Development of an Operational Spacecraft Alignment Algorithm

A Major Qualifying Project

Submitted to the Faculty of

Worcester Polytechnic Institute

in partial fulfillment of the requirements for the

Degree in Bachelor of Science

in

Mechanical Engineering

By

_____

Alexander Lemmon

Date: 8/10/2019

Project Advisor:

_____

Professor Torbjorn Bergstrom, Advisor

Abstract

The goal of this project was to develop a spacecraft alignment analysis algorithm in support of an industry servicing program and to investigate whether axiomatic design principles are applicable to software design. The algorithm uses statistical analysis, matrix manipulation, and interpolation tools to analyze multiple sets of telemetry in support of spacecraft control operators.

Acknowledgements

This project would not have been possible without the advice and assistance of Professor Torbjorn Bergstrom and Professor Christopher Brown, who were instrumental in their guidance and instruction with regards to good design practices and working in industry.

Additionally, I would never have been able to accomplish my work without the mentoring and assistance of the numerous engineers I was fortunate to have as colleagues during my time at Orbital ATK and later as part of Northrop Grumman. I learned more in a matter of months with them than in any class, and the experience and insights I gained were invaluable.

The following covers work done at Northrop Grumman during an internship. The program I was supporting was and remains highly proprietary and involves information sensitive to Northrop Grumman and its customer. All such details are omitted from this report, as is any information regarding past or present employees of the company. All details included in this report are already in the public domain and have been authorized for release. Except where otherwise cited, all information included herein is my own knowledge and the direct result of my work or shared with me by my colleagues and superiors.

Table of Contents

List of Figures

Background

The Mission Extension Vehicle (MEV) program began at Orbital ATK before the company was acquired by Northrop Grumman in 2018, becoming Northrop Grumman Innovation Systems (NGIS). Both Orbital ATK and Northrop Grumman were and remain major satellite manufacturers, holding numerous contracts with various governments and private entities around the world. In recent years, the space industry has been exploring ways to cut costs and test new technologies by extending the lifespan of spacecraft such as commercial telecommunications satellites, postponing the need to build and launch replacements. While a number of companies have been developing the means to accomplish this, Orbital ATK's MEV was first to market and continues to operate under the Northrop Grumman's new sector.

Space is an unforgiving environment to anything man-made, and it is not uncommon to lose control of spacecraft for reasons varying from long-term radiation damage to collisions with space debris or other spacecraft. The most common limiting factor of a communication satellite's lifespan, however, is exhausting its supply of propellant. This can refer both to the combustible fuel and oxidizer used by a rearward-facing liquid apogee engine (LAE) used for orbital maneuvers, or the inert pressurized gases used by reaction control systems (RCS) to control a spacecraft's attitude, or orientation in space. The latter is particularly important for communications satellites which have to maintain a very specific orientation to properly bounce signals from one part of the Earth to another, or to other satellites. Most of the time this attitude control is done by using gyros, also called reaction wheels, onboard the satellite, which are far more precise than gas jets. These reaction wheels, however, can only exert a force to orient the satellite when they accelerate, and when the motors that spin them can accelerate no further the RCS is forced to temporarily take over while the wheels spin down. This is called "dumping" the momentum from the reaction wheels, and just as they exert a force when they accelerate they exert an opposite force when they spin down quickly, which must be countered by the RCS jets. Once the supply of RCS propellant is exhausted, this process can no longer happen and the satellite's ability to orient itself is compromised.

Under international treaties, when a vehicle approaches the point where it can no longer be controlled, it must be removed as a potential hazard to other spacecraft. This is especially

important for communication satellites operating in geosynchronous Earth orbit (GEO), as this orbit has limited space and is divided into slots to keep satellites from interfering with each other's signals. These orbital slots are owned by different countries which lease them out to satellite operators, who in turn charge telecoms companies for their service. The presence of an uncontrollable satellite or worse, a debris field, permanently closes that given slot which cannot be used again until the technology to completely remove the hazard is developed and made available, all of which has a significant impact on the various stakeholders involved. Spacecraft in low Earth orbit (LEO) have it relatively straightforward; they can simply use the last of their fuel to enter a decaying orbit and either burn up in the Earth's atmosphere or survive reentry and be recovered, if they are designed to do so. Satellites in GEO are too far away for this approach to be feasible, as it would simply take too much fuel to return to Earth's atmosphere. For years, the solution has therefore been to designate a so-called "graveyard" orbit approximately 300 km beyond GEO where decommissioned satellites can be placed without risking collisions with operational spacecraft. This arrangement is analogous to a landfill on Earth, and the graveyard orbit is well-mapped and easily avoided by spacecraft venturing beyond it. Because most satellites must be decommissioned due to lack of propellant rather than damage or component failure, there are a number of fully functional satellites in the graveyard orbit that simply do not have enough fuel to be of any use.

Note: GEO can also refer to "geostationary Earth orbit" or "geosynchronous equatorial orbit". These terms all mean the same thing and are interchangeable.

Early ideas for extending the lifespan of existing satellites involved sending another spacecraft to rendezvous with them for refueling, exactly like roadside assistance might bring a container of fuel to a stranded vehicle on a highway. Unfortunately the technical challenges inherent in remotely refueling a satellite in space that was never designed to be serviced proved insurmountable, so the focus shifted to providing an alternate method of maneuvering the satellite. The MEV is a vehicle derived from an existing satellite structure, called a "bus", that rendezvouses with the client vehicle (CV), docks with it, and provides all necessary propulsion and attitude control, thereby allowing the CV to continue operating. As the majority of telecoms satellites have variants of the same basic LAE design, the MEV is able to dock with them all by extending an expanding probe into the rear of the engine and pulling back on it until three inert

prongs on the front of the MEV are pressed against the rear of the CV with enough force that the two spacecraft are locked together by friction. The MEV then essentially functions as a jetpack providing mobility for the satellite to which it is docked. The MEV is able to do this until something fails onboard the CV or the owner of the satellite stops paying Northrop Grumman for the service. The satellite is then placed in (or returned to) the graveyard orbit and the MEV proceeds to another satellite to begin again. The MEV can function in this role for far longer than the satellites it services due to its electric propulsion, which provides a fraction of the thrust of conventional propulsion but with far greater efficiency.

Figure 1: The MEV (foreground) docked with the client vehicle, forming the combined vehicle stack

Because the satellite being serviced was never designed to be docked with, the MEV's method of docking is, by necessity, an improvisation. Despite every effort on the part of the operators in the mission operations center (MOC) to ensure otherwise, there is no way of guaranteeing that the two spacecraft will be in perfect rotational alignment when they dock, nor is there any way of adjusting their alignment after docking short of undocking and trying again. Furthermore, the structural ring that the MEV presses against was designed for mounting the satellite onto its launch vehicle and normally serves no further purpose after the last stage of the rocket is jettisoned. As the satellite travels along its orbit, it receives various levels of sun exposure across different areas of its exterior. The difference in temperature between the parts of

the satellite facing the sun and the parts in shadow are extreme, and as the materials exposed to the sun's heat expand the entire structure warps slightly. These thermal expansions and contractions are accounted for in the design of the satellite bus and the components on board, but the mounting ring was never intended to have any use by the time these effects occur and therefore was not designed to account for them. With this ring now serving as part of the docking connection, any change in alignment between it and the rest of the structure or instruments results in an alignment error, or bias, between the CV and MEV. The two spacecraft are therefore known to have a fixed mechanical bias from docking and a diurnally varying bias based on the sun and the location of the joined spacecraft, referred to as the combined vehicle stack (CVS), in their orbit. Even if these alignment biases are extremely small, GEO orbit is approximately 36,000 kilometers from the Earth's surface so the geometry involved in pointing a communications satellite from that far away is unforgiving. Because the MEV is the vehicle controlling the orientation of the CVS, even a slight difference between its orientation and that of the CV can prevent the latter from functioning as an effective communications relay. The algorithm I worked on was intended to use telemetry from each vehicle to determine the alignment errors between them so that the MEV's controllers could compensate accordingly and ensure that the CV was properly oriented.

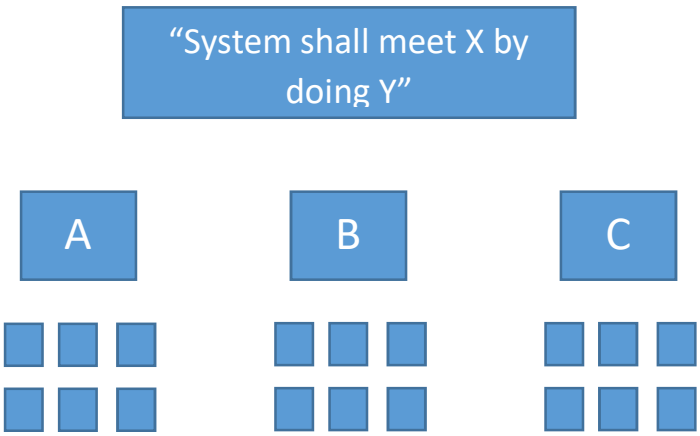
System Engineering Requirements

Before I was assigned to work on the algorithm, I needed to understand the various system requirements driving development of the MEV and how they fit together. This included training in IBM Rational Doors, software used in industry to keep track of such requirements. It functions as a library and organizational tool to not only list all of the requirements in one place, but also to manage the relationships between them all, as well as track compliance.

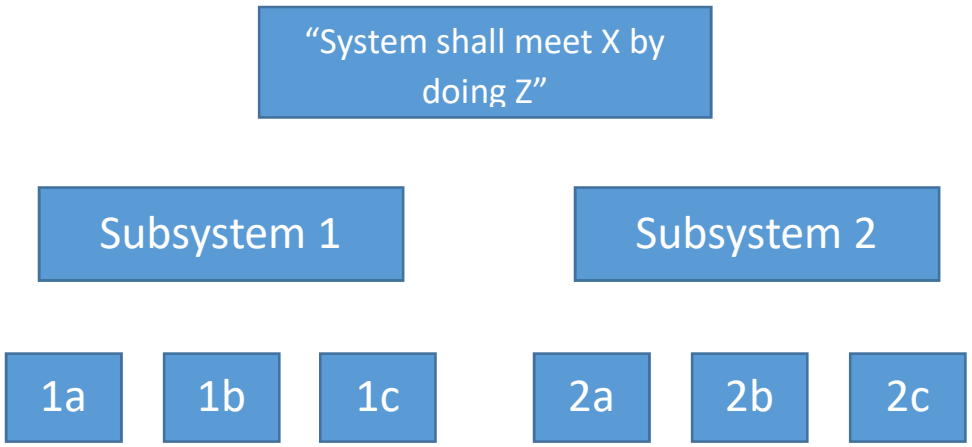
Most systems can be boiled down into a handful of very broad and basic requirements defined by the customer or end user.

As the operational context becomes better understood, additional requirements emerge that specify how the original, primary requirements will be met.

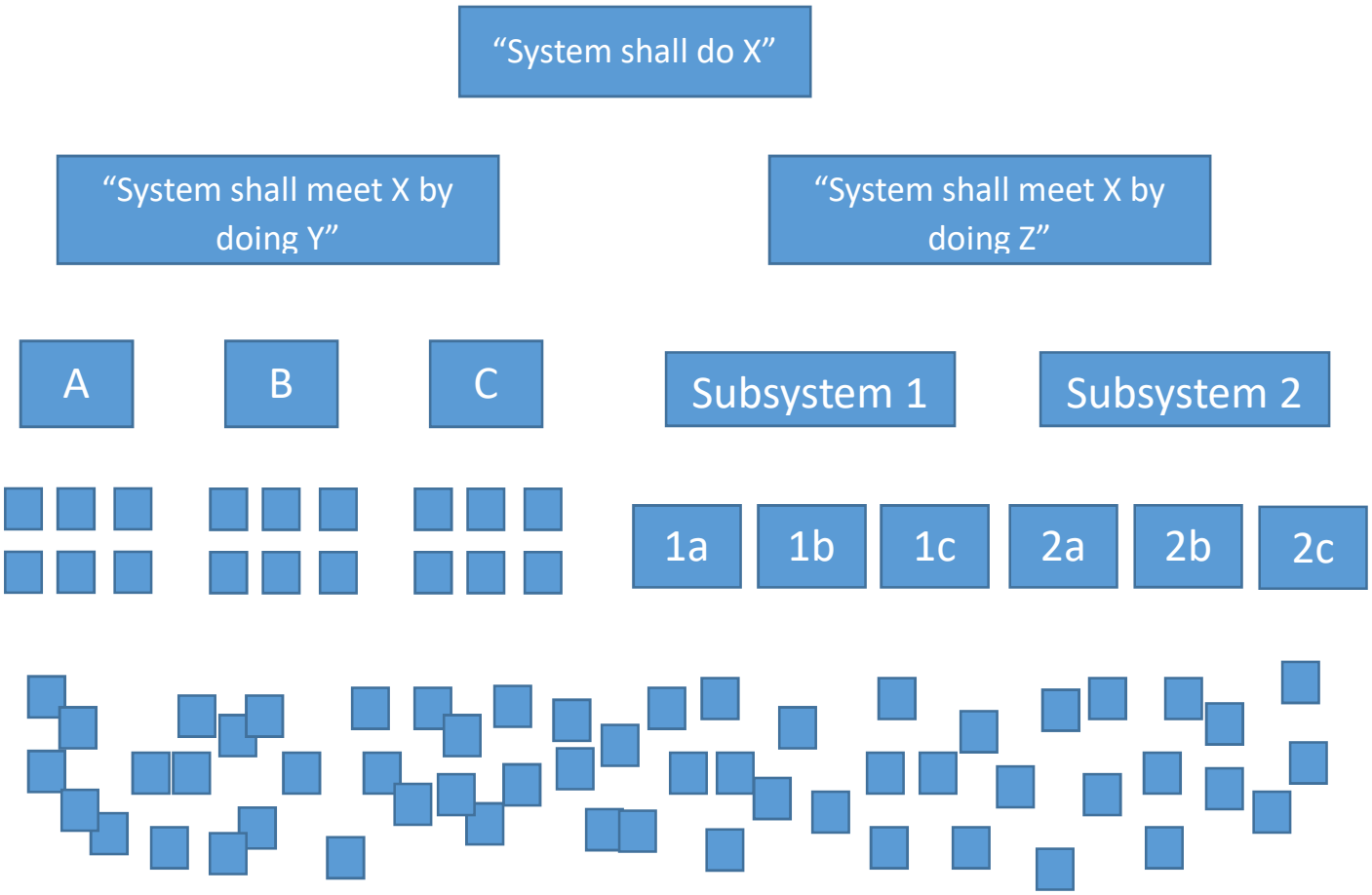
Each new requirement spawns further "child" requirements that elaborate on how the "parent" requirement will be met, with cascading effects.



Some requirements may be met by entirely new subsystems, which may be self-contained or tightly integrated with the larger system. Development of these subsystems may be undertaken by a separate, subordinate team at the same company, or it may be outsourced to another company entirely. An existing subsystem may already be available to purchase off the shelf, in which case it may need to be modified or its characteristics may drive the requirements of the system it is being integrated into instead of the other way around.



A system as complex as a spacecraft can very quickly involve thousands of requirements, many of which are interconnected, and even slight changes can have significant ripple effects. This can easily be overwhelming for small or poorly-organized teams and with many different groups working on different aspects of the same system, a common reference point for requirements becomes critical for effective communication and collaboration.



Besides keeping track of the various parent-child relationships, IBM Rational Doors also allows users to link documentation to each individual requirement including design notes, testing procedures and results, and certifications that a given requirement has indeed been met. This is called the verification and validation (V&V) process, and for several weeks I was tasked with linking these various forms of documentation with their respective requirements.

Algorithm Requirements and Challenges

As previously stated, the bulk of my work consisted of developing an algorithm that would receive telemetry from both the CV and MEV and determine the constantly varying alignment error between them. The challenges to doing this successfully were numerous: firstly, the vehicles lack the means of communicating with each other directly. Each spacecraft sends telemetry to its respective MOC separately, and then the CV's operators forward the flight data to the MEV's MOC at Northrop Grumman. The operators there use the algorithm to determine what adjustments need to be made to the MEV's attitude to ensure that the CV is properly oriented. Despite the number of steps, the data transfer and analysis happen nearly instantaneously and Northrop Grumman's operators merely need to issue the appropriate commands to the MEV. This does, however, mean that the human element needs to be accounted for in the design of the software, for instance minimizing the amount of work a user has to do to use the algorithm while simultaneously allowing for certain parameters to be adjusted for testing purposes. The code must also operate efficiently enough that it can process the incoming telemetry as it comes in and not bottleneck the flow of information.

An additional challenge was that the two streams of data being compared to each other were radically different. The data used to determine each spacecraft's orientation came from different sets of sensors, which resulted in information that was not immediately equivalent and that was transmitted in different formats. Further complicating things were the different sample rates of the two sets of telemetry, meaning that orientation data for one vehicle over a given interval would have a very different number of entries than the data for the other vehicle over the same interval. Properly comparing the two sets therefore required interpolating the data such that the orientation of one vehicle could always be determined for any corresponding data point for the other.

Lastly, the initial development of this algorithm was done in Matlab, a program I had never used before, and the sum of my programming experience was from some very basic robotics projects in high school. My design experience prior to this project focused on manufacturing, with some academic and practical experience in computer aided design and design for manufacturability, as well as some coursework in systems engineering. While some

overarching concepts regarding good design practices certainly did apply, nearly everything I needed to know how to do I learned on the spot, and I was fortunate enough to be surrounded by a number of experienced engineers from whom I could learn.

Technical Information

Before the two spacecraft's orientations could be compared, they needed to be in a common reference frame. The reference frame used in this case is known as local vertical local horizontal (LVLH) which is a rotating reference frame centered on the Earth. The X-axis points along the spacecraft's velocity vector and is tangential to its orbit, the Y-axis is normal to the spacecraft's orbit, and the Z-axis points at the center of the Earth. These axes correspond to roll, pitch, and yaw, respectively.

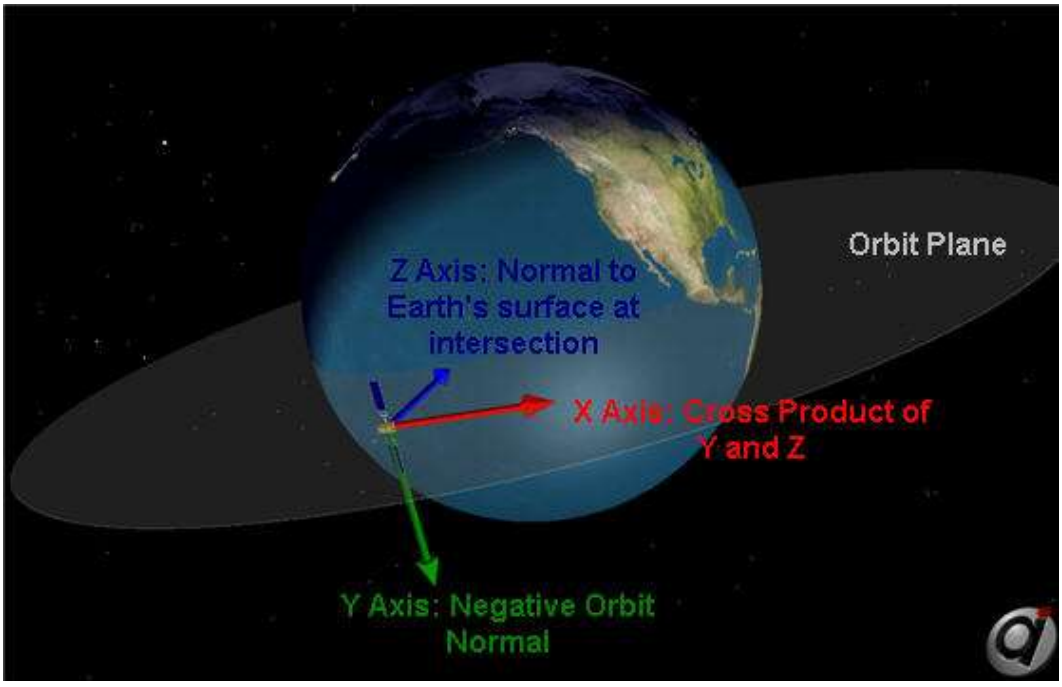


Figure 2: The LVLH reference frame

The CV's orientation is determined by combining data from multiple sensors. Measuring roll and pitch is relatively straightforward: an infrared camera known as an Earth sensor array (ESA) points directly at the Earth and measures the distance from the center of the Earth to the center of its field of view. A vertical offset corresponds to the vehicle's roll, and a horizontal offset to pitch.

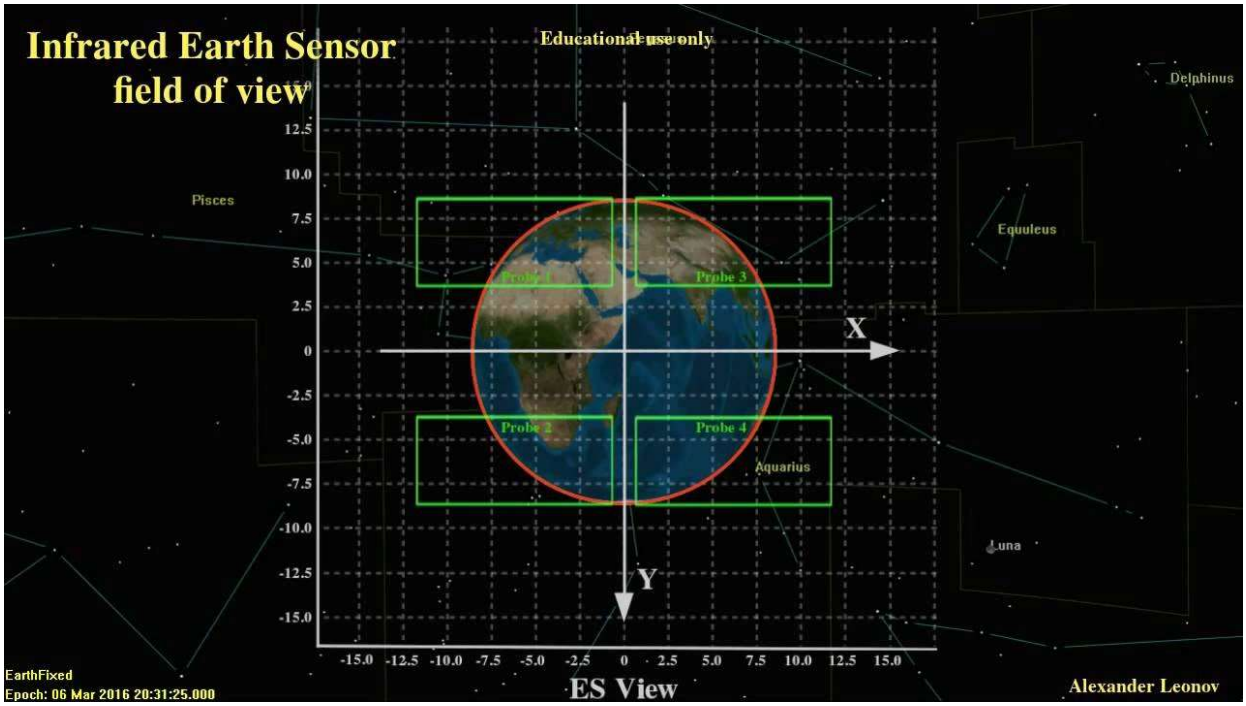


Figure 3: A representation of an ESA's field of view with LVLH axes. The Z-axis points toward the Earth.

Measuring yaw becomes more complicated. In this case the data comes from a digital sun sensor (DSS) which points in the positive X-axis in the vehicle's direction of travel. The greatest limitation is that the DSS can only return data when the sun is in its field of view, which occurs for about 4.5 hours in a single orbit (because the satellite is in GEO, it completes its orbit in exactly 24 hours). Furthermore, the sensor is only accurate when the sun is centered in its field of view, meaning the values received can only be taken at face value for several minutes each day. For the rest of the data gathered, the sensor values must be summed with sun declination figures that match the given data packet's timestamp. Sun declination is the angular distance of the sun's rays from the equator and accounts for the tilt of Earth's axis of rotation. Fortunately, these values are provided separately and do not need to be calculated by the algorithm. By adding the sun declination data to the raw telemetry from the DSS, I was able to determine the CV's true yaw and feed it into the algorithm.

In addition to not returning any data when the sun is not in view, the DSS also collects data at half the sample rate of the ESA. This means that every other packet sent by the satellite does not include any yaw information at all.

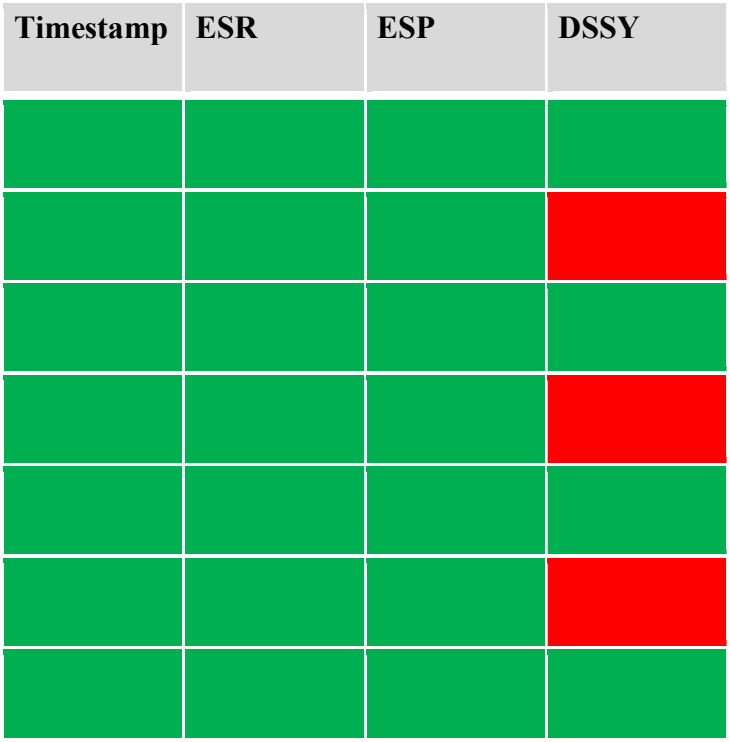


Figure 4: A representation of the roll, pitch, and yaw data sent from the CV. The boxes in red signify absent data.

At first these two problems seemed like they would require independent and convoluted workarounds, but a simple solution ultimately addressed both. It was determined that the sample rate of the DSS was sufficient for a reasonably reliable data set, so every entry that did not include data for all three axes was omitted. For the time period when the DSS did not return any data at all because the sun was outside its field of view (during most of the CV's orbit), the most recent yaw values would be used. While certainly imperfect, the other two axes could still be adjusted and this approach still allowed for bias correction within acceptable tolerances.

The MEV uses an altogether different method of determining its orientation. As with most modern spacecraft, the MEV is equipped with a star tracker that optically measures the positions of certain stars with respect to the vehicle. This information is compared to data from a star catalog to determine the spacecraft's position with much greater precision and consistency than the Earth and sun sensors. The most commonly used star catalogs are compiled by the United States Naval Observatory in Washington, DC, though catalogs from other sources do exist and are used elsewhere in the world. The MEV transmits this data in the form of quaternions which give an orientation vector. Quaternions are a four-element matrix which

include an X, Y, and Z value for the spacecraft's orientation in LVLH as well as a scalar 1 or 0 value which avoids a control singularity.

Scalar	X	Y	Z

Figure 5: A representation of the data in a single quaternion sent from the MEV.

This telemetry can be transmitted many times faster than the CV's telemetry, at up to 10 hertz. Transmitting data at this rate uses a lot of power, so this high rate is only typically used when high-resolution telemetry is needed, for instance during launch, orbital maneuvers, or when operating close to a satellite and extra care is required (this is referred to as RPOD, which stands for rendezvous, proximity operations, and docking). When none of these events are happening, the MEV transmits at significantly lower rates which suffice for mundane station keeping.

Design Considerations and Method

To summarize the criteria for the algorithm based on the aforementioned context, the algorithm needed to be able to work with varying sample rates to function consistently at any level of activity, without extra input from an operator. This would also have to be done with raw input that included multiple void data entries, the result of sensor windows, corrupted data packets, or inconsistent sample rates between multiple sensors. The operations done on the data needed to be able to handle different formats, particularly with regards to timestamps. Then there was the operational context to consider; during normal use, the algorithm was intended to run constantly using live telemetry transmitted from the MEV and the client's MOC, referred to as "online" operation. However it needed to also be able to operate in an "offline" capacity, where sets of data would be fed into it (likely from an Excel spreadsheet) for after-the-fact analysis. To clarify, this would be done by another instance of the algorithm, the same program would not have to handle both tasks simultaneously. Lastly, in an effort to make the algorithm as useful as possible and not needlessly overburden the flight controller using it, all of this had to be accomplished with minimal necessary inputs (additional optional inputs were both acceptable and desirable, for those instances when the flexibility proved necessary) as well as maximum computational efficiency to avoid informational bottlenecking.

This last design consideration proved to be a particular point of vulnerability given my lack of programming experience. In order to properly test each iteration of the algorithm as I updated things, I had to generate theoretical flight data for both vehicles to run through the program. This in turn meant performing detailed statistical analysis on sample data provided by the customer from a given period in their satellite's orbit. Because MEV had yet to fly, no such flight data existed on our end, so I had to use data generated by flight simulations used by other offices. To further test each iteration of the algorithm, I wrote separate scripts that generated new values based on the data received from the customer and the MEV simulation. These included the statistical outliers present in the original numbers, which the algorithm needed to be able to recognize and filter out.

This process required an understanding of what is considered "good" and "bad" data. These definitions are entirely dependent on context, but generally they include characteristics

such as noise, precision, and accuracy. For this project, accuracy was not a concern because the instruments and data already met the positioning and attitude control requirements of the customer, and I was merely analyzing telemetry which was already considered acceptable. Instrument precision did come into play, since the MEV's star tracker returned far more precise values than any of the CV's sensors. This meant that some information was lost when comparing telemetry from the two vehicles, but the resulting rounding errors fell well within the customer's desired tolerances. Given the interference inherent in transmitting data over 36,000 km of radiation-filled space and planetary atmosphere, however, the data fed into the algorithm had a substantial amount of noise. My approach to eliminating outliers involved taking a rolling average of a certain number of previous entries and if the latest entry deviated too far from that mean, then it would be discarded. If this degree deviation was consistent (and therefore keep throwing out packets), it would indicate significant change in attitude of one or both vehicles and would be both noticed and addressed by the MOC staff. Because the telemetry sent from both vehicles arrived as a packet, complete with values for all axes as well as the corresponding timestamp, I had to assume that if one value was suspect, the entire packet was corrupted and had to be thrown out. This also ensured that every category of information had the same number of entries (i.e. there where there same number of roll, pitch, and yaw entries for the CV so that there were no issues with mismatched values and timestamps). This was also how I was able to deal with the void entry problem from the CV sensors' different sample rates; if there wasn't a corresponding entry from the DSS, the ESA values weren't useful for the algorithm's purpose and could be safely discarded. While often painstaking, this process forced me to think about why any particular set of data looked the way it did and what conclusions could be drawn from it beyond the explicitly stated values. I also had to consider how any change to the data rippled through the algorithm, particularly with regards to calculating the rolling average and changing the length of the data sets I was working with by removing bad entries. Many hours were wasted trying to figure out why something wouldn't work when the reason was simply that a line of code was expecting a certain specific number of entries and I had written it to be too rigid.

Additionally, I spent countless hours reading through the documentation for particular commands and arguments, initially just to find the command that I needed to execute a particular task with the information I had on hand. Something that I didn't anticipate, however, was that all commands are not created equal and depending on how the back end worked one could

accomplish the same thing with significantly fewer calculations than another. Given the previously mentioned importance of minimizing the algorithm's runtime to avoid slowing down the flow of information, this was a critical piece of knowledge to have and I learned a lot from the other engineers around me about other best practices for maximizing the efficiency of the code. Chief among these was the process of vectorizing all of the data from both vehicles. Previously, I was handling the telemetry from each vehicle line by line for each packet…the thinking was to keep all the values for each axis and their shared timestamp together to avoid anything getting mixed up. The cost of this approach was that parts of the algorithm had to run in a loop, once for every single packet. There were over 56,000 packets. While theoretically acceptable for live data that would be arriving slow enough for any computer to keep pace, this would unnecessarily burden the processor that would almost certainly be handling multiple other tasks simultaneously on an operator's workstation. It was also completely unacceptable for bulk analysis of a given interval for offline operation, since just processing the 4.5-hour window's worth of telemetry I was given took so long that I would go on a lunch break and the algorithm would still not be finished by the time I returned. At this point I was shown how organizing each set of data (roll, pitch, yaw, timestamps, etc.) as a separate vector, defined as a one-by-X matrix where X was the number of entries that were fed into the algorithm, eliminated the need for loops and made the program thousands of times faster.

Throughout this process I also had to consider what each piece of the program I was writing did on its own, and was functionality would be necessary for the final version as opposed to development-specific scripts. A fair amount of the work I was doing was analyzing the provided flight data and writing scripts to generate new data that fit the mission profile, all of which was just to test the main algorithm to make sure it would function as intended with fresh, variable telemetry. None of this would be needed in the MOC where flight controllers would have all the live numbers they needed, but it was critical to ensure that the program would do its job even if it received unexpected telemetry. I also had to ensure that each subscript was compartmentalized from the others, both those in the main algorithm as well as those merely intended for development. This was to ensure both that the algorithm could function independently of the development scripts and that none of its elements would fail due to an error in another (i.e. the noise filter subscript would still filter out bad data packets regardless of an

error in interpolating the CV telemetry with that of the much higher resolution MEV instruments).

Another aspect of developing this algorithm that required a lot of on-the-spot learning on my part was the mathematical element. In order to properly compare their respective attitudes to one another, both the telemetry for both vehicles needed to be in a common reference frame, in this case the LVLH reference frame. The actual flight data the algorithm would be working with would already be in LVLH, and this was certainly true for the sample CV telemetry I had been given to work with since it, too, was past flight data. The MEV values, on the other hand, were not; the simulation they were drawn from was using a different reference frame, meaning that the values I received had to be converted. Despite some linear algebra work I had done in classes, the task at hand greatly exceeded my experience and I had to consult a number of online texts as well as some of my colleagues to fill in my knowledge gap. The first step was create a new quaternion matrix for the CV telemetry to match that from the MEV's star tracker. There was no script available to fully automate this task so I worked with another engineer to understand exactly what steps to take, including working with the limitations of the code. The basic relationship between Euler angles and quaternions is illustrated below:

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \arctan\frac{2(q_0 q_1 + q_2 q_3)}{1-2(q_1^2+q_2^2)} \\ \arcsin(2(q_0 q_2 - q_3 q_1)) \\ \arctan\frac{2(q_0 q_3 + q_1 q_2)}{1-2(q_2^2+q_3^2)} \end{bmatrix}$$

Where q0 is the quaternion scalar, and q1, q2, and q3 correspond to the X, Y, and Z values in Figure 5, respectively. Unfortunately, the arctan and arcsin functions in MATLAB did not return the full set of results from this calculation, so further research in the function documentation was required to find commands that did produce the full range of solutions. Furthermore, the formula above had to be modified for the new MATLAB functions being used.

With all the telemetry in the same format, the simulated MEV data needed to be converted to LVLH. At first, this seemed like a simple matrix transformation, but despite the existence of an internal company library of conversion scripts, nothing there fit the particular needs of the conversion at hand. This forced me to take a step back and try and visualize both the

two reference frames I was attempting to convert between as well as those included in the code library. After quite a bit of puzzle-solving (and some much-needed peer review) I was able to jury rig the necessary transformations by first converting to another, otherwise-irrelevant frame and then finally to LVLH.

Documentation and Handoff

The scope of this project extended beyond my time at the company, so I had to keep careful notes to thoroughly document everything I was working on and how it all worked so that someone else could continue where I left off or modify things as needed. Anyone with coding experience can attest to the importance of including comments in the code itself, but I also wrote numerous design notes to regularly document both my weekly progress as well as explaining the rationale for all of my decisions. Areas for improvement in the algorithm included reducing the overall runtime while continuing to ensure modularity. Separate tasks to be done in the future included developing a user interface to simplify the use of the algorithm, as well as converting all of the code into the software used in the MOC itself. This was a new, strange, but ultimately valuable experience, since as a student one is typically involved in a homework assignment or group project from start to finish, but in industry handoffs are relatively common and this affects how one does one's work.

Role of the Axioms

I became aware of axiomatic design shortly after beginning work on this project, but it quickly became clear that despite its origins as a design philosophy for manufactured goods, the principles involved would be of great benefit to a piece of software such as the one I was creating. To briefly summarize its basic tenets, axiomatic design posits that there are two fundamental axioms that contribute to a given design's success. The first is the independence axiom, which dictates that the functional requirements of a given system be independent of one another. The second is the information axiom, which encourages the minimizing of the information content of a system. In this context, "information" has a rather unique definition, but in plain English the second axiom governs the simplicity, and therefore the reliability, of a given design while the first compartmentalizes it so a change or failure in one area does not have significant ripple effects. These are rather universal concepts, as applicable to designing a car transmission as to planning a mother-in-law's birthday, so it stands to reason that software development would also benefit.

With regards to this algorithm, the independence axiom was met by the modularity of the program. There was a limit to the degree that all the components could be decoupled from one another, given that certain calculations and functions operated on the output of another command, but to the degree that it was possible every effort was made to allow each part of the data handling and analysis to operate independently. By far the more important of the two, however, was the information axiom. This had a direct impact not only on the complexity of the algorithm, but also on its runtime, a quantifiable measure of its efficiency. In particular, it involved organizing the code in a such a way as to minimize the number of variables that needed to be defined, not only reducing the burden on the end user but also allowing for some flexibility in certain parameters. For instance, one of my early mistakes was to write commands that expected a specific matrix length. Because the entire purpose of the noise filter was to eliminate bad entries, there was no way to predict exactly what length a data matrix would be, but by setting up the function to simply refer back to the actual length of the data set rather than a fixed value, a required input was eliminated as well as a potential point of failure. There were instances when these efficient practices undermined the independence axiom, since a particular approach would affect multiple elements of the algorithm fulfilling multiple functional

requirements, but the benefits were determined to outweigh the potential downsides and a conscious decision was made to prioritize the efficiency and reliability of the algorithm over modularity whenever the two conflicted.

Axiomatic Design in Software

One of the takeaways from this project regarding the use of axiomatic design principles in software is that while the concepts certainly apply, they do not always do so in the same way that they would for a mechanical design and can easily become mutually exclusive in certain circumstances. The axioms are therefore best understood not as hard lines dictating the minutia of how a specific design element, in this case a particular command or line of code, ought to function, but instead as a pair of general truisms that should guide the planning and development of the overall system from its conception. The assumptions behind them do hold true for software as well as they do for anything else. If the functionality of a given piece of software can be conceptually broken up into distinct roles ("the software does X, Y, and Z"), then the ability of the software to meet each requirement ought not to depend on its ability to meet any of the others. A complicating factor, however, would be a direct relationship between one or more of those requirements, particularly if they are sequential. This is analogous to solving a physics problem where one error early on cascades throughout the calculations despite otherwise flawless methodology, leading to an incorrect answer. This is likely to prove unavoidable in most software development, and should be carefully considered when defining the program's requirements from the beginning.

If the independence axiom may prove problematic, the importance and benefit of the information axiom cannot be overstated. The entire purpose of computing is to handle an enormous quantity of information at speeds far exceeding the capacity of even a large group of people, and while that may sometimes obscure the need for efficiency it in fact makes it even more important since even a small difference in a program's runtime will have an outsized impact on the amount of work done in a given time period. Software's simplicity is also directly related to the oft-overlooked human interface, both increasing the ease of use for an end user as well as for the development and maintenance teams. Coordinating efforts when building extremely lengthy and complex software is a dismal undertaking, matched only by the difficulty of editing and updating software written by since-departed developers, but an equally important consideration is that software that accomplishes the same tasks with fewer elements also tends to work better. The relationship between the two goes beyond mere convenience, since just as

minimizing the code reduces the possibilities of a computational error or missing required information, it also reduces the chance for human error.

Accomplishing this goal of minimizing the information content can consist of using optimal functions to fulfill certain tasks that are both computationally efficient and require the fewest possible inputs. Every input is either another set of data to handle or another variable to be defined (or potentially misdefined) by the user, and whenever possible these should be referenced back to values that the program can calculate for itself (or better yet has already calculated elsewhere). The number of outside dependencies should also be minimized. Most development teams have their own libraries of scripts, subscripts, and individual functions that are wonderful for avoiding the need to duplicate work, but if they are not fully defined within the software and are merely referenced from an outside source such as a network drive, then the entire software may cease working if the connection to that network drive is interrupted. In this example, a good solution would be to copy the subscripts into the working folder of the main program and reference them locally rather than clutter the latter with numerous lengthy function definitions, but this goes to show every added element or subrequirement in a system must be considered as a potential point of failure.

Bibliography

[1] Clark, S. (2016). *Orbital ATK books proton rocket for first commercial satellite servicing mission.* Retrieved from https://spaceflightnow.com/2016/10/14/orbital-atk-books-proton-rocket-for-first-commercial-satellite-servicing-mission/ (Accessed Aug 5, 2019)

[2] *Attitude reference frames.* Retrieved from https://ai-solutions.com/_help_Files/attitude_reference_frames.htm (Aug 5, 2019)

[3] *GEO satellites: Infrared earth sensor.* Leonov, A. (2016).[Video/DVD] YouTube.

[4] Huerta, J. *Introducing the quaternions.* Fullerton College. Retrieved from http://math.ucr.edu/~huerta/introquaternions.pdf (Accessed Aug 5, 2019)

[5] Technical overview of central body reference frames (coordinate systems). (2019). Retrieved from http://help.agi.com/stk/index.htm#stk/referenceFramesCentralBodies.htm (Accessed Aug 5, 2019)

[6] W. G. Breckenridge, (1979) "Quaternions proposed standard conventions," NASA Jet Propulsion Laboratory, Technical Report. (Accessed Aug 5, 2019)

[7] NASA Mission Planning and Analysis Division. "Euler Angles, Quaternions, and Transformation Matrices" (PDF). NASA. (Accessed Aug 7, 2019)

[8] Janota, A; Šimák, V; Nemec, D; Hrbček, J (2015). "Improving the Precision and Speed of Euler Angles Computation from Low-Cost Rotation Sensor Data" (Accessed Aug 11, 2019)

[9] MATLAB documentation. (2019). Retrieved from https://www.mathworks.com/help/matlab/index.html (Accessed Aug 11, 2019)

[10] Sarbu, I., & Sebarchievici, C. (2017). *Solar heating and cooling systems* Elsevier Inc. Retrieved from https://www.sciencedirect.com/book/9780128116623/solar-heating-and-cooling-systems (Accessed Aug 12, 2019)

[11] Brown, C. (2016). *Axiomatic design for MQPs.* (Accessed Aug 15, 2019)

[12] Everett, D. (2011). *Space mission engineering: The new SMAD* Space Technology Library.

[13] Wilson, J. *Hohmann Transfers* [Online]. Available at: http://jwilson.coe.uga.edu/EMAT6680Fa05/Bacon/hohmanntransfers.html (Accessed Aug 16, 2019)