Simultaneous Orbital and Attitude Propagation of CubeSats in Low Earth Orbit

Kareem Omar
University of Alabama in Huntsville
1218 Baker Lane, Nicholasville, KY 40356
kareem.omar@uah.edu

Faculty Advisor: Michael Briggs
University of Alabama in Huntsville

ABSTRACT
The proliferation of CubeSats has brought satellite engineering and mission design into the hands of students and engineering teams across the country and the world. The low mass and low altitude of CubeSats in LEO means that aerodynamic forces will cause orbital decay and deorbit, while aerodynamic torques can cause reaction wheels to saturate, necessitating magnetorquer to decouple angular momentum. Conversely, aerotorque can be leveraged for passive aerodynamic stabilization. In either case, understanding and predicting aerodynamic effects is critical to mission design. However, these effects are coupled to the instantaneous position, attitude, and velocity of satellites, making them difficult to model accurately. Furthermore, extant software typically focuses only on orbits, or only on generalized dynamics, not coupled orbit and attitude. The author thus presents a flexible, free and open source software infrastructure for simultaneous orbital and attitude propagation, catering to the unique requirements of CubeSats. Extreme performance allows simulation of an entire mission, from deploy to deorbit, in less than three minutes. Live 3-D visualizations provide instant feedback, encouraging experimentation with and learning about satellite design and orbital/attitude dynamics. Comprehensive documentation serves as a reference for those interested in the rich field of aerospace simulation.

INTRODUCTION
The medical field employs detailed simulations with rich visualizations to allow students to explore and practice in a safe environment. The author feels engineering and physics can similarly benefit from such simulations. The existing literature supports the idea of graphical applications, including games such as Kerbal Space Program, as a teaching tool for helping students intuitively grasp physics and engineering concepts and connect them to real-world effects. It is the intent of this paper, then, to extend the literature – and the body of free, open-source software – with a simultaneous orbital and attitude propagation tool, including aerodynamic modeling and rapid 3-D visualization, that can be used for CubeSat design exploration, aerospace education, and edification in the interdisciplinary study of the development of aerospace computer simulations.

COORDINATES AND REFERENCE FRAMES
Uniquely determining a position or other quantity containing both magnitude and direction in 3-dimensional space requires a coordinate representation containing 3 degrees of freedom (DoF), one for each dimension. Many such coordinate systems exist, each with advantages in particular applications. A reference frame is a specification of the location of the origin and orientation of the axes of one coordinate system relative to another, both of which may vary with time. Reference frames are a vital building block of aerospace and astrodynamics simulations, but they are not particular intuitive at first, so the author perceives value in providing a background and using them in a transparent manner in KPS to facilitate experimentation and understanding. KPS utilizes five right-handed Cartesian frames common to aerospace:

1. **Earth-Centered Inertial (ECI) Frame** – used for satellite position, velocity, and orientation
2. **Earth-Centered Earth-Fixed (ECEF) Frame** – used for Earth’s magnetic field
3. **Local Vertical, Local Horizontal (LVLH) Frame** – used for initial retrieval of Earth’s magnetic field from spherical harmonic model, which is returned in an LVLH frame
4. **Orbital Frame** – used for live 3-D rendering of satellite orientation, allowing easy viewing of stabilization and/or Earth-pointing behaviors
5. **Body Frame** – used for satellite angular velocity and specifying satellite geometry, center of mass, and moment of inertia matrix.

**ORBITAL DYNAMICS**

Satellites in Low-Earth Orbit are in captured elliptical orbits, which can be described by five parameters. The instantaneous position of a satellite within its orbit can be described by a sixth element, the true anomaly. The six elements taken together are known as the classical or Keplerian orbital elements. The advantage of the Keplerian orbit representation is that it specifies a closed form – given Keplerian elements at a particular time, one can, in constant time, compute the position of the satellite at any time in the future without requiring integration over the intervening time.

However, in reality, many effects, called perturbations, cause the first five Keplerian elements to change slightly over time. Perturbations vary in significance, and many can be ignored, depending on the application. Some perturbations can be approximately modeled such that the orbit remains closed-form with additional computation. However, if certain perturbations are desired, or particular accuracy is needed, applying closed-form corrections becomes difficult or impossible. In the case of KPS, full atmospheric drag modeling coupled to attitude (in which the attitude of the spacecraft can significantly affect the frontal drag area and the precise effects of this coupling are desired) requires integrating varying aerodynamic forces and torques over time, and magnetic torque is dependent on Earth’s fluctuating magnetic field and the satellite’s instantaneous attitude. A closed-form solution does not exist; instead, differential equations govern the satellite’s behavior and can be propagated forward in time via numerical integration.

The disadvantage of direct integration is that computation is now required per unit time propagated. However, once this price is accepted, more perturbations can be considered with relative ease. It is no longer necessary to apply complex corrections for approximate gravitational effects, for example; instead, integrate with the local gravitational field from a spherical harmonic gravity model at each point and let Newton’s laws do the rest. In this mode of propagation, the Keplerian orbital elements are replaced by two Cartesian 3-vectors in the ECI frame: one for position, denoted \( \vec{r} \), and one for velocity, denoted \( \vec{v} \). These are known as state vectors. See Table 1. The same fundamental ideas about a satellite are described by orbital elements and state vectors – where it is, and where it’s going if unperturbed.

<table>
<thead>
<tr>
<th>Keplerian Elements</th>
<th>State Vectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>( r_x = x )</td>
</tr>
<tr>
<td>( e )</td>
<td>( \vec{r} )</td>
</tr>
<tr>
<td>( i )</td>
<td>( r_y = y )</td>
</tr>
<tr>
<td>( \Omega )</td>
<td>( r_z = z )</td>
</tr>
<tr>
<td>( \omega )</td>
<td>( \vec{v} )</td>
</tr>
<tr>
<td>( \nu )</td>
<td>( v_x = \dot{r}_x = \dot{x} )</td>
</tr>
<tr>
<td></td>
<td>( v_y = \dot{r}_y = \dot{y} )</td>
</tr>
<tr>
<td></td>
<td>( v_z = \dot{r}_z = \dot{z} )</td>
</tr>
</tbody>
</table>

KPS operates exclusively with state vectors, but to facilitate understanding of the differences between the two representations, utilities are included allowing users to convert between state vectors and Keplerian elements, including live plotting of any Keplerian element over time for a simulation run.

**ATTITUDE DYNAMICS**

The attitude of a spacecraft refers to its orientation relative to a specified coordinate system. In KPS, attitude is referenced to the ECI frame. As described in the section on coordinate frames, the Body frame is attached to the spacecraft, translating and rotating with it, so to positively specify the satellite’s attitude is to specify the orientation of the Body frame relative to the ECI frame. In orbital dynamics, the position is specified as a 3-vector pointing from the origin of the ECI frame to the satellite. In attitude dynamics, the situation is more complicated, as one must describe the orientation of the Body frame relative to the ECI frame. The Apollo missions used Euler angles for this purpose. Other possibilities include rotation matrices and quaternions.

In KPS, attitude is specified as a quaternion which can rotate the ECI frame into the Body frame, or equivalently, rotate a vector in the Body frame into the ECI frame. The raw attitude quaternion and a quaternion able to rotate vectors from the Body frame into the Orbital frame are exported during simulation to allow users to investigate the quaternions’ properties. Computing the derivative of the attitude is not as straightforward as the equivalent problem in orbital dynamics, where the derivative of the position is simply the velocity. In contrast, the derivative of a quaternion is given by:

\[
\dot{q} = \frac{1}{2} q \vec{\omega}_b
\]  

(1)
where $\vec{\omega}_p$ is the angular velocity of the satellite coordinatized in the Body frame. A proof is beyond the scope of this paper. However, the author has written a nearly 60-page thesis to accompany this paper which contains comprehensive derivations of every equation used in KPS. Furthermore, in the KPS source code, anywhere an equation is used, a comment indicates the relevant section in the thesis, so the interested reader can quickly discover the origins of every implementation detail.

The second derivative of orientation, too, is more complicated. In orbital dynamics, forces are simply divided by satellite mass to obtain an acceleration, which can be integrated to directly obtain velocity. In contrast, the relation between torque and angular acceleration is given by:

$$\ddot{\vec{a}} = \ddot{\vec{\omega}} = I^{-1}(\ddot{\vec{r}} - \vec{\omega} \times I \vec{\omega})$$  \hspace{1cm} (2)

Equation 2 is used by KPS to compute angular acceleration based on applied torques, just as Newton’s Second Law is used to compute acceleration based on applied forces for orbital dynamics. Table 2 enumerates the external impetuses considered by KPS for orbital and attitude dynamics.

**Table 2: Orbital and Attitude External Impetuses**

<table>
<thead>
<tr>
<th>Orbital Dynamics – Forces</th>
<th>Attitude Dynamics – Torques</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aerodynamic Force</td>
<td>Aerodynamic Torque</td>
</tr>
<tr>
<td></td>
<td>Magnetorque</td>
</tr>
<tr>
<td>Gravitational Force</td>
<td>Gravity Gradient Torque</td>
</tr>
</tbody>
</table>

**NUMERICAL INTEGRATION**

Numerical techniques exist to compute to high accuracy a numerical solution for the integral of an ordinary differential equation (ODE) given initial conditions (ICs). If one knows how fast a function is changing, and where it started, one can plot the trajectory of the function. The simplest mechanism for this is known as Euler’s method. If rate of change is change per unit time, then multiply an amount of time by that rate gives you the change. Add this change on to the initial condition, and the function has been stepped forward, approximately, by that amount of time. Euler’s method is the simplest method in a class of integrators known as Runge-Kutta (RK) methods. It is a first-order method, meaning that the total error is proportional to the step size. This is unacceptable for precision work, and higher-order RK methods can do much better. In fact, if particularly fast and robust propagation is needed, combining a high-order RK method with an adaptive step size produces the best results. Adaptive ODE solvers continuously adjust the step size to be as large as possible without compromising accuracy, as determined by comparing the results of the different orders.

Perhaps the most common adaptive solver is the Runge-Kutta Dormand-Prince (RKDP) method, which computes fourth- and fifth-order solutions. The coefficients are carefully chosen to allow maximum reuse of computations between the two orders, and the last stage of one step is recycled in the first stage of the next step for additional savings. However, the other intermediate results are discarded after that step. The next step begins without any knowledge of the previous step, except the recommended step size, which diminishes both accuracy and performance. Solvers known as linear multistep methods take better advantage of information from previous steps. This greatly improves accuracy and performance, particularly when the cost of ODE evaluation is high – which it is for attitude propagation due to the complex aerodynamics, magnetic, and gravitational simulations. The author has created an adaptive variable order linear multistep predictor-corrector integrator for users, known as an Adams-Bashforth-Moulton (ABM) solver.

In the case of orbital dynamics, acceleration can be obtained from the forces applied to the satellite. Position is desired. Acceleration is the second derivative of position, however. The first derivative is velocity. The solution is to instead integrate a derivative vector that jointly considers position and velocity. Similarly, in attitude dynamics, angular acceleration can be obtained from torques applied to the satellite. Orientation, represented by an attitude quaternion, is desired. Figures 1 and 2 illustrate this process for orbital and attitude dynamics, respectively.

**Figure 1: Orbital propagation**
GRAVITATIONAL MODELING

KPS offers a variety of gravity models from which users can select, providing a tradeoff between performance and accuracy, but also allowing users to experiment with the different models and immediately see the effects on the propagation. The simplest model provided is a point-mass Earth. More complex models are implemented by expanding spherical harmonic terms.

MAGNETIC MODELING AND MAGNETORQUE

Satellites may wish to control their attitude by reacting against Earth’s magnetic field using magnetorquer. This requires knowledge of the Earth’s magnetic field pseudovector at the satellite’s position. To this end, as with the gravitational models, several choices are offered, of varying complexity and speed. Mission designers who wish their satellites to stabilize, passively or actively, in a particular orientation must be able to offload angular momentum received from satellite deployment or aerodynamic torques. Onboard magnetorquers can accomplish this without resorting to chemical thrusters by reacting against Earth’s magnetic field in opposition to the satellite’s angular velocity. Note that such magnetorquer, produced by two interacting magnetic fields, is orthogonal to both fields. The field produced by the magnetorquer is arbitrary, but that of Earth is fixed, so the satellite cannot perfectly oppose its angular velocity. However, as the satellite orbits, the Earth’s magnetic field changes, as does the angular velocity vector due to other torques, allowing magnetorquer to slowly but successfully diminish angular velocity over time. Equation 3 is used in KPS to generate magnetorquer, and the gain factor, \( k \), is specified by the user:

\[
\tau = \left( -k (\beta \times \omega) \right) \times \beta
\]  

(3)

ATMOSPHERIC MODELING

Atmospheric modeling is a vast and complicated field. Density and pressure vary rapidly with multiple cycles of different periods, from one day to one year to one solar cycle. Myriad factors influence the properties of air at different altitudes, and at sufficiently high altitudes, the composition itself begins to change. Its components occur in abnormal quantities and ionize. Consequently, the most accurate models require frequent inputs of solar activity and other corrective data, and are thus only accurate for the short time period specified in this data.

Because KPS is capable of rapidly simulating years of orbit, at present, such models are not an excellent fit, and approximate density modeling is sufficient. Time-specific models such as those described above may be supported in a future version of KPS. Currently, KPS uses the U.S. 1976 Standard Atmosphere in both its low- and high-altitude modes for altitudes from 0 to 1000 km. Above 1000 km, atmospheric effects are very slight, and are ignored. KPS is intended for CubeSats in LEO.

AERODYNAMICS

KPS implements fast aerodynamics in two different manners in the hope of providing both utility and instruction. The air density is known from the atmospheric model. The mean free path is quite large, meaning that atmospheric particles are sufficiently sparse that they hardly interact with each other, only with the satellite. This is critical, as at sea level, complex flows develop due to the vast number of collisions made between particles every second. In orbit, particles can be considered separate entities with which the satellite collides. If these particles are assumed to undergo specular reflection upon contact with the satellite, the momentum transfer per unit time, i.e. the force, is given by:

\[
\vec{F} = 2\rho A v \vec{v}_\perp
\]  

(4)

Where \( \rho \) is the air density, \( A \) is the exposed frontal area of the panel surface (i.e. the projected frontal area normal to the velocity of oncoming particles), \( v \) is the air’s relative velocity, and \( \vec{v}_\perp \) is the component of the air’s relative velocity normal to the panel. At this point the simulation differs based on the user’s choice of aerodynamics model. The analytical mode projects the satellite’s 3-D geometry into 2-D with z-ordering as viewed by the oncoming air, an example of which is shown in Figure 3 using KPS’ live 3-D orientation visualization. The aerodynamics engine performs this projection in real-time programmatically, computing the resultant forces and torques using Equation 4.
Each panel’s frontal area is analyzed to remove portions occluded by other polygons in the scene, and the remaining frontal areas experience forces at their centroids in accordance with Equation 4. Each of these forces, if not in line with the satellite’s center of mass, also confers a torque to the satellite. For those with an interest in aerospace software development, the polygon analysis process requires KPS to solve the polygon clipping problem (determining the intersection and difference of overlapping polygons), a classic problem in modeling, simulation, and computer graphics.

For edification in an alternate approach (and one which illustrates an implementation that solves a common problem – particle collision – in aerospace), KPS can also operate by generating grids of test particles to collide with the satellite, each representing a small panel of area to sample. Note that these test particles are simply for evenly colliding with the geometry of the satellite and are unrelated to physical atmospheric particles. Each of these test particles strikes the satellite, and Equation 4 is used to compute the force vector added by that individual particle, with $A$ equal to the area of the grid represented by that particle. These forces are summed to compute an approximation of the net force on the satellite, the accuracy of which increases with the number of particles. The particle collision process requires KPS to solve the point in polygon problem (determining whether the intersection of each particle and each satellite panel actually occurs within the boundaries of the panel), also a classic problem in computer graphics and simulation.

**CUDA**

CUDA is a parallel processing platform developed by NVIDIA\textsuperscript{12}. CUDA tools compile code written in C++ for execution on the Graphics Processing Unit (GPU) rather than the Central Processing Unit (CPU), where code typically executes. GPUs have evolved to meet the needs of gaming graphics, which typically follow a Single Instruction, Multiple Data (SIMD) paradigm in which little branching occurs and the same instruction must be rapidly (preferably simultaneously) executed on different data. See Figure 4.

Note that CPUs do also have some SIMD capabilities, though they cannot rival GPUs for massively parallel compute-bound tasks. With careful programming, certain massively parallel problems can be solved by GPUs much more quickly than by CPUs. The collision mode of the KPS aerodynamics simulation module is precisely such a massively parallel problem. The author understands that not all interested parties may have CUDA-capable devices, so CUDA is not required to run KPS. The software can run in a carefully optimized CPU-only collision mode, or in the (recommended) analytical mode. The author has made every effort to minimize differences between the CPU and CUDA versions of the collision-mode aerodynamics module so that otherwise opaque CUDA code can be compared to something more familiar, aiding study of CUDA programming practices.

The collision problem involves several steps in which identical operations are performed repeatedly on different data, making it suitable for parallel processing. Each CUDA thread follows the same instructions. Whereas the CPU aerodynamics module iterates over individual test particles to collide them, the CUDA version launches a CUDA kernel with one thread for...
each particle. However, note that the collision operations are not purely parallel. Each particle collision contributes a small force and torque to a running total. Thus a summation is required across all threads. This is an operation known as a reduction. Such an operation, which considers every element and produces a single final answer, but in which the order of access does not matter, can be performed in logarithmic time in CUDA. Consider Figure 5.

![CUDA reduction diagram]

**Figure 5:** CUDA reduction.

If the goal is to sum the eight values, the following procedure may be used:

- Split the data into two halves, “right” and “left”
- All “right” threads terminate; they are not needed
- Each “left” thread adds the corresponding value from the right half to itself
- Repeat these steps on the “left” data

In the example from Figure 6, the eight values are summed in just three instructions – logarithmic time. The efficiency of reductions in CUDA further contributes to its performance. Reductions are a common idiom in CUDA programming but are difficult to grasp and implement at first; the author hopes to help interested programmers break into this paradigm with KPS, as the system implements two different CUDA reductions as part of the CUDA collision module and are visible in the context of a working application for which the exact same operation is visible in CPU code for direct comparison.

**GRAVITY GRADIENT TORQUE**

The fact that gravity acts more strongly on the portions of a satellite nearer the Earth, as in Figure 6a, can produce small torques known as gravity gradient torques. Figure 6b shows the net forces by subtracting the middle’s average force from all three points. Figure 6c breaks these forces into components along the long axis of the beam, and perpendicular to it. The forces along the beam on either end cancel, leaving just the perpendicular components as seen in Figure 6d.

A counter-clockwise torque is produced, and can be approximated by:

\[
\begin{align*}
\tau_x &= \frac{3\mu}{\|r\|^2} r_b \cdot [I_z - I_y] \\
\tau_y &= \frac{3\mu}{\|r\|^2} r_b \cdot [I_x - I_z] \\
\tau_z &= \frac{3\mu}{\|r\|^2} r_b \cdot [I_y - I_x]
\end{align*}
\]

Where \( r_b \) is the satellite position vector in the Body frame. Equation 5 is used by KPS to compute gravity gradient torques. These torques are small and are sometimes ignored by larger satellites, but they can be significant, or even be leveraged to aid stability, in CubeSats. KPS allows the interested student to enable and disable them and immediately view the impact on satellite attitude control, or to leave them enabled but disable all other torques on the satellite to observe this otherwise difficult-to-discern behavior.

**INFRASTRUCTURE CAPABILITIES**

KPS consists of seven utilities. In addition to the main propagator, users can generate satellite configurations from basic polygons, examine satellite geometry in 3-D, generate initial state vectors for a desired orbit, convert between state vectors and Keplerian elements, and visualize propagation results in real-time or after simulation. The main propagator is written in C++ for performance and can execute on both Microsoft® Windows® and Linux. The other six utilities are each provided in both MATLAB® and Python for convenience, with identical functionality across both languages. Furthermore, the MATLAB® scripts are compatible with GNU Octave, so a user has three
different options, including fully open source alternatives, to run or, for the interested programmer, to study. Figures 7 and 8 contain examples of dart-style CubeSats viewed at two different panel angles, as generated by the polygon generator tool and viewed in the satellite geometry tool, allowing users to rapidly prototype different satellite designs and verify their correctness. The real-time visualization tool allows the user to view in real-time any combination of eighteen satellite parameters, including the Keplerian elements, altitude, position, velocity, and others, output by KPS during or after simulation. A series of demonstrations follows.

Figure 7: 3U Dart at 120° Panel Angle

Figure 8: 3U Dart at 160° Panel Angle

Figure 9 shows the orbital position of a satellite in a 500 x 500 km circular orbit over a 200,000 second simulation (2.3 days), with the point-mass gravity model selected. Aerodynamic forces are not sufficient to cause noticeable orbital decay in this time, so the integrator is essentially solving the two-body problem, and a perfect ellipse is produced. Figure 10 shows the orbital position again for the exact same parameters, except that point-mass gravity has been replaced by WGS84 ellipsoidal gravity modeling. The ellipsoidal Earth causes perturbations, and the satellite remains in a stable orbit but experiences drift in its Keplerian elements. Interested experimenters can modify parameters such as the gravitational model in this way and watch in real time as the satellite’s orbit responds to the new model, providing what the author intends as a novel visual approach to teaching both orbital mechanics and modeling techniques.

Figure 9: 200,000 Seconds of Circular Orbit using Point Gravity

Figure 10: 200,000 Seconds of Circular Orbit using WGS84 Gravity

KPS is capable of rapidly simulating significant amounts of mission time, including following a satellite to deorbit in only a few minutes of real time. This means that users can quickly investigate the impact of even very minor changes to satellite or simulation parameters on an entire mission, aiding satellite design
and understanding of the physics which underlie the models in use. Figure 11 shows a visualization configured to plot both altitude and $B^*$ for a satellite in 500 x 600 km elliptical orbit over its entire mission life. At first, the altitude oscillates between 500 and 600 km, as expected. Over time, two effects occur. Firstly, the entire orbit decays as the satellite loses orbital energy due to drag. This begins to occur more and more rapidly as the satellite drops into a denser and denser atmosphere, as seen in the rising $B^*$ term. Secondly, however, the orbit circularizes! Notice how much smaller the variation in altitude is at $7 \times 10^7$ seconds than at deploy. Because the force of drag is strongest at perigee, it has the effect of applying retrograde thrust at perigee, stealing velocity from the satellite and affecting how far it can reach on the other side of its orbit – which lowers the apogee. Factual statements of effects of this sort are encountered in orbital mechanics texts, but in KPS they are emergent behaviors naturally resulting from the integrity of the underlying physics engine, allowing students to inherently discover them as fundamental truths about satellite behavior resulting from first principles. The entire mission – 1.4 years of orbit and attitude propagation – was simulated by KPS in less than three minutes of real time.

The next series of demonstrations investigate satellite stability by plotting angular velocity, velocity in the body frame, and pointing error. Figure 12 shows a control test – the satellite is a simple, homogeneous rectangular prism. No magnetorque is applied. The satellite deploys with some initial tumble. As expected, these oscillations continue and no stabilization occurs. Notice that in Figure 12, the angular velocity, even though it is measured in the body frame, does not remain constant, despite the lack of net torques. The $z$-component does, but the $x$ and $y$ components oscillate about each other. This is the coupling of Euler's rotation equations of rigid body dynamics that occurs in a proper physics model. In KPS this behavior is introduced by the effects of Equation 2.
This coupling is responsible for the strange behavior of spinning objects and can be observed directly. If one tries spinning something like a phone or a book about each of its three axes in turn, one finds that it spins stably about its minimum and maximum moment of inertia axes, but is unstable if spun about its middle axis! This phenomenon is the result of Euler's equations of rotational dynamics. In addition to direct application in satellite design, students can observe the effects of this strange behavior on a real (or planned) satellite’s attitude dynamics in real time with KPS. If further investigation on this particular topic is desired, the flexibility of KPS means students can disable all forces and torques, for example, and isolate the strange behavior of the angular velocity for further observation and experimentation.

The aerodynamic “dart” configuration of Figures 7 or 8 can be used to generate a passively stabilizing satellite that points into the relative wind like an arrow. However, with aerodynamics alone, the dart will not stabilize. The extremely high mean free path means that air acts like bullets, imparting momentum, not like a flow. The air can still generate restoring torques, but it does not provide the damping that, say, an arrow fired at sea level would have due to skin friction and other flow phenomena. The dart configuration must be used in conjunction with magnetorque to produce prograde stabilization. Figure 13 adds magnetorque to the dart model. With the combination of torque and damping, the satellite stabilizes with its long axis pointing in the direction of travel, just like an arrow on Earth! Note that this process is an active one. If all torques on the satellite were to cease, the satellite would not maintain pointing, because as it travels around the Earth, it must constantly turn toward the Earth to follow the horizon as it falls away. This is sometimes known as Y-Thompson spin mode. The aerodynamic torques driving this stabilization are proportional to density, so at higher altitudes, residual pointing error tends to occur, and at some altitude (for a particular satellite configuration), stabilizing shapes like the dart will fail to stabilize altogether and tumble. The stabilization can even be observed occurring in real time in 3-D via KPS’ live orientation visualization mode.

For CubeSat designers interested in taking advantage of passive stabilization to reduce cost and design complexity – appealing prospects to CubeSat projects – KPS stands unique in its ability to rapidly assess what combinations of parameters are likely to produce stable pointing, under which conditions, for an arbitrary satellite shape and orbit configuration.

CONCLUSION

KPS allows users to simulate and visualize satellite trajectories and orientations, all through a completely free and open source framework. Real-time plotting, including 3-D animation of orientation and orbital path, allows users to immediately see the effects of changes in initial conditions or the configuration of the propagator, encouraging free experimentation with...
system parameters, and six ancillary utilities allow users to experiment with orbits and satellite geometry.

In addition, the software itself has been carefully designed as a modular system for inspection and modification for interested programmers, and a unifying background has been laid in this paper and a more comprehensive thesis written by the author. In this way the author hopes to generate two-fold interest: in using KPS directly as a teaching tool for fundamental aerospace and physics concepts as well as CubeSat mission design and analysis, and in using the design of KPS itself to aid in interdisciplinary study of the development of computer simulations for the aerospace industry.

KPS and the associated thesis are available upon request and/or at the author’s GitHub.14

ACKNOWLEDGMENTS
The author wishes to thank his cousin, Sanny Omar, who freely and selflessly shares his hard-won skill in aerospace and astrodynamics and has been responsible for a significant portion, if not a plurality, of the author’s aerospace knowledge over the years. The author also greatly wishes to thank his mentor, Dr. Michael Briggs, without whose guidance this research would not have been possible.

REFERENCES


found in this paper, and indeed all equations used in KPS. Available upon request and at author’s GitHub as part of KPS source code.


