HALO Orbit Determination, Integration, and Station Keeping

Ethan Geipel
April 25, 2019

Abstract

The primary goal for this report was to calculate, and integrate HALO orbits (determination) and to implement a station keeping method for a spacecraft in the Earth-Sun system. In this report, HALO orbits were successfully generated via initial third order approximation, and differential corrections were used to converge to a nominal periodic solution. These nominal orbits where then successfully used to implement a station keeping strategy that successfully maintains the orbit. Station keeping outlined in this report has ΔV and coast times that are comparable to real-life HALO missions such as the Solar Heliospheric Observatory (SOHO) mission. Over the course of 32 periods ($5700\ days$) the station keeping method implemented in this report required a total ΔV of $15.1\ m/s$, which can be accomplished using $16.2\ kg$ of fuel. Details of this process, comparisons to SOHO, and proposed future work are elaborated on within the report. The MATLAB code used to perform all of the above mentioned aspects to this report can be found in the Appendix, and some supplementary plots, figures, and animations of station keeping can be found at ethan-geipel.com/HALO.

To understand HALO orbits, it is important to understand what Lagrange points are. Lagrange points stem from the Circular Restricted Three Body Problem (CRTBP) as the solution where the gravitational forces between the two primary masses "cancel out", allowing the third (negligible mass) spacecraft to remain at a fixed point relative to the two primary masses. Five such points exist in the CRTBP, shown in Figure 1. While L4 and L5 are stable for systems where $\frac{m_1}{m_2} > \sim 25$, spacecraft aiming to remain at the colinear L points require a little more finesse to stay in place.

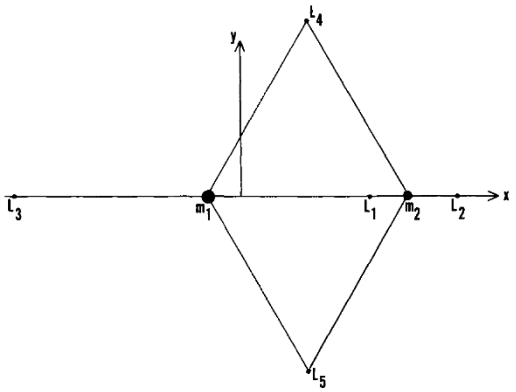

Figure 1 - Lagrange points in CRTBP. (Howell 1983)

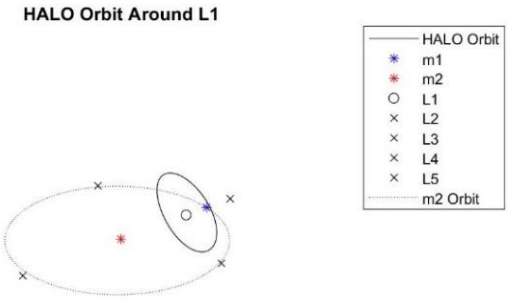

Figure 2 - HALO orbit around L1 ($\mu = 0.04$).

The primary strategy for maintaining an orbit near these colinear L points is to utilize the HALO orbit. HALO orbits are fascinating as they involve orbiting the spacecraft about the points in space where the mathematical elements L1, L2, and L3 exist – effectively orbiting an "empty" point in space (see Figure 2). Unfortunately, HALO orbits about the co-linear pints are typically unstable, like their L point counter parts. There is, however, enough existing orbital mechanics knowledge to allow for station keeping methods to be implemented in these orbits to keep the spacecraft in a relatively stable orbit at the cost of low ΔV burns every so often, as deviations grow.

The equations of motion for the CRTBP assumes two primary point-masses, and utilizes a rotating coordinate frame such that the two primary masses are colinear on the x-axis (centered at the barycenter). A third body of negligible mass can then be introduced, and the gravitational effects of the two primary masses can be determined.

Taking $x, y,$ and z to be relative to the barycenter of the system, the equations of motion that describe the CRTBP are the following:

$$x'' - 2y' = \frac{\partial \tilde{V}}{\partial x}, \qquad y'' + 2x' = \frac{\partial \tilde{V}}{\partial y}, \qquad z'' = \frac{\partial \tilde{V}}{\partial z} \tag{1}$$

Using μ as the ratio of the mass, $\frac{m_1}{m_2}$, the full set of equations includes:

$$\tilde{V} = \frac{1}{2}(x^2 + y^2) + \tilde{U}, \qquad \tilde{U} = \frac{1-\mu}{r_1} + \frac{\mu}{r_2},$$
$$r_2 = \sqrt{(x+\mu)^2 + y^2 + z^2}, \qquad r_3 = \sqrt{(x-1+\mu)^2 + y^2 + z^2} \tag{2}$$

Integration of these differential equations yields the trajectories of the third body in the rotating reference frame, relative to the two primary masses. Computational integration was done in MATLAB – the process for obtaining initial conditions, converging to a stable periodic orbit, and station keeping implementation are detailed in the upcoming Methodology section.

Methodology

In order to explore the definition, integration, and station keeping of HALO orbits, there were several steps that had to be linked together. First, the normalization of variables, integration of the State Transition Matrix and Equations of motion had to be accounted for. Second, the initial conditions of a HALO orbit had to be calculated based on characterizing parameters. Third, convergence of the initial HALO orbit to a periodic solution. Fourth, implementation of station keeping ΔV's to course-correct any deviations from the nominal orbit. Finally, the station keeping method could be examined and compared to known mission profiles, such as the Solar Heliospheric Observatory (SOHO) mission. Further details on each of these steps is shown below.

Normalization, State Transition Matrix, and Runge-Kutta-Merson Integration Method

Normalization: There are two methods of normalization used at different points in this paper. The first normalization occurs when implementing Richardson's third order approximation and initial condition calculations. The normalization used in the third order approximations is as follows.

Distance: $x = \frac{x}{r_e}$, where r_e is the M_2-$L1$ distance.

Time: $s = nt$, where n is the mean motion, and t is time.

The second normalization, occurs when integrating the equations of motion. All integration is under the following normalization.

Distance: $x = \frac{x}{r_{au}}$, where r_{au} is the distance between the two primaries.

Time: $G(m_1 + m_2) = n^2 a^3 = 1$, which is essentially that the mean motion is equal to 1.

State Transition Matrix: The State Transition Matrix (STM) that is integrated alongside the equations of motion was constructed as follows:

$$\Phi_{6\,x\,6} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{V_{xx}} & 2\mathbf{\Omega} \end{bmatrix} \quad (3)$$

Where $\mathbf{0}$ is a $3x3$ zero matrix and $\mathbf{I}$ is a $3x3$ Identity matrix. Additionally, $\Omega = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ and $\mathbf{V_{xx}}$ is a symmetric $3x3$ matrix of second partial derivatives $\mathbf{V_{xx}} = \begin{bmatrix} \frac{\partial^2 V}{\partial x^2} & \frac{\partial^2 V}{\partial x \partial y} & \frac{\partial^2 V}{\partial x \partial z} \\ \frac{\partial^2 V}{\partial y \partial x} & \frac{\partial^2}{\partial y^2} & \frac{\partial^2 V}{\partial z} \\ \frac{\partial^2 V}{\partial z \partial x} & \frac{\partial^2 V}{\partial z \partial y} & \frac{\partial^2 V}{\partial z^2} \end{bmatrix}$ where V is the potential Equation *(2)*.

Integration: For all integration performed in this paper, a variation on the standard Runge-Kutta integration technique was used – called Runge-Kutta-Merson. This integration takes the following form. (Lukehart 1963)

$$\begin{aligned} \eta_0 &= Y_0 & k_0 &= f(t_i, \eta_0, \mu) \times dt \\ \eta_1 &= \eta_0 + \tfrac{k_0}{3} & k_1 &= f\left(t_i + \tfrac{dt}{3}, \eta_1, \mu\right) \times dt \\ \eta_2 &= \eta_0 + \tfrac{k_0 + k_1}{6} & k_2 &= f\left(t_i + \tfrac{dt}{3}, \eta_2, \mu\right) \times dt \\ \eta_3 &= \eta_0 + \tfrac{k_0 + 3k_2}{8} & k_3 &= f\left(t_i + \tfrac{dt}{2}, \eta_3, \mu\right) \times dt \\ \eta_4 &= \eta_0 + \tfrac{k_0 - 3k_2 + 4k_3}{2} & k_3 &= f(t_i + dt, \eta_4, \mu) \times dt \end{aligned}$$

$$Y = \eta_0 + \frac{k_0 + 4k_3 + k_4}{6}$$

Where Y_0 is the initial conditions, Y is the returned integrated value, and function $f(t, \eta, \mu)$ implements the equation of motion, Equation *(1)*, and the STM, Equation *(3)*.

Richardson's Third Order Approximation for Initial Conditions

HALO orbits are often described by characteristic parameters such as their out-of-plane oscillation amplitudes: (A_x, A_y, A_z). Taking these oscillation amplitudes, the mean motion of the system, and the ratio of the mass (μ), the initial conditions for a HALO orbit can be determined. To obtain these initial conditions, a third order approximation of the equations of motion (above) is made. To the first order, the solutions are simply the following (Richardson 1979).

$$x = -A_x \cos(\lambda s), \qquad y = k\, A_x \sin(\lambda s), \qquad z = A_z \sin(\lambda s)$$

Derived by Richardson, the third order approximations are the following (see Richardson reference for analytic variable construction):

$$\begin{aligned} x &= a_{21}A_x^2 + a_{22}A_z^2 - A_x\cos(\tau_1) + (a_{23}A_x^2 - a_{24}A_z^2)\cos(2\tau) + (a_{31}A_x^3 - a_{32}A_xA_z^2)\cos(3\tau) \\ y &= kA_x\sin(\tau) + (b_{21}A_x^2 - b_{22}A_z^2)\sin(2\tau) + (b_{31}A_x^3 - b_{32}A_xA_z^2)\sin(3\tau) \\ z &= \delta_n A_z\cos(\tau) + \delta_n d_{21}A_xA_z(\cos(2\tau) - 3) + \delta_n(d_{32}A_zA_x^2 - d_{31}A_z^3)\cos(3\tau) \end{aligned} \qquad (4)$$

The velocities in the orbit are obtained by simply taking the derivative of the above position equations. At time $\tau = 0$, all $\sin(\tau)$ terms will go to zero, leading to the conclusion that $y(\tau = 0) = 0$, $\frac{dx}{dt}(\tau = 0) = 0$, and $\frac{dz}{dt}(\tau = 0) = 0$. The resulting initial conditions vector is therefore: $[x_0,\ 0,\ z_0,\ 0,\ \dot{y}_0,\ 0]$.

The initial conditions provided by Richardson's third order approximation are decent, but they are not fully periodic, as there are still slight deviations in the positions and velocities that cause the orbit to lose stability before the orbit has been completed.

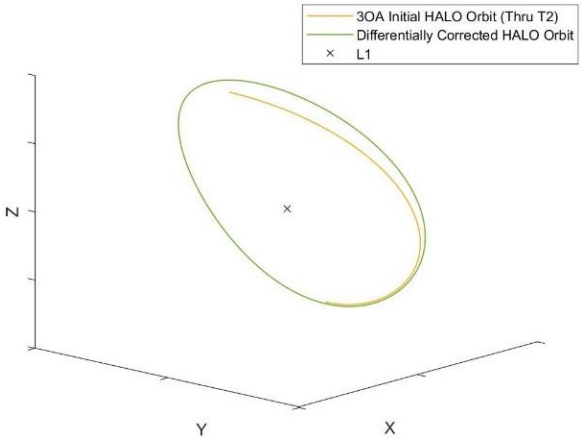

Figure 3 - Comparison of third order approximation to nominal, highlighting the need for differential correction.

Howell's Differential Correction and Convergence to Periodic HALO Orbit

To bridge the gap from the third order approximation to a satisfactory nominal HALO orbit, a method of differential corrections is implemented. There are multiple ways to accomplish this correction – in this report, a method outlined by Kathleen Howell is used. The algorithm Howell describes takes advantage of a known fact about HALO orbits which is that at $\tau = \frac{T}{2}$, the x and z velocity components should be equal to zero. By integrating the orbit until the y position has changed signs (indicating $\frac{T}{2}$), the values of the x and z velocities can be found and set as $\delta\dot{x} = -\dot{x}$ and $\delta\dot{z} = -\dot{z}$. From here, the differential correction can be calculated using the following (Howell 1983).

$$\delta X = \Phi(T/2, 0)\delta X_0 + \frac{\partial X}{\partial t}\delta(T/2)$$

There are two ways that this can be implemented via algorithm: 1) change z_0 and $\dot{y}_0$, or 2) change x_0 and $\dot{y}_0$. In the implementation of differential corrections used in this paper, both "versions" where implemented in alternating fashion until the desired tolerance was met. The equations to achieve both of these are as shown below, respectively.

$$\begin{pmatrix}\delta\dot{x}\\ \delta\dot{z}\end{pmatrix} = \left[\begin{pmatrix}\Phi_{43} & \Phi_{45}\\ \Phi_{63} & \Phi_{65}\end{pmatrix} - \frac{1}{\dot{y}}\begin{pmatrix}\ddot{x}\\ \ddot{z}\end{pmatrix}\begin{pmatrix}\Phi_{23} & \Phi_{25}\end{pmatrix}\right]\begin{pmatrix}\delta z_0\\ \delta\dot{y}_0\end{pmatrix}$$
$$\begin{pmatrix}\delta\dot{x}\\ \delta\dot{z}\end{pmatrix} = \left[\begin{pmatrix}\Phi_{41} & \Phi_{45}\\ \Phi_{61} & \Phi_{65}\end{pmatrix} - \frac{1}{\dot{y}}\begin{pmatrix}\ddot{x}\\ \ddot{z}\end{pmatrix}\begin{pmatrix}\Phi_{21} & \Phi_{25}\end{pmatrix}\right]\begin{pmatrix}\delta x_0\\ \delta\dot{y}_0\end{pmatrix} \quad (5)$$

After solving these equations for δz_0 and $\delta\dot{y}_0$ or δx_0 and $\delta\dot{y}_0$, the solutions are added back in to the initial conditions and the process is iterated through again. This continues until the errors $\delta\dot{x}$ and $\delta\dot{z}$ reach a tolerance of $\delta < 10^{-15}$.

Station Keeping ΔV Maneuver Calculations

The station keeping method implemented in this paper is based on the station keeping strategy used in the Genesis mission (launched August 2001). (Williams, et al. 2000) The strategy for station keeping relies on a simple equation for calculating the required ΔV.

$$\Delta V = -\mathbf{B}^{-1}\mathbf{A}\,\overline{\delta x} - \overline{\delta v} \quad (6)$$

Where $\overline{\delta x}$ and $\overline{\delta v}$ are the difference in spacecraft position and velocity from the nominal trajectory, respectively. The $3x3$ matrices **A** and **B** come from the STM:

$$\Phi(t_t, t_M) = \begin{bmatrix}A & B\\ C & D\end{bmatrix}$$

Where t_M is the time of the maneuver, and t_t is the time at some target location "downstream" of the maneuver location. In the station keeping strategy, it is advised to perform 3-4 station keeping maneuvers per HALO orbit, which works out to around 1 maneuver every 2 months. This maneuver frequency is quite intermittent which allows for minimal effect on science or other mission operations.

Station Keeping Perturbation

Testing the station keeping strategy was done by inducing a random perturbations of magnitudes within range:

$$\delta Y_0 = [|\delta x_0| < 1\ km \quad |\delta y_0| < 10\ km \quad |\delta z_0| < 1\ km \quad |\delta\dot{x}_0| < 1\ m/s \quad |\delta\dot{y}_0| < 10\ m/s \quad |\delta\dot{z}_0| < 1\ m/s]$$

Where $Y_0 = Y_0 + \delta Y_0$ induces the perturbation prior to integrating the equations of motion. Findings from this exploration are detailed further on in the Findings and Results.

Additional Notes

For future reference, there is a conversion between the L point frame of reference and that of the CRTBP which has to be accounted for when integrating results from Richardson. The "conversion" is as follows:

$$\dot{y}_{CRTBP} = x_{L1} \times au - \left(x_{L1} \times* au \times n + \dot{y}_{0,Richardson}\right)/n$$

Where x_{L1} is the normalized distance from barycenter to L1, n is the mean motion, and au is the distance between the two bodies (1 au in this case).

Also to note, the classical rocket equation $\Delta V = I_{sp} g_e \ln(m_0/m_f)$ was used to determine how much fuel is needed to maintain an orbit for a $m_0 = 1850\ kg$ spacecraft (SOHO).

Verification

To verify that the above steps were implemented correctly, the following "tests" were performed, using reference material provided by Richardson and Howell.

Verifying Third Order Approximation

To verify the third order approximation, the oscillation amplitudes that Richardson used in his report "HALO Orbit Formulation for the ISEE-3 Mission" (Richardson 1980).

$$A_z = 110{,}000\ km, \qquad A_x = 206{,}000\ km, \qquad A_z = 665{,}000\ km$$

The periodic orbit that Richardson identifies from these oscillation amplitudes is the following.

$$Y_{Richardson} = [2.45 \times 10^5\ km \quad 0\ km \quad -1.00 \times 10^5\ km \quad 0\ m/s \quad -291.7\ m/s \quad 0\ m/s]$$

From the third order approximation, the implemented function returns the following initial conditions.

$$Y_{HALO\_3OA.m} = [2.39 \times 10^5\ km \quad 0\ km \quad -1.14 \times 10^5\ km \quad 0\ m/s \quad -291.0\ m/s \quad 0\ m/s]$$

For third order approximation, these initial conditions are in line with what we might expect for a non-corrected periodic orbit.

Verifying Differential Corrections

To verify the implemented differential correction algorithm, the initial conditions from Richardson were input, with the hope that integration of the resulting initial conditions would result in a periodic orbit. The differential correction algorithm, alternating between the two variants in Equation *(5)*, reaches the tolerance of $\delta < 10^{-15}$ after roughly 6 iterations, and yields the following initial values.

$$Y_{differentialCorrection} = [2.47 \times 10^5\ km \quad 0\ km \quad -1.16 \times 10^5\ km \quad 0\ m/s \quad -292.2\ m/s \quad 0\ m/s]$$

These values, when integrated, result in a periodic orbit (see Figure 4). Variations of these differential corrections and $Y_{Richardson}$ are minimal with the exception of z_0, which is likely due to the alternating nature of allowing x_0 and z_0 to vary while the other is held "in place" – the initial velocity $\dot{y}_0$ differs by a half a meter per second as a result. A visual representation of the differential correction is shown in Figure 4 and Figure 5 below.

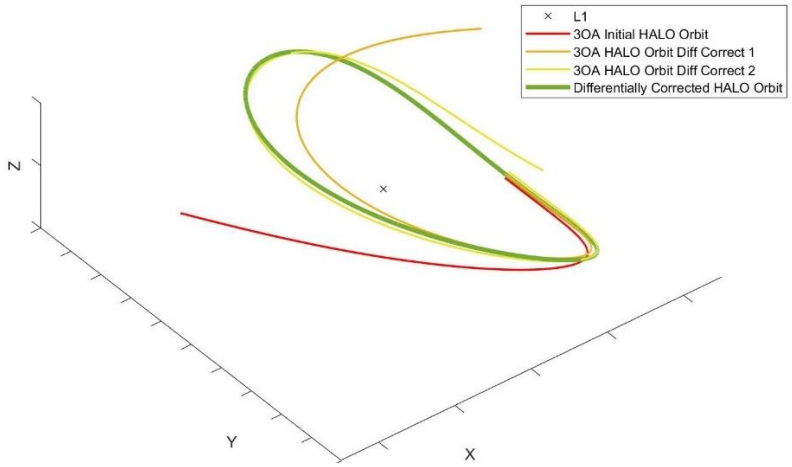

Figure 4 - Progression of differential correction algorithm.

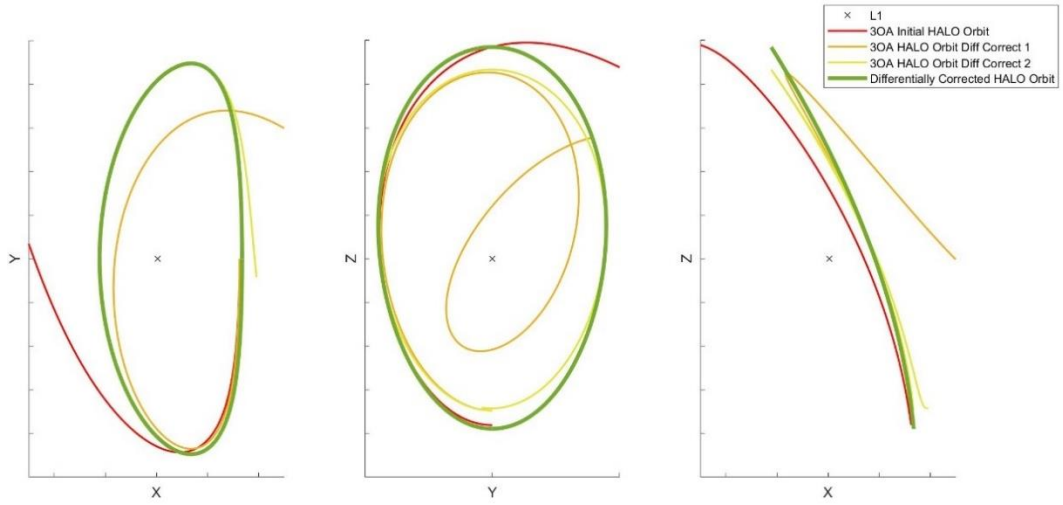

Figure 5 - Progression of differential corrections. (Planar views)

Station Keeping Verification

The station keeping strategy was verified by inducing a perturbation, $Y_0 = Y_0 + \delta Y_0$ (described above) and allowing the correction ΔV to be implemented whenever the orbit exceeds a position or velocity tolerance bound. The target point "downstream" of the maneuver for this verification occurred at approximately $t_t = 15\ days$. The best way to demonstrate the validity of this implementation is with a visual example. The red circles in the corrected plot indicate locations of station keeping maneuvers.

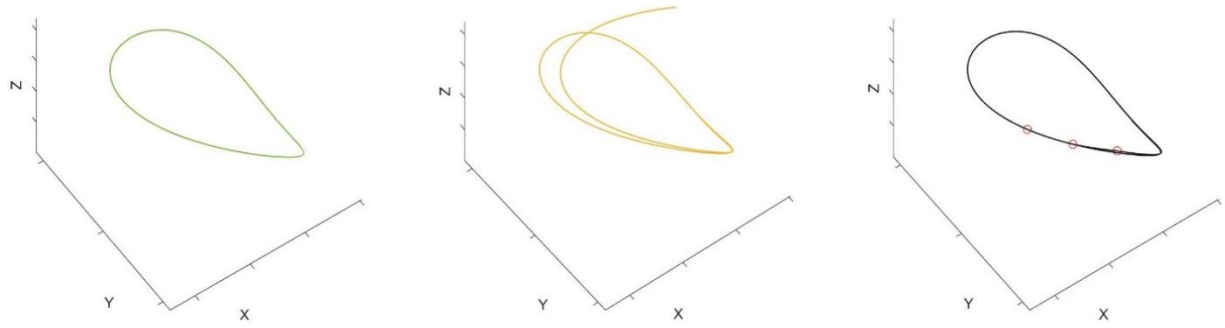

Figure 6 - Station keeping verification. Two periods after an induced perturbation. (Left: Nominal, Middle: Uncorrected, Right: Station-kept)

By inspection, the station keeping implementation was successful. However, the strategy implemented for this verification is not necessarily comparable to the station keeping strategy used for real-life missions. Creating a station keeping strategy that is comparable to previous missions (Genesis and SOHO) took several iterations, and is explored next in the Findings and Preliminary Results.

After verifying that all implemented code was functional, it was time to begin studying station keeping strategy and to compare it to previous mission results. The main real-life mission used as a reference is SOHO, which has available ΔV vs time information. Over time the ΔV values for SOHO have decreased in magnitude as the system is better characterized. For reference, the following burn profile for SOHO was used as for comparison (Roberts, 2011). The characteristic parameters for the SOHO orbit (very similar to the Richardson ISEE-3 mission) are:

$$A_x = 206{,}448\ km,\ A_y = 666{,}672\ km,\ A_z = 120{,}000\ km,\ \mu_{ES} = 3.0025 \times 10^{-6},\ n = 1.9912 \times 10^{-7}$$

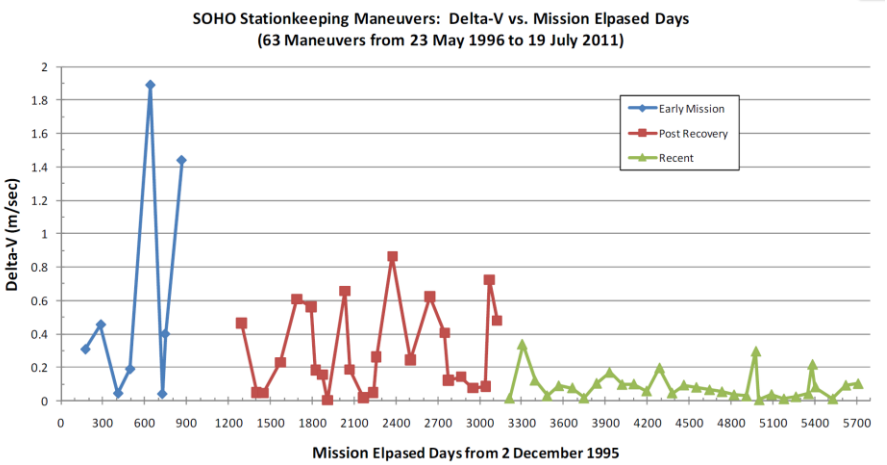

Figure 7 - SOHO mission ΔV profile.

A station keeping strategy was developed that attempted to combine features of Genesis and SOHO. From Genesis, the calculation of the required ΔV (Equation *(6)*) was used. From SOHO, the maneuver frequency and guidelines for acceptable ΔV magnitudes were used (Roberts, 2011). For SOHO, the goal was to maximize coast time while still having ΔV's lower than $0.5\ m/s$, which was considered "too large" by mission designers. From the above data provided by Roberts, most maneuvers performed were successfully at or below this level. Obviously, if the spacecraft gets far enough away from the orbit, a maneuver above $0.5\ m/s$ will be necessary. The reality of spaceflight is that many things can go wrong – in SOHO's case, the gap between days 900 and 1250 was due to the spacecraft being "lost" for a period of roughly 5 months before being re-established, and having new trajectories calculated to return it to a stable orbit. Eventually, a series of multiple "large" burns were implemented that successfully saved the mission. (Roberts, 2011)

To combat the possibility of an "erroneous" high value ΔV, a system was put in place to try to avoid the burn. The first time a ΔV is calculated to be $> 2.0\ m/s$, it is ignored. After a short delay ($10\ days$) a newly calculated ΔV was used. The implementation of this system appeared to have the effect of reducing the frequency of "large" ΔV's. The theory behind why this might have helped is that the target time t_t (from station keeping control) is fixed, so at certain locations within the HALO orbit, the deviation of t_t might appear larger. If the ΔV is calculated when t_t lines up with the peak velocity, a larger than necessary ΔV would be returned. By delaying an initial large ΔV, the subsequent ΔV might be smaller (when t_t is further offset form peak velocity). The relationship between t_t and station keeping behavior is explored more, prior to final results.

An initial test was performed with an initial perturbation. A minimum maneuver threshold of $\Delta V = 0.1\ m/s$ was enforced, meaning that if a ΔV was calculated below $0.1\ m/s$, the spacecraft would not perform the maneuver. This allows for long coast times without letting the ΔV build up too high – something that SOHO prioritized. Additionally, the target 2 month burn frequency was put in effect. Meaning that no two burns could occur within 60 days of each other. This strategy was integrated over 6 Periods and resulted in the orbital trajectory and ΔV profile shown below in Figure 8 and Figure 9. (Red circles indicate ΔV's).

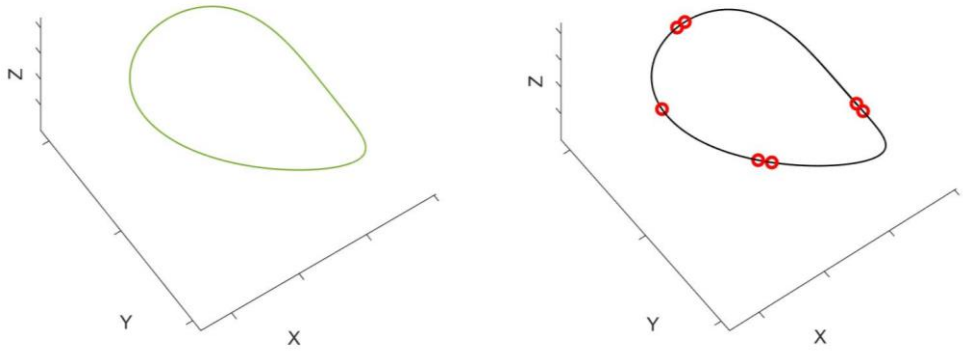

Figure 8 - SOHO Mimicked HALO Orbit with station keeping. (Left: Nominal, Right: Corrected).

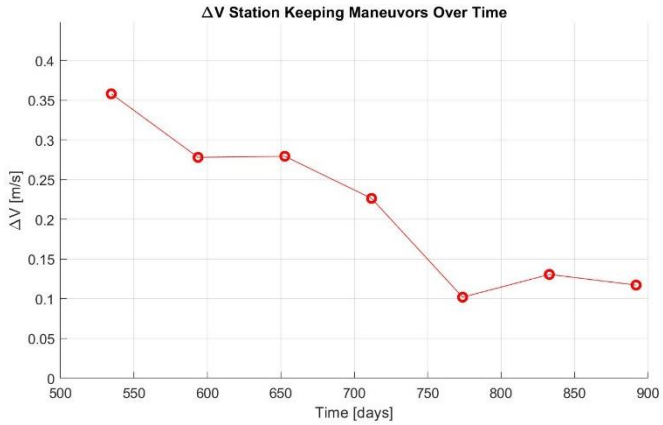

Figure 9 - ΔV vs time for station keeping of a mimicked SOHO HALO orbit.

With this station keeping control implemented, the maximum ΔV maneuver stays below $0.5\ m/s$ and a total $\Delta V = 1.49\ m/s$ over the course of 900 days. The roughly $1850\ kg$ SOHO spacecraft uses $4.5\ N$ hydrazine thrusters, which have an $I_{sp} \approx 175\ s$ – this translates to roughly $1.61\ kg$ of fuel (using the classic rocket equation).

Having verified the general process for implementing thresholds and frequency of burns, the mission was then integrated over $5700\ days$ – the amount of time available in the Roberts SOHO report. For this test, the same ΔV thresholds were used, and t_t was set at $60\ days$. For the longer duration integration, it seemed fitting to use the next burn location as the target for the current burn. This initial guess at a functional t_t proved to be unsuitable, as the station keeping maneuvers quickly kicked the spacecraft off of the nominal orbit. (Red circles indicate ΔV's).

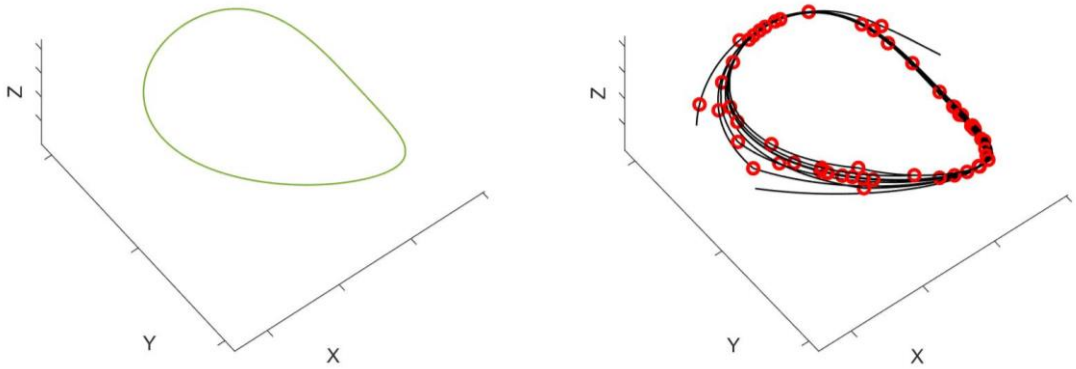

Figure 10 - Orbit trajectory for $t_t = 60\ days$. (Left: Nominal, Right: Corrected).

Efforts were then made to determine the optimal t_t for this station keeping strategy. This was done by varying t_t, and calculating the resulting ΔV_{total} after $2500\ days$ of control.

Table 1 - Total ΔV and average time between maneuvers for variable t_t.

$t_t\ (days)$	ΔV_{total}	$\Delta t_{average}\ (days)$
23	$193.38\ m/s$	60.5
26	$53.69\ m/s$	60.6
29	$35.12\ m/s$	61.4
32	$7.92\ m/s$	62.2
35	$7.10\ m/s$	66.2
38	$4.42\ m/s$	73.1
$41\ days$	$3.91\ m/s$	78.7
44	$5.82\ m/s$	75.6
47	$6.17\ m/s$	77.2
50	$17.65\ m/s$	71.4
53	$57.69\ m/s$	68.1

From this quick analysis, we see that a value for t_t that is too short causes the orbit to become unstable/escape. Likewise, if t_t is too large, the orbit falls apart. Therefore, a value for t_t was selected in the ΔV minimum location to be: $t_t = 41\ days$. With this value of t_t, the full mission duration could be integrated and the final results (trajectory, deviation, burn information, etc.) can be compared to SOHO performance.

This quick analysis does highlight an issue with the station keeping strategy implemented thus far, and that is that the t_t value is constant throughout the orbit and uses a simple deviation calculation. Ultimately, a successful mission profile (trajectory, ΔV profile, fuel mass, etc.) was generated, however, in the Conclusion and Future Work section, alternative methods of implementing a station keeping control is outlined.

In addition to total ΔV, the mean time between burns was also determined for the first $2500\ days$, as it is also an indicator of how well the station keeping satisfies the coast-time preferences for the mission. While each t_t has an average time between burns of at least $60\ days$, some perform better due to the $0.1\ m/s$ minimum ΔV to trigger a maneuver event.

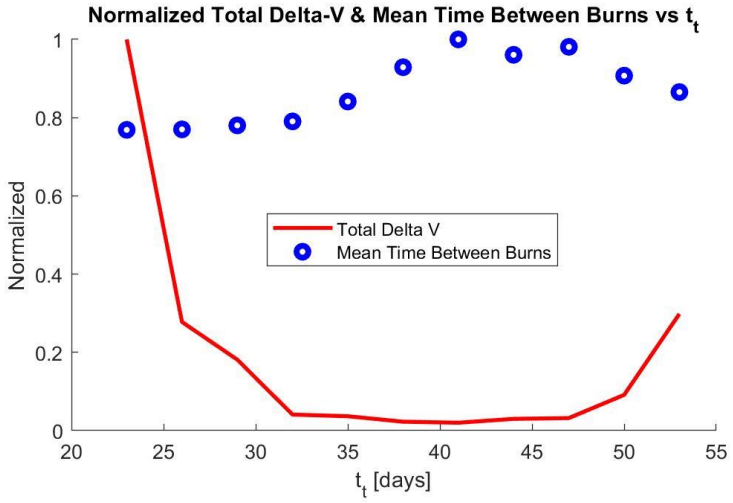

Figure 11 - Normalized plot of total ΔV and average time between burns vs t_t.

The above plot highlights the minimum ΔV and maximum coast time having an optimal value at $t_t = 41\ days$. This value was used in the following final integration.

With the newly selected value of t_t, the mission was then integrated over the full $5700\ days$, yielding the following successful trajectory and burn profile. (Red circles indicate ΔV's).

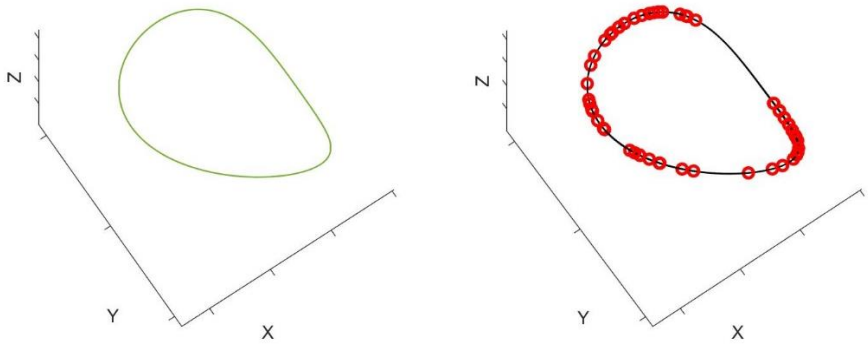

Figure 12 – SOHO-mimicked HALO orbit with station keeping. (Left: Nominal, Right: Corrected).

Visually, this station keeping strategy kept the spacecraft right in line with the nominal L1 orbit. To quantify the deviations, the following plot was constructed.

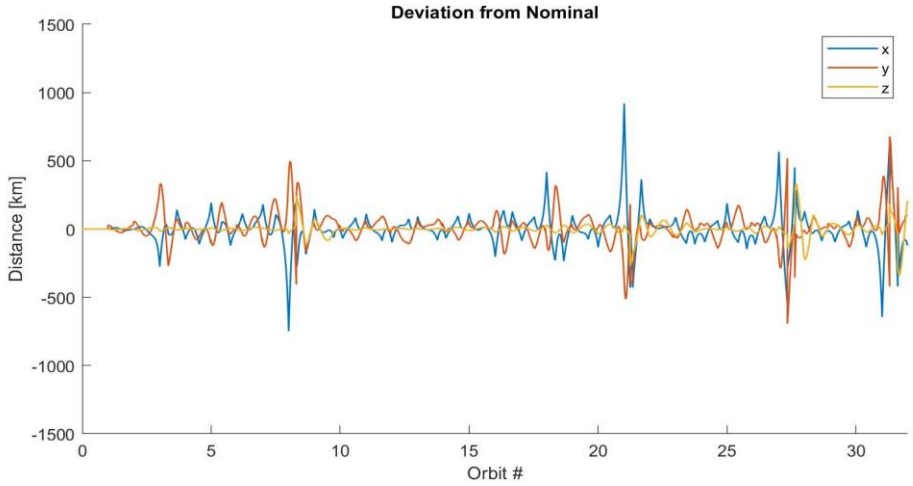

Figure 13 - Deviation from the nominal orbit.

From this plot we can determine that the spacecraft spends most of its time $< 250\ km$ away from the nominal target. Finally, the ΔV profile for this mission ended up being the following.

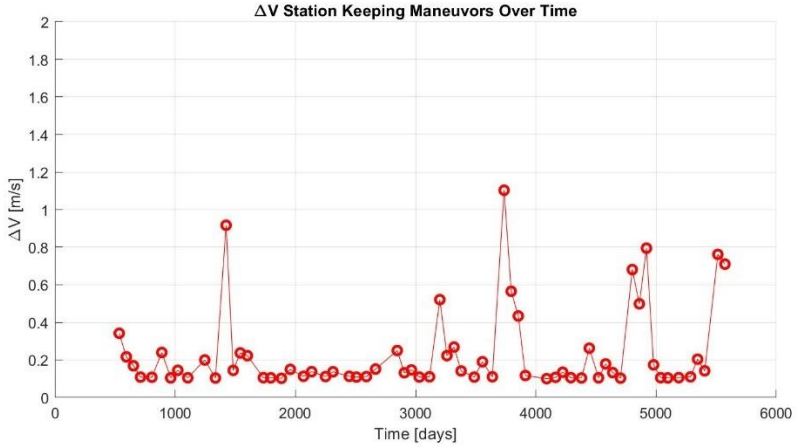

Figure 14 - ΔV vs time for station keeping of SOHO HALO orbit.

The total ΔV for this mission was calculated to be $15.07\ m/s$, which can be achieved using $16.17\ kg$ of propellant. Additionally, due to the $\Delta V_{min} = 0.1\ m/s$ threshold, the average time between burns was $78.7\ days$, with the longest time between two burns being $180\ days$. This is very similar performance to SOHO, which had a record of $146\ days$ between maneuvers (Roberts, 2011). Overlaying the ΔV profile of this mission with the SOHO profile shows, again, how similar the performance of this report is to the actual values.

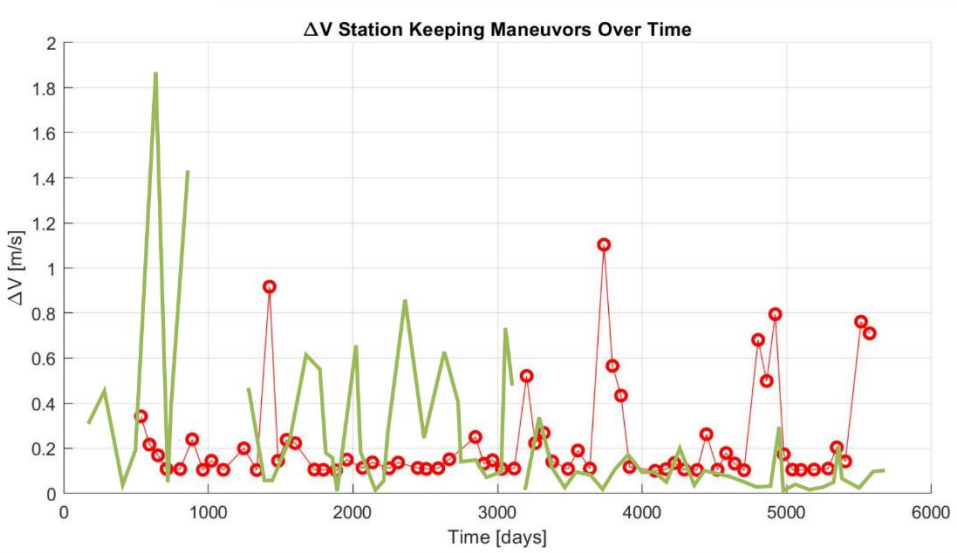

Figure 15 - Comparison of SOHO burn profile with the integrated profile (Green: SOHO [Fig. 7], Red: Ethan).

The ΔV values, while not linked to SOHO flight data, show that the strategy implemented in this report is an effective method for station keeping in SOHO-esque orbits that is comparable in terms of ΔV and coast time. Despite how well the station keeping performed, there is still room for improvements on the model. Improvements that can be made in the future are detailed further in the Conclusion and Future Work section.

Conclusion and Future Work

From the status update, the goal for this project was to: "calculate HALO orbits about the L1 Lagrange point, and to perform analysis on the station keeping requirements to maintain the orbit over time." To this end, the project succeeded – HALO orbits of any parameters (A_x, A_y, and A_z) can be generated with initial conditions. These initial conditions can then be made to converge to a nominal periodic solution. Integration can then be performed with an active station keeping algorithm that successfully maintains the orbit about the L point. Each of these systems were successfully implemented in MATLAB (see Appendix) and the effectiveness of the Genesis-SOHO inspired station keeping was verified by confirming how closely the total ΔV and coast times are to SOHO's actual values.

Future work for this project can take two forms: 1) improving the dynamic models and 2) improving the station keeping algorithm.

Currently the model assumes that only the two primary masses effect the spacecraft. For improvements to the dynamic system, other factors such the effects of Jupiter, the asteroid belt, and even special relativistic effects would need to be included into the model. By including a more comprehensive set of dynamic factors, the variations to the HALO orbit would better reflect reality. The above-mentioned factors would lead to different perturbing forces, ultimately leading to the spacecraft deviating from the nominal orbit in different ways.

For improving the station keeping algorithm, a more complex ΔV calculation could be used. Outlined in Roberts, the ΔV calculations for SOHO were more complex than the one implemented in this report. The algorithm used in SOHO integrates aspects of differential correction into the ΔV calculation algorithm. In the report, Roberts explains that the value of t_t is taken out until $T/2$, and a similar differential correction method is implemented to achieve $\dot{x} = \dot{z} = 0$ at the crossing. The ΔV is differentially corrected in accordance with $\delta\dot{x}$ and $\delta\dot{z}$, which would be iterated similar to the Howell method. Obviously the implementation of this method would be more involved that the determination of the nominal orbit, but the benefit would be more reliable ΔV's that avoid the current t_t problems discussed earlier in the Findings and Preliminary Results.

Of these two areas for future work, the station keeping improvements would likely have the biggest benefit to the overall performance of the integrated mission. It is likely that the rigid behavior of the current station keeping method contributes more "inaccuracy" than dynamical model.

Beyond the scope of this class, work will be focused on better understanding the differential ΔV correction method, and implementing it into the model. Given the simplistic dynamical model, this should lead to very low total ΔV's for long duration HALO orbit station keeping. After a more robust station keeping algorithm has been implemented, it would be interesting to compare results between the current model, and models that include more factors (such as the effects of Jupiter). A follow up report would aim at analyzing a further improved station keeping strategy, and comparing its performance between the Earth-Sun and Earth-Sun-Jupiter models.

MATLAB code for implemented mission features above can be found in the Appendix. Further plots and animations will be uploaded to ethan-geipel.com as purely fun/supplementary information to accompany info in this report.

References

Howell, K. C. (1983). *Three-Dimensional, Periodic, 'HALO' Orbits.* Stanford, Calif: Stanford University.

Lukehart, P. (1963). Algorithm 218. Kutta Merson. *Comm. Assoc. COmput. Mach.*, 737-738.

Richardson, D. L. (1979). *Analytic Construction of Periodic Orbits About the Collinear Points.* Cincinnati, OH: University of Cincinnati.

Richardson, D. L. (1980). *HALO Orbit Formulation for the ISEE-3 Mission.* Cincinnati, OH: University of Cincinnati.

Roberts, C. E. (2011). *Long Term Missions at the Sun-Earth Libration Point L1: ACE, SOHO, and WIND.* AAS 11-485: NASA.

Williams, K., Barden, B., Howell, K., Lo, M., & Wilson, R. (2000). *Genesis HALO Orbit Station Keeping Design.* Pasadena, CA and West Lafayette, IN: Jet Propulsion Laboratory, California Institute of Technology and School of Aeronautics and Astronautics, Purdue University.

Appendix – MATLAB Code

"Directory" – Please reach out if it appears any code is mission.
(efgeipel@gmail.com)

DeltaV_study.m – This is the main driver, ties together all the other functions. You can set the characteristic parameters at the top – hitting run generates the plots from the report, and gives values such as total deltaV, duration between burns, etc.

stateTransitionMatrix.m – Equations of motion and STM differential equations functions. Integrated by Runge_Kutta_merson.m or your favorte ode solver.

xSTM.m – Simple code to convert linear array to 6x6 matrix for STM.

StationKeep.m – This is the code where the station keeping method is implemented.

Runge_Kutta_Merson.m – The integrator used in the report.

HALO_3OA.m – A function that is passed the characteristic parameters and the initial conditions are passed back. Used at the start of DeltaV_study.m.

findT2.m – Used for differential corrections.

G_CRTBP.m – Function acquired from James Mireles (fau.edu) which is a cleaner, callable way to construct the STM matrices.

HALOcorrections_z_dy.m – Implementation of Howell's differential corrections.

HALOcorrections_x_dy.m – "Alternate" implementation of Howell's differential corrections.

%% DeltaV_study.m

```
clc; clear; close all
I6 = eye(6); Io = [I6(:,1); I6(:,2); I6(:,3); I6(:,4); I6(:,5); I6(:,6)];
fprintf('Cleared, closed, and ready to go.\n')
%% Initial Variable Set Ups
m1 = 1.989 * 10^30; m2 = 5.972 * 10^24; % Actual Masses (earth-sun)
G = (6.67408e-11); mu = m2/m1;
au = 1.49598e+8;
ne = -sqrt((m1+m2)*G/((au)*10^3)^3); n = 1;
numOrbs = 14; % Number of Periods to Integrate over
[L1, L2, L3] = getLpoints(mu);
% Howell mu = 0.04 Demo --
% mu = 0.04; au = 1;
% Y_3OA = [0.723268; 0; 0.040000; 0; 0.198019; 0];

% An orbit around the Earth!
% yinit = [au-mu*au+10000; 0; 5000; 0; (6.3)/ne; 0]/au;

% Richardson Demo --
% x0 = 2.452044207*10^5; y0 = 0; z0 = 1.003274912*10^5;
% dx0 = 0; dy0 = -2.91755907*10^2/1000; dz0 = 0;
fprintf('Starting 3rd Order Approximation...\n')
Ax = 205000; Az = 125000;
InitialConditions = HALO_3OA(mu, Ax, Az, ne);
x0 = InitialConditions(1);
z0 = InitialConditions(3);
dy0 = -InitialConditions(5);

Y_3OA = [L1*au+x0; 0; z0; 0; L1*au-(L1*au*ne+dy0)/ne; 0]/au;
i = 1; Y_final = zeros(1,42); t_0 = 0; h = 0.0005;
[T2, ~, ~] = findT2(Y_3OA, 0, 10, h, Io, mu, 1);
y_int = Y_3OA;
while t_0 < T2
    Y_final(i,:) = Runge_Kutta_Merson(@stateTransitionMatrix, t_0, [y_int; Io], h, mu, 1)';
        y_int = Y_final(i,1:6)'; Io = Y_final(i, 7:42)';
    t_0 = t_0 + h; i = i+1;
end
Y_3OA_traj = Y_final;
```

```
%% Differential Correction Process
t_end = 2*T2; y_guess = Y_3OA; zeroPoint = 30;
fprintf('Begining Differential Correction...\n')
for narrowDown = 1:1:20
    Io = [I6(:,1); I6(:,2); I6(:,3); I6(:,4); I6(:,5); I6(:,6)];
    [T2, y_STM_T2, STM_T2] = findT2(y_guess, 0, 3, 0.0005, Io, mu, n);
    if (mod(narrowDown,2) == 1) % z and dy
        fprintf('Fixing z and dy\n')
        dx_correction = HALOcorrections_z_dy(y_STM_T2, STM_T2, mu);
        y_guess = [y_guess(1); y_guess(2); y_guess(3)+ dx_correction(1); ...
                   y_guess(4); y_guess(5) + dx_correction(2); y_guess(6)];
    end
    if (mod(narrowDown,2) == 0) % x and dy
        fprintf('Fixing x and dy\n')
        dx_correction = HALOcorrections_x_dy(y_STM_T2, STM_T2, mu);
        y_guess = [y_guess(1)+ dx_correction(1) ; y_guess(2); y_guess(3); ...
                   y_guess(4); y_guess(5) + dx_correction(2); y_guess(6)];
    end
    t_end = 2*T2;
    if (abs(y_STM_T2(4)) < 1e-10 && abs(y_STM_T2(6)) < 1e-10)
        fprintf('Differential Corrections Complete!\n')
        break
    end
end
y_int = y_guess;
t_0 = 0; i = 2; h = 0.001;
Y_final = zeros(1,42);
Y_final(1,:) = [y_int; Io];
while t_0 < t_end + 1*h
    Y_final(i,:) = Runge_Kutta_Merson(@stateTransitionMatrix, t_0, [y_int; Io], h, mu, 1)';
    y_int = Y_final(i,1:6)'; t_0 = t_0 + h; Io = Y_final(i, 7:42)';
    i = i+1;
end


%% First Round of Plotting
figure(1) % Planar Views of the Stable (Differentially Corrected) Orbit
subplot(1,3,1)
plot(Y_final(:,1), Y_final(:,2), 'k', 'LineWidth', 1.5); hold on
plot(L1,0,'kx')
xlabel('X'); ylabel('Y'); axis square; box off; title('X-Y Plot')
xlim([min(Y_final(:,1))*.9975 max(Y_final(:,1))*1.0025]); ylim([-max(Y_final(:,2))*1.1
max(Y_final(:,2))*1.1])

subplot(1,3,2)
plot(L1(1)+Y_final(:,2), Y_final(:,3), 'k', 'LineWidth', 1.5); hold on
plot(L1,0,'kx')
xlabel('Y'); ylabel('Z'); axis square; box off; title('Y-Z Plot')
xlim([min(L1(1)+Y_final(:,2))*.9995 max(L1(1)+Y_final(:,2))*1.0005]);
ylim([min(Y_final(:,3))*1.05 max(Y_final(:,3))*1.05])

subplot(1,3,3)
plot(Y_final(:,1), Y_final(:,3), 'k', 'LineWidth', 1.5); hold on
plot(L1,0,'kx')
xlabel('X'); ylabel('Z'); axis square; box off; title('X-Z Plot')
xlim([min(Y_final(:,1))*.9975 max(Y_final(:,1))*1.0025]); ylim([min(Y_final(:,3))*1.05
max(Y_final(:,3))*1.05])

figure(2) % 3OA_T2 vs Differentially Corrected Stable Orbit
plot3(Y_3OA_traj(:,1), Y_3OA_traj(:,2), Y_3OA_traj(:,3), 'Color', [0.9290 0.6940 0.1250],
'LineWidth', 0.8);
hold on
plot3(Y_final(:,1), Y_final(:,2), Y_final(:,3), 'Color', [0.4660 0.6740 0.1880], 'LineWidth',
0.8);
plot3(L1, 0, 0, 'kx')
legend('3OA Initial HALO Orbit (Thru T2)', 'Differentially Corrected HALO Orbit', 'L1')
xlim([0.985 0.995]);
set(gca,'YTickLabel',[]); xlabel('X')
set(gca,'XTickLabel',[]); ylabel('Y')
set(gca,'ZTickLabel',[]); zlabel('Z')
```

```
%% Nominal Orbit Acquisition for numOrbs Number of Orbits
fprintf('Propogating for %2.0f periods (~%3.0f days)..\n',numOrbs, -numOrbs*T2*2/ne/24/60/60)
h = T2/1500; % "Time"-step size
t_0 = 0; t_end = 2*T2; Y_nominal = zeros(1,42); i = 1;
y0 = y_guess;
Io = [I6(:,1); I6(:,2); I6(:,3); I6(:,4); I6(:,5); I6(:,6)];
Y_nominal(1,1:6) = y_guess; Y_base = zeros(1,6);
% Iterates state and STM through 1 orbit to aquire the nominal trajectory
for i = 1:1:t_end/h + 10
    Y_nominal(i,:) = Runge_Kutta_Merson(@stateTransitionMatrix, t_0, [y0; Io], h, mu, n)';
        y0 = Y_nominal(i,1:6)'; Io = Y_nominal(i, 7:42)';
        t_0 = t_0 + h;
end % Steps

Y_xyz = zeros(2,6); STM = zeros(6,6,2); STMsave = zeros(6,6,2);
% Breaks up integration into State Vector and STM
for i = 1:1:length(Y_nominal)
    [Yi, STM] = xSTM(Y_nominal(i,:));
    Y_xyz(i,:) = Yi;
    STMsave(:,:,i) = STM;
end
Io = [I6(:,1); I6(:,2); I6(:,3); I6(:,4); I6(:,5); I6(:,6)];

% Repeats the nominal trajectory numOrbs times so station keeping has the
% nominal orbit to target. Even stable initial conditions deviates in the
% co-linear L point HALO orbits, so you can't just integrate the initial
% condition for 2+ periods.
Y_xyz = repmat(Y_xyz(1:end,:),numOrbs,1);
STMsave = repmat(STMsave(:,:,1:end),1,1,numOrbs);
Y_nominal = repmat(Y_nominal(1:end,:),numOrbs,1);
fprintf('Acquired nominal.\n')
```

```
%% Station Keeping Correction Method
% Initial Deviation, either random perturbation or 0 (since L(1-3) orbits
% are unstable anyway).
% initialDeviation = [rand(1)*1; rand(1)*10; rand(1)*1; ...
%     rand(1)*.0001*ne; rand(1)*.001*ne; rand(1)*.0001*ne]/au; % Random Deviation
initialDeviation = [0; 0; 0; 0; 0; 0]; % Zero Initial Deviation
y0 = Y_xyz(1,1:6)' + initialDeviation;
Io = [I6(:,1); I6(:,2); I6(:,3); I6(:,4); I6(:,5); I6(:,6)];

% Setting up variables/matrices used in station keeping strategy.
y0_nc = y0; Io_nc = Io; Y_delta_nc = zeros(1,42); Y_delta_nc(1,1:6) = y0;
t_0 = 0; burnCounter = 0; burnSpot = zeros(1,3);
DeltaVStore = zeros(1,6); DeltaVStoreP = zeros(1,6);
delay = 500; quickFix = 0; % Sets delay time between burns (1000 = 60 days)
Y_delta = zeros(size(Y_xyz,1),42);
Y_delta(1,1:6) = y0; Y_delta(1,7:end) = Io;
fprintf('Beginning Integration with Course Corrections...\n')
propOut = 675; % Time to propogate out to calculate deltaV's (41 days).
for i = 2:1:length(Y_xyz)
    % Integrate
    Y_delta(i,:) = Runge_Kutta_Merson(@stateTransitionMatrix, t_0, [y0; Io], h, mu, n)';
    Y_delta_nc(i,:) = Runge_Kutta_Merson(@stateTransitionMatrix, t_0, [y0_nc; Io_nc], h, mu, n)';
    y0 = Y_delta(i,1:6)';
    Io = Y_delta(i, 7:42)';
    y0_nc = Y_delta_nc(i,1:6)';
    Io_nc = Y_delta_nc(i,7:42)';

    if( delay < 1)
        t_0 = t_0-h;

        % Gets closest nominal location
        displaces = Y_delta(i,1:3)'-Y_xyz(:,1:3)';
        displaces = sqrt(displaces(1,:).^2+displaces(2,:).^2+displaces(3,:).^2);
        [a, b] = min(displaces);
        [deltaV, Y_prop] = StationKeep(Y_delta(i,1:6)', Y_delta(i,7:end)', propOut, t_0, h, i,
mu, 1, STMsave, Y_xyz, b);
        delay = 50;

        % Checks if conditions for making a manuever have been met (minimum
        % delta V threshold, 2 m/s max, etc.)
        if ((abs(norm(deltaV)*au*ne*1000) > 1e-1 && abs(norm(deltaV)*au*ne*1000) < 2) ||
(quickFix == 1) )
            burnCounter(end+1) = i;
            delay = 1000;
            fprintf('deltaV = %3.2f m/s ', norm(deltaV)*au*(-ne)*1000)
            fprintf('occuring at t = %4.0f days\n', -i*h/ne/24/60/60)
            y0 = y0 + deltaV'; y0_nc = Y_delta_nc(i,1:6)';
            Io = Y_delta(i, 7:42)';
            burnSpot(i,:) = y0(1:3);
            DeltaVStore(i,:) = deltaV;
            DeltaVStoreP(i,:) = deltaV;
            quickFix = 0;
        end
        % Strategy imposed for burns over 2m/s (none for t_t = 41 days).
        if (abs(norm(deltaV)*au*ne*1000) > 2)
            delay = 300;
            quickFix = 1;
        end

    end % delay < 1
    t_0 = t_0 + h; % Step forward in time.
    delay = delay - 1; % Handles the delay between burns.
end % 2:1:length(Y_xyz)

fprintf('Done with course correction!\n')
```

```
%% Process Deviations

% Actually - gets deviation from Orbit (the "right" way)
fprintf('Processing deviation data...\n')
for i = 1:1:size(Y_delta,1)
    if (mod(i,size(Y_delta,1)/4) == 0)
        fprintf('%1.0f percent..\n',100*i/size(Y_delta,1))
    end
    % Gets closest nominal location
    displaces = Y_xyz(1:size(Y_xyz,1)/numOrbs+10,1:3)'-Y_delta(i,1:3)';
    displaces = sqrt(displaces(1,:).^2+displaces(2,:).^2+displaces(3,:).^2);
    [a, b] = min(displaces);

    delta_xyz(i,:) = Y_xyz(b,1:3) - Y_delta(i,1:3);
    delta_dxyz(i,:) = Y_xyz(b,4:6) - Y_delta(i,4:6);

end
x_orbits = 0:numOrbs/(length(delta_xyz)):numOrbs;
fprintf('Finished processing deviation data!\n')


%% Second Round of Plotting
figure(3) % x,y,z Deviations from Nominal (position)
plot(x_orbits(:,1:end-1), au*delta_xyz(1:end,:), 'LineWidth', 1)
legend(['x'; 'y'; 'z']); xlim([0 numOrbs]); box off
xlabel('Orbit #'); ylabel('Distance [km]'); title('Deviation from Nominal')
%
figure(4) % x,y,z Deviations from Nominal (velocity)
plot(x_orbits(:,1:end-1), au*delta_dxyz(1:end,:)*(-ne)*1000, 'LineWidth', 1)
legend(['dx'; 'dy'; 'dz']); xlim([0 numOrbs]); box off
xlabel('Orbit #'); ylabel('Velocity [km/s]'); title('Deviation from Nominal')

% Axis Stuff
xlmax = max(Y_xyz(:,1)); xlmin = min(Y_xyz(:,1));
ylmax = max(Y_xyz(:,2)); ylmin = min(Y_xyz(:,2));
zlmax = max(Y_xyz(:,3)); zlmin = min(Y_xyz(:,3));
xlw = 1.25*(xlmax - xlmin)/2; xlm = (xlmax + xlmin)/2;
ylw = 1.25*(ylmax - ylmin)/2; ylm = (ylmax + ylmin)/2;
zlw = 1.25*(zlmax - zlmin)/2; zlm = (zlmax + zlmin)/2;

figure(5) % Trajectory Plots
subplot(1,3,1) % Nominal
plot3(Y_xyz(:,1), Y_xyz(:,2), Y_xyz(:,3), 'Color', [0.4660 0.6740 0.1880], 'LineWidth', 0.9)
hold on; plot3(au*L1, 0, 0); title('Nominal'); axis square
xlim([xlm-xlw xlm+xlw]); ylim([ylm-ylw ylm+ylw]); zlim([zlm-zlw zlm+zlw]);
set(gca,'YTickLabel',[]); xlabel('X')
set(gca,'XTickLabel',[]); ylabel('Y')
set(gca,'ZTickLabel',[]); zlabel('Z')

subplot(1,3,2) % Uncorrected Trajectory
plot3(Y_delta_nc(:,1), Y_delta_nc(:,2), Y_delta_nc(:,3), 'Color', [0.9290 0.6940 0.1250],
'LineWidth', 0.9)
hold on; plot3(au*L1, 0, 0); title('Not Corrected'); axis square
xlim([xlm-xlw xlm+xlw]); ylim([ylm-ylw ylm+ylw]); zlim([zlm-zlw zlm+zlw]);
set(gca,'YTickLabel',[]); xlabel('X')
set(gca,'XTickLabel',[]); ylabel('Y')
set(gca,'ZTickLabel',[]); zlabel('Z')

subplot(1,3,3) % Corrected Trajectory
plot3(Y_delta(:,1), Y_delta(:,2), Y_delta(:,3), 'k', 'LineWidth', 0.9)
hold on; plot3(au*L1, 0, 0)
xlim([xlm-xlw xlm+xlw]); ylim([ylm-ylw ylm+ylw]); zlim([zlm-zlw zlm+zlw]);
title('Corrected'); hold on; axis square
plot3(burnSpot(:,1),burnSpot(:,2),burnSpot(:,3), 'ro')
set(gca,'YTickLabel',[]); xlabel('X')
set(gca,'XTickLabel',[]); ylabel('Y')
set(gca,'ZTickLabel',[]); zlabel('Z')
% -- end subplots Figure 5

figure(6) % Nominal and Corrected Side-by-Side
subplot(1,2,1) % Nominal Trajectory
```

```
plot3(Y_xyz(:,1), Y_xyz(:,2), Y_xyz(:,3), 'Color', [0.4660 0.6740 0.1880], 'LineWidth', 0.9)
hold on; plot3(au*L1, 0, 0); title('Nominal')
xlim([xlm-xlw xlm+xlw]); ylim([ylm-ylw ylm+ylw]); zlim([zlm-zlw zlm+zlw]);
set(gca,'YTickLabel',[]); xlabel('X')
set(gca,'XTickLabel',[]); ylabel('Y')
set(gca,'ZTickLabel',[]); zlabel('Z')

subplot(1,2,2) % Corrected Trajectory w/ Course Correction Burns
plot3(Y_delta(:,1), Y_delta(:,2), Y_delta(:,3), 'k', 'LineWidth', 0.9)
hold on; plot3(au*L1, 0, 0); title('Corrected'); hold on;
xlim([xlm-xlw xlm+xlw]); ylim([ylm-ylw ylm+ylw]); zlim([zlm-zlw zlm+zlw]);
plot3(burnSpot(:,1),burnSpot(:,2),burnSpot(:,3), 'ro', 'LineWidth', 2)
set(gca,'YTickLabel',[]); xlabel('X')
set(gca,'XTickLabel',[]); ylabel('Y')
set(gca,'ZTickLabel',[]); zlabel('Z')
% -- end subplots Figure 6

fprintf('Plotting Complete!\n')


                                        %% Burn Profile
clear DVStoreP DeltaVval
x_days = x_orbits*(t_end/-ne/24/60/60);

for i = size(DeltaVStore,1):1:size(x_orbits,2)
    DeltaVStore(i,1:6) = zeros(1,6);
    DeltaVStoreP(i,1:6) = zeros(1,6);
end
DVStoreP = sqrt(DeltaVStoreP(:,4).^2+DeltaVStoreP(:,5).^2+DeltaVStoreP(:,6).^2);
counter = 1;
% Iterates through and determines location of actual burns vs empty slots
% in the array.
for i = 1:1:size(x_orbits,2)
    if (DVStoreP(i) == 0)
        DVStoreP(i) = -au;
    end
    if (DVStoreP(i) > 0)
        DeltaVval(counter,1) = DVStoreP(i)*au*abs(ne)*1000;
        DeltaVval(counter,2) = (i/size(x_orbits,2)*numOrbs)*(t_end/-ne/24/60/60);
        counter = counter + 1;
    end
end
for i = 1:1:size(DeltaVval,1)-1
    dtburn(i) = DeltaVval(i+1,2)-DeltaVval(i,2);
end

% Plots the burn profile.
figure(7)
plot(x_days,DVStoreP*au*abs(ne)*1000,'ro', 'LineWidth', 2);
hold on
ylim([0 max(DVStoreP)*au*abs(ne)*1000*1.25])
plot(DeltaVval(1:end-0,2),DeltaVval(1:end-0,1),'r')
xlabel('Time [days]')
ylabel('\DeltaV [m/s]')
title('\DeltaV Station Keeping Maneuvors Over Time')
box off

% Calculate total delta V and fuel required.
delVtotal = sum(DeltaVval(:,1));
m0 = 1850;
Isp = 175;
mf = m0/exp(delVtotal/Isp/(9.81));
dm = m0-mf;
fprintf('Total delta V: %1.2f m/s\n', delVtotal)
fprintf('Fuel used: %1.2f kg\n', dm)
fprintf('Mean coast time: %1.2f days\n', mean(dtburn))


%% Duration Between Burns
tdays = [23, 26, 29, 32, 35, 38, 41, 44, 47, 50, 53];
dVdays = [193.4, 53.7, 35.1, 7.92, 7.1, 4.4, 3.9, 5.8, 6.2, 17.65, 57.7];
```

```
meandays = [60.5, 60.6, 61.4, 62.2, 66.2, 73.1, 78.7, 75.6, 77.2, 71.4, 68.1];
% Plots duration between burns and total delta V for t_t range. (Hardcoded
% with results from using above code).
figure(8)
plot(tdays, dVdays/max(dVdays), 'r', 'LineWidth', 2); hold on
plot(tdays, meandays/max(meandays), 'bo', 'LineWidth', 3)
box off; legend('Total Delta V','Mean Time Between Burns')
title('Normalized Total Delta-V & Mean Time Between Burns vs t_t')
xlabel('t_t [days]'); ylabel('Normalized')

fprintf('End.\n')
fprintf('Thanks!\n')
fprintf('-Ethan Geipel.\n')
```

END OF DeltaV_study.m

StateTransitionMatrix.m – Misnomer! This also has the Equations of Motion in it! I just never bothered to change the name...

```
function Xreturn = stateTransitionMatrix(~, X, mu)
   %% Function Description
    % This function recieves the [state vector; Io] vector, which is made
    % up of, the state vector (duh) and the 36x1 phi matrix which is used
    % in the integration process for the STM calculation.
    % Currently, this function calculates the state vector using the CRTBP
    % process, which is the approach that is used in the HALO orbit
    % calculations.
    % This works when called independantly (nope), but also can be called
    % "indirectly" via the Runge_Kutta method.
    %% Ethan Geipel, March 2019
    %% The Function
    Xreturn = zeros(42,1);
    % Obtain r and v state vectors from initial expression.
    r = X(1:3); v = X(4:6);
    % Sets up Phi by reconstructing the 36x1 matrix version of it.
    PHIo(:,1) = X(7:12);
    PHIo(:,2) = X(13:18);
    PHIo(:,3) = X(19:24);
    PHIo(:,4) = X(25:30);
    PHIo(:,5) = X(31:36);
    PHIo(:,6) = X(37:42);
    % Used in constructing the G matrix.
    Omega = [0,1,0;-1,0,0;0,0,0];
    % Calls a function (obtained from James Mireles at FAU) which returns
    % the 3x3 matrix with the derivatives necessary to construct the G
    % matrix.
    dadr = G_CRTBP([r; v], mu);
    % Constructs the G matrix from above terms and generic code.
    G = [zeros(3) eye(3); dadr 2*Omega];
    % Multiplies G and Phi to iterate through the Phi process.
    PHI = G*PHIo;
    % Reconstructs the return matrix to 36x1 rather than 6x6.
    Xreturn(7:12) = PHI(:,1);
    Xreturn(13:18) = PHI(:,2);
    Xreturn(19:24) = PHI(:,3);
    Xreturn(25:30) = PHI(:,4);
    Xreturn(31:36) = PHI(:,5);
    Xreturn(37:42) = PHI(:,6);
    % Variables from matrix
     x = X(1);  y = X(2);  z = X(3);
    dx = X(4); dy = X(5); dz = X(6);
    % Distances, used in iterating through using CRTBP method.
    r1 = sqrt((mu+x)^2 + (y)^2 + (z)^2);
    r2 = sqrt((x-1+mu)^2 + (y)^2 + (z)^2);
    % Implementation of the CRTBP state vector integration process.
    Xreturn(1:6) = [dx;
                    dy;
                    dz;
                    2*dy + x-(1-mu)*(x+mu)/r1^3 - mu*(x-(1-mu))/r2^3;
                    -2*dx + y - (1-mu)*y/r1^3 - mu*y/r2^3;
                    -(1-mu)*z/r1^3 - mu*z/r2^3];

end
```

END StationTransitionMatrix.m

xSTM.m – converts 36x1 to 6x6 for STM work

```
function [Yi, STM] = xSTM(Y)
    Yi = Y(1:6);
    STM(:,1) = Y(7:12);
    STM(:,2) = Y(13:18);
    STM(:,3) = Y(19:24);
    STM(:,4) = Y(25:30);
    STM(:,5) = Y(31:36);
    STM(:,6) = Y(37:42);
end
```

END xSTM.m

StationKeep.m

```
function [deltaV, Y_prop] = StationKeep(y0, Io, propOut, t_0, h, j, mu, n, STMsave, Y_xyz, b )
    ti = t_0;
    yman = y0';
    for i = 1:1:propOut
        Y_prop(i,:) = Runge_Kutta_Merson(@stateTransitionMatrix, t_0, [y0; Io], h, mu, n)';
        y0 = Y_prop(i,1:6)'; Io = Y_prop(i, 7:42)';
        t_0 = t_0 + propOut*h;

     end

    M2 = j;
    T2 = j+ propOut;
    deltaV = zeros(1,6);
    deltaV(4:6) = zeros(1,3);
    if j+propOut < size(Y_xyz,1)
        STM_mod = STMsave(:,:,T2)/(STMsave(:,:,M2)); % M = manuevor, T = target
        A = STM_mod(1:3,1:3); B = STM_mod(1:3,4:6);
        deltaV(4:6) = -inv(B)*A*(yman(1:3)'-Y_xyz(b,1:3)')-(yman(4:6)'-Y_xyz(b,4:6)');
    end

end
```

End StationKeep.m

Runge_Kutta_Merson.m

```
function y = Runge_Kutta_Merson(func,t,y0,dt, mu, n)
   %% Function Description
    % This function recieves a function to be solved, the time to start
    % integration, the [state vector; Io] (for @sTM function, and the time
    % step to iterate with.
    % It implements the integration process from Runge-Kutta, and the core
    % code was "borrowed" from M. Mahooti via the MATLAB forums.
   %% Ethan Geipel, March 2019
   %% Global Variables (unused at the moment)
    % global mu
   %% The Function
    % dt = time step, Io = STM integration bit. y0 = [state vector; STM thing];
    % n = 1.99120675680384e-07; % n for Moon (?)
    % n = 0.0243736546460684; % n for Earth (?)
    % n=1; % For normalized case

    % Implementation of the Runge-Kutta integration approach
    eta0 = y0;
    k0   = dt*func(t,eta0, mu)*n;
    eta1 = eta0 + k0/3;
    k1   = dt*func(t+dt/3,eta1, mu)*n;
    eta2 = eta0 + (k0+k1)/6;
    k2   = dt*func(t+dt/3,eta2, mu)*n;
    eta3 = eta0 + (k0+3*k2)/8;
    k3   = dt*func(t+dt/2,eta3, mu)*n;
    eta4 = eta0 + (k0-3*k2+4*k3)/2;
    k4   = dt*func(t+dt,eta4, mu)*n;

    % Final return value
    y = eta0 + (k0+4*k3+k4)/6;

end
```

END Runge_Kutta_Merson.m

HALO_3OA.m - Initial conditions form third order approximation (For cleaner representation of analytic variables, see reference (Richardson, Analytic Construction of Periodic Orbits About the Collinear Points, 1979)**)**

```
function InitialConditions = HALO_3OA(mu, Ax, Az, ne) % Ax and Az is in km
    %% Get L points
    [L1, L2, ~] = getLpoints(mu);

    %% Analytic variables For Richardson Analytic Construction
    syms lam
    au = 1.49598e+08; % km
    re = au-L1*au;

    gam(1) = ((1-mu)-L1) / (1-mu-mu);
    gam(2) = -((1-mu)-L2) / (1-mu-mu);
    gam(3) = ((1-mu)-0) / (1-mu-mu);

    c = zeros(4,3);
    for n = 2:1:4
      c(n,1) = (1/(gam(1)^3))*( (1)^n*mu +  (-1)^n * (((1-mu)*gam(1)^(n+1))/((1-gam(1))^(n+1))));
      c(n,2) = (1/(gam(2)^3))*( (-1)^n*mu + (-1)^n * (((1-mu)*gam(2)^(n+1))/((1+gam(2))^(n+1))));
      c(n,3) = (1/(gam(3)^3))*( 1 - mu + ((mu*gam(3)^(n+1)))/((1+gam(3))^(n+1)) );
    end
    eqn = lam^4 + (c(2,1)-2)*lam^2-(c(2,1)-1)*(1+2*c(2,1)) == 0;
    lambda(:,1) = vpasolve(eqn, lam);
    eqn = lam^4 + (c(2,2)-2)*lam^2-(c(2,2)-1)*(1+2*c(2,2)) == 0;
    lambda(:,2) = vpasolve(eqn, lam);
    eqn = lam^4 + (c(2,3)-2)*lam^2-(c(2,3)-1)*(1+2*c(2,3)) == 0;
    lambda(:,3) = vpasolve(eqn, lam);
    lambda = lambda(2,:);

    del = lambda(1,:).^2 - c(2,:);
    del(3) = 2.66029*10^-6;

    k = 2.*lambda(1,:)./((lambda(1,:).^2+1-c(2,:)));

    d1 = 3*lambda(1,:).^2./k(1,:).*(k(1,:).*(6.*lambda(1,:).^2-1)-2.*lambda(1,:));
    d2 = 8*lambda(1,:).^2./k(1,:).*(k(1,:).*(11.*lambda(1,:).^2-1)-2.*lambda(1,:));

    a21 = 3.*c(3,:).*(k.^2-2)./(4.*(1+2.*c(2,:)));
    a22 = 3.*c(3,:)./(4*(1+2*c(2,:)));
    a23 = -(3*c(3,:).*lambda(1,:)./(4.*k(1,:).*d1(1,:))).*(3*k(1,:).^3.*lambda(1,:)-
6.*k(1,:).*(k(1,:)-lambda(1,:))+4);
    a24 = -3.*c(3,:).*lambda(1,:)./(4.*k(1,:).*d1(1,:)).*(2+3.*k(1,:).*lambda(1,:));

    b21 = -3.*c(3,:).*lambda(1,:)./2./d1(1,:).*(3.*k(1,:).*lambda(1,:)-4);
    b22 = 3.*c(3,:).*lambda(1,:)./d1(1,:);

    d21 = -c(3,:)./2./(lambda(1,:).^2);
    c31 = 3./64./(lambda(1,:).^2).*(4.*c(3,:).*a24(1,:) + c(4,:));

    a31 = -9.*lambda(1,:)./(4.*d2(1,:)).*( 4.*c(3,:).*(k(1,:).*a23(1,:)-b21(1,:))+
k(1,:).*c(4,:).*(4+k(1,:).^2) ) ...
        +((9.*lambda(1,:).^2 + 1 - c(2,:))./2./d2(1,:)).*( 3.*c(3,:).*(2*a23(1,:)-
k(1,:).*b21(1,:)) +c(4,:).*(2+3.*k(1,:).^2) );
    a32 = -(1./d2(1,:)).*( (9.*lambda(1,:)./4).*( 4.*c(3,:).*(k(1,:).*a24(1,:) - b22(1,:)) +
k(1,:).*c(4,:) ) ...
        +(3/2).*(9.*lambda(1,:).^2 + 1 - c(2,:)).*( c(3,:).*(k(1,:).*b22(1,:) + d21(1,:) -
2.*a24(1,:)) - c(4,:) ));

    b31 = 3./8./d2(1,:).*( 8.*lambda(1,:).*( 3.*c(3,:).*(k(1,:).*b21(1,:)-2.*a23(1,:)) -
c(4,:).*(2+3.*k(1,:).^2) ) ...
        +(9.*lambda(1,:).^2 + 1 + 2.*c(2,:)).*( 4.*c(3,:).*(k(1,:).*a23(1,:) - b21(1,:)) +
k(1,:).*c(4,:).*(4+k(1,:).^2) )  );
    b32 = (1./d2(1,:)).*( 9.*lambda(1,:).*( c(3,:).*(k(1,:).*b22(1,:) + d21(1,:) - 2.*a24(1,:)) -
c(4,:) )...
        +(3/8).*(9.*lambda(1,:).^2 + 1 + 2.*c(2,:)).*(4.*c(3,:).*(k(1,:).*a24(1,:) - b22(1,:)) +
k(1,:).*c(4,:) )   );

    a1 = -(3/2).*c(3,:).*(2.*a21(1,:) + a23(1,:) + 5.*d21(1,:)) - (3/8).*c(4,:).*(12-k(1,:).^2);
    a2 = (3/2).*c(3,:).*(a24 - 2.*a22(1,:)) + (9/8).*c(4,:);
```

```
    a1(3) = -1.25889*10^-5;
    a2(3) = 1.43702*10^-6;

    d31 = 3./64./(lambda(1,:).^2).*(4.*c(3,:).*a24(1,:) + c(4,:));
    d32 = 3./64./(lambda(1,:).^2).*( 4.*c(3,:).*(a23(1,:) - d21(1,:)) + c(4,:).*(4+k(1,:).^2) );

    s1 = 1./(2.*lambda(1,:).*(lambda(1,:).*(1+k(1,:).^2) - 2.*k(1,:))).*( ...
        (3/2).*c(3,:).*( 2.*a21(1,:).*(k(1,:).^2-2) - (a23(1,:).*(k(1,:).^2 + 2)) -
2.*k(1,:).*b21(1,:) )  ...
        -(3/8).*c(4,:).*(3.*k(1,:).^4-8.*k(1,:).^2 + 8));
    s2 = 1./(2.*lambda(1,:).*(lambda(1,:).*(1+k(1,:).^2) - 2.*k(1,:))).*( ...
        (3/2).*c(3,:).*( 2.*a22(1,:).*(k(1,:).^2-2) + (a24(1,:).*(k(1,:).^2 + 2)) +
2.*k(1,:).*b22(1,:) + 5.*d21(1,:))  ...
        +(3/8).*c(4,:).*(12- k(1,:).^2) );
    s1(3) = -1.59141*10^-6;
    s2(3) = 6.29433*10^-6;

    l1 = a1(1,:) + 2.*lambda(1,:).^2.*s1(1,:);
    l2 = a2(1,:) + 2.*lambda(1,:).^2.*s2(1,:);

    Ax_min = sqrt(abs(del(1,:)./l1(1,:)));
    Azt = sqrt((-del(1,:) - l1(1,:).*(Ax_min*re).^2)./l2(1,:));

    %% Construct the Path
    Lpoint = 1;
    Az = Az/re;
    Ax = -Ax/re;
    if (Lpoint == 1)
        Ay = k(1)*Ax;
    end
    if Lpoint == 2
        Ax = Ax_min(2); Ay = 0;
    end

    PHI = 0; PSI = [PHI + pi/2; PHI + 3*pi/2];
    w = 1 + s1(1,:).*(Ax^2) + s2(1,:).*(Az^2);

    %% Iterate Orbit Trace
    x = zeros(1,3); y = zeros(1,3); z = zeros(1,3);
    dx = zeros(1,3); dy = zeros(1,3); dz = zeros(1,3);
    i = 0;
    for t = 0:5*24*60*60/.2:180*24*60*60
        i = i + 1;
        % Time non-dimensionalizing
        s = ne*t;
        tau = w*s;
        tau1 = lambda(1,:).*tau + PHI;

        x(i,:) = a21(1,:).*Ax.^2 + a22(1,:).*Az.^2 - Ax.*cos(tau1) + (a23(1,:).*Ax.^2 -
a24(1,:).*Az.^2).*cos(2.*tau1) ...
            + (a31(1,:).*Ax.^3 - a32(1,:).*Ax.*Az.^2).*cos(3.*tau1);

        y(i,:) = k(1,:).*Ax.*sin(tau1) + (b21(1,:).*Ax^2 - b22(1,:).*Az^2).*sin(2.*tau1) ...
            + (b31(1,:).*Ax^3 - b32(1,:).*Ax.*Az^2).*sin(3.*tau1);

        deln = 2 - 3;
        z(i,:) = deln.*Az.*cos(tau1) + deln.*d21(1,:).*Ax.*Az.*(cos(2.*tau1) - 3) ...
            + deln.*(d32.*Az.*Ax^2 - d31(1,:).*Az^3).*cos(3.*tau1);

        dx(i,:) =  -Ax.*sin(tau1) + -2*(a23(1,:).*Ax.^2 - a24(1,:).*Az.^2).*sin(2.*tau1) ...
            + -3*(a31(1,:).*Ax.^3 - a32(1,:).*Ax.*Az.^2).*sin(3.*tau1);

        dy(i,:) = k(1,:).*Ax.*lambda(1,:).*cos(tau1) + 2*(b21(1,:).*Ax^2 -
b22(1,:).*Az^2).*lambda(1,:).*cos(2.*tau1) ...
            + 3*(b31(1,:).*Ax^3 - b32(1,:).*Ax.*Az^2).*lambda(1,:).*cos(3.*tau1);

        dz(i,:) = -deln.*Az.*sin(tau1) + -2*deln.*d21(1,:).*Ax.*Az.*(sin(2.*tau1)) ...
            + -3*deln.*(d32.*Az.*Ax^2 - d31(1,:).*Az^3).*sin(3.*tau1);

    end
```

```
    %% Plotting
    Plotter = 0;
    if Plotter == 1
        figure(8)
        plot(dx(:,1)/max(dx(:,1))); hold on
        plot(dy(:,1)/max(dy(:,1))); plot(dz(:,1)/max(dz(:,1)))
        legend(['dx'; 'dy'; 'dz'])

        figure(4)
        plot3(au*L1+re*x(:,Lpoint),re*y(:,Lpoint),re*z(:,Lpoint), 'k' )
        hold on; axis square; grid on
        plot(au*(1-mu),(0),'b*','LineWidth',4);
        plot(au*L1,0,'kx')

        figure(2)
        plot(re*x(:,Lpoint),re*z(:,Lpoint), 'k')
        hold on; axis square; grid on
        plot(re*(1-mu),(0),'b*','LineWidth',4);

        figure(3)
        plot(re*x(:,Lpoint),re*y(:,Lpoint), 'k')
        hold on; axis square; grid on
        plot((1-mu),(0),'b*','LineWidth',4); plot((-mu),(0),'b*','LineWidth',4)

        figure(1) % y-z
        plot(y(:,Lpoint),z(:,Lpoint),'k')
        hold on; axis square; grid on
        plot((-mu),(0),'b*','LineWidth',4)

        % General Plot Lims (If wanted)
        % axisdiv = 5*10^5;
        % xlim([-2*axisdiv 2*axisdiv]); xticks([-2*axisdiv -axisdiv 0 axisdiv 2*axisdiv])
        % ylim([-2*axisdiv 2*axisdiv]); yticks([-2*axisdiv -axisdiv 0 axisdiv 2*axisdiv])
    end

    %% Return Prints
    fprintf('x0 = %3.3f\n', x(1)*re)
    fprintf('z0 = %3.3f\n', z(1)*re)
    fprintf('dy0= %3.3f\n', dy(1)*re*ne)
    fprintf('Finished 3rd Order Approximation!\n')

    InitialConditions = [x(1)*re; 0; z(1)*re; 0; dy(1)*re*ne; 0];

end
```

END HALO_3OA.m

findT2.m - Used in Differential corrections

```
function [T2, y_STM_T2, STM_T2] = findT2(y0, t_0, t_end, h, Io, mu, n)
   %% Function Description
    % This function recieves the y0 = state vector, t_0 = start time
    % (normalized), t_end = end time (normalized), h = time step, and Io =
    % initial eye() matrix for STM integration.
    % This function then iterates through using Runge_Kutta method and is
    % "broken" when the y component of the state vector crosses 0 on the
    % "other side" of the HALO orbit.
    % It iterates through this process until a tolerance of delta y < 1e-11
    % is achieved after the y-crossing.
    % This function returns the time at this point, T2, and the State
    % Transition Matrix at this point, STM_T2.
    %% - Ethan Geipel, March 2019
    %% The function
    counter = 1; y_it = zeros(1,42);
    % Iterates through time 0 -> t_end
    while t_0 < t_end
        % Sends t_0, and y0; Io to the Runge_Kutta solver, which is then
        % passed along to the @stateTransitionMatrix function to be solved.
        y_it(counter,:) = Runge_Kutta_Merson(@stateTransitionMatrix,...
                                                t_0, [y0; Io], h, mu, n)';
        % Pulls out and updates Y and sets it to y0 for the next loop.
        y0 = y_it(counter,1:6)'; t_0 = t_0 + h; % Steps forward in time by h.
        % Pulls out and updates Io (becomes STM)
        Io = y_it(counter, 7:42)';

        % Checks for y-crossing at y=0.
        if (counter > 1)
            if (y0(2) * y_it(counter-1, 2) < 0)
                break
            end % y0(2) <0
        end

        % Iterates through the loop.
        counter = counter + 1;
    end
    % After y crossing has been detected, T2 is set to be the time t_0 at
    % which the y-crossing occurs.
    T2 = t_0 - h;
    KeepGoing = 0;
    % While loop to iterate T2 down such that the delta y value of the
    % y-crossing is < 1e-11.
    while KeepGoing < 1
        % Reverts t_0 back 50 time steps (so that you don't have to iterate
        % through all of the time span every iteration).
        t_0 = T2 - 20*h;
        % Reduces h by factor 1e-1 to achieve better resolution on T2.
        h = h/10;
        y0 = y_it(counter-20, 1:6)'; % Sets back the value of y0 wrt t_0 set
        Io = y_it(counter-20, 7:42)';% Sets back the value of y0 wrt t_0 set
        counter = counter-20; y_it = zeros(1,42);
        while t_0 < T2 + .2
            % Similar to above, sends time and state to R_G and executes
            % the function @sTM.
            y_it(counter,:) = Runge_Kutta_Merson(@stateTransitionMatrix,...
                                                    t_0, [y0; Io], h, mu, n)';
            % Pulls out state vector.
            y0 = y_it(counter,1:6)'; t_0 = t_0 + h; % Step forward in time, h
            % Pulls out Io for STM calculation
            Io = y_it(counter,7:42)';
            % Gets the full [state vector; STM] vector stored for later
            % calculations.
            ySTM = y_it(counter,:);

            % Detects y-crossing at y = 0.
%             if (y0(2) > 0)
            if (y0(2) * y_it(counter-1, 2) < 0)

                % fprintf('wow!')
                % Updates the value for T2 with the time of y-crossing
```

```
                % detection. (Will be more accurate due to h decrease.
                T2 = t_0;
                KeepGoing = KeepGoing + .1;
                % Calls function @xSTM(), sending the [state vector; STM]
                % at time T2, which sends back the 6x6 STM (at time T2).
                [y_STM_T2, STM_T2] = xSTM(ySTM(end,:));
                % Checks if the tolerance of delta y at y-crossing has been
                % met. Tolerance is 1e-11.
                if (abs(y0(2)) < 1e-15)
                    % fprintf('woah...')
                    KeepGoing = 1;
                end
                break
            end % y0(2) <0
            counter = counter + 1;
        end % while t_0 < T2 + .2
    end % while keepGoing < 1
end
```

END FindT2.m

G_CRTBP.m - Acquired from James Mireles (fau.edu)
[http://cosweb1.fau.edu/~jmirelesjames/matLabPage.html]

```
function GMatrix=G_CRTBP(x, mu)

%function returns the matrix 'G' for the CRTBP
%with parameter mu.


%the distances
r1 = sqrt((x(1)+mu)^2 + x(2)^2 + x(3)^2);
r2 = sqrt((x(1)-(1-mu))^2 + x(2)^2 + x(3)^2);

%If U is the potential for the CRTBP then this code is computing
%the derivative of the gradient of U.  The gradient has three
%components which we will call u1, u2, u3.  The differential of the
%gradient is the matrix of partials of these functions.  These will
%be denoted by u1_x, u1_y, and so forth.

u1_x = 1 - (1-mu)*(1/(r1^3) - 3*((x(1)+mu)^2)/(r1^5))- mu*(1/(r2^3) - 3*((x(1)-(1-
mu))^2)/(r2^5));

u2_y = 1 - (1-mu)*(1/(r1)^3 - 3*x(2)^2/r1^5) - mu*(1/r2^3 - 3*x(2)^2/r2^5);

u3_z=(-1)*(1-mu)*(1/(r1)^3-3*x(3)^2/r1^5)-mu*(1/r2^3-3*x(3)^2/r2^5);

u1_y=3*(1-mu)*x(2)*(x(1)+mu)/r1^5+3*mu*x(2)*(x(1)-(1-mu))/r2^5;

u1_z=3*(1-mu)*x(3)*(x(1)+mu)/r1^5+3*mu*x(3)*(x(1)-(1-mu))/r2^5;

u2_z=3*(1-mu)*x(2)*x(3)/r1^5+3*mu*x(2)*x(3)/r2^5;


%equality of mixed partials gives (as all the terms are already partials
%of the potential function);
u3_y=u2_z;

u2_x=u1_y;

u3_x=u1_z;



%Then (as mentioned) G is the matrix of partials

GMatrix=[u1_x, u1_y, u1_z;
         u2_x, u2_y, u2_z;
         u3_x, u3_y, u3_z];

end
```

```
                Differential Correction Methods (2 Variants, discussed in report)
function dx_correction = HALOcorrections_x_dy(Y_T2, STM_T2, mu) % Pass [x; y; z; dz; dy; dz], STM
@ T2, and mu
   %% Function Description
    % This function accepts the state vector and state transition matrix at
    % time T2, and uses these values to determine the corrections to x and
    % dy in order to trend towards a stable orbit.
    % Ideally, this will be called until the dx and dz values at T2 are
    % less than 1e-8.
   %% Ethan Geipel, March 2019
   %% The Function
   % Pull out the state vector components.
     x = Y_T2(1);  y = Y_T2(2);  z = Y_T2(3);
    dx = Y_T2(4); dy = Y_T2(5); dz = Y_T2(6);

    % Calculate distances for use in later calculations.
    r1=sqrt((mu+x)^2+(y)^2+(z)^2);
    r2=sqrt((x-1+mu)^2+(y)^2+(z)^2);

    % Calculate the x and z accelerations (from CRTBP equations)
    xdd = 2*dy + x-(1-mu)*(x+mu)/r1^3 - mu*(x+mu-1)/r2^3;
    xdd = 2*dy + x-(1-mu)*(x+mu)/r1^3 - mu*(x-(1-mu))/r2^3;
    zdd = -z*((1-mu)/r1^3 + mu/r2^3);
    zdd = -(1-mu)*z/r1^3 - mu*z/r2^3;

    % Setting up the expression to calculate correction values (this comes
    % from a paper on 3D HALO Orbit families by Kathleen "Kathy" Cowell).
    A = ([STM_T2(4,1), STM_T2(4,5); STM_T2(6,1), STM_T2(6,5)]-(1/dy)*[xdd; zdd]*[STM_T2(2,1),
STM_T2(2,5)]);
    B = [-dx; -dz];

    % Calculates the corrections and sets them to return.
    dx_correction = A\B; % Return del_x and del_dy

end

function dx_correction = HALOcorrections_z_dy(Y_T2, STM_T2, mu)
   %% Function Description
    % This function accepts the state vector and state transition matrix at
    % time T2, and uses these values to determine the corrections to z and
    % dy in order to trend towards a stable orbit.
    % Ideally, this will be called until the dx and dz values at T2 are
    % less than 1e-8.
   %% Ethan Geipel, March 2019
   %% The Function
   % Pull out the state vector components.
     x = Y_T2(1);  y = Y_T2(2);  z = Y_T2(3);
    dx = Y_T2(4); dy = Y_T2(5); dz = Y_T2(6);

    % Calculate distances for use in later calculations.
    r1=sqrt((mu+x)^2+(y)^2+(z)^2);
    r2=sqrt((x-1+mu)^2+(y)^2+(z)^2);

    % Calculate the x and z accelerations (from CRTBP equations)
    xdd = 2*dy + x - (1-mu)*(x+mu)/r1^3 - mu*(x+mu-1)/r2^3;
    zdd = -z*((1-mu)/r1^3 + mu/r2^3);

    % Setting up the expression to calculate correction values (this comes
    % from a paper on 3D HALO Orbit families by Kathleen "Kathy" Cowell).
    A = ([STM_T2(4,3), STM_T2(4,5); STM_T2(6,3), STM_T2(6,5)]-(1/dy)*[xdd; zdd]*[STM_T2(2,3),
STM_T2(2,5)]);
    B = [-dx; -dz];

    % Calculates the corrections and sets them to return.
    dx_correction = A\B; % Return del_z and del_dy

end

                        END DifferentialCorrection Functions
```