High Performance Computation with Small Satellites and Small Satellite Swarms for 3D Reconstruction

by

Caleb Ashmore Adams

(Under the Direction of Dr. Ramviyas Parasuraman)

Abstract

In this thesis research I discuss the design and implementation of 2 Earth observation Cube Satellites with a focus on the computational methods used and the design of their computer systems. The satellite computer systems are tested by simulating imaging of single view observations and multiview observations. Observations are simulated by imaging existing 3D models of the Earth's surface in 3D rendering software. A custom computer vision library, known as SSRLCV, is used to compute the final 3D models which are then compared to the ground truth. Restrictions, unique to the space environment, are mitigated with a specialized operating system, hardware, and software. Tests are run on the Nvidia TX2 and TX2i with timing, state, and power usage tracking. The Nvidia TX2i GPU accelerated SoC is modified for use in a Cube Satellite and is used as the platform for high performance onboard computation. The results show accurate 3D reconstruction of the surface of Earth feasible within 15 to 100 meters, depending on the camera system and altitude, while maintaining favorable power usage and computation time.

Index words: Small Satellites, Cube Satellites, Computer Vision, System on a Chip, Graphics Processing, Edge Computing, 3D Reconstruction, Digital Elevation Models, Remote Sensing, Satellite Swarms

High Performance Computation with Small Satellites and Small Satellite Swarms for 3D Reconstruction

by

Caleb Ashmore Adams

B.A. Bachelors of Science in Computer Science, University of Georgia, 2018

A Thesis Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

Masters of Science

Athens, Georgia

2020

High Performance Computation with Small Satellites and Small Satellite Swarms for 3D Reconstruction

by

Caleb Ashmore Adams

Approved:

Major Professors: Ramviyas Parasuraman

Committee: David Cotten
Michael E. Cotterell
WenZhan Song

Electronic Version Approved:

Ron Walcott
Interim Dean of the Graduate School
The University of Georgia
May 2020

High Performance Computation with Small Satellites and Small Satellite Swarms for 3D Reconstruction

Caleb Ashmore Adams

April 30, 2020

Acknowledgments

In late 2015 when I started the journey to crowdfund a cube satellite I had no reasonable expectation of success. If it were not for the arduous work and inordinate dedication of my peers and mentors I would not be here today. At the start, those individuals were Hollis (Nicholas) Neel and Ryan Babaie. I also have Graham Grable, Paige Copenhaver, Megan Le Corre, Kenny Cochran, Khoa Ngo, Paul Hwang, Nirav Ilango, Adam King, Trent Walls and Jaicob Stewart to thank for their dedication in growing our team and helping write the proposals for SPOC and MOCI. A big thank you to Bob Pickney, who gave our group a place to work in the Entrepreneurship building and listened to our crazy dream in those early days. I thank the countless students who have dedicated their time to the UGA SSRL over the years and have helped shape it into what it is today. Thank you Dr. Marshall Shepherd for connecting our student team with the faculty at the Center for Geospatial Research. Without Dr. David Cotten, who spearheaded our integration with the research apparatus of UGA, the SSRL would be little more than a student club. David, thank you for dealing with my hard headed nature over these years; I am happy to call you a friend and mentor. Thank you to Dr. Deepak Mishra, who was willing to build the SSRL into more than just a student organization with David. Thank you Roger Hunter for being a friend and mentor and helping the UGA SSRL feel welcome in the NASA community. I cannot express my gratitude enough to those at the University Nanosatellite Program who took a chance on giving us the funding and taught me everything I know about space systems. Thank you

Jeff, Sarah, Shivani, Kate, Kyle, Cammi, Van, Lee, and Jesse; without your guidance I would not be where I am today. Thank you Hollis, Katie, James, Jack, Mary, Casper, and Kaelyn for pushing the SPOC mission to the end. Thank you Jackson, Alex Lin, Alexander, Alex Holmes, Justin, Allen, Godfrey, and Matt for ensuring that MOCI continues to succeed in UNP. Thank you Jackson, Aaron, and Hollis for helping develop the technologies to enable MOCI with me. Thank you Dr. Cotterell for being so supportive to your students; I am happy to have started computer science in your classes and to end with you as an advisor. Thank you Dr. Ramviyas for being so supportive, allowing me to feel at home in the HeRo lab, and helping me focus on this thesis. Thank you to my family, who supported me over these years and helped me proofread this thesis; I love you guys. Lastly, thank you Julie - you probably want me to stop writing this thesis by now.

Contents

Acknowledgments **1**

1 Introduction **9**

1.1 The Cube Satellite and Access to Space 12

1.2 Planetary Observation from Satellite Platforms 14

1.3 Computation in Space 17

2 Small Satellite Systems **19**

2.1 Systems and Subsystems Requirements 19

2.2 The Spectral Ocean Color Satellite 21

2.3 The Multiview Onboard Computational Imager Satellite 25

3 Design and Architecture of Space Computation **29**

3.1 The SSRL CORGI/TX2i 29

3.2 Space Operating Linux 38

4 Onboard Computer Vision **41**

4.1 Role of Systems Dynamics 42

4.2 Camera Systems 43

4.3 Feature Detection, Extraction, and Matching 47

4.4 3D Reconstruction 55

4.5 Bundle Adjustment 66

5 Distributed Computation with Satellites **75**

5.1 Networking Assumptions 76

5.2 Satellite Swarm Architecture 77

6 Experiments and Results **81**

6.1 Initial Feasibility with VSFM 82

6.2 SSRLCV Simulations with Blender 91

6.3 Pipeline Timing 104

6.4 Hardware Experiments 105

7 Conclusion **109**

7.1 Limitations of Experiments 112

7.2 Publicly Released Software 113

7.3 General Considerations for In-Orbit 3D Reconstruction 113

List of Figures

1.1 NASA budget over time 10
1.2 Launches per year 11
1.3 Various cube satellite sizes 14
1.4 3D mapping mission timelines 16
1.5 Cube satellite size comparison 17

2.1 Mission development timeline 20
2.2 SPOC payload overview 22
2.3 The SPOC satellite 23
2.4 The Concept of Operations 24
2.5 The MOCI satellite 26
2.6 MOCI's optical system 27

3.1 SSRL CORGI TX2/TX2i interface board 30
3.2 CORGI Board Schematic 31
3.3 TX2 exposed with no TIM 36
3.4 TX2 exposed with TIM 37
3.5 TX2 exposed with TPP and TIM 37
3.6 TX2 exposed with TPP and no TIM 38

4.1 Radial distortion cases 45

4.2 Matched point locations 48
4.3 Image gradients of Mount Everest 49
4.4 The epipolar line 51
4.5 Line generation and reprojection 56
4.6 Nview triangulation 61
4.7 Camera derivatives 71
4.8 Ideal error functions 74

5.1 Swarm join multicast 78
5.2 Cooperation scenarios 80

6.1 Cloud reconstruction 1 85
6.2 Cloud reconstruction 2 87
6.3 Simple geometry simulation 87
6.4 Initial VSFM reconstruction 1 88
6.5 Initial VSFM reconstruction 2 88
6.6 Dense subpixel distributions 89
6.7 Stereo disparity of Mount Everest 90
6.8 ASTER GeoTIFF to PLY 91
6.9 Mount Everest Multiview Imagery 92
6.10 Orbital angle diagram 93
6.11 2 view noise visualization 96
6.12 N-view noise visualization 97
6.13 Average Linear Error for 2 and N-view 98
6.14 Visualization of reconstruction accuracy compared to truth 100
6.15 2 view reconstruction with truth measurements colored 101
6.16 Bundle adjustment graphs 103

6.17 TX2 5-view power usage over time 1 106
6.18 TX2 2-view power usage over time 2 107
6.19 TX2i 5-view power usage over time 108
6.20 TX2i 2-view power usage over time 108

7.1 SSRLCV's MOCI pipeline 112

List of Tables

1.1 Various Cube Satellite Specs 13

3.1 Recommended Dunmore Aerospace Satkit parts 34

6.1 Dense SIFT error distribution values 90

6.2 Slew times for image acquisition 95

6.3 Average linear error with variable noise 97

6.4 Point density with variable noise 98

6.5 3D reconstruction accuracy 100

6.6 Bundle adjustment results 102

6.7 Pipeline runtimes 104

6.8 Nvidia TX2 / TX2i power modes 105

Chapter 1

Introduction

During the early years of space exploration, advancements in computer technologies paralleled, and in some cases even surpassed, terrestrial advancements. Though the first transistor, the point contact transistor, was developed at Bell Labs in 1947, it took over a decade to develop the technology for practical uses [1]. In 1959 Bell Labs developed a new design, the Metal Oxide Semiconductor (MOS) Field Effect Transistor (FET), which rendered small scale Integrated Circuits (IC)s viable [2]. In 1958, in a response to the launch of the Sputnik satellite, the National Aeronautics and Space Administration (NASA) was formed. During what is now known as the Apollo Era, president John F. Kennedy famously orated NASA's goal to land a human on the Moon and bring them back safely. Among many others, one of the most significant challenges for this goal would be to develop the Apollo Guidance Computer (AGC). The AGC was the first computer in history to use silicon ICs; it was implemented entirely with 3 input NOR gates produced by Fairchild Semiconductor, marking a monumental shift in computer history.

Famously, the Space Race ushered in a spike in funding for science, technology, engineering, and mathematics in the United States of America. Though NASA's funding peaked in 1966 during the Apollo program, the USA continued to see benefits from these advancements

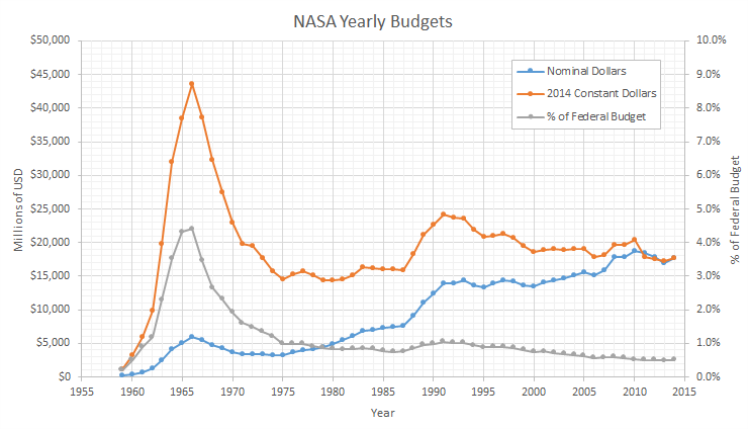


Figure 1.1: The NASA budget in Nominal Dollars, 2014 constant dollars (to adjust for inflation), and as a percent of the federal budget.

for many decades. Near the end of the Cold War, and certainly after the fall of the USSR in 1991, space exploration in the USA shifted towards a focus on sustainability, reusability, and robotic exploration. Human exploration was limited to the construction of the future International Space Station (ISS).

The ISS has become invaluable to the space research community. The completion of the ISS in 2011, though one could argue it was complete enough by 2001, has enabled unprecedented commercial and academic access to space research. The invention of the cube satellite in 1999 [3] and the advent of a usable space station can be thought of as the early enablers of what has now been termed *New Space* [4]. New Space is also characterized by an increase in launches to space (for the first time since the founding years of NASA) starting in the early 2000s. With increasing access to space via the space station, more and more hardware and software experiments were performed. Cube satellites, which certainly benefited from this increased access to space, grew symbiotically with the smartphone industry in the early 2000s - a trend of miniaturization that continues to this day.

Historically, burdensome communications constraints and poor onboard computational

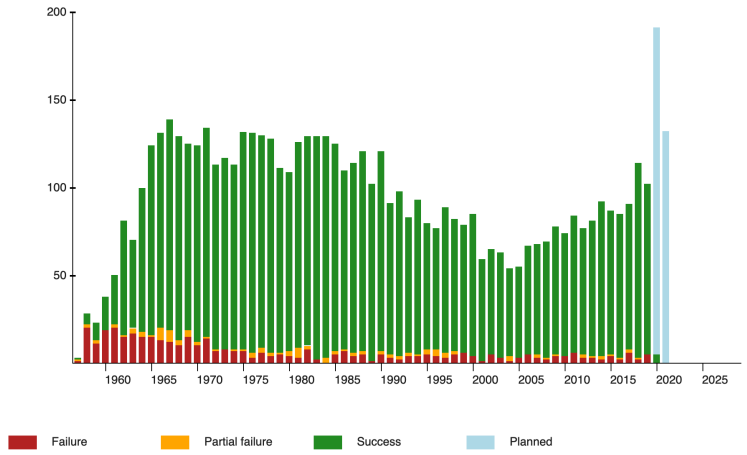


Figure 1.2: Total launches into space per year, after 1951.

power have forced satellite missions to focus on raw data generation. Though satellite sensors and instruments continue to improve, large sets of raw data are still transmitted back to ground operators on Earth to be analyzed [5]. This causes missions to transmit large amounts of data that may later go unused. Oftentimes the end products generated from such data are magnitudes smaller than the data as a whole. With the advent of commercial Internet of Things (IoT) devices, the improvements in smartphone computation, and the rise of the small satellite as a viable space platform, it is likely time to significantly rethink in-space computation. In the industry this is known as edge computation - generating useful information at the location where the data is collected. Only recently have new methods for extracting valuable data at the edge been explored, but these methods are traditionally deployed on terrestrial commercial electronics [6][7]. This research seeks to develop and explore edge computation with small satellites.

The solutions proposed within this research center around the development and use of a GPU accelerated System on a Chip (SoC) cubesat computer and the algorithms needed for their operation. The primary use of this computer system is to perform in situ Computer

Vision and Neural Network training onboard a cube satellite. Currently, the system is meant to be paired with a standard primary flight computer. The satellite computer system is to be tested by simulating imaging of single view observations, multiview observations, and swarm observations. In addition to hardware and architecture, benchmarks of performance will be measured by how well the system performs these observations.

1.1 The Cube Satellite and Access to Space

The Cube Satellite was jointly developed in 1999 by Cal Poly and Stanford, but the standards which define the weight of cube satellites are determined by the launch providers and the deployer manufacturers. Thus, over the years several standards have evolved based on the specifications of deployers and launch providers [3]. In many ways the Cubesat standards are, in the words of Hector Barbossa, "more what you'd call 'guidelines' than actual rules." These rules are released as public documents and CAD (Computer Aided Design) drawings within Interface Control Documents (ICD)s or Interface Definition Documents (IDD)s by the aforementioned organizations. To increase the likelihood of launch and to appeal to the maximum number of launch providers, most cube satellite developers begin by taking the strictest set of requirements from all available requirements to create the strictest set of guidelines to follow. Generally, cube satellites are built up of cube-like units where each unit, 1U, is generally 10cm $\times$ 10cm $\times$ 10cm and 1.33kg in mass. Additional units (1.5U, 2U, 6U) are not simply multipliers of a 1U as one might expect. Table 1.1 shows the variations of deployer specifications.

The most appealing property, and also the defining property, of the *small* satellite (i.e. the cube satellite) is its mass. A low mass is appealing because mass is the fundamental limitation when launching anything into space. Rockets, like you and I, obey the conservation of momentum; they are able to apply a net force to themselves by ejecting mass (rocket fuel)

Organization	Deployer	1U mass	3U mass	6U mass	1U size	6U size
Cal Poly [8] [9] [10]	PPOD	1.33 kg	4.00 kg	12.00 kg	$10cm\times$ $10cm\times$ $11.35cm$	$22.63cm\times$ $10cm\times$ $36.6cm$
NanoRacks [11]	NRCSD	2.40 kg	4.80 kg	8.40 kg	$10cm\times$ $10cm\times$ $11.35cm$	$22.7cm\times$ $10cm\times$ $34.05cm$
Rocket Labs [12]	MAXWELL	2.0 kg	5.5 kg	11.00 kg	$10cm\times$ $10cm\times$ $11.35cm$	$22.63cm\times$ $10cm\times$ $36.6cm$

Table 1.1: Variation on the Cube Satellite specifications.

at high speeds. The ideal rocket equation (1.1) considers that, in order to launch a payload of mass m_p from a rocket of mass m_r, one must also consider the mass of the propellent used to launch the payload, the velocity of the desired orbit, and the exhaust velocity of the engine v_e. The mass of the rocket filled with fuel can be considered m_f and the system's change in velocity to achieve the new orbit is Δv. The rocket equation is logarithmic and adding a small amount of payload mass drastically changes the final Δv, and therefore the final orbit.

$$\Delta v = v_e ln\left(\frac{m_f}{m_p + m_r}\right) \tag{1.1}$$

Launch providers operate within very strict mass tolerances for two primary reasons: to maximize fuel efficiency and to insure that the primary payload can achieve its desired orbit within certain bounds, often achieved with counter and dummy weights. Small satellites and cube satellites have historically been hitchhikers and were given launch opportunities only if the impact of their mass on the total rocket system was within a small tolerance. In some cases this additional mass is beneficial for launch providers, filling space that would

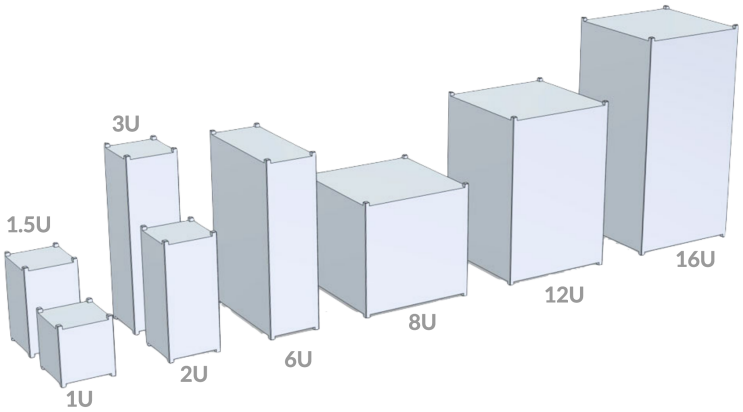


Figure 1.3: Various cube satellite sizes

otherwise be used for counterweights with paying customers. This means that cube satellite orbits are restricted to the desires of the primary payload. Most launches go into orbits which are classified as Low Earth Orbit (LEO) for ISS resupply and earth observation. All cube satellites, with the exception of the two MARCO martian cube satellites, are in earth orbit. Of those within earth orbit almost all are within LEO. LEO is classified as any orbit with an altitude of less than 2000km, for reference the ISS orbits at about 400 km.

1.2 Planetary Observation from Satellite Platforms

It is helpful to visualize an Earth observation satellite as a point projecting a rectangle, with rays extending like a pyramid onto the surface of the globe. Now consider the terms *nadir*, *ground-track*, *cross-track*, *swath*, and *in-track*. Nadir refers to the vector pointing directly "below" the satellite towards the Earth. Ground-track is the path that the satellite traces on the globe as it looks nadir. Cross-track is a direction, perpendicular to a ground track, of the projected pyramid on the globe. A satellite's swath is the measurement of this cross-track. In-track refers to the area the satellite observes as it covers area along the ground-track while

orbiting.

1.2.1 Optical Systems

There are two distinct optical systems relevant to this research: monochromatic imaging and multispectral imaging. Put simply, monochromatic imaging aggregates all wavelengths of light which enter the sensor into values of total intensity per pixel while multispectral imaging seeks to categorize different wavelengths of light into regions of intensities (the RBG cameras in our smartphones are examples of this).

Image resolution is measured with Ground Sample Distance (GSD), which represents the size of a single pixel on the ground. For example, if an image has a GSD of 1 meter and a resolution of 1024×1024 pixels, then the image covers an area of $1024m^2$. For a standard imaging system, one which is projective, this is a simple measurement of triangulation.

1.2.2 Global Digital Elevation Models

Digital Elevation Models (DEM)s are 3D models of a terrain's surface. Remote sensing literature also commonly mentions Digital Surface Models (DSM)s and Digital Terrain Models (DTM)s. The differences between these are purely semantic, at least to the engineer or computer scientist, and this research will thus use them interchangeably. Historical reference DEMs will include those generated by SPOT 5, DAICHI (ALOS), CARTOSat-1, and SRTM. Datasets will be prioritized first by public availability, and then by locational accuracy. Some datasets must be purchased. Note that the SRTM, radar based from the Space Shuttle, is a common source of elevation data, but is aging.

Figure 1.4 shows other satellites gathering DEM data contemporary with MOCI's lifetime. These can provide real-time data for comparison during the future lifetime of MOCI experiments. ASTER, a payload on the TERRA satellite, is a significant source of publicly

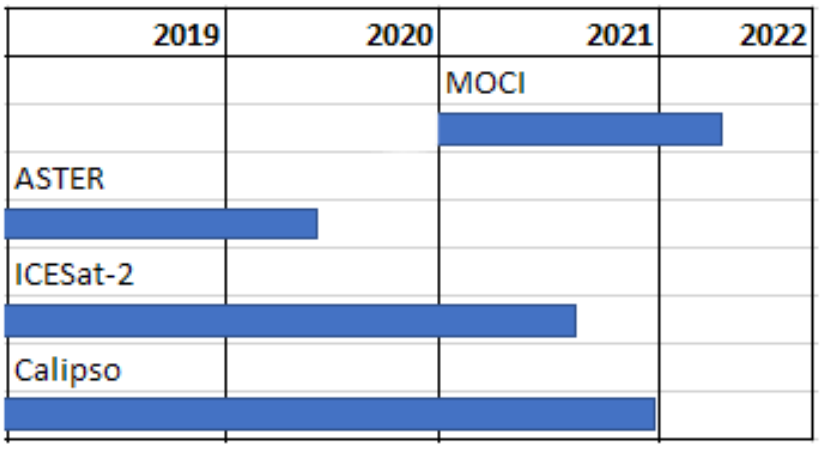


Figure 1.4: Other missions which produce digital terrain models, relative to the timeline of the MOCI satellite.

available DEMs. Note that Calipso will only be used for comparing DEMs of clouds.

1.2.3 Planet's Dove Constellation

Though traditionally most earth observation satellites have been large satellites, there is now significant competition from the small satellite world. The company Planet Labs (now just Planet) is flying a 3U earth observation cube satellite known as the DOVE. Planet has launched hundreds of these satellites, operating in what they term a "flock", over just the past few years. The orbits of these satellites are much lower than their competitors, meaning they re-enter the atmosphere much more quickly. This might seem at first to be disadvantageous, but the cost to launch and operate these cube satellites is minimal compared to the long development cycles of previous large systems, especially when considering the economics of scale in production. Additionally, Planet can slowly upgrade their flock with new technologies on the scale of months to years rather than decades. It is also interesting to consider that Planet could cover the globe with these satellites; in fact, it is their goal to image the entire

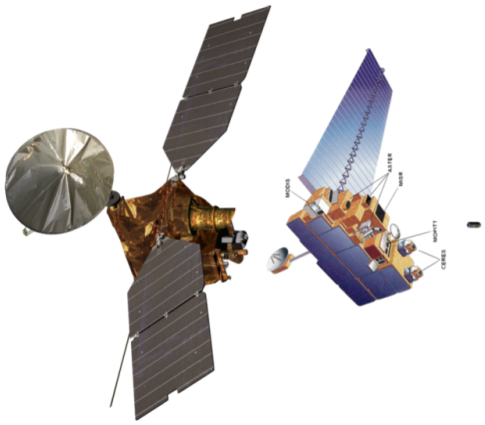


Figure 1.5: From left to right: the Mars Reconnaissance Orbiter, the Terra Satellite, a Planet Lab's Dove 3U cube satellite

earth at least once every day, something that has never been done before. Essentially, it is cheaper for Planet to have a factory like production line of small satellites which have 2-3 year lifespans and can cover the globe, rather than to spend 10 years developing a single satellite which will only see a fraction of the globe and use woefully outdated technologies by the time it launches.

1.3 Computation in Space

The title of this thesis mentions *high performance* computation. I by no means wish to imply that I have put a server grade super computer into space (yet!). The term high performance, in this case, instead refers to the fact that the systems and methods designed and developed within this thesis are scalable and written for the same architectures of super computers. Edge computing, which is the focus of this research, is not typically preferable to centralized methods when very few edge computers are used. Essentially, this research aids in enabling the new area of high performance edge computing in space.

There are, however, serious efforts to move *towards* utilizing server grade supercomputers

in orbit, but these efforts are limited to the International Space Station (ISS). The Center of Space, High-performance, and Resilient Computing (SHREC), a National Science Foundation (NSF) funded multi-university collaboration, focuses on developing such technologies for demonstration on the ISS. One example, and there are many from this collaboration, is the Spacecraft Supercomputing for Image and Video Processing (SSIVP) prototype [13].

Chapter 2

Small Satellite Systems

Designing a satellite, even a cube satellite, involves many complicated and interdependent pieces of hardware and software. It is due to this complexity that the study of designing satellites falls into the interdisciplinary fields of systems engineering and aerospace engineering. While the focus of this thesis is not on my contributions to the systems engineering of the satellites at the University of Georgia (UGA) Small Satellite Research Lab (SSRL), it is important to cover the basics that will be assumed knowledge in further chapters.

2.1 Systems and Subsystems Requirements

The design process of the SPOC and MOCI satellites began, like most satellite missions do, with the generation of high level mission requirements. From these high level mission requirements, systems requirements and then subsystems requirements are generated. In addition to the generation of requirements, satellite systems and space systems typically have a set of reviews that are intended to increase in complexity and aid in the finalization and implementation of designs and requirements [14] [15] [16]. At the beginning of the UGA SSRL the SPOC and MOCI satellites were a single mission, but later diverged thanks to

additional funding. In fact, the initial acronym for the MOCI satellite was *Mapping and Ocean Color Imager*. After the funding of the SPOC satellite and the subsequent division of mission objectives, MOCI was renamed and took on the 3D surface mapping mission objectives while SPOC took on the multispectral coastal imaging objectives. This is a prime example of how the systems engineering process works and how a focus on high level objectives can allow for specialization.

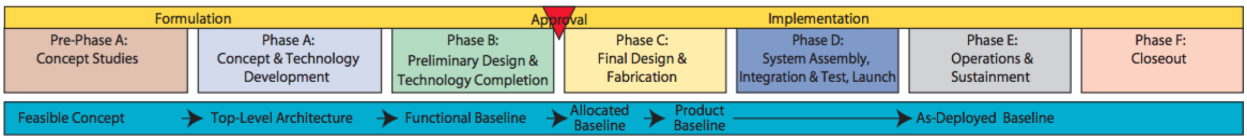


Figure 2.1: A typical timeline of cube satellite development

There are many ways to break down the complexity of satellite missions, all of which involve breaking up primary components into systems and subsystems based on requirements. At the UGA SSRL we have systems and subsystems as loosely defined in the following list. All systems, other than the payload, are considered *Core Avionics* and the payload is considered mostly isolated from those systems. This is done to eliminate essentially duplicate work between the MOCI and SPOC missions, though their core avionics do differ slightly.

- **OnBoard Computer (OBC):** The "brains" of the cube satellite
- **Electrical Power System (EPS):** The power distribution and regulation system of the satellite; solar panels are included
- **Batteries:** The physical batteries of the satellite, typically with integrated heaters
- **UHF Transceiver (UTRX):** The transceiver which operates on UHF for commands and telemetry
- **S-Band Transmitter (STRX):** The transmitter which operates on S-Band for payload data transmission

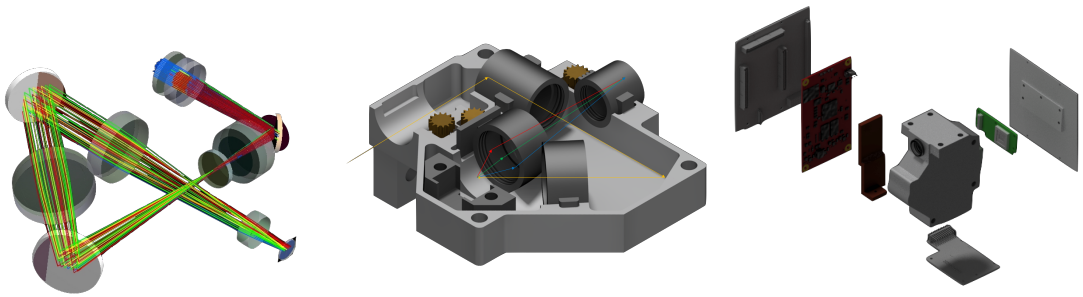
- **Antenna:** Antenna which interfaces to the UHF and S-Band systems
- **Attitude Determination and Control System (ADCS):** The system which provides position and orientation data as well as actuation and control
- **Payload:** The primary payload of the mission, the other systems should help this one function
- **Optics:** The optical assembly within the cube satellite
- **Frame and Housing:** The structural and physical components into which the electronics integrated. It includes thermal and radiation protection measures

During the development process it is common to build what is known as a *flatsat*. A flatsat is an engineering test unit which contains engineering level components for each system / subsystem in the satellite. The flatsat is used to develop and mature hardware and software for the cubesat.

2.2 The Spectral Ocean Color Satellite

The Spectral Ocean Color (SPOC) Satellite is a 3U cube satellite and the second mission of the UGA SSRL. The mission is funded via the NASA USIP (Undergraduate Student Instrument Project) and supported via the NASA CSLI (Cube Satellite Launch Initiative) and NASA ELaNa (Educational Launch of Nanosatellites) Program. It will be the first UGA SSRL mission to launch, with an expected launch date around summer 2020. SPOC has been under development for about 4 years and over that time scores of students have contributed greatly to significant portions of the mission. I was involved heavily in the initial creation of the mission, but can take little credit for the success of the optical instrument at the center of the mission.

The SPOC mission is focused around the development of a moderate resolution multi-spectral sensor. The cubesat is designed to acquire imagery of coastal ecosystems and ocean color across a wide range of spectral bands within the visible and near infrared. This mission directly supports NASA's 2014 strategic goals and objectives, specifically the Strategic Goal 2 to "advance understanding of Earth and develop technologies to improve the quality of life of on our home planet" and Objective 2.2 to "advance knowledge of Earth as a system to meet the challenges of environmental change, and to improve life on our planet". The primary sensor is known as *SPOCeye*, a nod to its similarity to the HawkEye sensor on the University of North Carolina-Wilmington's SeaHawk cube satellite. The SPOCeye sensor was developed jointly by UGA and Cloudland Instruments, who also aided in the development of the HAWKeye instrument.



(a) Ray traces of the spectral bands in the SPOCeye optics

(b) A late 2017 iteration of the SPOCeye optical housing

(c) The electronic of the SPOCeye instrument

Figure 2.2: The SPOCeye payload (an older 2017 breakup), around which the SPOC mission was built

The SPOCeye's sensor, attached at the focalplane of the optical train, is a 752×480 CMOS array. The optical train is essentially telescopic in design and spreads wavelengths between $433 - 866$ nm across the sensor vertically, resulting in a spectral resolution of 1.042 nm per row. The onboard picozed board, seen in figure 2.2c as the red component, bins

4 rows together to create a resultant spectral resolution of 4.16 nm. Additionally binning after this is optional, though we typically bin further to improve our signal to noise ratio for given spectral regions. The SPOCeye is limited to only selecting 16 bands at a time to save. We have defined SPOCeye to be an adjustable multispectral sensor, rather than a simple multispectral sensor, due to our ability to select any 16 bands within our spectral range. Furthermore, the sensor operates as a pushbroom scanner and must orient such that the spectral rows of the SPOCeye are parallel to the cross track and perpendicular to the in track / flight direction.

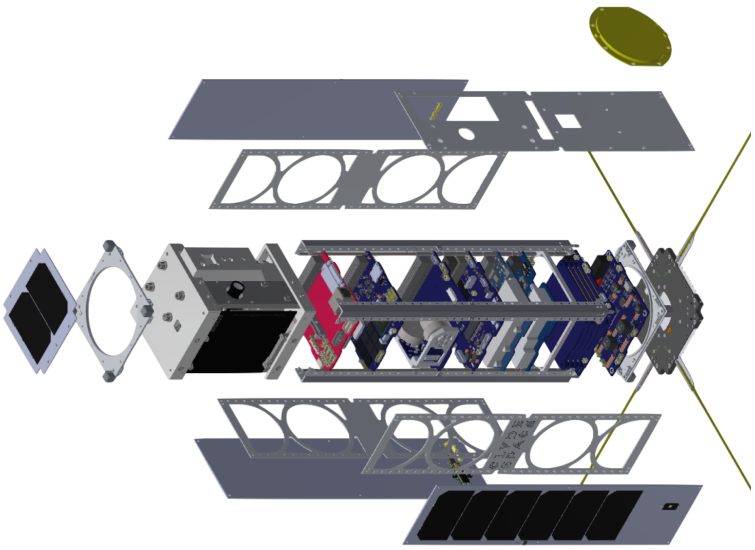


Figure 2.3: An expansion of the SPOC satellite, revealing the internal components of the cubesat.

The majority of the core avionics of the SPOC satellite are commercially available components from cube satellite vendors Clyde Space (now AAC Clyde) and ISISpace; these components can be seen in figure 2.3 along with the primary payload. The mission objectives of the SPOC satellite include monitoring the status of the coastal wetlands, the quality of estuarine water including wetland biophysical characteristics and phytoplankton dynamics, and the productivity of the near-coastal ocean. Such topics are covered in detail in Hollis Neel's thesis on the SPOC satellite [17].

To the end user, SPOC operates similar to a finite state machine. The states of the satellite are called modes and each mode has several sub-modes and transitions. Modes are designed with two ensure safe operations and to meet the mission requirements with minimal complications. The modes can be seen in detail in Figure 2.4. **Cruise Mode** is the nominal mode of the satellite and is designed as such so that the satellite is always power positive and in communication over the UHF radio. **Scan Mode** can be thought of as the data gathering mode of the SPOC satellite in which the craft performs a pushbroom scan of a target area. SPOC also contains a simple RGB imager which can be used as a view finder for geo-referencing imagery and for taking sample imagery. The simple imager is used in **Image Mode**, which operates similarly to the scan mode. **Data Downlink** mode takes advantage of the S-Band transmitter to downlink higher volumes of data to the SSRL ground station. **Safe Mode** is an emergency mode in which the satellite goes into a minimal power state and uses UHF comms to allow ground operators to debug the craft. An additional mode known as **Deployment Mode** is used once in the special case when the satellite is deployed from the International Space Station. This mode implements NASA required timers, delayed subsystem powerups, and communications delays until the satellite is far enough away from the station.

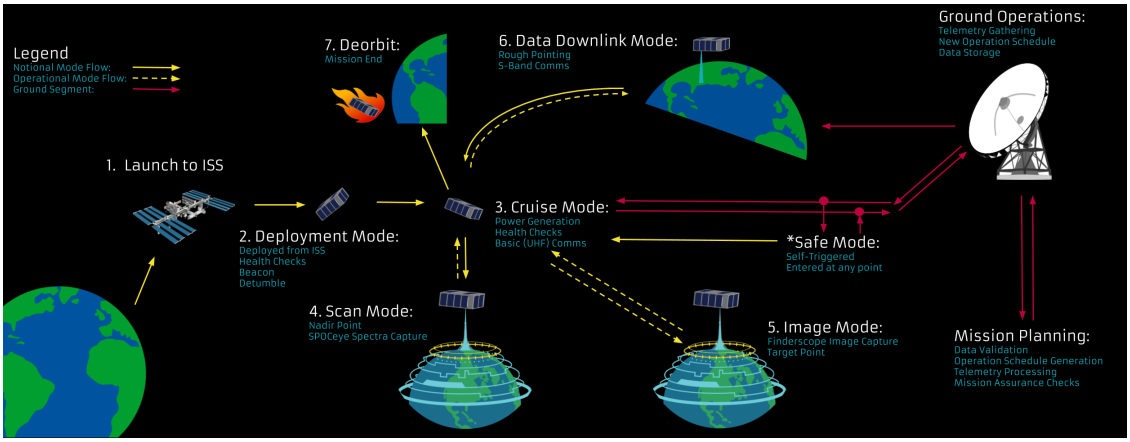


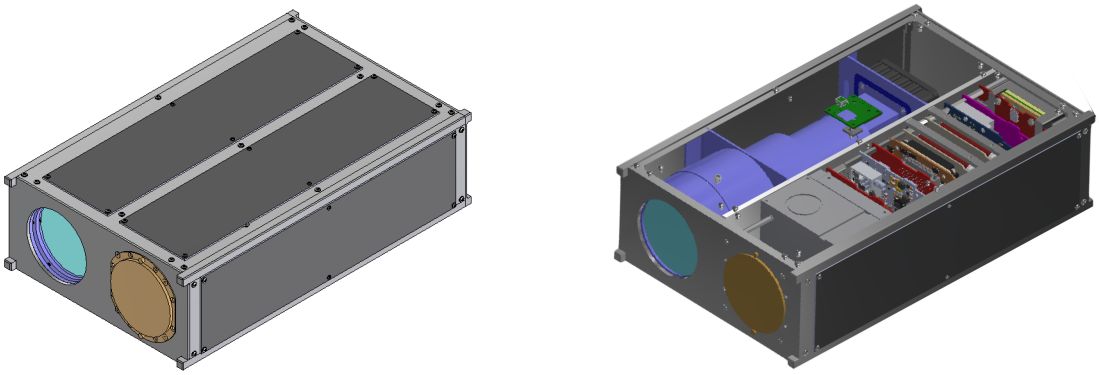
Figure 2.4: The Concept of Operations of the SPOC satellite

2.3 The Multiview Onboard Computational Imager Satellite

The Multiview Onboard Computational Imager (MOCI) satellite was the first funded mission of the UGA SSRL, but will be the second mission to launch. MOCI started design in early 2016 as the *Mapping and Ocean Color Imager* satellite, essentially combining the SPOC mission into it. MOCI later evolved to focus purely on in-situ computer vision. MOCI is funded by the University Nanosatellite Program (UNP) and supported by the Air Force Research Laboratory (AFRL) and is currently in the late stages of design with assembly scheduled for Summer 2020.

The MOCI mission will acquire imagery of the Earth's surface from (Low Earth Orbit) LEO and perform 3D surface reconstruction at a landscape scale using custom algorithms and modified off-the-shelf, high performance computational units. The MOCI mission will also identify and map or image surface objects, including but not limited to coastal environment phenomena, while training students in STEM-related fields. Efficient data compression, feature detection, feature matching, and SfM processing techniques of space-based imagery will be performed on board the spacecraft as a proof-of-concept of high performance, onboard processing capabilities. 3D models produced by the MOCI satellite will take the form of Point Clouds and Meshes as their end product for quick data downlink. The secondary goals of the project will be to lay the foundations for a self-sustaining Small Satellite Research Laboratory (SSRL) at the University of Georgia (UGA) involving numerous students and creating a free repository of imagery.

The data collected from MOCI will be processed onboard the satellite within the space environment and will likely be one of the first of its kind on a small satellite platform. This involves developing algorithms and technologies capable of detecting and matching features from multiple images, localizing features in 3D space, and computing accurate depth



(a) The 6U MOCI satllite (b) The exposed internals of MOCI

Figure 2.5: The MOCI satellite's 6U design

from point correspondence. By using high performance processors with these algorithms the amount of data and time needed to downlink completed data products will be greatly reduced.These goals align MOCI with the 2010-30 Air Force Science and Technology Horizon themes that can maximize capability superiority. Our system advances research by helping shift from Platforms to Capabilities, Control to Autonomy, Permissive to Contested domains, and Sensor to Information [18].

The internal hardware of the MOCI satellite is very similar to the internal hardware of the SPOC satellite; the key differences are with the Payload and ADCS systems. In the next chapter significant attention is given to the GPU SoC system onboard the MOCI satellite. The MOCI payload also consists of a 3U optical tube designed by RUDA cardinal which produces 4K imagery at a GSD of approximately 6.5 meters per pixel. The optical system has a field of view of approximately ± 2.4 degrees, which means strict requirements have been placed on pointing and controls to ensure image overlap is possible for 3D reconstruction. If image overlap cannot be guaranteed, then 3D reconstruction will not be possible. At the time of this thesis research, the MOCI ADCS is still undergoing finalization and the results

presented here are partially intended to solidify particular pointing requirements. The MOCI satellite intends to perform slew maneuvers to achieve multiple views of ground targets; the feasibility of such maneuvers is covered in the results of this thesis. Additionally, alternatives to slew maneuvers are also demonstrated. Prior to the research conducted here, the efficacy of particular slew maneuvers, or other satisfactory multiviews needed for 3D reconstruction, was largely speculation based on prior results from robotics and traditional planetary 3D reconstruction methods.

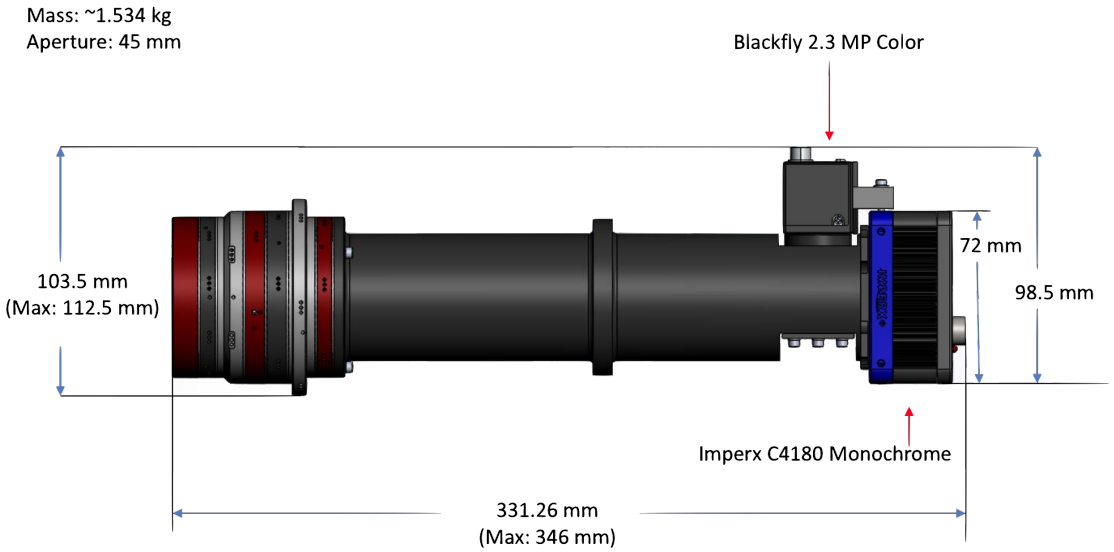


Figure 2.6: The optical assembly of the MOCI satellite, taking up 3U of the craft as seen in 2.5b.

The optical system of the MOCI satellite has changed significantly over its several years of development. The original optical system of MOCI, detailed in chapter 6 of this thesis, had a 60 meter GSD and only proved feasible for reconstructing objects on the scale of 10 km. The current optical design of the MOCI satellite was developed in a partnership with RUDA-Cardinal, who are the true optical experts. The system is designed around two primary imagers that have been used for space applications before. The optical system can be seen alone in figure 2.6 and integrated into the MOCI satellite in figure 2.5b. The resultant field of view of the optical system is $\pm 2.4^{\circ}$ with an effective focal length of 270 mm. The

effective focal length differs from the actual focal length in that it is not the true distance a ray (or photon) takes as it travels through an optical system. The effective focal length is important because it is used in software when modeling the camera system. The light, as it approaches the end of the optical tube, is split between an 80/20 beam splitter which sends 80% of the light to the primary sensor and 20% of the light to a secondary sensor. The primary sensor of the MOCI satellite is the Imperx C4180, a 4096×3072 monochrome sensor, which can be seen integrated at the far end of the optical tube in figure 2.6. The secondary camera is the FLIR Blackfly, which is a 648×488 RGB imager for use in neural network tests (not covered here) as well as point cloud coloring. The primary imager results in a GSD of about 6.5 meters.

Chapter 3

Design and Architecture of Space Computation

During the formative years of the UGA SSRL there were no existing solutions to our computational needs. In late 2016 this caused me to begin exploring methods to accelerate our onboard computational abilities with teams at the SSRL, which eventually led to the Nvidia Jetson family of products. The research here outlines the efforts to space rate and qualify a GPU accelerated SoC for use in LEO and is heavily influenced by my IEEE Aerospace Paper [19].

3.1 The SSRL CORGI/TX2i

The University of Georgia's Small Satellite Research Laboratory (SSRL) is utilizing a Nvidia TX2i Graphics Processing Unit (GPU) within the Multi-view Onboard Computational Imager (MOCI) Satellite. Although the system utilizes a GPU, an additional On Board Computer (OBC) is still required for control and communication with core avionics. The UGA SSRL has developed a board, the Core GPU Interface (CORGI), that is capable of inter-

facing the Nvidia TX1, Nvidia TX2, or Nvidia TX2i into a PC/104+ compliant CubeSat [20][21][22].

The GPU (Nvidia TX2/TX2i) being used is a complete System on a Chip (SoC), capable of running GNU/Linux on an ARMv8, with a 256 core Pascal GPU. For an exhaustive list of TX2/TX2i capabilities see the user guide [23]. Currently, the TX2/TX2i utilizes CUDA 10.0 [23].

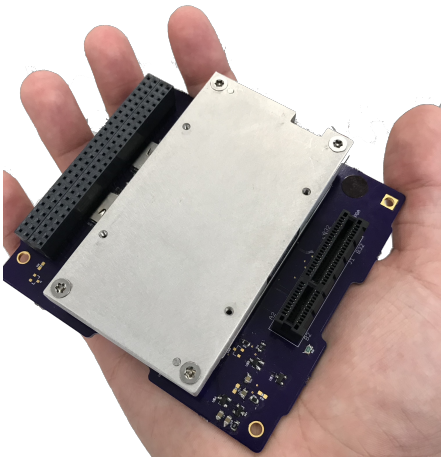


Figure 3.1: Me holding a previous iteration (2017 version) of the space ready TX2/TX2i concept, known as the CORGI (Core GPU Interface).

Additionally, I sought to adhere to the IEEE Std 1156.4-1997 standard for spaceborne computer modules. This standard provides requirement levels for thermal performance, pressure, shock, vibration, and radiation [24].

3.1.1 The Nvidia Tegra SoC Family

Unsurprisingly, the Nvidia Tegra product line is primarily intended for use in robotics and terrestrial edge computing, not in space computation. In fact, when I asked about this on the Nvidia forums two years ago, I was essentially met with the response, "Are you actually trying to do this?" [25]. The motivation is mostly because the Nvidia Tegra product line has many desirable properties: the CUDA programming language is the most popular GPU

programming language for research; it has a widely supported developer base; it is deployed in many similar terrestrial applications; and it requires less electrical engineering to integrate onto an interface board. While some accelerated platforms, such as Unibap's AMD GPU based computer [26], choose to build their computer architectures more or less from scratch we have chosen an interface board approach to speed up development time.

3.1.2 Pinout and Interfacing

The definitive pinout of the CORGI/TX2i system has changed drastically over the years, and I suspect that it will continue to change as it is developed further by students after me. This section is meant to give an overview on the goals and challenges of interfacing and should not be considered comprehensive. Internal UGA SSRL documentation should be consulted for current CORGI pinouts and interfacing.

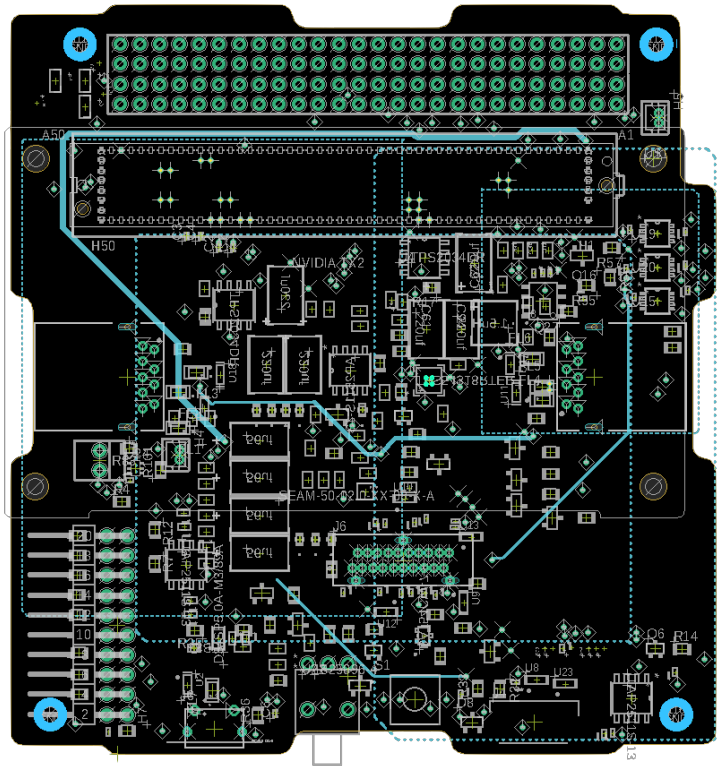


Figure 3.2: The CORGI board schematic, seen as version 3.1.2.

The CubeSat form factor is the primary consideration in the design of the CORGI. The form factor complies with available deployers such as the Poly Picosatellite Orbital deployers (P-POD), and the JEM Small Satellite Orbital Deployer (J-SSOD), and is modular to accommodate the payload and support I/O both spatially and functionally [27][28]. The CORGI board (see Figure 3.1) was minimally designed to meet our mission and development requirements. However, the design still required optimization and additional features to provide a comprehensive accelerated heterogeneous computing platform [26]. CORGI has useful features such as bi-directional logic shifters to convert the 1.8V (CMOS) logic from the onboard Nvidia Jetson TX2i [23] to peripheral logic. Shifters are required for serial data transfer, GPIO flipping, and IC enabling. The CORGI will maintain a PC/104+ form factor (90 x 96 mm)[22] with an ergonomic cutout to accommodate the high speed LVDS expansion header (Samtec LSHM-120-04.0-L-DV-A-N-K-TR) mounted beneath the TX2i module. Tight integration of both CPU and GPU allows support of all aspects of satellite performance, such as power, command and data handling, attitude determination, and payload.

As shown in Figure 3.2, H1 and H2 are the main PC/104+ CubeSat headers. These route peripheral components onto the bus, as well as routing the PC/104+. The H1 and H2 headers on the CORGI also act as pass-throughs, so there is no break in communication or power. In addition to an RJ45 Ethernet receptacle, the CORGI board also routes a display port for ground testing and has support and 2x USB Type C for debugging the TX2i.

3.1.3 System Hardware Mitigation and Shielding

For radiation and thermal mitigation in LEO we recommend utilizing the Dunmore Aerospace "Satkit" which contains the standard STARcrest MLI materials cut into manageable sizes for small satellite developers. This allows the development of thermal protection blankets according to various mission requirements. The kit includes an outer layer material, inner

layer material, first surface tape, and clear polyimide tape[29]. This kit is optimal for small satellite systems as it is small, cheap, and easily customizable. More specifications of the materials are listed below (See table 3.1) and values were calculated from the Solar Radiation, Earth's IR radiation, and Albedo Radiation equations [30].

$$q = G_s \alpha_s \cos\phi \tag{3.1}$$

$$q = \sigma T_e^4 \alpha_I R F e \tag{3.2}$$

$$q = G_s (AF) \alpha_s F e \cos\theta \tag{3.3}$$

The heat flux due to the LEO sources using equations (3.1), (3.2), and (3.3) was calculated with the help of data from literature [30]. Values have an expected error of ± 0.4. Values for the solar radiation were determined using the mean values provided by data from the World Radiation Center in Davos, Switzerland [31].

The necessary thickness of the radiation shield was calculated in part using values provided by Dunmore. The primary source of heat flux is solar and these values vary on an annual basis due to the an elliptical orbit. This means that the maximum and minimum amount of flux expected from the Sun would range between 1322 and $1414 w/m^2$. The target ambient temperature inside the CubeSat is 293.15 K (20 degrees C). This is calculated with the following formula:

$$Q = \frac{\Delta T K A}{L} \tag{3.4}$$

Radiation is calculated with equation (3.4), where Q is the heat flux into the system, K is the thermal conductivity of the material, A is the area in m^2, and L is the thickness of

the radiation shielding. Assuming a thermal conductivity of 0.014 and a ΔT of $101K$, the expected required thickness of the radiation shielding is $1.03438 \cdot 10^{-7} m^2$. Radiation from Free Molecular Heating (FMH) was determined to be negligible for the stage in which the CubeSat would be launched from the ISS. FMH is almost exclusively encountered during launch ascent just after the booster's payload fairing is ejected.

VDA / 200 GA Polyimide	
Tensile Strength	24000 psi
Elongation	50%
Thickness	50.8 micron
Density	1.42 g/cc
Yield	13.8 $m \cdot m/kg$
Weight/Area	72 g/m^2
Operating Temp	-250 - 400 C
Metalization	99.99 % pure aluminium
VDA / 200 GA Polyimide	
Tensile Strength	26000 psi
Elongation	110 %
Thickness	6.35 micron
Density	1.39 g/cc
Yield	124.9 $m \cdot m/kg$
Weight/Area	8 g/m^2
Operating Temp	-250 - 150 C
Metalization	99.99 % pure aluminium
VDA / 25 GA PET / VDA, Embossed & Perforated	
Thickness	76.2 microns
Yield	10.4 $m \cdot m/kg$
Weight/Area	96 g/m^2
Operating Temp	-40 - 220 C
Metalization	99.99 % pure aluminium
100 GA Polyimide / 966 PSA	
Thickness	76.2 microns
Yield	10.4 $m \cdot m/kg$
Weight/Area	96 g/m^2
Operating Temp	-40 - 220 C
Metalization	99.99 % pure aluminium

Table 3.1: Recommended Dunmore Aerospace Satkit parts

3.1.4 Radiation Mitigation

The radiation shielding thickness (See Equation (3.4)) is also driven by the 1997 IEEE Standard for Environmental Specifications for Spaceborne Computer Modules [24]. We design for level I radiation in preparation for the LEO environment. An additional benefit of the Dunmore Aerospace "Satkit" is that it meets this standard while accruing minimal mass gains.

Proper shielding does not have to incur heavy mass gains, and in fact shielding that is too thick can increase the effects of some kinds of radiation events. This is due to the higher levels of secondary particles created when a high-energy GCR particle impacts a thick shield [32]. If necessary, due to mass limitations, a properly designed shield may also act as a heat sink. This reduction of a device's operating temperature can greatly reduce the risks posed to that device by radiation [32].

When working with Commercial Off The Shelf (COTS) electronic components for use in the space environment, some of the problems that must be addressed are data integrity, performance, and accuracy in high radiation environments. In our design the TX2i is one of the more vulnerable parts with respect to this due to its highly dense hardware design [33]. As stated above, much can be done at a hardware level but software mitigation is often needed.

3.1.5 Thermal Mitigation

Simulations were conducted on the Jetson TX2, not the Jetson TX2i. The two boards, however, utilize the same Parker Series SoC and therefore have identical thermal properties within that region. Simulations have shown, given a large enough mounting structure, that the Nvidia TX2 is capable of dissipating heat effectively. To achieve this goal it is imperative that the TX2's Thermal Transfer Plate (TTP) is adequately interfaced into the heat

dissipating mass [20]. To interface the TTP with the heat dissipating mass, we highly recommend the use of a low thickness and high conductance thermal interface material (TIM) that adheres to the NASA standards for collected volatile condensible materials ($\leq$ 0.1% CVCM) and total mass loss ($\leq$ 1% TML) [34]. Previous findings have suggested that the Carbice Space TIM is ideal for these purposes due to its low thickness ($0.065mm$) and high conductance ($13,330Wm^{-2}K^{-1}$) [20].

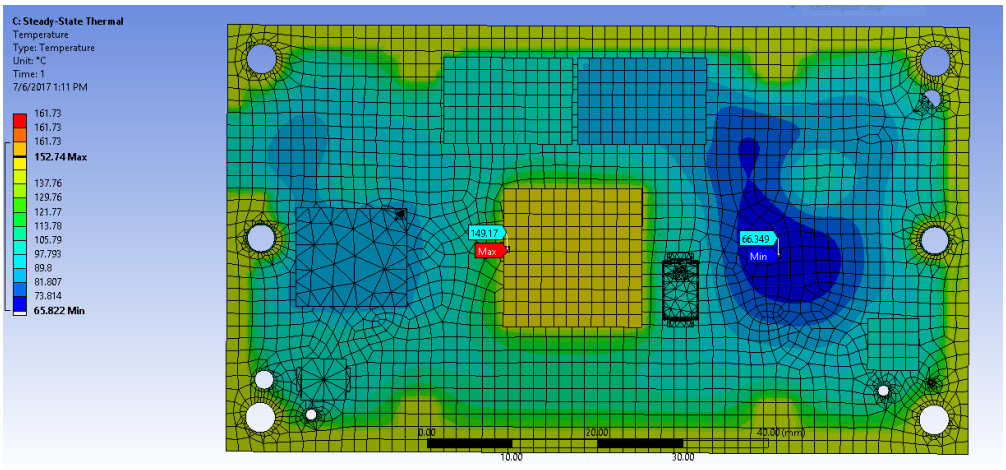


Figure 3.3: The Nvidia TX2 daughter-board with the Parker Series SoC exposed. No TIM is used in this simulation.

It is important to note that all thermal simulations are run with a heat load according to subsystem power draws and assume 0% efficiency for a worst case scenario. The steady state thermal simulation clearly shows that the GPU's core temperature has been lowered with the addition of the TIM. The overall maximum has been brought down from around 160°C to under 50°C. While this model did not include the entire spacecraft structure as a conduction medium, this would only serve to decrease the temperature further, as conduction is a much faster heat transfer mechanism than radiation.

However, some caution must be used with these results. The thermal environment of the satellite is inherently transient, so the ambient temperature assigned to the model here is likely inaccurate. This also means a "steady state" analysis might not necessarily be

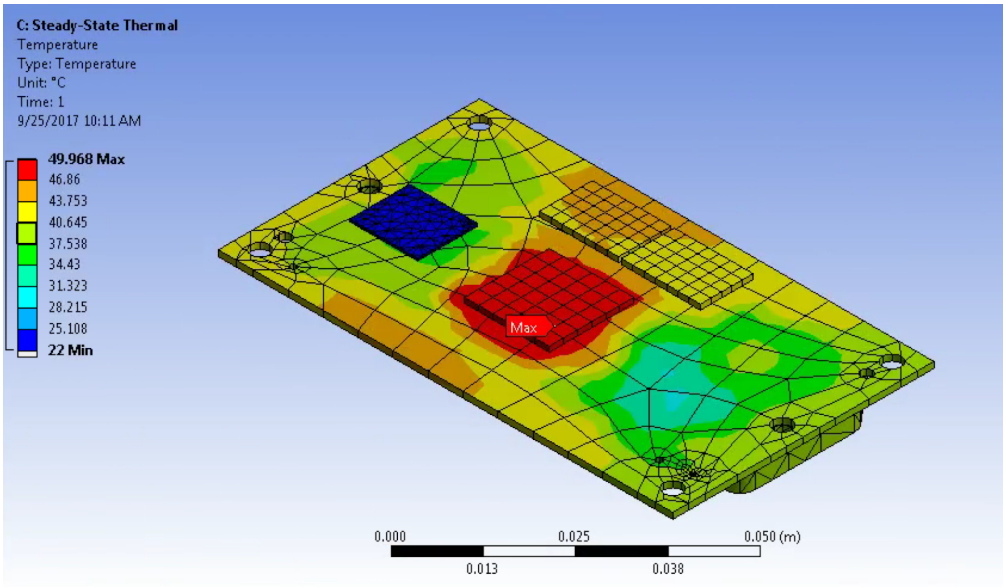


Figure 3.4: The Nvidia TX2 daughter-board with the Parker Series SoC exposed. TIM is used in this simulation

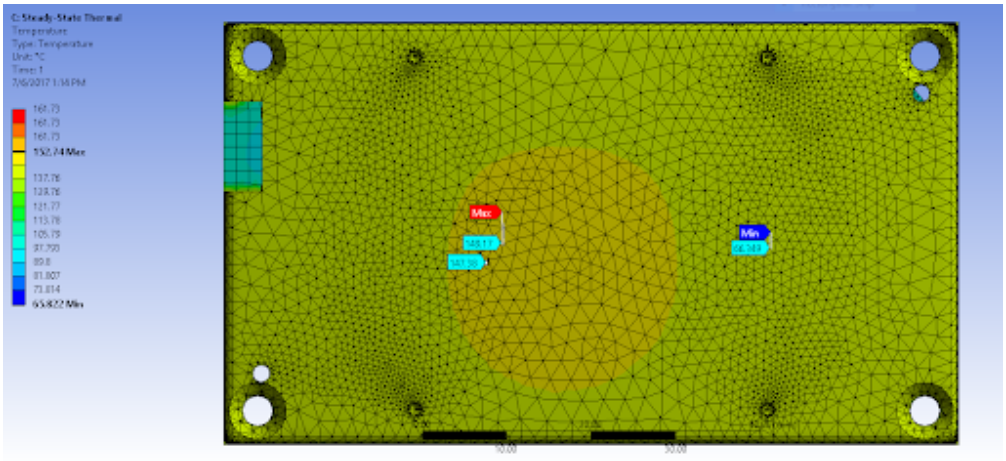


Figure 3.5: The Nvidia TX2 with TTP integrated and TIM used.

appropriate. However, the purpose of this analysis is not to provide conclusive thermal information about the TX2i/CORGI, rather to serve as a data point for design, and to show that under extremely approximate conditions, the TX2i/CORGI will be able to operate in orbit.

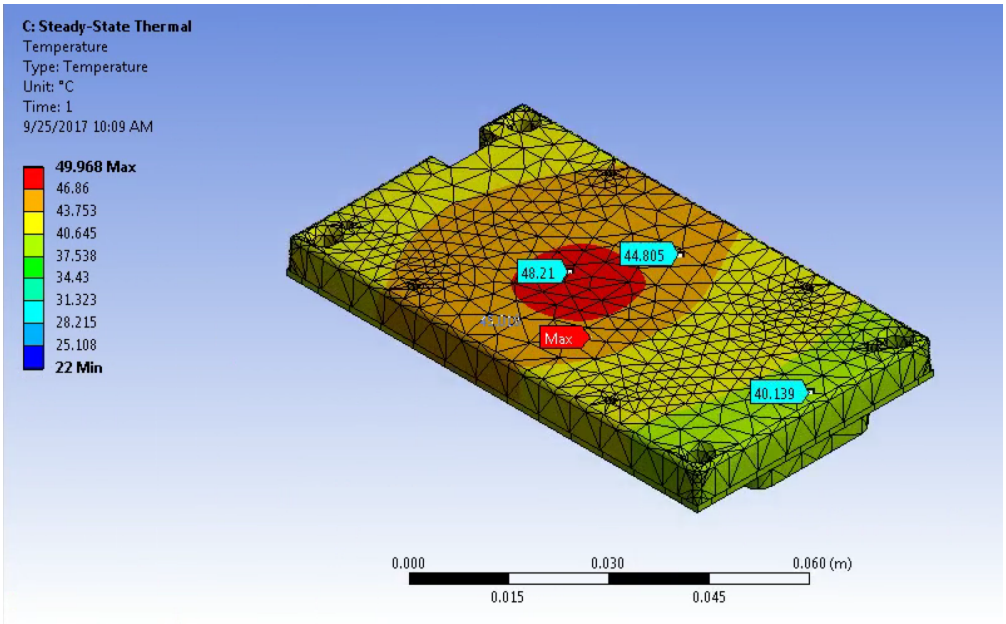


Figure 3.6: The Nvidia TX2 with TTP integrated, no TIM is used.

3.2 Space Operating Linux

The Nvidia Tegra line of products includes the Jetson TK1, TX1, TX2, TX2i, and Nano. This section only focuses on the TX2, TX2i, and Nano. All of Nvidia's Tegra products use a specialized fork of Ubuntu 16.04 called Linux for Tegra (L4T). Though L4T is well optimized for most terrestrial applications, it is not well suited for space applications. Thus, we have sought to modify L4T in the following ways: by implementing a RAM based file system for use on the TX2i, TX2, and Nano, by implementing a triplicated boot loader so a stable operating system can boot, and by enabling real time scheduling in the L4T linux kernel. The implementation of a RAM based filesystem is due to the fact that the system's RAM is much more resilient to radiation than a micro SD card. The triplicated bootloader is for a similar reason; because radiation is expected to degrade the operating system image, we seek to employ triple modular redundancy (TMR) at the bootloader level

as a mitigation technique. Lastly, real time scheduling is desired because operations in a space environment are extremely time sensitive. Instead of allowing the default scheduler to loosely time execution, we want to have start and stop time guarantees when running software on the TX2i / CORGI system. To avoid needless work, and to quickly test the goals of our custom operating system, we will use the Yocto Project's build tools. Yocto is an open source set of tools that helps developers create custom Linux-based systems. We will incorporate and slightly modify the open source U-boot bootloader to achieve TRM at the bootloader level. Lastly, we will utilize the PREEMPT_RT real time linux patch for the kernel.

The resulting specialized operating system is known as Space Operating Linux (SOL) and is a joint collaboration between the John's Hopkins Applied Physics Laboratory (JHU APL) and the UGA SSRL. The goal is to make a stable and minimal real-time scheduled linux system for a space-modified Nvidia TX2i GPU/SoC. The operating system will be based off of Ubuntu and L4T and will allow for responsible GPU computation in a space environment. The custom build is available on github or the UGA SSRL gitlab.

One of the major problems radiation presents for software is its effect on memory. Over time as bit flips [30] [24] aggregate, they will corrupt information and make some systems unusable. To address these issues, I have led an effort to modify the file system and bootloader. I use the principle of TMR to automatically correct damage. In addition to this, we will use the U-Boot bootloader software to modify the bootloading process with TMR [35]. The bootloader will store 3 copies of the operating system (OS) image for the TX2i and a hash for each image. At boot time, the bootloader will recalculate the hash for each OS image it attempts to load and compare the calculated hash against the stored hash to determine if the OS has been corrupted. If corruption is detected, then the boot loader will attempt to load the next OS image. If all OS images are determined to be corrupted, the bootloader shall attempt to construct an uncorrupted image by bit voting between the

corrupted images.

The Clyde Space OBC is the trusted control node, and test results show that the Smart Fusion 2 has high levels of radiation tolerance [36]. Thus, we plan to have the OBC operate as a watchdog for the TX2/TX2i. If the OBC detects that the TX2/TX2i has anomalous power draw levels (over $\approx$ 10 Watts), it will send a command (via the OBC) instructing the TX2i to reduce GPU usage until the power level stabilizes.

Chapter 4

Onboard Computer Vision

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

The mathematics of this section are implemented in a computer vision library known as SSRLCV (Small Satellite Research Lab Computer Vision) which is freely available and open

sourced. The implementation of the software library is not rigorously discussed, but results are analyzed in chapter 6.

4.1 Role of Systems Dynamics

It is critical for the satellite to have accurate position and orientation data so that the computer vision pipeline can reconstruct 3D surfaces. Without accurate estimates of location and orientation, costly estimation algorithms must be employed. Such algorithms are not necessary, as the satellite's position and orientation can assume knowledge within bounds. These bounds are inherently tied to systems dynamics and controls, as it is the Attitude Determination and Control System which provides these estimates.

4.1.1 TLEs and Propagators

A Two Line Elements (TLE) is a standard for encoding of orbital elements of an earth orbiting object at a given point in time. With only a TLE, a future state of the orbiting object can be determined within bounds. The algorithms which calculate these future states are known as simplified perturbation models, with the most commonly used being the Simplified General Perturbation 4 (SGP4) model. Though SGP4 is considered a model, it is also an algorithm. Though there are certainly more accurate models to use, the SGP4 algorithm has remained the most popular ever since the release of the FORTRAN source code in 1988. The commerically systems used by SPOC and MOCI utilize the SGP4 propogation algorithm.

4.1.2 The Kalman Filter

The Kalman filter is ubiquitous in robotics and motion planning and is fantastic for state estimation. A Kalman filter is a linear quadratic estimator (LQE) which uses a series of noisy measurements over time to update a predicted future state based on an assumed

initial model. Here the initial model is the SGP4 propagator and updates from observations of the satellite's state come from the various sensors interfaced to the ADCS. The algorithm even famously incorporated into the Apollo flight computer.

4.2 Camera Systems

An accurate characterization of the satellite's camera system is necessary for an accurate reconstruction of the observed geometry from cube satellites, as slight variations in camera parameters can cause massive changes in final 3D models [37] [38]. To avoid costly camera parameter estimation algorithms, the satellite's camera is pre-calibrated before launch. Additionally, camera parameter estimation is only necessary within certain boundaries, as the camera's systems are characterized on the ground before launch. Certain variables, such as focal length, may change slightly during operations due to temperature gradients.

4.2.1 The Pinhole Camera Model

The pinhole camera model is the simplest camera model and one that is most common in computer vision applications [38]. The MOCI satellite assumes a pinhole camera model within all of its software. The pinhole camera can be thought of as the interaction of light rays with an image plane as they converge to a single point behind the image place, known as the focal point.

The most common way to represent the pinhole camera is with projection matrices. The matrices can be broadly classified to encode intrinsic and extrinsic camera parameters, with the intrinsic parameters encoding optical information and the extrinsic parameters encoding world coordinate information. The intrinsic matrix is parameterized with a focal length in the x direction f_x, a focal length in the y direction f_y, and a principle point (x_0, y_0); this can be seen in equation (4.1).

$$K = \begin{bmatrix} f_x & 0 & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

Then, we can consider the extrinsic camera matrix as one which encodes rotation and translation information; for this we can use a rotation and translation as seen in equation (4.2).

$$[R|t] = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \end{bmatrix} \tag{4.2}$$

The resulting camera projection matrix in equation (4.3) can be used to calculate a given point in $\mathbb{R}^3$'s position in $\mathbb{R}^2$ on the camera plane.

$$\begin{aligned} P &= K \times [R|t] \\ P &= \begin{bmatrix} f_x & 0 & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \end{bmatrix} \end{aligned} \tag{4.3}$$

The final calculation for the 2D image projection, m, from a 3D world coordinate, P, where the final (x, y) of the projected point in the image is given by (u, v), is as follows in equation (4.4):

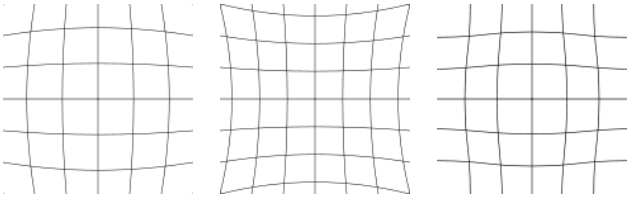


Figure 4.1: Some potential effects of radial distortion, from left to right: Barrel Distortion, Pin Cushion Distortion, and Mustache Distortion

$$m = K[R|t]P$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4.4}$$

4.2.2 Camera Distortions

We cannot assume that the image plane is undistorted. Therefore, assuming some accurate knowledge of the camera's intrinsic parameters, we must attempt to undistort our image. For us, this means we apply transformations to our distorted matched points in order to get the undistorted locations. There are several types of distortion, but we only account for radial distortion, which is often sufficient for most practical purposes.

To undistort our images, we need a formula which maps the distorted feature point coordinates to the undistorted coordinates. The formula we use is based on a simple Taylor series expansion with either one or two distortion coefficients, given by:

$$
\begin{aligned}
x_u &= x_d + (x_d - c_x)(K_1 r^2 + K_2 r^4) \\
y_u &= y_d + (y_d - c_y)(K_1 r^2 + K_2 r^4)
\end{aligned}
\tag{4.5}
$$

Equation (4.5) is defined where $r = \sqrt{x^2 + y^2} = \sqrt{(x_d - c_x)^2 + (y_d - c_y)^2}$ is the distance of the point from the distortion center (c_x, c_y) is the radial distortion center (sometimes different from the image center). K_1 and K_2 are the first and second order radial distortion coefficients, respectively. Ideally we try to find the distortion coefficients which minimize these problems via mathematical optimization. During the optimization, we define a cost function (to be minimized) as the reprojection error, which can be done in one of two ways:

1. Take a known simple scene or object, such as a grid, with precisely known 3D coordinates. This object is often called the calibration object. The reprojection error is the difference between our 2D images of the object versus the actual 3D coordinates projected onto these image planes. This is explained in detail in section on bundle adjustment.

2. The second method, sometimes called the total calibration or self-calibration method, does not require a calibration object. Instead it uses epipolar geometry and defines the reprojection error as the distance of a predicted 2D matched point to the epipolar line. For two views, the epipolar line is defined as the line in 3D space between the matched point on images, given its coordinates in the first view. For every point in view 1, we can compute the epipolar line in view 2 using a special matrix called the fundamental matrix. A diagram of this can be seen in 4.4.

By defining an optimization function to be the sum of squared reprojection error, the distortion coefficients can be estimated using the Levenberg-Marquardt algorithm which is discussed in more detail in a later section.

4.3 Feature Detection, Extraction, and Matching

Two widely used concepts to begin to determine 3D information, which take in arbitrary input images and attempt to identify common features between the images, are feature extraction and feature matching [39][40]. While this is intuitive, identifying features and determining a consistent method for matching two features is computationally expensive. This is why we choose to parallelize computations on a GPU (Graphics Processing Unit) with CUDA (Compute Unified Device Architecture). In addition, knowing the camera's geometry and optical properties are vital. Solutions such as bundle adjustment [41] can optimize the knowledge of the full system; however, in regards to the application of satellite imagery, many assumptions can be made to improve the capabilities of imaging systems constrained in an orbital environment.

Effective and efficient feature matching is key to advancing on-orbit imaging capabilities and terrestrial data gathering techniques. The ultimate goal for these imaging systems is the ability to register features and decide how to decipher a solution for the objects detected [42][43]. The methods described here use a standard Scale-Invariant Feature Transform (SIFT) algorithm from Lowe's original implementation [44][45][46][47].

We can assume that we are given two matched points, (x_0, y_0) and (x_1, y_1), for images I_0 and I_1 respectively. We also assume that the images I_0 and I_1 have n by m pixels. Again, the locations of the matched points will be given to us at the subpixel level in $\mathbb{R}^2$ with the origin in the top left by convention.

4.3.1 SIFT - the Scale Invariant Feature Transform

The Scale Invariant Feature Transform (SIFT) algorithm can be broken up into many component stages. First, the SIFT algorithm attempts to identify extrema in scale space. These points are considered candidate points for feature description and are what make the algo-

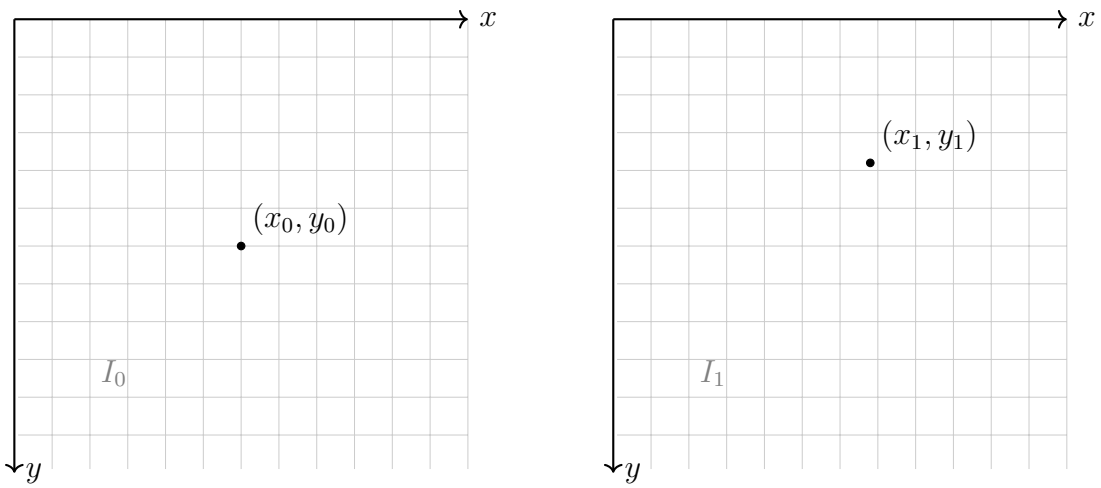


Figure 4.2: Matched points are on separate pixel grids in $\mathbb{R}^2$, but these pixel grids are not necessarily discrete. Subpixel feature detection methods are often employed to improve the accuracy of feature matching.

rithm scale invariant. Next, the algorithm achieves rotation invariance by assigning orientations to particular points from the scale space generation. Finally, the algorithm generates a feature descriptor in the form of a vector of 128 orientations, a histogram of oriented gradients. When this is performed across multiple images, nearly identical histograms are considered nearly identical points.

4.3.2 Dense SIFT

The SIFT feature descriptor is the current standard, with the exception that our method needs no scale value within the scale space. As described by Lowe [45], the descriptor is calculated in the form of a histogram of oriented gradients [45]. The standard SIFT algorithm uses feature detection before features are extracted. However, with dense SIFT variants, it is common to perform a per pixel feature extraction [47][46].

First, we proceed to compute a feature for each pixel in the image $I(x, y)$ using a 16x16

grid centered at the pixel in question. Since this grid has data values at each corner of every cell, we use this grid to make a 16x16 matrix such that each value of the matrix consists of the average of a cell's four corners of the grid.

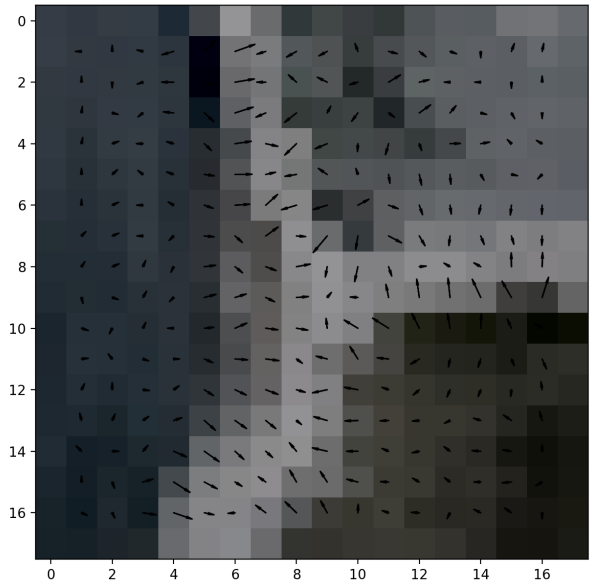


Figure 4.3: Example image gradients of the tip of Mount Everest calculated in a 16x16 grid

Next, we use this matrix to construct 16 4x4 sub-matrices. At each pixel in a sub-matrix, we calculate the orientation and magnitude of each pixel using the following equations from Lowe:

$$\begin{aligned} m(x,y) = &\big((I(x+1,y) - I(x-1,y))^2 \\ &+ (I(x,y+1) + I(x,y-1))^2\big)^{0.5} \end{aligned} \tag{4.6}$$

$$\theta(x,y) = tan^{-1}\left(\frac{I(x,y+1) - I(x,y-1)}{I(x+1,y) - I(x-1,y)}\right) \tag{4.7}$$

Then for each 4x4 sub-matrix, we create an 8-element vector, $v_{i,j}$, such that the value at each index, b, is the sum of each pixel's magnitude where the orientation, θ, is as follows:

$\frac{b\pi}{4} \leq \theta < \frac{(b+1)\pi}{4}$. Once this is computed, we store each vector in the final feature descriptor.

$$f(x,y) = \begin{bmatrix} v_{0,0} & v_{0,1} & v_{0,2} & v_{0,3} \\ v_{1,0} & v_{1,1} & v_{1,2} & v_{1,3} \\ v_{2,0} & v_{2,1} & v_{2,2} & v_{2,3} \\ v_{3,0} & v_{3,1} & v_{3,2} & v_{3,3} \end{bmatrix} \tag{4.8}$$

After obtaining the dense SIFT features for each pixel over a set of images, we proceed to match similar features over two images. These matched features represent similar geometric locations taken from different perspectives. To match these features, first we employ coarse pixel match location based constraints provided by epipolar geometry and outlier rejection.

4.3.3 The Matching Problem

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

$$\min_{(i',j')} ||f_1(i, j) - f_2(i', j')|| \tag{4.9}$$

The computational complexity of this process is $O(n^2m^2)$ because all features in image

I_1 must be searched against all features in image I_2. This is computationally expensive, even in a GPU accelerated framework. In the following sections we further expand on epipolar geometry and how it can be exploited in order to narrow this search space significantly.

4.3.4 Epipolar Geometry and the Fundamental Matrix

Epipolar geometry is the projective geometry between two views. When two cameras view a 3D scene from different positions, there are some geometric relations between the 3D points and their projections onto the images that define useful constraints between image points. Figure 4.4 depicts two cameras looking at some point X in world space.

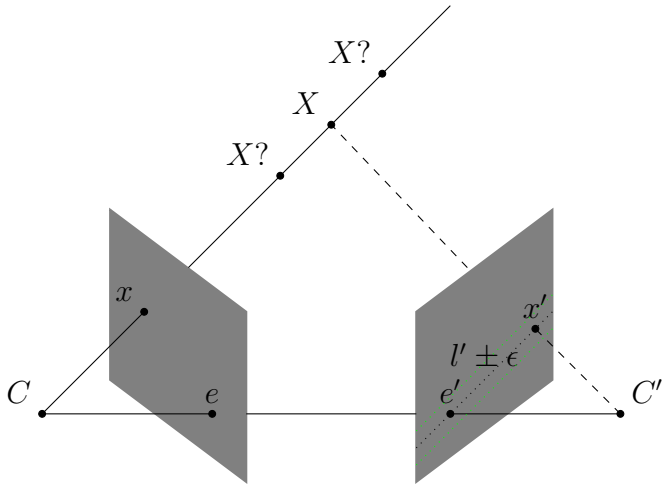


Figure 4.4: Epipolar geometry with epipolar line l' and constrained search path $l' \pm \epsilon$

Let C and C' be camera centers for our two views, and let x be some point on the first image. Figure 4.4 shows that x alone is not enough to determine a unique location of the 3D world point X as any points on the line segment from x to X in the picture would project to the same image location on image 1; however, each one maps to a different location in image 2. If we were to project this entire line to image 2, then we would end up with a line segment which represents the valid potential locations for the corresponding matched point x, depending on the exact location of X in space. This line, referred to as the epipolar line,

can be determined for every feature point. The fundamental matrix, denoted F, maps every point in image 1 to a corresponding epipolar line in image 2, and vice versa.

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

$$x'^T F x = 0 \tag{4.10}$$

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

4.3.5 Restricted Search Region from Epipolar Geometry

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

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

$l'_{i,j} = F\dot{f}_1(i,j)$. Define the set:

$$H_{i,j} \subset I_2 = \{h \in I_2 \mid d(h, l'_{i,j}) \leq \epsilon\} \tag{4.11}$$

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

$$\min_{(i',j') \in H_{i,j}} ||f_1(i,j) - f_2(i',j')|| \tag{4.12}$$

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

4.3.6 Bicubic Splines of Matching Windows

To enhance matching in the case of dense SIFT extraction, bicubic splines can be used within the matching windows. Cubic splines fit a piecewise polynomial function over some discrete set of input values [48]. This allows a surface to be generated over a potential feature match to identify a more accurate minimum than would be allowed by checking discrete values. In addition, it allows for the geometric match where a feature is projected within 3D space.

Once the pixel match has been found, we proceed to find the subpixel match. We will call the matched features $f_1(i_1, j_1), f_2(i_2, j_2)$. After this, we construct two odd $n \times n$ dimensional arrays M_1, M_2 for each image (odd because arrays are built around around a certain feature, associated with a certain pixel) and populate these arrays with the following information:

$$M_1(i,j) = ||f_1(i_1 + i - \frac{n-1}{2}, j_1 + j - \frac{n-1}{2}) - f_2(i_2, j_2)|| \tag{4.13}$$

$$M_2(i,j) = ||f_1(i_1, j_1) - f_2(i_2 + i - \frac{n-1}{2}, j_2 + j - \frac{n-1}{2})|| \tag{4.14}$$

For the purposes of this paper, we chose $n = 9$ (because it is large enough to have 2 immediate neighbors in all cardinal directions) as the size of this matrix. From here, we construct the data needed for the spline to overlay. For the interpolation to be successful, we need to know the edge values and derivatives of our function. To gather this data we create a 7x7 matrix, F, such that at each index in F, the function f represents the value and central derivative information of the desired function to be interpolated.

$$F(i,j) = \begin{bmatrix} f(0,0) & f(0,1) & f_y(0,0) & f_y(0,1) \\ f(1,0) & f(1,1) & f_y(1,0) & f_y(1,1) \\ f_x(0,0) & f_x(0,1) & f_{xy}(0,0) & f_{xy}(0,1) \\ f_x(1,0) & f_x(1,1) & f_{xy}(1,0) & f_{xy}(1,1) \end{bmatrix} \tag{4.15}$$

Each entry i, j, is determined by the following finite difference equations:

$$\begin{aligned} f(x,y) &= M(i+x+1, j+y+1) \\ f_x(x,y) &= M(i+x, j+y+1) - M(i+x+2, j+y+1)) \\ f_y(x,y) &= M(i+x+1, j+y) - M(i+x+1, j+y+2)) \\ f_{xy}(x,y) &= M(i+x, j+y) - M(i+x+2, j+y+2)) \end{aligned} \tag{4.16}$$

Now we create the bicubic spline, $S_{i,j}$ based on each entry in $F(i,j)$ and obtain the final bicubic spline, $B(x,y)$ where $-3 \leq x < 3$ and $-3 \leq y < 3$. This gives the final equation:

$$B(x,y) = S_{i,j}\big(x - (3-i), y - (3-j)\big) \tag{4.17}$$

From here we find the absolute minimum of that spline at some location x_0, y_0, and determine the subpixel location of feature i, j at pixel $(i + x_0, j + y_0)$. In the implementation we check to see if the subpixel matches lie on the boundary. If they do, we rely on the course resolution feature location. This occurs because an optimal match from I_1 to I_2 may not be the optimal match from I_2 to I_1.

4.4 3D Reconstruction

3D reconstruction consists of taking sets of matched points from the SIFT algorithm (other feature detection and matching algorithms could also work), moving them to their real locations in 3D space, and then deducing 3D structure from camera parameters match locations. In addition to the previous methods, MOCI seeks to test several of the following methods in orbit.

4.4.1 Stereo Disparity

Stereo disparity is perhaps the simplest form of 3D reconstruction possible given a set of corresponding match points in images. It is performed with rectified images that have match locations and by calculating the euclidean distance between both of the match points (assuming they are on the same image plane after rectification). This has the desired effect of producing a larger "disparity" between points closer in 3D space and a smaller "disparity" between points that are farther away in 3D space. All that is needed to create a realistic model from this method are the camera parameters of the system so that a scaling factor can be applied to the euclidean distance.

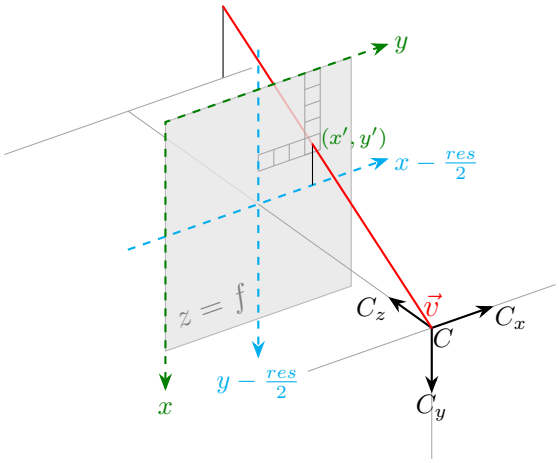


Figure 4.5: A visualization of line generation and reprojection from the image space to the world space

4.4.2 Triangulation for Reprojection

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

Generating Equations of Lines

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

The goal here is to generate a parametric equation of a line given camera position and

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

$$x' = dpix\left(x - \frac{xres}{2}\right) \qquad y' = dpix\left(y - \frac{yres}{2}\right) \tag{4.18}$$

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

$$\theta_z = \cos^{-1} \frac{\left(\begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} r_x & r_y & r_z \end{bmatrix}\right)}{\sqrt{\begin{bmatrix} r_x & r_y & r_z \end{bmatrix} \cdot \begin{bmatrix} r_x & r_y & r_z \end{bmatrix}}} \tag{4.19}$$

It is important to note that the sofware treats all planes in an identical way, rotating in several axis. Now, given a rotation in each plane $(\theta_x, \theta_y, \theta_z)$ (lets call this rotation matrix R_θ), we calculate the homogeneous coordinate $(r_x, r_y, r_z, 1)$ in $\mathbb{R}^4$ using linear transformations. The values (T_x, T_y, T_z) represent a translation in $\mathbb{R}^3$ and use camera position coordinates (C_x, C_y, C_z), the camera unit vectors representing orientation (u_x, u_y, u_z), and focal length f. Let the rotation matrix R in equation (4.20) represent a 3x3 rotation matrix generated from multiplying component rotation matrices of each axis.

$$R_\theta \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x_r \\ y_r \\ z_r \\ 1 \end{bmatrix} \tag{4.20}$$

$$\begin{bmatrix} C_x - (x_r + f \cdot u_x) \\ C_y - (y_r + f \cdot u_y) \\ C_z - (z_r + f \cdot u_z) \\ 1 \end{bmatrix} = \begin{bmatrix} T_x \\ T_y \\ T_z \\ 1 \end{bmatrix} \tag{4.21}$$

$$\begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r \\ y_r \\ z_r \\ 1 \end{bmatrix} = \begin{bmatrix} x'_r \\ y'_r \\ z'_r \\ 1 \end{bmatrix} \tag{4.22}$$

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

$$
\begin{aligned}
L_0 &= \begin{bmatrix} x_0 - C_{x0} \\ y_0 - C_{y0} \\ z_0 - C_{z0} \end{bmatrix} \begin{bmatrix} t_0 \\ t_0 \\ t_0 \end{bmatrix} + \begin{bmatrix} C_{x0} \\ C_{y0} \\ C_{z0} \end{bmatrix} \\
L_1 &= \begin{bmatrix} x_1 - C_{x1} \\ y_1 - C_{y1} \\ z_1 - C_{z1} \end{bmatrix} \begin{bmatrix} t_1 \\ t_1 \\ t_1 \end{bmatrix} + \begin{bmatrix} C_{x1} \\ C_{y1} \\ C_{z1} \end{bmatrix}
\end{aligned}
\tag{4.23}
$$

The host functions (this is CUDA terminology for a CPU located function) are simple and will not be addressed here, refer to standard CUDA memory management.

Minimum Distance Between Skew Lines

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

$$
L_0 = v_0 t_0 + C_0 \qquad L_1 = v_1 t_1 + C_1 \tag{4.24}
$$

To make sure that the lines are not parallel, which is unlikely, we must verify that their cross product is not zero. If $v_0 \times v_1 = 0$, then we have a degenerate case with infinitely many solutions. As long as we know this is not the case we can proceed. We know that the cross product of the two vectors $c = v_0 \times v_1$ is perpendicular to the lines L_0 and L_1. We know that the plane P, formed by the translation of $L1$ along c, contains C_1. We also know that

the point C_1 is perpendicular to the vector $n_0 = v_1 \times (v_0 \times v_1)$. Thus, the intersection of L_0 with P is also the point, s_0, that is nearest to L_1, given by the equation:

$$s_0 = C_0 + \frac{(C_1 - C_0) \cdot n_0}{v_0 \cdot n_0} \cdot v_0 \tag{4.25}$$

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

$$s_1 = C_1 + \frac{(C_0 - C_1) \cdot n_1}{v_1 \cdot n_1} \cdot v_1 \tag{4.26}$$

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

$$m = \begin{bmatrix} (s_0[x] + s_1[x])/2 \\ (s_0[y] + s_1[y])/2 \\ (s_0[z] + s_1[z])/2 \end{bmatrix} \tag{4.27}$$

4.4.3 N-View Reprojection

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

In figure 4.6, assume that point C is the correct real world point and has no orientation.

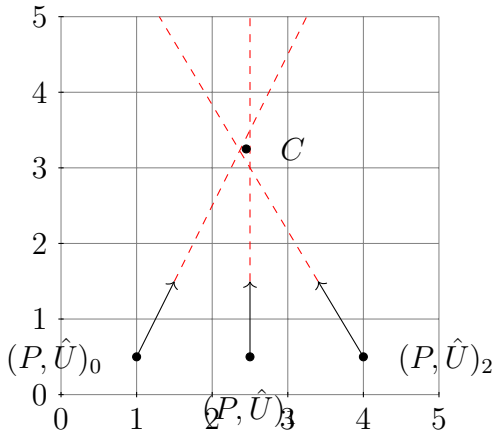


Figure 4.6: Nview triangulation / reprojection example within a single plane

The goal is to make a best guess at the value of C given our imperfect information.

3 by 3 Inversion Method

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

$$(m \times n) \cdot (m \times n) = ||m||^2 ||n||^2 - (m \cdot n)^2 \tag{4.28}$$

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

$$D_n = \frac{||(C - P_n) \times \hat{U}_n||}{||\hat{U}_n||} \tag{4.29}$$

We will want to use the square of the distance (as is common in many optimization

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

$$
\begin{aligned}
D_n &= \frac{||(C - P_n) \times \hat{U}_n||}{||\hat{U}_n||} \\
D_n^2 &= \Big(\frac{||(C - P_n) \times \hat{U}_n||}{||\hat{U}_n||}\Big)^2 \\
D_n^2 &= \frac{||(C - P_n) \times \hat{U}_n||^2}{||\hat{U}_n||^2} \\
D_n^2 &= \frac{||C - P_n||^2||\hat{U}_n||^2 - ||(C - P_n) \cdot \hat{U}_n||^2}{||\hat{U}_n||^2} \\
D_n^2 &= ||C - P_n||^2 - \frac{||(C - P_n) \cdot \hat{U}_n||^2}{||\hat{U}_n||^2} \\
\frac{dD_n^2}{dC} &= 2(C - P_n) - 2\hat{U}_n \frac{(C - P_n) \cdot \hat{U}_n}{||\hat{U}_n||^2}
\end{aligned}
\tag{4.30}
$$

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

$$
\begin{aligned}
0 &= \sum_{n=0}^{m} C - P_n - \hat{U}_n \frac{(C - P_n) \cdot \hat{U}_n}{||\hat{U}_n||^2} \\
0 &= \sum_{n=0}^{m} C - P_n - \frac{\hat{U}_n(C \cdot \hat{U}_n)}{||\hat{U}_n||^2} + \frac{\hat{U}_n(P_n \cdot \hat{U}_n)}{||\hat{U}_n||^2} \\
0 &= \sum_{n=0}^{m} C - P_n - \frac{\hat{U}_n \hat{U}_n^T C}{||\hat{U}_n||^2} + \frac{\hat{U}_n \hat{U}_n^T P_n}{||\hat{U}_n||^2} \\
0 &= \sum_{n=0}^{m} \Big(I - \frac{\hat{U}_n \hat{U}_n^T}{||\hat{U}_n||^2}\Big)C - \Big(P_n - \frac{\hat{U}_n \hat{U}_n^T P_n}{||\hat{U}_n||^2}\Big)
\end{aligned}
\tag{4.31}
$$

This is of the form $Ax = b$ because we now have $0 = Ax - b$. Thus, we can remove the

summations and get a system that results in taking an inverse of a 3 by 3 matrix.

Notice the possible expansion:

$$
\begin{aligned}
0 &= \sum_{n=0}^{m} \Big(I - \frac{\hat{U}_n \hat{U}_n^T}{||\hat{U}_n||^2}\Big)C - \Big(P_n - \frac{\hat{U}_n \hat{U}_n^T P_n}{||\hat{U}_n||^2}\Big) \\
0 &= \left(\Big(I - \frac{\hat{U}_0 \hat{U}_0^T}{||\hat{U}_0||^2}\Big) + \Big(I - \frac{\hat{U}_1 \hat{U}_1^T}{||\hat{U}_1||^2}\Big) + \Big(I - \frac{\hat{U}_2 \hat{U}_2^T}{||\hat{U}_2||^2}\Big) + ...\right)C \\
&\quad - \left(\Big(P_0 - \frac{\hat{U}_0 \hat{U}_0^T P_0}{||\hat{U}_0||^2}\Big) + \Big(P_1 - \frac{\hat{U}_1 \hat{U}_1^T P_1}{||\hat{U}_1||^2}\Big) + ...\right) \\
0 &= (A)C - (b) \\
AC &= b \\
C &= A^{-1}b
\end{aligned}
\tag{4.32}
$$

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

$$A = \sum_{n=0}^{m} \Big(I - \frac{\hat{U}_n \hat{U}_n^T}{||\hat{U}_n||^2}\Big) \tag{4.33}$$

$$b = \sum_{n=0}^{m} \Big(P_n - \frac{\hat{U}_n \hat{U}_n^T P_n}{||\hat{U}_n||^2}\Big) \tag{4.34}$$

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

4.4.4 Point Normal Estimation

The estimation of point normals is vital for accurate 3D meshing of point clouds, as normals give additional information on curvature and enable smoother non-linear mesh interpolation. Thus, a critical step in the computer vision pipeline of the MOCI satellite is to estimate point normals. Though point cloud meshing is not currently implemented in the MOCI computer vision software, it should be considered reasonable future work.

Aside from the point cloud coordinates, the only information needed in the point normal calculation is the camera position (C_x, C_y, C_z) that generated each point. The final result will be the same list of input point coordinates along with the computed normal vector of each point. The problem of determining the normal to a point on the surface is approximated by estimating the tangent plane of the point, and then taking the normal vector to the plane. However, there are two valid normals for the estimated tangent plane but only one is suitable for reconstruction. The correct orientation of the normal vector cannot be directly inferred, so an additional subroutine is needed to choose the correct normal vector.

Let a given point cloud be referenced as $PC = p_1, p_2, p_3, ..., p_n$ where a given point is $p_i = (x_i, y_i, z_i)$ and for each point $p_i \in PC$ we seek to find the correct normal vector $n_i = (n_x, n_y, n_z)$. Also note that each point has an associated camera of the form $C_i = (C_x, C_y, C_z)$.

First, the k nearest neighbors of point p_i must be retrieved; let these points be defined as $Q_{i,k} = q_1, q_2, q_3, ...q_k$ where any neighbor $q_i \in PC$. Then a centroid of the subset $Q_{i,k}$ is calculated with equation (4.35):

$$m = \frac{1}{k} \sum_{q \in Q} q \tag{4.35}$$

Next we seek to produce an approximation of a plane by calculating two vectors, v_1 and v_2, from the given subset of k points. First, let A be a k x 3 matrix built from the centroid being subtracted from each point in the nearest neighbor subset. To find the desired vectors

we must perform a singular value decomposition (SVD), seen in equation (4.36), and notice that the covariance matrix A^TA can be diagonalized so that the eigenvectors of the covariance matrix are the columns of vector V (or the rows of vector V^T).

$$\begin{aligned} A &= U\Sigma V^T \\ A^TA &= (U\Sigma^TU^T)(U\Sigma V^T) = V(\Sigma^T\Sigma)V^T \end{aligned} \tag{4.36}$$

In general, the best r-rank approximation of an (n x n) matrix, $r < n$, is found by diagonalizing the matrix as above, only keeping the first r columns of V (similarly only the first r rows of V^T), and only the first r diagonal elements of $\Sigma^T\Sigma$ (or only first r rows and columns), assuming that the values on the diagonal of Σ were in descending order. More precisely, for randomly ordered diagonal elements $(\sigma_i)^2 \in \Sigma^T\Sigma$ we keep only the maximum r many of them, along with their corresponding eigenvectors in matrix V. The reason for choosing the maximum valued eigenvalues is that it minimizes the amount of information lost in moving to a lower rank approximation matrix. Therefore, to produce the best approximation of a plane in $\mathbb{R}^3$ we would take the two eigenvectors, v_1 and v_2, of the covariance matrix (which are exactly the columns of V) with the highest corresponding eigenvalues. Those two eigenvectors span the plane we are looking for. Thus, the normal vector n_i is simply the cross product of these eigenvectors: $n_i = v_1 \times v_2$.

The reason for introducing the SVD is because in computing the covariance matrix A^TA we may lose some level of precision in the calculation. By simply factoring matrix A into its singular value decomposition and taking the cross product of the first two rows of V^T, we can avoid this problem.

As previously mentioned, there are two viable normals that could be computed with this method, but only one normal is the desired normal. To solve this issue we could simply compute the vector from the camera position C to point p_i such that $(C - p_i) \cdot n_i < 0$ holds.

If this does not hold then the vector can be flipped by changing the signs of its components. However, because there are likely to be many camera locations, say $C = C_{i,1}, C_{i,2}, C_{i,3}, ..., C_{i,N}$ for all N cameras of a given point p_i, a point's normal can be considered ambiguous if the following is true:

1. There exists a $\bar{C_1} \in C$ such that $(\bar{C_1} - p_i) \cdot n_i < 0$
2. There exists a $\bar{C_1} \in C$ such that $(\bar{C_1} - p_i) \cdot n_i > 0$

Such points cannot easily be oriented and thus additional computation is needed; fortunately, in most cases there are very few such normals. When these normals are discovered they are added to a queue of unfinished normals while the rest are placed in a list of correct normals. The algorithm iterates through the queue of ambiguous normals and tries to determine the orientation by looking at the neighboring points of p_i. If the neighboring points of p_i have already finished normals, then n_i is oriented such that it is consistent with the neighboring normals m_i by setting $n_i \cdot m_i > 0$. If the neighboring points do not have already finished normals, then we move p_i to the back of the queue, and continue until all normals are finalized.

4.5 Bundle Adjustment

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

4.5.1 Point Estimation Error

When points are estimated with the 2-view case, the minimal distance between skew lines is calculated by taking the Euclidean distance between the points. For that case the total error of the point cloud is simple to calculate and is just a summation of every point's individual error, or the average of that summation. The N-view case is similar; it can either be calculated with the Euclidean distance between projected estimated points onto the camera plane and their original match point (see 4.4) or by calculating the average. These methods are an analog for the commonly used reprojection error, which is not directly calculated in this case. Thus there are four possible error functions, though only the last two are used practically:

1. **Linear Error:** Only calculated in the 2-view case, linear error is the shortest distance between matched lines.

2. **Average Linear Error:** Calculated in the N-view case, average linear error is the average shortest distance from the estimated point to its corresponding line.

3. **Squared Linear Error:** This is the practical 2-view error measurement, used so that the error function can be used in convex optimization.

4. **Squared Average Linear Error:** This is the practical N-view error measurement, used so that the error function can be used in convex optimization.

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

4.5.2 Noise Removal

The distribution of error (calculator of error is above) in 3D reconstructions is typically considered to be Gaussian. Thus, statistical filtering methods can be used to remove outlier

error points. The methods used here start by calculating a sample variance, σ^2 , from a random set of points. Let a given sample point be s_i and the average of the total error be $\bar{s}$, then the sample variance can be calculated with equation (4.37)

$$\sigma^2 = \frac{1}{n}\sum_{i=0}^{n}\left(s_i - \bar{s}\right)^2 \tag{4.37}$$

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

4.5.3 Formulation as Gradient Descent

It is possible to formulate the bundle adjustment as an optimization to minimize one of the error functions listed above. Though many algorithms exist to solve this, Newtonian gradient descent is implemented here because of its simplicity. The most commonly used algorithm in bundle adjustment is the Levenberg-Marquardt (LM) algorithm [41], which differs from the standard Newtonian approach with its use of the second derivative, in this context the Hessian matrix, and contains a dampening parameter to slow the descent around a local minimum. It is certainly possible to implement the LM algorithm within the context of MOCI's bundle adjustment, this should be considered future work for SSRL lab members or graduate students. However, given that camera parameters are expected to be relatively accurate to start, the use of a second order Newtonian method with simple dampening might be considered sufficient for this application. If camera parameters are not known, but are instead estimated by some other computer vision procedure, then a more robust optimization algorithm is likely needed. Additionally, noise removal should not occur during the gradient descent as this can cause false minima and rapidly deteriorate into gratuitous point removal

until no points are left.

First, consider the standard Newtonian gradient descent and the parameters we are using for this descent in equation (4.38). The position and orientation of the satellite are much more uncertain than the focal length and field of view of the imager. Thus, intrinsic camera parameters are considered, at least for now, to be close enough to their optimal configurations that they should not be modified. Instead, extrinsic camera parameters (position and orientation) are to be modified when searching for a minimum error. Let the chosen error function be $F(C_n)$ where C_n represents a vector of all input camera parameters at iteration n.

$$C_{n+1} = C_n - \alpha_n H^{\dagger} F(C_n) \nabla F(C_n) \tag{4.38}$$

∇ is the gradient for the error function F with respect to all camera parameters in the camera vector C. ∇ is a vector of partial derivatives that are calculated via a finite central difference. A given element of ∇, say $\frac{\partial F}{\partial C_n[i]}$, where $C_n[i]$ is the ith element in the vector C_n, is calculated with a set size h along a vector e which is all 0's other than a single 1 at the ith index. This has the effect of only stepping a distance of h along the ith element of C_n, thus estimating the partial derivative, as seen in (4.39).

$$\begin{aligned}
\frac{\partial F}{\partial C_n[i]} &= \frac{F(C_n + he) - F(C_n - he)}{2h} \\
\frac{\partial^2 F}{\partial C_n[i]^2} &= \frac{-F(C_n + 2he) + 16F(C_n + he) - 30F(C_n) + 16F(C_n - he) - F(C_n - 2he)}{12h^2} \\
\frac{\partial^2 F}{\partial C_n[i] \partial C_n[j]} &= \frac{\begin{array}{l} F(C_n + h_i e_i + h_j e_i) - F(C_n + h_i e_i - h_j e_i) \\ \quad - F(C_n - h_i e_i + h_j e_i) + F(C_n - h_i e_i - h_j e_i) \end{array}}{4 h_i h_j}
\end{aligned} \tag{4.39}$$

Additionally, the Hessian is calculated via a finite central difference seen above in equation

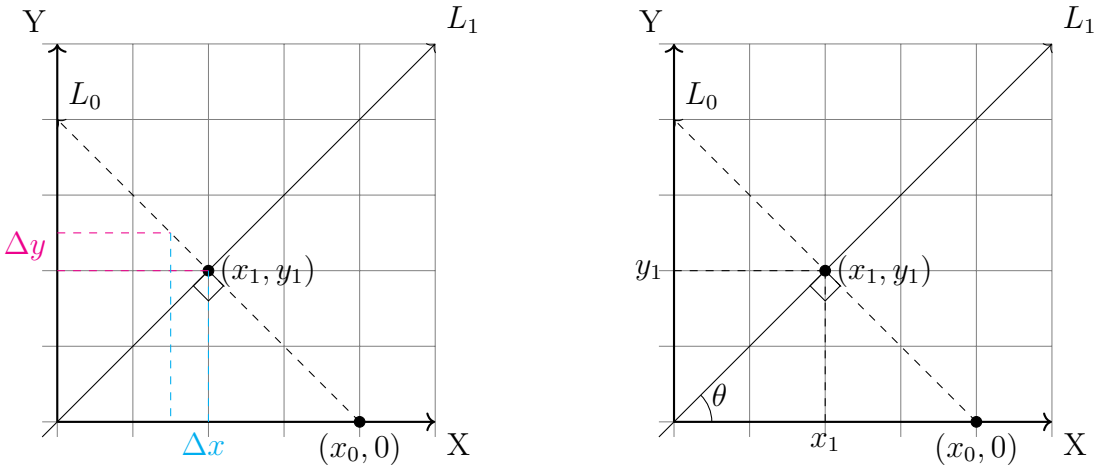
(4.39), which is defined similarly to the gradient only with more than one change to consider (a change in different parameters $C_n[i]$ and $C_n[j]$). The Moore–Penrose pseudoinverse of the Hessian $H^\dagger$ is then calculated to produce a stepsize adjustment for each step of the gradient descent. The pseudoinverse is calculated, rather than the direct inverse, because the Hessian may not always be invertable. The process for caclulating the inverse of the Hessian involves caclulating a singluar value decomposition (SVD), where $A = U\Sigma V^T$ for a given matrix A. To calculate the pseudoinverse $A^\dagger$, matrices U and V^T are transposed and the inverse Σ^{-1} is obtained by taking the reciprocal of each non-zero element within Σ. The equations cab be seen in (4.40).

$$\begin{aligned} A^\dagger &= (A^T A)^{-1} A^T \approx A^{-1} \\ A &= U\Sigma V^T \\ A^\dagger &= V\Sigma^{-1} U^T \end{aligned} \tag{4.40}$$

At each iteration of the descent, the dampening variable α, seen in the initial equation (4.38) is decreased by a ration of the previous error e_{n-1} and the currently computed error e_n such that $\alpha_n = \frac{e_{n-1}}{e_n}$. Here the assumption is that $e_{n-1} > e_n$; when this is no longer the case, the algorithm has found a local minima and exits.

Most bundle adjustment algorithms analyze how camera parameters are defined to help define derivatives. The camera model I have defined has special properties that are not derived elsewhere in computer vision literature, so a derivation of camera parameter derivatives was beneficial. Because I assume that intrinsic camera parameters are well calibrated, I have not derived their derivatives. Instead, I derive only the extrinsic translation and orientation derivatives by first deriving the error functions for each variable with respect to a single point.

The coordinate systems of the cameras are relative, so convenient axis and points can be



(a) The translation case

(b) The orientation case

Figure 4.7: Graphs used to aid in deriving camera parameter derivatives

chosen. Additionally, consider that only linear error is considered here. This linear error is just the euclidean distance, $d = \sqrt{(x_0 - x_1)^2 + (0 - y_1)^2}$, between point (x_1, y_1) and point $(x_0, 0)$.

The single point, seen in figure 4.7 as $(x_0, 0)$, should be thought of as a point on a line perpendicular to the XY plane in which figure 4.7 is graphed. That line and the line L_1 represent lines which should be used to produce a 3D point. The shortest distance between the line L_1 and the imagined 3D line passing through point $(x_0, 0)$ is along the line L_0 within the XY plane. Note that line L_0 must be perpendicular to line L_1 to represent the shortest distance between L_1 and $(x_0, 0)$.

The translation case is the easiest to consider. First, notice that a translation of line L_1 along the Z axis results in no change in linear error and any translation of line L_1 does not change L_0, thus L_0 is constant in the translation case. Notice that a translation Δx or Δy results in a right triangle where the sides are Δx and Δy. Thus, the resultant change in linear error for translation is just the length of the hypotenuse, $h = \sqrt{\Delta x^2 + \Delta y^2}$, and changes

linearly with translation. Consider that Δx and Δy are not independent and because slope $m = \frac{\Delta y}{\Delta x}$ we can substitute $\Delta y = m\Delta x$ so that we have the translation error as seen in equation (4.41).

$$\begin{aligned} h &= \sqrt{(\Delta x)^2 + (m\Delta x)^2} \\ e &= (\Delta x)^2 + (m\Delta x)^2 \end{aligned} \tag{4.41}$$

The derivatives of the translation case are constant and no adjustment of step size γ is necessary, but that is not a good thing. The resultant function to optimize is an absolute value so its derivative is undefined at its minimum. Such a function is not desirable, the constant step size for γ will result in slow performance and the convergence to the true minimum is not guaranteed. Simply not taking the square root in equation (4.41) allows for variable stepsize, faster convergence to the minimum, and guaranteed differentiability.

The orientation case is slightly more complicated, as the line L_0 changes with a change in θ. Notice that if both lines were in the same plane they would always have a point of intersection. Thus, rotating within that plane would cause no change in linear error. The goal is to have an error equation in terms of a change in θ. We consider the value x_0 to be constant just as before. First, let line L_1 be defined as $y = \tan\theta x$ and line L_0 be defined as $y = \frac{-1}{\tan\theta}(x - x_0)$ and then define the x intersection point x_1 in terms of θ and constant x_0 as seen in equation (4.42).

$$\begin{aligned} \tan(\theta)x_1 &= \frac{1}{\tan(\theta)}(x_1 - x_0) \\ x_1 &= \frac{x_0}{\tan^2(\theta) + 1} \end{aligned} \tag{4.42}$$

Then, we seek to define point y_1 in terms of θ and constant x_0 as seen in equation (4.43)

using substitution from equation (4.42).

$$\begin{aligned} y_1 &= \tan(\theta)x_1 \\ y_1 &= \frac{\tan(\theta)x_0}{\tan^2(\theta)+1} \end{aligned} \tag{4.43}$$

The Euclidean distance, $d = \sqrt{(x_0 - x_1)^2 + (0 - y_1)^2}$, represents error defined in terms of θ and constant x_0; this requires substitution and some algebra seen in equation (4.44).

$$\begin{aligned} d &= \sqrt{(x_0 - x_1)^2 + (y_0 - 0)^2} \\ d &= \sqrt{\left(x_0 - \frac{x_0}{\tan^2(\theta)+1}\right)^2 + \left(0 - \frac{\tan(\theta)x_0}{\tan^2(\theta)+1}\right)^2} \\ d &= \sqrt{x_0^2 + \frac{x_0^2}{(\tan^2(\theta)+1)^2} - \frac{2x_0^2}{\tan^2(\theta)+1} + \frac{\tan^2(\theta)x_0^2}{(\tan^2(\theta)+1)^2}} \\ d &= \sqrt{x_0^2 + \left(1 + \frac{1 - 2(\tan^2(\theta)+1) + \tan^2(\theta)}{(\tan^2(\theta)+1)^2}\right)} \\ d &= |x_0|\sqrt{\frac{1 - 2\tan^2(\theta) - 2 + \tan^2(\theta) + \tan^4(\theta) + 2\tan^2(\theta) + 1}{(\tan^2(\theta)+1)(\tan^2(\theta)+1)}} \\ d &= |x_0|\sqrt{\frac{\tan^2(\theta)}{\tan^2(\theta)+1}} \end{aligned} \tag{4.44}$$

To ensure convex optimization, the distance measurements are squared. Thus, the ideal error equations for the translation case (labeled e_t) and for the rotational case (labeled e_r) are seen in equation (4.45) and visualized in 4.8:

$$\begin{aligned} e_l &= (\Delta x)^2 + (m\Delta x)^2 \\ e_r &= x_0^2 \frac{\tan^2(\theta)}{\tan^2(\theta)+1} \end{aligned} \tag{4.45}$$

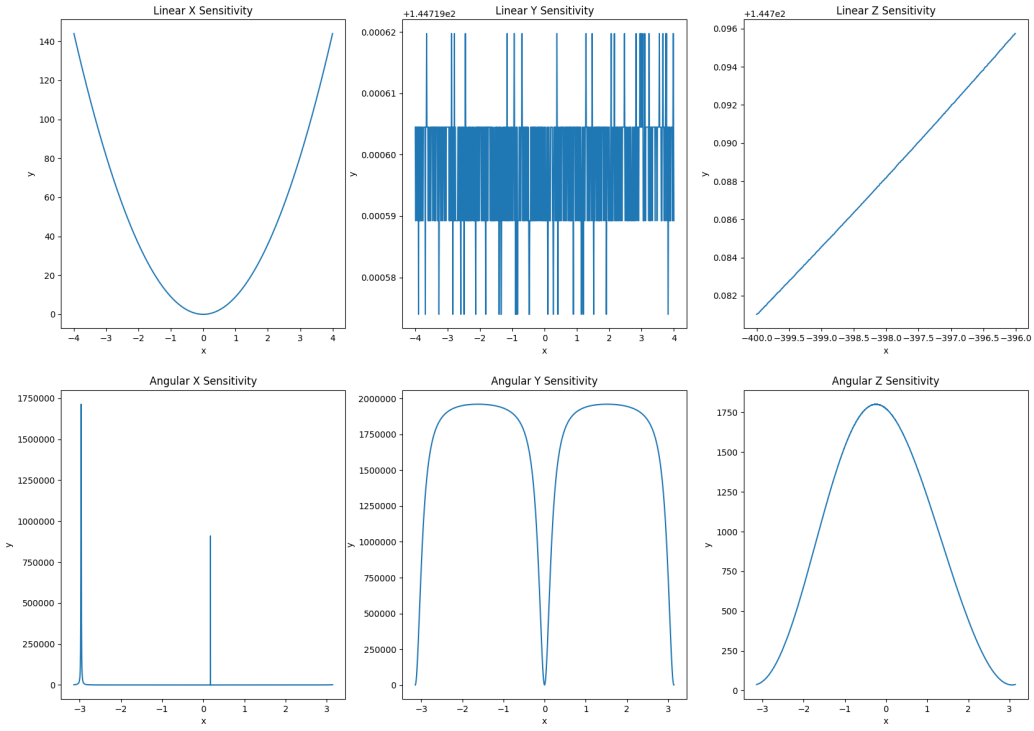


Figure 4.8: The ideal error functions calculated for each individual function

The ideal equations for the error functions are validated in figure 4.8 where each individual error function is calculated for a particular variable. To validate these error functions, the vertices of a cube were projected onto image planes. The camera extrinsic parameters were changed one at a time with small step sizes and the squared linear error was calculated at each step. Here the x linear sensitivity represents a change in the camera's x position (thus Δx is linear sensitivity) and a change in angle represents the camera's angular sensitivity (thus $\Delta\theta$ is angular sensitivity). The functions computed match the derived ideal case, even when computed on more complex datasets.

Chapter 5

Distributed Computation with Satellites

Distributed computation with satellites is, for the most part, limited by the communications constraints of the satellite systems. Though these constraints are a serious concern, there have recently been significant advances which may relieve communications bottlenecks enough so that distributed computation can be demonstrated on small satellites. Laser cross-link and high frequency bands, such as Ka-band and X-band, have seen promising advances which guarantee megabit to gigabit speeds. Additionally, satellite constellations, such as the Starlink, promis the development of wide area internet connections from satellite systems. For these reasons the focus of this research is not the feasibility of distributed communications, but rather the implementation of such communications over a traditional network stack. The management software written for this thesis is known as the SSRL Swarm Network Manager, or just SSRL swarmnet.

5.1 Networking Assumptions

This research assumes that the individual agents (the small satellites) within the swarm network are capable of communication over an internet-like stack, such as the OSI (Open Systems Interconnection) model and the TCP/IP (Transmission Control Protocol / Internet Protocol) model which is also known as the Internet Protocol (IP) suite. Though the swarmnet could support the OSI model, it was designed for use with the Internet Protocol version 4 (IPv4). Additionally, the swarm communications software assumes that all internet layer, link layer, and further lower layers are inherently supported and/or implemented by the operating system and/or satellite hardware. Consequently, ARP (Address Resolution Protocol) and routing are assumed to be features of the network that are abstracted away from individual agents. Furthermore, it is assumed that a network router is capable of managing multicast groups and can send an IP multicast over the internet layer.

The software assumes that all communications external to the satellite are accessible via a POSIX (Portable Operating System Interface) compliant API (Application Programming Interface). Thus, the network is accessible via the ISO C Library and C++ Standard Library calls so that the use of datagram sockets, stream sockets, and raw sockets are all supported. It is also assumed that the system will have ample access to port assignment and is capable of multi threading. Lastly, it is assumed that all agents run the same software packages and have knowledge of the internal state possibilities of every member of the swarm, their the relevant communication ports, and are thus homogenous.

These assumptions are made for three primary reasons:

1. Satellite communications hardware is developing rapidly and is outside of the scope of this research.

2. Future hardware is likely to adopt the common standards of internet communications.

3. The SSRL Swarm Network Manager is a proof of concept library.

5.2 Satellite Swarm Architecture

Though the satellite swarm architecture of the SSRL Swarm Network is currently single purpose, the methods described within this section are generalized for any application. All that is necessary to adapt the SSRL Swarm Network Manager to alternate applications is a modification of the state table, of which all agents have a local copy. After a state table modification, the logic of the swarm net can be used for data and telemetry sharing among the swarm network.

5.2.1 Agent Level Architecture

To join a swarm network an agent begins by joining a predefined multicast group at an IP address within the range `224.0.0.0 - 224.0.0.255`. There can be many multicast groups, but the agent may only join one at a time. It is expected that the programmer has divided the multicast groups by purpose, for the use case described here only one group is used, which is at the default `244.0.0.1`. Each agent has a specific state A_s which is encoded as a byte and can be determined via a lookup table. Additionally, each agent also has an internal member struct which contains a 16 byte string from the agent's name, the agent's state byte, and the agent's 4 byte IPv4 address. These member structs are multicast to every member of the swarm network at a predefined frequency, though the programmer may change this frequency when initiating the agent. The primary purpose of the multicast group is to distribute telemetry, device information, and IPv4 information to the members of the multicast group. The most commonly shared information is the member struct, though other telemetry can be shared. This information is delivered via a UDP packet with a payload of only 37 bytes. All agents make decisions based off of the information stored within these

packets. Should the program receives a `SIG INT` or other shutdown message, the swarm net manager will multicast that the agent is unavailable. If an agent has not communicated to the multicast group within a predefined amount of time, then the members of that group also consider that agent unavailable.

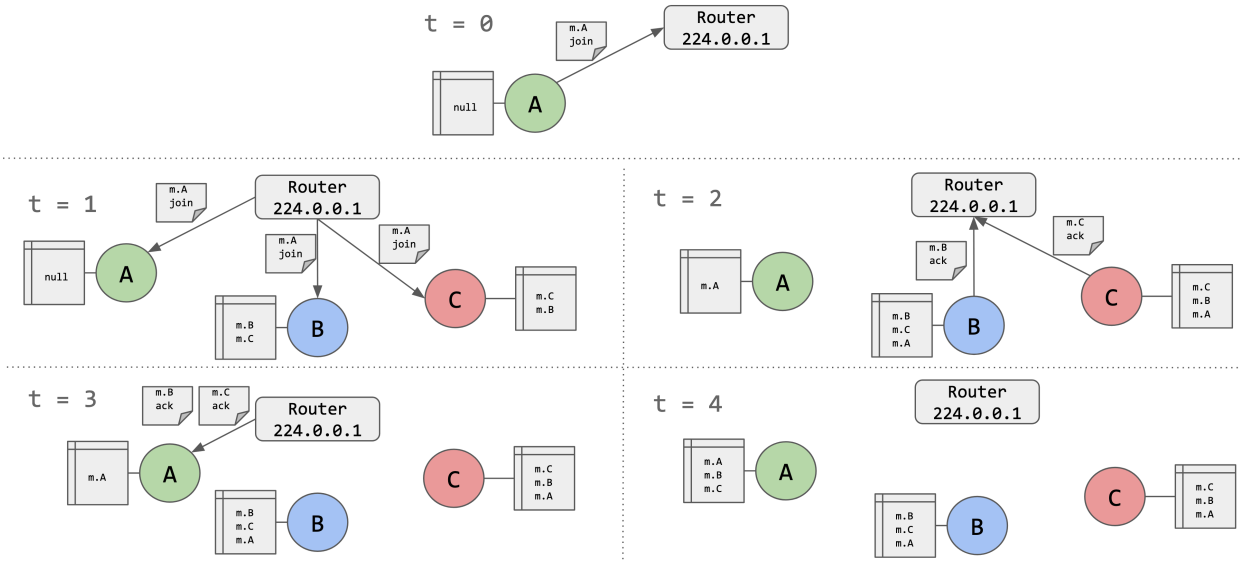


Figure 5.1: An example swarm join, where agent A multicats to join the swarm. The other agents in the swarm ACK with their member structs; agent A only updates its member struct after receiving its own JOIN.

Independent from the frequent state multicast, which runs in a separate thread, when an agent changes state it also multicasts the new state. This is done so that there is no delay between the agent's internal knowledge of its state change and the group's knowledge of the agent's state change. Relying only on the frequent state multicast could cause state mismatches between the group or an agent. Additionally, it would allow for an intermediary state, which the agent is only in briefly, to be missed by the group. Thus, a new thread is created and then destroyed to handle an adhoc state multicast. Such intermediary cases are not edge cases, as I will mention briefly in the following section, and are used to synchronize agents when cooperating. When in a particular intermediary synchronization state, an agent will open up a TCP socket for high volume data transmission.

5.2.2 Concept of Operations

The SSRL Swarm Net exists for multi agent cooperation to demonstrate how the MOCI software can scale to several satellite systems. Thus, the state tables which all agents hold internally are derived from MOCI's computer vision pipeline and the applications of that pipeline. Application specific states have been developed around the stages of MOCI's computer vision pipeline. Unlike MOCI, the Concept of Operations is defined at the swarm level. This is preferable for autonomy, as the benefits of satellite swarms are most clear when those swarms can be scaled effectively.

It is important to differentiate scenarios where swarm communication improves the group's 3D models and scenarios where communication cannot improve the group's 3D models. I consider an improved group 3D model to be a point cloud which is either larger in *contiguous* geographic footprint, has improved measurement accuracy, or both. To improve models or to join models into larger ones, there must be shared relevant information from the satellite systems. That information comes from measurements taken by the imaging systems. Thus, image overlap is the primary factor to consider when identifying scenarios for collaboration. There are three distinct scenarios I consider: A scenario where communication does not produce improved global models but local models can still be produced, a scenario where communication improves both local and global models, and a scenario where local models are impossible but global models are.

In addition to observation scenarios, there are also distinct stages within MOCI's 3D reconstruction pipeline where distribution is beneficial and where it is not. Stages which benefit most are feature generation, feature matching, and bundle adjustment; all other stages of the pipeline have no clear benefits from distribution. Sharing image information is not beneficial because the number of images to share may be prohibitively large, though it is necessary for the agents to share their local camera parameters. After features are detected and extracted, agents can share their relevant features with other agents so that

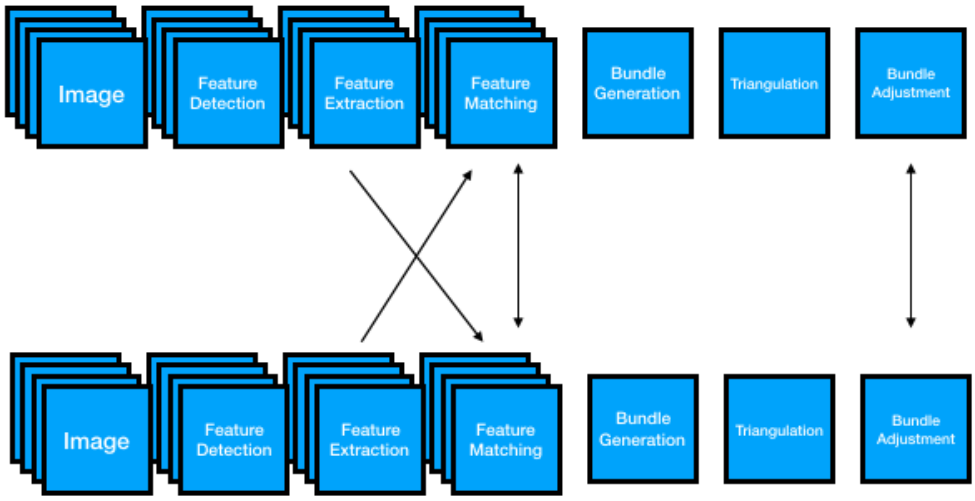


Figure 5.2: The directional arrows represent the flow of information between two agents who are cooperating.

cooperative matching can begin. In cooperative matching, the agents identify shared features and features from the other agents which match with locally available features. The agents run this independently and share their resulting matches with one another.

Chapter 6

Experiments and Results

The following chapter focuses on testing SSRLCV against the VSFM (Visual Structure from Motion) software package, NASA STRM data sets, ASTER datasets, and Google Earth Imagery. At the beginning of this research endeavor it was not clear that 3D reconstruction from LEO could work within the constraints of a cube satellite. The next section focuses on how I discovered what parameters needed to be tested, what objects could be feasibly reconstructed from orbit, and how existing technologies could be improved for this use case. Essentially, VSFM showed that SIFT feature detection and matching paired with triangulation and bundle adjustment could serve as a feasible pipeline for 3D reconstruction. The SSRLCV library was designed by considering how the initial VSFM results could be improved; the inner functions of SSRLCV are described in detail in chapter 4. SSRLCV uses a custom GPU accelerated SIFT implementation, matching, triangulation, filtering, and bundle adjustment.

The MOCI cube satellite requirements evolved, and became more strict, with the development of SSRLCV. At first MOCI was a 3U satellite with an optical system operating at 60 meters GSD. After initial simulations showed this configuration did not meet mission requirements, MOCI became a 6U satellite where an entire 3U is now used for an optical

system that operates at about 6.6 meters GSD. Diagrams of the MOCI system can be seen in chapter 2. It also became clear that a modified miniature GPU system would be ideal for onboard reconstruction. This system uses a custom PCB for the integration of a Nvidia TX2i into a cube satellite bus and is described in chapter 3. All tests after initial feasibility focus on that system.

6.1 Initial Feasibility with VSFM

The fundamental concept of the Multiview Onboard Computational Imager, MOCI, relies on the feasibility of Structure from Motion (SfM) from Low Earth Orbit. If this concept does not prove feasible, then the MOCI mission cannot be successful. Fortunately, it seems that this concept can be achieved by building on the foundations of some existing technologies and the methods mentioned in chapters 3 and 4. In this section I seek to quantify the initial feasibility of SfM with these technologies and explore alternative 3D reconstruction methods.

The reconstruction of the 3D geometry of objects, including the generation of point clouds, employs image processing concepts dating back to the 1950s. In traditional stereo photogrammetry, 3D structure was resolved from a series of overlapping, offset images. However, in order to produce usable models, knowledge of scene geometry, camera parameters, camera orientation, and ground control point (GCP) targets were required. I initially tested SfM, which differs from traditional photogrammetry because it does not require ground control, reference targets, or a prior knowledge of the camera exposure locations and attitudes. Instead, the geometry of the camera/scene parameters is resolved automatically with very little, if any, user interaction. The approach originated in the computer vision community and incorporates automatic feature detection and feature matching algorithms [44][49]. By using multiple overlapping images, most implementations of SfM incorporate simultaneous, highly redundant, iterative bundle adjustment procedures after extracting a set of features auto-

matically. As a result, very accurate point matching between photographs can be achieved, and a dense point cloud can be extracted. Previous SfM technique use in photogrammetry worked best with sets of highly overlapping images, with about 80% overlap, that capture the full 3D structure of a scene viewed from a wide array of positions [50].

Although internally consistent, models derived from SfM typically lack scale and orientation provided by GCPs. Consequently, resulting 3D point clouds are generated in a relative image-space coordinate system. Data must be aligned to a real-world, object-space coordinate system through the use of onboard GPS or georeferencing of ground based objects. A multidimensional data adjustment can be achieved by 3D similarity transforms using a few GCPs measured using known control points or GPS coordinates after the model is complete. The corresponding processing workflow would then include the consideration of a known control point and the definition of direction and dimension. In addition, control point insertion may involve the integration of 3D points measured on photos into the model solution.

Several software solutions exist to process a series of images and generate a point cloud dataset. Implementations include cloud based (Autodesk 123D Catch), free (as in price) software (Visual SfM/CMVS), open source (Meshlab, Insight3D), and commercial (Agisoft PhotoScan, Eos Systems PhotoModeler, University of Stuttgart SURE).

To first test the feasibility of the MOCI mission, open source options were used because of their adaptability. Additionally, to achieve multiple image angles MOCI will pivot as it approaches, passes over, and moves away from a target in question. This is not strictly necessary, but provides a method to vary image overlap percentage. When combined with image acquisition rates, this allows for multiple overlapping images, the key constraint in SfM and 3D construction from images. Finally, the onboard GPS and propagator will allow for images to be mapped on a usable coordinate system therefore providing GCPs for data analysis.

Initial feasibility tests of SfM in LEO were modeled with graphical simulations. The

concept was as follows: The Blender software package was used to generate a model of the Earth to scale. Other objects were also placed on the surface of the model. These objects were placed at scale, typically the size of mountain ranges. A camera, at the altitude of the satellite, was made to orbit the Earth, passing over the desired object or set of objects. The simulation allows the optical properties of this camera to be manipulated thus simulating the properties of essentially any optical system (within the constraints of a pinhole camera model). In the first case, I model the optical properties of GomSpace's NanoCam 1U, the camera initially selected for the MOCI mission. This camera has a 2048x1536 pixel array and has a GSD of about 60 meters per pixel. Blender then used this camera to take a series of images, mimicking the orbital imaging platform.

SfM was then performed on these images. Several tests were performed using the VSFM software developed by Chang Chang Wu [51]. Computer vision techniques involving image-based 3D modeling and Structure from Motion (SfM) were used to recreate the geometry of photograph acquisition and to reconstruct the 3D geometry of the Blender simulated imagers. SfM extracts (x, y, z) coordinates of objects that can be identified in more than one photograph and allows for the creation of textured models of the landscape. Algorithms for correspondence analysis and tie point identification (tie points are the photogrammetric term for matched feature points) usually employ a gradient descent based approach. These should not be affected by rotations of the camera positions, orientations of the camera positions, or scale of the image. This is known as rotation, affine, and scale invariance. Rotation and affine invariance were solved in the late 1980s but scale invariance was not solved until the invention of the SIFT feature detector and descriptor (e.g., SIFT - Scale-Invariant Feature Transform). The features resulting from this method show little or no change based oN-viewpoint or illumination, thus are very useful for point matching [45]. The SIFT method provides a set of image descriptors that each have an associated location; these descriptors obtained from two images form a set of keypoints. Based on the Euclidean distance between

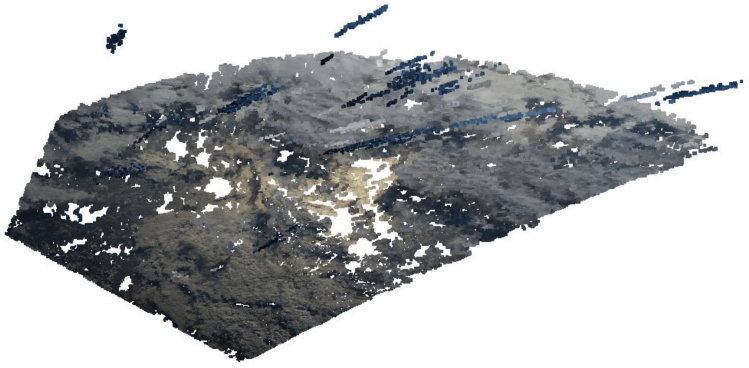


Figure 6.1: A 3D reconstruction of cloud tops, generated with VSFM, from images taken by the crew out of porthole windows. The point could consist of 301, 249 vertices.

matching point descriptors in the two images alone, up to 90% of incorrect key points are removed [52]. As a result, there is no need for rotating photographs and adjusting their scale during the pre-processing step of the workflow.

6.1.1 Initial VSFM Results

Before any quantifiable tests were performed, I performed SfM on some sample data sets that we thought might prove promising. Images from the Expedition 47 crew on board the International Space Station were used to generate a dense point cloud in Visual SfM. The sequence of shots was taken on March 25, 2016 from 11:45:02 to 11:57:17 GMT on a pass over the ocean with very dense cloud cover; this was thought to be an ideal image set for initial tests [53]. The set consisted of 30 images and Visual SfM was able to produce a dense point cloud of 301, 249 vertices.

Note that Visual SfM, as can be seen in figure 6.1, was able to distinguish several cloud layers in the upper atmosphere. Given that "anvil top" clouds, a nickname given to very large cumulonimbus clouds, have height between 6 km and 15 km, we can at least confirm

that SfM can distinguish a height differential of at least 15 km [54][55]. This of course depends fundamentally on the camera's spatial resolution but happily demonstrated promising results. Furthermore, the NC1U camera which MOCI was initially slated to use was discarded due to poor simulation results and a new camera was designed. The initial Blender simulations (done in late 2016) were only able to reconstruct objects on the scale of 10km while in a 400km orbit. Considering Mount Everest is about 8.8 km, this was unacceptable for the mission and is what led the mission to seek a custom optical system. The current optical system is briefly described in chapter 2 and will be used as a baseline in later sections.

There were, however, already significant issues with this initial point cloud and its computation. One such issue involves the computational power needed to calculate the point cloud. The above cloud was computed in 14 minutes on a PC with an NVidia GTX 980 graphics card. The graphics card on this system requires 165 Watts of power, produces a maximum temperature of 98 degrees Celsius [56], and cools convectively with a fan at sea-level atmospheric conditions. In late 2016, it was clear that this type of processing was not possible with existing 3U or 6U cube satellite hardware. Not only is this computationally difficult for a small satellite to handle, it also poses significant thermal risks, considering it will likely take more than 14 minutes to compute a similar cloud. In 2016 I concluded that the SSRL should place high priority on the design/development of a GPU for its MOCI satellite, as this presented significant challenges. Results from this system are discussed later in this chapter.

Another significant issue regarding the feasibility of on orbit 3D reconstruct is in regards to the already generated point cloud. In figure 6.1, and seen even more clearly in figure 6.2, there are artifacts of SfM processing, strange streaks of points high above the rest of the point cloud. It turns out that these streaks of points radiate out of each camera's position along the same direction that the camera's vector was pointing. Noise removal methods would be needed if accurate 3D reconstructions are to be obtained.

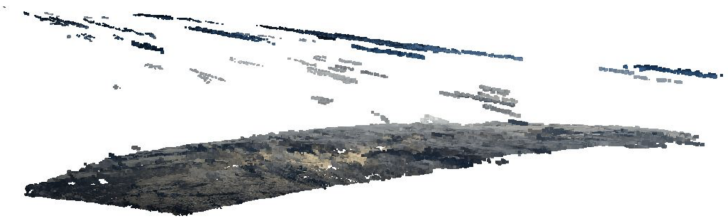


Figure 6.2: Noise radiating from cameras at the Space Station's position; this can cause issues while attempting to generate accurate 3D reconstructions.

At first, tests were primarily conducted by placing simple geometric solids on a roughly textured Earth while the Blender camera was made to move in a circular orbit at 400 km. VSFM was then used to generate a dense point cloud from those images, and next we used MeshLab to make a textured 3D mesh of the target and its surrounding area. At the time of this test SSRLCV was not developed and I had not yet convinced myself of the need to develop custom reconstruction software.

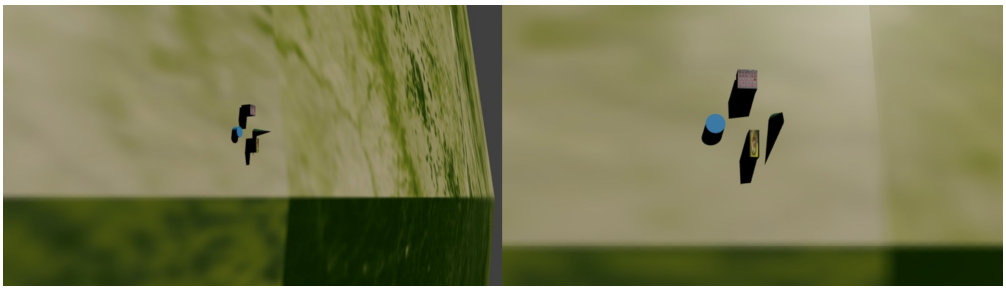


Figure 6.3: Simple geometric shapes being imaged with a 60m GSD imager from a 400 km orbit.

At this point the primary concern was image number, as it was previously believed that image number was a primary contributor to model quality (SSRLCV shows that this is not the case). Various image numbers were tested at various start and stop angles, producing a total of 22 test cases. An example initial simulation can be seen in figure 6.5.

Figure 6.4: An initial reconstruction with VSFM of simple geometry on the surface of a simulated body at a 400 km altitude. Camera position and orientation errors can be seen, as the camera path does not follow the arc of a circular orbit (especially on approach to nadir facing).

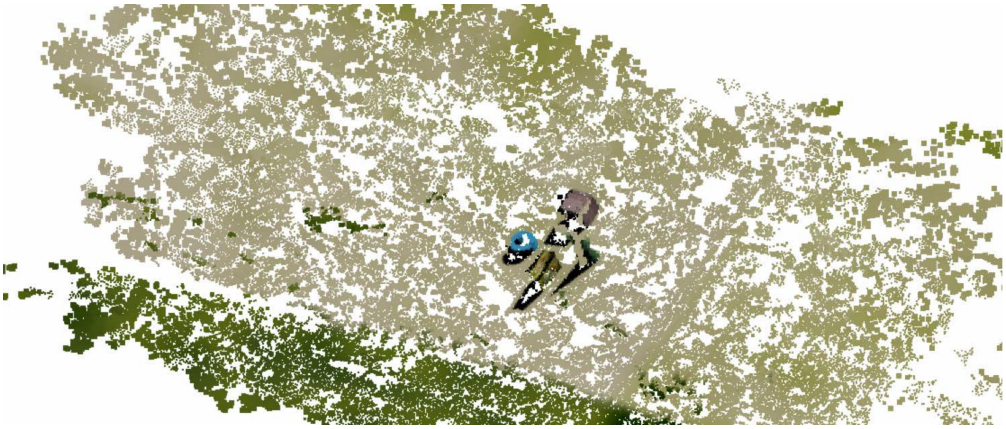
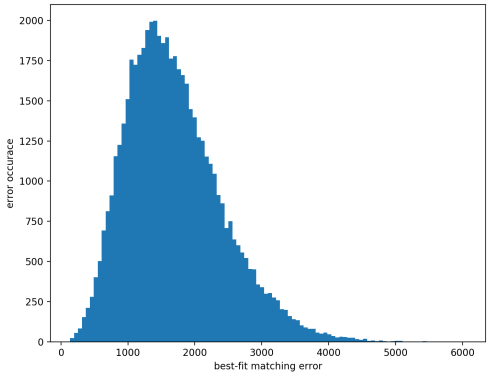


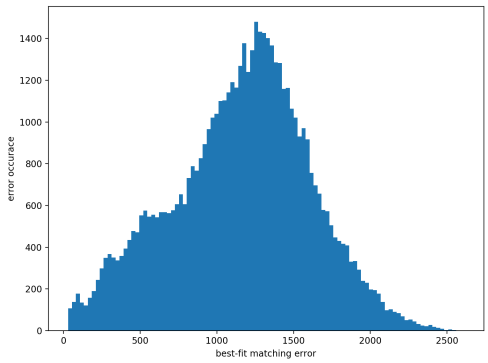
Figure 6.5: A full reconstruction of simple geometry with VSFM, showing that the scales used in orbital reconstruction were not outside the scope of common, openly available 3D reconstruction methods.

6.1.2 Stereo Disparity

To limit unknowns and have greater control over camera parameters, the images used to test our algorithms were simulated. These images were also produced with Blender, a standard 3D modeling software for animators, and utilized terrain data from the ASTERv3 global DEM [57]. Stereo disparity was calculated from a worst case low resolution (approximately



(a) Error distribution of best-fit feature descriptors without subpixel interpolation of feature vectors.



(b) Error distribution of best-fit feature descriptors with subpixel interpolation of feature vectors.

Figure 6.6: A demonstration of the improved errors when using subpixel interpolation methods in stereo disparity

315m GSD) simulation of Mount Everest imagery. The dense matching scheme for this image pair was naive brute force minimization matching at full pixel scale - there was no subpixel matching applied.

The histogram of best-fit match errors, shown in figure 6.6, shows how often differences between the sum of absolute differences (SAD) of feature vectors occur. The goal of the scale space transformations, described in section 2, is to decrease the magnitude of best-fit match errors so that subpixel matching can be more effective. The goal of the subpixel interpolation, described in section 4, is to move the histogram of SAD values towards the left (lowering error). The SAD is used over the sum of squared differences (SSD) because it better illustrates the magnitude of feature mismatching. The histograms in figure 6 and Figure 7 show a clear improvement when subpixel dense matching is used. This data is further illustrated in Table 2 below.

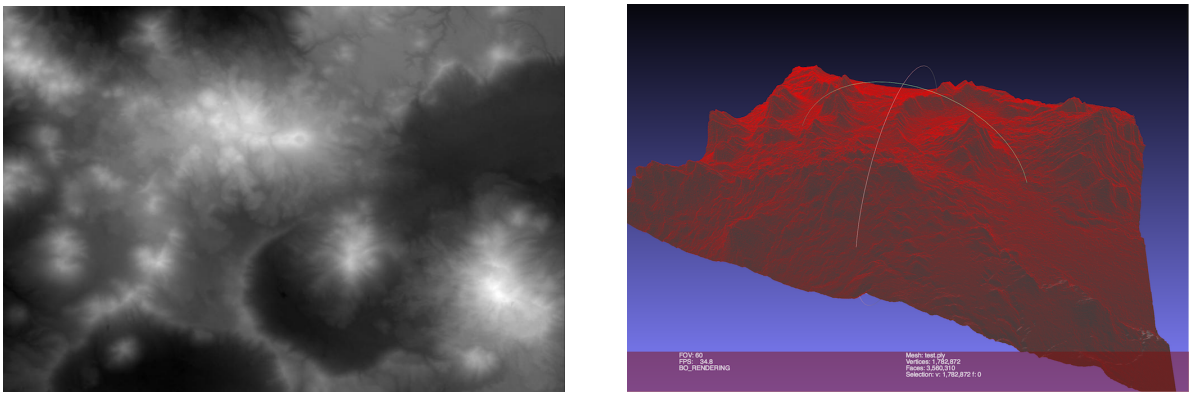
One significant difference between our method and the standard SIFT approach is that

Measure	Full Pixel Value	Subpixel Value
Average	1699	1157
Std. Dev.	723	451
Max	6050	2611
Min	136	136

Table 6.1: Statistical values of the SAD of best-fit feature vectors

Figure 6.7: A disparity map of Mount Everest calculated with our dense sift matches

we can generate $(n-18)^2$ features for an image. For the 254 x 254 image, $60,025$ features are generated. A standard SIFT approach used on the same data set (with default setting from the anatomy of SIFT implementation) only generates 690 features [46]. One downside to using our dense SIFT version is that our matches are less stable. However, if it is known that sequential images contain significant overlap (as is in our test cases), then there is a benefit to utilizing dense SIFT because a certain amount of image correspondence is expected. For example, dense SIFT can allow for stereo disparity rather than stereo multi-view and reprojection, greatly simplifying computer vision pipelines [58][21]. Stereo disparity, though it is not the focus of SSRLCV, can be desirable because it can be easily implemented on smaller, less optically complex cube satellites.



(a) The GeoTIFF height raster

(b) The PLY conversion

Figure 6.8: The Nevado Ojos del Salado Volcano in the Andes Mountain, the second highest mountain in the Western and Southern Hemisphere.

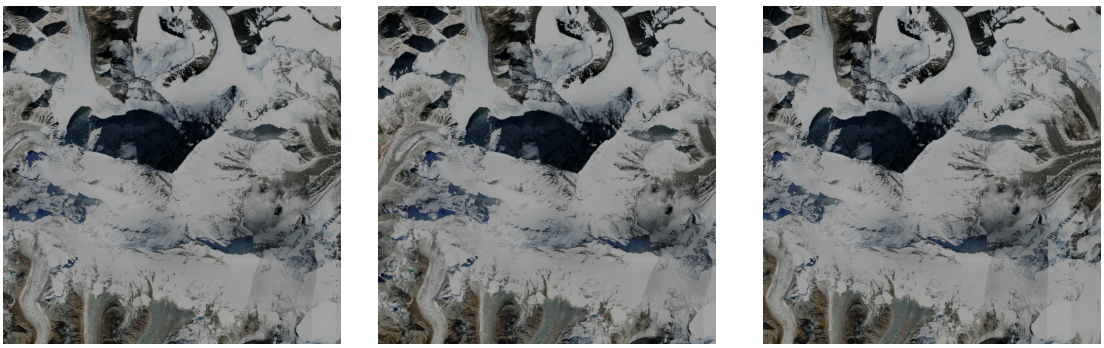
6.2 SSRLCV Simulations with Blender

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

The resulting ASTER meshes are between 15 and 90 meters in resolution, which means a given polygon face is a square $15m^2$ to a square $90m^2$. The resolution used for the manual ASTER to Blender testing is 30 meters. The rationale here was that, despite the models only being accurate within 30 meters, because the PLY format linearly interpolates the height

point locations in the raster it is possible to test for sub-30 meter accuracy. This means it is possible to generate a higher resolution model in the areas between raster height points. Additionally, the ASTER GeoTIFF and its resultant mesh can be viewed as a ground truth because the primary goal is to calculate height on a given surface. Different reconstruction methods can be compared by testing these same data sets with different methods.

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



(a) An Everest Rendering at nadir

(b) An Everest Rendering at $-10°$ off nadir

(c) An Everest Rendering at $10°$ off nadir

Figure 6.9: Example renderings of Mount Everest with real satellite data, aquired by Google and generated with the BlenderGIS addon. These images were simulated from a 400km circular orbit.

An additional advantage of using the BlenderGIS addon, instead of manually adding 3D models and textures, is that the Google imagery used is very high quality and scales

well to variable GSDs when modeling camera systems in orbit. Furthermore, the simulated images were generated with orientations from a slew maneuver where the camera was always looking at a fixed point on the ground. Examples can be seen in figure 6.9. This was done to ensure image overlap and have fine control over viewing angles. These simulations assumed a circular orbit of 400km and the desire was to input a certain θ viewing angle and output the satellite's orbital position at that viewing angle.

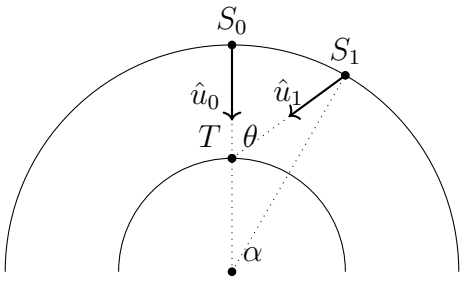


Figure 6.10: An example scenario showing the orbital plane where S_0 is the satellite looking nadir and S_1 is the new off-angle of the satellite. Here we desire to compute the rotation α with respect to the orbit center that S_1 should rotate (the local rotation θ is known) so that it is looking at the tracking point T.

First, consider that we need only look within the orbital plane of the satellite. To calculate the satellite's position within this orbital plane, we first consider the equation of a sphere with the radius of Earth, r_e, in km then consider the satellite has an orbit of h km above the surface of Earth. Next, consider the equation of a line with slope $m = tan(\theta - \frac{\pi}{2})$ where θ is the desired imaging angle. Substitution and distribution (note that all terms but x and y are constant) result in the quadratic seen in equation (6.1) which can be solved for the (x, y) position of the satellite within its orbital plane. I choose to define the tracking location T as the origin for simplicity.

$$
\begin{aligned}
x^2 + (y - r_e)^2 &= (h + r_e)^2 \\
y &= tan(\theta - \frac{\pi}{2})x \\
x^2 + (tan(\theta - \frac{\pi}{2})x - r_e)^2 &= (h + r_e)^2 \\
x^2 + (tan(\theta - \frac{\pi}{2})x)^2 - 2r_e tan(\theta - \frac{\pi}{2})x + (r_e^2 - (h + r_e)^2) &= 0 \\
(1 + tan(\theta - \frac{\pi}{2})^2)x^2 - 2r_e tan(\theta - \frac{\pi}{2})x + (r_e^2 - (h + r_e)^2) &= 0
\end{aligned}
\tag{6.1}
$$

If one wishes to calculate the angle α in refrence to the orbiting bodies center of mass, simply use equation (6.2) below:

$$
\alpha = arcsin\left(\frac{x}{r_e + h}\right) \tag{6.2}
$$

Given the orbital inclination, the point can be translated then rotated into its intended position. A simple python script (located in the SSRLCV utility repository) named `track_camera_gen.py`, was written to solve this and the output of that script is used as position and orientation data for the BlenderGIS simulations. From this information slew times, orbital arc lengths, and ground track lengths can be generated. If a circular orbit is assumed and a constant radius for earth can be considered accurate, the the simple arc length formula $L_{ground} = \alpha r_e$ can be used to find the ground track. Similarly, the formula $L_{orbit} = \alpha(r_e + h)$ can be used to find the orbit length. The velocity of a given orbit (using the assumtion that the mass of the cubesat is negligible) is caclulated with the equation (6.3) below:

$$
v \approx \sqrt{\frac{GM}{r}} \tag{6.3}
$$

In equation (6.3) the radius r is the combined altitude and earth radius, so $r = r_e + h$.

The standard gravitational paramter is $G \cdot M$ and is commonly used. Using the velocity v of the orbit, the time of a pass is caclulated simply with the equation $t = \frac{L_{orbit}}{v}$. The slew rate can be caclualted simply with the equation $v_{slew} = \frac{\theta}{t}$ for a slew to nadir or $v_{slew} = 2\frac{\theta}{t}$ a slew at the starting a angle θ. Nadir pointing, rather than point tracking, can be caclulated in a similar way; to caclulate a nadir slew simply use the angle α instead of θ.

Altitude (km)	θ° (deg)	time (s)	track slew rate (θ°/s)	nadir slew rate (α°/s)
400	5°	4.5624	1.0959	0.0648
400	10°	9.1890	1.0882	0.0648
400	15°	13.9474	1.0754	0.0648
400	20°	18.9126	1.0574	0.0648
400	25°	24.1717	1.0342	0.0648
450	5°	5.1515	0.9705	0.0641
450	10°	10.3746	0.9638	0.0641
450	15°	15.7449	0.9526	0.0641
450	20°	21.3458	0.9369	0.0641
450	25°	27.2741	0.9166	0.0641
500	5°	5.7446	0.8703	0.0634
500	10°	11.5683	0.8644	0.0634
500	15°	17.5542	0.8544	0.0634
500	20°	23.7942	0.8405	0.0634
500	25°	30.3943	0.8225	0.0634

Table 6.2: Various slew times for both nadir pointing and point tacking during image aquisition. The off angle θ is the angle seen in diagram 6.10; the times shown here are from the off angle to the point track nadir position (the surface normal/zenith of the track point).

The slew rates measured in table 6.2 are very achievabl for cube satellites. The Blender-GIS simulations use some of the various rates ablve. The Everest 2 view and 3 view sets use the 400 km 10° track slew rate. The Everst 5-view test set uses the 400 km 20° track slew rate. The Rainier 2 view and 3 view sets use the 400 km 5° track slew rate, while the Rainier 5-view sets use the 400 km 10° track slew rate.

6.2.1 Noise Removal

Here I consider noise to be any point which should not be within the pointcloud. Noise removal is a significant challenge when designing 3D reconstruction algorithms, and for the SSRLCV it is necessary to remove erroneous 3D points after the initial 3D triangulation. Points are assigned particular errors, described in section 4.5.1, and can be filtered based on these errors. The noise is assumed to be Gaussian. A combination of linear cutoff filtering (removes all points past a certain error) and statistical filtering (removes all points past a certain multiple of the standard deviation σ in the error distribution) yields very clean point clouds. Additionally, filtering at the feature matching level should be used. In the context of the SSRLCV software, this is known as seeding, where an image known to be unrelated to the satellite images is input as an example to counteract false positives. David Lowe, the creator of the SIFT algorithm, recommends this approach to improve matching [44] [45]. If no filtering is done at this stage, then the resultant point clouds are considerably noisier, thus reconstruction without a seed is never recommended.

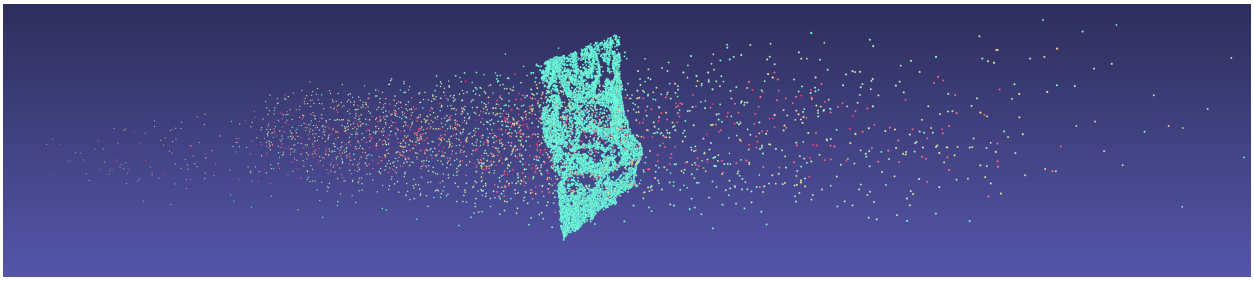


Figure 6.11: A visualization of the errors calculated for a 2-view case with filtering at the feature matching level. Red represents points with errors which are extrema and light teal represents errors which are close to zero.

When filtering 2-view reconstruction, the benefits of statistical filtering are most evident wheN-viewing the distribution of errors. The intuition here is to remove the erroneous points which lay outside of the Gaussian. At first the distribution is quite large, but still densely packed around the optimal error of zero. It is important to observe that there are very few

extremely erroneous points, but those points often contribute significantly to the total error of the point cloud. In order to perform a proper bundle adjustment, described in chapter 4.5, such points must be removed; otherwise their collective error may dominate the optimization resulting in a local minima of noise at the expense of valid points.

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

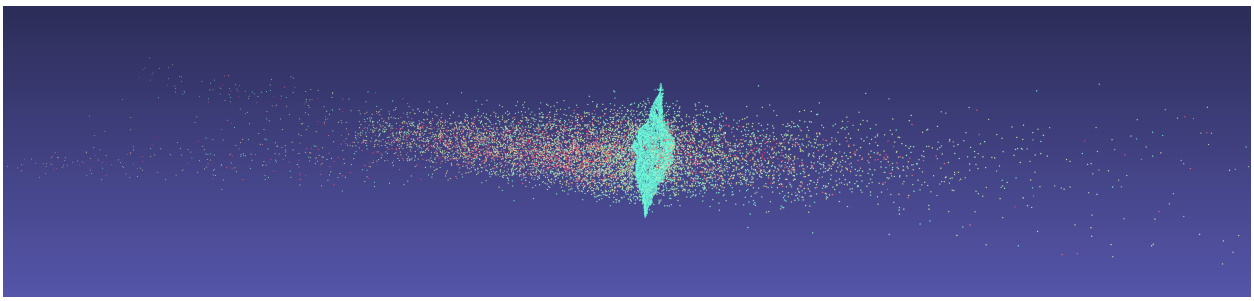


Figure 6.12: A visualization of the errors calculated for a 5-view case with filtering at the feature matching level. Red represents points with errors which are particularly bad and light teal represents errors which are close to zero.

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

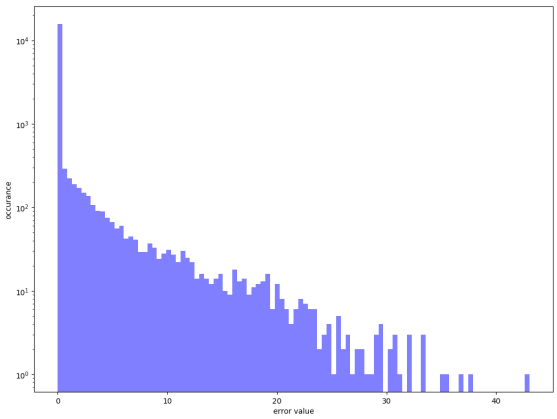
Table 6.3: Data on error removal correlated to point removal when filtering. Note all statistical filtering occurs after seed filtering, as this is the expected usage.

There is considerably more noise, and thus more error in the cloud, for N-view cases. Overall the 2-view cases produce less noisy clouds that are about as accurate as the N-view clouds after filtering. N-view clouds require significantly more filtering to extract a viable model. This is because the N-view triangulation produces a wider spread of errors than

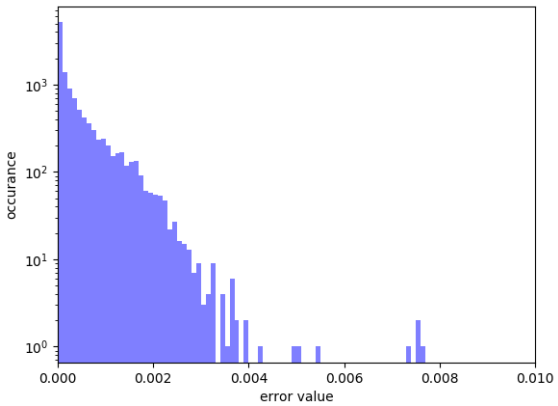
Dataset	Res	Views	unseeded	seeded	3σ filter	2σ filter	1σ filter
Everest	1024	2	0.7316	0.2266	0.2105	0.2079	0.2039
Everest	1024	3	0.7136	0.5124	0.4811	0.4646	0.4245
Everest	1024	5	0.7717	0.5796	0.5663	0.5506	0.5093

Table 6.4: Average distance, measured in km, of a point to 6 neighbors at certain error function filters.

the 2-view triangulation, seen in figure 6.14. This causes statistical filtering to time longer because it only removes a certain percentage of outliers at a time.



(a) The distribution of linear errors in a 3-view triangulation.



(b) The distribution of linear errors in a 2-view triangulation.

Figure 6.13: The difference of error histograms in the N-view and 2-view cases.

6.2.2 Reprojection and Triangulation

The tests with Blender and BlenderGIS seek to provide accuracy measurements which can be compared with current methods for the computer vision software developed for the MOCI mission. One of the most commonly cited existing reconstruction methods is the VSFM software released by Chang Chang Wu. Images rendered in BlenderGIS are output in the PNG format, but VSFM only accepts the JPG format. Images are converted from PNG

to JPG using FFMPEG at its highest conversion setting with the simple command `ffmpeg -i 1.png -qscale:v 1 1.jpg`. FFMPEG is a program which can be brew, apt, or yum installed on your favorite UNIX-like operating system.

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

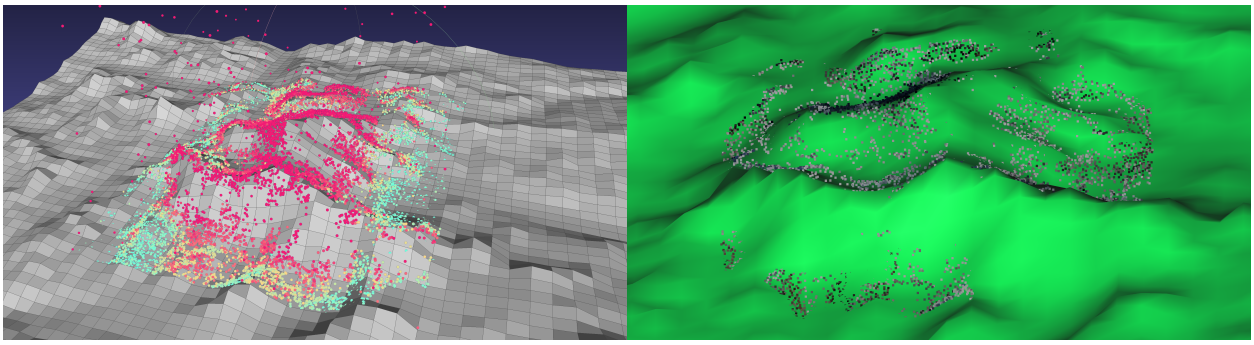
Meshlab is used to view the individual point cloud results. The error between the ground truth model and the initial 2-view point cloud is calculated as a sum of the distances between individual estimated 2-view points and their intersection with the 3D model below them.

Generally, view number is not correlated with an increase in model resolution for SSRLCV, but is correlated with model resolution for VSFM; this can be seen in table 6.5. In some cases, the noise removal process skews the ground truth error distribution to make the final SSRLCV model appear worse than it truly is. This effect is seen when the removal of a small number of points causes the large variance σ to decrease to a reasonable amount. The Everest 1024 5-view case, seen in table 6.5, has only 31 (only 0.12% of the $24,901$) points which cause the variance to increase an order of magnitude. This shows the importance of successive filtering, as these points should be removed during the filtering step. Similar results can be seen with a 2-view 4096 Everest case, thus successive filtering is necessary for all cases.

Errors to the ground truth are close in SSRLCV 2-view cases but not in VSFM 2-view cases. In fact, some VSFM cases fail to generate 3D point clouds because camera parameters cannot be accurately determined; this can be seen in table 6.5. These errors are visualized in figure 6.14 where the reconstructed points hug the ground truth. Overall, SSRLCV is

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

Table 6.5: Initial reconstructions, compared by average error to the ground truth and the sigma value of the error distribution. The number of points in the clouds is also provided. Cloud Compare's Iterative Closest Point (ICP) implementation was used to calculate the errors shown above. In some cases VSFM failed to produce a point cloud. Results with a $*$ were manually filtered after automatic filtering to show the ideal results of SSRLCV (VSFM results require no filtering and are thus already ideal and marked with " $-$ "; results in the row above should be used to compare).



(a) A 2-view 4096 × 4096 SSRLCV Everest reconstruction compared to the ground truth model.

(b) A 5-view 1024 × 1024 VSFM everest reconstruction compared to the ground truth model.

Figure 6.14: Here SSRLCV and VSFM are shown being compared to the NASA STRM data, used as a ground truth comparison.

both more dense and more accurate than VSFM. VSFM produces no noise in any model, but does so at the expense of generating dense points. SSRLCV generates as many points

as possible and allows the user of the pipeline to filter points at any stage. The benefit of such a design choice is accuracy, but the propagation of erroneous points into further stages of the pipeline can have tremendous consequences; this is explored in the next section.

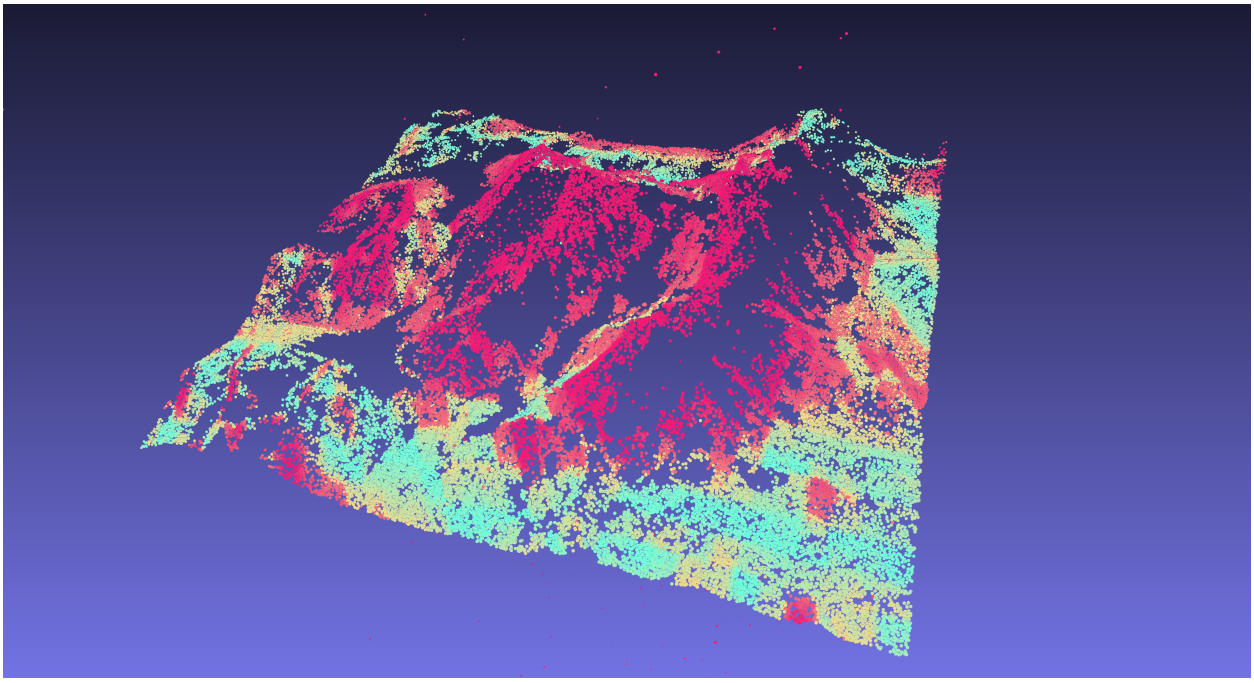


Figure 6.15: A 2 view reconstruction of 4096 x 4096 images of Mount Everest simulated at a 400 km ISS-like orbit; here the red points represent points with errors to the ground truth at or above 300 meters; the yellow represents errors around 150 meters and light teal color represents errors approaching zero.

The reconstructions from simulated Everest imagery work quite well, though an unexpected result was the larger errors at the peaks and higher altitudes as seen in figure 6.15. The figure also shows how the errors closer to the ground surface are better; this may be partially explained by the fact that the imagery used to perform the reconstruction is higher resolution than the Shuttle Radar Topography Mission (SRTM) which was used to compute the ground truth. This may also be explained by the fact that accurate regions tend to be more dense and not contain low density regions. The tips of the mountain contain regions of very low density and may be contributing to the error. VSFM produces a very sparse cloud at all locations, with almost no points at the sparse high altitudes of SSRLCV.

6.2.3 Bundle Adjustment

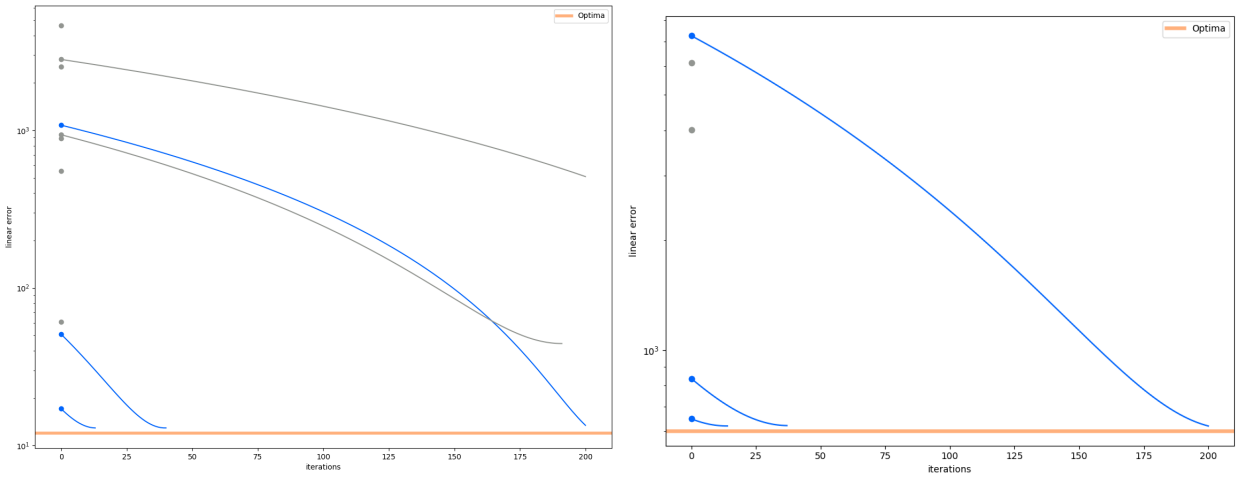
To test the Bundle Adjustment, noise was added to camera parameters so that the optimization could be observed. The Bundle Adjustment does improve the linear error of the point cloud but seems to get stuck in local minima. It is important to note that points and position estimation cannot be decoupled in the real world usage of MOCI, thus pointing estimation and position estimation are always considered together. Error values in position are from SGP4 and TLE various, which can diverge from the true location by up to 1 km a day [59].

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

Table 6.6: Initial bundle adjustment results where a noise vector is added to the camera parameters of one view in the set. Noise is added where the mean value is $\mu = 0$ and σ is defined as a positive and negative range. Error values are measured in Average Linear Error, mentioned in section on bundle adjustment in chapter 4. Each row was repeated several times with random noise added within the specified parameters. Successes, emasured in the right hand column, are tests that reach within $2\times$ the global optima.

Iterations can be seen in graphs 6.16, where noise was added as described in table 6.6. Convergence in not always guaranteed, which could be caused by some of the following issues:

improper definitions of error functions (perhaps the functions are not properly designed for convex optimization), floating point error associated with camera parameters, errors associated with the discrete approximation of the gradient and Hessian, or an elusive bug.



(a) 10 bundle adjustment tests run with noise from table 6.6; the values graphed here show the first tests in the table.

(b) 5 bundle adjustment tests run with noise from table 6.6; the values graphed here show the second tests in the table.

Figure 6.16: Here two batches of bundle adjustments are tested. Unfortunately, convergence seems less likely than desired, only occurring in some cases. The blue cases show convergence to the optima shown as an orange line, with the x axis representing the number of iterations. Bundle adjustment currently returns once a local minima is found.

Image size is not a contributor to bundle adjustment convergence; the largest factor seems to be the error values that are given as noise parameters. When the noise parameters are large, they result in a less optimal starting position for the bundle adjustment. Poor starting locations can be seen at the top of graph 6.16a, where only $\frac{1}{7}$ tests converged. On the other hand, locations at the bottom of the graph converged in $\frac{2}{3}$ cases. The 4096 case is similar, where lower errors converged but only some higher errors converged. However, because the sample size is small, these tests may not indicate any overall trend, though it is expected that noise closer to the real camera parameters should make convergence easier.

6.3 Pipeline Timing

The execution times of various pipelines in SSRLCV depend on the number of input images. More precisely, the execution times are bottlenecked by feature generation, feature matching, and bundle adjustment. Triangulation, filtering, and point normal estimation all occur very quickly. File IO is fast and not a major factor in execution time; it is essentially negligible. The tests seen in table 6.7 were run on an Nvidia TX2i.

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

Table 6.7: Runtimes for given sections of the pipeline on given datasets. Bundle adjustment was limited to 10 iterations and only tested on 2-view cases. In cells marked with " – " no bundle adjustment was run. Total runtime is listed on the right and may be slightly more than the individual sum due to small operations between pipeline stages. The total time also includes seed image feature generation. The same seed image was used for all tests (it runs in ≈ 9.448 seconds).

The MOCI mission is expected to process datasets which are closest to the Everest 4096 and Rainier 4096 sets used in table 6.7. The tests above indicate that MOCI's operations may take anywhere from 35 minutes to 1.4 hours. The sets above were selected because they are expected to yield best and worst case results for timing. The Everest dataset represents a best case computation for MOCI, where the low contrast of the snowy grey mountains would produce fewer features than the green high contrast Rainier dataset. One should note that the number of points in the point cloud is the same as the number of matches not filtered. The point number comparison of Rainier and Everest in table 6.5 shows that the

Rainier dataset generated about 2.1× more points than the Everest dataset. A common MOCI dataset will be somewhere in between these two examples.

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

6.4 Hardware Experiments

Individual cores can be shut off with the command `sudo nvpmodel N` where `N` is a mode number defined in table 6.8. The goal of initial experiments is to identify where the CORGI / TX2 / TX2i system will have significant power usage. Thermal properties are not modeled, but could be modeled at a later date by repeating these tests in the UGA SSRL's thermal vacuum chamber. Power is measured instead of thermal output because any non-vacuum measurements of thermal output will be inaccurate. Power, however, is directly correlated with thermal output and can be used to refine thermal models of the computation unit. For example, one could assume a 100% power to heat conversion to obtain worst case thermal models from the data below.

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

Table 6.8: Power Modes for the TX2 / TX2i

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

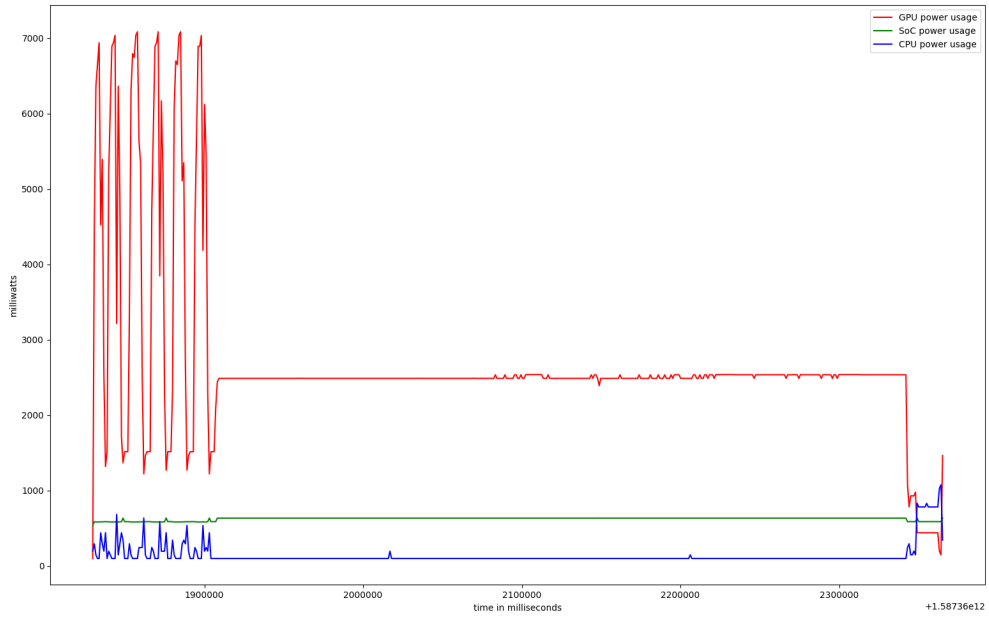


Figure 6.17: A 5-view reconstruct of 1024 $\times$ 1024 images of Everest showing the power consumption in milliwatts of the GPU, SOC, and CPU over the time the pipeline is executing. Tested on a development TX2.

The power graphs seen in 6.17 and 6.18 show the computer vision pipeline over time. The graphs track the power usage in milliwatts of the 3 primary systems on the Tegra, the SoC, the GPU, and the CPU. It is important to note that 6.17 and 6.18 do not contain bundle adjustment calculations in their pipelines. The six power spikes at the beginning of the pipeline in 6.17 are the result of feature generation; one spike is from the seed image and the

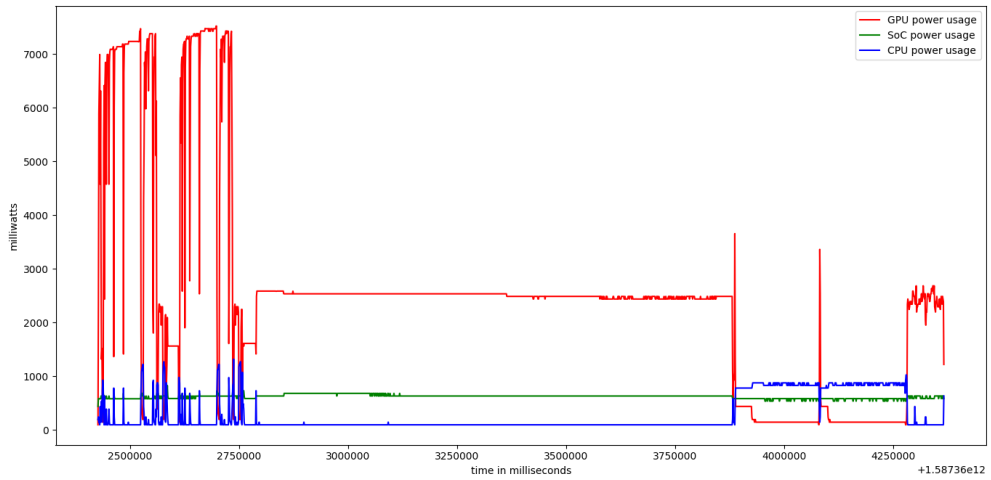


Figure 6.18: A 2 view reconstruction with 4096 × 4096 images of Everest showing the power consumption in milliwatts of the GPU, SOC, and CPU over the time the pipeline is executing. Tested on a development TX2.

other five spikes from the main images. Surprisingly, feature generation, not matching, is the most power intensive. The flat sustain of GPU usage come from filtering and triangulation. The CPU power usage tends to spike during memory transfers to and from the GPU while the SoC power usage tends to remain constant. The graphs 6.17 and 6.18 were generated from the TX2 and not the TX2i.

The TX2i produces power results that are *almost* identical to those of the TX2. These can be seen in graphs 6.19 and 6.20, but are about a Watt higher at the feature generation peaks. These results are promising, especially when considering the timing seen in table 6.7. Whereas Bundle Adjustment or Feature Matching may have initially seemed to be the most intensive sections of the pipeline, when considering power usage, Feature Generation emerges as the primary thermal, power, and temporal constraint. Matching and Bundle Adjustment, though they will run longer on average than Feature Generation, use less than half of the power. Thus, Matching and Bundle Adjustment will deplete MOCI's batteries at a slower rate and generate a less intense thermal load.

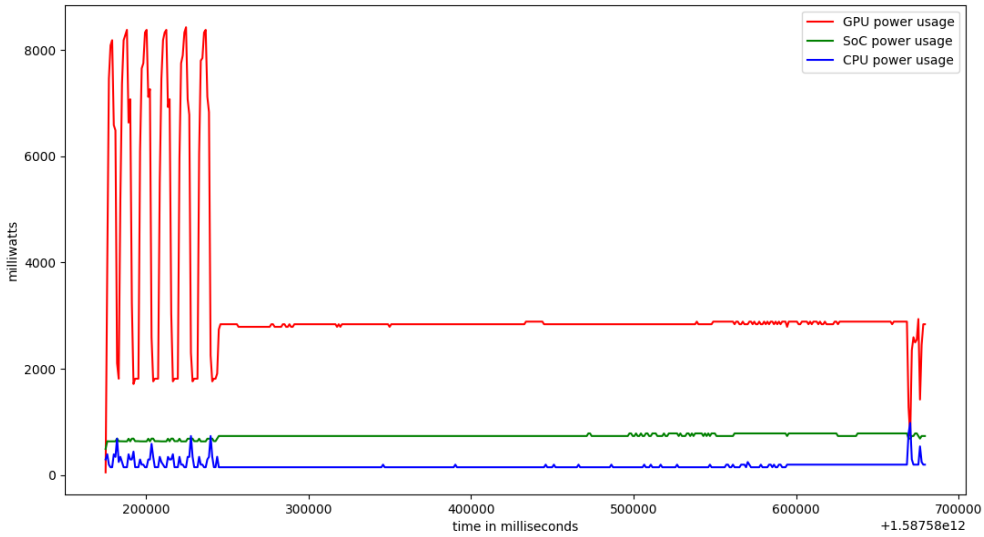


Figure 6.19: A 5-view reconstruction of 1024 × 1024 images run on the TX2i, the intended computation unit for the MOCI satellite, showing power usage over time. The power usage is almost identical to that of the TX2. Power consumption in milliwatts of the GPU, SOC, and CPU over the time the pipeline is executing.

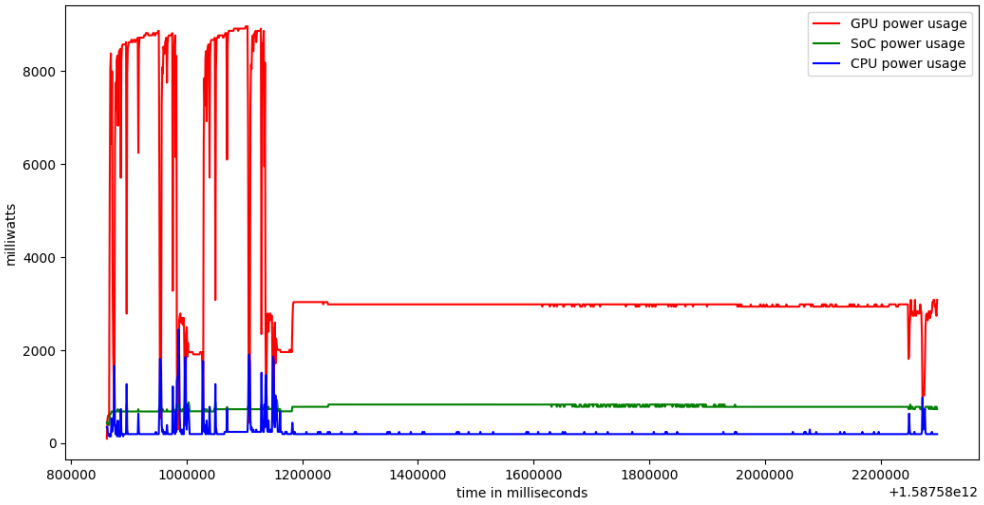


Figure 6.20: A 2-view reconstruction of 4096 × 4096 images run on the TX2i, the intended computation unit for the MOCI satellite, showing power usage over time. The power usage is almost identical to that of the TX2. Power consumption in milliwatts of the GPU, SOC, and CPU over the time the pipeline is executing.

Chapter 7

Conclusion

The results presented in this thesis demonstrate that, unlike previously assumed, the MOCI satellite does not need to have upwards of 30 images to produce an accurate 3D reconstruction to satisfy mission requirements. Instead, mission requirements can be met within reconstruction accuracy bounds with as little as two images. The resutls from prior tests may considerably relax pointing requirements for the mission. The only instance in which an image count greater than 2 is significantly beneficial is if the desire is to cover a larger land area, and thus produce a larger model. In retrospect this is obvious, as most topography uses 2-view methods in combination with long scans. The MRO HiRise instrument is an example of this, and it is able to produce 3D surface models of Mars at a resolution of 0.25 meters per pixel. Thus, if the MOCI mission is to improve the end data product, it makes more sense to explore longer-scale *scan like* images that can be used to produce larger models. In these cases it is only necessary to have any given image overlapping with at least 2 more, and no others, thus forming a chain of overlap. Any additional imagery seems only to contribute to noise and not stable feature points. An exploration of chained 2-view matching may also be worthwhile, as the N-view cases produce considerably more noise.

Based on the results in the previous chapter, there are a few new restrictions to the

MOCI mission:

- **SGP4 Propagator Constraints:** Errors due to the onboard SGP4 propagator can peak at $\approx$ 1 km error. Very little attention has been given to improving onboard propagation or correcting the propagation via TLEs or other telemetry in operations. To ensure optimal convergence on an accurate 3D model during bundle adjustment, propagation error should be minimized too.

- **N-View Reconstruction Time:** Even a 3 view reconstruction of 4096 $\times$ 4096 images takes over 3 hours. While MOCI could stop and restart such computations with checkpointing, this could pose problems because checkpointing cannot occur during the middle of a feature generation without significant additions to SSRLCV. Feature generation is the most power intensive stage of the pipeline and doing several long feature generation cycles will dump significant thermal loads into the system. Thus, if an N-view reconstruction is desired, MOCI must be capable of stopping (i.e. have the available power and thermal dissipation capability) after each feature generation and loading to restart at the next image after some time has passed. A stop and start is possible with current SSRLV checkpointing, but no analysis has yet been done to show MOCI has the power and thermal range to sustain several feature generation spikes. Future SSRL members should seek to test this in the thermal vacuum chamber.

- **Sub-Optimal Bundle Adjustment:** Unfortunately bundle adjustment is currently sub-optimal. It is not guaranteed to find a minimum and often when it does, it is only a local minimum. The MOCI team at the SSRL may choose to accept this and perform a bundle adjustment when possible. I recommend that quasi-Newton methods for optimization be explored because I speculate that the finite difference calculations of the Hessian and the gradient are causing the imperfect results.

- **Pipeline Bottlenecks:** The SSRLCV pipeline has power, thermal, and temporal

bottlenecks. Feature generation is the most intensive stage and is a power, thermal, and temporal bottleneck. Feature matching is primarily a temporal bottleneck, but does use about 3 Watts of power for a significant time. Bundle Adjustment is primarily a temporal bottleneck, but does use about 3 Watts of power for a significant time. Only some consideration has been given to how to start and restart the pipeline. Checkpointing is possible, but not widely implemented and certainly not yet widely tested. Significant development and testing should be done on starting and restarting the pipeline at the bottlenecks described here.

Based on the results in the previous chapter, there are a few relaxed restrictions on the MOCI mission:

- **Optimality of 2-View Reconstruction:** 2-view reconstruction is more than sufficient to achieve MOCI's mission objectives and the previous requirement of 30 images for a reconstruction should be relaxed. In fact, N-view reconstructions are currently just as accurate as 2-view reconstructions, but take much more time to compute and produce considerably more noise.

- **Non-Issue of Slew Maneuvers:** The off normal angles MOCI requires to achieve an accurate 3D reconstruction now range between 5° - 20° from a previous image, but are much less strict than previously considered. Thus, MOCI will be able to meet mission requirements without fast slews.

- **Non-Slew Options:** To lessen the slew restrictions further, MOCI operators could consider imaging a desired target on a different orbit rather than on the same orbit. The research conducted here has uncovered several 3D mapping missions which successfully produce models from different orbits.

- **Pointing Jitter:** As long as MOCI's bundle adjustment converges to a local minimum and camera position is accurately estimated. Any jitter in the extrinsic camera parameters is corrected for in bundle adjustment.

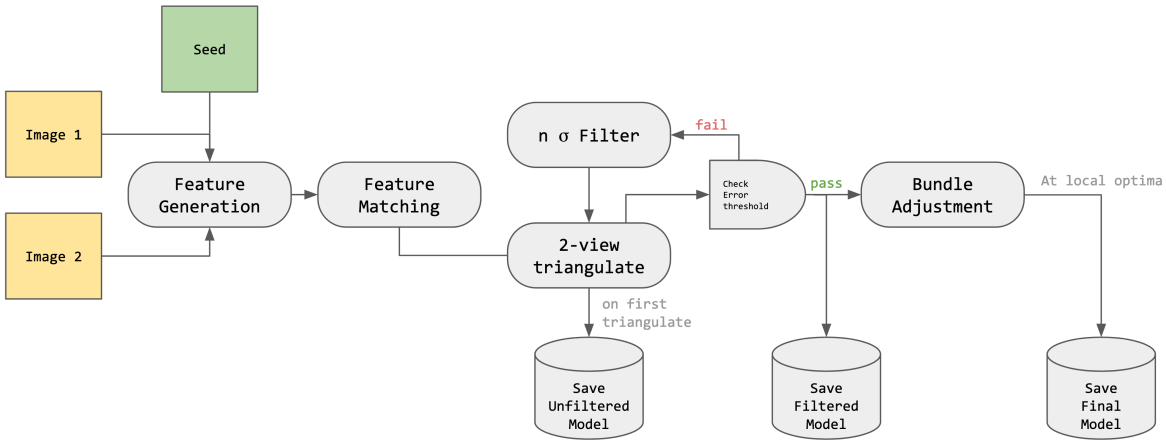


Figure 7.1: The recommended SSRLCV pipeline for the MOCI mission. Checkpointing should be possible at any stage or intermediate stage.

The tests in this thesis were not performed on a flatsat as is customary in the satellite design cycle. This is partially because a flatsat would not change the results of the computation but would make the testing of the computation much harder. I recommend that the UGA SSRL seek to implement several of the tests laid out in this thesis on a flatsat so that valuable experience can be gained.

7.1 Limitations of Experiments

The limitations of the experiments here leave much room for future work. No experiments were performed in vacuum, and the pipeline experiments here could be repeated in the SSRL vacuum chamber to validate or invalidate thermal models mentioned in this paper. Additionally, the software was not tested with randomized bit-flips at a rate that would be consistent with the radiation environment of LEO. Such an environment could be simulated

with additional software and may shed light on areas that need improvement. The SSRLCV software should also be tested on a drone testbed to validate with real-world data acquired in house, rather than relying on data provided by others.

7.2 Publicly Released Software

The computer vision software demonstrated here, known as SSRLCV, is open source and will be available on github at `https://github.com/uga-ssrl/SSRLCV`. The dataset used will also be publicly available on the UGA SSRL website at `http://smallsat.uga.edu/orbitalReconstructionDataset`. The swarm network demonstration software is also open source and will be available on github at `https://github.com/piepieninja/SSRL-Swarm-Net`.

7.2.1 Future Work

A pushbroom camera model was nearly completed for this thesis research, but I did not have the time to perfect the model. The MRO HiRISE system uses 2-view reconstructions with a pushbroom imager to generate the current best 3D models of Mars. It may be possible to reconstruct 3D models of earth using the SPOC satellite, despite the high GSD.

A GUI, which interfaces with SSRLCV, was partially developed and may still be released with SSRLCV via github.

7.3 General Considerations for In-Orbit 3D Reconstruction

This research highlights a few key factors of onboard 3D orbital reconstruction with small satellites. As is true with standard 3D reconstruction, image overlap is fundamental. How-

ever, increasing the number of views does not necessarily increase the accuracy of a final model. The primary indicator of the accuracy of the final model is the number of matched points used to triangulate 3D points and the GSD of the imagery used to acquire features. The more numerous and accurate these matches, the more dense and accurate the final model. Unfortunately, feature generation and matching can be quite time consuming (even when using a hardware accelerated platform) because a given feature must often be matched against all others; any reduction of the matching search space is beneficial. Additionally, noise in position and orientation estimation is an inevitable consequence of digitizing input signals; a small amount of noise in position or orientation results in significant triangulation error due to scale. Thus, formulating the minimization of such noise as an optimization problem is necessary; in the context of MOCI software this is known as bundle adjustment and is covered in detail in chapter 4. The bundle adjustment is the most temporally demanding operation of 3D reconstruction. Any effort to minimize function evaluations per iteration, calculate the camera gradient more efficiently, calculate the camera Hessian more efficiently, or restrict the search space is greatly beneficial.

When considering extensions of such technologies to small satellite swarms, one should seek to share information only if the information improves models. Thus, sharing features, collaborating on matching, and some flavor of distributing gradient descent are all highly likely to be beneficial. No other forms of collaboration are immediately beneficial.

When considering potential hardware improvements, the goal should be to further minimize computation time and power usage. FPGA and ASIC technologies offer interesting paths for future research. SIFT-like feature extraction implemented on ASICs (or FPGAs) could have considerably lower power usage and significantly faster runtimes than the methods implemented in this research. Optimization problems may still need the flexibility of a GPGPU, but shared memory could be used to offload expensive function evaluations to an FPGA or ASIC.

Bibliography

[1] L. Hoddeson, "The discovery of the point-contact transistor," *Historical Studies in the Physical Sciences*, vol. 12, no. 1, pp. 41–76, Jan. 1981. [Online]. Available: https://doi.org/10.2307/27757489

[2] R. Bassett, *To the Digital Age: Research Labs, Start-up Companies, and the Rise of MOS Technology (Johns Hopkins Studies in the History of Technology)*. The Johns Hopkins University Press, may 2007. [Online]. Available: https://www.xarg.org/ref/a/0801886392/

[3] H. Helvajian, *Small satellites : past, present, and future*. El Segundo, Calif. Reston, Va: Aerospace Press American Institute of Aeronautics and Astronautics, 2008.

[4] J. Guthrie, *How to make a spaceship : a band of renegades, an epic race, and the birth of private space flight*. New York: Penguin Press, 2016.

[5] B. Denby and B. Lucia, "Orbital edge computing: Machine inference in space," *IEEE Computer Architecture Letters*, vol. 18, no. 1, pp. 59–62, jan 2019. [Online]. Available: https://doi.org/10.1109/lca.2019.2907539

[6] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, oct 2016. [Online]. Available: https://doi.org/10.1109/jiot.2016.2579198

[7] G. Bersuker, M. Mason, and K. L. Jones, "Neuromorphic computing: The potential for high-performace processing in space," Nov 2018. [Online]. Available: https://aerospace.org/sites/default/files/2018-11/Bersuker_NeuromorphicComputing_11212018.pdf

[8] A. T. W. L. R. M. R. M. D. P. A. M. Simon Lee, Amy Hutputanasin, "Cal poly 1u to 3u cubesat design specification," Feb 2014. [Online]. Available: https://static1.squarespace.com/static/5418c831e4b0fa4ecac1bacd/t/56e9b62337013b6c063a655a/1458157095454/cds_rev13_final2.pdf

[9] "Cal poly 6u cubesat design specification," June 2018. [Online]. Available: https://static1.squarespace.com/static/5418c831e4b0fa4ecac1bacd/t/5b75dfcd70a6adbee5908fd9/1534451664215/6U_CDS_2018-06-07_rev_1.0.pdf

[10] R. M. D. P. W. Lan, R. Munakata, "Poly picosatellite orbital deployer mk.iii rev. e user guide," March 2014. [Online]. Available: https://static1.squarespace.com/static/5418c831e4b0fa4ecac1bacd/t/5806854d6b8f5b8eb57b83bd/1476822350599/P-POD_MkIIIRevE_UserGuide_CP-PPODUG-1.0-1_Rev1.pdf

[11] T. Prejean, "Nanoracks cubesat deployer (nrcsd) interface definition document (idd)," May 2018. [Online]. Available: https://nanoracks.com/wp-content/uploads/NanoRacks-CubeSat-Deployer-NRCSD-Interface-Definition-Document.pdf

[12] D. Gillies, "Rocket lab rideshare cubesat launch in maxwell," April 2018. [Online]. Available: https://nanoracks.com/wp-content/uploads/NanoRacks-CubeSat-Deployer-NRCSD-Interface-Definition-Document.pdf

[13] B. S. D. S. A. G. C. W. A. B. A. G. Sebastian Sabogal, Patrick Gauvin, "Ssivp: Spacecraft supercomputing experiment for stp-h6." AIAA, 2017.

[14] J. Wertz, *Space mission analysis and design.* Torrance, Calif. Dordrecht Boston: Microcosm Kluwer, 1999.

[15] P. Fortescue, *Spacecraft systems engineering*. Hoboken, N.J: Wiley, 2011.

[16] S. Hirshorn, *NASA Systems Engineering Handbook*. Washington DC: National Aeronautics and Space Administration, 2016.

[17] N. Neel, "Enhancing small satellite based remote sensing capabilities by utilizing advanced simulated data products for cross calibration techniques," Master's thesis, The University of Georgia, The address of the publisher, 5 2019.

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

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

[20] C. Versteeg, "Thermal management and design of high heat small satellite payloads," *32nd Annual AIAA/USU Conference on Small Satellites*, 2018.

[21] C. Adams, "A near real time space based computer vision system for accurate terrain mapping," *32nd Annual AIAA/USU Conference on Small Satellites*, 2018.

[22] (2003) Pc/104-plus specification. [Online]. Available: www.winsystems.com/wp-content/uploads/specs/PC104PlusSpec.pdf

[23] (2017) Jetson tx2 developer kit user guide. [Online]. Available: www.developer.nvidia.com/embedded/jetpack

[24] "Ieee standard for environmental specifications for spaceborne computer modules," *IEEE Std 1156.4-1997*, 1997. [Online]. Available: www.ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=603627&tag=1

[25] "Detailed materials list for thermal/vaccum/space-rating of tx2/tx2i," June 2018. [Online]. Available: https://forums.developer.nvidia.com/t/detailed-materials-list-for-thermal-vaccum-space-rating-of-tx2-tx2i/62250

[26] F. Bruhn, "Introducing radiation tolerant heterogeneous computers for small satellites," *32nd Annual AIAA/USU Conference on Small Satellites*, 2015.

[27] (2016) 6u cubesat design specification. [Online]. Available: http://org.ntnu.no/studsat/docs/proposal_1/A8%20-%20Cubesat%20Design%20Specification.pdf

[28] "Nanoracks cubesat deployer (nrcsd) interface definition document (idd). nr-nrcsd-s0003," 2018. [Online]. Available: www.nanoracks.com

[29] Dunmore aerospace satkit. [Online]. Available: www.dunmore.com/products/satkit-mli.html

[30] G. D. Badhwar, "The radiation environment in low-earth orbit," *Radiation Research*, vol. 148, no. 5, pp. S3–S10, 1997. [Online]. Available: www.jstor.org/stable/3579710

[31] "World radiation data centre," 2018. [Online]. Available: www.re3data.org/repository/r3d100010177

[32] R. H. Maurer, M. E. Fraeman, M. N. Martin, and D. R. Roth, "Harsh environments: Space radiation environment, effects, and mitigation," *Johns Hopkins APL Technical Digest*, vol. 28, no. 1, pp. 17–29, 2008.

[33] E. Wyrwas, "Body of knowledge for graphics processing units (gpus) nepp-bok-2018," 2018. [Online]. Available: www.nasa.gov/sites/default/files/atoms/files/2017-8-1_stip_final-508ed.pdf

[34] R. R. J. Roe, "Standard materials and processes requirements for spacecraft," *Technical Report NASA-STD-6016A*, November 2016.

[35] "The denx u-boot and linux guide (dulg) for canyonlands," 2018. [Online]. Available: www.denx.de/wiki/DULG/Manual

[36] (2014) Igloo2 and smartfusion2 65nm commercial flash fpgas interim summary of radiation test results. [Online]. Available: www.microsemi.com/document-portal/doc_view/134103-igloo2-and-smartfusion2-fpgas-interim-radiation-report

[37] E. Ontiveros, C. Salvaggio, D. Nilosek, N. Raqueño, and J. Faulring, "Evaluation of image collection requirements for 3d reconstruction using phototourism techniques on sparse overhead data," in *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XVIII*, S. S. Shen and P. E. Lewis, Eds. SPIE, May 2012. [Online]. Available: https://doi.org/10.1117/12.919319

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

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

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

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

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

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

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

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

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

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

[48] C. De Boor, "Bicubic spline interpolation," *Journal of mathematics and physics*, vol. 41, no. 1-4, pp. 212–218, 1962.

[49] M. Westoby, J. Brasington, N. Glasser, M. Hambrey, and J. Reynolds, "'structure-from-motion' photogrammetry: A low-cost, effective tool for geoscience applications," *Geomorphology*, vol. 179, pp. 300–314, Dec. 2012. [Online]. Available: https://doi.org/10.1016/j.geomorph.2012.08.021

[50] D. Cotten, T. Jordan, M. Madden, , and B. S., "Structure from motion and 3d reconstruction," *Manuel of Remote Sensing*, pp. 160–166, Jan. 2017.

[51] C. Wu, "Towards linear-time incremental structure from motion," in *2013 International Conference on 3D Vision*. IEEE, Jun. 2013. [Online]. Available: https://doi.org/10.1109/3dv.2013.25

[52] J. D. Matías, J. D. Sanjosé, G. López-Nicolás, C. Sagüés, and J. Guerrero, "Photogrammetric methodology for the production of geomorphologic maps: Application to the veleta rock glacier (sierra nevada, granada, spain)," *Remote Sensing*, vol. 1, no. 4, pp. 829–841, Oct. 2009. [Online]. Available: https://doi.org/10.3390/rs1040829

[53] (2016) Crew earth observations video page. [Online]. Available: https://eol.jsc.nasa.gov/beyondthephotography/crewearthobservationsvideos/

[54] D. A. Bennetts and M. Ouldridge, "An observational study of the anvil of a winter maritime cumulonimbus cloud," *Quarterly Journal of the Royal Meteorological Society*, vol. 110, no. 463, pp. 85–103, Jan. 1984. [Online]. Available: https://doi.org/10.1002/qj.49711046308

[55] T. W. Krauss, A. A. Sinkevich, N. E. Veremey, Y. A. Dovgalyuk, and V. D. Stepanenko, "Study of the development of an extremely high cumulonimbus cloud (andhra pradesh, india, september 28, 2004)," *Russian Meteorology and Hydrology*, vol. 32, no. 1, pp. 19–27, Jan. 2007. [Online]. Available: https://doi.org/10.3103/s1068373907010037

[56] (2016) Geforce gtx 980 specifications. [Online]. Available: http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-980/specifications

[57] "Nasa lp daac, 2015, aster level 1 precision terrain corrected registered at-sensor radiance. version 3." [Online]. Available: https://lpdaac.usgs.gov/dataset_discovery/aster

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

[59] R. H. T. S. K. David A. Vallado, Paul Crawford, "Revisiting spacetrack report #3: Rev 2," *Astrodynamics Specialist Conference*, 2006.