

**University of Windsor Space and Aeronautics Team (WinSAT):**

**Attitude Determination and Control System**

Capstone Design Summer 2020

ELEC-4000: Capstone Design Project

Department of Electrical and Computer Engineering

University of Windsor

August 7<sup>th</sup>, 2020



University  
of Windsor

**WINSAT**

**University of Windsor Space and Aeronautics Team (WinSAT):  
Attitude Determination and Control System**

BY

|                  |           |
|------------------|-----------|
| ASHETI, Mahmoud  | 104587881 |
| GILL, Harpreet   | 103809539 |
| MANGAT, Gurshawn | 104233452 |
| NASSAR, Kassem   | 103824480 |

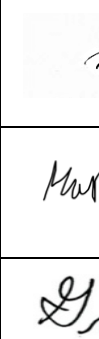
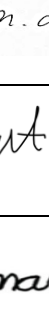

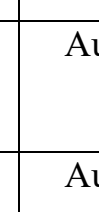
Supervisor: Dr. Shahpour Alirezaee

Department of Electrical and Computer Engineering  
University of Windsor

August 7<sup>th</sup>, 2020

**University of Windsor Space and Aeronautics Team (WinSAT):**  
**Attitude Determination and Control System**

“No action by any design team member contravened the provisions of the Code of Ethics and we hereby reaffirm that the work presented in this report is solely the effort of the team members and that any work of others that was used during the execution of the design project or is included in the report has been suitably acknowledged through the standard practice of citing references and stating appropriate acknowledgments”. The presence of the author's signatures on the signature page means that they are affirming this statement.

| <b>Name</b>      | <b>Initials</b> | <b>Student ID</b> | <b>Signature</b>  | <b>Date</b>                      |
|------------------|-----------------|-------------------|---|----------------------------------|
| ASHETI, Mahmoud  | MA              | 104587881         |    | August 7 <sup>th</sup> ,<br>2020 |
| GILL, Harpreet   | HG              | 103809539         |  | August 7 <sup>th</sup> ,<br>2020 |
| MANGAT, Gurshawn | GM              | 104233452         |  | August 7 <sup>th</sup> ,<br>2020 |
| NASSAR, Kassem   | KN              | 103824480         |  | August 7 <sup>th</sup> ,<br>2020 |

## **Abstract (GM)**

The University of Windsor Space & Aeronautics (WinSAT) team was challenged to design a 3U CubeSat which would have the capability of producing a space selfie, dubbed the “Selfie Sat”, which would be submitted to the Canadian Satellite Design Challenge (CSDC). Within the WinSAT team, there were several subdivisions, this report will showcase and highlight the work completed by the Attitude Determination and Control System (ADCS) subdivision over the 2020 Capstone term.

As the name states, the ADCS subdivision was tasked to provide determination and control of the CubeSat’s orbital attitude and position. Within the ADCS subdivision, the team split has two further divisions. The first team was tasked with Attitude Determination and the second team was tasked with Attitude Control. Attitude Determination encapsulates the two actuators, reaction wheels and magnetorquers, which were utilized in order to design the control system that would control the CubeSat’s orbital attitude. On the other hand, attitude control was tasked to collect input data from various sensors (gyroscope, magnetometer, and sun sensor) in order to output the CubeSat’s position, velocity, and orientation while in orbit.

To visualize this control and attitude determination, algorithms were developed via Python and MATLAB to acquire the required parameters for the two actuators and the sensor inputs for the design of the controller. Once these inputs and parameters were obtained, Systems Tool Kit (STK) was utilized to provide a visual aid to see the system in effect showcasing the orbit of the CubeSat and executing its BDOT algorithm for the Detumbling mode. Alongside, MATLAB scripts were generated in order to showcase Nadir pointing and Target Pointing via output graphs.

Within this report, the respective authors have been credited for the sections they have contributed to with their initials appearing within the header of the section.

# Table of Contents

|  |           |
|--|-----------|
| <b>1.0 Introduction (MA)</b> .....                                   | <b>1</b>  |
| <b>2.0 Benchmarking (HG)</b> .....                                   | <b>2</b>  |
| <b>3.0 Design Criteria, Constraints, and Deliverables</b> .....      | <b>3</b>  |
| <b>3.1 Constraints (GM)</b> .....                                    | <b>3</b>  |
| <b>3.2 Reaction Wheels (MA)</b> .....                                | <b>3</b>  |
| <b>3.3 Magnetorquers (GM)</b> .....                                  | <b>4</b>  |
| <b>3.4 Sensors (HG)</b> .....  | <b>4</b>  |
| <b>3.5 Controls (KN)</b> .....                                       | <b>6</b>  |
| <b>4.0 Design Methodology</b> .....                                  | <b>8</b>  |
| <b>4.1 Reaction Wheels (MA)</b> .....                                | <b>8</b>  |
| <b>4.2 Magnetorquers (GM)</b> .....                                  | <b>9</b>  |
| <b>4.3 Sensors (HG)</b> .....  | <b>11</b> |
| <b>4.4 Controls (KN)</b> .....                                       | <b>14</b> |
| <b>5.0 Physical Implementation/Simulation Model Validation</b> ..... | <b>19</b> |
| <b>5.1 Reaction Wheels (MA)</b> .....                                | <b>19</b> |
| <b>5.2 Magnetorquers (GM)</b> .....                                  | <b>19</b> |
| <b>5.3 Sensors (HG)</b> .....  | <b>19</b> |
| <b>5.4 Controls (KN)</b> .....                                       | <b>20</b> |
| <b>6.0 Experimental Methods/Model Validation</b> .....               | <b>22</b> |
| <b>6.1 Controls (KN)</b> .....                                       | <b>22</b> |
| <b>7.0 Design Specifications and Evaluation Matrix</b> .....         | <b>23</b> |
| <b>7.1 Sensors (HG)</b> .....  | <b>23</b> |
| <b>7.2 Controls (KN)</b> .....                                       | <b>23</b> |
| <b>8.0 Budget (MA)</b> .....   | <b>24</b> |
| <b>9.0 Conclusions (GM)</b> .....                                    | <b>25</b> |
| <b>References</b> .....  | <b>27</b> |
| <b>Appendices</b> .....  | <b>28</b> |

## **1.0 Introduction (MA)**

The WinSAT team is the University of Windsor's engineering team that was competing in the Canadian Satellite Design Challenge (CSDC). The team was tasked with designing, building, testing and possibly launching a 3U CubeSat into low orbit space. A CubeSat is measured as a single unit "U" which is constructed of a cube that is 10x10x10 cm. The CSDC had required the WinSAT team to design a 3U CubeSat sized at 34.05x10x10 cm. Within the WinSAT team, there are various sub-teams that include: Structural, Electrical Power Systems (EPS), Thermal, Command and Data Handling (CDH), Payload, Radio Communications, Business, and Attitude Determination and Control Systems (ADCS). This capstone team has accomplished all the research, designs, and deliverables for the ADCS sub-division. The main design objective of the ADCS team was divided into two categories: Attitude Control and Attitude Determination. The attitude control focuses on developing a control system using actuators to control the satellite's orbital attitude. Meanwhile, the attitude determination focuses on collecting data from the gyroscope, magnetometer, and sun sensor to output the satellite's position, velocity and orientation in orbit. The team developed a functional controller that was able to operate in the following three stages: Detumbling, Nadir Pointing and Target Pointing. Within the detumbling phase, the B-Dot algorithm used in the MATLAB script will take the satellite's initial uncontrolled angular velocity and will substantially decrease it to a near zero value. Within the Nadir pointing stage, the satellite will adjust the attitude from its original orientation to any other orientation to align the camera with Nadir (directly perpendicular to earth). Furthermore, the final stage will be the Target Pointing stage where the satellite will target its antenna to accurately point down towards the ground station for data transfer. The scope of the project included the design of the reaction wheels, the magnetorquers, the sun sensor and the controller. The first two team members worked on developing the final designs of the two actuators: reaction wheels and magnetorquers. However, due to global circumstances, they both finished designs of each actuator; however, they did not complete the physical models. As a result, they aided the other two members with the development of the sun sensor and the sliding mode controller. Two members had worked on constructing and simulating the sun sensor on both MATLAB and LTspice, while the other two members worked on designing and simulating the controller and the algorithms for each one of the three stages using MATLAB and STK.

## 2.0 Benchmarking (HG)

The sensor selection for the 3U CubeSat controls involved extensive literature review to determine commonly used sensors for other CubeSat missions. As the team read through the literature, the team would that the sensors use many other CubeSat missions included the gyroscope, magnetometer and a coarse sun sensor. The literature review consisted of an analysis of the different sensors and their application to determine their use in teams CubeSat. A commonly mentioned magnetometer in other CubeSat projects was the HMC5883L, a 3-axis sensor that should be used with the SparkFun breakout board. This sensor has been used in many CubeSat projects, resulting in an excellent flight heritage, low cost and optimal 3.3V operation [1]. Next the gyroscope, a commonly used sensor in other CubeSat missions, was the EVAL-ADXRS450Z-M, a simple breakout board with a digital output and a single axis sensing capability [1]. The review of many of the literature papers concluded with the team drawing up a summary table of the comparison of all the sensors considered and is shown in the figure below.

| Part #                               | Price     | Quantity | Net       | Voltage      | Current     | Power (W) | Interface | Mass | Dimensions    | Accuracy  |
|--------------------------------------|-----------|----------|-----------|--------------|-------------|-----------|-----------|------|---------------|---|
| Magnetometer                         |           |          |           |              |             |           |           |      |               |   |
| HMC6343                              | \$ 154.95 | 1        | \$ 154.95 | 3.3V         | 4.5mA       | 0.01485   | I2C       |      | 9x9x1.9mm     | 2deg RMS/1" yaw and roll                        |
| LSM303 - and Accelerometer           | \$ 14.95  | 1        | \$14.95   | 3.3V         | 0.1mA       | 0.00033   | I2C       |      | 3x5x1mm       | 1-2°  |
| MLX90393                             | \$ 14.95  |          |           |              | 3.3         |           | I2C       |      |               |   |
| HMC5883                              | \$ 35.00  | 1        |           | 2.16 to 3.6V | 100 µA      | 0.0003    | SPI/I2C   |      | 3.0x3.0x0.9mm | ±0.4mT  |
| Multisensor                          |           |          |           |              |             |           |           |      |               |   |
| ISM303DAC                            | \$ 6.76   | 1        | \$ 6.76   | 1.8V         |             | 0.0024156 | SPI/I2C   | -    | 2x2x1mm       |   |
| Accelerometer                        |           |          |           |              | 162µA       | 0.0002916 |           |      |               | 2-16+/- g                                       |
| Magnetometer                         |           |          |           |              | 1180µA      | 0.002124  |           |      |               | 49.152gauss                                     |
| MPU-9250                             | \$ 16.45  | 1        | \$ 16.45  | 2.5V         | 3.5mA (All) | 0.00875   | SPI/I2C   |      | 3x3x1mm       |   |
| Accelerometer                        |           |          |           |              | 450µA       |           |           |      |               | 2-16+/- g                                       |
| Magnetometer                         |           |          |           |              | 280µA       |           |           |      |               | 4800+/- uT                                      |
| Gyroscope                            |           |          |           |              | 3.2mA       |           |           |      |               | 250-2000deg/sec                                 |
| NXP Precision 9DoF                   | \$ 14.95  | 1        | \$14.95   |              |             |           |           |      |               |   |
| Gyroscope(FXAS21002)                 |           |          |           | 2-3.6V       | 2.7mA       | 8.19E-03  | SPI/I2C   |      |               | At +/- 2000 3.125 dps At +/- 250 dps 0.3906 dps |
| Accelerometer/Magnetometer(FXOS8700) |           |          |           | 2-3.6V       | N/A         | N/A       | SPI/I2C   |      |               | 0.1 µT/LSB                                      |

Figure 1: Literature review of the most commonly used sensors for CubeSat missions

The sensor shown in red is one that was very commonly used in other CubeSat teams, but was not available on the current market and was therefore not considered. The sensors highlighted in orange are those that have been thoroughly researched and an analysis of each data sheet of the sensor has been carried out to determine the best fit for the specific CubeSat mission objective in terms of accuracy, ease of interface and availability. Next, the sun sensor, as described above, was developed using photodiodes and, therefore, a comprehensive review of the operating principles of photodiodes and how they can be incorporated into CubeSats was carried out. For example, existing teams have used a variety of configurations in terms of placement and number used. Although the use of photodiode as a sun sensor presents a higher difficulty from the point of view of implementation, for example, all literature reviews

describe the accuracy as relatively low compared to an off-the-shelf, space ready coarse sun sensor [2]. However, these inaccuracies are the result of incorrect placement and the chosen photodiode.

### **3.0 Design Criteria, Constraints, and Deliverables**

#### **3.1 Constraints (*GM*)**

Regarding the constraints of the ADCS subdivision, these constraints were provided by the CSDC [3]. These constraints include were divided into five sections including: co-ordinate system for selfie-cam, attitude extent for selfie-cam operations, attitude control, attitude determination fault-tolerance, attitude control fault tolerance. The original plan for the ADCS subdivision was to maintain all set constraints in mind, though due to COVID-19 lockdown precautions, not all were considered or tested as the CSDC was no longer in consideration in relation to the deliverables. Though some points within the set-up constraints were maintained throughout the design phase and within the deliverables. A further breakdown of the specific design constraints are highlighted below in the following subsections.

#### **3.2 Reaction Wheels (*MA*)**

The main design criteria that was used for the design of the reaction wheels was to meet the required slew time in order to rotate each wheel in a 90-degree rotation about each axis. For the WinSAT 3UCubeSat, the required slew time was required that it was under one minute. With this, the reaction wheel was able to produce enough torque to quickly rotate the satellite when required. For instance, when the satellite reaches the detumbling mode, there needs to be a quick torque produced in order to quickly set the satellite with a controllable speed when in orbit. In the research paper, [2] and [4]-[6], there existed various formulas to follow in order to design the reaction wheels, mainly selecting the required materials, the torques produced and the momentum storage that was required in order to move the wheels. The debate of designing the reaction wheels included whether to include a ring with a disk design or to just make it one solid wheel. From the research and tests, the ring with a disk design was the most optimal because it required less momentum due to less material as a result of the hollow ring. Furthermore, by using the material with less inertia for the design, the result was that there was also less momentum required, which in turn created more torque. In order to optimize and finalize the reaction wheel design, the type of motor to be selected was crucial. The type of motor that had the highest rpm was the most optimal because it produced enough torque at a quick time. In order to select the correct motor, the amount of momentum stored was needed to be calculated for each slew time.



Once calculated, the result was that we can obtain a slew time of one minute by selecting a motor that had an rpm of 5000 or greater. Once our foundational rpm criteria were selected, the motor was then selected. The final design of the reaction wheels along with the selection of the motor were achieved by constructing a Python code in order to determine which combination of the three was most optimal: material and sizing, required torques, and required momentum storages. Therefore, based on the code results, the constraints that limited the design was the type of material to be used. In other words, the material with the motor selected together did not produce a passing design with respect to the momentum storage that was required by the satellite. The final deliverable of the reaction wheels was the preliminary SOLIDWORKS design that included the four reaction wheels placed onto the PCB board with their corresponding dimensions and angles. Due to global circumstances, this was the final deliverable for the reaction wheel design as the entire project had moved to simulation based and no physical models were constructed. However, because this was the case, the team was able to select the optimal material because it was only simulation based on STK. By doing this, the team was able to achieve the required momentum storage that was needed by the satellite.

### **3.3 Magnetorquers (*GM*)**

Regarding the CDSC, the magnetorquer design had the single constraint of requiring an auxiliary actuator in order to provide an alternative in the instance that one of the magnetorquers were to fail and be dysfunctional while in its orbit. This redundancy acts as a safety net to ensure that 3-axis control (roll, pitch, and yaw) will be maintained. For the magnetorquer design criteria, two metal core magnetorquers are positioned on the roll and pitch axis, and an air-core magnetorquer is placed on the yaw axis. In order to work, at any given moment, two of the three magnetorquers will be in effect. If the satellite is to reposition along any one of the three axes, the two not along the axis of rotation will be activated and produce a magnetic dipole moment which will interact with the Earth's magnetic field at any given position in order to reorient the CubeSat. For example, if the satellite is to rotate along the yaw axis, the magnetorquers along the pitch and roll axis will activate in order to produce the resultant rotation in the yaw axis. Beyond the specifications stated by the CDSC, the design of the magnetorquer was limited to fit within the limits of the PCB designed by the structural team. The preliminary designs can be seen in Figures 6-8.

### **3.4 Sensors (*HG*)**

As aforementioned, there are three sensors involved in the design of a 3UCubeSat, the magnetometer, the gyroscope and the sun sensor. The first two sensors were researched and bought off

the shelf in accordance with their flight heritage, accuracy and ease of interface. Flight heritage was an important aspect, as it provided the team with sensors that had completed the same mission as the WinSAT team. This indicates that the sensor can perform space missions and could be integrated with the CubeSat teams. As you know, the relative accuracy of the measurements with any sensor is very important in order to meet the specific mission objectives. For example, angular velocity, orientation and magnetic field strength measurements are very critical when determining the attitude determination of the satellite. Ultimately, the ease of interface was a criterion that the team focused on when selecting the gyroscope and magnetometer as the readings input from these sensors would go to the Command and Data Handling (CDH) team and would integrate the results and output them to the controller. Keeping in mind the design criteria mentioned above, the team selected the Adafruit Precision NXP 9-DOF Breakout Board-FXOS8700 + FXAS21002, which consists of a combination of a 3-axis magnetometer and a 3-axis gyroscope. This sensor provides a zero-rate level which is very important for orientation, as it represents the amount of angular velocity a gyroscope will report when the device is immobile [7]. This sensor provides a zero-rate level that is very important for orientation, as it represents the angular velocity of the gyroscope when the device is immobile. High zero-rate levels result in error in orientation determination and the distinction between zero-rate errors and angular velocity becomes non-trivial [7]. Overall, the selection of the gyroscope and magnetometer was limited to meeting the competition requirements and price constraints.

Next, the design of the sun sensor through the use of photodiodes involved various design selection criteria for design optimization, some of which included; the type of photodiode, the number of photodiodes per face and the orientation of the photodiode to produce the highest voltage readings for the microcontroller. The various types of photodiodes under consideration for the design included point and planar. The two types presented their own characteristics, such as point photodiode, which can be mounted on the CubeSat side, and soldered, but each photodiode produces four different outputs. Although this was a viable option, however, the four outputs per photodiode would result in 24 different outputs (1 photodiode per face) which were not appropriate and therefore the team decided to design the sun sensor using planar photodiodes. In addition to the type of photodiode, the number of photodiodes per face was analyzed by the following criteria, either one or two. The complexity of integration increases as the number of photodiodes increases, as shown in the following figures, showing the different equations for calculations [8].

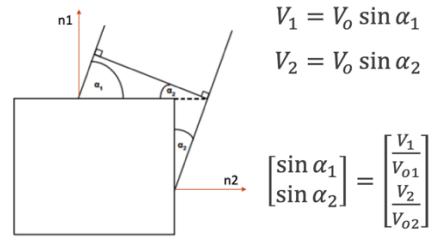


Figure 2: Equations for photodiode configuration for one photodiode per face [8]

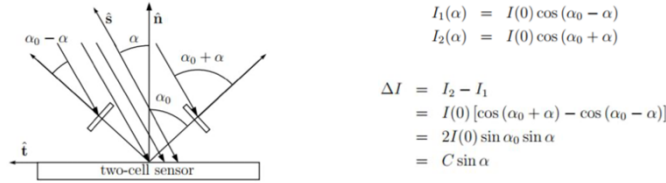


Figure 3: Equations for photodiode configuration for two photodiodes per face [8]

The logical approach used for selection resulted in the selection of a single photodiode (planar) per face as it reduced the complexity of the equations and also reduced the number of outputs to one per photodiode. As a result, the team purchased the SLCD-61N8 photodiode, which is a small silicone chip, with the bottom metallized and connected to the diode cathode. The upper side is the active area where the light-sensitive diode junction is located [9]. As the sun rays hit the light sensitive area, a current is generated and then sent to the microcontroller for readings, however, the initial current is too small to read and therefore the transimpedance amplifier (current to voltage converter) has been simulated to increase this minimum current to a voltage reading of approximately 5V. Therefore, by designing the sun sensor using planar photodiodes, the team was able to achieve the vector determination in order to use it as an input for an STK reading. The result outputted was a reading too low to be used and therefore the transimpedance amplifier was designed in order to obtain a voltage of 5V that can be used. The design of the sun sensor model in LTspice is made up of the photodiode configuration along with the amplifier circuit which consists of the op amp, resistor and capacitor. By utilizing LTspice, the team was able to conduct a result showing that the sun sensor can produce 5V which is required by the Electrical Power Systems (EPS) team.

### 3.5 Controls (KN)

The ADCS controller developed was required to maintain and switch between three different control states. These are Detumbling, Nadir Pointing and Target Pointing. In order to achieve these tasks, an appropriate model of the satellite's attitude dynamics and kinematics needed to be developed.

Since these models are very well studied and well known, the design criteria for the satellites model were very general. The equations of motion along with the dynamic equations are similar from CubeSat to CubeSat. Although these mathematical equations are very similar for every CubeSat. The appropriate changes were made to model our satellite, winSAT1. These changes were mainly the following criteria:

- The moment of inertia of the spacecraft in the body frame
- The 3x3 reaction wheel distribution matrix of the spacecraft

When developing the satellites controller, many different considerations were taken. For the detumbling controller, the design criteria were to be able to reduce the satellites angular velocity from a high angular rate, to a near zero angular rate in around 1-1.5 orbits. In order to achieve this, two different detumbling controllers were being compared for magnetic detumbling. The angular velocity feedback controller and the B-dot control law. The B-dot control law was then chosen due to its robustness and faster convergence rate to zero. These results were based on a report which implemented the two controls laws on a CubeSat with properties similar to winSAT1 and compared them [10]. An appropriate control gain needed to be determined in order to converge the angular rates to zero within the required time period. The initial control gain was determined using information gathered from previous research papers and then a trial and error method was used to find the gain required for the winSAT1 CubeSat to properly converge to zero. Likewise, there were multiples controllers studied to implement the Nadir Pointing and Target Pointing control stages. The main two were the Linear-Quadratic-Regulator (LQR) feedback controller and the Sliding mode controller. The final decision was made based on the settling time criteria of each controller. Through rigorous research, it was found that for a Low-Earth-Orbit (LEO) satellite such as winSAT1, the sliding mode controller converged to the desired Euler angle orientation. Moreover, it was found that the sliding mode controller also performs better when one of the reaction wheels loses functionality. With the LQR feedback controller, once a reaction wheel fails, large oscillations are observed in the final desired Euler angle orientation output. [11]. The sliding mode controller is used for both the Nadir Pointing and Target Pointing control stages. In this work, the sliding mode controller implementation was provided to us by Dr. Rahimi who was helping us with the project. For attitude determination, various methods were studied in order to determine the best suited algorithm for our purposes. These were mainly the TRIAD, Q-method and QUEST. After careful research, it was determined that the Q-method and QUEST delivered the most accurate results. However, the TRIAD method was finally used to determine the orientation of our satellite due to its simple algorithm. This algorithm required data of two inertial sensors as input. These sensors were readily available to us thus we chose this method.

## 4.0 Design Methodology

### 4.1 Reaction Wheels (*MA*)

As mentioned before, to design the reaction wheels, the required slew time had to be under one minute in order to produce enough torque to rotate in each respective axis. The design methodology for the reaction wheels commenced with two months of intensive research in order to comprehend what was required from the design. The main components that stood out from each research paper included the material and size of the reaction wheels, the required torque that was needed by the satellite, the required momentum storage that each wheel was able to hold, the disturbance torques, the motor selection, and the accuracy to which speed can be controlled. After the research was completed, the important formulas were noted and used to determine each parameter. These parameters included the total mass, the inertia of the disk, the inertia of the ring, the total inertia of the wheel, the total angular rotation time, the angular momentum of the reaction wheel, and final torque of the reaction wheel. By using a Python script in Appendix [A], this was all simply constructed and determined based on the initial input values that were given. From here, a comparison between the required parameters vs the actual parameters outputted was done in order to meet a certain safety factor. The required parameters of torque and momentum storage were given by the competition, and the actual parameters were determined based on research and student design. The team developed various scenarios with different combinations of the three parameters which included the sizing and material, the motor rpm, and the slew time. By adjusting every component, the team was able to test and observe each result and were to determine the most optimal design. However, the furthest that the team has achieved with the reaction wheel design was the preliminary SOLIDWORKS drawing that was developed by the structural team based on given parameters from the ADCS team. As a result, there was no actual physical model that was built simply due to the global circumstance which therefore resulted with only a simulation-based reaction wheel shown in the outputs of both MATLAB and STK. However, because the reaction wheel had gone fully simulation-based, there was the opportunity to explore even more optimal designs because the sizing, material and the rpm of the motor had limitless values that could be chosen. The team had chosen the most optimal design to use in STK simulations in order to produce the best sizing and material to use, and the most optimal motor rpm to produce the required torque and momentum storage.

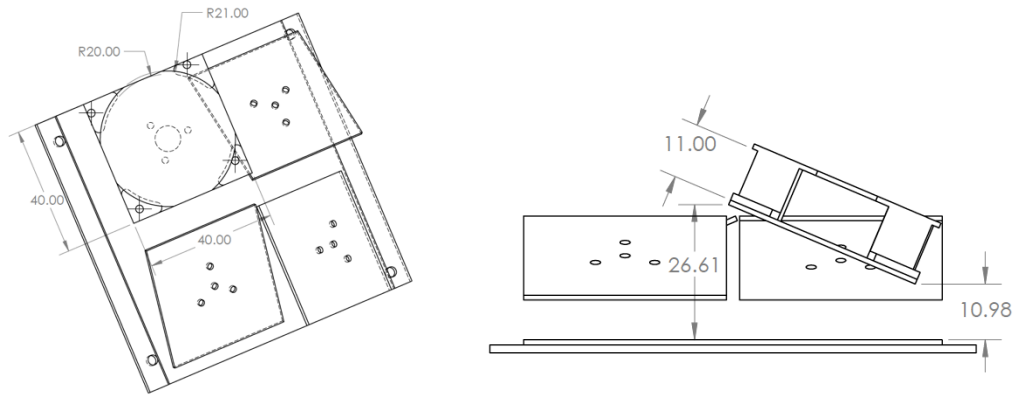


Figure 4: Preliminary SOLIDWORKS design of reaction wheel placements onto the PCB board

## 4.2 Magnetorquers (*GM*)

Before beginning any of the steps towards creating an acceptable design of the magnetorquers that would be on board the CubeSat, intensive literature review was done to further understand the purpose and role that the magnetorquers would serve. It was noted the goal of these actuators was to firstly aide in the detumbling of the CubeSat after it has been released from its launch vessel, once a stable speed is achieved for the CubeSat that we can control the secondary goal of the magnetorquers is set to begin. This secondary goal is to desaturate the work done by the reaction wheels. Seeing that magnetorquers are a very low power consuming actuator, this makes them an ideal solution to the problem. In terms of the major design criteria that was set out for the design of the magnetorquers, the magnetic dipole moment that would be generated from the two magnetorquer rods and the single air-core magnetorquer. In order for the magnetorquers to provide the required magnetic dipole moment for 3-axis control, a generated dipole moment of  $0.2 \text{ Am}^2$ , where A is the current in amperes and m is metres, would provide sufficient results in order for the magnetorquers to provide the attitude control required by the CubeSat [12]. To help realize the optimal design that would provide acceptable results, a Python script was generated. This Python script would be utilized to optimize the design so that the size of the magnetorquers and air-core magnetorquers could be minimized, in terms of core material length and radius as well as wire length and radius, and the generated magnetic dipole moment would be maximized. This design choice would provide a small amount of space taken up on its PCB and allow space for other components that require more space in other subdivisions of the WinSAT CubeSat team. The Python script that was generated can be seen in Appendix [B] along with the graphs produced by the Python script in Appendix [C] and the script output is included below.

```

Status: Pass - G: 32 AWG, M: 0.338879292061 Am^2, Mass: 0.080167868559 kg, Power: 0.213861038781 W, r_c: 4.44590232832 mm, l_c: 37.5552996543 mm, N: 8639.4646306
Status: Fail - G: 33 AWG, M: 0.367853732931 Am^2, Mass: 0.0799999992349 kg, Power: 0.20000000014 W, r_c: 6.52021534329 mm, l_c: 51.9949889297 mm, N: 5000.00252176
Status: Fail - G: 34 AWG, M: 0.463129388292 Am^2, Mass: 0.0573813332215 kg, Power: 0.199999999665 W, r_c: 5.1597004631 mm, l_c: 69.9999999961 mm, N: 5000.00011626
Status: Fail - G: 35 AWG, M: 0.40695164571 Am^2, Mass: 0.0363595295787 kg, Power: 0.200000070716 W, r_c: 4.10722280024 mm, l_c: 69.999999983 mm, N: 5000.00010395
Status: Fail - G: 36 AWG, M: 0.360825166994 Am^2, Mass: 0.0229079357053 kg, Power: 0.199999999985 W, r_c: 3.26010920367 mm, l_c: 69.9999999994 mm, N: 5000.00017598
Status: Fail - G: 37 AWG, M: 0.321100276066 Am^2, Mass: 0.0142028933281 kg, Power: 0.200000027188 W, r_c: 2.56700518956 mm, l_c: 70.0000000003 mm, N: 5000.01884032
Status: Fail - G: 38 AWG, M: 0.418056663643 Am^2, Mass: 0.00330555237671 kg, Power: 0.410722433548 W, r_c: 1.000000000058 mm, l_c: 69.9999999999 mm, N: 5000.0
Status: Fail - G: 39 AWG, M: 0.258723958743 Am^2, Mass: 0.00563714856388 kg, Power: 0.200000039641 W, r_c: 1.6172191929 mm, l_c: 70.0 mm, N: 5000.00021513
Status: Fail - G: 40 AWG, M: 0.231580793042 Am^2, Mass: 0.00355073719439 kg, Power: 0.199999999993 W, r_c: 1.28350758198 mm, l_c: 70.0 mm, N: 5000.0000921
mt_design.py:66: RuntimeWarning: invalid value encountered in log
  4*(log(lc/rc)-1.)/((lc/rc)**2 - 4*log(lc/rc))
mt_design.py:70: RuntimeWarning: invalid value encountered in log
  return 4*(log(lc/rc)-1.)/((lc/rc)**2 - 4*log(lc/rc))
Status: Pass - G: 20 AWG, M: 7.83588399006 Am^2, Mass: 0.626194020405 kg, Power: 6.18304038557 W, r_c: 2.14831708561 mm, l_c: 39.9858275624 mm, N: 10002.8151733
Status: Pass - G: 21 AWG, M: 800.859827352 Am^2, Mass: -340.017733648 kg, Power: -0.00386661864041 W, r_c: 75.575152521 mm, l_c: 3306.52919548 mm, N: -360604.573898
Status: Pass - G: 22 AWG, M: 3.64187012886 Am^2, Mass: 0.462710099248 kg, Power: 3.40214405662 W, r_c: 4.91186901432 mm, l_c: 39.9982088159 mm, N: 5000.11461112
Status: Pass - G: 23 AWG, M: 3.39263486816 Am^2, Mass: 0.434153266162 kg, Power: 2.23673596319 W, r_c: 2.96326335213 mm, l_c: 39.9997540551 mm, N: 9999.9933119
Status: Pass - G: 24 AWG, M: 3.07838627448 Am^2, Mass: 0.253120439209 kg, Power: 2.40919225889 W, r_c: 2.18109622135 mm, l_c: 40.0000024822 mm, N: 10000.0000093
Status: Pass - G: 25 AWG, M: 305.143534097 Am^2, Mass: -52.9090834989 kg, Power: -0.00398746179574 W, r_c: 33.5887441519 mm, l_c: 2464.39138702 mm, N: -311260.611016
Status: Pass - G: 26 AWG, M: 1.94966786065 Am^2, Mass: 0.158101500641 kg, Power: 1.53696908809 W, r_c: 2.15119210469 mm, l_c: 39.9999932296 mm, N: 9999.99946928
Status: Pass - G: 27 AWG, M: 1.52761731989 Am^2, Mass: 0.131170635476 kg, Power: 1.16932728344 W, r_c: 2.19080420004 mm, l_c: 39.9183285546 mm, N: 10230.9164103
Status: Pass - G: 28 AWG, M: 1.20520647835 Am^2, Mass: 0.10432531846 kg, Power: 0.931826875249 W, r_c: 2.23109048806 mm, l_c: 40.0000140827 mm, N: 10001.4056977
Status: Pass - G: 29 AWG, M: 0.0236829173355 Am^2, Mass: 0.104476101247 kg, Power: 0.571176258538 W, r_c: 5.76878588383 mm, l_c: 1.30116494366 mm, N: 5001.59938659
Status: Pass - G: 30 AWG, M: 0.700315290071 Am^2, Mass: 0.0800042670504 kg, Power: 0.492251189664 W, r_c: 2.65428197989 mm, l_c: 39.9996149397 mm, N: 10000.2853278
Status: Pass - G: 31 AWG, M: 0.505671380264 Am^2, Mass: 0.0829592971382 kg, Power: 0.308591237178 W, r_c: 3.36067270835 mm, l_c: 39.999985257 mm, N: 9999.99996613

```

Figure 5: Magnetorquer design Python script output

The design choice had been selected by filtering through the “Pass” cases that had been outputted by the code. Out of the “Pass” results, the decision was further refined to material accessibility, the easier the materials were to obtain the better the chances of the design specification to be selected. Prior to generating the Python code, the metal core material had already been selected to be Manganese-zinc ferrite (MnZn), from here the only other material left to decide was the gauge of the copper wire that would be wrapped around the ferrite core. Having access to 32 AWG (American Wire Gauge) since it was present on campus simplified the selection process further. From the figure above, the first design scenario was selected to proceed to a physical implementation. However, as mentioned prior, due to global circumstances, the physical models of the magnetorquers and air-core magnetorquer were to be left out as a completely simulated model of our design would be the only manageable deliverable. Though before the decision for global lockdown had been decided, with the assistance of the structural subdivision of the WinSAT CubeSat team, a preliminary digital design of the magnetorquer rods and air-core magnetorquer had been developed which can be observed below via SOLIDWORKS.

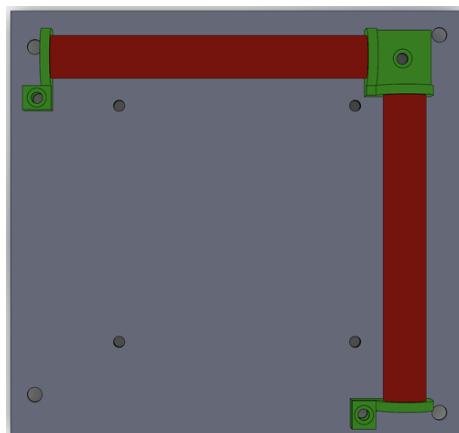


Figure 6: Top view of magnetorquer PCB showing the two metal core magnetorquers

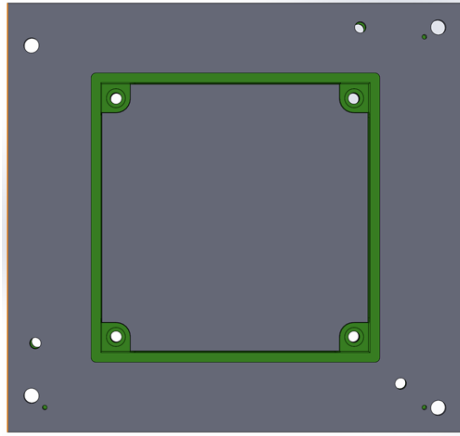


Figure 7: Bottom view of magnetorquer PCB showing the air-core magnetorquer frame

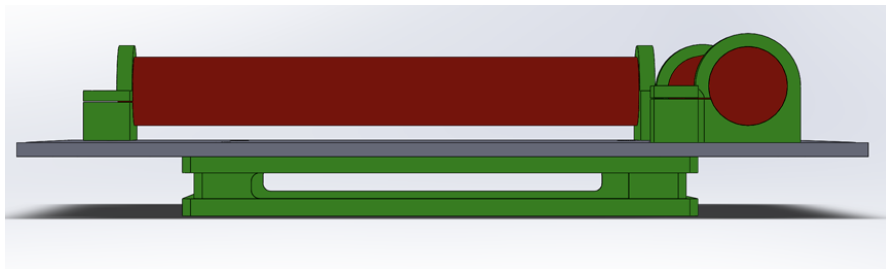


Figure 8: Side view of magnetorquer PCB

### 4.3 Sensors (*HG*)

As aforementioned, the sun sensor was designed using photodiodes and simulated in the LTspice. The initial steps taken were to choose the correct photodiode for the CubeSat mission, the SLCD-61N8 (planar photodiode). As the sun rays hit the photodiode, a low current is generated which are not read by the microcontroller. The transimpedance amplifier was therefore designed using a photodiode configuration, an op-amp, a resistor and a capacitor. The initial configuration of the photodiode consisted of the following parameters: Reverse Voltage ( $V_R$ ), Capacitance ( $C_D$ ), Shunt Resistance ( $R_{SH}$ ) and Peak Current ( $I_P$ ). Reverse voltage and capacitance values were recorded from the data sheet and found to be 20V and 100pF respectively [13].

However, the shunt resistance and peak current were calculated based on the following equation:

$$R_{SH} = V_{oc} / I_{oc}$$

After calculation the  $R_{SH}$  was calculated to be  $2.95e-6\Omega$ . Furthermore, the resistor and capacitor values were altered accordingly to produce an output voltage of 5V. The final resistor and capacitor values chosen were  $1e-6\Omega$  and  $1.01e-7F$  respectively. After conducting the simulation, the voltage was



amplified to approximately 5V as required. Below are two diagrams, one that shows the initial circuitry and the other that shows the final simulation results.

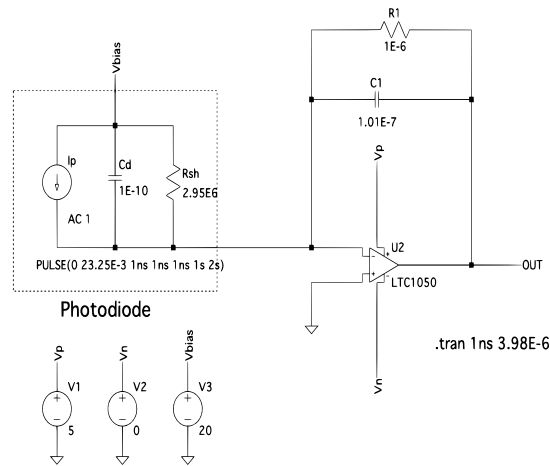


Figure 9: LTspice model for photodiode configuration and transimpedance amplifier



Figure 10: Simulation result displaying amplification of signal to 5V

Sun sensor readings are important for determining the orientation of the CubeSats with respect to the body frame, and therefore a MATLAB code was used to determine the position of the vector, which was then input to the controller for Nadir Pointing. As stated above, the current ( $I$ ) produced by the photodiode is correlated with the angle of the sun's rays being hit. The function to determine this correlation between the current angle of contact per CubeSat face is described by the sine functions shown below:

$$I_{left} = I_{max} \sin\phi \cos\theta$$

$$I_{front} = I_{max} \cos\phi \cos\theta$$

$$I_{top} = I_{max} \sin\theta$$

Figure 11: Current (I) equations for each face of the CubeSat [14]

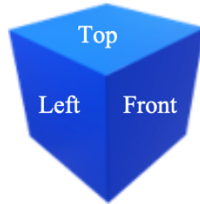


Figure 12: CubeSat Face Orientation [14]

In the equations shown above, there are two angles of contact between sun rays and photodiode, the Azimuth angle ( $\phi$ ) and the Elevation angle ( $\theta$ ). The Azimuth angle indicates the direction of the face, while the Elevation indicates how far you look in the sky. The combination of the two angles, Azimuth and Elevation, is therefore a measure to identify the position of a satellite flying overhead [15]. By measuring the current produced by the three photodiodes (maximum) that share the apex, the sun vector (single unit vector) was calculated using the following matrix:

$$v_S^B = \begin{bmatrix} X & 0 & 0 \\ 0 & Y & 0 \\ 0 & 0 & Z \end{bmatrix} \cdot \begin{bmatrix} I_{left} \\ I_{front} \\ I_{top} \end{bmatrix} \cdot \frac{1}{I_{max}}$$

Figure 13: Sun vector in 3D based on the current measurements [14]

Elevation angles were changed from 0 to 360<sup>0</sup> and current measurements per face matrix were simulated and plotted using MATLAB. You can see the MATLAB script in Appendix [D]. As the orientation of the satellite changes, the elevation angles per face increase and decrease with respect to time, and this correlation was seen in the MATLAB plot below.

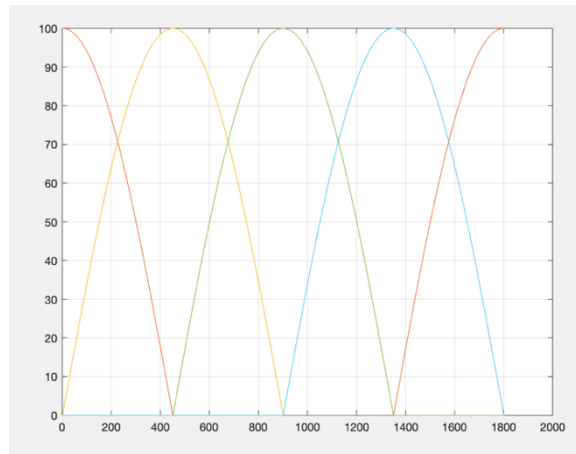


Figure 14: Change in current per face after elevation angle iterations between 0 to 360<sup>0</sup>

#### 4.4 Controls (*KN*)

The initial step taken to design a fully functional controller was to properly define the reference frames used in our satellite model. There are five major reference/coordinate frames included in winSAT1's model. These coordinate frames are:

##### ECI: Earth-Centered-Inertial Frame

This is a non-rotational frame where the origin is found at the center of earth. The Z-axis of this frame is found in the same direction as earth rotational axis, the X-axis is found along the vernal equinox and the Y-axis completes the orthogonal frame.

##### ECEF: Earth-Centered Fixed Frame

This frame is very similar to the ECI frame. However, in this case, the X-axis is pointed towards the 0-degree latitude and 0-degree longitude on earth's surface. This frame is not inertial, as it rotates with the earth along the Z-axis.

##### Orbit Frame

This frame has its origin located at the center of mass (COM) of the winSAT1 satellite. Thus, making it a non-inertial frame. The Z-axis of this frame is always pointed towards the earth's center. The Y-axis is in direction opposite to the orbit normal vector. This orientation is known as Nadir Pointing.

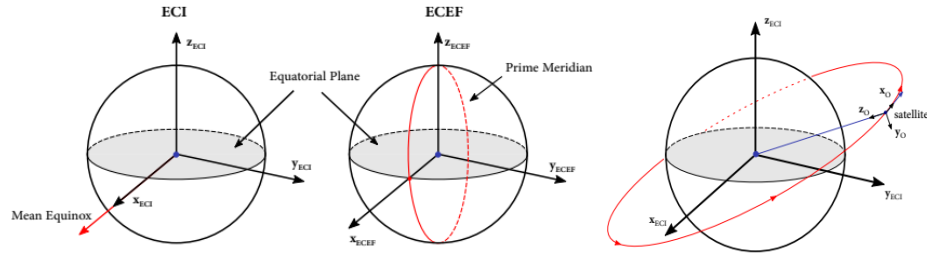


Figure 15: ECI, ECEF, and Orbit Frame [16]

### Body Frame

The body frame has its center the same as the orbit frame. Although, this coordinate system rotates with the satellite. The X, Y and Z axes are chosen in a manner where they coincide with the satellite's principle inertia axes.

### Target Frame

This frame is very similar to the orbit frame. However, the Z-axis of this frame is aligned with the target coordinate that is supplied to the satellite's controller. Target Pointing is defined within this reference frame.

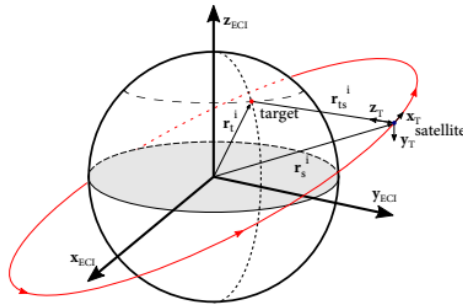


Figure 16: Target Frame [16]

Once the reference frames were properly defined. Rotation matrices which defined a vector in one frame to the other were developed in software. The second step was to develop the satellite's dynamic and kinematic models. A satellite orbiting earth can be modelled as a rigid body with an N-wheel reaction wheel cluster. Through review of [17], the following 3 dynamic equations which describe winSAT1 are:

$$\dot{h}_B = h_B \times J^{-1} (h_B - Ah_w) + \tau_d + \tau_w$$

Where  $h_B$  is the angular momentum vector of winSAT1 in the body reference frame given by:

$$h_B = J\omega_B + Ah_w$$

$H_w$  is the  $N \times 1$  vector of the axial angular momentum of the four reaction wheels.  $W_b$  is the angular velocity of winSAT1 in the body frame. This is provided by the gyroscope sensor.  $T_d$  and  $T_w$  are the disturbance torques and the reaction wheel torques respectively.  $A$  is the  $3 \times N$  reaction wheel distribution matrix,  $J$  is this moment of inertia of the spacecraft and reactions wheels given by:

$$J = J_s - AJ_w A^T$$

The satellites kinematic model will provide information between the angular speed and the rate of change of the attitude. In this work, the attitude will always be expressed in terms of quaternions and Euler angles. Through review of [17], the following 2 equations model the satellite kinematics:

$$\dot{\eta} = -\frac{1}{2}\epsilon^T \omega_{ib}^b \quad \dot{\epsilon} = \frac{1}{2}(\eta I_{3 \times 3} + S(\epsilon))\omega_{ib}^b$$

The kinematic and dynamic equations were then modelled in MATLAB software. Once the satellite model was developed. The third step was to develop the attitude determination model. This model allows for accurate determination of winSAT1's orientation and velocity throughout its orbit. To determine the satellite velocity in the ECI and ECEF frames, a well-known algorithm named simplified perturbation model 4 or SGP4 was used. The input to this algorithm is a set of orbit parameters known as Keplerian elements along with a Two-Line Element or TLE file which was generated from our STK satellite model. To determine the satellites orientation/attitude, the TRIAD algorithm was used. This algorithm takes in two sensor data readings, mainly the sun sensor and the magnetometer data in the body frame from the on board IMU. The algorithm also takes the same type of data, however this time from an inertial frame stemming from mathematical models of those same sensors. Once this data is passed into the algorithm, the current orientation of our satellite is known. The fourth and final step was to develop the control models which allow winSAT1 to navigate through the previously mentioned control stages. The first controller developed was the detumbling controller. Here the B-Dot control law was developed in MATLAB and its output was piped into STK for simulation and visualization of the algorithm working on our satellite. Through review of [16], the following is the B-dot control law which was modified for computation in MATLAB:

$$\mathbf{m} = -k \frac{1}{\|\mathbf{B}\|^2} \dot{\mathbf{B}}$$

Here,  $\mathbf{m}$  is the magnetic moment generated by the magnetorquer once a current is passed through the coils. This magnetic moment is generated in a direction opposite to that of the earth's magnetic field vectors. In order to slow down the angular velocity of the satellite, a torque is generated by crossing the magnetic moment with the magnetic field of earth. This produces a torque that acts in the direction opposite to that of the spin of the satellite. The follow is the torque generated:

$$\boldsymbol{\tau}_m^b = \mathbf{m}^b \times \mathbf{B}^b$$

It is assumed that for relatively high angular speeds, the rate of change of the earth's magnetic field can be defined as:

$$\dot{\mathbf{B}}^b \approx \mathbf{B}^b \times \boldsymbol{\omega}_{ib}^b$$

The output generated from STK for the detumbling controller is as follows:

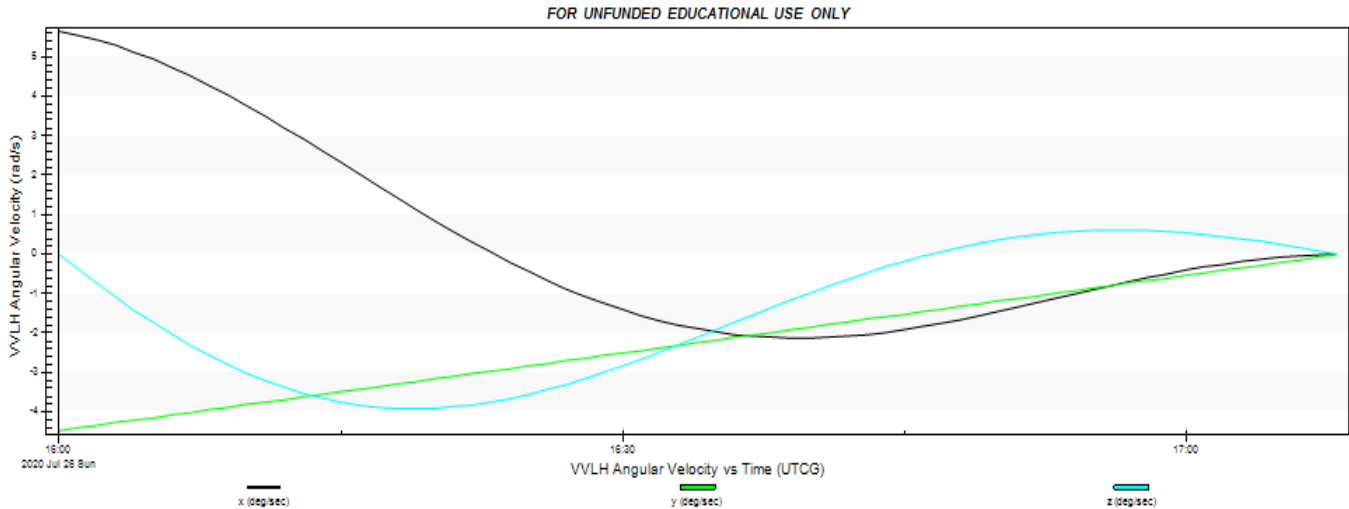


Figure 17: Detumbling algorithm converging the satellites angular velocity to near zero

In this figure, we can see that we have initialized the angular velocity of our satellite in the X, Y and Z axis respectively at specified rates. As time passes, the detumbling controller is producing a control torque continuously at every time step. With every time step, that output torque is then being passed back into STK and this torque is being applied to our satellite. It is important to note that the angular velocity being passed into the B-dot controller is being passed from STK on board gyro sensor to MATLAB. Thus, STK and MATLAB are working together to get the algorithm working. It is clear to see that as time passes, the angular velocity in every axis converge to zero. This confirms that the detumbling controller is working properly.

The Sliding mode controller was then developed in order to stabilize the attitude of our spacecraft. The following equations were obtained from the review of [17]. We must first define the linear sliding surface variable:

$$s = \boldsymbol{\omega}_B + \lambda \mathbf{q}_v$$

The feedback control law chosen is as follows:

$$\tau_{w-des} = -A^{\dagger} \left[ k_1 + (\rho + \eta) \frac{1}{\|s\| + \delta} \right] s \quad \rho = \sigma [ p_1 \|\omega\| + p_2 \|q\| + p_3 ]$$

Where k, P1, P2, P3 and sigma are positive constants. This equation describes the control torque that must be generated by the reaction wheels for a given desired attitude. The sigma value is determined as follows:

$$\dot{\sigma} = b_1 [ p_1 \|\omega\| + p_2 \|q\| + p_3 ] \frac{\|s\|^2}{\|s\| + \delta} - b_2 \sigma$$

Once we know the desired torque requirement, we must generate a corresponding voltage to the reaction wheel motors, the voltage needed is given by:

$$V = R_a K_m^{-1} J_w^{-1} \tau_{w-des} + K_m \omega_w$$

Where V is the output voltage, Ra is the coil resistance and Km is the motor torque constant matrix for each wheel. This control law was kindly provided to us by Dr. Rahimi to use for our simulations.

Below is the flow of MATLAB scripts which can be found in Appendix [F] – [K]. The software structure is as follows:

- test\_sgp4.m file was used to determine the velocity of the satellite at any point in its orbit.
- Winsat\_DetumbleTorque1.m file was used to integrate MATLAB with STK and run the detumbling algorithm.
- TRIAD.m was used to run the TRIAD algorithm.
- Tle.txt is the text file TLE used as input to test\_sgp4.m
- magFieldModelIECI.m was used as the inertial mathematical model for the earth's magnetic field and used as input to TRIAD.m
- SunSensorModel.m was a was used as the inertial mathematical model for the sun vector direction and used as input to TRIAD.m
- N\_CSA\_Script.m was used to initialize the initial parameters for Nadir Pointing and Target Pointing.
- N\_CSA\_Model.m was used to develop the satellite dynamic and kinematic model. Along with the Sliding mode controller.

- N\_CSA\_Model.mdl was used as the Simulink model which was used to simulate the model in MATLAB.
- N\_CSA\_Plotter.m was used to plot the outputs out N\_CSA\_model.m

## **5.0 Physical Implementation/Simulation Model Validation**

### **5.1 Reaction Wheels (*MA*)**

The simulation model was implemented using the STK software, MATLAB, and Python. This was done for every component of the ADCS including the reaction wheels, the magnetorquers, the sun sensor, and the controls algorithms. For the reaction wheels, the output of the Python script outputted what had passed or failed in terms of required torque and required momentum storage. In the STK software, the output of the voltage and wheel speeds shown for both Nadir and Target Pointing provide the results that the reaction wheels are producing enough torque at the required slew time to converge the quaternion error to zero in order to have both Nadir and Target Pointing stages.

### **5.2 Magnetorquers (*GM*)**

As mentioned prior within the report, the effects of COVID-19 caused lockdown regulations to be initiated worldwide. Due to the pandemic, the physical implementation to be done of the magnetorquers and the air-core magnetorquer were left undone, as all purchased and acquired materials were left on the campus of the University of Windsor, the magnetorquers were deemed as a non-essential part of the overall design. The reaction wheels provided sufficient simulated results and were able to be integrated into the controller algorithm. For this reason, the team had made the decision to that the magnetorquers could be removed from the final design and the reaction wheels could provide the results required successfully.

### **5.3 Sensors (*HG*)**

As it is known, due to the current global conditions, the physical implementation and testing of the sensors was not feasible and therefore simulation-based tests were carried out. As mentioned above, the two sensors, the magnetometer and the gyroscope were purchased off the shelf and therefore the ADCS team did not perform any sensor testing. However, the third sensor, the sun sensor, was tested and analysed using the LTspice program to design a transimpedance amplifier to convert the initial low photodiode current readings to a voltage reading of approximately 5V. These voltage readings for the design of the Sun Sensor are very important for determining the position of the CubeSat in relation to



the body frame. Using initial current readings, a MATLAB script was generated to determine the sun vector for positioning and input to the controller. The MATLAB script, as shown in the Appendix, shows variations in the current intensities of the CubeSat face as in the rotation. The maximum current ( $I_{max}$ ) was set for simulation purposes by the team in which the elevation angles were changed from 0 to  $360^{\circ}$  and input into the formula shown in Figure 13. Once the simulation was run, the position of the sun vector was shown for the time series (set to 0.1 seconds). The corresponding results were the same as the direct values of the sun vector at that specific angle of elevation and time in the STK indicating that the correct MATLAB script was generated.

### 5.4 Controls (KN)

The following were the outputs generated from the sliding mode control model.

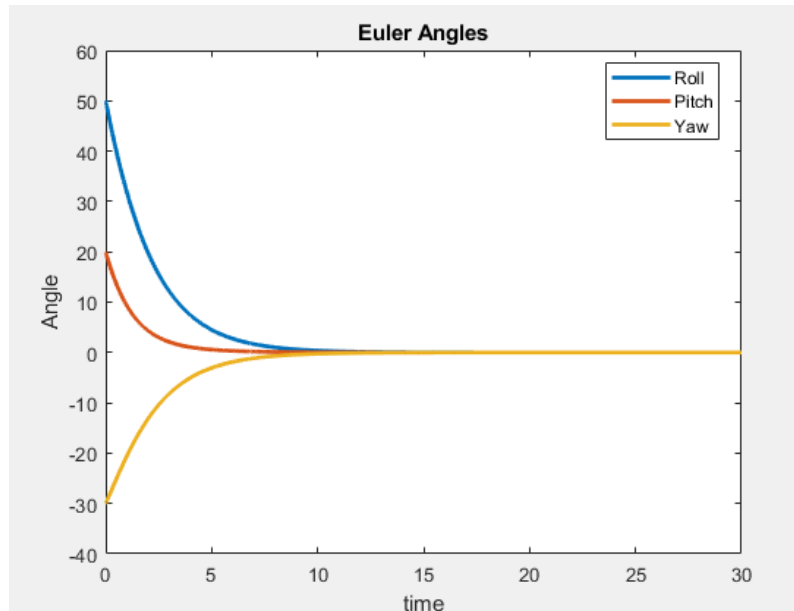


Figure 18: Euler angles converge to zero, thus pointing to Nadir

Here we can see a Euler Angle Vs Time plot for Nadir Pointing. The Euler angles are defined as Roll, Pitch and Yaw axes. It is seen that at time  $T=0$  seconds. We have a current attitude of 50 degrees Roll, 20 degrees Pitch and  $-30$  degrees Yaw. It is important to mention that to achieve Nadir Pointing. The body frame of the satellite must be aligned with the orbit frame. This alignment is defined by a 0 degree in Roll, Pitch and Yaw angles. In this plot, we can clearly see that we can converge to Nadir Pointing from any orientation in space with the sliding mode controller. The same is true for Target Pointing, the controller works the same, the only thing to change is the desired orientation.

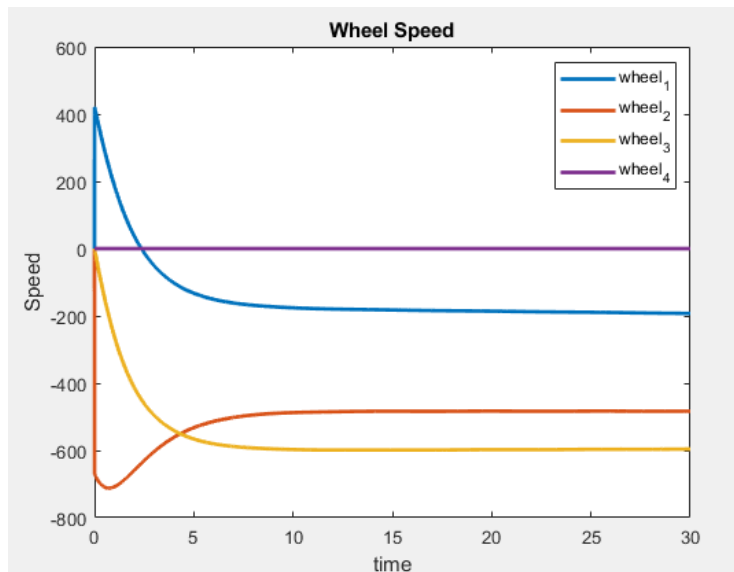


Figure 19: Wheel speed output

In this figure, the wheel speeds of every reaction are plotted against time for the Nadir Pointing case with initial Roll, Pitch and Yaw angles previously mentioned. We can see that the wheel speeds converged to a speed which is always required to be maintained to maintain Nadir Pointing. If any outside disturbance changes these wheel speeds, the Nadir Pointing orientation will diverge.

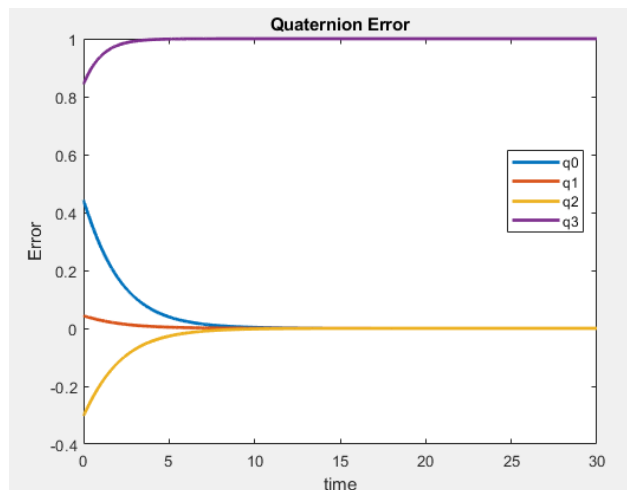


Figure 20: Quaternion error shows the change from the current quaternion to desired (converged at zero) quaternion

In this figure, the same initial conditions are the same as previously mentioned. Here we see the quaternion error plots. The quaternion error is defined as the difference between the reference quaternion/orientation and the current quaternion. It is clearly show that as time passes, this error converges to zero. Thus, supporting the result of Nadir Pointing.

## 6.0 Experimental Methods/Model Validation

### 6.1 Controls (KN)

In order to validate the design of the controller, a visualization tool called STK was utilized. STK allowed the team to import a real-life scale model of the physical CubeSat that would have been developed if not for the global circumstances. With the imported model, the trajectory and uncontrollable motion of the CubeSat resembled that of a real-life scenario. In Figure 21, the CubeSat's initial position is set at a given angular velocity of 7.2 degrees per second. This value was manually inputted into the system to provide a feasible starting point for the simulation. In Figure 22, the angular velocity has been greatly reduced to a value of 0.02 degrees per second. This reduction was achieved by importing the MATLAB script found in Appendix [F] which activates the control system's BDOT algorithm in order to detumble the CubeSat to a near zero value so that the CubeSat can be successfully manipulated and reoriented to carry out its mission.

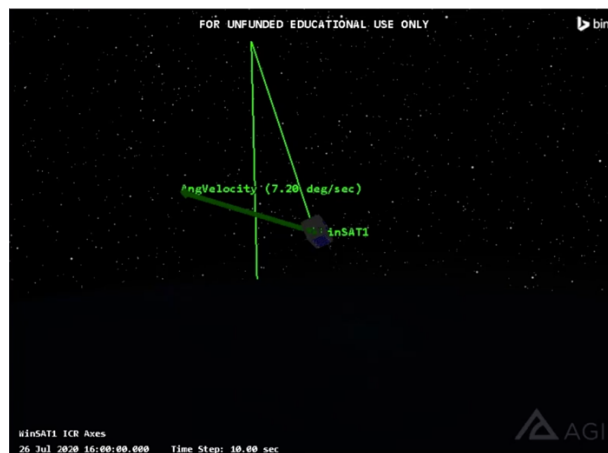


Figure 21: Starting position in STK simulation

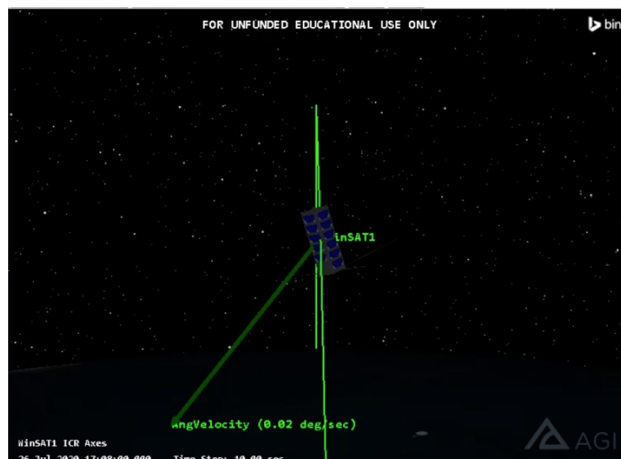


Figure 22: End position in STK simulation after BDOT algorithm execution

The time taken for the BDOT algorithm to achieve the near zero angular velocity can be seen in Figure 17, which showcases the 3 axes (Roll, Pitch, and Yaw) all converging to zero.

## **7.0 Design Specifications and Evaluation Matrix**

### **7.1 Sensors (HG)**

As discussed in sections above regarding the sensor design methodology and implementation, the sensors had proven to be designed as an accurate tool in recovering essential input data for the attitude control. The outputs of the three sensors provide the CubeSat with the ability to track its position, velocity and the orientation in its orbit. From the simulation constructed via LTspice, the photodiode transimpedance amplifier circuit was successful in producing a steady 5V. This voltage was key in being able to accurately pinpoint the position of the sun and aid with achieving Nadir Pointing. This is supported since the outputs achieved via the MATLAB script created for the sun vector determination matched the values that were achieved the successful creation of the STK simulation of the CubeSat. This supports the pointing accuracy achieved being less than or equal to 1.5 degrees which was highlighted in the constraints of the CSDC.

### **7.2 Controls (KN)**

Based on the graphs achieved for the various MATLAB scripts written for the control algorithm of the CubeSat, some observations of the design specification can be seen. In Figure 17 for the BDOT algorithm, on the basis of how long the system took to achieve convergence at zero of all 3 axes, it is determined that the algorithm will require approximately 68 minutes in order for the BDOT algorithm to have the 3 axes converge to zero and stabilize the CubeSat to a speed that we are able to manipulate so that we can achieve Nadir Pointing and Target Pointing. Nadir Pointing was achieved seeing that in Figure 18 showcasing the Euler angles all converging to a point of zero, (Roll, Pitch, Yaw) becomes (0, 0, 0). In accordance to the CSDC, this meets the standards set out in order achieve acceptable Nadir Pointing.

As well as this, since successful detumbling was achieved via BDOT algorithm. Target Pointing can also be achieved, as the same requirements have been accomplished in order to produce acceptable Nadir Pointing, which include a low controllable speed successful reading from the various sensors.

## 8.0 Budget (MA)

The initial budget was the \$200 that was distributed to each capstone member, which totaled to \$800 for the ADCS team. However, through fundraising and sponsorships, the WinSAT team was able to obtain \$20,000 which increased the budget for the ADCS team to \$2,000. Before the global situation occurred, the team was initial planning on purchasing the following components for each respective category; however, due to COVID-19, the team froze the purchases. For reaction wheels, the team was planning to purchase four motors and motor controllers. For magnetorquers, the team planned on purchasing the MnZn Ferrite cores. For sensors, the team had the initial plan on purchasing the photodiode planar chip and the NXP 9-DOF sensor.

Table 1: Motor selection and prices

| Motor   | Price         |
|---|---------------|
| EC 20 flat Ø20 mm, brushless, 5 Watt, with Hall sensors   | \$130.98 (x4) |
| Faulhaber: Series 1509B (6V)  | \$125.99 (x4) |
| EC-max 16 Ø16 mm, brushless, 5 Watt, with Hall sensors  | \$277.82 (x4) |
| CHIHAI CHF-130SA-ABHL 6V 9500rpm Gear Motor Permanent Magnet DC Hall Coded Reduction Encoder Gear Motor | \$8.54 (x4)   |
| ECX SPEED 8 M Ø8 mm, brushless, with Hall sensors   | \$318 (x4)    |

Table 2: Motor controllers and prices

| Motor Controller   | Price         |
|--|---------------|
| ESCON Module 50/5, 4-Q Servocontroller for DC/EC motors, 5/15 A, 10 - 50 VDC | \$226.59 (x4) |
| Faulhaber SC 1801 P  | \$199 (x4)    |
| Pololu Simple High-Power Motor Controller 18v15 (Fully Assembled)            | \$46.95 (x4)  |
| Digilent Pmod DHB1 Dual H-bridge Motor Controller                            | \$23.78 (x4)  |

Table 3: Magnetorquer material and price

| Magnetorquer Component | Price        |
|------------------------|--------------|
| MnZn Ferrite cores     | \$1.455 (x3) |

Table 4: Sensors components and prices

| Sensor Component   | Price       |
|--|-------------|
| Photodiode, Planar Chip, 60°, 1.7uA, 930nm, Planar-1               | \$2.91(x10) |
| Adafruit Precision NXP 9-DOF Breakout Board – FXOS8700 + FXAS21002 | \$14.95     |

NOTE: Items Selected for Purchase

As a result, if the team were to purchase the selected components, the totaled amount would be \$1,452.505 which would be in the team's updated budget. However, without the fundraisers and sponsorships, the team would be over budget and would have to select less expensive components.

## 9.0 Conclusions (GM)

Though the course of the Capstone project, much of the projected final product, results, and timeline had been offset by the unseen circumstances that COVID-19 has brought upon the world. As meeting in person to be able to work together and construct a physical model had been declared as a task that would no longer be possible, the ADCS subdivision, along with most of the other WinSAT subdivisions made the decision to move onwards with a completely simulated deliverable Capstone project. With this in mind, several aspects of the projects process of completion had changed all together. These included leaving behind the physical prototype of the ADCS system, actuators, and sensors.

Though with the changes brought onto the Capstone project, more ideal scenarios could be explored as with a completely simulated environment the limits of the real world no longer exist. However, real-world capabilities were kept in consideration throughout the construction of the completed sensor models as well as all aspects of the controller that dictates the CubeSat's movement while in its orbit. This was integral to the completion of the ADCS system as the intent is to provide a working base for future students who wish to take on the WinSAT satellite as their own Capstone project in the coming years, as this 2020 Capstone year is the first year in which this project has been attempted.

When comparing the deliverables achieved with the expected performance, the deliverables provide sufficient results and meet the expectations of both the ADCS teams demands, as well as the expectation set out by the CSDC. For the sun sensor, the single photodiode per face model was simulated and programmed successfully with a constant output voltage of 5V, Figure 10, after the current sent from the photodiode passes through the designed transimpedance amplifier, which feeds the 5V signal to the controller. Alongside this, the sun vector determination algorithm made in MATLAB had matching results with the output of STK which validated the pointing accuracy constraint set by the CSDC.

For the controller model, successful simulations and MATLAB scripts were constructed that prove the functionality of the control system in charge of dictating the CubeSat's orientation and motion while in LEO. STK provided an excellent method in which visualization of the BDOT algorithm could be observed, Figures 17, 21, and 22, and proof of its ability to work was achieved. Nadir and Target Pointing were successfully achieved as well and can be seen in Figures 18-20 as the Euler angles and

quaternion error converge at zero for the 3 axes. In accordance with the CSDC and theory studied during literature, this work is confirmed.

Unfortunately, the two actuators (reaction wheels and magnetorquers), were left incomplete due to COVID-19, the design work that had been completed had proved successful in terms of the theoretical work. The design incorporated an auxiliary reaction wheel as well as an auxiliary magnetorquer, as per the demands of the CSDC constraints which would have been utilized in the instance that some form of failure were to appear in one of the reaction wheels or magnetorquers that were set to be utilized as primary actuators.

## References

- [1] A. Farhat, J. Ivase, Y. Lu, A. Snapp, M. A. Demetriou, and M. Advisor, "Attitude Determination and Control System for CubeSat," pp. 180–184, 2013.
- [2] J. Gerber, "A 3-Axis Attitude Control System Hardware Design for a CubeSat by," no. December, p. 19, 2014.
- [3] Canadian Satellite Design Challenge Management Society, "The Canadian Satellite Design Challenge. Design , Interface , and Environmental Testing Requirements," no. 3, pp. 1–16, 2014.
- [4] E. Oland and R. Schlanbusch, "Reaction wheel design for CubeSats," *RAST 2009 - Proc. 4th Int. Conf. Recent Adv. Sp. Technol.*, no. November, pp. 778–783, 2009, doi: 10.1109/RAST.2009.5158296.
- [5] J. Rohde, "Kalman filter for attitude determination of student satellite," *Thesis*, no. July, 2007.
- [6] C. Hajiyeve and M. Bahar, "Attitude determination and control system design of the ITU-UUBF LEO1 satellite," *Acta Astronaut.*, vol. 52, no. 2–6, pp. 493–499, 2003, doi: 10.1016/S0094-5765(02)00192-3.
- [7] A. Industries, "Adafruit Precision NXP 9-DOF Breakout Board," *adafruit industries blog RSS*. [Online]. Available: <https://www.adafruit.com/product/3463>. [Accessed: 07-Aug-2020].
- [8] J. C. Springmann and J. W. Cutler, "Photodiode Placement & Algorithms for CubeSat Attitude Determination," *CubeSat Dev. Work.*, 2012.
- [9] M. Gruber, "Functional Testing of the MIST Satellite," 2018.
- [10] L. Mirko, "Comparison of Control Laws for Magnetic Detumbling Comparison of control laws for magnetic detumbling," no. OCTOBER 2012, 2015.
- [11] I. Kök, "Comparison and Analysis of Attitude Control Systems of a Satellite Using Reaction Wheel Actuators," pp. 17–18, 2012.
- [12] D. Miller, "Design optimization of the CADRE Magnetorquers," pp. 1–22, 2013.
- [13] S. P. Photodiode, "Slcd-61N8," no. 1, p. 104118.
- [14] Y. Winetraub and S. Bitan, "Attitude Determination – Advanced Sun Sensors for Pico-satellites," pp. 1–10, 2005.
- [15] "What are the "azimuth and elevation" of a satellite? | Memorial Spaceflights", *Celestis.com*, 2020. [Online]. Available: <https://www.celestis.com/resources/faq/what-are-the-azimuth-and-elevation-of-a-satellite/>. [Accessed: 07- Aug- 2020].
- [16] M. C. Santiago, "Cubesat Attitude Control System based on embedded magnetorquers in photovoltaic panels," 2018.
- [17] Godard, "CSA Report - Mathematical Model – Spacecraft Attitude Stabilization," 2009.



# Appendices

## Appendix A

```
import math as m
import numpy as np
from IPython import embed

#Constants
earthGravConst = 3.986e14 #Earth-Gravity Constant in m3/s2
earthRadius = 6378.14 #Radius of Earth in Km.
solarConst = 1367 #Solar Constant in [W/m2]
magmomentEarth = 7.96e15 #Magnetic moment of earth in [Tesla/m3]
rmsSINavg = 0.707 #Sinusoidal RMS average
speedLightVacuum = 299792458.0 #speed of light in vacuum [m/s]
safetyMargin = 1
'''
FireSat = satellite.Satellite({
    "mass" : 215,
    "inertiaZ" : 90,
    "inertiaY" : 60,
    "orbitAlt" : 700,
    "slewRate" : 30,
    "pointingAcc" : 0.1,
    "surfaceArea" : 2*1.5,
    "deltaCOGCOPsolar" : 0.3,
    "coefReflectivity" : 0.6,
    "angleIncidence" : 0,
    "residualDipole" : 1,
    "atmosDensityRho" : 1e-13,
    "dragCoefCd" : 2.0,
    "surfAreaAero" : 3,
    "satVelocity" : 7504,
    "deltaCOPCOGaero" : 0.2,
    "marginFactor" : 0,
    "slewTime" : 600,
    "orbitalPeriod" : 1482,
    "yawRollAccuracy" : 0.1
})
'''
WinSAT = {
    "mass": 3.0, #Mass [kg]
    "inertia_z": 0.014, #Moment of Inertia [Kg.m2] Ix = Iz
    "inertia_y" : 0.0037, #Moment of Inertia [Kg.m2]
    "orbital_altitude" : 408, #Alt [Km], Circular Orbit
    "slew_rate" : 0, #0.1 [deg/s]
    "pointingAccuracy" : 0.1, #[deg]
    "targetingMaxAngle": 25, #[deg]
    "surface_area" : 0.034, #Surface area cross section of [m^2, 0.340m by 0.1m]
    "deltaCOMCOP": 0.051, #Center of mass to Center of pressure difference in [m] -
based on SMAR -> Cubesat ratio
    "coef_of_reflectivity": 0.6, #Coefficient of Reflectivity
    "angle_of_incidence": 0, #Angle of incidence of the sun in [deg]
    "magnetic_dipole": 1.0, #Spacecraft magnetic dipole [A.m2]
    "atmospheric_density_rho": 1e-13, #Atmospheric density Rho [kg/m3]
    "drag_coefficient": 2.0, #Drag Coefficient usually between 2 and 2.5
    "surface_area_aero":0.034, #Surface Area in [m^2, 0.340m by 0.1m]
    "deltaCOPCOGaero": 0.051, #Center of gravity to Center of aerodynamic pressure
difference in [m]
}

```

```

#REFER TO TABLE 11.9A IN SMAD FOR LIST OF EQUATIONS
"""MAX GRAVITY TORQUE GENERATED BY GRAVITY GRADIENT DISTURBANCE"""
def gravityGradient(sat):
    orbitRadius = (earthRadius+sat['orbital_altitude'])*1000
    baseFunc = ((3*earthGravConst)/(2*((orbitRadius)**3))) * (sat['inertia_z'] -
sat['inertia_y'])
    nadirMode = baseFunc*(np.sin(np.deg2rad(2*sat['pointingAccuracy'])))
    targettingMode = baseFunc*(np.sin(np.deg2rad(2*sat['targetingMaxAngle'])))
    return max(targettingMode,nadirMode)
print ("Max torque generated by gravity gradient: " + str(gravityGradient(WinSAT)) +
" N.m")
"""MAX GRAVITY TORQUE GENERATED BY SOLAR RADIATION DISTURBANCE"""
def solarRad(sat):
    F = (solarConst/speedLightVacuum) * (sat['surface_area']) *
(1+sat['coef_of_reflectivity']) * (np.cos(np.deg2rad(sat['angle_of_incidence'])))
    return F*(sat['deltaCOMCOP'])
print ("Max torque generate by solar radiation is: " + str(solarRad(WinSAT)) + "
N.m")
"""MAX GRAVITY TORQUE GENERATED BY MAGNETIC FIELD DISTURBANCE"""
def magTorque(sat):
    orbitRadius = (earthRadius+sat['orbital_altitude'])*1000
    earthMagfield = 2*(magmomentEarth/(orbitRadius**3)) #For polar orbits-- half
this for equatorial orbit
    return earthMagfield*sat['magnetic_dipole'] #Worst case polar mag field in [N.m]
print ("Max torque generated by magnetic field is: " + str(magTorque(WinSAT)) + "
N.m")
"""MAX GRAVITY TORQUE GENERATED BY AERODYNAMIC DISTURBANCE"""
def aerodynamicDrag(sat):
    orbitalVelocity = np.sqrt(398600.5/(6378.14+sat['orbital_altitude']))
    F =
0.5*(sat['atmospheric_density_rho']*sat['drag_coefficient']*sat['surface_area_aero']
*(orbitalVelocity**2))
    return F * (sat['deltaCOMCOP'])
print ("Max torque generated by aerodynamic pressure is: " +
str(aerodynamicDrag(WinSAT)) + " N.m")
""" SIZING ADCS HARDWARE """
print("----- SIZING ADCS HARDWARE -----")
#List of disturbances
disturbanceList = [gravityGradient(WinSAT), solarRad(WinSAT), magTorque(WinSAT),
aerodynamicDrag(WinSAT)]
greatestdisturbanceTd = max(disturbanceList)
"""Calculates torque required to counter balance the effect of a disturbance"""
...
def requiredcounterTorque(sat):
    Trw = greatestdisturbanceTd*sat.margin_factor
    return Trw
print ("Counter torque required from reaction wheel due to greatest disturbance is:
" + str(requiredcounterTorque(FireSat)) + " N.m")
""" Calculates torque required to slew to a required angle """
def slewTorque(sat):
    Tslew = (4*sat.slew_rate*(np.pi/180)*sat.inertia_z)/(sat.slew_time)**2
    return Tslew
print ("Counter torque required from reaction wheel to slew is: " +
str(slewTorque(FireSat)) + " N.m")
"""Estimates the wheel momentum stored in reaction wheel for one orbital period"""
def momentumStorage(sat):
    wheelmomentumH = (greatestdisturbanceTd*sat.orbital_period*rmsSINavg)/(4)
    return wheelmomentumH
print ("Wheel momentum stored in one orbit is: " + str(momentumStorage(FireSat)) + "
N.m.s")

```

```

"""Angular momentum required for a specific Roll and Yaw with a given degree
accuracy"""
def requiredAngmomentum(sat):
    angularmomentumreq =
    (greatestdisurbanceTd*sat.orbital_period)/(sat.yaw_roll_accuracy*(np.pi/180))
    return angularmomentumreq
print ("Angular momentum required to Yaw or Roll with specific accuracy is: " +
str(requiredAngmomentum(FireSat)) + " N.m.s")
"""Magnetic torquer ability to counteract worst case disturbance during momentum
dumping"""
def magneticTorquer(sat):
    orbitRadius = (earthRadius+sat.orbital_altitude)*1000
    earthMagfield = 2*(magmomentEarth/(orbitRadius**3))
    magDipoleD = (greatestdisurbanceTd/earthMagfield) + 3
    return magDipoleD
print("the magnetic torquing ability of the magnetorquer is: "+
str(magneticTorquer(FireSat)) + " N.m.s")
'''
densityOfMaterial = 8940 #Density of Aluminum [kg/m^3] (661 T1)
radiusOfDisk = 0.015 #Radius of the disk [m]
heightOfDisk = 0.001 #Height of disk [m]
radiusOfRing = 0.019 #Radius of ring [m]
heightOfRing = 0.01 #Height of ring [m]
slewRequirement_rad = np.deg2rad(90) #[rad]
slewRequirement_time = 20.0 #[s]
MOI_satellite = np.eye(3)*[0.0140011346, 0.0137275532, 0.00369381176]
orbitalPeriod_ISS = 92.68*60. #[s]
pointingRequirement_rad = np.deg2rad(1.0) #[rad]
pyCfgAngle = np.deg2rad(67) #67 degrees in radians
motorNominalSpeedTimeConst = (30.3e-3/0.63 * (4780/9350.)) #based on EC 20 flat
datasheet, linearly relating time const to nom speed (mech time const = time to 63%
of no load speed)
singleRWSpeed = 4780*(2*np.pi/60) #rad/s

massOfRing = densityOfMaterial*np.pi*(radiusOfRing**2 -
radiusOfDisk**2)*heightOfRing #Mass of the ring [kg]
massOfDisk = densityOfMaterial*np.pi*radiusOfDisk**2 * heightOfDisk #Mass of the
disk [kg]
motorAngularVelocity = [singleRWSpeed]*4#Maxon EC 20 flat, from RPM spec to rad/sec
totalMassRW = massOfRing + massOfDisk #Total Mass of the Reaction Wheel
inertiaOfRing = massOfRing*(radiusOfRing**2 + radiusOfDisk**2)/2 #Inertia of the
Ring
inertiaOfDisk = (massOfDisk*radiusOfDisk**2)/2 #Inertia of the Disk
inertiaTotalRW = inertiaOfRing + inertiaOfDisk #Total inertia of the Reaction Wheel
matrixAw = np.array([[np.cos(pyCfgAngle), 0, -np.cos(pyCfgAngle), 0],
[0, np.cos(pyCfgAngle), 0, -np.cos(pyCfgAngle)],
[np.sin(pyCfgAngle), np.sin(pyCfgAngle), np.sin(pyCfgAngle),
np.sin(pyCfgAngle)]])
#Maximum Torque and Momentum of the Reaction Wheel
#wheelMomentum = matrixAw*(inertiaTotalRW * motorAngularVelocity);
wheelTorques = [inertiaTotalRW*(singleRWSpeed/motorNominalSpeedTimeConst)]*4;
wheelMomentums = [inertiaTotalRW*singleRWSpeed]*4
#Maximum Torque and Momentum in the Satellite Frame
#Max X & Y occur at [+,+,-,-] and max Z at [+,+,+,+]
wheelTorquesMax_SatelliteFrame = [wheelTorques[1]*np.cos(pyCfgAngle),
wheelTorques[2]*np.cos(pyCfgAngle), 4*wheelTorques[1]*np.cos(pyCfgAngle)]
wheelMomentumMax_SatelliteFrame = [wheelMomentums[1]*np.cos(pyCfgAngle),
wheelMomentums[2]*np.cos(pyCfgAngle), 4*wheelMomentums[1]*np.cos(pyCfgAngle)]
#Minimum Wheel Torque in the Satellite Frame
wheelTorqueMin = min(wheelTorquesMax_SatelliteFrame)

```

```

#Maximum Disturbance Torque
disturbanceTorque = greatestdisurbanceTd
#Satellite's Moment of Inertia
#Required Slew Torque and Momentum Storage
requiredSlewTorque =
np.array(4*slewRequirement_rad*MOI_satellite/(slewRequirement_time**2))
#minSlewTorque = np.amax((requiredSlewTorque))
arr = np.array (requiredSlewTorque)
requiredSlewTorqueX = arr[0,0]
requiredSlewTorqueY = arr[1,1]
requiredSlewTorqueZ = arr[2,2]
requiredSlewTorqueList = [requiredSlewTorqueX, requiredSlewTorqueY,
requiredSlewTorqueZ]
minSlewTorque = np.amin(requiredSlewTorqueList)

minwheelMomentumMax_SatelliteFrame = np.amin(wheelMomentumMax_SatelliteFrame)

requiredMomentumStorage = (orbitalPeriod_ISS/4)*(disturbanceTorque*0.707)
#wheelTorques_SatelliteFrame = matrixAw*wheelTorques;
#Comparison of the Required and Found (Torque and Momentum)

print 'The Greatest Disturbance Torque is', greatestdisurbanceTd
print 'The Torque of Each RW is:', wheelTorques

print 'The Maximum Momentum of the Wheel in the Satellite Frame is:',
wheelMomentumMax_SatelliteFrame
print 'The Momentum of Each Wheel is:', wheelMomentums

print 'The Required Momentum Storage is:', requiredMomentumStorage
print 'The Required Slew Torque is:', requiredSlewTorque

print 'The Maximum Torque of the Wheel in the Satellite Frame is:', wheelTorqueMin
print 'The Minimum Value of all Maximum Momentums in the Satellite Frame is',
minwheelMomentumMax_SatelliteFrame
print 'The Minimum Value of all Required Slew Torques is:', minSlewTorque

#embed()

print 'Wheel Disturbance Torque: {} - {} > {}'.format('Pass' if wheelTorqueMin >
safetyMargin*greatestdisurbanceTd else 'Fail' ,wheelTorqueMin,
safetyMargin*greatestdisurbanceTd)
print 'Slew Torque: {} - {} > {}'.format('Pass' if wheelTorqueMin >
safetyMargin*minSlewTorque else 'Fail' , wheelTorqueMin, safetyMargin*minSlewTorque)
print 'Momentum Storage: {} - {} > {}'.format('Pass' if
minwheelMomentumMax_SatelliteFrame > safetyMargin*requiredMomentumStorage else
'Fail' ,minwheelMomentumMax_SatelliteFrame, safetyMargin*requiredMomentumStorage)

```

## Appendix B

```
from numpy import *
import matplotlib.pyplot as plt
from IPython import embed
from scipy.optimize import minimize

#based on: http://www.raumfahrt.fh-aachen.de/compass-
1/download/Diploma_Thesis_Ali_Aydinlioglu.pdf

#https://en.wikibooks.org/wiki/Engineering_Tables/Standard_Wire_Gauge
V_bus = 5 #V
u_0 = 1.25663706e-6 #m kg s-2 A-2 - permeability of free space

guageCSA = {
20: 0.5176e-6,
21: 0.4105e-6,
22: 0.3255e-6,
23: 0.2582e-6,
24: 0.2047e-6,
25: 0.1624e-6,
26: 0.1288e-6,
27: 0.1021e-6,
28: 0.0810e-6,
29: 0.0642e-6,
30: 0.0509e-6,
31: 0.0404e-6,
32: 0.0320e-6,
33: 0.0254e-6,
34: 0.0201e-6,
35: 0.0160e-6,
36: 0.0127e-6,
37: 0.0100e-6,
38: 0.0080e-6,
39: 0.0063e-6,
40: 0.0050e-6,
}

materialDict = {
    'Copper': {
        'density': [8.92e3, 'kg/m^3'],
        'permiability': [1.256629e-6, 'H/m'],
        'resistivity': [1.55e-8, 'Ohm*m'],
        'u_r': [1.256629e-6/u_0, 'u/u0']
    },
    #'Al': {
    #    'density': [2.7e-3, 'g/mm^3'],
    #    'resistivity': [2.5e-5, 'Ohm*mm'],
    #    'tempCoeffResistivity': [3.90e-3, '1/K'],
    #},
    'MnZnFerrite': {
        'density': [4.837e3, 'kg/m^3'],
        'u_r': [2300, 'u/u0'],
    }
}

'''
xMap:
0 - radius of the core
1 - N number of turns of wire
'''
```

```

#guage, m^2

allResults = []
for guage, a_w in guageCSA.items():
    def demagnetizingFactor(params):
        rc, lc, N = params
        try:
            4*(log(lc/rc)-1.)/((lc/rc)**2 - 4*log(lc/rc))
        except Exception as e:
            print e
            embed()
        return 4*(log(lc/rc)-1.)/((lc/rc)**2 - 4*log(lc/rc))

    def magDipoleMoment(params, mat_core=materialDict['MnZnFerrite'],
mat_wire=materialDict['Copper']):
        rc, lc, N = params
        M = (pi*(rc**2)*N*V_bus/wireResistance(params))*(1+((mat_core['u_r'][0]-
1)/(1.+(mat_core['u_r'][0]-1)*demagnetizingFactor(params))))
        return M

    def objective(params):
        return -magDipoleMoment(params)

    def powerConstraint(params, power=0.2, mat_core=materialDict['MnZnFerrite'],
mat_wire=materialDict['Copper'], get=False):
        rc, lc, N = params
        result = V_bus**2/wireResistance(params)
        if get:
            return result
        return power - result

    def numTurnsConstraint(params, numTurns=10000):
        rc, lc, N = params
        return numTurns - N

    def dimensionConstraint(params):
        rc, lc, N = params
        return lc - rc

    def wireResistance(params, mat_core=materialDict['MnZnFerrite'],
mat_wire=materialDict['Copper']):
        rc, lc, N = params
        #R_m - resistance of wire material - Ohm*m
        #a_w - guage of wire - m^2
        return 2*pi*rc*N*mat_wire['resistivity'][0]/a_w

    def massConstraint(params, mass=0.08, mat_core=materialDict['MnZnFerrite'],
mat_wire=materialDict['Copper'], get=False):
        rc, lc, N = params
        l_wire = 2*pi*rc*N
        result = (mat_core['density'][0]*pi*(rc**2)*lc +
a_w*l_wire*mat_wire['density'][0])
        if get:
            return result
        return mass - result

    cons = [{'type': 'ineq', 'fun': massConstraint},
            {'type': 'ineq', 'fun': powerConstraint},
            {'type': 'ineq', 'fun': numTurnsConstraint},
            {'type': 'ineq', 'fun': dimensionConstraint}]

```

```

#rc, lc, N
x0 = [4.75e-3, 70e-3, 5000]

bnds = ((1e-3, 10e-3), (40e-3, 70e-3), (1000, 50000))

sol = minimize(objective, x0, method='SLSQP', constraints=cons, bounds=bnds)

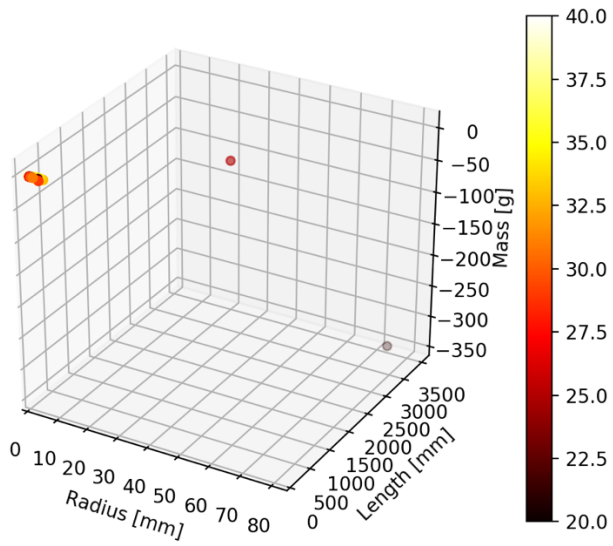
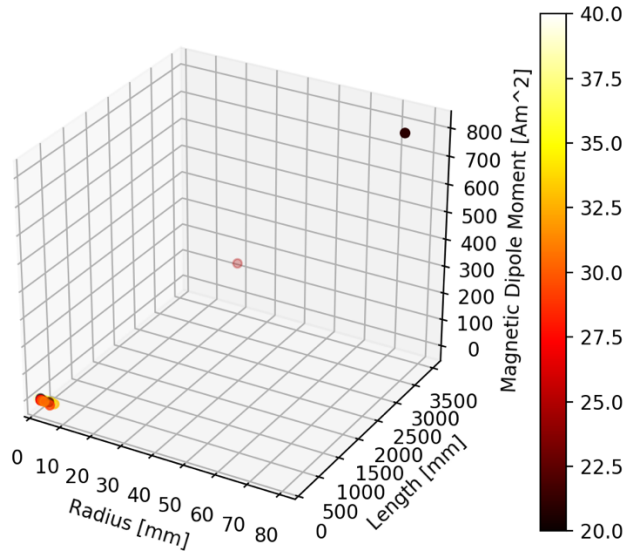
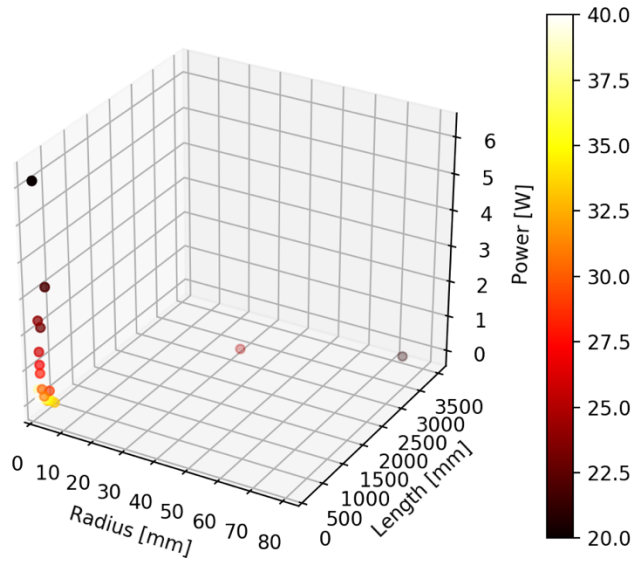
xOpt = sol.x
paramOpt = -sol.fun
status = 'Pass' if sol.status else 'Fail'
print 'Status: {} - G: {} AWG, M: {} Am^2, Mass: {} kg, Power: {} W, r_c: {}
mm, l_c: {} mm, N: {}'.format(status, gauge, paramOpt, massConstraint(xOpt,
get=True), powerConstraint(xOpt, get=True), xOpt[0]*1e3, xOpt[1]*1e3, xOpt[2])
    allResults.append([gauge, massConstraint(xOpt, get=True),
powerConstraint(xOpt, get=True), paramOpt]+list(xOpt))
allResults = array(allResults).T

from mpl_toolkits.mplot3d import Axes3D
zPlots = {
    'Magnetic Dipole Moment [Am^2]': allResults[3],
    'Mass [g]': allResults[1],
    'Power [W]': allResults[2],
}
#gauge, massConstr, powerConstr, magMoment, r_c, l_c, N
for zTitle, zs in zPlots.items():
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    xs = allResults[4]*1e3
    ys = allResults[5]*1e3
    values = allResults[0]
    ax.set_xlabel('Radius [mm]')
    ax.set_ylabel('Length [mm]')
    ax.set_zlabel(zTitle)
    ax.set_xlim3d(0, max(xs)*1.1)
    ax.set_ylim3d(0, max(ys)*1.1)
    p = ax.scatter3D(xs, ys, zs=zs, c=values, cmap='hot')

    fig.colorbar(p, ax=ax)
plt.show()
plt.pause(.001)
embed()

```

# Appendix C





## Appendix D

```
clc;
% Body Frame Definitions:
%{
    X+ : Nadir pointing
    Z+ : To Top of Sat: Antenna Deployment System
    Y+ : Orthonormal
%}
%To_From_csys
%csys_name

%Elevation Angle = theta (u) Defined at the height of the observer
%Azimuth Angle = Phi (v) Defiend as the angle between the x-y axis
% Theta = u phi = v

dT = 0.01; % Defined as the change in time
omega = 20; %deg/s % Change in the angle (degrees) per second
dTheta = omega*dT; % Change in angle theta with respect to omega and time
u = 0:dTheta:360; % The angle of rotation (from 0-360 degrees)
%v = 0:dt:90;
v = zeros(size(u)); % Phi angle (angles of elevation)
I_max = 100; % Setting the maximum current (The team set this value just for
simualtion purposes)
%I_Theta = I_max*sind(u);
I_measured = I_max*[sind(v).*cosd(u); cosd(v).*cosd(u); sind(u); -
(sind(v).*cosd(u)); -(cosd(v).*cosd(u)); -(sind(u))]; % Matrix for each face of the
cubesat
I_measured = max(I_measured,0) % Defining I measured matrix based on the I_measured
formula above and setting all negative values to zero
plot(transp(I_measured)); % Plot the I_measured
grid on;

% R_Positive_X=[1 0 0;0 cosd(u) -sind(u);0 sind(u) cosd(u);]
% R_Negative_X=[1 0 0;0 cosd(u) sind(u);0 sind(u) cosd(u);]
% R_Positive_Y=[cosd(u) 0 sind(u);0 1 0; -sind(u) 0 cosd(u);]
% R_Negative_Y=[cosd(u) 0 -sind(u);0 1 0; sind(u) 0 cosd(u);]
% R_Positive_Z=[cosd(u) -sind(u) 0;sind(u) cosd(u) 0; 0 0 1;]
% R_Negative_Z=[cosd(u) sind(u) 0;-sind(u) cosd(u) 0; 0 0 1;]
```

# Appendix E

Sample result of vector (I\_measured for the first 39 iterations)

I\_measured =

Columns 1 through 13

|          |         |         |         |         |         |         |         |         |         |         |         |         |
|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 0        | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       |
| 100.0000 | 99.9994 | 99.9976 | 99.9945 | 99.9903 | 99.9848 | 99.9781 | 99.9701 | 99.9610 | 99.9507 | 99.9391 | 99.9263 | 99.9123 |
| 0        | 0.3491  | 0.6981  | 1.0472  | 1.3962  | 1.7452  | 2.0942  | 2.4432  | 2.7922  | 3.1411  | 3.4899  | 3.8388  | 4.1876  |
| 0        | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       |
| 0        | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       |
| 0        | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       |

Columns 14 through 26

|         |         |         |         |         |         |         |         |         |         |         |         |         |
|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       |
| 99.8971 | 99.8806 | 99.8630 | 99.8441 | 99.8240 | 99.8027 | 99.7801 | 99.7564 | 99.7314 | 99.7053 | 99.6779 | 99.6493 | 99.6195 |
| 4.5363  | 4.8850  | 5.2336  | 5.5822  | 5.9306  | 6.2791  | 6.6274  | 6.9756  | 7.3238  | 7.6719  | 8.0199  | 8.3678  | 8.7156  |
| 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       |
| 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       |
| 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       |

Columns 27 through 39

|         |         |         |         |         |         |         |         |         |         |         |         |         |
|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       |
| 99.5884 | 99.5562 | 99.5227 | 99.4881 | 99.4522 | 99.4151 | 99.3768 | 99.3373 | 99.2966 | 99.2546 | 99.2115 | 99.1671 | 99.1216 |
| 9.0633  | 9.4108  | 9.7583  | 10.1056 | 10.4528 | 10.7999 | 11.1469 | 11.4937 | 11.8404 | 12.1869 | 12.5333 | 12.8796 | 13.2256 |
| 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       |
| 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       |
| 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       |

## Appendix F

### Winsat DetumbleTorque1.m

```
function [output] = Winsat_DetumbleTorque1(input)

switch input.method

    case 'compute'
        computeData = input.methodData;

        % Output Control Torque in form expected by Attitude Simulator
        MagField = computeData.MagFieldIGRF(4:6);
        Angualr_velocity = computeData.Angvelocity;
        output.Torque = -1.1 * (eye(3)-MagField/norm(MagField)*MagField')
*Angualr_velocity;

    case 'register'

        param1 = { 'ArgumentName','time',...
            'Name','Epoch',...
            'ArgumentType','Input'};

        param2 = { 'ArgumentName','MagFieldIGRF',...
            'Type','Vector',...
            'Name','MagField(IGRF)',...
            'RefType','Attitude',...
            'Derivative','Yes',...
            'ArgumentType','Input'};

        param3 = { 'ArgumentName','Angvelocity',...
            'Type','Vector',...
            'Name','AngVelocity',...
            'RefType','Attitude',...
            'Derivative','NO',...
            'ArgumentType','Input'};

        param4 = { 'ArgumentName','Torque',...
            'Type','Parameter',...
            'Name','Torque',...
            'BasicType','Vector',...
            'ArgumentType','Output'};

        output = {param1, param2, param3, param4};

    otherwise
        output = [];

End
```

## Appendix G

### Test sgp4.m

```
clc
clear
format long g

global const
SAT_Const

ge = 398600.8; % Earth gravitational constant
TWOPI = 2*pi;
MINUTES_PER_DAY = 1440.;
MINUTES_PER_DAY_SQUARED = (MINUTES_PER_DAY * MINUTES_PER_DAY);
MINUTES_PER_DAY_CUBED = (MINUTES_PER_DAY * MINUTES_PER_DAY_SQUARED);

% TLE file name
fname = 'tle.txt';

% Open the TLE file and read TLE elements
fid = fopen(fname, 'r');

% 19-32    04236.56031392    Element Set Epoch (UTC)
% 3-7      25544      Satellite Catalog Number
% 9-16     51.6335     Orbit Inclination (degrees)
% 18-25    344.7760    Right Ascension of Ascending Node (degrees)
% 27-33    0007976    Eccentricity (decimal point assumed)
% 35-42    126.2523    Argument of Perigee (degrees)
% 44-51    325.9359    Mean Anomaly (degrees)
% 53-63    15.70406856 Mean Motion (revolutions/day)
% 64-68    32890      Revolution Number at Epoch

while (1)
    % read first line
    tline = fgetl(fid);
    if ~ischar(tline)
        break
    end
    Cnum = tline(3:7);           % Catalog Number (NORAD)
    SC   = tline(8);           % Security Classification
    ID   = tline(10:17);       % Identification Number
    year = str2num(tline(19:20)); % Year
    doy  = str2num(tline(21:32)); % Day of year
    epoch = str2num(tline(19:32)); % Epoch
    TD1   = str2num(tline(34:43)); % first time derivative
    TD2   = str2num(tline(45:50)); % 2nd Time Derivative
    ExTD2 = tline(51:52);      % Exponent of 2nd Time Derivative
    BStar = str2num(tline(54:59)); % Bstar/drag Term
    ExBStar = str2num(tline(60:61)); % Exponent of Bstar/drag Term
    BStar = BStar*1e-5*10^ExBStar;
    Etype = tline(63);         % Ephemeris Type
    Enum  = str2num(tline(65:end)); % Element Number

    % read second line
    tline = fgetl(fid);
    if ~ischar(tline)
        break
    end
    i = str2num(tline(9:16));   % Orbit Inclination (degrees)
```

```

    raan = str2num(tline(18:25)); % Right Ascension of Ascending Node
(degrees)
    e = str2num(strcat('0.',tline(27:33))); % Eccentricity
    omega = str2num(tline(35:42)); % Argument of Perigee (degrees)
    M = str2num(tline(44:51)); % Mean Anomaly (degrees)
    no = str2num(tline(53:63)); % Mean Motion
    a = ( ge/(no*2*pi/86400)^2 )^(1/3); % semi major axis (m)
    rNo = str2num(tline(64:68)); % Revolution Number at Epoch
end
fclose(fid);

satdata.epoch = epoch;
satdata.norad_number = Cnum;
satdata.bulletin_number = ID;
satdata.classification = SC; % almost always 'U'
satdata.revolution_number = rNo;
satdata.ephemeris_type = Etype;
satdata.xmo = M * (pi/180);
satdata.xnodeo = raan * (pi/180);
satdata.omegao = omega * (pi/180);
satdata.xincl = i * (pi/180);
satdata.eo = e;
satdata.xno = no * TWOPI / MINUTES_PER_DAY;
satdata.xndt2o = TD1 * 1e-8 * TWOPI / MINUTES_PER_DAY_SQUARED;
satdata.xnnd6o = TD2 * TWOPI / MINUTES_PER_DAY_CUBED;
satdata.bstar = BStar;

tsince = 1440; % amount of time in which you are going to propagate satellite's
state vector forward (+) or backward (-) [minutes]

[rtime, vtime] = sgp4(tsince, satdata);

% read Earth orientation parameters
fid = fopen('eop19620101.txt','r');
% -----
% | Date      MJD      x          y          UT1-UTC      LOD          dPsi          dEpsilon
dX          dY          DAT          "          "          s          s          "          "
% |(0h UTC)          "          "          "          "          "          "          "          "
"          "          s          "          "          "          "          "          "
% -----
eopdata = fscanf(fid,'%i %d %d %i %f %f %f %f %f %f %f %f %i',[13 inf]);
fclose(fid);

if (year < 57)
    year = year + 2000;
else
    year = year + 1900;
end

[mon,day,hr,minute,sec] = days2mdh(year,doy);
MJD_Epoch = Mjday(year,mon,day,hr,minute,sec);
[mon,day,hr,minute,sec] = days2mdh(year,doy+tsince/1440);
MJD_UTC = Mjday(year,mon,day,hr,minute,sec);

% Earth Orientation Parameters
[x_pole,y_pole,UT1_UTC,LOD,dpsi,deps,dx_pole,dy_pole,TAI_UTC] =
IERS(eopdata,MJD_UTC,'1');
[UT1_TAI,UTC_GPS,UT1_GPS,TT_UTC,GPS_UTC] = timediff(UT1_UTC,TAI_UTC);

```

```
MJD_UT1 = MJD.UTC + UT1.UTC/86400;  
MJD_TT = MJD.UTC + TT.UTC/86400;  
T = (MJD_TT-const.MJD_J2000)/36525;  
[positionECI, velocityECI] = teme2eci(rteme,vteme,T,dpsi,deps)  
[recef,vecef] = teme2ecef(rteme,vteme,T,MJD_UT1+2400000.5,LOD,x_pole,y_pole)
```

## Appendix H

### TRIAD.m

```
function A_traid =
TRIAD(sunSensorVector_BF,magFieldStrength_BF,sunSensorVector_ECI,magFieldStrength_EC
I)
% Abr = TRIAD(fb, mb, fn, mn)
% Function implements TRIAD algorithm using measurements
% from three-component accelerometer with orthogonal axes and vector
% magnetometer
%
% Input arguments:
% sunSensorVector_BF - Sun vector in body frame [3x1]
% magFieldStrength - Magnetic field vector in body frame [3x1]
% sunSensorVector_ECI - Sun vector in ECI frame [3x1]
% magFieldStrength_ECI - Magnetic field vector in ECI frame [3x1]
%
% Output arguments:
% Abr - estimated Direction Cosines Matrix (DCM)
traid1 = [sunSensorVector_BF/norm(sunSensorVector_BF),...

cross(sunSensorVector_BF,magFieldStrength_BF)/norm(cross(sunSensorVector_BF,magField
Strength_BF))];
traid1 = [traid1, cross(traid1(:,1),traid1(:,2))];

traid2 = [sunSensorVector_ECI/norm(sunSensorVector_ECI),...

cross(sunSensorVector_ECI,magFieldStrength_ECI)/norm(cross(sunSensorVector_ECI,magFi
eldStrength_ECI))];
traid2 = [traid2, cross(traid2(:,1),traid2(:,2))];

A_traid = traid1*traid2';
End
```

## Appendix I

### SunSensorModelECI.m

```
function [sunVectorX, sunVectorY, sunVectorZ] = sunSensorModelECI(tleFile)

%Open text file
fileID = fopen(tleFile,'r');
tline = fgetl(fileID);

%We want to extract line 19-33 from out ISS_TLE file to find Julian Date
year = tline(19:21);
month = tline(21:23);

julianDate = (367*year) - ((7*year) + (month + 9)/12)/4 + (275*month)/9 +1721013.5 +
hour/24 + minute/1440 + second/86400;

%Given the julian date - compute unit vector direction to the sun

Tut1 = (julianDate - 2451545)/36525;
meanLongitudeOfSun = 280.4606184 + 36000.7705361*Tut1;
meanAnomalyOfSun = 357.5277233 + 35999.05034*Tut1;
eclipticLongitudeOfsun = meanLongitudeOfSun +
1.914666471*sin(meanAnomalyOfSun)+0.918994643*sin(2*meanAnomalyOfSun);

elipsoid = 23.439291 - 0.0130042 * Tut1;

%calculates the sun vector components
sunVectorX = cos(eclipticLongitudeOfsun);
sunVectorY = cos(elipsoid)*sin(eclipticLongitudeOfsun);
sunVectorZ = sin(elipsoid)*sin(eclipticLongitudeOfsun);
end
MagFieldModelECI.m
function [magFieldStrengthX,magFieldStrengthY,magFieldStrengthZ] =
magFieldModelECI(positionECI)

    coelevationAngle = 196.54;           %[degrees]
    eastLongitudeDipoleAngle = 108.43;   %[degrees]

    unitDipoleDirection =
[sin(coelevationAngle)*cos(eastLongitudeDipoleAngle), sin(coelevationAngle)*sin(eastL
ongitudeDipoleAngle), cos(coelevationAngle)];
```



## Appendix J

### N\_CSA\_Script

```
clear; clc;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% ORBITAL PARAMETERS %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
mue = 398600;
Re = 6378;
Rs = 500;
Rp = Re+Rs;
n = sqrt(mue/(Rp^3));
thtd2 = n^2;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% MOMENT OF INERTIA %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Js = diag([0.002928049; 0.014683125; 0.001477392]); % WinSAT1%
Jw = diag([1e-5, 1e-5, 1e-5, 1e-5]); % Reaction Wheels %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% REACTION WHEEL PARAMETERS %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Ra = 0.6; % Coil Resistance
Km1 = 0.0082; % Motor Constant - Wheel 1
Km2 = 0.0080; % Motor Constant - Wheel 2
Km3 = 0.0071; % Motor Constant - Wheel 3
Km4 = 0.0080; % Motor Constant - Wheel 4
dp_x = 1.3; % Positive Deadzone - Wheel 1
dn_x = -1.1; % Negative Deadzone - Wheel 1
dn_y = -1.15; % Positive Deadzone - Wheel 2
dp_y = 1.2; % Negative Deadzone - Wheel 2
dp_z = 1.2; % Positive Deadzone - Wheel 3
dn_z = -1.1; % Negative Deadzone - Wheel 3
dp_o = 1.1; % Positive Deadzone - Wheel 4
dn_o = -1.15; % Negative Deadzone - Wheel 4
V_max = 4.50; % Wheel - Maximum Voltage
wsd = 0.01; % Lowest speed
alpha = 45*pi/180; % In-plane angle
beta = 45*pi/180; % Out-of-plane angle

CASE = 1; % 1 = Orthogonal, 2 = Standard four Wheels, 3 = Pyramid

Ap = [Ra; Km1; Km2; Km3; Km4; dp_x; dn_x; dp_y; dn_y; dp_z; dn_z; ...
      dp_o; dn_o; V_max; wsd; alpha; beta; CASE];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% CONTROL GAINS %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
p1 = 1;
p2 = 1;
p3 = 1;
b1 = 300;
b2 = 0.001;
k1 = 0.06;
bh0 = 0.0;
lambda = 1;
delta = 1e-3;
alpha = 0.00001; %added for 3-axis magntic control
beta = 0.1; %added for 3-axis magntic control

Cg = [p1; p2; p3; b1; b2; k1; delta; lambda; thtd2; alpha; beta];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% INITIAL CONDITIONS %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
roll0 = 50*pi/180;
pitch0 = 20*pi/180;
yaw0 = -30*pi/180;
q0 = angle2quat(yaw0,pitch0,roll0);

%sat position in orbit frame
%now instead we will use,
%ang velocity in body frame XYZ
%magnteic field val in body XYZ
%sun vect values in body XYZ
%time since epoch
%3U cubesat TLE file in ISS orbit
%OUTPUT - current quat in orbit frame. nadir = [1,0,0,0]
%OUTPUT - current quaternion attitude

q10 = q0(2); %0.4;
q20 = q0(3); %0.4;
q30 = q0(4); %0.4;
q40 = q0(1); %sqrt(1 - q10^2 - q20^2 - q30^2);

w10 = 1.000; %initial value for magnetic control (deg/s)
w20 = -0.500; %initial value for magnetic control (deg/s)
w30 = 0.300; %initial value for magnetic control (deg/s)

X0 = [ q10; q20; q30; q40; w10; w20; w30; 0; 0; 0; 0 ];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESIRED CONDITIONS
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
roll0d = 0*pi/180;
pitch0d = 0*pi/180;
yaw0d = 0*pi/180;
q0d = angle2quat(yaw0d,pitch0d,roll0d);

qd = [q0d(2); q0d(3); q0d(4)]; %[q1d; q2d; q3d];
q4d = q0d(1); %sqrt(1 - (qd'*qd));
qD = [qd; q4d];
wD = [0; 0; 0];

sim('N_CSA_Model.mdl',[0 30]);
save 'results.mat'
N_CSA_Plotter();

```

## Appendix K

### N\_CSA\_Model.m

```
function [State_dot, Q_er, w_sat, Voltage, Tau_ap, w_I, Euler, w_speed, sigma_dot]
...
    = N_sim_model(States, qD, wD, Jw, Js, Cg, Ap,t,sigma)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% REACTION WHEEL PARAMETERS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Ra      = Ap(1);           % Coil Resistance           %
Km1     = Ap(2);           % Motor Constant - Wheel 1   %
Km2     = Ap(3);           % Motor Constant - Wheel 2   %
Km3     = Ap(4);           % Motor Constant - Wheel 3   %
Km4     = Ap(5);           % Motor Constant - Wheel 4   %
dp_x    = Ap(6);           % Positive Deadzone - Wheel 1 %
dn_x    = Ap(7);           % Negative Deadzone - Wheel 1 %
dp_y    = Ap(8);           % Positive Deadzone - Wheel 2 %
dn_y    = Ap(9);           % Negative Deadzone - Wheel 2 %
dp_z    = Ap(10);          % Positive Deadzone - Wheel 3 %
dn_z    = Ap(11);          % Negative Deadzone - Wheel 3 %
dp_o    = Ap(12);          % Positive Deadzone - Wheel 4 %
dn_o    = Ap(13);          % Negative Deadzone - Wheel 4 %
V_max   = Ap(14);          % Wheel - Maximum Voltage    %
wsd     = Ap(15);          %                               %
Km_mat  = diag([Km1, Km2, Km3, Km4]); %                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% REACTION WHEEL CONFIGURATION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
alpha   = Ap(16);          % In-plane angle             %
beta    = Ap(17);          % Out-of-plane angle         %
cbca    = cos(beta)*cos(alpha); %                               %
cbsa    = cos(beta)*sin(alpha); %                               %
sb      = sin(beta);       %                               %
CASE    = Ap(18);          % 3-configurations           %
%
if CASE == 1
    R_dist = [ 1 0 0 0 ; % Three orthogonal wheels %
               0 1 0 0 ; %
               0 0 1 0 ]; %
elseif CASE == 2
    R_dist = [ 1 0 0 -cbca ; % Standard four wheels %
               0 1 0 -cbsa ; %
               0 0 1  sb  ]; %
elseif CASE == 3
    R_dist = [ -cbca -cbca  cbca  cbca ; % Pyramid configuration %
               cbsa -cbsa -cbsa  cbsa ; %
               sb    sb    sb    sb  ]; %
else
    R_dist = [ 1 0 0 0 ; % Three orthogonal wheels %
               0 1 0 0 ; %
               0 0 1 0 ]; %
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% CONTROL GAINS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
p1      = Cg(1);           %                               %
p2      = Cg(2);           %                               %
p3      = Cg(3);           %                               %
b1      = Cg(4);           %                               %
b2      = Cg(5);           %                               %
```

```

k1      = Cg(6);
delta  = Cg(7);
lambda = Cg(8);
thtd2  = Cg(9);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%                               INTEGRATED STATES
%
q1 = States(1);
q2 = States(2);
q3 = States(3);
q4 = States(4);
w   = States(5:7);
q   = [q1; q2; q3];
w_speed = States(8:11);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%                               ESSENTIAL TERMS AND MATRICES
%
q_cross = [ 0  -q3  q2  ;
           q3   0 -q1  ;
          -q2  q1   0  ];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%                               ROTATION MATRIX - BODY TO ORBITAL
%
R_bo = (q4^2 - q'*q)*eye(3,3) + 2*(q*q') - 2*q4*q_cross;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%                               ANGULAR VELOCITY - ORBITAL TO INERTIAL
%
w_oI = R_bo*[0; -sqrt(thtd2); 0];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%                               ANGULAR VELOCITY - BODY TO INERTIAL
%
w_I = w + w_oI;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%                               QUATERNION KINEMATICS
%
q_dot = 0.5*(q_cross + q4*eye(3,3))*w;
q4_dot = -0.5*transpose(q)*w;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%                               DESIRED ATTITUDE
%
qd   = [qD(1); qD(2); qD(3)];
q4d  = qD(4);

q_e  = q4d*q - q4*qd + cross(q,qd);
q4_e = q4d*q4 + qd'*q;

qe_c = [ 0      -q_e(3)  q_e(2)  ;
        q_e(3)  0      -q_e(1)  ;
        -q_e(2)  q_e(1)  0      ];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%                               CONVERSION TO EULER ANGLES
%
roll  = (180/pi)*atan2(R_bo(2,3),R_bo(3,3));
pitch = -(180/pi)*asin(R_bo(1,3));
yaw   = (180/pi)*atan2(R_bo(1,2),R_bo(1,1));

Euler = [roll; pitch; yaw];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%                               ROTATION MATRIX - BODY TO DESIRED
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

Re_bi = (q4_e^2 - q_e'*q_e)*eye(3,3) + 2*(q_e*q_e') - 2*q4_e*qe_c;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%                               ANGULAR VELOCITY ERROR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
w_e   = w - Re_bi*wD;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%                               SLIDING PLANE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Sp = w_e + sign(q4_e)*lambda*q_e;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%                               CONTROL ALGORITHM
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
eta   = p1*norm(w) + p2*norm(q) + p3;
rho   = sigma*eta;
K_t   = 1/(norm(Sp) + delta);
R_Pin = R_dist'*inv(R_dist*R_dist');
U_des = (k1 + (rho + eta)*K_t)*R_Pin*Sp;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%                               ADAPTIVE LAW
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sigma_dot = -b1*sigma + b2*eta*(norm(Sp)^2)/(norm(Sp) + delta);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%                               WHEEL VOLTAGE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Voltage = Ra*inv(Km_mat)*U_des + Km_mat*w_speed;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%                               VOLTAGE DEADZONE LIMITATION (Lower limits)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (Voltage(1) <= dp_x) && (Voltage(1)>0) && (abs(w_speed(1)) < wsd)
    Voltage(1) = dp_x;
elseif(Voltage(1) >= dn_x) && (Voltage(1)<0) && (abs(w_speed(1)) < wsd)
    Voltage(1) = dn_x;
end
if (Voltage(2) <= dp_y) && (Voltage(2)>0) && (abs(w_speed(2)) < wsd)
    Voltage(2) = dp_y;
elseif(Voltage(2) >= dn_y) && (Voltage(2)<0) && (abs(w_speed(2)) < wsd)
    Voltage(2) = dn_y;
end
if (Voltage(3) <= dp_z) && (Voltage(3)>0) && (abs(w_speed(3)) < wsd)
    Voltage(3) = dp_z;
elseif(Voltage(3) >= dn_z) && (Voltage(3)<0) && (abs(w_speed(3)) < wsd)
    Voltage(3) = dn_z;
end
if (Voltage(4) <= dp_o) && (Voltage(4)>0) && (abs(w_speed(4)) < wsd)
    Voltage(4) = dp_o;
elseif(Voltage(4) >= dn_o) && (Voltage(4)<0) && (abs(w_speed(4)) < wsd)
    Voltage(4) = dn_o;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%                               VOLTAGE SATURATION LIMITS (Upper bound)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (abs(Voltage(1)) > V_max) && (Voltage(1) < 0)
    Voltage(1) = -V_max;
elseif (abs(Voltage(1)) > V_max) && (Voltage(1) > 0)
    Voltage(1) = V_max;
end
if (abs(Voltage(2)) > V_max) && (Voltage(2) < 0)
    Voltage(2) = -V_max;
elseif (abs(Voltage(2)) > V_max) && (Voltage(2) > 0)
    Voltage(2) = V_max;

```

```

end
if (abs(Voltage(3)) > V_max) && (Voltage(3) < 0)
    Voltage(3) = -V_max;
elseif (abs(Voltage(3)) > V_max) && (Voltage(3) > 0)
    Voltage(3) = V_max;
end
if (abs(Voltage(4)) > V_max) && (Voltage(4) < 0)
    Voltage(4) = -V_max;
elseif (abs(Voltage(4)) > V_max) && (Voltage(4) > 0)
    Voltage(4) = V_max;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% GRAVITY GRADIENT TORQUE
%
GGT = 3*thtd2*cross(R_bo(:,3), (Js*R_bo(:,3)));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% APPLIED TORQUE
%
Tau_ap = U_des; % (1/Ra)*Km_mat*(Voltage - Km_mat*w_speed); % Applied Torque
Jmat = Js - R_dist*Jw*R_dist'; % Combined Inertia Matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% SPACECRAFT DYNAMICS
%
w_0 = sqrt(thtd2)*1000;
EDT = [ 0*4e-6 + 2e-6 * sin(w_0*t);
        0*6e-6 + 3e-6 * sin(w_0*t);
        0*3e-6 + 3e-6 * sin(w_0*t)];
w_dot = cross(w,w_oI) + inv(Jmat)*(-cross(w_I, (Js*w_I+R_dist*Jw*w_speed))...
    + GGT + l*EDT - R_dist*Tau_ap);
Ow_dot = inv(Jw)*Tau_ap - R_dist'*(w_dot - cross(w,w_oI));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% OUTPUT FOR INTEGRATION
%
State_dot = [q_dot; q4_dot; w_dot; Ow_dot];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Q_er = [q_e; q4_e];
w_sat = w;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

N_CSA_Plotter.m
function [] = N_CSA_Plotter()
clc;
close all;
load 'results.mat'

line_w=2;
fig_max_row = 3;
fig_max_col = 3;

h1 = figure('NumberTitle', 'off', 'Name', 'Quaternion Error');
%placefigure(h1,[fig_max_row fig_max_col 1 1]);
plot(quat_er,'linewidth',line_w);
title('Quaternion Error');
xlabel('time');
ylabel('Error');
legend('q0','q1','q2','q3','location','best');

h2 = figure('NumberTitle', 'off', 'Name', 'Angular Velocity Error');
%placefigure(h2,[fig_max_row fig_max_col 1 2]);
plot(angular_er,'linewidth',line_w);

```

```

title('Angular Velocity Error');
xlabel('time');
ylabel('Error');
legend('\omega_1', '\omega_2', '\omega_3', 'location', 'best');

h3 = figure('NumberTitle', 'off', 'Name', 'Input Voltage');
%placefigure(h3,[fig_max_row fig_max_col 1 3]);
plot(input_v, 'linewidth', line_w);
title('Input Voltage');
xlabel('time');
ylabel('Voltage');
legend('wheel_1', 'wheel_2', 'wheel_3', 'wheel_4', 'location', 'best');

h4 = figure('NumberTitle', 'off', 'Name', 'Applied Torque');
%placefigure(h4,[fig_max_row fig_max_col 2 1]);
plot(applied_torque, 'linewidth', line_w);
title('Applied Torque');
xlabel('time');
ylabel('Torque');
legend('wheel_1', 'wheel_2', 'wheel_3', 'wheel_4', 'location', 'best');

h5 = figure('NumberTitle', 'off', 'Name', 'Angular Velocity');
%placefigure(h5,[fig_max_row fig_max_col 2 2]);
plot(inertial_angular_v, 'linewidth', line_w);
title('Interior Angular Velocity');
xlabel('time');
ylabel('Angular Velocity');
legend('\omega_1', '\omega_2', '\omega_3', 'location', 'best');

h6 = figure('NumberTitle', 'off', 'Name', 'Euler Angles');
%placefigure(h6,[fig_max_row fig_max_col 2 3]);
plot(euler_angles, 'linewidth', line_w);
title('Euler Angles');
xlabel('time');
ylabel('Angle');
legend('Roll', 'Pitch', 'Yaw', 'location', 'best');

h7 = figure('NumberTitle', 'off', 'Name', 'Wheel Speed');
%placefigure(h7,[fig_max_row fig_max_col 3 1]);
plot(wheel_speed, 'linewidth', line_w);
title('Wheel Speed');
xlabel('time');
ylabel('Speed');
legend('wheel_1', 'wheel_2', 'wheel_3', 'wheel_4', 'location', 'best');

```

**Appendix L**

Orange represents time before COVID-19

Blue represents time during COVID-19

Green represents the testing phase

Table 5: Gantt chart for project timeline of completion

|                                       | Jan<br>202<br>0 | Feb<br>2020 | March<br>2020 | April<br>2020 | May<br>2020 | June<br>2020 | July<br>2020 | Aug<br>2020 |
|---------------------------------------|-----------------|-------------|---------------|---------------|-------------|--------------|--------------|-------------|
| Comprehensive Research                | Orange          |             |               |               |             |              |              |             |
| Component Documentation/Selection     |                 | Orange      |               |               |             |              |              |             |
| Design Phase                          |                 | Orange      |               |               |             |              |              |             |
| Sun Sensor & Controller Prototyping   |                 |             | Blue          | Blue          | Blue        | Blue         |              |             |
| Simulation of Sun Sensor & Controller |                 |             |               |               |             |              | Green        | Green       |