Reusable and Testable Attitude Determination Software for CubeSats based on Low-Cost Sensors

A Senior Project by

Luc Bouchard

Advisor, Dr. Matt Moelter

Department of Physics, California Polytechnic University SLO

April 8, 2019

Contents

List of Figures

1 Introduction

Attitude determination and control (ADC) software discerns and corrects the orientation of spacecraft on orbit. It is an essential component of many space missions; however, it is often tricky to implement due to its mathematical complexity and testability issues. In this work, we describe Kalman-filter based attitude determination software for Exocube 2, a mission in development at PolySat, the Cal Poly CubeSat research lab, and we provide a software-based testing framework to validate the ADC software. Furthermore, we show how this software can be easily reused on all PolySat spacecraft, reducing risk and development time on future missions. This attitude estimation solution relies on low-cost instrumentation that is already part of the generic PolySat bus, magnetometers and solar angle sensors, enabling even simple missions to perform attitude estimation with little additional development.

1.1 Background

CubeSats are 10 cm cube spacecraft that began as a collaborative effort between Dr. Jordi Puig-Suari of Cal Poly and Dr. Bob Twiggs of Stanford University in 1999. Puig-Suari and Twiggs sought to reduce small satellite development time and cost by providing a rigorous mechanical standard that, if properly met, ensured satellites could fly on any launch vehicle with a CubeSat deployer at no risk to other payloads on the same launch (Cal Poly CubeSat Lab, 2014).

The Cal Poly CubeSat Lab, also known as PolySat, has been developing CubeSats since 2001, steadily advancing its technical capability with each completed mission. However, one capability that has eluded PolySat since its founding is on-orbit attitude estimation. An attempt to support this capability was made for PolySat's ExoCube mission (Mehrparvar, 2013); however, ExoCube suffered an antenna deployment failure that severely limited potential operations, causing ADC validation to be descoped (Saunders, 2016). Futhermore, the ADC software developed for ExoCube was specific to the mission, and porting the software to different spacecraft would require significant development effort.

1.2 Exocube 2

Because of ExoCube's aforementioned antenna deployment failure, a reflight of the ExoCube mission is under development, known as ExoCube 2. Just like the original ExoCube, ExoCube 2 will fly a mass spectrometer suite in low earth orbit (LEO) to measure neutral and ionized molecule densities in the exosphere and thermospehere. This information is essential to physics-based weather models, and similar measurements have not been made since the Dynamics Explorer 2 mission that launched in 1981 (Sellers, 2013).

Ions and neutral molecules will impact Exocube 2's ion neutral mass spectrometer (INMS) detector at a location dependent on their incident velocity, mass, and charge. Because Exocube 2 is making measurements while orbiting, it will sample molecules moving at random velocities within the atmosphere. Thus, incident species will create a distribution on the detector that results from their velocity distribution. Since kinetic gas theory provides a model for the velocity distribution of molecules of varying masses at a particular temperature, the mass of the incident species can be determined from the measured velocity distribution. Of course, this assumes that the INMS is exposed to the full velocity distribution for all species.

To sample the full velocity distribution, ExoCube 2 will point the aperture of the INMS along the orbital velocity vector of the spacecraft (ram facing direction). If the vector normal to the plane of the aperture is at an angle relative to the ram facing direction, the measured velocity distribution will be skewed, and the species could be misidentified (Sellers, 2013). To torque the spacecraft into the ram facing direction, the attitude of the spacecraft must first be determined. The algorithm behind this determination step is the topic of this paper. An estimate of the attitude will be made using magnetometers and solar angle sensors alone. Analysis shows that to obtain useful data from the instrument, its attitude must be known to within $\pm 5°$ of the true attitude (Mehrparvar, 2013). The determination algorithm described here must meet this requirement.

2 Spacecraft Dynamics

2.1 Reference Frames

Three reference frames will be utilized throughout this paper, earth-centered inertial (ECI), local vertical local horizontal (LVLH), and body. Their definitions follow, and Figure 1 shows 2D illustrations of the frames.

2.1.1 ECI

The earth-centered inertial frame, also known as the geocentric equatorial frame, is a non-rotating right handed Cartesian coordinate system with its origin at the center of the Earth. The fundamental plane (X_{ECI}, Y_{ECI}) consists of the Earth's equatorial plane with the principal direction (X_{ECI}) pointed at the vernal equinox. The right handed coordinate system is completed by Z_{ECI} which is orthogonal to the Earth's equatorial plane and coincides with the Earth's axis of rotation (Curtis, 2010, p. 204).

2.1.2 LVLH

The LVLH frame is an Earth centered, orbit based, and rotating frame. The origin is centered at the center of mass of the spacecraft with the Z_{LVLH} axis pointing towards the center of the Earth (nadir). The Y_{LVLH} axis is aligned with the negative of the orbital angular momentum vector (cross-track). The right handed coordinate system is completed by X_{LVLH} which is aligned with the velocity vector (in-track/ram) direction for circular orbits.

2.1.3 Body

The body frame is aligned geometrically with the spacecraft. The orthogonal set of axes is defined with the origin at the geometric center of the 3U structure with X_{Body}, Y_{Body}, and Z_{Body} axes normal to the X, Y, and Z sides of the CubeSat as defined by the CubeSat Standard.

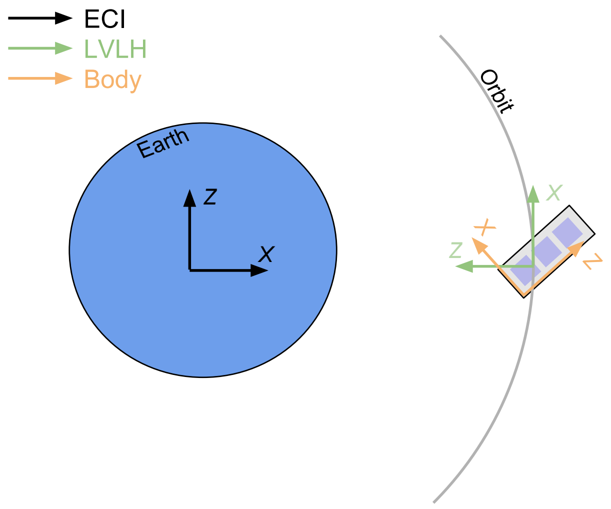

Figure 1: 2D depictions of all three reference frames.

2.2 Quaternions

This paper will use quaternions to represent spacecraft attitude. Quaternions are four element vectors that can be used to represent a frame rotation. Thus, the spacecraft's attitude will be represented as the frame transformation from ECI to body, or LVLH to body. In this paper, the last element of the four element vector will be the scalar component, and the first three will be the vector component. They will be notated as follows:

$$q = [q_i, q_j, q_k, q_r] = [\vec{q}, q_r] \tag{1}$$

To represent attitude, the quaternion elements will be defined as:

$$\begin{aligned} \vec{q} &= \hat{u}\sin(\tfrac{\theta}{2}) \\ q_r &= \cos(\tfrac{\theta}{2}) \end{aligned} \tag{2}$$

where θ is the angle with which to rotate the frame, and $\vec{u}$ is a unit vector to rotate the frame around. Multiplication of two quaternions, a and b is defined by the Hamilton product:

$$a * b = \begin{pmatrix} a_r b_i + a_i b_r + a_j b_k - a_k b_j \\ a_r b_j - a_i b_k + a_j b_r + a_k b_i \\ a_r b_k + a_i b_j - a_j b_i + a_k b_r \\ a_r b_r - a_i b_i - a_j b_j - a_k b_k \end{pmatrix} \tag{3}$$

The conjugate of a quaternion q, q^{-1}, is defined as:

$$q^{-1} = [-\vec{q}, q_r] \tag{4}$$

To apply the transformation described by a quaternion q to a vector v, the following must be performed:

$$v' = q * [\vec{v}, 0] * q^{-1} \tag{5}$$

The transformed vector is given by the vector component of the quaternion v'. Note that if two quaternions, a and b, are multiplied (i.e. $a * b$), their product represents the rotation by b followed by the rotation by a.

2.3 Rigid Body Dynamics

We start with Euler's equation for rotational motion, which states that:

$$\dot{\vec{L}} + (\vec{\omega} \times \vec{L}) = \vec{M}_{net} \tag{6}$$

where $\vec{L}$ is the angular momentum vector of the body in the body frame, $\vec{\omega}$ is the angular velocity vector of the body frame with respect to an inertial frame, and $\vec{M}_{net}$ is the net torque on the body in the same inertial frame. Note that $\vec{L}$ is given by:

$$\vec{L} = I\vec{\omega} \tag{7}$$

where I is the inertia matrix of the body. If the body frame is aligned with the principle axes of the body, then the inertia matrix becomes diagonal. Assuming the principle moments of inertia are given by λ_{xx}, λ_{yy}, and λ_{zz}, equation (6) can be converted into component form:

$$\begin{aligned} \lambda_{xx}\dot{\omega}_x - (\lambda_{yy} - \lambda_{zz})\omega_y\omega_z &= M_x \\ \lambda_{yy}\dot{\omega}_y - (\lambda_{zz} - \lambda_{xx})\omega_z\omega_x &= M_x \\ \lambda_{zz}\dot{\omega}_z - (\lambda_{xx} - \lambda_{yy})\omega_x\omega_y &= M_x \end{aligned} \tag{8}$$

2.4 Quaternion Kinematics

The time derivative of a quaternion can be related to the angular velocity of the target frame through the following:

$$\frac{dq}{dt} = \dot{q} = \frac{1}{2}\begin{pmatrix} 0 & \omega_z & -\omega_y & \omega_x \\ -\omega_z & 0 & \omega_x & \omega_y \\ \omega_y & -\omega_x & 0 & \omega_z \\ -\omega_x & -\omega_y & -\omega_z & 0 \end{pmatrix} q \tag{9}$$

3 Kalman Filter

In determining the attitude of a spacecraft, several challenges must be overcome to get a stable and consistent attitude estimate. The most obvious of these challenges is noise in the magnetic field and solar angle measurements. The spacecraft's magnetometers, while being limited in resolution for one, are also constantly being exposed to magnetic field fluctuations generated by surrounding electronics. On top of this, for magnetic field measurements to be useful in an attitude estimate, they must be compared to ECI magnetic field reference vectors for the given location. Since these reference vectors are generated using approximate models like IGRF, they have their own associated uncertainty. Solar angle sensors suffer from similar uncertainties, but also have the added challenge of being unusable when the sun is eclipsed by the earth. Thus, the determination algorithm must be able to accommodate the spontaneous loss of solar angle measurements.

To solve these challenges, an Extended Kalman Filter (EKF) was designed and implemented. Kalman Filters use Bayesian statistics to combine noisy sensor readings with a mathematical model of the system in question (Labbe, 2018).

3.1 Dynamics Model

The filter used in this work employs a seven-dimensional state vector, $\vec{x}$ given by:

$$\vec{x} = (\omega_x, \omega_y, \omega_z, q_r, q_i, q_j, q_k) \tag{10}$$

where the angular velocities are of the body frame with respect to ECI and the quaternion values rotate from ECI to body. We assume that the dynamics of the system can be described as:

$$\dot{\vec{x}} = \vec{f}(\vec{x}) + \vec{w} \tag{11}$$

where $\vec{f}(\vec{x})$ is a potentially nonlinear function of the attitude state vector, and $\vec{w}$ is a vector that represents any force contributions that are not modeled or are simply random (for example, a spontaneous increase in solar pressure). This equation of motion can be propagated using Euler or RK methods. The $\vec{w}$ contribution is obviously ignored when propagating attitude state.

The system dynamics can also be described in state-space (discrete-space) as:

$$\vec{x}_k = F\vec{x}_{k-1} + \vec{w} \tag{12}$$

where F is the state transition matrix, as it transitions the system from the $k-1$ state to the k state. It is given by the matrix exponential. In this work, the predict step of the Kalman filter is accomplished by propagating the dynamics using the Euler method. Using equations 8 and 9, we find the following equation for $\vec{f}(\vec{x})$:

$$\dot{\vec{x}} \approx \vec{f}(\vec{x}) = \begin{pmatrix} \dot{\omega}_0 \\ \dot{\omega}_1 \\ \dot{\omega}_2 \\ \dot{q}_s \\ \dot{q}_i \\ \dot{q}_j \\ \dot{q}_k \end{pmatrix} = \begin{pmatrix} \lambda_{xx}^{-1}(\lambda_{yy} - \lambda_{zz}) * \omega_y * \omega_z \\ \lambda_{yy}^{-1}(\lambda_{zz} - \lambda_{xx}) * \omega_x * \omega_z \\ \lambda_{zz}^{-1}(\lambda_{xx} - \lambda_{yy}) * \omega_x * \omega_y \\ \frac{1}{2}(\omega_z q_j - \omega_y q_k + \omega_x q_r) \\ \frac{1}{2}(-\omega_z q_i + \omega_x q_k + \omega_y q_r) \\ \frac{1}{2}(\omega_y q_i - \omega_x q_j + \omega_z q_r) \\ \frac{1}{2}(-\omega_x q_i - \omega_y q_j - \omega_z q_k) \end{pmatrix} \tag{13}$$

Note that the above assumes the body frame is aligned with the spacecraft's principle axes and that there are no control torques. Since this paper is focused on determination, we will ignore control torques. The state transition matrix, F, for this system is approximately given by:

$$F \approx I + \frac{\partial \vec{f}(\vec{x}_{k-1})}{\partial \vec{x}} dt \tag{14}$$

where I is the identity matrix and dt is the differential time between the $k-1$ state and the k state.

3.2 Measurement Model

The measurement vector of the EKF, $\vec{z}$ is given by:

$$\vec{z} = (b_x, b_y, b_z, s_x, s_y, s_z) \tag{15}$$

where $\vec{b} = (b_x, b_y, b_z)$ is the magnetic field measurement and $\vec{s} = (s_x, s_y, s_z)$ is a unit vector that points towards the sun, which is provided by the solar angle sensor, all expressed in the body frame. We assume measurements made by the spacecraft at a particular attitude can be described by the following:

$$\vec{z} = \vec{h}(\vec{x}) + \vec{v} \tag{16}$$

where $\vec{h}(\vec{x})$ computes the expected sensor readings from the state vector, and $\vec{v}$ is a random vector that describes uncertainties in sensor readings. The measurement model, $\vec{h}(\vec{x})$, is given by:

$$\vec{z} \approx \vec{h}(\vec{x}) = \begin{pmatrix} R_{body\leftarrow eci} & 0_{3x3} \\ 0_{3x3} & R_{body\leftarrow eci} \end{pmatrix} \vec{z}_{ECI} \tag{17}$$

where $R_{body\leftarrow eci}$ is the rotation matrix that rotates from ECI to body, 0_{3x3} represents the 3x3 zero matrix, and $\vec{z}_{ECI}$ is the measurement vector in ECI given by some reference model that depends only on the position of the spacecraft and not the attitude. The measurement matrix, H, is defined as the matrix that satisfies the following:

$$\vec{z} = H\vec{x} + \vec{v} \tag{18}$$

It is approximately given by:

$$H \approx \frac{\partial \vec{h}(\vec{x})}{\partial \vec{x}} \tag{19}$$

The equations described above are used in the standard EKF algorithm (Labbe, 2018):

3.2.0.1 Predict

$$\vec{x}_- = \vec{x}_+ + \vec{f}(\vec{x}_+)dt \tag{20}$$

$$P_- = FP_+F^T + Q \tag{21}$$

3.2.0.2 Update

$$K = P_-H^T(HP_-H^T + R)^{-1} \tag{22}$$

$$P_+ = (1 - KH)P_- \tag{23}$$

$$\vec{x}_+ = \vec{x}_- + K(\vec{z} - \vec{h}) \tag{24}$$

where the $+$ and $-$ symbols denote the aposteriori (posterior) and apriori (prior) states respectively, Q is the process noise matrix, K is the Kalman gain matrix, R is the sensor noise matrix, and P is the state covariance matrix.

4 Software Architecture

As mentioned previously, the ADC software developed for the original ExoCube was not rigorously tested and was not designed to be reused on future PolySat missions. This work set out to design the ADC software so that it is easy to reuse and extend on future missions, compatible with the larger PolySat code base, and testable in the most flight-like configuration possible. Although the control algorithm for ExoCube 2 is beyond the scope of this paper, its place in the software architecture is discussed for the sake of completeness.

Although high-level languages and environments like MATLAB, Simulink, and Python provide robust ecosystems for developing and validating ADC algorithms, PolySat spacecraft are barred from flying these tools due to the highly resource constrained PolySat flight computer. For this reason, the ADC software was written in C to minimize memory footprint and maximize computational efficiency. This choice had the added benefit of being compatible with the wider PolySat code base (Manyak, 2011).

4.1 Reusability and Extensibility

Reusability and extensibility were achieved by implementing a basic model of object orientation in C (Schreiner, 2018). More specifically, we defined several generic interfaces that together accomplish all the features of the ADC software. A high-level diagram of some these interfaces and their interactions is shown in Figure 2. Each block represents a different interface. In this UML-style diagram, the arrows do not represent inheritance, but rather reflect the dependency hierarchy within the software. In other words, `ADCS State` stores instances of `Mission Determination` and `Mission Controller`, `Mission Controller` contains multiple `Controller` objects, etc. Because these blocks represent interfaces, different implementations can be easily swapped in and out, enabling ADC developers to reuse past implementations and provide new implementations with little effort. A configuration file based plugin system was developed to simplify this implementation swapping. Note that the software utilizes many more generic interfaces than depicted, but they were left out of the diagram to reduce clutter.

The two branches depicted in the diagram represent the two primary components of the software, determination and then control. The yellow arrows at the bottom represent the flow of data within the software—the attitude estimate created by the determination side is passed to the control side, and the control torques generated by the control side are passed back to the determination side to be used in the predict step of the Kalman filter.

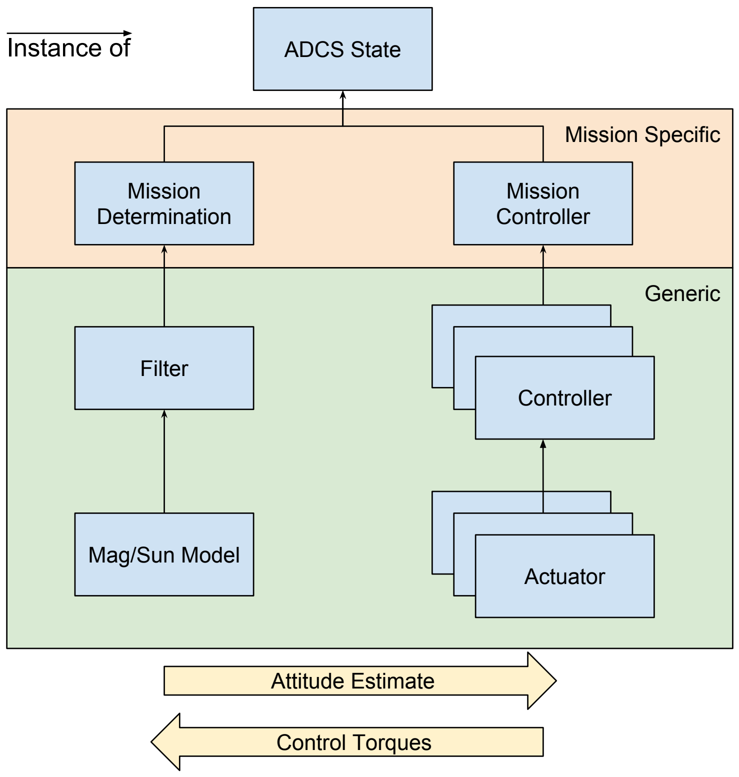

Figure 2: Object model for the ADC software architecture.

4.1.1 Control Architecture

To create a complete picture of the software architecture, all of the interfaces will be described from the bottom of Figure 2 upward, starting on the control side. The lowest level interface is the `Actuator`, which is a lightweight API abstracting the actuator drivers away (e.g. magnetorquer and reaction wheel drivers). Users can pass torque or magnetic dipole moment vectors to these objects and expect the appropriate physical action to take place. Note that the `Actuator` interface actually refers to a set of interfaces, as the API to generically control reaction wheels is fundamentally different from that of magnetorquers.

Next we move to the `Controller` interface, which utilizes the `Actuator` interfaces to control the attitude or angular velocity of the spacecraft. The `Mission Controller` object passes the `Controller` an attitude estimate, relevant sensor readings, other data that could be useful within a control law, and potentially a desired orientation or angular velocity for the spacecraft. The `Controller` object is where the actual control law is implemented. For example, there is an implementation of a `BDOT Controller`, which torques the magnetorquers in the direction opposite that of the change in magnetic field to slow down the spacecraft's angular velocity. One could also imagine implementing a `Reaction Wheel Controller`, which utilizes reaction wheels and some control law to torque the spacecraft into a desired orientation. The torques and dipole moments generated by the `Controller` is passed back to the determination branch so that they can be included in the Kalman filter predict step.

The `Mission Controller` is responsible for switching between different `Controller`'s as dictated by the mission requirements. For example, if a spacecraft first needs to detumble using BDOT before starting its primary control law, the `Mission Controller` would run the `BDOT Controller` until some convergence criteria was met, and then initialize the primary control law. The `Controller`'s themselves are thus agnostic to these mission-specific mode changes.

The larger ADC code base is responsible for calling methods within the `Mission Controller` at the appropriate time. The `Controller` and further the `Mission Controller` are expected to return the amount of time that should elapse before being called again. This allows the `Controller` to vary its time interval. For example, within the `BDOT Controller`, the algorithm samples the magnetometers twice to compute the change in magnetic field, and then proceeds to torque the magnetorquers in the appropriate manner. The time interval between magnetometer samples may be different than the torque duration, thus the `BDOT Controller` needs to vary the time between calls. The `Controller` also returns a flag indicating whether the determination algorithm should be run or not. This temporary suspension of the determination algorithm has many uses. For example, in magnetorquer-dependent `Controller`'s like the `BDOT Controller`, the determination algorithm should not be run while the magnetorquers saturate the magnetometers.

4.1.2 Determination Architecture

The lowest level of the determination branch is the `Magnetic Model` and `Sun Model`. These are interfaces that take the spacecraft's position and the current time and produce ECI magnetic and solar reference vectors respectively. These are needed by the `Filter` class, which is where the Kalman filter algorithm described above resides, to perform the update step. Wrapping solar and magnetic models in a generic interface allows ADC developers to utilize the reference model that best satisfies the mission requirements.

Implementations of the `Filter` interface are passed sensor readings and control torques and are expected to produce an attitude estimate. The `Mission Filter` adds an extra degree of mission-specific flexibility on top of the `Filter` by allowing missions to switch between `Filter` implementations if needed. Furthermore, if the mission needs to model disturbance torques that are not modeled in generic `Filter` implementations, the `Filter` interface provides an API to allow `Mission Determination` implementations to provide these contributions to the dynamics.

As mentioned previously, the determination code is executed at the command of the control algorithm. It has no mechanism to change the rate at which it is called.

4.1.3 Configuration Files

This software utilizes a robust network of configuration files to allow specifying which `Filter`, `Controller`, `Actuator`, etc. implementations to use, thus building out a basic plugin or extension system. As this code base matures, and more robust `Filter` and `Controller` algorithms are prototyped and tested, mission developers will be able to support full attitude determination and control by writing just a series of configuration files and bare-bones implementations of the `Mission Determination` and `Mission Controller` interfaces, which should be similar to example implementations. This enables developers to maximize code reuse and implement only the ADC functionality that is specific to the mission, minimizing risk and development time.

4.2 Testing and Verification

Since it is impossible to fully recreate orbital conditions on earth, and even crude approximations are expensive, validating the performance of ADC software can be a challenge, especially for an undergraduate research lab with a small budget. For this reason, a software framework was developed so that all hardware interactions could be virtualized and modeled using a spacecraft dynamics simulator. The high-level architecture of this framework is shown in Figure 3.

The core of the hardware virtualization happens within `libpolydrivers`, PolySat's hardware abstraction library used across its flight software. `libpolydrivers` abstracts all sensor and actuator hardware to a consistent interface. Thus, drivers were implemented for every sensor and actuator used by the ADC software so that, instead of interacting with physical hardware, the driver sends a JSON packet to a simulation server where the appropriate action is taken. For example, if the ADC software was attempting to interact with a magnetometer, the simulation server would respond with the current magnetic field estimate in the body frame for the given position. If the software was attempting to interact with a reaction wheel, the simulation server would model the induced torque.

Because the hardware virtualization happens within the driver abstraction layer, the user must only change a configuration value to switch into simulation mode. No changes to the ADC software are required. This means that the actual flight ADC software build can be tested without any changes—all the logic to perform the simulation happens within the driver library, which is a dynamic library loaded at runtime. Furthermore, because the virtualization is network based (it sends a JSON request to an external server), the ADC software

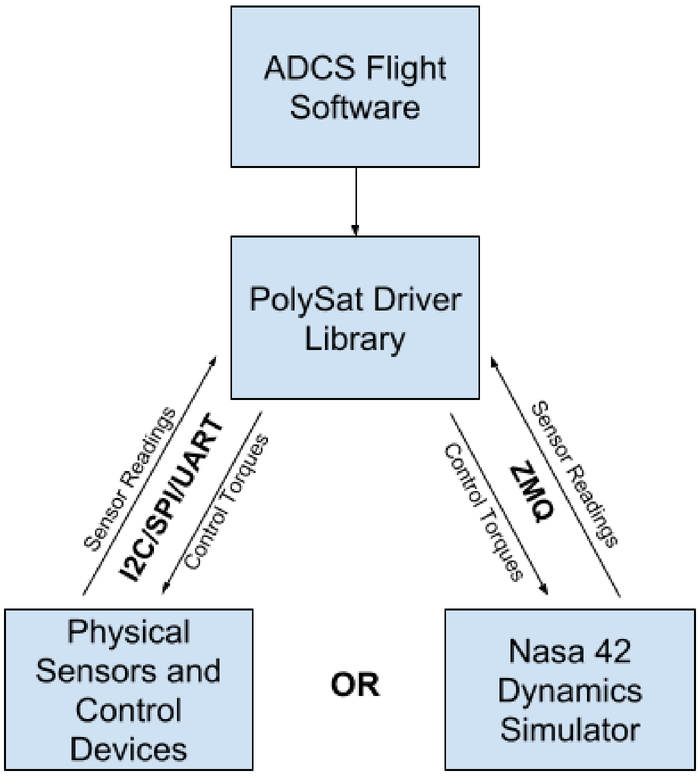

Figure 3: ADC software testing framework architecture.

can be running on the resource-constrained flight hardware while the simulation server runs on an external computer, significantly expediting simulation time while further increasing flight fidelity.

The simulation server is based off of NASA's open-source spacecraft dynamics simulator entitled "42." 42 is a fast, comprehensive dynamics simulator written in C. A Python binding was written atop 42 so the server infrastructure could be built in a high level language, making server development much more flexible. Network requests are sent through ZMQ sockets as they are more flexible than standard sockets and have a wide array of platform and language support.

4.2.1 Simulation Control Flow

When the ADC software is started in simulation mode, `libproc`, the core PolySat process library, is initialized to use "virtual time," which means the software executes as quickly as possible, ignoring any sleeps or delays, and keeps track of what the time would be if the process was executing in "normal time." Note that the user provides an initial time as a configuration value.

Simultaneously, the simulation server is started as a separate process, either on the same computer or on an external computer. The simulation server is provided with the spacecraft's initial conditions and the time that was used as the starting point in the ADC software. When the ADC software reaches its first instance of hardware interaction, a JSON packet is sent through a ZMQ socket from the ADC software to the simulation server. The packet includes the current virtual time. When the simulation server receives this packet, it commands 42 to propagate forward to the virtual time specified in the packet and performs the expected action—either modeling an actuator or responding with a sensor reading. This process repeats for as long as the simulation was configured to execute.

5 Results

Once the Kalman filter was implemented and debugged, and the testing infrastructure described previously was functional, the determination algorithm was tested in a multitude of scenarios to characterize its performance. The orbit with which the software was tested was chosen to be similar to what ExoCube 2 will experience on flight—a roughly circular orbit with an 81° inclination and an altitude of approximately 500 km.

5.1 Magnetometer and Solar Angle Sensor Determination

The filter's performance was first evaluated with both the magnetometer and solar angle measurements available. To most effectively test the filter, the spacecraft state vector was initialized to a random orientation and a random angular velocity vector of magnitude less than 6 $\frac{deg}{sec}$, which is consistent with CubeSat deployment angular velocity estimates (Sellers, 2013). Gaussian noise was added to the simulated sensor readings, with 5° standard deviation added to the magnetometer measurements and 3° standard deviation added to the solar angle measurements. These uncertainty values are consistent with the instrumentation on ExoCube 2. Figure 4 shows convergence of the filter for seven different trials, where the error angle is defined as the angle between the z-axis of the actual and estimated body frames. The Kalman filter parameters used in these simulations are shown in Table 1.

Table 1: Kalman filter parameters used in Figure 4 simulations. The values listed for the process noise and covariance matricies have mixed units as they apply to both angular velocity ($\frac{rad}{sec}$) and quaternions (unitless). Both the Solar and Magnetic measurement noise values are applied to unit reference vectors, and thus they are unitless.

Process Noise Matrix Diagonal (Q)	5×10^{-8}
Initial Covariance Matrix Diagonal (P)	5×10^{-6}
Magnetic Field Measurement Noise (R_{mag})	0.5
Solar Angle Measurement Noise (R_{sol})	0.4

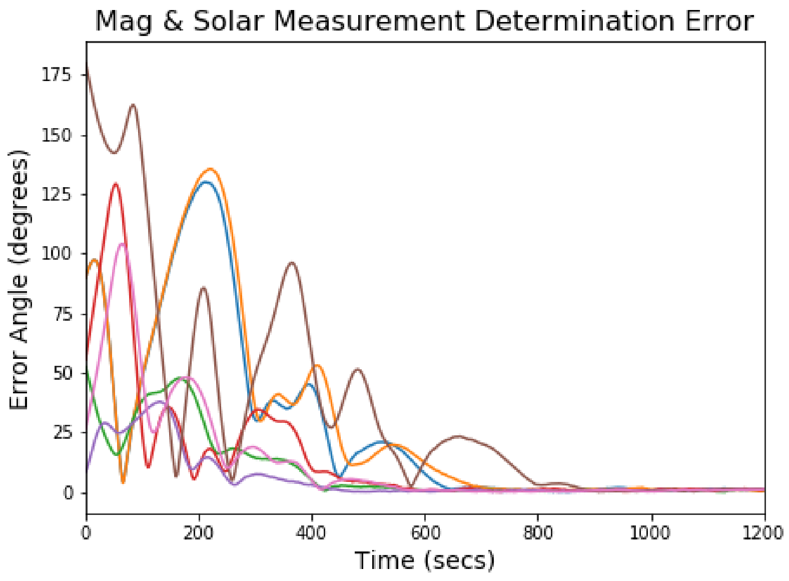

Figure 4: Kalman filter convergence across several simulations utilizing both magnetometer and solar angle readings. Each curve represents a separate simulation. The error angle is defined as the angle between the z-axis of the actual and estimated body frames.

For each situation, the Kalman filter converges in approximately 900 seconds or 15 minutes. The mean error angle after convergence across all trials is 2.3°, which satisfies the determination requirement for ExoCube 2.

5.2 Magnetometer-Only Determination

Since ExoCube 2 will be passing into eclipse while orbiting, it will periodically lose its solar angle reading. For this reason, the performance of the Kalman filter was evaluated in the same situation described in the previous section, but without solar angle measurements. Again, the Kalman filter parameters used are shown in Table 1. Data from these trials are shown in Figure 5.

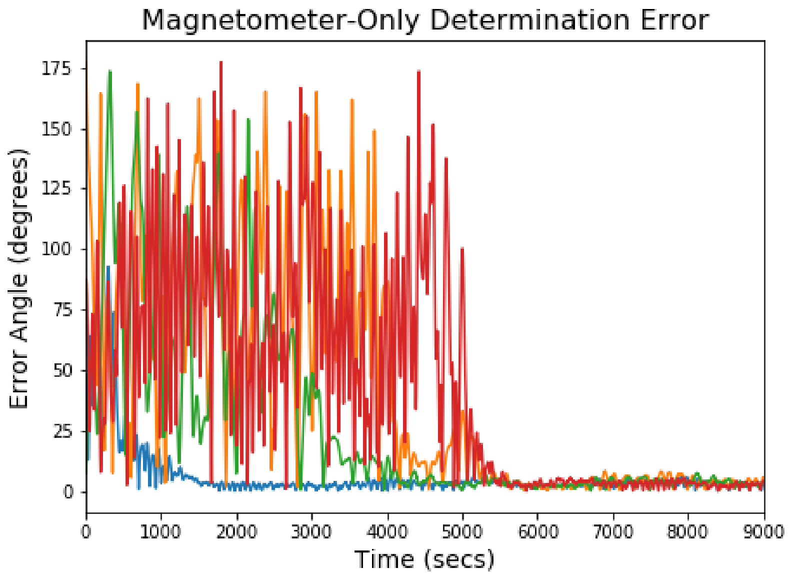

Figure 5: Kalman filter convergence across several simulations with just magnetometer readings. Each curve represents a separate simulation.

With just magnetometer readings, convergence time was much more variable. Although several of the simulations depicted in Figure 5 converge in approximately 6000 seconds (about one orbit of the earth), some simulations took far longer to converge, and some took far shorter. However, ExoCube 2 operations will be designed such that the Kalman filter is not initialized until solar angle readings are available, so the magnetometer-only convergence time is not of great importance. What is of significant importance is the ability of the Kalman filter to maintain attitude knowledge when solar angle readings are not available. After convergence, the Kalman filter in these simulations maintained a mean error angle of 3.6°, which is also within ExoCube 2's ±5° requirement. Thus, the Kalman filter meets ExoCube 2 requirements even when solar angle measurements are unavailable.

One limitation of the magnetometer-only determination algorithm occurs when the spacecraft rotates at very low angular rates ($< 1\frac{deg}{sec}$). At low angular rates with high-uncertainties, there exists a larger set of attitudes that provide the same magnetic-field measurements over longer time scales. This makes Kalman filter convergence very slow or sometimes impossible. Potential mitigation of this issue is discussed in the Future Work section.

6 Conclusion

This work implemented a Kalman filter in C to determine the attitude of PolySat's ExoCube 2 using magnetometer and solar angle measurements. This implementation was tested extensively using a software-based testing framework built off NASA's open-source spacecraft dynamics simulator "42", and the software was designed to be reusable on future PolySat missions using a robust software architecture. Simulating the uncertainties expected from ExoCube 2's instrumentation, the Kalman filter converged to the spacecraft's true attitude in approximately 15 minutes and maintained an attitude knowledge with 2.3° mean error. If solar angle measurements were lost due to the spacecraft passing into eclipse, the Kalman filter still maintained an attitude knowledge with a mean of 3.6° error. Both of these uncertainties are within ExoCube 2's ±5° attitude knowledge requirement.

Because the Kalman filter converged even when just magnetometer readings were available, this software could be used in any PolySat spacecraft with little development time, as all PolySat spacecraft include magnetometers as part of the generic hardware. Thus, the software described in this paper is an enabling technology for the PolySat lab.

6.1 Future Work

This work put little effort or analysis into the selection of the Kalman filter parameters. More thoughtful selection of the process noise matrix (Q) and the initial covariance matrix (P) could yield significant improvements in the filter's convergence time and steady-state error.

Furthermore, the seven-dimensional filter employed in this work has a singular covariance matrix due to the norm-constraint on the quaternion (Lefferts, Markley, & Shuster, 1982). This can lead to numerical instabilities that produce non-optimal filter performance. One algorithm to avoid these instabilities is the Multiplicative Extended Kalman Filter (MEKF). This may have the added benefit of improving filter performance at low angular velocities, which could resolve the problem described in the Magnetometer-Only Determination section above.

7 Acknowledgments

I first want to thank all the PolySat faculty and students that came before me and laid the groundwork for this research, specifically Dr. Jordi Puig-Suari, Ryan Sellers, and Arash Mehrparvar.

I absolutely must thank Dr. John Bellardo for lending me his infinite wisdom not just for this project, but throughout my undergraduate career.

Dr. Matt Moelter, thank you dearly for your mentorship over these last two quarters.

Thanks to all the wonderful student engineers I've worked with at PolySat over the last four years. Special thanks to Liam Bruno for leading my feeble physics brain through the treacherous landscape of spacecraft controls.

Thanks to my girlfriend, Mari Friedman, and to anyone else who has supported me through this project and through my time at Cal Poly.

And finally, thanks to my parents, Helene and Louis Bouchard, who are ultimately responsible for the existence of this project. Thanks for your endless support.

References

Cal Poly CubeSat Lab. (2014, February). CubeSat Design Specification Rev. 13. California Polytechnic State University, San Luis Obispo.

Curtis, H. D. (2010). *Orbital Mechanics for Engineering Students.* Burlington: Elsevier Ltd.

Labbe, R. R. (2018). *Kalman and Bayesian Filters in Python.* Self Published.

Lefferts, E., Markley, F., & Shuster, M. (1982). Kalman Filtering for Spacecrafts Attitude Estimation.

Manyak, G. (2011, June). *Fault Tolerant and Flexible CubeSat Software Architecture.* California Polytechnic State University, San Luis Obispo.

Mehrparvar, A. (2013, November). *Attitude Estimation for a Gravity Gradient Momentum Biased Nanosatellite.* California Polytechnic State University, San Luis Obispo.

Saunders, A. (2016, April). A Failure Analysis of the Exocube CubeSat. CubeSat Developers Workshop.

Schreiner, A.-T. (2018). *Object-Oriented Programming With ANSI-C.*

Sellers, R. (2013, March). *A Gravity Gradient, Momentum-Biased Attitude Control System for a CubeSat.* California Polytechnic State University, San Luis Obispo.