THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF INDUSTRIAL AND MANUFACTURING ENGINEERING

THREE-DIMENSIONAL MODEL RECONSTRUCTION FROM IMAGES USING POSITION TRACKING AND STRUCTURE FROM MOTION

BENJAMIN JACQUES SATTLER
Spring 2017

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Mechanical Engineering
with honors in Industrial Engineering

Reviewed and approved* by the following:

Saurabh Basu
Assistant Professor of Industrial and Manufacturing Engineering
Thesis Supervisor

Catherine Harmonosky
Assistant Professor of Industrial and Manufacturing Engineering
Honors Advisor

* Signatures are on file in the Schreyer Honors College.

ABSTRACT

Currently, reverse engineering techniques require a combination of laser scanners, coordinate measuring systems, and human interaction to generate usable files. All of these methods are both cost prohibitive and require many hours to complete. The end result is a three-dimensional model with varying degrees of accuracy. Recreating three-dimensional models is extremely beneficial in cases where the original manufacturer is no longer in business or if the part was manufactured prior to modern three-dimensional modeling techniques.

This thesis investigates the accuracy of model generation using photogrammetry algorithms. A digital single-lens (DSLR) camera or a camera found on a modern cell phone are used to keep the cost and barrier to entry low. Initial work completed compared the accuracy of off-the-shelf software before moving on to customized algorithms. New methods combine position tracking from an inertial measurement system (IMU) alongside Structure from Motion (SfM) techniques to create accurate three-dimensional models.

The two software packages evaluated are PhotoModeler made by EoS Systems Inc. and Remake made by AutoDesk. Three objects, each presenting different challenges to the photogrammetric method, are used to conclude which software package is more accurate. On all three tests, Remake was the most accurate, at best achieving tolerances of $\pm^{0.6833}_{1.2598}\,mm$ and at worst $\pm^{1.6688}_{1.0084}\,mm$. After conducting tests on a newly created SfM algorithm written in MathWorks Inc's. MATLAB, the length of a 76.2 mm cube was calculated to be 76.3 mm.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Appreciation and thanks goes to

Dr. Saurabh Basu

for mentoring me and dedicating time to this research, as well as his dedication to the Department of Industrial and Manufacturing Engineering

Dr. Catherine Harmonosky

for her dedication to research and the Department of Industrial and Manufacturing Engineering

Mr. Michael Immel

for continuously pursuing new ideas and inspiring the work contained within this thesis

Ms. Debra Rodgers

for facilitating the coordination with the Department of Industrial and Manufacturing Engineering

Chapter 1

Introduction

Reconstructing computerized three dimensional models is one part of reverse engineering a product and can be needed for many different reasons. However, this process is often difficult and expensive, requiring special equipment such as laser scanners and coordinate measuring machines as well as specialized personnel. In addition, completing model reconstruction is a time intensive process. New techniques, such as photogrammetry, are emerging to replace traditional reverse engineering methods. A comparison of photogrammetry and laser scanning can be seen in Table 1. Photogrammetry uses overlapping images and vision processing to create three-dimensional scene reconstruction [1]. Traditionally, photogrammetry uses aerial photographs to map large areas such as agriculture fields, stadiums, architecture, and mines [2]. An example of this can be seen in Figure 1. The blue dots represent each camera position. The goals of this work include evaluating the accuracy of current products available as well as methods for improvement by tracking the position of each image.

Figure 1: An example of photogrammetry using aerial photography. The blue points are camera locations.

Table 1: A comparison of photogrammetry and laser scanning provided from Barsanti [3].

	Photogrammetry (Image-Based modeling)	**Laser Scanner (Range-Based modeling)**
Characteristics		
Cost of the instruments (HW and SW)	Low	High
Manageability/Portability	Excellent	Sufficient
Time of data acquisition	Quite short	High
Time for modeling	Quite short, experience required	Often long
3D information	To be derived	Direct
Distance’s dependence	Independent	Dependent
Dimension’s dependence	Independent	Dependent
Material’s dependence	Almost independent	Dependent
Geometry’s dependence	Dependent	Almost/totally independent
Texture’s dependence	Dependent	Independent
Scale	Absent	Implicit (1:1)
Data volume	Dependent on the images resolution and on the measurements	Dense point cloud
Detail’s modeling	Good/excellent	Generally excellent
Texture	Included	Absent/Low resolution
Edges	Excellent	Quite problematic
Statistics	From each calculated point	Global
Open-source software	Some	A few

As photogrammetry has previously been used to model large objects, with a scale of many meters, the accuracy of these methods on small industrial parts, on the scale of centimeters, is being evaluated. This process is known as close-range photogrammetry. Two different software packages, Eos Systems PhotoModeler and Autodesk Remake, will be evaluated. Eos Systems PhotoModeler is designed and recommended for use in architectural, accident scene, and archeological image reconstruction [4]. Autodesk Remake is marketed for similar archeological purposes, but also for creating prototypes and preparing models for additive manufacturing [5]. A variety of objects of known dimensions will be used to evaluate the accuracy of both software packages. Once understood, techniques will be evaluated to test potential improvements.

While the exact technique used by Eos Systems and Autodesk is proprietary information, structure from motion (SfM) is one of the most commonly employed techniques in the past 15 years for scene reconstruction [6]. As such, this method is utilized in this thesis and will be the basis for iteration. SfM uses the images to calculate the translation and rotation of one camera to another by computing the fundamental matrix. The fundamental matrix relates the points in two images using epipolar geometry, as explained in Chapter 3. This method of calculating the fundamental matrix to find camera position was pioneered by Luong in the mid 1990's [7]. While this method works well for calculating the rotation of the cameras, translation can only be calculated with a scaling factor. It is common to compute the translation vector with a length of one to facilitate post process scaling [8]. If the position of the relative camera position is known, the result of the SfM algorithm will not need to be scaled. This should increase the accuracy as there is no need for user input on the final results, thus reducing the chances of human error. None of the software tested included this capability, and thus a new algorithm was created.

Chapter 2

Evaluation of Current Technology

To accurately evaluate both PhotoModeler and Remake, a consistent hardware and environmental setup were used. A single camera, with a set focal length, aperture, and ISO, in a studio lit environment was used to take all pictures of all objects. Three different objects were selected to be modeled, each attempting to capture different types of challenges. The first object was a 76.2 millimeter (mm) cube, selected to test the accuracy of hard edges as well as to present a baseline. The second object was similar to the first, but contained depth information. This was achieved by machining conics and semicircles into the faces of a 76.2 mm cube. Due to the added cuts and overhangs, the possibility of shadows greatly increased which often presents challenges to the reconstruction process of photogrammetry [2]. The third object selected contained complex curves in addition to depth information and sharp edges. This was selected to test the accuracy of reconstruction of objects with mainly non-straight edge features. All three objects were machined to within .05 mm of the original design, as measured by a coordinate measuring machine. Figure 2 shows the three objects being used for this thesis. To compare the results, all output models were compared to the true model. From this comparison, a tolerance was calculated. This method was selected as tolerancing parts is common in industry and often drives the manufacturing process used to create the product.

Figure 2: The three machined objects used for close-range testing.

Camera Parameters

The camera chosen for this project was a Cannon EOS 6D DSLR. A fixed, 100 mm focal length lens was used along with an aperture set to f/32.0. The shutter speed was 1/10 second and an ISO of 3200. The EOS 6D has a resolution of 20.2 megapixels and 35 mm, full frame image sensor. These camera parameters were selected based on previous published studies as well as the lighting in the room [9].

Due to the ability to capture sharp images in relatively small working spaces, the macro lens was chosen. The highest accuracy of photogrammetry can be achieved when 50-80% of the image pixels are of the desired object [10]. A macro lens allows more of the image to be of the desired object, and has the effect of being “zoomed in” on the object. The fixed focal length was chosen as a non-moving focal length is one of the assumptions made in three-dimensional scene reconstruction algorithms, which will be discussed in further detail in Chapter 3-Camera Calibration.

Data Collection

To decrease any variance between images, a studio was used in an effort to create even and smooth lighting. In addition, the objects were placed on a plain, uniform color background. This environment produces the best results as it decreases shadows and increases the contrast between the desired object and the background [9]. Both the lighting and background can be seen in Figure 3.

Figure 3: Studio setup, eliminating shadows and producing even lighting.

To keep conditions as consistent as possible from object to object, the object remained stationary at the center of the setup and the camera was mounted to a tripod. A delay was used on the shutter, so the act of pressing the button to capture the picture would not cause vibrations and thus blurriness in the image. Two concentric circles were drawn on the ground, the outer circle being for the single back tripod leg and the inner circle being for the front two tripod legs. For each object, a total of 128 images were captured. The circle was divided into 32 evenly spaced sections, with a picture being taken every 11.25 degrees. A circumferential path of pictures was taken at 4 different heights, corresponding to the total of 128 images. Based off testing by Behrouzi and researchers at the University of Arkansas, the line of sight to the camera should be no more than 60 degrees [9] [2]. As such, the four passes were made at 60 degrees, 40 degrees, 20 degrees, and 0 degrees respectively. Due to the variance in object size and vertical location of the camera, the distance from the lens of the camera to the object ranged from 38 to 66 centimeters. The position of the camera and the tripod can be seen in Figure 4.

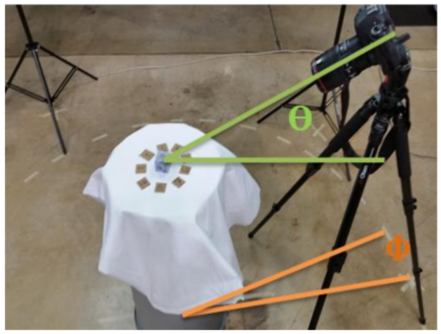

Figure 4: The position of the camera where Ө is the angle between the object and the camera's line of sight and where Φ is the constant 11.25° angle between each image.

In addition to the setup of the camera, the objects also needed preparation. Photogrammetry uses vision processing and feature tracking, discussed in further detail in Chapter 3, which needs unique features to track across images. Highly reflective and glossy surfaces, also known as non-Lambertian surfaces, do not allow for easy feature tracking and produce poor results [11]. While methods for image reconstruction on non-Lambertian surfaces are an area of research, these will not be considered for the scope of this thesis. As the objects being used started as polished aluminum, adding a matte coating was necessary to create a Lambertian surface. This was achieved by simply coating each object with a thin layer of a baby power. A chalk spray was also tested, but the baby powder was ultimately selected due to better results, less expense, and ease of sourcing.

The final component in this setup is a measuring device. The ruler is included for post processing the data. As stated in Chapter 1, the results of the scene reconstruction algorithm are not scaled and a known distance must be included for the final model to produce metric results. As the purpose of this section of the thesis is to test the accuracy of the final object size, it would be biased to scale the model based off the known dimensions of the object itself.

Software Used and Steps Taken

While PhotoModeler and Remake were the main software packages being tested, ultimately additional software was needed to generate point clouds as well as compare results. Each software package required slightly different processes, which are explained in the following sections. All non-open source software was purchased by the Department of Industrial and Manufacturing Engineering at The Pennsylvania State University.

PhotoModeler

Unlike Remake, PhotoModeler requests the camera being used to be calibrated. This is not required, but highly recommended by EoS Systems. While not specifically stated by EoS Systems, this step is needed to calculate the internal parameters of the camera. Internal parameters include image sensor size, lens focal length, as well as distortion characteristics: in total, there are 11 unknown variables that need to be solved [12]. Camera calibration will be discussed further in Chapter 3- Camera Calibration. PhotoModeler provides a printable calibration target for the user as well as a built-in calibration algorithm.

Once the camera has been calibrated, photographing each object can begin. Upon uploading the images, the software analyzes each picture, differentiating between the object and the background. After doing this for each image, features between each image are matched and then triangulation can begin. The process of triangulation and generation of a three-dimensional model occurs entirely on the user's computer. For the processing of the 128 photos used for this research, a minimum of 8 gigabytes (GB) of random access memory (RAM) was needed but 16 GB is recommended [13].

The results require post processing as well. After triangulation, there is typically still some of the background that needs to be removed. PhotoModeler has point cloud as well as mesh editing tools, allowing the user to remove the background. The biggest drawbacks are a lack of a “fill feature”, allowing the user to define a bottom plane and create a solid body, and the lack of a “hole fill” feature, allowing the user to quickly remove holes. As mentioned previously, the original result is scaled, and has no units. The user must define two points in the model and provide a distance. An example of an unmodified output from PhotoModeler can be seen in Figure 5.

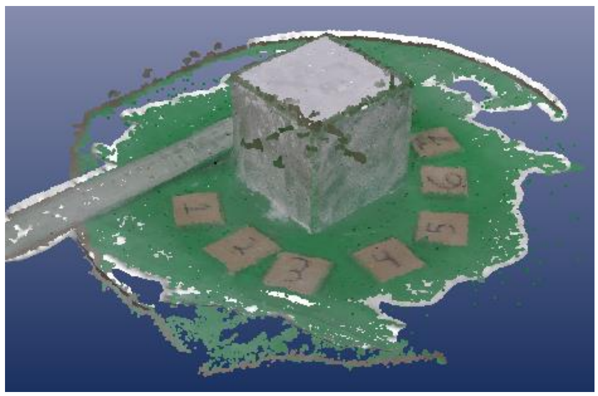

Figure 5: An unmodified output from PhotoModeler. This 3D object has no scale, and the background has not yet been removed.

Remake

While similar to PhotoModeler in many ways, Remake does not request a calibration file. This feature alone greatly increases the ease of use for this software but increases the processing time. Not having a calibrated camera does come at a cost, and that comes at computation expense. Remake requires at least 64 GB of RAM and Autodesk recommends 128 GB of RAM, making this software not feasible for current laptop technology, and even most desktop computers [14].

To mitigate the requirement for such large amounts of RAM, Autodesk provides cloud computing services. While this provides a solution for one problem, it also presents new issues as well. The largest advantage the cloud computing option has is freeing up the local machine for other tasks. In addition, this feature lowers the cost for a user, and computers with less RAM are typically less expensive. However, using the cloud also presents an uncontrollable variable for the user. Once a project is uploaded, the project is added to a queue to be processed. While this can be monitored within the program, the user has no control as to if the project is actively being solved. For all the uses in the project, this downside provided no hindrance as no project took longer than 4 hours to solve. An example of the results from Remake can be seen in Figure 6.

Once triangulation is complete and the file is on the local machine, the mesh can be manipulated manually. Similar to PhotoModeler, basic mesh removal tools exist to facilitate background removal. Remake does have a "fill feature" as well as a "hole fill" feature, making the post processing extremely easy. As will all reconstruction algorithms, the image has no scale, and this must be entered manually.

Figure 6: An unmodified output from Remake. This 3D object has no scale, and the background has not yet been removed.

The final output from Remake is a stereolithography (STL) file. While this file is beneficial for computer aided design (CAD) programs and additive manufacturing, it is difficult to evaluate the results. Conversion of the STL is discussed in the next section.

Point Cloud generation

In order to compare the results from PhotoModeler and Remake, a point cloud was deemed to be the most effective way. The process of comparing the results file to ground truth is discussed in the next section.

PhotoModeler generates both a point cloud and an STL file. While simply using the point cloud file would be ideal, further mesh refinement beyond the capabilities of the software was required. The most important addition was a closeout layer on the bottom of the STL. This was done using an open source, mesh editing software, MeshLab. Care was taken to change the rest of the mesh as little as possible, to have as close to no effect on the accuracy of the original file. Meshlab is able to save files as both STLs as well as point clouds. This software was used to save the output of PhotoModeler as a point cloud after modifications were made. Even though no modifications were needed to be made outside of Remake, Meshlab was still used. The results file from Remake was simply opened and then saved as a point cloud. It is important to note that all scaling was done in the original software, either PhotoModeler or Remake.

Comparing to Ground Truth

The first step in analyzing the output of the two software packages evaluated is creating a ground truth. As the objects were relatively simple, CAD files were made (for the object with complex curves, this was required for machining) and saved as STL files. Using the open source software CloudCompare, point clouds can be compared to an STIL file by using point cloud registries. The output of the point cloud registry process is a data point corresponding to each location in the point cloud with distance information to the ground truth STL file.

Using the output of the point cloud registry, maximum, minimum, and average variance were calculated. In addition, histograms were plotted along with calculations of standard deviations. All of this data was compared across the three objects and from both PhotoModeler and Remake to evaluate accuracy.

Results

In addition to the accuracy computed through point cloud registries, STL quality was also evaluated. STL quality was evaluated by comparing the number of holes in the generated mesh, the number of inverted triangles, the number of overlapping triangles, the number of bad edges, and the number of intersecting triangles.

While there is some variance in the results, Autodesk Remake was the most accurate in every case when looking at the point cloud compared to ground truth. When comparing the quality of the STL mesh, Remake always had less holes and fewer overlapping triangles. For the two cubes, Remake had fewer inverted triangles and fewer bad edges, but performed worse on the third object with complex curves. PhotoModeler had less intersecting triangles in all cases. This is most likely due to the way the different software programs added the bottom of the model. Table 2 shows the full results.

Table 2: Results for all three objects in both PhotoModeler and Remake.

	Software	**PhotoModeler**			**Remake**		
		Object 1	Object 2	Object 3	Object 1	Object 2	Object 3
STL/Point Cloud	Number of Planar Holes	0	1	0	0	0	0
	Number of Inverted Triangles	0	342	0	0	0	56
	Number of Overlapping Triangles	384	1960	775	358	125	569
	Number of Bad Edges	0	768	3	0	68	30
	Intersecting Triangles	328	476	155	999	1332	1488
Computer Comparison	Max Pos Variance (mm)	2.6314	2.6010	1.7678	0.6833	1.3589	1.6688
	Max Neg Variance (- mm)	1.9126	2.3368	2.6594	1.2598	1.1354	1.0084
	Average Variance (mm)	0.1600	0.1168	0.0076	-0.1219	-0.0229	-0.0635
	Standard Deviation (mm)	0.2235	0.2743	0.3429	0.3404	0.1981	0.3226

It is surprising that both PhotoModeler and Remake performed the worst on the object selected to be the baseline when comparing average variances and maximum positive variance. However, the results are quite varied if standard deviation is compared. Ultimately, in manufacturing the overall tolerance of a part is one of the most critical pieces of information. As such, the minimum and maximum were used to create tolerance windows. For object 1, Remake was the most accurate, with a tolerance of $\pm_{1.2598}^{0.6833} mm$. Remake also held a tighter tolerance on object 2 at $\pm_{1.1354}^{1.3589} mm$. The tighter tolerance for object 3, of $\pm_{1.0084}^{1.6688} mm$ was also achieved by Remake. These results provide a baseline so a decision can be made if the photogrammetry process is suitable for different tolerancing applications. A visual representation of the results can be seen in Figure 7 and Figure 9. Figure 8 and Figure 10 show the histograms of the deviations of the models created by photogrammetry compared to ground truth.

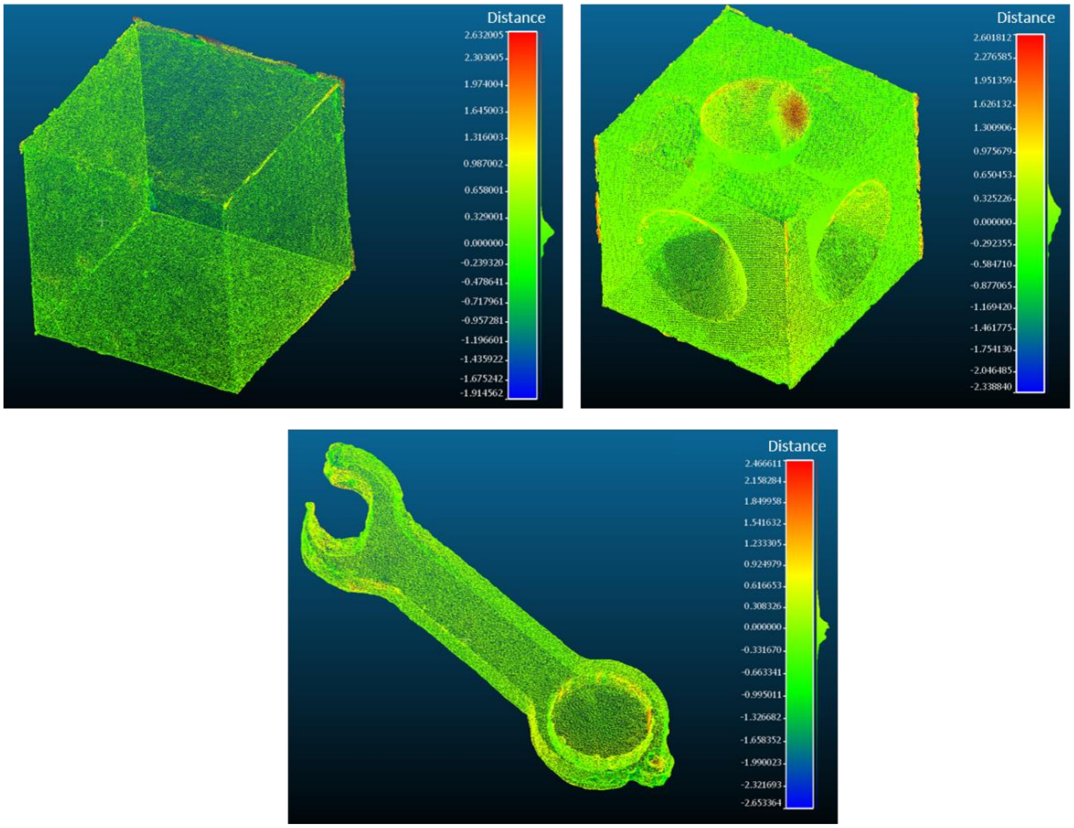

Figure 7: Results of all three objects from PhotoModeler.

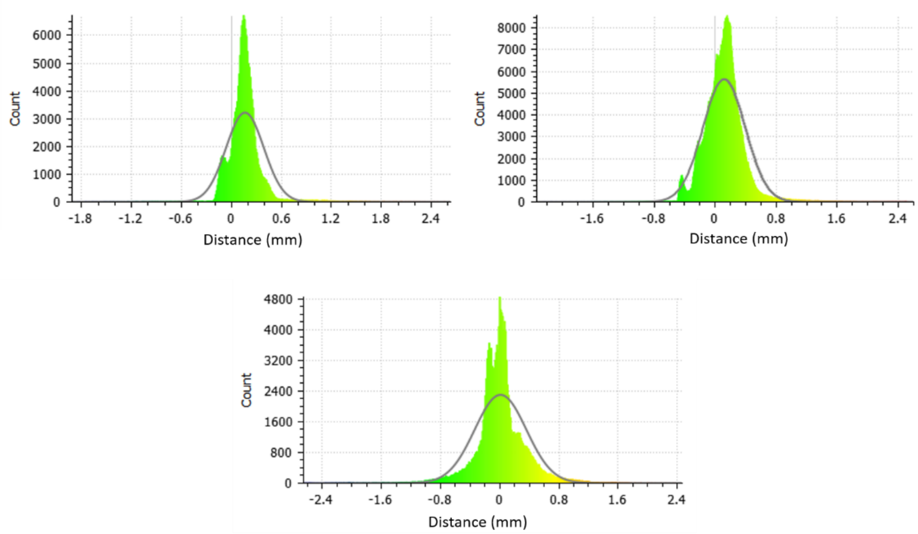

Figure 8: Distribution of variance for objects 1, 2, and 3 from PhotoModeler respectively.

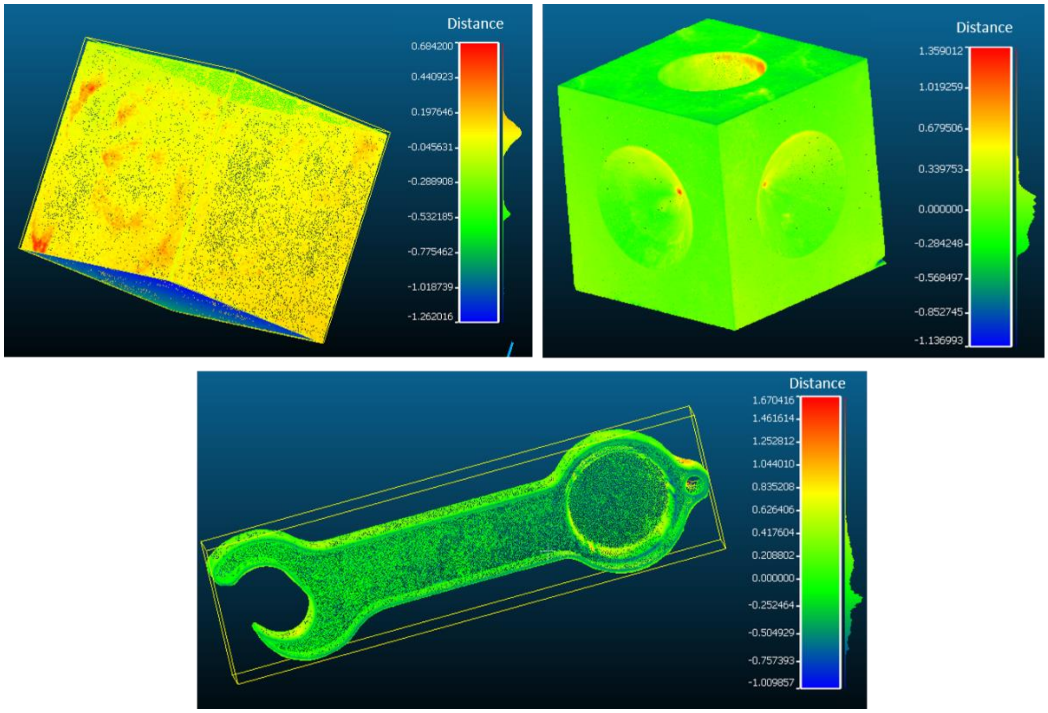

Figure 9: Results of all three objects from Remake.

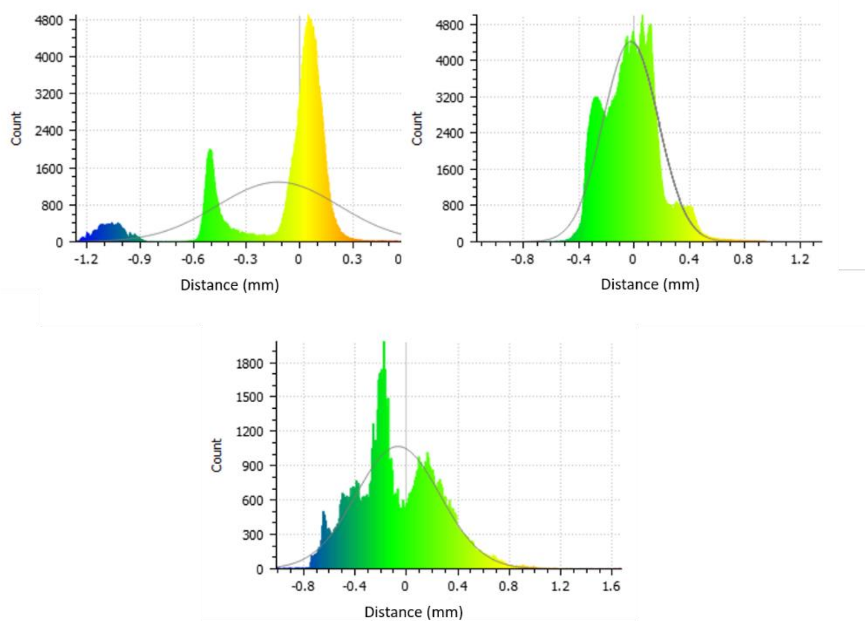

Figure 10: Distribution of variance for objects 1, 2, and 3 from Remake respectively.

Chapter 3

Improvements with Position Data

Now that the baseline from two photogrammetry software packages is complete, efforts to improve the accuracy can be made. In order to do this, a complete understanding of the Sfm algorithm is needed. This begins with understanding the camera model and camera calibration. The correction factors computed in the calibration step feed into SfM. Once this is mastered, position data is included to improve accuracy. This step is critical as it takes human input out of the equation, the largest source of error. This technique is the basis of stereo cameras and has not been applied to a monocular set of images.

Camera Calibration

In order to calibrate a camera, it is important to first understand the model of a camera. One of the simplest and most common models is known as the pin-hole camera model. The pin-hole camera assumes a small hole in a plane that the rays of an image pass through to create an inverse image on the opposing side. The pin hole of the model corresponds to the lens of the camera, and the image plane is the sensor chip. If the focal point is a known value and the distance to the image plane is known, then a three-dimensional world point P{XYZ} can be described as a two-dimensional image point p{x y}. This concept is depicted in Figure 11. To solve for the coordinates for p{x y}, the following equations can be used:

$$x = f * \frac{X}{Z}$$

$$y = f * \frac{Y}{Z}$$

where *f* is the focal length of the camera and X, Y, and Z are the world coordinate points. From Figure 11 there are two key lessons. The first is there are an infinite amount of points world points P{XYZ} that correspond to image point p, as long as it falls along the ray displayed in red. This will become important in Chapter 3- Structure from Motion with Unknown Position. The second is that in real life, the point p{x y} is a single pixel on a sensor chip and to increase light there is a lens in front of the image plane, or the sensor chip. This lens is not perfect and distorts the image, requiring correction factors. [12]

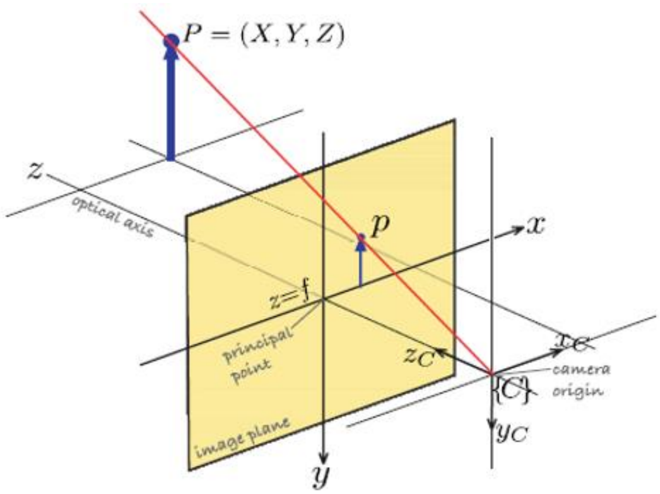

Figure 11: Pin-hole camera model relating world point P{XYZ} to image point p{x y}. Image sourced from [12].

Typically, a camera is described with two matrices; the matrix K to describe the camera parameters of focus, the height of each pixel, the width of each pixel, and the {x y} point where axis z_c crosses the image plane, as well as the matrix ξ_c to describe the pose of the camera. The pose is fully defined by six variables corresponding to the translation and orientation of the camera. While in theory these 11 variables are known, in practice they are unknown due to variations in manufacturing standards. Solving for these 11 variables comprises one step of the camera calibration. The matrix K is depicted below:

$$K = \begin{bmatrix} f/\rho_w & 0 & u_0 \\ 0 & f/\rho_h & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

Where *f* is the focal length, p_w is the width of each pixel, p_h is the height of each pixel, u_o and v_o represent the point where axis z_c crosses the image plane. [12] To correlate the image plane to a pixel array and to calculate the u_o and y_o values:

$$u = \frac{x}{\rho_w} + u_0$$

$$v = \frac{y}{\rho_h} + v_0$$

Where u and v correspond to the pixel position that relate the sensor array position to the point p{x y} on the image plane. [12]

The second step is calculating distortion. Distortion is seen in two main ways, tangential and radial. A tangential distortion causes the image to shift off center while radial distortion causes points to shift along radial lines originating at the {x y} point where axis z_c crosses the image plane. The radial distortion usually has a larger effect on the image. For example, radial distortion is one of the common characteristics of a fisheye lens. Characterizing radial distortion is completed with three variables and tangential distortion with two variables. Computing these five variables is the second part of camera calibration. The distortion δ_u and δ_v can be explained by:

$$\begin{bmatrix} \delta_u \\ \delta_v \end{bmatrix} = \begin{bmatrix} u(k_1 r^2 + k_2 r^4 + k_3 r^6) \\ v(k_1 r^2 + k_2 r^4 + k_3 r^6) \end{bmatrix} + \begin{bmatrix} 2p_1 uv + p_2(r^2 + 2u^2) \\ p_1(r^2 + 2v^2) + 2p_1 uv \end{bmatrix}$$

Where the first matrix represents the radial distortion and the second matrix represents the tangential distortion. The k values are the radial coefficients and the p variables are the tangential coefficients that need to be determined. Typically, three coefficients are used for radial distortion and two coefficients are used for tangential distortion. [12]

While camera calibration can be done with a single image containing known three-dimensional data, the process is much easier with multiple known two-dimensional data images. For the purpose of this thesis, the MathWorks MATLAB single camera calibration application was used. Numerous images were taken of a calibration checkerboard of known size. The calibrator detects the points of the checkerboard and compares the detected points to where the points should lie. By using multiple images, the entire calibration matrix can be computed [15]. Figure 12 shows an example of a calibration picture with the corners of the calibration matrix detected.

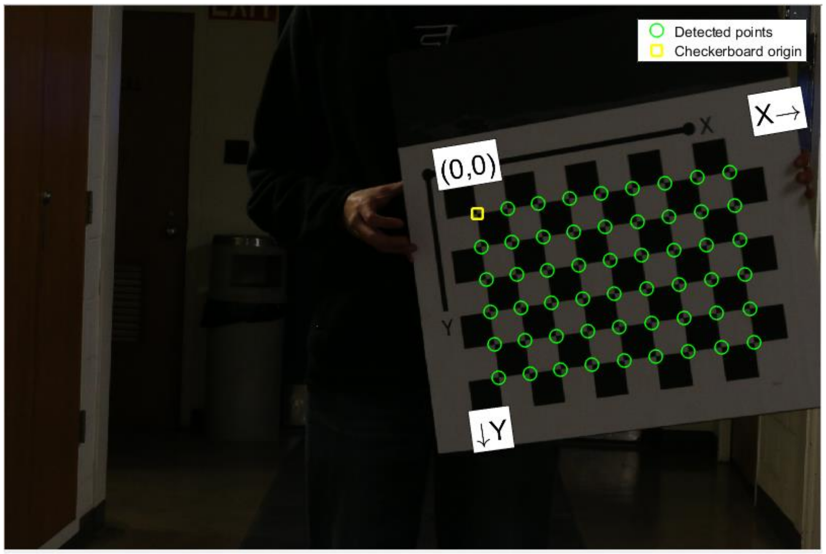

Figure 12: An example of the calibration board used with the MathWorks camera calibration application.

Structure from Motion with Unknown Position

The basic premise of SfM is using a set of images to correlate points in the image frame p{x y} to the world coordinates P{XYZ}. As was depicted in Figure 11, a single image cannot be used to calculate a world point, and a minimum of a second image is needed. For the purpose of this thesis, using simply two images will be employed, which will be discussed in further detail later in this section. If we call the origin of the first camera C_1 and the origin of the second camera C_2, then the intersection of rays C_1P and C_2P will correlate to a unique world point [8]. This basic concept is seen in Figure 13. By using this method for hundreds of points, a point cloud of the world can be generated.

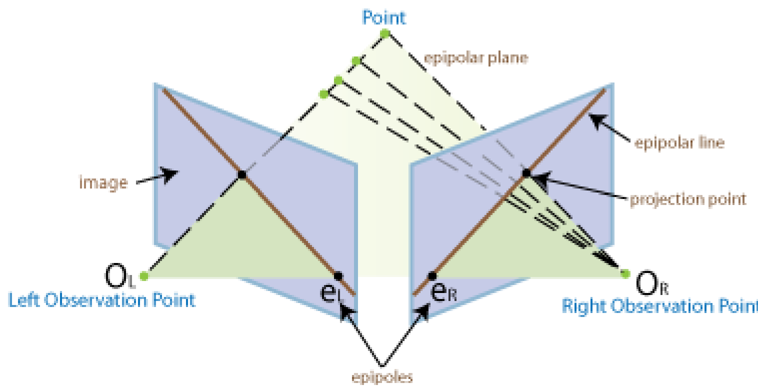

Figure 13: SfM depiction, with unique points P, and where O_L and O_R correspond to C_1 and C_2 respectively. Image sourced from [8].

Figure 13 also shows the epipoles, epipolar lines, and the epipolar plane. These three features are used to identify the pose of the second camera relative to the first. For the purpose of this explanation, the points p_L and p_R correspond to the point label projection point in Figure 13 on the left and right images respectively. The projection point is where the ray O_LP and O_RP pass through left and right respective image planes. At the most basic level, the epipolar line is the projection of the opposite OP ray. For example, the epipolar line on the right image is the projection of ray O_LP onto the right image. The epipolar line on the left image is the projection of ray O_RP onto the left image. The epipolar line can be described even further though by looking at the projection points p_L and p_R, along with epipoles e_L and e_R. The epipoles are simply the projections of the opposite projection point. So, e_R is the projection of p_L on the right image plane and e_L is the projection p_R on the left image frame. This means the rays e_Lp_L and e_Rp_R are the epipolar lines for the left and right images respectively. Finally, the plane created by O_LO_RP (which passes through e_L and e_R) is the epipolar plane. Once all the epipolar features are computed, a 3x3 matrix known as the Fundamental matrix can be calculated. Using the Fundamental matrix, p_L, and p_R, the pose and translation of the second camera to the first can be calculated. [7]

Computing the Fundamental matrix itself is outside the scope of this thesis, but with the help of prewritten MATLAB functions the orientation and transpose of the second camera can still be calculated. The function *estimateFundamentalMatrix* is used to compute the Fundamental matrix, which is then given to the function *relativeCameraPose* along with the matched image points and the calibrated camera matrix, the orientation and transpose of the second camera to the first camera is calculated [16]. However, the scale of the transpose cannot be calculated without further information. This is the same basic principle that makes it impossible to tell if an apparent small object is simply large and far away or an apparent large object is simply small and very close. As such, the transpose vector is calculated with a length of one, and the image is not metric [8].

In order to put all the theory stated above into practice, a basic SfM algorithm was written. To keep the method as simple as possible, only two views were used. While including multiple images to create a denser point cloud is feasible, to create a baseline reconstructed model this added complexity was avoided. Figure 14 to Figure 18 shows the output of that algorithm. Figure 14 shows the original images and Figure 15 shows the same images after distortion has been removed. Due to the high quality DSLR being used, there is not a lot of distortion present. Figure 16 shows the first set of points matched overtop of the two images, as well as the direction of point travel. These points are the epipolar points used to calculate the Fundamental matrix. Once the camera positions are calculated, another set of matched points are calculated to create a dense point cloud. Since the camera position has been calculated, the minimum quality of each detected point can be reduced. These new points are shown in Figure 17. Finally, Figure 18 shows the reconstructed scene. In this case, the front of the cube can be clearly seen, along with the two positions of the camera when the images were captured. It can also be seen from the scale of the axis, that the face of the cube measures close to 1 square unit, which makes sense as this image has not been scaled in any way.

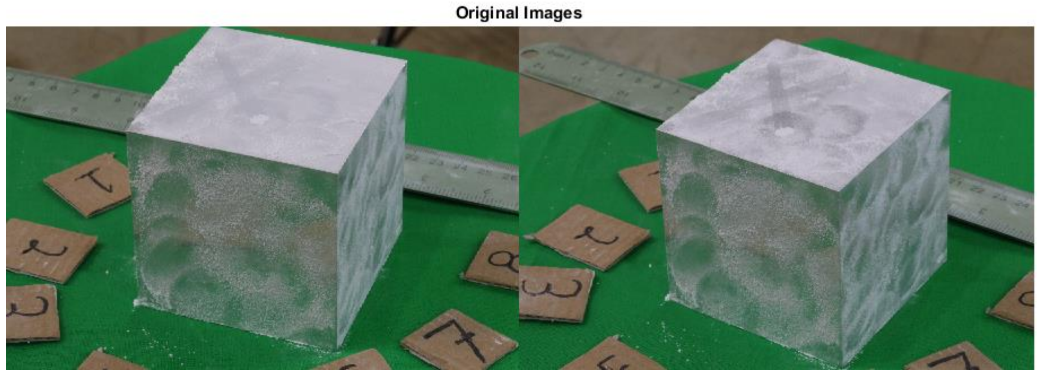

Figure 14: Original images (distorted) used in a two view SfM.

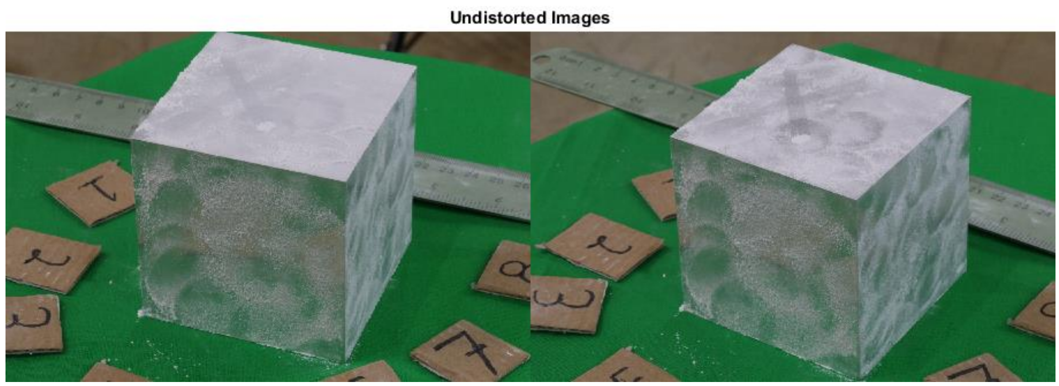

Figure 15: Undistorted images used for two view SfM.

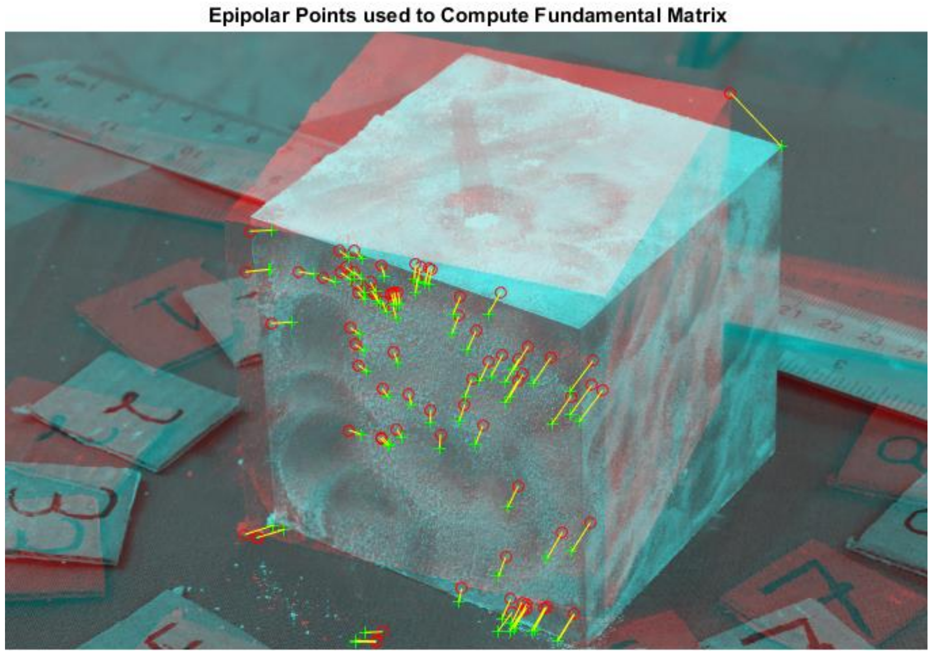

Figure 16: Epipolar points used to compute the Fundamental matrix.

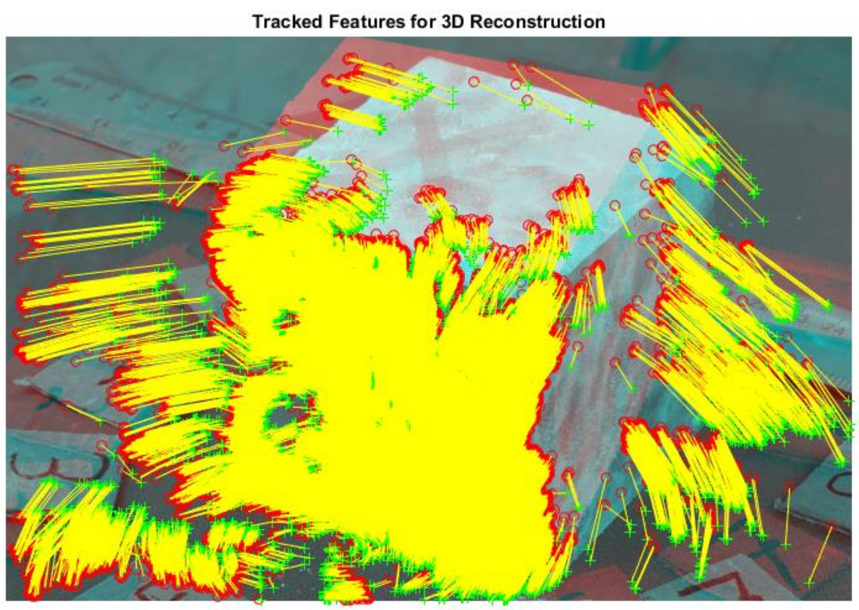

Figure 17: A total of 71,983 points were tracked between the two images.

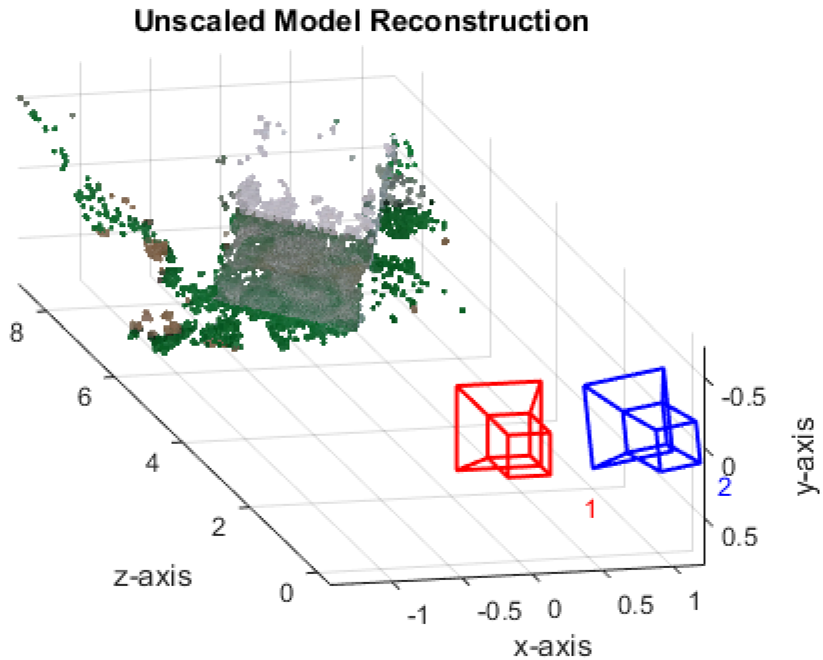

Figure 18: Unscaled point cloud calculated from the two view SfM algorithm.

Structure from Motion with Known Position

To create a properly scaled image, the transpose vector simply needs to be multiplied by the actual distance the camera moved. Since the images were taken from the pictures taken in Chapter 2-Data Collection, the distance from one camera to a second can be calculated. By estimating the location of the image sensor, and an 11.25 degree shift of the camera around the object, the approximate distance of the transpose is 73.66 mm. Since the SfM algorithm calculates the transpose with a length of one, simply scaling the transpose by a factor of 73.66 will yield a metric result. The only change made from the algorithm completed in the previous section is the addition of the scaling factor to the transpose vector. As such, Figure 14 to Figure 16 remain identical, and only the output changes. The results can be seen in Figure 19. The width of the block is approximately 76.3 mm when it should be 76.2 mm. The width can be calculated from the two points shown, by finding the difference vector and calculating the length. These equations can be seen below.

$$\begin{bmatrix} \Delta X \\ \Delta Y \\ \Delta Z \end{bmatrix} = \begin{bmatrix} -40.106 \\ -20.043 \\ 483.362 \end{bmatrix} - \begin{bmatrix} 27.584 \\ -5.370 \\ 451.358 \end{bmatrix}$$

$$\begin{bmatrix} \Delta X \\ \Delta Y \\ \Delta Z \end{bmatrix} = \begin{bmatrix} -67.69 \\ -14.673 \\ 32.004 \end{bmatrix}$$

$$length = \sqrt{67.69^2 + 14.673^2 + 32.004^2} = 76.298\ mm$$

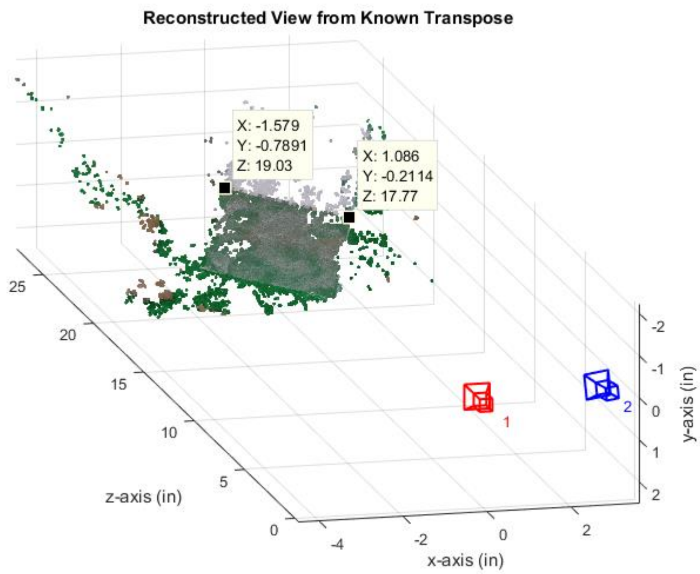

Figure 19: Scaled SfM results, allowing for measurements.

With the basic two image SfM algorithm functional, increased point density can be reconstructed by using an increased number of views. To test the addition of multiple views, only the first rotation of images from Chapter 2-Data Collection will be used: yielding 32 pictures. Due to increased computation cost, the image set was further reduced to 29 pictures. The most significant difference between the multiple view algorithm compared to the two-image algorithm is a data set to store camera poses as well as tracked points [17]. In addition, points are tracked across more than just the adjacent image. For example, features in image 7 could be matched to features in image 2. The results from this algorithm can be seen in Figure 20. One can easily see a problem seems to have arisen. The camera poses should be arranged in essentially a circular plane around the object. The phenomenon seen in Figure 20 is a well-known problem that occurs due to error build up as each pose is calculated. A technique pioneered in 1999 known as Bundle Adjustment can be used to reduce the error build up [18]. While portions of this technique were used, it can be seen that adjustment of the camera pose was only effective until approximately the fourth image.

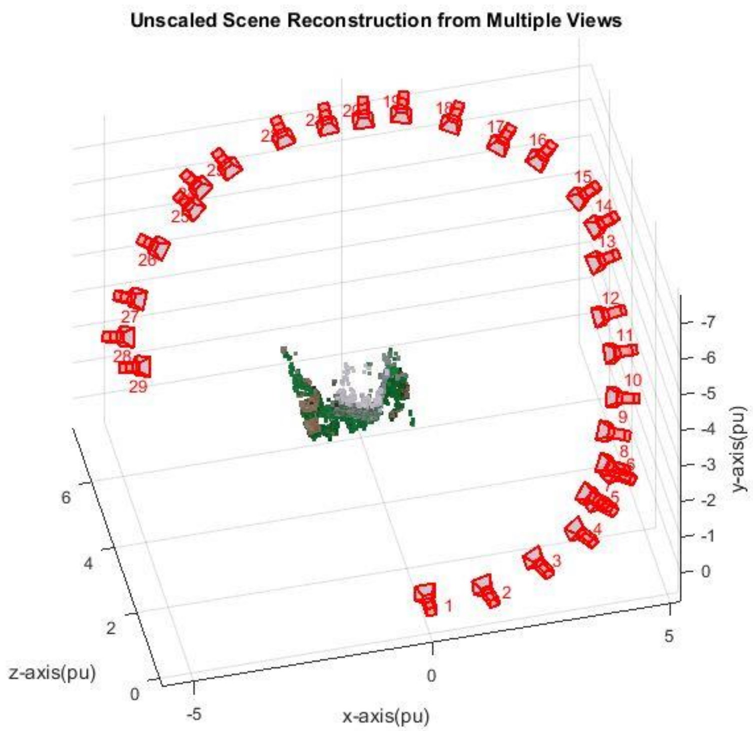

Figure 20: Attempt to use multiple view with SfM.

Due to the intricacies of Bundle Adjustment and a potential reduction in error by calculating position data via alternative methods, multi-view SfM algorithms will not be pursued further. Instead, the focus will return to improving the scaling coefficient used in the two-image algorithm, as this can later be applied to the multi-image SfM. While the results in Figure 19 are extremely accurate, deviation can occur. The step most likely to introduce error is the measurement and calculation of the position of the camera. Estimations were made as to the location of image sensor and very primitive methods of measuring the angle between the camera positions were used. Even though the calculations provided proper scaling, the fact the measurement is being completed by a human adds a possibility of error.

Position Tracking

As seen in Chapter 3-Structure from Motion with Known Position, knowing the transpose of one camera to another can yield extremely promising results. If the true transpose of the camera is known, the result of the SfM algorithm is metric. In order to improve the position tracking from human measurement, this section will focus on using an Inertial Measurement Unit (IMU). As modern cellular devices contain both IMUs as well as cameras, these devices make the perfect test subject. To obtain accurate data from an IMU, the first step is understanding how to model one and calculate the necessary variables. The output from the IMU is accelerometer data in three axes and a gyroscope to provide angular acceleration. The data was filtered to remove the noise prior to calculating the position.

Inertial Measurement Unit Modeling

The first step in obtaining accurate accelerometer data is removing gravity. To remove the gravity vector, the orientation of the phone must be known, which can be calculated from the gyroscopic information. The output of the IMU is $\dot{\phi}, \dot{\theta}, and\ \dot{\psi}$, corresponding to the roll rate, pitch rate, and yaw rate respectively. Roll pitch and yaw can be obtained by [19]:

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \delta t + \begin{bmatrix} \phi_{k-1} \\ \theta_{k-1} \\ \psi_{k-1} \end{bmatrix}$$

Where ϕ, θ, and ψ are the roll pitch and yaw of the phone. These values are then used to remove the gravity component, as seen below [19]:

$$\begin{bmatrix} \dot{u}_{true} \\ \dot{v}_{true} \\ \dot{w}_{true} \end{bmatrix} = \begin{bmatrix} \dot{u}_{meas} + g * sin(\theta) \\ \dot{v}_{meas} + g * cos(\theta)\sin(\phi) \\ \dot{w}_{meas} - g * cos(\theta)\cos(\phi) \end{bmatrix}$$

Where $\dot{u}_{true}, \dot{v}_{true}, \dot{w}_{true}$ are the acceleration in the x, y, and z axis with gravity removed, and $\dot{u}_{meas}, \dot{v}_{meas}, \dot{w}_{meas}$ are the measured accelerations from the IMU. The effects of gravity on the IMU acceleration data can be seen in Figure 21.

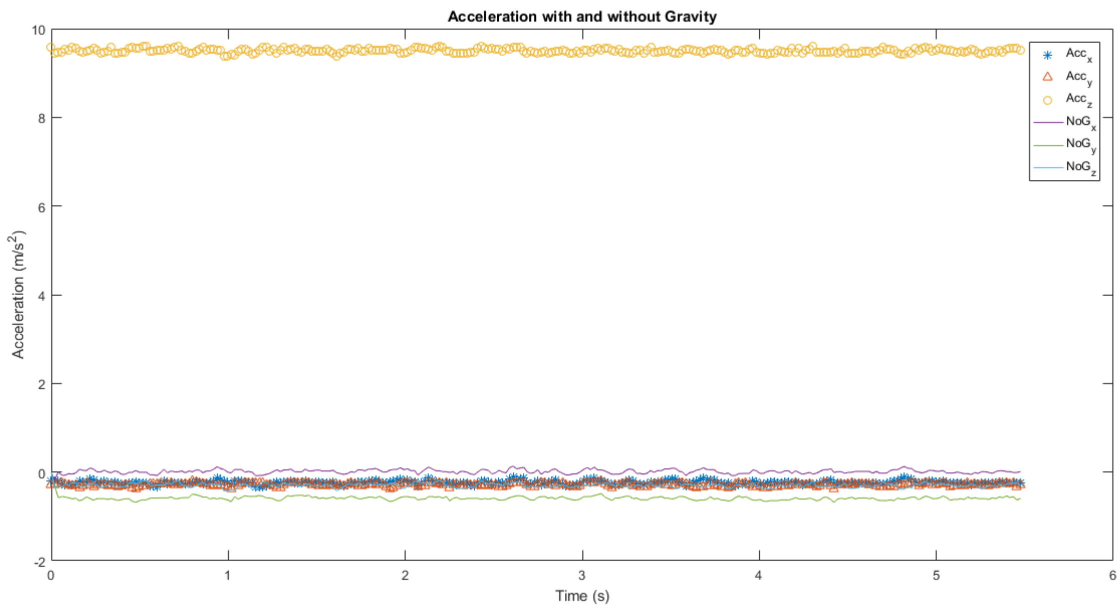

Figure 21: IMU Acceleration data before and after accounting for gravity.

In addition to removing gravity, it can be observed from Figure 21 that even with gravity accounted for, all three axes have an offset bias. In an attempt to make the sensor output as accurate as possible, the average acceleration was calculated and stored as a bias parameter. This bias was then applied to the data prior to integrating to find position.

Calculating Position

To calculate the position, simple numeric integration was used. First, the initial conditions for velocity were set as 0 m/s. Once completing the second integration, the position of the phone is found, however the position calculated is in body coordinates. To be useful for the SfM algorithm the position is needed in global coordinates. While a Euler direction cosine matrix could be used, the chances of the phone rotating into gimbal lock (when the rotation is zero degrees, and causing an unsolvable matrix) is quite probable. As such, the quaternion was utilized for the transformation from body to global coordinates. The following equations were used to calculate body position and velocity, as well as global velocity and position [19].

$$\begin{bmatrix} u_k \\ v_k \\ w_k \end{bmatrix} = \begin{bmatrix} \dot{u}_{true} \\ \dot{v}_{true} \\ \dot{w}_{true} \end{bmatrix} * dt + \begin{bmatrix} u_{k-1} \\ v_{k-1} \\ w_{k-1} \end{bmatrix}$$

$$\begin{bmatrix} x_k \\ y_k \\ z_k \end{bmatrix} = \frac{1}{2} * \left(\begin{bmatrix} u_k \\ v_k \\ w_k \end{bmatrix} + \begin{bmatrix} u_{k-1} \\ v_{k-1} \\ w_{k-1} \end{bmatrix} \right) * dt + \begin{bmatrix} u_{k-1} \\ v_{k-1} \\ w_{k-1} \end{bmatrix}$$

Where u_k, v_k, w_k are the velocities in the x, y, and z directions respectively and x_k, y_k, and z_k describe the phone position in body coordinates. Then, using the ϕ, θ, and ψ (roll pitch and yaw), the quaternion vector can be calculated as defined by [20]:

$$\{\hat{q}\} = \begin{Bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{Bmatrix} = \begin{Bmatrix} l * sin\left(\frac{\theta}{2}\right) \\ m * sin\left(\frac{\theta}{2}\right) \\ n * sin\left(\frac{\theta}{2}\right) \\ cos\left(\frac{\theta}{2}\right) \end{Bmatrix}$$

Where l, m, n describe the unit vector $\hat{\boldsymbol{u}}$ in the body coordinate system. To transform from the body *xyz* coordinate system to the global *XYZ* system the direction cosine matrix can be calculated as [20]

$$[\boldsymbol{Q}]_{xX} = \begin{bmatrix} {q_1}^2 - {q_2}^2 - {q_3}^2 + {q_4}^2 & 2(q_1q_2 - q_3q_4) & 2(q_1q_3 + q_2q_4) \\ 2(q_1q_2 + q_3q_4) & -{q_1}^2 + {q_2}^2 - {q_3}^2 + {q_4}^2 & 2(q_2q_3 - q_1q_4) \\ 2(q_1q_3 - q_2q_4) & 2(q_1q_3 - q_2q_4) & -{q_1}^2 - {q_2}^2 + {q_3}^2 - {q_4}^2 \end{bmatrix}$$

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} = [\boldsymbol{Q}]_{xX} \begin{bmatrix} u_k \\ v_k \\ w_k \end{bmatrix}$$

$$\begin{bmatrix} X_k \\ Y_k \\ Z_k \end{bmatrix} = \frac{1}{2} * \left(\begin{bmatrix} \dot{X}_k \\ \dot{Y}_k \\ \dot{Z}_k \end{bmatrix} + \begin{bmatrix} \dot{X}_{k-1} \\ \dot{Y}_{k-1} \\ \dot{Z}_{k-1} \end{bmatrix} \right) * dt + \begin{bmatrix} \dot{X}_{k-1} \\ \dot{Y}_{k-1} \\ \dot{Z}_{k-1} \end{bmatrix}$$

Where $\dot{X}, \dot{Y}, and\ \dot{Y}$ describe the velocity of the phone and X_k, Y_k, and Z_k describe the position of the phone in global coordinates [19].

Results

While numerous different tests were completed, all yielded extremely similar results. Due to compounding error that is integrated twice, the final calculated position is extremely variable and does not accurately represent the actual IMU movement. Figure 22 shows the calculated position of a stationary phone over the course of only five and a half seconds. While the calculated position originally was clustered within an approximately 0.5mm by 0.1 mm by 0.1 mm box, it can be seen that when the IMU was stopped recording the position was becoming rapidly inaccurate. If allowed to continue, the calculated position would have predicted meters of movement within a matter of minutes.

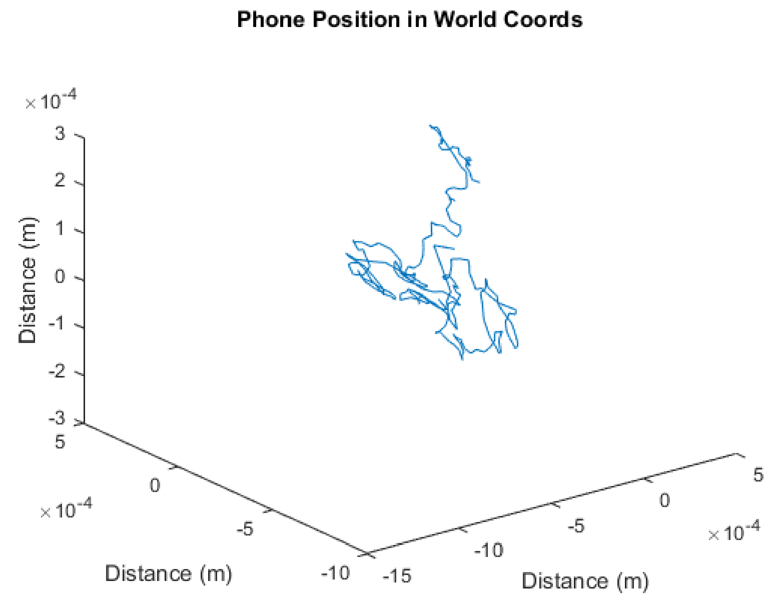

Figure 22: Phone position calculated from IMU.

Unfortunately, the results of this effort did not contribute to calculating the position of the camera and help increase the accuracy of the SfM algorithm. Figure 22 depicts a well-known phenomenon known as velocity random walk, or randomly walking bias, which is when the calculated velocity and position appear to randomly walk or change due to IMU bias [21].

Chapter 4

Conclusions

In the event the original three-dimensional computer model is unavailable, reverse engineering techniques are used to create the needed geometry. Traditionally this has been done using coordinate measuring machines, laser scanners, physical measuring, or some combination of these processes. Recently, photogrammetry has emerged as a method to create accurate reconstruction of relatively small scale items for reverse engineering.

A base line evaluation of the photogrammetric method was the first step; evaluating the accuracy of two purchased software packages. Eos Systems PhotoModeler and Autodesk Remake were used to create models for three different types of objects. The first object was meant as a baseline and was a simple 76.2 mm cube. The second object was also a 76.2 mm cube but contained subtracted conics and hemispheres to induce shadows. The final object was a thin, long object with complex curves and cuts. This object was meant to be the most challenging.

Each of the three objects was selected to test how the software packages would react to different features. Both software packages performed well, consistently producing results that were within 2.5 mm of the original design. Unfortunately, no real statistical significance could be found when comparing the results of objects 1, 2, and 3 from the same software. Remake consistently performed better, and the worst tolerance model was $\pm^{1.6688}_{1.0084}\,mm$.

The biggest drawback of both PhotoModeler and Remake was a lack of scaling. In order to produce a metric model, a scale had to be included by the user, thus leading to potential inaccuracies. In an effort to remove this variable, a SfM algorithm was developed using MATLAB. The original task was simply reconstructing a scene from two images. Once successful, the position of the camera was tracked and this data was also utilized in the algorithm. By including information about the camera position, the output of the model was true to size. While results were preliminary, the face of the 76.2 mm cube was calculated to be 76.3 mm by the SfM algorithm.

In an effort to increase the accuracy further, a tracking system was developed using an IMU. Unfortunately, this goal was unsuccessful. Due to inaccuracies and bias in the IMU, computing a precise location is not possible.

While the goal of having a single system capable of taking pictures and tracking position was unsuccessful, the overall goal of the thesis was accomplished. The current available software was evaluated to a high level of accuracy across a broad range of objects with various types of features. Furthermore, the goal of creating a SfM algorithm that produced a metric reconstruction without a scaling being set by the user was successfully implemented.

Chapter 5

Future Work and Areas for Expansion

In order to simplify and test the proof of concept for an improved SfM algorithm, two main assumptions were made. The first assumption was the user has access to an easily calibratable camera and the second was a known camera position. Both of these assumptions are not necessarily accurate and can be improved upon in the future.

Phone Camera Calibration

One of the most important steps in calibrating a camera is maintaining a constant focal length on the lens. In addition, keeping the aperture at a constant diameter is also important [12]. If the transition is made from a DSLR to the camera on a phone, these assumptions may be difficult to keep. Many phone cameras do not allow the user the ability to change the focus point with hardware, but instead use software techniques to change the focus. The effects of this have not been investigated on the quality of the calibration. In addition, DSLR cameras typically use glass lenses that are well manufactured, helping to reduce distortion. The differences between Figure 15 and Figure 16 is a perfect example of this. Most cell phone camera lenses are plastic, which have worse distortion properties. In addition, the camera calibration matrix can be more likely to change overtime with a lower quality lens. In the future, work achieving a calibration matrix for a phone camera and comparing the results to a DSLR could be completed.

Advanced Accelerometer Modeling and Improved Position Tracking

While using a cell phone's IMU to accurately calculate position was unsuccessful, other research has been done on how to improve this process. Typically, a real IMU deviates from an ideal IMU due to bias, scaling, and noise [19]. In addition, methods have been developed to measure bias stability, and many of the modeling parameters needed are often included in an IMU's data sheet [22]. While including a more accurate model of the IMU was beyond the scope of this thesis, knowing the noise, bias, and stability variables of the IMU used could produce accurate position data that could be used in the SfM algorithm. If both advanced accelerometer modeling and phone camera calibration are successful, then a smart phone alone could perform metric scene reconstruction.

Appendix A

Structure from Motion with 2 Views – Metric Output

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                      Pennsylvania State University
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                              COPYRIGHT 2017
%                      Pennsylvania State University
%                        University Park, PA 16802
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FILENAME: SFM_2_pics
%
% DESCRIPTION: This code loads two pictures and creates a point cloud. To
%              do this, it first calculates the orientation and pose of the
%              second camera relative to the first. The point cloud that is
%              output is not metric.
%
% REFERENCES: 1.  The MathWorks, "Structure from Motion," The MathWorks Inc
%                 2017. [Online]. Available:
%                 https://www.mathworks.com/help/visiion/ug/structure-from
%                 motion.html
%
%             2.  The MathWorks, "Structure from Motion from Two Views," The
%                 MathWorks Inc., 2017. [Online]. Available:
%                 https://www.mathworks.com/help/visiion/ug/structure-from-
%                 motion-from-two-views.html
%
% DATE                 AUTHOR                 REVISION
% 04-APRIL-2017       BENJAMIN SATTLER         INITIAL RELEASE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% INPUTS: Provide description of script inputs if applicable.
%         1. Image 1        :  First image used
%         2. Image 2        :  Second image used
%         3. Camera Matrix  :  Camera calibration
%
% OUTPUTS: Provide description of script outputs if applicable.
%         1. PtCloud        :  Calculated point cloud of the world
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Clean Workspace
% This section of the code simply closes all figures, clears all variables,
% and clears the command window

clear all, close all, clc

```

```
%% Load the Images
% This section of the code uses a directory to create an image set that
% imports the two input images

images = imageDatastore('C:\Users\bzs52\Documents\Senior Year\Dr. Basu\SFM');
I1=readimage(images,1);                                 % Read image 1
I2=readimage(images,2);                                 % Read image 2
figure                                                  % Create a figure
imshowpair(I1,I2,'montage');                            % Show the images
title('Original Images');                               % Title the figure

%% Load Camera Params
% This section of code loads the camera parameters of a calibrated camera

load ('C:\Users\bzs52\Documents\Senior Year\Dr. Basu\Cal Pics\EoS 6D
big\matlab.mat');

%% Remove Lens Distortion
% This section of code removes distortion from the images

I1=undistortImage(I1,cameraParams);                     % Undistort image 1
I2=undistortImage(I2,cameraParams);                     % Undistort image 2
figure                                                  % Create figure
imshowpair(I1,I2,'montage');                            % Show the images
title('Undistorted Images');                            % Title the figure

%% Find Points in Both Images for Epipolar Points
% This section of code finds trackable points, only used to calculate the
% epipolar points to find the Essential Matrix

imagePoints1 = detectMinEigenFeatures(rgb2gray(I1),'MinQuality',0.05);
figure                                                  % Create figure
imshow(I1,'InitialMagnification',50);                   % Show image
title('150 Strongest Corners from the first Image');    % Title figure
hold on
plot(imagePoints1.selectStrongest(150))                 % Plot detected
points

% Define the parameters of the point tracker
tracker=vision.PointTracker('MaxBidirectionalError',1,'NumPyramidLevels',5);
imagePoints1=imagePoints1.Location;                     % Save point locatons
initialize(tracker,imagePoints1,I1);                    % Start the point
tracker
[imagePoints2,validIdx]=step(tracker,I2);               % Use the tracker to
correlate points
matchedPoints1=imagePoints1(validIdx,:);                % Save image 1 points
if valid match
matchedPoints2=imagePoints2(validIdx,:);                % Save image 2 points
if valid match
figure                                                  % Create figure
showMatchedFeatures(I1,I2,matchedPoints1,matchedPoints2);%Show the images and
matched points
title('Tracked Features');                              % Title the figure
```

```
%% Calculate the Essential Matrix
% Using the matched points, calculate all parameters of the Essential
% Matrix

% Use the estimateEssentialMatrix to find epipolar points and a 3x3 matrix
[E, epipolarInliers]=estimateEssentialMatrix(...
    matchedPoints1,matchedPoints2,cameraParams,'Confidence',99.99);

inlierPoints1=matchedPoints1(epipolarInliers,:);        % Calc epipolar
points on image 1
inlierPoints2=matchedPoints2(epipolarInliers,:);        % Calc epipolar
points on image 2
figure                                                  % Create figure
showMatchedFeatures(I1,I2,inlierPoints1,inlierPoints2); % Show images and
epipolar points
title('Epipolar Inliers')                               % Title the figure

%% Use Essential Matrix to find Camera Orientation and Translation
% This section of code uses the Essential Matrix calculated in the previous
% section to find the relative orientation and transpose of the second
% camera to the first camera

[orient, loc]=relativeCameraPose(E,cameraParams,inlierPoints1,inlierPoints2);
loc= loc*2.9364;                                        % 2.9364 is the
distance the camera
                                                        % moved between each
image.
                                                        % Calc from
experimental setup

%% Reconstruct the Scene
% This section of code first removes the outer edge of the image, then
% finds new points to track. Since camera position is already known, the
% quality of the points can be reduced to help find more points. Finally,
% the 3D position of the points is calculated

roi=[40,40,size(I1,2)-40,size(I1,1)-40];               % Define the section
of the image to use
% Detect new points. As mentioned above, the quality of points can be lower
imagePoints1 = detectMinEigenFeatures(rgb2gray(I1), 'ROI', roi, ...
    'MinQuality', 0.001);
% Define the parameters of the point tracker
tracker = vision.PointTracker('MaxBidirectionalError', 1, 'NumPyramidLevels',
5);
imagePoints1 = imagePoints1.Location;                  % Save the point
locations
initialize(tracker, imagePoints1, I1);                 % Start the image
tracker
[imagePoints2, validIdx] = step(tracker, I2);          % Use the tracker to
correlate points
matchedPoints1 = imagePoints1(validIdx, :);            % Save image 1 points
if valid match
```

```
matchedPoints2 = imagePoints2(validIdx, :);             % Save image 2 points
if valid match
camMatrix1=cameraMatrix(cameraParams,eye(3),[0 0 0]);  % Create first camera
matrix
[R,t]=cameraPoseToExtrinsics(orient,loc);               % Calculate
orientation and pose of cam 2
camMatrix2=cameraMatrix(cameraParams,R,t);              % Create second camera
matrix
% Use triangulation to find the 3D location of each point
points3D = triangulate(matchedPoints1, matchedPoints2,camMatrix1,camMatrix2);
numPixels=size(I1,1)*size(I1,2);                        % Calc total number of
pixels
allColors=reshape(I1,[numPixels,3]);                    % Put all points into
a MxN matrix
% save the RGB number of all the pixels
colorIdx = sub2ind([size(I1, 1), size(I1, 2)], round(matchedPoints1(:,2)),
... round(matchedPoints1(:, 1)));
color = allColors(colorIdx, :);                         % Save color to point
ptCloud=pointCloud(points3D,'Color',color);             % Create the point
cloud

%% Display the Point Cloud
% Visualize the camera locations and orientations along with the world

cameraSize = 0.3;                                       % Set the camera size
figure                                                  % Create the figure
% show the first camera
plotCamera('Size', cameraSize, 'Color', 'r', 'Label', '1', 'Opacity', 0);
hold on
grid on
% show the second camera
plotCamera('Location', loc, 'Orientation', orient, 'Size', cameraSize, ...
    'Color', 'b', 'Label', '2', 'Opacity', 0);

% Show the point cloud
pcshow(ptCloud, 'VerticalAxis', 'y', 'VerticalAxisDir', 'down', ...
    'MarkerSize', 45);
camorbit(0, -30);                                       % Rotate the plot
camzoom(1.5);                                           % Zoom in on the plot
xlabel('x-axis (in)');                                  % Label the x-axis
ylabel('y-axis (in)');                                  % Label the y-axis
zlabel('z-axis (in)');                                  % Label the z-axis
title('Reconstructed View from Known Transpose');       % Title the figure
```

Appendix B

Structure from Motion with Multiple Views

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                      Pennsylvania State University
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                              COPYRIGHT 2017
%                      Pennsylvania State University
%                        University Park, PA 16802
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FILENAME: SFM_mult_pics
%
% DESCRIPTION: This code loads two pictures and creates a point cloud. To
%              do this, it first calculates the orientation and pose of the
%              second camera relative to the first. The point cloud that is
%              output is not metric.
%
% REFERENCES: 1.  The MathWorks, "Structure from Motion," The MathWorks Inc
%                 2017. [Online]. Available:
%                 https://www.mathworks.com/help/visiion/ug/structure-from
%                 motion.html
%
%             2.  The MathWorks, "Structure from Motion from Multiple Views,"
%                 The MathWorks Inc., 2017. [Online]. Available:
%                 https://www.mathworks.com/help/visiion/ug/structure-from
%                 motion-from-multiple-views.html
%
% DATE                  AUTHOR                 REVISION
% 04-APRIL-2017       BENJAMIN SATTLER         INITIAL RELEASE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% INPUTS: Provide description of script inputs if applicable.
%         1. Images         :  Directory of Image Pathway
%         2. Camera Matrix  :  Camera calibration
%
% OUTPUTS: Provide description of script outputs if applicable.
%         1. PtCloud        :  Calculated point cloud of the world
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Clean Workspace
% This section of the code simply closes all figures, clears all variables,
% and clears the command window

clear all, close all, clc

%% Load the images
```

```
% This section of the code uses a directory to create an image set that
% imports the two input images

imds = imageDatastore('C:\Users\bzs52\Documents\Senior Year\Dr.
Basu\SFM\Block Pics');
images=cell(1,numel(imds.Files));   % create empty array for images
for i=1:numel(imds.Files)           % for loop to run through all images
    I=readimage(imds,i);            % read in image
    images{i}=rgb2gray(I);          % convert image to grayscale and save in
array
end

%% Load Camera Params
% This section of code loads the camera parameters of a calibrated camera

load ('C:\Users\bzs52\Documents\Senior Year\Dr. Basu\Cal Pics\EoS 6D
big\matlab.mat');

%% Remove Lens Distortion
% This section of code removes distortion from the images

I=undistortImage(images{1},cameraParams);

%% Find Point Correspondences Between the Images
% This section of code finds trackable points

roi = [50,50,size(I, 2)-2*50,size(I, 1)- 2*50];                    % Set region
of interest
prevPoints=detectSURFFeatures(I, 'NumOctaves', 8, 'ROI', roi);  % detect
points
prevFeatures = extractFeatures(I, prevPoints, 'Upright', true); % Extract
features
vSet = viewSet;                                                  % Create set
for other views
viewId = 1;                                                      % First view
ID
% Add in the information from the first image to the set
vSet = addView(vSet, viewId, 'Points', prevPoints, 'Orientation', ...
    eye(3, 'like', prevPoints.Location), 'Location', ...
    zeros(1, 3, 'like', prevPoints.Location));
%% Add the rest of the views
% This section of code brings in the rest of the images and detects the
% points. It also matches the points to the previous feature

k=0;
for i = 2:numel(images)                                  % Loop through every
image
    I=undistortImage(images{i},cameraParams);            % Undistort the image
    currPoints=detectSURFFeatures(I, 'NumOctaves', 8, 'ROI', roi);% Detect
points
    currFeatures =extractFeatures(I, currPoints, 'Upright', true);% Detect
features
```

```
    indexPairs=matchFeatures(prevFeatures, currFeatures, ...      % Find
matches
        'MaxRatio', .7, 'Unique',  true);
    matchedPoints1=prevPoints(indexPairs(:, 1));      % Save match if valid
in image 1
    matchedPoints2=currPoints(indexPairs(:, 2));      % Save match if valid
in image 2
    % Calculate the orientation and location of one camera to the previous
    [relativeOrient,relativeLoc,inlierIdx]=helperEstimateRelativePose(...
        matchedPoints1,matchedPoints2,cameraParams);
    vSet=addView(vSet,i,'Points',currPoints);         % Add the points to the
set
    vSet=addConnection(vSet,i-1,i,'Matches',indexPairs(inlierIdx,:));
    prevPose=poses(vSet,i-1);                         % Look at previous pose
    prevOrientation=prevPose.Orientation{1};          % Look at the previous
orientation
    prevLocation=prevPose.Location{1};                % Look at the previous
transpose
    orientation=relativeOrient*prevOrientation;       % Calculate the new
orientation
    location=prevLocation+relativeLoc*prevOrientation;% Calc new location
    vSet=updateView(vSet,i,'Orientation',orientation, ...
        'Location',location);
    tracks = findTracks(vSet);                        % Find points in all
views
    camPoses = poses(vSet);                           % Load camera poses
    % Calculate the world points
    xyzPoints = triangulateMultiview(tracks, camPoses, cameraParams);
    % Use Bundle Adjustment to account for erros
    [xyzPoints, camPoses, reprojectionErrors] = bundleAdjustment(xyzPoints,
...
        tracks, camPoses, cameraParams, 'FixedViewId', 1, ...
        'PointsUndistorted', true);
    vSet = updateView(vSet, camPoses);                % Save adjusted cam
poses
    prevFeatures = currFeatures;                      % Make current features
previous feats
    prevPoints   = currPoints;                        % Make current points
previous points
    k=k+1;                                            % Update k
end

%% Display Camera Poses
% This section of code simply displays the camera positions and points used
% to calculate those positions
camPoses = poses(vSet);               % Save the camera positions
figure;                               % Create a figure
plotCamera(camPoses, 'Size', 0.2);    % Plot the cameras
hold on                               % Keep the plot up
goodIdx = (reprojectionErrors < 5);   % Calculate if good point or not
xyzPoints = xyzPoints(goodIdx, :);    % Save the valid XYZ points
% Display valid points
pcshow(xyzPoints, 'VerticalAxis', 'y', 'VerticalAxisDir', 'down', ...
    'MarkerSize', 45);
```

```
grid on                              % Display a grid
hold off                             % Turn the hold off
loc1 = camPoses.Location{1};         % Identify cam position one
xlim([loc1(1)-5, loc1(1)+4]);        % Set the x axis
ylim([loc1(2)-5, loc1(2)+4]);        % Set the y axis
zlim([loc1(3)-1, loc1(3)+20]);       % Set the z axis
camorbit(0, -30);                    % Change orientation of plot
title('Camera Location');            % Title the plot

%% Compute Dense Reconstruction
% Go through the images again. This time detect a dense set of corners,
% and track them across all views using vision.PointTracker.

I=undistortImage(images{1},cameraParams);                 % Undistort first
image
prevPoints=detectMinEigenFeatures(I,'MinQuality',0.001);% Detect points
% Create the point tracker
tracker=vision.PointTracker('MaxBidirectionalError',1,'NumPyramidLevels', 6);
prevPoints=prevPoints.Location;                           % Set first points
initialize(tracker,prevPoints,I);                         % Init tracker
vSet=updateConnection(vSet,1,2,'Matches',zeros(0, 2));   % Make part of set
vSet=updateView(vSet,1,'Points',prevPoints);              % Store points in set

for i = 2:numel(images)                                   % Loop through all
images
    I=undistortImage(images{i},cameraParams);             % Undistort current
pic
    [currPoints,validIdx]=step(tracker,I);                % Track the points
    if i<numel(images)                                    % Check where in loop
        vSet=updateConnection(vSet,i,i+1,'Matches',zeros(0, 2)); % Zero
Images
    end
    vSet=updateView(vSet,i,'Points',currPoints);          % Update the set
    matches=repmat((1:size(prevPoints,1))',[1,2]);        % Tile matches matrix
    matches=matches(validIdx,:);                          % Save valid matches
    vSet=updateConnection(vSet,i-1,i,'Matches',matches);% Save matches in set
end

tracks=findTracks(vSet);                                  % Track points in the
set
camPoses=poses(vSet);                                     % Read camera poses
xyzPoints = triangulateMultiview(tracks, camPoses,...     % Calc XYZ points
    cameraParams);
% Use bundleAdjustment to reduce errors
[xyzPoints, camPoses, reprojectionErrors] = bundleAdjustment(...
    xyzPoints, tracks, camPoses, cameraParams, 'FixedViewId', 1, ...
    'PointsUndistorted', true);

% Get color from images
for i=1:length(tracks)
    matches(i,:)=[double(tracks(1,i).Points(1,1))
double(tracks(1,i).Points(1,2))];
end
```

```
numPixels=size(I, 1)*size(I, 2);                        % Calc total pix
number
Q=readimage(imds,1);                                    % Read RGB value
allColors = reshape(Q, [numPixels, 3]);                 % Save the color IDs
colorIdx = sub2ind([size(I, 1), size(I, 2)], round(matches(:,2)), ...
    round(matches(:, 1)));
color = allColors(colorIdx, :);
ptCloud=pointCloud(xyzPoints,'Color',color);            % Add color to pt
cloud

%% Display Point Cloud
% Visualize the camera locations and orientations along with the world
figure;                                                 % Create figure
plotCamera(camPoses, 'Size', 0.2);                      % Plot cameras
hold on                                                 % Keep plot
goodIdx = (reprojectionErrors < 5);                     % Calc if good point
pcshow(ptCloud, 'VerticalAxis', 'y', ...                % Disp point cloud
    'VerticalAxisDir', 'down', 'MarkerSize', 45);
grid on                                                 % Plot with grid
hold off                                                % Turn hold off
loc1 = camPoses.Location{1};                            % Identify camera 1
xlim([loc1(1)-5, loc1(1)+4]);                           % x axis
ylim([loc1(2)-5, loc1(2)+4]);                           % y axis
zlim([loc1(3)-1, loc1(3)+20]);                          % z axis
camorbit(0, -30);                                       % Define orientation
title('Unscaled Scene Reconstruction from Multiple Views');
```

Appendix C

Position Tracking Algorithm

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                      Pennsylvania State University
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                              COPYRIGHT 2017
%                      Pennsylvania State University
%                        University Park, PA 16802
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FILENAME: from_file_gravity_removed
%
% DESCRIPTION: This code loads data recorded using the MATLAB mobile
%              application. The end result is plots of the phone's
%              acceleration in body coordinates as well as the position of
%              the phone in both body and global coordinates
%
% REFERENCES: 1.  C. D. Monaco, Detecting the Instability of Oncoming
%                 Vehicles Using Optical Flow and Map-Based Context,
%                 University Park: Penn State Electronic Theses and
%                 Dissertations for Graduate School, 2016.
%
% DATE                  AUTHOR                REVISION
% 04-APRIL-2017        BENJAMIN SATTLER         INITIAL RELEASE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% INPUTS: Provide description of script inputs if applicable.
%         1. Acceleration :  System acceleration [m/s^2]
%         2. Orientation  :  System roll pitch and yaw [degrees]
%
% OUTPUTS: Provide description of script outputs if applicable.
%         1. NoG          :   Accel data w/ gravity removed [m/s^2]
%         2. NoG_zeroed   :   Accel data w/ gravity & bias removed [m/s^2]
%         3. uvw          :   Velocity in body coordinates [m/s]
%         4. xyz          :   Position in body coordinates [m]
%         5. XYZ_vel      :   Velocity in global coordinates [m/s]
%         6. XYZ          :   Position in global coordinates [m]
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Clean Workspace
% This section of the code simply closes all figures, clears all variables,
% and clears the command window

clear all, close all, clc

%% Load Data
% This section of the code loads the .mat file recorded from Matlab mobile

```

```
load 'stationary_phone.mat'      % Load the data file
acc=acc';                        % Transpose the data
o=o';                            % Transpose the data
o(2,:)=-o(2,:);                  % Convert to SAE coords

%% Remove Gravity
% This section calculates the actual acceleration of the phone by removing
% the measured gravity

g=9.81;                          % [m/s]
NoG=zeros(3,length(t));          % init No gravity variable

for i=1:length(t)                % in this loop, calc dt, and remove gravity
    if(i<length(t))              % check the quantity of i
        dt(i+1)=t(i+1)-t(i);     % calculated the change in time
    end
    if(i==1)                     % check the quantity of i
        NoG(:,i)=[0;0;0];        % Set initial condition to 0
    else
        % Use the roll, pitch, and yaw from MATLAB Mobile to remove effect
        % of gravity. Modified from [1]
        NoG(:,i)=acc(:,i)+[g*sind(o(3,i));g*cosd(o(3,i)).*sind(o(2,i));-
g*cosd(o(3,i)).*cosd(o(2,i))];
    end
end

%% Remove Bias
% This section of code averages the entire calculated acceleration to find
% the offset bias. This bias is then removed from the data

bias=mean(NoG');               % calculate the bias of the IMU
NoG_zeroed=NoG-bias';          % subtract bias from the data

%% Filter the Data
% This section of code filters the data using a simple moving average

coeff50hz = ones(1, 50)*(1/50);
avgNoG_zeroed = filter(coeff50hz, 1, NoG_zeroed);

%% Calculate Body
% In this section of the code, the change of time, and dt between is
% measurement is calculated. dt is then used to calculate the body velocity
% and the position. A quaternion is used to transform the data from body to
% global coordinates.

dt=zeros(length(t),1);           % init dt var
uvw=zeros(3,length(t));          % init uvw var
for i=1:length(t)                % Calculate uvw,xyz,quatern,& world coords
    if(i==1)                     % check quantity of i
        uvw(:,i)=[0;0;0];        % set initial condition to 0
    else
        % Lines 41,42, and 45 modified from [1]
        uvw(:,i)=avgNoG_zeroed(:,i)*dt(i)+uvw(:,i-1);        % calc velocity
```

```
        xyz(:,i)=0.5*(uvw(:,i)+uvw(:,i-1))*dt(i)+uvw(:,i-1); % body position
        q=angle2quat(o(1,i),o(2,i),o(3,i),'zyx');            % quatern vector
        XYZ_vel(:,i)=quatrotate(quatinv(q),uvw(:,i)');       % World velocity
        XYZ(:,i)=0.5*(XYZ_vel(:,i)+XYZ_vel(:,i-1))*dt(i)+XYZ_vel(:,i-1); % world coords
    end
end

%% Plot Data
% This section of code simply plots the data into relevant figures

h1=figure;                    % Create a figure
plot(t,NoG_zeroed(3,:))       % Plot the Z accel with gravity and bias removed vs time
title('Accel in Z')
hold on
plot(t,avgNoG_zeroed(3,:))    % Plot the filtered accel data on same graph
movegui(h1,'northwest')       % Move graph to top left corner of screen

h2=figure;                    % Create a figure
plot(t,NoG_zeroed(2,:))       % Plot Y accel with gravity and bias removed vs time
title('Accel in Y')
hold on
plot(t,avgNoG_zeroed(2,:))    % Plot the filtered accel data on same graph
movegui(h2,'north')           % Move graph to top center of screen

h3=figure;                    % Create a figure
plot(t,NoG_zeroed(1,:))       % Plot X accel with gravity and bias removed vs time
title('Accel in X')
hold on
plot(t,avgNoG_zeroed(1,:))    % Plot the filtered accel data on same graph
movegui(h3,'northeast')       % Move graph to the top right corner of screen

h4=figure;                    % Create a figure
plot(XYZ(1,:),XYZ(2,:))       % Plot the XY position of phone in world coordinates
title('Phone Position in World Coords')
movegui(h4,'southwest')       % Move graph to bottom left corner of screen

h5=figure;                    % Create a figure
plot(xyz(1,:),xyz(2,:))       % Plot xy position of phone in body coordinates
title('Phone Position in Body Coords')
movegui(h5,'south')           % Move graph to the bottom center of screen

h6=figure;                    % Create a figure
plot3(XYZ(1,:),XYZ(2,:),XYZ(3,:))   % Plot 3D position of phone in world coords
title('Phone Position in World Coords')
movegui(h6,'southeast')       % Move graph to bottom left corner of screen
```

BIBLIOGRAPHY

[1] E. P. Baltsavias, "A Comparison Between Photogrammetry and Laser Scanning," *ISPRS Journal of Photogrammetry and Remote Sensing,* vol. 54, no. 2-3, pp. 83-94, 1999.

[2] University of Arkansas, "Digital Photogrammetry," Geospatial Modeling & Visualization, 2017. [Online]. Available: http://gmv.cast.uark.edu/photogrammetry/. [Accessed 3 August 2016].

[3] S. Gonizzi Barsanti, F. Remondino and D. Visintini, "Photogrammetry and Laser Scanning for Archaeological Site 3D Modeling - Some Critical Issues," *CEUR-WS,* vol. 948, 2012.

[4] Eos Systems Inc., "PhotoModeler Product Overview," Eos Systems Inc., 2016. [Online]. Available: http://www.photomodeler.com/products/default.html. [Accessed 02 April 2017].

[5] Autodesk Inc., "Autodesk Remake," Autodesk Inc., 2015. [Online]. Available: https://remake.autodesk.com/about. [Accessed 02 April 2017].

[6] R. Hartley and A. Zisserman, Multiple View Geometry in Computer Vision, Cambridge: Cambridge University Press, 2000.

[7] Q. Luong and O. Faugeras, "The Fundamental Matrix: Theory, Algorithms, and Stability Analysis," *International Joumal of Computer Vision,* vol. 17, pp. 43-75, 1996.

[8] The MathWorks, "Structure from Motion," The MathWorks Inc., 2017. [Online]. Available: https://www.mathworks.com/help/vision/ug/structure-from-motion.html. [Accessed 02 April 2017].

[9] A. A. Behrouzi and D. A. Kuchma, "Instruction Manual: Photogrammetry as a Non-Contact Measuring System in Large Scale Structural Testing," *Network for Earthquake Engineering Simulation*, 2014.

[10] EoS Systems Inc., "Factors Affecting Accuracy in Photogrammetry," PhotoModeler, 2017. [Online]. Available: http://info.photomodeler.com/blog/kb/factors_affecting_accuracy_in_photogramm/. [Accessed 02 April 2017].

[11] I. Ihrke, K. N. Kutulakos, H. P. A. Lensch, M. Magnor and W. Heidrich, "Transparent and Specular Object Reconstruction," in *EuroGraphics*, Crete, 2008.

[12] P. Corke, Robotics, Vision and Control, Berlin: Springer, 2013.

[13] PhotoModeler, "What are the computer requirements to run PhotoModeler," EoS Systems Inc., 2017. [Online]. Available: http://info.photomodeler.com/blog/kb/what_are_the_computer_requirements_to_run/. [Accessed 07 April 2017].

[14] AutoDesk, "System requirements for Autodesk ReMake 2017," Autodesk Knowledge Network , 15 May 2015. [Online]. Available: https://knowledge.autodesk.com/support/remake/troubleshooting/caas/sfdcarticles/sfdcarticles/System-requirements-for-Autodesk-ReMake-2017.html. [Accessed 02 April 2017].

[15] The MathWorks, "Single Camera Calibration App," The MathWorks, Inc., 2017. [Online]. Available: https://www.mathworks.com/help/vision/ug/single-camera-calibrator-app.html. [Accessed 02 April 2017].

[16] The MathWorks, "relativeCameraPose," The MathWorks Inc., 2017. [Online]. Available: https://www.mathworks.com/help/vision/ref/relativecamerapose.html. [Accessed 02 April 2017].

[17] The MathWorks, "Structure From Motion From Multiple Views," The MathWorks Inc., 2017. [Online]. Available: https://www.mathworks.com/help/vision/examples/structure-from-motion-from-multiple-views.html. [Accessed 03 April 2017].

[18] B. Triggs, P. McLauchlan, R. Hartley and A. Fitzgibbon, "Bundle Adjustment — A Modern Synthesis," in *International Workshop on Vision Algorithms*, Corfu, 1999.

[19] C. D. Monaco, Detecting the Instability of Oncoming Vehicles Using Optical Flow and Map-Based Context, University Park: Penn State Electronic Theses and Dissertations for Graduate School, 2016.

[20] H. D. Curtis, Orbital Mechanics for Engineering Students, Waltham: Elsevier Ltd., 2014.

[21] N. El-Sheimy, H. Hou and X. Niu, "Analysis and Modeling of Inertial Sensors Using Allan Variance," *IEEE Transactions on Instrumentation and Measurement,* vol. 57, no. 1, pp. 140-149, 2008.

[22] O. J. Woodman, "An Introduction to Inertial Navigation," Univeristy of Cambridge, 2007.

ACADEMIC VITA

Academic Vita of Benjamin Jacques Sattler

EDUCATION

Bachelor of Science in Mechanical Engineering Spring 2017
The Pennsylvania State University, University Park, PA
Schreyer Honors College: Industrial Engineering
Dean's List: FA12, SP13, FA13, FA14

THESIS

Three-Dimensional Model Reconstruction from Images
Using Position Tracking and Structure from Motion
Thesis Supervisor: Dr. Saurabh Basu

CAREER EXPERIENCE

SpaceX – Propulsion Production FA 15 -SP 16
Design Tooling to Increase Efficiency of Manufacturing Rocket Engines

- Interface with individuals designing flight components to meet manufacturing requirements
- Design tools varying from shipping rocket stages, to qualification stands, to assembly aids
- Define load cases and perform analysis on all designs ensuring safe operation
- Create engineering drawings for fabrication

Roush – Engine and Transmission Calibration SP 15 –FA 15

- Calibration and mapping of engine dyno, chassis dyno, and in-vehicle engines
- Improving accuracy of inferred engine parameters to decrease number of needed sensors
- Track-side support during vehicle testing to diagnose problems as they arose

RESEARCH EXPERIENCE

Undergraduate Student Research
BATTERY Lab SP 14 –SP 15
The Pennsylvania State University, University Park, PA

- Engage in systems engineering research under Timothy Cleary
- Design of large vehicle battery pack and integration
- Crash safety and effects of shock and vibration on battery lifespan

Hydrogen Hybrid Research Lab FA 12 – SP 15
The Pennsylvania State University, University Park, PA

- Engage in interdisciplinary research under Dr. Joel Anstrom
- Restoring a GM Ev1 and updating to state-of-the-art technology
- Organizing renewable vehicle competition- built charging station for 12 electric vehicles

NSF-Funded Undergraduate Research (REU) Summer 2013
Modeling of Ignition Time and Temperature of Biodiesel Surrogates
University of Connecticut, Storres, CT

- Engage in chemical and thermodynamics research under Dr. Tianfeng Lu
- Computational combustion simulation of biodiesel. MATLAB based models
- Use of CHEMKIN database and software package to find ignition timing and temperature

LEADERSHIP

EcoCAR 3 Team Leader/Project Manager 2014/2015
Penn State Advanced Vehicle Team

- Lead and review writing of team reports and assist with all necessary waivers
- Guide and focus team to meet deadlines and deliverables dictated by competition
- Create control algorithms which were implemented into team's vehicle
- Use of MATLAB and Simulink to make accurate vehicle models for SIL and HIL setups
- Logging and reading of in-vehicle CAN messages for vehicle refinement and calibration
- Technical and software skills: welding, machining, MATLAB, Simulink, Siemens NX

PUBLICATIONS

Design and Implementation of a Series Plug-In Hybrid Electric Vehicle for the EcoCAR 2 Competition (SAE International)
Paper #: 2014-01-2909 Published 2014-10-13

Motivational Tactics and Techniques for Largely Volunteer-Based Organizations (2015 PMI Global Conference Proceedings)