University of Colorado
Department of Aerospace Engineering Sciences
Senior Projects - ASEN 4018
Project Final Report (PFR)

Visual In-Situ Sensing for Inertial Orbits of NanoSats (VISION)

Sunday 22nd March, 2020

Project Customer

Name: Dr Penina Axelrad Email: penina.axelrad@colorado.edu

Group Members

Name: Max Audick Email: maximillian.audick@colorado.edu	Name: Cameron Baldwin Email: caba3252@colorado.edu
Name: Adam Boylston Email: adbo5502@colorado.edu	Name: Zhuoying Chen Email: zhch1699@colorado.edu
Name: Tanner Glenn Email: tagl1811@colorado.edu	Name: Ben Hagenau Email: beha7507@colorado.edu
Name: Adrian Perez Email: adpe2067@colorado.edu	Name: Andrew Pfefer Email: anpf9194@colorado.edu
Name: Ian Thomas Email: iath2777@colorado.edu	Name: Bao Tran Email: tran.tran@colorado.edu
Name: Theodore Trozinski Email: thtr7807@colorado.edu	Name: Mathew van den Heever Email: mava5537@colorado.edu

Contents

Nomenclature

$\mathcal{S}$ = VISION Sensor Frame
$\mathcal{O}$ = Deployer Orbit Frame
$\mathcal{D}$ = Deployer Body Frame
$\mathcal{N}$ = Earth-Centered Inertial Frame

1. Project Purpose

Andrew Pfefer & Ian Thomas

1.1. Problem or Need

The growth of the small satellite industry has given universities and other smaller organizations an opportunity to go to space that otherwise did not exist just a few years ago. This democratization of science and space travel is a result of the drastic drop in material and hardware, as well as the cost to launch. With smaller payloads, a deployer can accommodate more customers and this lowers the cost for everyone. However, this does not come without some inherent problems. Most ride share agreements give NanoSats little guarantee or control of their orbits which is dictated by the main payload. Without the same resources the launch providers give to the main payload, many NanoSats default to a lower priority. This becomes an issue when the NanoSats are deployed with very little orbital determination and are difficult to track and communicate with from ground-based antennas. Some NanoSats are completely lost after deployment which is devastating situation for the organization that paid for the satellite and the individuals who devoted time to design, build, and test their spacecraft. In December of 2018, SpaceX launched the SSO-A SmallSat Express, sending 64 small satellites into orbit. 4 months later, the AirForce had yet to identify 19 of those satellites, leaving the operators unsure of which satellite was theirs[10].

VISION aims to lower the risk of losing small satellites and to track them via ground station within a few orbits after deployment. The team has developed a low-cost, minimally invasive payload that can be integrated onto an existing deployment infrastructure to accurately predict the orbits of deployed NanoSats. The launch providers can then offer this data as a service to customers who would prefer to pay a small upfront investment to ensure the success of their mission.

VISION is a heritage project where the basic premise was developed as a proof-of-concept by a previous team. VISION is continuing the project to a point of a working prototype. This device will house all of the necessary components to complete the mission and each will have a space-grade counterpart for a more space ready iteration of the project. This is crucial to continue the project but is also an important functional feature. The device is designed with the intent of being a multi-platform device capable of interfacing with different CubeSat deployment mechanisms. A finished VISION project would serve as a foundation for larger project with more funding that would be able to advance the TRL of the entire system.

1.2. Previous Work

For satellite operators that rely on TLEs, the accuracy is paramount. There has been significant research and application to improve this. This project is a continuation of the VANTAGE project who successfully created a point of concept for the method in which VISION is modeled. They worked with Dr. Penina Axelrad as well as John Gaebler who specializes in multi-target tracking algorithms[3]. The VISION project is the next stage of the project which incorporates the previous work into a fully autonomous prototype.

The image recognition aspect of the project has had previous work, including by a current member of VISION. Adam Boylston developed an algorithm for centroid recognition that has been used by VANTAGE in their software package[5]. The results of this project are expected to be paramount for further research on centroid recognition moving forward.

The VISION project will also be implementing a non-linear Batch Filter to filter the data for improved propagation accuracy. This form of filtering has been in aerospace applications for years, however this is typically not performed at the undergraduate level. This required trial and error with different techniques until the batch was ultimately selected based on the advise of our customer and advisor.

Depth estimation is also heavily researched as well. There are currently different techniques available including stereoscopic imaging as well as infrared depth detection. All of which is being used in a wide array of commercial products. There has been implementation of this technology in motor vehicles for assisted parking as well as commercial video gaming in the XBox Kinect.

2. Project Objectives and Functional Requirements

Adrian Perez, Andrew Pfefer & Ian Thomas

This section defines the objectives and functional requirements for the VISION project. The objectives were developed in order quantify achievable goals through the duration of the project. The project is a continuation of the VANTAGE

project last year. These goals are built off of the previous work but are exclusive to VISION. These are broken down into varying "Levels of Success" where there are distinct reachable goals for different subgroups of the project. This section also defines the functional requirements by discussing both the overall mission and team concept of operations as well as the functional block diagram.

2.1. Project Objectives

The following table lays out project VISION's levels of success for three different subgroups that ultimately drive the project. This project is a heritage project with structural, software and hardware aspects. The goals of each of these can be seen in the tables where a level 1 will indicate basic satisfactory for the project. Additional levels of success are expected to be reached but are not necessary for project completion.

Structures/Mechanical	
Level 1	• Mission-ready chassis that meets material requirements. • Chassis can interface with one deployer. • All components fit within volumetric and mass constraints of chassis/mechanical structure.
Level 2	• Component Testing in Relevant Environment (Launch and mission environment component testing, TRL-5).
Level 3	• System Testing in Relevant Environment (Launch and mission environment system testing, TRL-6).
Software/Dynamics	
Level 1	• Estimates TLEs of deployed cubesat shaped objects using known deployer state and initial relative position, velocity and attitude.
Level 2	• Estimates TLEs of deployed CubeSat shaped objects using simulated optical sensor data and simulated deployer position, velocity, to a position accuracy of 10 centimeters at a distance of 10 meters. • Assigns identification markers to each individual CubeSat shaped object.
Level 3	• Estimates TLEs of deployed CubeSat shaped objects using experimental optical sensor data and experimental deployer position, velocity, and attitude. • Follows and tracks the assigned marker on each CubeSat shaped object.
Electronics/Sensors	
Level 1	• Interfaces with the deployer's electrical and telemetry systems. • Provides visual verification of successful deployment to simulated deployer interface. • On-board embedded system interfaces and receives data from optical sensors for processing.
Level 2	• Data buffer size downlink/relay data to the deployer in 15 minutes. • Estimates its own single-axis inertial attitude to within 20 degrees [1 standard deviation]. • Stores proof of evidence and processed data on-board for the duration of the data processing and downlink period.
Level 3	• Interfaces with multiple deployers' electrical and telemetry systems. • Estimates its own single-axis inertial attitude to within 15 degrees [1 standard deviation] • Components and system testing in relevant environment (launch and mission environment system testing, TRL-4/5).

Table 1: VISION Levels of Success

Unfortunately, due to the interruption in project development from COVID-19, not all expected levels of success were met. All level 1 objectives under Structures were met through inspection or analysis. While the chassis was never fully finished, missing drilling holes for assembly bolts, the chassis was large enough for all components to fit within, as well as met mass requirements and contained mounting locations designed using the NanoRacks Deployment bay ICD. But because random vibrational testing and system integration testing was never performed, levels 2 and 3 were not met. If testing was not interrupted, using both component and system level testing the VISION team expected to fully meet levels 2 and 3.

The first levels of success for the software and state estimation subsystems were met using unit code functional tests. Using a non-linear batch filter, and known deployer inertial state data, the TLEs could be produced as an output of the software algorithm. Level 2 required TLES to be produced using a combination of data collected from both simulation software and subsystem level testing. Using heritage collected data using the same sensors, and outputs from Blensor and C4D, accurate TLEs were also produced completing the level 2 objectives of this subsystem. But while our testing was interrupted, level 3 objectives were still a stretch as simulation was still needed in order to mimic

accurate behavior of the CubeSat orbits. Therefore level 3 objectives of the software subsystem were not expected to be met.

While all electronic and sensor components were acquired and tested at a component level, functional subsystem and system level testing was halted. This led to only partial completion of level 1 objectives under the Electronics/Sensors subsystem. Using a single power source, and a Raspberry Pi, integration with the mock deployer could be demonstrated. If testing was not halted, using the system integration test, Visual Verification, Data Buffer size and down-link, storage space are all expected to have been met if the components worked within specification and successfully met test requirements.

Below in table 11 are the functional requirements used to derive the design requirements, as well as dictate the verification process needed to meet the customers requirements. These were derived from the levels of success and customer requirements. In the following sections we will show how they were flowed down into design requirements and how the system met or would have met them.

Req. ID	Req. Text	Rationale
FR-1	The tracking system shall observe 6 separate CubeSats from the deployment platform.	The customer wants to be able to track an entire deployment from one NanoRacks bay. Currently a NanoRacks bay is 6U. The maximum number it can deploy is 6, 1U CubeSats.
FR-2	VISION shall report Two-Line Element (TLE) estimates of deployed CubeSats.	The customer wants the orbital estimation in the form of a Two-Line Element (TLE) because it is a generally accepted form of orbital expression in industry. Furthermore, the TLE will be the result of data processing and therefore requires much less bandwidth to downlink from the launch provider to a ground station than the entirety of captured the data from VISION. TLEs would allow a more timely and accurate ground tracking of individual CubeSats, decreasing the likelihood of loosing any after deployment.
FR-3	VISION shall report proof of deployment.	Proof of deployment is required in order to make VISION commercially viable. Future launch customers want verification of a successful launch, proof of deployment will show launch customers the initial status of their product.
FR-4	VISION shall integrate the functionality of both software and hardware within a single package.	The customer wants a fully autonomous package. To deliver this the system must be fully integrated into a single package.
FR-5	The system shall integrate with a deployment system defined by an Interface Control Document (ICD).	Integration with the deployment system is critical for operation of VISION. VISION must be able to communicate with the launch deployer to report TLE and proof of deployment as well as receive power from the launch deployer.
FR-6	Components within VISION shall be space-grade or interchangeable with comparable space-grade components.	As required by the customer, the TRL of VISION needs to advance to near or complete space-readiness, equivalent to TRL-4 or TRL-5. However, due to budgetary constraints, it is unlikely that all components can be made to be space-grade. Therefore, as much as possible, VISION will be space-grade, or TRL-4 equivalent, and for expensive components, a comparable component will be used.

Table 2: Functional Requirements

2.2. Concept of Operations

Figure 1 shows the full Concept of Operations (CONOPS) for the VISION project. As noted in the key, the purple color denotes operations that VISION will handle whereas the magenta color is what a deployment vehicle would handle and is outlined in a team Interface Control Document (ICD). The deployer would handle launch in order

to get themselves and out interfaced payload into Low Earth Orbit. From there VISION will stay dormant until it receives a boot command from the deployer signaling an upcoming deployment of CubeSats. As the CubeSats are deployed, VISION will observe them with its sensor suit and collect data. It then processes the data and packages it into the universal data type known as a Two-Line Element (TLE). VISION then delivers the TLEs to the deployer who downlinks it to the ground station where it can be disseminated to tracking stations. This will allow for faster, more accurate tracking of these CubeSats and will lower the risk that they are lost in the deployment process.

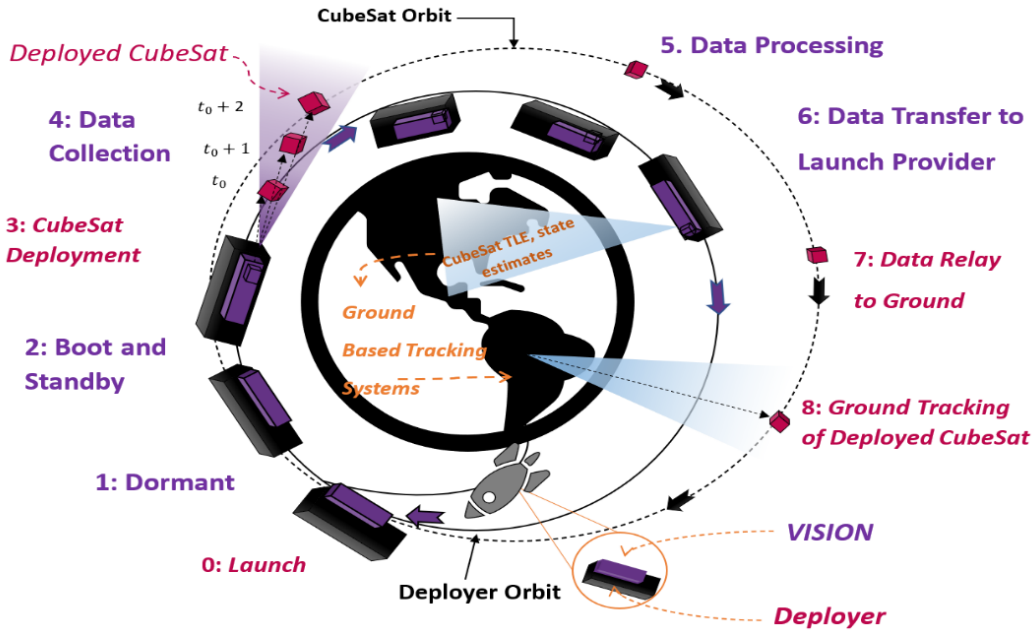

Figure 1: Project CONOPS

However, since the project will not have the opportunity to test this in a real space environment, VISION has developed a team CONOPS for this school year. The team CONOPS can be found below in figure 2. Looking first at the purple dotted area, we see the main functionality of VISION. This is are the software functions that provide the TLE deliverable. For the purposes here, we can simplify by saying that the heritage project, VANTAGE, provides the ”centroid identification” function. VISION then takes these and uses a state estimation filter to vastly improve the accuracy of these measurements. It then calculates the orbital parameters and packages them into a TLE.
In order to verify this process, the team has two separate avenues. On the top we have software verification which uses photo-realistic camera simulation software to replicate deployment of CubeSats in a space environment. This data is then passed to the VISION system software which will deliver TLEs. The TLEs are then compared with the actual orbital conditions inputted into the simulation to verify the accuracy of the software. The second avenue is a physical system integration test which uses the physical system hardware to collect data. This will verify the integration of the entire system as well as allow for actual data to be used to tune the software and make it even more accurate.

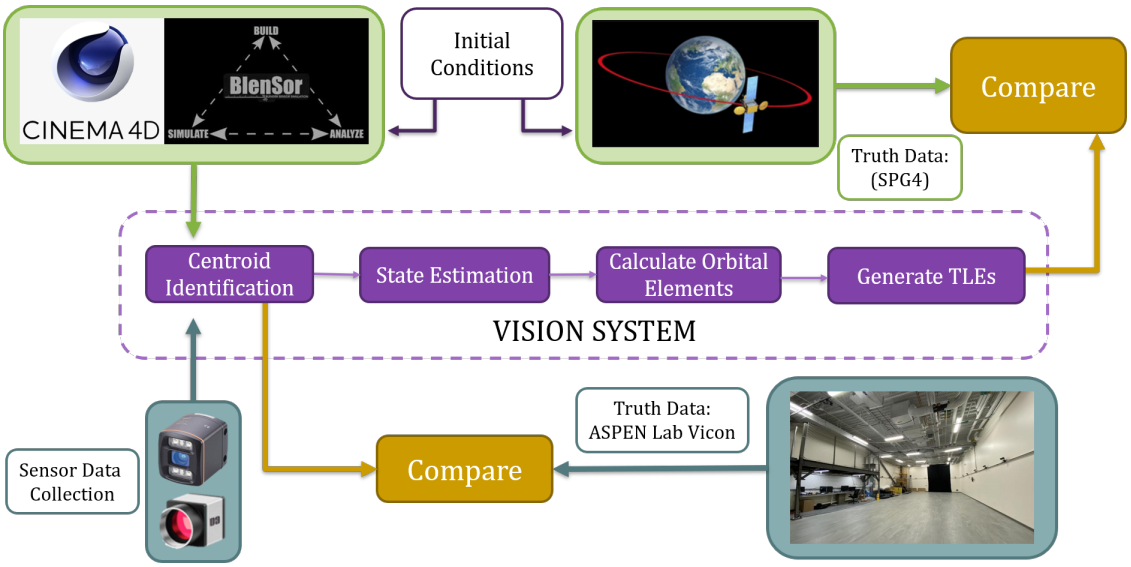

Figure 2: Team CONOPS

Figure 3 is the VISION system functional block diagram. This diagram shows how the system will achieve the functionality described above in the project and team CONOPS. Everything contained within the red barrier is a part of the VISION system. The blocks within the system are color coded to denote which components were acquired from the heritage project, procured, or developed. Going further into detail we will explain this diagram starting in the bottom right where, externally, CubeSats are deployed. The system uses a TOF and Monochrome camera to capture data of the deployment. These two pieces of hardware were inherited from VANTAGE. This data is then sent to heritage software which calculates the position of the centroids of the CubeSats at each observation. This data, along with data from the GPS receiver that we procured, is then processed through VISION-developed software to estimate the state and derive a prediction of their orbit which is then synthesized into a TLE. These TLEs as well as select pictures are then sent through a software interface to the deployment provider. The software is run on a procured processor and the entire system is powered by a developed printed circuit board with acquired hardware. External to our system, the deployment provider can communicate our deliverables to the ground.

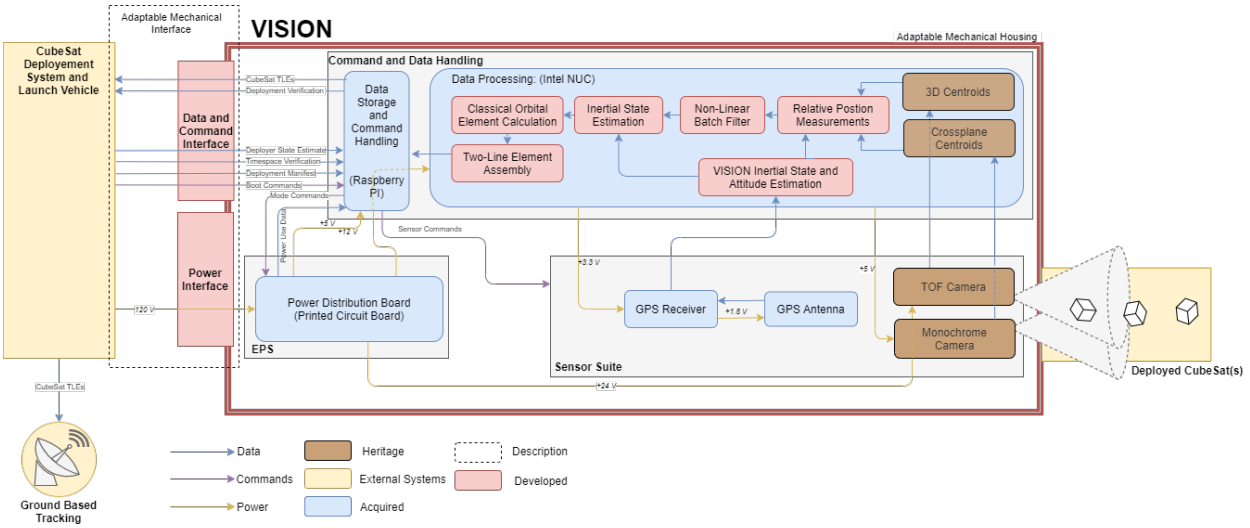

Figure 3: Functional Block Diagram

2.3. Deliverables

The VISION team must deliver products required by the customer and the ASEN 4028 class. Our customer, Professor Axelrad, has requested an integrated protoype of a near space-ready (or space-grade component interchangeable) system that will deliver Two-Line Element estimations and photographs serving as proof of deployment. The team will deliver an autonomous system including the physical system and all supporting software and test equipment. The ASEN 4028 requires deliverables throughout the 2 semester project duration. These include reports and presentations on the progress of the project to be reviewed by the Project Advisory Board (PAB). All of these deliverables will be completed to the standards provided by the faculty and delivered in a professional manner.

In light of the COVID-19 crisis, the team will not be able to complete all customer deliverables due to the closed labs. Instead, the team will try to explain what we would have been done and what the results were expected to be. We will also provide all work to allow future progress to continue on the project.

3. Design Process and Outcome

Adam Boylston, Ben Hagenau, Max Audick, Theodore Trozinski, Cameron Baldwin, Bao Tran,& Zhuoying Chen

3.1. Structures

3.1.1. Overview

The chassis framework (structure) must be able to incorporate and condense components from all other systems vital to the mission into a single package. These components also require thermal management to ensure successful operation. For comprehensive mission success, the chassis must be compatible with all system components, have them fit within the volumetric constraint of the design, and survive the launch environment without structural failure. The chassis is also designed to allow for modularity when interfacing with multiple deployers.

3.1.2. Design Requirements

Req. ID	Req. Text	Rationale
DR-4.1	The chassis envelope shall enclose all components excluding protruding instrument sensors.	The testing and operation of the system requires an integrated hardware and software package to allow for end-to-end testing.
DR-5.1	The chassis shall mechanically integrate with one deployment system as defined in the Mechanical Interface Control Document (MICD).	The customer has explicitly stated that the system needs to mechanically interface with at least one deployment system
DR-5.2	The system shall have a mass of less than 8 kg.	The system needs to be able to meet the mass requirements of a 6U CubeSat in order to ensure interfacing is possible in the case of an internal or modular,interfacing design.
DR-5.3	The system shall have dimensions less than 6000 cm3.	The system needs to be able to meet the mass requirements of a 6U CubeSat in order to ensure interfacing is possible in the case of an internal or modular,interfacing design. The system needs to meet the volumetric requirements of a 6U CubeSat in order to ensure interfacing is possible in,the case of an internal or modular interfacing design.
DR-6.1	Chassis structure shall reach an advanced TRL certification, or equivalent.	This is a requirement of the customer that the system advances in TRL from the 2018-2019 VANTAGE project. An advanced TRL certification or equivalent is a feasible goal with VISION's given budgetary, time, and resource constraints.
DR-6.2	Chassis structure shall maintain structural integrity, within a safety factor of 2.	The chassis structure must maintain structural integrity in order to perform,mission operation and meet the TRL, or equivalent rating. A factor of safety,of 2 was chosen as this is general engineering practice.

3.1.3. Trade Studies

There was only 1 trade study performed for structures. This related to where on the deployer VISION would interface. There were 4 configurations considered. The first was an internal mounting, where VISION would be integrated inside a launch tube, replacing the CubeSats. The second was an external mount, with VISION attached to the outside of the deployment tube. The third was a complete tube replacement, where VISION would replace an entire deployment tube. This was the configuration the 2018-19 team chose. The final option was a modular configuration, where VISION could mount both externally and internally.

Structural Interfacing Method Trade Study Table

		Internal			External			Tube Replacement			Modular		
Metric	Weight	Performance	Score	Points	Performance	Score	Points	Performance	Score	Points	Performance	Score	Points
Required Volume	10%	5	0	0	10	10	1	10	10	1	5	0	0
Tolerancing/Manufactu	5%	3	0	0	5	10	0.5	5	10	0.5	4	5	0.25
Development Time	20%	8	10	2	3	2.86	0.57	4	4.29	0.86	1	0	0
Universality	20%	9	8.57	1.71	4	1.42	0.28	3	0	0	10	10	2
Structural Integrity	15%	8	6.67	1.00	8	10	1.5	8	10	1.5	6	0	0
Opportunity Cost	15%	2	1.11	0.17	10	10	1.5	1	0	0	6	5.56	0.83
Thermal Management	15%	2	0	0	10	10	1.5	10	10	1.5	7	6.25	0.94
Wt. Total	**100.00%**			**4.88**			**6.86**			**5.36**			**4.02**

Figure 4: Structural Interfacing Method Trade Study Results

The winner of the trade study was the external integration method. This allows for liberal volumetric constraints, low development time, high universality, and low opportunity cost. With this in mind, the chassis was also designed to fit within a 6U deployment tube as a contingency.

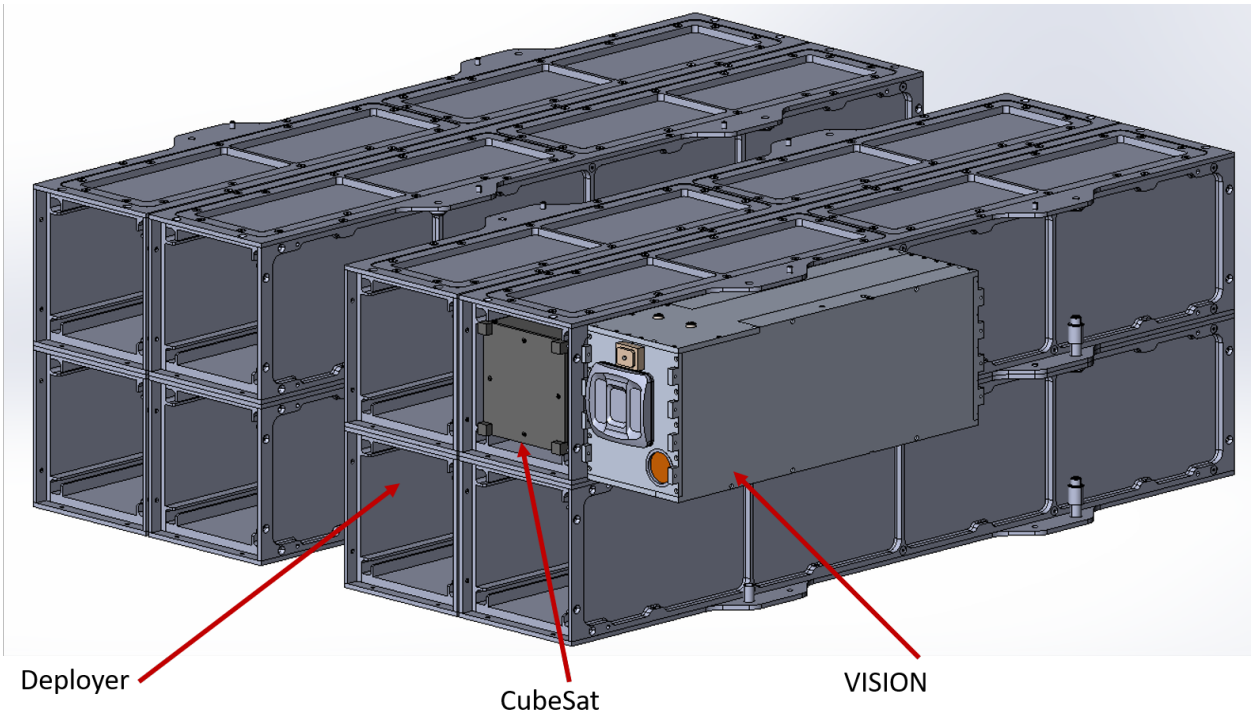

Figure 5: Final Design Choice

Several key options for the physical design of the structural chassis were considered. Each design alternative was evaluated for its performance of the assigned metrics. The overall best design option is an externally mounted chassis. This design is decidedly much more versatile and could be incorporated with multiple launch providers without as much design change as the Tube replacement design. Furthermore, the external mount would not take up payload space for deployment providers, which makes it more attractive to possible partners. The second highest design option is the tube replacement option but it falls short in the opportunity cost section as it would replace up to 6 1U CubeSat customers. Internally mounted is the third-best option. This design would be limited in both the amount of volume that would usable, and the tolerances that the system would have to meet. The system would be inside the deployment provider and, therefore, would have limited ability to use a radiator to manage its temperature. The least performing option is a design that could mount internally or externally. This design mainly draws the most negative aspects as the other options as it shares their limits. The main negative aspect is the development time which is a combination of the other designs. This alternative would carry the most risk and complexity and is eliminated from consideration.

3.1.4. Design Results

The VISION chassis has dimensions of 36.63 x 13.8 x 11.1 cm. This gives a total volume of 5611 cm^3. The material selected was Al-6061 as it is lightweight, inexspensive, and meets our material requirements. This brings the total mass of the chassis to 2.14 kg.

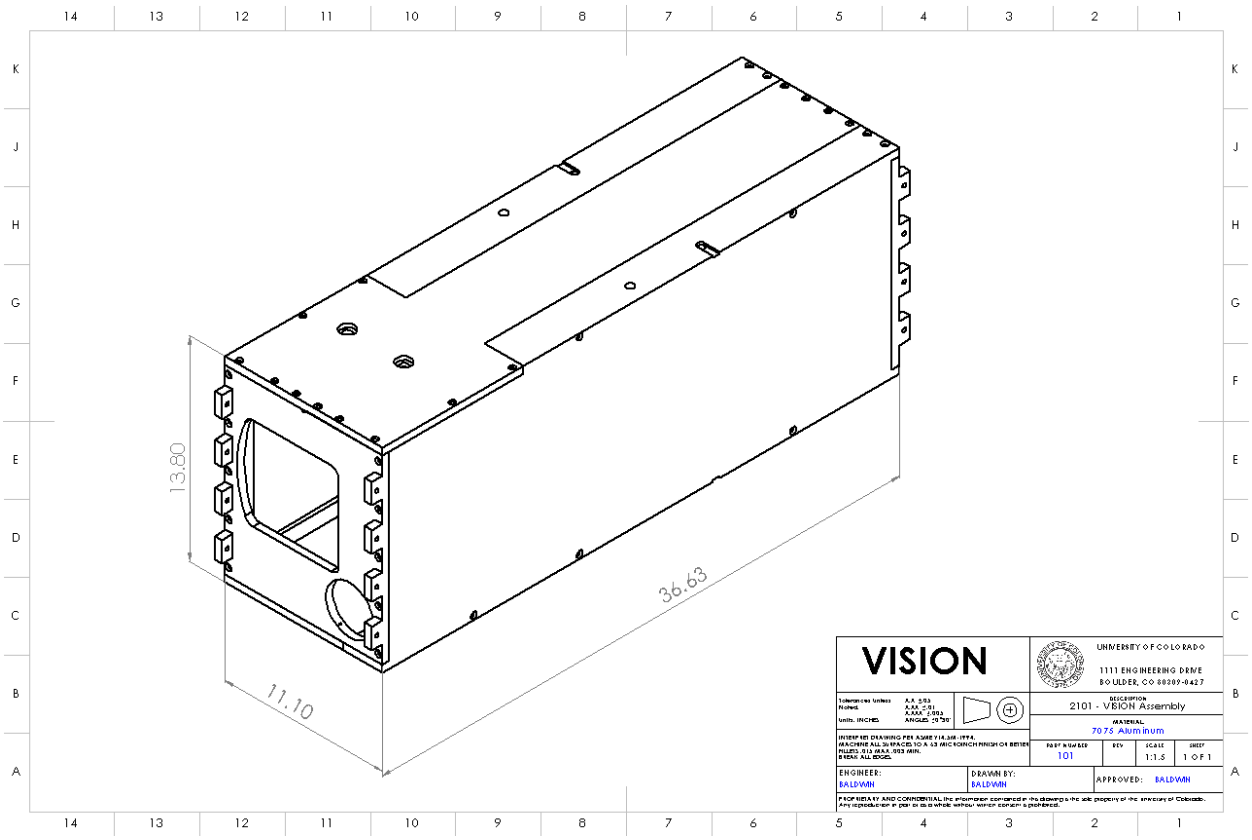

Figure 6: Basic Dimensions

The VISION chassis was designed to be made of 6 separate components. This was to allow for manufacturability. The CAD and manufactured pieces can be seen below in figures 7 and 8.

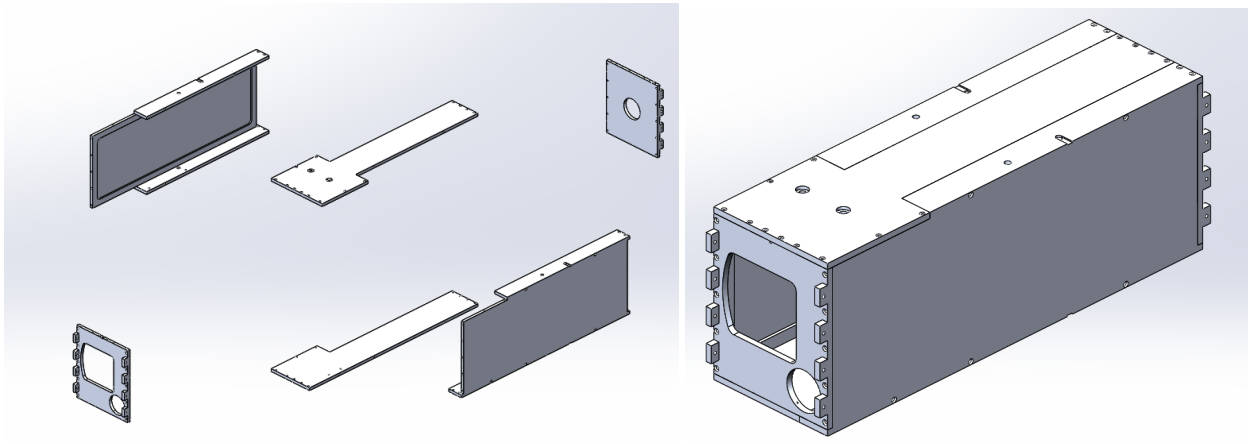

Figure 7: Chassis Exploded & Collapsed Views

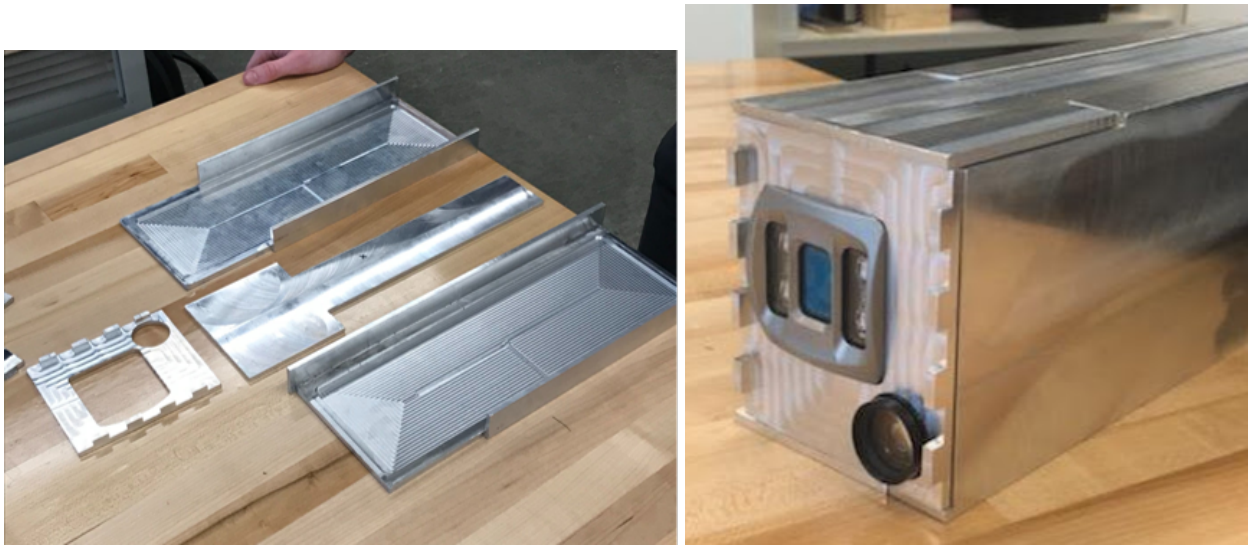

Figure 8: Chassis Components & Fit Test

The chassis has multiple integration configurations for modularity when interfacing with deployers. These are circled in red in figure 9.

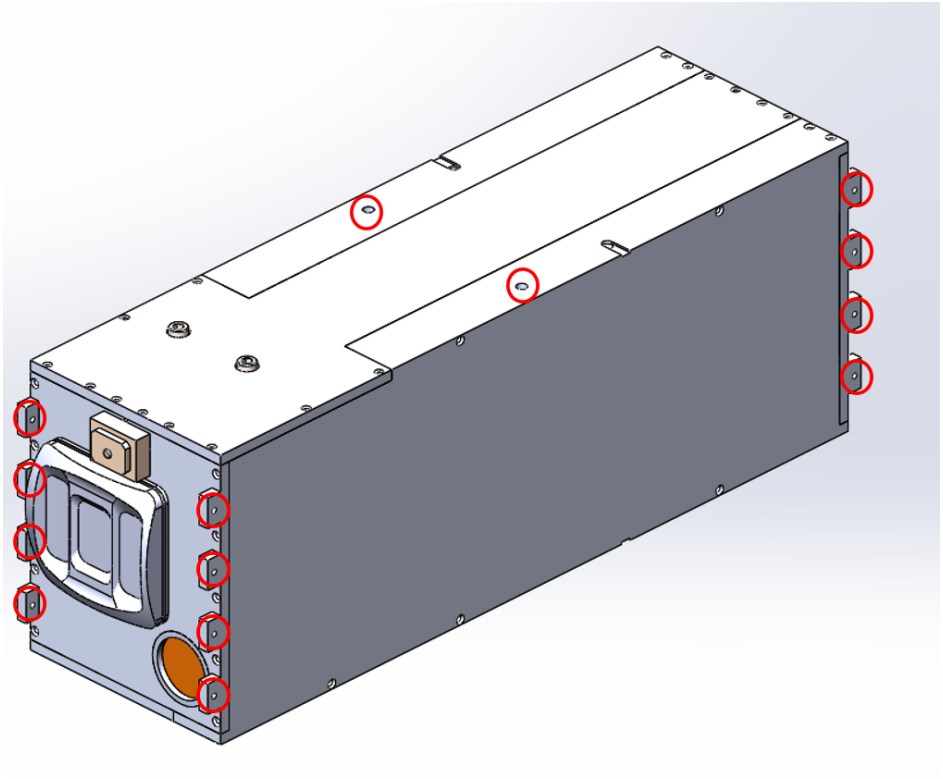

Figure 9: Integration points

When assembled, the chassis would enclose all internal components, excluding the sensors. A fully integrated VISION package is shown below in figure 10.

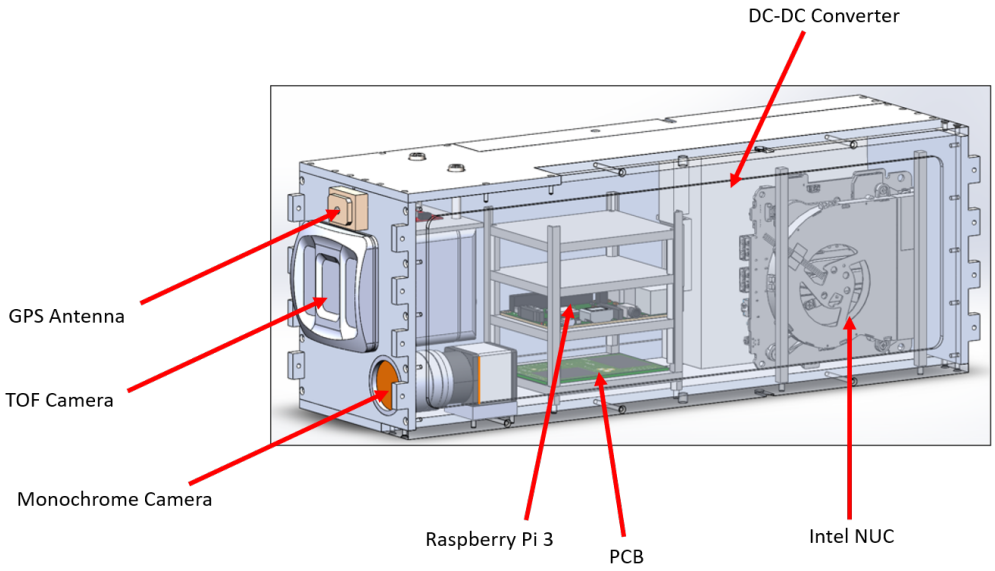

Figure 10: Internals

3.2. Electrical Systems

3.2.1. Overview

The main objective of the electrical subsystem is to satisfy the integration and interfacing requirements. This requires voltage distribution within the chassis as well as compatibility with the deployer's electrical and data interfaces. VISION was designed based on NanoRacks' ICD for power and data parameters. The power is received as a single input and is then conditioned to meet VISION's hardware requirements.

3.2.2. Design Requirements

1. **FR.4:** VISION shall integrate the functionality of both software and hardware within a single package.

   The design requirements derived from the functional requirement 4 are shown in Table.3. In order to meet FR-4, VISION must be able to operate under the supplied power from NanoRacks' deployers. Besides, VISION should be able to store all the collections and results to process and transfer to the deployer.

Functional Requirement ID	Design Requirement ID	Design Requirement
FR-4	DR-4.2	The system shall operate with no more than 120VDC, 3 Vpp ripple voltage, and 5A.
FR-4	DR-4.3	The system shall draw no more than 520 Watts.
FR-4	DR-4.4	The system shall store images, raw data, and estimates of one deployment cycle, on-board, for the duration of the data processing and down-link period.

Table 3: Electrical Design requirements

2. **FR.5:** The system shall integrate with a deployment system defined by an Interface Control Document (ICD).

   The design requirements derived from the functional requirement 4 are shown in Table.4. In order to satisfy the functional requirement, the electrical subsystem should be able to interface with the deployer to transfer the estimated results.

Functional Requirement ID	Design Requirement ID	Design Requirement
FR-5	DR-5.4	The electrical power distribution subsystem shall interface with a PC. Note: This interface will simulate all data and power communications between a potential deployment system.

Table 4: Electrical Design requirements

3. **FR.6:** Components within VISION shall be space-grade or interchangeable with comparable space-grade components.

   The design requirements derived from the function requirement 6 are shown in Table 5. In order to satisfy the function requirement, the electrical components in VISION should be compatible with space-rated hardware so that it can easily change to space-rated project in the future.

Functional Requirement ID	Design Requirement ID	Design Requirement
FR-6	DR-6.3	The electrical components shall have similar protocols and functions as space,rated components.

Table 5: Electrical Design requirements

3.2.3. Trade Studies

Embedded System

This project is software heavy and the main function is to take images from sensors and process them to estimate the TLE's. Therefore, the success of this project highly depends on the performance of the embedded system. Besides, the system needs to provide proof of deployment by the customer's request. Therefore, it shall be able to store multiple images and videos of at least one deployment cycle to be able to transmit to the deployer. The evidence with high quality requires large amount of memory space.

Moreover, since VISION is a closed package, heat generation is also an important consideration to keep the system work under operating environments. Besides, the embedded system is expected to be one of or even the greatest power draw for the project payload. While NanoRacks has informed the team that 520 W is the maximum allowable power draw. This relatively low weight reflects the team's large power budget. Another major requirement for this project is to be ready for the flight experiment by customer's request. So the embedded system needs to be space rated. But the team is not able to purchase space rated hardware due to the limited budget. Therefore, interchangeability with space rated hardware is highly weighted.

Embedded System Trade Study Table

		Nividia Jetson Nano			Intel NUC8I7BEH			Intel NUC8I7HNK			Intel NUC8I7HVK		
Metric	Weight	Performance	Score	Points	Performance	Score	Points	Performance	Score	Points	Performance	Score	Points
Integration	15%	~ 4 [Weeks]	5	0.75	~ 2 [Weeks]	7.5	1.13	~ 2 [Weeks]	7.5	1.13	~ 2 [Weeks]	7.5	1.13
Performance	30%		5.4	1.62		4.566666667	1.37		6.8	2.04		6.8	2.04
RAM	10%	4 GB	1.3	0.13	32 GB	10	1.00	32 GB	10	1.00	32 GB	10	1.00
GPU Rank	10%	240	4.9	0.49	293	3.7	0.37	116	7.5	0.75	98	7.9	0.79
Benchmark CPU	10%	5732	10	1.00	3662	0	0.00	4271	2.9	0.29	4181	2.5	0.25
Storage Capabilities	10%	16 GB	1.3	0.13	>128 GB	10	1.00	>128 GB	10	1.00	>128 GB	10	1.00
Heat Generation	15%	10 W	10	1.50	28 W	8	1.20	65 W	3.9	0.59	100 W	0	0.00
Power Consumption	10%	20 W	9.2	0.92	78 W	7	0.70	230 W	1.2	0.12	212 W	1.8	0.18
Interchangeability	20%		2	0.40		6	1.20		6	1.20		6	1.20
Wt. Total	100.00%			5.32			6.60			6.07			5.55

Figure 11: Embedded System Trade Study Part 1

Embedded System Trade Study Table										
		Intel NUC8i7BEH			Intel NUC7i7DNHE			Intel NUC7i7DNBE		
Metric	Weight	Performance	Score	Points	Performance	Score	Points	Performance	Score	Points
Integration	15%	~ 2 [Weeks]	7.5	1.13	~ 2 [Weeks]	7.5	1.125	~ 2 [Weeks]	7.5	1.125
Performance	30%		4.566666667	1.37		0.966666667	0.29		2.533333333	0.76
RAM	*10%*	*32 GB*	10	*1.00*	*16 GB*	5	*0.50*	*32*	10	*1.00*
GPU Rank	*10%*	*293*	3.7	*0.37*	*390*	1.7	*0.17*	*396*	1.5	*0.15*
Benchmark CPU	*10%*	*3662*	0	*0.00*	*2875*	-3.8	*-0.38*	*2842*	-3.9	*-0.39*
Storage Capabilities	10%	>128 GB	10	1.00	>128 GB	10	1.00	>128 GB	10	1.00
Heat Generation	15%	28 W	8	1.20	15 W	9.4	1.41	15 W	9.4	1.40
Power Consumption	10%	78 W	7	0.70	65 W	7.5	0.75	65 W	7.5	0.75
Interchangeability	20%		6	1.20		6	1.20		6	1.20
Wt. Total	100.00%			6.60			5.78			6.24

Figure 12: Embedded System Trade Study Part 2

At the beginning of this project, the team had four options, Nividia Jetson Nano and three Intel NUC models. The chosen option from the Fig.11 is Intel NUC8i7BEH. However, the team forgot to consider the dimensions of the processing unit and the selected Intel NUC8i7BEH exceed the dimensions of the structures. Since the team had started working on the Intel NUC kit about interfacing with different sensors and communicating with the control unit, the team decided to consider the Intel NUC board instead because of their smaller dimensions. In the end, by using the same metric, the NUC7i7DNBE get 6.24 as the second place of all the options and even better than the previous other two Intel NUC model in part one. Therefore, the team decided to use Intel NUC7i7DNBE as the final product.

Data Collection

The primary sensors on VISION observe the CubeSats directly to determine the relative position and velocity of the CubeSats as they move away from a deployer. The heritage project VANTAGE used a Time of Flight (TOF) Camera and a Monochrome Camera. However, new requirements made it necessary to determine whether or not sensors provided were still adequate.

During the preliminary design phase of the project, a trade study was performed to weigh the pros and cons of numerous sensor sets for this primary array. Six total options were considered. Each system has its pros and cons, impacting other aspects of the project greatly.

The first possible configuration, shown in figure 13, is based on the legacy equipment from VANTAGE. It uses data from one wide angle optical sensor and a Time of Flight (TOF) camera. A TOF camera uses pulses of Infrared (IR) light to determine the distance to an object based on the time the IR particles take to leave the sensor, bounce off the object and return to the sensor. The wide angle optical camera will use visual data to augment the data from the TOF camera to provide a more accurate estimate. This system is easier to implement, as the work done by VANTAGE significantly lowers its development time. However, the wide angle visual camera provides little help for observations beyond the first few meters. The TOF camera, however, has a high power draw, increasing the heat produced by the system and the overall power draw from the deployer.

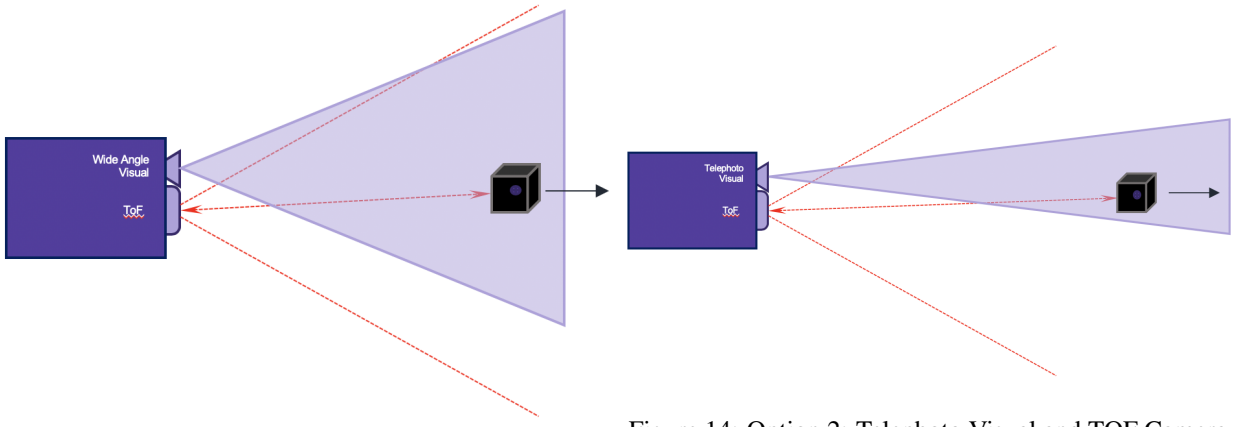

Figure 13: Option 1: Wide Angle Visual and TOF Camera

Figure 14: Option 2: Telephoto Visual and TOF Camera

Next, a telephoto camera is considered in conjunction with the TOF. This way, the visual camera is able to better complement the short term accuracy of the time of flight. This approach is illustrated in figure 14. A narrower FOV will allow the sensor to observe the CubeSats for a longer time, but the observation will begin later, as the CubeSats

will have to travel into the FOV of the camera. This also makes the system sensitive to overlap, which may limit the more accurate observations to just the trailing CubeSat. The zoom camera approach will require a similar development time to the wide angle/TOF approach used by the VANTAGE team, as the software differences are minimal.
The next system used two monochrome cameras and a time of flight camera. This option provides the best visual coverage, as displayed in figure 15, combining excellent initial wide angle differentiation of CubeSats with the longer narrow FOV observation.

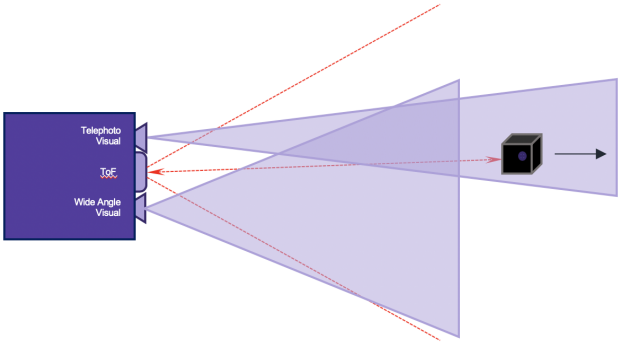

Figure 15: Option 3: Wide Angle Visual, Telephoto Visual and TOF Camera

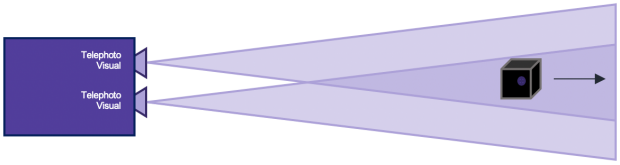

Figure 16: Option 4: Two Stereoscopic Visual Cameras

The final three options sacrifice the accuracy of the TOF Camera in order to lower the overall power draw. The first, is a stereoscopic setup shown in figure 16, where the different angles of an object in view are used to determine position relative to the sensor. The second is just a single monochrome sensor, which would save space and power, but with the worst performance. This system is seen in figure 17. The final system uses two staggered monochrome cameras to lower power draw, while capturing the CubeSats both near and far.

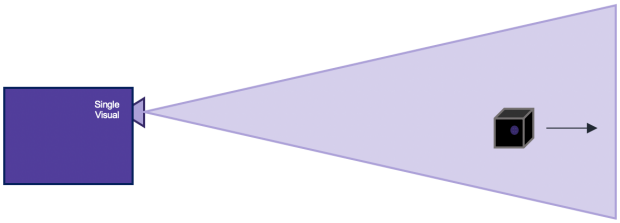

Figure 17: Option 5: Single Visual Camera

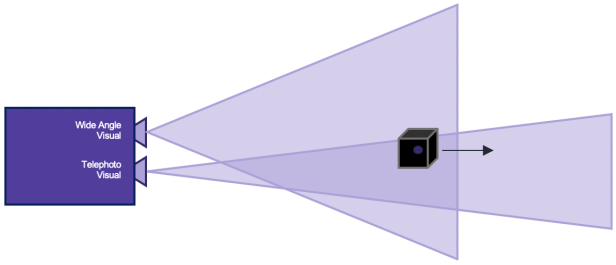

Figure 18: Option 6: Staggered Dual Visual Cameras

The results of this trade study are shown below in figures 19 and 20, where the approach used by VANTAGE is seen to be the best option continuing forward.

Sensor Suite Trade Study Table (Part 1)

		Wide Angle Visual and ToF			Telephoto Visual and ToF			Dual Visual with ToF		
Metric	Weight	Performance	Score	Points	Performance	Score	Points	Performance	Score	Points
Volume [cm^3]	15%	500	2	0.3	500	2	0.3	600	0	0
Power [W]	15%	9	3	0.45	9	3	0.45	12	0	0
Short Distance Accuracy (<10m)	30%	10	10	3	4	4	1.2	10	10	3
Long Distance Accuracy (>10m)	15%	6	6	0.9	10	10	1.5	10	10	1.5
Implementation Time	25%	100	10	2.5	100	10	2.5	150	7.5	1.875
Wt. Total	**100.00%**			**7.15**			**5.95**			**6.38**

Figure 19: Sensor Suite Trade Study Results (Part 1)

Sensor Suite Trade Study Table (Part 2)										
		Stereoscopic			Single Visual			Staggered Dual Visual		
Metric	Weight	Performance	Score	Points	Performance	Score	Points	Performance	Score	Points
Volume [cm^3]	15%	200	8	1.2	100	10	1.5	200	8	1.2
Power [W]	15%	10	2	0.3	2	10	1.5	4	8	1.2
Short Distance Accuracy (<10m)	30%	0	0	0	6	6	1.8	6	6	1.8
Long Distance Accuracy (>10m)	15%	0	0	0	3	3	0.45	5	5	0.75
Implementation Time	25%	300	0	0	150	7.5	1.88	160	7	1.75
Wt. Total	**100.00%**			**1.5**			**7.13**			**6.7**

Figure 20: Sensor Suite Trade Study Results (Part 2)

This solution is the easiest to implement, as the VANTAGE team was based on this design, and it scores highly in short distance accuracy. These two metrics carry the highest weight, so it is beneficial for the design that this solution scored highly there. It does however require a large amount of space compared to just a single visual camera, and has a high power draw. A single visual camera came in a very close second, scoring highly for volume and power. However, it loses out when it comes to accuracy, as this approach requires no overlap between CubeSats. Staggered dual visual was anticipated to score higher, but it suffered from the same drawback as the single visual– it is crucial for visual sensors to see an entire edge to determine distance. This drawback causes these two approaches to lose out to the Time of Flight solutions, which are more robust.

3.2.4. Design Results

Electrical Power System

In order to integrate all the electrical components and have them operate, the power distribution system needs to be designed. The following table shows the input voltage and power consumption for each component.

Components	Input Voltage	Maximum Power consumption	Data Interfacing
NUC board 7i7DNBE	12-19V	65W	USB3.0 & Ethernet
Raspberry Pi 3 B+	5V	5W	USB2.0 & Ethernet
ToF camera	24V	40W	Ethernet
Monochrome camera	5V	5W	USB3.0
GPS receiver & antenna	3.3V	1.2W	on-board
Total Maximum Power consumption		116.2W	

Table 6: Power & Data interface

By mimicking NanoRocks deployers, VISION shall be able to operate at 120 VDC with a 3 Vpp ripple voltage, at 5A and with a power draw less than 520W. The Table.6 shows that the maximum power consumption from our system is much lower than the maximum given power consumption. Therefore, the system satisfies DR.4.3.

Moreover, in order to interface with multiple deployers, the team decided to use the most common interfaces. The chassis has three power ports: +V, -V, and GND. Initially, the provided power flows through the first DC/DC converter (MEAN WELL DDR-120D-24) to step down the voltage from 120V to 24V to power the TOF camera. There are four output ports from the first DC/DC converter. Two of them (+V,-V) supports the ToF camera, and the other two support the power distribution board. There are 2 additional converters on the power distribution board, and both of them are TDK-Lambda I6A DC/DC. They step down the voltage from 24V to both 12V and 5V. These are used to power the NUC and the raspberry pi respectively. The monochrome camera and GPS receive power and data from the NUC via USB3.0 ports. This design satisfies DR.4.2.

The following schematic shows the schematic of the power distribution board (printed circuit board). PS2412 is the DC/DC converter that converts the power from 24V to 12V. PS245 is the DC/DC converter that converts the power from 24V to 5V. J1 is the power connector with 5.5/2.1mm diameter to power on the Intel NUC board. J2 is the USB 3.0 connector to power on the raspberry pi. The PCB layout is shown in Fig.22.

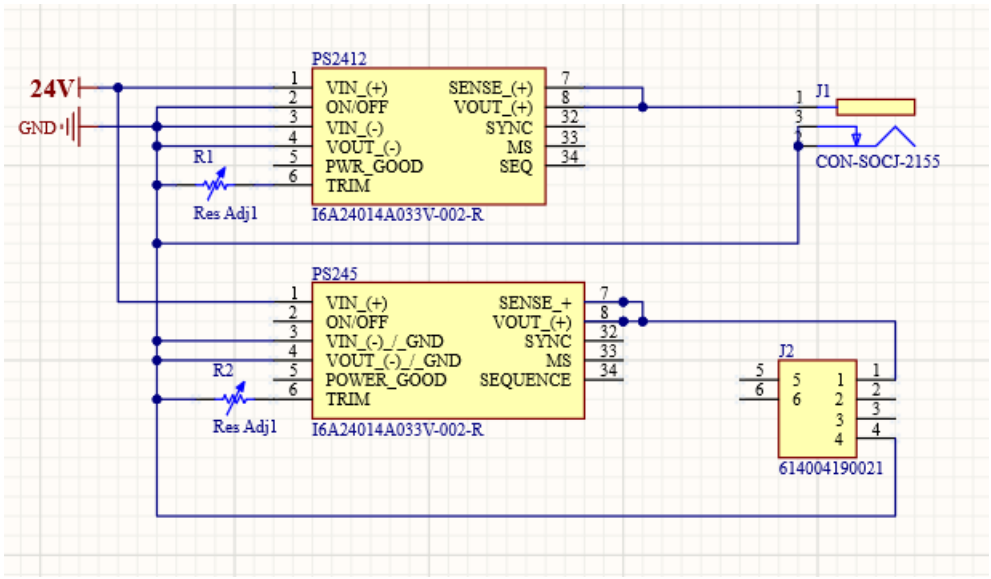

Figure 21: Schematic for PCB

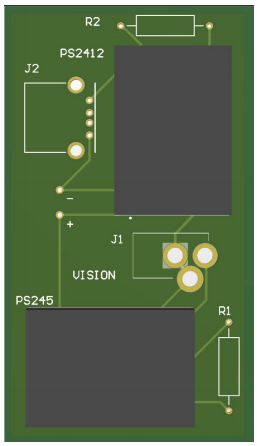

Figure 22: PCB layout (switch PS2412 245)

Electrical Telemetry System

The VISION shall be connected with a 5VDC USB 2.0 data bus. The telemetry system shall be able to work by USB 2.0 transfer.

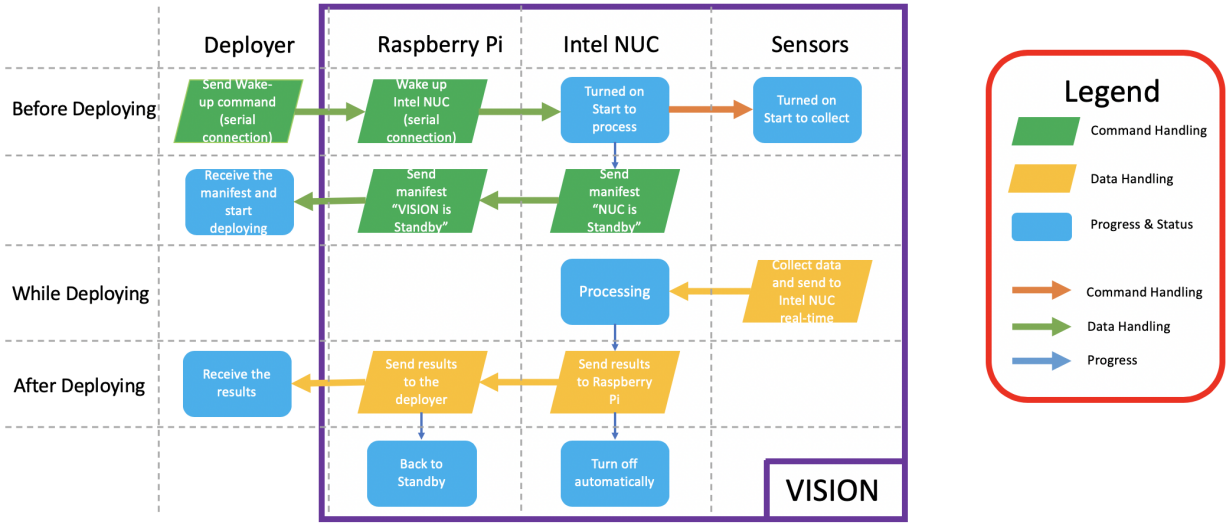

Figure 23: Telemetry System

The telemetry system of VISION is working under the flowchart shown above. Before the deployer starts to launch CubeSats, the deployer needs to send a wake-up command to the raspberry pi. As the raspberry pi receive the command, it wakes up the Intel NUC. At the same time, the sensors suite (Monochrome camera, ToF camera, and GPS) wake up. Besides, after the NUC is awake, it sends a message ”NUC is standby” back to the raspberry pi. Then raspberry pi receives this message and sends a message to the deployer about the manifest that VISION is ready to work. Then the deployer can launch CubeSats. The sensors collect data and transfer them to the NUC real-time, and the NUC processes those data to get estimates. After deploying all the CubeSats, NUC finishes all the data processing tasks firstly and sends them to the raspberry pi. Then it turns off automatically and waits for the next deployment cycle. After the raspberry pi receives all the data, it transfers them to the deployer and back to standby to wait for the next iteration.

3.3. Software

3.3.1. Overview

The stated goal of the VISION project is to produce TLE that improve the trackability of CubeSats after deployment. This is directly achieved by VISION's software package which processes raw sensor measurements to produce TLE for each deployed CubeSat. This process is broken into two parts:

1. Centroid determination
2. State estimation

The centroid determination algorithm was developed by VANTAGE and processes raw TOF and monochrome measurements to compute centroids for each CubeSat as they drift away from the deployer. The team will briefly explain the design of the centroid calculation, and for further details please refer to VANTAGE's Project Final Report. The centroid determination suite follows the steps:

1. Input raw sensor data
2. Initialize deployer and CubeSat parameters
3. Calculate centroids from optical camera
4. Calculate centroids from TOF camera
5. Combine and validate centroids

The state estimation algorithm is developed by the VISION team and uses those centroids measurements to compute estimates for the relative orbital states of the CubeSats which are in turn used to compute the orbital elements of each CubeSat. This information is used along with pre-assigned quantities to populate TLE for each deployed CubeSat. The state estimation algorithm is broken into five steps:

1. Transformation from the Sensor frame into the deployer orbit frame
2. Estimation
3. Transformation from the deployer orbit frame into the Earth inertial frame
4. computation of orbital elements
5. TLE assembly

The state estimation algorithm handles CubeSat states in the sensor frame, the deployer orbit frame, and the inertial frame. A general visual representation of the frames used is shown in the Fig. 24 and will aid in comprehension of the state estimation algorithm described below.

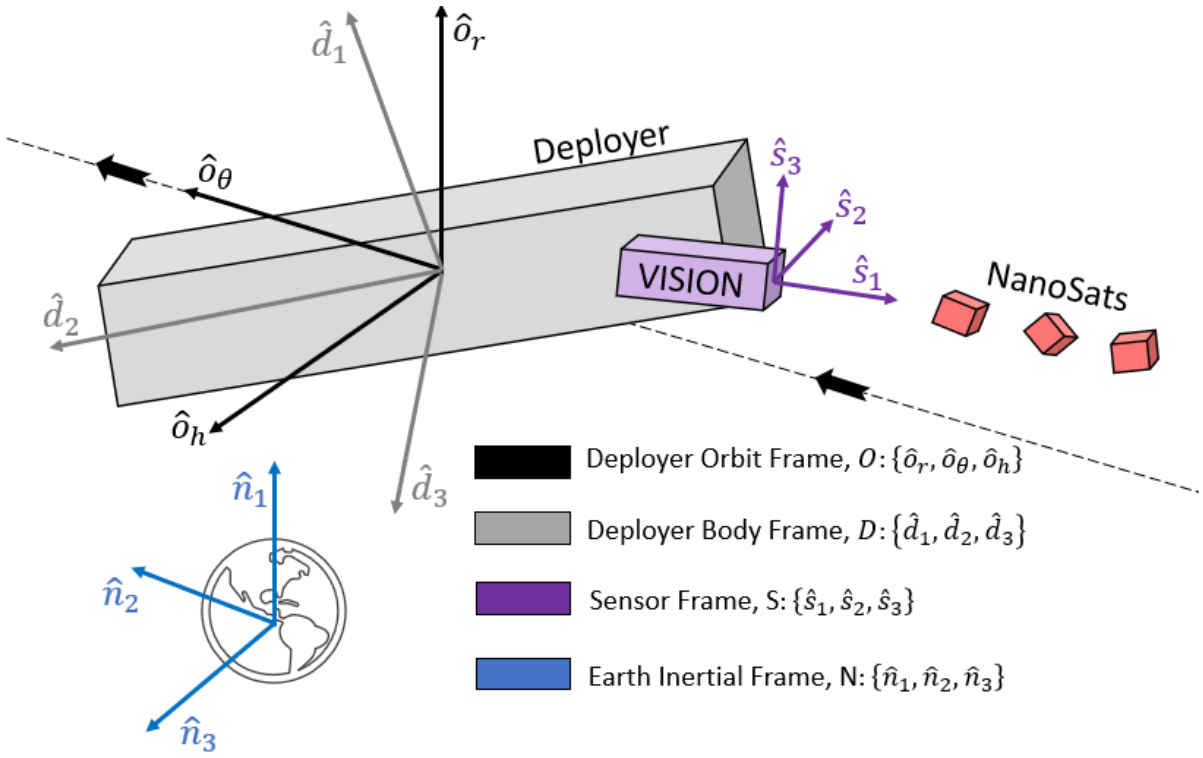

Figure 24: Frame Definitions Used in State Estimation Algorithm

3.3.2. Design Requirements

The centroid determination suite must satisfy the functional requirement

1. FR-1: The tracking system shall observe 6 separate CubeSats from the deployment platform.

The design requirements derived from the functional requirement are shown in Table 7. In order to satisfy FR-1, VISION must be able to differentiate each object as they are being deployed in order to identify CubeSats, which is the rationale for DR-1.1. DR-1.2 allows VISION to calculate the centroid of each CubeSat as they are being deployed in order to track each CubeSat. DR-1.3 will allow VISION to identify each CubeSat as it is being deployed by its size and order of deployment.

Table 7: Centroid Determination Design Requirements

Functional Requirement ID	Design Requirement ID	Design Requirement
FR-1	DR-1.1	VISION shall characterize and differentiate up to 6 CubeSats of 1U and up to 3U sized CubeSats.
FR-1	DR-1.2	VISION shall estimate the centroid of each CubeSat of 1U up to 3U size.
FR-1	DR-1.3	VISION shall utilize a deployment manifest provided by the deployer to identify each object.

VISION's state estimation package must satisfy functional requirements

1. FR-2: VISION shall report Two-Line Elements for each deployed CubeSat
2. FR-4 VISION shall integrate the functionality of both software and hardware within a single package.

In order to drive the design to meet these functional requirements, the design requirements in Table **??** were developed and shall be met by the VISION state estimation package.

Functional Req ID	Design Req ID	Design Requirement
FR-2	DR-2.2	VISION shall estimate the orbit frame position of each CubeSat such that the estimate covariance conforms to the Position Uncertainty Map.
FR-2	DR-2.6	VISION shall produce Two-Line Elements for each deployed CubeSat.
FR-4	DR-4.5	All VISION software shall be integrated into a single algorithm that is run on VISION's processing unit

DR-2.2 is defined to ensure that the TLE estimates of deployed CubeSat states produced by VISION's software package are accurate enough that they may be used effectively to track CubeSats later in time. Have accurate estimates is necessary for the results from the VISION project to be relevant and useful when enabling ground stations to better track deployed CubeSats. DR-2.4 is critical to ensure that the estimates produced by the VISION package can be used to make up all observed quantities used in TLE. VISION needs to be able to estimate all required elements of a TLE using in-situ measurements collected by its sensor suite in order to produce TLE for each deployed CubeSat. DR-4.5 is necessary to ensure that the VISION software can be integrated into one working software package and then executed using VISION hardware. This is required for VISION to be a fully integrated system.

The Uncertainty Map defining the accuracy requirement is shown in Fig. 25. This figure depicts standard deviations in the along-track and cross-plane in the deployer orbit frame. The green region shows combinations of these standard deviations such that if CubeSat state estimate error is sampled from those distributions, then the propagation error has a 95% chance of being less than the a ground station with a 70 degree field of view after being propagated for 3 orbits. 70 degrees is the smallest field of view of any professionally registered ground station as stated by Vallado's Fundamentals of Astrodynamics and Applications. In other words, if the standard deviations produced by the filter have a covaraiance that matches the estimate error magnitude and the standard deviations computed from those covariances lie in the green region, then VISION's estimates can be used to track CubeSats within 3 orbits after deployment with 95% confidence. It is conservatively assumed that the velocity error is sampled from a distribution that is two times the position uncertainty divided by the sample period of the time of flight sensor.

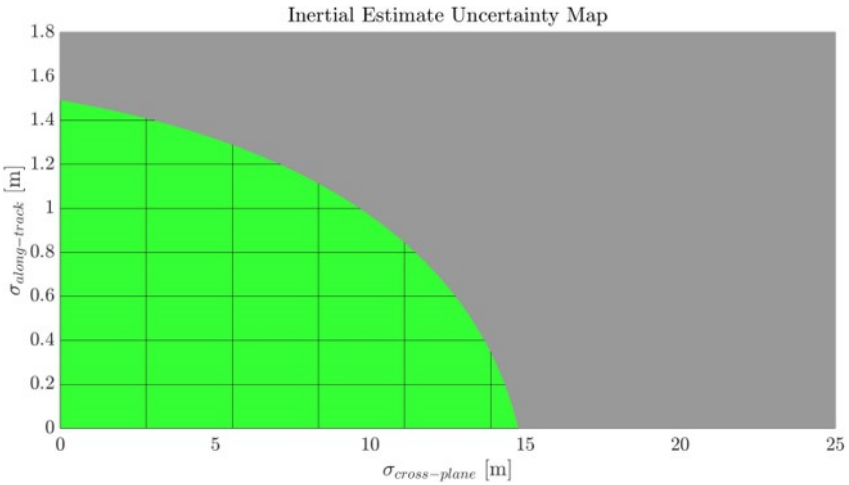

Figure 25: Uncertainty Map

3.3.3. Trade Studies

Programming Language

This trade study highlighted the advantages and disadvantages of each programming language in the aspects that related to VISION's mission scope. The main capabilities considered in this trade study were image processing, resource ecosystem, readability/development time, embedded system implementation, and run-time. The decision of programming language was to be implemented to new flight software and ensured that VISION achieved an optimized balance between development time, run-time speed, and fulfillment of design requirements.

MATLAB is a high-performance language for mathematical computing, more specifically matrix operations. MATLAB is a very well-documentation language that makes prototyping and debugging very fast and easy. Because of this, simulations and testing algorithms in MATLAB are quick and efficient because it is a programming interface and does not need to compile. MATLAB's toolboxes are very robust because of the extensive development and verification

from credible sources, which reduces development and implementation time for the user. Toolboxes such as the Image Processing Toolbox are very powerful because of MATLAB's ability to perform fast numerical matrix manipulation, which is the main component of image processing. Since MATLAB is a high-level language and interactive environment, it has one of the slowest overall run-times due to the complexity of the built in interpreter. This results in high computational cost (RAM) and overhead memory allocation, which is not ideal for product development. Another downside is that it does not have an open resource ecosystem and the user is limited to the toolboxes and packages available. Python is one of the most popular and widely used programming language. Because of Python's growing popularity, it is an open source and community development with extensive support libraries. In recent years the number of data manipulation and visualization packages for Python has increased exponentially. Examples of these are NumPy, SciPy, Scikit-learn, and Matplotlib, which can work in conjunction with OpenCV for a powerful Computer Vision and Machine Learning environment. Python is also built for readability with the use of block-based indentation, simple syntax (like using square brackets and parenthesis), and augmented assignments. This greatly improves programmer's productivity and development time because there is very little time wasted in organization, readability, and debugging efforts. The main downside of Python compared to C++ is its speed because it is not a compiler language. However, C/C++ extensions for Python can be used with the package Cython, providing speed increases of up to 100x native Python. Python is also known for simple and flexible object-oriented programming (OOP) because of its ability to handle and manipulate data structures with ease from libraries such as Pandas.

C++ is a compiled language and is often used in applications where speed is critical. There are many advantages to a compiled language like C++, such as having compact code with faster run-times and object oriented capabilities. This is obviously an optimal choice in mobile computing, and that is why 95% of embedded system programs are in C/C++. Additionally, C++ is used in many production-grade Computer Vision systems because it has great libraries such as OpenCV, which has excellent performance and is easy to put into a production environment. OpenCV is also written in C/C++ which means that developers can easily modify the source library for a specific need. The biggest downside to C++ is that it is slow to write, error prone, and frequently unreadable. Since it is a hard environment to visualize and debug, it is not an ideal language for beginners or group collaborations. This would have dramatically increased the team's effort in development and learning time, which was a huge disadvantage.

Development time was the biggest factor and had the heaviest weight. Python has straightforward and readable syntax, and can be written fairly quickly. C++ has the worst development time for the same reasons as its slow learning time, which would impact development time as well. MATLAB's programming environment is very well documented, and easy to work with, however data structure handling can be inefficient and difficult to implement.

Python has the biggest open source ecosystem, which means it has the most available libraries and packages. Some popular and useful packages include NumPy, SciPy, Scikit-learn, Cython, and Matplotlib. These packages can easily give Python an advantage over other languages in specific areas. On the other hand, MATLAB has the smallest available libraries because it does not operate on an open source ecosystem, and cannot easily extend its capabilities outside of mathematical matrix computation. C++ is in between these two due to its open source, but has fewer than Python.

Finally, C/C++ are the most commonly used in embedded systems because of their fast run-time, dominating 95% of the embedded systems market. While Python can't run as fast, it is is the fastest growing language used in embedded computing due to its popularity. This means that many commercial off-the-shelf (COTS) computing boards will be able to run Python. Even though MATLAB is not a production-ready language, it has the capability to translate code into very compact languages such as C, C++, and HDL that can be run on embedded systems. However, these conversions mainly work on the algorithmic portion of scripts and not other functions such as image processing. This defeats the purpose of writing in a different language and leaves MATLAB as the worst choice for our embedded system.

The metrics and weight determination for the programming language can be seen in Table 12.

Flight Software Trade Study Table										
		MATLAB			Python			C++		
Metric	Weight	Performance	Score	Points	Performance	Score	Points	Performance	Score	Points
Run Time	25%	0	0	0	6	6	1.5	10	10	2.5
Learning Time	15%	10	10	1.5	7	7	1.05	0	0	0
Development Time	30%	10	10	3	9	9	2.70	0	0	0
Libraries/Packages	15%	0	0	0	10	10	1.50	7	7	1.05
Embedded System Implementation	15%	0	0	0	5	5	0.75	10	10	1.5
Wt. Total	**100.00%**			**4.50**			**7.50**			**5.05**

Figure 26: Programming Language Trade Study Results

State Estimation

One of VISION's primary objectives is to report TLE for each deployed CubeSat. In order to perform this function VISION will use its own inertial state and the relative states of the CubeSat. Obtaining accurate relative state estimates starts with the data collected using VISION's sensors followed by relative positions extracted from sensor data using image processing techniques. Both of these processes introduce significant amounts of uncertainties in the calculated relative position that get worse as the CubeSats drift farther from the deployer. This error directly impacts the accuracy of calculated TLE and if large enough can cause TLE to impede ground based tracking capabilities by providing incorrect orbit information. In order to minimize the error introduced by VISION's sensors and image processing algorithms a more robust state estimation method is required. This will enable vision to take advantage of a priori knowledge of system dynamics and filter all sensor data to minimize the effects of sensor noise. The filter used will greatly affect the computational resources required by VISION, optimality of the estimate, and the resources invested by the software team to develop a state estimation algorithm. For these reasons a rational variety of state estimation algorithms and filtering methods are explored.

In order to quantify the efficacy of each state estimation method to VISION's requirements, the following metrics are considered: robustness, optimizability, learnability, and computational expense. *Robustness* is defined as a sensors ability to produce accurate estimates using noisy measurements and model error. *Optimizability* is defined as the state estimation algorithm's ability to provide an optimal estimate. *Learnability* is defined as the software team's ability to learn and develop the state estimation method being used. Finally, *computational expense* is defined as the time required to compute the state estimates and the memory required to store intermediate calculations and measurement data during the estimation process.

State Estimation Method Quantitative Trade Study Table																
		Weighted Least Squares			Recursive Least Squares			Classical Kalman Filter			Extended Kalman Filter			H_infinity Filter		
Metric	Weight	Performance	Score	Points	Performance	Score	Points	Performance	Score	Points	Performance	Score	Points	Performance	Score	Points
Robustness	30%	2	0	0	2	0	0	6	5	1.5	8	7.5	2.25	10	10	3
Optimizability	20%	6	3.33	0.67	4	0	0	8	6.67	1.33	8	6.67	1.33	10	10	2
Learnability	30%	10	10	3	8	7.5	2.25	6	5	1.5	4	2.5	0.75	2	0	0
Computational Expense	20%	6	3.33	0.67	10	10	2	8	6.67	1.33	6	3.33	0.67	4	0	0
Wt. Total	**100.00%**			**4.33**			**4.25**			**5.67**			**5.00**			**5.00**

Figure 27: State Estimation Trade Study

The trade study found that the Kalman filter will most likely be the optimal method of state estimation for this project. The next-highest scores are held by the EKF and H_∞ filter. Should a nonlinear dynamics model be required, the EKF (or some other nonlinear Kalman filter variant) would likely be the optimal method of state estimation. Therefore, our initial selection for state estimation filter was the Kalman filter. However, it was later decided to switch to a nonlinear batch filter. This filter type was not listed in the trade study; however, testing found that this filter produced better results than the Kalman filter, especially for eccentric orbits. Additionally, it was determined that all of the data was being processed as a batch, so this filter was better applicable to the system than a Kalman filter.

3.3.4. Design Results

The resulting design can be broke into three main categories:

1. Centroid determination

2. State estimation

3. TLE assembly

Figure 28 is a top-level overview of the software suite. VISION will keep the centroiding algorithm from VANTAGE which is all in MATLAB. VANTAGE's post processing will output the calculated centroids of each CubeSat from optical data and point cloud data. The centroids will then be the input into the state estimation unit. The whole state estimation suite will calculate the CubeSat inertial state, which will eventually be used to generate TLEs.

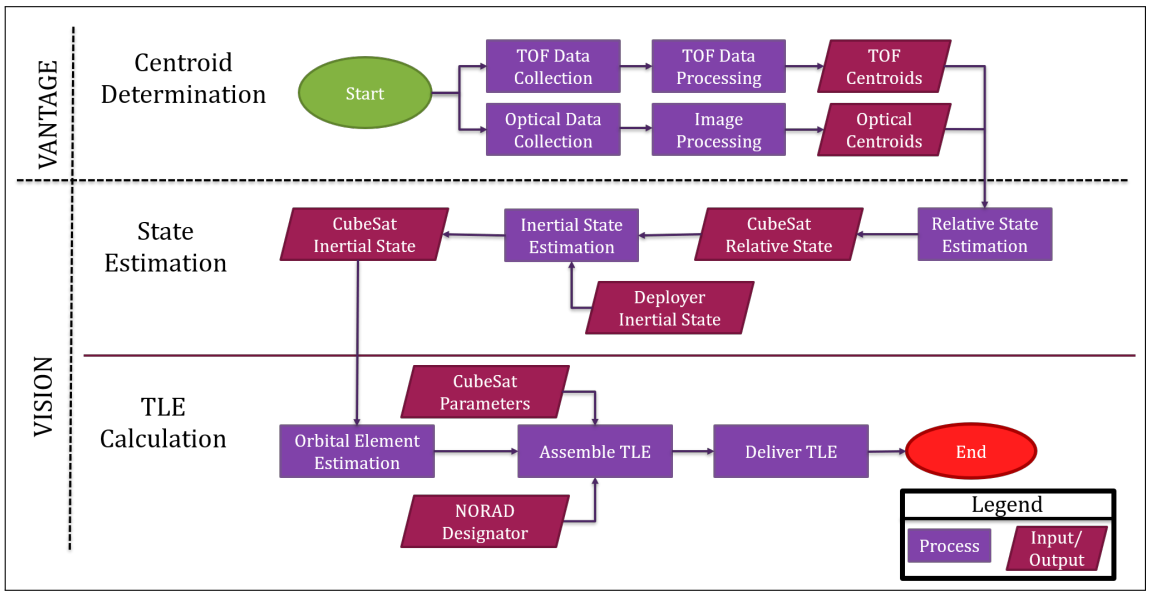

Figure 28: High-Level Software Flowchart

It is important to note that the centroid determination algorithm was developed by last year's team will not be discussed in detail here. At a systems level, the heritage centroid determination algorithm process raw sensor measurements to compute centroids for each CubeSat over the time span of the CubeSats being deployed. These centroids are then processed individually for each CubeSat with additional information of the deployer's inertial state to estimate the inertial states of the CubeSats. These inertial states are then converted to a set of orbit elements for each CubeSat which make up the bulk of each TLE. For details on the centroid methods, please refer to VANTGE's Project Final Report.

The VISION software package starts by receiving all raw measurements collected during the data collection period. This is comprised of point clouds from the time of flight camera and images from the monochrome camera. This raw measurement data package is expected to be consist of 600 monochrome frames and 270 TOF frames for a typical 10 second data collection period. A conservative estimate places the total size of the raw measurements to be approximately 2 GB with 3 MB being contributed from the monochrome camera and 90 KB being contributed by the time of flight camera. Heritage code developed by last year's team, VANTAGE, processes the point clouds to distinguish between CubeSats and identify centroids for each TOF frame for each CubeSat. VANTAGE's centroids determination algorithm then uses monochrome cross-plane observations to correct the cross-plane positions of the centroids computed from the time of flight camera point clouds. More details of this algorithm can be found in VANTAGE's Project Final Report.

VISION's state estimation algorithm is initialized by receiving centroids for each CubeSat at each time of flight measurement time. These centroid measurements are received in the sensor frame. It also receives the deployer inertial cartesian state and attitude over the data collection period. The centroids are grouped for each CubeSat and to be processed individually by VISION's state estimation algorithm. The deployer state information is interpolated using cubic splines to the measurement times so that they can be used with the CubeSat centroids when performing frame rotations and when computing CubeSat inertial states. VISION's state estimation algorithm also receives a deployment manifest which includes the order at which CubeSats are deployed and the mounting position and orientation of the VISION package on the deployer.

In order to leverage the general relative equations of orbital motion in VISION's statistical estimation algorithm, the centroid measurements that are in the sensor frame must be transformed into the deployer's orbit frame which

is the frame that the desired equations of motion are defined in. In order to transform the the sensor frame centroid measurements into the deployer orbit frame, deployer inertial states and attitude as well as VISION's mounting position and orientation on the deployer are required. The algorithm developed to perform this rotation is described in Algorithm 1. Note that DCM denotes Directional Cosine Matrix.

Algorithm 1: Sensor Frame to Deployer Orbit Frame Transformation Algorithm

Input CubeSat centroid measurements in the sensor frame relative to the sensor frame, deployer inertial state and attitude, and VISION mounting orientation and position.

1. **Compute** CubeSat positions relative to the deployer frame in the sensor frame using VISION's mounting position and CubeSat centroid measurements.
2. **Compute** 3-2-1 DCM rotating from the sensor frame to the deployer body frame using VISION's mounting orientation.
3. **Compute** DCM rotating from Deployer frame to inertial frame using the inertial state of the deployer.
4. **Compute** 3-2-1 DCM rotating from the deployer body frame to the deployer inertial frame using deployer inertial attitude.
5. **Compute** DCM rotating from sensor frame to deployer orbit frame using DCM from steps 2-4.
6. **Compute** Sensor frame centroids relative to the deployer orbit frame into the deployer orbit frame using DCM from step 5.

Return CubeSat centroid measurements in the deployer orbit frame relative to the deployer orbit frame.

The design for this algorithm is greatly driven by the information available to perform the rotations which forces the sensor frame centroid measurements to be rotated into the deployer body frame, then the inertial frame, and then the deployer orbit frame.

With measurements now in the deployer orbit frame the relative orbital states of the CubeSats are estimated using a nonlinear batch filter. The nonlinear batch filter is selected as its a means of performing statistical state estimation because it can be used with the nonlinear general equations of relative motion that introduce the lowest amount of process noise for all elliptical deployer orbit scenarios. Furthermore, this selection takes advantage of the fact that VISION processes data in batch after the data collection period which takes advantage of the ability to iterate over the information multiple time to get an accurate initial guess in order to improve the estimates. For this reason, batch style processing in general performs better when a smaller number of measurements are obtained than sequential filters. The batch filter uses the general equations of relative orbital motion which describe the relative cartesian state of a deputy satellite (in this case the deployed CubeSats) relative to a chief satellite (in this case the deployer) in the deployer's orbit frame while both the chief and deputy are in Keplerian orbits. Equations 1, 2, and 3 describe the cartesian formulation for this dynamical model.

$$\ddot{x} - 2\dot{f}(\dot{y} - y\frac{\dot{r}_c}{r_c}) - x\dot{f}^2 - \frac{\mu}{r_c^2} = -\frac{\mu}{r_d^3}(r_c + x) \tag{1}$$

$$\ddot{y} + 2 * \dot{f}(\dot{x} - x\frac{\dot{r}_c}{r_c}) - y\dot{f}^2 = -\frac{\mu}{r_d^3}y \tag{2}$$

$$\ddot{z} = -\frac{\mu}{r_d^3}z \tag{3}$$

where x, y, z are the CubeSat cartesian position relative to the deployer, $\dot{f}$ is the deployer true anomaly rate, mu is Earth's gravitational parameter, r_d is the deputy's (CubeSat's) distance from Earth, and r_c is the chief's (deployer's) distance from Earth. Note that r_d is a function of the relative state of the CubeSat and the inertial state of the deployer so the inertial states of the CubeSats do not need to be known at this point in the algorithm. Equations 1, 2, and 3 are used to compute the dynamics matrix, A, which is the Jacobian matrix of the described system. The measurement sensitivity matrix used by the filter takes a simple form because three of the deployed CubeSat states that are being estimated are directly measured. The measurement sensitivity matrix, H, takes the form in Eq. 4 and is used to map a state into measurement space which in this case is the three coordinates of relative orbital position.

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{4}$$

The nonlinear batch filter is described by an inner and outer loop. The filter is initialized with a nominal state and an expected estimate covariance matrix. The inner loop is defined by two steps, a time update step, a measurement update, and an observation accumulation step. At the beginning of each inner loop, the nominal trajectory is propagated

to the next time at which a measurement occurs. At this point, the nominal trajectory is transformed in the state space using the H matrix and a residual is computed as the difference in the actual measurement at that time and the nominal state in measurement space. During observation accumulation step, the measurement covariance matrix is inverted, converted to the state space using H, and propagated to the initial time using State Transition Matrices (STM) that are computed using the dynamics matrix liearized about the nominal trajectory. This accumulates all measurement uncertainties at the initial time for the state estimate in the form of an information matrix which is the inverse of a covariance matrix. A second part of the observation accumulation step is to convert the previously computed measurement residual into state space and propagate it back to the initial time using the same STM used to propagate the measurement covariance matrix. This is repeated for each inner loop to accumulate the net deviation from the nominal initial state that the filter believes to be the true trajectory. This inner loop is repeated until the update to the initial nominal state reaches a minimum thereby estimating the best guess for a deviation from the nominal state that when added to the nominal state yields an estimate for the true initial state. This estimate for the true initial state can be propagated using the nonlinear equations of motion to obtain estimates for all desired during the measurement collection period. Algorithm 4 describes the nonlinear batch filter algorithm designed for VISION.

Algorithm 2: VISION Nonlinear Batch Filter Algorithm

Input CubeSat centroid measurements in the deployer orbit frame, interpolated deployer inertial states, measurement times, nominal initial state, initial state estimate covariance, measurement covariance matrix.
while *update to nominal initial state is unconverged* **do**
  1. initialize CubeSat initial state as best estimate
  2. initialize state transition matrix as identity
  3. initialize information matrix as inverse of initial state estimate covariance
  4. initialize accumulated deviation
  for *all measurement times* **do**
    5. propagate nominal trajectory to the next measurement time
    6. propagate STM to the next measurement time
    7. propagate deployer state to the next measurement time using current state measurement
    8. compute measurement residual
    9. accumulate initial state estimate information matrix
    10. accumulate deviation from nominal
  end
  11. Compute Cholesky decomposition of initial state estimate information matrix
  12. Compute initial state covariance matrix using results from step 11
  13. Compute estimated update to nominal initial state
  14. Compute updated nominal initial state
end
Return nominal initial state estimate, nominal initial state estimate covariance

This is a modified version of the general nonlinear batch filter algorithm described in Tapley, Shutz, and Born[16], tailored to work with the selected dynamics and is unique in the way that it handles the general relative equations of motion. Not only does it propagate the nonlinear dynamics of the deployed NanoSats but it does so using propagated measurements of the deployer's inertial state. Furthermore, the algorithm leverages Choleskey factorization when inverting matrices to reduce numerical precision error introduced by inverting ill-conditioned matrices. A further desired extension to this algorithm is to estimate the deployer inertial state as well as the CubeSat relative states. This would allow for a better estimate of the deployer's states using measurements GPS measurements obtained during the deployment period. This would accommodate the fact that the deployer state is not known with absolute precision.

With CubeSat relative states estimated in the deployer orbit frame, the algorithm now transforms these estimates into inertial states in the inertial frame which is required when computed orbital elements from a cartesian state. The algorithm written to perform this function is described in more detail by Algorithm 3.

Algorithm 3: Orbit Frame to Inertial Orbit Frame Transformation Algorithm

Input CubeSat relative orbit frame states, inertial deployer states, attitudes, and attitude rates
1. Compute DCM rotating from deployer orbit frame to inertial frame using inertial states or the deployer.
2. Rotate CubeSat relative orbit frame positrons into the inertial frame using results from step 1.
3. Compute deployer true anomaly rate (used in step 5) using inertial deployer states.
4. Rotate CubeSat relative orbit frame velocity into the inertial frame relative to the inertial frame using transport theorem.
Return CubeSat inertial states.

With inertial states for each CubeSat, the orbital elements can be computed using the conversion algorithm found in Schaub and Junkins[15]. Only the initial state estimate is converted into a set of classical orbital elements because it is the most accurate estimate produced by the filter. If other states were used, they would incorporate process noise from unmodelled perturbations introduced when integrating the initial state estimate. Furthermore, the time associated with the first measurement is used as the epoch associated with the orbital elements.

The TLE assembly algorithm uses estimates with the designators and drag parameters assigned to each CubeSat prior to deployment to create TLE. The elements used in the TLE are not exactly the classical orbital elements that have been computed at this point however they can be computed using the classical orbital elements. In the TLE, the mean anomaly and mean motion replace the true anomaly and the semi-major axis from the computed classical orbital element set, respectively. Both of these quantities are computed using the the classical orbital element set and Earth's gravitational parameter via methods found in Schaub and Junkins[15]. Each quantities in the TLE is truncated to the correct number of digits as defined by Celestrak and the final TLE for each CubeSat is reported as a string.

Object oriented software design was used to create the integrated software package. This allows for an organized and efficient code structure. VISION's software package contains three main classes: *Deployer*, *Satellite*, and **NLB**. An additional class used to store state information, *State*, is also used to provide a general means of storing state information. These classes along with their attributes and methods are depicted in Fig. 29. It is important to note outside of these three classes, the centroid determination algorithm received as heritage code is treated as its own black box function and there are two additional functions, *format_satellite()* and *format_deployer* which are used to format the centroids and deployer state information received at the start of the state estimation algorithm.

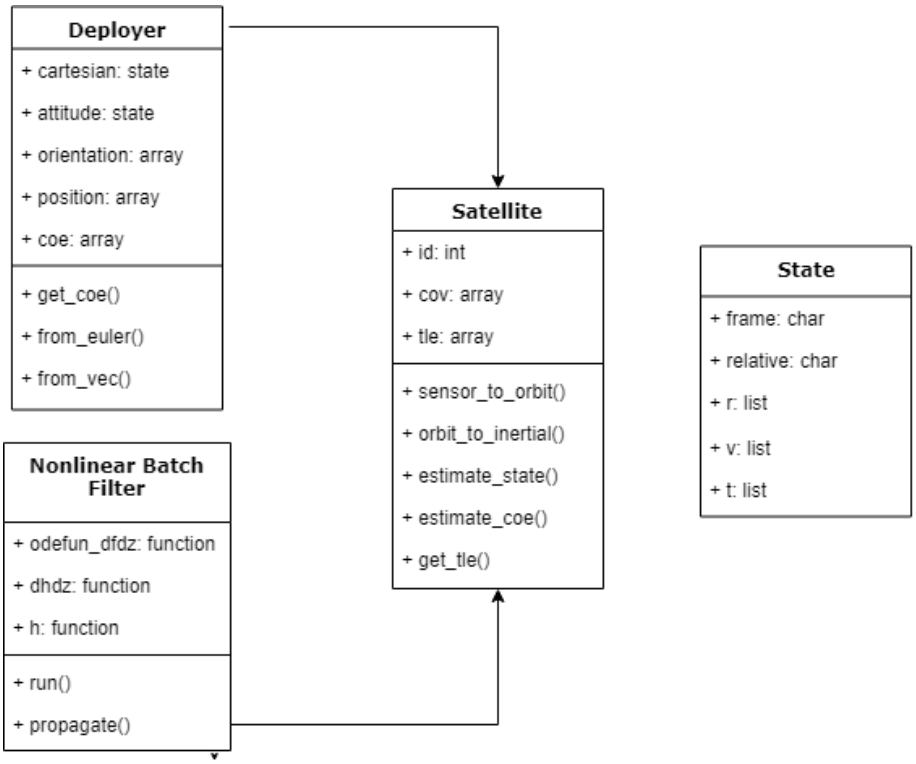

Figure 29: Class Definitions

The deployer class is used to store the cartesian state of the deployer (*cartesian*), the attitudes and attitude rates of the deployer (*attitude*), the mounting orientation and position of VISION on the deployer relative to the deployer center of mass (*orientation* and *position*), and the orbital elements of the deployer (*coe*). Note that the attitude and cartesian states are saved as objects of the *State* class which stores the frame that the states are represented in (*frame*), the frame that the states are relative to (*relative*), the position state (r), the state rates (v), and the time associated with those states, (t). The *Deployer* class also method that convert all states into orbital elements (*get_coe()*), compute a DCM from Euler angles (*from_euler()*), and that create a DCM that converts from orbit frame to inertial frame using inertial state vectors (*from_vec()*). In practice, a *Satellite* class is created for each satellite that is deployed. The *Satellite* class inherits from the *Deployer* class and also introduces unique attributes that have a satellite id (*id*), list

of covariance matrices produced by the nonlinear batch filter (*cov*), and a variable to store the computed TLE for that CubeSat (*tle*). The *Satellite* class has methods to transform centroid measurements from the sensor frame to the deployer's orbit frame (*sensor_to_orbit()*), transform relative orbit frame states to inertial states (*orbit_to_inertial()*), running the nonlinear batch filter (*estimate_state()*), estimate orbital elements (*estimate_coe()*), and assemble the TLE (*get_tle()*). Figure 30 depicts the function interactions inside the state estimation algorithm in a mission scenario. Note that the upper box in each function description contains the function inputs and the bottom box contains outputs.

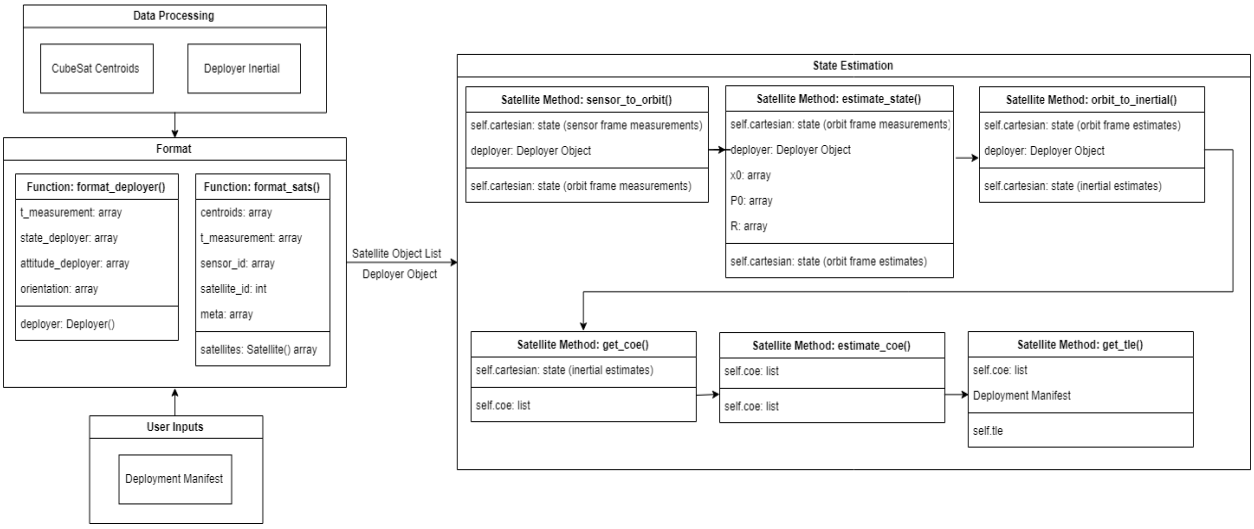

Figure 30: State Estimation Block Diagram

The state estimation process starts by receiving centroids for each CubeSat in the from heritage code, deployer inertial states during the deployment period, and the deployment manifest. The centroids and deployment manifest are passed into *format_satellites()* and produces a list of populated satellite objects, one for each deployed satellite. The measurement times and deployer inertial states are then passed into *format_deployer()* which interpolates the deployer states and attitudes to the measurement times and outputs a populated deployer object. The list of Satellite objects and and Deployer object are then used to estimates the states of each CubeSat. This is done individually for each CubeSat using the methods built into the Satellite objects. The state estimation portion of Fig. 30 shows the function call sequence for one satellite object to go from centroid measurements to TLE. This process starts with transforming the centroid measurements in the sensor frame relative to the sensor frame to the deployer orbit frame in the deployer orbit frame using *sensor_to_orbit()*. Method *estimate_state()* is then used to run the nonlinear batch filter on the centroid measurements in the orbit frame to produce position and velocity estimates in the orbit frame. Note when a frame transformation done, the *cartesian* State object attribute of the *Satellite* class is over written and the *State* object attributes *frame* and *relative* are changed to reflect the new frame that the state is defined in. At this point, the orbit frame state estimates are converted to the inertial frame using *orbit_to_inertial()*, inertial states are converted to orbital elements using *get_coe()*, the orbital element set selected to be used in the TLE and the epoch are assigned using *estimate_coe()*, and TLE are assembled using *get_tle()*.

Due to the COVID-19 crisis the majority of this design was completed however additional work needs to be done to complete the integration portion of the software design. This includes creating a black box for the centroid determination heritage algorithm. Because this algorithm is written in MATLAB, the code must be run from a python wrapping in order to integrate it with VISION's state estimation algorithm. Finally, the full software package must be ported onto the NUC processing unit and set up to run from a single execution file that is ran when directed by the Raspberry Pi after the data collection period is complete. The final improvement to the design that was not completed is to implement deployer state estimation into the nonlinear batch filter.

The software design above uses a nonlinear batch filter to meet the design requirement DR-2.2 which requires the state estimation algorithm to reach a specified level of accuracy such that the estimates are effective when being used to track the CubeSats after deployment. The design satisfies DR-2.4 by using a TLE assembly algorithm to format the required estimates, designators, and drag parameters into the correct format for TLE. DR-4.5 is met by this design by running the centroid determination algorithm with the state estimation algorithm on VISION's flight processor run from a single execution file. Though this last requirement was not met in work due to the COVID-19 crisis it was met by design.

4. Manufacturing

Adam Boylston, Cameron Baldwin, Ben Hagenau, Max Audick, Theodore Trozinski, Bao Tran,& Zhuoying Chen

4.1. Structures

4.1.1. Manufacturing Plan

The structures manufacturing consisted of machining the chassis components. There were six total components to be manufactured. There was a total of 3 pieces of aluminum purchased for machining. The side pieces were cut from a single piece of 6061 Aluminum and manufactured on the CNC, with some facing on the mills. The front and back pieces were CNC'd. Finally, the top and bottom pieces were manufactured on the mills. All manufacturing took place in the Aerospace Machine Shop and was verified with Matt Rhode. All CNC'ing used toolpaths generated from the CAD models. All milling was done based on drawing specifications.

4.1.2. Manufacturing Process

In the end, all six components were manufactured, however, holes were not able to be drilled and tapped due to the manufacturing halt.

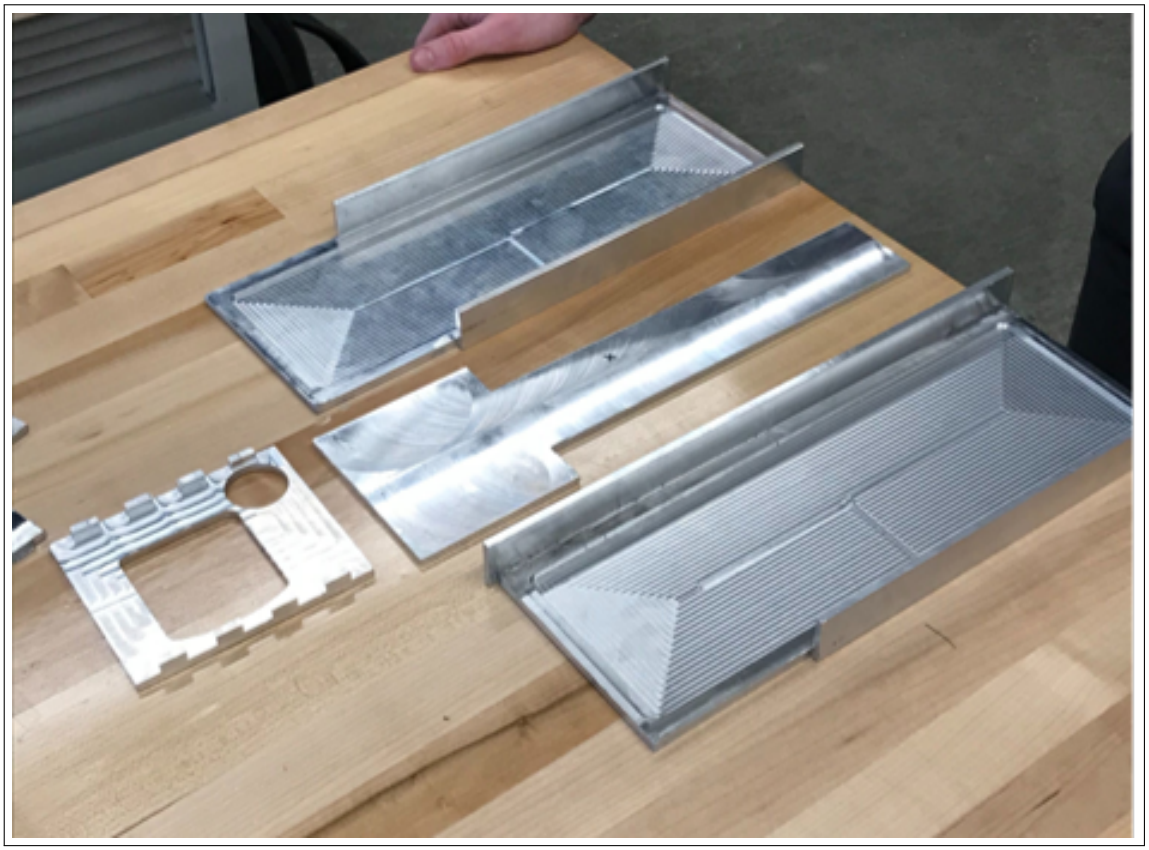

Figure 31: Chassis Components

There were difficulties in facing the larger pieces, like the sides. Vibrations during machining meant that the side pieces did not receive an even face. There was also an incident where the facing tool bit into one of the side pieces, however this was remedied with rubber cement. To mitigate vibrations on other pieces, facing was performed on the mills, where more clamping options were available, and it was easier to visually inspect for consistent facing.

4.1.3. Structures Integration

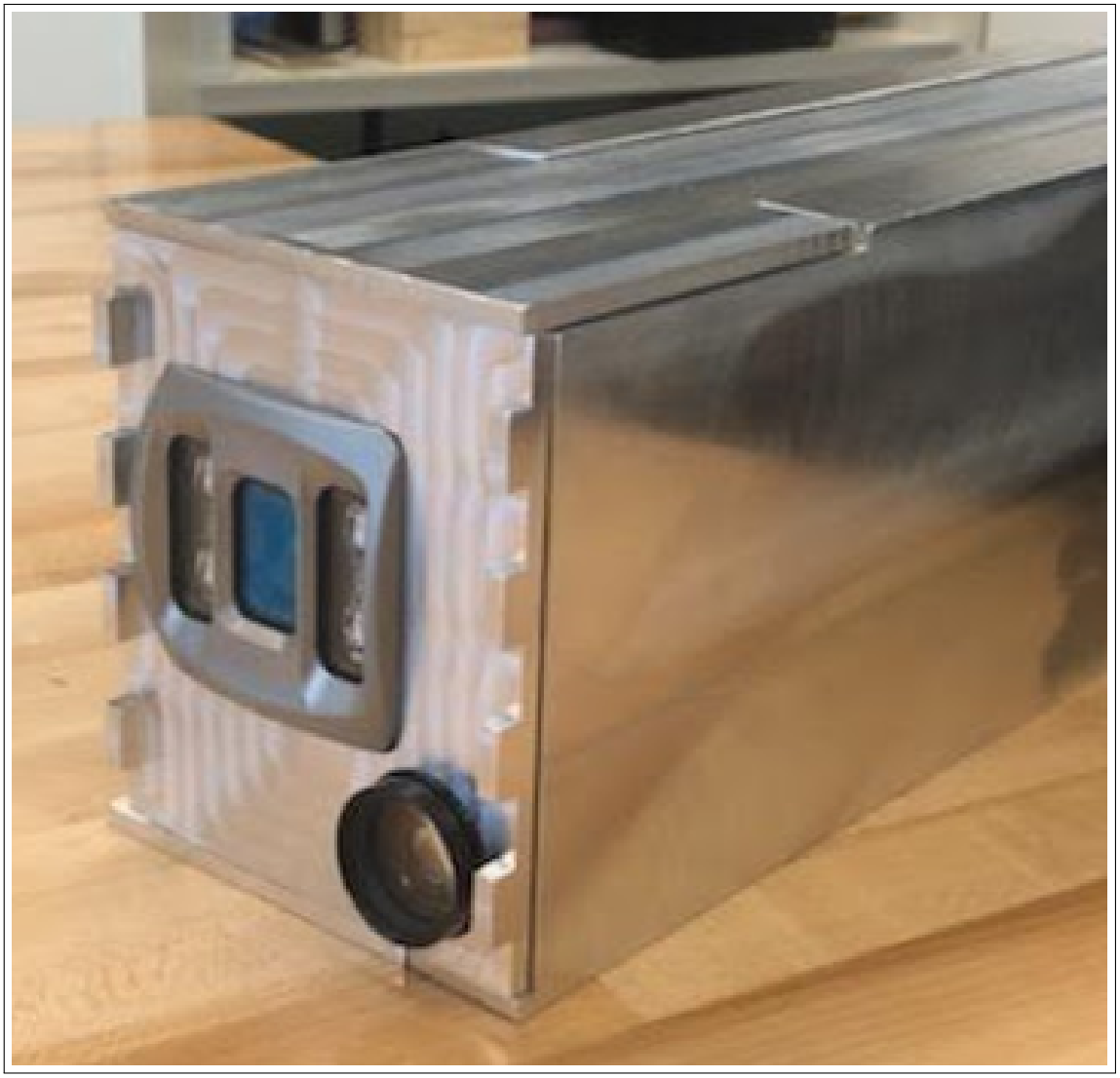

Figure 32: Fit Test

The next step, had the screw holes been drilled and tapped, would be a full system assembly test, wehre all components would be integrated into the chassis. This would have looked like Figure 10. Shown above in Figure 32 is a fit test of our monochrome and ToF cameras.

4.2. Electrical System

4.2.1. Procurement and Assembly Plan

Electrical Power System

For electronics, the most important part of manufacturing is the power distribution board. The previous team, Vantage, used the breadboard to distribute the power and integrate the electrical components together. When the team received their packages, the components on the breadboard has become unstable and the wires have mixed with each other. Therefore, the team decided to use the printed circuit board to keep the system clean and stable.

The PCB design is shown in Figure.22. The team used the **Altium Designer 20** on Windows to build the schematic and the PCB layout. Then the team contacted **Advanced Circuits** to print a 2-layer PCB board. Then it was being

soldered on the Electronics Workshop in Aero Building. The Table.8 shows the purchased items that are soldered on the PCB board.

Functions	Components
DC/DC converter 120V - 24V	DDR-120D-24
DC/DC converter 24V - 12V and 5V	TDK-Lambda I6A24014A033V-002-R
USB Connector	USB Connectors WR-COM USB Type A THT Horizontal
Power Connector	PCB Female DC Power Jack Socket Connector
Resistors	$1K\Omega$ and $14.6K\Omega$

Table 8: Purchased Components

Electrical Telemetry System

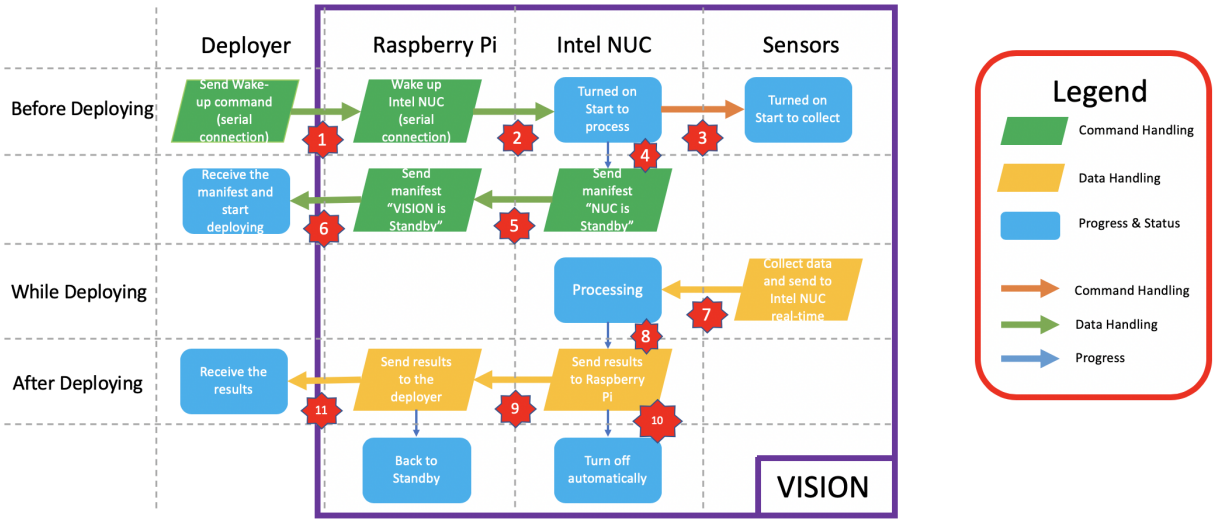

Figure 33: Telemetry System

The Figure.33 shows the telemetry system. Each red stars show the critical steps are needed to complete the design. Those needed steps are illustrated below with required actions.

1. The deployer needs to send a wake-up command to let the raspberry pi know that the deployers are going to launch the CubeSats. It is connected by a USB 2.0 port so that the team plans to use the serial connection to communicate between the deployer and the raspberry pi.

2. The raspberry pi receives the command and wakes up the Intel NUC. It is completed by the **Wake on Magic Packet** installed and activated on the Intel NUC.

3. After the Intel NUC is turned on, the sensors are woke up by the NUC. The monochrome camera and the GPS are powered by NUC so that there are not extra actions needed. But the ToF camera is powered from the first DC/DC converter directly. Therefore, the team plans to add an on/off control board between the 120V - 24V DC/DC converter and the ToF camera, which are controlled by the NUC.

4. This step involves the automation of the NUC. After the NUC is turned on, MATLAB will start to run automatically.

5. The MATLAB script includes a batch file (.bat) to send commands to the raspberry to let it know the VISION package is ready to work.

6. When the raspberry pi received the message, it sends a command to the deployer to let it know the VISION is at standby.

7. This step is completed by the MATLAB script.

8. This step is completed by the MATLAB script.
9. After the NUC finishes processing all the data, it runs a batch file(.bat) to execute some ”SCP” commands to transfer the estimated results to the raspberry pi.
10. At the end of the batch file, it includes a command that turns off the NUC.
11. After the raspberry pi receives all the data, it transfers them to the deployer.

4.2.2. Assembly Process

The Figure.34 shows the final product of the PCB board. It distributed the power to the raspberry pi and the Intel NUC successfully. However, the team did face some challenges while designing and soldering.

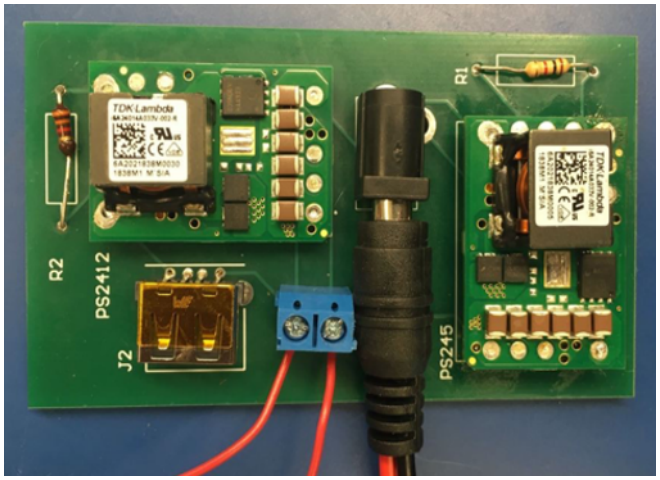

Figure 34: Final Product of PCB board

While designing the PCB board, the team used the Altium designer 20 which is suggested by PAB members Trudy Schwartz. In order to reduce the time of designing the PCB board, the team wanted to use the Altium library, but the computer in the university cannot release the administers' account to the students, so that the team couldn't download the library to use. Therefore, the team contacted Professor Eric Bogatin in Electrical Engineering Department to ask for permission to download it on our personal laptops. Thanks to Professor Eric Bogatin to give us access to the software so that the team saved a lot of time on designing the PCB board.

Moreover, while soldering the components on the PCB board, the team made a mistake that switched the label of the two DC/DC converter. Therefore, the input voltage of the raspberry pi exceeds the operating voltage range and burned it. The team spent a lot of time to resolder and make the board work. Thanks for Trudy Schwartz to help us with troubleshooting and soldering the components.

4.2.3. Electrical Integration

Electrical Power system 6 The Figure.35 shows the final product of the integrated electrical power system. The system has three power input ports (+V,-V,GND) to interface with the deployer. In the system, the power connects to the first DC/DC converter to step down from 120V to 24V to power the ToF camera and the PCB board. The PCB board supplies 12V from a power connector to support the Intel NUC, and 5V from the USB connector to support the raspberry pi. The GPS and the monochrome camera receive power from the Intel NUC by its USB 3.0 ports.

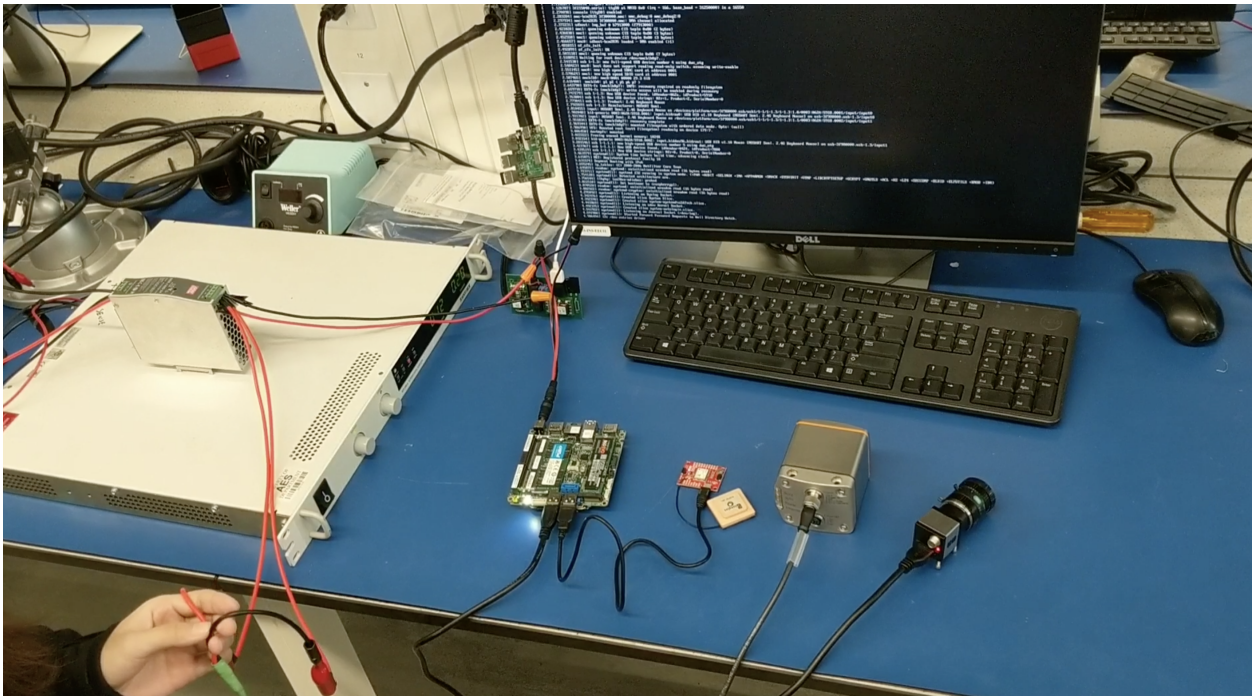

Figure 35: Final Product of the Electrical Power System

Electrical Telemetry System

As shown in Figure.33, there are 11 steps are needed to integrate into the Intel NUC and the raspberry pi. The following table shows the progress of each step.

Steps	Progress	Required actions
Step 1	Incomplete	Dig into the serial connections
Step 2	Complete	Run a command to wake up Intel NUC when needed
Step 3	Incomplete	Integrate an on/off control board
Step 4	Complete	Integrate the batch file into the MATLAB script
Step 5	Incomplete	Needs to decide what kinds of command to be sent
Step 6	Incomplete	Dig into the serial connections
Step 7	Complete	No more extra actions from Electrical team
Step 8	Complete	No more extra actions from Electrical team
Step 9	Complete	Integrate the batch file into the MATLAB script
Step 10	Complete	Integrate the batch file into the MATLAB script
Step 11	Incomplete	Dig into the serial connections

Table 9: Progress of each step and required actions

4.3. Software

VISION's software development is comprised of three high level efforts:

- Integration of heritage code
- Development of state estimation algorithms
- Development of deployment scenario simulation

In order to integrate heritage code which performs the centroid determination, the algorithms must be understood, implemented independently, and then run from a wrapper so that it can operate in conjuncture with the state estimation algorithm. The state estimation algorithm was developed from scratch by the VISION team and required a ground

up system design, development, testing, and implementation. The final manufacturing effort was put into developing the deployment simulation used to test VISION's software package. This includes a truth orbital state simulation that simulates the true orbital states of the deployer and deployed CubeSats. This also includes the implementation of off the shelf software Cinema 4D, which visually simulates CubeSats being deployed, and BlenSor which is another software BlenSor used to simulate the raw sensor measurements expected to be obtained from the TOF camera. This simulation development allows VISION to fully simulate the real dynamics of relative orbital motion as well as the expected raw sensor measurements using software to enable rapid testing of the state estimation algorithm.

4.3.1. Architecture Plan

VANTAGE, the precursor to VISION, provided our team with the centroid determination software. This consisted of automation code for the cameras written in Python and processing code for the point clouds and monochrome images written in MATLAB. The automation code was able to set parameters of the cameras and control when they took pictures. The image processing code went through a series of techniques such as binarization, thresholding, and edge detection to detect the centroid of the system of CubeSats in each frame of the monochrome video. This measurement was combined with the centroid estimates from the TOF (Time of Flight) processing to obtain the most accurate estimation of the position of each CubeSat over time.

The first step to develop VISION's software package is to understand and incorporate VANTAGE's code that was passed down to the team. There is not much architecture to this process but it is crucial to the project development. The team spent a significant amount of time debugging the heritage code to be able to fully run and use it for centroiding. VISION kept most of the centroiding code, but the team did implement small modifications to the data formatting in order to incorporate the state estimation. How VISION utilizes the centroid determination suite as a box with sensor data inputs and outputs the calculated centroids of each deployed CubeSats.

The first part in developing the deployment scenario simulation is to generate true orbital data for the deployer and deployed CubeSats. This simulation is developed to receive and input of deployer orbital elements, VISION's mounting configuration on the deployer, and initial CubeSat deployment states in the sensor frame. From these inputs, the initial conditions for each CubeSat are transformed into the the deployer orbit frame and the inertial frame using the coordinate frame transformation methods already validated when developing the state estimation algorithm. At this point, the inertial conditions for the CubeSat and the Deployer are propagated forward using known Keplerian orbital dynamics. This ensures that the dynamics in the simulation are close to what is expected on orbit. Once true inertial states for all relevant vehicles is computed, the CubeSat inertial truth states must be converted to inertial states relative to the deployer and then transformed into the sensor frame. The true sensor frame positions relative to the deployer can now be used to generate visual CubeSat deployments following this true trajectory using Cinema 4D which is the next step in the simulation. The next part of simulating the deployment scenario is the simulate the sensors. Cinema 4D is used to create files that are directly readable by Cinema 4D's render tool and by BlenSor. Then, Cinema 4D's render tool can be used to create simulated optical camera data, and the BlenSor runner script can be used to create simulated ToF camera data.

The state estimation development effort required all code to be written to match the design described in the design section. This requires the development of supporting functions, frame transformations, and the nonlinear batch filter. The supporting functions included routines that were called in the more complicated frame transformation and nonlinear batch filter algorithms. These are comprise of the DCM construction methods (*from_vec()* and *from_euler()*, the conversion between cartesian states and orbital elements (*get_coe()*), and a routine used to propagate states and covariance in the filter (*propagate()*). The frame rotation algorithm development include the manufacturing of both *sensor_to_orbit()* and *orbit_to_inertial()* used to transform CubeSat states from the sensor frame to the deployer orbit frame and from the deployer orbit frame the inertial frame, respectively. The development of the nonlinear batch filter is broken into individual segments that are validated individually: linearized dynamics model, dynamics propagation, and filter algorithm. Each of these processes are manufactured and validated to function properly individually so to make the debugging process more effective and mitigate the challenges that accompany integrating all of these components into one filter.

4.3.2. Development Process

State Estimation Development

The development of the supporting functions described above was performed using a unit testing approach. This approach required the development of a testing architecture that is run alongside the actual algorithms that completes checks on each subroutine whenever the code is run to ensure that they are working properly. This mitigates the potential challenges of debugging errors that arise when integrating many software components into one. Furthermore,

when developing the supporting functions themselves it allows for each to be debugged easily against a truth case that is used in the unit test.

When developing the frame transformation routines, the mathematical processes derived to perform these transformations had to be checked using hand calculations. Fortunately, the supporting function used in the rotations were already validated via unit testing so the only place for error was the logic itself. Because the frame transformation algorithms developed are novel and tailored specifically to the information VISION has on mission, the transformation algorithms could not be checked using an already known truth case. In order to validate that the transformations are correct, a separate algorithm is written to revert the frame transformations using a different method. By rotating from frame A to frame B using the software package algorithm and then back to frame B to frame A using a separate algorithm allowed for the debugging of the software package algorithm. Hand calculations were also used to validate each step within the frame transformation algorithms. Using these two development methods, the frame transformation algorithms were validated and debugged effectively.

In order to ensure that the general steps in the nonlinear batch filter, which are identical in all nonlinear batch filters, was working correctly, a general nonlinear batch filter algorithm was developed as a black box that can be used to handle any number of different dynamics models and scenarios depending on the inputs. By developing a baseline algorithm that is generic, these common processes within the batch filter were validated using a known dynamics and measurement scenario for a orbit determination scenario. This allowed the general nonlinear batch algorithm to be debugged effectively against a truth case and provided a correctly functioning baseline that could be modified to to meet the specific requirements of VISION's mission. In order to implement the specific dynamical model needed for VISION's mission, the dynamics Jacobian matrix, A, and the measurement sensitivity matrix, H, needed to be derived. These linear approximations of the dynamics and sensor measurements were validated outside of the filter before being implemented by propagating perturbed trajectories using them and comparing the results to the nonlinear dynamics and measurement model. Once the desired behavior was obtained, the linearized model could be implemented in the filter without introducing new errors. Finally, the propagation step within the filter was validated by implementing the filter without measurement update step and observing the nominal trajectory propagated by the filter's propagation subroutine. This was then compared to a known truth propagated using a method that was already validated to ensure that the propagation step did no introduce any errors in the filter with VISION's dynamics. This was a specifically important step in the manufacturing because the propagating step used had to be significantly modified from the general baseline nonlinear batch filter propagation step to accommodate the propagation of deployer states as well as the CubeSat nominal trajectory. With each individual component validated, the state estimation algorithm was finally assembled using components that had been individually validated making the integration of the algorithm's components without significant challenges.

Simulation Development

When developing the orbital truth simulation, the routines validated when developing the state estimation algorithm were leveraged so that work was not repeated. The first process required when developing the orbit simulation was to transform the initial conditions in the sensor frame into the inertial frame relative to the inertial frame to that they can be propagated using known orbital dynamics equations. This was done using the already validated frame transformation methods developed for the state estimation algorithm. The next step was to propagate the inertial initiation conditions of the deployer and the CubeSats and then use these inertial states to obtain relative orbit frame states. In order to ensure that this step was correct, the inertial initial conditions were propagated using Keplerian orbit equations of motion. The states of the CubeSats were then found relative to the deployer in the inertial frame and the same code used to validate the coordinate frame transformations was used to transform the relative inertial states to relative orbit frame states. The true orbit frame of the CubeSats relative to the deployer were then propagated forward using the general equations of relative orbital motion and the results were compared to those found using the inertial states. Once the results from these two methods matched, the propagation and conversion from inertial state to orbit frame was confirmed. Then the orbit frame states were transformed into the sensor frame using the routine already developed for validating the coordinate frame transformations used in the state estimation algorithm. This proved to be a surprisingly smooth manufacturing process because a lot of modular code that had already been validated was used to develop the simulation. Much of the sensor simulation code was moved into the simulation code or modified to meet VISION's requirements. VANTAGE's code for running Cinema 4D took in arguments for running a simulated straight-line trajectory, in which the trajectory was calculated in the runner script via start and end location. VISION required that the trajectories followed orbital dynamics. Therefore, this code was modified to read in data from a .csv file instead. Additionally, part of VANTAGE's code determined the start location of the CubeSats in a deployer-centered frame in the proper coordinate frame for Cinema 4D, so the relevant methods were moved into the simulation code before the .csv with centroid locations was generated.

The BlenSor script written by VANTAGE was generally functional for VISION's goals. The bulk of the work done

with this script was to automate the process. The script and process flow were modified so that the filename used in the script didn't need to be modified on each run, and a batch script was created to automate the entire process of running BlenSor.

Centroid Determination Development
In order to confirm that VANTAGE's code was being run correctly we attempted to compare our results to what they received last year. The first step was to ensure we were generating the same simulations as VANTAGE was, and in the beginning we were getting wildly different results that were easily determined to be wrong simply by watching the video. This error was attempted to be debugged by nearly half of our time with no luck, and members of VANTAGE were unable to help over email. Eventually we set up a face to face meeting with a member of VANTAGE and were able to solve the issue. The issue ended up being a cube that C4D automatically placed in the simulation when the program was opened. This was a known problem to last year's team, however when they wrote code to automatically delete the cube on startup they failed to note that change in the README or upload the code. This opened our eyes to the challenge of reusing code for heritage projects. None of the members of our team had experience with C4D so debugging was very challenging, but after talking to last year's team briefly they were able to solve the issue. This was not the only issue getting heritage code running, in fact the TOF automation code proved to be very difficult as well. We were very quickly able to get the monochrome camera automated. We were able to change parameters like aperture and exposure time, and take pictures by just running a script on a laptop. Since the monochrome camera code was ready to go out of the box we expected that the TOF would be straight forward as well, but it ended up being very time consuming. The culprit ended up being a package that didn't function properly in Python 3, however since we needed Python 3 for the state estimation we had to find a workaround. Other packages were researched and used, but none ended up saving the correct point cloud data. Unfortunately production was halted before we were able to get the TOF fully automated with code, however we were able to see a live stream of the point clouds on the laptop it was connected to. This was the most difficult step for software, and after this we only had to upload the automation code to the NUC to do a system test.

4.3.3. Software Integration

In order for the software subsystem to function as a whole, VISION must integrate the individual software efforts in one fluent process. This includes the collection of sensor data, centroid determination, state estimation filter, and TLE generation.

The first component in the integration process is the sensor data collection. VISION has two types of sensor data: simulation and experimental, but there is not much of a difference in the way the data is handled and processed. For the simulated sensor data, Cinema 4D and BlenSor were already functional; however, some changes were made to make the process of using them smoother. The processes for running both programs were modified so that files didn't have to be moved and so that file names in the code no longer needed to be changed between runs. Instead, a working "current" directory was created to store all files being generated by the current data processing run. The Python script run by Cinema 4D was modified to read in the data created by the new simulation script. Finally, a batch script was created to fully automate the process of running BlenSor with no user interaction besides running and closing the script.

For the experimental sensor data, there will be a main driver script that calls the monochrome and TOF camera to start data collection. This period will last for 10 seconds. The data will be stored in a folder along with the manifest that will be read in by the centroid determination code and ultimately processed. The folder will be wiped before every deployment cycle for a clean directory. After the centroids are found, the processed data will be saved and read by the state estimation unit, and finally the TLE generation will follow suit.

Below is a simple pseudocode of the automation process. Before the script is called, the NUC will be powered on which will power on the other sensors as well. On startup, the batch script will be executed as displayed below:

Wake up NUC and all sensors

Open batch script on startup

Algorithm 4: VISION Flight Software Automation

```
folder = "D:/VISION/SensorData" ;                          // location of sensor data
if receive deployment signal then
    monoCam(folder) ;                    // start data collection monochrome camera
    TOFcam(folder) ;                            // start data collection TOF camera
    sleep(30) ;                                              // time sleep 30 seconds
    centroidMain() ;                // call centroid determination code main script
    sleep(120) ;                                             // time sleep 2 minutes
    stateEst() ;                                 // call state estimation main script
    sleep(60) ;                                               // time sleep 1 minute
    TLEgen() ;                                          // call TLE generation script
    relay() ;            // transfer calculation to deployer and enter sleep mode
else
    remain idle;
```

At the time the project halted, the monochrome camera was successfully operated using Python code on a laptop, and the TOF camera was able to show point clouds live on a laptop however they were unable to be saved. Because of this problem, the software integration process was halted and this is where we left off when the project halted. The batch script has not be fully written due to the delay in the TOF camera data collection. Windows and Python were successfully installed on the NUC, however the data processing code was not uploaded or run on the NUC.

4.4. System Integration

As a continuation project, System Integration was a critical part of the VISION package. The software for VISION can be separated into three separate sets, excluding integration. First, the centroid determination, or CubeSat tracking, code came from the VANTAGE team. The second set is automation, where the processes performed by the sensors and the post processing is called upon without human interference. Finally, the third set is the state estimation, where a non-linear batch filter is implemented to improve the relative state estimation of the CubeSats. The integration of all of this, mainly discussed above, is only a part of the system integration, however.

The entire integration process begins with the Raspberry Pi, which sends a boot command to the Intel NUC. As the NUC turns on, power is provided through its USB ports to the monochrome camera and the GPS board, which boot whenever a nominal power supply is supplied to them. The GPS antenna, a ceramic LNA patch antenna, is connected directly to the GPS board, and is powered by the board itself. Next, the TOF is supplied 24 Volts and 5 amps, and takes about 25-30 seconds to boot up. As these components are starting up, a two minute wait period is built in. This is primarily to allow the GPS to lock onto enough satellites to produce a location estimate. During testing of the GPS, two minutes was found to be more than enough under typical ground scenarios with a good, unobstructed view of the sky.

Once the two minute wait period is completed, the system is ready to collect data. In space, the data collection period will begin with a signal from the deployer itself. For the purposes of ground testing, this would be a command sent from a PC to the NUC on-board. From here, the TOF, monochrome, and GPS are collecting and saving data to the NUC. The post processing begins two minutes after the deployment begins, giving the system ample time to save the data to the NUC. The initial post processing consists of TOF and monochrome centroiding from the VANTAGE team in 2018/2019. The NUC gives two minutes for this process to occur, before calling the state estimation script, which uses a nonlinear batch filter to produce a state estimate for the CubeSats. Finally, the TLE generation occurs, which uses timing from the GPS device to propagate orbits and deliver them to the deployer, along with images of the CubeSats during deployment.

In order for this process to occur, the software must also be connected to the hardware. It is critical to know the location of the monochrome and TOF relative to each other, so that cross track corrections from the monochrome can properly aid the TOF.

5. Verification and Validation

Matt van den Heever, Cameron Baldwin, Adrian Perez, Ben Hagenau, Max Audick,& Theodore Trozinski

5.1. Background

The VISION team used a tailored version of the aerospace industry standard verification and validation V model to help guide and dictate the operations of systems engineering throughout the project. Using a flowdown starting from the customer's needs or requirements, a project scope was then defined, giving the needed inputs to then define the system specification. On the left side of the V, requirements starting from the customer down to the allocated component levels dictated the designs of the entire system and so forth. At the bottom of the V, we started production of all software and hardware. And on the right side of the V, as the tests are performed starting from the component level all the way up to the system level, the process of verification that our design complies with all levels of requirements defines the most important outputs of our testing and designing phases. Once all designs have been verified by test, the last step then includes validation that the design meets the customers overall requirements, and delivers a compliant product. The following section will now describe the individual verification of system requirements handled by each test.

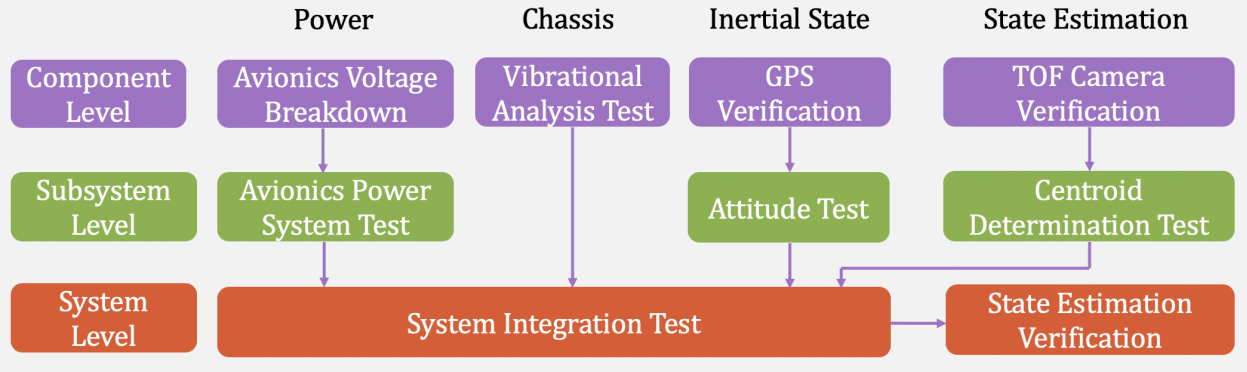

Figure 36: Test Plan Schematic

5.2. Vibrational Environmental Testing

5.2.1. Motivation

In order to meet requirement DR-6.1

- *DR-6.1:Chassis structure shall reach an advanced TRL certification, or equivalent.*

The structure had to be tested in a relevant environment. Since a flight-ready VISION system would need to be launched into orbit, the team would have needed to make sure the chassis structure is capable of handling the vibrational loads of a launch. The team performed a vibrational analysis predicting the first 5 resonant frequencies and would have used the vibrational test to verify those predicted frequencies to include in an ICD.

5.2.2. Procedure

The first step was an FEM analysis of the chassis. This was to determine the resonant frequencies in order to test integrity. They were verified in FEM using multiple meshes and checking convergence, as well as a simplified cantilevered beam test. The resonant frequencies are shown below:

Mode	1	2	3	4	5
Frequency [Hz]	887.73	926.21	1,020.9	1,044.1	1,132

The team would then have manufactured an adapter plate to interface between the vibration table in the PILOT lab and the VISION system. The first test would be a frequency sweep on the structure. The vibration of the structure would have been monitored for the entirety of the test using accelerometers and the strobe method. After this, the resonant frequencies would be tested as well. Finally, an Acceleration Spectral Density test would take place. This is a measure of acceleration versus frequency, and can help model launch environments.

5.2.3. Expected Results

The expected results would have been acceleration data for the frequency sweep, with increases near the resonant frequencies. It would also be expected to see mode shapes from the FEM analysis. An example of these is shown in Figure 37. Using the strobe, these vibration shapes could be seen and verified. This would also help in determining component placement. For all of these tests, the team would also check for any structural damage, as well as the torque of the screws before and after the tests. If necessary, damping could be put in place to reduce vibrational loadings.

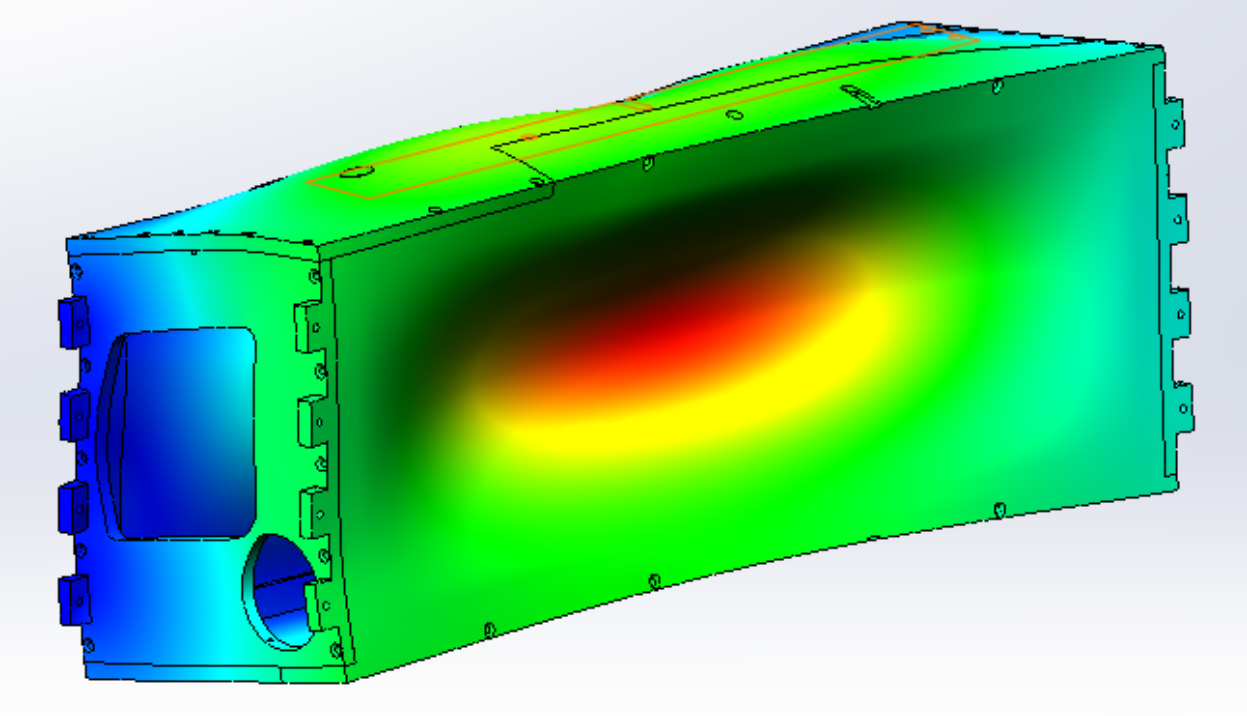

Figure 37: Example Mode Shape (Mode 1)

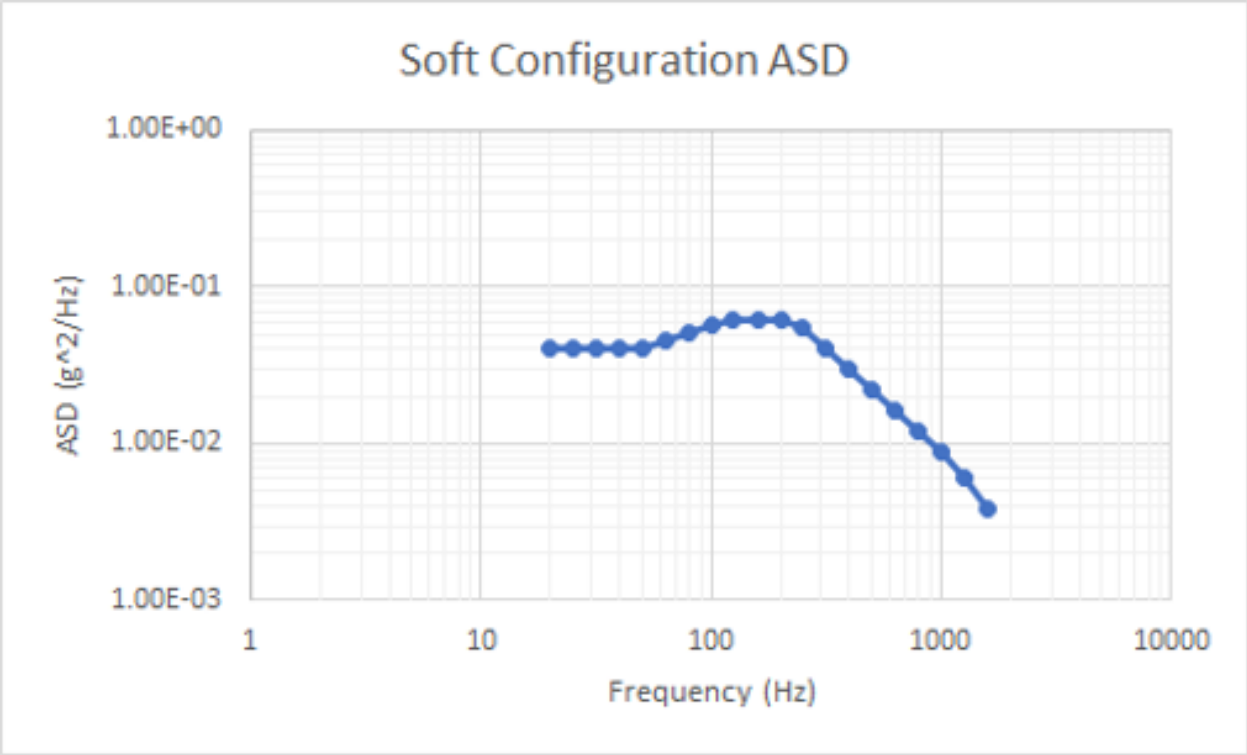

Figure 38: Example ASD (NanoRacks Launch)

5.2.4. Measurement Uncertainties

The major uncertainties in the testing method would have been the vibration control and vibration measurement device. During the test, the vibration table could be actuating a different frequency than is being commanded which could have caused issues determining what frequencies caused resonance.

5.3. Avionics Testing

5.3.1. Motivation

In order to meet requirements:

- *DR-4.2:The system shall operate with no more than 120 VDC, 3 Vpp ripple voltage, and 5 A.*
- *DR-4.3: The system shall draw no more than 520 Watts.*

The entire avionics subsystem must be tested using through the use of volt and ammeters. VISION must be able to utilize 120 VDC to power all electronic components, including the visual and time of flight cameras, an on board processor (Intel NUC), micro-controller (Raspberry Pi) and GPS.

5.3.2. Procedure

First the voltage was stepped down from 120 volts to 24 volts in order to operate the TOF camera. This was done using a DC/DC converter. The team tested each of the respective voltage step downs using the respective DC/DC converter, a breadboard, power resistors, and an oscilloscope. The next voltage step downs were from 24V to 5 volts to power the Raspberry Pi, and 24 volts to 12 volts to power our Intel NUC. Once verified, we replaced the breadboard with a printed circuit board and verified the voltages again. To test the power draw of the system We chose 4 amps, which results in 480 watts, which is less than the 520 watt we would have access to when integrating with a launch provider like NanoRacks. Finally, all the sensors were connected and powered on to tested the systems ability to power our entire system.

5.3.3. Results

After completing the avionics voltage break down test verifying that all of the output voltages were within the acceptable operational voltage ranges for each of our hardware components, the team performed the final avionic test. This was done by checking the outputs of the power distribution PCB and then powering all of the hardware. The operational voltage range for the time of flight camera is between 20.4 volts and 28.4 volts. The team obtained an output voltage of 24.9 volts. Likewise the Raspberry Pi has an operational voltage range between 4.7 volts to 5.2 volts, and the team found an output voltage of 4.8 volts. The Intel NUC has an operational voltage of 12 volts to 24 volts and the team measured an output voltage of 12.3 volts. After verifying that the PCB outputted the proper voltages and would not damage any components, it was connected and tested in the final avionics test as shown in figure 39. The result was that the system was able to power on all of the teams hardware without any issues, even when providing the system with 480 watts, which is 40 less than we would have access to.

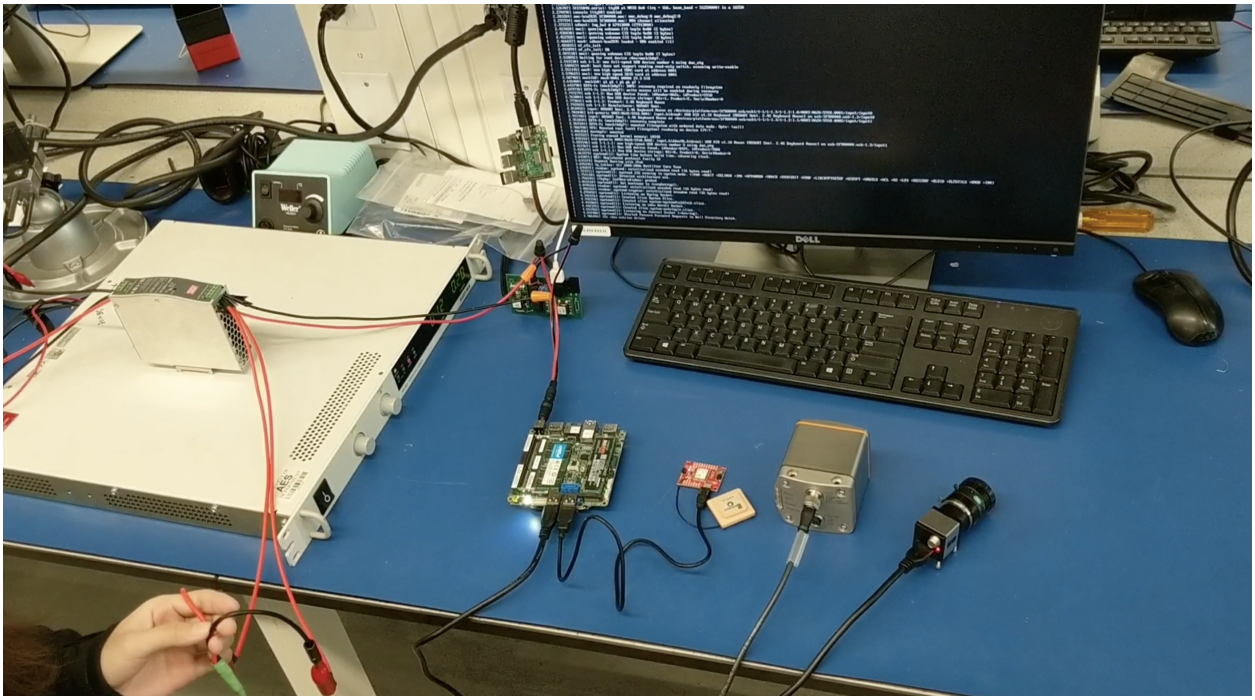

Figure 39: Final Avionics Test

5.3.4. Measurement Uncertainties

The main source of uncertainty in the measurements we made as a team stem from errors associated with the oscilloscopes measurements. Essentially the measurements made by the team are only as accurate as those that can be made by the oscilloscope. How ever, the oscilloscope was able to measure our voltages to multiple decimal points, therefore there is very little uncertainty associated with this test.

5.4. Image Processing and Centroid Determination Testing

5.4.1. Motivation

In order to meet requirements:

- *DR-1.1: VISION shall characterize and differentiate up to six CubeSats of sizes between 1U and 3U*
- *DR-1.3: VISION shall utilize a deployment manifest provided by the deployer to identify each object*
- *DR-3.1: VISION shall deliver a still image of each individual CubeSat in a deployment*

The Image Processing and Centroid Determination testing contains a test of two systems– the monochrome camera and image post processing software, and the Time of Flight camera and its centroiding software. A test of the visual

camera and image processing system would have been performed first. Centroid determination from the monochrome camera is critical correct the Time of Flight camera in the cross-plane. The main source of measurement in the along-track direction would have been made by the time of flight camera, however the aforementioned sensor is not as accurate at measuring changes in the cross-plane, similar to a calibration.

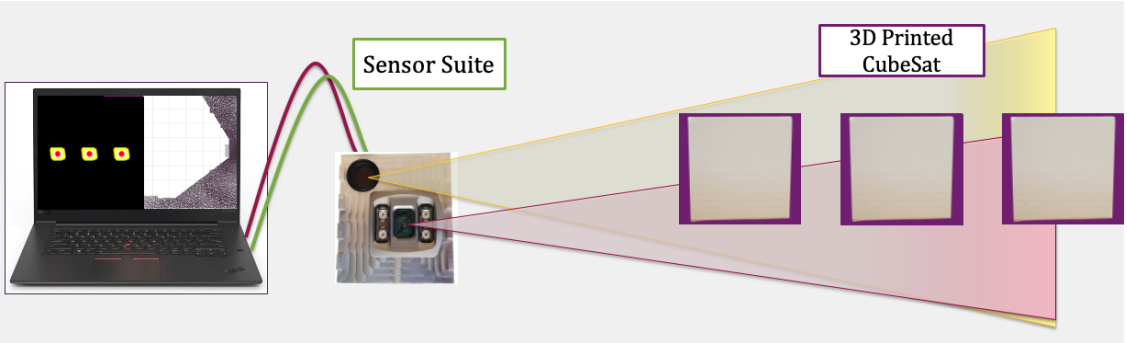

Figure 40: Centroid Determination Test

5.4.2. Procedure

The image processing and centroid determination test is a multistage test which will be performed in the senior projects room. The first stage involves orienting a CubeSat of varying sizes (1U-3U) in numerous different orientations and capturing images with the visual camera. These images will be run through the image processing software in order to extract an estimation of the location of the centroid of each of these objects. The estimated centroid location for each orientation would have been compared to a manually determined centroid location which the team would have made prior to the data capture.

This would have allowed the team to validate the fidelity of the centroid determination software, including OpenCV. The second stage of this test is test a multi-body system, in which multiple mock CubeSats are placed in a variety of orientations. The visual camera and time of flight would have imaged the multi-body system and utilized the captured data, along with a deployment manifest and the teams image processing software to first differentiate between the CubeSats in each test, and then to determine a centroid for each mock CubeSat in the test. If the location of the mock CubeSats did not allow for the software to determine a centroid for each individual CubeSat, the software would have determined a centroid for the system of CubeSats which would instead be utilized for the necessary cross-plane corrections.

5.4.3. Expected Results

The team expected for the image processing software to have been able to differentiate all imaged CubeSats within the ten meters of the visual camera and to have been able to determine the centroid of the CubeSats in various backgrounds.

5.4.4. Measurement Uncertainties

The largest uncertainty during this test would have stemmed from manually determining the location of a centroid for each test. This could have been mitigated by using the Vicon system in the Aspen lab by determining the centroid of the each of the mock CubeSats through the use of fiducial markers or by orienting the mock CubeSats in such a way that the centroid can be more easily determined, such as by first orienting the CubeSats with a single face in the camera view and then by simply rotating the mock CubeSats such that the centroids location do not differ.

5.5. Integrated System Testing

5.5.1. Motivation

In order to meet requirements:

- *DR-2.7: VISION shall calculate and package TLE estimates within 15 minutes of the end the deployment sequence. Note: Relevant data collection during the deployment sequence is considered finished when all CubeSats can be differentiated by more than1 pixel between frames*
- *DR-4.1: The chassis envelope shall enclose all components, excluding protruding instrument sensors*
- *DR-4.4: The system shall store images, raw data,and estimates of one deployment cycle,on-board, for the duration of the data processing and down link period*

An integrated system test would have been performed. The integrated system test would have verified all of VISION's developed software as well as the performance of all avionics and sensors. The test would have also provided the team with data from the sensors which would have been used to quantify the uncertainty profiles of the sensors which is needed as an input for the state estimation software.

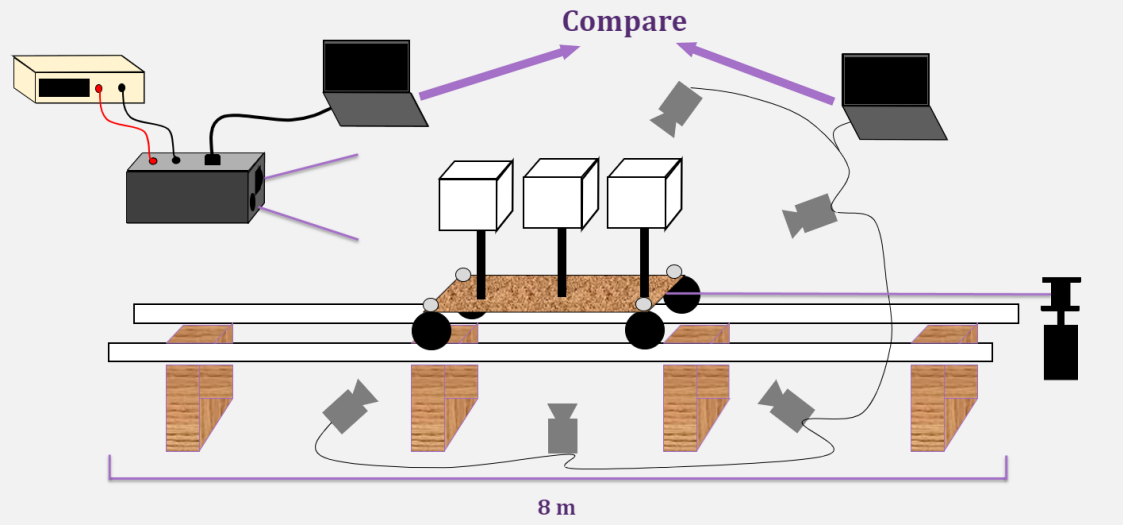

Figure 41: Integrated Systems Test Theoretical Setup

5.5.2. Procedure

The integrated system test utilizes a winch and rail system which would have been driven by a DC motor. An image of a segment of the rail system with a single cart can be seen in figure below.

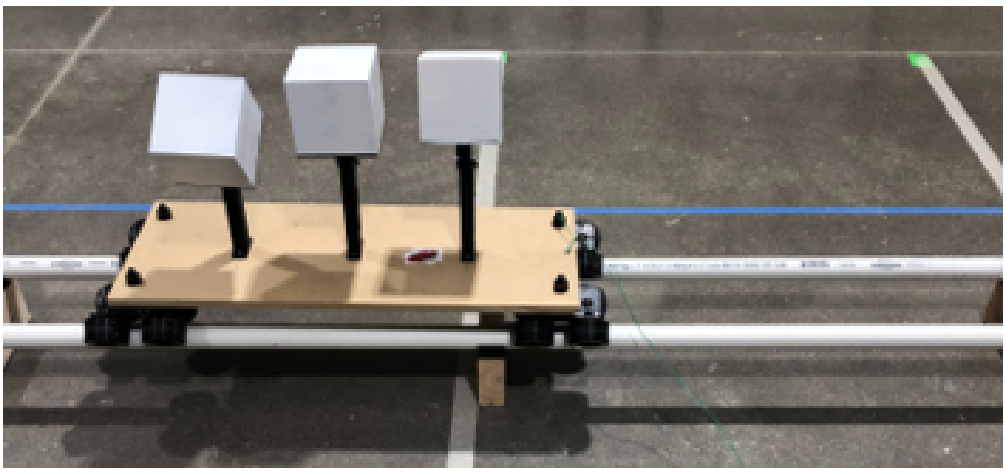

Figure 42: Integrated Systems Testing Rail

The rails were composed of pvc pipe and connectors which were sanded down to be level with the rest of the pipe. The stands were constructed out of 2x4's with 3D printed holding parts which were screwed into the wood, and

provide a grove for the rails to sit within. The motor, which is attached to the winch, will be given a specific voltage to drive it at a constant number of revolutions per minute in order to allow the winch to pull a cart at a constant speed, which will be verified for different tests, between speeds of 0.5 and 2 meters per second. A separate system, the Vicon system will also capture the motion of the cart with sub-millimeter accuracy. The measurements made by the Vicon system will be compared to the determined relative position and velocity determined by the VISION package and the teams developed time of flight software and image processing software. This test would have also allowed the team to verify the time it takes for all on-board processing and calculations to be calculated, stored and reported.

5.5.3. Expected Results

The team expected the on-board processing to be completed within 15 minutes after deployment. This test would have also been able to prove the ability of VISION's sensors to differentiate between all the mock CubeSats during the deployment. Finally the team expected this test of further verify many of the subsystem test such as the avionics power test by proving all the hardware would indeed be powered by 120VDC and 520 watts.

5.5.4. Measurement Uncertainties

The uncertainty in this test is extremely small since the Vicon system has an extremely accurate sensor profile. Most of the uncertainty would have stemmed from man made errors in the setting up of the Vicon system such as errors in leveling or straitening the rails, or errors made in setting the origin for the system.

In order to validate that the results from the sensor simulation are significant, VISION must show that the error profile of the simulated sensor measurements matches that of the true sensor hardware results. In order to do this, a large number of hardware tests are performed and the error at each time for each test is computed using the Vicon lab truth. Accumulating a large number centroid measurement errors over each test, the Anderson-Darling test can be used to ensure that the centroid measurement error is indeed gaussianly distributed. This is important because the nonlinear batch filter is derived under the assumption that the measurement error is sampled from a gaussian distribution. Once a gaussian distribution is confirmed, the standard deviation for the centroid measurement error is computed and the results are compared to that produced by the sensor simulation. The sensor simulation is then adjusted to have the same error distribution so that the results form tests that use the sensor simulation can be considered reflective of the real world. This test was not performed in because of the COVID-19 pandemic however preliminary results were updated using a small number of data sets collected by last year's team. The recovered distribution results from the Anderson-Darling test are shown below. The figure shows the error distribution in the positions along each of the sensor frame axis.

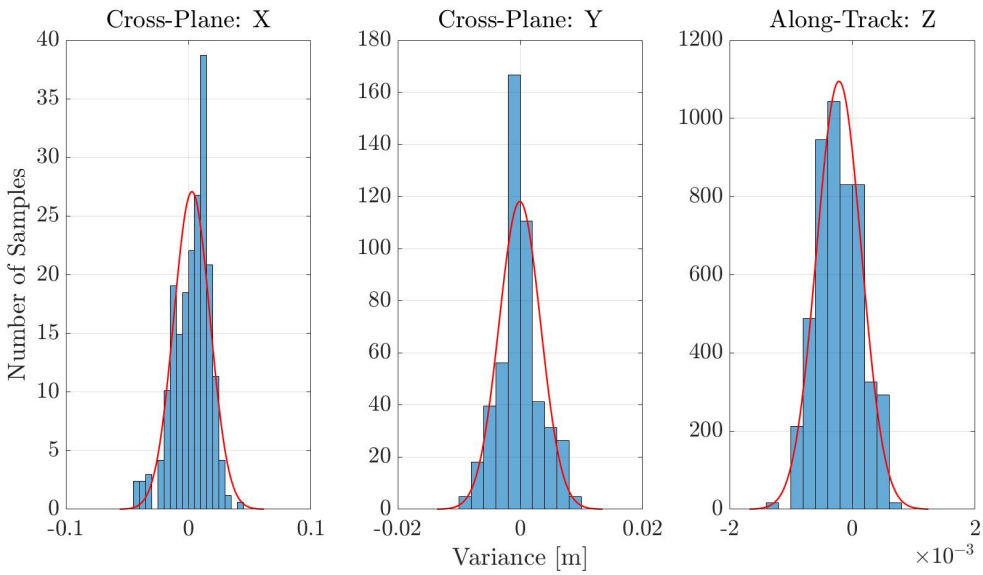

Figure 43: Anderson-Darling Test Distributions

The quantified results show that error distributions are 80% gaussian with a 95% significance level. This near gaussian distribution is expected because gaussian distributions do not exist perfectly in the real world. The computed

standard deviations are 0.03, 0.01, and 0.01 meters for the sensor frame x, y, and z axis respectively. These results should not be taken as accurate because this analysis was performed on a small number of measurements. In order to obtain useful results, a significantly larger number of measurements should be used.

5.6. Software

5.6.1. Motivation

The goal of software testing is to validate DR-2.2 and DR-2.4

- *DR-2.2:VISION shall estimate the orbit frame position of each CubeSat such that the estimate covariance conforms to the Position Uncertainty Map.*
- *DR-2.6: VISION shall produce Two-Line Elements for each deployed CubeSat.*

Which state requirements on the accuracy of the state estimations produced by VISION and to ensure that the TLE produced are complete and correctly formatted. The test can be described by five main steps: simulating true orbital data, simulating sensor measurements, centroid determination, state estimation, and results analysis. In order to obtain significant results a large number of deployment scenarios were analyzed. Due to the COVID-19 pandemic, only 25 scenarios were studied. Furthermore, many different deployment conditions are studied to encapsulate all cases that are expected for a for a deployment into an Earth bound orbit. This requires simulated deployments from deployer orbits that have eccentricities ranging from 0.0 to 0.9 and semi-major axis ranging from 7000 km to 10000 km. Due to the COVID-19 pandemic, test results were only produced for eccentricities ranging from 0.0 to 0.25 and deployer semi-major axis of 7000 km. This covers the behavior expected for any Low Earth Orbit (LEO) deployment.

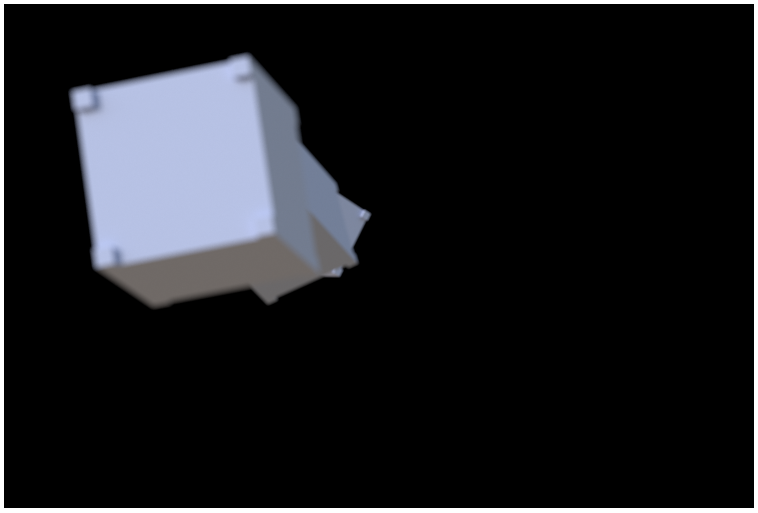

Figure 44: Cinema4d Sample Output Frame

5.6.2. Procedure

The first step in the test procedure is to generate true orbital data for deployed CubeSats and the deployer. This is done using the true orbital state simulation. The test controls are the deployer orbital elements, VISION's mounting position and orientation on the deployer, and CubeSat initial conditions in the sensor frame. When generating truth data, deployer orbital elements are varied to cover all potential Earth bound deployment scenarios. The two elements controlled to do this are the semi-major axis and eccentricity varied to cover the range described above. The dynamics simulation produces a set of states representing "truth" data for each simulated satellite's centroid positions over time. These are processed by a Python script run via Cinema 4D to create a .c4d file and a .fbx file that each contain that same data in formats that are directly readable by Cinema 4D and BlenSor, respectively. Cinema 4D's render feature can then be used to create simulated optical camera data, while BlenSor can be run via a batch script to create simulated ToF camera data. Next, a modified version of VANTAGE's image processing code is run to produce a set of states representing "measurement" data for each simulated satellite's centroid positions over time. With centroid measurements in the sensor frame, the state estimation algorithm is used to process the centroids and produce TLE

for each CubeSat. In the process, intermediate values are extracted from the state estimation algorithm to check the validation of requirements. The orbit frame relative state covariance and estimate error are output. These are used to perform truth model [2] analysis, visually validate the the errors and covariances match, and to validate DR-2.2 using the Uncertainty Map. Additionally, the computed orbital elements extracted to be compared against the truth to observe how the state estimate error propagates to the computed orbital elements. To better study the performance of the filter, the propagation error is studied in more detail by propagating forward the state estimates and comparing them to the truth. In order to meet requirement DR-2.6, the TLE produced by VISION would be propagated using the Special General Perturbations 4 (SGP4) propagator which is the formalized propagator of TLE, to ensure that the TLE produced satisfy the required format. This was not tested due to the COVID-19 pandemic.

5.6.3. Estimate Accuracy

The uncertainty map requirement is defined such that relative orbit frame position estimates sampled from a gaussian distribution described by a combination of along-track standard deviations and cross-plane standard deviations that lie in the green region, then estimates with errors sampled from those distributions, when propagated forward three orbits, can be tracked by the official ground station with the smallest known field of view. This conservatively assumes that the error in the velocity is sampled from a deviation that is found to be two times that of the position uncertainty. The results for the 25 data sets produced lie in the highlighted purple region with a max along-track standard deviation of 0.31 meters and max cross-plane standard deviation of 0.29 meters. These results prove to satisfy the requirement DR-2.2.

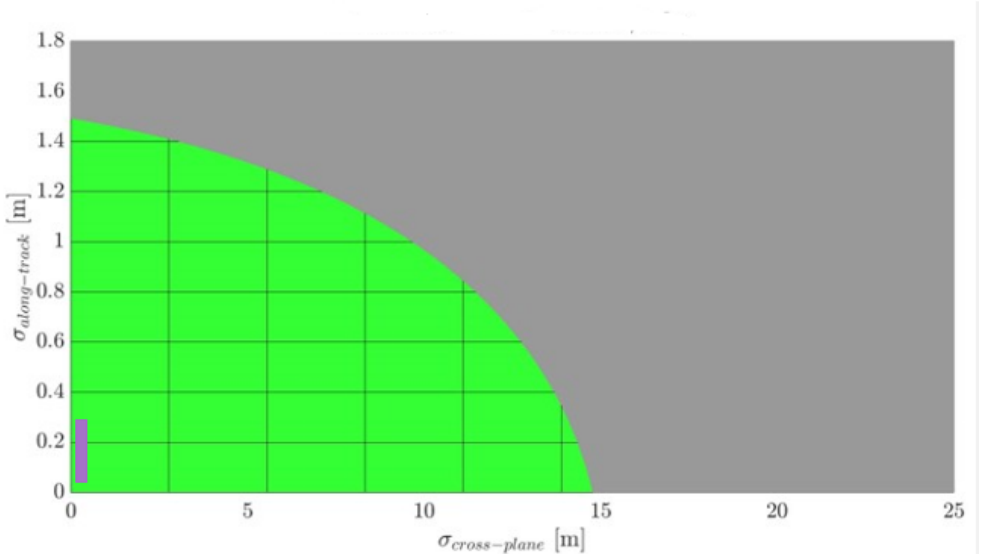

Figure 45: Uncertainty Map Results

In order to show the functioning of VISION's software package, the results after each of the major steps of the filter are shown in the following figures for a representative deployment scenario. Figure 46 depicts the black truth state used to compute the simulated sensor measurements and the blue centroids computed using heritage code to determine the centroids of the CubeSat at each measurement time.

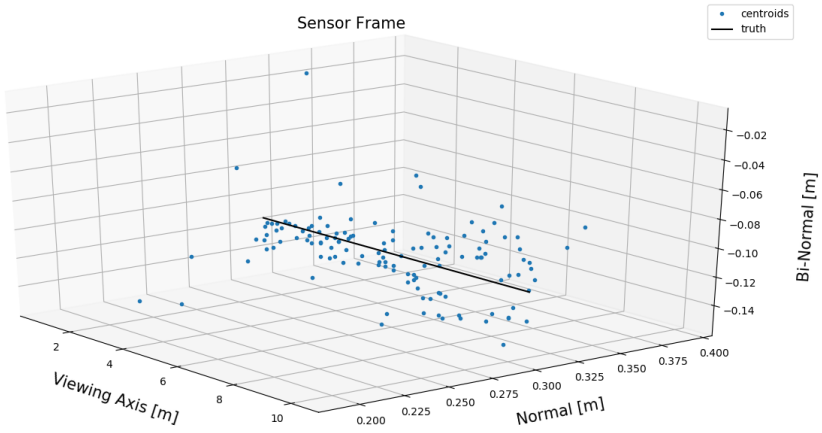

Figure 46: Sensor Frame Centroid Measurements

Figure 47 depicts the true black trajectory, the blue centroid measurements, and now orange centroid estimates in the orbit frame. As can be seen, the accuracy of the estimates is significantly higher than that of the centroid measurements as the estimates stick very near to the black truth line.

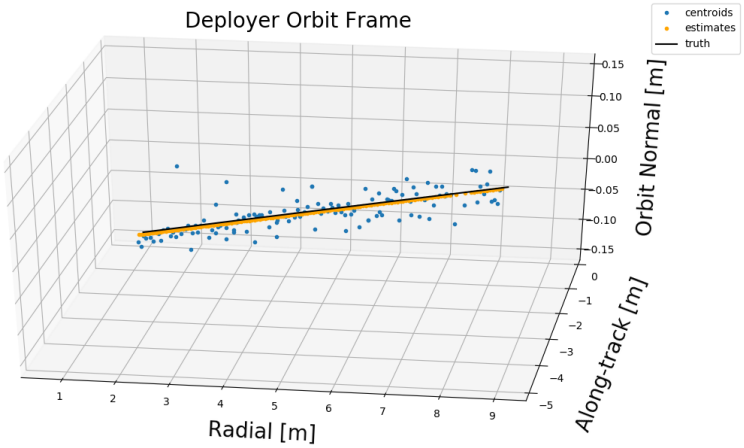

Figure 47: Orbit Frame Estimates

Using the truth model, the errors in the position and velocity estimates are computed. These errors are shown along with their 3σ covaraince bounds produced by the filter in Figures 48 and 49, respectively.

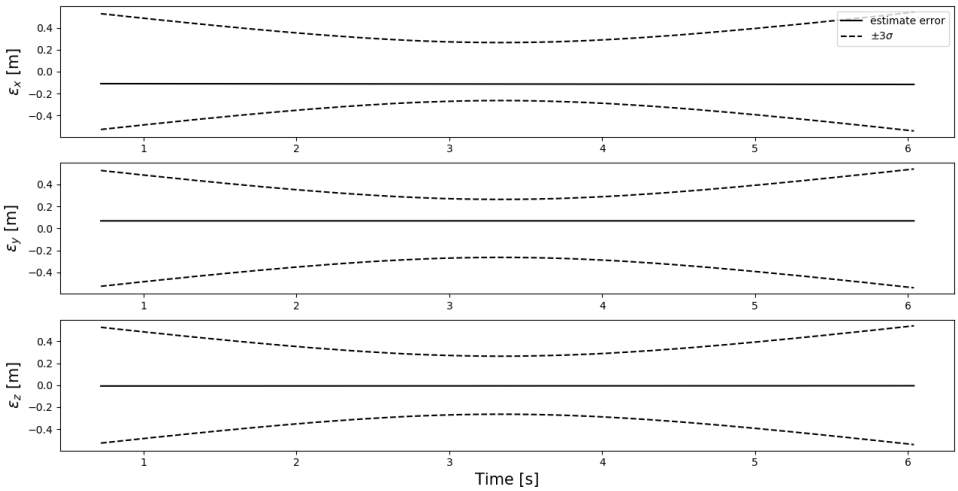

Figure 48: Orbit Frame Position Estimate Error

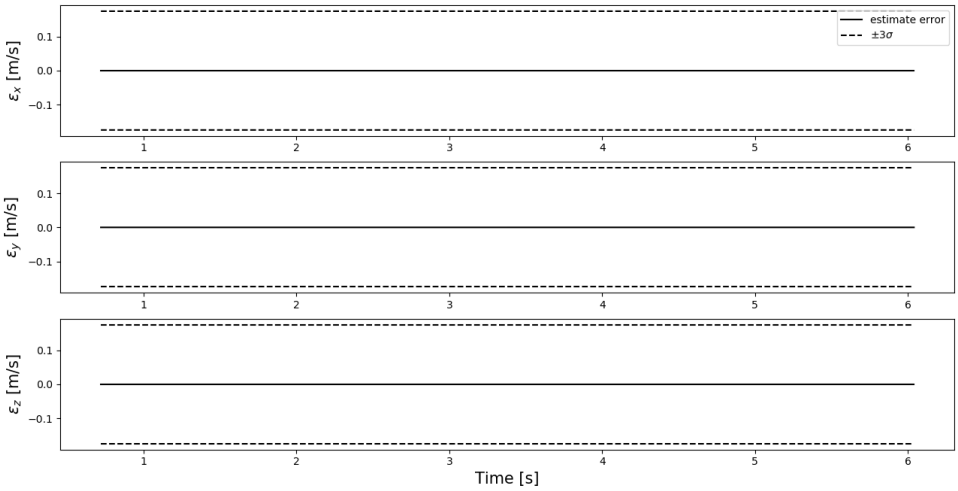

Figure 49: Orbit Frame Velocity Estimate Error

After transforming these estimates into inertial states they are propagated forward using the J2 perturbation to a Keplerian orbit to produce the following orbital motion. This is shown in Fig. 50.

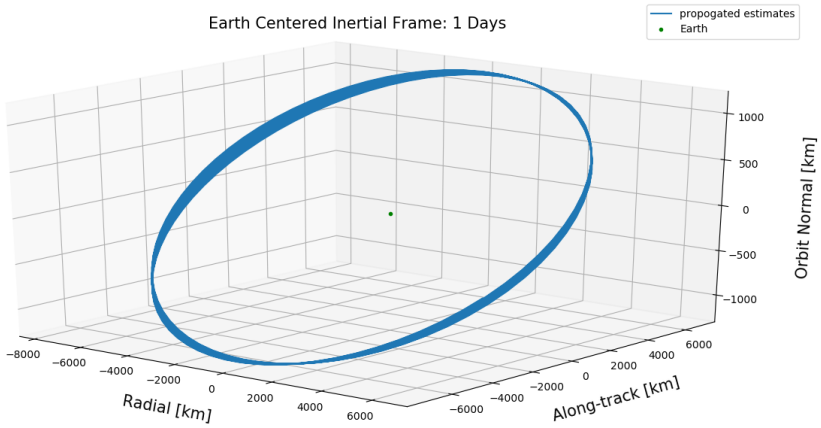

Figure 50: Inertial orbit

5.6.4. χ^2 Testing

In order to ensure that the filter results are valid, the state estimate error is used to compute the normalized estimate error squared (NEES) statistic at each time step. This when sampled for a high number of simulations from a zero mean gaussian distribution, the NEES statistic should create a χ^2 distribution of degrees of freedom equal to the number of elements in the estimated state, in this case 6. This must be the case it the filter is working correctly. For a nonlinear batch filter, the only directly estimated state is the initial condition so the NEES statistic is computed for the initial condition over the 25 simulates run. Filter input parameters, initial state covariance and and measurement covariance are then modified until the NEES statistic matches the desired results thereby tuning the filter. Conceptually, satisfying the NEES statistic requirement of forming a χ^2 distribution defined by a degrees of freedom of 6, the filter is shown the have covariance bounds that accurately reflect the error that they describe. This is important in ensuring that the filter is no diverging but also that it is not over or under confident in its estimates. The results from the 25 deployment scenarios after the filter was tuned showed that the filter converges to the correct number of degrees of freedom, converging to degrees of freedom of 5.9389. Unfortunately, in order for these results to be significant over double the number of deployment scenarios sets used would be required. The tuned measurement covariance matrix found to meet the requirements is shown in Eq. 5.

$$R = \begin{bmatrix} 0.2^2, 0, 0 \\ 0, 0.3^2, 0 \\ 0, 0, 0.4^2 \end{bmatrix} \tag{5}$$

Figures 51 and 52 presents the error and covariance in the initial condition produced by the filter for the 25 deployment scenarios analysed for position and velocity, respectively. These errors and covariance are used to compute the NEES statistic directly for each iteration. As a sanity check, these figures depict 3σ bounds and errors visually resemble a gaussian distribution.

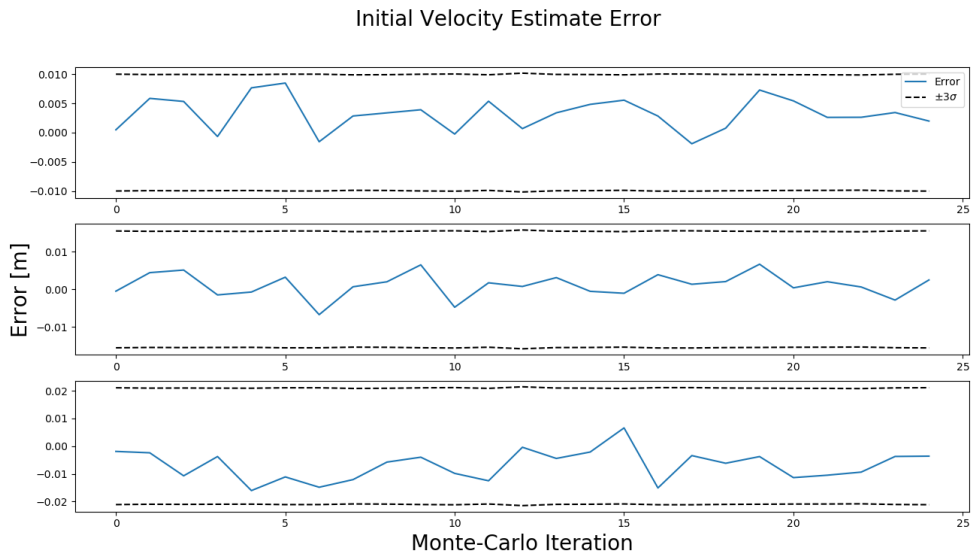

Figure 51: Position NEES Rsults

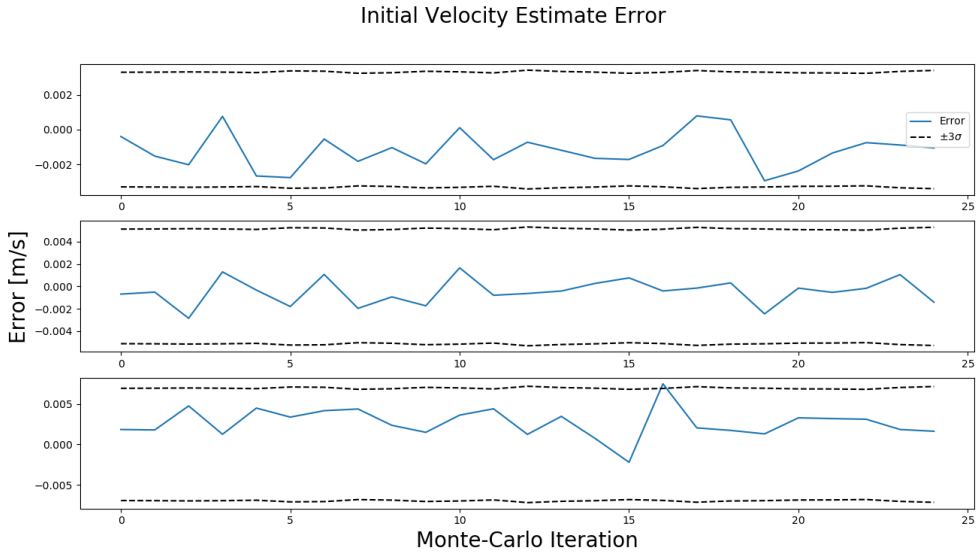

Figure 52: Velocity NEES Rsults

It is important to note that this tuning parameter changes significantly with when the certainty in the initial condition is changed as that directly effects the performance of the filter. It is recommended that a base line is found when working on the filter in the future to re-tune the filter to satisfy the χ^2 results.

5.6.5. Two-Line Element Assembly

The components estimated in the TLE using the data collected are the orbital elements of the deployed CubeSat. Over the 25 deployment scenarios studied, the error and standard deviation in the orbital element estimates are in Table 10.

Table 10: Orbital Element Estimate Error Profiles

Element	Mean Error	Standard Deviation	units
semi-major axis	2.0	1.4	meters
eccentricity	$2.4e^{-7}$	$1.8e^{-7}$	N/A
inclination	$3.2e^{-7}$	$1.7e^{-7}$	degrees
right ascension of the ascending node	$6.9e^{-7}$	$3.7e^{-7}$	degrees
argument of periapsis	$2.3e^{-4}$	$5.2e^{-4}$	degrees
true anomaly	$2.3e^{-4}$	$5.2e^{-4}$	degrees

Due to testing ending early because of the COVID-19 pandemic, the TLE were never run using the SGP4 propagation tool to validate the formatting of the TLE. For this reason, DR-2.4 remains untested

5.6.6. Propagation Error

Each set of orbital elements computed using the 25 deployment scenarios was propagated forward to quantify to propagation error. A representative plot of these results for a scenario with semi-major axis error of 1.27 meters is shown in Fig. 53. The propagation error is the black line and the dashed lines are the $\pm 3\sigma$ bounds propagated forward in time. This example shows a propagation error of about 175 meters per day.

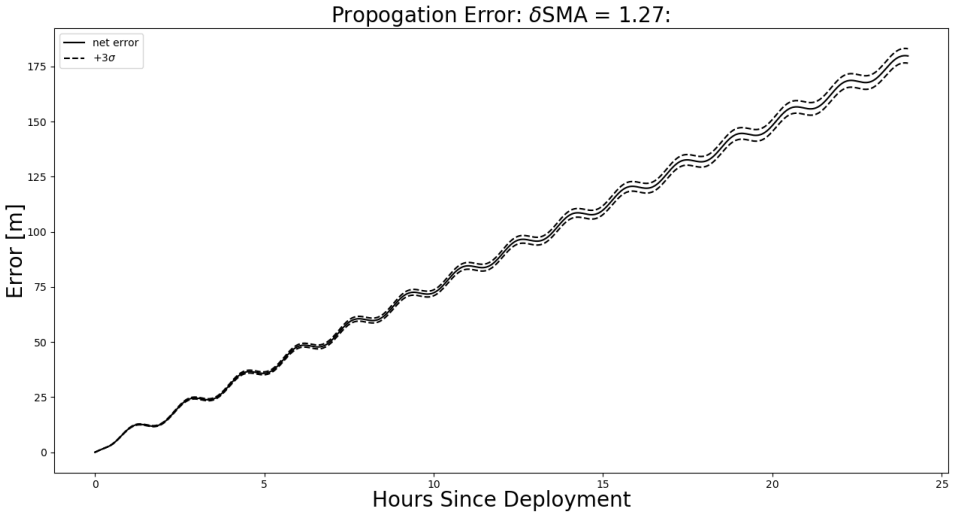

Figure 53: Example Propagation Error

After quantifying the propagation error for each deployment scenario, an error distribution was computed for the propagation error resulting in Fig. 54. The mean observed propagation error is 302 meters per day with a standard deviation of 220 meters per day.

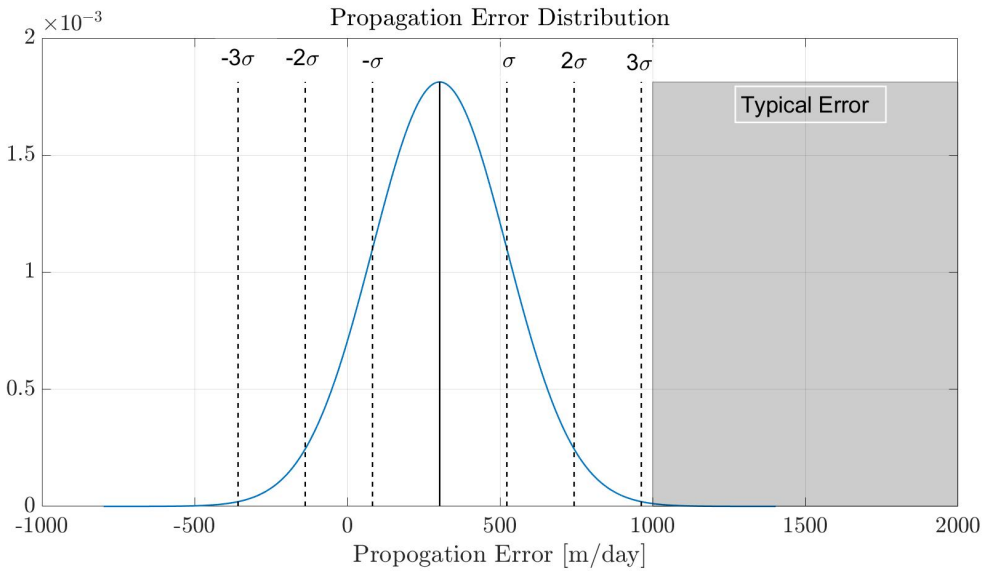

Figure 54: Propagation Error Distribution

When these results are compared to the typical propagation error for small satellites reported on Celestrak, which is between 1000 and 2000 meters per day, the results by VISION show almost a 47% improvement. TLE are only as accurate as how recently they were updated and with how much data they were updated with. Small satellites have a very low priority for this so they have a large propagation error. This shows the significance of implementing the VISION system which provides immediate and accurate data to produce these TLE for Small satellites allowing them to be significantly more trackable in the future. Another interesting note is that the secular drift in the propagation error observed in Fig. 53 is introduced by error in the semi-major axis which causes the orbital period of the estimated satellite state to be different from the truth. This change in period drives the drift in the error. So reducing error in semi-major axis is the best way to improve these estimates. Furthermore, the semi-major axis error is extremely sensitive to error in the estimated relative velocity. Because CubeSat velocity is not directly measured by VISION, significant improvements can be made to the velocity estimates by adding direct measurements of the relative velocity. By reducing error in relative velocity, error in semi-major axis decreases along with the propagation error.

6. Risk Assessment and Mitigation

Adrian Perez

6.1. Risk Management and Tracking Process

6.1.1. Identification

As a group of students with minimal engineering experience, the VISION team understood the importance of integrating a risk management strategy as a key tool in order to minimize the likelihood of key failures or road blocks in the development of the project. This started by pairing potential risks with every design choice that was proposed. As the preliminary designs of each subsystem were chosen, the risks associated with each component or software function were already identified. While the risks were not used as metrics for design trade studies, it was important to identify all risks that had some possibility of occurring throughout all phases of development.

Identification of risk usually stems from experience with a given technology or process used. Each subsystem lead, and their counterparts, used any experience they had in proposing not only design choices, but also areas of failure or potential problems. But experience was not the only factor in identifying potential risks associated with a given design choice, as team members also used advice from PAB members and reviews of other sources across the internet as a resource to identify risks that the VISION team could not foresee.

6.1.2. Evaluation

Once the preliminary design was chosen, the identified risks were then evaluated in terms of severity, or the effect they would have on the overall cost and development of the project, and likelihood of occurrence, or the probability they would in fact transpire if no action was taken. An example of this evaluation can be seen below in the following table:

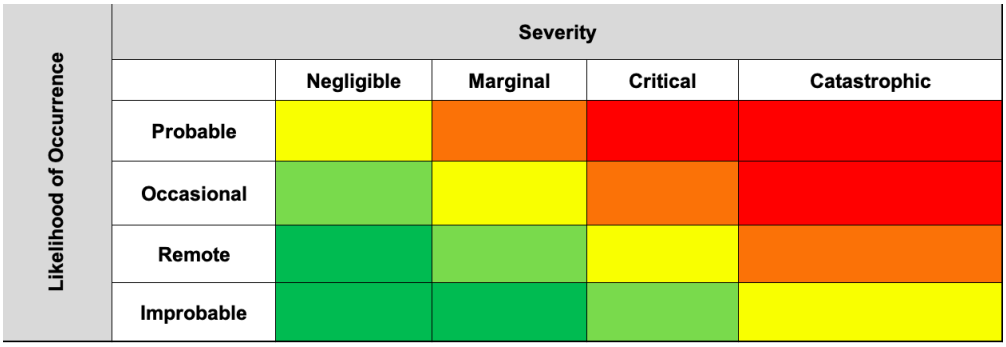

Figure 55: Risk Evaluation Table

Risks are usually never quantitative events, or explained in quantitative terms. This table/chart allowed for the VISION team to review each associated risk with the overall design in quantitative terms, using qualitative metrics. The table was tailored using industry standard risk templates, and uses a color metric of green, yellow, orange, and red to represent the seriousness of the risk. Any risks that laid in the red or orange regions required immediate action, while the green and yellow areas designated non-seriousness risks. But for every risk identified, the management team felt it was necessary to create a mitigation plan for each, in order to reduce both the severity and likelihood. The VISION team understood it was beneficial to the team to mitigate all risk as early as possible, as a given risk only grows in cost to the project as the design matures.

6.1.3. Tracking

The tracking of the risks, their evaluated score in terms of severity and likelihood, as well as the mitigation paths for each respective entry, was compiled in a risk mitigation matrix shown in the appendix in figure 69. Each risk carried a score, based on the evaluation assessment performed at the beginning of the fall semester, and was applied a new score based on the effect the mitigation method had on the overall severity and likelihood of each risk. If this new score was not within the appropriate region described earlier, a new mitigation path was then proposed and applied to the risk. This process was repeated until the team felt comfortable with the remaining effect the risk had on the overall project and cost. This matrix was reviewed on a weekly basis, as the design progress and matured both conceptually, as well as physically when manufacturing started in the spring semester.

6.1.4. Results

While full operation was never developed due to the interruption of COVID-19, the status of each risk described in table 69 will be reviewed in this section.
LANG CODE: When the software team decided to keep the heritage code blocks that were written in two languages, there was believed to be risk in compiling multiple languages with a single processing unit. This risk directly affected processing time and ease in developing an autonomous unit. A mitigation path of using a MATLAB API in Python, thus compiling all MATLAB code in python and one overall language, this risk would be mitigated. Unfortunately, while we were able to compile and run code on the NUC, the code was never ran from front to back as component level testing was never finished. This mitigation path was believed to efficiently reduce cost of risk, but was never proven by test.
DISASSM: Although accounting for the destruction or degradation of a component through misuse is not an decision that required engineering intuition, the VISION team felt it necessary to identify this as an error, as several components in the system greatly outweighed the majority of the rest of the system in cost. Thus the VISION team recognized in the event of misuse of the most critical component to hardware operation, being the NUC, a critical cost would occur. The team came up with two mitigation paths. These involved setting an allocated budget for a replacement unit, in the event of failure/degredation to the unit, but also buying the same model NUC as last year without the casing as

VISION will manually mount the processing unit to a custom fitting. These two paths helped to a reduction of risk, where the NUC was never damaged and did not need to be replace.

POW: One disadvantage of developing an autonomous system, powered from a single power source, is the risk of a short circuit, and failure of power distribution to all other electronics. While not very likely as the operating conditions of all tests were well within the performance specifications of all electronics, a mitigation path of printing multiple Printed Circuit Boards reduced the potential cost if the PCB was ever fried. The PCB never encountered a short circuit, and the effects of the risk were never observed, but the cost of the risk was mitigated successfully as the VISION team was prepared.

DEPL ORB: The Non-Linear Batch Filter was completely developed by the state estimation software team over these two semesters. In this development, an initial first guess of using HCW equations to describe the shape or eccentricity the CubeSats were expected to orbit in once deployed. But after exposure to other uncertainty filters describing orbital mechanics, the team felt there was risk in inaccurately describing the behavior of the CubeSats, all stemming from the use of HCW equations which assumed circular eccentricity. Thus, as a mitigation path, the Tschauner-Hempel dynamics equations replaced the HCW equations, and added the dynamics of non-circular eccentricity. While there was some non-gaussian error produced by the uncertainty filter, the cost associated with the risk of using HCW was reduced by adding complexity to the behaivor of the CubeSat orbits, and therefore deemed successful.

VIBE: This next risk was evaluated in terms of added cost of not meeting customer requirements. One requirement that was imposed on the VISION team by the customer was advancing the Test Readiness Level of any major subsystem. While this cost did not impose a fiscal consequence, it did infer the deduction of points of a grade in the Senior Projects Class. Therefore, as a mitigation path forward, the VISION team developed a Random Vibe testing procedure for the aluminum chassis, which would have been performed had this semesters operations not been interrupted. This mitigation path was believed to have reduced the cost, showing the effort to complete this requirement, but was never proved unfortunately.

CKF Tune: One major risk associated with the use of an uncertainty filter, is the needed tuning using multiple data sets, in order to prevent insufficient performance of the filter. So the VISION team delegated a mitigation path of simulating and recording sufficient data that would be used for tuning. While this risk was high in cost, the team felt capable of producing enough data sets to mitigate this risk. As all testing was halted in mid-March, VISION was not able to perform 3 sets that would have produced data sets that would be directly used to tune the filter. Therefore VISION cannot directly guarantee that all added risk associated with inaccuracy that stem from the filter are mitigated, but the VISION still believes the best path forward would have been to record as many data sets that mimicked the CubeSat orbital behavior as possible.

7. Project Planning

Ian Thomas & Andrew Pfefer

7.1. Organizational Chart

The duration and scope of the project required multiple levels of management. This was broken down as an organizational chart with can be seen in figure 56. This organizational breakdown was paramount in completing the tasks required for a successful project.

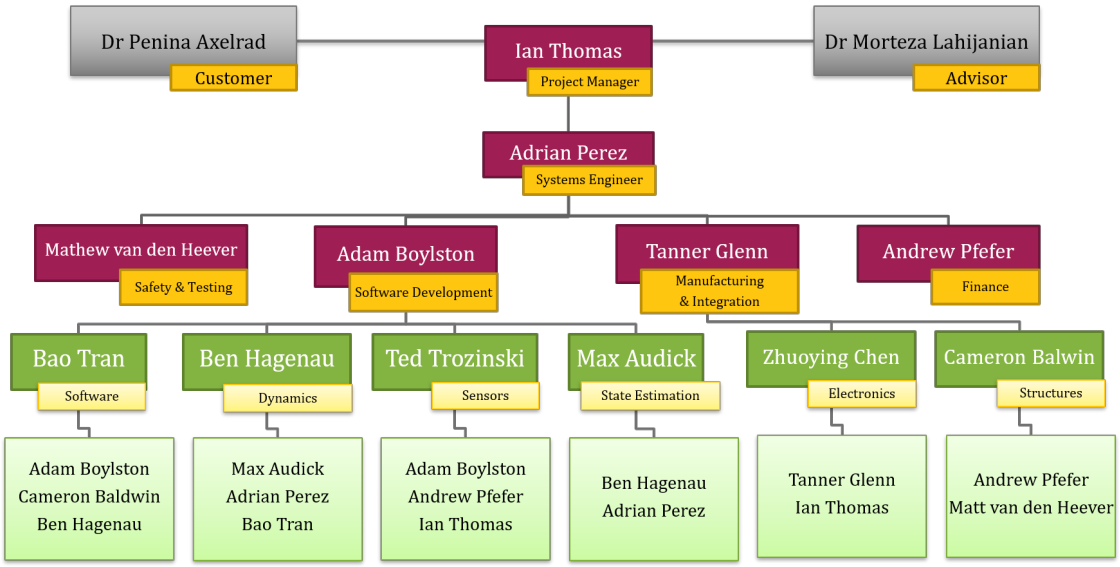

Figure 56: VISION Organizational Chart

7.2. Work Break Down Structure

The project was broken into more manageable phases with different goals and managerial practices in each phase. This is shown in the high-level WBS in figure 57 below.

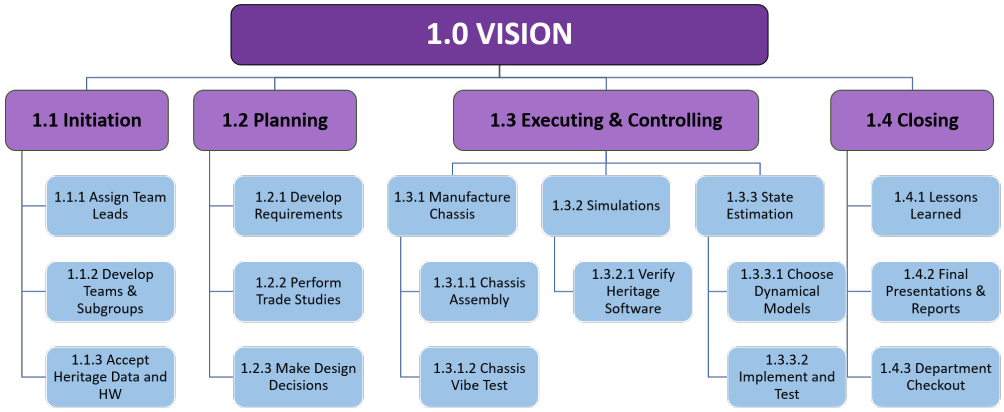

Figure 57: VISION Work Breakdown Structure

The project initiation and planning phases were primarily done in the first part of last semester, most of the subsequent tasks were completed as a team and are discussed in previous sections. The majority of the work is done in the the *Executing & Controlling* phase. This is broken down more specifically in figures 64, 65 and 66 found in the appendix.

7.3. Work Plan

The WBS gives a top-level guide to the respective subgroup, These ultimately drive the development of the work plan. A very high level-view of the spring semester can be seen in figure 58. Following the WBS, the work plan for the

integration (including manufacturing), software and testing are broken down into different tasks. The integration and software schedules can be seen in figures 63 and 60 respectively. Due to the complexity of the project there is more than one critical project element, therefore there is more than one critical path. These are highlighted with the red line following the tasks. The yellow line indicates the progress made until the COVID-19 pandemic stopped the project, this is indicated by the vertical red line.

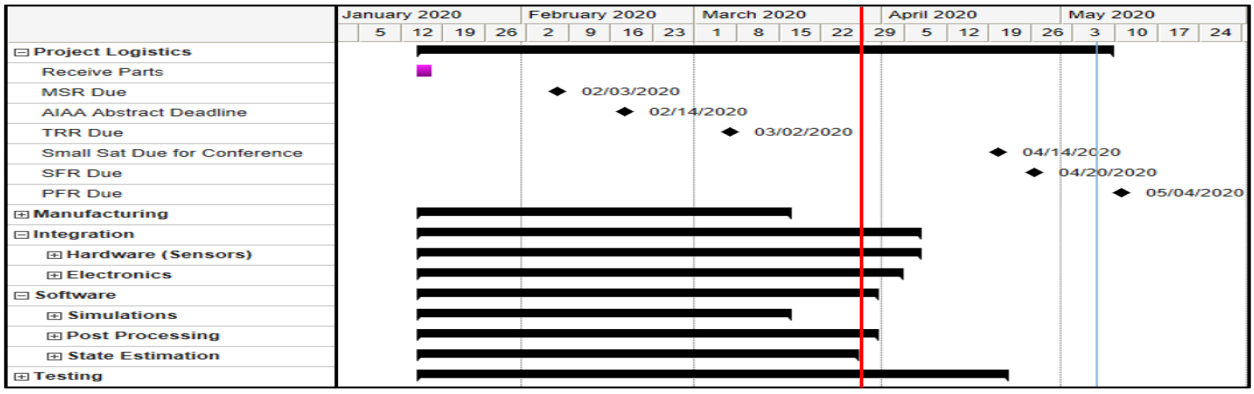

Figure 58: VISION Work Plan: Overview

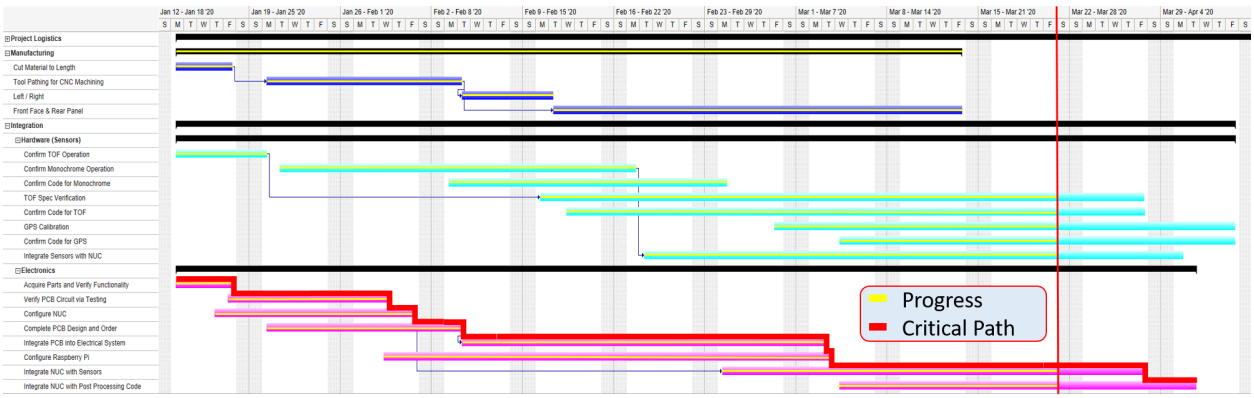

Figure 59: VISION Work Plan: Integration

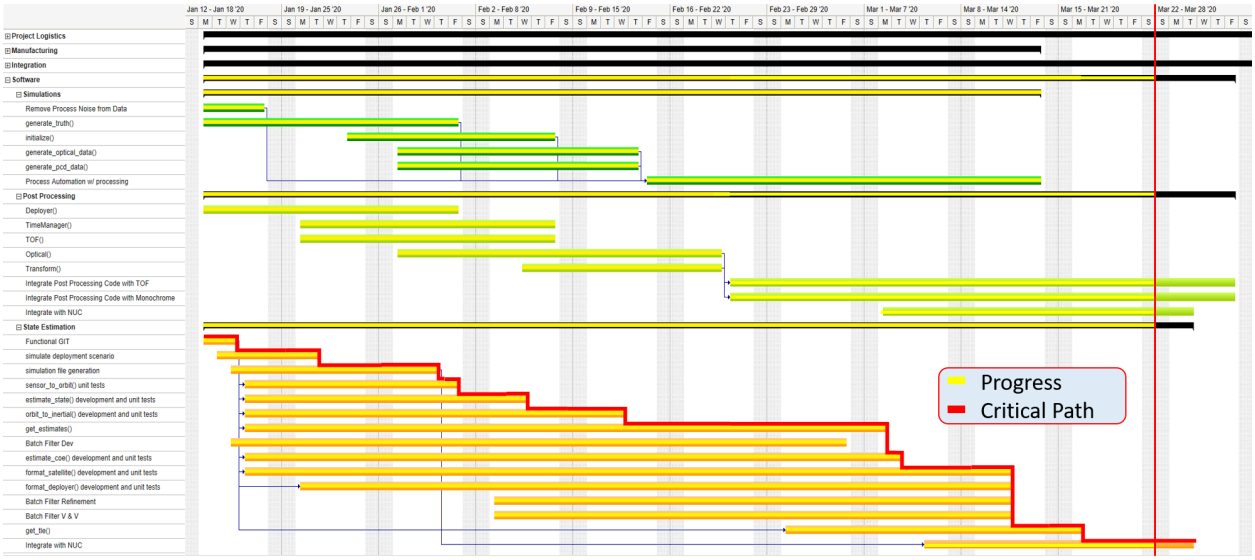

Figure 60: VISION Work Plan: Software

7.4. Cost Plan

In order to develop a cost plan for the VISION project, the team created a budget for the critical design review (found in appendix figure 67) encompassing all anticipated hardware and stock to create the system. This budget took into account uncertainties in price as well as shipping expenses. The team also decided to include redundant hardware in case the parts were damaged during the project. This included a second processor, backup aluminum stock, and extra funds in case the heritage hardware such as the time of fight camera were damaged. This budget was approved from the critical design review.

As testing plans were finalized and the electronics system chose all of the final hardware, procurement started. Figure 61 shows the components above $25 purchased for the project. This list only includes a little less than half of the number of components that were purchased. An exhaustive budget list can be found in appendix figure 68.

Vision Budget Document							
Subteam	Item	Qty.	Price	Shipping	Total Price	Buy or Inherit	Purchase Location
Electronics	Intel NUC8i7BEH	1	$641.84	$0.00	$641.84	Buy	https://www.amazon.com/NUC7I7DNBE-Deskto
	16GB RAM DDR4 1.2V	1	$57.99	$6.40	$64.39	Buy	https://www.bhphotovideo.com/c/product/13338
	250GB SSD	1	$65.99	$12.00	$77.99	Buy	https://www.bhphotovideo.com/c/product/13825
	120V-24V DC/DC Converter	1	$70.00	$5.00	$75.00	Inherit	N/A
	TDK-Lambda	1	$33.55	$7.99	$41.54	Buy	https://www.mouser.com/ProductDetail/TDK-La
	TDK-Lambda converters	1	$33.00	$5.00	$38.00	Buy	https://www.mouser.com/ProductDetail/967-I6A
	PCB	3	$33.00	$10.00	$109.00	Buy	www.advancedcircuits.com
	Raspberry Pi 3 B+	1	$35.00	$3.00	$38.00	Inherit	N/A
	Raspberry Pi	1	$43.75	$43.75	$87.50	Buy	https://www.mouser.com/ProductDetail/Adafruit
	Ethernet to usb3.0 adapter	1	$14.99	$14.99	$29.98	Buy	https://www.amazon.com/dp/B07MK6DJ6M/ref=
Structures	6061 Al 8"x8"x.25" (Front/Back)	2	$18.31	$70.92	$107.54	Buy	https://www.mcmaster.com/9246K11
	6061 Al 6"x48"x.25" (Bottom/Top)	1	$55.96	$0.00	$55.96	Buy	https://www.mcmaster.com/9246K424
	6061 Al. 6"x"48"1.25" (Left/Right)	1	$224.92	$0.00	$224.92	Buy	https://www.mcmaster.com/9246K625
	M2 Screws (20mm), x25	3	$9.40	$0.00	$28.20	Buy	https://www.mcmaster.com/91294A539
Sensors	Time of Flight Camera	1	$1,547.00	$0.00	$1,547.00	Inherit	N/A
	Camera Sensor	1	$725.00	$0.00	$725.00	Inherit	N/A
	Camera Lens	1	$500.00	$0.00	$500.00	Inherit	N/A
	GPS Antenna	1	$50.81	$21.56	$72.37	Buy	https://www.taoglas.com/product/agpsf-36c-07-
	GPS Receiver	1	$199.95	$0.00	$199.95	Buy	https://www.sparkfun.com/products/15005?_qa=
Test	Internal 1 inch PVC coupling	20	$1.77	$10.60	$46.00	Buy	https://www.pvcfittingsonline.com/1-pvc-internal
	1" x 10' PVC	10	$2.70	$0.00	$27.00	Buy	Home Depot
	track wheels	4	$39.00	$0.00	$156.00	Buy	https://www.bhphotovideo.com/c/product/11890
	uxcell DC 24V 111RPM	1	$34.99	$0.00	$34.99	Buy	https://www.amazon.com/uxcell-50Kg-cm-Self-L
Total					**$2,307.26**		
Surplus					**$2,692.74**		

Figure 61: VISION Budget items $25+

Figure 62 shows a breakdown of the cost by subsystem. The most noticeable portion is the surplus which is the majority of the $5000 budget. The Electronics subsystem had the highest expense which was mainly the processor. The team kept a large surplus to buy down risk if the processor was damaged and needed to be replaced. Next, structures was the second highest expense and was dominated by the cost of Aluminum stock. The last note is that the testing budget was higher than expected which was a comment at CDR.

Subteam Subtotals	
Electronics	$1,187.11
Structures	$439.15
Sensors	$280.27
Testing	$400.73
Surplus	$2,692.74

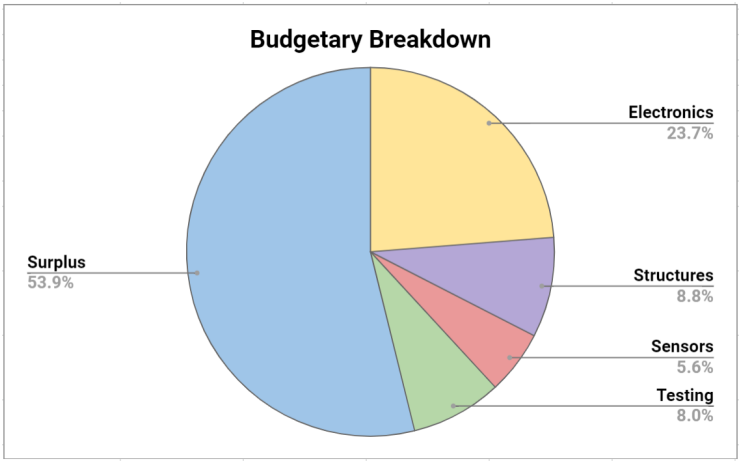

Figure 62: Subsystem Cost Breakdown

7.5. Testing Plan

As with the previous sections, the testing plan was included in the Gantt chart for the project. This plan is laid out in the figure below.

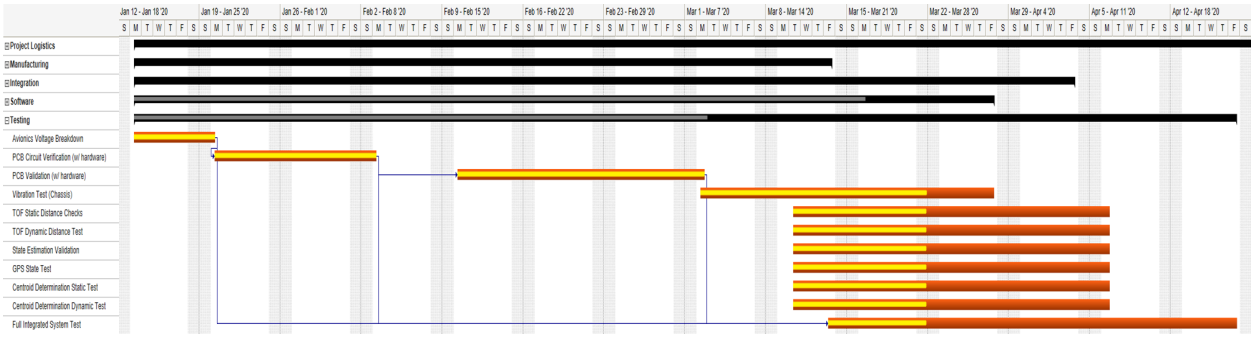

Figure 63: VISION Work Plan: Integration

The testing was planned to be started or completed by the time the COVID-19 crisis halted the project. The electrical testing was completed and the other tests were in the process of being developed. This includes plans and procedures, as well as the manufacturing of certain testing devices. The testing had not gotten to formal data collection for a variety of reasons but the schedule allowed for a few weeks for each test to be completed. The sensor and centroiding tests were all planned to be completed during the same testing periods, meaning the set-up and restrictions were the same for all. This gives us confidence that there would have been plenty of time to complete all of the remaining component testing before our integration test. The integration test itself required designing and manufacturing, all of which had been completed prior to the stoppage. The testing ultimately was held up by software compatibility issues, though those issues had a built in margin so as to not completely overtake the testing schedule.

8. Lessons Learned

Andrew Pfefer & Bao Tran

Throughout the 2 semester class, the entire team has learned many lessons about the engineering process and team dynamics. In this section we will try to capture the major lessons we learned from the process and mistakes that were made. The team implemented a lessons learned documentation inspired by the NASA lessons learned database and the write up can be found in appendix section 11.5.

The most important lesson team VISION learned was how to communicate our project. Early on in the project it seemed trivial to communicate our plan for the project. However, after the first presentation, we learned that we needed to very carefully plan our story for each presentation and clearly introduce, explain, and conclude each idea with logical transitions between topics. This technique change was by far the most valuable and impractical change we made during the project. Not only did this improve our presentation performance, and therefore grades, but it allowed the PAB to give us better questions and feedback that vastly improved our project. To future teams that may read this section, improving this is the best way to improve your project. To achieve this we employed a few strategies we will share here. When developing the presentations and papers, we sat down as an entire team and talked through the story we would use to convey our project. This includes a very clear introduction which is the foundation of the remainder of the presentation. Next, the team created a draft presentation or paper and took it to our advisor, customer, and multiple PAB members of diverse backgrounds to make sure that the story made sense and did not jump around. This was also useful to verify that the technical level of the contents were not too high level or low level. Finally we would incorporate the feedback into our presentation. We teamed up with other senior project teams to ”red team” which allowed us to get feedback from others in the same situation and a different perspective. After this process, we felt extremely comfortable in the actual presentation.

Another important lesson the team learned was the circumstances that follows with being a heritage project. There was a disconnect in the continuation of the project from the previous team to the current team due to the change of personnel, experience, and lack of communication between the two teams. VISION encountered many problems with understanding and running VANTAGE’s code, which seems to take the whole Fall semester. The struggles range from installing the simulation software to running it on our own computers. Everytime an error was encountered, a member of the team had to reach out to the previous team to debug this problem. This repeated a few times and every iteration accumulated to much time wasted. These problems originated from the lack of communication between the teams, which mostly includes documentation. The team understand this problem is more common in academia, since industry does not replace the whole team at a time and has more requirements in documentation. In order to mitigate this problem, the team emphasizes the importance of documentation and communication with the prospect team.

The team also had a mishap that delayed parts of the project and required purchasing of new hardware. We will relay the situation and lessons learned here but have a thorough account in the aforementioned appendix section. On February 10th, a test of the printed circuit board in the Electrical Power System was taken place to verify the design and components of the board. A detailed testing plan was written in order to make sure that the board was attached properly and the outputs would be tested before attaching expensive hardware. In this event, the test was conducted incorrectly by connecting a Raspberry Pi before checking the outputs of the PCB. After turning on the power, the Raspberry Pi started smoking and the test was stopped. It turned out after inspection that the two DC/DC converters on the board were installed in the wrong positions and the full 25 VDC source was provided to the Raspberry Pi greatly exceeding it's operating voltage range and destroying the hardware. Fortunately, the team possessed backup boards and ordered a new Raspberry Pi. The lesson that was learned from this event was to follow the testing plan to the letter. At the end of the day, it was the excitement of verifying the board that caused this oversight and destruction of hardware. After this event, all further tests and especially electronic tests followed the procedure very closely.

In light of the COVID-19 crisis that terminated our projects about 75% through completion, the team has some advice for future teams. Our advice is to not take the project for granted. As difficult and time consuming as it can be, it is a very rewarding experience that all 12 of us regret not being able to complete. It is a wonderful opportunity to display what you have learned the prior 3 years as well as learn much more than that as you understand what a real engineering project is like to complete.

9. Individual Report Contributions

Name	PFR Contribution Sections	Project Contibution
Max Audick	3.3.3, 4.3, 5.5.1	Sensor Simulation Adaptation, State Estimation, Kalman Filter, Software Integration
Cameron Baldwin	3.1, 4.1, 5.2, ICD sections: 1.1, 1.2, 2.1, 2.2, 2.3	Structures Design, Manufacturing, Vibrational Analysis
Adam Boylston	3.3.3, 4.3.2, 4.3.3	Programming Language Trade Study, Centroid Determination Development, Software Integration
Zhouying Chen	3.2, 4.2	
Tanner Glenn	4.1.1, 4.1.2, ICD sections: 2.4, 3.1, 3.3, 3.4	Preliminary electronics design and testing, manufacturing, system testing helper. Lead water boy.
Ben Hagenau	3.3.1, 3.3.2, 3.3.4, 4.3, 5.4.4, 5.5.1, 5.5.2, 5.5.3, 5.5.4, 5.5.5	state estimation algorithm design, Orbit simulation code, nonlinear batch filter code, coordinate frame transformation code, state estimation analysis code, state estimation tuning code, accuracy requirement derivation, class structure, state estimation test design, supporting state estimation methods.
Adrian Perez	2.1, 6	System engineering
Andrew Pfefer	1.1, 2.1, 2.2, 2.3, 7.4, 8	Budgets, procurement, Design of System Integration Test track, Implementing GPS receiver and antenna, Assistant *to the* Project Manager
Ian Thomas	1.2, 7.1, 7.2, 7.3, 7.5	Project Management,Electrical Team, Testing Team
Bao Tran	3.3.1, 3.3.2, 3.3.3, 3.3.4, 4.3.1, 4.3.3	Simulation, Centroid Determination, Sensor Automation, Software Integration
Theodore Trozinski	3.2.3, 4.3.3, 4.4, 5.3.1, 5.3.2, 5.5.6	Data Collection, Sensor Research, Integration and Automation, Verification and Validation Centroid Testing
Mathew van den Heever	5.1, 5.2, 5.3, 5.4, 5.5	Test Procedures, Avionics system manufacturing, test equipment manufacturing, PAB meeting coordinator, Avionics Testing

Table 11: Functional Requirements

10. Acknowledgements

Team VISION would like to thank Dr. Penina Axelrad, Dr. Morteza Lahijanian, and the entire PAB for their invaluable help throughout the year. This project would not have been possible without them.

11. Appendix

11.1. Structures

University of Colorado
Department of Aerospace Engineering Sciences
Senior Projects - ASEN 4018
Interface Control Document (ICD)

Visual In-Situ Sensing for Inertial Orbits of NanoSats (VISION)

Monday 16th December, 2019

Project Customer

Name: Dr Penina Axelrad Email: penina.axelrad@colorado.edu

Group Members

Name: Max Audick Email: maximillian.audick@colorado.edu	Name: Cameron Baldwin Email: caba3252@colorado.edu
Name: Adam Boylston Email: adbo5502@colorado.edu	Name: Zhuoying Chen Email: zhch1699@colorado.edu
Name: Tanner Glenn Email: tagl1811@colorado.edu	Name: Ben Hagenau Email: beha7507@colorado.edu
Name: Adrian Perez Email: adpe2067@colorado.edu	Name: Andrew Pfefer Email: anpf9194@colorado.edu
Name: Ian Thomas Email: iath2777@colorado.edu	Name: Bao Tran Email: tran.tran@colorado.edu
Name: Theodore Trozinski Email: thtr7807@colorado.edu	Name: Mathew van den Heever Email: mava5537@colorado.edu

Contents

Nomenclature

$\mathcal{S}$ = VISION Sensor Frame
$\mathcal{O}$ = Deployer Orbit Frame
$\mathcal{D}$ = Deployer Body Frame
$\mathcal{N}$ = Earth-Centered Inertial Frame

1. Introduction

1.1. Purpose

This Interface Control Document (ICD) defines the requirements and methods of interfacing the VISION system for operations with a CubeSat deployment structure.

1.2. Scope

This ICD is the sole description of requirements for end users of VISION. The physical, functional, and environmental requirements for operation of VISION are described within this document. The defined requirements will apply to all phases of operation, from integration and assembly to flight operations.

2. VISION Overview

This section outlines the Visual In-situ Sensing for Inertial Orbits of NanoSats (VISION) system and describes various parameters and design features necessary to meet the requirements of Section 3.

2.1. Overview

VISION (Fig.1) is a single package system designed to track CubeSats during deployment, determine their inertial orbits, propogate them forward, and then relay this data to the deployer. This allows for a smooth transition from deployment to tracking operations. VISION is a modular package with multiple attachement points for easy integration to most systems.

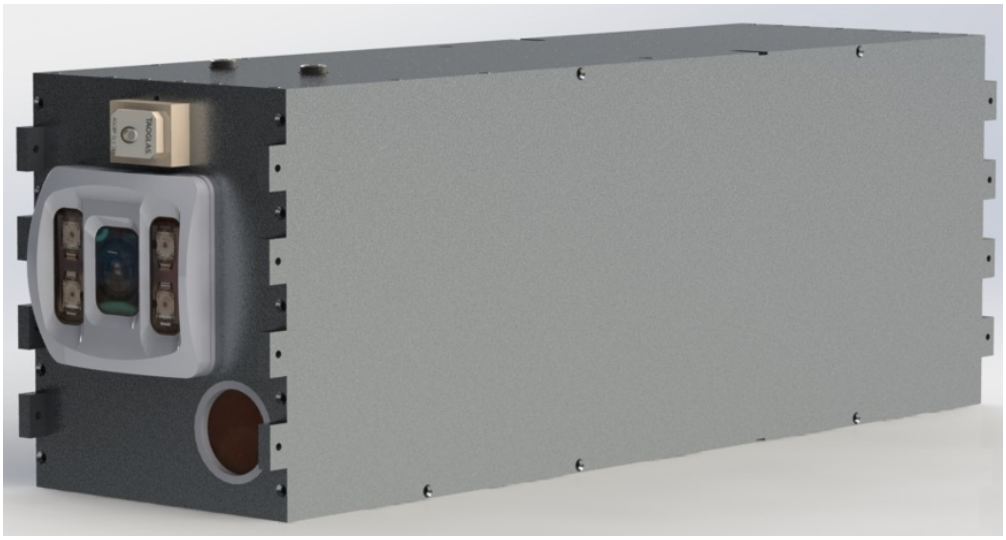

Figure 1: VISION System

2.2. Coordinate System

VISION's coordinate system is defined below, Figure 2. The location of the origin is not considered at this time, but is to give reference for geometry of the package.

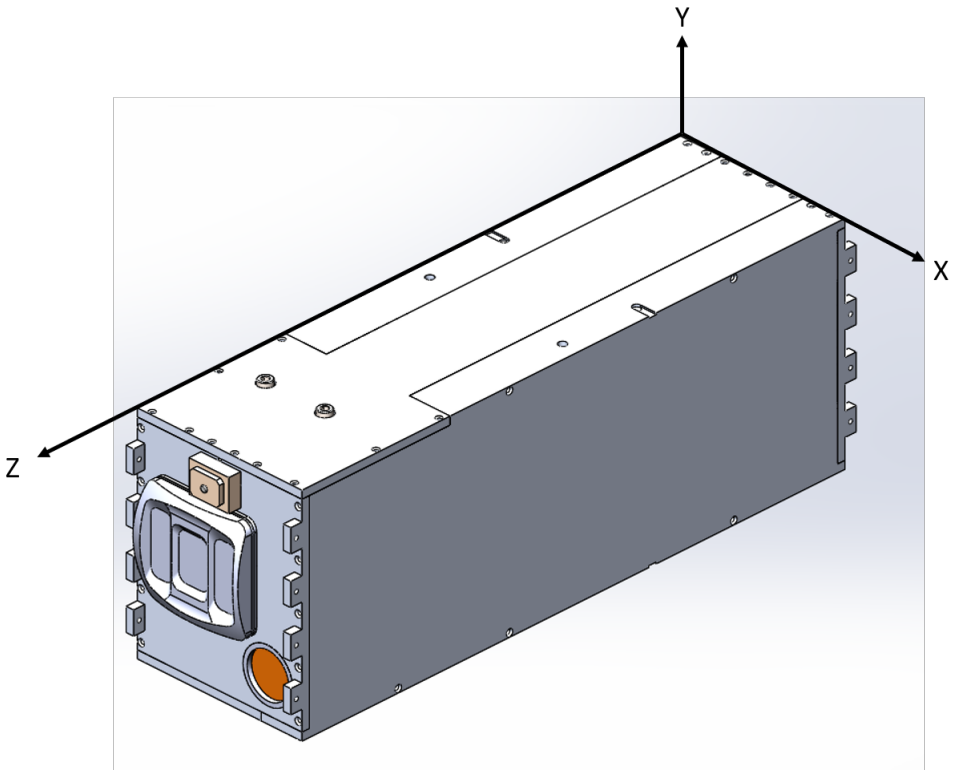

Figure 2: VISION Coordinate System

2.3. Design Features

VISION's sensor suite includes 3 snesors. The Time of Flight camera, the monochrome camera, and the GPS antenna. The Time of Flight camera is capturing the deployment of the CubeSats and generating relative estimates of their centroids and positions. The monochrome camera is used to correct cross-plane measurements from the ToF and improve accuracy. The GPS antenna is allowing for inertial estimates of VISION's own attitude.

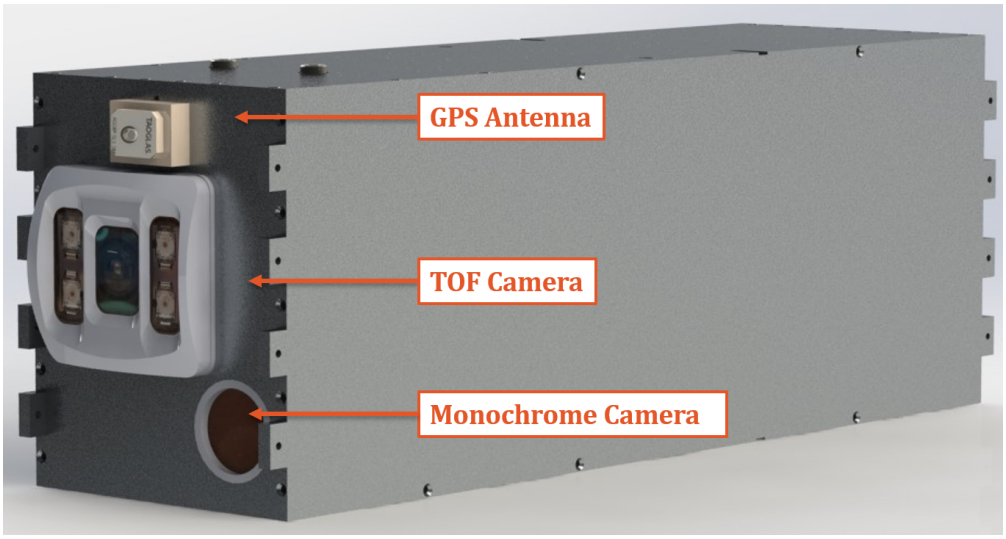

Figure 3: VISION Sensors

2.4. Operations

The VISION package was designed to function completely autonomously. A full cycle of operations includes boot and calibrate, data collection, data processing, data packaging, and finally data transfer. Boot and calibrate will accept power from the deployer and condition the signal for each of the sensors. Each sensor will be turned on in a standby mode until all systems are operating nominally. At which point, the data collection will begin and all sensors will collect 1 cycle's worth of data. Immediately following, the data will be processed by the NUC, packaged into a TLE, and transferred to the deployer. The VISION package has no telecommunications on board and is fully reliant on the cubesat deployer to downlink the data package to a ground station.

3. Interfacing Requirements

3.1. Structural and Mechanical Subsystem Interfacing

VISION is designed to interface modularly on multiple deployment systems. There are a total of 18 points that can be used to integrate with (circled in red). These integration points are for M-3 screws. The design also allows for the package to be integrated onto either side of a deployer. At any one time, only 9 of the 18 integration points will be used. If the operator chooses to forgo using the top integration points, the package can then be integrated in multiple orientations.

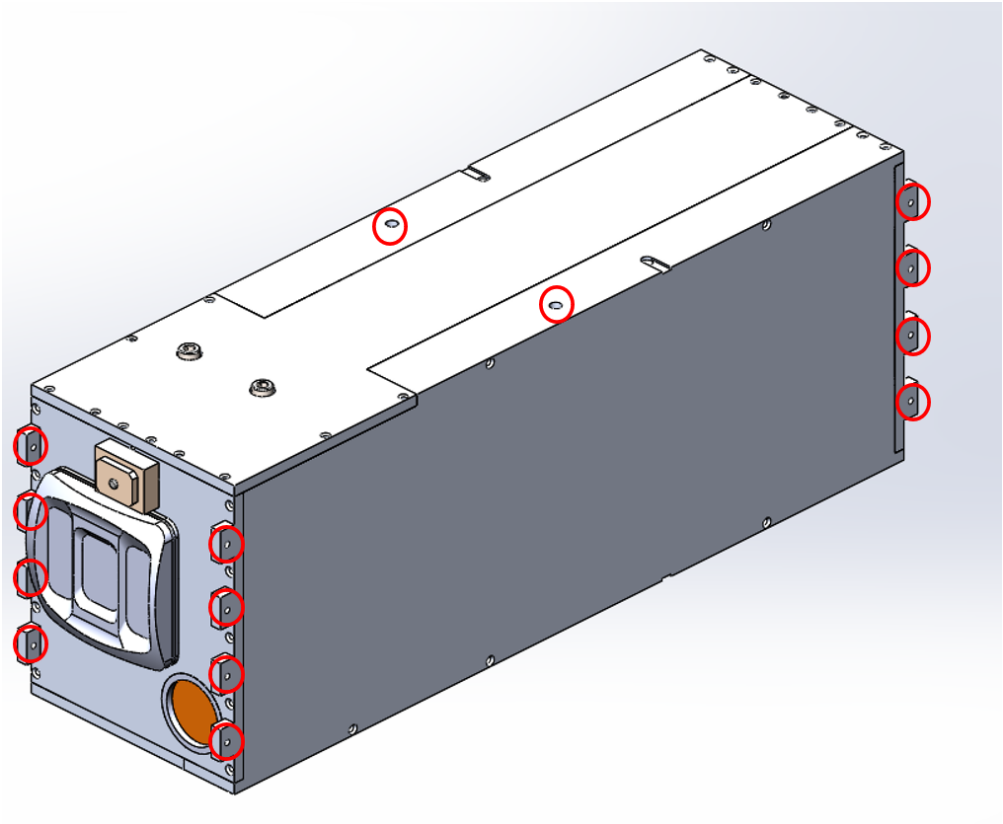

Figure 4: Integration Points

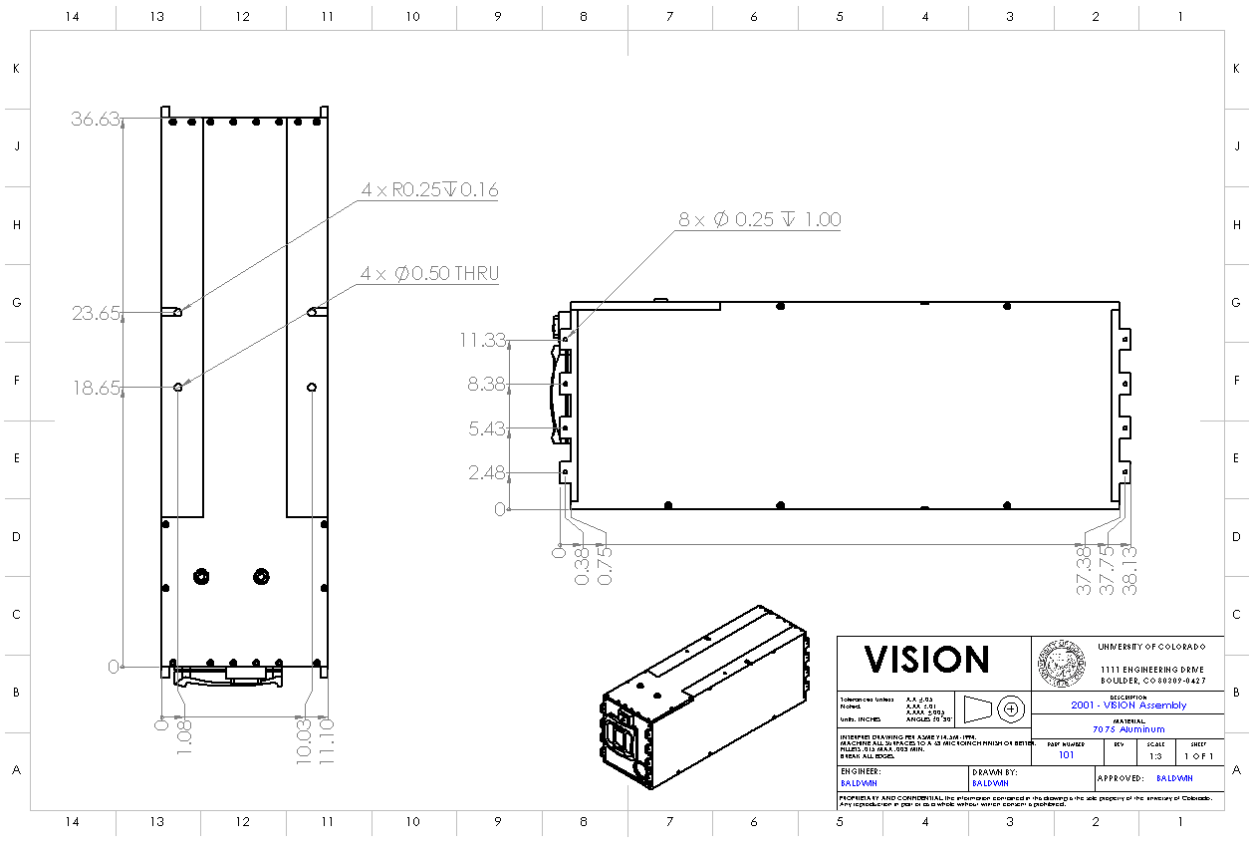

Figure 5: Integration Dimensions

Figure 5 shows a schematic of the integration points for the VISION package. The M3 screws used for inetegration are equidistantly spaced with approximately 2.95 cm between each screw hole on the front and back of the package. The integration points on the top of the package are located 19.48 cm from the back of the package where the coordinate frame is set. In order to interface with VISION, the deployer should use 4 of the 18 integration points.

3.2. Avionics Subsystem Interfacing

Electrical services are available via banana interface connectors. The connectors are located at the backside of the VISION chassis. The Interface to the deployer shall be a 120VDC power inlet with a minimum power of 120W @ 180V. Additionally, a 5VDC USB 2.0 data bus shall be connected with the deployer.

3.2.1. Electrical Provisions – 120V

The VISION shall be connected with 120VDC with a minimum power of 180W. The contacts shall be banana plugs for the power in and the floating ground. Our design is based on the NanoRacks deployer. Although the NanoRacks cannot provide large range of power, the VISION system is able to handle the power with large range.

Voltage range	80 – 140V
DC current	1.3A
Inrush Current	5A
Rated Power	120W
Peak Power	180W

Table 1: The 120VDC Power characteristics

3.2.2. Command and Data Interfaces

The VISION shall be connected with a 5VDC USB 2.0 data bus. The telemetry system shall be able to work by USB 2.0 transfer.

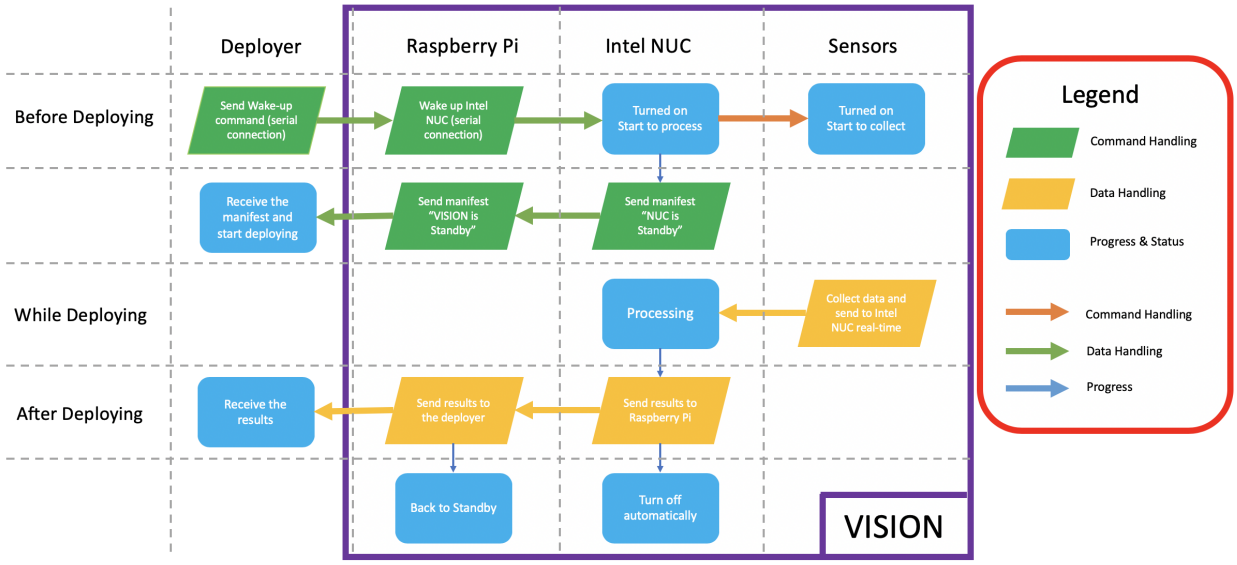

Figure 6: Telemetry System

The telemetry system of VISION is working under the flowchart shown above. Before the deployer starts to launch CubeSats, the deployer needs to send a wake-up command to the raspberry pi. As the raspberry pi receive the command, it wakes up the Intel NUC. At the same time, the sensors suite (Monochrome camera, ToF camera, and GPS) wake up. Besides, after the NUC is awake, it sends a message "NUC is standby" back to the raspberry pi. Then raspberry pi receives this message and sends a message to the deployer about the manifest that VISION is ready to work. Then the deployer can launch CubeSats. The sensors collect data and transfer them to the NUC real-time, and the NUC processes those data to get estimates. After deploying all the CubeSats, NUC finishes all the data processing tasks firstly, and sends them to the raspberry pi. Then it turns off automatically and waits for the next deployment cycle. After the raspberry pi receives all the data, it transfers them to the deployer and back to standby to wait for the next iteration.

In order to interface with the VISION system, the deployer shall have a USB 2.0 bus to handle command and data. Besides, it shall have two parts of the code. Firstly, having a "wake-up" command to let the raspberry pi or the VISION system know the deployer is going to launch the CubeSats. Secondly, after receiving the message that "VISION is standby", it shall be able to launch the CubeSats automatically. These two parts of code are needed for the deployer to work with VISION as a fully autonomous system.

3.3. Environmental Interfacing

During the testing portion of the project life cycle, the VISION package was set to undergo vibrational testing. Specifically, a frequency sweep, resonant frequency, and sample ASD launch vibration profile were going to be completed. This section of the ICD would include the results of these tests as well as highlight structural limitations for the consumer. Since the project was ended prior to testing, none of the vibrational results were obtained and cannot be included in the ICD.

3.4. Mission Operation Interfacing

The VISION package is designed to track 1-6 separate CubeSats simultaneously. These Cubesats can also vary in size between 1-6U. For optimal functionality, the VISION package should only be used to track satellites within the number and size constraints. Additionally, the VISION package should be integrated directly to the CubeSat deployer with minimal obstructions in the package's field of view. Furthermore, gaps of 15 minutes between deployments is optimal and will allow the package to finish data processing before starting another cycle of data collection.

11.2. Software

Metric	Driving Req's	Weight	Rationale
Execution Time	FR.2, FR.3	25%	Execution time is a critical part of the mission. Due to the time requirement of sending our deliverables to the deployer before the subsequent deployment, the faster the code can run, the better.
Learning Time	FR.2	15%	Learning time only affects the first few weeks of code development so it is weighted less heavily than other factors. Similarity to languages already known by the team will decrease the amount of time spent learning the language, and allow more time for writing of the actual code.
Development Time	FR.2	30%	Development time will have the biggest factor in VISION's ability to succeed and fulfill the functional requirements. This focuses on the ease of working with the language, and will affect the speed at which the code is written for the duration of the project. Factors such as readability/visualization, debugging, and compatibility to different environments affects this parameter.
Libraries	FR.1, FR.2	15%	The amount of libraries available in each language is very important to the capabilities of the code, and Python and C++ simply have more libraries available than MATLAB. However, all three languages can accomplish this project with the available packages, so since this metric isn't mission critical is is only weighted 15%.
Embedded System Compatibility	FR.4, FR.6	15%	Only certain embedded boards are capable of running MATLAB (through Simulink), while most can run Python and C++. The code being able to run on our computing board is very important, however the electronics requirements will drive the choice of computer which leads to compatibility being a smaller software consideration.

Table 12: Weights of Programming Language Metrics

11.3. Project Planning

11.3.1. WBS

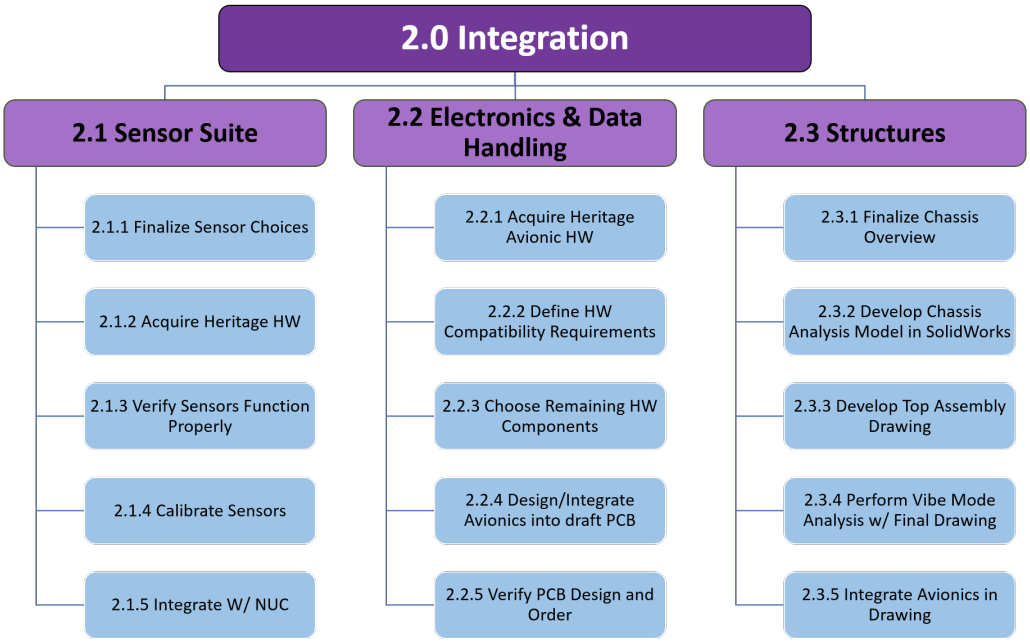

Figure 64: VISION WBS: Integration

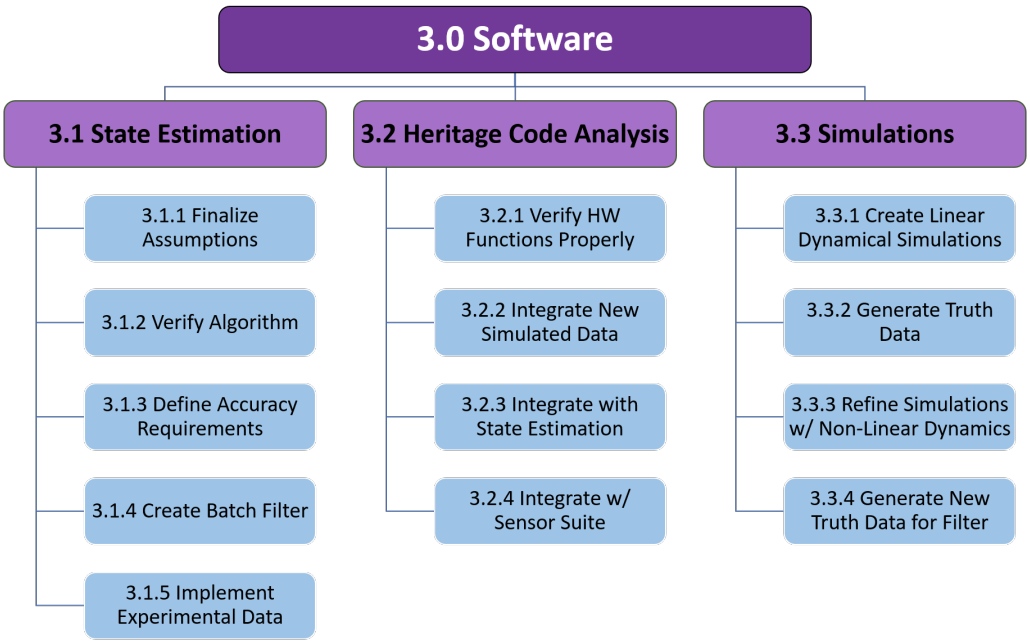

Figure 65: VISION WBS: Software

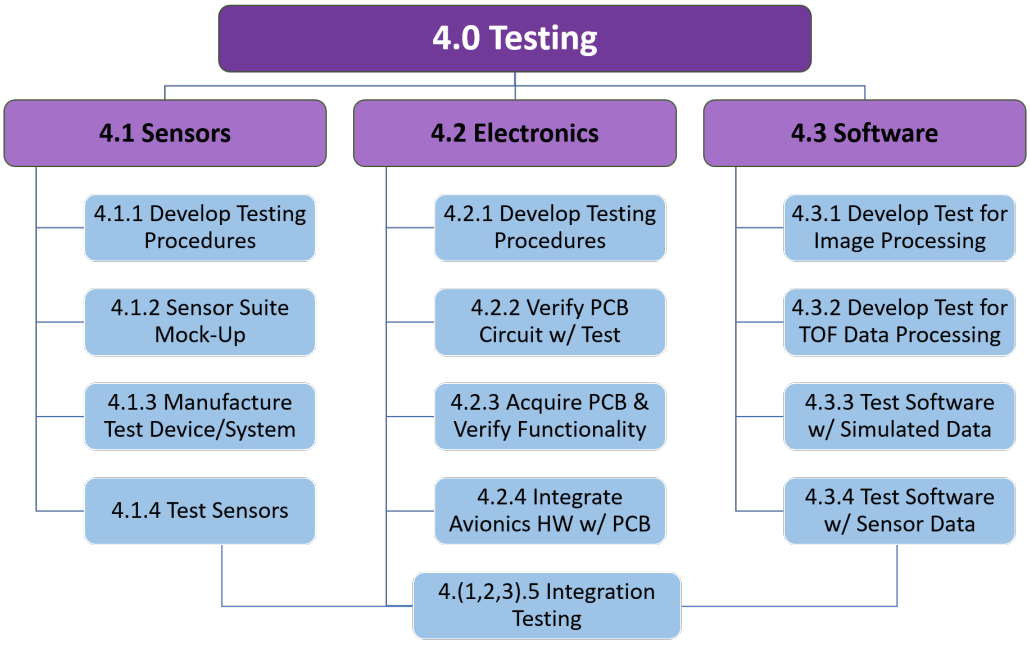

Figure 66: VISION WBS: Testing

11.3.2. Budgetary

Figure 67: VISION Budget as of CDR

Vision Budget Document										
Subteam	Item	Qty.	Price	Price Uncertainty (+/-)	Highest Price	Buy or Inherit	Binary	Purchase Location	Expected Purchase Date	Expected Lead Time
Electronics	Intel NUC7i7DNB 16GB + 500 GB SSD	2	$642.00	$40.00	$1,364.00	Buy	1	https://www.walmart.cc	12/19,3/20	4-6 days
	16 GB Ram Card	1	$60.00	$6.40	$66.40	Buy	1	https://www.bhphotovic	12/19	2-3 days
	500 GB SSD	1	$70.00	$12.00	$82.00	Buy	1	https://www.bhphotovic	12/19	2-3 days
	120V-24V DC/DC Converter	1	$70.00	$5.00	$75.00	Inherit	0	N/A	N/A	N/A
	24V-12V DC/DC Converter	1	$33.00	$5.00	$38.00	Buy	1	https://www.mouser.cc	12/19	2-3 days
	24V-5V DC/DC Converter	1	$33.00	$5.00	$38.00	Buy	1	https://www.mouser.cc	12/19	2-3 days
	12V-3.3V DC/DC Converter	1	$8.00	$5.00	$13.00	Inherit	1	N/A	N/A	2-3 days
	PCB	2	$33.00	$33.00	$132.00	Buy	1	www.advancedcircuits.	1/19	2-3 days
	Raspberry Pi 3 B+	1	$35.00	$3.00	$38.00	Inherit	0	N/A	N/A	N/A
	Ethernet Cable	2	$3.00	$0.00	$6.00	Inherit	0	N/A	N/A	N/A
	Ethernet to USB 3.0	1	$20.00	$2.00	$22.00	Buy	1	https://www.amazon.cc	12/19	2-3 days
	USB 3.0 to USB 3.0 Adapter	1	$7.00	$1.00	$8.00	Buy	1	https://www.amazon.cc	12/19	2-3 days
	USB 3.0 to micro-B Cable	1	$7.00	$1.00	$8.00	Inherit	0	N/A	N/A	N/A
	USB 3.0 to micro-USB	1	$7.00	$1.00	$8.00	Inherit	0	N/A	N/A	N/A
	Other PCB Components	1	$50.00	$5.00	$55.00	Buy	1	www.advancedcircuits.	1/19	2-3 days
	GPS Connectors	0	$10.00	$5.00	$0.00	Buy	1	https://www.taoglas.co	1/19	2-3 days
Structures	6061 Al 8"x8"x.25" (Front/Back)	2	$18.31	$1.80	$40.22	Buy	1	https://www.mcmaster.	1/20	2-3 day
	6061 Al 6"x48"x.25" (Bottom/Top)	1	$55.18	$5.50	$60.68	Buy	1	https://www.mcmaster.	1/20	2-3 day
	6061 Al. 6"x"48"1.25" (Left/Right)	1	$225.00	$22.50	$247.50	Buy	1	https://www.mcmaster.	1/20	2-3 day
	7075 Al 6"x6"x.25" (Front/Back)	2	$23.85	$5.96	$59.63	Buy	1	https://www.mcmaster.	2/20	2-3 day
	7075 Al 6"x24"x.25" (Bottom/Top)	2	$80.30	$20.08	$200.75	Buy	1	https://www.mcmaster.	2/20	2-3 day
	7075 Al 6"x24"x1.5" (Left/Right)	2	$322.88	$80.72	$807.20	Buy	1	https://www.mcmaster.	2/20	2-3 day
	Damping Gasket	1	$77.37	$19.34	$96.71	Buy	1	https://www.mcmaster.	2/20	2-3 day
	M2 Screws (20mm), x25	3	$9.40	$1.89	$33.87	Buy	1	https://www.mcmaster.	1/20	2-3 day
	M2 Screws (10mm), x25	3	$7.41	$1.89	$27.90	Buy	1	https://www.mcmaster.	1/20	2-3 day
Sensors	Time of Flight Camera	1	$1,547.00	$0.00	$1,547.00	Inherit	0	N/A	N/A	N/A
	Camera Sensor	1	$725.00	$0.00	$725.00	Inherit	0	N/A	N/A	N/A
	Camera Lens	1	$500.00	$0.00	$500.00	Inherit	0	N/A	N/A	N/A
	GPS Antenna	2	$50.00	$5.00	$110.00	Buy	1	https://www.taoglas.co	1/20	2-3 days
	GPS Receiver	2	$200.00	$5.00	$410.00	Buy	1	https://www.sparkfun.c	1/20	2-3 days
Test	Radial Encoder	1	$100.00	$50.00	$150.00	Buy	1	https://www.sparkfun.c	2/20	2-3 days
	1" PVC Tubing	6	$3.30	$1.00	$25.80	Buy	1	https://www.homedepc	2/20	0 days
	Brushless Motor	1	$80.00	$20.00	$100.00	Buy	1	https://www.amazon.cc	2/20	1 day
	Fishing Line	1	$20.00	$5.00	$25.00	Buy	1	https://www.amazon.cc	2/20	2 days
	2"x4"x10'	3	$3.00	$1.00	$12.00	Buy	1	https://www.homedepc	2/20	0 days
	3D Printing Material	4	$20.00	$5.00	$100.00	Buy	1	https://www.amazon.cc	2/20	3 days
Total			$3,729.46	$584.20	**$4,313.66**					
Surplus					**$686.34**					

Figure 68: Final VISION Budget

Vision Budget Document							
Subteam	Item	Qty.	Price	Shipping	Total Price	Buy or Inherit	Purchase Location
Electronics	Intel NUC8i7BEH	1	$641.84	$0.00	$641.84	Buy	https://www.amazon.com/NUC7I7DNBE-Deskto
	16GB RAM DDR4 1.2V	1	$57.99	$6.40	$64.39	Buy	https://www.bhphotovideo.com/c/product/13338
	250GB SSD	1	$65.99	$12.00	$77.99	Buy	https://www.bhphotovideo.com/c/product/13825
	120V-24V DC/DC Converter	1	$70.00	$5.00	$75.00	Inherit	N/A
	TDK-Lambda	1	$33.55	$7.99	$41.54	Buy	https://www.mouser.com/ProductDetail/TDK-La
	TDK-Lambda converters	1	$33.00	$5.00	$38.00	Buy	https://www.mouser.com/ProductDetail/967-I6A
	12V-3.3V DC/DC Converter	1	$8.00	$5.00	$13.00	Inherit	N/A
	PCB	3	$33.00	$10.00	$109.00	Buy	www.advancedcircuits.com
	Raspberry Pi 3 B+	1	$35.00	$3.00	$38.00	Inherit	N/A
	Ethernet Cable	2	$3.00	$0.00	$6.00	Inherit	N/A
	Raspberry Pi	1	$43.75	$43.75	$87.50	Buy	https://www.mouser.com/ProductDetail/Adafruit
	USB 2.0 A Male to A Male	1	$4.94	$4.94	$9.88	Buy	https://www.amazon.com/dp/B009GUXG92/ref=
	USB3.0 A Male to A Female Cable	1	$6.49	$6.49	$12.98	Buy	https://www.amazon.com/AmazonBasics-Extens
	Ethernet to usb3.0 adapter	1	$14.99	$14.99	$29.98	Buy	https://www.amazon.com/dp/B07MK6DJ6M/ref=
	12V Male+Female 2.1x5.5MM DC Power Jack Plug Adapter Connector	1	$4.50	$4.50	$9.00	Buy	https://www.amazon.com/Ksmile%C2%AE-Fem
	DC Power Pigtail Cable Wire, 12V 5A Male & Female Connectors	1	$9.89	$9.89	$19.78	Buy	https://www.amazon.com/43x2pcs-Connectors-
	Omnihil Adapter Plug Converter 5.5 x 2.1mm Female Plug to 5.5x2.5 Male Plug	1	$6.96	$6.96	$13.92	Buy	https://www.amazon.com/OMNIHIL-Adapter-Co
	PCB Female DC Power Jack Socket Connector	1	$5.99	$5.99	$11.98	Buy	https://www.amazon.com/Uxcell-a15012900ux0
	USB 3.0 to micro-B Cable	1	$7.00	$1.00	$8.00	Inherit	N/A
	USB 3.0 to micro-USB	1	$7.00	$1.00	$8.00	Inherit	N/A
	PCB USB2.0 connecter	1	$1.35	$1.35	$2.70	Buy	https://www.mouser.com/ProductDetail/Wurth-E
	USB Connectors WR-COM USB Type A	3	$1.59	$7.99	$12.76	Buy	https://www.mouser.com/ProductDetail/Wurth-E
	Screw Terminals 5mm Pitch (2-Pin)	3	$0.95	$0.00	$2.85	Buy	https://www.mouser.com/ProductDetail/SparkFu
Structures	6061 Al 8"x8"x.25" (Front/Back)	2	$18.31	$70.92	$107.54	Buy	https://www.mcmaster.com/9246K11
	6061 Al 6"x48"x.25" (Bottom/Top)	1	$55.96	$0.00	$55.96	Buy	https://www.mcmaster.com/9246K424
	6061 Al. 6"x"48"1.25" (Left/Right)	1	$224.92	$0.00	$224.92	Buy	https://www.mcmaster.com/9246K625
	M2 Screws (20mm), x25	3	$9.40	$0.00	$28.20	Buy	https://www.mcmaster.com/91294A539
	M2 Screws (10mm), x25	3	$7.51	$0.00	$22.53	Buy	https://www.mcmaster.com/91294A006
Sensors	Time of Flight Camera	1	$1,547.00	$0.00	$1,547.00	Inherit	N/A
	Camera Sensor	1	$725.00	$0.00	$725.00	Inherit	N/A
	Camera Lens	1	$500.00	$0.00	$500.00	Inherit	N/A
	GPS Antenna	1	$50.81	$21.56	$72.37	Buy	https://www.taoglas.com/product/agpsf-36c-07-
	GPS Receiver	1	$199.95	$0.00	$199.95	Buy	https://www.sparkfun.com/products/15005?_ga=
	Serial Breakout Board	1	$7.95	$0.00	$7.95	Buy	https://www.sparkfun.com/products/14050
Test	Internal 1 inch PVC coupling	20	$1.77	$10.60	$46.00	Buy	https://www.pvcfittingsonline.com/1-pvc-internal
	1/2 2x2 MDF	1	$6.93	$0.00	$6.93	Buy	Home Depot
	2x4-96" Stud	2	$3.22	$0.00	$6.44	Buy	Home Depot
	3/4" x 10' PVC	10	$1.86	$0.00	$18.60	Buy	Home Depot
	Wood Screws	1	$8.72	$0.00	$8.72	Buy	Home Depot
	1/2" x 10' PVC	1	$1.88	$0.00	$1.88	Buy	Home Depot
	7/16" Wrench	2	$3.97	$0.00	$7.94	Buy	Home Depot
	1/4" Locking Nut	2	$1.18	$0.00	$2.36	Buy	Home Depot
	1/4" Hex Bolt	4	$0.52	$0.00	$2.08	Buy	Home Depot
	Foam	1	$5.98	$0.00	$5.98	Buy	Home Depot
	WD-40	1	$8.87	$0.00	$8.87	Buy	Home Depot
	White Spray Paint	1	$6.98	$0.00	$6.98	Buy	Home Depot
	1" x 10' PVC	10	$2.70	$0.00	$27.00	Buy	Home Depot
	track wheels	4	$39.00	$0.00	$156.00	Buy	https://www.bhphotovideo.com/c/product/11890
	White PLA 1KG Spool	1	$23.99	$0.00	$23.99	Buy	https://www.amazon.com/Overture-Filament-Pro
	Black PLA 1KG Spool	1	$23.99	$0.00	$23.99	Buy	https://www.amazon.com/HATCHBOX-3D-Filam
	uxcell DC 24V 111RPM	1	$34.99	$0.00	$34.99	Buy	https://www.amazon.com/uxcell-50Kg-cm-Self-L
	Rankie USB 3.0 Cable, Type A to Type A, 1-Pack 6 Feet	2	$5.99	$0.00	$11.98	Buy	https://www.amazon.com/Rankie-Cable-Type-1-
Total					**$2,307.26**		
Surplus					**$2,692.74**		

11.4. Systems Engineering

Figure 69: Tabled Risk Mitigation Assessment

RISK ID	IF	THEN	ORIGINAL SEVERITY	ORIGINAL PROBABILITY	RISK SCORE	MITIGATION STRATEGY	POST-MITIGATION SEVERITY	POST-MITIGATION PROBABILITY	POST-MITIGATION RISK SCORE
LANG CODE	Different code languages are not properly integrated with different compilers and IDEs	VISION can lose functionality during software system integration and automation	2	4	8	Python API for MATLAB using matlab.engine class package	1	2	2
DISASSM	NUC is damaged during any testing/operation	VISON may not be able to process onboard data	3	4	12	VISION will purchase spare unit for replacement if damage occurs	2	3	6
IN POS	If Inertial Position is not calculated	Unanswered uncertainty will accrue in TLE estimates	3	4	12	VISION will assume deployment company will provide inertial state in or before deployment manifest	1	1	1
POW	If any component shorts out	the entire electronics system will fail as there is a single power source	4	3	12	Building a redundant printed circuit board so that in the case of a short operation can continue	1	2	2
DEPL ORB	HCW equations assume a circular deployer orbit	Non-zero deployer eccentricity introduces model error to HCW CKF	2	5	10	Using Tschauner-Hempel dynamics equations	1	2	2
UNC PRF	If processed sensor uncertainty profile is non-gaussian and has a non-zero mean	CKF performance degrades with non-gaussian distributions	3	3	9	Implementing a built up filter using least squares which enables a Kalman Filter	2	2	4
OBJ DETCT	If object detection fails	VISION will be unable to assign centroids	2	3	6	Find the centroid of the system of the visible CubeSats	2	1	2
VIBE	If VIBE testing is not performed	VISION will fail to certify structure a TRL	2	4	8	VISION will use AERO provided VIBE table to perform basic testing	1	2	2
CH MNUF	If chassis manufacturing encounters road blocks	Cost of time and manpower to compensate for lack of structure needed for testing	2	4	8	Chassis structure will be segmented into multiple parts	1	2	2
CKF Tune	Ability to select initialization parameters for CKF greatly affects performance	CKF performance degrades with poorly tuned initialization parameters	2	3	6	CKF extensions such as U-D factorization and square-root preprocessing.	1	2	2
CU ADAPT	Inability to handle variable number of CubeSats and blank data	will cause filter not to work	2	4	8	Design modular algorithms for data preprocessing and state estimate filtering	1	1	1
SAMP RTE	Filter does not compensate for different sample rates of visual/tof sensors	Inability to utilize both sensors effectively will decrease filter performance.	4	2	8	Format incoming data by time, flag with sensor type, modify filter to handle differential information breadth.	1	1	1

11.5. Leassons Learned

Date: 2/10/20
Title: PCB Test

Event: Electronic and testing team members received all electrical components to be soldered to the printed circuit board (PCB), and determined the placement of all components before soldering. The components were all soldered to the PCB and all connections were checked. The PCB was then connected to a power supply, as well as several other pieces of hardware, including a Raspberry Pi. The PCB was then supplied with roughly 25VDC. Upon supplying the PCB with 25VDC, the raspberry pi started to smoke, and the power supply was quickly put into standby. All hardware was disconnected and the PCB was again supplied 25VDC and the output voltages from the two DC/DC converters were measured. It was at this point the team discovered that the two DC/DC converters had been switched. This means the raspberry pi was provided 12VDC instead of 5VDC. Lessons Learned/recommendations:

- A testing officer should be present and sign off on every step of a procedure before following steps are taken
- The output voltages of the PCB should have been checked before any hardware was tested
- Never trust the “definitely correct” schematic or layouts, check everything with suspicion
- Make sure the labels are all correct and double or triple checked with someone who doesn’t know at all with explaining with them the whole procedure.
- Configurations should be checked/tested before made permanent

11.6. Test Procedures

Electronic Subsystem DC/DC Testing Procedure

Test 1

January 15, 2020

Objective: To verify that our electronic board functions as expected in order to have the PCB manufactured.

Purpose: The purpose of this test is to verify the DC/DC converters functionality as well as to ensure that all voltage breakdown are correct.

Materials:

1. 120-24V DC/DC converter _____
2. 24-12V DC/DC converter ____
3. 24-5V DC/DC converter ____
4. 12-3.3V DC/DC converter ____
5. 2x 2.2 kOhm resistors ____
6. 2x 36 Ohm resistors ____
7. 2x 20 Ohm resistors ____
8. 1x 1.2 kOhm resistors ____
9. 1x 200 Ohm resistor _____
10. 1x 10 kOhm resistor ____
11. 3 Ohm power resistor ____
12. 6 Ohm power resistor ____
13. 10 Ohm power resistor ____
14. 14 Ohm power resistor ____

Power and Equipment

15. 120V power source_____
16. ESD mat ____
17. Agilent Digital Multimeter ____
18. Banana Clips ____
19. Alligator clips ____
20. Oscilloscope ____
21. Bread board ____

Personnel Roles:

One member from each of the following sub teams must be present at the test.

Electronics: Responsible for performing the test and making measurements. _____

Safety: Responsible for maintaining the safety of all personal during the test _____

Testing: Responsible for recording the results from test. _____

Set-Up:

Pre-Test procedure

1. Safety Checks
    a. Tape leads to the power supply ____

2. Set up power supply
    a. Connect red wire to + output ____
    b. Connect black wire to - output ____
    c. Ensure wires are secured ____
    d. Connect red alligator clip to red wire ____
    e. Connect black alligator clip to black wire ____
    f. Hook up banana clips to voltmeter ____
    g. Connect wires in parallel to be connect to breadboard ____

Testing Procedure

Start Time: ____ AM/PM

Test Seven

1. Set up breadboard for seventh configuration as shown in set up diagram _____
   Notes:

2. Second party check configuration _____
   Notes:

3. Turn on power supply _____
   Notes: Make sure the ground is connected _____

4. Put power supply into standby _____
Notes:

5. Safety Check: Make sure the power supply is in standby _____
Notes:

6. Check Power supply is at 120V _____
   Notes:

7. Connect power supply to breadboard _____
   Notes:

8. Turn off standby for power supply ____
Note:

9. Check 120-24V DC/DC converter output _____
   Notes:
   - The output voltage should be 24V.
     - Actual output voltage:
   - The ripple voltage of the output voltage
     - Actual Vp-p,ripple:

10. Check 24-12V DC/DC converter output _____
    Notes:
    - The output voltage should be 12V.

    - Actual output voltage:
- The ripple voltage of the output voltage
    - Actual Vp-p,ripple:

11. Check 24-5V DC/DC converter output _____
    Notes:
    - The output voltage should be 5V.
        - Actual output voltage:
    - The ripple voltage of the output voltage
        - Actual Vp-p,ripple:

12. Check 12-3.3V DC/DC converter output _____
    Notes:
    - The output voltage should be 3.3V.
        - Actual output voltage:
    - The ripple voltage of the output voltage
        - Actual Vp-p,ripple:

13. Turn off power supply _____

Notes:

14. Safety Check: Make sure power supply is off ____

Notes:

Clean-Up

1. Return 120V power supply _____
2. Return power resistors _____
3. Return resistors _____
4. Return capacitors _____
5. Turn off all voltmeters and oscilloscopes _____
6. Clean up work space _____

Post-Test Analysis

1. Determine if DC/DC system works as expected _____
   Notes:

2. Note the date of the next test. ____/____/2017
   Notes:

Electronic Subsystem Nuc Test Procedure

January 15, 2020

Objective: To verify that our Intel NUC functions as expected

Materials:

1. Intel NUC
2. USB3.0 Cable
3. Power Cable

Personnel Roles:

One member from each of the following sub teams must be present at the test.

Electronics: Responsible for performing the test and making measurements. _____

Testing: Responsible for recording the results from test. _____

Testing Procedure

Start Time: ____ AM/PM

1. Connect the provided power cable to the NUC _____
   Notes:

2. Check that NUC operates while being powered by the provided cable _____
Notes:

3. Connect a self-provided power cable to the NUC _____
   Notes:

4. Check that NUC operates while being powered by a self-provided power cable _____
Notes:

5. Check Ethernet ports functionality _____
   Notes:

Post-Test Analysis

1. The NUC powered by the provided cable _____
   Notes:

2. The NUC was powered by the self-provided cable _____
Notes:

3. The Ethernet ports function as expected _____
Notes:

4. Note the date of the next test. ____/____/2020

Electronic Subsystem PCB Test Procedure

Test 1

February 10, 2020

Objective: The objective of this test is to verify that our PCB provides the correct voltages to our hardware.

Purpose: The purpose of this test is to verify that all hardware receives the appropriate amounts of power necessary to operate from the PCB and function as expected.

Materials:

1. PCB ____
2. 120-24V DC/DC converter _____

Power and Equipment

3. 120V power source_____
4. ESD mat ____
5. Agilent Digital Multimeter ____
6. Banana Clips ____
7. Alligator clips ____
8. Oscilloscope ____
9. Bread board ____

VISION Hardware and Sensors

10. Time of Flight Camera ____
11. NUC ____
12. Raspberry Pi ____
13. Monochrome Camera ____
14. GPS ____

Personnel Roles:

One member from each of the following sub teams must be present at the test.

Electronics: Responsible for performing the test and making measurements. _____

Safety: Responsible for maintaining the safety of all personal during the test _____

Testing: Responsible for recording results and ensuring the test procedure is followed _____

Testing Procedure

Start Time: ____ AM/PM

1. Set power supply to 24.9VDC ____

Notes:

2. Put power supply in standby ____

Notes:

3. Connect the power supply to the PCB ____

Notes:

4. Take the power supply off standby____

Notes:

5. Check that the 24-5V DC/DC converter output voltage is 5VDC ____

Notes:

6. Check that the 25-12V DC/DC converter output voltage is 12VDC ____

Notes:

7. Put the power supply back into standby ____

Notes:

8. Connect Raspberry Pi to output from PCB ____

Notes:

9. Take power supply off standby____

Notes:

10. Check the Raspberry Pi powers on ____

Notes:

11. Put the power supply back on standby____

Notes:

12. Connect NUC to output from PCB ____

Notes:

13. Take power supply off standby____
Notes:

14. Check the NUC powers on ____
Notes:

15. Put the power supply back on standby____
Notes:

16. Connect GPS to NUC ____
Notes:

17. Take the power supply off standby____
Notes:

18. Check the GPS powers on ____
Notes:

19. Put the power supply back on standby____
Notes:

20. Connect Monochrome camera to NUC ____
Notes:

21. Take the power supply off standby____
Notes:

22. Check the Monochrome camera powers on ____
Notes:

23. Put the power supply back on standby____
Notes:

24. Disconnect all non-permanently connected wires ____
Notes:

Clean-Up

1. Turn off all voltmeters and oscilloscopes _____
2. Clean up work space _____

Post-Test Analysis

1. Determine if the respective output powers from the DC/DC convertors will power the respective hardware ____

Notes:

2. Determine if the PCB correctly powers all equipment ____
   Notes:

System Integration Test Procedure

Objective:

Purpose:

Materials:

1. 20x Rails _____
2. 20x Rail Stands ____
3. Winch ____
4. Cart ____
5. 3x Mock CubeSats ____
6. 120V Power Supply ____
7. Fiducial Markers ____
8. Vision Package ____
9. Voltmeter ____
10. Vicon System ____
11. Personal Computer ____

Personnel Roles:

One member from each of the following sub teams must be present at the test.

Electronics: Responsible for setting up the power for the winch and Vision package. _____

Software: Responsible for providing Vision with the bootup commands. _____

Safety: Responsible for maintaining the safety of all personnel and equipment during the test _____

Testing: Responsible for recording results and ensuring the test procedure is followed _____

Set-Up:

Pre-Test procedure

1. Set up 120V power supply
    a. Connect the red wire to + output ____
    b. Connect the black wire to - output ____
    c. Ensure wires are secured ____
    d. Connect voltmeter in parallel with power output ____

2. Safety Checks
    a. Ensure 120V power supply is off____
    b. Ensure all connections and exposed wires have been taped to avoid direct contact____
    c. Tape leads to the top of the power supply _____
    d. Plug in power supply____
    e. Put power supply in standby____
3. The rail and winch system
    a. Connect each of the rail segments to their corresponding partners ____
    b. Lay the rails on the stands ____
    c. Level the rails so they are as flat
    d. Straighten the rails
4. Vicon system
    a. Boot-up the system ____
    b. Use the wand in order to start the calibration process ___
    c. Set the origin point ____

Testing Procedure

Start Time: ____ AM/PM

1. Set the cart on the rail ____

Notes:

2. Set VISION in front of the rails ____

Notes:

3. Ensure VISION is properly aligned ____

Notes:

4. Power on the winch system ____

Notes:

5. Pull the cart along the rails ____

Notes:

6. Valide the velocity the cart was pulled at was also the desired velocity ____

Notes:

7. Reset the cart at the start of the rail ____

Notes:

8. Send the boot up command from the personal computer to the Raspberry Pi ____

Notes:

9. Validate that the Raspberry Pi has booted up correctly ____

Notes:

10. Validate that all other hardware as powered on and is ready for data capture ____

Notes:

11. Begin data capture on both the vicon system and Vision package ____

Notes:

12. Pull the cart along the rail system____

Notes:

13. Ensure the cart stops before the end of the rail____

Notes:

14. Stop the Vicon data capture____

Notes:

15. Check the software has finished running ____

Notes:

16. ____

Notes:

Clean-Up

1. Return 120V power supply _____
2. Turn off all voltmeters _____
3. Take rails off stands ____
4. Take rails appart ____
5. Remove all hardware and testing equipment from the Aspen Lab _____
6. Clean up work space _____

Post-Test Analysis

1. Determine if VISION could provide an image of each CubeSat _____

Notes:

2. Validate subsystem tests results ____

Notes:

Time of Flight Camera Test Procedure

Objective: Verify the accuracy from the manufacturer's claims

Purpose: Capture and determine a noise profile for the filter, we expect it to be mostly Gaussian

Materials:

1. Time of flight camera _____
2. Personal computer ____
3. Mock cubesats ____
4. Power supply ____
5. Cables for power supply ____

Personnel Roles:

One member from each of the following sub teams must be present at the test.

Software: Responsible for running the software and for any analysis _____

Safety: Responsible for maintaining the safety of all personnel and equipment during the test _____

Testing: Responsible for recording results and ensuring the test procedure is followed _____

Set-Up:

Pre-Test procedure

1. Set up 120V power supply
    a. Connect the red wire to + output ____
    b. Connect the black wire to - output ____
    c. Ensure wires are secured ____
    d. Connect voltmeter in parallel with power output ____

2. Safety Checks
    a. Ensure 120V power supply is off____
    b. Ensure all connections and exposed wires have been taped to avoid direct contact____
    c. Tape leads to the top of the power supply _____
    d. Plug in power supply____
    e. Put power supply in standby____

Testing Procedure

Start Time: ____ AM/PM

1. Set up the time of flight camera with 24 volts and 4 amps ____

Notes:

2. Set up mock CubeSat____

Notes:

3. Set up the time of flight camera so that it is point at the mock cubesats ____

Notes:

4. Start the automated software ____

Notes:

5. Ensure that data is being captured ____

Notes:

6. Run the saved data through the post-processing software ____

Notes:

7. Go back to "2." and perform 2nd test with a new mock CubeSat orientation ____

Notes:

8. Go back to "2." and perform 3rd test with a new mock CubeSat orientation ____

Notes:

9. Go back to "2." and perform 4th test with a new mock CubeSat orientation ____

Notes:

10. Go back to "2." and perform 5th test with a new mock CubeSat orientation ____

Notes:

Clean-Up

1. Return 120V power supply _____
2. Turn off all voltmeters _____
3. Remove all hardware and testing equipment from the Lab _____
4. Clean up work space _____

Post-Test Analysis

1. Compare the results from the post-processing software with the manual thruth data _____

Notes:

References

[1] ASM Material Data Sheet, http://asm.matweb.com/search/SpecificMaterial.asp?bassnum=MA7075T6.

[2] Aboaf, A.; Renninger, N.; Lufkin, L. 2019. "Design of an In-Situ Sensor Package to Track CubeSat Deployments," *Proceedings of the AIAA/USU Conference on Small Satellites*, FJR Student Paper Competition, 141. http://digitalcommons.usu.edu/smallsat/2019/all2019/141/.

[3] Axelrad, P., and Behre, C. P., "GPS Based Attitude Determination for Spinning Satellites," Oct. 1997.

[4] Baldwin, Cameron W. Chassis1_4_Freq-Frequency. SolidWorks, 2019, Chassis1_4_Freq-Frequency.

[5] Boylston, A., J.A. Gaebler, and P. Axelrad, "Extracting CubeSat Relative Motion Using In Situ Deployment Imagery," *Proceedings of 42nd Annual AAS Guidance & Control Conference*, Breckenridge, CO, AAS 19-016, 10 pages, Feb 2019.

[6] BUDYNAS, RICHARD G., and WARREN C. YOUNG. ROARKS FORMULAS FOR STRESS AND STRAIN. 7th ed., MCGRAW-HILL EDUCATION, 2002.

[7] Fitzgerald, Joe. "Why Is There So Much TLE Confusion When New Cubesats Are Launched?" *AMSAT*, February 20, 2018. www.amsat.org/why-is-there-so-much-tle-confusion-when-new-cubesats-are-launched/

[8] Foust, Jeff. "More Startups Are Pursuing CubeSats with Electric Thrusters". *SpaceNews*. July 23, 2018, from https://spacenews.com/more-startups-are-pursuing-cubesats-with-electric-thrusters/

[9] Gaebler, J.A and P. Axelrad, "Improving Orbit Determination of Clustered CubeSat Deployments using Camera-Derived Observations" *Proceedings of 42nd Annual AAS Guidance & Control Conference*, Breckenridge, CO, AAS 19-041, February 2019.

[10] Grush, Loren, "Why The Air Force Still Cannot Identify More Than A Dozen Satellites From One December Launch", *The Verge*. April 2, 2019, from https://www.theverge.com/2019/4/2/18277344/space-situational-awareness-air-force-tracking-sso-a-spaceflight-cubesats

[11] Jackson, Jelliffe. "Project Definition Document (PDD)", University of Colorado–Boulder, Retrieved August 29, 2019, from https://canvas.colorado.edu/

[12] Lan, W. *Poly Picosatellite Orbital Deployer Mk. III Rev. E User Guide* CubeSat - California Polytechnic State University. *Revised:* March 4, 2014.

[13] Pandey, Parul. "10 Python image manipulation tools". *opensource.com*. March 18, 2019, from https://opensource.com/article/19/3/python-image-manipulation-tools

[14] Riesing, Kathleen, "Orbit Determination from Two Line Element Sets of ISS-Deployed CubeSats" *Proceedings of 29th Annual AIAA/USU Conference on Small Satellites*, Logan, UT,August 2015.

[15] Schaub, Hanspeter and Junkins. "Analytical Mechanics of Space Systems." (2003).

[16] Tapley, Shutz and Born. "Statistical Orbit Determination."

[17] Wong, William G. "Python's Big Push into the Embedded Space". *ElectronicDesign*. August 29, 2018, from https://www.electronicdesign.com/embedded-revolution/python-s-big-push-embedded-space

[18] Wu, Elaine. "NVIDIA JETSON NANO DEVELOPER KIT DETAILED REVIEW". *seeed Studio Blog*. April 3, 2019, from https://www.seeedstudio.com/blog/2019/04/03/nvidia-jetson-nano-developer-kit-detailed-review/

[19] "CubeSat Concept and the Provision of Deployer Services". *eoPortal Directory*. Retrieved from: https://directory.eoportal.org/web/eoportal/satellite-missions/c-missions/cubesat-concept

[20] Van Atten, W.,*SYSTEMS ENGINEERING LECTURES* Sep.2019.

[21] NVIDIA Jetson Nano. *User Guide*. From: https://elinux.org/Jetson_Nano

[22] Intel NUC (Intel NUC8i7BEH, Intel NUC8i7HNK, Intel NUC8i7HVK). *User Guide*. From: https://www.intel.com/content/www/us/en/products/boards-kits/nuc/kits.html

[23] Xilinx FPGA board (Xilinx Zynq-7000 SoC ZC702, Xilinx Zynq Ultrascale+ MPSoC). *User Guide*. From: https://www.mouser.com/Xilinx/Embedded-Solutions/Engineering-Tools/Programmable-Logic-IC-Development-Tools/_/N-cxcznZ1yzvvqx?P=1yzohtwZ1y8efd7&FS=True

[24] GPU rank website . From: https://www.notebookcheck.net/Mobile-Graphics-Cards-Benchmark-List.844.0.html

[25] "GeekBench" – CPU benchmark. From: https://browser.geekbench.com/

[26] Image Processing Toolbox – MATLAB. From: https://www.mathworks.com/discovery/digital-image-processing.html

[27] Embedded Encoder – MATLAB. From: http://msdl.cs.mcgill.ca/people/mosterman/presentations/date07/tutorial.pdf

[28] C++ OpenCV edge detection. From: https://docs.opencv.org/trunk/da/d22/tutorial_py_canny.html

[29] "O3D313 Time of Flight Camera." Ifm Electronic, 2017, www.ifm.com/us/en/product/O3D313.

[30] "Ueye Camera Manual." IDS, en.ids-imaging.com/manuals-ueye-software.html.

[31] "Celestrack", https://celestrak.com/columns/v04n03/.