Implementation of a Low-Thrust Trajectory Optimization Algorithm for Preliminary Design

Jon A. Sims,[*] Paul A. Finlayson,[†] Edward A. Rinderle,[‡] Matthew A. Vavrina,[§] and Theresa D. Kowalkowski[**]
Jet Propulsion Laboratory, Pasadena, California, 91109-8099

A tool developed for the preliminary design of low-thrust trajectories is described. The trajectory is discretized into segments and a nonlinear programming method is used for optimization. The tool is easy to use, has robust convergence, and can handle many intermediate encounters. In addition, the tool has a wide variety of features, including several options for objective function and different low-thrust propulsion models (e.g., solar electric propulsion, nuclear electric propulsion, and solar sail). High-thrust, impulsive trajectories can also be optimized.

I. Introduction

FOR interplanetary missions, highly efficient electric propulsion systems can be used to increase the mass delivered to the destination and/or reduce the trip time over typical chemical propulsion systems.[1,2] This technology has been demonstrated on the Deep Space 1 mission[3] – part of NASA's New Millennium Program validating technologies that can lower the cost and risk and enhance the performance of future missions. With the successful demonstration on Deep Space 1, future missions can consider electric propulsion as a viable propulsion option. In fact, Dawn, a Discovery Program mission, plans to use electric propulsion to transfer to and between orbits at the two most massive asteroids, Vesta and Ceres.[4]

Electric propulsion systems, while highly efficient, produce only a small amount of thrust. As a result, the engines operate during a significant fraction of the trajectory. This characteristic makes it much more difficult to find optimal trajectories. The methods for optimizing low-thrust trajectories are typically categorized as either indirect or direct. Indirect methods are based on calculus of variations, resulting in a two-point boundary value problem that is solved by satisfying terminal constraints and targeting conditions.[5] These methods are subject to extreme sensitivity to the initial guess of the variables – some of which are not physically intuitive. Adding a gravity assist to the trajectory compounds the sensitivity. Direct methods parameterize the problem and use nonlinear programming techniques to optimize an objective function by adjusting a set of variables. A variety of methods of this type have been examined with varying results.[6-9] These methods are subject to the limitations of the nonlinear programming techniques.

In this paper we describe the implementation of a direct method intended to be used primarily for preliminary design of low-thrust interplanetary trajectories, including those with multiple gravity assists. Preliminary design implies a willingness to accept limited accuracy to achieve an efficient algorithm that executes quickly. The basic approach and testing of a prototype was described by Sims and Flanagan.[10] The underlying structure in the current implementation remains the same, although a few of the optimization variables have changed slightly. Many new features and functionalities have been added, and a graphical user interface has been developed to assist users in creating starting guesses and visualizing and analyzing results.

[*] Senior Member of Engineering Staff, Guidance, Navigation, and Control Section, 4800 Oak Grove Drive, Mail Stop 301-140L, Senior Member AIAA.
[†] Senior Member of Engineering Staff, Guidance, Navigation, and Control Section, 4800 Oak Grove Drive, Mail Stop 301-140L.
[‡] Senior Member of Engineering Staff, Guidance, Navigation, and Control Section, 4800 Oak Grove Drive, Mail Stop 301-140L.
[§] Associate Member of Engineering Staff, Guidance, Navigation, and Control Section, 4800 Oak Grove Drive, Mail Stop 301-140L.
[**] Member of Engineering Staff, Guidance, Navigation, and Control Section, 4800 Oak Grove Drive, Mail Stop 301-150, Member AIAA.

II. Approach

A. Trajectory Structure

The trajectory is divided into legs that begin and end at *control nodes* (Fig. 1). Typically, the control nodes are associated with planets or small bodies, but they can be free points in space (i.e., non-body control nodes). On each leg is a single *match point*, and the trajectory is propagated forward in time from the leg's earlier control node to the match point and backward from the leg's later control node to the match point. This type of multiple shooting technique with intermediate control nodes is very similar to that implemented in the program CATO.[11] This technique significantly reduces the sensitivity of the propagation to intermediate flybys compared to propagating strictly forward in time from the beginning of the trajectory to the end. Several optimization variables are available at each control node, providing considerable and direct control over the trajectory when needed.

Continuous thrusting is modeled as a series of impulses. The legs are subdivided into *segments* with an impulsive ΔV in the middle of each segment. When modeling low-thrust propulsion systems, the magnitude of the impulse is limited by the amount of ΔV that could be accumulated over the duration of the segment.

The propagation between control nodes, impulses, and match points is according to a two-body model with respect to a primary body. Flybys of secondary bodies are modeled as instantaneous changes in the direction of the $\mathbf{V}_\infty$ (relative velocity vector). For interplanetary trajectories, the Sun is the primary body and planets (or small bodies such as asteroids and comets) are the secondary bodies. We can also model planetocentric trajectories with a planet as the primary body and the moons of the planet as secondary bodies.

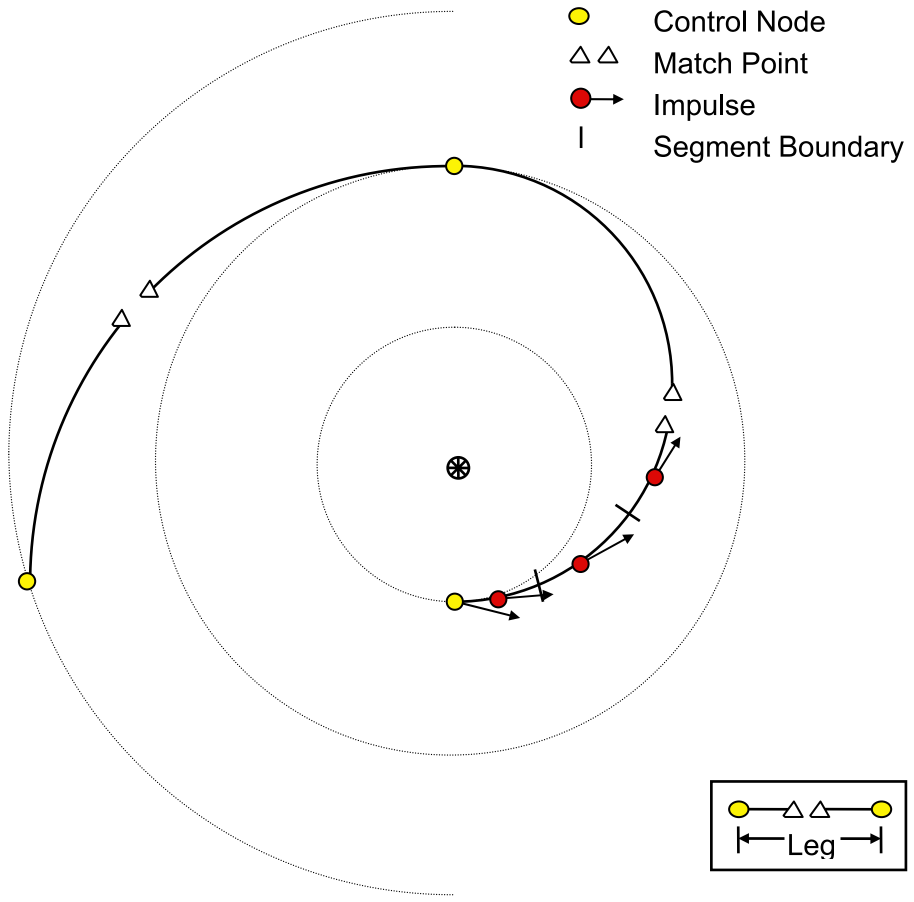


Figure 1. Trajectory Structure.

A feasible trajectory is continuous in position, velocity, and mass at the match point on each leg. The trajectory depicted in Fig. 1 is in an intermediate, unconverged state with noticeable position discontinuities at the match points.

An encounter with a body represented by a control point can be a flyby or a rendezvous. For a rendezvous, the trajectory will match both the position and velocity of the body at the time of arrival. The mass of the arrival body is not modeled in this rendezvous case. For a flyby, the trajectory will match the position of the body at the time of arrival but not necessarily the velocity. For a flyby of an intermediate body, the effect of the mass of the body is modeled by a rotation of the $\mathbf{V}_\infty$, and the flyby altitude is determined based on the mass.

B. Optimization

The trajectory structure described above leads to a constrained, nonlinear optimization problem which we solve using the nonlinear programming software SNOPT.[12]

1. Variables

The independent variables at each control node associated with a body, such as a planet, are the velocity of the spacecraft relative to the control node, the mass of the spacecraft, and the corresponding epoch. At an intermediate control node, the variables are, in general, different upon arrival at the control node compared to departure from the control node. For example, during a flyby, the relative velocity vector changes. For an intermediate body rendezvous, the departure time will be different than the arrival time. The mass of the spacecraft can also change due to a mass drop such as a probe release, for example. For these reasons, each intermediate control node has two sets of variables: one at arrival and one at departure. For the initial control node, only the departure variables apply, and for the final control node, only the arrival variables apply. The independent variables at a non-body control node are the spacecraft mass, position, and velocity relative to the primary, and the corresponding epoch. Again, intermediate non-body control nodes have two sets of variables.

The other major set of independent variables is the components of the thrust vector on each segment. These thrust vector variables represent the vast majority of independent variables. By discretizing the thrust profile as a function of time, we have converted the infinite-dimensional continuously varying thrust problem into a finite-dimensional problem, but the number of these variables still drives the overall scale of the problem and makes it numerically challenging.

Other potential independent variables are the reference power of the spacecraft and the specific impulse of the thrusters in the case that the specific impulse is constant throughout the trajectory.

2. Bounds and Constraints

Upper and lower bounds are placed on all of the independent variables. In addition, constraints can be placed on functions of these independent variables. The value of each constraint function must lie between an upper and lower limit. A primary constraint on the optimization is that the position, velocity, and mass of the spacecraft must be continuous at the match points, i.e., the state from the forward propagation to a match point must be the same as that from the backward propagation to that point. The magnitude of the thrust on each segment is constrained due to the limited power available for thrusting. For the initial control node, the mass can be constrained to lie on or below a specified launch vehicle performance curve, and the magnitude and declination of the departure $\mathbf{V}_\infty$ can be constrained. At any rendezvous body, the magnitude of the $\mathbf{V}_\infty$ is constrained to be zero. At an intermediate body flyby, the magnitude of the arrival $\mathbf{V}_\infty$ and the flyby radius can be constrained, and the magnitude of the departure $\mathbf{V}_\infty$ is constrained to be equal to that of the arrival $\mathbf{V}_\infty$.

Constraints can be placed on flight time and propellant mass between any two control nodes, and the minimum distance to the Sun can also be constrained.

3. Cost functions

The following optimizations can be performed:

1. Maximize the final net spacecraft mass (final spacecraft mass – propulsion system mass)
2. Minimize the initial spacecraft mass (subject to a lower bound on the final spacecraft mass)
3. Minimize the total trip time
4. Optimize a weighted combination of the final net spacecraft mass and the total trip time
5. Maximize a function corresponding to the change in the Earth miss distance after impacting a small body. This is the cost function that was used in the 1st Global Trajectory Optimisation Competition sponsored by the Advanced Concepts Team of the European Space Agency.[††]

[††] http://www.esa.int/gsp/ACT/mission_analysis/globaloptimisationcompetition.htm

4. Optimization

The optimizer adjusts the independent variables to satisfy all of the bounds and constraints while simultaneously optimizing the cost function. If the optimizer can find a set of variables that satisfies the bounds and constraints, the trajectory defined by those variables is said to be feasible. If, in addition, the optimizer succeeds in locally optimizing the selected cost function, then the trajectory is said to be optimal and the optimization problem has been solved. There may be many locally optimal solutions within the space defined by the set of variables and the corresponding bounds and constraints. The solution actually found depends to a large extent on the initial guess provided for the variables.

During the optimization process, the program computes not only the values of all constraints and the cost function, but it also computes the partial derivatives of all constraints and the cost function with respect to each variable. The derivatives are formulated analytically (not by finite differencing), and their computation represents the bulk of the effort and code. The optimizer uses these derivatives to refine iteratively the values of the variables. We have found analytic derivatives to be a key component leading to fast execution and robust convergence.

III. Program Components

The trajectory structure and optimization described above has been implemented in a program called MALTO (Mission Analysis Low Thrust Optimization). The MALTO software set consists of four major components: Core (the computational engine), Executive (interface between user and Core), GUI (graphical user interface), and LowPost (post-processing application). The Core, Executive, and LowPost are written in Fortran 95. The GUI is written in MATLAB. Each of these components are described in more detail below.

A. Core

The Core is the primary computational engine for MALTO. It interfaces directly with the optimizer. For a given set of variables provided by the optimizer, the Core computes the values of the constraint and objective functions and the partial derivatives of all constraints and the cost function with respect to each variable, and returns these to the optimizer. These computations require the propagation of the trajectory as described above.

1. Overview

The variables defining a trajectory (described in Section II B) are represented by the independent variables vector **x**. Given an initial guess for **x**, it is the Core's task, combined with the optimizer SNOPT, to find a value of **x** that represents a feasible and locally optimal solution to the trajectory optimization problem.

The independent variables contained in **x** are constrained to be within upper and lower bounds:

$$\mathbf{x}_{LB} \le \mathbf{x} \le \mathbf{x}_{UB} \qquad (1)$$

where $\mathbf{x}_{LB}$ and $\mathbf{x}_{UB}$ are vectors of upper and lower bounds for the components of **x** and must be provided as inputs to the Core. For a given **x**, Core routines compute the trajectory constraint function vector **F**(**x**). The components of **F** represent trajectory values that must be constrained for the trajectory to be feasible. For example, contained within **F** (i.e., as components of **F**) are the position, velocity and mass mismatches at the trajectory match points (these are constrained to be zero to enforce position, velocity and mass continuity along the trajectory). Other components of **F** represent the vector magnitude of the Δ**V**s for each thrust segment (these are constrained to be less than or equal to the maximum ΔV attainable by the low thrust acceleration over the segment duration). These are only two of many constraint functions contained within **F**. The upper and lower limits that constrain the values of **F**(**x**) are inputs to the Core and represented by $\mathbf{F}_{LL}$ and $\mathbf{F}_{UL}$:

$$\mathbf{F}_{LL} \le \mathbf{F}(\mathbf{x}) \le \mathbf{F}_{UL} \qquad (2)$$

If Eq.1 and Eq. 2 are satisfied, the trajectory is said to be feasible. Beyond feasibility, the other goal of the optimizer is to make the trajectory optimal, which is done by maximizing some property of the trajectory. For example, the goal may be to maximize the final spacecraft mass, or to minimize the flight time (this is achieved by maximizing the negative of the flight time). The property to be maximized is computed by the cost function C(**x**), which is a scalar and is also a function of the independent variables **x**.

$$C(\mathbf{x}) = \text{maximum} \qquad (3)$$

It is the job of the optimizer to maximize the cost function C(**x**) while simultaneously satisfying the constraints of the problem (Eqs. 1 and 2). To achieve this, the optimizer iteratively adjusts the components of **x** taking steps toward an optimal solution. For each step, new values for **F**(**x**) and C(**x**) are computed. In addition to computing the values **F**(**x**) and C(**x**), the Core also computes the partial derivatives d**F**/d**x** (the Jacobian of **F**) and dC/d**x** (the gradient of C) which are required by the optimizer to determine the direction and magnitude for the next step. These derivatives are computed analytically (rather than using a finite differencing scheme). Robust convergence depends upon the quality of these derivatives and much of the computational effort of the Core is dedicated to their accurate and efficient computation. The Core (and the optimizer) also exploit the fact that the Jacobian d**F**/d**x** is often very sparse (i.e., most of the elements of d**F**/d**x** are zero). Exploiting the sparsity of the Jacobian greatly enhances optimization performance.

If the constraints are satisfied (Eq. 1 and Eq. 2) and the cost is locally maximized (Eq. 3), then the optimization problem is solved and a *local* solution has been found. There may be many local solutions. The solution actually found is often determined by the starting guess used. This makes the selection of an appropriate starting guess an important part of the solution process.

2. Computation of Constraint Functions and Cost

The main task of the Core software is the computation of the constraint function vector **F**(**x**) and the scalar cost function C(**x**) as well as their partial derivatives d**F**/d**x** and dC/d**x**.

The components of **F** that correspond to the position, velocity and mass mismatches at the match points are computed by propagating the trajectory forward in time from the initial control node of a given leg to the match point time, and backward in time from the final control node of a given leg to the match point time. The match point mismatch is simply the difference between the forward propagation values and the corresponding backward propagation values:

$$\mathbf{F}_{mismatch} = \begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \\ m \end{bmatrix}_{forward} - \begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \\ m \end{bmatrix}_{back} \tag{4}$$

As shown in Fig. 1, the trajectory is propagated from the control node to the match point as a sequence of two-body (conic) coasts separated by impulsive ΔVs representing the low thrust acceleration over a thrust segment. The initial conditions (position, velocity and mass) for a given propagation from control node to match point are determined from the independent variables in **x**. The components of the thrust segment ΔVs are also components of **x** and are constrained by available low thrust acceleration at any particular point in the trajectory. If the propulsion is solar electric, the available electrical power for the low thrust engine is a function of the distance from the Sun and is computed by the Core solar power model. The user can choose from built-in models or define his own. For nuclear electric, the available electrical power is a user-supplied constant determined by the reactor. For either case, power requirements for spacecraft systems can be modeled and subtracted from the power available for thrust. The maximum low thrust force and propellant mass flow rate are determined as functions of available electrical power by the engine model. The Core has several built-in engine models and also allows the user to specify a custom engine model. The maximum thrust force as determined by the engine model is used to compute the maximum allowed ΔV for a given thrust segment (which becomes part of $\mathbf{F}_{UL}$ in Eq. 2). The propellant mass flow rate is used to decrement the spacecraft mass to account for fuel used during the segment.

The Core also allows modeling solar sail trajectories. In this case, the thrust vector components for each segment represent the sail orientation vector (a unit vector perpendicular to the sail surface). For this case, the sail force is a function of the distance from the Sun and the sail orientation (as defined by the sail normal components for a particular segment). For solar sail, the mass flow rate is zero and spacecraft mass remains constant.

A model for launch vehicle performance computes the maximum mass a particular launch vehicle can deliver to a specified C_3. The user can select from a list of built-in launch vehicles or define a new one. A spiral capture/escape model allows the spacecraft to either be captured to or escape from a circular orbit around a body using a low thrust spiral. The fuel and time required for the spiral capture/escape are computed using an approximate analytic model. The Core also allows for an impulsive capture or escape to or from a specified elliptical orbit around a body.

B. Executive

The Executive processes user inputs into a form appropriate for the Core, orchestrates the running of the other elements of the MALTO set, and processes and outputs the results from an optimization run.

There are a variety of input files for MALTO, two of which are required, several others are optional. The primary input file is the Save Inputs file. Actually, it is both an input and an output file. As an input file, it specifies the trajectory path and types of encounters, and contains the relevant model parameters (e.g., thrusters, power, launch vehicle), initial guesses and bounds for all variables, and constraints on the trajectory. It is typically generated through use of the graphical user interface (described below). As an output file, the Save Inputs file includes all of those items except that the values of the optimization variables are no longer initial guesses but instead are the final values from the MALTO run.

The other required input file is the SNOPT Specifications file. This file allows users to change the default values used by SNOPT during the optimization process. Typical options include the maximum number of major, minor, and total iterations, the function precision, and the feasibility and optimality tolerances.

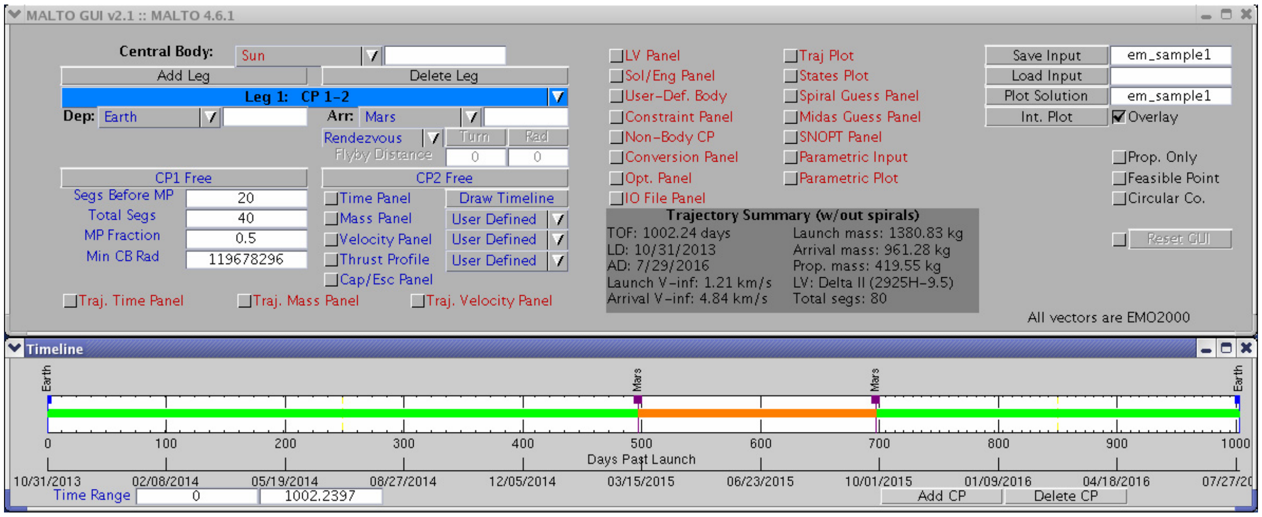
Four optional input files that can be used to customize the MALTO run are the Parametric Input file, the Models file, the Checkpoint file, and the MASL Data file. MALTO has an automated parametric capability for doing broad trade studies, and the Parametric Input file is used to specify the values of the parameters to search over and the names of the variables to include in the output file. The Models file can be used to provide MALTO with custom launch vehicle, solar array, and thruster data. The Checkpoint file is generated from a previous MALTO run and includes the values of the optimization variables at intermediate points in the optimization run, allowing the optimization to be restarted from one of those points. The MASL Data file allows users to change the default body data (e.g., mass). Ephemeris files for planets, moons, asteroids, and comets can also be used as input.

MALTO produces several standard output files. The Save Inputs and Checkpoint files were described above as input files. The Summary file lists the final values of optimization variables and the cost function and provides a summary of the converged trajectory, including control point encounters and thruster toggling. In addition, the Summary file contains a copy of the SNOPT Specifications file as well as the SNOPT exit condition. SNOPT also writes its own output file providing the details of the optimization itself and information for each major iteration. The Spacecraft Ephemeris file is a binary file containing the state (position and velocity) of the spacecraft as a function of time that is used by many mission analysis software tools. The Low-Thrust file is a binary file of information used by the post-processor LowPost. Both the Spacecraft Ephemeris file and the Low-Thrust file are used by the GUI for plotting the trajectory. The GUI also uses the Events file, which is a list of key events, event types, and durations. The Intermediate Plotting file is used primarily for plotting the trajectory at intermediate iterates during the optimization process. For parametric runs, a specified list of parameters is written after each step in the run to a Parametric Output file.

C. Graphical User Interface

The MALTO graphical-user interface (GUI) is run through MATLAB and allows users to graphically set up new input files, modify existing input files, and analyze MALTO results. The GUI features many drop-down menus, buttons that launch input panels, and direct input fields, as well as a summary of the current trajectory. The GUI also contains a graphical representation of the trajectory timeline that, in addition to providing a useful visualization of the entire trajectory and important trajectory events, allows direct access to and manipulation of many of the trajectory parameters (Fig. 2).

A key benefit of using the GUI to set-up new MALTO input files is that the MATLAB environment enables the use of formulas and functions in the input fields, which decrease the time to set up a problem and reduce user input errors. Furthermore, the GUI provides tools for analyzing MALTO outputs via a trajectory summary box in the main panel (Fig. 2), trajectory plotting (example in Fig. 3), and plots of various trajectory quantities, such as thrust or orbital elements (example in Fig. 4). Since the MALTO output file has the same format as an input file, a user can load a MALTO result into the GUI, modify it to solve a somewhat different problem, and create a new MALTO input file for the new case.



Central Body: Sun

Leg 1: CP 1-2

Dep: Earth

Arr: Mars

Rendezvous

Flyby Distance 0 0

Turn Rad

CP1 Free

CP2 Free

Segs Before MP 20

Total Segs 40

MP Fraction 0.5

Min CB Rad 119678296

User Defined

User Defined

User Defined

Save Input em_sample1

Plot Solution em_sample1

[✓ Overlay]

Trajectory Summary (w/out spirals)

```
TOF: 1002.24 days            Launch mass: 1380.83 kg
LD: 10/31/2013               Arrival mass: 961.28 kg
AD: 7/29/2016                Prop. mass: 419.55 kg
Launch V-inf: 1.21 km/s      LV: Delta II (2925H-9.5)
Arrival V-inf: 4.84 km/s     Total segs: 80
```

All vectors are EMO2000

Time Range 0 1002.2397

Figure 2. Main MALTO GUI panel and timeline.

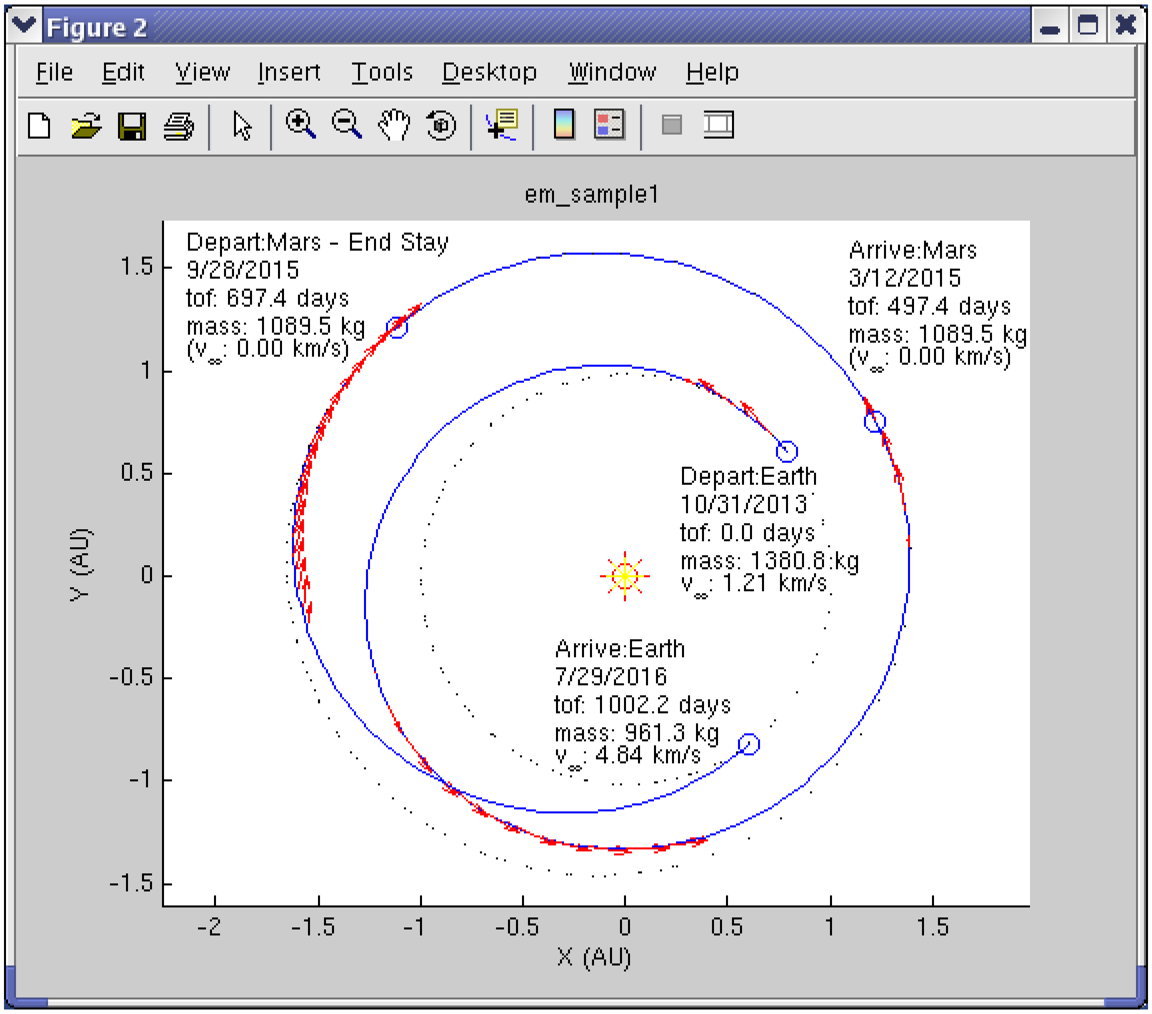


Figure 3. Example trajectory plot from MALTO GUI of a hypothetical Mars sample return mission; arrows indicate the thrust direction.

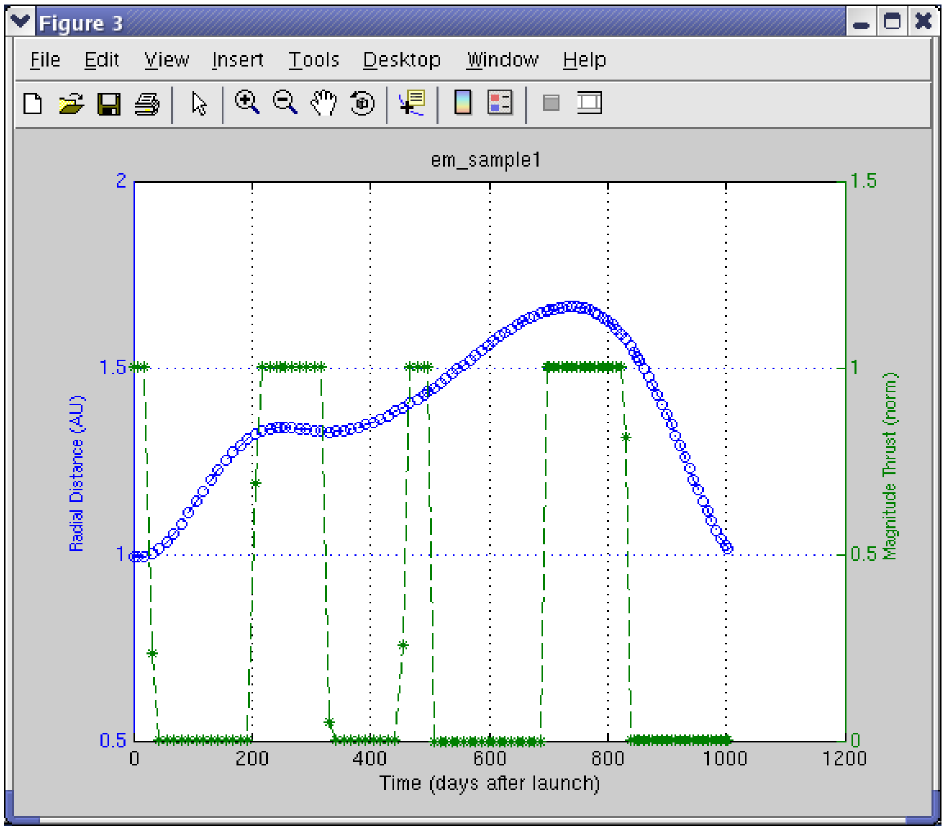


Figure 4. Example output data plot from MALTO GUI of the distance from the Sun and normalized thrust for a hypothetical Mars sample return mission.

The GUI facilitates the generation of an initial guess via two different guess tools: the MIDAS guess tool and the spiral guess tool. MIDAS[13] is a fast-converging patched conic trajectory optimization program based on primer vector theory. The MIDAS guess tool parses MIDAS detailed output files and spreads out the ΔV of any deep space maneuvers over a longer time period to produce an initial guess for the thrust profile. The thrust profile, along with the event dates and V_∞ values from MIDAS, provide MALTO with a low-thrust analogue to an impulsive solution and have shown to be useful starting points in the design process. The second tool, the spiral guess tool, also generates a thrust profile that may be used as an initial guess. The spiral guess tool assumes thrust along or against the velocity vector on each segment. While this is a very simple control law, it is a powerful tool particularly for building initial guesses with multiple revolutions about the Sun.

Other plotting features of the MALTO GUI include intermediate trajectory plotting, parametric output plotting, and 3-D plotting. The intermediate plotting feature allows users to quickly create simple trajectory plots while MALTO is running. Overlaying with the trajectory plots of previous iterations, users can evaluate the progress of a case and determine whether MALTO is converging to the desired type of solution. The parametric output plotting capability takes a tabular parametric output file and offers up each value in that file for users to select from in creating a plot. This is a quick way to gauge trends and overall behaviors exhibited in a parametric analysis. Finally, all of the available plotting features, with the exception of intermediate plotting, can be plotted in three dimensions. In fact, the normal trajectory plots (such as the one in Fig. 3) are always plotted in three dimensions.

D. LowPost

LowPost processes data from a MALTO run, generating a table of user-specified trajectory data as a function of flight time. The inputs for LowPost are the Low-Thrust file, the Spacecraft Ephemeris file, and user inputs describing the desired output. LowPost writes an ASCII file containing a table of values (up to 99 columns) of the parameters requested. LowPost runs in either interactive or batch mode. In interactive mode, it solicits the user for the inputs it needs. In batch mode, the user sets up a file of input data. The type of data that is generated includes dates, flight times, mass, thrust and power information, and angles and ranges between bodies.

IV. Testing and Applications

MALTO has been subjected to extensive testing and comparison to other low-thrust trajectory optimization tools. In Ref. 10, the initial prototype for MALTO was compared to SEPTOP, a low-thrust trajectory optimization program using an indirect method. (SEPTOP was used in Ref. 5 and is briefly described there.) SEPTOP is the result of a long evolution of low-thrust trajectory optimization software and has been used extensively to design a variety of missions. The trajectory results from the MALTO prototype compared very well to those from SEPTOP with final mass differences typically within less than 0.1 kg. MALTO itself has shown much less convergence sensitivity than SEPTOP and can handle many more intermediate flybys. In fact, MALTO has been used to optimize a trajectory with a dozen intermediate massive body flybys.

MALTO has already been used for a variety of applications, including mission proposals, advanced concept studies, and several mission studies for the Prometheus Program, especially the Jupiter Icy Moons Orbiter mission.[14-17] MALTO also played a key role in JPL's winning entry for the Global Trajectory Optimization Competition.[18]

V. Conclusion

We have developed, implemented, and tested a direct method for preliminary design of low-thrust interplanetary trajectories. The resulting tool is easy to use, has robust convergence, executes quickly, and can handle many intermediate encounters. It also has an extensive set of features that make it applicable to a variety of real world mission design activities.

Acknowledgment

The research described in this publication was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

[1]Williams, Steven N., and Coverstone-Carroll, Victoria, "Benefits of Solar Electric Propulsion for the Next Generation of Planetary Exploration Missions," *Journal of the Astronautical Sciences*, Vol. 45, No. 2, April-June 1997, pp. 143-159.

[2]Sauer, Carl G., Jr., and Yen, Chen-Wan L., "Planetary Mission Capability of Small Low Power Solar Electric Propulsion Systems," IAA-L-0706, IAA International Conference on Low-Cost Planetary Missions, Laurel, MD, April 12-15, 1994.

[3]Rayman, Marc D., and Lehman, David H. "NASA's First New Millennium Deep-Space Technology Validation Flight," IAA-L-0502, Second IAA International Conference on Low-Cost Planetary Missions, Laurel, MD, April 16-19, 1996.

[4]Rayman, Marc D., Fraschetti, Thomas C., Raymond, Carol A., and Russell, Christopher T., "Dawn: A Mission in Development for Exploration of Main Belt Asteroids Vesta and Ceres," *Acta Astronautica*, Vol. 58, 2006, pp. 605-616.

[5]Sauer, Carl G., Jr., "Solar Electric Performance for Medlite and Delta Class Planetary Missions," Paper AAS 97-726, AAS/AIAA Astrodynamics Specialist Conference, Sun Valley, Idaho, August 4-7, 1997.

[6]Betts, John T., "Optimal Interplanetary Orbit Transfers by Direct Transcription," *Journal of the Astronautical Sciences*, Vol. 42, No. 3, July-September 1994, pp. 247-268.

[7]Coverstone-Carroll, Victoria, and Williams, Steven N., "Optimal Low Thrust Trajectories Using Differential Inclusion Concepts," *Journal of the Astronautical Sciences*, Vol. 42, No. 4, October-December 1994, pp. 379-393.

[8]Kluever, Craig A., "Optimal Low-thrust Interplanetary Trajectories by Direct Method Techniques," *Journal of the Astronautical Sciences*, Vol. 45, No. 3, July-September 1997, pp. 247-262.

[9]Tang, Sean, and Conway, Bruce A., "Optimization of Low-Thrust Interplanetary Trajectories Using Collocation and Nonlinear Programming," Journal of Guidance, Control, and Dynamics, Vol. 18, N0. 3, May-June 1995, pp. 599-604.

[10]Sims, Jon A., and Flanagan, Steve N., "Preliminary Design of Low-Thrust Interplanetary Missions," Paper AAS 99-338, AAS/AIAA Astrodynamics Specialist Conference, Girdwood, Alaska, August 16-18, 1999.

[11]Byrnes, Dennis V., and Bright, Larry E., "Design of High-Accuracy Multiple Flyby Trajectories Using Constrained Optimization," AAS/AIAA Astrodynamics Specialist Conference, AAS 95-307, Halifax, Nova Scotia, Canada, August 14-17, 1995.

[12]Gill, Philip E., "User's Guide for SNOPT Version 7: A FORTRAN Package for Large-Scale Nonlinear Programming," University of California, San Diego, December 4, 2004.

[13]Sauer, Carl G., "MIDAS: Mission Design and Analysis Software for the Optimization of Ballistic Interplanetary Trajectories," *Journal of the Astronautical Sciences*, Vol. 37, No. 3, 1989, pp. 251-259.

[14]Parcher, Daniel W., and Sims, Jon A., "Venus and Mars Gravity-Assist Trajectories to Jupiter Using Nuclear Electric Propulsion," AAS/AIAA Space Flight Mechanics Meeting, AAS 05-112, Copper Mountain, Colorado, January 23-27, 2005.

[15]Parcher, Daniel W., and Sims, Jon A., "Earth Gravity-Assist Trajectories to Jupiter Using Nuclear Electric Propulsion," AAS/AIAA Astrodynamics Specialist Conference, AAS 05-397, Lake Tahoe, California, August 7-11, 2005.

[16]Parcher, Daniel W., and Sims, Jon A., "Gravity-Assist Trajectories to Jupiter Using Nuclear Electric Propulsion," AAS/AIAA Astrodynamics Specialist Conference, AAS 05-398, Lake Tahoe, California, August 7-11, 2005.

[17]Kowalkowski, Theresa D., Kangas, Julie A., and Parcher, Daniel W., "Jupiter Icy Moons Orbiter Interplanetary Injection Period Analysis," AAS/AIAA Space Flight Mechanics Meeting, AAS 06-187, Tampa, Florida, January 22-26, 2006.

[18]Petropoulos, Anastassios E., Kowalkowski, Theresa D., Vavrina, Matthew A., Parcher, Daniel W., Finlayson, Paul A., Whiffen, Gregory J., and Sims, Jon A., "1st ACT Global Trajectory Optimisation Competition: Results Found at the Jet Propulsion Laboratory", *Acta Astronautica* (to be published).