

# Modeling and Simulation for Multi-Missions Space Exploration Vehicle

Max Chang<sup>1</sup>

*DARTS (Dynamics and Real Time Simulation) Lab  
Jet Propulsion Laboratory, California Institute of Technology*

Mentor: J (Bob) Balaram

**Asteroids and Near-Earth Objects [NEOs] are of great interest for future space missions. The Multi-Mission Space Exploration Vehicle [MMSEV] is being considered for future Near Earth Object missions and requires detailed planning and study of its Guidance, Navigation, and Control [GNC]. A possible mission of the MMSEV to a NEO would be to navigate the spacecraft to a stationary orbit with respect to the rotating asteroid and proceed to anchor into the surface of the asteroid with robotic arms. The Dynamics and Real-Time Simulation [DARTS] laboratory develops reusable models and simulations for the design and analysis of missions. In this paper, the development of guidance and anchoring models are presented together with their role in achieving mission objectives and relationships to other parts of the simulation. One important aspect of guidance is in developing methods to represent the evolution of kinematic frames related to the tasks to be achieved by the spacecraft and its robot arms. In this paper, we compare various types of mathematical interpolation methods for position and quaternion frames. Subsequent work will be on analyzing the spacecraft guidance system with different movements of the arms. With the analyzed data, the guidance system can be adjusted to minimize the errors in performing precision maneuvers.**

---

<sup>1</sup> USRP Fall 2011 Intern, DARTS Lab, Jet Propulsion Laboratory, Embry-Riddle Aeronautical University

## I. Introduction

In the Dynamics and Real-Time Simulation (DARTS) laboratory, simulation development is supported by several existing models. The C++ implementation of multi-body dynamics is in a module called Darts++ and supports the main dynamics computation. Near Earth Asteroid (NEO) #25143 (or Itokawa) models are used for the development of the NEO simulation. Itokawa has been visited by the Japan Aerospace Exploration Agency (JAXA)'s unmanned spacecraft Hayabusa and its gravity and detailed shape models are available for use in the simulation. A Python-based class design of spacecraft Guidance, Navigation & Control (GNC) is used to implement spacecraft on-board functions.

## II. Near Earth Object Simulation

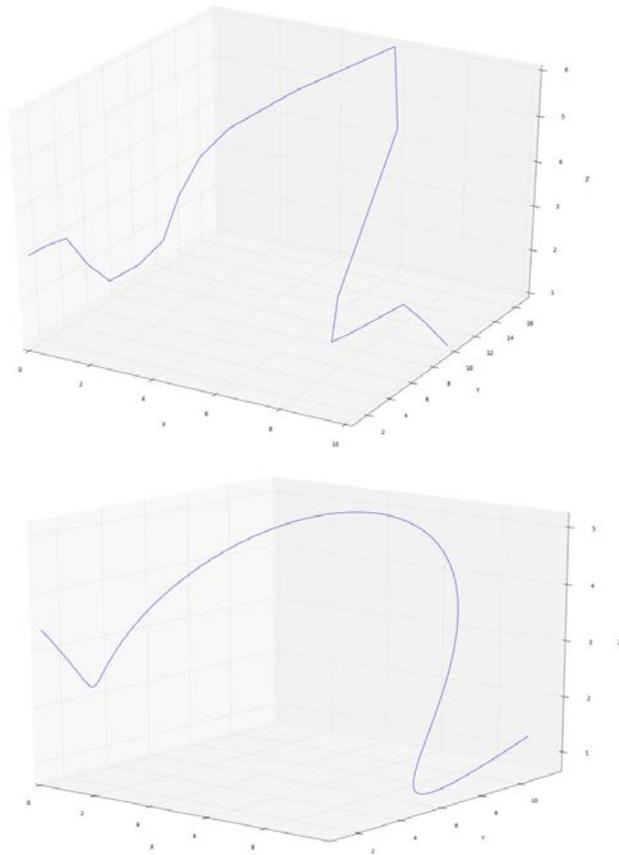
The NEO simulation models the interaction between the Multi-Mission Space Exploration Vehicle (MMSEV) and the asteroid. Guidance for station keeping with the rotating asteroid and approach to anchoring into the asteroid were the main simulation objectives for analysis. Trajectory design for these guidance objectives was supported by various interpolation algorithms implemented into the simulation.

### A. Interpolation Algorithm

Several types of interpolation methods are available for the analysis of minimizing the approximation errors. Depending on the desired result, the appropriate interpolation method should be selected to approximate the desired guidance trajectory.

Linear interpolation was developed as a reference method for the analysis. The linear interpolation function was constructed from a third party scientific computing tools package, NumPy<sup>4</sup>. Equal intervals were selected from the provided waypoints for the linear interpolation function. Linear interpolation was calculated directly from one waypoint to the following waypoint. Even though linear interpolation could be computed easier and faster, this method was not precise with the resulting approximation as shown in Fig. 1. The main concern from choosing linear interpolation method would be the abrupt change in direction at certain waypoints.

Polynomial interpolation was more accurate for analysis compared to linear interpolation. The polynomial interpolation function utilized was also from the NumPy library function. The function only used the interpolation interval, the waypoints, and the polynomial degree equation to fit the data. Polynomial interpolation could smooth the abrupt changes that occurred in linear interpolation. However, testing with polynomial interpolation has discovered the Runge's phenomenon which shows oscillation errors at the beginning and the end of the result. To reduce the Runge's phenomenon errors, Chebyshev nodes are used instead to interpolate with equal length intervals. Multiple tests with different ranges of waypoints have shown the polynomial interpolation method suitable for limited scenarios. Complex waypoints requiring higher degree could develop errors beyond the reduction capability made possible from using Chebyshev nodes.

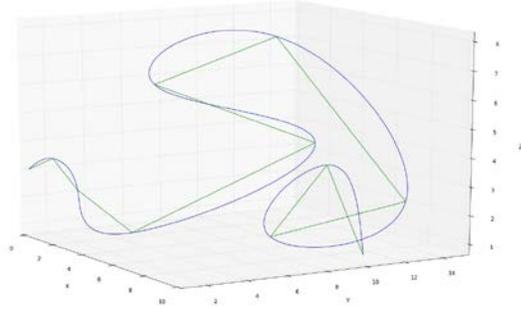


**Figure 1: The top plot shows the linear interpolation of the given waypoints and the bottom plot shows the polynomial interpolation of the same waypoints to the fifth degree.**

The Chebyshev nodes can be found using Eq. (1)<sup>3</sup>

$$x_i = \cos\left(\frac{(2(n-i)+1)\pi}{2n+2}\right), \quad i = 0, 1, \dots, n \quad (1)$$

The more accurate interpolation method for complex waypoints would be the cubic spline interpolation method. “For the piecewise cubic curves, if the geometrical constraints are modified for one of the cubic functions composing the curve, then only that piece of the curve and its immediate neighbors can be affected.”<sup>1</sup> Additionally, a cubic spline interpolation would not have the Runge’s phenomenon errors associated with the polynomial interpolation. The cubic spline curve would be calculated by solving the 4 unknowns of the cubic polynomial in Eq. (2) with represent an equation for the trajectory from one point to the next point.



**Figure 2: The green plot shows the linear interpolation of the provided waypoints. The blue plot shows the spline interpolation passed through all given waypoints.**

$$S_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i \quad (2)$$

To solve for the cubic spline parameters, the initial parameter was set to  $l_0 = 1, u_0 = z_0 = 0$  and the parameters from Eq (3) to Eq.(7) .

$$h_i = x_{i+1} - x_i \quad (3)$$

$$l_i = 2(x_{i+1} - x_{i-1}) - h_{i-1}u_{i-1} \quad (4)$$

$$u_i = \frac{h_i}{l_i} \quad (5)$$

$$a_i = \frac{3}{h_i} (y_{i+1} - y_i) - \frac{3}{h_{i-1}} (y_i - y_{i-1}) \quad (6)$$

$$z_i = \frac{a_i - h_{i-1}z_{i-1}}{l_i} \quad (7)$$

With the parameters calculated, the unknowns for the cubic polynomial equation were solved using interval from  $j = n - 1, n - 2, \dots, 0$  for Eq. (8),(9),(10).

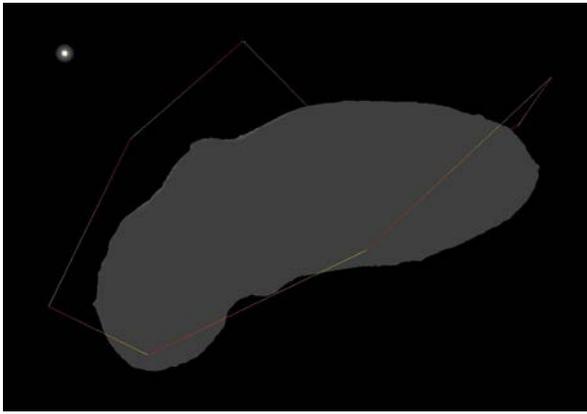
$$c_j = z_j - u_j c_{j+1} \quad (8)$$

$$b_j = \frac{y_{j+1} - y_j}{h_j} - \frac{h_j (c_{j+1} + 2c_j)}{3} \quad (9)$$

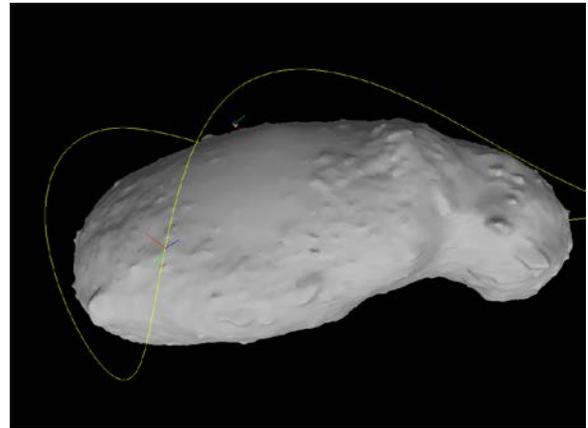
$$d_j = \frac{c_{j+1} - c_j}{3h_j} \quad (10)$$

The spline interpolation would result with an overall smooth curve through all the waypoints for the complex waypoints interpolation shown in Fig. 2.

The interpolation methods have been used to calculate trajectories in the simulation. Waypoints relative to the Itokawa asteroid were given, and the trajectory plots compared the different interpolation methods. The linear interpolation gives the expected result but is not feasible for actual spacecraft maneuvers as seen in Fig. 3. The sharp change in direction would require the spacecraft to fire thrusters longer and consume more fuel. For polynomial interpolation, the complex waypoints have caused unacceptable errors due to the Runge’s phenomenon. Thus, the interpolation result was unable to maintain a stable trajectory. The spline interpolation showed in Fig. 4 displayed the most promising trajectory with smooth curves through the waypoints.



**Figure 3: This simulation snapshot shows the interpolated trajectory from the provided waypoints.**



**Figure 4: This simulation snapshot shows the spline interpolation trajectory from the provided points. The frame axes show on the trajectory path was displaying the rotation from SLERP.**

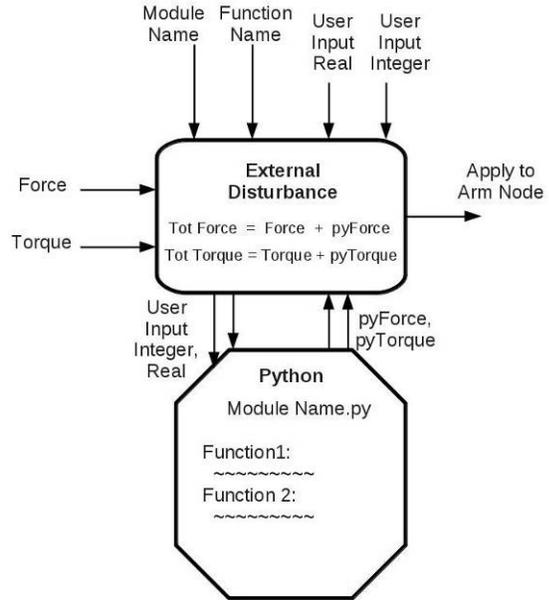
For three dimensional (3D) rotation, spherical linear interpolation (SLERP) is applied to calculate the quaternion to achieve the desired rotation. Unlike the spline or polynomial interpolation, SLERP would require the frame’s position in 3D or X, Y, Z and also the Euler’s angle  $\phi$ ,  $\theta$ , and  $\psi$ . The Spatial Operator Algebra (SOA) library has a quaternion function that can easily convert from Euler’s angle to quaternion. However, the function has a minor estimation error that could cause the norm of quaternion to be not equal to one. Normalizing the quaternion has solved the problem with the diminutive difference. Given at least two quaternion( $q_1, q_2$ ), SLERP was calculated using Eq. (11),(12) and the interpolation interval .<sup>2</sup>

$$q_1 \bullet q_2 = \cos(\delta) \quad (11)$$

$$slerp(q_1, q_2, u) = \frac{\sin([1-u]\delta)}{\sin(\delta)} q_1 + \frac{\sin(u\delta)}{\sin(\delta)} q_2 \quad (12)$$

**B. Models and Assemblies**

The NEO missions will require the MMSEV to deploy its arms and anchors into the surface of the asteroid. At the end-effector of each arm, an *ExternalDisturbance* model is attached to a node on the arm. The *ExternalDisturbance* actuator model applies forces and torques to the actuator node on which the model is attached. The torque and force data flow-ins are used to specify the applied force vector and the applied torque vector. The original *ExternalDisturbance* model had parameters of flow-in force, flow-in torque, and the frame of the disturbance. This would only allow a constant *ExternalDisturbance* force and torque with respect to time. To make the model more flexible in applying changing forces and torques with respect to time, additional forces and torques to be added to the *ExternalDisturbance* model are calculated using a user-provided Python function. The C++ modules are extended with a Python interpreter with new modules to run the Python function within the C++ module. The modified *ExternalDisturbance* model is added to the model library with new parameters for Python function module name, Python function name, user input integer numbers, and user input real numbers. The user input integer numbers and user input real numbers are passed through to the Python function to enable control of the function. The Python function calculates the additional forces and torques from contact with the ground (spring-damper in the vertical axis, and damper in horizontal axes), and returns the forces and torques calculated back to the *ExternalDisturbance* model. The *ExternalDisturbance* model sums the forces and torques from the original flow-in and the Python function return values and applies it to the arm node. A schematic diagram is shown in Fig. 5 to illustrate all the parts of the *ExternalDisturbance* model.



**Figure 5: This is a schematic diagram of the External Disturbance class model. It shows the input/output of the model and the interface with a Python class.**

One problem faced during the development of extending C++ with Python was in passing the entire user input array for both integers and real numbers to the Python function. It would only function correctly with passing one number and was incapable of working with an entire number array. After investigation, the problem was determined to be caused from initialization of the user input parameter with an incorrect size. After directly assigning the correct number of user input values, the Python function was able to pass through all the user input integer and real numbers. Another problem encountered was on getting the Python function to return correct forces and torques to the

```

'TopStandoffArm' : {
  'class' : 'MMSEVArmAssembly',
  'nlinks' : 7,
  'motorType' : 'PrescribedPin',
  'poses' : {
    'zero' : [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    'deployed1' : [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0],
    'deployed2' : [2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0],
  },
  'context' : { 'parentBody' : 'TopStandoffBaseBody2' },
  'params' : { 'Bodies' : bodies.Library['MMSEV']['Bodies'],
    'Nodes' : bodies.Library['MMSEV']['Nodes']['TopStandoffArm'],
    'StowWaypoints' : arms.Library['MMSEV Waypoints']['TopStandoffArm Stow Waypoints'],
    'UnstowWaypoints' : arms.Library['MMSEV Waypoints']['TopStandoffArm Unstow Waypoints'],
    'UserInputInt' : UserInputIntParam(params={'name':[1, 0, 0]},source=None),
    'UserInputReal' : UserInputRealParam(params={'name':[1.0, 0, 0]},source=None),
    'PyExtDistModuleName' : PyExtDistModuleNameParam(params={'name':'InteractionTest'},source=None),
    'PyExtDistFunctionName' : PyExtDistFunctionNameParam(params={'name':'pyInteraction'},source=None)
  },
},

```

**Figure 6: An example of a configuration section declared parameters to be passed down to the assembly and ExternalDisturbance modules.**

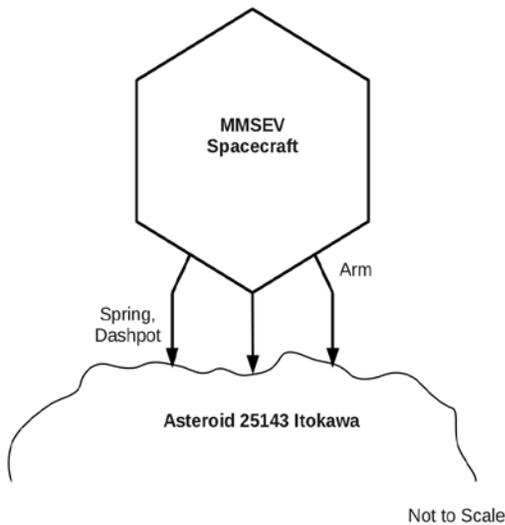
*ExternalDisturbance* model. The problem was caused by an error in assigning and calling the return value name. A specific process to call a return value from Python to C++ was originally implemented incorrectly and caused the wrong return value to the *ExternalDisturbance* model. After getting help from other colleagues and studying other similar examples of embedding Python in a C++ model, the *ExternalDisturbance* model is now able to receive the calculated forces and torques from the Python function.

Next, the *ExternalDisturbance* model C++ class is built into the assembly for the MMSEV anchoring arms. When each arms was assembled, the additional parameters for disturbance forces and torques calculation are added with flags to address each arm to the corresponding end-effectors’ node. The adjustable input parameters are passed down from the configuration to the arms assembly, and from the arm assemblies down to the external disturbance model. Figure 6 shows the configuration for one of the arm assembly with the parameters to call the Python function within the file.

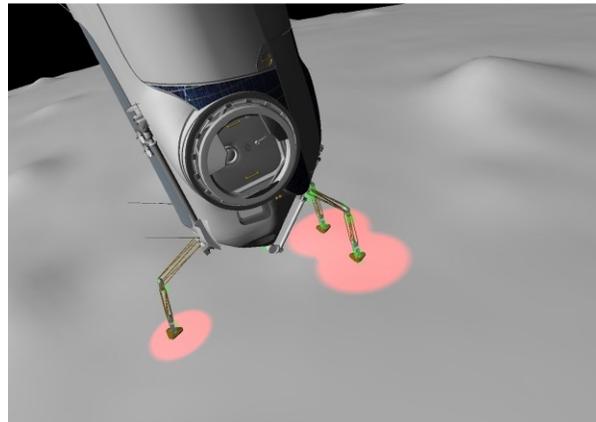
**C. Anchoring State**

Asteroids have their own rotation which makes it difficult to station keep and land. To save fuel during station keeping in orbits around the asteroid and to conduct sample collection, anchoring into the asteroid was one of the main objectives. MMSEV would be anchoring into the asteroid with three arms whose interaction with the surface is modeled as a spring and damper at the end-effector as shown in Fig. 7. When the spacecraft is in the position to deploy the arm for the anchoring state, geometric collision is detected of the end of the arm with the surface of the asteroid. The spacecraft would then enter an anchoring state when all three arms have anchored into the surface.

For simulation visualization, anchoring locations are highlighted with respect to the *ExternalDisturbance* forces. As the forces increased, the highlighted radius would also increase by a ratio. In Figure 8, the left arm has a smaller disturbance force than the right two arms; therefore, the highlight radius of the left contact point is smaller. The highlighted graphics could display the change in the disturbance forces when other maneuvers are performed by the spacecraft.



**Figure 7: An illustration of the spacecraft anchored to the asteroid with the spring and damper on the end effector of the three arms.**



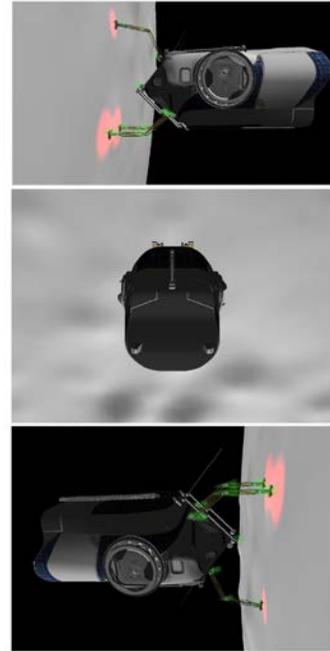
**Figure 8: A simulation snapshot displayed the contact points with the highlight radius reference to disturbance forces.**

During testing and analysis in the anchoring state, the numerical results show several unexpected large force vectors. When the spacecraft simulation is set to a purely dynamic mode after anchoring, the spacecraft would go unstable from the unexpected large forces generated by the spring. The spring and damper constant used to calculate the disturbance forces and torques could be one source of error. The possible problem causing the errors are the frame transformation calculation used to obtain the force vectors. When the force interaction function problems have been fixed, further analysis could be directed at applying different spring and damper coefficients. The analysis could help determine the appropriate spring and damper for anchoring into the surface of an asteroid.

#### D. Camera View and Guidance Chase frame

To make a movie to present the simulation work, a camera view frame has to be easily maneuvered during the real-time simulation. Using the trajectory interpolation algorithm developed, the camera view frame is interpolated between the desired camera view locations. The camera attached to a scene frame relative to the spacecraft and its frames are updated in real-time. Furthermore, the desired camera positions and rotation angle are relative to the spacecraft body. The movie maker can easily maneuver the camera view with key camera locations. The function would interpolate the maneuver to transition smoothly from one camera frame to the next one. Different camera view frames are shown in Fig. 9 to present possible scenes of the movie.

Guidance frames are developed with similar tools as the camera view frames. A guidance frame is used as a spacecraft chase frame and the planned course was compared with the actual spacecraft course. The guidance frame function is developed to update in real-time with the simulation and to display a chase frame for ease of comparison. The guidance frame is created with reference as the Itokawa body frame. The nominal frame and quaternion information were also with respect to the body frame of Itokawa. The guidance frame function could also be applied to maneuver the spacecraft's arms with the implementation of an inverse kinematics algorithm.



**Figure 9: Top figure shows the left camera view. Middle figure shows the back camera view. Bottom figure shows the right camera view.**

### III. Conclusion

Guidance interpolation was developed using linear, polynomial and spline interpolation. Spline interpolation is more accurate than the other two methods. A force interaction model was implemented into the arm assemblies. Visualization of anchoring location was added with a visual indication of the force vector. Subsequent work would focus on fixing the existing known errors, conducting more analysis with different interaction models, and developing robotic arm's movement with existing tools.

#### Acknowledgments

This project was carried out at the Jet Propulsion Laboratory, California Institute of Technology and was sponsored by the NASA University Student Research Program. The author would like to thank mentor Bob Balaram for his guidance and support. Also, gratitude to Abhinandan Jain, Calvin Kuo, Steven Myint, and all the other DARTS Lab members for their assistance with this project and Petra Kneissl-Milanian for organizing the USRP program.

#### References

- <sup>1</sup>Lengyel, Eric, *Mathematics for 3D Game Programming & Computer Graphics*, 2<sup>nd</sup> ed., Charles River Media, Boston, 2003, Chaps. 15, pp. 468-472.
- <sup>2</sup>Parent, Rick, *Computer Animation: Algorithms and Techniques*, 2<sup>nd</sup> ed., Morgan Kaufmann, Burlington, 2008, Chaps 3, pp. 110 – 112.
- <sup>3</sup>Stewart, Gilbert W., *Afternotes on Numerical Analysis: a Series of Lectures on Elementary Numerical Analysis Presented at the University of Maryland at College Park and Recorded after the Fact.*, Society for Industrial and Applied Mathematics, Philadelphia, 1996, pp. 152.
- <sup>4</sup>Numpy, Software Package, Ver. 1.5.1, Numpy Developers, 2010.