High Resolution Low-Bandwidth Real-Time Reconnaissance using Structure from Motion with Planar Homography Estimation

Christian M.A. Arnold

Follow this and additional works at: https://scholar.afit.edu/etd

Part of the Aeronautical Vehicles Commons, and the Digital Communications and Networking Commons

Recommended Citation

https://scholar.afit.edu/etd/2243

This Thesis is brought to you for free and open access by the Student Graduate Works at AFIT Scholar. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of AFIT Scholar. For more information, please contact richard.mansfield@afit.edu.
HIGH RESOLUTION LOW-BANDWIDTH REAL-TIME RECONNAISSANCE USING STRUCTURE FROM MOTION WITH PLANAR HOMOGRAPHY ESTIMATION

THESIS

Christian M. A. Arnold, 2nd Lieutenant, USAF
AFIT-ENG-MS-19-M-007

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.
The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, the United States Department of Defense or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.
HIGH RESOLUTION LOW-BANDWIDTH REAL-TIME RECONNAISSANCE
USING STRUCTURE FROM MOTION WITH PLANAR HOMOGRAPHY
ESTIMATION

THESIS

Presented to the Faculty
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Computer Science

Christian M. A. Arnold, BS
2nd Lieutenant, USAF

March 2019

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.
HIGH RESOLUTION LOW-BANDWIDTH REAL-TIME RECONNAISSANCE
USING STRUCTURE FROM MOTION WITH PLANAR HOMOGRAPHY
ESTIMATION

THESIS

Christian M. A. Arnold, BS
2nd Lieutenant, USAF

Committee Membership:

Scott L. Nykl, Ph.D.
Chair

Scott R. Graham, Ph.D.
Member

Robert C. Leishman, Ph.D.
Member
Abstract

Aerial real-time surveillance exists in a paradigm balancing the constraints of delivering high quality data and transporting data quickly. Typically, to have more of one, sacrifices must be made to the other. This is true of the environment in which an Unmanned Aerial Vehicle (UAV) operates, where real-time communication may be done through a low-bandwidth satellite connection resulting in low-resolution data, and serves as the primary limiting factor in all intelligence operations. Through the use of efficient computer vision techniques, we propose a new Structure from Motion (SfM) method capable of compressing high-resolution data, and delivering that data in real-time. Specifically demonstrating more than a 90% reduction of original video imagery size while operating at 4 Hz, which equates to an 80x computation time speed-up compared to traditional SfM methods, with an added benefit of presenting the original 2D intelligence data as a 3D virtual model.
# Table of Contents

Abstract ........................................................................................................ iv

List of Figures ............................................................................................. vii

List of Tables .............................................................................................. xi

0.1 Introduction .......................................................................................... 1
  Aerial Surveillance .................................................................................. 1
  Application of Real-Time Structure from Motion .................................. 2
  Contributions ......................................................................................... 4

0.2 Background ......................................................................................... 6
  Technical Overview .............................................................................. 6
  Processing Vision .................................................................................. 6
  Digital Image Representation .............................................................. 6
  The Projection Matrix ........................................................................... 7
  Handling Camera Distortion ................................................................ 9
  Types of Distortion .............................................................................. 10
  Quartic Model for Radial Distortion .................................................... 11
  Camera Calibration ............................................................................... 12
  Finding Image Features ....................................................................... 13
  Feature Detectors ............................................................................... 13
  Essential Matrix .................................................................................. 16
  Using Keypoints ................................................................................... 17
  Random Sampling Consensus (RANSAC) to Find Essential Matrix 19
  Extracting Depth Information .............................................................. 20
  Epipolar Geometry ............................................................................ 20
  Dense Reconstruction .......................................................................... 22
  Mesh Model Creation .......................................................................... 24
  Mesh Texturing ................................................................................... 27
  Model Size and Orientation ............................................................... 27
  Homography Transformation ............................................................... 30
  Related Work ..................................................................................... 31
  SFM Design Decisions ....................................................................... 32
  Other Methods Useful to Redesigning SfM .......................................... 33
  Software Tools for Computer Vision and SfM .................................... 34

0.3 Methodology ....................................................................................... 36
  Structure from Motion with Planar Homography ............................... 36
  Estimation ......................................................................................... 36
  Operating Assumptions ....................................................................... 36
  Virtual Environment ............................................................................ 39
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camera Calibration</td>
<td>41</td>
</tr>
<tr>
<td>Input Data Collection</td>
<td>43</td>
</tr>
<tr>
<td>Orbital Flight Path</td>
<td>43</td>
</tr>
<tr>
<td>OpenCV Point Triangulation</td>
<td>45</td>
</tr>
<tr>
<td>Plane Finding</td>
<td>54</td>
</tr>
<tr>
<td>Homography Transform and Texturing</td>
<td>59</td>
</tr>
<tr>
<td>Plane Refinement</td>
<td>61</td>
</tr>
<tr>
<td>Generating Traditional SfM Reconstruction for</td>
<td></td>
</tr>
<tr>
<td>Evaluation Criteria</td>
<td>64</td>
</tr>
<tr>
<td>0.4 Results and Discussion</td>
<td>66</td>
</tr>
<tr>
<td>Gathering Results</td>
<td>66</td>
</tr>
<tr>
<td>Hardware/Software Environment</td>
<td>66</td>
</tr>
<tr>
<td>Datasets</td>
<td>66</td>
</tr>
<tr>
<td>Discussion</td>
<td>68</td>
</tr>
<tr>
<td>Tall Building</td>
<td>70</td>
</tr>
<tr>
<td>City Block</td>
<td>71</td>
</tr>
<tr>
<td>Arab House</td>
<td>71</td>
</tr>
<tr>
<td>0.5 Conclusions and Future Work</td>
<td>86</td>
</tr>
<tr>
<td>Conclusions</td>
<td>86</td>
</tr>
<tr>
<td>Future Work</td>
<td>87</td>
</tr>
<tr>
<td>Bibliography</td>
<td>89</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>SFM Example: Input Unordered Images, Output Reconstruction</td>
<td>3</td>
</tr>
<tr>
<td>2.</td>
<td>Pinhole Camera Model [1]</td>
<td>8</td>
</tr>
<tr>
<td>3.</td>
<td>Left: No distortion, Middle: Barrel distortion, Right: Pincushion distortion [1]</td>
<td>10</td>
</tr>
<tr>
<td>4.</td>
<td>Chessboard Calibration Image</td>
<td>14</td>
</tr>
<tr>
<td>5.</td>
<td>Rotated Chessboard Calibration Image</td>
<td>14</td>
</tr>
<tr>
<td>6.</td>
<td>FAST Bresenham circle where dashed line represents a 12-point segment indicating a corner</td>
<td>16</td>
</tr>
<tr>
<td>7.</td>
<td>Array of input images to be used in SfM Pipeline</td>
<td>17</td>
</tr>
<tr>
<td>8.</td>
<td>Visual representation of all keypoints found in an image using the FAST feature detector</td>
<td>18</td>
</tr>
<tr>
<td>9.</td>
<td>Result of Symmetry Matching corresponding keypoints between two different perspectives</td>
<td>18</td>
</tr>
<tr>
<td>10.</td>
<td>Application of RANSAC to determine line of best fit</td>
<td>19</td>
</tr>
<tr>
<td>11.</td>
<td>Removal of outliers after applying RANSAC</td>
<td>20</td>
</tr>
<tr>
<td>12.</td>
<td>Epipolar Geometry</td>
<td>21</td>
</tr>
<tr>
<td>13.</td>
<td>Perspective Rectification to create parallel Epipolar lines</td>
<td>22</td>
</tr>
<tr>
<td>14.</td>
<td>Point Cloud with only ground model superimposed</td>
<td>23</td>
</tr>
<tr>
<td>15.</td>
<td>Point Cloud with full truth model superimposed</td>
<td>23</td>
</tr>
<tr>
<td>16.</td>
<td>Dense Reconstruction View 1</td>
<td>25</td>
</tr>
<tr>
<td>17.</td>
<td>Dense Reconstruction View 2</td>
<td>26</td>
</tr>
<tr>
<td>18.</td>
<td>Refined Mesh Model</td>
<td>28</td>
</tr>
<tr>
<td>19.</td>
<td>Textured SfM Model</td>
<td>28</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>20.</td>
<td>Texture Map for SfM model.</td>
<td>29</td>
</tr>
<tr>
<td>22.</td>
<td>SfM with PHE Overview</td>
<td>37</td>
</tr>
<tr>
<td>23.</td>
<td>Truth Model rendered in Aft Burner</td>
<td>40</td>
</tr>
<tr>
<td>24.</td>
<td>Detected Corners on Calibration Object</td>
<td>42</td>
</tr>
<tr>
<td>25.</td>
<td>Standard orbital flight path with camera frustums</td>
<td>44</td>
</tr>
<tr>
<td>26.</td>
<td>Example of images collected by virtual camera to be used for feature matching and plane texturing</td>
<td>44</td>
</tr>
<tr>
<td>27.</td>
<td>OpenCV Camera-Centric Coordinate System [1]</td>
<td>47</td>
</tr>
<tr>
<td>28.</td>
<td>OpenCV Camera Rotation Diagram</td>
<td>49</td>
</tr>
<tr>
<td>29.</td>
<td>Unrectified Match</td>
<td>49</td>
</tr>
<tr>
<td>30.</td>
<td>Rectified Match</td>
<td>49</td>
</tr>
<tr>
<td>31.</td>
<td>Triangulated Points without Reprojection</td>
<td>51</td>
</tr>
<tr>
<td>32.</td>
<td>Triangulated Points with some Correction</td>
<td>51</td>
</tr>
<tr>
<td>33.</td>
<td>Reprojected Points with Model</td>
<td>53</td>
</tr>
<tr>
<td>34.</td>
<td>Reprojected Points without Model</td>
<td>53</td>
</tr>
<tr>
<td>35.</td>
<td>Reprojected Points without Model</td>
<td>55</td>
</tr>
<tr>
<td>36.</td>
<td>Point cloud of one perspective before plane detection</td>
<td>57</td>
</tr>
<tr>
<td>37.</td>
<td>Point cloud of best plane detected from one perspective projection</td>
<td>57</td>
</tr>
<tr>
<td>38.</td>
<td>Before homography transformation</td>
<td>62</td>
</tr>
<tr>
<td>39.</td>
<td>After homography transformation</td>
<td>62</td>
</tr>
<tr>
<td>40.</td>
<td>Homography transforms introduce error by displaying objects behind the desired plane as though they were part of the plane</td>
<td>63</td>
</tr>
</tbody>
</table>
Figure | Page
---|---
41. | Combined rectified perspectives with differing colored pixels removed ........................................ 63
42. | Example of geopoints on truth model .......................... 69
43. | Example of geopoints on PHE reconstruction .................... 69
44. | Tall Building Scene with only the truth model visible........ 73
45. | Tall Building Scene with the truth model overlaid with the traditional SfM model ......................... 73
46. | Tall Building Scene with only the traditional SfM model visible ................................................. 74
47. | Tall Building Scene with the truth model overlaid with the PHE SfM model .................................... 74
48. | Tall Building Scene with only the PHE model visible ........... 75
49. | Tall Building Scene with the truth model overlaid with the traditional SfM model and the PHE model .......... 75
50. | Tall Building Metric Analysis of Spatial Accuracy of X-face between truth model and traditional SfM ........ 76
51. | Tall Building Metric Analysis of Spatial Accuracy of X-face between truth model and PHE .................. 76
52. | Tall Building Metric Analysis of Spatial Accuracy of Y-face between truth model and traditional SfM .......... 77
53. | Tall Building Metric Analysis of Spatial Accuracy of Y-face between truth model and PHE .................. 77
54. | City Block Scene with only the truth model visible .......... 78
55. | City Block Scene with the truth model overlaid with the traditional SfM model .................................. 78
56. | City Block Scene with only the traditional SfM model visible ....................................................... 79
57. | City Block Scene with the truth model overlaid with the PHE SfM model ...................................... 79
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>58.</td>
<td>City Block Scene with only the PHE model visible</td>
<td>80</td>
</tr>
<tr>
<td>59.</td>
<td>City Block Scene with the truth model overlaid with the traditional SfM model and the PHE model</td>
<td>80</td>
</tr>
<tr>
<td>60.</td>
<td>City Block Metric Analysis of Spatial Accuracy of X-face between truth model and traditional SfM</td>
<td>81</td>
</tr>
<tr>
<td>61.</td>
<td>City Block Metric Analysis of Spatial Accuracy of X-face between truth model and PHE</td>
<td>81</td>
</tr>
<tr>
<td>62.</td>
<td>City Block Metric Analysis of Spatial Accuracy of Y-face between truth model and traditional SfM</td>
<td>82</td>
</tr>
<tr>
<td>63.</td>
<td>City Block Metric Analysis of Spatial Accuracy of Y-face between truth model and PHE</td>
<td>82</td>
</tr>
<tr>
<td>64.</td>
<td>Arab House Scene with only the truth model visible</td>
<td>83</td>
</tr>
<tr>
<td>65.</td>
<td>Arab House Scene with the truth model overlaid with the traditional SfM model</td>
<td>83</td>
</tr>
<tr>
<td>66.</td>
<td>Arab House Scene with only the traditional SfM model visible</td>
<td>84</td>
</tr>
<tr>
<td>67.</td>
<td>Arab House Scene with the truth model overlaid with the PHE SfM model</td>
<td>84</td>
</tr>
<tr>
<td>68.</td>
<td>Arab House Scene with only the PHE model visible</td>
<td>85</td>
</tr>
<tr>
<td>69.</td>
<td>Arab House Scene with the truth model overlaid with the traditional SfM model and the PHE model</td>
<td>85</td>
</tr>
</tbody>
</table>
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Storage Size Comparison</td>
<td>70</td>
</tr>
<tr>
<td>2.</td>
<td>Computation Time Comparison</td>
<td>71</td>
</tr>
</tbody>
</table>
0.1 Introduction

Aerial Surveillance.

The history of aerial surveillance dates back to the invention of lighter than air vehicles, with an operator flying a balloon high enough to either take a rudimentary photograph or detailed sketch of the surrounding environment. While aerial reconnaissance has evolved significantly over the past 200 years, there is one component that has remained constant since its inception: the transportation of that aerial data is relatively slow compared to the collection method. This problem has only been exacerbated by the rapid growth and adoption of computer-based technologies for reconnaissance with the prolific use of Unmanned Aerial Vehicles (UAVs) and satellite remote sensing. The primary constraint in building these modern platforms in not the fidelity of the camera, but signal strength of the antenna, and the bandwidth a system is capable of transmitting data. Effectively, there is a bitrate problem in modern Intelligence, Surveillance, and Reconnaissance (ISR) pipelines.

In order to transmit a higher quality image, a system either needs a wider frequency band, an increase the signal strength, or more efficient data encoding method, as defined by the Shannon-Hartley theorem, where $B$ is the size of the frequency band, $S$ is signal strength (dB), $N$ is noise strength (dB), and $C$ is the channel capacity [3].

$$C = B \log_2(1 + \frac{S}{N})$$

The first option (increasing $B$) is limited in modern telecommunications as frequency allocation is auctioned off to the highest bidder, placing high monetary value on a finite resource [4]. Although it is a solution, it is not always viable, and costs will only increase as more industries continue to rely on specific frequency allocations. In the second option, by increasing the signal strength, the system maintains a higher Signal to Noise Ratio (SNR). However any improvement made to the SNR is applied
logarithmically according to Shannon-Hartley, and will thus have diminishing returns as the cost of higher gain antennas increases linearly. Thus historically, the greatest improvements in telecommunications have come from trying to squeeze the same amount of data in a smaller package through the use of clever encoding. As computing power increases, so does the possibility of using more complex algorithms in an effort to achieve a greater size reduction of original data.

**Application of Real-Time Structure from Motion.**

A recent method analyzed for its data size reduction abilities is the use of Structure from Motion (SfM) which has demonstrated 49%-60% size reduction from the original Intelligence, Surveillance, and Reconnaissance (ISR) images [5]. SfM is a set of computer vision algorithms commonly used to create static three-dimensional models, known as a reconstruction, from a scene using only an unordered array of detailed two-dimensional photographs. From these photographs, relative camera locations of each image are recovered, and through the use of Epipolar Geometry, depth information is extracted from the comparison of different perspectives. An example of the input and output of this process is illustrated by Figure 1.

In addition to the size reduction gained from processing images in a SfM pipeline, depth data is extracted from the original dataset that was not originally available, increasing the utility of ISR data. Although 15-25 minutes was required to reduce a data set of 90 images, it is possible to improve the computation time of SfM by exploiting some underlying assumptions about the data being imaged.

The strategic use of virtual environments and a graphical virtualization engine was vital to the development of this proposed method. The use of a virtual environment enabled the repeatable analysis of information in a deterministic manner, eliminating many sources of error where all parameters of each experiment were explicitly declared.
Figure 1. SfM Example: Input Unordered Images, Output Reconstruction
and could be easily modified as needed. The virtual world also enabled a cost-effective testing environment, as no physical hardware other than a laptop was needed to conduct each experiment. Additionally, the experimental environment for each test was not subject to real-world problems like weather and scheduling to fly a drone to collect aerial photography, as well as the regulatory process that would be necessary to accomplish these tasks.

This thesis contributes an assessment of a new SfM pipeline, purpose-built with fast computation speed as a primary measure of success. This paper compares the implementation of using SfM with Planar Homography Estimation (PHE) to that of a traditional SfM pipeline as well as an evaluation of achieved size reduction of original ISR image sets and the computation time required.

**Contributions.**

1. Proposed novel Structure from Motion algorithm using Planar Homography Estimation to detect and visualize vertical planes that represent walls of man-made structures.

2. Demonstrated the applicability of using homography transformations to generate textures for a 3D object of unknown geometry.

3. Performed analysis of computational time complexity and showed significant speed-ups up to 80x compared to that of traditional SfM.

4. Performed analysis comparing data size of SfM/PHE reconstructions versus traditional SfM reconstructions and original simulated video feed, demonstrating a possible 90% size reduction of original data with new method.

5. Performed analysis of spatial accuracy of reconstructions built by SfM/PHE and traditional SfM with that of original truth model, highlighting SfM/PHE
potential to be more accurate than traditional SfM.
0.2 Background

Technical Overview.

Understanding the process by which Structure from Motion (SfM) works requires in-depth knowledge of the entire digital image processing pipeline. Essentially SfM is a process that “reverses” the action of taking a picture of a three-dimensional scene, so the best way to explain SfM is to first understand the process it is attempting to reverse.

Processing Vision.

Computer Vision and Image processing seek to mimic what a human vision system does so well, and make sense of the myriad of photons colliding with sensory mechanisms. Nature has evolved this ability over billions of years, through many iterations. Yet, the largest part of the human brain, the cerebral cortex, is entirely dedicated to information-processing, so to even nature, vision is difficult problem. At the same time, vision processing is so intuitive to humans that it can be difficult to describe the extreme difficulties faced by researchers implementing even the most basic vision algorithms.

Digital Image Representation.

Computer algorithms are, in general, rigid and usually output quantized, precise numeric information. In image processing, information is conveyed at a quantized pixel level, where each pixel has an inherent position value, and, typically, a finite 24-bit color scale (this may vary based on the type of image) where three 8-bit numbers (storing values between 0 and 255) are used to store the primary color values of red, blue, and green. Naturally, these specifications are modeled after the human limitation to effectively only see combinations of three primary colors that compose a
very narrow band of the electromagnetic spectrum. Each pixel is arrayed to store the information of an image, and it is through this medium that a SfM pipeline operates to recover camera positions and extract depth information.

The Projection Matrix.

Mathematically, the camera used to take an image can be represented by a Projection Matrix that encodes the perspective of the three-dimensional object in a two-dimensional plane. A standard assumption made in all computer vision applications is that the camera can be approximated mathematically as a ideal pinhole camera, an example of which is shown in Figure 2 [6]. This assumption is then used to associate the data stored in pixel space with that of the camera through the means of an intrinsic camera matrix, $K$, $K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$, with $f_x$ and $f_y$ being independent focal lengths in the pinhole camera model, $s$ representing the skew of the image sensor, and $(c_x, c_y)$ defining the optical center expressed in image coordinates. A common assumption when computing $f_x$ and $f_y$ is that $f_x = f_y$ and they are a multiple of the focal ratio, $f_r$, and the largest dimension (height or width) of the image. $f_r$ can be computed as a function of the camera Field of View (FoV) where

$$f_r = \frac{1}{\tan\left(\frac{1}{\text{FoV}_{\text{radians}}}\right)},$$

and

$$f_x = f_y = f_r \cdot \max(\text{imageheight}, \text{imagewidth}).$$
Finally, the aggregation of $K$ with a joined rotation-translation matrix forms the Projection Matrix,

$$P = K[R|T].$$

With a projection matrix, a direct mapping can be made from 3-space, $(x, y, z)$, to pixel space, $(u, v)$. However, it is important to note that 3-space in the camera frame is slightly different than that of standard 3D simulations. Instead of the $z$-axis representing the up-direction as it commonly does in many 3D engines, it instead denotes distance from the camera aperture, where a positive $z$-value is a point that is in front of the camera. $x$ and $y$ are then used to indicate the vertical and horizontal translations centered from the camera’s frame. The full mapping of the projection matrix is represented by the following.
\[
\begin{bmatrix}
  u \\
  v \\
  1
\end{bmatrix}
= \begin{bmatrix}
  f_x & s & c_x \\
  0 & f_y & c_y \\
  0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  r_{11} & r_{12} & r_{13} & t_1 \\
  r_{21} & r_{22} & r_{23} & t_2 \\
  r_{31} & r_{32} & r_{33} & t_3
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}.
\]

However in the process, camera pose information is lost and must be recovered to make an accurate reconstruction from pixel space back to 3-space. In order to recover camera pose, several other factors must be accounted for including possible camera distortion in the original imagery. Essentially, the process of creating a virtual image from a 3D scene is reversed where 3D points must be computed from a 2D uv-map.

\[
\begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}
= [K][R][T]
\begin{bmatrix}
  u \\
  v \\
  1
\end{bmatrix}.
\]

**Handling Camera Distortion.**

Inherent in every image collection system is distortion by the lens and imperfections in the collection sensor. This is also true in the natural world with respect to biological eyes, however, these distortion are “easily” corrected by the brains of organisms, such that the perception of reality is never lost and images appears undistorted. Distortions in geometric optics are a direct result of rectilinear projection, where in the process of measuring light from a 3D world, inherent distortion is applied to the image itself. A related effect is commonly seen in cartography where it is difficult to flatten a geodesic like the Earth into a two-dimensional map without distorting the geography it represents.
Types of Distortion.

Although there exists irregular distortion in images primarily due to the collection sensor, the vast majority of distortion in computer vision follow a simple pattern. The most common is radial distortion and can be classified as either barrel or pincushion distortion as shown in Figure 3.

Fortunately, correcting radial distortion is quite simple in practice, and the quartic model is a 4th order polynomial used to warp each pixels’ location to correct this distortion [6]. First, let \((x', y', z')\) be the pixel coordinates obtained after perspective division

\[
\begin{bmatrix}
x' \\
y' \\
z'
\end{bmatrix}
= \begin{bmatrix} R & x \\
& y \\
& z
\end{bmatrix} + t
\]

such that

\[
x'' = x'/z' \\
y'' = y'/z'.
\]

![Figure 3. Left: No distortion, Middle: Barrel distortion, Right: Pincushion distortion](image.png)

[1]
Quartic Model for Radial Distortion.

According to the quartic model, radial distortion is observed from the center of the image by an amount proportional to their radial distance, \( r^2 = x'^2 + y'^2 \). In its simplest permutation, the model only requires two \( \kappa \)-values to describe the observed distortion.

\[
x''' = x''(1 + \kappa_1 r^2 + \kappa_2 r^4)
\]
\[
y''' = y''(1 + \kappa_1 r^2 + \kappa_2 r^4)
\]

The \( \kappa \)-values directly represent the magnitude of radial distortion present in the image. Barrel distortion is observed when \( \kappa_1 > 0 \) and pincushion distortion is observed when \( \kappa_1 < 0 \). The second \( \kappa \)-value corrects a second-order curvature that defines a “Mustache” distortion. Additionally, the quartic model can be expanded, adding \( \kappa \)-values to further improve the accuracy of the model and correct higher-order distortions.

However, the simplified model provided does not describe the true level of distortion observed by complex lenses. A more complete model includes tangential distortion, or a distortion that occurs across the image [7]. This type of distortion is defined using \( p \)-values and applied with respect to the planar coordinate position as well as the its radial position,

\[
2p_1 x'' y'' + p_2 (r^2 + 2x'^2)
\]
\[
2p_2 x'' y'' + p_1 (r^2 + 2y'^2).
\]

OpenCV uses these two models to compensate for distortion, but also permits the inclusion of four addition \( \kappa \)-values which are optional [1]. The final distortion is represented as

\[
x''' = x'' \frac{1 + \kappa_1 r^2 + \kappa_2 r^4 + \kappa_3 r^6}{1 + \kappa_4 r^2 + \kappa_5 r^4 + \kappa_6 r^6} + 2p_1 x'' y'' + p_2 (r^2 + 2x'^2)
\]
\[
y''' = y'' \frac{1 + \kappa_1 r^2 + \kappa_2 r^4 + \kappa_3 r^6}{1 + \kappa_4 r^2 + \kappa_5 r^4 + \kappa_6 r^6} + 2p_2 x'' y'' + p_1 (r^2 + 2y'^2),
\]
such that

\[ u = f_x x'' + c_x \]
\[ v = f_y y'' + c_y. \]

**Camera Calibration.**

With an appropriate model to describe the properties of the camera and image for the purposes of SfM, the next step in recovering the pose of the camera is accurately calculating these parameters. This highlights a major limitation when using SfM, or any similar method, in that parameters for the camera that stored the image data must be known, or they must be assumed. The latter brings with it the possibility of introducing significant error if the camera’s intrinsic parameters are not computed properly. Compound this problem with the fact that even cameras of the same model will have different variations, calibrating for the exact camera used to capture the images is necessary. Once a calibration is complete, those values can be stored indefinitely, so long as the camera does not experience excessive physical disturbances.

The computation of intrinsic camera parameters is commonly performed using a known calibration target that covers the entirety of an image from multiple perspectives. The classic approach involves a chessboard image, shown in Figure 4, with a known number of chessboard cells. Based on the parameters of each cell, a direct comparison can be accomplished between an ideal chessboard with no distortion, and the provided image of the chessboard. It is important that the captured chessboard extends to the full dimensions of the image, as well as different orientations of the chessboard, to produce the most accurate computation. An example of this process is shown in Figure 5. The chessboard provides an easily identifiable gradient image that is very simple for edge detection algorithms to pinpoint corners in the image. This process is handled natively in OpenCV using the `cv::findChessboardCorners` and
the cv::calibrateCamera functions. Papers by Slama [7] and Tsai [8] go into greater
detail of the specifics of computing the intrinsic camera values and the distortion
coefficients for use by OpenCV.

Finding Image Features.

Detecting specific and uniquely cross-identifiable features in images is paramount
for SfM pipelines, as it is the primary process that first allows unordered images
to be sorted based on similar detected features compared to other images, using
these features to then recover relative 3D pose of the camera that took the image,
and finally using those detected features to create a sparse 3D reconstruction of the
imaged scene, also known as a 3D point cloud. Feature detection holds a central role
in the computer vision field, beginning with simple edge detection algorithms and
moving to more complex unique feature detection methods.

Feature Detectors.

There are several popular versions of feature detectors available, all of which work
in very similar ways. At their core, feature detectors compute a corner response
function ($C$) throughout an image, and those pixels where $C$ exceeds a threshold, or
are locally maximal, are identified as features [9]. In the computer vision literature,
corner and feature are used somewhat interchangeably. One of the original corner
detectors, of which many feature detectors are a descendant of, is the Moravec corner
detector, where the corner response function is a simple minimization function com-
puting the sum-squared-difference between candidate pixels thus extracting corners
at locations which change the most through translation [10]. Further improvements
were made with Harris [11], Shi-Tomasi [12], and Lowe (SIFT) [13] feature detection
algorithms. In computer vision, Scale-Invariant Feature Transform (SIFT) is cur-
Figure 4. Chessboard Calibration Image

Figure 5. Rotated Chessboard Calibration Image
rently the ubiquitous base comparison as the “Gold Standard” for feature detection algorithms, however its main drawback lies in its computation speed.

However, some feature detectors have been designed with computation speed in mind, capable of processing images at real-time frame rates. Features from Accelerated Segment Test (FAST) builds upon the previously mentioned feature detection algorithms with some notable optimizations to produce similar feature detection as SIFT in a fraction of the time [14]. FAST classifies corners by measuring pixel difference from the center of a circle, where $f_c$ is the pixel value at the center of the circle, and $f_p$ and $f'_p$ are the pixel values at either end of the circle, such that if a line were drawn from $f_p$ and $f'_p$, the midpoint of that line would describe $f_c$. Therefore the response function is defined as

$$ C = \min_P (f_p - f_c)^2 + (f'_p - f_c)^2. $$

This equation asserts that a corner exists where $C$ is large. The response function is performed on the perimeter of a Bresenham circle with radius 3, shown in Figure 6. Thus this test is only performed on 16 neighboring pixels for each pixel in the image. Furthermore, a continuous 12 point line segment along the Bresenham circle must be achieved to define a corner according to the FAST algorithm. This 12 point line segment is illustrated by the dotted line in Figure 6 [9].

Finally, the algorithm generates a keypoint descriptor that acts as a unique tag based on the feature properties to be compared against other similar keypoints from different images. The keypoint descriptor is a vector that describes the corner’s “direction” in the image, and the more similar these vectors are, the more likely two keypoints correspond to the same point in different images. Traditionally, SfM assumes the provided input of images is an unordered array. When performing feature matching between images to recover camera pose and 3D points in a sparse reconstruction, the SfM pipeline must compare each image to the remaining set, resulting in
a $O(n^2)$ time complexity for the comparison. Additionally, traditional SfM pipelines require a large data set as too few images will result in degraded information models that do not accurately reflect the imaged object’s geometry. Figure 7 shows an example of a simple array of images that an SfM pipeline would iterate through to build a 3D reconstruction.

**Essential Matrix.**

Another important parameter for defining camera properties in computer vision is the *essential matrix*, $E$, which describes the three-dimensional structure of a scene from a pair of perspective projections where the spatial relationship between the two is unknown [15]. The essential matrix specifically defines a relationship of relative camera rotation and translation matrices between two perspective projections,

$$E = [t]_x \times R.$$

Following Longuet-Higgins, this information is very quick to compute, as such, the essential matrix is extremely valuable for camera pose recovery and 3D point
Figure 7. Array of input images to be used in SfM Pipeline

triangulation [15].

Using Keypoints.

In every image, a set of keypoints or features are identified (Figure 8), where each set is compared with a set of keypoints, $K$. Although the features detected are from images of a virtual model, they are perfectly analogous to how features would be detected in real-world imagery. A similar example using real-world imagery is demonstrated by Roeber et al. [5]. However, not every keypoint found in one image is the same as those found in the other, so each potential list of keypoints must be adequately matched and trimmed to ensure accurate results for the pose recovery and 3D sparse reconstruction. The first method typically used is a symmetry test which verifies that the matches found in $K_a$ matches to $K_b$, and $K_b$ conversely matches to $K_a$. The result of symmetry matching is shown in Figure 9 where each corresponding match is connected by a randomly colored line. While the symmetry match was able to eliminate keypoint features that were not present in the images of the city block from different perspectives, a large percentage of the matches presented remain erroneous, and would cause vastly incorrect results.
Figure 8. Visual representation of all keypoints found in an image using the FAST feature detector

Figure 9. Result of Symmetry Matching corresponding keypoints between two different perspectives
Random Sampling Consensus (RANSAC) to Find Essential Matrix.

After the symmetry test, it is assumed that nearly 40\% of all matched keypoints are considered outliers, which makes it difficult to determine a line of best fit when it comes to experimentally determining the essential matrix. However, the correct matrix can be approximated iteratively by testing a random sampling of possible essential matrices for a pair of perspective projections, and weighting each option by the distance to keypoints that match with each tested essential matrix. This process is known as Random Sampling Consensus (RANSAC). Figure 10 shows a representation of each keypoint in 2-space with almost 40\% of the points shown as outliers. Because the algorithm is constrained by psuedo-random probability, the result is non-deterministic and the best result is not guaranteed, however, accurate results can be achieved within a bounding probability based on the number of iterations, $k$, where

$$k = \frac{\log(1-p)}{\log(1-w^n)}.$$

In this equation $w$ is the assumed number of correct matches in the data set, and $n$ is defined as the minimum number of matches desired to fit the model. For
example, if 60% of matches are correct \((w = 0.6)\), and five matches are desired to fit the model \((n = 5)\), then \(k >= 86\) iterations to achieve 99.9% certainty.

With the essential matrix calculated, further outliers in the paired perspective projections can be eliminated as demonstrated in Figure 11. All keypoints are still present from the older images, but only matches that have been deemed as inliers now have corresponding randomly colored lines between them. Because this perspective pair represents a horizontal translation and rotation of the camera, all the matches in this image follow that horizontal translation pattern, but if the camera were to move in a different manner, the lines may not necessarily be horizontal as shown.

**Extracting Depth Information.**

**Epipolar Geometry.**

Epipolar Geometry is the mathematical constraint used to extract depth information from perspective pair images and is the next step in performing point triangulation of keypoints in SfM, and is covered heavily by Hartley and Zisserman for use in Multiple View Geometry [2]. Typically used to recover the depth information of stereo cameras with known rotations and translations, the usage of Epipolar Geometry is necessary for point triangulation in SfM tools where camera pose is unknown.

The model assumes two pinhole cameras which have the same point \(P\) in front

![Figure 11. Removal of outliers after applying RANSAC](image-url)
of the camera apertures. In a real-world setup, the camera will produce an image that exists behind the focal point of the camera that has a symmetry about that focal point. However, for the purposes of Epipolar Geometry, a virtual image plane is projected in front of the camera along the optical center, $O$. From the optical center of each perspective a line can be drawn from $O$ to $P$ in 3-space, where this line passes through the perspective image plane is known as $p$. Additionally, a line can be drawn between the respective optical centers, and where this line intersects each image plane is known as $E$, the epipolar line, which intersects the image plane at the epipoles (not to be confused with the essential matrix). Both $p$ and $E$ may not necessarily lie within the bounds of the image projection, but they will exist along the image plane nonetheless. For simplicity, both $p$ and $E$ are assumed to be within the bounds of the image projection for Figure 12. Finally, a line can be drawn between $p$ and $E$ for each respective perspective images, and this line is known as an Epipolar.
Using this Epipolar line and the known position of each camera, depth information can be extracted. A useful optimization for triangulating points using Epipolar lines is ensuring the lines are parallel, and mathematically aligning the corresponding points such that either the x-component or y-component is constant. This process is known as rectification and the result of which is shown in Figure 13. The black borders and warped images embedded in the border visually demonstrate the effects of rectification. The exact process of this mathematical alignment is discussed in greater depth in Chapter 3. The rectification process distorts the perspective pair images by mathematically flattening the two 3D image planes through a common double-width aperture to produce perfectly parallel Epipolar lines. Once this is completed for every pair of images in the SfM toolset, 3D points can be triangulated. When the sparse reconstruction is complete in the form of a 3D point cloud from epipolar reprojection and triangulation, shown in Figure 14, with Figure 15 showing the truth model for reference.

**Dense Reconstruction.**

Once the sparse reconstruction is completed, the point cloud can be further refined by interpolating 3D points between those in the sparse point cloud. This is typically

Figure 13. Perspective Rectification to create parallel Epipolar lines
Figure 14. Point Cloud with only ground model superimposed

Figure 15. Point Cloud with full truth model superimposed
performed by pixel analysis in the original imagery, and separating the process into patches via Patch-based Multi-View Stereo (PMVS) [16]. A higher fidelity point cloud is required to further compute the geometry of the object being imaged. PMVS inputs the sparse reconstruction along with recovered perspective projections and creates a patch of 3D points (notice how some points seem to strongly overlay each other in Figures 16 and 17, while also rejecting outliers.

The exact mechanisms of this process will not be further explored as part of this paper, so an example of this process is shown via the OpenMVS pipeline, which handles dense reconstruction, mesh-modelling, texturing, and mesh-simplification [17]. Figures 16 and 17 show two separate perspectives of the dense reconstruction. The point cloud is so dense that points further away appear nearly opaque as the rendering engine can no longer graphically differentiate the points. This step in the SfM pipeline is very computationally intensive and can require up to 25 percent of the processing time for the entire SfM toolchain to complete.

**Mesh Model Creation.**

After the dense point cloud is recovered, it is possible to create a full 3D model of the imaged environment. A 3D model is composed of an array of connected triangles that fit together to form a continuous surface. This is incredibly useful for the purposes of SfM as this is where the Photogrammetry component of SfM meets the components of computer graphics processing. Each triangle in the array that constitutes the model is composed of three vertices previously defined as the various points recovered in the dense reconstruction. The process of mesh creation from a point cloud is fairly simple as each point, or vertex in this case, is connected to its nearest neighbor using Delaunay triangulation [18]. However, this method produces a simplified mesh model, where most use-cases for SfM might require a mesh model
Figure 16. Dense Reconstruction View 1
that has more detail. A more complicated mesh refinement is done by creating simple sub-divisions of each triangle in the simplified mesh. Figure 18 illustrates the results of the Delaunay triangulation method after mesh subdivision.

**Mesh Texturing.**

Once the model has been created, the final texturing of the model can begin. Applying textures to the surface of a model involves wrapping a flat 2D image to a 3D space. This is helpful as the textures for the model can be built from a combination of information from the recovered camera position and the original images used as input, through a process known as raster registration. For a discrete face in the 3D model, a normal vector is computed and a corresponding image that was composed from a similar direction that is the inverse of the computed normal face vector is suitable for use as a texture map. Further edits can be made to this image to account for specific geometries of the object, but the underlying premise remains true. Additionally, there are several processes to accomplish this task to account for various different conditions, but those will not be discussed by this paper [19]. The final result of the textured model is shown by Figure 19, with Figure 20 showing the computed texture map for the entire model computed as one file. However, in the case of this model, the combined OpenMVG-OpenMVS pipeline produced interesting artifacts in the texturing that resulted in parts of the ground model sharing the same textures as the buildings, it is unknown what could have caused this issue, however, for the purposes of this paper, such artifacts will not contaminate results [20] [17].

**Model Size and Orientation.**

One notable artifact of the SfM process is that all camera poses recovered via RANSAC computation are *relative*, meaning that it is impossible recover the absolute
Figure 18. Refined Mesh Model

Figure 19. Textured SfM Model
Figure 20. Texture Map for SfM model
positions of each camera without some external information. As such, the generated models typically have a random orientation and random scale applied to them. This means that these models will require external information if they are to be useful for aerial reconnaissance. Correcting for orientation could be as simple as storing magnetic orientation of the camera, as well as ensuring each picture captured is aligned with a vertical axis based on the direction of gravity, essentially recording the up direction for each image. Scale and translation of the image would be a more difficult process, but models could be compared to existing computed geometry for landscapes. Another solution would be to simply tag each camera position with a GPS coordinate, but this approach lacks versitility in a GPS denied environment.

**Homography Transformation.**

The proposed method for improving SfM specifically for the purpose of reducing the data transmission requirements for aerial imagery involves replacing the mesh model creation with Planar Homography Estimation (PHE), which relies heavily on Homography Transformation to texture detected planes. Homography is fundamental to producing visual perspective of a scene, and a Homography Transformation is a matrix transformation that enables the warping of perspectives in images. The Homography is defined as a transformation between two planes [1]:

\[
s \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.
\]

A homography on a perspective projection image can be used to rectify portions of that image such that they align with a specific geometry. An example of this process is shown in Figure 21 showing the removal of perspective in reference to the rectangular portion of the building. Notice how the outlined rectangle in the right image is now
parallel to the viewer’s image plane.

**Related Work.**

Refining the design of SfM for the specific task of near real-time aerial imaging requires understanding the physical constraints for a UAV, as well as design choices to better optimize the algorithm. This section discusses the design decisions, as well as previous attempts to use methods unrelated to SfM to achieve similar results. However, most significant for this paper is the work done by Jonathan Roeber who demonstrated the use of SfM as a method of data size reduction for aerial imagery [5]. The results reported by Roeber were extremely promising, showing a size reduction of nearly 60% compared to the original set of imagery. The largest drawbacks in his approach were the physical accuracy of the model, where distortion in the reconstruction was unintentionally introduced, as well as the computation time required to create a 3D reconstruction from a set of 90 images.

Analysis of the source code from Roeber’s work revealed a small error in the aspect ratio of the digital camera used to collect images. Despite this correction, distortions are still present in the traditional SfM reconstructions, and the proposed method
by Roeber to correct them remains valid. When using the OpenMVG-OpenMVS pipeline, a different toolset from the one Roeber used, similar distortion is noticeable, reaffirming the conclusion that this type of distortion is inherent to the SfM process.

**SfM Design Decisions.**

SfM can be quantified as a larger problem that can be broken into several smaller problems, and each step in the SfM process can be accomplished in various different ways. Additionally, most freely available SfM pipelines are composed of several smaller programs independently developed and combined as one. No single pipeline contains the best method for generating an arbitrary 3D model from images, each comes with trade-offs. This is best highlighted by the different choices made by designers for different underlying algorithms that accomplish the same task. For instance, in contrast to the Delaunay triangulation method for mesh model creation, the Poisson Reconstruction method for generating a mesh model typically produces a model that is too detailed for aerial reconnaissance purposes, and pipelines using this method would have their defined use-cases specifically tailored to that type of modeling [21].

Another important decision in making a SfM pipeline is which feature/descriptor extractor to use, therefore different analyses of the performance of each is important in choosing which one to use. According to an analysis of various image matching (feature detection) techniques, based on evaluations of Scale-Invariant Feature Transform (SIFT), Speeded-Up Robust Features (SURF), and Oriented FAST and rotated BRIEF (ORB), ORB is the fastest algorithm, but performance is slightly worse than SIFT. ORB is likewise comparable to SURF in most scenarios [22]. These results are also supported by another study of the computation time in OpenCV presented by John Raquet [14].
A noteworthy finding from the study is that ORB tends to concentrate on features in the center of the image, while key point detectors for SIFT and SURF are distributed throughout the image. Another noteworthy finding is that ORB has a tendency to degrade in its feature detection capability when images forming angles larger than 45 degrees are used, therefore it is important for this algorithm that the source images come from sets of images with small angular offsets between each image pair.

**Other Methods Useful to Redesigning SfM.**

Szpak demonstrated using homography to transfer a texture derived from one image and applying it to an image of the same object taken from a different perspective [23]. The transfer areas were designated manually and not automatically generated, but highlight the utility and plausibility of using similar techniques to propagate a texture taken from a certain perspective to other perspectives. Simon developed a method to correlate the known position of a camera from a pre-determined scene and calculate homography transformations between that original image to any image taken in the same general area to create a sense of spatial awareness, and tag certain buildings in augmented reality. Specific areas are defined as predetermined planes of registration to find features within to correlate the original image with the new one to generate the pose of the new camera [24]. These two studies exist as proofs of concept towards using planar homography as a replacement method for texturing the surface of recovered geometry.

Hung proposed a direct pose estimation technique for 3D objects without the use of feature detection. The proposed algorithm performs favorably against “state-of-the-art feature-based estimation approaches,” and is also theoretically highly parallelizable for GPUs. However, this requires a target image to be inputted to compare
against the camera image. This limitation means this method may not be valuable for initial reconstruction but could be useful to ID building faces after they have been observed [25].

Next, Hodan proposed a method for texture-less object detection and evaluates its suitability for augmented reality applications. This method creates a more geometrically accurate edge trace than the standard Canny edge detector. This also uses a Hough-based representation for efficient directional search of edgelets, giving a 5 times speed-up, which could be used for narrowing the search area for specific features in SfM [26].

**Software Tools for Computer Vision and SfM.**

The SfM toolchain used for comparison against the SfM/PHE method proposed by this paper is a pipeline which combines the OpenMVG toolchain, used for feature matching, extraction, camera pose recovery, and sparse reconstruction, with OpenMVS, which generates and textures the final reconstruction model finishing the SfM pipeline [20], [17]. These two tools are freely available, and relatively straight-forward to use. These toolsets were chosen over the ones used by Roeber because they can be seamlessly integrated through a python script, and can produce a full reconstruction without any manual input. However, the final model seems to be more prone to error, as the algorithm does not properly trim the model to the object of interest due to interference from the virtual world’s skybox, which consists of similar textures rendered as a static box around the camera based on the configured view distance in the virtual world. Additionally, all of these tools run natively on Windows operating systems, which was chosen as the primary environment for development and testing of the real-time SfM method.

The OpenCV library is the most important tool used in this paper [27]. It provided
a readily available library feature extraction and camera pose recovery, as well as basic tools to create a sparse reconstruction. All of the testing was done in the Afttr Burner Engine, a in-house virtualization engine at the Air Force Institute of Technology developed by Dr. Scott Nykl [28].
0.3 Methodology

**Structure from Motion with Planar Homography Estimation.**

Based on previous research, for a Structure from Motion (SfM) algorithm to be used for the purposes of reducing the size of aerial imagery is ideal to explore different technologies or methods that could improve its impact and develop a faster process of generating a 3D reconstruction. This effectively requires a complete re-write of the SfM pipeline. The proposed process is reliant on the tools available in the OpenCV library and the results of this method are visualized through the Atr Burner Virtualization Engine [27], [28].

In order to achieve significant speed-up, the sparse reconstruction was chosen to be processed in the same manner as it is in other SfM pipelines, but with the proposed method using the FAST feature detector and the BRIEF feature descriptor based on previous analyses showing superior computation-time compared to other methods [22] [14]. Once a sparse reconstruction is generated, Planar Homography Estimation (PHE) is used, which employs RANSAC to detect planes among the point cloud. From the 3D points in those planes, the homography matrix is computed that relates the 3D points projected onto an XY-plane with those coordinates of the features detected in the original images. This homography is then applied to one of the input images to rectify the area of the detected plane, which can then be used as a texture for the detected plane. This process is repeated for each subsequent view. For an overview of the entire SfM with PHE process, reference Figure 22.

**Operating Assumptions.**

Because the method is designed to run in real-time, the process will not have access to future images, so it is designed to work sequentially, only processing the feature matches found in the current and most previous images. This has the benefit
Figure 22. SfM with PIIE Overview
of cutting out the lengthy $O(n^2)$ process necessary to match an unordered list of images with images from similar perspectives used in traditional SfM pipelines.

To facilitate ease of comparison and to eliminate a source of error in the new reconstruction, it is assumed the exact camera position is known in model space within the Aflr virtualization engine. The errors in determining camera pose from a set of features is well-studied, and not the subject of this thesis. The process of making an estimation of the camera pose from the essential matrix is on the magnitude of single milliseconds so eliminating that calculation will not have a dramatic effect on the final computation time. Retaining access to the exact position of the camera also eliminates all issues with scaling and orientation of the reconstruction with respect to the virtual world, as all calculations can now be performed with the absolute translation and rotation. Furthermore, streamlining testing as a manual alignment of the reconstruction with the truth model is no longer required.

The proposed method is a proof of concept, as such, it is currently only designed to handle specific types of truth model geometry. The method assumes that each model contains vertical walls, common to that of man-made structures, as this algorithm is purpose-built for imaging man-made objects. Aerial surveillance is commonly performed in urban environments, as such the assumptions applied to this model would be applicable for a surveillance regime that specifically targets man-made structures at human scale. However, this limitation is self-imposed and not representative of the ultimate capability of the proposed method, but merely as an approach to limit the scope of initial testing.

Finally, based on initial camera calibration testing, the virtual camera used in the virtualization engine is well characterized, and to save time between tests, the intrinsic parameters and distortion coefficients are manually stored in the source code. However, the program still retains functionality to perform a live recalibration of the
camera and use that data instead.

**Virtual Environment.**

A virtual environment is the preferred method to test the new SfM implementation. Virtual environments are superior to real-world testing in this case as it can accurately model and visualize image projections from any perspective. These flight paths and images can be deterministically repeated over several iterations, removing a source of error that is inherent in real-world testing. It also enables more agile testing where parameters for the simulation can be dynamically changed, as well as being an effective use of resources. There are many barriers to performing this kind of experiments if limited to real-world hardware, such as obtaining licenses and permissions to operate a UAV to collect aerial images, and errors introduced through poor navigation that would make every test a slightly different sample, reducing reproducibility. Therefore for early proof of concepts such as SfM with PHE presented in this thesis it is more efficient to perform preliminary testing in a virtual-world to refine the process. While it could be beneficial to gather data from real-world test harnesses that do introduce inherent errors, it is not necessary to test at this time, as other computer vision algorithms have been demonstrated on real-world objects, and the extrapolation can be safely made that the same would remain true for the proposed method in this thesis.

Each test is performed using the After Burner virtualization and simulation engine, which is an OpenGL based library that has a variety of utilities to handle all graphics and simulation needs in an efficient manner. It is a full virtualization engine with built-in support for resource-management, input/output for software and hardware devices, and hierarchical object management in a virtual world.

The test scene represents a cityscape, with buildings ranging from a simple tall
building to a complex of eleven buildings on a single block. This is designed to test a variety of scenarios that may be present in an urban environment with many structures close together with varied geometries and urban environments with buildings that maintain larger separation. All of these building are box-like, with some containing more complicated geometries, but all buildings imaged have vertical faces, which will be the target of the proposed Real-time SfM method. This should create a varied test environment where the accuracies can be compared to that of traditional SfM, as well as to the original truth model. Figure 23 shows the geometry of the most complex truth model that will be tested. This truth model is similar to Roeber’s demonstration of SfM as a data size reduction method. The only difference is the ground model has been aligned with the building.

Lastly, the image collection apparatus is represented by a small UAV model that will fly around the test environment and capture virtual images of the truth model. The camera is simulated based on the pinhole camera model, and therefore many of
the issues that are inherent to real-world cameras are also applicable to the virtual camera. Furthermore, because the camera is virtual, all parameters of the camera are easily modifiable, but for simplicity the camera’s parameters remain static through every test. As such, the intrinsic characteristics and distortion coefficients must also be characterized to be used for multi-view geometry applications.

**Camera Calibration.**

Camera calibration is performed using readily available methods in the OpenCV library. The calibration object is a chessboard of variable height and width and is dynamically created based on program inputs. In the case of this program test, the calibration chessboard is 32 squares wide by 18 squares tall and properly scaled to cover the entirety of the virtual camera’s field of view for the most accurate calibration results. Additionally, as defined by the OpenCV library, the calibration object must also have a white border around the entirety of the calibration to eliminate false positives on the corner matching. The virtual camera then rotates about the calibration object taking pictures of it in a uniform pattern until the camera completes one orbit about the object. 32 unique perspectives are collected for the calibration object.

The OpenCV library function cv::findChessboardCorners then detects the chessboard corners in the image as features as shown in Figure 24, and because the calibration object has a known width and height, that information can be used to recover the pose of the camera to obtain a camera matrix, and determine if any distortion exists in the image. This process is repeated across every image and then the best fit intrinsic parameters and distortion coefficients are outputted.

Based on a short trial of three experiments, the camera intrinsic parameters are very close to what the predicted $K$-matrix would be based on mathematical assumption that $f_x = f_y$ for a camera that produces images 1920 pixels wide by 1080 pixels
tall and a horizontal Field of View (FoV) of 80 degrees. Note that because the images are stored as zero-indexed arrays, the first pixel begins at 0 and the last will be one less than the total length for the following equations. The expected intrinsic camera values are as follows:

\[
f_r = \frac{1}{\tan\left(\frac{1}{80}\right)} = 0.59631,
\]

thus

\[
f_x = f_y = 1919 \times 0.59631 = 1144.3
\]

Likewise, the optical center of each image is simply half that of each dimension where

\[
c_x = \frac{1919}{2} = 959.5
\]

\[
c_y = \frac{1079}{2} = 539.5
\]

The experimental results were exactly the same for each trial providing confidence for the results, and they were very close to the predicted K-matrix as well. The calculated values are shown below.
Similarly, the experimentally derived distortion coefficients were promising, with minimal distortion indicating that a near-perfect pinhole camera had been achieved. This matches a manual pixel analysis of the calibration object, where all pixels where found to be in perfect alignment, again indicating a near-perfect camera.

With this preliminary data, the intrinsic camera parameters and distortion coefficients could remain as constant variables through future testing without the need to re-calibrate the camera before every test.

**Input Data Collection.**

**Orbital Flight Path.**

A single circular flight path that orbits a central point was chosen as the method of data imagery collection with 30 evenly-spaced images. This serves the purpose of ensuring that every face of the model is imaged equally, creating a best case scenario for creating a well-matched point cloud, which can be used to further detect building walls, represented as vertical planes in the scene. The flight path is inspired by the previous work of Roeber and Ekholm, who used similar flights paths to gather imagery of the truth model [5] [29].

However, this flight path differs from Roeber in the sense that it is a purely circular orbit with no spiral pattern. The flight path is predefined by a configuration file with parameters for the altitude (in meters) of the orbit, the radius (in meters) of the orbit, and the number of images collected $n$. Additionally, the orbit requires a desired target, which in this case is defined as the look target relating to the fixed 3D point $(x, y, z)$ at which each camera should be looking. Figure 25 shows a standard
Figure 25. Standard orbital flight path with camera frustums

Figure 26. Example of images collected by virtual camera to be used for feature matching and plane texturing
flightpath around a truth model, and each perspective projection is represented by camera frusta originating from a point along the flight path. Figure 26 shows an example of captured images using the described orbit.

**OpenCV Point Triangulation.**

Once the images are collected they can then be processed to create a sparse point cloud which is later used to detect vertical planes in the scene. This uses of the OpenCV library for Image Projection and 3D Vision [27]. Specifically the cv::TriangulatePoints function and its pre-requisite inputs to reproject 2D keypoints into a 3D point cloud representing the original scene. This is a mathematically rigorous task with little room for error, and is explained in detail. Point triangulation is a derivative of stereo imaging, as such it relies on the past work of Trucco and Verri [30], Hartley and Zisserman [2], Forsyth and Ponce [31], and Shapiro and Stockman [32].

A set of stereo images must be computed for each perspective of the desired scene. This requires designating a left and a right image to represent a left and a right camera. To use the same camera for several different perspectives, the assumption that the scene being imaged is static must be enforced. A moving scene would result in inaccurate triangulation of keypoint features.

As a note, coordinates will need to be transformed between the Aftir Engine coordinate system and the OpenCV camera-centric coordinate system. Figure 27 is a representation of 3-space in the OpenCV coordinate system, where the z-axis denotes distance from the camera, and the x and y-axes represents horizontal and vertical translations, respectively. This is different from the Aftir coordinate system where the z-axis is a vertical translation in reference to an xy-ground plane. The following defines the transformation matrix from Aftir to the camera frame,
\[
F_{\text{OpenCV}} = \begin{bmatrix}
0 & -1 & 0 \\
0 & 0 & -1 \\
1 & 0 & 0
\end{bmatrix}.
\]

Additionally, all matrix multiplication described follows column major, post multiplied.

When using the OpenCV library with Aptr there are four major steps you must consider:

1. Mathematically \textit{undistort} the given images using the intrinsic matrix \( K \) based on a previous calibration of the camera. This is done by comparing the perfect image of a known object to the recorded image by the camera and calculating the discrepancies between known feature points. These known feature points are commonly the corners of a checkerboard and OpenCV has libraries for this function under \texttt{cv::CalibrateCamera}, explained previously in this section.

2. Next, the images must be \textit{rectified} such that they are mathematically represented along the same plane. The goal of this step is to row-align the images such that the epipolar lines are entirely parallel with each other. I.e., the two images in a perspective need to be coplanar and the corresponding rows must be collinear relative to each other. In practice, this is accomplished through the use of \texttt{cv::stereoRectify} to calculate the corresponding rotation and projection matrices that need to be applied to the original image, and \texttt{cv::undistortPoints} to apply those changes to previously calculated keypoints found in each image.

(a) To accomplish this, the relative position of the respective camera positions must be recovered. In a system where the exact position is unknown (similar to most SfM applications) the relative position and rotation can be computed as a derivative of the Essential Matrix \( E \) of a given camera
pairing. In OpenCV, this is done via cv::findEssentialMat which requires a set of keypoints from each image and uses RANSAC or some other consensus algorithm to compute the inliers or “good matches” between two images. Then with an essential matrix, $E$, cv::recoverPose returns a valid relative rotation $R$ (3 x 3) and translation $T$ (3 x 1), however, recovery of $R$ and $T$ from the Essential Matrix $E$ does not preserve the global scale and orientation of the reconstructed object, so that will need to be corrected at a later time. Figure 28 illustrates an example of the relative rotation and translation between two images. $H_f$ is described by rotation $R_1$ and $T_1$, and $H_r$ as $R_2$ and $T_2$.

(b) The alternative is to calculate the relative position and rotation from known values within the system. For this thesis, this is the primary method used to limit nondeterminism in the point cloud reconstruction pipeline.
This is a simplifying assumption that preserves the scale and orientation.

(c) In this scenario, the relative rotation matrix $R$ is computed as the relative rotation from $R_2$ to $R_1$.

$$R = R_1 \times R_2^T$$

Next, the relative translation from Camera 1 to Camera 2 is calculated as the difference of Camera 2’s position and Camera 1’s position rotated about Camera 1’s rotation matrix.

$$T = R_1 \times (T_2 - T_1)$$

Optionally, the effects of these changes can be viewed to verify our accuracy by using cv::initUndistortRectifyMap to apply the new rotation and projection matrices computed by cv::stereoRectify. Below is an example of an unrectified pair of images and a rectified pair of images (Figures 29, 30). Notice how the rectified pair appears to be along a plane centered between the two perspectives. This is important for orienting the point cloud later.

3. Next, the task of triangulation via cv::triangulatePoints is performed. This step is relatively straightforward as OpenCV only requires the projection matrix $P_1$ and $P_2$ (generated by cv::stereoRectify) and a set of rectified points (generated by cv::undistortPoints) to properly extract the depth information from the images and generate a 4D homogeneous point cloud. For the purposes of this thesis, a standard 3D point cloud is desired, so the 4D homogeneous data structure is converted into a 3D point cloud by applying the scale factor $w$ (the fourth index in the 4D array) to each 3D component.
Figure 28. OpenCV Camera Rotation Diagram

Figure 29. Unrectified Match

Figure 30. Rectified Match
Notice in Figure 31 how the points at the very top exhibit z-fighting in the model, as this represents the global z-axis in the model around which the cameras pivot. This will not occur in most SfM scenarios, but this example was specifically set up to demonstrate this phenomenon as a quick sanity check.

4. After a set of 3D points, $P$, have been obtained, the scene can be reconstructed through a process called \textit{reprojection}. Up to this point, the points are projected with the global origin as a place-holder for the source camera, they must be flipped along a mirror axis and as the points are assumed to be reprojected from the perspective of their source camera. Unfortunately, there is no standard tool in OpenCV to accurately transform the points computed from \texttt{cv::triangulatePoints} into a reprojected point cloud that snaps over the surface of the originally imaged object. For that, the rotation is manually computed from each camera point for reprojection.

For reprojection, begin by converting the 3 x 1 point matrix into the desired coordinate from OpenCV to Aftr since the two use a different coordinate system. This is done via the transpose of the OpenCV coordinate transformation matrix, $F_{\text{OpenCV}}$.

\[ F_{\text{OpenCV}}^T \times P \]

Next, take the rotation matrix for Camera 1, represented in the Aftr coordinate frame, $R_1$ and compute the negated transpose and apply it to an identity matrix rotated about the Y-axis, $M$. This step effectively mirrors the points so that they can be reprojected from the perspective of Camera 1 as opposed to the origin. Next, this term is applied to the previous calculation. The aggregation of these terms is shown in Figure 32.
Figure 31. Triangulated Points without Reprojection

Figure 32. Triangulated Points with some Correction
\[-R_1^T \times M \times F_{OpenCV}^T \times P\]

Next, apply a half rotation, \( \mathbf{R}_{half}^T \), of the angle between the left camera and right camera to re-orient the points with respect to the left camera, as they were previously oriented such that the mid-point between Camera 1 and Camera 2 was the origin of projection.

\[\mathbf{R}_{half}^T \times -R_1^T \times M \times F_{OpenCV}^T \times P\]

Finally, add the translation vector \( T_1 \) of Camera 1 to each point to complete the reprojection.

\[T_1 + \mathbf{R}_{half}^T \times -R_1^T \times M \times F_{OpenCV}^T \times P\]

Figures 33 and 34 shows the end result using the assymetrical checkerboard object.
Figure 33. Reprojected Points with Model

Figure 34. Reprojected Points without Model
Plane Finding.

The sparse reconstruction extracted from the input imagery contains valuable information about the model. Many different methods of analysis can now be used to make sense of the 3D point cloud. With a sparse reconstruction, the human eye is able to assume the geometry of complex Figures present in the scene as shown in Figure 35 where the exterior geometry of the city block is visible. While specific details cannot yet be determined, an observer might be able to recognize several building facades along the edge of the block, and well as one large protruding building on the right corner of the block.

Exploiting this data using computer vision algorithms can be done in several ways. One approach involves attempting to match basic object primitives to the geometry present in the sparse reconstruction [33]. However, a simpler method was chosen where an algorithm attempts to find the most prominent set of point-defined planes in the sparse reconstruction and isolate them for further processing.

Using a similar RANSAC process as the feature matching to determine the essential matrix for each perspective projection, three points are chosen at random from the sparse reconstruction. Checks are made to ensure that the chosen points are not duplicates of each other as well as ensuring that points are not colinear. Both of these conditions would result in every remaining point in the sparse reconstruction matching being coplanar to the original three points. Other intrinsic errors are introduced in the process of extracting the 3D geometry of the scene, and the algorithm accounts for these variations through a threshold to determine coplanar and colinear points. These thresholds are dynamically computed based on given inputs for the size of the scene imaged. This is done by computing two vectors, \( V_1, V_2 \), from the three points, \( p_1, p_2, p_3 \), and comparing the magnitude of the cross product of these two vectors to ensure they are greater than the given colinear threshold, \( \sigma_{linear} \). Ideally, \( \sigma_{linear} \)
Figure 35. Reprojected Points without Model

would be zero, but to account for small variances in the data, it is calculated with respect to the size of the detected plane. Mathematically, a good plane is determined as

\[ V_1 = p_1 - p_0 \]
\[ V_2 = p_2 - p_0 \]
\[ \sigma_{\text{linear}} = \frac{|V_1|+|V_2|}{2} \]

where

\[ |V_1 \times V_2| > \sigma_{\text{linear}}. \]

Once three physically separate points are chosen, then the number of points that are members of the chosen plane are counted. This is done by computing third vector, \( V_3 \), as the difference of \( p_1 \) and \( p_6 \). Ideally, a plane will always have a volume of zero, however, in practice due to the errors introduced through the sparse reconstruction, it is highly unlikely than any experimental point will perfectly align with the rest of the
plane. Therefore, the computed volume from three distinct vectors from a common origin can be used to verify that the tested point is near the randomly chosen plane. If that volume is within some $\sigma_{planar}$ of zero, then it is considered a member of that plane.

$$V_3 = p_3 - p_0$$

$$\sigma_{planar} = |V_1 \times V_2|$$

$$(V_1 \times V_2) \cdot V_3 > \sigma_{planar}.$$ 

This RANSAC plane detection process is repeated for twice the number of 3D points present in the point cloud to ensure the accuracy of the detected plane. Each iteration checks for the plane with most members and stores the largest set as the best plane. Alternatively, if a candidate has the same number of members as the best plane, then the total volume of each plane is compared, and whichever has the smallest total volume becomes the next iteration’s best plane. For simplicity in the method, the actual volume of each detected plane is not computed. Instead, a comparison of the sum of each volume previously computed to determine whether a point was within $\sigma_{planar}$ is used. This is a valid comparison under the assumption that the candidate and the previous iteration’s best plane both have the same number of members, and therefore, the plane with the smallest summed volume is also the plane with the smallest total volume. Figures 36 and 37 show the results of this process. The bounding boxes for the point clouds are enabled and demonstrate how large the original point cloud is before the plane detection occurs.

Next, a check is made to ensure the detected plane is a vertical wall. This is done by comparing the $z$-value of the computed normal vector, $N$, of the plane (computed from the normalized $V_1 \times V_2$), and verifying that it is near 1. In the case of the method, a $z$-value greater than 0.95 in the normal vector is accepted.
Figure 36. Point cloud of one perspective before plane detection

Figure 37. Point cloud of best plane detected from one perspective projection
Next, the displacement component of the plane, $P$, is computed as the maximum dot product of a member point, $P_m$, and the normal vector, satisfying the equation for a point that lies in a plane,

$$A = N_x$$

$$B = N_y$$

$$C = N_z$$

$$D = \max(p_m \cdot N)$$

$$Ax + By + Cz + D = 0.$$  

Finally, the best plane detected by RANSAC is compared against previously detected planes from different perspectives to determine if it a duplicate. Further computations for a plane that has been previously detected in the scene is unnecessary unless the newer detected plane is better than previous ones. First the normal vectors for the candidate plane, $C_n$, and each previously detected planes, $C_n$, are compared by computing the magnitude of the combined vectors versus the individual magnitudes of each vector. If the magnitudes are within 90% of the length of the two vectors combined, then they are considered similar. Next, if the displacement values, $C_d$ and $D_d$, are close to each other within some threshold, $\sigma_{\text{dist}}$, the two planes are deemed similar.

$$\sigma_{\text{dist}} = (\max(P_m) - \min(P_m)) \cdot 0.025$$

$$|C_n + D_n| > (|C_n| + |D_n|) \cdot 0.90$$

$$|C_d - D_d| < \sigma_{\text{dist}}.$$  

However, if the candidate plane contains more 3D points it will be added to the list of detected planes and replace the previous plane that is similar to it.

In this implementation of the method, only one plane is detected between two perspective projection pairs. It could be altered to detect further smaller planes
present in each pair, but for simplicity of testing, only the most prominent plane in each pair is needed.

**Homography Transform and Texturing.**

The homography transformation is possible once planes have been detected within the sparse reconstruction. The method for computing a homography transformation between two sets of points is already given via the OpenCV library [27], however, the two sets of points need to be represented as a 2D array of xy-points in an image. The first set can be directly derived from previous computations where the 2D feature keypoints were computed in the image, but the second set must come from the 3D points. In this instance, the 2D feature keypoints represent the previously warped perspective, and the 3D points can be altered to represent a perfect planar projection of a desired area in an image. In this case, the desired area is the vertical face of a building, described previously as the detected plane in the sparse reconstruction.

For this the 3D points from the plane can be projected as 2D points as members of an rectified xy-plane, \( P \). First the x and y vectors for the plane are computed. Because the planes represent vertical walls, for this purpose they are assumed to be

\[
P_y = (0, 0, 1),
\]

thus the x-component can be computed as the cross product of the normal vector of the plane and the y-component,

\[
P_x = P_n \times P_y.
\]

Next, the corners of the 2D projection are derived from the 3-space min and max points present in the detected plane, where the bottom left and top right corners, \( BL \) and \( TR \), of the 2D plane is computed as
\[ BL = (P_x \cdot P_{min}, P_y \cdot P_{min}) \]
\[ TR = (P_x \cdot P_{max}, P_y \cdot P_{max}). \]

It is unknown exactly how the plane is oriented in 3-space, so a simple sign modifier for the x and y components of each 2D points is computed as

\[
M_x = \frac{TR_x - BL_x}{\|TR_x - BL_x\|}, \\
M_y = \frac{BL_y - TR_y}{\|BL_y - TR_y\|},
\]

which is then applied to the respective components of the corners and to the 2D projection of the remaining points in the detected plane. The 2D projection for each point, \( p \), in the detected plane, \( P \), is

\[
p_x = M_x \cdot (P_x \cdot P_m) \\
p_y = M_y \cdot (P_y \cdot P_m)
\]

Next, the scale and offset for the projected points need to be computed as they exist in a different reference frame from that of the original perspective projection pair. This is done by computing vectors, \( v_o \) and \( v_p \), for the \textit{original} keypoints, \( k \), and the \textit{projected} 3D points, \( p \), of the detected plane.

\[
v_o = k_0 - k_1 \\
v_p = p_0 - p_1
\]

where the scale, \( s \), and the offset, \( o \), are

\[
s = \frac{|v_o|}{|v_p|} \\
o = k_0 - s \cdot p_0
\]

The scale and offset can then be applied to all the points in the detected plane. At this point, the two sets of points can be used by the \texttt{cv::findHomography} function.
in the OpenCV library to calculate the homography matrices for the left and right perspective projections.

Then using cv::warpPerspective, the computed homography can be applied to the original images, which warps the perspective of the original image as a fully-rectified texture. Figures 38 and 39 shows the results of the homography transformation to visually align the detected plane with the viewer's image plane. The black borders and warped images embedded in the border visually demonstrate the effects of rectification.

Finally, the image is then cropped to the size of the detected plane and used as a texture for a quadrilateral object in the reconstruction. For this method, no further image processing is performed, so the full quadrilateral is not properly trimmed according to geometry within the plane. This introduces some errors in the model reconstruction for complex structures such as cityscapes where some building may be taller than others, and do not form a perfect quadrilateral. An example of this error is shown in Figure 40. While this looks fine at first, the rest of the algorithm will simply apply this entire image as a texture for the detected plane, resulting in spatial error for the objects imaged behind the detected plane, such as the roofs of the buildings, and the buildings in the back row. This could be fixed in the future with better image processing algorithms to remove pixels that differ based on different perspectives as demonstrated in Figure 41. This is a crude example based on removing pixels that had a different color between two similar perspectives, however, more robust solutions were not explored in this thesis.

**Plane Refinement.**

If a better plane is discovered that occupies the same location as another plane, it must be replaced. A simple method that only replaces the reconstruction quadri-
Figure 38. Before homography transformation

Figure 39. After homography transformation
Figure 40. Homography transforms introduce error by displaying objects behind the desired plane as though they were part of the plane.

Figure 41. Combined rectified perspectives with differing colored pixels removed.
lateral, \(d\), with the candidate quadrilateral, \(c\), when they overlap is used.

A similar check as the plane detector to find similar planes among the plane that is to be added to the reconstruction model,

\[
\epsilon_{\text{dist}} = (\max(P_m) - \min(P_m)) \cdot 0.025 \\
|c_n + d_n| > (|c_n| + |d_n|) \cdot 0.90 \\
|c_D - d_D| < \sigma_{\text{dist}}.
\]

Using the above equation, it determines if a candidate plane and an existing plane share similar normal vectors and displacement components. If this is the case, then an additional check is performed to verify that the planes themselves overlap.

\[
x_\chi = c_x \cdot d_x - c_{\text{width}} - d_{\text{width}} \\
y_\chi = c_y \cdot d_y - c_{\text{height}} - c_{\text{height}}
\]

If either \(x_\chi\) or \(y_\chi\) are less than zero, then the two planes intersect and the algorithm defaults to the candidate plane replacing the current plane in the reconstruction.

Once all the perspective projection pairs have been processed, the reconstruction for SfM with PHE is complete.

**Generating Traditional SfM Reconstruction for Comparison.**

For an evaluation of the efficacy of using SfM with PHE, a traditional SfM Reconstruction is also generated to be used for comparison. This is done using the OpenMVG and OpenMVS pipelines linked to each other through a single python script [20], [17]. The exact same input images are used in this pipeline, using the same intrinsic camera parameters.

Great care is used to ensure that the alignment of the traditional SfM reconstruction matches the truth model for the best possible spatial comparison.
Evaluation Criteria.

Multiple comparisons will be made for each data set. First a data size comparison between the storage size of the input images, the size of traditional SfM reconstructions, and the size of the SfM with PHE reconstructions will be done. Many different methods to compare the size of the original data set and the reconstruction size were considered, but to create the best raw comparison, no changes were made to the data sets and the reconstructions. In a real scenario, the original data would likely be transmitted via some video compression technique from the aerial surveillance platform, however, the compression gained by any pre-transmission compression method, could likewise be applied to the final reconstruction. For that reason, a raw data size comparison is used for this experiment, rather than applying some sort of image/video compression technique that might be seen in the real-world.

Next, a computation time analysis is performed, measuring the length of time required to complete a traditional SfM reconstruction and a new SfM PHE reconstruction. This is a rough analysis, as the time and power criteria will be different for different machines and graphical processing units, as such these results will be purely relative for the purposes of comparing the two methods.

Finally, both the traditional and PHE reconstructions are assessed for their spatial accuracy by using virtual georeference points, which has also been used in related works [5] [34] [35]. The georeference points are manually chosen on key points in the truth model, and the same points are mapped similarly to both reconstructions. A sum-of-squared differences is used as the error metric, where a lower number corresponds to better spatial accuracy. For this experiment, only the XYZ error is tested.
0.4 Results and Discussion

Gathering Results.

Hardware/Software Environment.

The experiments were conducted on a Thinkpad P51 laptop similar to the same hardware used by Roeber [5]. It uses an Intel i7-7820HQ CPU clocked at 2.90GHz, with physical 4 Core, 8 Logical Processors using Intel Hyperthreading, and 16 GB RAM. OpenMVG and OpenMVS support multi-threaded processing, so they will have a small advantage over the SfM PHE method which currently does not implement multi-threaded computations. However, a future implementation of this method would be capable of multi-threading as well as potentially utilizing graphical processing units and vectorized SIMD instructions for even greater speed-ups. Traditional SfM has another advantage over PHE in that most implementations use smaller texture sizes for various parts of the reconstruction geometry. The current implementation does not resize the original in any way, it merely crops the image size to the size of the plane after it has been rectified using a planar homography. In the future, it would be possible to implement a similar size reduction in textures as a texture for 3D model does not need the same image size to convey a similar level of detail as a normal 2D image.

Datasets.

One dataset was chosen for each virtual environment. Each dataset assumed only one circular pass could be made, rather than in previous work done by Roeber where many images were used from multiple different angles [5]. The datasets generated in these experiments are similar to an actual reconnaissance flight which may not be able to gather a perfect dataset on the desired target. Because of this, notice-
able degradation in the model accuracy of the traditional SfM reconstructions was observed.

The specific parameters of each flight path were chosen based on desired test parameters for each model. For instance the Arab House flight path is an orbit around a building at a very low altitude, testing what the algorithm might produce when a surveillance platform is used near eye level. Whereas the Tall Building and the City Block models demonstrate a flight path from a reconnaissance platform flying high above a city. No specific flight path was chosen because it was the best one for that environment, but merely chosen based on the best ability to capture the entire object in one image.

A final dataset removes the circular orbit, and uses a point-to-point flight path in its place, where the camera flies from point to point between buildings, imaging an “Urban Canyon” that is created by small streets between large buildings.

Work done by Roeber demonstrated that the exact flight paths did not have much of an effect on the final model, but what exactly was captured in each image was most important. Additionally, due to limitations in the feature matching algorithms, each image pair must come from camera positions that are within 45 deg of each other. Any incidence angle greater than that will result in a strong degradation of the amount of features that can be matched. This effect is also present in traditional SfM, so for each orbit, 30 pictures are taken, thus an angle of 12 deg was used between each image.

Geopoints were collected manually by finding easily identifiable features between the truth model and reconstructions. Typically, this meant various windows and building corners present in the scene. An example of this process is shown in Figures 42 and 43. Yellow and orange points are visible in these pictures, with yellow indicating the geopoints for the reconstruction, and orange for the truth model. In this
example, the orange geopoints are very difficult to see as they line up near perfectly with the yellow geopoints points.

**Discussion.**

All the results discussed in this section are shown below as various tables and Figures. Comparisons on the physical data size, computation time, and spatial accuracy were made. For every reconstruction with and without PHE, the size of each model was significantly smaller than the original data set containing 30 images from different perspectives around the model. However, PHE differed from traditional SfM in that it performed its task much faster, achieving significant speedups, up to 80 times that of traditional SfM. Based on visual inspection, the traditional SfM appeared to be less spatially accurate than PHE, however, for instance where PHE failed to properly find a plane, or where objects behind the plane were included in the texture, PHE accuracy fell. Also, a metric analysis was not performed for the Arab House due to the absence of easily identifiable features to measure between all models.

The Arab House proved to be the most difficult for both algorithms to create reconstructions for as the virtual model was very simplistic and contained many repeated features, but once again, the PHE reconstruction appears to better than the traditional SfM reconstruction. Additionally, several of the reconstructions built with traditional SfM had to be redone, due to very large distortions present in the reconstruction. This is likely due to the automated algorithm that chooses what is the most prominent object in the scene, and it is likely that the virtual skybox was causing these errors. This is supported by Roeber’s previous work, where the virtual skybox was intentionally left blank due to errors it introduced.

Because this algorithm is only designed to work for vertical walls of man-made structures, the roofs of these structures are noticeably missing. This was an inten-
Figure 42. Example of geopoints on truth model

Figure 43. Example of geopoints on PIE reconstruction
tional design decision when building the algorithm to enable ease of testing, and because an implementation was not made that estimates the full geometry of an object based on its point cloud. This is a separate component to the problem that is left as future work.

**Tall Building.**

The singular tall building shown in Figure 44 represents a best case for this specific implementation of PHE, as it contains a singular cuboid geometry. This is because the implementation uses very simplistic methods to recover the full geometry of a scene. Future implementations could include better imaging algorithms to trim planes to the specific geometry of complex objects.

PHE was able to achieve the greatest size reduction with the Tall Building, reducing the final reconstruction 98.6% smaller than the original 30 image data set, all while doing it in approximately 7 seconds. If use for a video feed is assumed, then for this instance, PHE would be able to process the scene at 4 frames per second.

Even though the full geometry is not represented by the PHE reconstruction, what was represented matched the truth model nearly perfectly, showing little to no error, see Figure 51. More error was likely introduced through the manual process of collecting the geopoints than due to inaccuracies in the reconstruction. In contrast, the traditional SfM reconstruction had a significant lean present in the model, in addition to the model appearing very rough in general. This observation is supported empirically by Figure 50 and through visual inspection of Figures 46 and 48.

<table>
<thead>
<tr>
<th>Scene</th>
<th>Dataset</th>
<th>Trad. SfM</th>
<th>Improvmt</th>
<th>SfM w/ PHE</th>
<th>Improvmt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tall Building</td>
<td>178 MB</td>
<td>13.6 MB</td>
<td>92.4%</td>
<td>2.53 MB</td>
<td>98.6%</td>
</tr>
<tr>
<td>City Block</td>
<td>178 MB</td>
<td>26.9 MB</td>
<td>84.9%</td>
<td>9.29 MB</td>
<td>94.8%</td>
</tr>
<tr>
<td>Arab House</td>
<td>178 MB</td>
<td>9.47 MB</td>
<td>94.7%</td>
<td>11.8 MB</td>
<td>93.4%</td>
</tr>
</tbody>
</table>
Table 2. Computation Time Comparison

<table>
<thead>
<tr>
<th>Scene</th>
<th>Traditional SfM</th>
<th>SfM with PHE</th>
<th>Speed Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tall Building</td>
<td>282.7s</td>
<td>7.045s</td>
<td>40.12x</td>
</tr>
<tr>
<td>City Block</td>
<td>783.1s</td>
<td>9.842s</td>
<td>79.56x</td>
</tr>
<tr>
<td>Arab House</td>
<td>495.3s</td>
<td>5.973s</td>
<td>80.92x</td>
</tr>
</tbody>
</table>

City Block.

The city block represents a typical use case for aerial surveillance on man-made objects. It contains many buildings with various and complex geometry. In its current implementation, PHE reconstructions might not be fully adequate to replace the video feed for aerial reconnaissance, showing its limitations to only find simple rectangular planes to texture. However, for the areas that are represented in the reconstruction, they are more accurate than that of traditional SfM, which is prone to greater distortions. Additionally, the PHE reconstruction might also be limited by the nature of the virtual model, which contains fewer physical features for the feature matcher to recognize. So the limitations present here may not fully represent its capability with real-world imagery.

None the less, the physical size of the model was still smaller than both the original data set and the traditional SfM reconstruction, PHE was computationally faster as well. Similar distortions are present in the traditional SfM reconstruction as shown in Figure 56 and supported empirically by the graphs in Figures 60, whereas the PHE reconstruction had no such distortions present.

Arab House.

The arab style house demonstrates how PHE might perform with images taken from street level, where the geometry of the roof may not be known. Typically, this is also accomplished easily by traditional SfM algorithms, but in this instance,
the model textures were so similar that they resulted in many inaccuracies in the reconstruction process, as such, only a partial model was recovered.

Interestingly, the size of the traditional SfM reconstruction was smaller than that of the PHE reconstruction, however, this may be attributed to the incomplete reconstruction that was generated. Even though the PHE reconstruction was larger than the traditional SfM reconstruction, it was still significantly smaller than the data set, and the computation time of PHE was still much faster than traditional SfM.
Figure 44. Tall Building Scene with only the truth model visible

Figure 45. Tall Building Scene with the truth model overlaid with the traditional SfM model
Figure 46. Tall Building Scene with only the traditional SfM model visible

Figure 47. Tall Building Scene with the truth model overlaid with the PHE SfM model
Figure 48. Tall Building Scene with only the PHE model visible

Figure 49. Tall Building Scene with the truth model overlaid with the traditional SfM model and the PHE model
Figure 50. Tall Building Metric Analysis of Spatial Accuracy of X-face between truth model and traditional SfM

Figure 51. Tall Building Metric Analysis of Spatial Accuracy of X-face between truth model and PHE
Figure 52. Tall Building Metric Analysis of Spatial Accuracy of Y-face between truth model and traditional SfM

Figure 53. Tall Building Metric Analysis of Spatial Accuracy of Y-face between truth model and PHE
Figure 54. City Block Scene with only the truth model visible

Figure 55. City Block Scene with the truth model overlaid with the traditional SfM model
Figure 56. City Block Scene with only the traditional SfM model visible

Figure 57. City Block Scene with the truth model overlaid with the PHE SfM model
Figure 58. City Block Scene with only the PHE model visible

Figure 59. City Block Scene with the truth model overlaid with the traditional SFM model and the PHE model
Figure 60. City Block Metric Analysis of Spatial Accuracy of X-face between truth model and traditional SfM

Figure 61. City Block Metric Analysis of Spatial Accuracy of X-face between truth model and PHE
Figure 62. City Block Metric Analysis of Spatial Accuracy of Y-face between truth model and traditional SfM

Figure 63. City Block Metric Analysis of Spatial Accuracy of Y-face between truth model and PHE
Figure 64. Arab House Scene with only the truth model visible

Figure 65. Arab House Scene with the truth model overlaid with the traditional SfM model
Figure 66. Arab House Scene with only the traditional SfM model visible

Figure 67. Arab House Scene with the truth model overlaid with the PHE SfM model
Figure 68. Arab House Scene with only the PHE model visible

Figure 69. Arab House Scene with the truth model overlaid with the traditional SfM model and the PHE model
0.5 Conclusions and Future Work

Conclusions.

Through this research, a new method for collecting and representing aerial reconnaissance data was proposed using Structure from Motion (SfM) with Planar Homography Estimation (PHE). This system was built with computational speed in mind, and it was able to achieve its goal of providing similar data to traditional SfM algorithms at a fraction of the computation time. This demonstrates its potential for use to augment or replace aerial reconnaissance tool chains, by enabling the possibility of pre-processing data into a smaller package before it is transmitted from the remote device, which builds from the previous work presented by Roeber [5].

The speed at which this method was developed would not be possible without the strategic use of virtual environments and a graphical virtualization engine to test the new method. The use of a virtual environment enabled the repeatable analysis of information in a deterministic manner, eliminating many sources of error where all parameters of each experiment were explicitly declared and could be tweaked on the fly if needed. The virtual world also enabled a cost-effective testing environment, as no physical hardware other than a laptop was needed to conduct each experiment. Additionally, the experimental environment for each test was not subject to real-world problems such as weather and scheduling a flight to collect aerial photography, as well as the regulatory process that would be necessary to accomplish these tasks.

Many metrics were used to measure the effectiveness for SfM with PHE versus traditional SfM and a standard data set to simulate the requirements for a video feed. From the data collected, SfM with PHE shows promise, but requires further development before it can be used as an alternative to traditional SfM or replace video imagery. However, it does come with the benefits of creating a model that reflects greater spatial accuracy of the scene versus traditional SfM, and is able to create a
reconstruction is a timely manner that would be acceptable for aerial reconnaissance, depending on the application.

**Future Work.**

As stated previously, there is still much work to do, and many different areas that could use improvement. To improve spatial accuracy, further exploration of image processing algorithms to detect the edges of a rectified face of a building could be used, similar to Figure 41 in chapter 3. Ideas such as using Hough lines and pixel comparisons between different frames were explored as part of this research, but ultimately not included as an adequate working model was not achieved in a timely manner. Other exploration could look into using the images parallax as a means to determine what parts of the image belong to the detected plane, and what parts of the image lie in front of or behind the rectified plane.

To improve the computational time required for this method further, multithreading strategic areas of the algorithm that perform repeatative tasks could be beneficial, but was never explored as part of this research. Furthermore, it should be possible to accelerate computations via a graphical processing unit in some areas of SfM with PHE. This is possible due to many sections in the algorithm where calculations are performed on a pixel-by-pixel level, and would benefit greatly from vectorized execution.

More varied flight paths should be tested in the future, like a point-to-point flight path that can specifically mimic data collection in an urban canyon as the drone flies over streets between buildings.

An analysis of the energy required to run this algorithm was not performed, which would be important if this is designed to run on a remote device that is reliant on a finite energy supply. However, evidenced by the short computation times, it is
unlikely that SfM with PHE would have a dramatic effective energy usage for an aerial surveillance platform. Furthermore, energy usage could be improved via intelligent implementation of accelerations by low-energy graphical processing units such as an Nvidia Jetson.

Additionally, the method could be improved by implementing a method that further reduces the sizes of textures, as the majority of the reconstruction storage size is due to the size of the images. There are arguments to be made that if reducing the image is possible, then it should also be possible with the source imagery. While this is true, having access to the 3D aspect of the reconstruction means that much less texture information is required to convey the same meaning to an intelligence analyst as traditional 2D images.

Further effort to make the algorithm more robust, beyond just identifying and reconstructing vertical walls. SfM with PHE should be capable of handling a variety of geometries, and more importantly, completing the geometry of imaged objects, such that roofs and building corners are included is vital if SfM with PHE is to augment or replace current aerial surveillance techniques and reduce required bandwidth for that data. Included in that should be determining the feasibility of long-range sensory imagery for SfM, where the remote sensor is hundreds of kilometers from its target, like that of a satellite. Most imaging regimes will not involve a remote asset that is able to get in close with the target object.

Finally, real-world tests must be performed once much of the above is explored or implemented. While traditional SfM has proven effective for real-world objects, and there is sufficient reason to believe that SfM with PHE would perform similarly, it is also important to fully characterize the algorithms behavior and make potential tweaks to handle instances under non-ideal conditions, such as excessive shadows or glare.
Bibliography


19. Michael Waechter, Nils Moehrle, and Michael Goesele. Let there be color! Large-scale texturing of 3D reconstructions. In Lecture Notes in Computer Science
(including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), volume 8693 LNCS, pages 836–850, 2014.


4. TITLE AND SUBTITLE

High Resolution Low-Bandwidth Real-Time Reconnaissance using Structure from Motion with Planar Homography Estimation

6. AUTHOR(S)

Arnold, Christian M. A., 2d Lt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Intentionally left blank

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

Aerial real-time surveillance exists in a paradigm balancing the constraints of delivering high quality data and transporting data quickly. Typically, to have more of one, sacrifices must be made to the other. This is true of the environment in which an Unmanned Aerial Vehicle (UAV) operates, where real-time communication may be done through a low-bandwidth satellite connection resulting in low-resolution data, and serves as the primary limiting factor in all intelligence operations. Through the use of efficient computer vision techniques, we propose a new Structure from Motion (SFM) method capable of compressing high-resolution data, and delivering that data in real-time. Specifically demonstrating more than a 90% reduction of original video imagery size while operating at 4 Hz, which equates to an 80x computation time speed-up compared to traditional SFM methods, with an added benefit of presenting the original 2D intelligence data as a 3D virtual model.

15. SUBJECT TERMS

Structure from Motion; Modeling and Simulation; Geolocation; Intelligence, Surveillance, and Reconnaissance; Unmanned Aerial Vehicle; Algorithm Design; Homography

16. SECURITY CLASSIFICATION OF:

<table>
<thead>
<tr>
<th>a. REPORT</th>
<th>b. ABSTRACT</th>
<th>c. THIS PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>U</td>
<td>U</td>
</tr>
</tbody>
</table>

17. LIMITATION OF ABSTRACT

UU

18. NUMBER OF PAGES

105

19a. NAME OF RESPONSIBLE PERSON

Scott Nykl. PhD. AFIT/ENG

19b. TELEPHONE NUMBER (include area code)

(937) 255-3636; x4395; scott.nykl@afit.edu