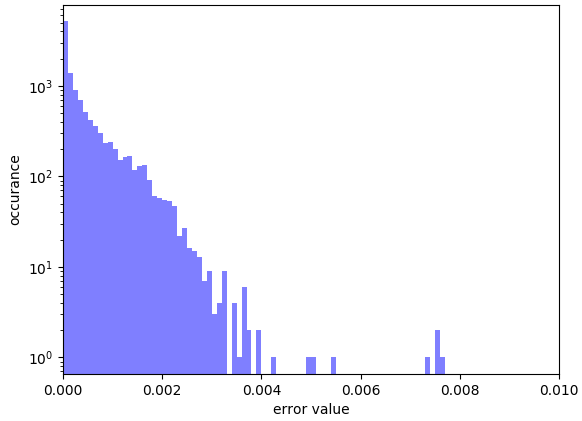


Figure 8. The distribution of linear errors in a 2-view triangulation.

reconstruction methods is the Visual Structure from Motion (VSFM) software released by Chang Chang Wu. Images rendered in BlenderGIS are output in the PNG format, but VSFM only accepts the JPG format. Images are converted from PNG to JPG using FFMPEG at its highest conversion setting with the simple command `ffmpeg -i 1.png -qscale:v 1 1.jpg`. FFMPEG is a program which can be brew, apt, or yum installed on your favorite UNIX-like operating system.

The SSRLCV 2-view reconstruction outperforms the VSFM 2-view reconstruction by producing a more dense initial point cloud and a more accurate initial point cloud. This is likely due to the fact that SSRLCV requires camera parameters before a reconstruction where VSFM only estimates the camera parameters. Both SSRLCV and VSFM modify these initial camera parameters in the bundle adjustment, but SSRLCV starts with estimated camera parameters which are likely quite accurate and VSFM does not.

Meshlab is used to view the individual point cloud results. The error between the ground truth model and the initial 2-view point cloud is calculated as a sum of the distances between individual estimated 2-view points and their inter-

Dataset	Res	Views	unseeded	seeded	3σ filter	2σ filter	1σ filter
Everest	1024	2	538618.1875	506.1283	19.8736	10.6172	6.9021
Everest	1024	3	27499.1660	14654.0087	5474.9116	3766.6467	1601.0681
Everest	1024	5	93405.1875	52093.9765	28640.6523	19155.5859	8549.2802

Table 1. Data on error removal correlated to point removal when filtering. Note all statistical filtering occurs after seed filtering, as this is the expected usage.

Dataset	Res	Views	unseeded	seeded	3σ filter	2σ filter	1σ filter
Everest	1024	2	0.7316	0.2266	0.2105	0.2079	0.2039
Everest	1024	3	0.7136	0.5124	0.4811	0.4646	0.4245
Everest	1024	5	0.7717	0.5796	0.5663	0.5506	0.5093

Table 2. Average distance, measured in km, of a point to 6 neighbors at certain error function filters.

section with the 3D model below them.

Generally, view number is not correlated with an increase in model resolution for SSRLCV, but is correlated with model resolution for VSFM; this can be seen in table 3. In some cases, the noise removal process skews the ground truth error distribution to make the final SSRLCV model appear worse than it truly is. This effect is seen when the removal of a small number of points causes the large variance σ to decrease to a reasonable amount. The Everest 1024 5-view case, seen in table 3, has only 31 (only 0.12% of the $24,901$) points which cause the variance to increase an order of magnitude. This shows the importance of successive filtering, as these points should be removed during the filtering step. Similar results can be seen with a 2-view 4096 Everest case, thus successive filtering is necessary for all cases.

Errors to the ground truth are close in SSRLCV 2-view cases but not in VSFM 2-view cases. In fact, some VSFM cases fail to generate 3D point clouds because camera parameters cannot be accurately determined; this can be seen in table 3. These errors are visualized in figures 9 and 10 where the reconstructed points hug the ground truth. Overall, SSRLCV is both more dense and more accurate than VSFM. VSFM produces no noise in any model, but does so at the expense of generating dense points. SSRLCV generates as many points as possible and allows the user of the pipeline to filter points at any stage. The benefit of such a design choice is accuracy, but the propagation of erroneous points into further stages of the pipeline can have tremendous consequences; this is explored in the next section.

The reconstructions from simulated Everest imagery work quite well, though an unexpected result was the larger errors at the peaks and higher altitudes as seen in figure 11. The figure also shows how the errors closer to the ground surface are better; this may be partially explained by the fact that the imagery used to perform the reconstruction is higher resolution than the Shuttle Radar Topography Mission (SRTM) which was used to compute the ground truth. This may also be explained by the fact that accurate regions tend to be more dense and not contain low density regions. The tips of the mountain contain regions of very low density and may be contributing to the error. VSFM produces a very sparse cloud at all locations, with almost no points at the sparse high altitudes of SSRLCV.

Bundle Adjustment

To test the Bundle Adjustment, noise was added to camera parameters so that the optimization could be observed. The Bundle Adjustment does improve the linear error of the point cloud but seems to get stuck in local minima. It is important to note that points and position estimation cannot be decoupled in the real world usage of MOCI, thus pointing estimation and position estimation are always considered together. Error values in position are from Simplified General Perturbations 4 (SGP4) and Two Line Elements (TLEs) positions, which can diverge from the true location by up to 1 km a day [15].

Iterations can be seen in graphs 12 and 13, where noise was added as described in table 4. Convergence in not always guaranteed, which could be caused by some of the following issues: improper definitions of error functions (perhaps the functions are not properly designed for convex optimization), floating point error associated with camera parameters, errors associated with the discrete approximation of the gradient and Hessian, or an elusive bug.

Image size is not a contributor to bundle adjustment convergence; the largest factor seems to be the error values that are given as noise parameters. When the noise parameters are large, they result in a less optimal starting position for the bundle adjustment. Poor starting locations can be seen at the top of graph 12, where only $\frac{1}{7}$ tests converged. On the other hand, locations at the bottom of the graph converged in $\frac{2}{3}$ cases. The 4096 case is similar, where lower errors converged but only some higher errors converged. However, because the sample size is small, these tests may not indicate any overall trend, though it is expected that noise closer to the real camera parameters should make convergence easier.

6. PIPELINE TIMING

The execution times of various pipelines in SSRLCV depend on the number of input images. More precisely, the execution times are bottlenecked by feature generation, feature matching, and bundle adjustment. Triangulation, filtering, and point normal estimation all occur very quickly. File IO is fast and not a major factor in execution time; it is essentially negligible. The tests seen in table 5 were run on an Nvidia TX2i.

The MOCI mission is expected to process datasets which are

Dataset	Res	V.	SSRLCV Avg. Dist	σ_{SSRLCV}	N_{SSRLCV}	VSFM Avg. Dist	σ_{VSFM}	N_{VSFM}
Everest	1024	2	114.603 m	186.709	11,768	288.296 m	315.317	1,797
Everest	1024	3	53.8238 m	531.069	13,131	142.311 m	198.477	5,993
Everest	1024	5	149.08 m	2842.87	24,901	78.4359 m	159.465	7,747
Everest*	1024	5	55.3631 m	135.74	24,870	–	–	–
Everest	4096	2	57.2854 m	1486.79	73,376	165.679 m	220.515	1,655
Everest*	4096	2	47.2689 m	126.547	73,233	–	–	–
Rainier	1024	2	178.548 m	2260.29	12,888	*failed*	*failed*	*failed*
Rainier	1024	3	135.168 m	793.099	13,357	*failed*	*failed*	*failed*
Rainier	1024	5	121.992 m	481.374	24,005	263.361 m	368.128	224
Rainier	4096	2	101.204 m	2322.29	154,280	*failed*	*failed*	*failed*

Table 3. Initial reconstructions, compared by average error to the ground truth and the sigma value of the error distribution. The number of points in the clouds is also provided. Cloud Compare's Iterative Closest Point (ICP) implementation was used to calculate the errors shown above. In some cases VSFM failed to produce a point cloud. Results with a $*$ were manually filtered after automatic filtering to show the ideal results of SSRLCV (VSFM results require no filtering and are thus already ideal and marked with " $-$ "; results in the row above should be used to compare).

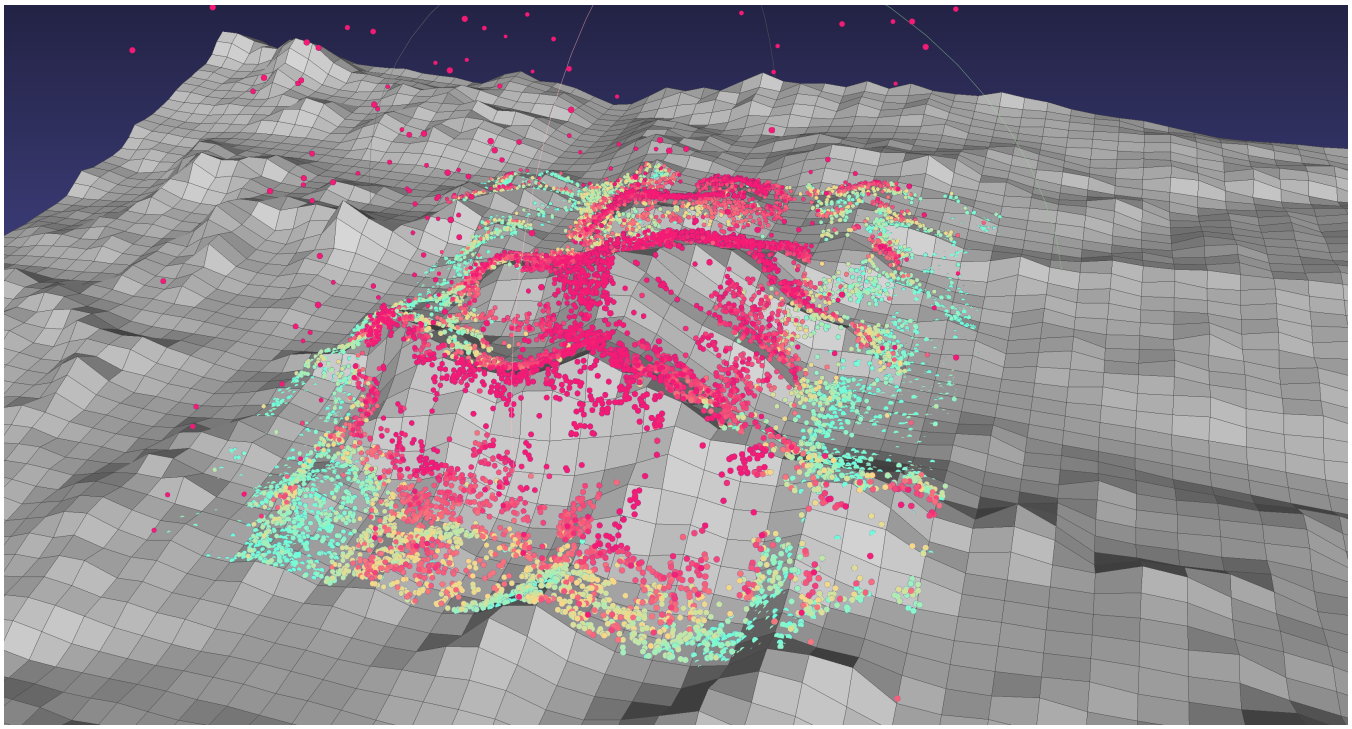


Figure 9. A 2-view 4096 $\times$ 4096 SSRLCV Everest reconstruction compared to the ground truth model.

closest to the Everest 4096 and Rainier 4096 sets used in table 5. The tests above indicate that MOCI's operations may take anywhere from 35 minutes to 1.4 hours. The sets above were selected because they are expected to yield best and worst case results for timing. The Everest dataset represents a best case computation for MOCI, where the low contrast of the snowy grey mountains would produce fewer features than the green high contrast Rainier dataset. One should note that the number of points in the point cloud is the same as the number of matches not filtered. The point number comparison of Rainier and Everest in table 3 shows that the Rainier dataset generated about $2.1\times$ more points than the Everest dataset. A common MOCI dataset will be somewhere in between these two examples.

The primary bottlenecks of the MOCI pipeline are Bundle Adjustment, Feature Matching, and Feature Generation. MOCI will likely not be capable of running a full pipeline at once, the payload will likely have to shut down and restart the pipeline at a given stage. Feature generation can be checkpointed after each image's features are generated. Bundle Adjustment can be checkpointed after each Newtonian iteration. Matching cannot currently be checkpointed, though it is possible to do by adding on to the existing SSRLCV library.

Initial Error	1σ	Starting Error	Ending Error	Agv. Iter.	Avg. Good Iter.	Avg. Good Result	Success
Everest 1024, 2 View, 10 tests at 200 max iterations							
6.9021	$x \pm 250$ m $y \pm 250$ m $z \pm 250$ m $\theta_x \pm 0.00053$ $\theta_y \pm 0.00053$ $\theta_z \pm 0.00053$	1354.9344	924.0120	65	85	13.1173	30%
Everest 4096, 2 View, 5 tests at 200 max iterations							
506.7010	$x \pm 250$ m $y \pm 250$ m $z \pm 250$ m $\theta_x \pm 0.00053$ $\theta_y \pm 0.00053$ $\theta_z \pm 0.00053$	3776.0946	2399.1825	51	85	621.0281	60%

Table 4. Initial bundle adjustment results where a noise vector is added to the camera parameters of one view in the set. Noise is added where the mean value is $\mu = 0$ and σ is defined as a positive and negative range. Error values are measured in Average Linear Error, mentioned in section on bundle adjustment in chapter 4. Each row was repeated several times with random noise added within the specified parameters. Successes, measured in the right hand column, are tests that reach within $2\times$ the global optima.

Dataset	Res	Views	Feat.	Mat.	Tri.	Filt.	B.A.	Total
Everest	1024	2	27.045 s	47.421 s	0.121 s	0.635 s	115.02 s	199.69 s
Everest	1024	3	40.365 s	176.556 s	0.155 s	2.852 s	–	230.043 s
Everest	1024	5	66.994 s	436.103 s	0.285 s	3.28 s	–	699.498 s
Everest	4096	2	352.404 s	1091.193 s	0.61 s	3.28 s	699.498	2156.436 s
Rainier	1024	2	26.692 s	40.238 s	0.101 s	0.622 s	110.4 s	183.92 s
Rainier	1024	3	40.374 s	157.101 s	0.165 s	2.768 s	–	210.058 s
Rainier	1024	5	67.558 s	413.127 s	0.276 s	5.188 s	–	495.883 s
Rainier	4096	2	389.809 s	3309.492 s	1.189 s	6.601 s	1443.502s	5159.922 s

Table 5. Runtimes for given sections of the pipeline on given datasets. Bundle adjustment was limited to 10 iterations and only tested on 2-view cases. In cells marked with " – " no bundle adjustment was run. Total runtime is listed on the right and may be slightly more than the individual sum due to small operations between pipeline stages. The total time also includes seed image feature generation. The same seed image was used for all tests (it runs in ≈ 9.448 seconds).

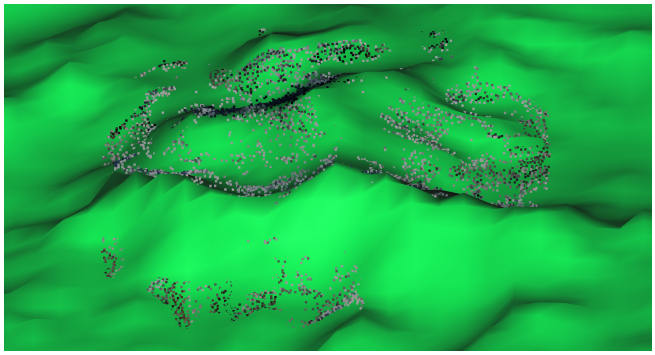


Figure 10. A 5-view 1024×1024 VSFM everest reconstruction compared to the ground truth model.

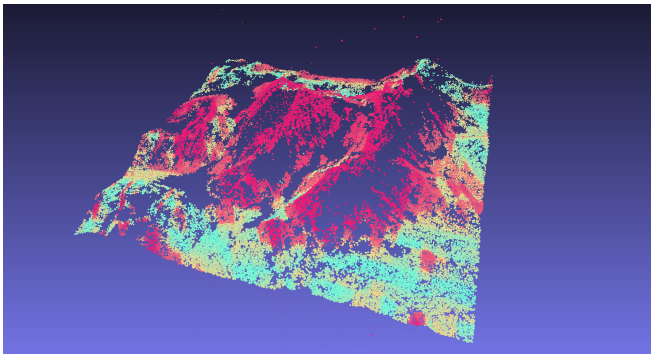


Figure 11. A 2 view reconstruction of 4096 x 4096 images of Mount Everest simulated at a 400 km ISS-like orbit; here the red points represent points with errors to the ground truth at or above 300 meters; the yellow represents errors around 150 meters and light teal color represents errors approaching zero.

Hardware Experiments

Individual cores can be shut off with the command `sudo nvpmodel N` where `N` is a mode number defined in table 6. The goal of initial experiments is to identify where the

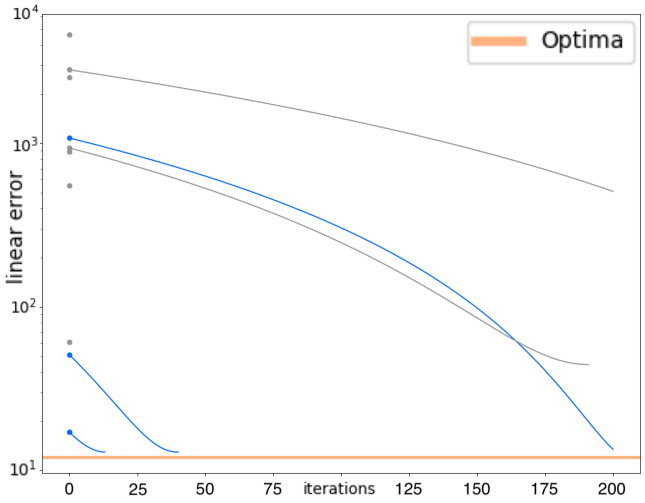


Figure 12. 10 bundle adjustment tests run with noise from table 4; the values graphed here show the first tests in the table.

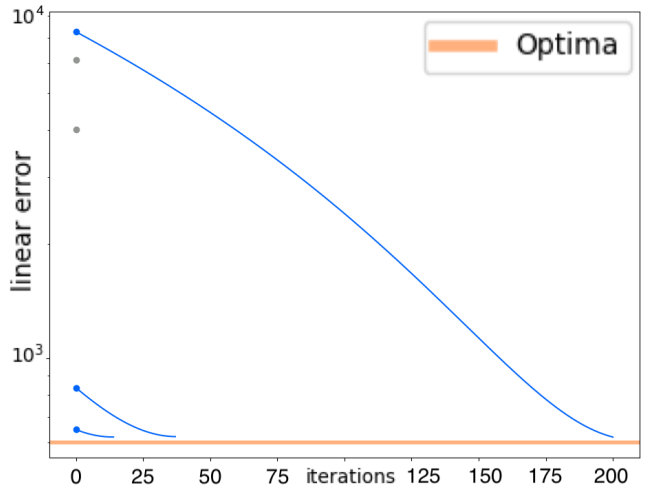


Figure 13. 5 bundle adjustment tests run with noise from table 4; the values graphed here show the second tests in the table.

CORGI / TX2 / TX2i system will have significant power usage. Thermal properties are not modeled, but could be modeled at a later date by repeating these tests in the UGA SSRL's thermal vacuum chamber. Power is measured instead of thermal output because any non-vacuum measurements of thermal output will be inaccurate. Power, however, is directly correlated with thermal output and can be used to refine thermal models of the computation unit. For example, one could assume a 100% power to heat conversion to obtain worst case thermal models from the data below.

The computer vision pipeline is run in these different power configurations with various inputs. Additionally, it is required by the MOCI mission that the payload does not exceed a certain wattage, namely the wattage of MAX-Q. Because the Jetson TX2 has built-in utilities to monitor power consumption, the power consumption was able to be directly monitored in an idle state. A logger is provided with the SSRLCV software which can monitor state transitions, voltage, current, and power consumption over time. Idle computation level is included and a CSV file is generated as the result of the program. Average power consumption lies at 3.195 Watts and peak power consumption is always below 8 watts.

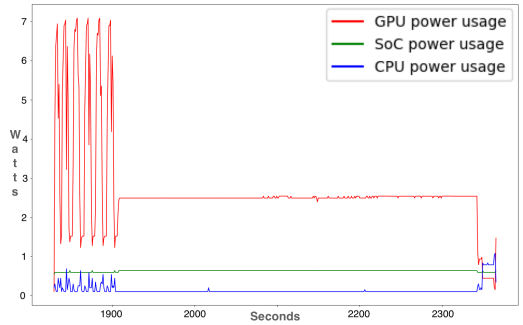


Figure 14. A 5-view reconstruct of 1024 × 1024 images of Everest showing the power consumption in milliwatts of the GPU, SOC, and CPU over the time the pipeline is executing. Tested on a development TX2.

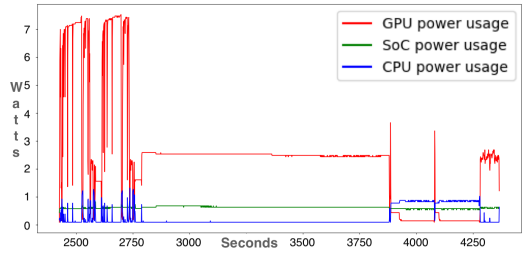


Figure 15. A 2 view reconstruction with 4096 × 4096 images of Everest showing the power consumption in milliwatts of the GPU, SOC, and CPU over the time the pipeline is executing. Tested on a development TX2.

The power graphs seen in 14 and 15 show the computer vision pipeline over time. The graphs track the power usage in milliwatts of the 3 primary systems on the Tegra, the SoC, the GPU, and the CPU. It is important to note that 14 and 15 do not contain bundle adjustment calculations in their pipelines. The six power spikes at the beginning of the pipeline in 14 are the result of feature generation; one spike is from the seed image and the other five spikes from the main images. Surprisingly, feature generation, not matching, is the most power intensive. The flat sustain of GPU usage comes from filtering and triangulation. The CPU power usage tends to spike during memory transfers to and from the GPU while the SoC power usage tends to remain constant. The graphs 14 and 15 were generated from the TX2 and not the TX2i.

The TX2i produces power results that are *almost* identical to those of the TX2. These can be seen in graphs 16 and 17, but are about a Watt higher at the feature generation peaks. These results are promising, especially when considering the timing seen in table 5. Whereas Bundle Adjustment or Feature Matching may have initially seemed to be the most intensive sections of the pipeline, when considering power usage, Feature Generation emerges as the primary thermal, power, and temporal constraint. Matching and Bundle Adjustment, though they will run longer on average than Feature

Mode	Name	Denver 2	Hz	ARM A57	Hz	GPU Hz
0	Max-N	2	2.0	4	2.0	1.3
1	Max-Q	0	-	4	1.2	0.85
2	Max-P Core-All	2	1.4	4	1.4	1.12
3	Max-P ARMv8	0	-	4	2.0	1.12
4	Max-P ARMv8	1	-	4	2.0	1.12

Table 6. Power Modes for the TX2 / TX2i

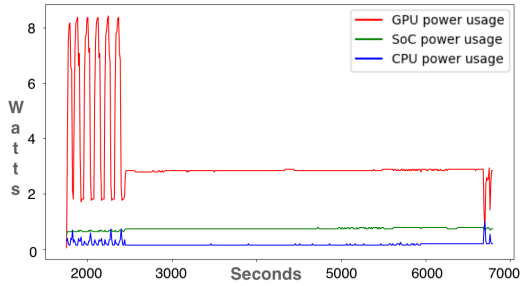


Figure 16. A 5-view reconstruction of 1024 × 1024 images run on the TX2i, the intended computation unit for the MOCI satellite, showing power usage over time. The power usage is almost identical to that of the TX2. Power consumption in milliwatts of the GPU, SOC, and CPU over the time the pipeline is executing.

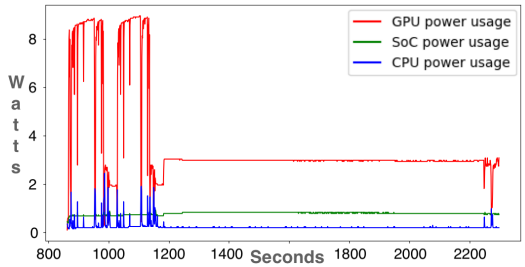


Figure 17. A 2-view reconstruction of 4096 × 4096 images run on the TX2i, the intended computation unit for the MOCI satellite, showing power usage over time. The power usage is almost identical to that of the TX2. Power consumption in milliwatts of the GPU, SOC, and CPU over the time the pipeline is executing.

Generation, use less than half of the power. Thus, Matching and Bundle Adjustment will deplete MOCI's batteries at a slower rate and generate a less intense thermal load.

APPENDICES

A. CODE AND RESOURCES IN THIS PAPER

The resources used in this paper can be requested at any point by contacting the authors of this paper or visiting the SSRL Software page at `http://smallsat.uga.edu/software`. The computer vision software demonstrated here, known as SSRLCV, is open source and is available on github at `https://github.com/uga-ssrl/SSRLCV`. To see sample data, generate more sample data, or for detailed instructions on how sample data is obtained see `https://github.com/uga-ssrl/SSRLCV-Sample-Data`

ACKNOWLEDGMENTS

The authors would like to thank the Georgia Space Grant Consortium for funding these GPU research projects and the Air Force Research Laboratory's University Nanosat Program for giving us tremendous opportunities and for funding the projects that led us to this point. The authors would also like to thank Hollis (Nicholas) Neel and Aaron Martinez for helping with checking the math in this paper. A special thank you to Roger Hunter for helping the UGA SSRL over all these years.

REFERENCES

[1] C. Adams, A. Spain, J. Parker, M. Hevert, J. Roach, and D. Cotten, "Towards an integrated GPU accelerated SoC as a flight computer for small satellites," in *2019 IEEE Aerospace Conference*. IEEE, Mar. 2019. [Online]. Available: https://doi.org/10.1109/aero.2019.8741765

[2] C. Adams, "High performance computation with small satellites andsmall satellite swarms for 3d reconstruction," Master's thesis, The University of Georgia, University of Georgia, Athens, GA 30602, United States, 5 2020.

[3] "Technology horizons : a vision for air force science and technology 2010–30," 2010. [Online]. Available: http://www.defenseinnovationmarketplace.mil/resources/AF_TechnologyHorizons2010-2030.pdf

[4] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, ISBN: 0521540518, 2004.

[5] A. J. Rossi, "Abstracted workflow framework with a structure from motion application. thesis," 2014. [Online]. Available: https://scholarworks.rit.edu/theses/7814

[6] R. T. Collins, "A space-sweep approach to true multi-image matching," in *Proceedings CVPR IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jun. 1996, pp. 358–363.

[7] A. Baumberg, "Reliable feature matching across widely separated views," in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No.PR00662)*, vol. 1, Jun. 2000.

[8] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle Adjustment — A Modern Synthesis," in *Vision Algorithms: Theory and Practice*, ser. Lecture Notes in Computer Science, B. Triggs, A. Zisserman, and R. Szeliski, Eds. Springer Berlin Heidelberg, 2000, pp. 298–372.

[9] B. Zitova and J. Flusser, "Image registration methods: a survey," *Image and vision computing*, vol. 21, no. 11, pp. 977–1000, 2003.

[10] H. Kuuste, T. Eenmaee, V. Allik, A. Agu, and R. Vendt, "Imaging system for nanosatellite proximity operations," *Proceedings of the Estonian Academy of Sciences/#/Proceedings of the Estonian Academy of Sciences*, vol. 63, no. 2, pp. 250–257, 2014.

[11] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2. Ieee, 1999, pp. 1150–1157.

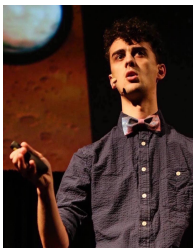
[12] ——, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[13] I. R. Otero and M. Delbracio, "Anatomy of the sift method," *Image Processing On Line*, vol. 4, pp. 370–396, 2014.

[14] C. Liu, J. Yuen, and A. Torralba, "Sift flow: Dense correspondence across scenes and its applications," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 5, pp. 978–994, 2011.

[15] D. Vallado, P. Crawford, R. Hujsak, and T. Kelso, "Revisiting spacetrack report #3: Rev 2," *Astrodynamics Specialist Conference*, 2006.

B. BIOGRAPHY



Caleb Adams *Caleb Adams was the co-founder of the UGA Small Satellite Research Laboratory, which now has 2 satellite missions. Caleb has a Masters in Computer Science from UGA and now works as a Civil Servant at NASA's Ames Research Center where he helps develop Distributed Spacecraft Autonomy and Aerial Perception systems. His research focuses on small satellites, distributed computing, and perception systems.*

Jackson Parker *Jackson Parker received his M.S. in Electrical and Computer Engineering in spring of 2020 from the University of Georgia, during which he acted as Payload Software Architect and Systems Engineer for the MOCI mission. Currently, he works at NASA Ames Research Center helping the Payload Accelerator for Cubesat Endevors group design and develop highly robust flight software.*

David Cotten *David L. Cotten, Ph.D., received a B.S in Physics from Louisiana State University in 2005. He currently serves as the manager and Co-PI of the UGA's Small Satellite Research Laboratory and is an Assistant Research Scientist at the Center for Geospatial Research in the Geography Department at UGA. He graduated from UGA in 2011 with his doctorate in Physics and Astronomy. His research as a Post-Doctoral Associate focused on surface-atmosphere exchange, and he is currently using remote sensing (multispectral/hyper spectral sensors) and micrometeorology techniques to quantify carbon storage in wetland regions at both the local and regional scales. Dr. Cotten is also using unmanned aerial vehicles, air photos, and satellite imagery to build 3D models of terrestrial objects using photogrammetric Structure from Motion.*