Volume II: Appendices

Simulation Framework for Rapid Entry, Descent, and Landing (EDL) Analysis

December 17, 2009

NESC Request No.: 09-00530

Table of Contents

Appendix A. Gravity Turn Terminal Descent Guidance Theoretical Development 1
Appendix B. POST2 Mass Model User's Guide 5
Appendix C. Animation Tool v2.3 25
Appendix D. Animation GUI v1.0 38
Appendix E. POST2 Scripts Users' Guide 51

Appendix A. Gravity Turn Terminal Descent Guidance Theoretical Development

Written by David W. Way; Edited by Scott A. Striepe

The gravity turn terminal descent guidance capability is for use during an all-propulsive terminal descent phase of an EDL. The routine is based on the use of gravity to turn the vehicle velocity and thrust to control the vehicle velocity magnitude.

Background and Theory

The term "gravity turn" is defined here as a trajectory in which the only force acting normal to the velocity vector, and thus providing the angular acceleration required to rotate the vehicle flight path angle, is gravity. The gravity turn problem is to determine the magnitude of the thrust vector (the direction is constrained by the problem to be aligned opposite the velocity vector) required to achieve a specified terminal condition (altitude and velocity) at a near vertical orientation. The governing equation of motion can be written

$$\dot{\vec{V}} = \left(\frac{1}{m}\right)\vec{T} + \vec{g} \tag{1}$$

During the powered portion of the terminal descent, vehicle dynamics are dominated by the propulsive thrust and drag is neglected. In order to solve the problem, the vector equation of motion must be written in scalar form. The free-body diagram of a gravity turn is shown in Figure A.1.

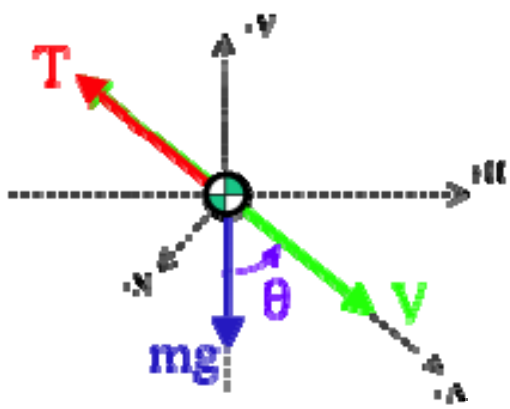

Figure A.1. Free-Body Diagram

By definition, the velocity vector, V, in Figure A.1 is the atmospheric relative velocity vector. However, since the atmospheric winds are usually not known, the planet relative velocity vector is used. The primary constraint of a gravity turn, therefore, is that the thrust vector is directed opposite the direction of the relative velocity vector.

Figure A.1 is used to write Equation 1 in the normal, axial, vertical, and horizontal directions. Equations 2 and 3 are the acceleration normal to the velocity vector and axially along the velocity vector, respectively. The term axial refers to the nominal thrust axis of the vehicle, which is assumed to be aligned with the planet relative velocity as depicted in Figure A.1.

$$\dot{V}_N = a_N = g\sin\theta \tag{2}$$

$$\dot{V}_A = a_A = g\cos\theta - \left(\frac{T}{m}\right) \tag{3}$$

where g is the surface gravity acceleration, T is the vehicle thrust, and m is the vehicle mass. Note that θ goes to zero as the gravity turn approaches the target final altitude. Equations 4 and 5 are the components of the acceleration in the local vertical and local horizontal directions, respectively.

$$\dot{V}_V = a_V = \left(\frac{T}{m}\right)\cos\theta - g \tag{4}$$

$$\dot{V}_H = a_H = -\left(\frac{T}{m}\right)\sin\theta \tag{5}$$

These equations apply only to the specific gravity turn case, where the thrust vector is aligned in the direction opposite the relative velocity vector and the drag is negligible

The axial acceleration, Equation 3, is integrated over the time-to-go, t_{go}, to get an expression for the final velocity, V_f, Equation 6.

$$V_f = V_0 + g\int_0^{t_{go}} \cos\theta dt - \int_0^{t_{go}} \left(\frac{T}{m}\right) dt \tag{6}$$

The time-to-go is defined as the time interval between the current time and the final time at which the target altitude and velocity are achieved.

The vertical acceleration, Equation 4, is integrated over the time-to-go, to get an expression for the final vertical velocity, Equation 7.

$$V_{Vf} = -V_0 \cos\theta_0 - g t_{go} + \int_0^{t_{go}} \left(\frac{T}{m}\right) \cos\theta dt \tag{7}$$

Because the target condition is assumed to be in a near-vertical attitude (-90 deg flight path angle), the target vertical velocity is identical to the target velocity. Therefore, either Equation 6 or Equation 7 may be used to express the desired constraint on the terminal velocity. Since large amounts of kinetic energy may be masked from the guidance at shallow flight path angles, where the vertical component of velocity is small, system performance is improved by controlling total velocity through Equation 6, rather than vertical velocity (Equation 7). Equation 7 is integrated again over the time-to-go interval to get an expression for the final altitude, h_f, Equation 8.

$$h_f = h_0 - V_0 \cos\theta_0 t_{go} - \frac{1}{2} g t_{go}^2 + \int_0^{t_{go}} \int_0^{t_{go}} \left(\frac{T}{m}\right) \cos\theta dt^2 \tag{8}$$

Evaluating the Integrals

A few notes are in order about evaluating the integrals contained in Equations 6 and 8. First, these integrals may not be evaluated unless the flight path angle and acceleration profiles are known, as well as the time-to-go. However, a simple change of integration variable from time, t, to a non-dimensional time scale over the time-to-go, τ removes the time interval dependence. Equation 9 shows how the time interval dependence is removed from one of the integrals.
For

$$\tau = \frac{t}{t_{go}}, \qquad \int_0^{t_{go}} \cos\theta dt = \int_0^1 \frac{\cos\theta}{\left(d\tau / dt\right)} d\tau = t_{go} \int_0^1 \cos\theta d\tau = t_{go} I_{\cos\theta} \tag{9}$$

Second, the trust profile may be both normalized by a scale factor, T_{sf}, and multiplied by a throttle setting, ϕ, as shown in Equation 10. The throttle (or gain) allows the guidance to target the terminal conditions by linearly scaling the nominal thrust profile. Equations 11 and 12 demonstrate these substitutions for the remaining two integrals.

$$\left(\frac{T}{m}\right) = \phi T_{sf} \left(\frac{\hat{T}}{m}\right) \tag{10}$$

$$\int_0^{t_{go}} \left(\frac{T}{m}\right) dt = \phi T_{sf} t_{go} \int_0^1 \left(\frac{\hat{T}}{m}\right) d\tau = \phi T_{sf} t_{go} I_{(T/m)} \tag{11}$$

$$\int_0^{t_{go}} \int_0^{t_{go}} \left(\frac{T}{m}\right) \cos\theta dt^2 = \phi T_{sf} t_{go}^2 \int_0^1 \int_0^1 \left(\frac{\hat{T}}{m}\right) \cos\theta d\tau^2 = \phi T_{sf} t_{go}^2 II_{(T/m)\cos\theta} \tag{12}$$

The usefulness of these three integrals is that they depend only on the shape of the assumed flight path angle profile and the desired thrust-to-mass profile over the non-dimensional time-to-go interval. The actual value of these integrals may be calculated off-line or as a separate subroutine and supplied to the guidance algorithm as parameters or gains. Therefore the guidance may be written in a general fashion that is independent of the assumptions imbedded in the definition of the integrals. Since the integrals represent certain assumptions on the profiles, the guidance is easily adapted to different assumptions or desired profiles by supplying new subroutines to evaluate the integrals.

Substituting for the Integrals

Equations 13 and 14 are the result of substituting the integral definitions (Equations 9, 11 and 12) into the equations of motion (Equations 6 and 8). These are the final forms of the equations of motion. They express the altitude and velocity loss as a function of the current velocity and flight path angle state, the engine throttle setting (thrust profile multiplier), time-to-go, and the three profile integrals that depend only on the shape of the assumed flight path angle profile and the desired thrust-to-mass profile over the time-to-go. Since the profile integrals are supplied as parameters, the only unknowns are the throttle setting and time-to-go.

$$\Delta V = \left(V_0 - V_f\right) = \left[\phi T_{sf} I_{(T/m)} - g I_{\cos\theta}\right] t_{go} \tag{13}$$

$$\Delta h = \left(h_0 - h_f\right) = V_0 \cos\theta_0 t_{go} + \left(\frac{1}{2} g - \phi T_{sf} II_{(T/m)\cos\theta}\right) t_{go}^2 \tag{14}$$

The order in which Equations 13 and 14 are solved depend upon the mode in which the guidance is operating. There are two guidance modes: a predictive mode in which the guidance is predicting the time-to-go and altitude loss for the assumed thrust profile, and a proactive mode in which the guidance determines the throttle setting required to achieve the terminal conditions. The predictive mode is used for determining the initiation of the powered descent. The proactive mode is used to guide the vehicle during the powered descent.

Appendix B. POST2 Mass Model User's Guide
Written by John J. Wagner; Edited by Scott A. Striepe

I. Introduction

The purpose of the mass modeling task was to equip the standard Program to Optimize Simulated Trajectories II (POST2)[B.1] code with the capability of utilizing mass metamodels for standard trajectory analysis. This capability allows the user to simultaneously design both the trajectory and an appropriately sized vehicle capable of following the intended trajectory. Several mass models were developed and integrated into the default mass initialization routines and test cases were developed which employ these mass model in a variety of scenarios. These scenarios correspond to various EDL technology packages currently under consideration for landing large payloads on the surface of Mars. The following sections explain the origin of the various mass models, their assumptions and functional limitations, and the results of the developed sample test cases.

II. Simulation Architecture

The standard POST2 calculation routines were preserved wherever possible when implementing the mass model subroutines. The basic framework architecture may be seen in Figure B.2.1. The intent of the architecture was to avoid a high degree of coupling between the mass model subroutines and the standard POST trajectory functions. The user begins by entering the necessary mass and trajectory input variables and initial values for the targeting algorithm. The wgtini.f routine is called which initializes the vehicle weight model. This file has been modified to call a special subroutine, massEDL.f which is used to perform a dynamic mass allocation. Instead of reading a deterministic mass or weight from the input deck, mass model input information is collected and used to determine the initial component masses (e.g. the aeroshell, descent stage, aeroshell reaction control system (RCS), etc.). These component masses are used to set the stage weights (WGTSG(i)) using the vehicle component weight model.

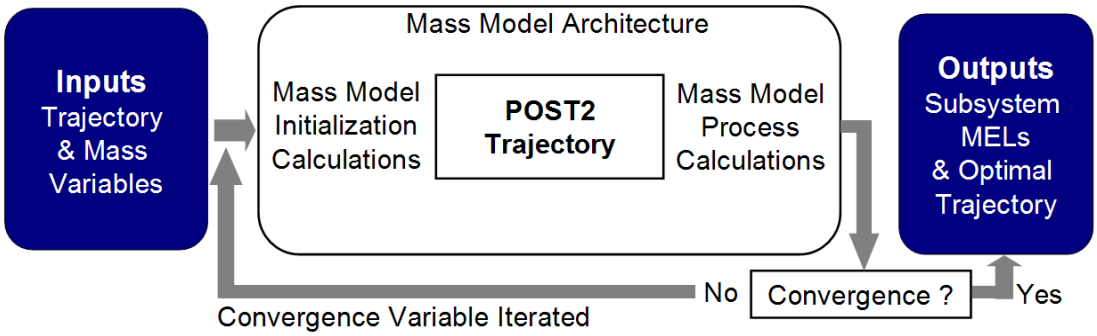

Figure B.2.1. Framework Mass Model Architecture

After the stage weights are set in massEDL.f, the POST trajectory is then allowed to run as normal. In the final usable phase (i.e., just before the final event), the mass convergence routine which resides in the special calculations routine calspe.f is called. This routine examines the mass results from the trajectory and computes the convergence variables. The standard target algorithms (projected gradient or NPSOL) are used to determine if convergence has been attained. If not, the mass control variables are iterated and the process repeats. Once convergence has been reached, the program outputs a standard output deck containing the standard trajectory variables as well as mass variables. All variables in the mass modeling variable structure begin with "MEDL_M". For example, the current value of the entry mass during iteration is referred to as MEDL_MGUESS.

III. Simulation Concepts & Modeling Approach

Three simulations for exploring potential EDL technology packages have been developed. These include;

- An ellipsled rigid aeroshell model and associated RCS model.
- A flexible, inflatable aeroshell model (Mars Inflatable Aeroshell System (MIAS)) and associated RCS model.
- An unconstrained all propulsive model with no aeroshell or thermal protection system.

The operational concepts of each of these simulation types may be seen in Figure B.3.1. All three simulations begin in the same 250 km x 1 Sol orbit and immediately perform an instantaneous deorbit maneuver at apoapsis to place the vehicle on an appropriate descent transfer trajectory. The notional trajectories shown in the figure begin at entry interface. The ellipsled simulation (blue curve) uses bank angle to control entry acceleration during a high altitude constant G phase. Following this phase the vehicle flies full lift-up until aeroshell jettison. Approximately 3.5 km above ground, the ellipsled aeroshell is jettisoned and the descent stage engines are immediately ignited. The engines are commanded to hold 3 (Earth) G's until reaching the next event. At 12.5 meters above the surface, the vehicle executes a brief hover phase, maintaining a constant velocity of 2.5 m/s for 5 seconds. The POST targeting algorithm perturbs the deorbit ΔV and the 3.5 km engine start/aeroshell jettison altitude to ensure that the hover phase ends at zero geodetic altitude (GDALT = 0).

In the MIAS simulation (red curve), a coasting phase occurs after entry interface where the aeroshell is used to decelerate the vehicle. MIAS aeroshell jettison and the initiation of powered flight is controlled with an event timer whose duration is controlled by the targeting algorithm to ensure the landing condition is met. For this simulation, the deorbit ΔV is fixed. During the

powered flight phase, generalized acceleration steering is used to hold a constant sensed acceleration of 3 (Earth) G's.

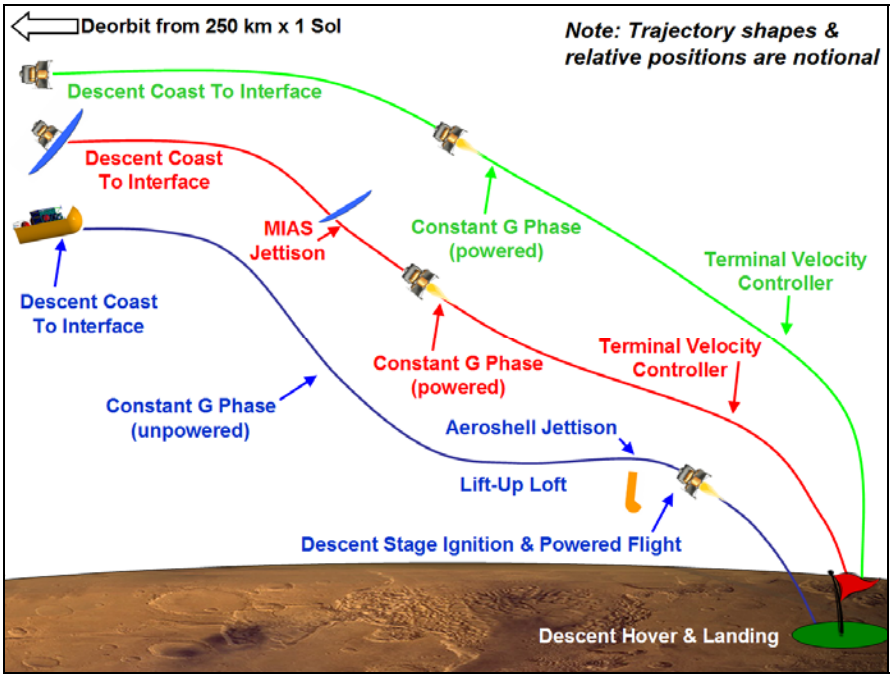

Figure B.3.1. Simulation Concept of Operations

Below 15 km, a terminal velocity controller is activated. This controller follows a linear velocity profile to the surface and prevents large required accelerations late in the trajectory. While this approach incurs more gravity losses than an uncontrolled approach, it also absorbs parameter variations and prevents small changes in initial conditions from generating disproportionately large changes at landing. The purpose of the controller is to enhance the robustness of the simulation and to add a degree of realism since a controlled landing approach would be utilized on any piloted Mars landing missions. Following this controlled approach phase, the vehicle uses the same brief hover phase utilized by the ellipsled to land.

The all propulsive simulation (green curve) utilizes many of the same flight phases as the MIAS simulation. It begins in the same orbit, employs a constant G powered deceleration phase, uses the same terminal velocity controller, and the same landing approach hover as the MIAS simulation. The primary difference is the lack of an aerodynamic deceleration phase and aeroshell jettison event. It is important to note that this simulation does not include any maximum heat rate or heat load constraints. Since the mass model does not include an allotment for any thermal protection, this simulation offers a best-case performance result. Other all

propulsive concepts require thermal constraints to be considered during entry which add ΔV losses and ultimately propellant mass to the converged solution.

Based on the above operational concepts, mass models must be developed for several architecture elements:

- Descent stages for all three simulations
- Ellipsled aeroshell
- Ellipsled RCS system
- MIAS aeroshell
- MIAS RCS system

Descent Stage Modeling Approach

Weight and sizing models for suitable descent stages were developed in EXAMINE, an Excel-based weights and sizing algorithm developed by D.R. Komar (NASA LaRC). The EXAMINE models were used to generate descent stage inert masses as a function of several input variables. Since the output data depends on more than one input variable, multivariate regression analysis was chosen as the primary descent stage modeling tool. Regression equations are compact, may be quickly evaluated, and are easily integrated into the POST simulation framework. However, since regression equations are created from a discrete set of model data, they have finite input variable ranges which, if violated, can lead to large modeling errors.

Multivariate regression analysis leads to a polynomial equation typically of the form

$$R = \exp\left[b_o + \sum_{i=1}^{h} b_i x_i + \sum_{i=1}^{k} b_{ii} x_i^2 + \sum_{i=1}^{l} b_{ij} x_i x_j \right] \qquad (1)$$

where
h represents the number of linear coefficients
k represents the number of pure quadratic coefficients
l represents the number of cross term coefficients
b_o represents the intercept
b_i represents the linear coefficients
b_{ii} represents the pure quadratic coefficients
b_{ij} represents the cross term coefficients
x_i and x_j represent the ith and j^{th} input variables respectively
R represents the desired response.

Once the input variables $x_i, \ldots, x_j$ are known, the response R (which is the descent stage inert mass in this case) may be directly computed by evaluating equation (1). The statistical analysis

software package JMP® by SAS was used to construct the multivariate regression equations using a standard least squares approach. Due to the low computational cost of each EXAMINE simulation, a full factorial experimental design was selected for all the descent stage models: that is, data was generated for each combination of parameters since these computations could be completed using only a small amount of computation time.

The resulting response surfaces were tested against several figures of merit to ensure that an accurate representation of the point-design data was achieved. These figures of merit include:

(a) *Coefficient of Determination*

The coefficient of determination (also known as R^2) is a measure of the variation in the response around the mean due to the model and not random error. This metric should approach unity to ensure a good fit.

(b) *Actual-vs-Predicted Data*

Inspection of the actual data against the predicted data, usually in the form of a scatter plot, visually displays the deviation of the model from the true behavior. The predicted data is obtained by re-entering the data used to generate a candidate model back into the model itself. A actual-predicted data scatter plot should closely adhere to a positively inclined 45° line with an intercept through the origin. The data should be evenly spread along the line with minimal clustering.

(c) *Model Fit Residuals*

The residuals illustrate the absolute error associated with the assumed model for each predicted value. Inspection of the residuals of the predicted data, typically in the form of a residual-vs-predicted scatter plot, should show a random dispersion of the data points about zero with no distinguishable pattern and a small magnitude relative to the predicted value.

(d) *Model Fit Error*

The model fit error is the percent deviation of the predicted data relative to the actual data used to create the model. A distribution of the model fit errors graphically depicts the percent error of the predicted data with respect to the actual data. The distribution should approximate a standard normal distribution such that the mean is close to zero and the standard deviation is less than or equal to unity.

(e) *Model Representation Error*

The model representation error is the percent deviation between randomly generated test data and the actual data used to create the model. A distribution of the model representation error

graphically depicts the percent error of the random data with respect to the actual data. This distribution should also approximate a standard normal distribution.

Response surfaces meeting the above criterion are typically sufficient for engineering approximations and conceptual design purposes. The models themselves and the critical fitting statistics are available in section IV.

Ellipsled Aeroshell Modeling Approach

The model data for the ellipsled structural and thermal protection masses was generated by Dave Kinney (NASA Ames). This data was compressed into a table look-up model for integration into the mass subroutines (see Section IV).

Ellipsled RCS, MIAS RCS, and MIAS Aeroshell Modeling Approach

Each of these models has output data which is a function of a single input variable. Therefore these elements may be modeled using a monovariate linear least squares approach. (Note: The multivariate regression analysis used to generate the response surfaces uses linear least squares to fit a set of output data which depends on several input variables simultaneously.) The MIAS aeroshell and RCS models are derived from the original Astrium report (see references B.2 and B.3). The ellipsled RCS model is based on the RCS system masses of robotic Mars missions developed by Juan Cruz (LaRC).

IV. Default Simulation Mass Models

This section presents the various models implemented in the default simulations as well as fitting statistics and other relevant data.

Descent Stage Response Surfaces

The descent stages modeled in EXAMINE for the three default simulations are closely related to one another. All are based on a legged lander concept (see Figure B.4.1) similar in form to the Apollo lunar module descent stage. The descent stage consists of:

- Four spherical composite propellant tanks with a multi-layered insulation (MLI). Overwrap and thermal control system (radiators) to inhibit propellant boiloff.
- A thrust structure which acts as the vehicle frame.
- A landing gear assembly.
- Four pump-fed LOX/CH4 rubber engines (sized using required vehicle T/W and given engine uninstalled T/W):
    - Oxidizer: Fuel (O/F) ratio = 3.5
    - 600 psi chamber pressure

    - Assumed nozzle expansion ratio = 200
- A power management and distribution system (PMAD).
- A small avionics package to route throttling and engine commands from the payload to the engine control/health management systems.
- A 30 percent dry mass margin (matches DRA 5/6).
- Propellant tanks and feed/fill/drain system sized to include volume of residuals and pressurants.

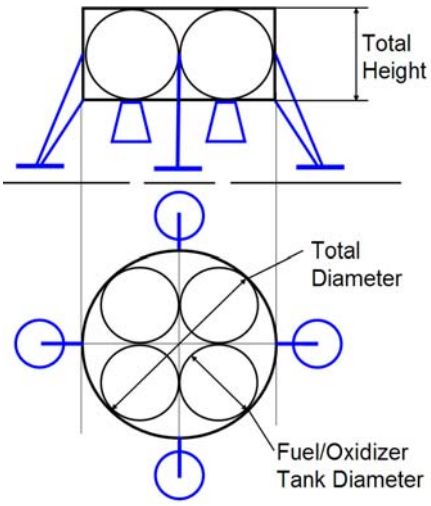

Figure B.4.1. EXAMINE Descent Stage Conceptual Layout

The three descent stages also make several common hardware assumptions:

- Keep-alive and flight power are drawn from the payload power supply bus.
- RCS and associated hardware is mounted on the payload. This is largely due to the assumption that the ascent stage will require an RCS system and that cargo-only missions may add an RCS to the payload.

Table B.4.1. Descent Stage Inert Mass Response Surface Input Variable Ranges

Ellipsled					
Variable	Minimum	Baseline	Maximum	Units	Comments
Descent ΔV	500	1250	2000	m/s	Ideal ΔV based on constant I_{sp}
Deorbit ΔV	0	25	50	m/s	Ideal ΔV based on constant I_{sp}
T/W_{system}	3	4.5	6	Earth G's	
T/W_{engine}	30	60	90		
Mission Payload Mass	20	45	70	mT	
Aeroshell Mass	30	50	70	mT	
MIAS					
Variable	Minimum	Baseline	Maximum	Units	Comments
Descent ΔV	200	1100	2000	m/s	Ideal ΔV based on constant I_{sp}
Deorbit ΔV	0	25	50	m/s	Ideal ΔV based on constant I_{sp}
T/W_{system}	1	3.5	6	Earth G's	
T/W_{engine}	30	60	90		
Mission Payload Mass	20	45	70	mT	
Aeroshell Mass	0.7	5.35	10	mT	based on available MIAS test data
All Propulsive					
Variable	Minimum	Baseline	Maximum	Units	Comments
I_{sp}	369	634.5	900	sec	
T/W_{engine}	80	140	200		
T/W_{system}	1	2.5	4	Earth G's	
Total ΔV	3000	4500	6000	m/s	

The differences between the three models are the type and ranges of the input variables. Table B.4.1 lists the input variables for each of the decent stage response surfaces along with their associated ranges and units. Note that both the ellipsled and MIAS cases have the same input variables while the all propulsive case uses slightly different inputs.

This disparity in the input variables is due to the lack of an aeroshell in the all propulsive simulation. Because the all propulsive descent stage has no aeroshell, both the deorbit and descent maneuvers are performed with the same payload, the mission payload. The ellipsled and MIAS simulations perform the deorbit maneuver carrying both the mission payload and the aeroshell while the descent maneuver is performed with only the mission payload. This situation requires the tracking of two different ΔV measures for these simulations whereas only one is necessary in the all propulsive case. Note that only one mission payload is considered in the all propulsive simulation which corresponds to the 40 mT baseline payload mass used in Mars Design Reference Architecture (DRA) 5.0 (Table B.4.2). Also note that only one I_{sp} is

considered for the ellipsled and MIAS simulations corresponding to the 369 second I_{sp} used in DRA 5.0. While the all propulsive response surface can accommodate a range of I_{sp}, only LOX/CH4 propellant densities are used for vehicle sizing. Therefore, this model looses fidelity for fuel/oxidizer combinations significantly different from LOX/CH4. Since the required propellant mass is computed by POST2, propellant bulk density errors will only effect the descent stage inert mass modeled by the all propulsive response surface.

Table B.4.2. Descent Stage Inert Mass Response Surface Default Values

Ellipsled & MIAS		
Item	Value	Comments
Isp [sec]	369	***Note: only one I_{sp} considered***
# engines	4	
engine type	pump-fed	
O/F Ratio	3.5	
Propellant	O2/CH4	
Chamber Pressure [psi]	600	
Nozzle Area Ratio	200	
All Propulsive		
Item	Value	Comments
Payload Mass [mT]	40	***Note: only one payload considered***
# engines	4	
engine type	pump-fed	
O/F Ratio	3.5	
Propellant	O2/CH4	
Chamber Pressure [psi]	600	
Nozzle Area Ratio	200	

The EXAMINE-derived response surface models for the descent stage inert masses may be seen in table B.4.3. These equations may be reconstructed into their polynomial form using equation (1). For example, the ellipsled descent stage inert mass may be expressed as

$$m_{inert} = \exp[8.014791 + 0.000322\Delta V_{descent} + 0.000333\Delta V_{deorbit} + \ldots$$
$$+4.36E\text{-}05(\Delta V_{descent})(T/W_{system}) + \ldots] \quad \text{(kg, 2)}$$

Table B.4.3. EXAMINE Derived Descent Stage Inert Mass Response Surfaces

Ellipsled		**MIAS**		**All Propulsive**	
Coefficient	*Value*	*Coefficient*	*Value*	*Coefficient*	*Value*
Intercept	8.014791	Intercept	7.50698341	Intercept	9.69143058
DescentDV	0.000322	descentDV	0.00020052	TWsys	0.6560922
DeorbitDV	0.000333	deorbitDV	0.00087792	Tweng	-0.0017234
Twsys	0.558493	Twsys	0.75175856	Isp	-0.0057324
Tweng	-0.02758	Tweng	-0.021925	DV	0.00041462
PL	5.11E-05	PL	0.00005348	Twsys*Tweng	-0.0015231
AS	2.69E-07	descentDV*Twsys	3.4128E-05	Twsys*Isp	-0.0005215
DescentDV*Twsys	4.36E-05	descentDV*Tweng	-2.2012E-06	Tweng*Isp	3.7750E-06
DescentDV*Tweng	-3.3E-06	Twsys*Tweng	-0.005155	Twsys*DV	5.2370E-05
Twsys*Tweng	-0.00487	descentDV*PL	5.2589E-10	Tweng*DV	-4.4098E-07
DescentDV*PL	8.46E-10	Twsys*PL	4.5708E-07	Isp*DV	-1.5094E-06
DeorbitDV*PL	-2.84E-09	Tweng*PL	-7.7423E-08	Twsys*Twsys	-0.0362629
Twsys*PL	3.59E-07	descentDV*descentDV	9.8210E-09	Tweng*Tweng	5.2231E-06
Tweng*PL	-9.62E-08	deorbitDV*deorbitDV	-4.1254E-05	DV*DV	6.9153E-08
DescentDV*AS	-5.46E-11	Twsys*Twsys	-0.0759616	Isp*Isp	1.1632E-05
DeorbitDV*AS	2.49E-09	Tweng*Tweng	0.00039685	Twsys*Tweng*Tweng	2.9848E-06
PL*AS	-3.05E-12	PL*PL	-5.9390E-10	Twsys*Tweng*Isp	7.6563E-07
DescentDV*DescentDV	-5.71E-08	deorbitDV*deorbitDV*deorbitDV	5.4153E-07	Tweng*Tweng*Isp	-5.8419E-09
Twsys*Twsys	-0.03608	descentDV*descentDV*Twsys	3.7979E-09	Twsys*Isp*Isp	5.1789E-07
Tweng*Tweng	0.000523	descentDV*Twsys*Twsys	1.3742E-06	Tweng*Isp*Isp	-4.6339E-09
PL*PL	-5.70E-10	Twsys*Twsys*Twsys	0.00494902	Isp*Isp*Isp	-7.8643E-09
DescentDV*DescentDV*DescentDV	2.37E-11	descentDV*descentDV*Tweng	-2.1900E-10	Twsys*Twsys*DV	6.6688E-07
DescentDV*DescentDV*Twsys	1.27E-08	descentDV*Twsys*Tweng	-4.1817E-07	Twsys*Tweng*DV	-8.4039E-08
DescentDV*Twsys*Twsys	4.17E-06	Twsys*Twsys*Tweng	1.2759E-05	Tweng*Tweng*DV	8.0749E-10
Twsys*Twsys*Twsys	0.002461	descentDV*Tweng*Tweng	2.3237E-08	Twsys*Isp*DV	-9.7616E-08
DescentDV*DescentDV*Tweng	-7.93E-10	Twsys*Tweng*Tweng	3.1034E-05	Tweng*Isp*DV	8.5557E-10
DescentDV*Twsys*Tweng	-1E-06	Tweng*Tweng*Tweng	-2.2634E-06	Isp*Isp*DV	1.4370E-09
Twsys*Twsys*Tweng	-0.00013	descentDV*Twsys*PL	-1.2170E-10	Twsys*DV*DV	4.7424E-09
DescentDV*Tweng*Tweng	5.65E-08	Twsys*Twsys*PL	-3.4885E-08	Tweng*DV*DV	-4.2210E-11
Twsys*Tweng*Tweng	4.19E-05	descentDV*Tweng*PL	2.9097E-12	Isp*DV*DV	-8.9390E-11
Tweng*Tweng*Tweng	-3.4E-06	Twsys*Tweng*PL	-7.6450E-10		
DescentDV*DescentDV*PL	-9.85E-14	Tweng*Tweng*PL	2.3984E-10		
DescentDV*Twsys*PL	-1.82E-10	descentDV*PL*PL	-2.1960E-15		
Twsys*Twsys*PL	-5.30E-08	Twsys*PL*PL	-1.1990E-12		
DescentDV*Tweng*PL	5.24E-12	Tweng*PL*PL	2.9282E-13		
Twsys*Tweng*PL	1.81E-09	PL*PL*PL	2.9364E-15		
Tweng*Tweng*PL	2.69E-10				

Ellipsled		MIAS		All Propulsive	
Coefficient	*Value*	*Coefficient*	*Value*	*Coefficient*	*Value*
Twsys*PL*PL	1.72E-12				
Tweng*PL*PL	2.57E-13				
PL*PL*PL	2.82E-15				

Variable Key:
DescentDV = descentDV = Descent ΔV
DeorbitDV = deorbitDV = Deorbit ΔV
Twsys = T/W_{system}
Tweng = T/W_{engine}
PL = mission payload
AS = aeroshell mass (structure, thermal protection, & aeroshell RCS)
Isp = specific impulse, I_{sp}
DV = total ΔV (all propulsive only)

These response surfaces reside in calspe.f and are called after each trajectory run using by setting the convergence flag MEDL_MCONV to unity. The response surface independent variables listed in Table B.4.1 are drawn from information in the input deck and from values computed during the trajectory. After the inert mass is computed, the current payload capability of the vehicle is computed by subtracting the inert mass from the landed mass which POST obtains at the end of each trajectory iteration.

Descent Stage Response Surface Fitting Metrics

The goodness-of-fit metrics discussed in Section 3 were used to evaluate the accuracy of the response surface curve fits. Analysis of goodness-of-fit is primarily a graphic procedure based on recognition of data patterns and abnormalities. To spare the reader the details of the statistical analysis, the fitting graphics have been omitted. The essential fitting results which convey the accuracy of the response surfaces are available in Table B.4.4. In order for a curve fit to be valid, the value of R^2 should approach unity. In addition, the spread of the residuals scaled by the minimum predicted value (presented in the second row of Table B.4.4) should generally lie below 5 percent. All of the response surfaces meet these criteria.

The remaining three rows of Table B.4.4 present the critical parameters describing the model representation error (MRE) distribution. This distribution is created by generating random input vectors to the response surfaces and calculating the percent error between the actual inert mass and the predicted inert mass for each vector. Valid response surface error distributions should closely approximate a standard normal distribution with a mean near zero and a standard deviation near unity. Recall that for a normal distribution, 99.7 percent of the possible errors should be within ±3σ of the mean. Note that the standard deviation presented in Table B.4.4 is a sample standard deviation which approximates the population standard deviation, σ. All of the MRE's presented here have a shape closely approximating a normal distribution with means and standard deviations as listed in Table B.4.4. The ellipsled and MIAS cases both have tight error distributions with low maximum observed errors. The all propulsive case, however, has a wider distribution due to the wide variable ranges examined.

Table B.4.4. Descent Stage Inert Mass Response Surface Fitting Statistics

Item	Ellipsled	MIAS	All Propulsive	Comments
R^2	0.999685	0.999899	0.998085	
Residual Plot	0.869565	0.757575758	2.78	%
Mean % Error	0.00716	-0.025582	0.2058705	of the model representation error distribution
Standard Deviation (% Error)	1.065	0.7905835	2.5882445	of the model representation error distribution
Max Observed % Error	5.638	4.08	-9.533	of the model representation error distribution

The MRE data should not be confused with the model fit error (MFE) distribution which describes the error between the response surface and the data used to generate the response surface. Generally, the MRE is a wider distribution (i.e., has a larger standard deviation) and is therefore a more conservative benchmark for determining response surface accuracy. Since, for the three surfaces presented here, the differences between the means of the MFE and MRE distributions were small and because the MRE's all had larger standard deviations, only the MRE's are presented. This is to avoid potential misuse of the MFE as the accuracy benchmark in place of the MRE.

Ellipsled Aeroshell & RCS

An ellipsled aeroshell is similar in form to a sphere-capped cylinder with either an open or closed leeward section (Figure B.4.2) Both the descent stage and the mission payload are loaded into the cargo bay area of the ellipsled for the entire first segment of the mission up to aeroshell jettison during descent and landing. The RCS system on the ellipsled is an integral part of the aeroshell structure and is used for directional control during the entry phase of the mission.

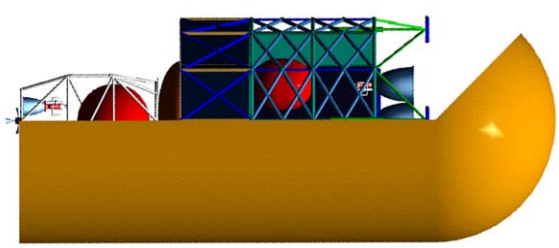

Figure B.4.2. Conceptual Ellipsled Aeroshell with Notional Payload

The ellipsled aeroshell mass model is in a table look-up format as a function of the orbital period, the aerodynamic reference length, and the mission payload mass (Table B.4.5). Note that only

discrete values for the mission payload (20,000 kg, 40,000 kg, & 70,000 kg) and the reference length (10m, 12m) may be used; i.e., this model is not valid for interpolation between these payloads and reference lengths. The stated aeroshell masses include both the structural mass and the mass of the required thermal protection system needed for atmospheric entry.

Table B.4.5. Ellipsled Aeroshell Mass (Structure & Thermal Protection)

Payload	Reference Length	Orbit Period	Aeroshell Mass
kg	m	min	kg
20000	10	>1000	37972.32
20000	10	<1000	40283.18
20000	12	>1000	55247.8
20000	12	<1000	56360.1
40000	10	>1000	37100
40000	10	<1000	41726.9
40000	12	>1000	57408.84
40000	12	<1000	59356.5
70000	10	>1000	44736.92
70000	10	<1000	43892.48
70000	12	>1000	60650.4
70000	12	<1000	63851.1

The ellipsled aeroshell RCS mass model is a regression equation based on previous Mars robotic missions. This equation, once reduced to simplest terms, is a simple scaling factor applied to the entry mass. The RCS system mass may be computed as

$$m_{RCS} = 0.01996(m_{entry}) \qquad \text{(kg, 3)}$$

where m_{entry} is the current value of MEDL_MGUESS.

MIAS Aeroshell & RCS

The MIAS is a Lavochkin/Babakin scaled conceptual design (Figure B.4.3) based on a small reentry test vehicle, the Inflatable Reentry and Descent Technology (IRDT) demonstrator. Similar in function to the ellipsled aeroshell, the MIAS shields the payload from the entry thermal loads and is jettisoned during the landing phase. The primary differences between the MIAS and the ellipsled are the geometric configurations and in the inflatable/deployable nature of the MIAS concept. An inflatable aeroshell has the potential to reduce the Earth launch mass of the entry stack. The subsequent flow-down mass impacts of such a reduction make inflatable

aeroshell options attractive and the subject of much current research. The MIAS RCS, like the ellipsled RCS, is an integral part of the MIAS structure.

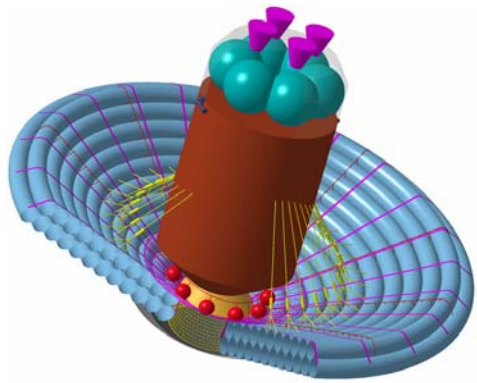

Figure B.4.3. MIAS Conceptual Design Layout

The MIAS aeroshell and RCS mass models were generated from data available in the original MIAS Design Definition Report from Astrium GmbH (see reference 3). The report presents conceptual mass equipment lists (MELs) for three entry masses, two discovery class missions (4mT & 6mT) and one exploration class mission (60-70 mT). The correlation between the three breakdowns is highly linear both on the system and on the subsystem levels. Consequently, the mass models derived from these mass MELs are also linear. The MIAS aeroshell mass may be expressed as a function of the entry mass;

$$m_{MIAS} = 133.07(m_{entry}/1000) + 151.57 \qquad \text{(kg, 4)}$$

where m_{entry} is the current value of MEDL_MGUESS in kg. The RCS mass for the MIAS aeroshell may be expressed as

$$m_{MIAS\text{-}RCS} = 14.269(m_{entry}/1000) - 17.39 \qquad \text{(kg, 5)}$$

where m_{entry} is the current value of MEDL_MGUESS in kg. Since most of the original data appears to be a linear extrapolation between the three given MELs, small extrapolations past the entry mass upper limit (70 mT) would appear to be no more or less valid than entry masses falling within the model limits.

V. Simulation Test Cases

Three POST2 input decks are available to model the three simulations discussed above. These input files were provided as part of the simulation architecture for the Rapid EDL Analysis assessment and can be accessed on the NASA Langley Atmospheric Flight and Entry Simulation Branch computer under the REDLASim directory on the root level. Each simulation may be

used as an example or starting point for future simulations or as a test to ensure the mass/sizing calculation processes are correct.

General Mass/Sizing Computational Scheme

All simulations use the same general scheme to implement the mass models. When the simulation begins, POST2 begins the mass initialization procedure for the first trajectory iteration. It is at this time that mass EDL.f is called by the weight initialization routine, wgtini.f. The massEDL.f routine uses the current entry mass guess (MEDL_MGUESS) as the mass of the total entry vehicle stack. It computes the aeroshell and aeroshell RCS masses based on the entry mass and any other required inputs. Since the mission payload is a user-defined input, the only remaining element of the entry vehicle stack whose mass is unknown is the descent stage total mass (which includes both the inert and propellant masses). The response surfaces are not used at this time since the propellant mass needed to execute the trajectory is not yet known.

The descent stage total mass is computed by subtracting the aeroshell, aeroshell RCS, and mission payload masses from the current entry mass guess. The sum of the mission payload mass and descent stage total mass are stored as the first step weight (WSTPD1) and the sum of the aeroshell mass and aeroshell RCS masses are stored under the second step weight (WSTPD2). The masses are, of course, converted to weight just prior to storage as a step weight. This allows the aeroshell and its RCS system to be jettisoned simultaneously by removing the second step weight from the model. For the all propulsive simulation, which lacks an aeroshell and an aeroshell RCS system, the second step weight is set to zero.

At touchdown, the convergence calculations in calspe.f are activated by setting the convergence flag (MEDL_MCONV) to unity. Now that the trajectory is known, all of the required response surface inputs (see Table B.4.1) are known and the descent stage inert mass (MEDL_MDSIM) may be computed. This inert mass is subtracted from the touchdown mass (MEDL_MTDWN) and the result is the payload capability (MEDL_MPLCALC) of the entry stack for the current iteration. The standard POST targeting algorithms treat this payload capability as any other target variable. If the payload capability does not equal the desired mission payload (MEDL_MPL) then the entry mass guess is iterated and the trajectory is recomputed.

Expected Simulation Results

The nominal input decks for the three default simulations should run to completion and converge on stable solutions if all computations are implemented as intended. The essential results for the parameters of interest are available in Table B.5.1.

Note that the MIAS payload mass is half of the payload selected for the ellipsled and all propulsive simulations. The MIAS payload is necessarily small to prevent gross extrapolation of the MIAS descent stage response surface. Recall that this surface is only valid up to an aeroshell mass of 10 mT. Doubling the mission payload would generate an increase in required entry mass which would increase the aeroshell mass and violate the variable boundary.

Table B.5.1. Nominal Simulation Results for Main Variables of Interest

POST Variable	Units	Ellipsled	MIAS	All Propulsive	Comments
medl_mcount	-	11	11	10	iteration counter
medl_mguess	kg	107759.542	65546.4048	307862.432	guessed entry mass/initial mass or vehicle stack
medl_mpl	kg	40000	20000	40000	mission payload mass, convergence target variable
medl_mashtot	kg	39250.3586	9791.72223	0	aeroshell + aeroshell RCS mass
medl_mdestot	kg	28509.183	35754.6826	267862.432	descent stage total mass (propellant + inert mass)
medl_mdsim	kg	17184.9759	12671.3262	51875.7505	descent stage inert mass (mdestot - usable propellant)
medl_mtdwn	kg	57184.2071	32680.5538	91875.5957	touchdown mass
medl_mplcalc	kg	39999.2312	20009.2276	39999.8452	calculated payload capability at touchdown for current iteration
medl_mperiod	min	0	0	0	period for ellipsled for aeroshell sizing table*
medl_mdv	m/s	630.266204	1917.25542	0	descent ΔV (total ΔV - deorbit ΔV)
medl_mpropcon	kg	0	0	215986.837	all propulsive consumed propellant mass
dvimag	m/s	14.9781669	13.4	13.498365	deorbit ΔV
xmax2	Earth G	3.00018485	3.24536806	2.62041539	maximum sensed acceleration
videal	m/s	645.244371	1930.65542	4375.7333	total ΔV

**Note this is always defaulted to zero except in the one printblock where the calculation is called.*

This limitation on the MIAS mass model is discussed in greater detail below (see section on Coincidental Extrapolation below). Several other details in table B.5.1 are significant;

- The value of MEDL_MGUESS in the final printblock is the converged entry mass
- MEDL_MPERIOD is always zero in the final printblock. This variable is only required for the ellipsled aeroshell sizing and appears as non-zero in the first phase of the nominal trajectory evaluation. For the default 250 km x 1 Sol orbit, the nominal period is 1476.4 minutes.

- MEDL_MDV represents the descent ΔV (i.e., the total ΔV – deorbit ΔV). The all propulsive simulation has no aeroshell to jettison and therefore all maneuvers are performed with the same payload. Therefore, the distinction between the descent and deorbit ΔV is unimportant in this case. To prevent confusion, MEDL_MDV is set to zero for this simulation and the value of VIDEAL is used to size the descent stage.
- MEDL_MPROPCON was added to the namelist while developing the all propulsive simulation as a check on the mass model development process. Consequently, it is only reported for the all propulsive simulation. The consumed propellant for any simulation may alternately be determined by subtracting the descent stage inert mass (MEDL_MDSIM) from the descent stage total mass (MEDL_MDESTOT).

Multibody Simulations & Single Iteration Cases

A multibody simulation was created late in the study to assess the feasibility of running parallel sets of mass computations in the same trajectory. This simulation *multi2.inp* pairs the MIAS default test case with the all propulsive test case. The mass results obtained for the multibody case are consistent with those obtained for the individual test cases as expected. This proves that the mass computations may be successfully utilized in a multibody simulation environment. The provided test case is not optimized or targeted to avoid convergence stability issues stemming from the inherent differences in the event structures for the two test cases. For more information on constructing multibody simulations, see the POST2 User's Manual.

Three single iteration cases were created for each of the default simulation types (MIAS, all propulsive, and ellipsled). These cases obtain the same results as the three default simulations except the single iteration cases converge immediately whereas the default simulations require multiple iterations.

Simulation Characteristics, Assumptions, & Limitations

The default POST2 input decks are all 3-DoF simulations which use Mars GRAM 2005 to determine a suitable average atmospheric profile. The atmosphere is static with no random number seeds to allow repeatable results to be obtained. The aerodynamic characteristics of the default entry vehicles may be found in Table B.5.2. The drag coefficient for the MIAS is taken from the MIAS design definition report for a zero angle of attack case. The drag coefficient remains relatively constant from the hypersonic regime down to low supersonic conditions.

Due to the lack of drag data for a legged lander in a base-first entry attitude, the lander was assumed to have drag characteristics similar to a blunt cone. Empirical measurements of blunt cone configurations in the hypersonic and high supersonic regimes indicate a drag coefficient near unity[B.4]. The ellipsled drag characteristics are taken from baseline DRA 5.0 simulations

which employ a 10x30 m ellipsled. Note that the ellipsled as currently defined flies a lifting trajectory in contrast to the MIAS and all propulsive simulations.

The nose radius of the entry vehicle is used by POST2 to compute the thermal loading using a Mars-adapted Sutton-Graves approach. The nose radius of the ellipsled was also taken from baseline DRA 5.0 simulations whereas the MIAS and all propulsive nose radii are assumed values. Note that the last row of Table B.5.2 gives the aeroshell diameter which is directly computed from the stated reference area in the second row. These reference diameters are assumed values which were found to yield stable mass/sizing solutions.

Table B.5.2. Entry Vehicle Aerodynamic Characteristics

POST Variable	**Units**	**Ellipsled**	**MIAS**	**All Propulsive**	**Variable**
LREF	m	10	10	10	aerodynamic reference length
SREF	m^2	78.53982	314.16	176.71	reference area
RN	m	6.5	10	10	aeroshell nose radius
CDT	-	2.96	1.55	1	constant drag coefficient
CLT	-	1.39	n/a	n/a	constant lift coefficient
n/a	m	10	20*	15	aeroshell diameter

**Represents the inflated MIAS diameter.*

The default simulations in their current form share several common assumptions which limit the fidelity and applicability of the results. The primary emphasis in the development of the test cases was centered on mass model development and integration. Consequently, these assumptions are not critical to the mass/sizing capability of the simulations. The following issues should be assessed before any of the default simulations are used for research purposes:

- For simplicity and reduced run time, jettisoned aeroshells are not tracked and pose a possible landing site impact hazard.
- To ensure convergence stability, simple constant drag models are used.
- Detailed guidance and control models are not implemented, again for simplicity and reduced run time.
- No aeroshell separation dynamics are modeled.
- Due to the lack of a detailed supersonic retropropulsion model, all drag coefficients are set to zero during powered flight. Consequently all lift and drag forces are zero during powered flight phases.

Coincidental Extrapolation

It is important to note the coupling between the MIAS descent stage mass model and the MIAS linear regression models. The data used to generate the descent stage mass model accounts for aeroshell masses up to 10 mT. A 10 mT MIAS aeroshell corresponds to a maximum allowed entry mass (MEDL_MGUESS) of roughly 66.9 mT. For MIAS aeroshell masses greater that 10 mT, the MIAS descent stage will be forced to extrapolate. This problem does not exist for the ellipsled simulation since the ellipsled aeroshell model is a table look-up and is therefore bounded. The ellipsled descent stage mass model can accommodate aeroshells beyond these bounds.

The above issue has been discussed in order to illustrate a potentially critical issue, namely coincidental extrapolation in which valid inputs to one model generate invalid inputs for another model. In the case of the MIAS model, the sensitivity of the descent stage inert mass to the aeroshell mass is very small. When the descent stage model data was analyzed in JMP, the coefficients associated with the aeroshell mass were all found to be statistically insignificant. Therefore, the aeroshell mass does not explicitly appear in the MIAS descent stage response surface. Since this is the case, a small degree extrapolation of the current MIAS response surface is permissible. Note, however, that for sufficiently large aeroshell masses, the effect of the aeroshell mass on the descent stage inert mass will no longer be negligible. Extrapolation beyond 5-10 mT over the stated ranges is discouraged.

VI. Conclusions

This document has summarized the key features and implementation methodology of an iterative mass/sizing routine embedded in the POST2 computational framework. The topics and examples discussed herein are intended to serve as a guidepost for the user in modifying the existing mass/sizing routines to suit current research efforts.

VII. References

B.1 Striepe, S., Powell, R., Desai, P., Queen, E., Brauer, G., Cornick, D., Olson, D., Petersen, F., Stevenson, R., Engel, M., Marsh, S., & Gromko A. *Program to Optimize Simulated Trajectories (POST II) Volume II: Utilization Manual.* Version 1.1.6.G. January, 2004.

B.2 Finchenko, V., Terterashvili, A., Stelter, C., & Wilde, D. *MIAS Design Development Plan*. Astrium GmbH, Doc. No. MIAS-RIBRE-RP-0003. NASA Contract No. 901128. December, 2002.

B.3 Finchenko, V., Ivankov, A., da Costa, R., & Stelter, C. *MIAS Design Definition Report.* Astrium GmbH, Doc. No. MIAS-RIBRE-RP-0002. NASA Contract No. 901128. December, 2002.

B.4 Intrieri, P., and Kirk, D. *High-Speed Aerodynamics of Several Blunt-Cone Configurations.* J. Spacecraft, Vol. 24, No. 2, 1987, pp 127-132.

Appendix C. Animation Tool v2.3

Written by Behzad Raiszadeh and Jeremy Shidner; Edited by Scott A. Striepe

I. Introduction

Typically the output from trajectory simulation programs is analyzed using two-dimensional time history plots. It is often desired to animate the trajectory where position and orientation of the bodies in flight are put into motion. The following is a MATLAB™ based tool that generates a movie from the trajectory data. The animations generated using this tool serve as an engineering analysis tool to gain further insight into the dynamic behavior of flight vehicles. This tool is able to animate a single body as well as multiple vehicles in flight, and has been tailored for output generated from POST2 simulations.

II. Method

This tool has been designed such that no code modification is required by the user. The animation tool obtains most of the required inputs from the POST2 input file, POST2 output file, and the Matfile (Figure C.1). The user is required to provide master_setup.txt, geom_data and camera_properties.txt files for additional parameters.

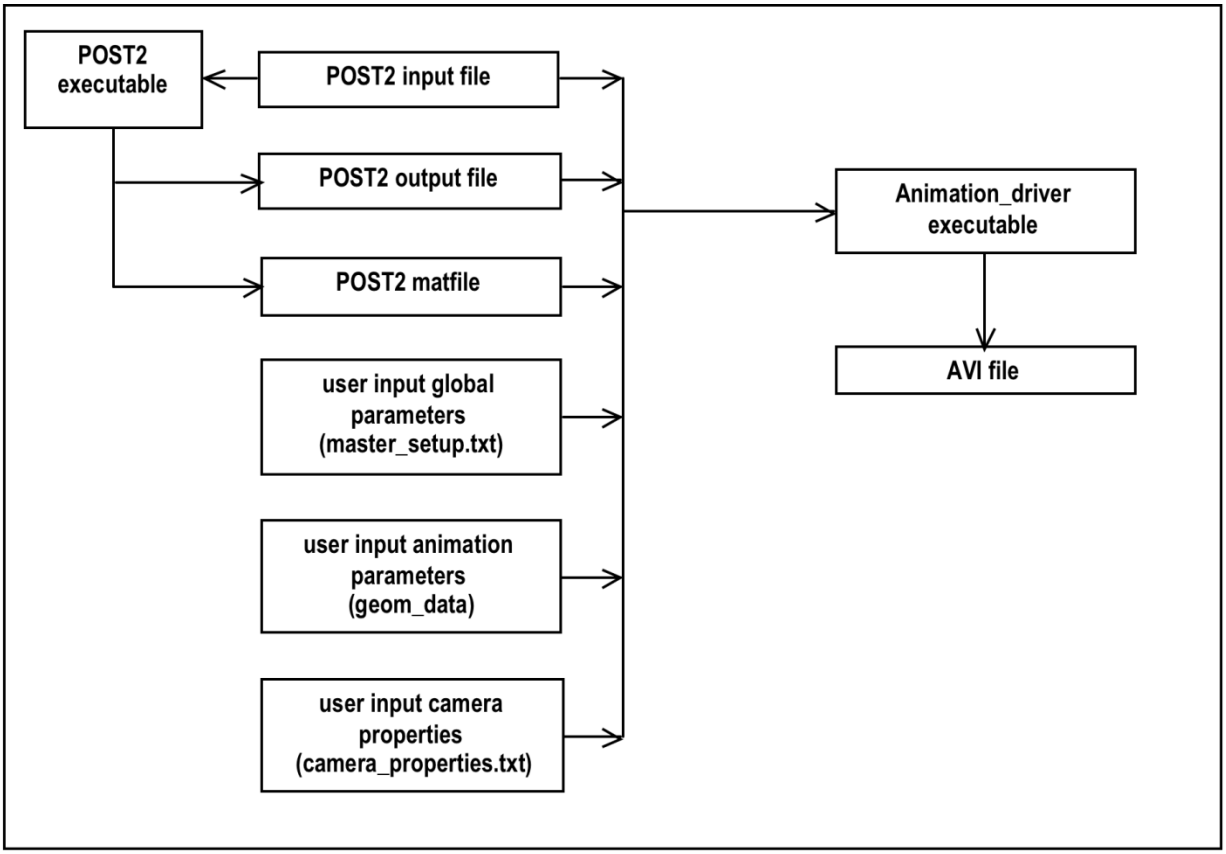

Figure C.1 Animation tool flow diagram

III. Step-by-Step Instructions

This section outlines the procedure, step-by-step, to make a trajectory animation. The animation tool has been developed in MATLAB™. All the source files have been compiled using the MATLAB™ Compiler. The compiled executable, named animation_driver.exe, is a stand-alone application and is used to generate trajectory animations in AVI format. The AVI file is compatible with both PCs and Macs.

Step 1: Installation

MATLAB™ Component Runtime (MCR) module needs to be installed on the host machine. A MATLAB™ license is not required for the MCR installation. The MCR installer utility program (MCRInstaller.exe) will be provided as part with the animation tool. The MATLAB™ Component Runtime installation procedure is provided in MATLAB™ Compiler User's Guide

[C.2], page 4-6. It is also available on the web. The instructions are provided for the Windows platform.

http://www.mathworks.com/access/helpdesk/help/toolbox/compiler/compiler_ug_collection.html

Windows installation
The following components make up the animation application on Windows platforms.

MCRInstaller.exe Self-extracting MATLAB™ Component Runtime library utility; Platform-dependent file that must correspond to the end user's platform.

animation_driver.exe	Animation application
animation_driver.ctf	Component Technology File archive; Platform-dependent file that must correspond to the end user's platform.
data_dict.txt	Holds all search and assignment variables for the Animation application.
PlumeGeom.mat	Specialized rocket plume data file.

The above files may be found at:

/new_home/rais/matlab/animation_V2.3/Executable/

Copy the above files to the local drive in the same directory as the trajectory data. The animation program is provoked by double-clicking the animation_driver.exe file.

Step 2: Set up local directory

The user needs to set up a local directory containing all the case specific files, such as the POST2 input file, output file, Matfile, and other setup files. These files contain mission specific data such as the trajectory, movie setup (frame rate, frame size, background color, lighting, etc), and links to where the geometries for the vehicles are stored. The following is a listing of local files:

POSTfile.inp
POSTfile.out
POSTfile.mat
master_setup.txt
camera_properties.txt
geom_data

Samples of these files can be found in the following directory:

/new_home/rais/matlab/animation_local_dir/

Step 3: POST2 trajectory data

The POST2 input file, output file, and the Matfile need to be copied to the local directory where they are processed and prepared for animation.

POST2 Input file

Some parameters are extracted from the POST2 input file. This includes planet rotation rate, mass of each vehicle, vehicles activation/deactivation, event sequencing, line activation/deactivation, line attach point locations, etc. The input parser also looks through all the included files to gather information. The user needs to make sure the path to the included files remains accurate if the input file and include files are copied from their original directory. If the input file contains a "* milestone read" statement, it should be replaced by "* include milecreate_fname.inp", where "milecreate_fname.inp" is the input file that created the binary milestone.

POST2 Matfile

The animation tool expects the following variables in the POST2 Matfile for each active vehicle. Ensure they are included in the profile. For POST2 variable definitions see Reference C.1. The burden of choosing a proper time interval between the data points is placed on the user. Choosing a proper time interval is a function of the desired movie frame rate and the apparent speed of the movie. A suggested time interval is 1/25th second, so that there are 25 frames available to the tool for every second of real time. See section 3.4 for further discussion.

time – trajectory time
longi – inertial longitude
decln – declination
ib matrix – inertial to body direction cosine matrix (9 elements)
xi, yi, zi – cartesian coordinates of the vehicle CM in inertial frame
gammar – planet relative flight path angle
azvelr – azimuth of the planet relative velocity vector
xcg, ycg, zcg – location of cg with respect to the BR frame

An example print block that may be used in the POST2 run can be found in the following directory: /misc/home0/rais/matlab/animation_local_dir/

POST2 output file

The event numbers were obtained from the POST2 input file, and the corresponding event times are extracted from the POST2 output file.

Step 4: Setting up master_setup.txt

Next, the user needs to fill in appropriate data for the master_setup.txt input file (shown in Figure C.2). This file lets the user specify data that is global in nature, such as the background color, number of movie frames per second, lighting parameters, movie name, and link to of various input files. The variable names are descriptive for convenience. All colors are input as three integers representing the Red, Green, and Blue (RGB) intensities ranging from 0 to 255. The entries in **bold** are required entries, and the entries in *italic* are informational and will be displayed with a dial or as a bar graph. Display of informational parameters is completely optional. Other parameters in plain font are required parameters but will default to the values listed below if the user does not provide input.

```
post_input_file        ="c1_animation.inp"      // mandatory entry
post_output_file       ="c1_animation.out"      // mandatory entry
post_matfile           ="c1_animation_25fps.mat"// mandatory entry
movie_name             = mera_take2.avi         // default output.avi if not input
frames_per_second      = 25                     // mandatory entry
projection             = perspective            // other option: orthographic
num_pixels_horizontal  = 800                    // movies always 3 X 2 aspect ratio
ground_geom            ="landing_patch.mat"     // ground patch geometry file

light_color            = 255  255   255         // by default, only one light source
light_color2           = 210  210   160         // additional optional light source
light_position_ned     =   0    0  -500         // default position of light 1
light_position_ned2    =  -4   30  -100         // position of additional light
background_color       =   0    0     0         // background black by default
gauge_color            = 150  150   150         // all gauges in gray by default

starting_index         = 1                      // starting index defaulted to 1
ending_index           = 3516                   // defaulted to the length of trajectory
step                   = 1                      // use higher numbers to reduce run time

altimeter_var          = gdalt                  // altitude up to 100 km
radar_altimeter_var    = hgtagl                 // altitude up to 3000 m
G_meter_var            = asmg                   // acceleration in Earth Gs
mach_meter_var         = mach                   // mach meter
airspeed_var           = velr                   // airspeed gauge
horiz_vel_var          = N_vel E_vel            // horizontal velocity gauge
fuel_gauge_var         = wprop                  // fuel usage
fuel_capacity          = 234                    // amount of fuel when full
vertical_speed_var     = wr                     // vertical speed gauge
attitude_indicator_var = pitr rolr              // attitude indicator gauge
turn_coordinator_var   = beta turn_rate         // turn coordinator gauge

altimeter_assigned_spot        = 5              // default altimeter spot
radar_altimeter_assigned_spot  = 6              // default radar altimeter spot
G_meter_assigned_spot          = 7              // default G meter spot
mach_meter_assigned_spot       = 8              // default mach meter spot
airspeed_assigned_spot         = 9              // default airspeed gauge spot
```

```
horiz_vel_assigned_spot      = 10          // default horizontal vel gauge spot
fuel_gaguge_assigned_spot    = 11          // default fuel usage spot
vertical_speed_var           = 12          // default vertical speed gauge spot
attitude_indicator_var       = 13          // default attitude indic gauge spot
turn_coordinator_var         = 14          // default turn coord gauge spot

display_var                  = gammar_1    // displayed as a bar graph
display_var2                 = velr_1      // display of a second bar graph
display_var_put_right        = 1           // displayed on the right side
```

Figure C.2. master_setup.txt listing

The frames_per_seconds parameter specifies the number of frames that are combined to make one second of animation. If the reciprocal of this parameter is equal to the time interval provided by the Matfile, then the movie will appear to be in real time. Other combinations of frame rate and output time increment can be used to simulate slow motion or fast-forwarding effects. For example, if frames_per_seconds = 20 and Matfile time interval is 0.1, the movie will appear to be twice the normal speed. The recommended values for frames_per_seconds are 20 and 25. The movie will be choppy for small frame rates around 10, and the monitor displays cannot keep up with frame rate of 50 and higher.

Ground_geom specifies a Matfile that holds ground geometry. For Mars animations, this information is obtained from MOLA data. The program that generates the Matfile is described at: /misc/home0/rais/matlab/animation_mola/README.txt

Ground geometry needs to cover the latitude and longitude range as spanned by the POST2 output file.

Starting_index, ending_index and step specify a subset of the trajectory to be animated. If left out, starting_index will default to one, ending_index will default to the length of the trajectory, and step equals one. By default, the entire trajectory is animated.

Display_var = 'POST variable' fields are optional. The POST2 variables specified in this field are displayed on the bottom of the screen as moving bar graphs. The variable names need to be exactly as they appear in the POST2 Matfile. By default, the bar graphs move horizontally from left to right when increasing in value. Bars graphs are stacked from bottom to top if more than one is requested. A total of 5 bar graphs may be requested. If desired, the bar graphs can be displayed on the right of the screen. This is accomplished by exercising the display_var_put_right = 1 option.

The animation tool supports the display of some parameters by dials and gauges. The following parameters are available for dial display: time, net velocity, horizontal velocity, acceleration in Gs, Mach number, altitude up to 100 km, altitude up to 3000 m, spent fuel, vertical speed, pitch,

roll, beta, and turn rate. Dials and gauges are automatically assigned a spot on the screen, but the spot assignment can be overridden by the user in the animation_setup.txt file. Figure C.3 shows the default dial and gauge spot assignments.

5 altimeter	9 airspeed	10 horizontal velocity	11 fuel gauge	1 clock
6 radar altimeter				2 not used
7 G meter				3 not used
8 mach meter	12 vertical speed	13 attitude indicator	14 turn coord	4 not used

Figure C.3. Default Gauge Locations

Figure C.4 is an example of all the gauges being used on a Mars entry animation. All the gauges are placed in their default location in this example.

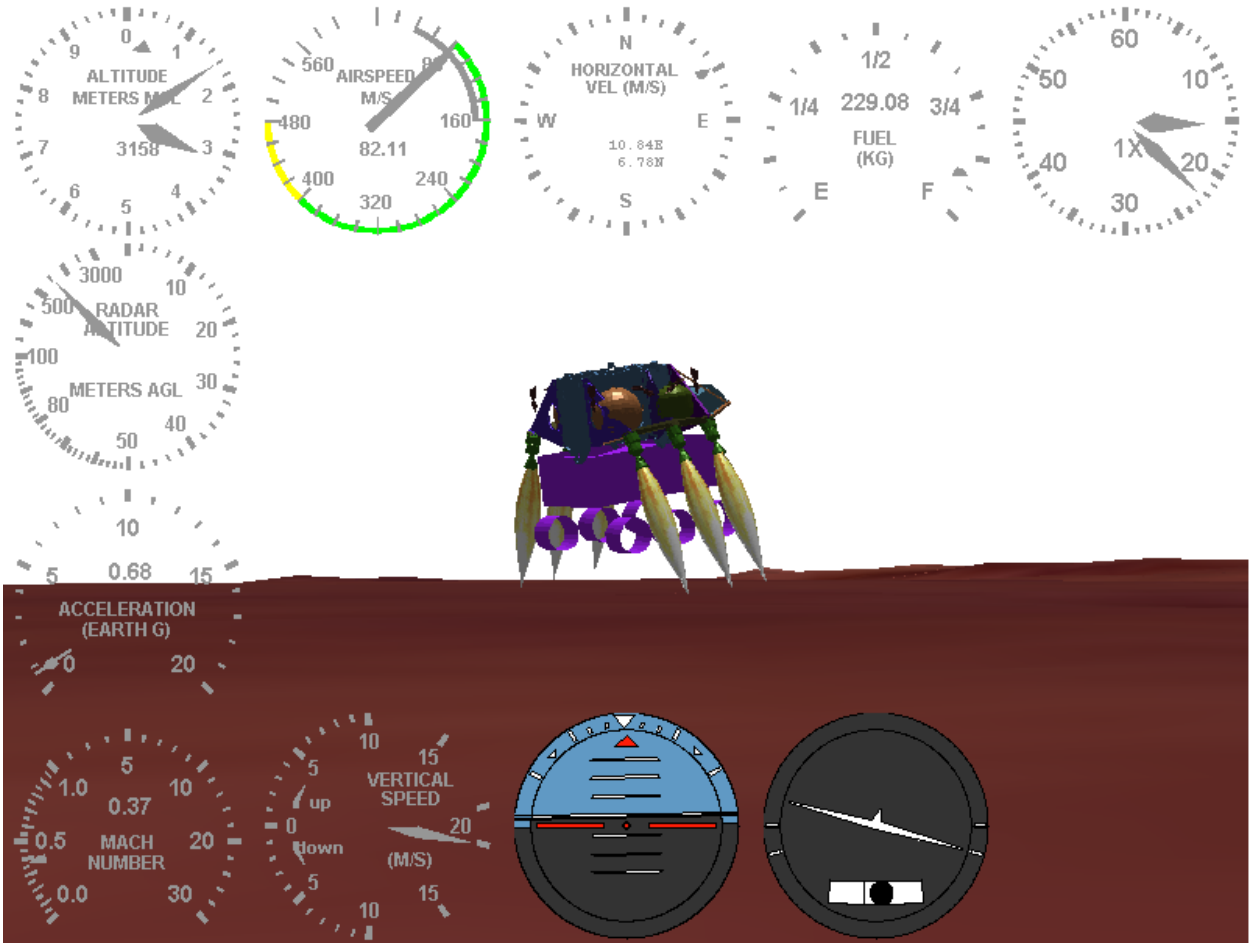

Figure C.4. Example of Gauges

Step 5: Setting up geom_data

The geom_data text file is responsible for setting up the geometrical representation of vehicles in flight (see Figure C.5). The syntax of geom_data is similar to the POST2 input deck. Vehicle geometries are input at the start of POST2 events. The animation tool expects a geometry input for each vehicle when first activated. Geometry input for an existing vehicle overwrites the previous input. Overwriting previous geometry can be used to reflect vehicle appearance changes such as jettisoning of spent stages, heatshield separation, parachute deployment, etc. The animation tool processes this file along with the POST2 input file.

```
event         = 1                    / event to add vehicle
vehicle       = 1                    / vehicle number relative to POST2
veh_geom_file  = "patch_data/capsule"      / relative path to vehicle geometry
incl_cg_calc   = 1                   / include in composite cg calculation
incl_body_axes = 1                   / display body axes
eng_scale      = 1 1 2 2             / scaling of rocket engine plumes
throttle_var01 = gvrc1               / thruster throttling variable 1
throttle_var02 = gvrc2               / thruster throttling variable 2
throttle_var03 = gvrc3               / thruster throttling variable 3
throttle_var04 = gvrc4               / thruster throttling variable 4

event         = 45                   / event to add vehicle
vehicle       = 2                    / vehicle number relative to POST2
veh_geom_file  = "patch_data/mer_parachute" / relative path to vehicle geometry
incl_cg_calc   = 1                   / include in composite cg calculation
incl_body_axes = 1                   / display body axes
eng_scale(2)   = 3                   / scale rocket engine 2 by factor of 3
throttle_var01 = gvrc1               / thruster throttling variable 1
throttle_var02 = gvrc2               / thruster throttling variable 2
throttle_var03 = gvrc3               / thruster throttling variable 3
throttle_var04 = gvrc4               / thruster throttling variable 4
```

Figure C.5. geom_data sample

The geometry is input as a collection of MATLAB™ patch elements. For more information on patches refer to MATLAB™ graphics manual. Patches are ideal for visualizing 3D objects such as aerospace vehicles and spacecrafts of arbitrary shapes. When drawing patches, the vertices must be input in POST2 Body Reference (BR) frame. The animation tool makes appropriate transformations to shift the geometrical representation of the vehicles to a planet relative coordinate system. The animation tool loads the geometrical data file specified on the right hand side of the veh_geom_file entry above. The file that is loaded contains an array of structures stored in geom_data with each element of the array making up a part. Each element of the geom_data array contains the coordinates of the vertices, polygons and color information. Color can be input in two ways. The first method is to specify a uniform color for a part using geom_data(n).color = [R G B] option. The animation tool also accommodates for each face or vertex to have its own color. This is accomplished by providing an additional color_array matrix of data in geom_data structure .The number of entries in color_array field must correspond to the number of entries in the faces or vert0 variables, with each entry being the RGB intensity. The wireframe model of the Mars Exploration Rover entry capsule shown in Figure C.6 contains 1152 vertices, and 1104 polygons.

geom_data(1).vert0: [1152x3 double]
geom_data(1).faces: [1104x4 double]
geom_data(1).color: [226 204 190]

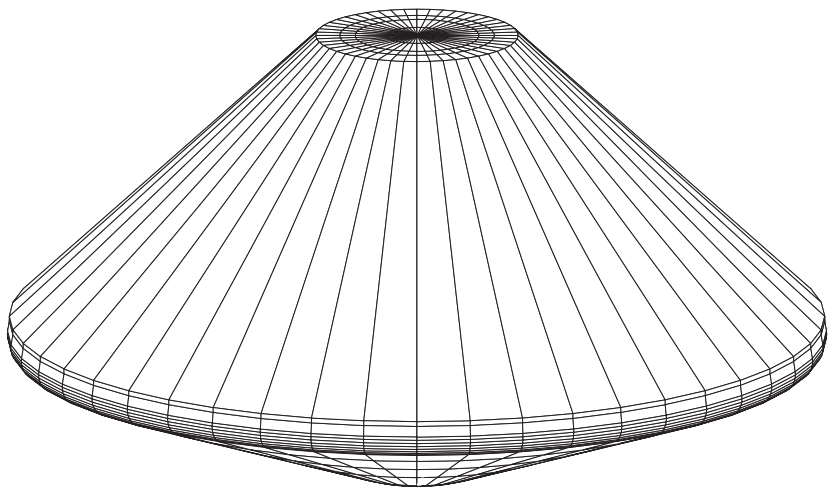

Figure C.6. MER Entry Capsule Model

Camera position and target are specified with respect to the composite vehicle center of mass. For a single vehicle in flight, composite vehicle center of mass is the same as the vehicle center of mass, but for multiple vehicle trajectory simulation, the composite vehicle center of mass is calculated at each time step. The user can exclude a vehicle from composite center of mass calculation by setting the incl_cg_calc flag to zero in the geom_data file. This is useful when a body is being jettisoned, where the desired focus of the camera is the main vehicle, not the jettisoned body.

Step 6: Setting up camera_properties.txt

The camera special effects are accomplished by manipulating the camera_properties.txt file. This file contains the camera position and camera target with respect to the composite vehicle center of mass as a function of time or event number. Event number input may be denoted as 'evxxx-y', where xxx is the specific event number and y is time in seconds that may be added or subtracted to the event time. The camera generally moves through space along with the vehicle. The user specifies where the camera is located with respect to the composite vehicle center of mass. The camera position and target points are input in the geographic frame or in the relative velocity frame. North, east, and down directions form the Cartesian X, Y, and Z axes in the geographic frame. In the velocity frame, X-axis is in direction of the relative velocity vector; Y-axis is in the local horizontal plane, and Z-axis completes the right-handed coordinate system. The camera viewing angle is also input in camera_properties.txt file. The camera viewing angle can be used to simulate the effect of zooming in and out. All the properties are input as a function of

trajectory time. The animation tool linearly interpolates in between discreet time intervals, so all camera movements appear smooth. In Figure C.7 is a sample of the camera_properties.txt file:

```
% Variables:
% CVA            : Camera viewing angle
% posi_cs        : camera position coordinate system
%                 ned -> Cameral location defined in NED frame with respect to composite CG
%                 fix -> Camera location stays fixed in ECR frame
%                 vel -> Camera location defined in relative velocity frame
% camera_position : Camera position, if posi_cs = fix camera_position is in NED frame
% targ_cs        : Camera target coordinate system
%                 ned -> Camera target defined in NED frame with respect to composite CG
%                 fix -> Camera target stays fixed in ECR frame
%                 vel -> Camera target defined in relative velocity frame
% camera_target   : Camera target coordinates in targ_cs coordinate system
% CFT            : Camera fixed time. Used only when posi_cs or targ_cs = fix
%
% time     CVA  posi_cs  <-  camera_position ->   targ_cs    <- camera_target ->        CFT
%-----------------------------------------------------------------------------------------------
  "0.0"    45    ned      0.0    -5.0    -20.0      ned      0.0      0.0     0.0      "0.0"
 "300.0"   45    ned      0.0    -5.0    -20.0      ned      0.0      0.0     0.0      "0.0"
 "320.0"   45    ned    -50.0     0.0    -15.0      ned      0.0      0.0     0.0      "0.0"
"1135.0"   45    vel    -10.0    10.0      0.0      vel      0.0      0.0     0.0      "0.0"
"1140.0"   45    vel    -50.0    10.0      0.0      vel      0.0      0.0     0.0      "0.0"
"1143.0"   45    vel    -50.0    10.0      0.0      vel      0.0      0.0     0.0      "0.0"
"1144.0"   10    vel    -50.0    10.0      0.0      vel      0.0      0.0     0.0      "0.0"
"1149.0"   10    vel    -50.0    10.0      0.0      vel      0.0      0.0     0.0      "0.0"
"1150.0"   45    vel     15.0    10.0      0.0      vel      0.0      0.0     0.0      "0.0"
"1155.0"   45    vel     15.0    10.0      0.0      vel      0.0      0.0     0.0      "0.0"
"1157.0"   45    vel    -50.0     0.0      0.0      vel      0.0      0.0     0.0      "0.0"
"1170.0"   45    vel    -50.0     0.0      0.0      vel      0.0      0.0     0.0      "0.0"
"ev300-2"  40    ned    -50.0     0.0    -30.0      vel    -10.0      0.0     0.0      "0.0"
"1177.0"   40    ned    -50.0     0.0    -30.0      vel    -17.0      0.0     0.0      "0.0"
"1178.0"   40    ned    -70.0     0.0     20.0      vel    -17.0      0.0     0.0      "0.0"
"1185.0"   40    ned    -70.0     0.0     20.0      vel    -17.0      0.0     0.0      "0.0"
 "ev850"    5    fix    -20.0     0.0      0.0      vel    -17.0      0.0     0.0  "ev850+6"
"1205.0"    5    fix    -20.0     0.0      0.0      vel    -17.0      0.0     0.0  "ev850+6"
"1206.0"   27    ned    -70.0     0.0    -90.0      vel    -25.0      0.0     0.0      "0.0"
"2000.0"   27    ned    -70.0     0.0    -90.0      vel    -25.0      0.0     0.0      "0.0"
```

Figure C.7. Sample camera_properties.txt file

In the camera position options (shown in Figure C.7 above), the camera location and target points are defined with respect to the composite center of mass. The animation tool also supports a fixed option for the camera where either the camera location or camera target or both remain fixed in space. This feature is activated by specifying the "fix" option in the camera_position and/or camera_target coordinate system flag. When this option is chosen, the system also looks up the Camera Fixed Time (CFT) field in the table. The CFT field is the time or event when the vehicle position is looked up as the fixed location for the camera. This option simulates the effect of the vehicle flying past the camera, with the camera tracking. Setting the position and target

coordinate systems to fix (columns 3 and 7) simulates the effects of a completely stationary camera.

Step 7: Run animation_driver

The animation tool is now ready to build all the data structures for generation of the movie. Double-click on the animation_driver executable to initiate the program. The animation_driver executable opens a disk operating system (DOS) window and goes through the POST2 input files, master_setup.txt, geom_data, camera_properties.txt, POST2 output file, and the POST2 Matfile to build up appropriate data structures. The movies are made in AVI format, and are compressed using Cinepak codec on the PCs. Figure C.8 is a sample frame taken from one of the Mars entry animations.

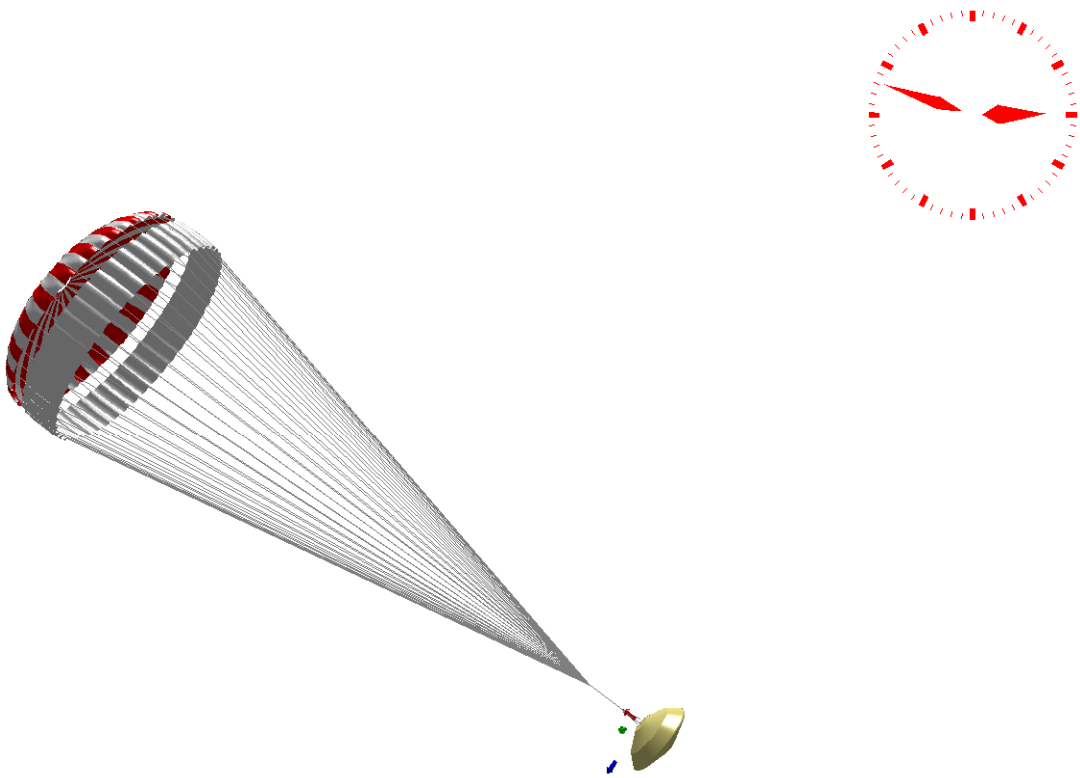

Figure C.8. A Sample Frame from Mars Entry Simulation

References

[C.1] Program to Optimize Simulated Trajectories: Volume II, Utilization Manual, prepared by: R.W. Powell, S.A. Striepe, P.N. Desai, P.V. Tartabini, E.M. Queen; NASA Langley Research Center, and by: G.L. Brauer, D.E. Cornick, D.W. Olson, F.M. Petersen, R. Stevenson, M.C. Engel, S.M. Marsh; Lockheed Martin Corporation, Version 1.1.1.G, May 2000.

[C.2] MATLAB™ Compiler User's Guide, Version 4. October 2004

Appendix D. Animation GUI v1.0
Written by Jeremy Shidner; Edited by Scott A. Striepe

I. Introduction

As demand for trajectory visualization has become more apparent, the needs of the EDL designer has grown substantially for animations. In the past, each animation would take excessive amounts of time to link the POST2 simulation to the animation tool (See Appendix C). By integrating specific components of the animation tool into a graphical user interface (GUI), the time needed to complete a trajectory animation, from POST2 data, can be reduced from weeks to a matter of hours. Specifically, input files are generated by the GUI and executed by the animation tool's driver.

II. Method

In addition to the POST2 files required by the animation tool, the GUI will build a structure of data in the MATLAB™ file handles.mat as well as create the required files by the animation tool, master_setup.txt, geom_data, and camera_properties.txt. The file, handles.mat, will hold all necessary inputs that are specific to the animation being developed (Figure D.1). Due to specialized inter-dependence between the handles.mat file and the files being used by the animation tool, saved settings cannot be used by other POST2 trajectory data. This limitation means that the same animation settings and files cannot be used for an alternate set of trajectory data and animation tool files. The animation data being used must remain in the local animation directory for correct operation.

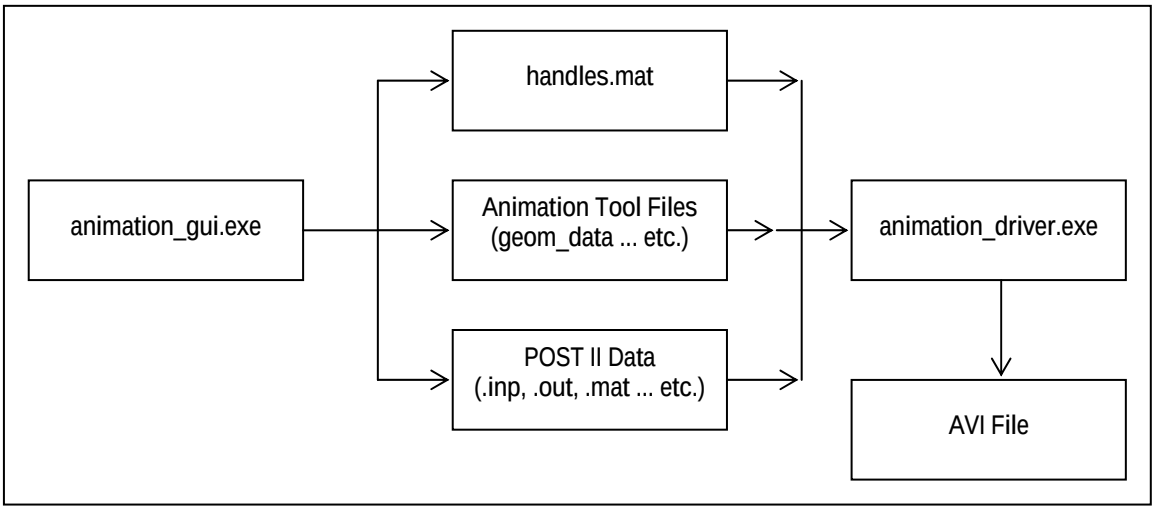

Figure D.1. Animation GUI Flow Diagram

III. Step-by-Step Instructions

This section outlines the procedure, step-by-step, to use the GUI to interface with the animation tool. The animation GUI has been developed in MATLAB™. All the source files have been compiled using the MATLAB™ Compiler. The compiled executable, named animation_gui.exe, is a stand-alone application and is used to interface with the animation tool.

Step 1: Installation

MATLAB™ Component Runtime (MCR) module needs to be installed on the host machine. A MATLAB™ license is not required for the MCR installation. The MCR installer utility program (MCRInstaller.exe) will be provided as part with the animation GUI. The MATLAB™ Component Runtime installation procedure is provided in MATLAB™ Compiler User's Guide [D.1], page 4-6. It is also available on the web. The instructions are provided for the Windows platform.

http://www.mathworks.com/access/helpdesk/help/toolbox/compiler/compiler_ug_collection.html

Windows Installation

The following components make up the animation GUI on Windows platforms.

MCRInstaller.exe Self-extracting MATLAB™ Component Runtime library utility; Platform-dependent file that must correspond to the end user's platform.

animation_gui.exe	Animation GUI executable
animation_gui.ctf	Component Technology File archive; Platform-dependent file that must correspond to the end user's platform.
animation_driver.exe	Animation tool executable
animation_driver.ctf	Component Technology File archive; Platform-dependent file that must correspond to the end user's platform.
data_dict.txt	Holds all search and assignment variables for the Animation application.
PlumeGeom.mat	Specialized rocket plume data file.
Geometries/	Directory containing archived vehicle geometry.

The animation GUI is provoked by double-clicking the animation_gui.exe file.

Step 2: Set Up Local Directory

The user needs to set up a local directory containing all the case specific files, such as the POST2 input file, output file, Matfile, and other associated case files. These files contain mission specific data such as trajectory and links to where specific files are stored. The following is a listing of local case files:

POSTfile.inp
POSTfile.out
POSTfile.mat
POSTrefile.dat

Once the case files are placed in the local directory, the animation GUI files listed in section 3.1.1 need to be placed in the local directory as well. MCRInstaller.exe does not need to be present in the local directory for correct operation. A sample setup can be found at the following directory:

/new_home/rais/matlab/animation_gui_local_dir/

For further description of the use of these case files, refer to Appendix C.

Step 3: Execute animation_gui.exe

Once the local directory has been set up, the animation GUI can be executed. An introduction window will be loaded first, as shown in Figure D.2.

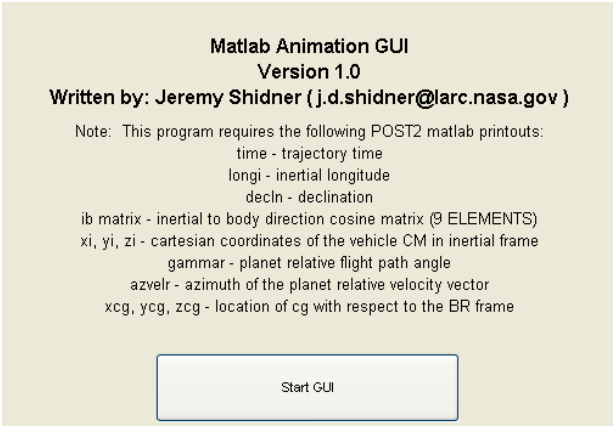

Figure D.2. Animation GUI Introduction Window

The introduction window notifies the user of the required inputs by the animation tool. The window is also the identifier for the version and contact information for GUI support. Processes run in the background initializing the handles.mat and usage variables for the GUI. Clicking the 'Start GUI' button initializes the main menu.

Step 4: GUI Main Menu

Before the main menu is shown, the GUI will detect the current animation progress. If previous work has been performed in the local directory, the relevant information will be loaded and displayed on the main menu shown in Figure D.3.

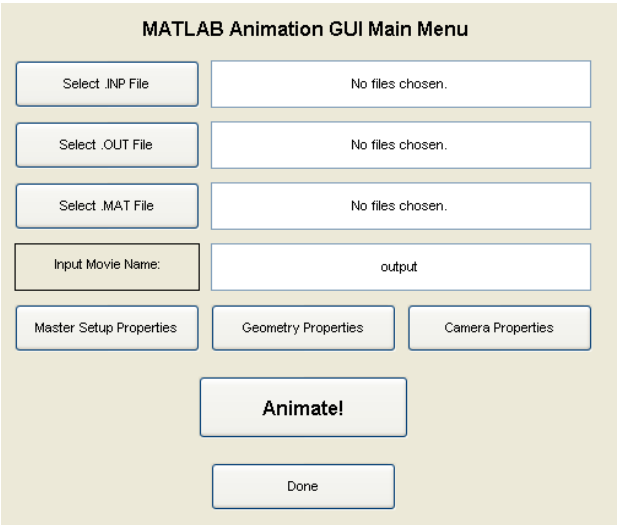

Figure D.3. GUI Main Menu

The main menu holds all selections for the required inputs. If any action that cannot be executed in the main menu is performed, text will appear next to the (*Animate!*) button notifying the user of the requirements for correct operation. The four inputs to the main menu are the POST2 input file, output file, MATLAB™ data file, and desired output movie name. The movie name should only be alpha-numeric characters, and does not require the '.avi' extension. To select the POST2 files for use, click the corresponding button, and select the file from the resultant pop-up window. The POST2 Matfile will require further refinement of the parameters once the file has been selected. The *Run Time Setup* window will appear as shown in Figure D.4.

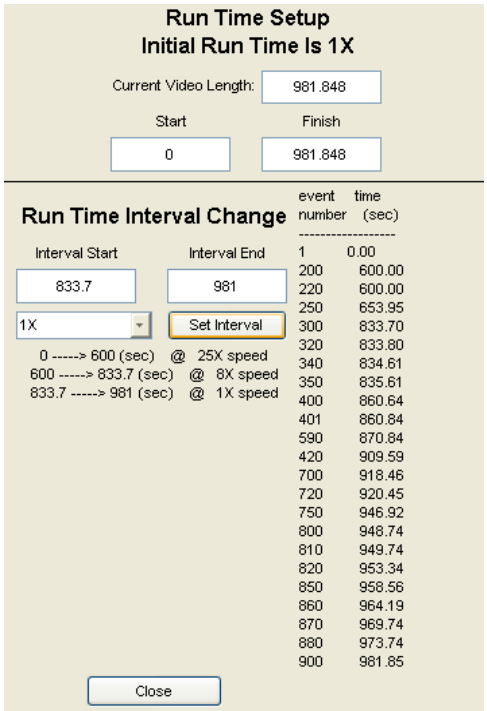

Figure D.4. POST2 Matfile Time Interval Sorting

The purpose of this window is to declare the speed at which the video will playback. *Current Video Length* is displayed at the top, and corresponds to the difference in time of the first and last indices of the time variable in the MATLAB™ data file. The times corresponding to those indices are displayed beneath the current video length and can be changed by entering new times in the corresponding text boxes. The *Run Time Interval Change* portion is dedicated to declaring the speed the video runs at. Event times are listed on the right, as taken from the corresponding POST2 '.out' file. This can be useful for determining phases of the trajectory when declaring run times. To set an interval, enter an interval start and interval end time in the corresponding text boxes, select a run speed in the drop down menu below the interval start box, and click *Set Interval*. Interval speed does not need to be selected for the whole video length; however, as the final sorting of video variables will default to 1X (real time). Once the intervals have been input, click *Close* to return to the main menu.

Step 5: Master Setup Properties

After the input variables have been selected in the GUI main menu, the next item to address is master setup properties. The master setup properties are broken into two divisions: master setup and variable display selection. The first window to be displayed is the master setup window.

Master Setup

The master setup window will read in current master setup properties stored in handles.mat and display the current version on the right side of the window. All properties that are adjustable are displayed on the left hand side as shown in Figure D.5.

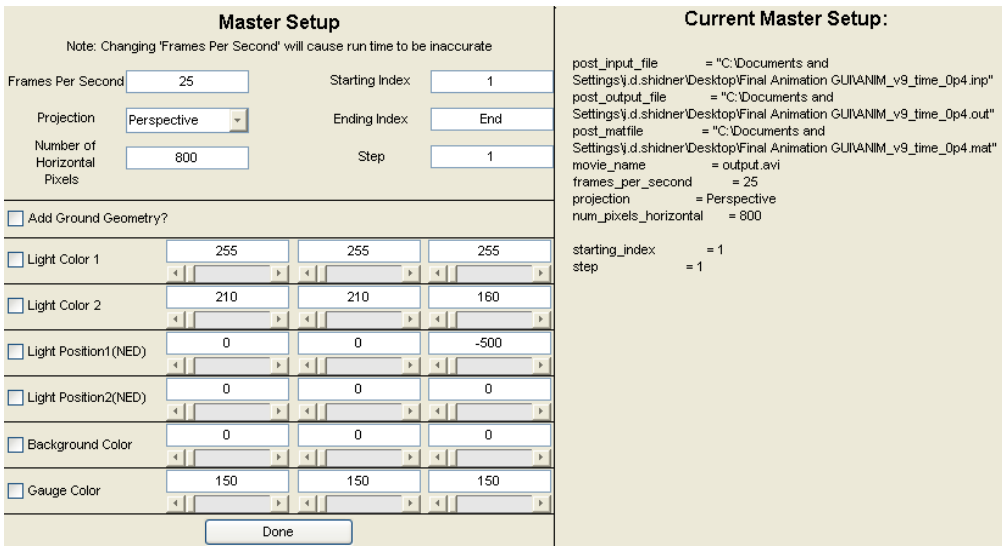

Figure D.5. Master Setup Window

The animation tool will default to the settings first displayed in master setup. For most cases, these defaults are satisfactory for the typical animation, and do not need to be changed. Though, if the user wishes, these settings can be changed to reduce the computation time or add different effects to the color and lighting of the animation.

First, note that changing the *frames per second* of the animation will affect the clock used to display time and the run time of the animation. It is recommended that the *frames per second* setting not be changed, but rather the *step* taken by the animation tool. *Step* corresponds to the indices of the animation. If a quick first cut of the animation is desired, one may set the *step* to a large number to capture less frames, but span the whole length of the animation. For example, if the POST2 MATLAB™ Matfile has 4,000 indices, one may set the step to 40, yielding 100 frames of animation. This would create a 4 second video that spans the entire length of the trajectory. If there is only a set range of indices desired for the animation, say 2,000 to 3,000, one may set the *starting index* and *ending index* to 2,000 and 3,000 respectively, such that only those indices will be animated.

If greater animating speed or smaller file size is desired, one may also set the *number of horizontal pixels* to a smaller number. The animation tool defaults to a 4 x 3 aspect ratio yielding a default setting of 800 x 600 pixels. This may be reduced or increased for greater definition.

Projection simply states the view taken by the camera as 'perspective' or 'orthographic'. What this means is that the 'perspective' camera will only capture as large a view as declared by the camera-viewing angle (CVA), while 'orthographic' will capture the entire view, not limited by the viewing angle.

The option to *add ground geometry* will display a pop-up menu asking for a ground geometry file. Ground geometry generation and definition is explained in section 3.4 of reference [1]. The ground geometry will always be a MATLAB™ Matfile, and must span the corresponding latitude and longitude ranges of the POST2 trajectory. This means that a ground geometry that worked for one animation will not necessarily work for another.

Color is dictated by red, green, and blue (RGB). Color can be set for four items: two light sources, background color, and gauge color. To change a color, click on the checkbox for the corresponding item, and move the slider bars left or right to the desired setting. *Light position* can also be adjusted by moving the slider bars. *Light position* is given in meters with respect to North, East, and Down of the spacecraft body. If the slider bars do not cover the desired light position range, one can manually input the light position by typing the desired value in the text box above the corresponding slider bar.

Once all the values have been dictated, click the *Done* button to close the master setup window and open the variable display selection window.

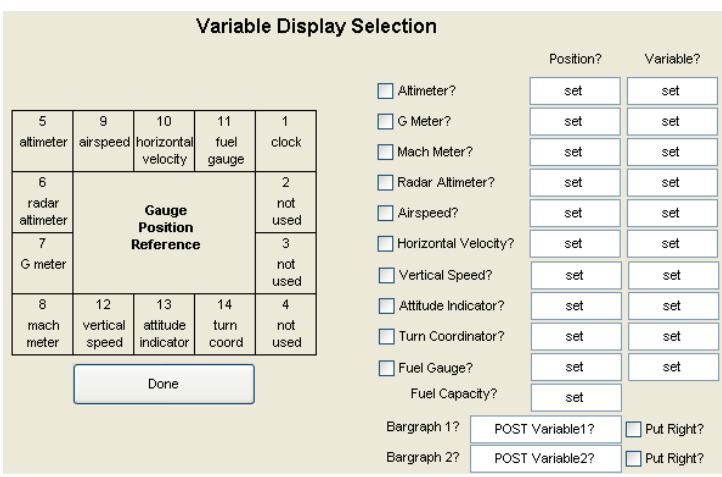

Figure D.6. Variable Display Window

Variable Display Selection

The variable display window will read in current variable display properties stored in handles.mat and display the current version on the right side of the window. All properties that are adjustable are displayed on the right hand side as shown in Figure D.6.

The variable display selection window allows the user to select which gauges are desired in the animation and assign a position and variable to the selected gauge. The left side of the window holds the gauge positions and defaults. Positions are declared by numbers, which are assigned to each position shown in the 'Gauge Position Reference'. Each gauge must also have a corresponding variable name input. This variable name is the name used in the POST2 MATLAB™ data file.

To select a gauge, click on the corresponding check box. When a gauge is selected, the corresponding default position and variable name will be assigned to the newly selected gauge. To change the position, enter the number of the desired position in the gauge's corresponding *Position* text box. To change the POST2 Matfile variable name, overwrite the default variable name in the gauge's corresponding *Variable* text box.

There are two special gauges in the variable display selection window. First, the *Fuel Gauge* requires an extra input for *Fuel Capacity*. This input is directly related to the POST2 output variable used for the propellant weight. The input fuel capacity will subtract off the value of the propellant weight from the total fuel capacity at every frame in the animation. Since the fuel gauge is displayed as kilograms in the animation, it is recommended that the POST2 Matfile propellant weight variable be in terms of kilograms as well.

Second, the option to display bar graphs is given by the GUI. This allows for any variable to be displayed in the animation screen. To display a bar graph, enter the POST2 Matfile variable name into the corresponding text box. The bar graph will display as a moving bar along the bottom of the screen. If other gauges have been placed on the bottom, overlap will occur, so it is advised to the user to not place gauges along the bottom row of the animation window. If the user desires, the bar graphs can be placed on the right side of the screen. This is accomplished by clicking the *Put Right* option.

Once the gauges have been selected for display, click *Done* to close the variable display selection window. For further explanation of the master setup and variable display properties, consult Appendix C.

Step 6: Geometry Properties

After the master setup properties have been selected, the next item to address is geometry properties. Geometry properties will declare the specific geometries used by the vehicles simulated in POST2. The geometry GUI is loaded by clicking *Geometry Properties* from the main menu window. The geometry data setup window that appears will read in current geometry properties stored in handles.mat and display the current version on the bottom left side of the GUI window. Event data will be read in from the POST2 '.out' file and displayed on the bottom right side of the GUI window. All properties that are adjustable are displayed on the top as shown in Figure D.7.

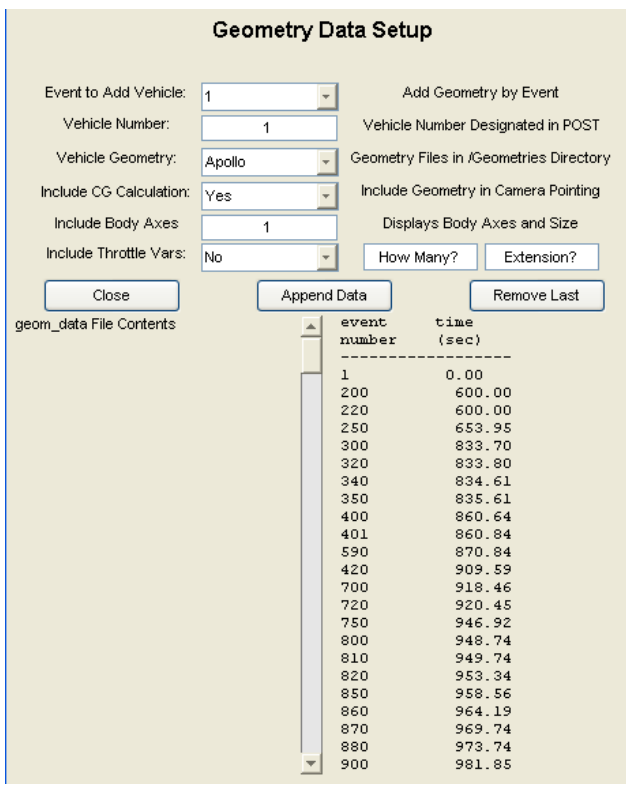

Figure D.7. Geometry Data Setup Window

The geometry properties are added by selecting the event, vehicle number, and geometry of the vehicle to then append to the geometry data file. Other features that are input define the representation of the vehicle in the animation.

The event listing will determine when the vehicle being appended is displayed in the animation.

Typically, this relates to such events as parachute openings or airbag deployment. The GUI will automatically read in the events from the POST2 '.out' file and load the events in the drop down menu next to *Event to Add Vehicle*.

The *Vehicle Number* corresponds to the vehicle number designated in the POST2 '.inp' file.

The *Vehicle Geometry* may be selected from five archived vehicles or a user-defined geometry. The five archived vehicles are Apollo, MER, MRO, Moonrise, and MSL derived capsules. These vehicle geometries are selected by clicking the drop-down menu, and selecting the corresponding vehicle. If a geometry file has been generated aside from the five archived vehicles, the user may select *Browse...* from the drop-down menu to open a search window. The user can then find the geometry file and use the corresponding file for the *Vehicle Geometry*. The vehicle selected will be displayed to the right of the drop-down *Vehicle Geometry* menu.

The option to *Include CG Calculation* is important for camera pointing properties. If selected, the camera target will include the center of gravity of the vehicle in each animation frame. If not selected, the camera will only focus on the vehicles with active cg's. This can be useful when peripheral vehicles are added to the primary vehicle in the animation (i.e.,– a parachute).

The *Include Body Axes* option is a scale representation of the body axes displayed in the animation. These axes are specifically taken from the coordinates used in drawing the vehicle geometry and are not representative of any other coordinate system. For vehicles that do not have specific body axes, this property is best set to zero to reduce confusion (i.e.,– airbags).

The *Include Throttle Vars* option will enable thruster firing visualization in the animation. The GUI will automatically use the POST2 Matfile throttle variable of '*gvrc*xx' where xx is the thruster number represented in the POST2 '.inp' file. The number of thrusters must be declared in the first box to the right of this option entitled (*How Many?*). If this option is invoked, the user will need to be sure that the thruster locations are either in the POST2 '.inp' file or in an included file that is present in the local directory. If an extension is present on the POST2 Matfile throttle variables, the extension must be declared in the second box to the right of this option entitled (*Extension?*).

Once the geometry properties have been selected for a vehicle, the data must be appended to the geometry data file. This is accomplished by clicking the *Append Data* button. The geometry data file contents will be displayed on the bottom left side of the window. If an incorrect geometry property is input, the *Remove Last* button will delete the last appended vehicle from the

geometry data file. Once the geometry properties have been input, clicking the *Close* button will close the window. For further explanation of geometry properties, consult Appendix C.

Step 7: Camera Properties

After the geometry properties have been selected, the next item to address is camera properties. Camera properties will declare the specific path the camera takes in the animation to view the geometry. The camera properties GUI are loaded by clicking *Camera Properties* from the main menu window. The camera properties setup window that appears will read in current camera properties stored in handles.mat and display the current version on the left and top right side of the GUI window. Event data will be read in from the POST2 '.out' file and displayed on the bottom right side of the GUI window. Any erroneous actions made in the camera properties GUI will be addressed by text that appears in the bottom right of the window, beneath the event readout. All properties that are adjustable are displayed on the left as shown in Figure D.8.

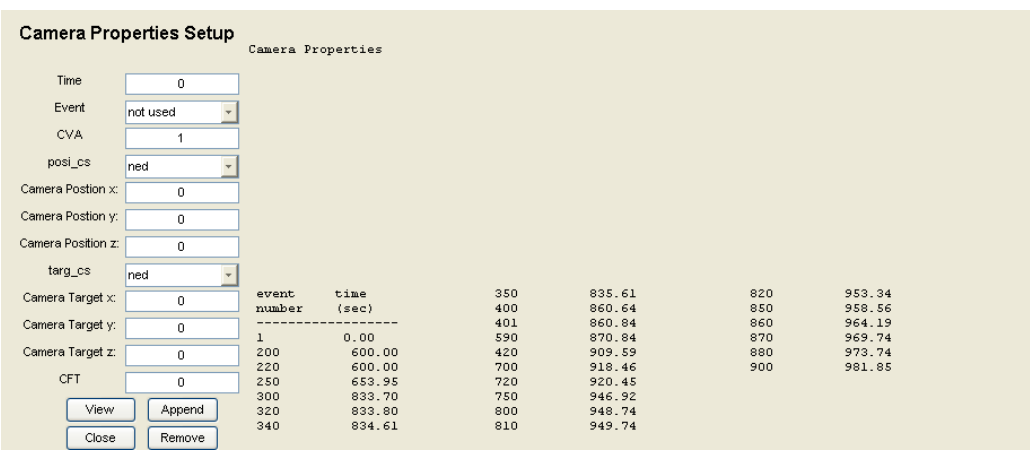

Figure D.8. Camera Properties Setup Window

The camera properties address five options, the camera time, camera-viewing angle (CVA), camera position, camera target, and camera fixed time (CFT). Each option is changed by either selecting from a drop-down menu, or by manual input. As in geometry properties, the user can either append or remove each set of camera properties desired in the camera properties data file. An option to view the camera setting is available by clicking *View*.

The *Time* option is related specifically to the trajectory time. The event times are listed on the bottom right of the window for reference. The user can also specify the time by *Event* by selecting from the drop down menu. For each time entry, the animation tool will interpolate the camera position between the previous times so that a seamless transition is made from position to position.

The *CVA* option declares the camera-viewing angle in degrees from the camera position. This has the effect of zooming in and out on the target vehicle.

The camera position is dictated by four options, *posi_cs, Camera Position x, Camera Position y, and Camera Position z*. The first option, *posi_cs*, declares the coordinate system of the camera position. The three remaining options declare the camera position with respect to the vehicle center of gravity. Position is measured in the units used to draw the vehicle geometries, typically meters.

The camera target is dictated by similar options, *targ_cs, Camera Target x, Camera Target y, and Target Position z*. The first option, *targ_cs*, declares the coordinate system of the camera target. The three remaining options declare the camera target position with respect to the vehicle center of gravity. Target is measured in the units used to draw the vehicle geometries, typically meters.

The final option, *CFT*, is the time or event when the vehicle position is looked up as the fixed location for the camera. The *CFT* option may be input using event times as well, but has to be input manually as '*ev*xxx' where xxx represents the event number.

Once the camera properties have been input, the user may view a frame of animation using their current camera properties. This is accomplished by clicking *View*. As data must now be processed, an actual frame can take on the order of minutes to actually appear. During this time, the user should refrain from pressing any other buttons in the GUI. Once the frame has appeared, the user may inspect the image and adjust the camera properties accordingly.

Once the camera properties have been selected, the data must be appended to the camera properties data file. This is accomplished by clicking the *Append* button. The camera properties data file contents will be displayed on the top right side of the window. If an incorrect camera property is input, the *Remove* button will delete the last appended camera setting from the camera properties data file. Once the camera properties have been input, clicking the *Close* button will close the window. For further explanation of camera properties, consult Appendix C.

Step 8: Animate!

When all the above steps have been completed, the animation GUI will have developed the necessary data files to provide the animation tool with a complete animation definition. These data files are master_setup.txt, geom_data, and camera_properties.txt. Refer to Appendix C for the structure and importance of these data files.

To generate the animation, click the (*Animate!*) button in the main menu window. The animation GUI will then pass the data files to the animation tool executable, and generate the movie. It is recommended the computer being used to animate the POST2 trajectory have at least 512 Mb of random access memory (RAM). If the number of frames being animated is too large, the animation tool will error because there will not be enough physical memory available. A good ratio to go by is for every 1500 frames of animation, there needs to be at least 256 Mb of RAM.

Figure D.9 shows a sample frame taken from a trajectory animation using the animation GUI.

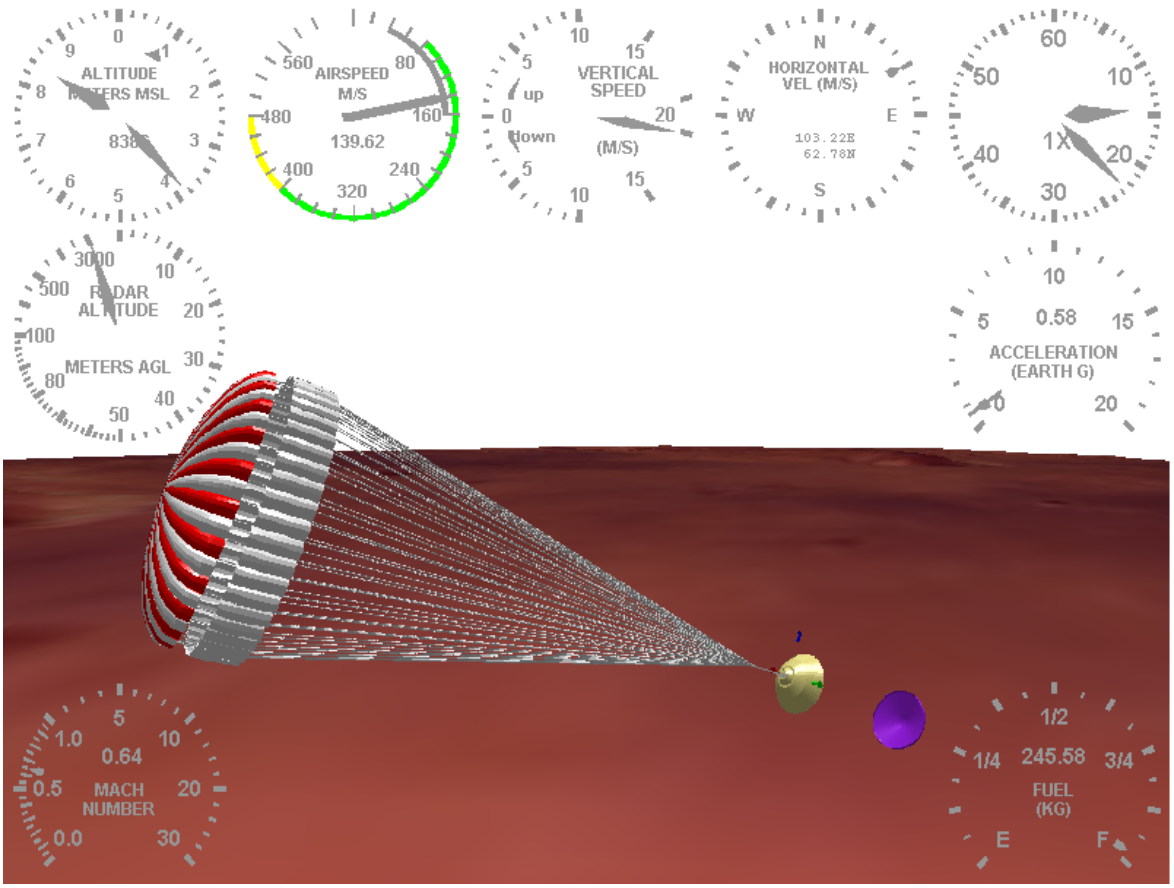

Figure D.9. Sample Animation Frame

References

[D.1] MATLAB™ Compiler User's Guide, Version 4. October 2004

Appendix E. POST2 Scripts Users' Guide
Written by Loreyna Yeung; Edited by Scott A. Striepe

I. Single Post Run

Files Location: /app/production/programs.c2/Single_Post_Run

Files Requirements:

1) run_s_post.pl: Driver script to generate PERL scripts and submit a single POST job to the queue.
2) s_post.tpl: Template to create s_post.pl.

Both files can be place in a common area such as the 'bin' directory as long as the user environment path is set correctly. The file run_s_post.pl can be modified to user specified POST executable path, name, and menu display selections.

How to Run:
Default: run_s_post.pl uses low priority queue (short).

Example: run default POST executable and driver script is in the bin directory.
run_s_post.pl filename.inp 0

(Note: input filename extension can be omitted)

Example: run menu mode to select POST executable and driver script is in the working directory.
./run_s_post.pl filename

Options:
-s To save outputs by creating a temporary area under user's working directory and place all the outputs after program run.
Example: run POST executable selection #2 + save + verbose
./run_s_post.pl filename 2 -vs

-l To run local by setting everything on the local node's /node/tmp directory and only brings back the output when it is done.
Example: default + local + verbose
./run_s_post.pl filename 0 -vl

-v To displays more information during program run.
Example: run POST executable selection #2 + verbose
./run_s_post.pl filename 2 -v

-q To sets user queue type to be run (Default queue type is Max_All_5min)
Example: run default POST executable + Max_N64_1hr + verbose
./run_s_post.pl filename 0 –vq Max_N64_1hr

-m To create Matlab file.

Monte Carlo

Input Generation

Files Location; /app/production/programs.c2/MPI_Monte_Carlo/

Files Requirements:
monte_inputs.pl - A script to generate the monte input file.
ovat_inputs.pl - A script to generate the ovat input file.
perl_make.pl - A script to generate a PBS script to run the above PERL script on the cluster.

How to Run

"./make" - will generate the milestone, monte_inputs.dat, ovat_inputs.dat and executes the monte_carlo run to generate the output_variables file.

./make <option> - will generate only the option you give it.

<options>:
milestone - creates the milein.mc
inputs - creates the monte_inputs.dat
ovat - creates the ovat_inputs.dat
monte - runs the monte_carlo and generates the output_variables file

Log file
q_input.log - logs the status of the run and errors if any.

Debug
Check the q_input.log first. This file will most likely tell you what went wrong while generating your inputs.

PERL Scripts

Files Location: /app/production/programs/MPI_Monte_Carlo

Files Requirement

qsub_mpi_nqueue.pl - driver script to generate PBS and PERL scripts
to submit Monte Carlo POST jobs to the queue.
qtemplate.tpl - contains all the templates to create PBS script,
to generate q_p2_x.pl for low priority (short or ALL_MAX_5min) queue,
to generate q_p2_x.pl for high priority (n64 or Max_N64_1hr) queue,
p2_qsub.pl - script to execute POST run (edit POST executable path)

How to Run

There are two ways to run these scripts:
1.
a) Copy qtemplate.tpl to the 'bin' directory and set to an appropriate environment path in the '.cshrc' file (don't forget to source the file). The driver script will always use qtemplate.tpl in the 'bin' directory unless it is in the working directory

b) Copy qsub_mpi_nqueue.pl either to the 'bin' directory, or the working directory or both. To run script is in the 'bin' directory, follow the same setup as above. To run script in the working directory, type:
"./qsub_mpi_nqueue.pl" at command prompt to run.

c) To setup system to look for the driver script in the working directory then the 'bin', environment path must is set correctly in '.cshrc' file. Follow example below.
Example: set path=(. ~ ~/bin $path $HOME /usr/local/bin)
Note: the '.' is current directory, and '~' is home directory.

d) Copy p2_qsub.pl to the working directory.
Make sure permission is set correctly.
For more submission examples, please see below.

2.
a) Copy qsub_mpi_nqueue.pl and qtemplate.tpl to the production area other than the 'bin' directory.

b) Setup path to the production area in qsub_mpi_nqueue.pl (instruction is in the script)

c) Create a project directory and copy p2_qsub.pl to this location.

d) Optional, create a link to the production area in the project directory.
Default: qsub_mpi_nqueue.pl uses both high priority queue (Max_N64_1hr or n64) and low priority queue (ALL_MAX_5min or short).

Example, default + verbose ./qsub_mpi_nqueue -v

Options

-j To use new job name instead of using default job name "q_post" job name can be written in file 'job.name'. If -j option is not given in the command line, the program will look for 'job.name'. If 'job.name' file doesn't exist, job name will go back to default 'q_post'

IMPORTANT NOTE: The specific name may be up to and including 15 characters in length. It must consist of printable, non-whitespace characters with the first character alphabetic, and contain no "special characters".

Example: run only on ALL_N64_1hr queue and automatic node selection with job name 'msl'

./qsub_mpi_nqueue -pajmsl

-c To set user's specified queue size on ALL_N64_1hr queue for all jobs

Example: use 25 node only

./qusb_mpi_nqueue -c25

-a Automatically select maximum available node on 64 node primary queue, and program will not prompt user for number of nodes input.
-f To run entire input cases on low priority queue, Max_All_5min.
-p To run entire input cases on the one hour time limit high priority queue.
-v To displays more information during program run
-s To save all the outputs to working directory
-t To set the job length for each case run and default is set to 30 seconds:

Example: rerun + joblength + low priority queue(short):

./qsub_mpi_nqueue -r1,2,5:10 -t15

-r To rerun specified input cases; user must separate each case with a comma and set the range using colon.

Example: rerun + low priority queue (short)
./qsub_mpi_nqueue -fr1,2,5:10

-q To run subset of cases; User must separate each case with comma and set the range using colon.
Example: verbose+ subset+ low priority queue (short)
./qsub_mpi_nqueue -vfq1:3,4

Test Suite

Files Location: /app/production/programs.c2/Test_Suite

File Requirements:

1) go_n_tpl: This is a template c-shell script that executes old and new POST.
2) tqsub_p2_tpl.pbs: A template PBS script.
3) tq_p2_tpl.pl: A template script used to run go_n_tpl.
4) mace: A shell script used to compare the old and the new post runs.
5) old_pst: Link the production POST to "old_pst".
6) new_pst: Link your POST to "new_pst".
7) test cases: Link /app/production/post2.testing/cluster/tests to "tests"
8) include: Link /app/production/post2.testing/cluster/include to "include"

All these files are needed in the same directory with the old and new post executables.

Usage:
No space after between the option and the parameter input "< >."

-v Verbose

Example: run_testsuite.pl -v

-l<set number> Subset of test suite, <set number> is the subset number you want to run.

Example: run_testsuite.pl -vaq1:3:6,100

-i<cluster number> Runs the test suite on specific cluster:

-i default submits job to eseb-fs2
-i1 submits job to "esebcluster"
-i2 submits job to "eseb-fs2"
-i3 submits job to "i3-head" Note: must submit from eseblab100 to 103, i3lab1 to i3lab4
-i4 submits job to "i4-head"

Example: run_testsuite.pl -vi3

-b Both new and old output, generates both new and old output for any inputs (default or project series).

-a Add all projects with series number greater or equal to 100 and runs projects series in addition to the default set of series. The location of the project series path needs to be specified either at the command line using the -c option, or the program will prompt for a path.

-p<output_path> output path for old POST output, specify the location of the old POST output path that will be use to compare with the new output for the default series (1 to 10). Absolute or relative path can be used.

-c<project_input_path> Project series input path; specify the location where the project series inputs are located. Absolute or relative path can be used.

Example: -cseries/101, the project series is located in the series/101 directory.

-o<project_output_path> Project series output path, specify the location of the project series output path. Absolute or relative path can be used.

-j<project_series_input_path> Just project series, generate output for project series only. The project series input location is required.

-s Single CPU mode, run job on a stand-alone single CPU mode

MACE
Files Location: /app/production/programs/Test_Suite

Note: The profile compare scripts were combined into just one file "mace", and new options were added to account for the new capabilities. By default, the script compares only series 1 to 10 unless user specifies other option.

Files requirements:

-PCNMLT
-profcomp
-rcompare_8
-rcompare_9

How to Run: no dash (-) nor comma (,) requires for using the options below.

J Just series, compare only project series.

ex. mace j

b Both default (1 to 10) and project series.

Example: mace b

<selected_series> no option requires, just enter each series number with a space.

Example: mace 2 3 9 100 (compare series 2, 3, 9, and 100)

* Check the files good.txt, bad.txt, exceed.txt, and report.txt for the compare in "outfiles/tests."

Delete PBS Jobs Utilities

annihilate.pl

Files Location: /app/production/programs/Utilities

Description: performs a qdel to a job and deletes the zombie jobs on each node that the job was running on and cleans up the /nodes/tmp area.

How to Run:

annihilate.pl <job id>, where job id is the id given by PBS
or

annihilate.pl <node numbers> where node numbers is the cluster nodes

Example: job ID: 149912.esebcluster
annihilate.pl 149912

Example: to clean node number 2, 4, 6, and 7
annihilate.pl 2,4,6,7

Example: to clean node number from 5 to 10
annihilate.pl 5:10

eradicate.pl
Files Location: /app/production/programs/Utilities

Description: Cleans up the '/nodes/tmp and /tmp' area by determine what files or directories you own and remove it from the area.

How to Run:
eradicate.pl<user's id>