

Multi-Mission Space Exploration Vehicle Concept Simulation of Operations in Proximity To a Near Earth Object

Introduction

This paper details a project to simulate the dynamics of a proposed Multi-Missions Space Exploration Vehicle (MMSEV), and modeling the control of this spacecraft. A potential mission of the MMSEV would be to collect samples from a Near Earth Object (NEO), a mission which would require the spacecraft to be able to navigate to an orbit keeping it stationary over an area of a spinning asteroid while a robotic arm interacts with the surface.

In order to simulate the complex systems of the proposed MMSEV, its propulsion system, and the non-uniform gravity of the asteroid Itokawa, tools from the DARTS Lab have been used. The Dynamics and Real-Time Simulation (DARTS) Laboratory develops simulation tools for the development and testing of spaceflight and planetary exploration systems. The software developed is used to test and verify flight software and hardware, as well as technology development. DARTS provides a computational engine for multibody dynamics. Dshell was also developed in the DARTS lab, and provides a simulation framework for spacecraft simulation. Dynamics Simulator for Entry, Descent, and Surface landing (DSEENDS) is a simulator based on Dshell designed specifically for Entry, Descent, and Landing (EDL) scenarios¹. During the course of this project DSEENDS was used initially and later replaced with DshellCommon, a more recently developed framework.

The capabilities which have been added to the simulation during the project include thruster allocation and control, reaction wheel control, and station keeping about a non-spinning asteroid. The code structure that existed for this simulation beforehand has been modified to have a more modular structure, and the additional code added has a similarly modular structure. Parametric and small Monte-Carlo studies have also been done using this simulation. Multiple methods of control allocation are available.

Code Structure

The code is structured in classes, each which contain multiple methods. A class inheritance diagram is shown in Figure 1. The ActuatorGraphics class contains the methods for displaying an illustration of what the thrusters and reaction wheels are doing. These methods are called within the ThrusterControl and RWControl classes, which contain the methods that calculate the control actions based on inputs of the desired state of the spacecraft. These inputs are determined in the Control and StationKeeping classes. The Flight class contains methods for determining the current Local-Vertical-Local-Horizontal (lvlh) frame, which is used to orient the spacecraft relative to the asteroid. It also includes methods which set up the bodies in the simulation to have prescribed hinges, allowing velocity and position states to be set kinematically, or unprescribed hinges which make the

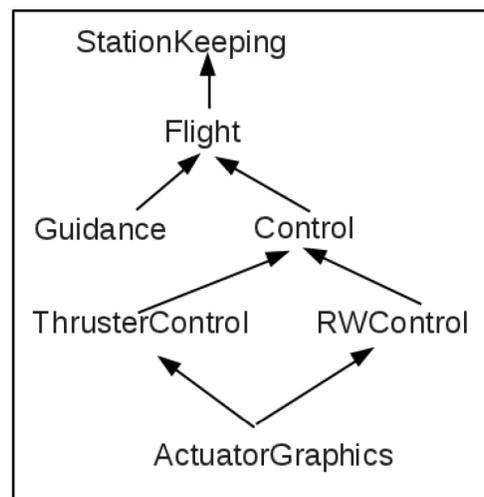


Figure 1: Class Diagram

simulation use dynamics to determine the velocity and position. The Guidance class contains methods for determining the current position, orientation, velocities and accelerations of the spacecraft, as well

as the default desired position, velocity, etc. Flight and StationKeeping are called in the top-level simulation when the spacecraft enters a particular state, determined by what is happening in the simulation. For example, when the spacecraft reaches a desired distance from the asteroid it might enter the StationKeeping state and start calling the methods included in that class.

This code was developed starting with an existing simulation which combined the Flight, Guidance, and Control classes into a single file, and did not include StationKeeping, ThrusterControl, or ActuatorGraphics. RWControl was originally a method within the Control class.

Control Systems

Thrusters are control devices which use the expulsion of mass to provide a force. Different types of thrusters can be used for translational movement, or in combination for attitude control. In the proposed system simulated, the same thrusters were simulated as simultaneously controlling the position and the rotation. Reaction wheels are flywheels controlled with electric motors that provide changes in angular momentum in order to implement attitude control.²

The sections of code which implement the reaction wheel and thruster control are each organized into their own class of methods. Each takes the desired position, velocities, accelerations, and rotations as inputs. This allows different control goals, such as a particular orientation or velocity, to be implemented with short sections of code that are stored in other classes. Both methods use gains along with the mass and inertia of the spacecraft to determine the torque, and in the case of the thrusters also the force, based on differences between the desired and actual position and velocities as well as the mass properties of the spacecraft.

The Reaction Wheel Control class code, RWControl, collects information about the masses and locations of the reaction wheels and uses it to calculate the necessary rotation rates based on the required torques.

The Thruster Control class code collects information about the location and direction of the thrusters in the system and uses this information to determine a combination of thruster levels which will provide the necessary torque as well as force. This is implemented using control allocation, which is described in more detail in the following section.

Thruster Allocation Methods:

Thruster allocation is the calculation of which thrusters need to be fired and at what level in order to achieve a desired set of forces and torques. The thruster control code collects the relevant values from the nominal simulation parameters such as the number of thrusters, locations, directions, specific impulses, and maximum thrust level. This information is collected into matrices that describe the ability of each thruster to provide forces and torques in three dimensions. These are combined into a 6-by-n *effectiveness matrix*. This is used in combination with the desired forces and torques to find a solution for the thruster levels. The solution for the thruster levels, which is non-unique due to the large number of thrusters in the system (24 for the proposed

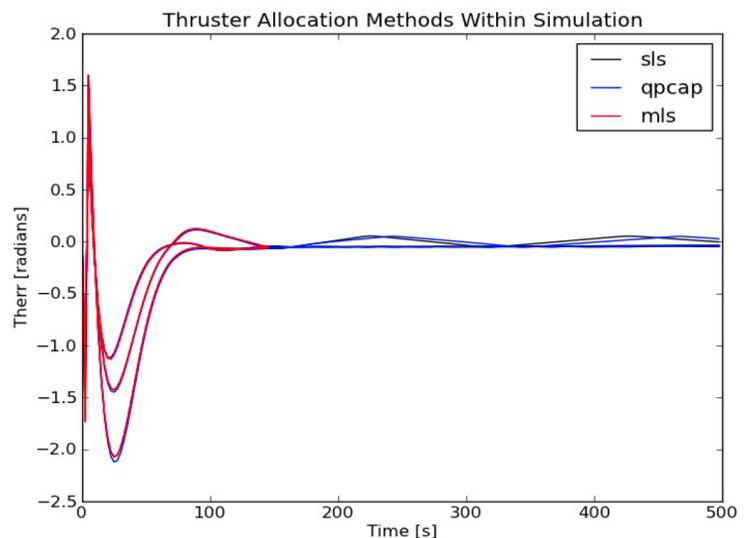


Figure 2: Thruster allocation comparison

case studied in this project), can be found in several different ways. The first method, which was also used in the included plots unless otherwise mentioned, was developed in the DARTS lab in the MATLAB programming language and translated into Python as part of this project. Other methods came from the Mathworks file exchange³, also written in MATLAB and translated into Python so that they could be used in the simulation.

Several of the methods available from this source were implemented in MATLAB for a number of attitude-adjustment scenarios drawn from the simulation, and three were translated into Python and inserted into the simulation. Figure 2 Shows the results of simulating the same situation with three different control allocation methods. The sequential least squares (sls), and a quadratic programming algorithm (qpacap) which was provided from the DARTS laboratory produce very similar torques for the whole simulation, and while the minimal least squares (mls) method experienced an error midway through the simulation up to that point it also produced similar torques. At the current time, further development is needed in two of the methods implemented in python, while the sls method and the qpacap method are functioning without error.

Capabilities of Simulated Reaction Wheel and Thrust Control

In order to determine whether the reaction wheels were acting as desired, gravity was temporarily removed from the simulation and an initial angular velocity was applied. Using only reaction wheel control a goal was set of returning the angular velocity to zero and the attitude to a desired orientation. In addition to illustrating the success in attitude control, the plots in Figure 3 also show the effect of a dead-band on the attitude control. A dead-band sets a minimum error which the control system will attempt to correct, creating a finite region where no control is attempted. The effects of this can be seen in the way that the attitude error oscillates in a tightly bound region.

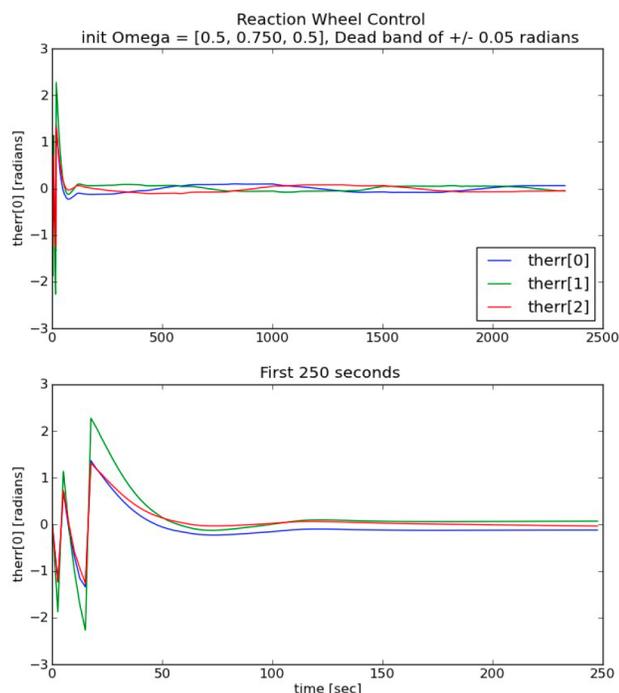


Figure 3: Reaction Wheel Control Demonstration: initial omega = [0.5, 0.75, 0.5], Dead band of +/- 0.05 radians, initial velocity of 5e-8 m/s

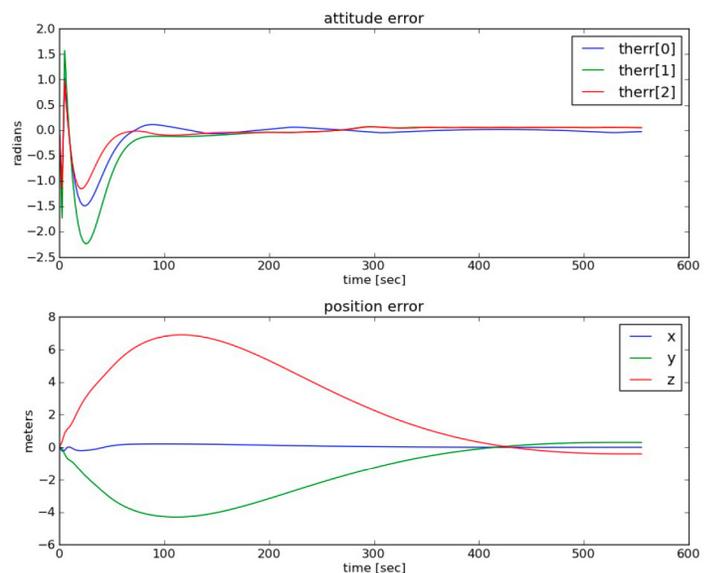


Figure 4: Thruster Control Demonstration

A similar simulation with using only thruster control was also completed. This simulation had a position goal, and a non-zero initial velocity for the spacecraft to correct in addition to the same angular perturbation used in the reaction wheel case. The results are shown in Figure 4. Although acting more

slowly than the reaction wheels, the thrusters were able to correct the attitude as well as the position. Through adjusting the position gains the response of the position was later improved. Once the basic control of the simulated spacecraft was shown in this way, the more complex goal of station keeping was attempted.

Station Keeping

Station-keeping is when a spacecraft keeps a given altitude and orientation relative to an object in space. This can mean a NEO-stationary orbit where the acceleration of gravity is theoretically sufficient at a certain altitude to keep the spacecraft at a constant position relative to the object. In reality, adjustments for non-uniform gravity and attitude adjustments are also needed. At any other altitude additional fuel is needed to make up the difference between the acceleration required to keep a stationary position relative to the object and the acceleration provided by gravity. Station-keeping is useful for observation and for near-surface operation.

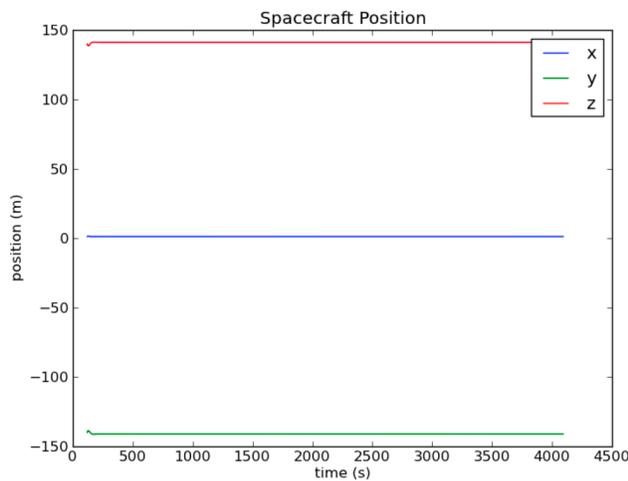


Figure 5: Non-rotating Station Keeping

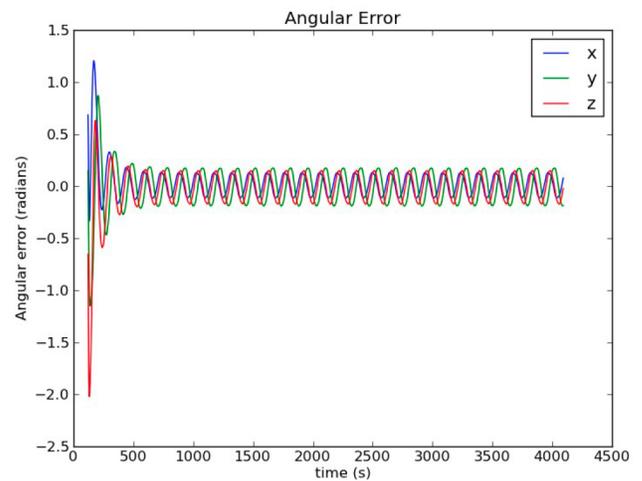


Figure 6: Non-rotating Station Keeping

With the rotation of the simulated asteroid, a model of the real asteroid Itokawa, set to zero radians per second, station-keeping appears to be satisfactorily achieved. Figure 5 shows the spacecraft staying in a single location long-term, and Figure 6 shows the spacecraft staying dead-banded about the desired attitude. When the rotation of the asteroid is set to a finite rate, instabilities occur. The position of the spacecraft is successfully held at a constant position relative to the asteroid, however the attitude of the spacecraft is not controlled well, and in many cases begins to spin with increasing velocity. This is an issue which has not yet been resolved. Several possibilities have been explored, including changing the gains and the way in which the rotation rate error is treated.

Parametric Sweeps

A parametric sweep is where a design parameter of a simulation is changed in order to observe the effects as an aid to design decisions. A parametric sweep of control gains was performed on the thruster attitude control system. The control gains are calculated from a chosen frequency (f) and damping ratio (ζ) as shown in the equations below.

$$\omega_n = 2 * \pi * f \quad T = 10 / \zeta * \omega_n \quad K_p = \omega_n^2 + 2 * \zeta * \omega_n / T \quad K_d = 2 * \zeta * \omega_n + 1 / T$$

A sweep of the attitude control frequency shows the expected changes in the period of

oscillations of the error in the attitude, shown in Figure 7. Although the position control gains were not altered in this sweep changing the attitude gain also has an effect on the position, although a less pronounced one, as shown in Figure 8. This occurs because the thruster allocation couples the two problems of position and attitude control.

An interesting result of altering the gains is a significant effect on the fuel use, shown in Figure 9. As the frequency is raised, the control actions become more frequent and aggressive, as the gains on both the angular velocity and the angular error are increasing. With more frequent and larger demands being made on the thrusters, the spacecraft runs out of fuel much more quickly. In order to save fuel a slower but more efficient choice is a smaller frequency, in other words smaller gains.

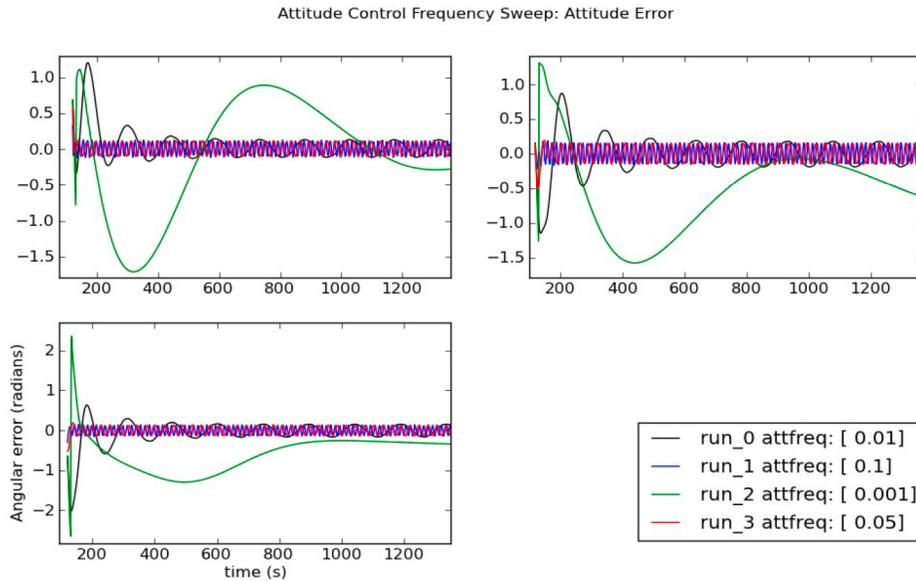


Figure 7: Three components of Attitude Error

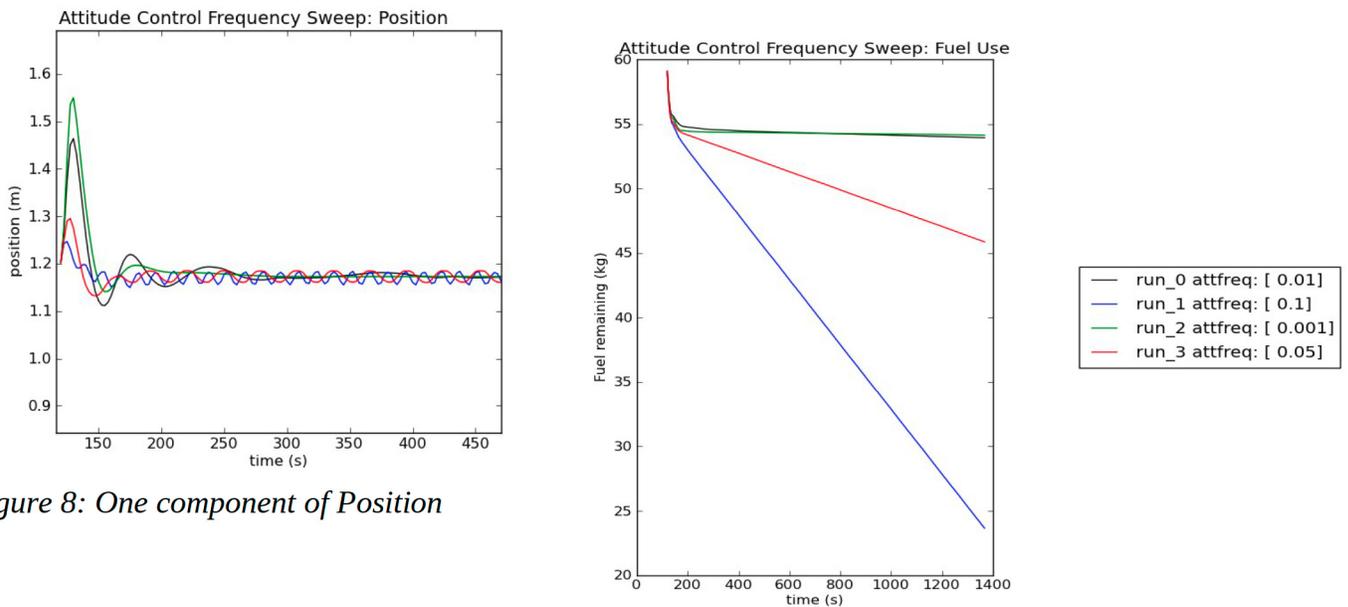


Figure 8: One component of Position

Figure 9: Fuel Consumption

A sweep of attitude damping ratios shows that a damping ratio closer to one provides more favorable results. In the plots of attitude error in Figure 10 the amplitude of the oscillations is decreased as the damping ratio approaches one. This is expected, as the case where the damping ratio is equal to one is expected for the error to decrease to zero without oscillations. Figure 11 illustrates the effect on the position of the spacecraft, and it can be seen that the smaller damping ratios reduce the ability to maintain a position. A larger damping ratio also decreases fuel consumption, as shown in Figure 12. It should be noted that very small damping ratios cause extremely undesirable results, where the oscillations in the attitude error do not converge within the time simulated. The damping ratio for other simulations unless otherwise mentioned is 0.707 for both attitude and position.

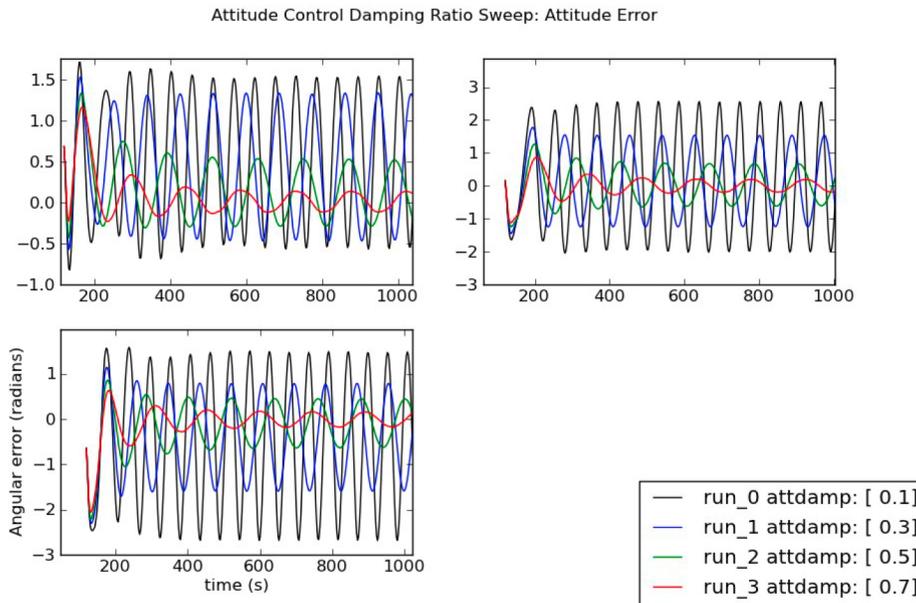


Figure 10: Three components of attitude error

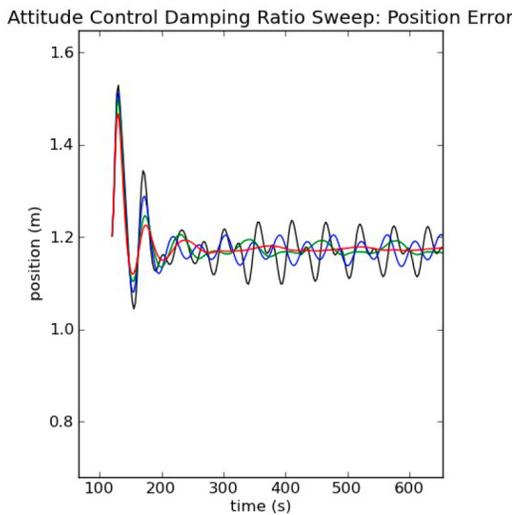


Figure 11: One component of position

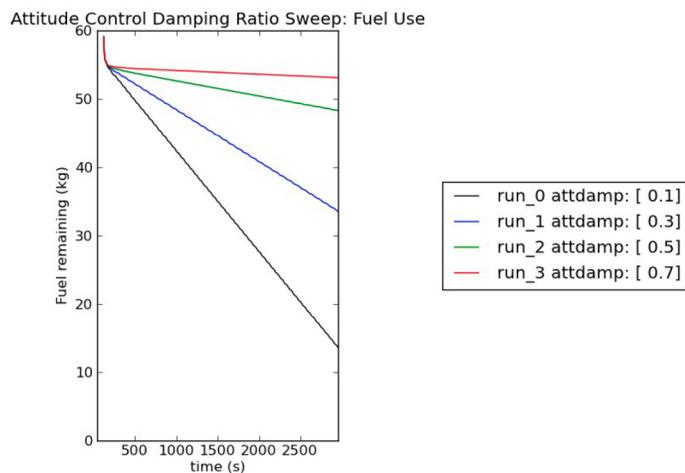


Figure 12: Fuel consumption

Monte Carlo Simulations

A Monte-Carlo simulation is where the inputs of a simulation such as the characteristics of control actuators or external forces are randomly altered to observe the effects of uncertainty. One potential use of these simulations is to account for errors in the construction of a spacecraft, for example a thruster may be installed with a slight error in its position or thrust direction. Since the control system is unaware of these errors and cannot take them into account, the control action differs from the ideal action.

Monte-Carlo simulations altering some characteristics of the spacecraft have been performed. In these simulations, the thruster control code assumed the nominal values, and made decisions on the percent of the maximum thruster level to apply to each thruster. One of these Monte-Carlo simulations varied the maximum level of a single thruster. With a mean as the nominal value of 3000 Newtons, and a standard deviation of 5 Newtons, 25 normally distributed thruster levels were given to the thruster located on the front, right, top corner of the spacecraft which points forward. Figure 13 Shows the results of comparing the position of the spacecraft to the nominal case. The plotted values are the norm of the difference between the position and the position from the nominal case, divided by the norm of the position from the nominal case. After an initial peak, the error from the nominal case reduces due to feedback.

Figure 14 shows the comparison between the angular error of the spacecraft compared to the angular error of the nominal case. These results show that within this simulation an uncertainty in the maximum level of only one thruster produces a very small uncertainty in the position – less than 0.0001 of the position expected from the nominal case after a short period of time, although the difference within the first few hundred seconds is much larger. The uncertainty in the angular error is a much larger fraction of the angular error itself, almost one third, and grows over time. The fuel consumption was also recorded, and at the end of 4000 seconds the total fuel used had a range of approximately 0.1 kg centered about the nominal case, where 7.5 kg had been used.

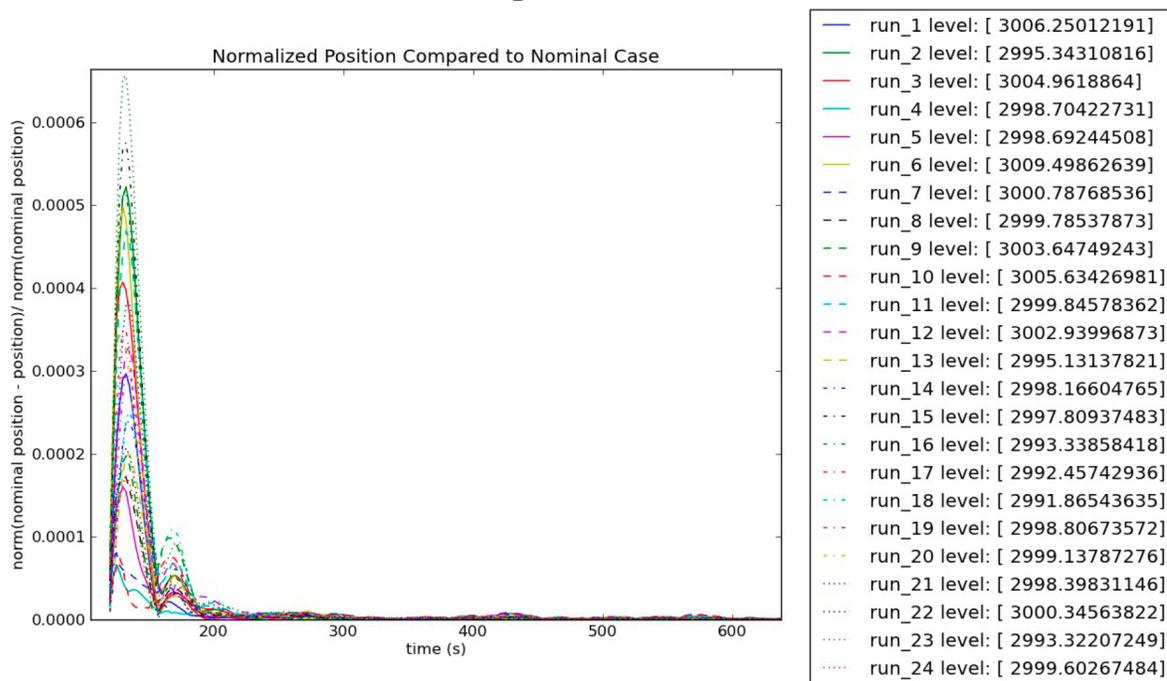


Figure 13: Relative position change with varied maximum thruster level

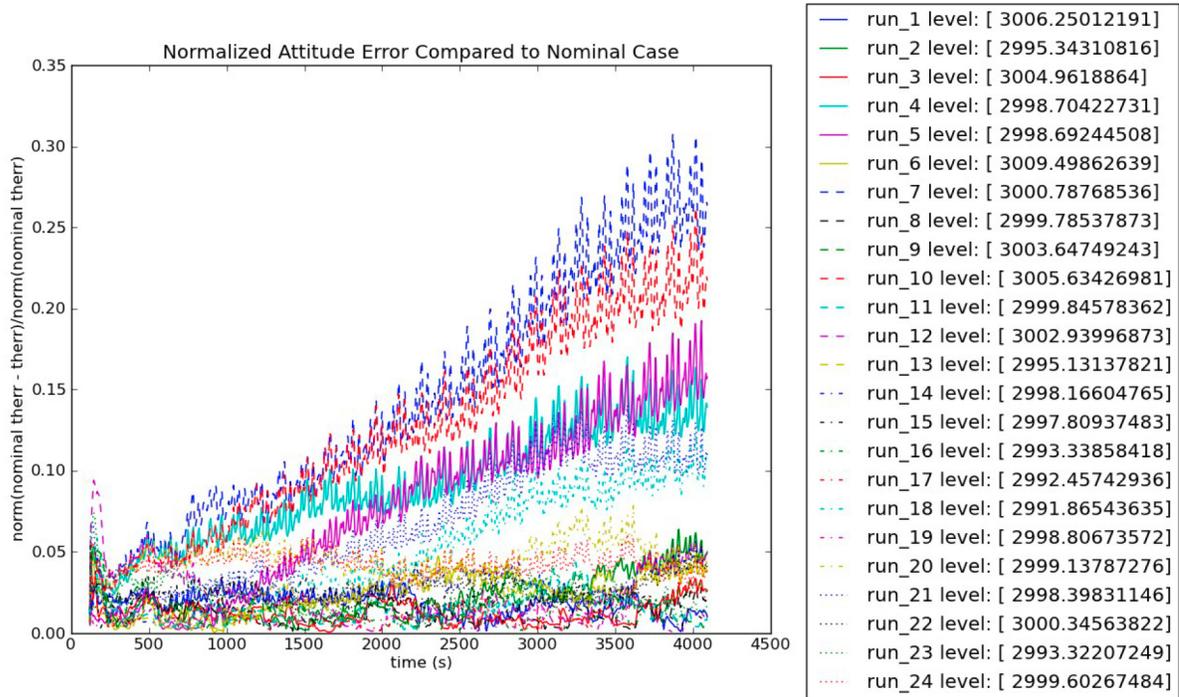


Figure 14: Relative attitude error with varied maximum thruster level

A Monte-Carlo simulation altering the location of the same thruster was also examined. The nominal position of this thruster is [0.837, -1.304, 0.593] meters, and the position was varied about this as the mean with a standard deviation of 0.1 meters. Similar to the first Monte-Carlo study, the resulting uncertainty in the final position, shown in Figure 15, is much less significant than the uncertainty in the angular error, shown in Figure 16. In this set of simulations, the fuel consumption at the end of 4000 seconds the total fuel used had a range of approximately 0.2 kg centered about the nominal case.

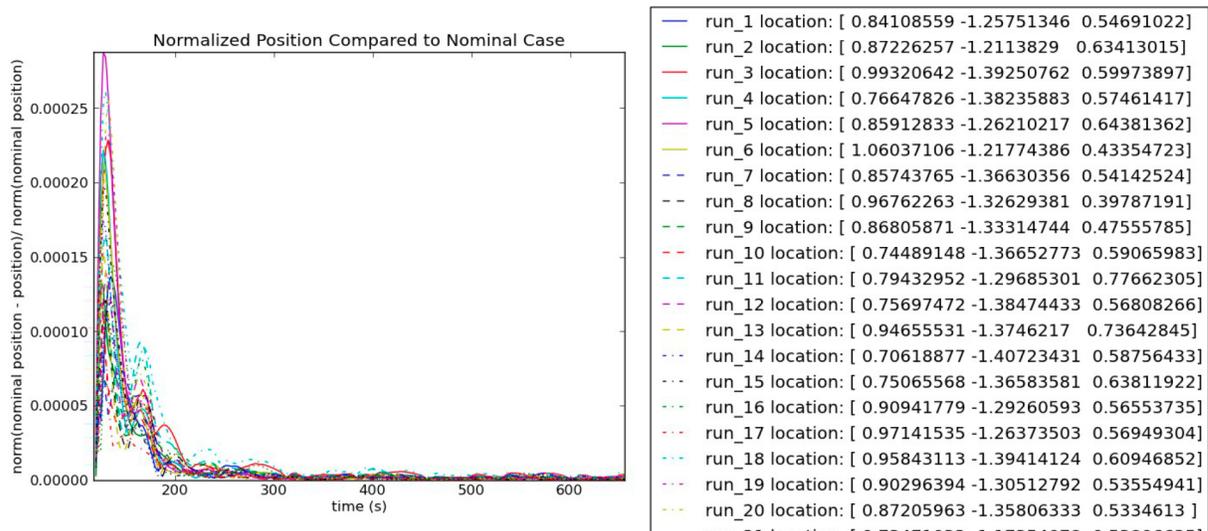


Figure 15: Relative position change with varied thruster location

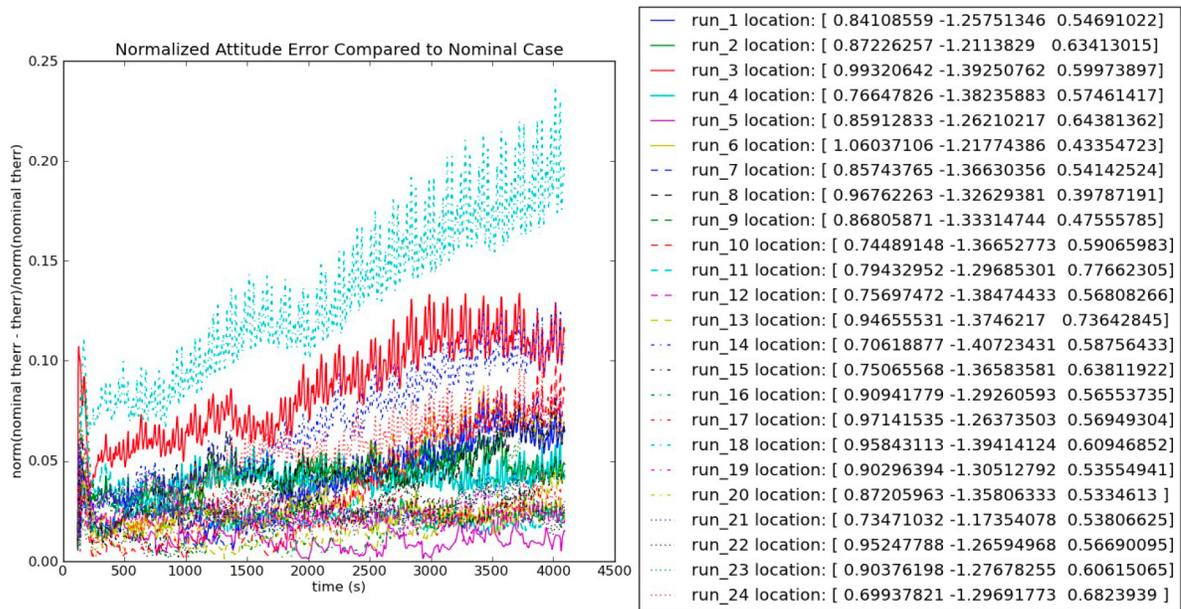


Figure 16: Relative attitude error with varied thruster location

Considering the results of both of these Monte-Carlo simulations, it appears that the orientation of the spacecraft is relatively sensitive to uncertainty in thruster properties while the position and fuel consumption are less sensitive.

DSENGS vs DshellCommon

Both DSENGS and DshellCommon are simulation frameworks based on Dshell which provide the ability to simulate spacecraft within the DARTS system. This includes various types of actuators such as thrusters, and the ability to create the simulation objects of the spacecraft and targets. The main difference between the DshellCommon system and the DSENGS system is that the DshellCommon system is not limited to only one spacecraft and one target. The simulation of the proposed MMSEV was moved to this new framework when it became possible to do so mid-way through the project. This required several changes including the configuration of the system as well as alterations to the names of variables in the simulation. To confirm that the new simulation was operating accurately, and that no errors had been introduced in switching over to the new system, the results of a simulation performed in each framework were compared. Small differences still exist, on the order of centimeters in the trajectory of the spacecraft, however these have been explained by a difference in the way that the two systems accommodate changes in fuel mass – the old system applies the fuel use after a simulation time-step, which the new one does it beforehand. This slightly changes the acceleration experienced by the spacecraft.

Conclusions

Over the course of this project, several capabilities have been added to the Itokawa simulation and this capability has been used to conduct parametric and Monte-Carlo studies. A more modular code structure was introduced. The capabilities added include thruster control, an alternative option for control allocation, and station-keeping. The non-rotating Station-keeping case showed satisfactory performance and the rotating case is in need of completion. Thruster allocation was implemented using several different methods. These methods were implemented in a modular way as well, and could be adapted to control allocation for other control systems in the future since they are not dependent on the use of thrusters. The ability of the simulated control system to correct for selected attitude, position,

and velocity perturbations was demonstrated.

Parametric sweeps of attitude control gains shows expected responses as well as demonstrating that the position control and attitude control are linked. They also shows how the gains can effect fuel consumption. The ability to run Monte-Carlo simulations was demonstrated with changes to a single thruster.

Future studies could explore more sophisticated Monte-Carlo simulation, as well as a more complete study of control gains. In addition, there are several more control allocation methods available from QCAT³ which could be included as options in the simulation. The ability to station keep over a rotating object should also be investigated further.

Acknowledgments

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the Jet Propulsion Laboratory Student Internship Program (JPLSIP) and the National Aeronautics and Space Administration. I would like to thank my mentor and comentor Bob Balaram and Marco Quadrelli, as well as the other members of the DARTS Lab.

Works Cited

1. "DSEENDS: Spacecraft Dynamics Simulator". Jet Propulsion Laboratory. July 2011
<<http://dshell.jpl.nasa.gov/DSEENDS/index.php>>
2. Sidi, Marcel J. Spacecraft Dynamics & Control: A Practical Engineering Approach. New York, NY: Cambridge University Press, 1997
3. Harkegard, Ola. "QCAT". Matlab Central; File Exchange. 11 Mar. 2004
<<http://www.mathworks.com/matlabcentral/fileexchange/4609-qcat>> Web. July 2011