Engineering a Reliable Software Stack for Computer Vision on Small Satellites

by

Eric Nathan Miller

(Under the Direction of Deepak Mishra)

Abstract

In the majority of satellite missions, data is collected onboard and processed on the ground. This bottlenecks the amount of scientific data that can be downlinked without compression. However, as computing systems have progressed, missions have begun using high-performance computers onboard for *in-situ* computation, reducing the need to downlink raw data. The Multiview Onboard Computational Imager (MOCI) will house its computer vision pipeline onboard an NVIDIA Jetson TX2i GPU in order to create digital elevation models from and recognize objects within high-resolution imagery, while only needing to downlink the final data products. For reliable mission success in a constrained satellite body and unforgiving environment in space, measures are taken to ensure both that the underlying system architecture is fault-tolerant in Low Earth Orbit and that the higher level algorithms can operate within architectural limits and agnostic of sensor noise.

Index words: [onboard processing, gpu, edge computing, radiation mitigation, computer vision, structure from motion]

Engineering a Reliable Software Stack for Computer Vision on Small Satellites

by

Eric Nathan Miller

B.S. Computer Science, University of Georgia, 2022
B.S. Mathematics, University of Georgia, 2022

A Thesis Submitted to the Graduate Faculty of the
University of Georgia in Partial Fulfillment of the Requirements for the Degree.

Master of Science

Athens, Georgia

2022

Engineering a Reliable Software Stack for Computer Vision on Small Satellites

by

Eric Nathan Miller

Major Professor: Deepak Mishra

Committee: In Kee Kim
Frederick Maier

Electronic Version Approved:

Ron Walcott
Vice Provost for Graduate Education and Dean of the Graduate School
The University of Georgia
December 2022

Acknowledgments

I would like to thank my major advisor and Small Satellite Lab principal investigator Dr. Deepak Mishra, not only for supporting my own research but also for fostering the lab's success as a whole. I would also like to thank my committee members Dr. Frederick Maier and Dr. In Kee Kim in guiding me through the MSAI program and specific research questions.

I would also like to thank the lab alumni who have mentored me and involved me in research early in undergrad: James Roach, Godfrey Hendrix, Alex Lin, and especially Caleb Adams and Jackson Parker, who have continued to advise me and the Payload Team. Thank you to current lab members Cameron Bonesteel, Ryan Hughes, Ellemieke Van Kints, Omer Cinar, and lab manager Sydney Whilden, who have all contributed to the results presented here and will continue to guide MOCI to success through launch.

I would also like to thank those external to UGA who have helped set up the radiation tests and contributed to the results. Thank you to Chris Heistand, who co-developed the SOL operating system while and after working at APL, and Lance Simms, who made major contributions to the codebase later on. Thank you to Sarah Katz from APL and Edward Wyrwas from NASA Goddard for helping us prepare for radiation tests and analyze results. Thank you to TRIUMF, particularly Camille Belanger-Champagne and Mike Trinczek, for allowing us to book your facilities for testing and making the process seamless.

Finally, I would like to thank my family for their tremendous support throughout this process.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

On February 18, 2021, the Mars 2020 mission's Perseverance rover touched down on Martian terrain, equipped with a state-of-the-art imaging system. In supplement to the cameras used to document the rover's descent and its collection of samples, the vehicle houses two cameras for navigation and four for hazard avoidance. Each camera is 20 megapixels, requiring buffered image transmission into the rover's flight software, as well as lossy compression for downlinking. However, when coupled with the rover's enhanced AutoNav features since NASA's MSL and MER missions, these imaging subsystems enable semi-autonomous driving and arm movements, reducing both the amount of computation required by and the time of transmission to human engineers [1, 2].

Mars 2020 highlights two key points regarding the current and future states of scientific missions in space. Primarily, due to advances in sensor technology, as well as larger mission scopes, extraterrestrial endeavors are becoming highly data-intensive, with image and/or raw data transfers becoming a bottleneck to mission progress. Moreover, Perseverance among its predecessors illustrates the ever-expanding breadth of humanity's scientific exploration, requiring vehicles to reach further into space, where response times to and from Earth become slow and unreliable.

These two factors motivate projects to have a higher capacity for on-board data processing and autonomy. The feasibility of various imminent projects, such as a human mission to Mars, will require a decreased reliance on spacecraft-to-Earth communication. Accordingly, onboard applications of Artifi-

cial Intelligence in data analysis, distributed systems, swarm intelligence, and fault tolerance can permit spacecrafts to autonomously make necessary responses to their environments [3].

1.1 The Advent of CubeSats

As Poghosyan and Golkar (2017) detail, another principal factor in the advancement of space technologies has been the industry's expansion into the private sector. Since 1957, conventional satellite missions have been restricted to government-funded agencies, as the sheer size of the satellites (meant to house multiple instruments to optimize cost) required both a large team and budget. However, with the technical challenges of engineering large spacecrafts, as well as the availability and reduced size of commercial-off-the-shelf (COTS) products in recent decades, the space industry has inclined towards smaller satellites.

With small satellites' trending popularity, Stanford and California Polytechnic standardized the CubeSat as a composition of 1U volumetric units, each $10 \times 10 \times 10$ cm^3 and up to 1.33 kg in mass. Since their inception in 1999, CubeSats have ranged in application from the original proof-of-concept and educational technology demonstrations to full-scale scientific discovery missions. Their ease of development and low cost have enabled smaller countries and even educational institutions to devise and launch spacecrafts with state-of-the-art COTS technology. In particular, CubeSat missions have become a low-risk avenue for developers to test incremental updates in scientific payload technology [4].

1.2 GPUs and their Applicability in Spacecrafts

Vuduc and Choi (2013) discuss the history of graphics processing units (GPUs) from their earliest applications, which, as their name suggests, was for 3D graphics modeling, particularly in gaming systems. As GPUs began to greatly outperform CPUs (by factors of 10–100) in parallel computational tasks required for graphics rendering, their capabilities captured the interest of developers in other fields. Due to the difficulty of developing GPUs for tasks outside of the graphics realm, NVIDIA developed the Compute Unified Device Architecture (CUDA) in the early 2000s as a high-level platform for developers to interface with GPUs. This gave rise to a new software-level paradigm, the general-purpose GPU (GPGPU),

which allows programmers to utilize GPU hardware to explicitly parallelize tasks of their choosing. In particular, GPU hardware has proven useful for single instruction multiple data (SIMD) designs, where one operation can be designed to perform simultaneously over a large set of data values. CUDA has now grown to include a user base in the millions and is largely used in GPU-based research [5].

GPUs are commonly utilized for AI applications, particularly in deep learning and computer vision, where intensive linear algebra computations that would heavily bottleneck CPU programs can be refactored into SIMD designs and accelerated via GPGPU programming. Hence, for spacecraft missions implementing such AI applications, such as those involving large volumes of image data, GPUs are often used for on-ground processing. However, due to the constraints of data transmission discussed earlier, teams have recently become motivated to incorporate GPUs onboard. GPUs' ease of programming and commercial availability make them suitable low-cost candidates for such missions, but the available hardware is primarily designed for terrestrial operation—that is, agnostic of the radiation environment that exists outside of Earth's atmosphere [6].

1.2.1 CUDA-based Architecture

Since the research presented here is conducted on NVIDIA GPUs, an understanding of CUDA-based architectures is beneficial. The GPU consists of a cluster of streaming multiprocessors (SMs), each containing a collection of cores to which threads can be assigned. Blocks of threads are assigned by the block scheduler to a specific SM, and then the warp scheduler delegates groups of threads within the block to specific cores. Finally, the dispatch unit relays the instructions to each core. Memory in L2 cache is shared among all SMs on the GPU, whereas each SM has its own L1 cache shared among its cores. Each SM also has one register file, but register scope is restricted to individual cores/threads [7].

1.3 Radiation-Induced Effects

The majority of satellites operate in Low Earth orbit (LEO), where three main sources of radiation are of relevance:

1. galactic cosmic radiation (GCR)
2. trapped radiation belts
3. solar energetic particles (SEPs)

GCR includes particles of a wide range of energies originating from beyond the solar system, and whose flux inversely correlates with solar energy. Trapped radiation belts have minor effects in low altitudes, except at high inclination (near the poles). SEPs are directly correlated with the solar cycle, as the events originate from the sun (*e.g.*, solar flares) [8].

Radiation in such an environment has two forms of effects on electronic devices: total ionizing dose (TID) and single-even effects (SEEs). The former refers to a build up of trapped charges that, over time, lead to functional hardware shifts. SEEs may take the form of, among others, single-event upsets (SEUs) such as bit-flips that cause corruption or single-event latch-ups (SELs) that lead to circuitry-level failures. Radiation mitigation measures, on a physical and hardware level, include device shielding, hardware-level redundancy, and error detection and correction (EDAC). The latter two tend to carry budgetary and performance-related overheads, generally creating an inverse relationship between computational efficiency and reliability [6].

1.4 The Multiview Onboard Computational Imager (MOCI)

The Multiview Onboard Computational Imager (MOCI) is a 6U CubeSat in development by the University of Georgia (UGA) Small Satellite Research Lab (SSRL). The aim of MOCI is to perform onboard computer vision, namely structure from motion (SfM) and object recognition, to avoid nominally downlinking full-sized images. The accuracy of the in-house SfM pipeline, which produces digital elevation models (DEMs) from multiple images of a ground target location, is the primary mission objective. Detection of objects such as ships and planes using a third-party model trained in-house constitutes a secondary objective. Both algorithms are described in greater detail in chapter 4.

Figure 1.1 shows simplified CAD diagrams of MOCI with and without its side panel. The CubeSat's structure essentially consists of two side-by-side 3U modules. The left side, as shown, houses MOCI's

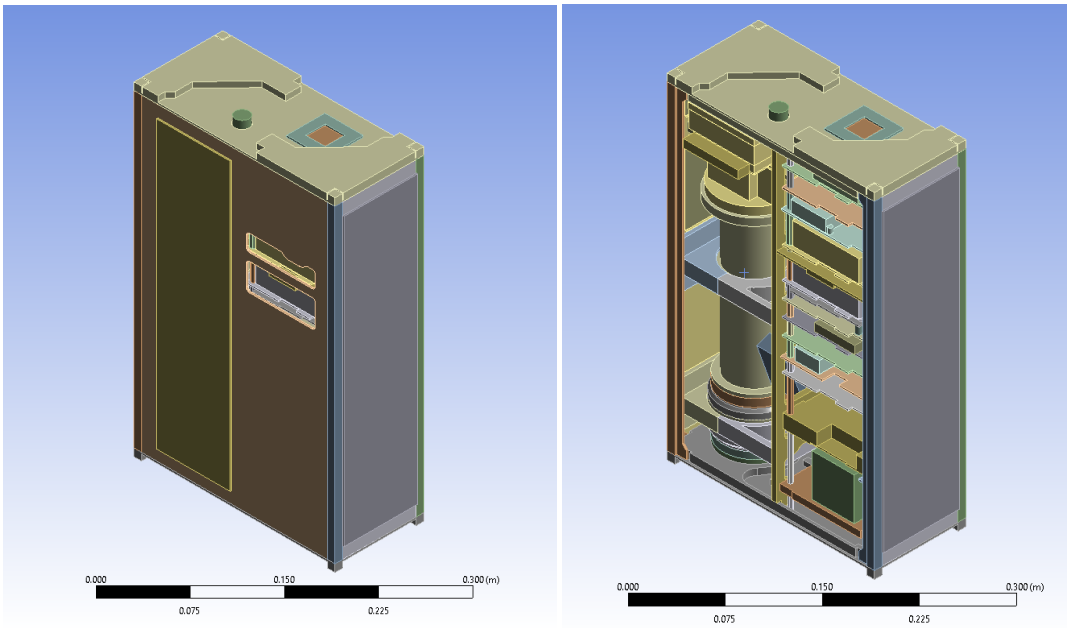


Figure 1.1: MOCI CAD diagrams (simplified).

optical train, containing two camera sensors. The other side consists of the satellite's avionics stack, containing the following components, along with custom interface boards:

- **Electrical Power System (EPS)**
- **Onboard Computer (OBC):** Command and data handling.
- **Payload Computer:** Data (image) processing.
- **Attitude Determination and Control System:** Actuators and sensors for pointing.
- **UHF Transceiver:** Uplink and downlink.

The payload subsystem, the subject of this research, consists of both the payload computer—an NVIDIA Jetson TX2i—and the optical train.

1.4.1 Optical Train

The optical train contains two sensors. The 4k, monochrome imager is intended to be employed for SfM applications, whereas 2k lower-resolution color imager is intended for object detection. At an assumed 400km altitude, MOCI's Imperx monochrome imager has a ground sample distance (GSD) of 6.67m whereas the Blackfly color imager has a GSD of 8.68m.

1.4.2 The NVIDIA Jetson TX2i

The NVIDIA Jetson TX2i is a state-of-the-art System-on-Module (SoM) whose System-on-Chip (SoC) integrates a 64-bit ARMv8 multi-processor (Dual-Core Denver 2 and Quad-Core Cortex-A57) and a 256-core CUDA-compatible NVIDIA Pascal GPU. The module also contains a memory controller providing error-correcting code (ECC) on LPDDR4 SDRAM, as well as an eMMC flash memory card. A block diagram of the module is shown in figure 1.2 [9].

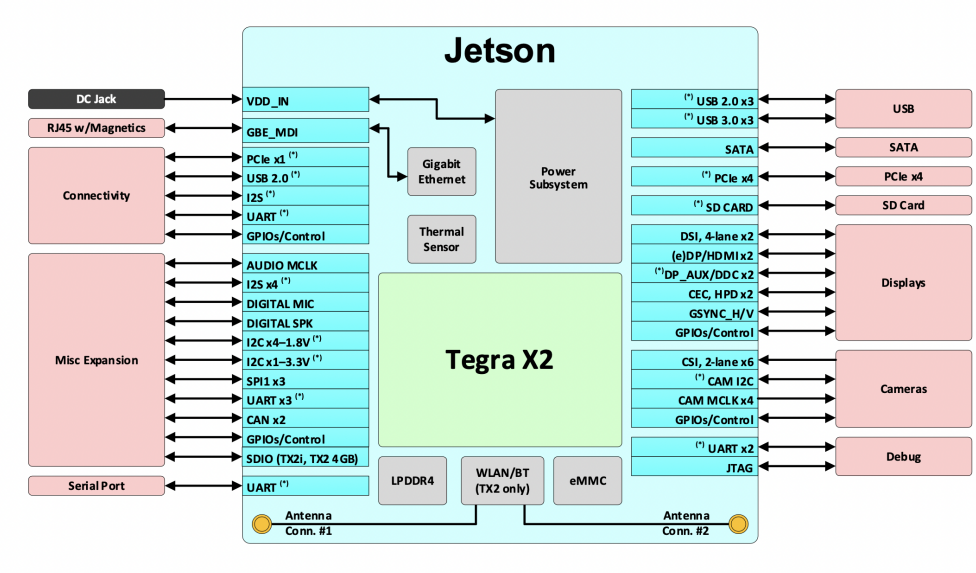


Figure 1.2: Block diagram of the TX2i module.

Jetson series GPUs have notably been used in various embedded computer vision and robotics tasks, due to their small form factor and low power consumption, the latter of which is offered by a shared memory space between the GPU/CPU. Although virtually no commercial GPUs are intended for use in

space—and such use cases void their warranties—the TX2i contains upgrades over its TX2 counterpart that permit its use in industrial grade environments, rendering it a relatively suitable candidate for on-board processing [10].

On MOCI, the custom Core GPU Interface (CORGI) will interface the TX2i to the OBC, with UART being the main line of communication. In this paradigm, the OBC will act as controller and TX2i as peripheral, responding to any commands related to data processing or transferring as they arrive. In the case that the TX2i fails during the mission, MOCI's Peripheral USB Interface (PUSBI) will reroute the optical system to the OBC, such that images can still be taken onboard and processed on the ground. This potential stage of the mission would greatly reduce the rate of scans, as downlinks would take drastically longer times without onboard processing.

With recent interest in flying a Jetson GPU, various radiation tests have been presented and published on the TX2i and related modules over the past two years. Slater et. al. tested the Jetson Nano under various TIDs (around 20 krad) and concluded that, with a $1/10$ inch of aluminum shielding, the SoM could be expected to last through at least a 1.5–2 year mission [6].

Heistand et. al. from the Johns Hopkins Applied Physics Laboratory (APL) conducted both TID and SEE tests on the TX2i module. The former test revealed that the TX2i could handle moderate radiation dosage (23 krad) before failure while powered on, a notably higher dose when off (45 krad), and lower doses (9.7–20 krad) during soft and hard power cycles—the latter of which had extremely high susceptibility. Proton SEEs caused multiple forced reboots before eventually resulting in board failure after an average of four runs. From the collected data, the APL team predicts a 57% and 73% survivability rate in LEO at solar minimum and maximum, respectively, without shielding or mitigation techniques. Additionally, while the majority of logged software events were CPU and/or memory errors, the majority of TX2i failures—including all permanent failures—were linked to flash errors (read/write) [11].

In orbit, while many of these issues can be mitigated through physical shielding, the TX2i still remains a weak point of future missions. This calls for software-level tolerance as a final line of defense against SEEs.

1.5 Research Objective

The primary goals of this research are aimed at improving the robustness of MOCI's payload to ensure operability, reliability, and survivability in orbit. However, the methods and results presented here extend outside the scope of MOCI, with general applications to radiation mitigation, computer vision, and onboard processing. There are two main objectives:

System Reliability: A custom operating system is employed to provide near real-time operation and software-level radiation mitigation on the payload computer.

Computer Vision Optimization: A handful of optimizations are evaluated and employed to fit MOCI's computer vision pipelines into the runtime, power, memory, and pointing constraints of the mission.

The former objective is discussed in chapter 3, where an operating system design is explained and evaluated. Proton-beam irradiation tests were performed on the TX2i and, along with previous tests performed by other entities, are evaluated in order to justify the radiation mitigation strategy. In opposition to previous radiation tests on the TX2i, irradiation collimated to the SoC itself—without spallation into peripherals—showed considerably low estimated error and failure rates in LEO, with zero permanent device failures. Hence, radiation mitigation for the TX2i should continue to be tailored towards mitigating effects on peripherals such as flash. Software mitigation can increase reliability, but custom hardware (circuitry) solutions should be considered for future, longer-lifetime missions.

Onboard computer vision algorithms are discussed in chapter 4, where solutions to in-orbit constraints are presented and evaluated, particularly in regard to SfM. General pipeline optimizations reduce runtime and power usage of an existing pipeline (formulated by [12]) by approximately a factor of four. Custom pose estimation techniques suited towards MOCI's pointing constraints allow for orientation computation within 0.0001 degrees accuracy and offloads much of the computation that is generally accounted for by bundle adjustment, the final stage in traditional pipelines.

Chapter 2

Related Works

The increase in data-intensive missions and onboard processing has motivated a variety of novel algorithms and optimizations to reduce the associated processing and downlinking bottlenecks. In conjunction, state-of-the-art software and hardware architectures have been theorized and designed in order to give rise to such computations, particularly through radiation mitigation techniques.

Many of these concepts have been considered and built upon in the design presented here. Others, although not feasible on MOCI due to various mission constraints, are also acknowledged.

2.1 Embedded Architectures and Radiation Mitigation

George and Wilson (2018) conducted a survey of onboard computing technologies that tackle the problem of data processing within the constrained and harsh environments of small satellites, particularly by employing reconfigurable and hybrid computing techniques. Field-programmable gate arrays (FPGAs) are a common option for a software-reconfigurable architecture, allowing for task-specific parallelization with minimal power usage. However, FPGAs are especially prone to SEU damage, as their configuration memory is a single point of failure. Hybrid architectures, including SoCs, may include some combination of CPUs, GPUs, and FPGAs to optimize performance, or combinations of high-performance COTS processors and reliable rad-hard processors for fault-tolerance. In addition to hardware-level fault tolerance, including rad-hardened circuitry and redundancy, software-level mitigation provides low-cost solutions

to radiation effects. Such techniques include error encoding, repetition of processes, checkpointing, and exception handling [13].

NASA provides recommendations for an iterative satellite design process with regard to radiation effects. This commences with modeling and evaluating the radiation environment (based on inclination, altitude, shielding, the solar cycle, etc.) in order to create high-level mission radiation requirements. Then, individual devices may be evaluated through either archival tests on similar parts or novel radiation tests, and modeling tools are used to characterize circuit degradation and SEE rates in the satellite's environment. Finally, results are brought into the design and engineering process in order to either replace unacceptable parts or perform radiation mitigation techniques [14].

In MOCI's case, the TX2i has been identified as a semi-critical component. To aid in preventing immediate failure, shielding and software mitigation techniques are employed, the latter of which is discussed in the following chapter. However, the satellite's Peripheral USB Interface (PUSBI) allows its optical payload to be rerouted directly to the OBC to extend the mission's lifetime past the TX2i's eventual failure.

Sample architectures and mitigation techniques are discussed from low to high level. While lower-level hardware designs are generally ideal for radiation hardness, higher-level software solutions are often more efficient and portable with lower costs.

2.1.1 Hardware

Given the topic of this research, the primary hardware paradigms discussed here are GPU-centric architectures. As discussed in the previous chapter, standard GPU designs are highly susceptible to radiation-induced effects. This primarily includes SEUs that can corrupt data and in many cases, due to the complex architecture of the GPU and scheduler, result in single event functional interrupts (SEFIs) [15]. Such susceptibilities tend to push spacecraft engineers towards heterogeneous architectures that allow for the reliability of more tolerant devices coupled with the performance gains of GPUs for tasks such as image processing.

Adams *et al.* (2019) theorized a future iteration upon MOCI's Core GPU Interface (CORGI) called the Accelerated Flight Computer (AFC). Like CORGI, the AFC is to contain a hybrid architecture including a SmartFusion2 FPGA SoC as a trusted control node and a payload including one (or more) Jetson TX2i modules (as shown in figure 2.1) all in a modular design compatible with standard CubeSat form factors. Improvements upon CORGI include an ethernet line for bulk data transfer and shared radiation-hardened NOR flash between the devices, as well as radiation shielding. Additionally, software level mitigation may be applied to the TX2i, and the SmartFusion2—which has been shown to have relatively high radiation tolerance—may as a watchdog over the GPU system [16].

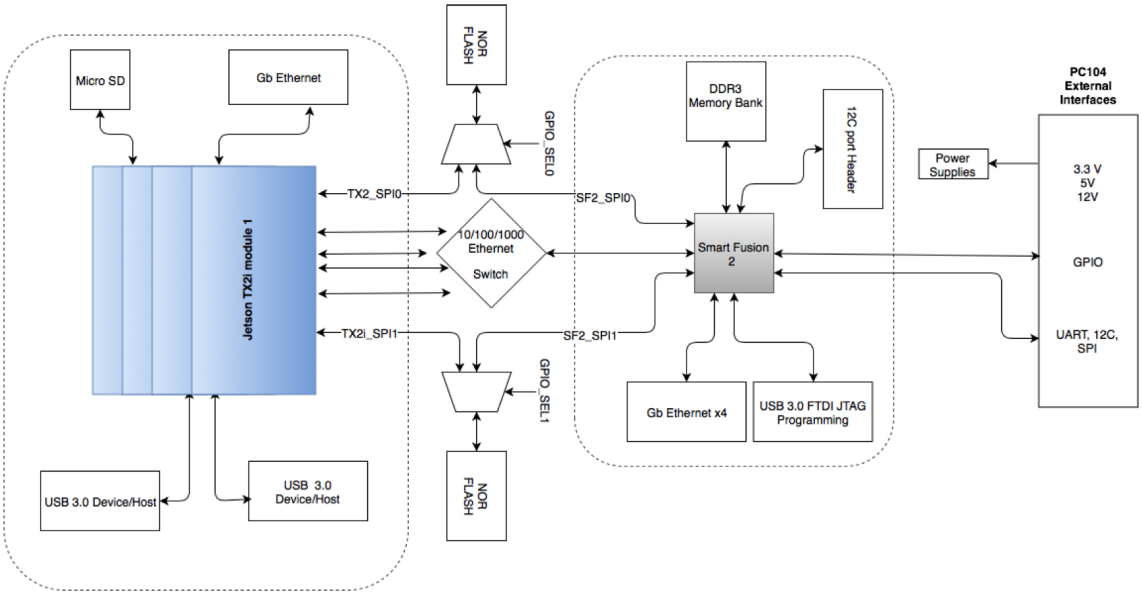


Figure 2.1: Hardware architecture of the Accelerated Flight Computer [16].

In production, the Unibap e2160 has a similar heterogeneous architecture, shown in figure 2.2. Like the AFC, the e2160 houses a SmartFusion2 FPGA SoC as a control node for a GPU SoC payload. However, due to higher radiation tolerance, the module opts for an AMD rather than an NVIDIA chip. Recent increases in performance, as well as the development of the ROCm software stack (which allows CUDA code to be compiled for AMD architectures), render AMD GPUs viable options for payload computers going forward. The e2160 additionally supports the attachment of additional accelerators, as shown on the left side of the architecture diagram [17].

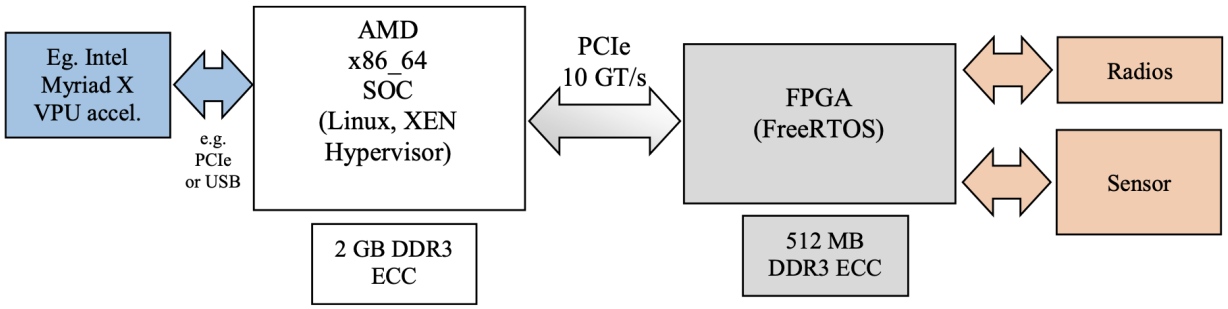


Figure 2.2: Architecture of the Unibap e2160 [17].

2.1.2 Middleware

In his 2006 dissertation, Troxel introduced the Comprehensive Approach to Reconfigurable Management Architecture (CARMA), a middleware designed for heterogeneous and reconfigurable architectures for space applications, such as those discussed above. CARMA includes a Fault-Tolerance Manager (FTM) running on a radiation hardened processor to monitor each component and attempt recovery when applicable [18].

Troxel's dissertation became the basis for Troxel Aerospace, whose SEE Mitigation Middleware (SMM) has successfully been deployed to the Unibap e2160. This middleware primarily operates on top of the operating system, but has extensions within and below the OS to provide system-level fault tolerance. In tests, Troxel's SMM was shown to reduce SEFI rates by a factor of 720 [17].

2.1.3 Software

Software mitigation techniques range in application, from process duplication to redundancy to error encoding methods. While these techniques are generally not as effective as hardware techniques, as lower-level failures would render software mitigation futile, they carry lower costs and design complexities.

There are a number of ways SEUs can impact GPU performance. While an upset in a single core only impacts one thread, corruption to L1 or L2 cache or scheduler errors may have more drastic effects. Error

correcting code (ECC) is common in protecting these memory spaces, but scheduler and logic errors are left unprotected from SEUs, creating the potential for incorrect data products or control flow errors leading to kernel panics. In order to mitigate these, two strategies include algorithm-based fault tolerance (ABFT) and duplication with comparison (DWT). The former is preferred for performance purposes, but since it is algorithm-specific, it is not applicable in all cases [7].

ABFT strategies generally involve input encoding along with potential algorithmic changes in such a manner that allows the validity of the output to be checked. Due to high failure rates of Fast Fourier Transform (FFT) algorithms under neutron radiation, Pilla *et al.* devised a custom ABFT technique to make their FFT algorithm fault-tolerant. This enhancement incurred a significantly smaller overhead than alternative ECC-based methods [19].

Garrett (2021) employed a novel technique called Resilient TensorFlow (RTF), a fault-tolerant iteration upon the TensorFlow backend. Since TensorFlow is the most commonly used machine learning framework, the implications of Garrett's improvements span the vast scope of machine learning algorithms, as opposed to ABFT approaches that are algorithm-specific. RTF works by replacing vulnerable TensorFlow operations with one of two alternative kernels that leverage the GPU's redundant architecture for fault tolerance. One of these techniques, the distributed thread blocks approach, allocates operations across the device to limit dependency on shared resources and results in an 80% decrease of highly vulnerable operations with minimal performance overhead [20].

2.2 Structure from Motion

2.2.1 History

The techniques used for structure from motion (SfM) stem from earlier developments in the broader field of photogrammetry, which provides the mathematical basis for perspective and camera geometry. In the late 1950s, nearly two decades before SfM's official inception, Thompson formulated an iterative method for computing the relative orientation of two cameras from five point correspondences to find the solution to five third-order equations [21].

Marr and Poggio (1971) formulated a method of 3D construction based on stereo disparity. Similar to the concept of human depth perception, this technique requires two images with known relative orientation as well as known point correspondences in order to discern depth from relative point translation differences [22]. Marr additionally introduced the *primal sketch*, which uses grey level intensity changes to produce localizations and descriptions of edges, as a method for forming point correspondences [23].

In 1979, Ullman formally introduced the early concept of SfM. A motivating phenomenon was the two cylinder experiment, pictured in figure 2.3, where 100 points on concentric cylinders were displayed on a screen as they rotated along a single vertical axis. Since no outlines of the cylinders were present, still images appeared to contain random noise, but the human visual system was able to perceive the cylinder's structure from the moving video.

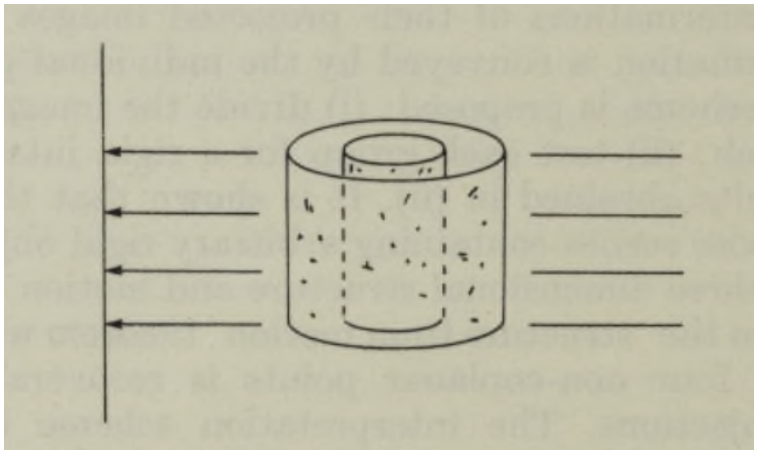


Figure 2.3: Ullman's two cylinder demonstration [24].

Under the assumption of rigidity, without which there could be an infinite number of motion paths, Ullman proved that there is a unique interpretation of structure given 4 non-coplanar points on three orthographic projections. Moreover, he extended this result to perspective scenes using a polar-parallel approach, which combines perspective with orthographic geometry. Ullman noticed that each neighborhood of nearby points in the scene could be approximated by a local orthographic approach, with a single axis of projection. This can be completed for multiple neighborhoods across the scene, but by using different axes per neighborhood based on perspective geometry—that is, the ray from the camera center

through the center of the given neighborhood. This approach is less complex and less prone to noise than a purely perspective approach [24].

The secondary goal of SfM, aside from reconstruction of three-dimensional points, is to recover camera geometry. In 1981, Longuet-Higgins provided a major contribution to such techniques by showing that eight point correspondences between two perspective images can simplify the problem of relative camera geometry to a system of linear equations [25]. This iteration upon Thompson's 5-point correspondence technique allows for more feasible computation in SfM pipelines and continues to be used in the modern day.

An integral aspect of most structure from motion implementations is bundle adjustment, another photogrammetric concept where both 3D projections and camera structure are refined to minimize a cost function. The name lends itself to the concept that an individual point is projected to an image by the three-dimensional ray to its camera center, forming a bundle of such rays for each point. This problem is generally formulated as a least squares optimization. Despite the high dimensionality of the parameter space, bundle adjustment is not an overly expensive operation due to the problem's inherent sparsity [26].

2.2.2 State of the Art

In 2018, Bianco *et al.* performed an evaluation of the most popular commercial SfM pipelines. Most pipelines consist of a similar sequence of operations, as follows:

- Feature extraction
- Feature matching
- Geometric verification
- Image registration
- Triangulation
- Bundle adjustment

Bianco's evaluation criteria include camera pose accuracy (position and orientation), and comparison to ground truth (after alignment and iterative registration). Some pipelines also enable exports to MVS pipelines, which produce denser point clouds that can be evaluated similarly. Bianco tested six pipelines—both SfM and MVS versions—against Blender-simulated data, and COLMAP had the best overall results. Another important lesson was that no pipelines could produce adequate reconstructions of a simulated fire hydrant due to its uniformity [27]. This shows how essential a role the first stage of the pipeline, feature extraction, plays in producing a feasible reconstruction.

2.2.3 Application to Remote Sensing

SfM-based reconstructions in the 1990s constituted a major breakthrough in digital elevation model (DEM) generation, which became feasible on low-end computers and without specialized sensors or knowledge, as SfM software became commercially available. SfM's speed compared to earlier photogrammetric methods, and even more so compared to on-site tacheometers, also allows for the automated and dense monitoring of morphological changes over time [28].

The most notable space-based application of SfM uses imagery from NASA's Terra satellite. The mission includes the Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER), which houses two visible and near-infrared sensors for topographic mapping, as shown in the bottom-right side of figure 2.4 [29].

These two sensors include band 3N (nadir-viewing) and 3B (backward-viewing), at an angle of 27.7 degrees off-nadir. As depicted in figure 2.5, the combination of sensors allows for on-track stereo imaging with a 0.6 base-height ratio without slewing. Imaging targets on-track, rather than re-imaging over multiple passes, increases the likelihood of cloud-free image pairs. Processing facilities with Japan and the United States produce DEMs using commercial SfM software with the goal of producing a single, global DEM. This model provides supplementary topographical data to that supplied by previous approaches, such as the Shuttle Radar Topography Mission (STRM). Despite being more accurate, interferometric techniques require more complex sensors, such as synthetic aperture radars (SAR), and processing that is prone to coordinate localization errors [30].

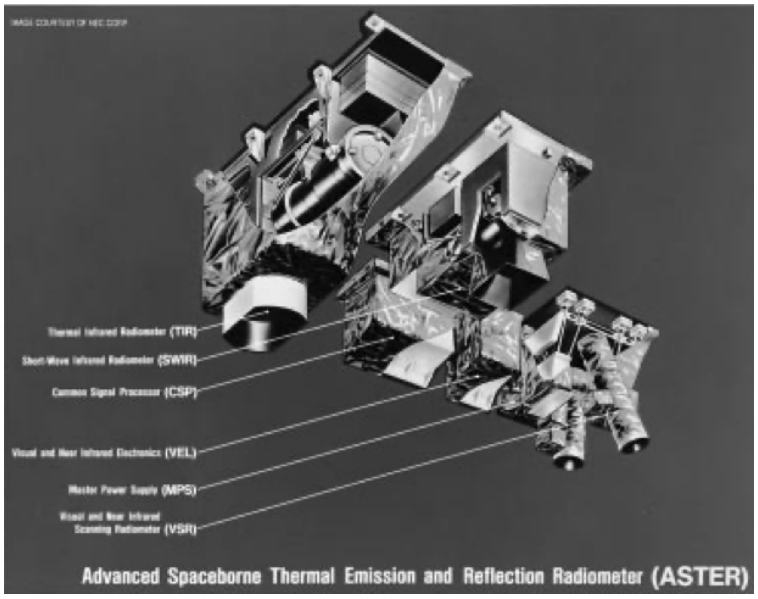


Figure 2.4: ASTER's onboard sensors [29].

2.2.4 Modern Techniques

Simulatenous Localization and Mapping (SLAM) uses similar techniques to SfM in the field of robotics with the primary goal of localizing the agent within the structure during its reconstruction [31].

More recently, deep learning approaches—most notably neural radiance fields NeRF—have shown tremendous improvements upon classical reconstruction models. NeRF utilizes multi-layer perceptrons to parametrize a scene's emitted radiance as a continuous function of viewing location and orientation [32].

2.3 Onboard Processing

The architectural improvements for space-based processing discussed earlier in this chapter have allowed recent missions to perform computer vision and autonomous processing techniques *in-situ*. Some notable early examples, particularly in the small satellite realm, are discussed here.

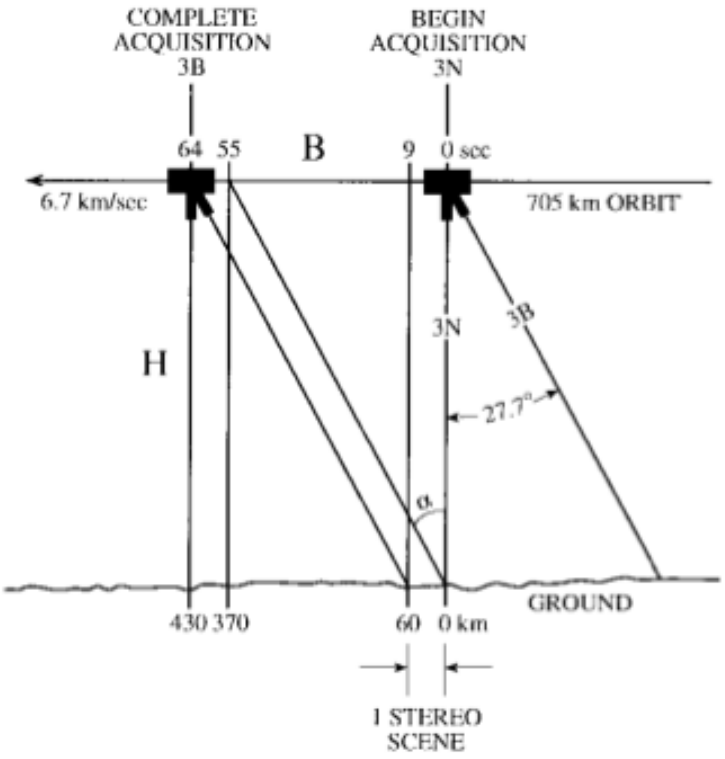


Figure 2.5: Along-track stereo using ASTER's 3N and 3B bands [30].

Functioning from 2013 to 2015, the Intelligent Payload Experiment (IPEX) was a 1U CubeSat that used both AI image processing techniques (random forest classification) and an AI-based scheduler to determine allocate onboard resources autonomously [33]. Multiple recent small satellite ship-detection algorithms employ YOLOv4 and similar CNN-based object detection techniques onboard [34, 35]. Adjacent to the satellite realm, at the cross-section of computer vision and autonomous processing, Choi *et al*. (2022) proposed the use of YOLOv4 for automated docking of unmanned aerial vehicles (UAVs) [36].

Chapter 3

Embedded Software and Radiation Mitigation

3.1 Operating System Design

Out of the box, the Jetson TX2i runs Linux for Tegra (L4T), a fork of the commercial Linux distribution Ubuntu 16.04. Space Operating Linux (SOL) is an open-source, space-faring alternative operating system to L4T to run on future Jetson-bearing missions with a primary focus on TX2i missions. SOL will first fly on the University of Georgia Small Satellite Research Laboratory's (SSRL) Multiview Onboard Computational Imager (MOCI) mission. The SSRL intends for MOCI to be a proof-of-concept for the feasibility of GPU-based AI and computer vision in space, and the mission is particularly motivated by the reduction of data downlink that is afforded by onboard data processing. Within the scope of MOCI, the TX2i is to act as a payload computer that is commanded by a separate onboard computer (OBC), which is responsible for the satellite's command and data handling. The mission will provide integral data on GPU performance in the conditions of LEO and an open-source payload software suite, both of which may be used in the payload design of future, larger-scope spacecraft missions.

The primary goal of SOL is to increase the payload's reliability in orbit by providing software-level radiation mitigation and real-time scheduling, while also maintaining the functionality necessary for

conducting AI-based missions. In particular, SOL make efforts to decrease reliance on flash eMMC memory, which has been shown in radiation tests to be the most vulnerable.

3.1.1 Yocto and OS Minimization

SOL is primarily developed under the Yocto framework, an open-source project used to construct custom embedded Linux distributions from the ground up. The fundamental units for Yocto development are BitBake recipes, each of which provides instructions on retrieving, patching, building, and installing a particular software package for the target device [37]. Related recipes are grouped into layers, such as `meta-tegra`, which contains L4T-based software recipes intended for Jetson devices.

The `meta-sol` layer is developed on top of `meta-tegra` to provide SOL's custom software packages and patches. By doing so, SOL contains only the Tegra software packages that are essential in flight, as well as any necessary modifications to the existing software. Additionally, many existing Unix utilities are replaced by their alternatives in BusyBox, which provides lightweight implementations of such commands. These measures inherently minimize the size of the operating system and file system, hence reducing its usage of flash memory space.

3.1.2 Bootloader Redundancy

While a major goal of SOL is to reduce flash memory usage, the eMMC card is the device's only form of non-volatile memory and hence must be used for any permanent storage. Therefore, redundancy measures are necessary to ensure reliability of essential sectors of flash storage, particularly from corruption caused by accumulated SEEs such as bit-flips. At the most granular level, NVIDIA provides A/B redundancy, where all partitions are duplicated so that, on-boot, if the "A" side fails, the "B" side can be loaded instead.

SOL seeks to emulate triple modular redundancy (TMR) on the single eMMC card for the kernel and root file system. This method was originally proposed by Adams et. al. [16] but was since expanded to include redundancy measures beyond the kernel, motivated by the fact that corruption to the file system itself could hinder the kernel image from being accessed altogether. To achieve this, essential files for boot that are conventionally stored in the root file system—the kernel, device tree, and initial RAM disk image,

as well as a tarball of the root file system itself—are directly written as binary large objects (BLOBs) into specified points of the main file partition, which is then triplicated. Each BLOB is stored along with an associated MD5 checksum to check for corruption easily, and there is a fifth, 60-byte `info` BLOB at the start of the partition that lists the exact file sizes of the following four. An example configuration is outlined in table 3.1, although exact size allocations are dependent on the build and determined within BitBake configurations. Aside from essential software correction, this partition is intended to be updated irregularly, as any temporary data or configuration files needed across boots shall be written to other data partitions.

Table 3.1: Example layout of one of three identical flash partitions for TMR.

File	Offset (# Blocks)	Max Size (# Blocks)
info	0	1
info (hash)	1	1
kernel image	2	90,000
kernel image (hash)	90,002	1
device tree BLOB	90,003	2,000
device tree BLOB (hash)	92,003	1
initial ram disk	92,004	5,000
initial ram disk (hash)	97,004	1
root file system TAR	97,005	1,500,000
root file system TAR (hash)	1,597,005	1

The final stage bootloader for NVIDIA Jetson devices, U-Boot, is responsible for loading the kernel and booting into the initial RAM disk. U-Boot is conveniently open-source[38], and patches can be applied within the Yocto source tree. Hence, SOL is able to apply custom TMR operations on-boot beginning at this stage of the boot process, with a patch that makes use of the custom partition layout. To load each BLOB—aside from the root file system, which is not needed at this stage—U-Boot checksums each of the three versions across the triplicated partition. If any of the checksums match the stored sum, the BLOB in that partition is assumed not to be corrupted and can be loaded. In the case that all three checksums are mismatched, bit voting is used to reconstruct the BLOB from the three corrupted versions, a method that is probabilistically resilient to bit-flips and potentially minor sector failures. A bit voting logic gate is illustrated in figure 3.1. Note that, in order to speed up the boot process, which as discussed earlier is a vulnerable state for the SoM [11], the reconstruction is only used in volatile memory in U-Boot—

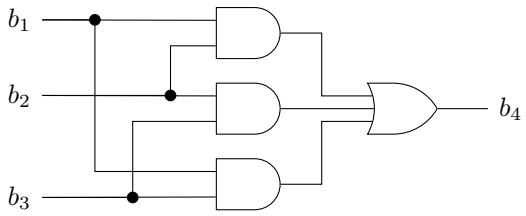


Figure 3.1: A simple bit voting logic gate using AND and OR blocks. The output is the majority "vote" of the three input bits.

not yet written back to flash—for the sake of time. Once this process has been completed for each of the four BLOBs, U-Boot boots into the initial RAM disk, using the RAM disk image, the device tree, and the kernel image.

Once booted into the initial RAM disk, the same logic is executed with a few key differences. First, this uses all five BLOBs, as the root file system is mounted at this stage. Additionally, instead of stopping checksums once a valid BLOB is found, all three checksums are performed so that any corrupt BLOBs can be corrected in flash. With the kernel loaded at this stage, this can be sped up via multi-processing.

3.1.3 RAM-based File System

Upon entry to the initial RAM disk, a temporary file system (tmpfs) is initialized in RAM. Then, after the emulated TMR process detailed above, the root file system is extracted from its tarball into the RAM disk. Hence, all reads from and writes to the root file system occur in volatile memory, greatly reducing flash usage post-boot.

To allow for writes back to the persistent hard-disk for essential data products, additional EXT4 file systems are mounted at `/config` and `/data`. While data stored here is not redundant by default, these partitions give missions ease of access to persistent storage when necessary, leaving it up to the specific use case how often flash should be used. For instance, for power-heavy software, developers may need to store the final data products between reboots if data cannot be transferred immediately. In this case, intermediate data products can be stored within the RAM-based file system, while the final data product may be copied to `/data` for downlinking or further processing on the next boot-up. To provide integrity,

to an extent, to these partitions, `fsck` (a file system checker) is implemented on boot to correct any recoverable inode reference issues.

The main downside to using a RAM-based file system is apparent with a large file system. In particular, including the central NVIDIA CUDA libraries for most GPU programs to be supported generates a file system of over a gigabyte in size, which uses up a static 15% of available RAM. Hence, software applications must have a relatively low memory usage in order not to exceed the given RAM availability.

3.1.4 Boot Time Considerations

In addition to the increased RAM usage induced by a large file system, file system size directly influences boot time. In the nominal boot case, where no corruptions have occurred, a large portion of boot time is spent hashing each BLOB, and the root file system takes the longest. While parallelization is employed in the initial ram disk to expedite the boot process, this portion can take over 30 seconds to complete. This approximately doubles boot time (from about 40 seconds to 80 seconds) in the nominal case for large file systems, and to a higher extent when there is corruption to be corrected. In most cases, this tradeoff is worth the increased system reliability, but such a decision is dependent on mission requirements.

3.1.5 Real-Time Linux Patch

In spacecraft processors, real-time scheduling is generally necessary to ensure that tasks are executed when expected. The `meta-sol` layer includes the `PREEMPT_RT` patch that enables the real-time scheduling paradigm on the Linux kernel. Hence, software commands may be sent over the mission's payload interface without disrupting the OBC's real-time operations. Additionally, this allows for predictable and consistent functionality in the TX2i's software system.

3.2 Experimental Design

The following three experiments test the SOL-loaded TX2i's efficacy on various aspects of providing a platform for LEO-based computation. In order to do so, three experiments are run to answer the following questions, respectively:

1. Does the TX2i's SoC itself perform well enough to reliably function as a payload computer?
2. Can bit flips induced in the file system, assuming eMMC still operates nominally, be corrected through SOL's boot scheme?
3. What impact does the real-time patch have on command execution latency and reliability on the TX2i?

In Experiment 1, a proton beam is used to perform stress tests for a more in-depth analysis of SoC performance than previous literature generates. Experiment 2 discusses a fault injection procedure for verification of SOL's redundant boot scheme. The final experiment compares latency on L4T and SOL.

3.3 Experiment 1: Proton Beam Irradiation

3.3.1 Procedures

Proton beam acceleration tests were completed at TRIUMF's Proton Irradiation Facility in Vancouver, BC, on beam line 2C [39]. During the time of testing, the beam line operated at two proton energies: 63 MeV and 105 MeV.

Four TX2i devices under test (DUTs) were used for this experiment. DUTs 1 and 2 were both loaded with SOL and were in new condition, aside from flashing and preparation for the experiment. DUTs 3 and 5 were L4T devices used as controls/baselines for comparison. Both L4T devices had been used strenuously beforehand; DUT 5 experienced a few years worth of software/hardware testing at UGA, and DUT 3 was previously killed and revived after a 50-kRad TID test performed by APL in the same

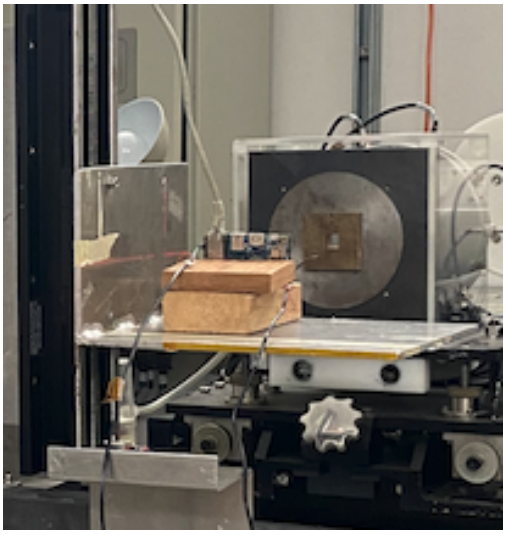


Figure 3.2: The 1 cm $\times$ 1 cm square beam collimator used for aligning the beam to the SoC.

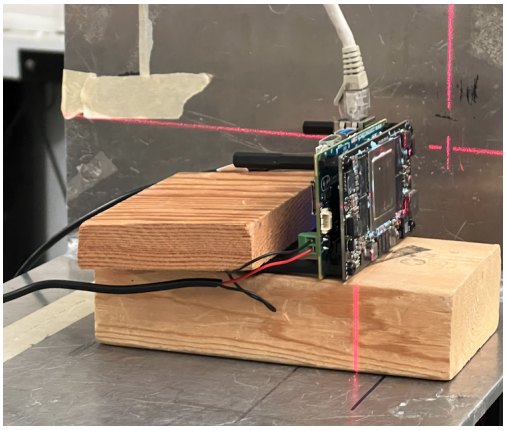


Figure 3.3: Alignment of one DUT across from the proton beam.

report as their proton tests [11]. Although brand-new devices would be preferred, availability was severely limited due to supply chain shortages.

Prior to arriving at the TRIUMF facility, each DUT's heat sink was removed, and thermal paste was taken off chip surfaces. Each TX2i was connected to a Connect Tech, Inc. Orbitty carrier board. From the board ran a serial line for monitoring kernel messages and panics over UART0, an Ethernet cable for commencing stress tests and additional logging, and a power supply. All three cables ran outside the beam room into a control room for ease of use during and between beam runs. The collimator, shown in figure 3.2, created a 1 cm $\times$ 1 cm square irradiation area that could be aligned to cover the SoC's die. Alignment to the center of the beam could be performed via a 3-axis laser system as shown in figure 3.3.

While the test setup was constructed to replicate APL's tests [11] as closely as possible, beam parameters and environmental factors could not all be controlled for. In particular, a narrower collimation at TRIUMF than that used by APL contributed to less spallation onto peripherals. As flash memory, directly below the chip, was hypothesized to contribute to the majority of APL's permanent DUT failures, the current tests could not be expected to stress such peripherals to the same extent. Although this disparity did not allow for testing of SOL's flash memory fault tolerance, it did allow for more thorough tests to be completed on the chip itself.

A similar, but more extensive software suite to APL's was loaded onto each device. This software is open-source in the `meta-sol` repository and includes mechanisms for logging and four types of stress tests. The first test, `stress-ng` is a general stress test on the CPU and memory units. CUDA sample programs, namely merge sort and bandwidth tests, were used to stress the GPU. In-house computer vision algorithms were *not* tested, as the accelerated flux produced by the proton beam is not representative of what the high-intensity algorithms would experience in LEO. Finally, `rt-migrate-test` for testing real-time priority scheduling and `memtester` as a more intense memory stressor were both included, although each was only used once. Time series data regarding power usage, thermals, device activity (including CPU and memory usage), and test-metadata were logged to an InfluxDB instance running on the laptop connected to the DUT over Ethernet. A time synchronization command would be run at the start of

each test, and the time was output to each kernel message log in order to associate times between kernel panics and Influx data (up to a second).

3.3.2 Results

Table 3.2: Proton test log.

DUT	Run No.	Runtime (min:sec)	Energy (MeV)	Avg. Flux (p/cm^2/sec)	Fluence (p/cm^2) Run	Fluence (p/cm^2) Cumulative	Dose (kRad Si) Run	Dose (kRad Si) Cumulative	Test Type	Failure Type Primary	Failure Type Secondary
L4T: 3	1	1:40	63	5.83E+07	5.83E+09	5.83E+09	7.73E-01	7.73E-01	Stress	GPU PMU	GPU FIFO
	2	4:05	63	3.53E+06	8.65E+08	6.69E+09	1.15E-01	8.88E-01	Stress	CPU	ROC:CCE
	3	0:19	63	3.68E+07	7.00E+08	7.39E+09	9.28E-02	9.81E-01	Stress	CPU	CPU MEM
	4	0:26	63	3.64E+07	9.45E+08	8.34E+09	1.25E-01	1.11E+00	Stress	Watchdog	CPU MEM
	5	0:06	63	4.04E+07	2.42E+08	8.58E+09	3.21E-02	1.14E+00	Stress	ROC:CCE	CPU MEM
	6	0:53	63	3.69E+07	1.96E+09	1.05E+10	2.60E-01	1.40E+00	Stress	CPU MEM	CPU
	7	0:05	63	3.84E+07	1.92E+08	1.07E+10	2.54E-02	1.42E+00	Stress	ROC:CCE	CPU
	8								Reboot		
SOL: 1	9	0:22	63	3.48E+07	7.65E+08	7.65E+08	1.01E-01	1.01E-01	Stress	CPU MEM	CPU
	10	0:42	63	3.40E+07	1.43E+09	2.19E+09	1.89E-01	2.91E-01	Stress	CPU	CPU MEM
	11	0:22	63	3.21E+07	7.07E+08	2.90E+09	9.38E-02	3.85E-01	GPU	CPU	-
	12	0:18	63	3.17E+07	5.71E+08	3.47E+09	7.58E-02	4.60E-01	GPU	CPU	-
	13	0:13	63	2.98E+07	3.87E+08	3.86E+09	5.13E-02	5.12E-01	Memory	ROC:CCE	CPU
	14	0:07	63	3.06E+07	2.14E+08	4.07E+09	2.84E-02	5.40E-01	RT	ROC:CCE	CPU
	15	0:11	63	3.21E+07	3.53E+08	4.42E+09	4.68E-02	5.87E-01	Log-only	-	-
	16	0:36	63	1.05E+07	3.77E+08	4.80E+09	5.00E-02	6.37E-01	Log-only	CPU	-
	17	0:23	63	1.11E+07	2.55E+08	5.06E+09	3.38E-02	6.71E-01	Log-only	CPU	ROC:CCE
	18	1:13	63	4.38E+06	3.20E+08	5.38E+09	4.24E-02	7.13E-01	Log-only	CPU	-
	19	3:05	63	4.56E+06	8.44E+08	6.22E+09	1.12E-01	8.25E-01	GPU+Stress	-	-
	20	0:52	63	3.72E+06	1.94E+08	6.41E+09	2.57E-02	8.51E-01	GPU+Stress	CPU	ROC:CCE
	21								Reboot		
L4T: 5	22	0:49	105	1.22E+07	5.97E+08	5.97E+08	5.40E-02	5.40E-02	Stress	CPU	ROC:CCE
	23	0:35	105	1.19E+07	4.17E+08	1.01E+09	3.77E-02	9.18E-02	GPU	CPU MEM	CPU
	24	0:40	105	1.20E+07	4.80E+08	1.49E+09	4.35E-02	1.35E-01	Stress	CPU	-
	25	0:27	105	1.19E+07	3.23E+08	1.82E+09	2.92E-02	1.64E-01	Stress	ROC:CCE	CPU MEM
	26	3:12	105	1.18E+07	2.26E+09	4.08E+09	2.05E-01	3.69E-01	Stress	ROC:CCE	CPU MEM
	27	2:43	105	1.20E+07	1.96E+09	6.04E+09	1.77E-01	5.47E-01	Stress	ROC:CCE	CPU MEM
	28								Reboot		
SOL: 2	29	0:45	105	1.09E+07	4.93E+08	4.93E+08	4.46E-02	4.46E-02	GPU	ROC:CCE	CPU
	30	0:39	105	1.08E+07	4.22E+08	9.15E+08	3.82E-02	8.28E-02	Stress	CPU	CPU MEM
	31	3:21	105	1.11E+07	2.22E+09	3.14E+09	2.01E-01	2.84E-01	GPU+Stress	CPU MEM	CPU
	32	3:10	105	1.14E+07	2.16E+09	5.30E+09	1.96E-01	4.80E-01	GPU+Stress	ROC:CCE	CPU
	33	0:16	105	1.09E+07	1.74E+08	5.48E+09	1.57E-02	4.96E-01	GPU+Stress	-	-
	34	10:03	105	1.11E+07	6.68E+09	1.22E+10	6.05E-01	1.10E+00	Off	None	None

Table 3.2 details the characteristics of all 34 runs completed at TRIUMF. Any reboot, whether soft (automatically forced by the kernel) or hard (where the device froze and a power cycle was necessary) constituted a new run. Runs 8, 21, and 28 were simply reboots to ensure nominal operations after each DUT completed its irradiation. In run 34, the DUT was kept powered off to be exposed to a constant radiation stream for ten minutes, although this did not produce any off-nominal behaviour upon boot-up.

No permanent failure occurred on any DUT, whereas APL's devices saw permanent failure after an average of four reboots. This is likely due to the previously mentioned difference in collimator sizes, and

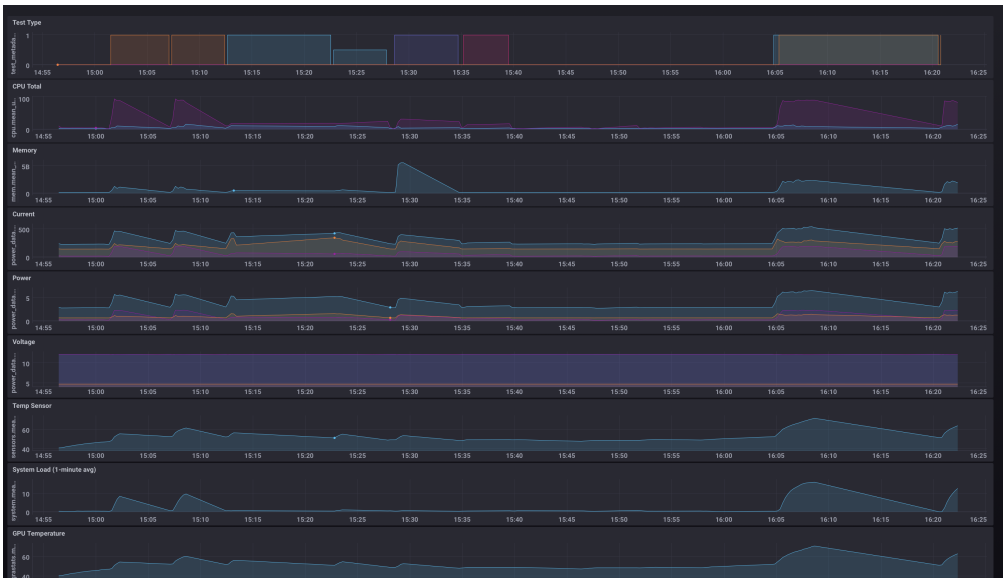


Figure 3.4: Chronograf visualization of InfluxDB radiation data.

the implications are discussed in the following section. Error types and kernel panics found in the kernel log are similar to those described by APL, although there were no flash errors logged at TRIUMF. The most common errors were CPU errors (`SError`), ROC:CCE machine check errors, and CPU memory errors. There was also an assortment of GPU errors, including memory management errors and single-bit upsets. Many of these occurred even when no GPU-specific tests were running.

Figure 3.4 illustrates the ease-of-visualization of InfluxDB time series data through Chronograf. Data points can be easily correlated with testing instance and type as well as corresponding lines in the associated kernel logs. No clear degradation occurred to any sensor or component over each DUT's irradiation runs, nor did any SEE-induced anomalies appear in the on-chip sensor data. This supports the possibility of using such sensor data in real-time and over telemetry to characterize onboard anomalies reliably.

SEE cross sections, defined as the number of events per amount fluence, are used for analysis and comparison to previous tests. For direct comparison with APL's data, the cross sections for reboot events are plotted as a function of proton energy in figure 3.5. Cross sections are consistent between SOL and L4T data captured at TRIUMF, as well as in comparison to APL's cross sections. Notably, the best-performing runs in the 60-63 MeV range, indicated by the lowest cross sections, were from DUT 3 at

TRIUMF, the device that had been previously killed by 50 kRad of ionizing dose. A hypothesis for the occasional out-performance of L4T devices over SOL devices on similar tests is the potential that the `PREEMPT_RT` updates allowed for earlier detection and correction (via reboot) of SOL devices, whereas L4T devices continued running until hard failure.

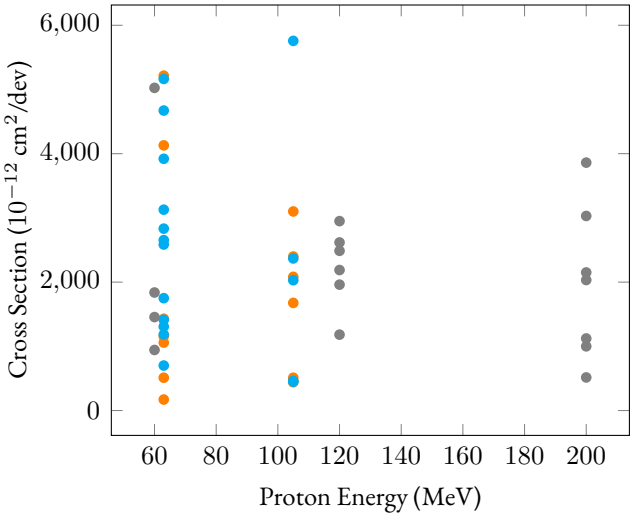


Figure 3.5: Cross sections of reboot events of both SOL (blue) and L4T (orange) devices, in comparison to data from APL's radiation tests on L4T devices (grey).

CREME96 [40], an online suite of tools currently hosted by Vanderbilt and often used for various SEU analysis, was used to further break down cross sections into rate estimates. First, the TRP tool was used to generate trapped proton calculations for ISS orbit—500 kilometers with 51.6-degree inclination—in both solar minimum and maximum environments. The Geomagnetic Transmission Routine (GTRN) then factored in the shielding effects of earth's magnetic field. Next, the FLUX utility was used to generate a model of ionizing materials on the satellite's surface, and the nuclear transport routine (TRANS) was used to account for a standard 100 mils of aluminum shielding. Finally, the PUP utility could be used to predict rate estimates of the TX2i under such orbital and shielding conditions. Based on the estimates shown in table 3.3, less than four reboots are expected per year of up-time on SOL. As discussed in reference to the cross-section plot, although rates are better for L4T devices, this may be a result of early detection and fixes done in SOL to prevent the hard system freezes that occurred in most L4T runs.

Table 3.3: Reboot rate estimates in LEO.

OS	Reboots per Day		Reboots per Year	
	Solar Min.	Solar Max.	Solar Min.	Solar Max.
SOL	8.72E-03	5.21E-03	3.18	1.90
L4T	2.80E-03	1.50E-03	1.02	0.55

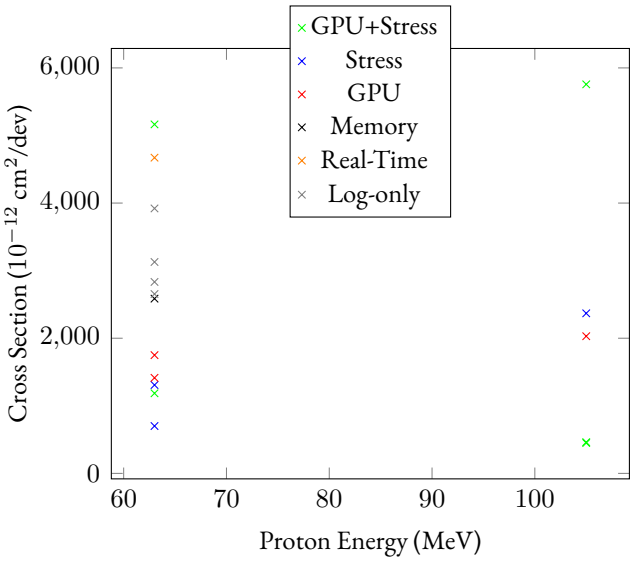


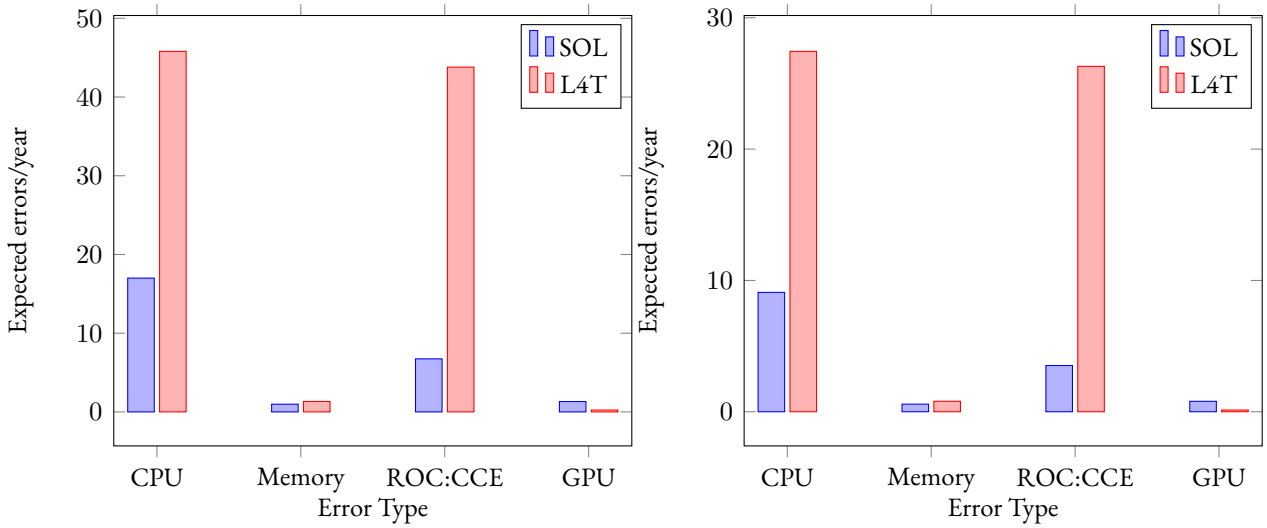
Figure 3.6: Cross sections of SOL-loaded DUTs classified by test type.

Cross sections for SOL devices are plotted separately by test type in figure 3.6. Aside from consistent performance when no stress tests were running—that is, log-only runs—there were not any clear patterns in terms of performance under a particular test. Surprisingly, there were cases where stress and GPU tests performed better than log-only runs. Additionally, combinations of stress and GPU tests generally performed longer than either test alone, even though Chronograf verified that the device was stressed to a higher extent during combinatory tests.

To analyze specific error types and their number of occurrences, a similar cross sectional analysis as above was completed using kernel message occurrences as events. These messages were grouped into the categories of CPU errors (SError), memory errors, ROC:CCE errors, and GPU errors. Since many runs did not experience every error type, cross sections were calculated by-device based on cumulative fluence and total errors over all runs. These cross sections were input into the CREME96 PUP tool using the same

orbital and shielding environment as above. Daily rates are then multiplied by 365 in order to convert to a yearly rate estimation per error type.

Results for both SOL and L4T devices are shown in figure 3.8. While reboot rate estimates were higher for SOL devices, error estimates are significantly lower. This further supports the hypothesis that the real-time patch provides better early detection and correction of such errors. GPU error rate estimates are slightly higher for SOL devices, but this is likely due to sampling error, as only one L4T run underwent a GPU test.



(a) Solar Minimum

(b) Solar Maximum

Figure 3.8: Expected error rates within one year of up time at both solar minimum and maximum.

3.4 Experiment 2: TMR Verification

3.4.1 Procedures

This section provides a brief overview of fault injection techniques used to test bit flips within the scope of the SOL's redundant boot scheme. The general procedure involved determining where each BLOB lies in flash memory and directly corrupting bytes of data in specified locations through UNIX utilities

such as `dd`. For each test-case, the device would be rebooted, and logs would be checked to ensure that the expected corrections occurred at boot-time. This was done for all combinations of the three BLOB copies as well as the five different BLOB files, repeatedly, to verify robust fault tolerance in the expected cases.

For a more realistic test, random bits were corrupted throughout eMMC to examine bootloader responses and time to failure.

3.4.2 Results

In the first test, expected corrections occurred on each reboot, and the device could operate after each boot without issue, even when all BLOBs had been corrupted. For completeness, similar tests were run on L4T devices, and in any case of corruption to a core file—such as the kernel image—the device would not reboot.

When corrupting bits through eMMC, both SOL and L4T devices failed after 10,000 bit flips between boots. The SOL device had multiple detected corruptions that were corrected on previous boots with lower amounts of bit flips, whereas the L4T device incurred un-recovered damage to libraries in its root file system. It is important to note that such effects will accumulate on L4T, but be corrected incrementally (based on how often boots occur) on SOL.

3.5 Experiment 3: Latency Comparisons

3.5.1 Procedures

To test the applicability of the `PREEMPT_RT` patch in SOL, a latency comparison was assessed across SOL and L4T devices. The `rt-migrate-test` schedules tasks with various priorities in parallel to ensure that higher-priority tasks have low latency. The same test can be run on non-real-time environments for comparison purposes, although priority cannot be guaranteed on the standard Linux kernel.

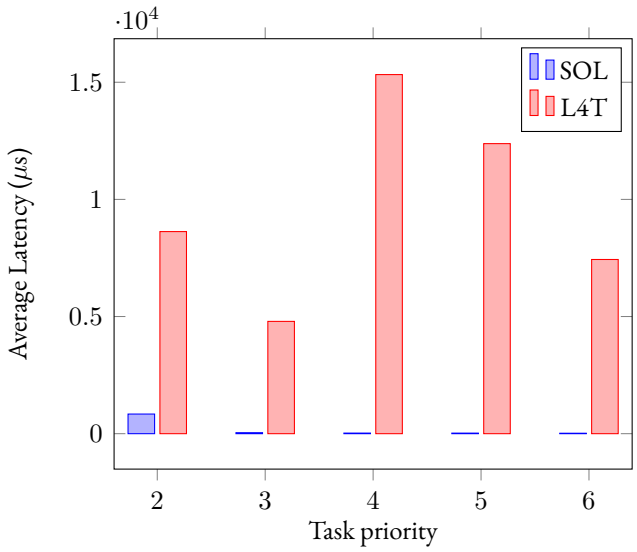


Figure 3.9: Average latency from `rt-migrate-test`.

Additionally, to test latency in a more realistic scenario, response times to an OBC, using MOCI's payload communication protocol, were recorded for both devices. The communication occurs over UART, and a simple ping command was sent by the OBC to be acknowledged by the TX2i. This was repeated for 50 trials on each device.

3.5.2 Results

Figure 3.9 shows the average latency of tasks at each priority setting. As expected, latency is notably lower on SOL devices and decreases with increasing priority. There was also notable consistency in terms of responses; priority 6 tasks ranged from 14-63 microsecond latency on SOL devices compared to 23-31268 microseconds on L4T.

Ping tests from the OBC averaged a 474.26 ms response on the SOL device and 496.98 on L4T, over 50 trials each. However, the logged time is relative to when each debug statement was received from the OBC rather than the time the packets were actually returned. Since print statements are relatively low priority, there is likely some noise within the data. For more precise measurements on future tests,

flight software should be updated to include a reading from the OBC's real-time clock in such debug statements.

3.6 Discussion

The following discussion is framed around answering and expanding upon the three questions posed in the beginning of the experiments section.

3.6.1 Performance of System on Chip (SoC)

With a narrower collimator than that used by APL in their previous report [11], the tests at TRIUMF allowed for a more thorough investigation of the SoC itself under proton irradiation. Most notable is the fact that, with less spallation into peripherals, there were no permanent failures. While this does not conclusively prove that the flash eMMC card is the culprit of APL's permanent failures, it does confirm that the issues are off-chip—and hence likely not an issue with the flash memory controller. This lends support to potential future projects involving hardware level customizations to the SoM, including replacing the flash memory module with rad-hard storage for future missions.

Also notable is that, without such permanent failures, the DUTs at TRIUMF went through approximately twice as many radiation runs as those tested by APL, allowing for the more detailed analysis to be performed above. Rate estimates on both reboot events as well as general errors show that, especially on SOL, minimal on-chip errors will occur for nominal run times.

These findings partially oppose the general movement in the space computing community towards other processors such as AMD, as seen in [17], by showing that the TX2i SoC is reliable under radiation. However, future projects may require support from NVIDIA to decouple the chip from the unreliable peripherals.

3.6.2 Correcting Flash Errors

Due to the fact that only the SoC itself was tested on SOL devices and no detected flash errors occurred, there cannot be a direct assessment of how well the boot redundancy scheme performs after SEUs. However, some inferences can be made regarding the results from APL's tests.

Two of APL's failed DUTs were recoverable through a re-flash, indicating that there were upsets in the eMMC flash card that could be corrected through a write operation. Since this is the case, software TMR should theoretically be able to fix the same issue.

The boot redundancy scheme does, however, work in a probabilistic manner. Only essential high-level binaries in the APP partition (which is reduced to 4 GB) is triplicated, whereas all lower level partitions rely on NVIDIA's A/B redundancy. The next largest partition required at boot time is the U-Boot kernel at 84 MB, significantly less likely to experience SEUs, especially in both A and B slots. However, if cboot—which loads U-Boot—were modified in a similar manner to apply triple modular redundancy to the U-Boot binary, which could be a future revision on SOL, all non-triplicated essential partitions would be under 10 MB each.

3.6.3 Command Execution Time

Results from `rt-migrate-test` have shown that the real-time patch drastically reduces device latency on essential tasks, which is especially necessary in the case where the TX2i is implemented as a payload computer. Untested and unexpected edge cases that occur due to increased latency could be detrimental in flight.

3.7 Conclusion and Future Work

High performance computation for computer vision and automation tasks are becoming increasingly more prevalent in the aerospace industry. CubeSats in particular, through short-lifetime missions, are

accelerating the speed at which new techniques can be tested and evaluated. Many of these missions lean toward NVIDIA Jetson system on chips due to their form-factor, availability, and ease-of-development.

Proton tests completed by the Applied Physics Laboratory, as well as the Small Satellite Research Laboratory in this paper, show that while the SoC is fairly resilient to single event effects, peripherals are significantly more vulnerable on the Jetson TX2i. Moreover, flash memory is the most likely culprit.

Space Operating Linux provides robust software-based solutions to single event upsets in flash as well as general latency improvements for near real-time processing. The operating system is built through Yocto and kept open-source so that payload developers may easily adapt the source code for their mission needs.

Future projects are directed toward further analysis of InfluxDB data retrieved from proton testing with the eventual goal of creating a model of anomaly detection that can be flown on the TX2i. If this becomes possible, the TX2i can have a background daemon that decides when to pause/restart processing due to various environmental factors. On MOCI, such sensor data telemetry will be downlinked with boot logs, kernel messages/panics, and external radiation sensor data to add to the data-pool. Data from the TRIUMF radiation tests also further supports the possibility of a future project that realizes a new board, with rad-hard components, built around the Tegra SoC.

Chapter 4

Computer Vision Algorithms and Optimizations

MOCI's high-level mission objectives are to produce high-accuracy DEMs and recognize ground-level objects *in situ*. The former goal is of greater focus in the SSRL and in this research, as it uses an in-house structure from motion (SfM) framework first developed by previous master's students [12, 10]. This chapter discusses limitations of the initial SfM—in regards to onboard constraints and sensor noise—of the pipeline and presents solutions to make the model space-ready. A similar, but narrower, evaluation is performed on MOCI's object detection framework to ensure its operability in a low-memory system.

4.1 Development Operations

In order to allow for reliable team development on the `SSRLCV` codebase, which houses the SSRL's computer vision framework, continuous integration and deployment approaches have been enforced. These methods are discussed here briefly in order to provide context to the development and testing environment mentioned in further sections.

4.1.1 CUDA Programming Model

One of the primary reasons NVIDIA GPUs are often preferred for embedded edge computing applications over other companies—and even other hardwares, such as FPGAs—is their ease of programming. This can be attributed to CUDA, an API created by NVIDIA to enable a higher-level paradigm of development. MOCI's SfM pipeline, `SSRLCV`, was developed through the CUDA extension to C++. This permits any developer with a working knowledge of C++ to have a seamless transition to GPGPU programming, with the addition of a few more concepts.

CUDA code can be compiled for either the host (CPU) or device (GPU), or both. *Kernel* methods are those designed to run on the device to carry out a particular—usually single instruction multiple data (SIMD)—task. The developer specifies the number of threads per block and blocks per grid in the caller function on the host, and any data accessed by the kernel must be copied into device memory space. Each thread executes the same kernel on a different core, and threads on the same block are guaranteed to be dispatched on the same SM, allowing for synchronization across the block within a kernel [41].

4.1.2 Memory Enhancements

A useful feature in modern C++ is the concept of smart pointers. In `SSRLCV`, they have been extended to fit the CUDA memory model and automatically allocate, de-allocate, and transfer memory on both the CPU and GPU in an effort to reduce memory leaks. This wrapper also allows for such memory allocations to be logged over time to monitor both CPU and GPU usage of shared memory.

4.1.3 Continuous Integration

`SSRLCV` is primarily hosted on an internal Gitlab instance. Using Gitlab's built-in continuous integration features, merge requests deploy automated unit tests to the University's high performance computing cluster, Sapelo2, to ensure that intermediate data products and results of component operations only change when they are expected to. All merge requests must pass every test as well as a human code review

before its acceptance in order to guarantee only high quality code reaches the main branch (and eventually flight).

While the majority of tests are run on the computing cluster—which includes higher-end GPUs—for ease of development, major updates are manually tested on a TX2i to ensure expected operation in-flight. An important note is that, while optimizations may speed up some operations on one GPU, the speed-up may not scale linearly to—or even occur at all on—another GPU. This is due to the multiple architectural factors that impact performance, causing disparities in metrics such as memory bandwidth and floating point operations per second (FLOPS).

4.2 Perspective Geometry

SfM fundamentally relies on perspective geometry, and implementations generally employ a pinhole camera model, or variations thereof. The basic assumptions of this model are illustrated in figure 4.1. The origin of this model is the *camera center*, and the camera's *principal axis* extends out through the lens. The *image plane* lies a focal length out from the camera center on the principal axis, and a 3D point $\mathbf{X}$ is projected onto the given image at the location $\mathbf{x}$ given by the point of intersection between the ray from the camera center to that point and the image plane. Conversely, a given pixel location $\mathbf{x}$ can correspond to any 3D point along the given ray [42].

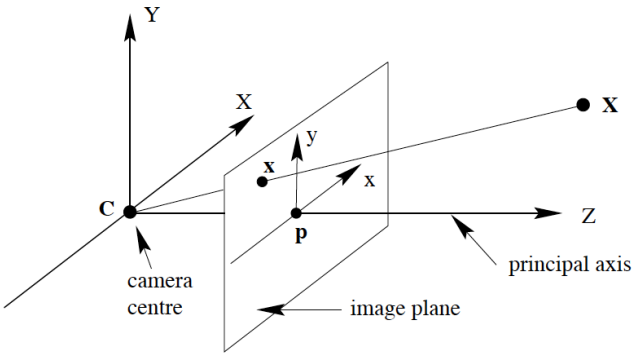


Figure 4.1: The pinhole camera model [42].

4.2.1 Homogeneous Coordinates

Homogeneous coordinates provide elegant parametrizations to the relationships present in perspective geometry. Euclidean space $\mathbb{R}^n$ can be extended to the corresponding projective space $\mathbb{P}^n$ where coordinates are represented by an $(n+1)$-vector. A point's canonical representation is given by appending a 1 to the end of its Euclidean vector, and nonzero scalar multiples represent the same point. Ideal points are those whose final coordinate is zero, and these have no correspondence in $\mathbb{R}^n$. This can be visualized using a similar model to figure 4.1 as $\mathbb{P}^2$, where points in projective space can be considered rays from the origin and correspond to a point in $\mathbb{R}^2$ given by the intersection of that ray with the plane $z = 0$. Note that the camera center and points at $z = 0$ are ideal points, as they do not have projections onto the image plane. However, such points may be visible from other camera perspectives, as can be represented by projective transformations such as a rotation and translation [42].

In `SSRLCV`, homogeneous coordinates are not directly used in most cases, and instead ray tracing techniques are employed for efficiency. However, there are some cases discussed below in regard to relative camera orientations where the use of homogeneous coordinate representations does prove beneficial.

4.2.2 The Camera Matrix and Fundamental Matrices

Using homogenous coordinates, the correspondence between a 3D point $\mathbf{X}$ and its image correspondence $\mathbf{x}$ can be parametrized as

$$\mathbf{x} = P\mathbf{X} = K[R \mid \mathbf{t}]\mathbf{X}$$

where R and $\mathbf{t}$ are a Euclidean rotation and translation from the world to camera frame (with camera center at the origin), respectively. K is a calibration matrix of the form

$$K = \begin{bmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix},$$

where α_x and α_y are the focal length in terms of pixel dimensions, s is a skew parameter, and (x_0, y_0) is the location of the principal point on the image (*i.e.*, its intersection with the principal axis, in pixel coordinates).

In the case of two images, a fundamental matrix F can be derived describing their relative geometry, called *epipolar* geometry. Namely,

$$\mathbf{x}'^{\top} F \mathbf{x} = 0$$

for corresponding points $\mathbf{x}$ and $\mathbf{x}'$ on the two images. Additionally, $F\mathbf{x}$ maps a point on the first image to a line of potential correspondences on the second, called the *epipolar line*.

4.3 Structure from Motion Pipeline

MOCI's SfM pipeline, as originally developed by previous master's students, contains five main stages [12, 10]. While this research presents and evaluates alternate implementations to solve various constraints and improve accuracy, the overall pipeline structure generally remains the same. The original implementation is summarized below:

4.3.1 Feature Generation

The first stage of the pipeline is to extract features and descriptors for both images. MOCI employs the Scale Invariant Feature Transform (SIFT), which although slower and more power-heavy than other feature detectors, is tremendously more robust. Accuracy cannot be compromised as this stage, as shortcomings at this stage propagate to errors in the point cloud.

SIFT was originally devised by Lowe in 1999 and is a common, robust feature generator for structure from motion pipelines [43, 27]. The algorithm convolves images with a Gaussian filter with multiple values of σ and computes a difference of Gaussians to approximate a Laplacian of Gaussians operation. In a pyramidal scheme, the images are downsampled, and operations are repeated for scale invariance. Extrema are detected at each scale by comparing each pixel to its 26 neighbors—eight on the same image

and nine on each neighboring blur. A histogram of oriented gradients can then be computed at each extremum as a feature descriptor at that keypoint [44].

4.3.2 Feature Matching

Once features are produced in all images, they can be matched to perform pixel correspondences between overlapping images. The original implementation compares SIFT descriptors exhaustively across images by Euclidean Distance to find keypoint's best match in the next image. This match is discarded if the Euclidean distance either surpasses a set absolute threshold, or if the ratio of the Euclidean distance to that of the best match in another unrelated image—called a *seed*—surpasses a set relative threshold.

4.3.3 Triangulation

Using the pinhole camera model, matches among two or more images can be re-projected into three-dimensional space. Rays are projected out from each camera center through its corresponding pixel. In a continuous environment without noise, these rays would intersect at the point's true 3D location. To account for imprecision, this point is taken to minimize its distance from each ray.

4.3.4 Filtering

Although filtering is not usually considered a stage in SfM pipelines, it is important in ensuring that erroneous features (noise) do not obstruct the point cloud. As the next stage is a gradient-descent problem that is highly susceptible to outliers.

4.3.5 Bundle Adjustment

Bundle adjustment seeks to correct the camera parameters (intrinsic and extrinsic) as well as point locations in order to correct for the linear error between skew lines. In the initial pipeline, only a minimal version of bundle adjustment was implemented using Newton's iterative method for least squares minimization of re-projection error. The method is highly susceptible to rotational errors, particularly around the x

and y axes, and only generally converges to a local minimum. This has also only been implemented for 2-view cases.

Bundle adjustment, or some form of iterative estimation, is generally a required aspect of SfM pipelines, especially as most applications do not have known camera parameters to a high accuracy [27].

4.3.6 Limitations

The obvious limitations in this implementation are that it pushes runtime, memory, and power constraints of the MOCI mission. Memory is especially limited due to the RAM-based file system present in Space Operating Linux (see chapter 3).

Other limitations come in terms of accuracy. For complex images, such as those previously simulated of Mount Rainier, the point cloud is barely visible due to the high noise density.

The three main bottlenecks in the pipeline, as shown in [12], are feature generation, feature matching, and bundle adjustment. Feature generation has the highest power usage of any stage due to the memory transfers involved in Gaussian convolutions. Feature matching and bundle adjustment are the longest stages, in some cases exceeding the maximum given runtime of 45 minutes.

4.4 Pointing Accuracy

Due to its high pointing accuracy and small form factor, the CubeSpace ADCS was chosen for MOCI. However, its most accurate sensor, the CubeStar star tracker, cannot be used at MOCI's SfM slew rates, introducing noise to attitude estimates. This noise is coupled with time synchronization issues on both the ADCS and TX2i's end, as there is no direct line of communication between the two, and the ADCS samples attitudes at only 1 Hz.

4.4.1 SfM Slew

The proposed SfM slew on MOCI starts and ends at 15-degree angles θ from the ground target on either side. One side of the slew is shown in figure 4.2. However, as previously mentioned, this would exceed the

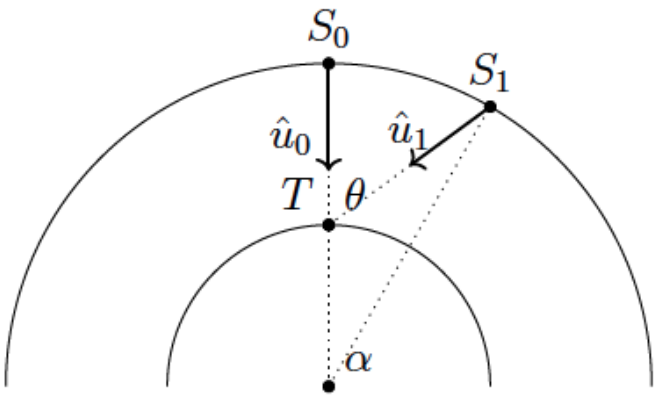


Figure 4.2: Proposed SfM slew [12].

star tracker's maximum angular rate of 0.3 degrees per second. Other slew options have been considered and are presented below. However, each option brings its own potential issues.

Off-Track Slewing Options

Slew options have also been simulated for targets that do not lie directly in the orbital plane. However, these only slightly decrease the pointing accuracy and do not approach the 0.3 degree requirement for accurate pointing.

Non-Slew Options

Adams proposes the option of pointing nadir throughout imaging. However, due to the low B/H ratio of this option, accurate results would not be plausible. Existing models like ASTER have used a B/H ratio of 0.6 [30], and MOCI would achieve a similar result with the proposed 30 degree slew. However, with the small field of view, nadir pointing only keeps targets within frame for under five seconds (depending on how much overlap is desired), yielding a fraction of the B/H ratio.

There is also the option of simply taking images of the same target over multiple passes, at different pre-set track angles relative to nadir. However, this requires high positional accuracy and is less likely to

produce a set of unobstructed (cloud-free) images of the target. This problem was also considered by ASTER, which performed both on-track and repeated imaging but had a longer lifetime [30].

4.4.2 Custom Pose Estimation

As discussed above, the alternatives to on-track slewing are likely to produce undesirable results. Hence, a custom pose estimation algorithm is employed and evaluated. This has been done successfully in commercial SfM software and is considered a pipeline stage in most implementations, with pose estimation considered one of the main objectives of SfM [27].

4.5 Adjusted and Optimized Pipeline

An adjusted pipeline is presented below in order to decrease the intensity of bottlenecks on the pipeline and adjust for noisy sensor data. The next section provides a memory, runtime, power, and accuracy comparison to the previous iteration.

In addition to architectural differences, the pipeline also has the capability of checkpointing states at the end of each stage, which can be specified through command-line flags. This serves the purpose of allowing the pipeline to restart on reboot if runtime limits are exceeded or upon environmentally caused kernel latch-ups. Ideally, checkpointing should only occur after stages with smaller data products, rather than those like feature generation with the largest products, in order to reduce writes to the eMMC card.

4.5.1 Feature Generation

The hypothesized reason for the runtime and power bottlenecks seen in feature generation was the size of bulk memory transfers. Required for parallelizing the algorithm. Optimizations were made in the logic flow to ensure that memory was only copied and reallocated when needed.

Additionally, to reduce runtime and load on the GPU, the Gaussian kernel has been separated as followed into its one-dimensional parts:

$$\frac{1}{2\pi\sigma^2}e^{-(x^2+y^2)/2\sigma} = \left(\frac{1}{\sigma\sqrt{2\pi}}e^{-(x^2)/2\sigma}\right)\left(\frac{1}{\sigma\sqrt{2\pi}}e^{-(y^2)/2\sigma}\right).$$

This modification can hence be convolved as two, one-dimensional kernels for a decrease in runtime complexity [43].

4.5.2 Pose Estimation

The goal of this stage is to offload the optimization of camera pose from the bundle adjustment by producing relatively accurate parameters to start. This is done before feature matching so that relative orientations may be used for the optimizations in that stage discussed next.

Initial Guess: Random Sample Consensus

An initial estimate of the attitude can be determined through a random sample consensus of matches. First, pairwise matches are computed with a low threshold so that only matches with highly similar SIFT descriptors are considered in pose estimation. This is done both to compute the matches at a faster rate and to reduce the number of outliers.

Next, the matches are randomly split into sets of seven pairs. Labeling each correspondence as $\mathbf{x}_i = (x_i, y_i)$ and $\mathbf{x}'_i = (x'_i, y'_i)$, the equations $\mathbf{x}_i'^{\top} F \mathbf{x}_i = 0 \ \forall i$ can be parametrized as

$$A\mathbf{f} = \begin{bmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_7x_7 & x'_7y_7 & x'_7 & y'_7x_7 & y'_7y_7 & y'_7 & x_7 & y_7 & 1 \end{bmatrix} \mathbf{f} = \mathbf{0}$$

where $\mathbf{f}$ is the column vector containing elements of F in row-major order. Let $\mathbf{f}_1, \mathbf{f}_2$ span the (rank-2) null space of A. Then, for some scalar multiple α, $\mathbf{f} = \alpha\mathbf{f}_1 + (1-\alpha)\mathbf{f}_2$, with the constraint that $\det F = 0$. This constraint forms a cubic equation in α, yielding either 1 or 3 real solutions [45].

For each matrix, the Sampson Cost Function

$$\sum_i \frac{(\mathbf{x}_i'^{\top} F \mathbf{x}_i)^2}{(F\mathbf{x}_i)_1^2 + (F\mathbf{x}_i)_2^2 + (F^{\top}\mathbf{x}_i')_1^2 + (F^{\top}\mathbf{x}_i')_2^2}$$

is used to compute the number of inliers. The fundamental matrix with the most inliers is then converted to a relative pose in terms of position and orientation, using known intrinsic camera parameters. Using a random sample consensus, as well as using 7-point rather than 8-point correspondences, increases resilience to outliers [42].

Iterative Refinement

Let $\mathbf{x}_i, \mathbf{x}_i'$ be measured correspondences, specifically the inliers chosen by a random sample consensus. By triangulating this match into $\mathbb{R}^3$ and re-projecting them onto the two images, a pair of estimated matches $\hat{\mathbf{x}}_i, \hat{\mathbf{x}}_i'$ is computed. As a function of extrinsic camera parameters $\mathbf{p} \in \mathbf{R}^6$—namely, the second camera's translation and rotation components relative to the first's—the geometric distance between re-projected points can be parametrized as

$$\mathbf{f}(\mathbf{p}) = \begin{bmatrix} x_{0,1} - \hat{x}_{0,1} \\ x_{0,2} - \hat{x}_{0,2} \\ x'_{0,1} - \hat{x}'_{0,1} \\ x'_{0,2} - \hat{x}'_{0,2} \\ \vdots \\ x_{N,1} - \hat{x}_{N,1} \\ x_{N,2} - \hat{x}_{N,2} \\ x'_{N,1} - \hat{x}'_{N,1} \\ x'_{N,2} - \hat{x}'_{N,2} \end{bmatrix}.$$

Levenberg-Marquardt (LM) iterations are used to minimize the quantity

$$||\mathbf{f}(\mathbf{p})||^2 = \sum_{i=1}^{N} (x_{0,1} - \hat{x}_{0,1})^2 + (x_{0,2} - \hat{x}_{0,2})^2 + (x'_{0,1} - \hat{x}'_{0,1})^2 + (x'_{0,2} - \hat{x}'_{0,2})^2$$

$$= \sum_{i=1}^{N} d(\mathbf{x}_i, \hat{\mathbf{x}}_i)^2 + d(\mathbf{x}'_i, \hat{\mathbf{x}}'_i)^2.$$

LM optimization seeks to find a step-size Δ for the parameter vector $\mathbf{p}$ at each iteration that satisfies $(J^\top J + \lambda I)\Delta = -J^\top \mathbf{f}(\mathbf{p})$, where J is the Jacobian matrix of $\mathbf{f}$, $J^\top J$ serves as a Hessian approximation, and λ is decreased on iterations where the error is reduced but is increased otherwise. Higher values of λ approach gradient descent, whereas lower values approach Gauss-Newton iteration. This allows optimization to make progress when distant from a solution, and then rapidly approach the solution once the parameters lie within its neighborhood [42]. Finite differences are used in this case to approximate partial derivatives in the Jacobian matrix.

Initial experiments with simulated data showed accuracy within 0.2 degrees and 40 km, which is less than optimal and produced skewed point clouds. However, supplying positional data and only estimating orientation caused estimations to reach < 0.0001 degree accuracy, as LM iterations would no longer converge to local minima or have problems converging on ridges formed by the interdependent parameter space. Supplying accurate positional data on MOCI should not be an issue, with accurate GPS data onboard as well as the option for SGP4 propagation [46].

4.5.3 Feature Matching with Strong Epipolar Geometry

With relatively accurate attitude knowledge, epipolar geometry—the relative geometry between images—can be used to restrict the search space for feature matching for a decrease in both runtime and noise. This concept uses epipolar geometry to constrain matches to the epipolar line, with an allowance of $\pm\varepsilon$, as shown in figure 4.3. This concept was proposed in [12] but not implemented.

In a doubly constrained version, which the author refers to as *strong epipolar geometry*, using earth-centered earth-fixed (ECEF) coordinates, the search space can be restricted along another dimension. The highest given distance from earth's center (the origin in ECEF) to the surface is Mt. Chimborazo at 6384.4 km [47], whereas the lowest distance is earth's semi-minor axis 6356.77 km [48]. Hence, points on the ray from the camera's origin through the queried pixel are only valid within this range. This restriction creates two segments of the ray (in the general case), but the further segment is discarded as it is on the

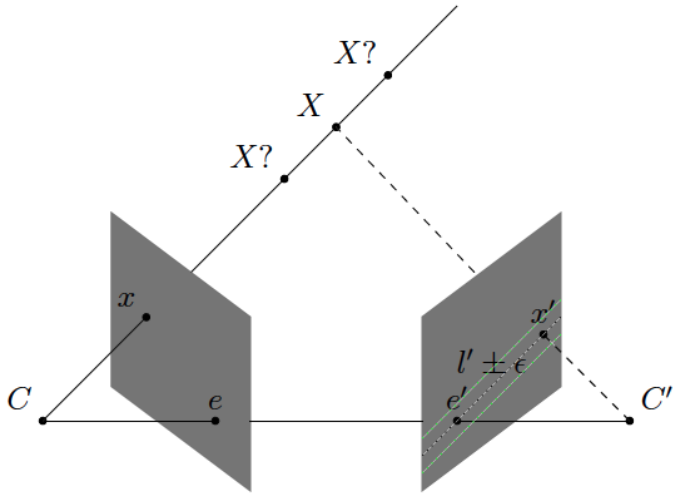


Figure 4.3: Matching constrained to the epipolar line [12].

other side of the earth. Hence, its projection onto the second image is simply a line segment, as shown in figure 4.4.

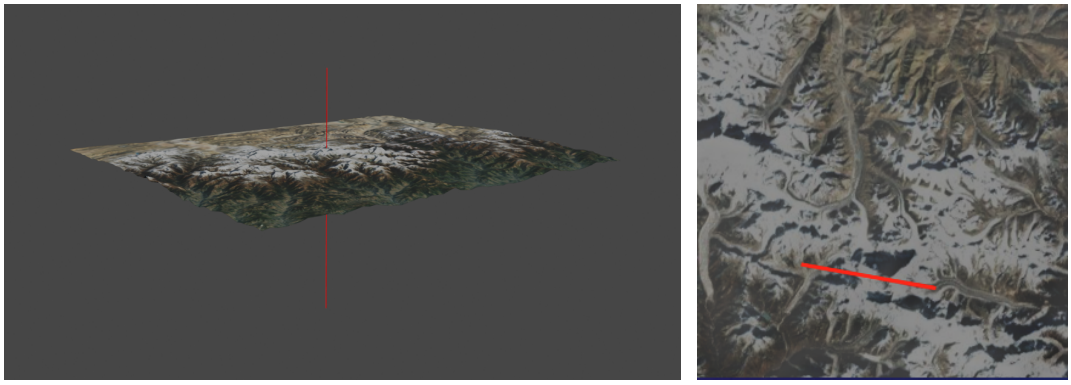


Figure 4.4: Visualization of strong epipolar geometry constraints.

Command-line flags allow for allowances to be set both in terms of pixels around the line segment and kilometers out from the original three-dimensional ray before projection. The default values are set at 25 pixels and 5 kilometers, respectively. A 15 degree view off nadir only changes the distance-to-target by $\sim$ 2 km, so MOCI has the option of assuming the first image was always taken from nadir with the given error allowance.

4.5.4 Triangulation

There are no changes needed in the original triangulation algorithm, due to its low impact on pipeline performance. However, initial point clouds can be saved immediately (without filtering) due to the increase in accuracy.

4.5.5 Filtering

Due to the large decrease in noise afforded by epipolar geometry, the aggressive filtering methods applied earlier start to filter accurate terrain points. Hence, filtering is decreased due to the advantage of downlinking data that can easily be hand-filtered over point clouds with lost data.

4.5.6 Bundle Adjustment

For experimentation, bundle adjustment has been removed from the new pipeline. Since it adjusts the same parameters as done in the pose estimation stage, it is unlikely to find a better solution with more outliers. With checkpointing, this can be enabled at runtime onboard if positional estimates are seen to be inaccurate.

4.6 Experiments and Results

The data used in this section was generated in Blender from the Shuttle Radar Topography Mission (STRM), accurate to 30m [49]. The two targets considered are Mount Everest and Mount Rainier. The two targets are considered the best and worst case scenarios, respectively, in terms of performance, as Rainier's forestry produces over double the features produced by the snow-covered Everest. Most MOCI targets are expected to fall within this range of features.

Table 4.1 shows the accuracy (compared to ground truth) for 2-view 4k reconstructions of Mount Everest and Mount Rainier. Rows with asterisks contain results after manual filtering. Notably, the Everest data produces point clouds with higher accuracy than the original STRM data. Additionally,

the table shows a comparison of results based off of feature matching strategies, revealing that the strong epipolar constraints only leave 9-10 erroneous points, compared to the hundreds produced by a brute force approach.

Rainier data shows lower accuracy, which is likely due to a combination of a larger amount of noise as well as the lower base-height ratio employed on this data set. One image was taken from a nadir-pointing camera on both datasets, whereas the second image was 5 degrees off nadir for Rainier imagery but 10 degrees for Everest imagery.

Table 4.1: Accuracy comparison of 2-view 4096×4096 reconstructions between the brute force and constrained matching strategies.

	Original			Constrained		
Dataset	Average Distance	Std. Deviation	Points	Average Distance	Std. Deviation	Points
Everest	263.451 m	3446.65 m	93,883	37.5961 m	128.426 m	93,215
*Everest	19.0908 m	21.5174 m	93,203	18.7601 m	16.6673 m	93,206
Rainier	337.373 m	17631.1 m	244,350	90.8851 m	84.0431 m	243,590
*Rainier	85.5802 m	71.2258 m	243,965	85.0615 m	69.5612 m	243,580

The point cloud producing the 18.76 average distance to ground truth is shown in figure 4.5. A visual comparison of the noise produced by the two matching strategies is shown in figure 4.6. The outer yellow box (which goes out of frame) includes all noise from the brute force matching strategy, whereas the points produced by epipolar matching are contained closer to the terrain within the smaller yellow box.

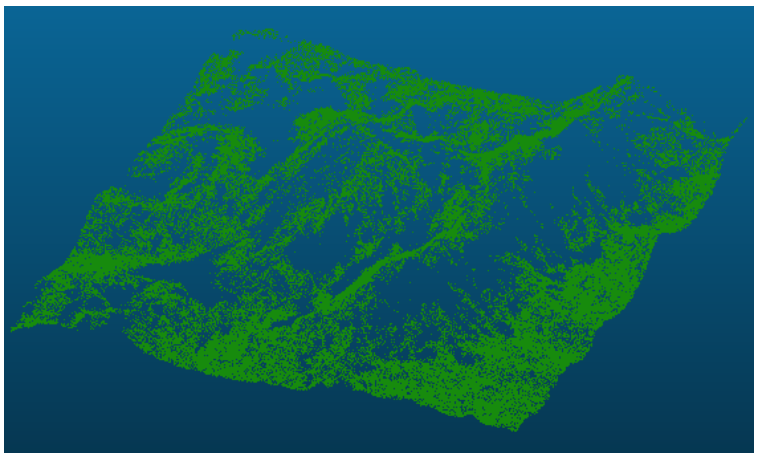


Figure 4.5: Reconstruction of Mount Everest from two 4k images.

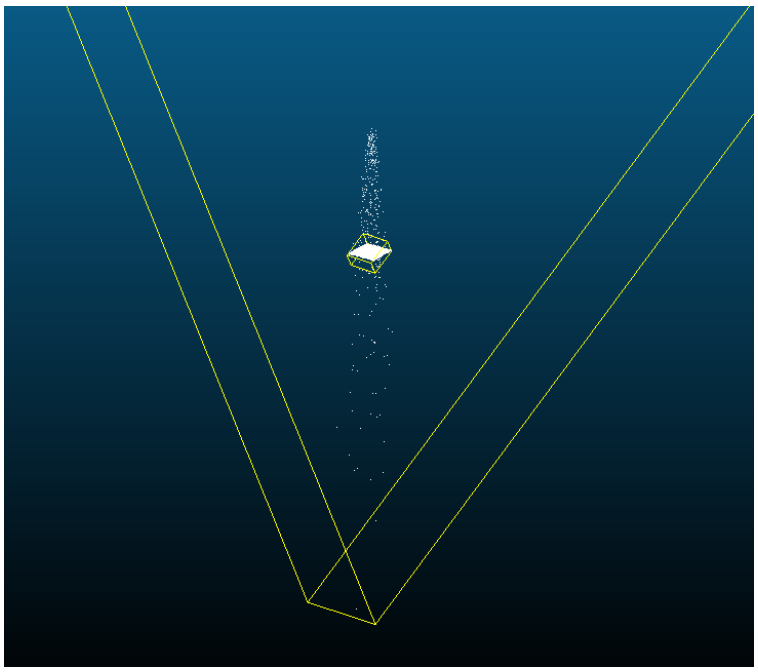


Figure 4.6: Visualization of noise induced by brute-force and epipolar matching strategies.

Runtime comparisons between the original and optimized pipelines are shown in table 4.2. Speedups in feature generation are seen from the separation of the Gaussian kernel, and drastic feature matching speedups are incurred from epipolar constraints. Bundle adjustment was only run for 3 iterations on Everest and 2 on Rainier out of a maximum of 10, due to the locations being known. In both cases, computing pose-estimation for noisy data would be preferred to running a full 10 iterations of bundle adjustment to solve the same problem.

Table 4.2: Runtime comparison of 2-view 4096×4096 reconstructions between the original and optimized pipeline.

Dataset	Pipeline	Feature Gen.	Pose	Match	Triangulation	Filter	B.A.	Total
Everest	Initial	404.522 s	—	2193.598 s	0.774 s	4.218 s	1341.276 s	3944.39 s
	Optimized	204.32 s	510.6 s	119.276 s	1.159 s	2.676 s	—	838.034 s
Rainier	Initial	486.438 s	—	10460.812 s	1.833 s	10.818 s	2317.871 s	13277.819 s
	Optimized	286.892 s	2457.882 s	576.946 s	2.040 s	4.815 s	—	3328.579 s

Peak RAM usage for each stage from the same set of runs is shown in table 4.3. There were no major memory improvements, and in some cases more memory is used in the optimized pipeline. This is likely

due to the fact that, in cases such as the separated Gaussian convolution, reductions in runtime complexity resulted in slight memory complexity increases.

Table 4.3: Peak RAM comparison of 2-view 4096×4096 reconstructions between the original and optimized pipeline.

Dataset	Pipeline	Feature Gen.	Pose	Match	Triangulation	Filter	B.A.
Everest	Initial	3.252 GB	—	0.35 GB	0.346 GB	0.396 GB	0.711 GB
	Optimized	3.2 GB	0.484 GB	0.476 GB	0.507 GB	0.514 GB	—
Rainier	Initial	3.376 GB	—	0.539 GB	0.539 GB	0.656 GB	0.987 GB
	Optimized	3.406 GB	0.667 GB	0.623 GB	0.691 GB	0.699 GB	—

Finally, the energy consumption in Watt-hours is displayed for the same data in table 4.4. Major reductions were seen in both feature generation and matching, correlated with the decrease in runtime, especially for Rainier data. Additionally, note that the initial pipeline's power consumption totals are only in the best-case, as bundle adjustment was run for a minimal number of iterations.

Table 4.4: Energy consumption comparison of 2-view 4096×4096 reconstructions between the original and optimized pipeline.

Dataset	Pipeline	Feature Gen.	Pose	Match	Triangulation	Filter	B.A.	Total
Everest	Initial	0.623 Wh	—	2.61 Wh	0.0007 Wh	0.003 Wh	0.932 Wh	4.169 Wh
	Optimized	0.177 Wh	0.627 Wh	0.196 Wh	0.001 Wh	0.002 Wh	—	1.004 Wh
Rainier	Initial	0.690 Wh	—	12.51 Wh	0.001 Wh	0.008 Wh	1.605 Wh	14.814 Wh
	Optimized	0.248 Wh	3.13 Wh	0.991 Wh	0.002 Wh	0.004 Wh	—	4.374 Wh

4.7 Object Recognition

Object detection is a secondary mission objective on MOCI and uses existing commercial architectures.

4.7.1 YOLOv4 Architecture

You Only Look Once (YOLO) is a state-of-the-art real-time object detection architecture. The third iteration, YOLOv3, primarily uses a backbone of convolutions to perform feature extraction, along with skip layers to reduce the diminishing gradient problem. At three different scales along these convolutions,

a $1 \times 1 \times [3 * (4 + 1 + 80)]$ kernel is applied for feature detection, resulting in three bounding box predictions for every point at that scale, where there are 4 parameters (2-dimensional center and box size), 1 value for the "objectness" of the prediction, and 80 values corresponding to class probabilities on the original dataset. Moreover, the center coordinates of the object are represented as offsets from the cell center, and the width and height of each of the three bounding boxes are represented as offsets from three corresponding anchor dimensions at the given scale. The anchors are priors calculated by k-means clustering on the dataset [50, 51].

Like previous iterations, YOLOv4 is a one-stage detection model, which consists of three main stages past input, shown in figure 4.7. YOLOv4 uses a modified backbone to that employed on YOLOv3, consisting of cross-stage partial connections, weighted residual connections, mish activation, and additional data augmentation. The model's neck consists of spatial pyramid pooling and a path aggregation network. YOLOv3's detector is used for the three prediction stages [52].

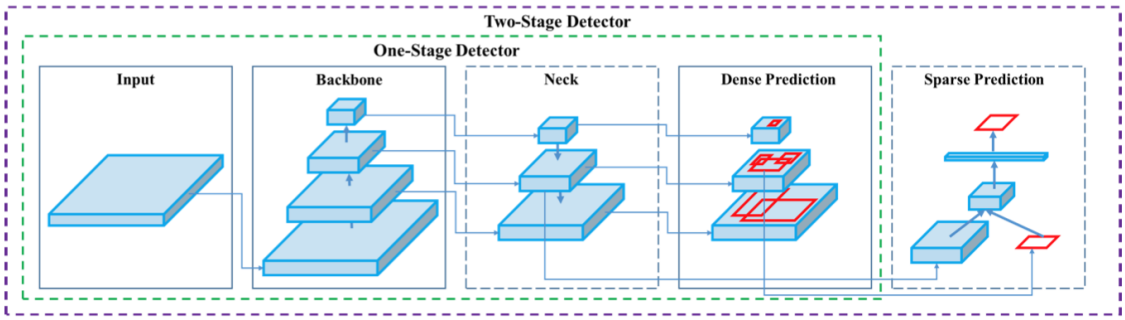


Figure 4.7: One-stage and two-stage detectors [52].

4.7.2 Dataset

For onboard detection of bridges, sports fields, storage tanks, ships, planes, and harbors, the dataset from [10] was used without modification, aside from minor labelling fixes. The dataset includes images from the DOTA and Airbus Ship datasets with bounding box labels. Images were rescaled to 7 GSD to approximately match the resolution of MOCI's color imager.

4.7.3 Training

Training parameters were set to the default options provided by the YOLOv4 developers [52] over 500,500 epochs and using mosaic data augmentation. Due to the small size of objects in the dataset, a necessary pre-training step was to resample anchors by k-means sampling [51]. Any attempts to train without this step led to diverging loss.

4.7.4 Memory, Accuracy, and Runtime Tradeoff

A number of different options were considered for onboard object detection with YOLOv4. The developers provide two versions of the model: the full-sized version, as well as a YOLOv4-Tiny model. The tiny version is a lightweight version, often used for embedded applications, with only two detection layers.

Additionally, the `darknet` executable developed for running YOLOv3, YOLOv4 and similar models offers multiple compilation options for CPU/GPU versions, as well as support for libraries such as OpenCV and cuDNN.

In evaluating these architectures, important factors to consider include runtime, memory, and accuracy. Memory is especially important in this case, as SOL employs a RAM-based file system. Hence, both runtime memory and static storage must be considered. For this reason, the cuDNN- and OpenCV- accelerated options are not considered, as the associated libraries would constantly fill hundreds of megabytes in RAM without use for any other applications.

Table 4.5: Runtime, memory, and accuracy comparison of trained YOLO models for MOCI.

		Static Storage					Performance	
Model Size	Device	Executable	Weights	Total	Runtime Memory	Detection Time	IOU	Recall
Tiny	CPU	0.77 MB	23.7 MB	24.5 MB	167 MB	6.0 s	57.87%	64.13%
	GPU	4.9 MB	23.7 MB	28.6 MB	284 MB	0.7 s	57.87%	64.13%
Full	CPU	0.77 MB	256 MB	257 MB	1161 MB	56.5 s	71.62%	81.98%
	GPU	4.9 MB	256 MB	261 MB	1501 MB	1.0 s	71.62%	81.98%

A full comparison of image inference time, memory consumption, and accuracy are shown in table 4.5. While the YOLOv4-Tiny architecture provides the best inference time and memory consumption for the application, its accuracy is subpar for use on MOCI. This is likely due to a limited data set, and

low sensor data, which can be improved for future missions. Xu *et al*., for instance, achieve 91% recall with YOLOv4-Tiny using SAR imagery [34].

In terms of the full sized model, the GPU-accelerated version provides tremendous runtime and improvements over the CPU version for minimal additional memory overhead. Within the constraints of MOCI, the 261 MB of static storage and 1501 MB of runtime memory are a non-issue, and this architecture will therefore be used on the mission.

Figure 4.8 shows sample detections of ships and sports fields from the custom data set.

Figure 4.8: Sample object detection results.

Chapter 5

Conclusion

The Multiview Onboard Computational Imager (MOCI), the second CubeSat in development by the University of Georgia, intends to capture imagery and perform computer vision pipelines onboard. *In-situ* computation reduces the amount of data that needs to be downlinked, and hence more accurate models can be produced at a lower cost. The primary points of failure for MOCI's payload include radiation-induced device failure, an overuse of available resources, or sensor inaccuracy.

Both low-level and high-level software techniques are employed to remedy these issues. A custom operating system, Space Operating Linux (SOL), is designed to provide system-level fault tolerance to radiation-induced effects and a real-time patch to ensure reliable response to mission-sensitive commands. Radiation tests on the SoC show promising results for CPU and GPU performance in low earth orbit, cementing that former issues were related to peripherals such as flash. This supports the use of a redundant flash scheme and RAM-based file system to reduce flash usage in order to prolong mission lifetime.

On a higher level, optimizations and analyses were conducted in respect to MOCI's computer vision pipelines in order to ensure the system can reliably operate with constraints in runtime and power, as well as memory constraints induced by SOL, and to reduce reliance on potentially faulty sensor data. The mission's custom SfM pipeline was modified to reduce both runtime and power usage by nearly a factor of four. The faster runtime allows for checkpointing to occur at a lower rate, abiding by the practice in SOL of minimizing flash usage. Memory usage was also monitored and determined not to exceed the afforded

RAM, and changes to feature matching strategies and iterative pose estimation allow for high-accuracy models to be produced, regardless of sensor noise. For object recognition, a model was chosen with the highest accuracy to where it could also run within these constraints.

5.1 Limitations and Future Work

5.1.1 Radiation Mitigation

There are still many avenues for future work in radiation mitigation and determining the radiation tolerance of the TX2is. When the modules are shipped back to the SSRL from TRIUMF, the team plans on de-lidding the chips to analyze the spallation area and confirm assumptions that peripherals were not affected.

For future tests, if resources permit, the eMMC should be directly irradiated in order to prove that it is the primary reason for failures seen in previous tests. In such a test, it may be possible to test the redundant memory scheme in SOL against a control.

Such tests may increase support from NVIDIA or other manufacturers in developing custom circuitry. Since the SoC has been shown to be more tolerant than expected to radiation, and the peripherals are especially vulnerable, there has been demand in the space processing community to develop custom boards with rad-hard components around the SoC. While software mitigation does prove to be beneficial, there is only so much that can be done without hardware mitigation.

Finally, MOCI's flight will provide invaluable data to the community on the TX2i's true performance in low earth orbit. While radiation tests and models like CREME-96 allow the community to provide a best estimate of radiation effects, there are some limitations. For instance, the utilities run during radiation tests are generally lower in performance than what would be used onboard, as the high flux under a beam is not realistic as to what a full program would experience. MOCI's flight data will supplement the radiation test presented here and could potentially be used for autonomous fault detection and mitigation algorithms on future satellites.

5.1.2 Pose Estimation

Pose estimation is an essential aspect of the SfM pipeline, as inaccurate relative positioning of the images will skew and distort the point cloud, at best. It is necessary not only to ensure that the positional data from the GPS is accurate, but also to perform edge testing to ensure reliability of the estimated angles. It may be beneficial to provide a geometric ground for the initial estimate, such as the approximate ECEF location of the target. Additionally, the team may opt to provide ADCS-generated attitudes as a failsafe mechanism.

5.1.3 Memory Usage

While erroneous memory usage, such as memory leaks, have been corrected in this work, there have been no attempts to lower the peak memory usage of the pipeline that occurs within feature generation. Future attempts to do so, without compromising the accuracy of the pipeline, would prove extremely beneficial to the fields of SfM and photogrammetry as a whole. This can occur as changes in SIFT hyper-parameters or even a different model, such as FAST or ORB [53, 54].

5.1.4 Scalability

So far, the SfM pipeline has only been tested on simulated satellite imagery of high-relief surfaces. For future research, scaling to additional data types should be considered:

- **Low-relief terrain:** terrain without notable elevation changes could pose two major issues in the current pipeline. The likely possibility is that SIFT will produce a low amount of features, resulting in a sparser point cloud. Another possibility is that many features are produced, but due to similarity across the image, the features are mismatched to form incorrect correspondences between images.
- **Lower-altitude imagery:** future projects within the SSRL call for the use of drone imaging for SfM computation. While this will produce higher resolution images (with lower GSD), problems in accuracy could result from a breakdown in pinhole camera geometry due to surface proximity.

- **Distortion and sensor noise:** since the data used for this research is simulated, the imagery is not subject to real-world issues such as camera distortion and sensor noise. Ideally, the design of the optical train on MOCI should minimize these effects, but they may still occur to a low extent on MOCI and future missions. One option for correcting these is to account for such errors during bundle adjustment, which would then separate the computation done at the end of the pipeline from that done during pose estimation. Hartley and Zisserman also discuss methods to account for projective distortion within the camera model itself [42].

Bibliography

[1] JN Maki et al. "The Mars 2020 Engineering Cameras and microphone on the perseverance rover: A next-generation imaging system for Mars exploration". In: *Space science reviews* 216.8 (2020), pp. 1–48.

[2] Neil Abcouwer et al. "Machine Learning Based Path Planning for Improved Rover Navigation". In: *2021 IEEE Aerospace Conference (50100)*. 2021, pp. 1–9. DOI: 10.1109/AERO50100.2021.9438337.

[3] Daniela Girimonte and Dario Izzo. "Artificial intelligence for space applications". In: *Intelligent Computing Everywhere*. Springer, 2007, pp. 235–253.

[4] Armen Poghosyan and Alessandro Golkar. "CubeSat evolution: Analyzing CubeSat capabilities for conducting science missions". In: *Progress in Aerospace Sciences* 88 (2017), pp. 59–83.

[5] Richard Vuduc and Jee Choi. "A Brief History and Introduction to GPGPU". In: *Modern Accelerator Technologies for Geographic Information Science*. Ed. by Xuan Shi, Volodymyr Kindratenko, and Chaowei Yang. Boston, MA: Springer US, 2013, pp. 9–23. ISBN: 978-1-4614-8745-6. DOI: 10.1007/978-1-4614-8745-6_2. URL: https://doi.org/10.1007/978-1-4614-8745-6_2.

[6] Windy Slater et al. "Total Ionizing Dose Radiation Testing of NVIDIA Jetson Nano GPUs". In: Sept. 2020, pp. 1–3. DOI: 10.1109/HPEC43674.2020.9286222.

[7] Daniel Alfonso Gonçalves Gonçalves de Oliveira et al. "Evaluation and Mitigation of Radiation-Induced Soft Errors in Graphics Processing Units". In: *IEEE Transactions on Computers* 65.3 (2016), pp. 791–804. DOI: 10.1109/TC.2015.2444855.

[8] Gautam D. Badhwar. "The Radiation Environment in Low-Earth Orbit". In: *Radiation Research* 148.5 (1997), S3–S10. ISSN: 00337587, 19385404. URL: http://www.jstor.org/stable/3579710.

[9] *Jetson TX2 Series Module Data Sheet*. https://developer.nvidia.com/embedded/downloads. [Online]. Apr. 2021.

[10] Jackson O Parker. "Design and Implementation of a 6U Cubesat for Low Earth Orbit Computer Vision". MA thesis. University of Georgia, 2020.

[11] Christopher Heistand, Sarah Katz, and Amanda Voegtlin. "NVIDIA Jetson TX2i Radiation Report". Single Event Effects (SEE) Symposium / Military and Aerospace Programmable Logic Devices (MAPLD) Workshop. 2020.

[12] Caleb Ashmore Adams. "High Performance Computation with Small Satellites and Small Satellite Swarms for 3D Reconstruction". MA thesis. University of Georgia, 2020.

[13] Alan D. George and Christopher M. Wilson. "Onboard Processing With Hybrid and Reconfigurable Computing on Small Satellites". In: *Proceedings of the IEEE* 106.3 (2018), pp. 458–470. DOI: 10.1109/JPROC.2018.2802438.

[14] Kenneth A LaBel et al. "Emerging radiation hardness assurance (RHA) issues: a NASA approach for space flight programs". In: *IEEE Transactions on Nuclear Science* 45.6 (1998), pp. 2727–2736.

[15] Edward Wyrwas. *Body of knowledge for graphics processing units (GPUs)*. Tech. rep. 2018.

[16] Caleb Adams et al. "Towards an integrated GPU accelerated SoC as a flight computer for small satellites". In: *2019 IEEE Aerospace Conference*. IEEE. 2019, pp. 1–7.

[17] Fredrik C Bruhn et al. "Enabling radiation tolerant heterogeneous GPU-based onboard data processing in space". In: *CEAS Space Journal* 12.4 (2020), pp. 551–564.

[18] Ian A. Troxel. “CARMA: Management infrastructure and middleware for multi-paradigm computing”. PhD thesis. University of Florida, Jan. 2006.

[19] L. L. Pilla et al. “Software-Based Hardening Strategies for Neutron Sensitive FFT Algorithms on GPUs”. In: *IEEE Transactions on Nuclear Science* 61.4 (2014), pp. 1874–1880. DOI: 10.1109/TNS.2014.2301768.

[20] Tyler Garrett and Alan D. George. “Improving Dependability of Onboard Deep Learning with Resilient TensorFlow”. In: *2021 IEEE Space Computing Conference (SCC)*. 2021, pp. 134–142. DOI: 10.1109/SCC49971.2021.00021.

[21] E. H. Thompson. “A RATIONAL ALGEBRAIC FORMULATION OF THE PROBLEM OF RELATIVE ORIENTATION”. In: *The Photogrammetric Record* 3.14 (1959), pp. 152–157. DOI: https://doi.org/10.1111/j.1477-9730.1959.tb01267.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1477-9730.1959.tb01267.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1477-9730.1959.tb01267.x.

[22] D Marr and T Poggio. “Cooperative computation of stereo disparity”. In: *Appl. Phys* 42 (1971), p. 3451.

[23] David Marr. “Early processing of visual information”. In: *Philosophical Transactions of the Royal Society of London. B, Biological Sciences* 275.942 (1976), pp. 483–519.

[24] S. Ullman and Sydney Brenner. “The interpretation of structure from motion”. In: *Proceedings of the Royal Society of London. Series B. Biological Sciences* 203.1153 (1979), pp. 405–426. DOI: 10.1098/rspb.1979.0006. eprint: https://royalsocietypublishing.org/doi/pdf/10.1098/rspb.1979.0006. URL: https://royalsocietypublishing.org/doi/abs/10.1098/rspb.1979.0006.

[25] H Christopher Longuet-Higgins. “A computer algorithm for reconstructing a scene from two projections”. In: *Nature* 293.5828 (1981), pp. 133–135.

[26] Bill Triggs et al. “Bundle adjustment—a modern synthesis”. In: *International workshop on vision algorithms*. Springer. 1999, pp. 298–372.

[27] Simone Bianco, Gianluigi Ciocca, and Davide Marelli. “Evaluating the Performance of Structure from Motion Pipelines”. In: *Journal of Imaging* 4.8 (2018). ISSN: 2313-433X. DOI: 10.3390/jimaging4080098. URL: https://www.mdpi.com/2313-433X/4/8/98.

[28] Jim Chandler. “Effective application of automated digital photogrammetry for geomorphological research”. In: *Earth Surface Processes and Landforms* 24.1 (1999), pp. 51–63. DOI: https://doi.org/10.1002/(SICI)1096-9837(199901)24:1<51::AID-ESP948>3.0.CO;2-H. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/%28SICI%291096-9837%28199901%2924%3A1%3C51%3A%3AAID-ESP948%3E3.0.CO%3B2-H. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291096-9837%28199901%2924%3A1%3C51%3A%3AAID-ESP948%3E3.0.CO%3B2-H.

[29] Y. Yamaguchi et al. “Overview of Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER)”. In: *IEEE Transactions on Geoscience and Remote Sensing* 36.4 (1998), pp. 1062–1071. DOI: 10.1109/36.700991.

[30] R. Welch et al. “ASTER as a source for topographic data in the late 1990s”. In: *IEEE Transactions on Geoscience and Remote Sensing* 36.4 (1998), pp. 1282–1289. DOI: 10.1109/36.701078.

[31] Hugh Durrant-Whyte and Tim Bailey. “Simultaneous localization and mapping: part I”. In: *IEEE robotics & automation magazine* 13.2 (2006), pp. 99–110.

[32] Ben Mildenhall et al. “Nerf: Representing scenes as neural radiance fields for view synthesis”. In: *Communications of the ACM* 65.1 (2021), pp. 99–106.

[33] Steve Chien et al. “Onboard autonomy on the intelligent payload experiment cubesat mission”. In: *Journal of Aerospace Information Systems* 14.6 (2017), pp. 307–315.

[34] Pan Xu et al. “On-board real-time ship detection in HISEA-1 SAR images based on CFAR and lightweight deep learning”. In: *Remote Sensing* 13.10 (2021), p. 1995.

[35] Austin P Arechiga, Alan J Michaels, and Jonathan T Black. "Onboard image processing for small satellites". In: *NAECON 2018-IEEE National Aerospace and Electronics Conference*. IEEE. 2018, pp. 234–240.

[36] Andrew Jaeyong Choi, Jeonghwan Park, and Jae-Hung Han. "Automated Aerial Docking System Using Onboard Vision-Based Deep Learning". In: *Journal of Aerospace Information Systems* 19.6 (2022), pp. 421–436.

[37] *Yocto Project Reference Manual*. https://docs.yoctoproject.org/current/ref-manual/index.html. [Online].

[38] *Das U-Boot – the Universal Boot Loader*. https://www.denx.de/wiki/U-Boot. [Online]. Nov. 2019.

[39] EW Blackmore, B Evans, and M Mouat. "Operation of the TRIUMF proton therapy facility". In: *Proceedings of the 1997 Particle Accelerator Conference (Cat. No. 97CH36167)*. Vol. 3. IEEE. 1997, pp. 3831–3833.

[40] A.J. Tylka et al. "CREME96: A Revision of the Cosmic Ray Effects on Micro-Electronics Code". In: *IEEE Transactions on Nuclear Science* 44.6 (1997), pp. 2150–2160. DOI: 10.1109/23.659030.

[41] *Cuda C++ Programming Guide*. URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.

[42] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Second. Cambridge University Press, 2003.

[43] David G Lowe. "Object recognition from local scale-invariant features". In: *Proceedings of the seventh IEEE international conference on computer vision*. Vol. 2. Ieee. 1999, pp. 1150–1157.

[44] Tony Lindeberg. "Scale invariant feature transform". In: (2012).

[45] R.I. Hartley. “Projective reconstruction and invariants from multiple images”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16.10 (1994), pp. 1036–1041. DOI: 10.1109/34.329005.

[46] David Vallado and Paul Crawford. “SGP4 orbit determination”. In: *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*. 2008, p. 6770.

[47] Arjun Tan. “Mt. Everest vs. Mt. Chimborazo: Which is Higher?” In: ().

[48] Wen-Hao Yeh et al. “Ray tracing simulation in nonspherically symmetric atmosphere for GPS radio occultation”. In: *TAO: Terrestrial, Atmospheric and Oceanic Sciences* 25.6 (2014), p. 801.

[49] Tom G Farr et al. “The shuttle radar topography mission”. In: *Reviews of geophysics* 45.2 (2007).

[50] Joseph Redmon et al. “You only look once: Unified, real-time object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[51] Joseph Redmon and Ali Farhadi. “YOLOv3: An Incremental Improvement”. In: *arXiv* (2018).

[52] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. “Yolov4: Optimal speed and accuracy of object detection”. In: *arXiv preprint arXiv:2004.10934* (2020).

[53] Deepak Geetha Viswanathan. “Features from accelerated segment test (fast)”. In: *Proceedings of the 10th workshop on image analysis for multimedia interactive services, London, UK*. 2009, pp. 6–8.

[54] Ethan Rublee et al. “ORB: An efficient alternative to SIFT or SURF”. In: *2011 International conference on computer vision*. Ieee. 2011, pp. 2564–2571.