To the Graduate Council:

I am submitting herewith a thesis written by Brian Kendell Funk entitled "Trajectory Design and Propulsion System Comparison for a Near Earth Asteriod Rendezvous Mission." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Aerospace Engineering.

Dr. Gary Flandro, Major Professor

We have read this thesis
and recommend its acceptance:

Accepted for the Council:

Associate Vice Chancellor
and Dean of The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master if Science degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Brian K Funk

4/23/93

TRAJECTORY DESIGN AND PROPULSION SYSTEM COMPARISON FOR A NEAR EARTH ASTEROID RENDEZVOUS MISSION

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Brian Kendell Funk

May 1993

DEDICATION

This Thesis is dedicated to my fiancee

Carolyn M. Cleary

and

to my parents

Kennerly W. Funk and Janet K. Funk

ACKNOWLEDGMENTS

I would like to thank H. Dan. Thomas and A. Allen McCarley for developing the impulsive thrust program and the low thrust optimization program with me. I also want to thank H. Dan. Thomas for letting me use his trajectory animation programs in my oral defense. My advisor, Dr. Gary Flandro, provided a lot of insight and encouragement throughout the development of the programs and the writing of my thesis. Carl Sauer of the Jet Propulsion Laboratory answered the question that enabled us to finish the low thrust trajectory optimization program. I would also like to thank Dr. Robert Roach and Dr. Roy Schulz for being on my thesis committee.

ABSTRACT

If the cost of space flight were drastically reduced, mining operations on metal rich asteroids could be profitable. The objective of this thesis is to design an asteroid rendezvous mission and to compare possible propulsion system options. The two propulsion systems studied in detail are a solar sail hybrid system and a liquid chemical system. The asteroid selected is the near Earth asteroid, Eros.

In order to compare these propulsion schemes, two programs were written for use with desktop computers to model the optimal interplanetary trajectories of these propulsion systems. The impulsive thrust program uses Battin's modified Gauss algorithm to solve for the three-dimensional space trajectory based on the two-body problem. The low-thrust program utilizes Pontryagin's maximization principle to optimize the solar sail trajectory generated by a fourth order Runge-Kutta integration algorithm and the method of steepest descent, and Newton's method to solve the resulting two point boundary value problem. Both of these programs contain animated graphics illustrating the spacecraft's trajectory, the Earth's orbit, and Eros's orbit.

Utilizing these tools, optimal mission profiles for each propulsion scheme were selected based upon the lowest required total energy . If the total mission energy was the same for multiple missions, minimum time of flight became the basis for selection. The minimum energy impulsive mission has a launch date of January 30, 2005, and a flight time of 291 days. The total C3 for this mission is 25.391 $(km/s)^2$, which requires a total delta V of 6.6393 km/s. Since the delta V required by the chemical booster is the same for all of the solar sail hybrid missions (3.1532 km/s), the minimum flight time missions were selected. The primary mission for the low acceleration (1 mm/s^2 characteristic acceleration) sail has a launch date of June 6, 2002 and a flight time of 397 days. The primary mission

for a 2 mm/s^2 characteristic acceleration sail has a launch date of July 25, 2002 and a flight time of 286 days.

The figures of merit used to compare the three propulsion systems were the initial mass in low Earth orbit (IMLEO), and the Earth to Eros flight time. The IMLEO for the January 30, 2005 mission chemical system is 2730 kg. The 2mm/s^2 characteristic acceleration solar sail hybrid system has a mission flight time comparable to the chemical system, but the IMLEO is much greater at 6240 kg. The smaller 1 mm/s2 characteristic acceleration solar sail hybrid requires an addition 106 days of flight time, but at a significant IMLEO mass savings. The IMLEO of the 1 mm/s^2 characteristic acceleration solar sail hybrid is equal to 1590 kg. Since the flight time is not as critical as the mission cost for the unmanned asteroid rendezvous mission, the 1 mm/s^2 sail hybrid vehicle is the best propulsion system option. The small solar sail has an additional benefit in that it is reusable.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

a	Semi-major axis
$\bar{a}$	Acceleration
ad	Sail area density
a_o	Characteristic acceleration of the solar sail
a_r	Radial acceleration
a_ϕ	Tangential acceleration
a_ψ	Normal acceleration
C3	Square of the relative heliocentric velocity
e	Eccentricity
$\bar{e}_r$	Unit vector in the radial direction
$\bar{e}_\phi$	Unit vector in the tangential direction
$\bar{e}_\psi$	Unit vector in the normal direction
g	Gravity
GM	Universal gravitational constant times the mass of the Sun
H	Hamiltonian of the system
HCA	Heliocentric angle
i	Inclination
$\bar{i}$	Unit vector in the radial direction
IMLEO	Initial mass in low Earth orbit
I_{sp}	Specific impulse
JPL	Jet Propulsion Lab
ks	Solar sail structural factor
M_{dry}	Spacecraft's dry mass
M_o	Spacecraft's initial mass

M_{PL}	Payload mass
M_{sail}	Solar sail mass
m	Spacecraft mass
$\bar{n}$	Unit vector in the direction normal to the plane of the solar sail
NASA	National Aeronautics and Space Administration
NERVA	Nuclear Engine for Rocket Vehicle Application
P_r	Auxiliary variable, costate of the radius
P_{sol}	Solar radiation pressure
P_ϕ	Auxiliary variable, costate of the celestial longitude
P_ψ	Auxiliary variable, costate of the celestial latitude
P_{Vr}	Auxiliary variable, costate of the radial velocity
$P_{V\phi}$	Auxiliary variable, costate of the tangential velocity
$P_{V\psi}$	Auxiliary variable, costate of the normal velocity
Q_i	Generalized force or moment
q_i	Generalized coordinate
r	Radial direction
r_o	Radial distance from the Sun to the Earth
S_o	Solar sail surface area
SNRE	Small Nuclear Rocket Engine
T	Kinetic energy
t	Time
V_r	Radial velocity
V_ϕ	Tangential velocity
V_ψ	Normal velocity
Y	Direction 90 degrees counter clockwise from the line to Aries
Z	Direction out of the ecliptic plane

β	Solar sail clock angle
γ	Line to Aries
ΔV	Required velocity change
δ	Solar sail thickness
θ	Solar sail cone angle
Π	Sum of longitude of the ascending node and the argument of the periapsis
ρ	Density of the solar sail material
σ	Ratio of the spacecraft's initial mass and the spacecraft's dry mass
σ_{arrive}	Mass ratio at arrival delta V
σ_{depart}	Mass ratio at departure delta V
ϕ	Celestial longitude
ψ	Celestial latitude
Ω	Longitude of the ascending node
ω	Argument of the periapsis

1. INTRODUCTION

1.1 Statement of Problem

The National Aeronautics and Space Administration (NASA) is interested in studying the composition of asteroids because they "are believed to be among the oldest bodies in the solar system and are believed to have remained unchanged since their formation in the early solar nebula." Williams, Knocke, and Bright (1, p. 957). Several of the current asteroid mission designs result in a flyby of the asteroid in order to reduce the cost of the mission. The cost of a rendezvous mission is greater than the flyby mission cost, but there are benefits associated with a rendezvous mission, such as circumnavigation of the asteroid and possibly robotic ascent to the surface for composition analysis, internal structural analysis, and heat flow analysis. According to spectrographic data, some asteroids are composed of metals that are used extensively on earth. Thus, a secondary mission objective is determining the feasibility of asteroid mining. In order to conduct an asteroid rendezvous mission, the cost must be minimized. One way to reduce the cost is to develop efficient propulsion systems. If asteroid mining is to be realized, low cost and reusable propulsion systems must be developed. Two of the more promising low cost propulsion schemes appear to be nuclear thermal and solar sail. One of the major drawbacks of the solar sail is its inability to operate efficiently near the Earth. It takes a very long time for it to spiral to escape velocity because of the significant amount of time spent in the Earth's shadow. In order to reduce trip times and make the sail a more competitive option, a chemical system is used to boost the sail to escape velocity. Thus the sail analyzed in this study is actually a hybrid system. The main objective of this thesis is to design an asteroid rendezvous mission to Eros for the 1996 - 2012 time period, and to compare possible propulsion schemes using this mission. The rendezvous mission is a light weight, low cost, scientific mission. To date, most of the nuclear thermal rocket

engines have been designed with the manned Mars mission in mind. There is an engine designed for unmanned missions; however, the mass of this engine is too large for the mission studied in this thesis. The mass of the nuclear thruster alone is comparable to the initial mass of the chemical system including the thrusters, the spacecraft, the propellant, and the payload. Thus, the initial mass of the nuclear system is greater than the chemical system. The primary basis for comparison of the propulsion system options is initial mass, which relates directly to mission cost; therefore, a mission using nuclear propulsion would cost more than a mission using chemical propulsion. Because of the high masses associated with the existing designs and the anti-nuclear political climate, the nuclear thermal propulsion system was not studied in detail for this asteroid rendezvous mission.

The asteroid Eros was selected as the target body because of its size and its well known orbit. Eros has an ellipsoid shape with a major diameter of 36 km and a minor diameter of 14 km. Eros's orbit is well known and precisely defined. The orbital elements are presented in Appendix A. "The ascending and descending nodal distances are 1.40 and 1.57 A.U." Sauer (2, p. 999). Although Eros passes within 0.2 A.U. of Earth's orbit, it represents a significant mission design challenge because of its high inclination (10.832). The energy requirements are higher for orbital transfers out of the ecliptic plane. Eros may be a more difficult target than some of the smaller near earth asteroids, but its well defined orbit makes it a less risky target. In addition, its size simplifies the job of the spacecraft's acquisition sensor.

1.2 Propulsion System Background

The two advanced propulsion schemes studied in this thesis are not new ideas. The United States had a solid core nuclear rocket program called Rover that began in 1955. This NASA - Los Alamos Scientific Laboratory program lead to graphite core reactor tests beginning in 1959 with the Kiwi-A. This reactor "was designed for 100 megawatts of

thermal energy" Rom (3, p.10). The next series of tests were on a 1100 megawatt reactor. Once the problems associated with this reactor were solved, a breadboard engine called NRX-EST was developed and tested. This breadboard engine led to the development of the Nuclear Engine for Rocket Vehicle Application (NERVA). "This engine was designed and operated at 1500 MW with a thrust of 333 kN. The engine was designed for a 10-hour life and 20 operating cycles." Frisbee (4, p. 5). This engine was the only nuclear rocket ever tested. The rocket used graphite fuel elements through which hydrogen gas was heated and ejected through a nozzle. A smaller nuclear rocket engine was designed by the Los Alamos National Laboratory. The Small Nuclear Rocket Engine (SNRE) was designed for unmanned missions. The NERVA engine was designed with manned missions in mind. In fact, the motivation for the NERVA engine was a manned mission to Mars. The SNRE, "based on Pewee technology, was a 370 MW engine with a thrust of 72.6 kN, a 2-hour life, and 20 operating cycles" Frisbee (4, p. 5). This engine had a thrust comparable to the Pratt & Whitney RL 10-A, of 73.4 kN, according to Table 17-3 in Wertz and Larson (5, p. 585). Because of budgetary constraints, the nuclear rocket program was canceled in the mid-seventies. Because of the revitalization of the manned Mars mission initiated by the Bush administration, nuclear thermal propulsion has seen renewed interest recently. Since the asteroid mission is unmanned, the SNRE design represents the type of nuclear thermal system that could be used. According to Frisbee, the SNRE engine weighed 2600 kg. The weight of this engine is too large for the light weight, low cost asteroid rendezvous mission being considered in this thesis. As mentioned before, the mass of the SNRE engine is comparable to the initial mass of the chemical spacecraft including propellant and the mission payload. Thus, the initial mass in low Earth orbit and the mission cost of a nuclear propelled spacecraft is higher than the chemical spacecraft for this mission. As the mission payloads increase, the weight associated with the nuclear system becomes less significant. Since nuclear systems have a higher Isp (825

s for the NERVA and 900 s for the SNRE, Frisbee (4, p. 5)) than chemical systems, they require less propellant mass for a given delta V. This propellant mass savings can reduce the initial mass and cost of heavy missions as illustrated in reference 4. Thus, the nuclear thermal engines designed to date appear to be better suited for manned or large unmanned missions.

The solar sail concept which "works by intercepting the sunlight photon pressure acting over the sail area imparting momentum to the vehicle", Friedman (6, p. 1), also has a fairly long history. The concept was first mentioned by Russian scientists in 1924. According to Louis Friedman, "Fridrickh Arturovich Tsander wrote, 'For flight in interplanetary space I am working on the idea of flying, using tremendous mirrors of very thin sheets, capable of achieving favorable results.' " Friedman (7, p. 9). Konstantin Tsiolkovsky also mentioned the use of solar pressure in the mid 1920's. According to Friedman (7, p. 10), the first American to write about solar sailing was Carl Wiley who published an article entitled, "Clipper Ships of Space" under the pseudonym "Russell Sanders". This article appeared in the May 1951 issue of Astounding Science Fiction. Dr. Richard Garwin is responsible for the first solar sail article published in a professional journal. The article was published in 1958, and it "included preliminary calculations of sail-vehicle performance" Friedman (7, p. 10). The early solar sail designs utilized spin to add stability. The people credited with the spinning sail idea are Ted Cotter, and Philippe Villers. NASA became interested in the concept in the mid 1960's but this interest decreased with the shrinking post Apollo space program. According to Friedman (7, p. 11), the solar sail concept was reincarnated by JPL in the mid 1970's. Jarome Wright at Battelle Memorial Institute in Ohio, designed a Halley's comet rendezvous mission using a solar sail for propulsion. This mission coupled with the advancements in the area of large space structures and the space shuttle development spurred the JPL solar sail design study in 1977 and 1978. Two designs were studied, a square sail and a heliogyro sail. The

heliogyro was invented by Richard MacNeal and John Hedgepath "between 1965 and 1967" Friedman (7, p. 11). The result of the preliminary design study was that the solar sail concept was feasible for the Halley's comet rendezvous mission. At the same time, NASA Lewis was proposing a solar electric propulsion system for the Halley mission. Because the solar sail was considered to be "not sufficiently mature", Friedman (6, p. 13), the solar electric system was chosen as the propulsion scheme for the Halley rendezvous mission which was subsequently scrapped. To date there are no NASA funded solar sail studies. The only solar sail work currently being done in the United States is funded by the World Space Foundation. This group is composed mostly of JPL engineers who are trying to continue solar sail development. In this thesis, an ideally reflecting square sail, similar to the Halley mission design sail, is used in the propulsion system comparison for an Earth to Eros rendezvous mission.

1.3 Methodology for Orbit Determination

The first task of the study was to develop a computer code that models an impulsively thrusted spacecraft's three-dimensional space trajectory. A patched conic trajectory was used to simplify the analysis. Parking orbits were assumed at the launch point in order to calculate the spacecraft's ΔV requirement. The program uses Battin's modified Gauss algorithm for solving Lambert's problem. The code also contains animated graphics to illustrate the relative location and motion of the inner planets, the spacecraft, and the asteroid, Eros. In addition to the graphical output and the required ΔV's, the program calculates all of the orbital elements required to specify a three-dimensional trajectory. In order to select the optimal mission time frame, the program must iterate the launch date while holding the time of flight constant. This program is used to generate graphs of constant flight time showing the variation of total mission C3 or energy as a function of launch date. These graphs illustrate the approximate launch date and flight

time for a minimum energy mission; however, energy contour graphs are needed to find the actual launch date and time of flight that result in a minimum energy mission. The energy contours are obtained by modifying the original program to assume a given total C3 and use Newton's method to iterate the flight time for each launch date. The calculation of the flight times corresponding to a specified C3 is repeated for several launch dates around the approximate date found in the C3 variation graphs described above. The minimum C3 flight times corresponding to the approximate launch date are used as initial guesses for the Newton's method. The contour plots are typically concentric with the minimum total C3 curves in the middle. Thus, the lines representing the launch date and flight time corresponding to minimum energy solution intersect at a point in the middle of the contours. These programs are used to model the baseline chemical system.

The second task of the study was to develop a computer program that models low-thrust trajectories. This program uses a forth order Runge-Kutta scheme to integrate the equations of motion of a solar sail and generate possible trajectories. The algorithm for the Runge-Kutta scheme appears in the book, Numerical Analysis by Burden and Faires (8). The trajectories are optimized using Pontryagin's maximization principle. The implementation of this principle results in a two point boundary value problem that requires the Hamiltonian of the system to be maximized throughout the trajectory. The method of steepest descent and Newton's method are utilized to iterate the unknown initial values of the auxiliary variables, the launch date, and the time of flight until the spacecraft's final 3-D position and velocity meet that of the target body. The method of steepest descent is used to improve the initial guesses until they are within the range of convergence for Newton's method. The algorithms for these methods also appear in Numerical Analysis by Burden and Faires (8). The steepest descent algorithm is on pages 554 and 555. The Newton's method algorithm is on page 539. It finds the new initial guesses by inverting the Jacobian matrix. The partial derivatives in the Jacobian matrix are obtained numerically by stepping

the initial conditions, one at a time, and calculating the resulting change in the final conditions. The solar sail trajectories are optimal in that they are minimum time solutions. This condition is met by requiring the Hamiltonian of the system to be positive. In this program, the Hamiltonian of the system is assumed to be equal to one at the final time. In order to have a solvable two point boundary value problem the number of unknown initial conditions must equal the number of specified final conditions. For the three dimensional trajectory optimization problem there are seven unknown initial conditions: launch date, flight time and five auxiliary variables. The seven specified final conditions are the three components of position, the three components of velocity, and the transversality condition required for a minimum time solution (Hamiltonian = 1). The initial guesses for the 3-D solutions are not known so the problem must be solved in steps. The first step is to find the minimum time trajectory between circular coplanar orbits. This only requires three initial guesses (P_r, P_{V_r}, and P_{V_ϕ}). The next step is to find the launch dates that result in the proper alignment between Earth and the target body. The flight time and the change in celestial longitude are determined by the circular coplanar trajectory solution; thus, the proper alignment occurs when the longitude between Earth's initial position and the target's final position is equal to the trajectory celestial longitude change. The flight time and the three initial condition solutions from the circular coplanar problem, and the launch dates from the alignment analysis are then used as initial guesses for the 2-D elliptical orbit trajectory optimization. The initial condition, launch date, and flight time solutions from the 2-D problem become the initial guesses for the 3-D problem. With these five initial guesses specified, there are only two more initial guesses to obtain. For small inclination orbits, small values can be assumed for the remaining two initial auxiliary variables (P_ψ and P_{V_ψ}). For large inclination orbits, like Eros, these values must be obtained using trial and error. Fortunately the minimum time solutions are periodic in launch window repeatability: therefore the initial value solutions associated with a certain Earth - target

alignment can be used as initial conditions for the subsequent alignments. For example, there are three basic Earth to Eros trajectory solutions, and the geometry associated with each trajectory reoccurs approximately every seven years. All of the possible minimum time 3-D trajectories in the 1996 - 2012 time frame are calculated using these programs. Two values of the solar sail characteristic acceleration are investigated: 1mm/s^2 and 2 mm/s^2; thus, two primary solutions are obtained. These accelerations were chosen because the 2 mm/s^2 acceleration is near the maximum attainable using current materials, and the 1 mm/s^2 acceleration is the value assumed for the Halley rendezvous mission solar sail. The primary figure of merit for the mission selection is the launch energy. Since the solar sail is a chemical hybrid, cost is minimized by minimizing the total energy. If the launch energies for the missions are the same, the secondary figure of merit for the solar sail mission selection is flight time.

After the impulsive and solar sail missions were designed, the propulsion systems were compared on the basis of their initial mass in low Earth orbit (IMLEO) and their mission flight time. Using the mass of a typical asteroid mission spacecraft and payload, the mass of the fuel required was calculated. The mass of the payload, the mass of fuel required and the mass of the propulsion system was used to estimate the initial mass in low Earth orbit. Another comparison was made on the basis of time of flight, which is limited by the propulsion system's capabilities.

2. THEORY

2.1 Equations of Motion

The motion of a spacecraft in an inverse square gravity field can be described by three second order differential equations. This section shows the derivation of these equations using Lagrange's equation for non-conservative forces:

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}_i}\right) - \frac{\partial T}{\partial q_i} = Q_i \qquad (2.1.1)$$

where T is the kinetic energy, q_i is the generalized coordinate, and Q_i is the generalized force or moment. The equations of motion will be derived for a spherical coordinate system because it is the most convenient. The spherical coordinate system utilized in this analysis is illustrated below in Figure 2.1.1. This coordinate system is different from the typical spherical coordinate system in that ψ is measured up from the γ - Y plane. This coordinate system is standard for orbital mechanics.

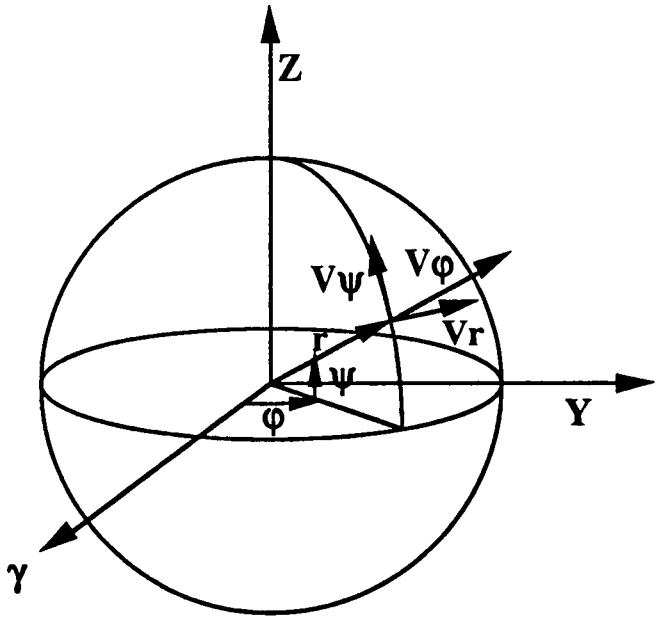

Figure 2.1.1: Spherical Coordinate System

For convenience, the equations of motion are derived using dimensionless variables. These variables are non-dimensionalized using the following formulas:

$$r = \frac{r^*}{r_o} \tag{2.1.2}$$

$$V = \frac{V^*}{\sqrt{\frac{GM}{r_o}}} \tag{2.1.3}$$

$$a = \frac{a^*}{\frac{GM}{r_o^2}} \tag{2.1.4}$$

$$t = \frac{t^*}{\sqrt{\frac{r_o^3}{GM}}} \tag{2.1.5}$$

where the asterisk denotes the dimensional variables. The scaling variable r_o is the distance between the Earth and the sun. Thus, the scaling variables in equations (2.1.2), (2.1.3), and (2.1.4) are the orbital radius of the Earth, the circular velocity of a point mass at Earth's orbital radius, and the acceleration of a point mass due to the sun's gravity at Earth's orbital radius respectively. The scaling variable in equation (2.1.5) is proportional to the period of an orbit at Earth's distance from the sun. For interplanetary motion, the central body is the sun; thus, the GM term in the above formulas is the universal gravitational constant (G) times the mass of the sun (M).

From the general form of Lagrange's equation (2.1.1), the first step in the derivation is to find an expression for the kinetic energy in a spherical coordinate system. The spherical velocity components are as follows:

$$V_r = \dot{r} \tag{2.1.6}$$

$$V_\phi = r\cos(\psi)\dot{\phi} \tag{2.1.7}$$

$$V_\psi = r\dot{\psi} \tag{2.1.8}$$

Since the kinetic energy is equal to:

$$T = \frac{1}{2}m(V_r^2 + V_\phi^2 + V_\psi^2) \tag{2.1.9}$$

it can be expressed in the following form:

$$T = \frac{1}{2}m(\dot{r}^2 + r^2\cos^2(\psi)\dot{\phi}^2 + r^2\dot{\psi}^2) \tag{2.1.10}$$

In order to obtain the generalized coordinates, the degrees of freedom of the system must be known. There must be one generalized coordinate for every degree of freedom. The spacecraft has three degrees of freedom; therefore, three generalized coordinates are needed. In the spherical coordinate system, the generalized coordinates are the radial distance, the celestial longitude, and the celestial latitude. The three equations of motion are obtained by using equation (2.1.1) for each of the generalized coordinates. The first generalized coordinate is the radial distance. Lagrange's equation for this coordinate is:

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{r}}\right) - \frac{\partial T}{\partial r} = Q_r \tag{2.1.11}$$

where the generalized force is equal to the sum of the spacecraft's thrust in the radial direction and the gravitational attraction of the central body:

$$Q_r = -\frac{GMm}{r^2} + ma_r^* \tag{2.1.12}$$

This is the dimensional form of the generalized force equation. The accelerations are non-dimensionalized using equation (2.1.4). Thus, equation (2.1.12) becomes:

$$Q_r = -\frac{m}{r^2} + ma_r \tag{2.1.13}$$

The r_o term drops out of the equation because r_o is equal to 1 AU. The partial derivatives of the kinetic energy with respect to r and the time rate of change of r are:

$$\frac{\partial T}{\partial r} = mr\cos^2(\psi)\dot{\phi}^2 + mr\dot{\psi}^2 \tag{2.1.14}$$

$$\frac{\partial T}{\partial \dot{r}} = m\dot{r} \tag{2.1.15}$$

Equations (2.1.13), (2.1.14), and the time derivative of equation (2.1.15) are substituted directly into equation (2.1.11). After dividing through by the spacecraft's mass (m), equation (2.1.11) becomes:

$$\ddot{r} - r\cos^2(\psi)\dot{\phi}^2 - r\dot{\psi}^2 + \frac{1}{r^2} = a_r \tag{2.1.16}$$

Lagrange's equation for the celestial longitude is:

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\phi}}\right) - \frac{\partial T}{\partial \phi} = Q_\phi \tag{2.1.17}$$

Because the celestial longitude coordinate is an angle, Q_ϕ must be a generalized moment. In this case, the moment is due to the spacecraft's thrust in the longitudinal direction:

$$Q_\phi = mr\cos(\psi)a_\phi \tag{2.1.18}$$

The gravitational attraction term does not appear in this equation because it only acts in the radial direction. The partial derivatives of the kinetic energy with respect to the longitude and the time rate of change of the longitude are:

$$\frac{\partial T}{\partial \phi} = 0 \tag{2.1.19}$$

$$\frac{\partial T}{\partial \dot{\phi}} = mr^2\cos^2(\psi)\dot{\phi} \tag{2.1.20}$$

Equation (2.1.18) and the time derivative of equation (2.1.20) are substituted into equation (2.1.17). After dividing through by [mr cos(ψ)], equation (2.1.17) becomes:

$$r\cos(\psi)\ddot{\phi} + 2\dot{r}\dot{\phi}\cos(\psi) - 2r\dot{\phi}\dot{\psi}\sin(\psi) = a_{\phi} \tag{2.1.21}$$

This procedure is repeated again for the third generalized coordinate, the celestial latitude. Lagrange's equation for this coordinate is:

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\psi}}\right) - \frac{\partial T}{\partial \psi} = Q_{\psi} \tag{2.1.22}$$

The generalized moment is due to the spacecraft's thrust in the normal direction:

$$Q_{\psi} = mra_{\psi} \tag{2.1.23}$$

The partial derivatives of the kinetic energy with respect to the latitude and the time rate of change of the latitude are as follows:

$$\frac{\partial T}{\partial \psi} = -mr^2\cos(\psi)\sin(\psi)\dot{\phi}^2 \tag{2.1.24}$$

$$\frac{\partial T}{\partial \dot{\psi}} = mr^2\dot{\psi} \tag{2.1.25}$$

Once again, the partial derivative equations, (2.1.19) and (2.1.20), and the generalized moment equation (2.1.18) are substituted into equation (2.1.17). After dividing through by (mr), the final equation of motion has the form:

$$r\ddot{\psi} + 2\dot{r}\dot{\psi} + r\dot{\phi}^2\cos(\psi)\sin(\psi) = a_{\psi} \tag{2.1.26}$$

In order to generate the trajectory of a low thrust spacecraft, the equations of motion (2.1.16), (2.1.21), and (2.1.26) must be integrated numerically. The low thrust trajectory optimization program in Appendix B uses a fourth order Runge-Kutta scheme. Before the equations of motion can be integrated, they must be reduced from second order to first

order. Since a second order differential equation can be replaced by two first order equations, the three second order equations of motion can be replaced by six first order equations. These equations are as follows:

$$\frac{dr}{dt} = V_r \tag{2.1.27}$$

$$\frac{d\phi}{dt} = \frac{V_\phi}{r\cos(\psi)} \tag{2.1.28}$$

$$\frac{d\psi}{dt} = \frac{V_\psi}{r} \tag{2.1.29}$$

$$\frac{dV_r}{dt} = \frac{V_\phi^2 + V_\psi^2}{r} - \frac{1}{r^2} + a_r \tag{2.1.30}$$

$$\frac{dV_\phi}{dt} = \frac{V_\phi V_\psi \tan(\psi)}{r} - \frac{V_r V_\phi}{r} + a_\phi \tag{2.1.31}$$

$$\frac{dV_\psi}{dt} = -\frac{V_r V_\psi}{r} - \frac{V_\phi^2 \tan(\psi)}{r} + a_\psi \tag{2.1.32}$$

2.2 Solar Sail Acceleration and Control

The equations of motion derived in the previous section are for any spacecraft operating in an inverse square gravity field; thus, the accelerations in equations (2.1.30) - (2.1.32) are general accelerations. In order to model the solar sail, expressions for the sail's acceleration must be substituted into equations (2.1.30) - (2.1.32). The solar sail acceleration is a result of the solar photon pressure on its surface. The sail's acceleration vector is aligned with the vector normal to the sail's surface ($\bar{n}$) as illustrated in Figure (2.2.1).The direction of the sail's acceleration or thrust vector is specified using the two angles pictured in Figure 2.2.1. The first angle is the thrust vector cone angle (θ), which is the angle between the incident solar radiation ($\bar{e}_r$) and the sail's normal vector ($\bar{n}$). The

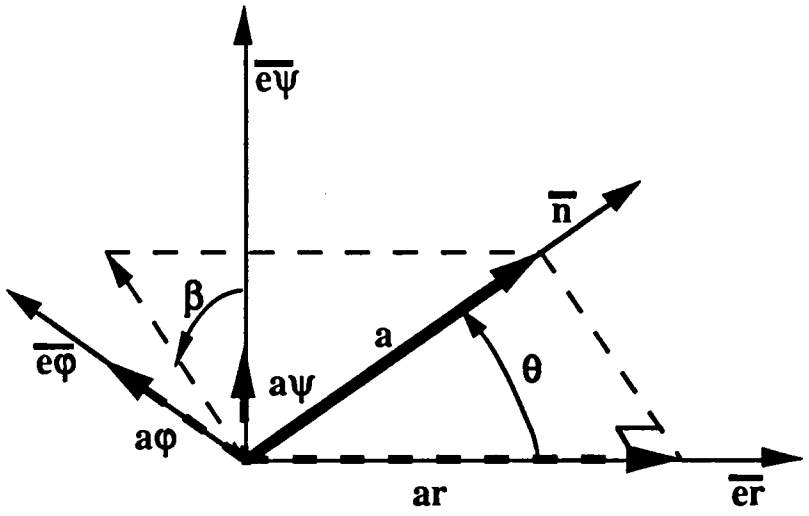

Figure 2.2.1: Solar Sail Control Angles

second angle is the thrust vector clock angle (β), which is the angle between the $\bar{e}_\psi$ axis of the spherical, body-fixed coordinate system and the projection of the sail's normal vector ($\bar{n}$) into the plane normal to the incident sunlight.

In order to simplify the analysis, an ideally reflecting, flat, square sail is assumed. The same model was also used by A. N. Zhukov and V. N. Lebdev (9, p.41), Grodzovskii (10, p. 316), and Carl G. Sauer (11, p. 2). The sail acceleration equation as it appears in Grodzovskii (10, p. 316) is:

$$\bar{a} = a_o\left(\frac{r_{earth}}{r}\right)^2(\bar{n}\cdot\bar{i})^2\bar{n} \qquad (2.2.1)$$

where a_o is the characteristic acceleration of the sail, which is defined as the acceleration of the sail when it is oriented normal to the incident sunlight at 1 AU. The characteristic acceleration can be calculated using the following formula:

$$a_o = \frac{P_{sol}S_o}{M_{sail} + M_{PL}} \qquad (2.2.2)$$

This expression is merely force (solar radiation pressure times the sail area) divided by mass (the mass of the sail and the mass of the payload). The sail area (S_o) is assumed to be constant; thus, sail wear is not considered. Equation (2.2.2) contains a reflectivity term which is equal to one for the case of an ideally reflecting sail. Since the mass of the sail (M_{sail}) is equal to the product of the sail area (S_o), the density of the sail material (ρ), the sail's thickness (δ), and the sail structural factor (ks), which is proportional to the sail area according to Sauer (11, p. 2), equation (2.2.2) can also be expressed in the following form:

$$a_o = \frac{P_{sol} S_o}{S_o \rho \delta (1 + ks) + M_{PL}} \tag{2.2.3}$$

The mass of the sail related structure and spars make up approximately fifty percent of the total mass of the JPL Halley rendezvous solar sail vehicle according to J. Wright and J. Warmke (12, p. 2); thus a sail structural factor of 0.5 was used in this analysis.

In order to utilize equations (2.1.30) - (2.1.32), expressions for the vector components of equation (2.2.1) must be found. The sail's normal vector is:

$$\bar{n} = \cos(\theta)\bar{e}_r + \sin(\theta)\sin(\beta)\bar{e}_\phi + \sin(\theta)\cos(\beta)\bar{e}_\psi$$

and $\bar{i} = 1\bar{e}_r$; thus:

$$\left(\bar{n} \cdot \bar{i}\right)^2 = \cos^2(\theta)$$

After substituting these terms into equation (2.2.1), the sail's acceleration can be expressed in the following form:

$$\bar{a} = a_r \bar{e}_r + a_\phi \bar{e}_\phi + a_\psi \bar{e}_\psi \tag{2.2.4}$$

where:

$$a_r = a_o\left(\frac{r_{earth}}{r}\right)^2 \cos^3(\theta) \tag{2.2.4a}$$

$$a_\phi = a_o\left(\frac{r_{earth}}{r}\right)^2 \cos^2(\theta)\sin(\theta)\sin(\beta) \tag{2.2.4b}$$

$$a_\psi = a_o\left(\frac{r_{earth}}{r}\right)^2 \cos^2(\theta)\sin(\theta)\cos(\beta) \tag{2.2.4c}$$

The solar sail equations of motion are obtained by substituting equations (2.2.4a), (2.2.4b) and, (2.2.4c) into equations (2.1.30), (2.1.31), and (2.1.32) respectively. The final form of the solar sail equations of motion are:

$$\frac{dr}{dt} = V_r \tag{2.2.5}$$

$$\frac{d\phi}{dt} = \frac{V_\phi}{r\cos(\psi)} \tag{2.2.6}$$

$$\frac{d\psi}{dt} = \frac{V_\psi}{r} \tag{2.2.7}$$

$$\frac{dV_r}{dt} = \frac{V_\phi{}^2 + V_\psi{}^2}{r} - \frac{1}{r^2} + \frac{a_o}{r^2}\cos^3(\theta) \tag{2.2.8}$$

$$\frac{dV_\phi}{dt} = \frac{V_\phi V_\psi \tan(\psi)}{r} - \frac{V_r V_\phi}{r} + \frac{a_o}{r^2}\cos^2(\theta)\sin(\theta)\sin(\beta) \tag{2.2.9}$$

$$\frac{dV_\psi}{dt} = -\frac{V_r V_\psi}{r} - \frac{V_\phi{}^2 \tan(\psi)}{r} + \frac{a_o}{r^2}\cos^2(\theta)\sin(\theta)\cos(\beta) \tag{2.2.10}$$

2.3 Solar Sail Trajectory Optimization

One of the objectives of this comparison study is to obtain minimum time, solar sail rendezvous trajectories to the asteroid 433 Eros. These optimal trajectories are calculated using calculus of variations. Specifically, Pontryagin's maximization principle is utilized to

calculate the optimal control functions, and reduce the problem to a two point boundary value problem. The maximization principle utilized in the sail trajectory optimization program is based upon Theorem 2 in Pontryagin (13, p.20):

> "Let $u(t)$, $t_0 \leq t \leq t_1$, be a permissible control taking the phase point from the position x_0 to x_1, and $x(t)$ the corresponding trajectory, so that $x(t_0) = x_0$, $x(t_1) = x_1$. A necessary condition for the control $u(t)$ and the trajectory $x(t)$ to be time optimal is that there exists a non-zero continuous vector function $\psi(t) = [\psi_1(t), \psi_2(t), ..., \psi_n(t)]$, corresponding to the functions $u(t)$ and $x(t)$ such that:
>
> (1) for all t, $t_0 \leq t \leq t_1$, the function $H(\psi(t), x(t), u)$ of the variable $u \in U$ attains a maximum at the point $u = u(t)$:
>
> $$H(\psi(t), x(t), u(t)) = M(\psi(t), x(t));$$
>
> (2) at the final instant t_1, the relationship
>
> $$M(\psi(t_1), x(t_1)) \geq 0$$
>
> is satisfied. Furthermore, if the quantities $\psi(t)$, $x(t)$, $u(t)$ satisfy systems (18), (19) and condition (1), the function $M(\psi(t), x(t))$ of variable t proves to be constant, so that the verification of [condition (2)] can be carried out at any instant t, $t_0 \leq t \leq t_1$, and not necessarily at the instant t_1."

This theorem states that in order to obtain a time optimal trajectory, the Hamiltionian of the system $H(\psi(t), x(t), u(t))$ must be maximized using the control functions, and the value of $H(\psi(t), x(t), u(t))$ at the final time must be greater than or equal to zero. The control functions that maximize the Hamiltonian are obtained by setting the partial derivatives of the Hamiltonian with respect to the control variables (θ and β) equal to zero. The specification that the Hamiltonian be greater than or equal to one is the transversality condition that insures time minimization. The Hamiltionian is defined by the following equation:

$$H(P,x,u) = \sum_{i=1}^{n} P_i f^i(x,u) \qquad (2.3.1)$$

The auxiliary variables (P_i) in equation (2.1.3) correspond to the ψ terms in Theorem 2. The variable P_i is utilized in order to avoid confusion between the auxiliary variables and the celestial latitude (ψ). The term $f^i(x,u)$ represents the equations of motion (2.2.5) - (2.2.10). From equation (2.3.1) the Hamiltonian for the solar sail is:

$$H = P_r V_r + P_\phi\left(\frac{V_\phi}{r\cos(\psi)}\right) + P_\psi\left(\frac{V_\psi}{r}\right) +$$

$$P_{V_r}\left(\frac{V_\phi^2 + V_\psi^2}{r} - \frac{1}{r^2} + \frac{a_o}{r^2}\cos^3(\theta)\right) +$$

$$P_{V_\phi}\left(\frac{V_\phi V_\psi \tan(\psi)}{r} - \frac{V_r V_\phi}{r} + \frac{a_o}{r^2}\cos^2(\theta)\sin(\theta)\sin(\beta)\right) +$$

$$P_{V_\psi}\left(-\frac{V_r V_\psi}{r} - \frac{V_\phi^2 \tan(\psi)}{r} + \frac{a_o}{r^2}\cos^2(\theta)\sin(\theta)\cos(\beta)\right) \qquad (2.3.2)$$

From Pontryagin (13, p. 19), the Hamiltonian system is defined by the following equations:

$$\frac{dx^i}{dt} = \frac{\partial H}{\partial P_i}, \qquad i = 1, 2,..., 6 \qquad (2.3.3)$$

$$\frac{dP_i}{dt} = -\frac{\partial H}{\partial x^i}, \qquad i = 1, 2,..., 6 \qquad (2.3.4)$$

Equations (2.3.3), and (2.3.4) correspond to equations (18) and (19) which are referenced in Theorem 2. Equation (2.3.3) represents the equations of motion (2.2.5) - (2.2.10). Equation (2.3.4) is utilized to generate the differential equations that describe the variation of the auxiliary variables with time:

$$\frac{dP_r}{dt} = \left(\frac{V_\phi}{r^2\cos(\theta)}\right)P_\phi + \left(\frac{V_\psi}{r^2}\right)P_\psi +$$
$$\left(\frac{V_\phi^2 + V_\psi^2}{r^2}\right)P_{V_r} - \left(\frac{2}{r^3}\right)P_{V_r} + \left(\frac{2a_o}{r^3}\cos^3(\theta)\right)P_{V_r} -$$
$$\left(\frac{V_r V_\phi}{r^2}\right)P_{V_\phi} + \left(\frac{V_\phi V_\psi \tan(\theta)}{r^2}\right)P_{V_\phi} + \left(\frac{2a_o}{r^3}\cos^2(\theta)\sin(\theta)\sin(\beta)\right)P_{V_\phi} -$$
$$\left(\frac{V_r V_\psi}{r^2}\right)P_{V_\psi} - \left(\frac{V_\phi^2 \tan(\theta)}{r^2}\right)P_{V_\psi} + \left(\frac{2a_o}{r^3}\cos^2(\theta)\sin(\theta)\cos(\beta)\right)P_{V_\psi} \qquad (2.3.5)$$

$$\frac{dP_\phi}{dt} = 0 \qquad (2.3.6)$$

$$\frac{dP_\psi}{dt} = \left(\frac{V_\phi \sin(\theta)}{r\cos^2(\theta)}\right)P_\phi - \left(\frac{V_\phi V_\psi}{r\cos^2(\theta)}\right)P_{V_\phi} + \left(\frac{V_\phi^2}{r\cos^2(\theta)}\right)P_{V_\psi} \qquad (2.3.7)$$

$$\frac{dP_{V_r}}{dt} = -P_r + \left(\frac{V_\phi}{r}\right)P_{V_\phi} + \left(\frac{V_\psi}{r}\right)P_{V_\psi} \qquad (2.3.8)$$

$$\frac{dP_{V_\phi}}{dt} = -\left(\frac{1}{r\cos(\theta)}\right)P_\phi - \left(\frac{2V_\phi}{r}\right)P_{V_r} + \left(\frac{V_r}{r}\right)P_{V_\phi} -$$
$$\left(\frac{V_\psi \tan(\theta)}{r}\right)P_{V_\phi} + \left(\frac{2V_\phi \tan(\theta)}{r}\right)P_{V_\psi} \qquad (2.3.9)$$

$$\frac{dP_{V_\psi}}{dt} = -\frac{P_\psi}{r} - \left(\frac{2V_\psi}{r}\right)P_{V_r} - \left(\frac{V_\phi \tan(\theta)}{r}\right)P_{V_\phi} + \left(\frac{V_r}{r}\right)P_{V_\psi} \qquad (2.3.10)$$

According to equation (2.3.6), P_ϕ is equal to a constant. P_ϕ is constant because the celestial longitude (ϕ) does not appear explicitly in the Hamiltonian. Zhukov and Lebdev (9, p. 43) and Grodzovskii (10, p. 318) set P_ϕ equal to zero and neglect it. The same assumption is utilized in this study. These equations are integrated numerically along with the equations of motion, and they are an essential part of the two point boundary value problem.

The control functions are optimized if condition (1) in Theorem 2 is satisfied. The two optimal control functions for the solar sail are obtained using the following equation:

$$\frac{\partial H}{\partial u_i} = 0 \qquad i = 1, 2 \qquad (2.3.11)$$

where u_1 is the sail cone angle (θ), and u_2 is the sail clock angle (β). The partial derivative of the Hamiltonian with respect to the sail cone angle yields a second order polynomial in $\tan(\theta)$, after some algebraic manipulation:

$$2\left(P_{V_\phi}\sin(\beta) + P_{V_\psi}\cos(\beta)\right)\tan^2(\theta) + 3P_{V_r}\tan(\theta) - \left(P_{V_\phi}\sin(\beta) + P_{V_\psi}\cos(\beta)\right) = 0 \qquad (2.3.12)$$

A quadratic equation results which gives the expression for $\tan(\theta)$:

$$\tan(\theta) = \frac{-3P_{V_r} + \sqrt{9P_{V_r}^2 + 8(P_{V_\phi}\sin(\beta) + P_{V_\psi}\cos(\beta))^2}}{4(P_{V_\phi}\sin(\beta) + P_{V_\psi}\cos(\beta))} \qquad (2.3.13)$$

The positive root is used because it insures that the cone angle (θ) remain between the 0° and 90° limits imposed by the sail (the thrust vector cannot point towards the sun). The partial derivative of the Hamiltonian with respect to the sail clock angle (β) yields a much simpler equation:

$$\tan(\beta) = \frac{P_{V_\phi}}{P_{V_\psi}} \qquad (2.3.14)$$

The clock angle can also be calculated using the following equations, which are equivalent to equation (2.3.14):

$$\sin(\beta) = \frac{P_{V_\phi}}{\sqrt{P_{V_\phi}^2 + P_{V_\psi}^2}} \qquad (2.3.14a)$$

$$\cos(\beta) = \frac{P_{V_\psi}}{\sqrt{P_{V_\phi}{}^2 + P_{V_\psi}{}^2}} \qquad (2.3.14b)$$

The solar sail trajectory optimization program in Appendix B utilizes equations (2.3.14a) and (2.3.14b). It is important to note that the expressions for the control angles are functions of the auxiliary variables. Thus, the optimal control functions are dependent upon the initial values of the auxiliary variables.

The optimized trajectory problem is a two point boundary value problem because the position and velocity conditions at both endpoints of the trajectory must be satisfied. The spacecraft's initial position and velocity vectors are assumed to be equal to the earth's position and velocity vectors. Thus, the initial conditions for the equations of motion (2.2.5) - (2.2.10) are specified by the components of the earth's position and velocity vector. Since P_ϕ is equal to zero, equation (2.3.6) can be neglected. This leaves five auxiliary variable equations (2.3.5) and (2.3.7) - (2.3.10). The initial values of these variables are unknown and vary for different trajectories because the required control angles for each trajectory are different . In order to meet the rendezvous conditions, the spacecraft's final position and velocity vectors must match Eros's. Time minimization is assured by meeting the transversality condition (2) specified in Theorem 2, which states that the maximized Hamiltonian be greater or equal to zero at the final time. Pontryagin (13, p. 70) also says that, "optimal motions can always be accomplished in such a way that $H(y(t), x(t), u(t)) = 1$." Therefore, the transversality condition is assumed to be $H = 1$ at the final time. This assumption was also made by Sackett (14, p. 24). In order to solve this two point boundary value problem, the number of unknown initial values must equal the number of specified final conditions. Since seven final conditions must be met (three position components, three velocity components, and the transversality condition), and only five initial conditions are unknown (five auxiliary variables), two more unspecified

initial conditions are required. These conditions are the launch date and the time of flight. The initial conditions of the auxiliary variables, the launch date, and the time of flight are calculated using the method of steepest descent and Newton's method. Once the seven unknown initial conditions are found, the trajectory is completely specified.

3. IMPULSIVE PROPULSION SYSTEM MISSION DESIGN

3.1 Impulsive Propulsion System Trajectory Design

The impulsive program listed in Appendix B utilizes a modification of Gauss's method to solve Lambert's two body problem and generate possible Earth to Eros rendezvous trajectories in the 1996 - 2012 time frame. The modification to Gauss's method is outlined in chapter seven of Richard H. Battin's An Introduction to The Mathematics and Methods of Astrodynamics (15, ch 7). The program that appears in the Appendix is the version that contains the animated graphics routines. In order to find the minimum energy trajectories, this program had to be modified to assume a constant flight time and iterate launch dates, and to assume a launch date and a value for the total C3 (launch C3 + arrival C3) and use Newton's method to iterate time of flight. This program produces energy contours which are used to find the minimum energy mission. Total C3 is used because the mission is a rendezvous mission; thus, both the launch and arrival C3 must be minimized. A flyby mission only requires minimization of the launch energy. The minimum energy mission is important because it is also the least expensive mission. Since the Eros rendezvous mission is unmanned, cost and payload are the most important factors.

The first step in finding the mission launch window is identifying the dates when Earth and Eros are aligned in such a way that the trajectory between the two bodies requires the least amount of energy. One way to find the optimal launch dates is to use the program described in the previous paragraph that assumes a constant time of flight and steps the launch date through the time frame of interest. The results from this program are displayed in Figures 3.1.1 and 3.1.2. These figures show the variation of total C3 over the 1996 - 2012 time frame for constant flight times of 200, 300, 400, and 500 days. The three lowest total C3 missions occur in 1998, 2005, and 2012. The constant flight time curves of interest are the 200, 300, and 400 day curves because the 500 day flight time trajectories

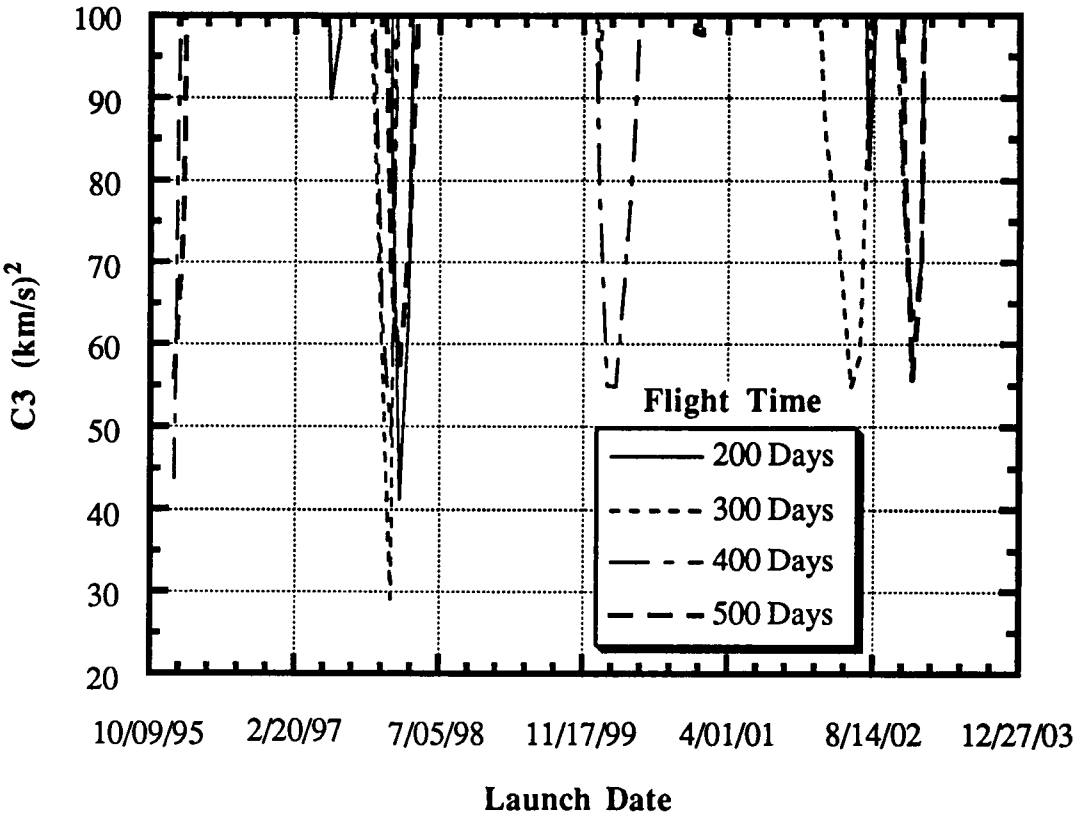

Figure 3.1.1: Variation of Total C3 with Launch Date for Various Flight Times Between 10/09/95 and 12/27/03

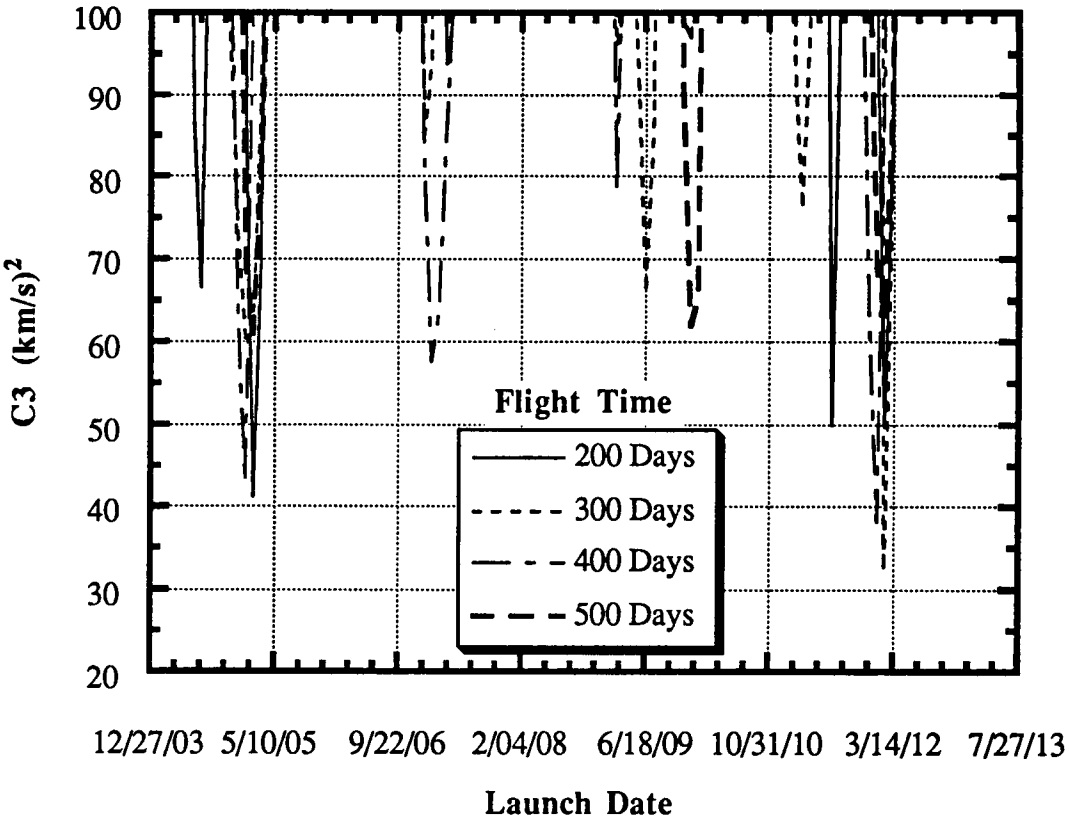

Figure 3.1.2: Variation of Total C3 with Launch Date for Various Flight Times Between 12/27/03 and 7/27/13

require a C3 of greater than 60 $(km/s)^2$. The constant flight time curves are periodic in time. The minimum energy launch dates occur every seven years. These figures identify the minimum energy launch dates for these specified flight times. Since they are based on assumed fixed flight times, they do not represent those flight times and launch dates that result in the minimum energy missions.

In order to find the minimum energy mission, total energy contour plots must be generated. These plots are created using the program described in the first paragraph that assumes a launch date and a value for the total mission C3, and iterates to find the flight time required. Figures 3.1.1 and 3.1.2 are used to find the possible launch dates and the Newton's method initial guesses for flight time. The total energy contours for the Eros rendezvous mission are illustrated in Figures 3.1.3, 3.1.4, and 3.1.5. The three energy contour figures correspond to the three minimum energy regions in Figures 3.1.1 and 3.1.2. As expected, the minimum energy flight times are somewhere between the 200 and 400 day flight times that are shown in Figures 3.1.1 and 3.1.2. The three contour plots illustrate the deceptive nature of the constant flight time curves. From Figure 3.1.2, the January 2005 mission appears to have a higher energy requirement than the January 1998 and January 2012. In actuality, Figure 3.1.4 shows that the energy requirement for January 2005 mission is as low or lower than the other two possible mission launch dates (Figures 3.1.3 and 3.1.5).

All of the energy contour plots have a concentric figure eight shape with the lower energy contours in the middle. The contours appear to all cross at a single point. This point represents the 180 degree transfer angle. The minimum energy trajectory between concentric circular orbits is the Hohmann transfer ellipse. The transfer angle for the Hohmann ellipse is 180 degrees and the ellipse is tangent to both circular orbits. The Hohmann transfer is not possible for non-circular orbits but the minimum energy principle applies. Figures 3.1.3, 3.1.4, and 3.1.5 show the minimum energy missions are clustered

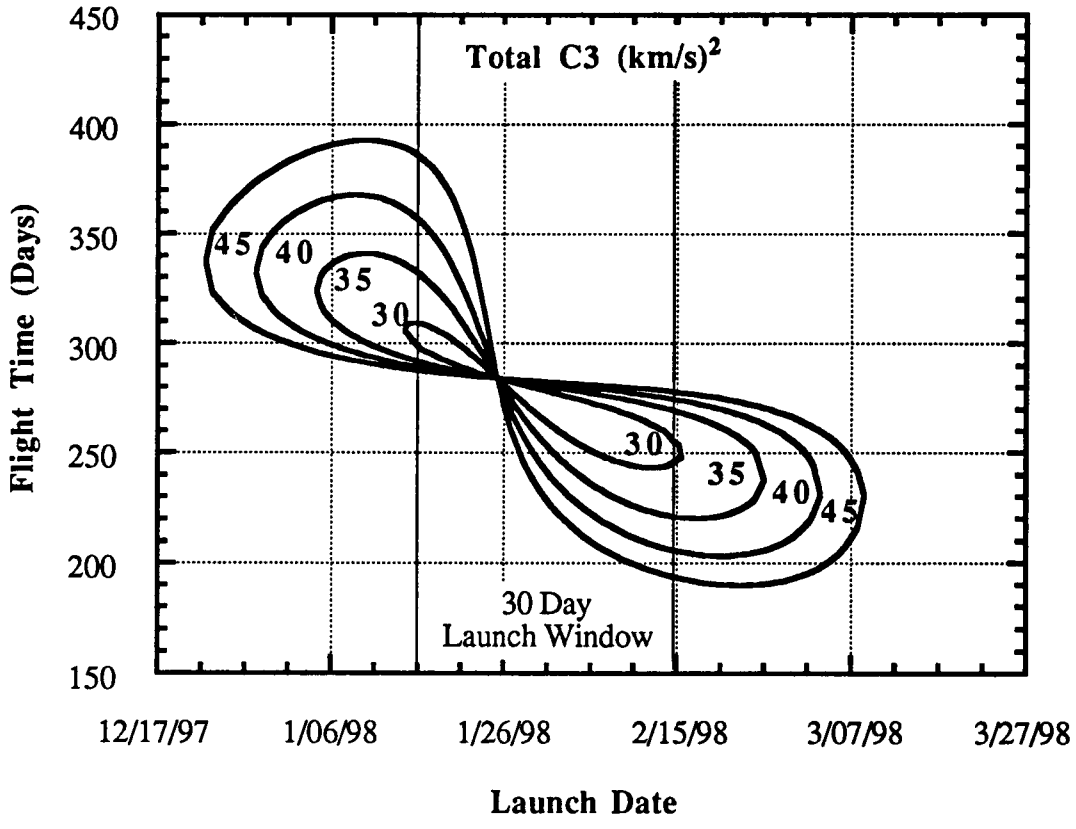

Figure 3.1.3: Total Energy Contours for the Eros Mission in 1998

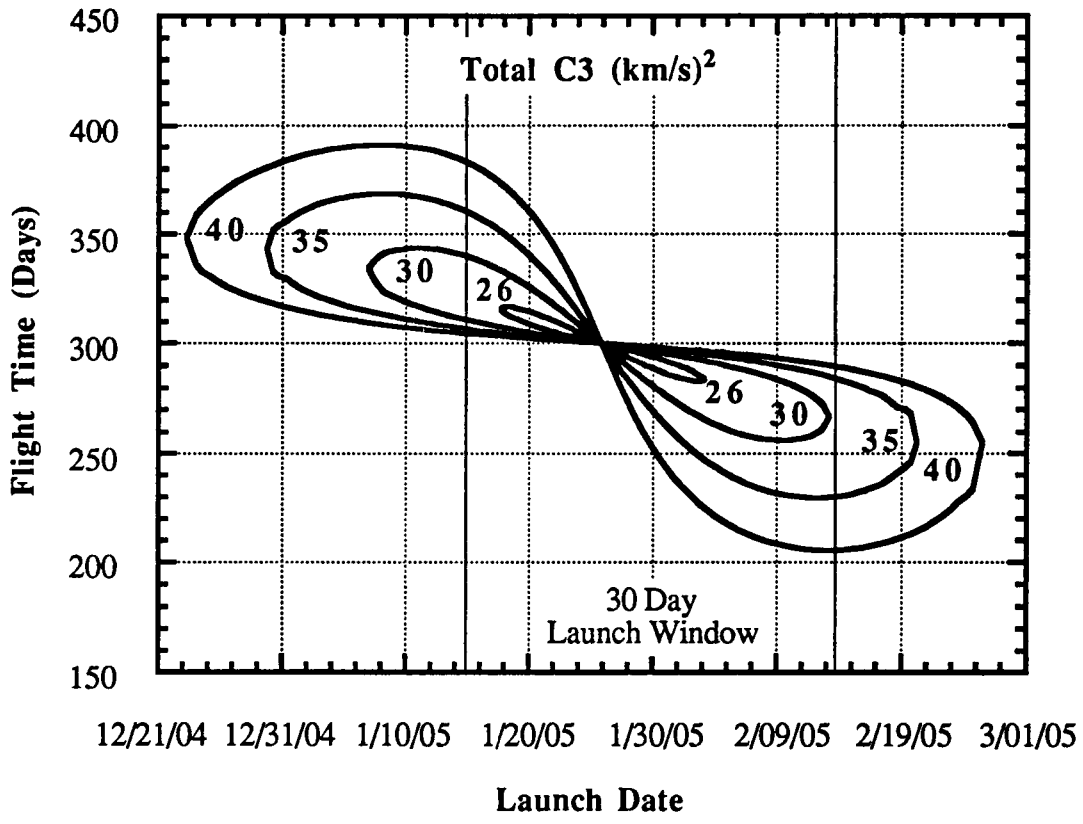

Figure 3.1.4: Total Energy Contours for the Eros Mission in 2005

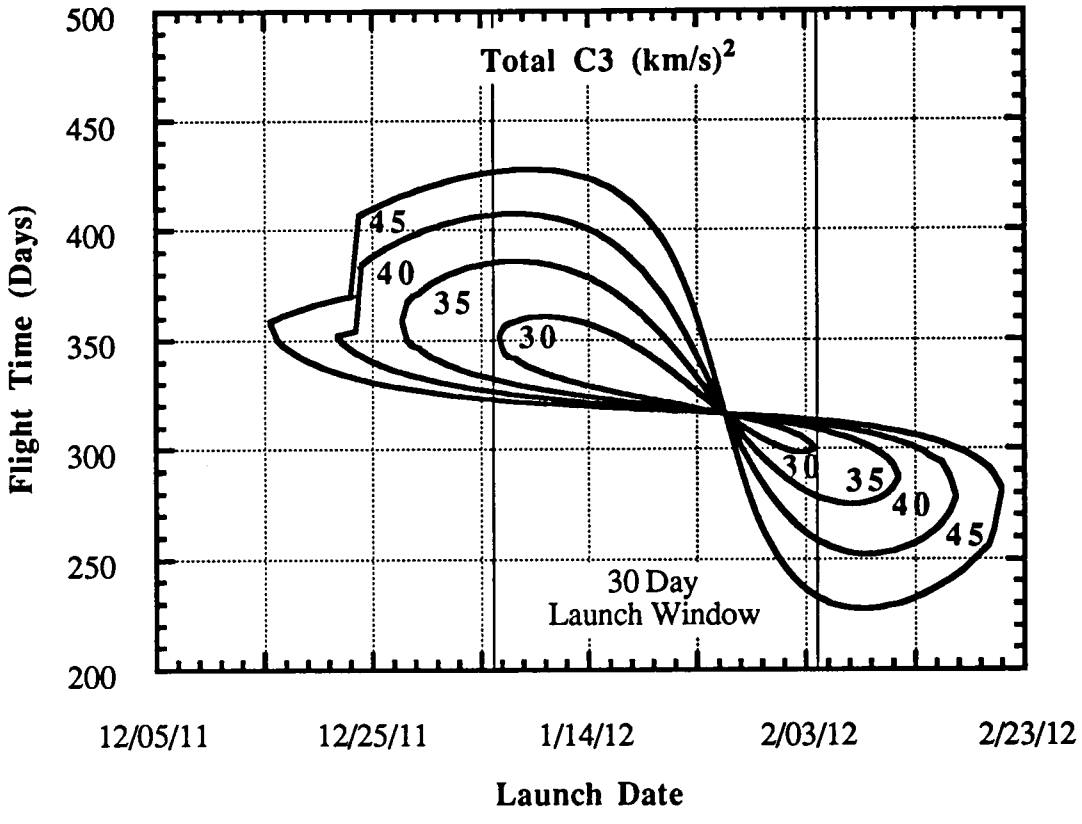

Figure 3.1.5: Total Energy Contours for the Eros Mission in 2012

around the 180 degree transfer angle. The required mission energies increase as the launch date moves away from the launch date associated with the 180 degree transfer. The contours to the left of the 180 degree point represent type II, or greater than 180 degree transfer angle trajectories. The contours to the right of the 180 degree point represent the type I, or less than 180 degree transfer angle trajectories. The type II trajectories require a longer flight time, which is also illustrated in Figures 3.1.3, 3.1.4, and 3.1.5.

Figure 3.1.3 was used to select the 1998 mission launch date and flight time. By selecting January 31, the total mission C3 will remain under 30 $(km/s)^2$ for the non-optimum dates in the 30 day launch window. The contour areas for the type I orbits are larger than the contour areas for the type II orbits; thus, the penalty associated with missing the optimal launch date is not as great for the type I orbits in 1998.

The contour areas are more balanced between the 2005 mission type I and type II orbits in Figure 3.1.4. A 291 day mission with a launch date of January 30, 2005 was chosen because the trajectory is a type I. In this case, the energy required for the type I and type II trajectories is the same; therefore, it makes sense to choose the lower flight time trajectory.

The 2012 mission energy contour plot (Figure 3.1.5), shows the type II trajectories to be better in terms of non-optimal launch date mission energy. Unlike Figure 3.1.3, the contour areas for the type II trajectories are larger than the contour areas for the type I trajectories. The January 20, 2012 launch date was chosen in order to center the 30 $(km/s)^2$ C3 contour in the 30 day launch window.

The energy requirements and the delta V requirements for the three possible missions are summarized in Table 3.1.1, and the 30 day launch window delta V requirements for each mission are listed in Table 3.1.2. The 2005 mission has the lowest total delta V requirement at 6.6392 km/s. The second lowest total delta V mission is the 1998 mission at 6.8726 km/s, and the highest total delta V mission is the 2012 mission at

Table 3.1.1: Eros Rendezvous Mission Trajectory Options

Launch Date	1/31/98	1/30/05	1/20/12
Flight Time (Days)	270	291	330
Launch C3 $(km/s)^2$	19.505	18.270	17.929
Launch ΔV (km/s)	4.0239	3.9707	3.9560
Parking Orbit Altitude (km)	500	500	500
Parking Orbit Inclination (Deg.)	45.321	37.070	33.643
Parking Orbit Ω (Deg.)	270	270	270
Arrival Date	10/28/98	11/17/05	12/15/12
Arrival C3 $(km/s)^2$	8.115	7.121	8.714
Arrival ΔV (km/s)	2.8487	2.6686	2.9519
Total ΔV (km/s)	6.8726	6.6393	6.9079

Table 3.1.2: Thirty Day Launch Window Data for the Eros Rendezvous Missions

Launch Date	1/31/98	1/30/05	1/20/12
Early Launch Date	1/16/98	1/15/05	1/05/12
Flight Time (Days)	305	320	350
Total ΔV (km/s)	6.8901	6.9202	7.4071
Late Launch Date	2/15/98	2/14/05	2/04/12
Flight Time (Days)	250	265	300
Total ΔV (km/s)	7.4048	7.0937	7.1404

6.9079 km/s. Since the total delta V is directly related to the cost of the mission, the 2005 mission is probably the least expensive. Most of the total mission energy is required to escape from the Earth. In fact, the launch energy is roughly twice the arrival energy in all cases. Table 3.1.2 illustrates the total delta V penalty associated with launching at the beginning or the end of the 30 day launch window. This penalty is on the order of hundreds of meters per second for all of the potential missions.

All of the missions start from a 500 km circular parking orbit. Because of the high inclination of Eros's orbit (10.832 degrees), the spacecraft cannot depart from the standard 28.5 degree inclination orbit without doing a plane change maneuver. Thus, in order to do a tangential injection, the inclination of the parking orbit must be increased. According to the International Reference Guide to Space Launch Systems (16, p. 205 and p. 268), the Delta launch vehicle and the Titan IV launch vehicle are capable of attaining a parking orbit inclination of 51 degrees. This value is used as the upper limit for the parking orbit inclination. If the inclination of the parking orbit must be greater than this value for tangential injection, the parking orbit inclination is assumed to be 51 degrees and the required plane change delta V is calculated. If tangential injection is possible for inclinations between 28.5 degrees and 51 degrees, the minimum possible parking orbit inclination is assumed. For all three missions the parking orbit inclination required for tangential injection is between the 28.5 and 51 degree limits. The specific values are listed in Table 3.1.1. Ordinarily, there are two possible parking orbits with the same inclination that enable tangential injection. These orbits differ in the location of the ascending node. Minimizing the required inclination causes these two orbits to be coincident with a longitude of the ascending node of 270 degrees measured relative to the projection of the Earth's velocity vector on the equatorial plane. The delta V at arrival is just the vector difference between the spacecraft and Eros. The spacecraft does not burn into a parking orbit because the asteroid is too small. The final burn is used to eliminate the relative

velocity between the spacecraft and the asteroid. All of the maneuvering on station is done with the attitude control thrusters.

The heliocentric conic data for the three trajectories is listed in Table 3.1.3. All of the trajectories are very similar. They all leave Earth when Earth is near its perihelion and arrive at Eros when it is near its aphelion. In addition, the rendezvous occurs near the ecliptic plane, minimizing the effect of Eros's large orbital inclination. The periodic alignment of Earth and Eros is evident in these results. An ecliptic projection of the spacecraft trajectory, and the orbits of Mercury, Venus, Earth, Mars, and Eros for each mission is presented in Figures 3.1.6, 3.1.7, and 3.1.8. The 1998 and the 2005 missions are characterized by the lower flight time type I trajectories, whereas, the 2012 mission is characterized by a higher flight time type II trajectory. Although the 1998 mission has the lowest flight time at 270 days, the 2005 mission is the best mission option because it has the lowest delta V requirement.

3.2 Spacecraft Mass Calculation

The initial mass in low Earth orbit is equal to the sum of the payload mass, the spacecraft bus mass, the propulsion system mass, and the required propellant mass. The payload mass and the spacecraft bus mass are taken from a paper entitled, "Mission Options for Rendezvous with the Most Accessible Near-Earth Asteroid - 1989 ML", by Jim V. McAdams (17, p. 983). Table 3.2.1 is the same as Table 3 in the aforementioned report. The sum of the payload mass and the spacecraft mass used in this study does not include the mass of the propulsion system or the mass of the propellant shown in Table 3.2.1; therefore, the sum of the payload and spacecraft masses are equal to 340 kg. The propulsion system mass and the propellant mass illustrated in Table 3.2.1 has no relevance to Eros rendezvous mission studied in this report so these values must be calculated. In order to find the best case for a chemical propulsion system based on LH_2/LO_2

Table 3.1.3: Heliocentric Conic Data for the Eros Rendezvous Missions

Launch Date	1/31/98	1/30/05	1/20/12
Initial Radius (A.U.)	0.985	0.985	0.985
Initial Celestial Longitude (Deg.)	130.401	128.675	117.792
Initial Celestial Latitude (Deg.)	0	0	0
a (A.U.)	1.390	1.384	1.390
Eccentricity	0.2916	0.2889	0.2922
Heliocentric Angle (Deg.)	171.09	174.01	189.33
Ω (Deg.)	310.401	308.675	297.792
Π (Deg.)	132.239	123.176	115.894
ω (Deg.)	181.838	174.501	178.102
Inclination (Deg.)	3.694	3.306	3.083
Final Radius (A.U.)	1.783	1.783	1.782
Final Celestial Longitude (Deg.)	301.505	302.693	307.107
Final Celestial Latitude (Deg.)	- 0.5720	-0.3450	- 0.4995

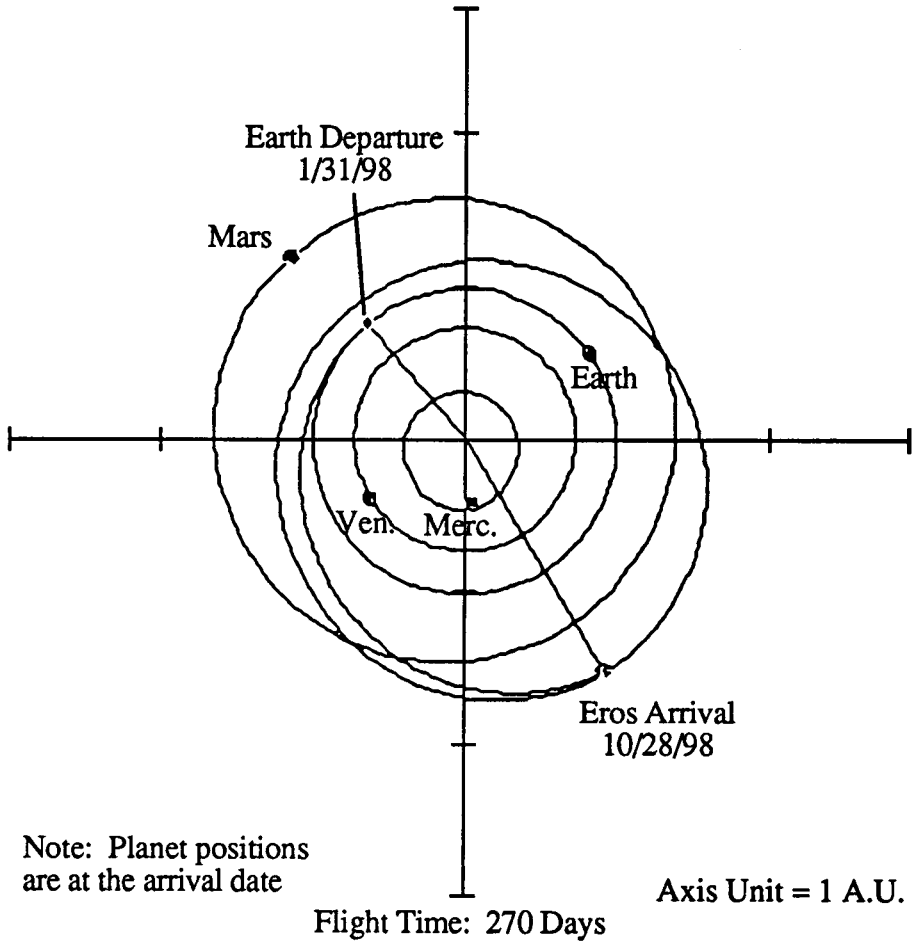

Figure 3.1.6: Earth to Eros Impulsive Trajectory in 1998

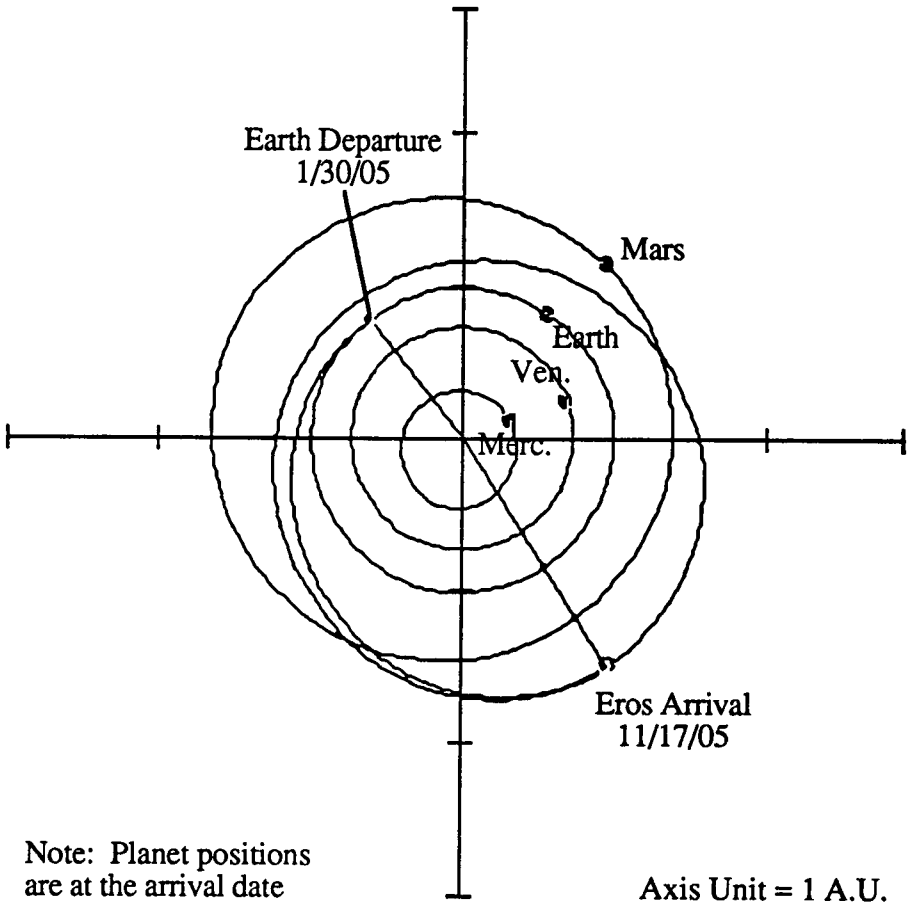

Figure 3.1.7: Earth to Eros Impulsive Trajectory in 2005

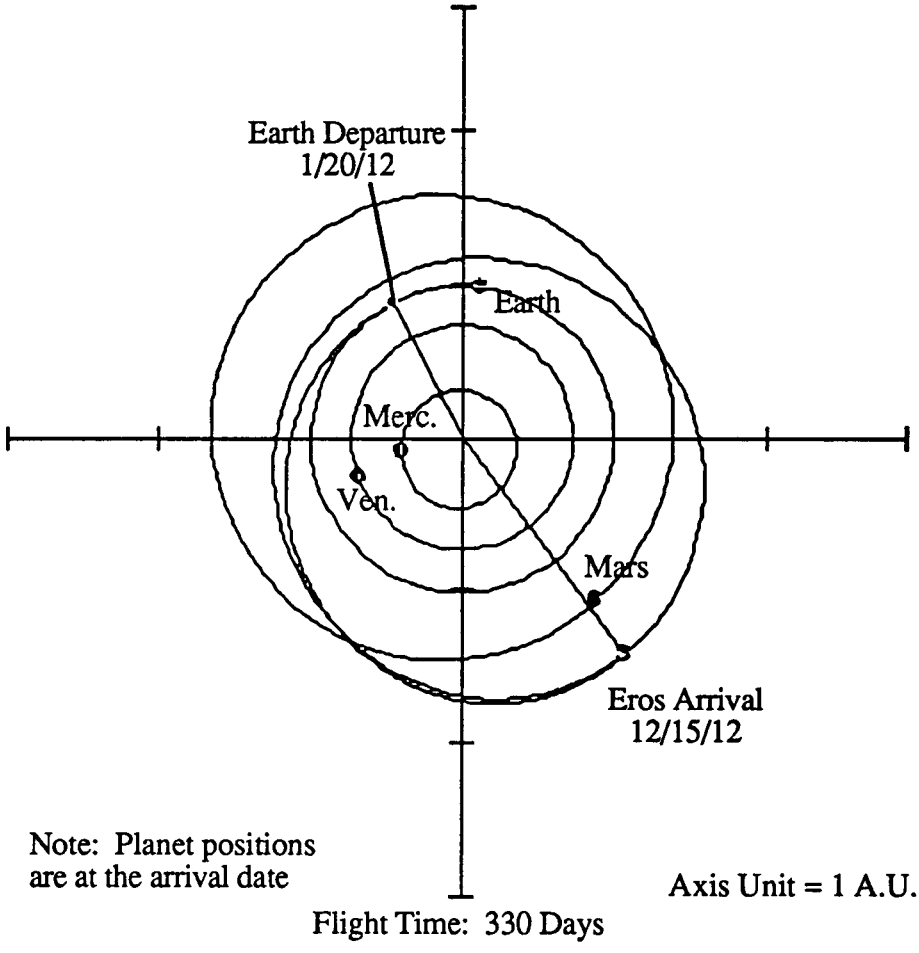

Figure 3.1.8: Earth to Eros Impulsive Trajectory in 2012

Table 3.2.1: NEAR Spacecraft System Mass Summary, Ref. 17

System	Mass (kg)
Instruments	60
Propulsion	100
Power	39
Radio Frequency	26
Attitude Control	25
Command & Data Handling	29
Thermal	10
Harness	18
Structure	76
Dry Mass	383
15% Contingency	57
Propellant	400
Total Injected Mass	840

propellants, a specific impulse (Isp) of 470 s is assumed. This is the same value assumed by Frisbee (4, p. 4). According to Wertz and Larson, "Liquid propulsion systems are typically 85 - 93% propellant by mass.", (5, p. 602). For the purposes of this study, the propellant is assumed to be approximately 85% of the propulsion system by weight. Equation 3.2.1 is used to calculate the propellant mass required for a given delta V:

$$\sigma = e^{(\Delta V / g I_{sp})} \tag{3.2.1}$$

where:

$$\sigma = \frac{M_o}{M_{dry}}$$

The propellant mass is equal to the difference between the mass before the burn (M_o) and the dry mass after the burn (M_{dry}). The chemical system is assumed to be two stage; thus, the first stage which is used to escape Earth is discarded after its burn. Using equation 3.2.1, and the required delta V's for the 2005 mission, the departure mass fraction (σ_{depart}) is equal to 2.3660, and the arrival mass fraction (σ_{arrive}) is equal to 1.7839. The mass of the second stage propulsion system and propellant are calculated first. In order to find an accurate propellant mass, the dry weight of the propulsion system must be assumed. This is done by calculating the propellant required to perform the delta V using a total dry mass equal to the 340 kg spacecraft and payload mass. The approximate propulsion system dry mass can then be calculated using the assumption that the propellant is 85% of the total propulsion system mass. These calculations approximate the propellant mass to be 267 kg and the propulsion system dry mass to be 47 kg. The actual propellant mass and propulsion system dry mass is greater than the values calculated because the propulsion system dry mass is not included in the 340 kg. In order to get more accurate results, the stage 2 propulsion system dry mass is assumed to be 100 kg; thus the mass of the spacecraft after the burn (M_{dry}) is 440 kg. From Equation 3.2.1, the mass before the burn (M_o) is 785 kg. The propellant mass required is 345 kg, which is 78% of the total

propulsion system mass of the second stage. According to Wertz and Larson, "the fuel mass fraction can be considerably lower in small systems." (5, p. 602). The second stage is a small system so the lower mass fraction is acceptable. A similar procedure is used to find the propellant mass and the dry engine mass required for the Earth escape burn. In this case, the mass before the second burn ($M_o = M_{dry} = 785$ kg) is used to find the initial approximation for the propellant mass. The initial approximation for the propulsion system dry mass is 189 kg. From this value, the propulsion system dry mass is assumed to be 250 kg; thus the spacecraft dry mass after the burn is actually equal to 1035 kg. Using the departure mass ratio (σ_{depart}) the mass of the spacecraft before the burn is 2445 kg. The propellant mass required is 1414 kg which is 85% of the total propulsion system mass. The propellant required for the mission is 1759 kg. An additional 15% safety margin and a 1% residual are assumed. Thus, the total required propellant is 2040 kg. The IMLEO for the chemical mission is 2730 kg. The mass summary appears in Table 3.2.2.

Table 3.2.2: Chemical System Minimum Energy Mission to Eros Mass Summary

Component	Mass (kg)
Spacecraft Bus and Payload	340
Stage 1 Dry Propulsion System	250
Stage 1 Propellant[1]	1414
Stage 2 Dry Propulsion System	100
Stage 2 Propellant[2]	345
Propellant Margin and Residual	281
Chemical Spacecraft IMLEO	2730

[1] Stage 1 propellant is 85% of the total stage 1 propulsion system

[2] Stage 2 propellant is 78% of the total stage 2 propulsion system

4. SOLAR SAIL MISSION DESIGN

4.1 Solar Sail with a Characteristic Acceleration of 1 mm/s^2

The solar sail trajectory optimization theory described in Section 2.3 was utilized by the low thrust trajectory optimization program in Appendix B to generate minimum time trajectories for an Earth to Eros mission in the 1996 - 2012 time frame. The solar sail mission design was conducted for two values of the characteristic acceleration, 1 mm/s^2 and 2 mm/s^2. This section deals with the 1 mm/s^2 characteristic acceleration results. The 2 mm/s^2 characteristic acceleration results will be presented in the next section.

During the 1996 - 2012 time frame, there are several possible Earth to Eros mission opportunities. These mission opportunities are determined by the favorable locations of Earth and Eros at the launch and arrival dates respectively. For circular and coplanar orbits, only one minimum time trajectory exists. If the orbits of the Earth and Eros are assumed to be circular and coplanar, the launch dates for the rendezvous mission occur whenever the Earth and Eros are separated by the change in celestial longitude specified by the minimum time trajectory. Thus, once the minimum time trajectory between the concentric circular orbits is calculated, rendezvous mission launch dates are obtained by fixing the starting point of the trajectory to Earth's position and changing the Earth's position until the end point of the trajectory matches Eros's position. As described in Section 1.3, the circular, coplanar trajectory solution is used as a first approximation to the trajectory optimization solution between eccentric, coplanar orbits, which is subsequently used as the first approximation to the fully three-dimensional trajectory optimization solution. The initial guesses for launch dates are determined by the proper angular separation between Earth and Eros, which occurs approximately every two and a half years. Over the 1996 - 2012 time frame there are seven local minimum time mission opportunities. The launch dates and flight times for the three-dimensional Earth to Eros

Table 4.1.1: Minimum Time Mission Opportunities for a Solar Sail with a Characteristic Acceleration of 1 mm/s^2

Calendar Launch Date	Julian Launch Date	Time of Flight (Days)
2/02/98	2450846	520
3/29/00	2451632	471
6/02/02	2452427	397
1/24/05	2453394	518
3/17/07	2454177	484
5/19/09	2454970	408
1/13/12	2455939	508

rendezvous missions are presented in Table 4.1.1. If the orbits of Earth and Eros were circular and coplanar, the flight times for the different mission opportunities would be the same. Because the orbits of Earth and Eros are eccentric and do not lie in the same plane, some launch dates result in a more favorable alignment between the two bodies. The different Earth - Eros alignments occur periodically. As shown in Table 4.1.1, similar mission flight times occur every seven years. There are three possible Earth to Eros mission geometries: a January launch and 500 day flight time, a March launch and 480 day flight time, and a May launch and 400 day flight time. If the launch costs of these missions are the same, the 400 day flight time missions are the best mission opportunities because shorter flight times can be realized at no additional cost.

The solar sail's initial position, velocity, and control angle settings are listed for each mission opportunity in Table 4.1.2, and the initial conditions for the auxiliary variables are listed for each mission opportunity in Table 4.1.3. The initial conditions for

Table 4.1.2: Initial Conditions of the State Variables and Control Angles for the 1 mm/s^2 Characteristic Acceleration Solar Sail Missions

Launch Date	Radius (A.U.)	ϕ (Deg.)	V_r (km/s)	V_ϕ (km/s)	θ (Deg.)	β (Deg.)
2/02/98	0.9852	131.2596	0.2364	30.2271	35.8669	147.6930
3/29/00	0.9980	187.1875	0.4952	29.8389	30.4512	143.2829
6/02/02	1.0139	249.9856	0.2711	29.3716	28.8712	9.5353
1/24/05	0.9842	122.1258	0.1628	30.2593	35.6310	143.6717
3/17/07	0.9946	175.0496	0.4733	29.9429	32.9550	149.6358
5/19/09	1.0115	237.1848	0.3575	29.4428	32.3405	4.0116
1/13/12	0.9834	110.8002	0.0663	30.2822	35.4098	139.1422

Table 4.1.3: Initial Conditions of the Auxiliary Variables for the 1 mm/s^2 Characteristic Acceleration Solar Sail Missions

Launch Date	P_r	P_ψ	P_{Vr}	$P_{V\phi}$	$P_{V\psi}$
2/02/98	3.8970	-2.3727	-0.3373	8.6016	-13.6028
3/29/00	3.6992	-19.2446	2.2486	7.6795	-10.2965
6/02/02	1.8638	-13.6521	2.8361	1.9826	11.8029
1/24/05	4.6046	0.5793	-0.1999	9.2761	-12.6149
3/17/07	3.0563	-16.0184	1.1843	7.3027	-12.4649
5/19/09	4.0116	-18.1592	4.2253	5.0492	7.9746
1/13/12	5.3245	3.8422	-0.0771	9.9392	-11.4912

the position and velocity of the solar sail are assumed to be equal to the Earth's position and velocity on the launch date. The initial values for the celestial latitude (ψ) and the velocity in the direction normal to the ecliptic plane (V_ψ) are not included in Table 4.1.2 because their initial value is always equal to zero. Since the equatorial plane of the spherical coordinate system used in this analysis is defined by the ecliptic plane, the solar sail never has an initial celestial latitude or an initial component of velocity in the direction normal to the ecliptic plane. The earth's orbit is nearly circular; thus, the initial conditions for the radius and the tangential velocity component do not change much from mission to mission. The radial velocity components are always positive, so the mission launch dates are all after the Earth passes its perihelion.

The solar sail's final position and velocity are listed for each of the mission opportunities in Table 4.1.4. According to Carl Sauer, "This large flight time difference

Table 4.1.4: Final Conditions of the State Variables for the 1 mm/s^2 Characteristic Acceleration Solar Sail Missions

Launch Date	Radius (A.U.)	ϕ (Deg.)	ψ (Deg.)	V_r (km/s)	V_ϕ (km/s)	V_ψ (km/s)
2/02/98	1.2597	60.0568	9.7934	-5.0394	27.7410	-2.2569
3/29/00	1.1422	139.6465	-2.8628	1.6397	30.1888	-5.5686
6/02/02	1.3259	201.0893	-10.5517	5.5187	26.4205	-1.1456
1/24/05	1.3422	41.5361	10.7502	-5.5718	26.1183	-0.6035
3/17/07	1.1336	126.8205	-0.4446	0.3793	30.3787	-5.8077
5/19/09	1.2681	188.0067	-9.7177	5.1283	27.5515	-2.3176
1/13/12	1.4611	19.3669	10.4654	-5.4861	23.9679	1.1772

illustrates the fact that as the eccentricity of the target body increases, it becomes increasingly easier to rendezvous with a target before aphelion and more difficult to rendezvous after aphelion." (11, p. 5). The final radial velocity is an indicator of the location of the target body relative to its aphelion at rendezvous. A negative final radial velocity indicates the rendezvous occurs after aphelion, whereas a positive final radial velocity indicates the rendezvous occurs before aphelion. As illustrated in Table 4.1.4, the missions launched in 1998, 2005, and 2012 are characterized by a rendezvous after aphelion. The other missions are characterized by their rendezvous before aphelion. From Table 4.1.1, the flight times for the rendezvous after aphelion missions are greater than the before aphelion rendezvous missions. There is also a significant difference in flight time between the various before aphelion rendezvous missions. By comparing the flight times in Table 4.1.1 with the final radii in Table 4.1.4, it appears that the flight time increases the closer the target body is to its perihelion. The mission with the lowest time of flight is launched on 6/02/02. The rendezvous with Eros occurs at a radius of 1.3259 A.U.. The mission with the second lowest time of flight is launched on 5/19/09. The rendezvous occurs at a radius of 1.2681 A.U.. Since the rendezvous of the 2009 mission occurs at a smaller radius than the 2002 mission, the 2009 mission rendezvous is closer to Eros's perihelion. The rendezvous of the 2000 and 2007 missions occur even closer to Eros's perihelion (1.1422 and 1.1336 respectively). The higher flight times are required for the near perihelion rendezvous missions because the rendezvous velocity requirements are more difficult to obtain. In order for the solar sail to obtain the fairly high rendezvous velocities, the sail must spiral outside the orbit of the target body before the rendezvous. As the sail spirals out, the tangential velocity is sacrificed in order to increase the radial velocity and the radius. By the time the sail gets to the target body's radius, the sail's tangential velocity is too slow to rendezvous with the target; thus, the sail spirals outside the target's orbit and increases its tangential velocity by spiraling back in. The flight times

of the near perihelion rendezvous missions are greater than the missions that rendezvous closer to the aphelion of Eros because the sail must spiral farther outside the target body's orbit. This behavior is analyzed in more detail in the next two paragraphs.

The variation of the solar sail's position, velocity, control angles, and auxiliary variables with flight time for the primary and secondary missions are illustrated in Figures 4.1.1 - 4.1.6 and 4.1.7 - 4.1.12 respectively. The corresponding graphs for each mission look very similar, indicating that the overall mission geometries are similar. In both missions, the radius increases, decreases, and increases again (Figures 4.1.1 and 4.1.7). This trend is also exhibited by the radial velocity (Figures 4.1.4 and 4.1.10). The tangential velocity behaves in the opposite manner (Figures 4.1.4 and 4.1.10). As the radial velocity and radius increases in the early part of the trajectories, the tangential velocity decreases. According to Figures 4.1.5 and 4.1.11, the cone angle remains fairly constant over the early part of the trajectories, and the clock angle varies from a small positive value to small negative values, which indicates that almost all of the early acceleration is in the positive radial and normal directions. As expected, the celestial latitude increases over the initial portion of the trajectories (Figures 4.1.3 and 4.1.9). It is interesting to note that the sail initially travels out of the ecliptic plane in the opposite direction from the final rendezvous celestial latitude. In the middle portion of the trajectories, the sail begins to spiral inward in order to increase its tangential velocity. Both the cone and clock angles increase (Figures 4.1.5 and 4.1.11), causing the radius (Figures 4.1.1 and 4.1.7) and the radial velocity (Figures 4.1.4 and 4.1.10) to decrease. As the tangential velocity increases, the clock angle continues to increase past 90 degrees and the cone angle begins decreasing again (Figures 4.1.5 and 4.1.11). This causes the normal velocity (Figures 4.1.4 and 4.1.10) and the celestial latitude (Figures 4.1.3 and 4.1.9) to decrease. During the later portion of the trajectories, the cone angle remains fairly constant

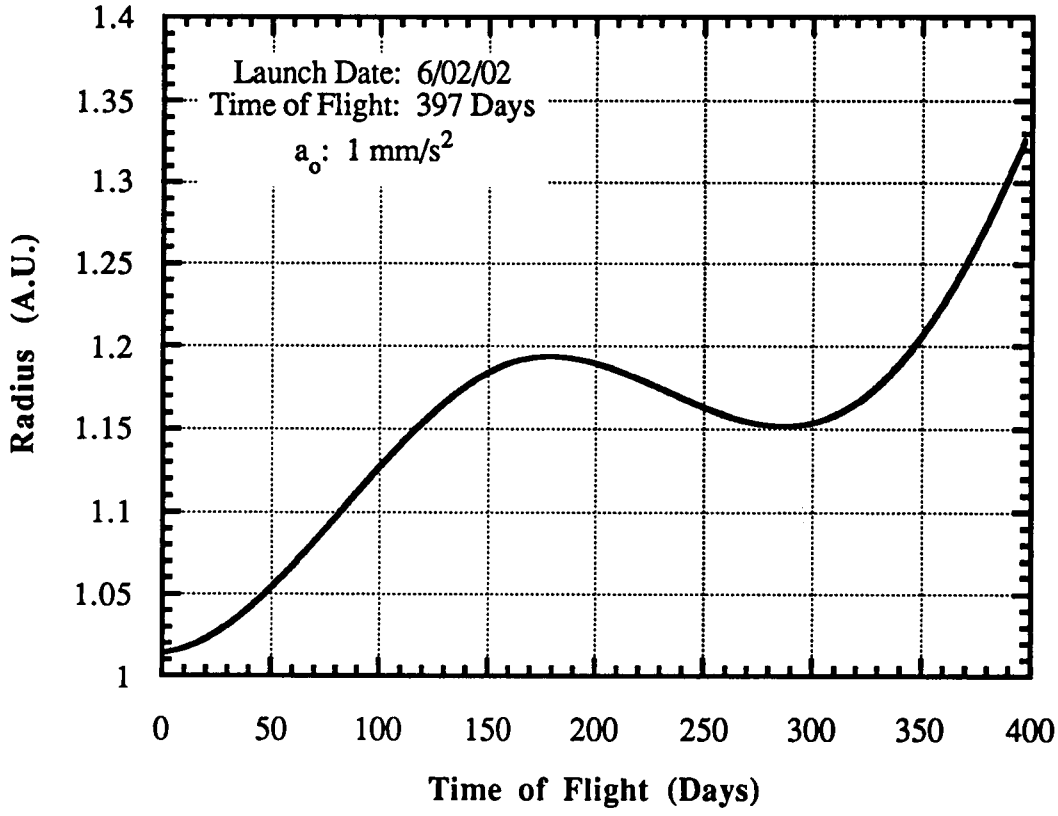

Figure 4.1.1: Solar Sail Radius History for the Primary Mission

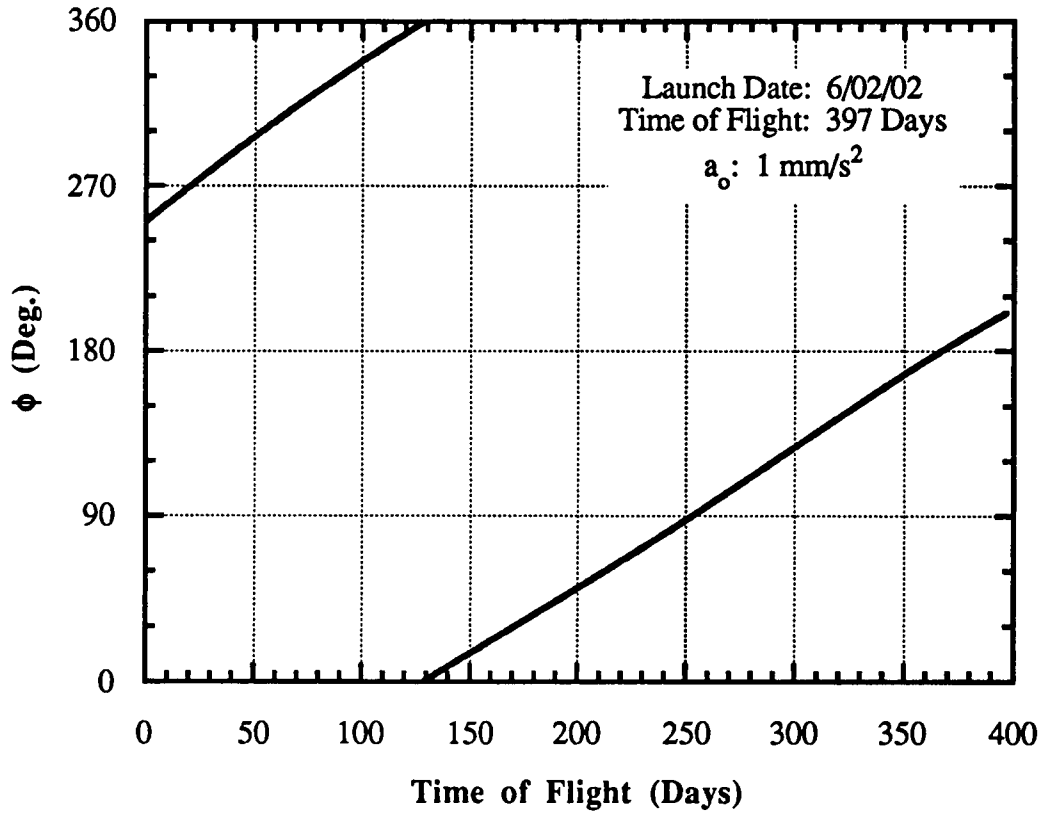

Figure 4.1.2: Solar Sail Celestial Longitude History for the Primary Mission

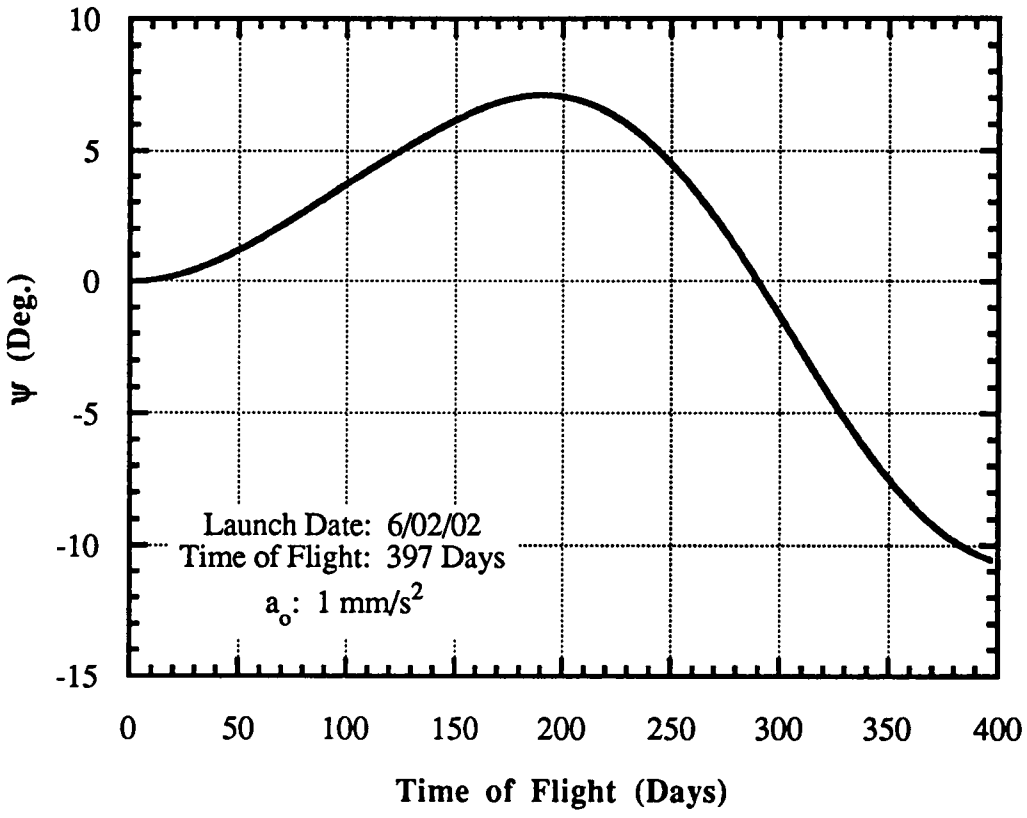

Figure 4.1.3: Solar Sail Celestial Latitude History for the Primary Mission

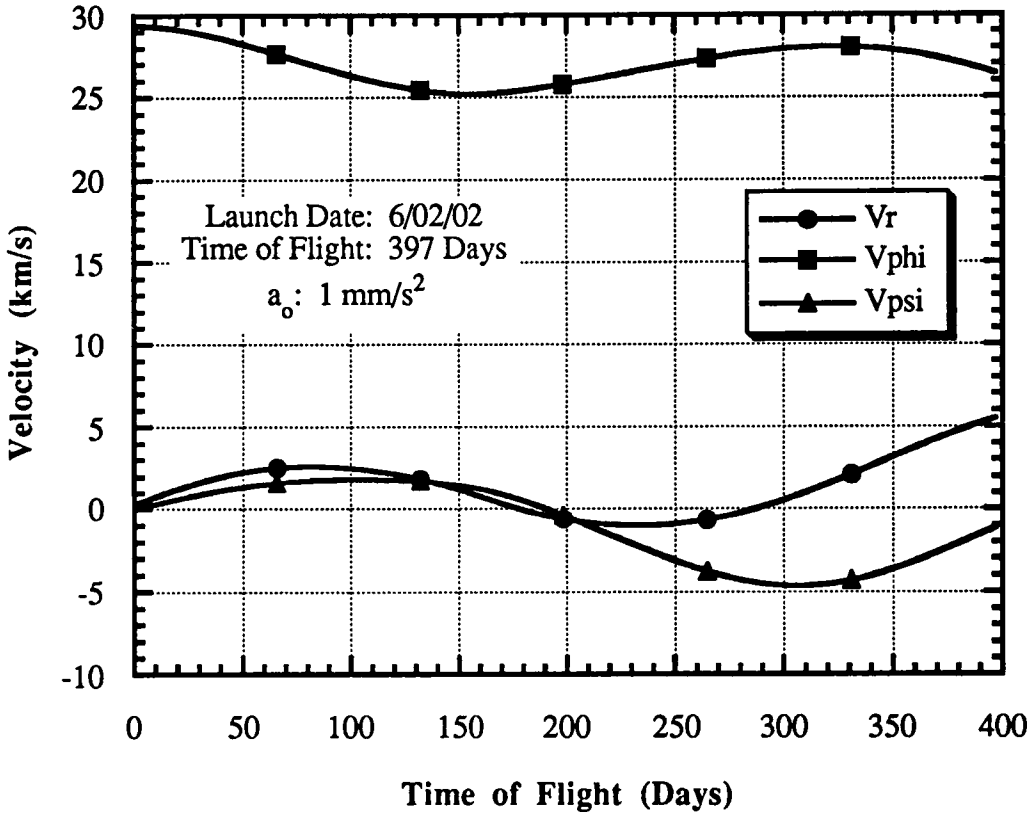

Figure 4.1.4: Solar Sail Velocity Component Histories for the Primary Mission

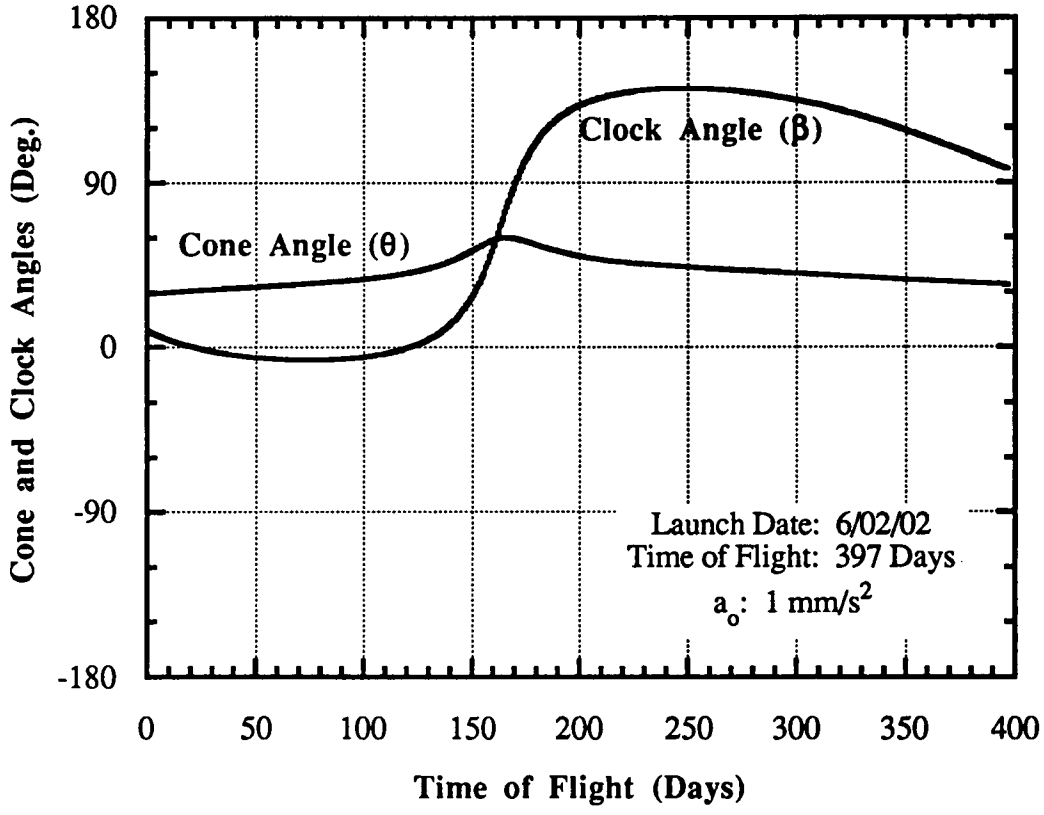

Figure 4.1.5: Solar Sail Control Angle Histories for the Primary Mission

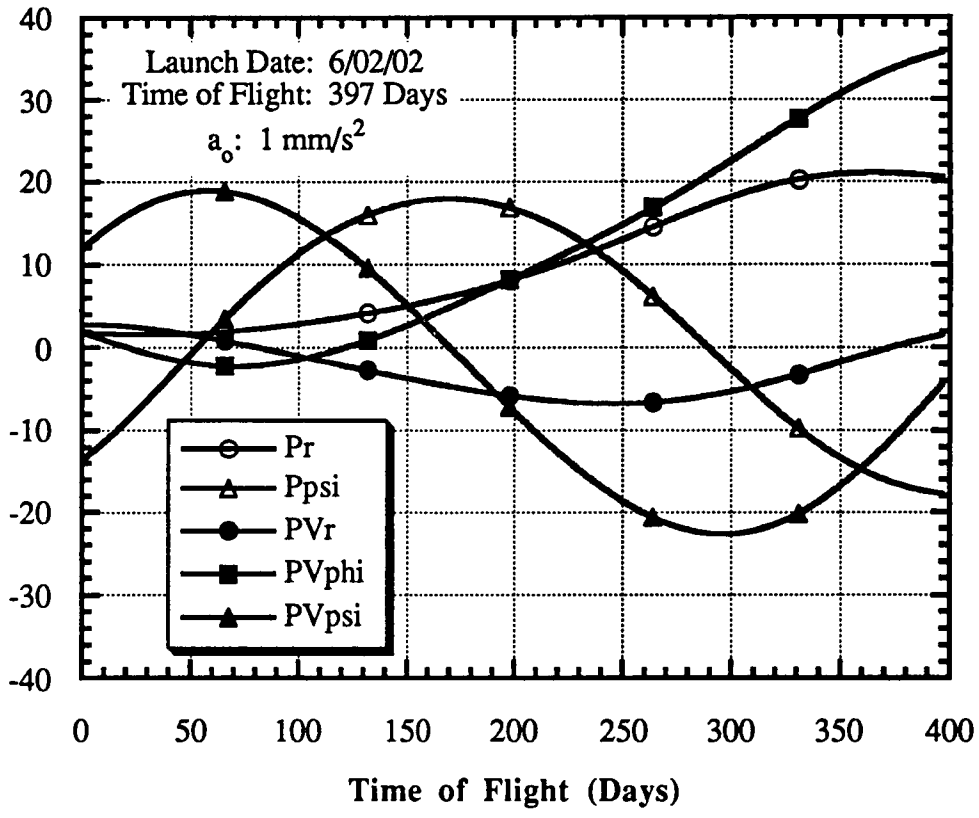

Figure 4.1.6: Solar Sail Auxiliary Variable Histories for the Primary Mission

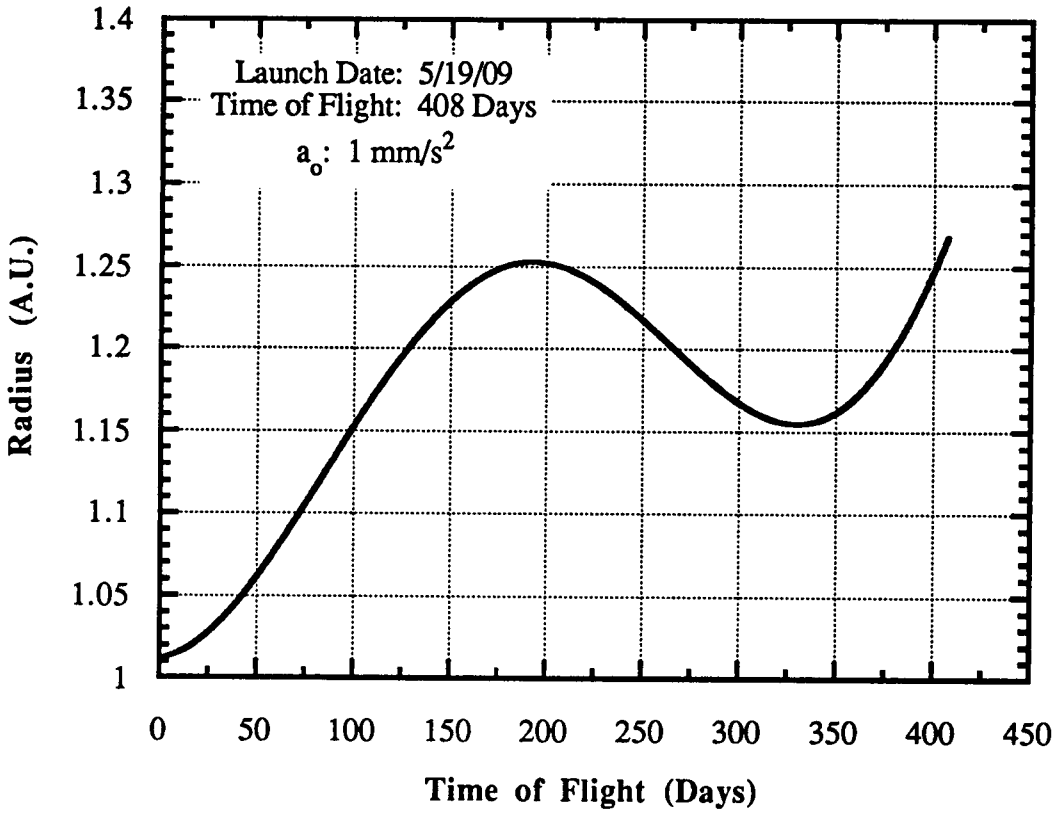

Figure 4.1.7: Solar Sail Radius History for the Secondary Mission

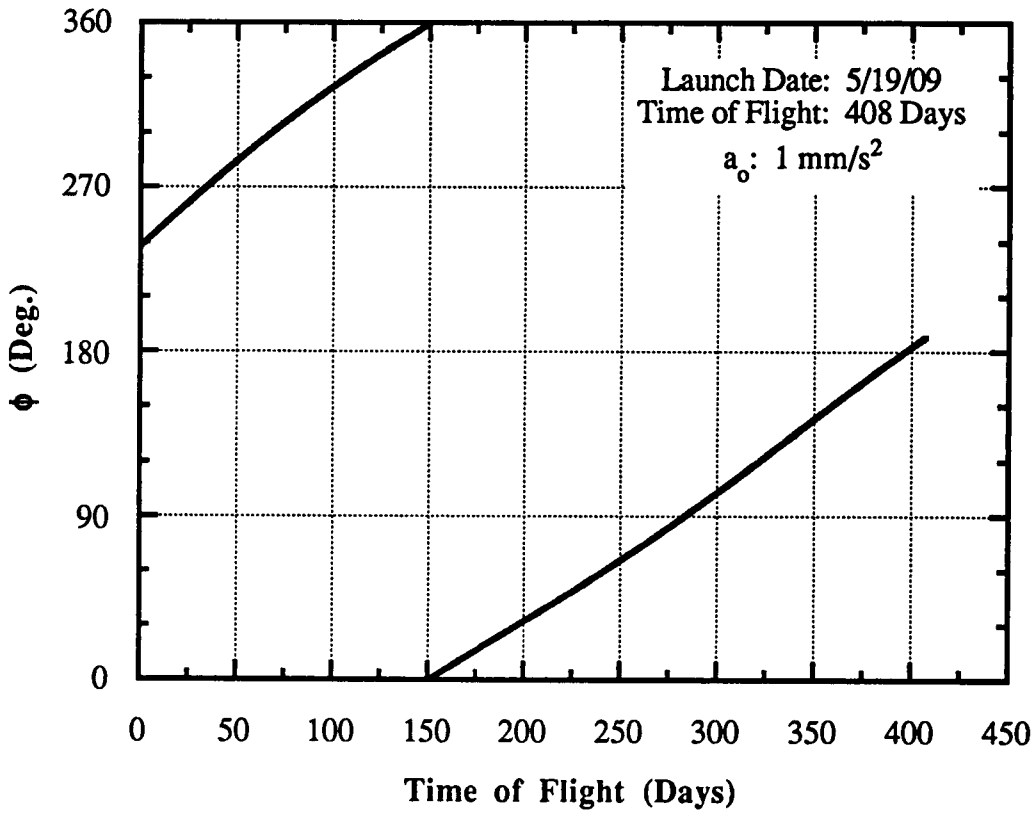

Figure 4.1.8: Solar Sail Celestial Longitude History for the Secondary Mission

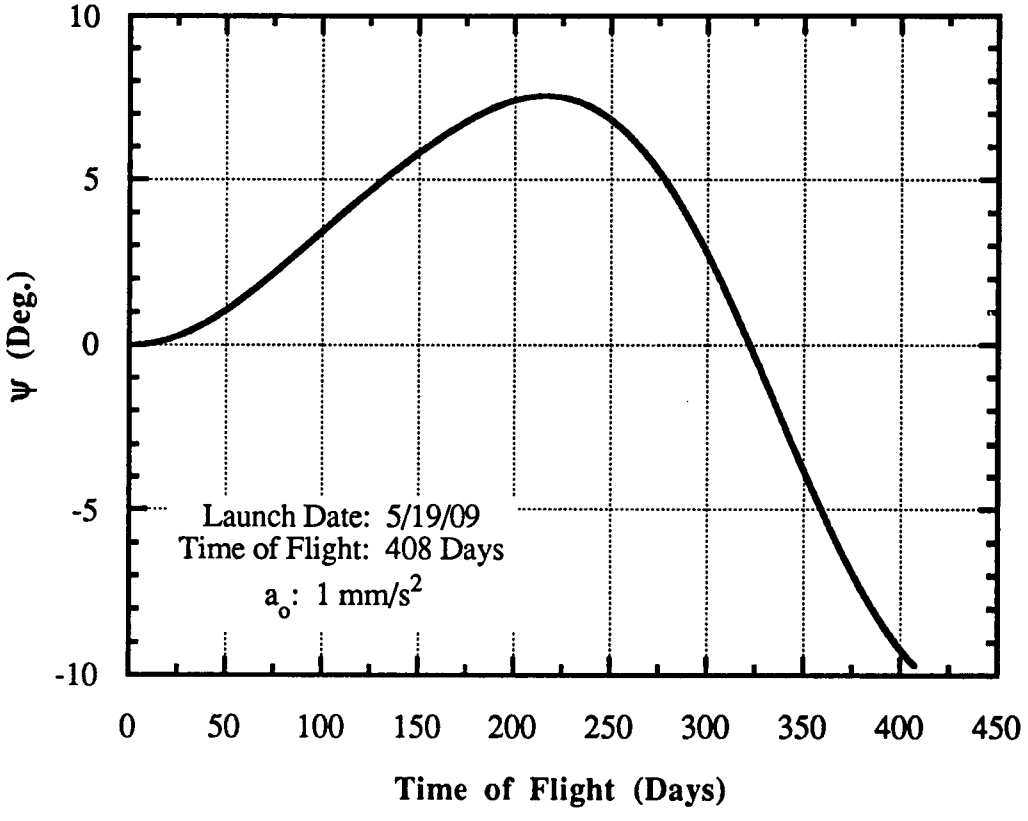

Figure 4.1.9: Solar Sail Celestial Latitude History for the Secondary Mission

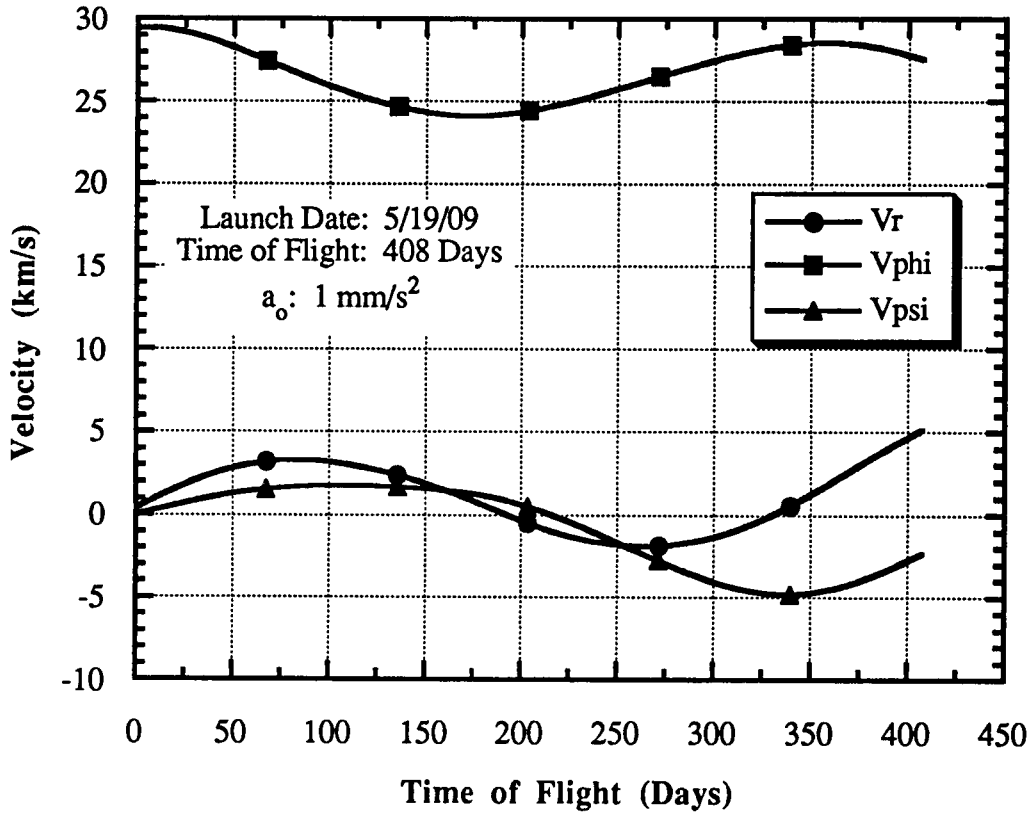

Figure 4.1.10: Solar Sail Velocity Component Histories for the Secondary Mission

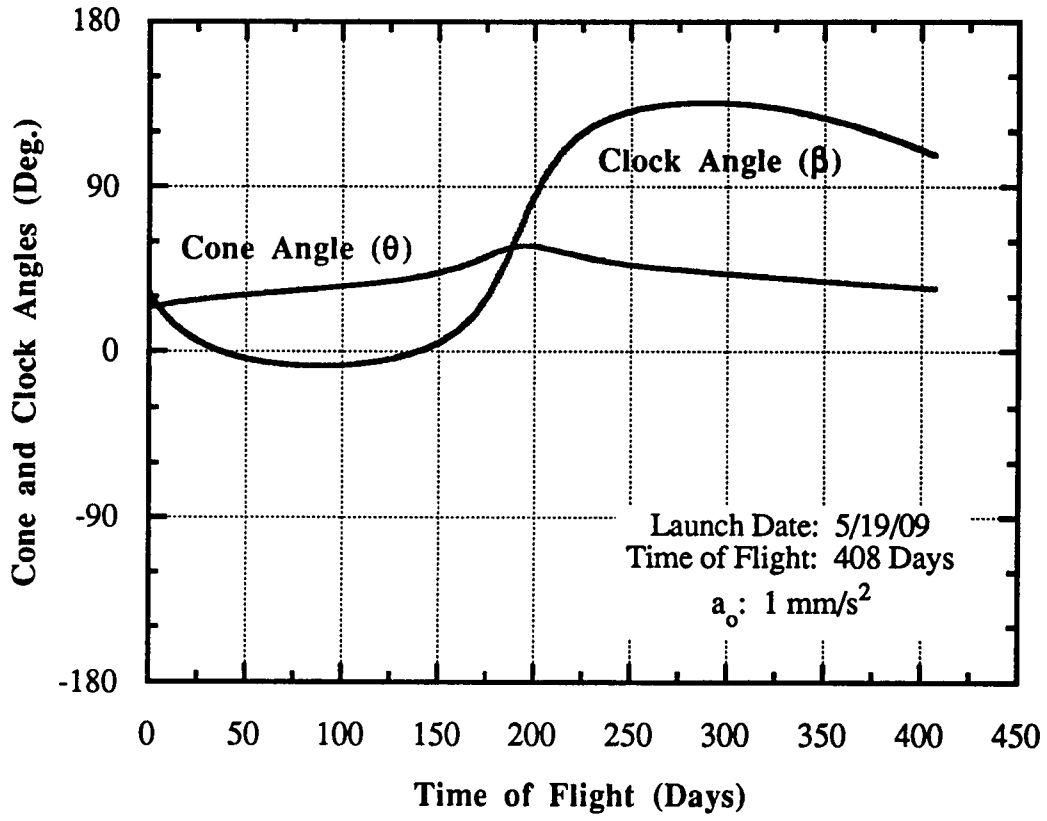

Figure 4.1.11: Solar Sail Control Angle Histories for the Secondary Mission

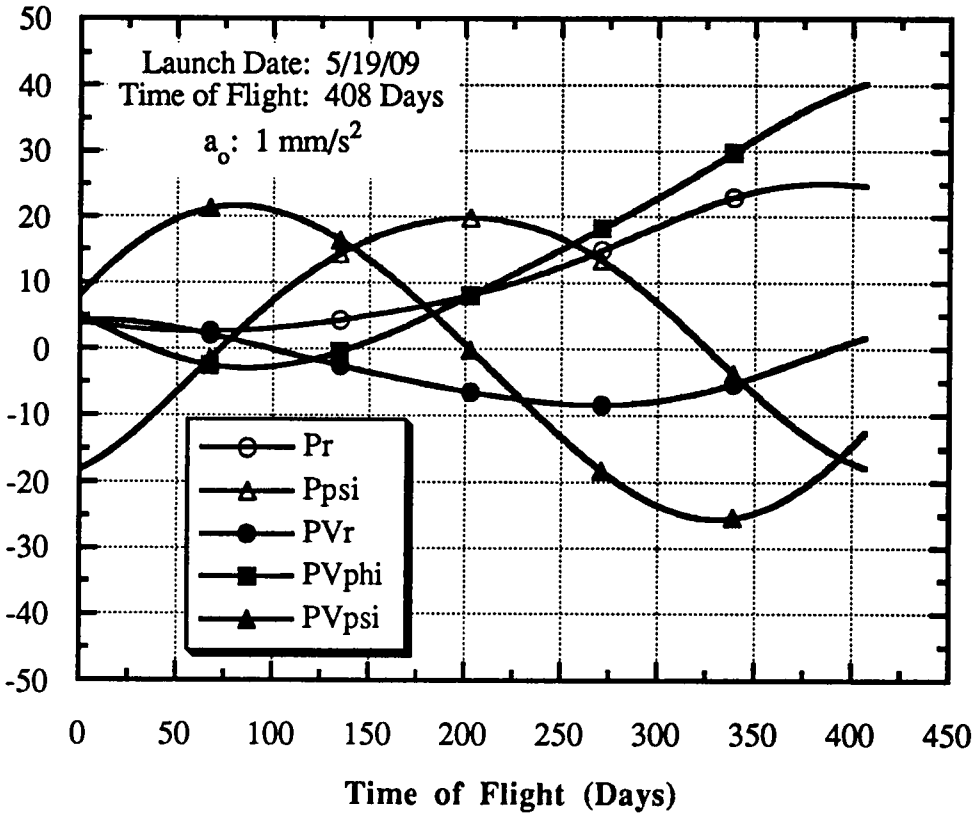

Figure 4.1.12: Solar Sail Auxiliary Variable Histories for the Secondary Mission

and the clock angle reaches a maximum and then decreases again (Figures 4.1.5 and 4.1.11). This causes the radius (Figures 4.1.1 and 4.1.7) and radial velocity (Figures 4.1.4 and 4.1.10) to increase again.

There are some differences between the primary and secondary trajectories. The differences that have been discussed to this point are the 11 day difference in flight time and the rendezvous location relative to the perihelion of the target body. As stated earlier, flight times of the near perihelion rendezvous missions are greater than the missions that rendezvous closer to the aphelion of Eros because the sail must spiral farther outside the target body's orbit. The differences in flight time between the primary and secondary missions can be explained by this phenomena. According to Figures 4.1.1 and 4.1.7, the sail travels farther away from the sun before spiraling back in to increase its tangential velocity in the secondary mission. The tangential velocity change over the duration of the trajectory is also slightly greater in the secondary mission (Figures 4.1.4 and 4.1.10).

An ecliptic projection of the primary and secondary mission trajectories are illustrated in Figures 4.1.13 and 4.1.14 respectively. These Figures also show the Earth's orbit and the ecliptic projection of Eros's orbit. As described earlier, the sail travels outside Eros's orbit before rendezvous in both cases. In addition, the rendezvous occurs closer to Eros's perihelion in the secondary mission (Figure 4.1.14).

4.2 Solar Sail with a Characteristic Acceleration of 2 mm/s^2

This section presents the results generated by the low thrust trajectory optimization program, assuming a solar sail with a characteristic acceleration of 2 mm/s^2. The increased characteristic acceleration is achieved by either increasing the sail area, decreasing the payload mass, or decreasing the sail mass. Since the payload mass is specified by the mission and the sail material is assumed to be the same for the two sails, the characteristic

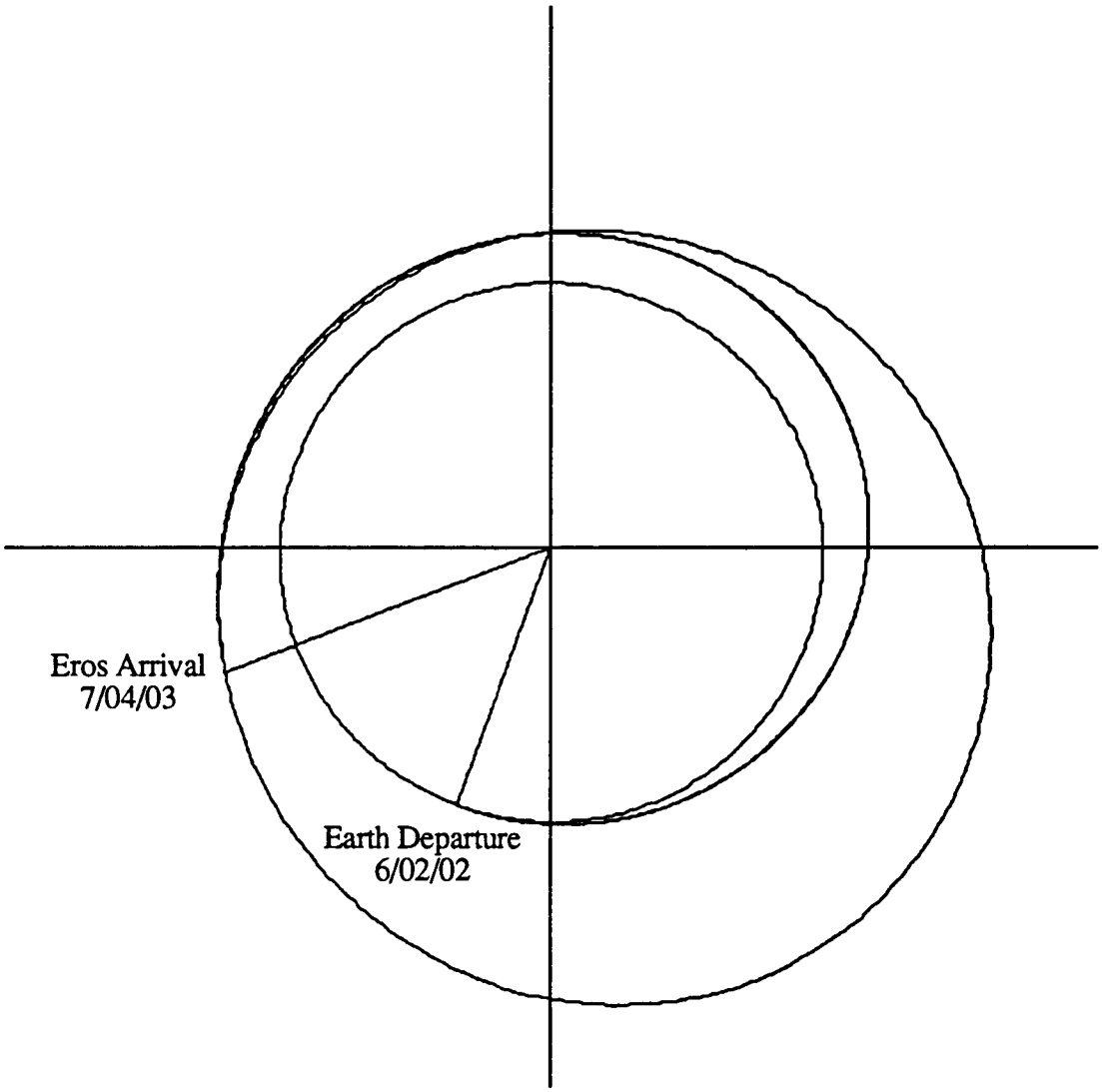

Figure 4.1.13: Ecliptic Projection of the Primary Eros Mission Trajectory

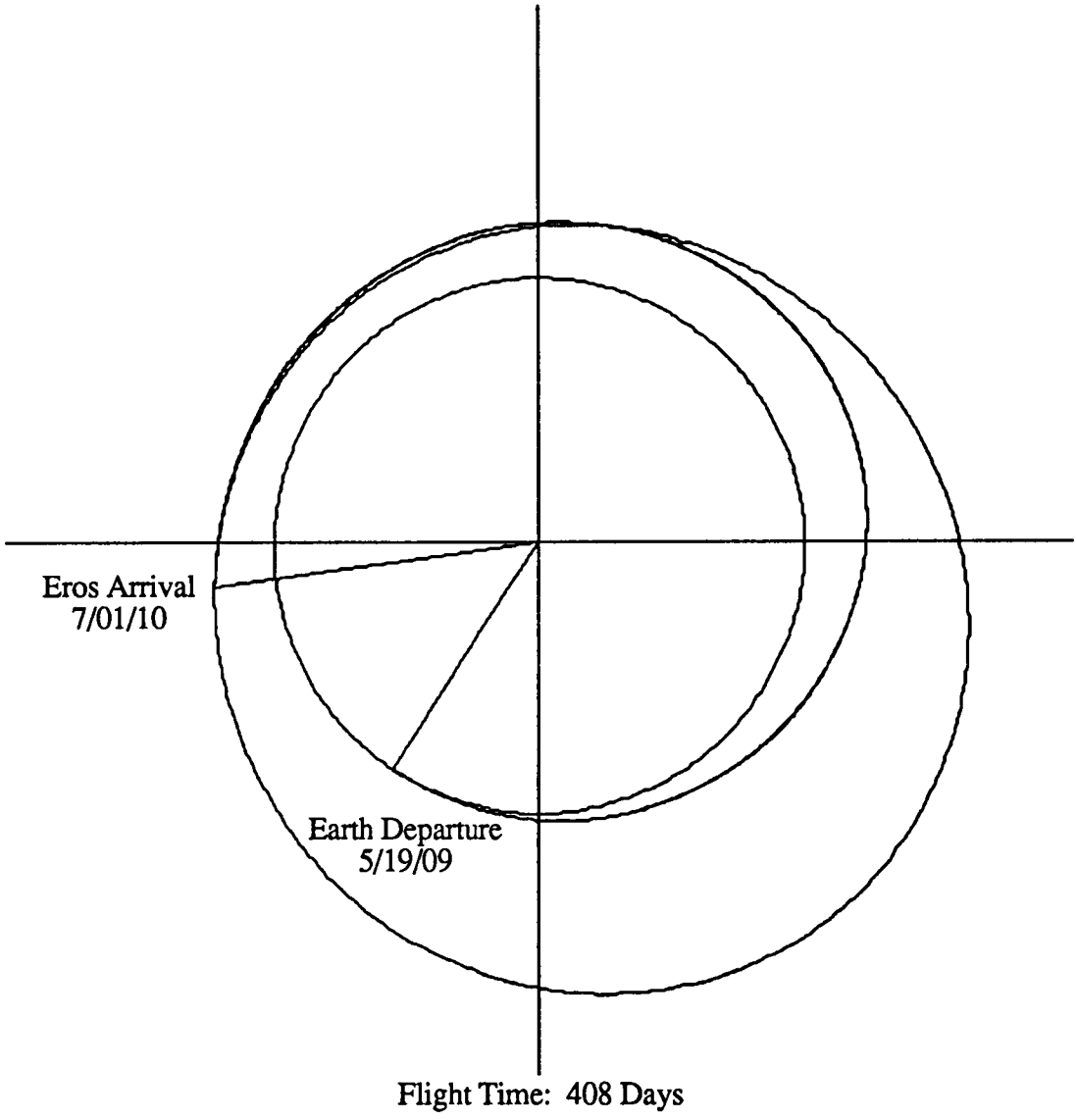

Figure 4.1.14: Ecliptic Projection of the Secondary Eros Mission Trajectory

acceleration is increased by increasing the sail area. The mission time frame investigated begins in January of 1996 and ends in December of 2012.

There are seven local minimum time mission opportunities which occur during the specified mission time frame. The various flight times for these mission opportunities are listed below in Table 4.2.1. This table follows the trends illustrated in Table 4.1.1. There are three different mission geometries that occur every seven years: a February/March launch, a May launch, and a July launch. The two lowest flight time missions occur in 2002 and 2009 respectively.

The solar sail's initial position, velocity, and control angle settings are listed for each mission opportunity in Table 4.2.2, and the initial conditions for the auxiliary variables are listed for each mission opportunity in Table 4.2.3. Since the initial values for the sail's position and velocity are equal to the Earth's position and velocity on the launch date the initial values for the celestial latitude (ψ) and normal velocity component (V_ψ) are always equal to zero. These values are not included in Table 4.2.2. There are two

Table 4.2.1: Minimum Time Mission Opportunities for a Solar Sail with a Characteristic Acceleration of 2 mm/s^2

Calendar Launch Date	Julian Launch Date	Time of Flight (Days)
3/27/98	2450900	406
5/17/00	2451681	366
7/25/02	2452480	286
3/13/05	2453443	398
5/09/07	2454230	377
7/06/09	2455018	303
2/23/12	2455981	370

Table 4.2.2: Initial Conditions of the State Variables and Control Angles for the 2 mm/s^2 Characteristic Acceleration Solar Sail Missions

Launch Date	Radius (A.U.)	ϕ (Deg.)	V_r (km/s)	V_ϕ (km/s)	θ (Deg.)	β (Deg.)
3/27/98	0.9975	185.3449	0.4934	29.8545	21.9413	132.4005
5/17/00	1.0112	235.8899	0.3644	29.4499	14.8777	73.8213
7/25/02	1.0159	300.5810	-0.1505	29.3146	26.8525	7.6544
3/13/05	0.9938	171.9709	0.4644	29.9678	23.9237	138.6236
5/09/07	1.0092	227.3517	0.4112	29.5087	15.1502	89.4718
7/06/09	1.0167	283.0051	0.0009	29.2914	24.2056	14.9169
2/23/12	0.9890	152.8642	0.3796	30.1107	26.1845	145.1648

Table 4.2.3: Initial Conditions of the Auxiliary Variables for the 2 mm/s^2 Characteristic Acceleration Solar Sail Missions

Launch Date	P_r	P_ψ	P_{Vr}	$P_{V\phi}$	$P_{V\psi}$
3/27/98	2.2790	-2.1168	2.1593	2.8529	-2.6051
5/17/00	2.4189	-2.2317	2.7777	2.4756	0.7182
7/25/02	1.3588	1.3519	1.7216	0.7147	5.3177
3/13/05	2.1942	-1.8601	1.9394	2.8136	-3.1941
5/09/07	2.4344	-2.3634	2.7423	2.6102	0.0241
7/06/09	1.6763	0.0626	2.0692	1.2056	4.5255
2/23/12	2.0433	-1.4657	1.6538	2.6986	-3.8778

characteristics of the initial conditions that stand out for the two optimum mission opportunities. The first characteristic is the launch occurs when the Earth near its aphelion. In fact, the Earth is past its aphelion when the primary mission is launched. The second characteristic is the initial clock angle is small indicating very little tangential acceleration. These characteristics also correspond to the 1mm/s^2 optimal mission initial conditions. The Earth is not as close to its aphelion on the 1 mm/s^2 launch dates as it is on the 2 mm/s^2 launch dates.

The final position and velocity at the rendezvous point are displayed in Table 4.2.4. Once again, it is more difficult to rendezvous with the target body after it passes its aphelion than before its aphelion. As illustrated in Table 4.2.1 and Table 4.2.4, the two lowest flight time missions are characterized by a rendezvous with Eros before it reaches its aphelion. The mission opportunity in 2002 has a lower flight time than the 2009 mission opportunity. The rendezvous point associated with the 2009 mission occurs closer to Eros's perihelion than the 2002 mission's rendezvous point. The 1 mm/s^2 sail missions also behave in this manner. Carl Sauer also says, "For low values of thrust acceleration and a high eccentricity of the target body, it becomes impractical to rendezvous with the body after aphelion." (11, p. 5). For the 1 mm/s^2 acceleration case, there are four missions in which rendezvous occurs before aphelion, and three missions in which rendezvous occurs after aphelion passage. For the 2 mm/s^2 acceleration case, there are two pre-aphelion rendezvous missions and five post-aphelion rendezvous missions.

The variation of the solar sail's position, velocity, control angles, and auxiliary variables with flight time for the primary and secondary missions are illustrated in Figures 4.2.1 - 4.2.6 and 4.2.7 - 4.2.12 respectively. The corresponding figures between the two missions, with the exception of the control angle graphs (Figures 4.2.5 and 4.2.11), are all similar; thus, the mission geometries must be similar. The difference in the control angle histories (Figures 4.2.5 and 4.2.11) is the clock angle behavior. The initial and final

Table 4.2.4: Final Conditions of the State Variables for the 2 mm/s^2 Characteristic Acceleration Solar Sail Missions

Launch Date	Radius (A.U.)	ϕ (Deg.)	ψ (Deg.)	V_r (km/s)	V_ϕ (km/s)	V_ψ (km/s)
3/27/98	1.4485	21.5802	10.5652	-5.5302	24.1879	1.0143
5/17/00	1.1662	91.0983	6.0123	-3.0205	29.6923	-4.7171
7/25/02	1.1707	157.0485	-5.8781	3.1953	29.5710	-4.7441
3/13/05	1.5621	2.1235	9.1793	-4.8620	22.3327	2.2585
5/09/07	1.1922	80.2837	7.5998	-3.8634	29.1426	-3.9616
7/06/09	1.1445	141.6653	-3.2328	1.8317	30.1381	-5.5008
2/23/12	1.7072	334.9084	5.5331	-3.0188	20.2657	3.3287

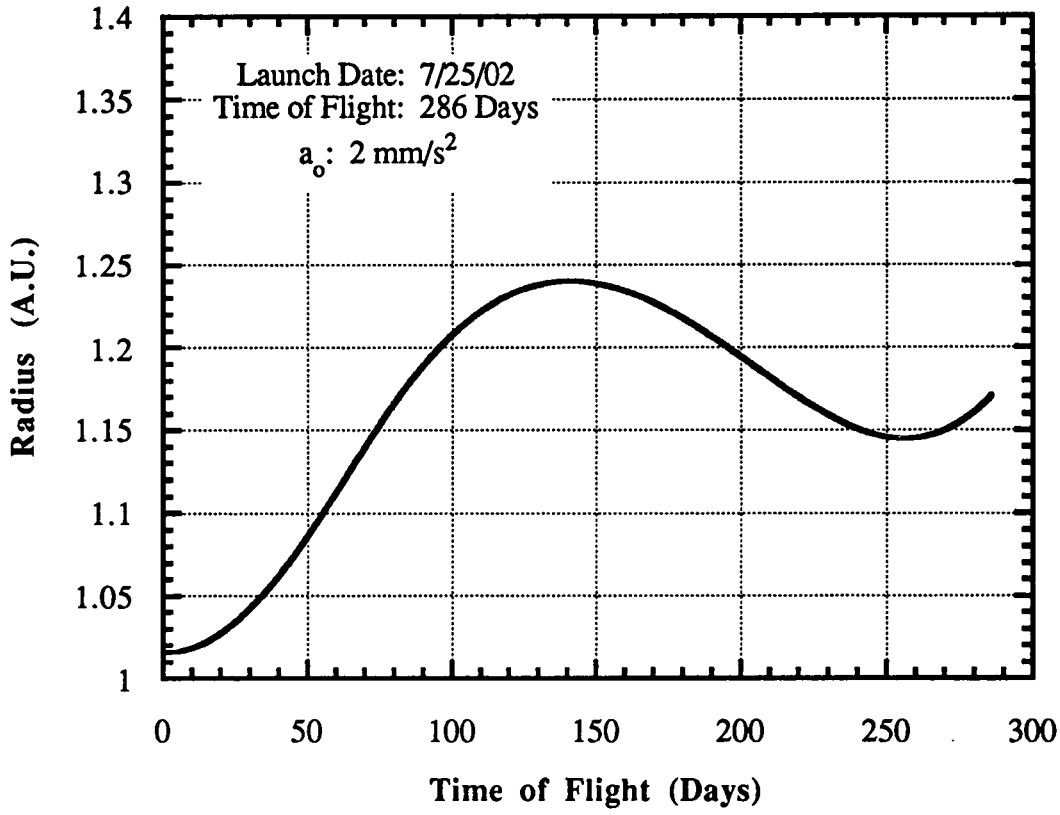

Figure 4.2.1: Solar Sail Radius History for the Primary Mission

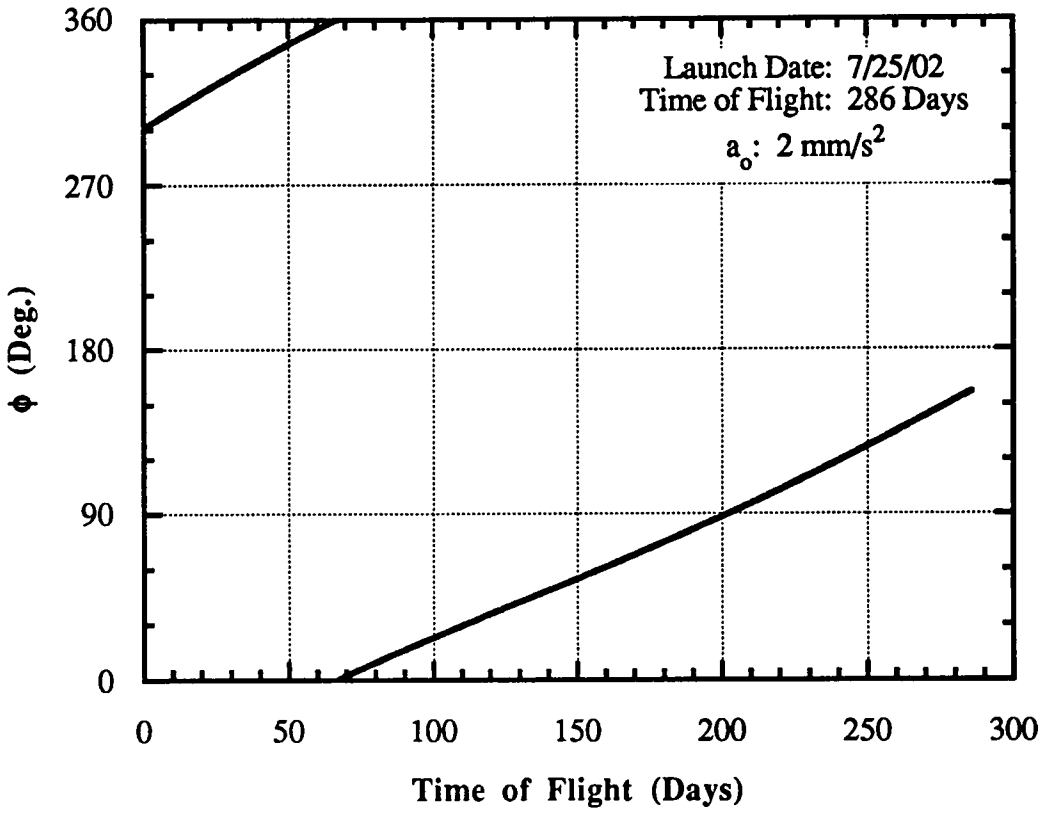

Figure 4.2.2: Solar Sail Celestial Longitude History for the Primary Mission

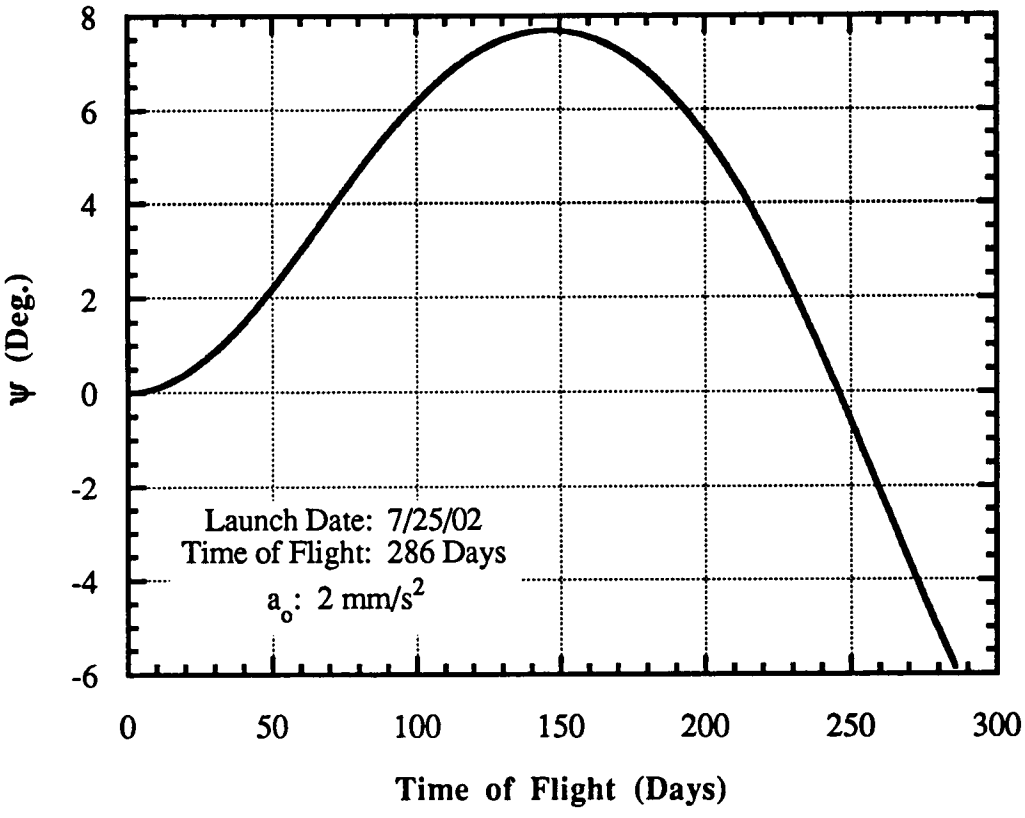

Figure 4.2.3: Solar Sail Celestial Latitude History for the Primary Mission

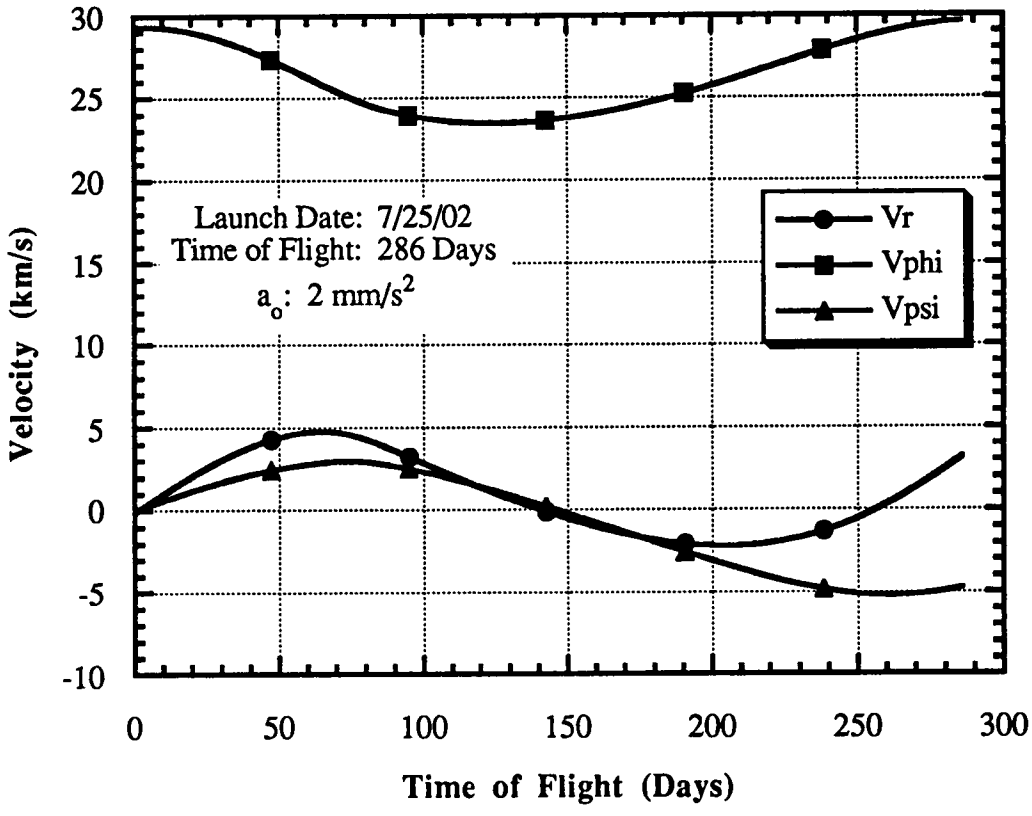

Figure 4.2.4: Solar Sail Velocity Component Histories for the Primary Mission

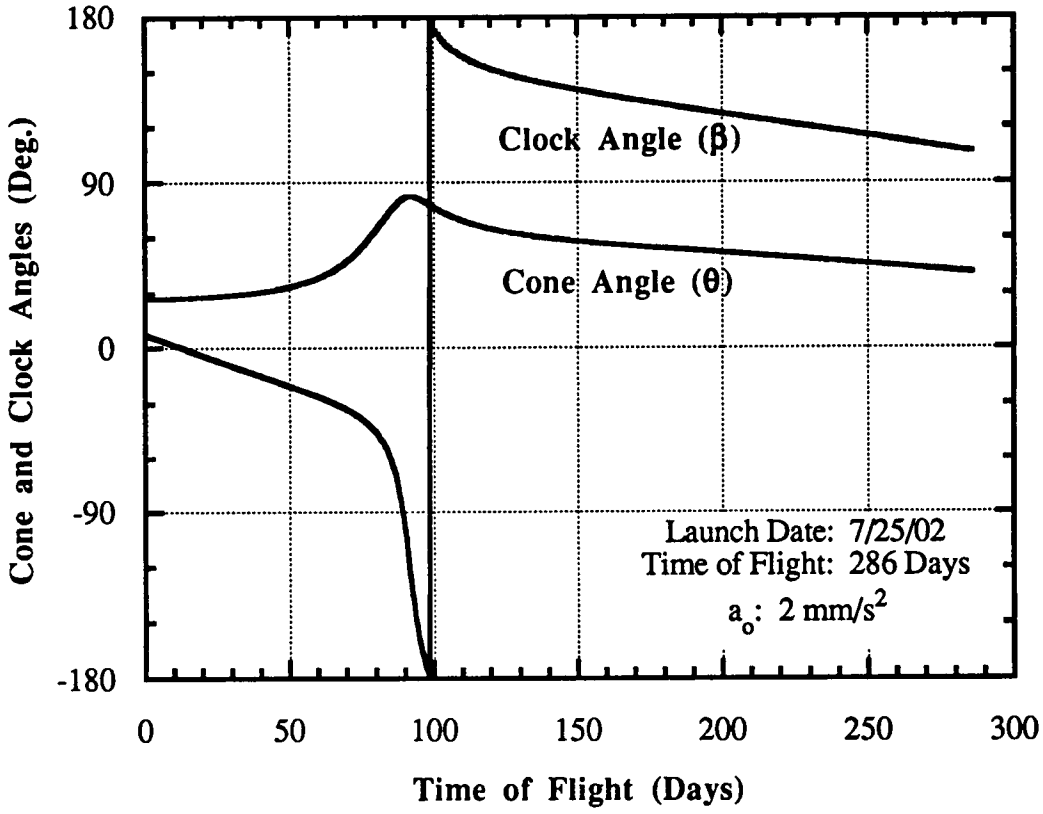

Figure 4.2.5: Solar Sail Control Angle Histories for the Primary Mission

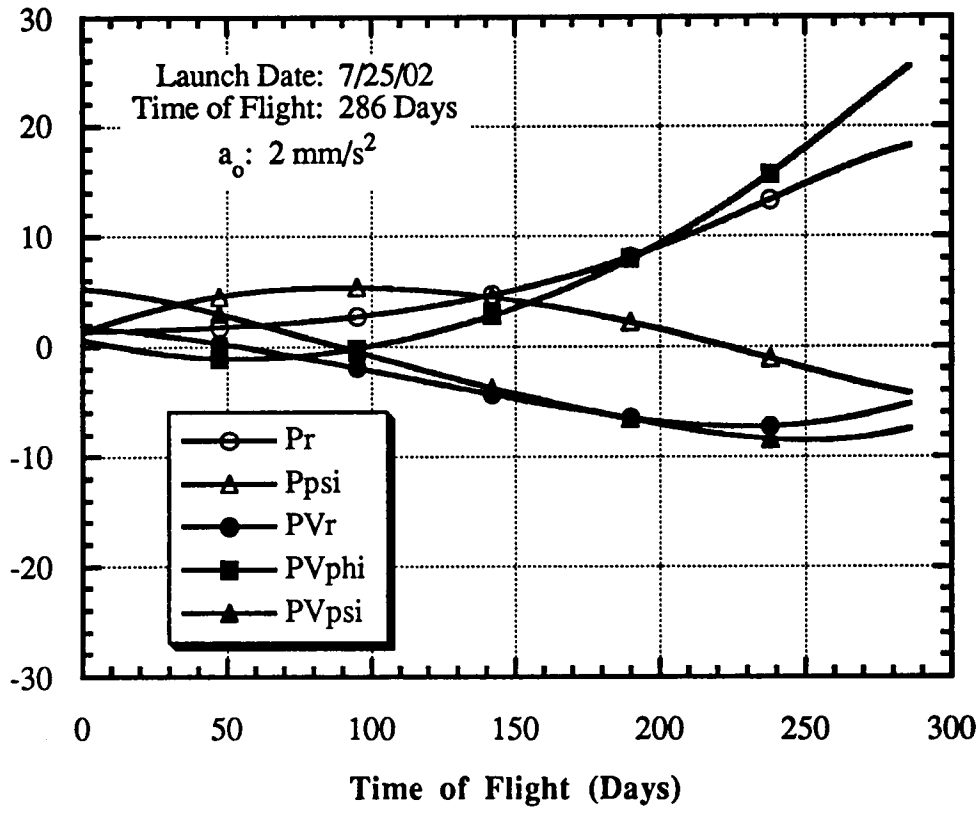

Figure 4.2.6: Solar Sail Auxiliary Variable Histories for the Primary Mission

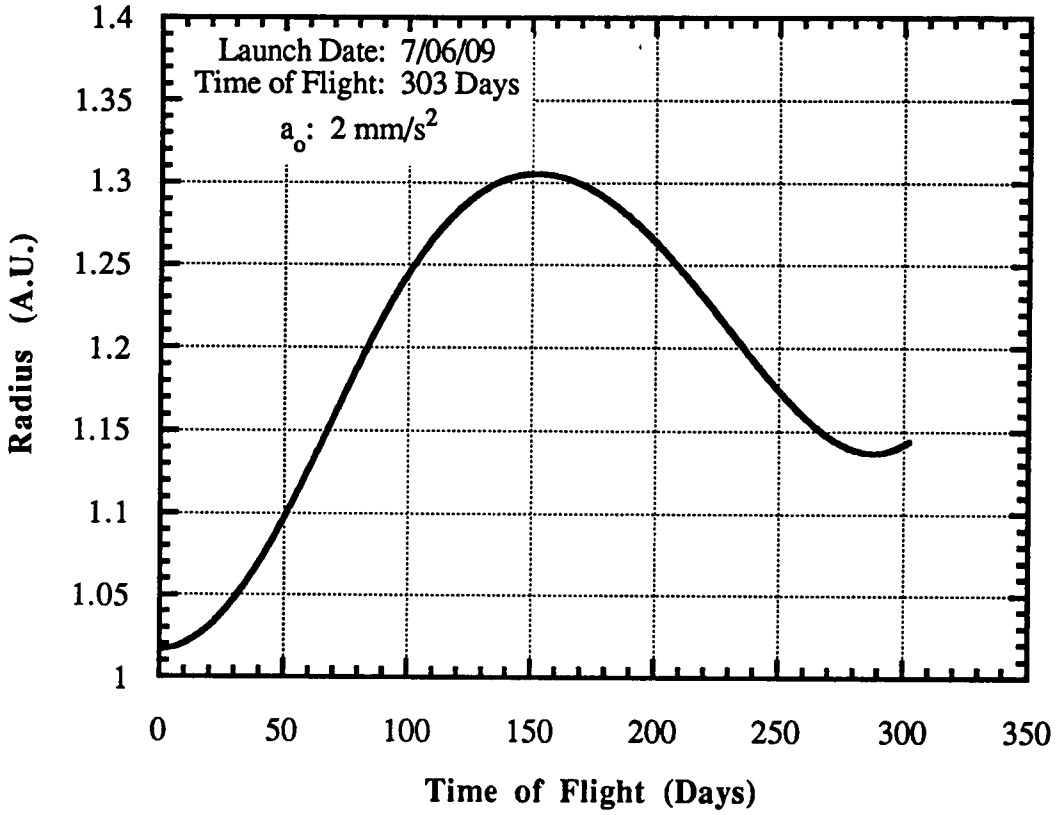

Figure 4.2.7: Solar Sail Radius History for the Secondary Mission

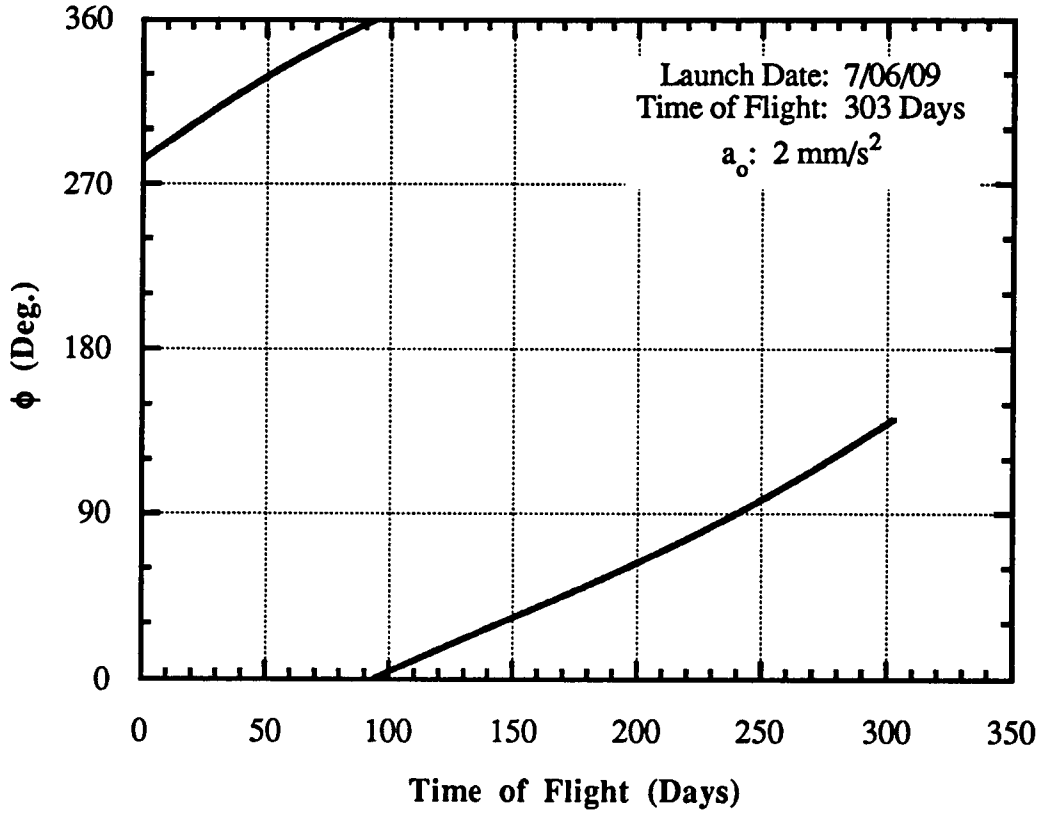

Figure 4.2.8: Solar Sail Celestial Longitude History for the Secondary Mission

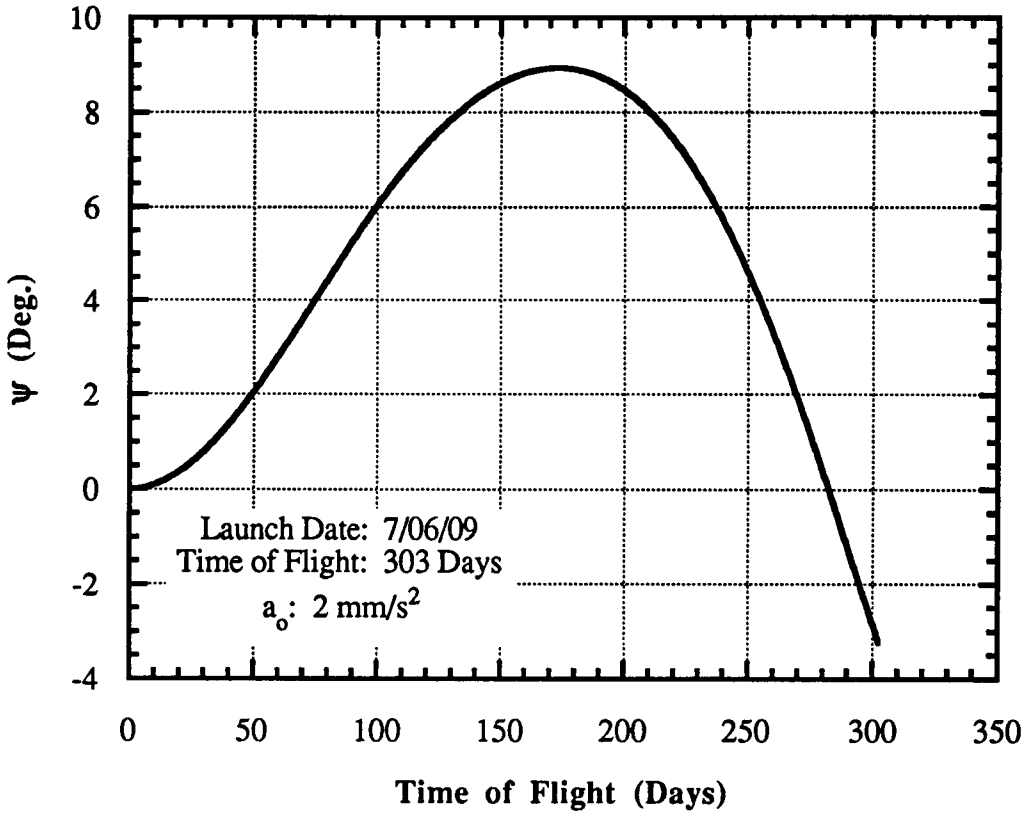

Figure 4.2.9: Solar Sail Celestial Latitude History for the Secondary Mission

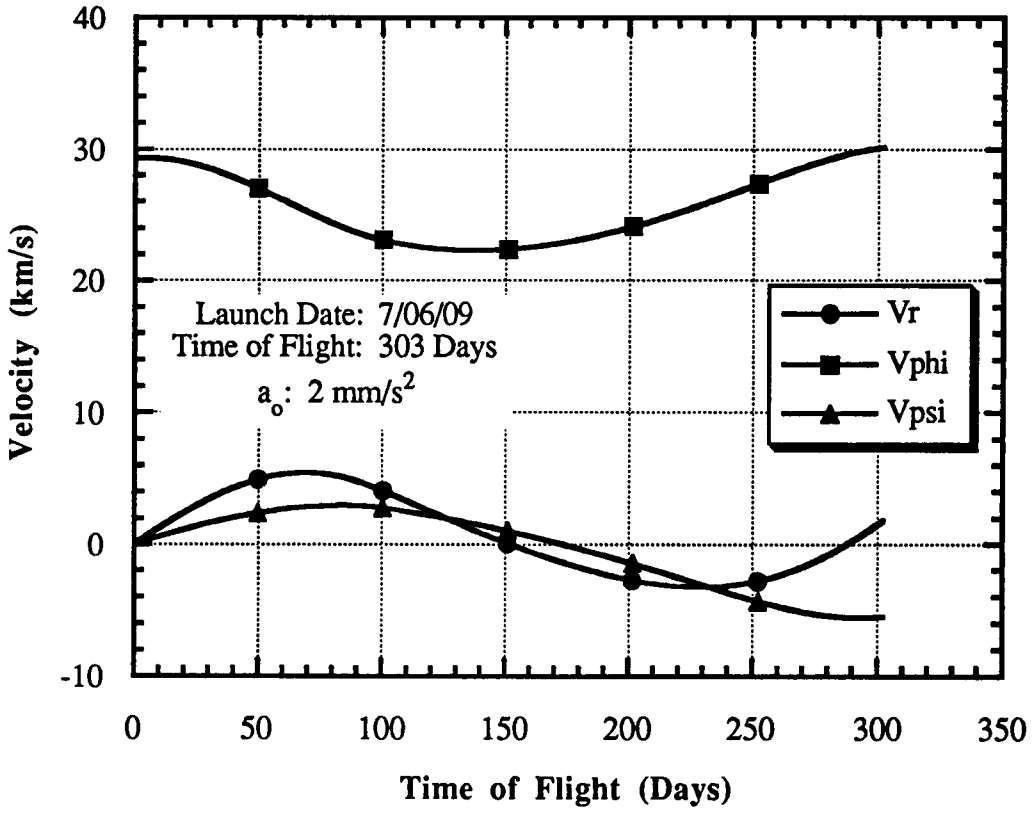

Figure 4.2.10: Solar Sail Velocity Component Histories for the Secondary Mission

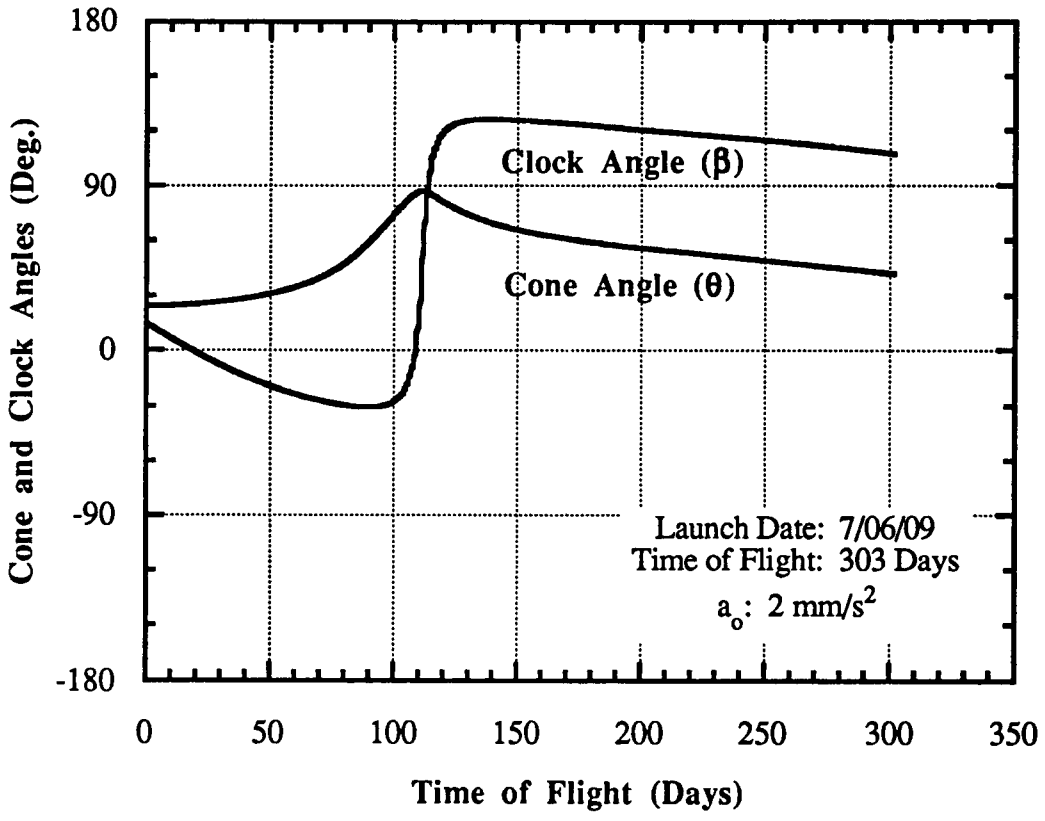

Figure 4.2.11: Solar Sail Control Angle Histories for the Secondary Mission

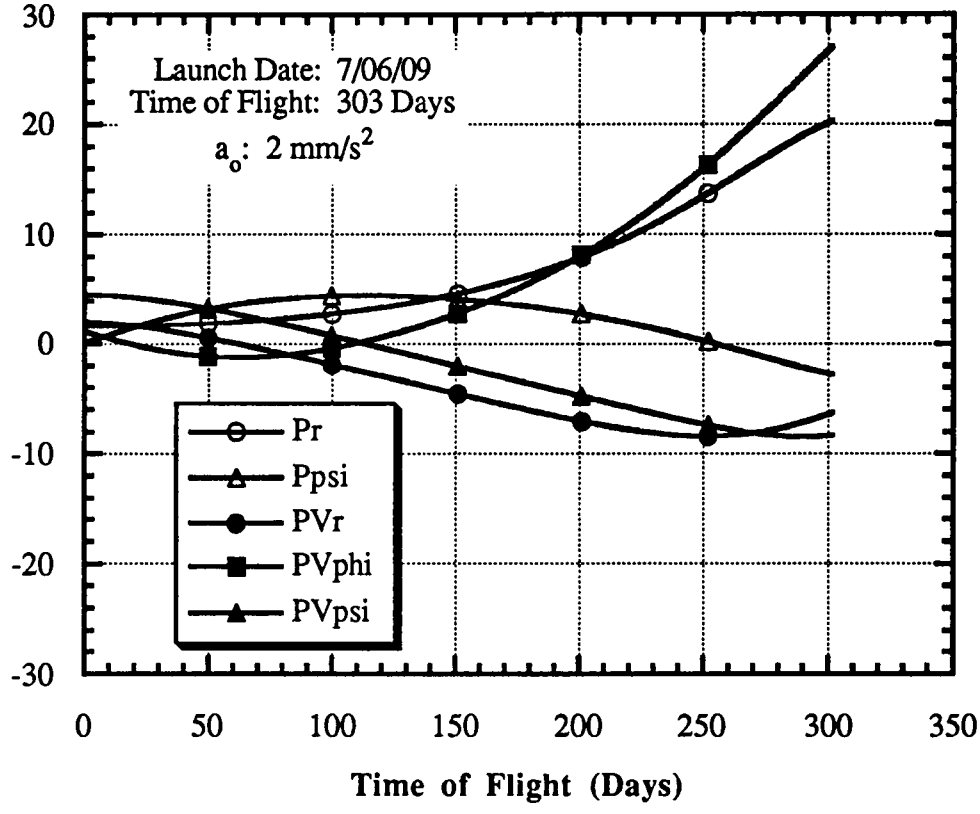

Figure 4.2.12: Solar Sail Auxiliary Variable Histories for the Secondary Mission

portions of the clock angle histories look very similar which accounts for the similarity between the two trajectories. The middle portion of the trajectory, which only consists of about 40 days, is different for the two histories. During the primary mission the clock angle continues to decrease through the negative tangential acceleration region (0 to -180 degrees), where as the clock angle stops decreasing and increases to a positive value. Both clock angles change very rapidly around the 100 day portion of the trajectory. There is some question as to how quickly the sail can reorient itself. In order to simplify the analysis, no limitation is placed on the sail's reorientation capabilities.

The sail parameter histories for the 2 mm/s^2 acceleration case are similar to the 1 mm/s^2 parameter histories. In both cases, the radius increases, decreases, and then increases again (Figures 4.2.1 and 4.2.7). The decrease in radius is utilized to increase the tangential velocity that was initially sacrificed to increase the radial and normal velocities (Figures 4.2.4 and 4.2.10). The solar sail initially flies to a positive celestial latitude even though the rendezvous latitude is below the ecliptic plane (Figures 4.2.3 and 4.2.9). Once again, the higher flight time of the secondary mission is a result of the rendezvous location. The rendezvous point of the secondary mission is closer to Eros's perihelion than the rendezvous point of the primary mission; thus, the rendezvous velocity is a little higher for the secondary mission. From Figures 4.2.1 and 4.2.7, it is apparent that the sail spirals farther away from the sun before spiraling back in to the rendezvous point in the secondary mission. This maneuver is utilized to increase the tangential velocity to the value required for rendezvous.

The 2 mm/s^2 characteristic acceleration trajectories for the primary and secondary missions are displayed in Figures 4.2.13 and 4.2.14 respectively. These figures show Earth's orbit and ecliptic projections of Eros's orbit and the solar sail trajectory. Both missions are characterized by a portion of the trajectory outside the orbit of Eros. These

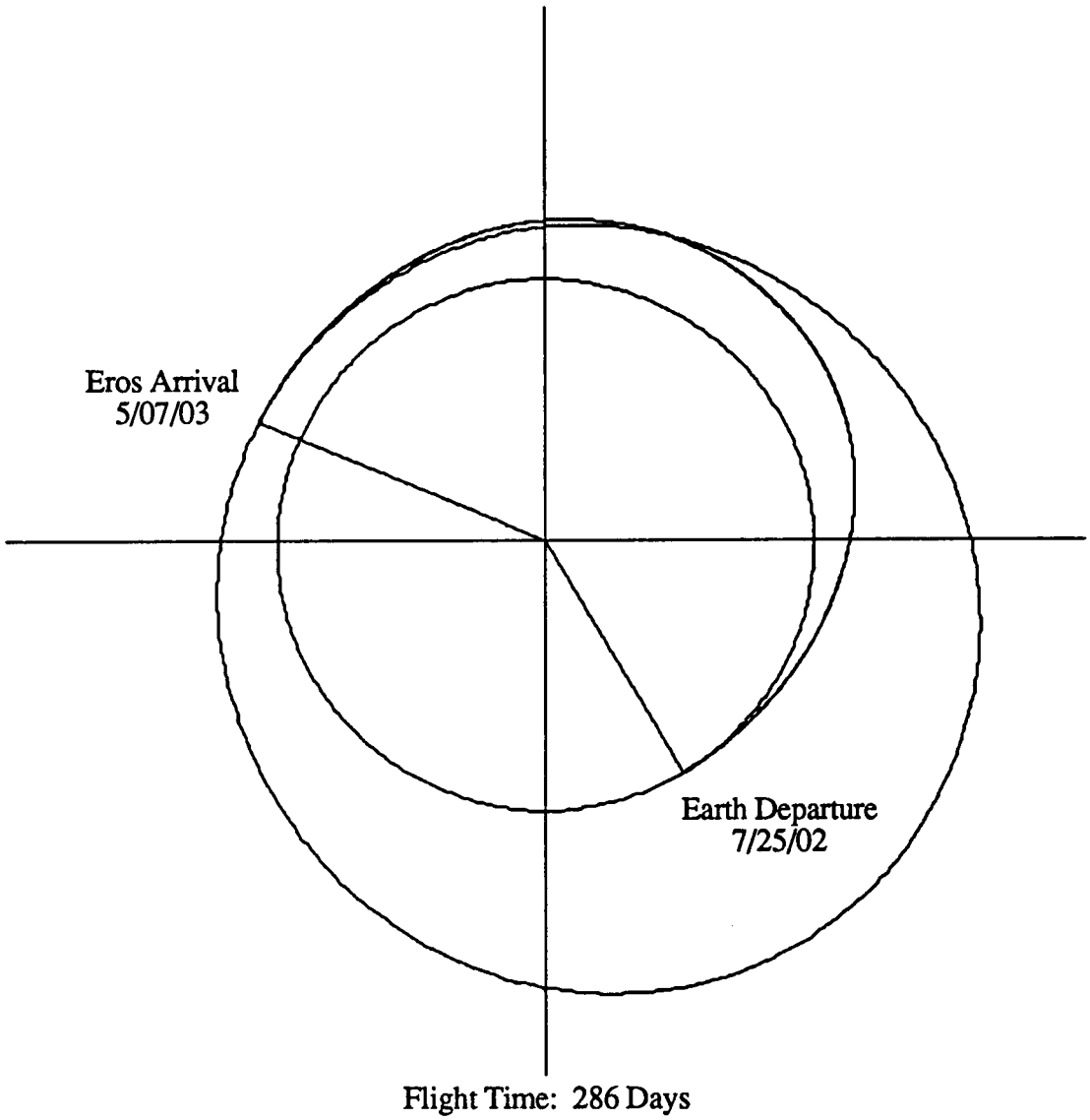

Figure 4.2.13: Ecliptic Projection of the Primary Eros Mission Trajectory

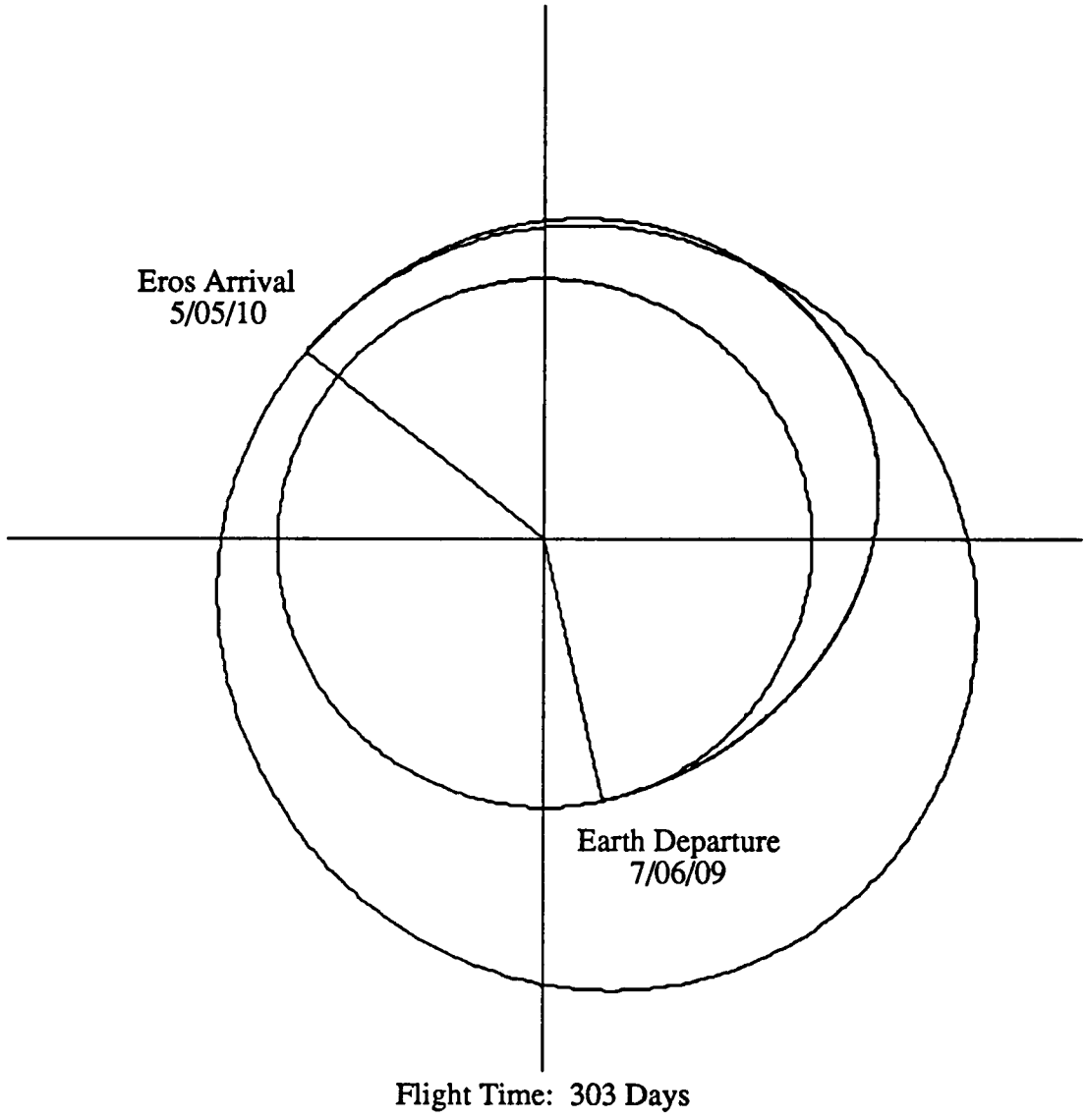

Figure 4.2.14: Ecliptic Projection of the Secondary Eros Mission Trajectory

figures also illustrate the fact that the rendezvous location of the secondary mission is closer to Eros's perihelion than the primary mission's rendezvous location.

4.3 Solar Sail Results Summary

Although there are seven mission opportunities for both characteristic acceleration cases, there are two missions for each acceleration that have significantly lower flight times than the other missions. The primary mission in each case is the absolute minimum flight time trajectory, and the secondary mission is the second lowest flight time trajectory. Both cases are presented because of the probable difficulty in launching a mission by the year 2002. The trajectory data for these missions is presented in Table 4.3.1. The solar sail analyzed in this study is actually a hybrid device. The initial conditions for the interplanetary portion of the trajectory are the Earth's position and velocity on the launch date; thus the sail must accelerate from its initial parking orbit velocity to Earth's heliocentric velocity. The sail can spiral to escape velocity on its own, but it takes a significant amount of time because of shadowing caused by the Earth. According to R. H. Frisbee (18, p.2-6), it takes about 700 days for a 1 mm/s2 characteristic acceleration sail to escape Earth and about 300 days for a 2 mm/s2 sail. R. H. Frisbee also says, "a sail cannot operate below an altitude of about 2000 km due to air drag" (18, p.2-1). Because of this problem, an alternative propulsion system must be used to boost the sail from the typical 500 km parking orbit to an altitude of 2000 km. Since an alternative propulsion scheme must be used anyway, it makes sense to use the same scheme to escape the Earth's gravitational field and significantly reduce the total trip time. In this study, a liquid oxygen, liquid hydrogen chemical system is used to boost the sail from a typical 500 km, 28.5 degree circular parking orbit. In all of the cases, tangential injection is possible; thus, the required ΔV is only the difference between the Earth's parabolic escape velocity and the

Table 4.3.1: Solar Sail Trajectory Data for the Primary and Secondary 433 Eros Rendezvous Missions (2006 - 2012)

Mission	Primary	Secondary	Primary	Secondary
Launch Date	6/02/02	5/19/09	7/25/02	7/06/09
Arrival Date	7/04/03	7/01/10	5/07/03	5/05/10
a_o (mm/s^2)	1	1	2	2
Time of Flight (Days)	397	408	286	303
Injection	Tangential	Tangential	Tangential	Tangential
Parking Orbit Ω (Deg.)	23.2723 or 156.7277	24.4985 or 164.8531	202.5885 or 337.4115	189.5285 or 350.4715
Required ΔV (km/s)	3.1532	3.1532	3.1532	3.1532

circular parking orbit velocity. If tangential injection is possible, there are two possible parking orbits with the same inclination that can be used. These parking orbits are characterized by the longitude of the ascending node. These values are listed in Table 4.3.1 along with the required ΔV. The longitude of the ascending node that is presented in Table 4.3.1 is measured relative to the projection of the Earth's velocity vector in the equatorial plane. Since the mission cost is the same from a ΔV standpoint, the best mission for each sail size is the minimum time mission.

The two major design parameters for the solar sail are its thickness and its surface area. In order to increase the characteristic acceleration of the sail, either the sail surface area must be increased, or the thickness must be reduced. Although increasing the sail area increases the characteristic acceleration of the vehicle, it also increases the total mass of the sail. Decreasing the thickness increases the characteristic acceleration while reducing the

mass. Thus, the performance of the sail is improved and the cost of deployment is reduced. Figure 4.3.1 illustrates the sail's maximum characteristic acceleration sensitivity to its thickness. Early sail designs assumed that the sail would be constructed on the ground, folded, boosted to an operational orbit, and deployed. According to Garvey, "Deployable sails must be at least 1 micron thick to survive folding and unfolding." (19, p. 2-3). A way to reduce the thickness is to fabricate the sails in orbit so they do not have to be folded. In this analysis, the sail must be deployable because of the high acceleration of the chemical motor used to escape the Earth's gravitational field. In the late 1970's, JPL conducted a preliminary design of a solar sail for the Halley's comet rendezvous mission. From Staehle's report, the sail was to be constructed of 2.5 micron Kapton ® coated with Aluminum, and have an area density (including support structure) of 0.0048 kg/m2, (20, pp. 328 - 329). The area density (ad) is calculated using the following formula:

$$ad = \rho\delta(1 + ks) \qquad (4.3.1)$$

where ρ is the density of the sail material, δ is the thickness, and ks is the structural factor. The area density is related to the maximum achievable characteristic acceleration of the solar sail in the following manner: $(a_o)_{max} = P_{sol} / ad$. The solar radiation constant (P_{sol}) is equal to 9.0 N/km^2. This relationship is illustrated by Figure 4.3.1. Using this formula, which assumes the reflectivity of the sail is 1.0, the maximum achievable characteristic acceleration of the Halley's rendezvous mission sail is 1.879 mm/s^2. The Halley's comet rendezvous sail is representative of 1982 technology. According to Friedman, "it seems reasonable to consider this thickness of 2.0 μm for the first operational solar sail" (7, p. 29). In this analysis, a 2 μm thick sail constructed of coated Kapton ® with a reflectivity of 1.0 is assumed. Thus, the area density of this sail is 0.00384 kg/m^2. This sail also meets the 1 μm thickness deployment requirement.

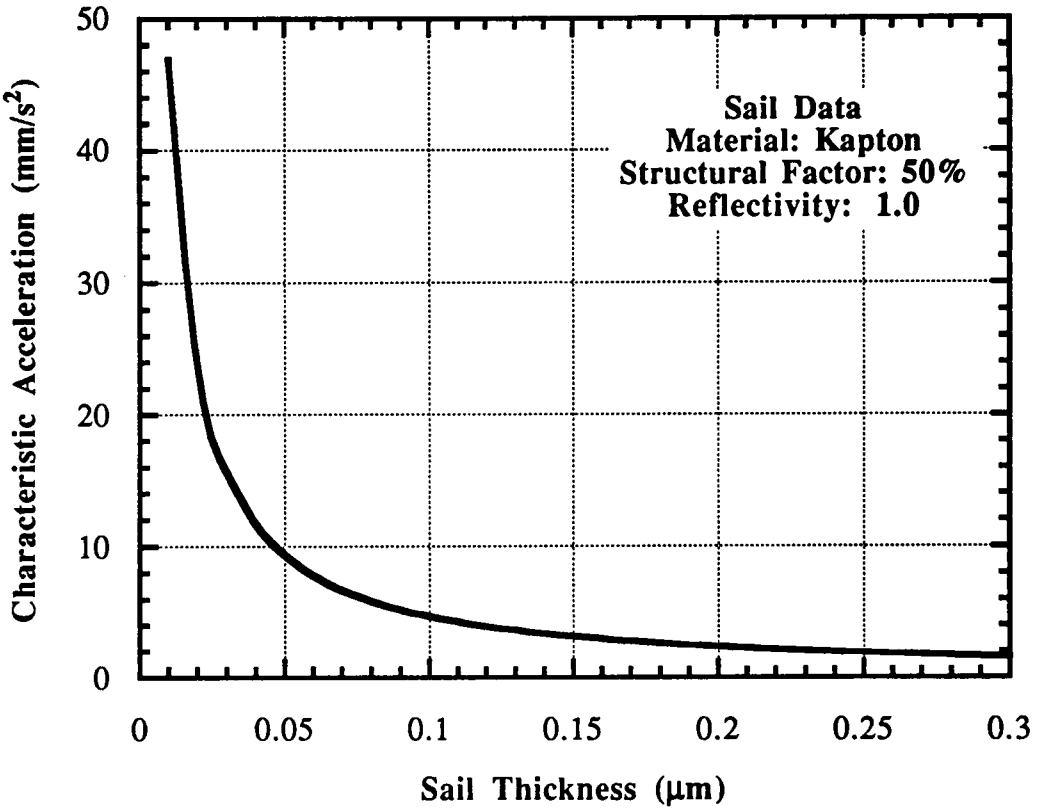

Figure 4.3.1: Maximum Characteristic Acceleration Sensitivity to Sail Thickness

In order to calculate the initial mass in low Earth orbit (IMLEO), the sail area and mass must be calculated. Equation 2.2.3 can be modified to solve for the sail surface area:

$$S_o = \frac{a_o(M_{PL})}{P_{sol} - a_o\rho\delta(1+ks)} \tag{4.3.2}$$

The sail mass (M_{sail}) is calculated using the following equation:

$$M_{sail} = S_o\rho\delta(1+ks) \tag{4.3.3}$$

Using Equation 4.3.2, the sail area required to achieve a characteristic acceleration of 1 mm/s^2 is 65891 m^2, and the sail area required to achieve an acceleration of 2 mm/s^2 is 515152 m^2. The corresponding sail masses for the 1 mm/s^2 and 2 mm/s^2 sails are 253 kg and 1978 kg respectively. Thus, doubling the characteristic acceleration by increasing the sail area (assuming the same area density) results in a mass that is 8 times the original mass.

In order to calculate the initial mass in low Earth orbit, the mass of the chemical escape booster and the mass of the propellant required must be calculated. Once again the mass of the spacecraft and the payload are 340 kg, and the Isp of the chemical system is assumed to be 470 s. In this case, the mass of the solar sail is added to the spacecraft and payload mass. The mass of the 1 mm/s^2 sail , the spacecraft, and the payload is equal to 593 kg, and the mass of the 2 mm/s^2 sail, the spacecraft, and the payload is equal to 2318 kg. Using Equation 3.2.1, and the delta V required for Earth escape (3.1532 km/s), the sail mass ratio (σ_{depart}) is 1.9816. Once again, the method utilized in section 3.2 is used to find the propulsion system dry mass and the propellant mass required for the launch delta V. The total system dry mass of the 1 mm/s^2 sail is initially assumed to be 593 kg. This value gives a required propellant mass of 582 kg. The mass of propellant is assumed to be 85% of the total propulsion system mass; thus, the first approximation for the propulsion system dry mass is 103 kg. Since the actual propellant mass and the propulsion

system dry mass are greater than these values, the propulsion system dry mass is assumed to be 150% of its initial value. Therefore, the propulsion system dry mass is 150 kg. This value is then added to the initial approximation for the total system dry mass (593) and the initial total system mass before the burn is recalculated using the departure mass ratio (σ_{depart}). For the 1 mm/s^2 sail, the initial total system mass before the burn is 1472 kg, and the propellant mass required is 729 kg. In order to check the propulsion system dry mass assumption, the mass of propellant is divided by the total propulsion system mass (propellant mass and propulsion system dry mass). In this case, the propellant is 83 % of the total propulsion system mass. Once again, a propellant safety margin mass of 15 % and a residual mass of 1 % are assumed. The mass of this extra propellant is 118 kg. The IMLEO of the system is equal to the sum of the initial total system mass and the extra propellant mass. The IMLEO of the 1 mm/s2 characteristic acceleration solar sail hybrid is 1590 kg.

The same procedure is used to calculate the 2 mm/s^2 sail hybrid IMLEO. The initial assumption for the total system dry mass is 2318 kg. In order to boost this mass to escape velocity, 2275 kg of propellant are required. This propellant mass is 85 % of a 2676 kg propulsion system; thus, the initial approximation for the propulsion system dry mass is 400 kg. Once again, the actual propulsion system dry mass is assumed to be 150 % of the original approximation. Based on this assumption, the propulsion system dry mass is 600 kg, and the total system dry mass is 2918 kg. Using the departure mass fraction (σ_{depart}), the initial total system mass is 5783 kg. The actual propellant mass required is 2864 kg, which is 83 % of the total propulsion system mass. The safety margin and residual propellant mass is 457 kg. The IMLEO of the 2 mm/s2 characteristic acceleration solar sail hybrid system is 6240 kg. The component masses for the two sail systems are summarized in Table 4.3.2.

Table 4.3.2: Component Mass Summary for the Solar Sail Hybrid Systems

Characteristic Acceleration (mm/s^2)	1.0	2.0
Sail Area (km^2)	0.0659	0.5152
Sail Mass (kg)	253	1978
Payload & Spacecraft Bus Mass (kg)	340	340
Propulsion System Dry Mass (kg)	150	600
Propellant Mass (kg)	729	2864
Propellant Margin and Residual Mass (kg)	118	457
Sail System IMLEO	1590	6240

5. CONCLUSIONS

Data from an asteroid rendezvous mission could increase our understanding of the origin of the solar system. The mission would enable scientists to study solar system bodies that are believed to be unchanged since their formation. In light of current national budgetary constraints, a mission of this type would never be realized unless the cost was sufficiently low. Thus, low cost propulsion systems must be developed so that the space exploration initiatives can be continued. A promising system in this regard is the solar sail, which requires no propellant for interplanetary flight. The two main criticisms of the solar sail are its long flight times, and the difficulties associated with the dynamics and the deployment of the system. The dynamics and deployment problem is challenging but it is solvable. The long time of flight is addressed in this thesis. In order to reduce the Earth escape time, a chemical booster is employed. As illustrated in the results, the mission flight times associated with the primary and secondary missions of the 2 mm/s^2 characteristic acceleration sail are comparable to the chemical mission flight times. The 2 mm/s^2 sail primary mission is recommended for launch on July 25, 2002 and the mission flight time is 286 days. The minimum cost chemical mission would be launched on January 30, 2005, and the mission flight time is 291 days. The cost of the mission is directly related to the IMLEO of the total system. The cost of the 2 mm/s^2 sail hybrid mission is much greater than the chemical mission because the IMLEO of the 2 mm/s^2 sail hybrid at 6240 kg, is more than twice as much as the chemical system's IMLEO of 2730 kg. The mission cost of the smaller 1 mm/s^2 sail hybrid system is much less than both the large sail hybrid and the chemical systems. The IMLEO of the 1 mm/s^2 sail hybrid system is 1590 kg. The primary 1 mm/s^2 sail hybrid mission would be launched on June 2, 2002, and has a flight time of 397 days. Thus, reducing the sail mission cost results in a flight time increase. Since the mission cost is more important than the mission flight time for an unmanned

mission, the small solar sail is the best mission propulsion system option. The sail material and thickness assumed in this thesis represent 1982 technology. Advances in light weight materials could be implemented to reduce the sail mass and thickness. This would reduce the cost of the high characteristic acceleration sail missions. As demonstrated, the solar sail flight time can be reduced to be comparable to the chemical systems. Another benefit associated with the solar sail is its reusability. The solar sail is ideally suited for sample and return missions. Sails also show promise as a large cargo vessel for manned Mars mission. Because of the potential benefits associated with the solar sail, research in this area should be restarted.

The low thrust trajectory optimization program in Appendix B was developed from basic principles and it represents original work by the author, A. Allen McCarley, and H. Dan. Thomas. This application of Pontryagin's maximization principle is not found in standard texts. This program is comparable to the program written by Carl Sauer at the Jet Propulsion Laboratory and it extends the analysis done by Zhukov and Lebdev (9) to three dimensions.

LIST OF REFERENCES

LIST OF REFERENCES

(1) Williams, S. N., Knocke, P. C., and Bright L. E., "Interplanetary Mission Design for the Comet Rendezvous Mission", AAS 91-396, 1991.

(2) Sauer, Carl G., "Low Energy Near-Earth Asteroid Flyby Missions", AAS 91-398, 1991.

(3) Rom, Frank E., "Nuclear Rocket Propulsion", NASA TM X-1685, Conference of NonChemical Space Propulsion, 12 January 1967.

(4) Frisbee, R. H., Blandino, J. J., Sercel, J. C., Sargent, M. G., Gowda, N., "Advanced Propulsion Options for the Mars Cargo Mission", AIAA 90-1997, AIAA/SAE/ASME/ASEE 26th Joint Propulsion Conference, Orlando, FL, 16 - 18 July 1990.

(5) Wertz, James R. and Larson, Wiley J. (editors), Space Mission Analysis and Design, Kluwer Academic Publishers, Boston, MA, 1991.

(6) Friedman, L., Carroll, W., Goldstein, R., Jacobson, R., Kievit, J., Landel, R., Layman, W., Marsh, E., Ploszaj, R., Rowe, W., Ruff, W., Stevens, J., Stimpson, L., Trubert, M., Varsi, G., Wright, J., and MacNeal, R., "Solar Sailing - The Concept Made Realistic", AIAA 78-82, AIAA 16th Aerospace Sciences Meeting, Huntsville, AL, 16 - 18 January 1978.

(7) Friedman, Louis, Starsailing: Solar Sails and Interstellar Travel, John Wiley and Sons Inc., New York, NY, 1991.

(8) Burden, Richard L. and Faires, J. Douglas, Numerical Analysis, PWS-Kent Publishing Company, Boston, MA, 1989.

(9) Zhukov, A. N. and Lebdev, V. N., "Variational Problem of Transfer Between Heliocentric Corcular Orbits by Means of a Solar Sail", Translated from Kosmicheskie Issledovaniya, Vol. 2, No. 1, pp. 45 - 50, January-February 1964.

(10) Grodzovskii, G. L., Ivanov, Yu. N., and Tokarev, V. V., Mechanics of Spaceflight Low-Thrust, Translated from Russian, Israel Press, Jerusalem, 1969.

(11) Sauer, Carl G., "Optimum Solar - Sail Interplanetary Trajectories", AIAA 76 - 792, AIAA/AAS Astrodynamics Conference, San Diego, CA, 18 - 20 August 1976.

(12) Wright, J. and Warmke, J., "Solar Sail Mission Applications", AIAA 76 - 808, AIAA/AAS Astrodynamics Conference, San Diego, CA, 18 - 20 August 1976.

(13) Pontryagin, L. S., Boltyanskii, V. G., Gamkrelidze, R. V., and Mishchenko, E. F., The Mathematical Theory of Optimal Processes, The MacMillian Company, New York, NY, 1964.

(14) Sackett, Lester L., "Optimal Solar Sail Planetocentric Trajectories", R-1113, Charles Stark Draper Laboratory, Inc., September 1977.

(15) Battin, Richard H., An Introduction to the Mathematics and Methods of Astrodynamics, American Institute of Aeronautics and Astronautics, Inc., New York, NY, 1987.

(16) International Reference Guide to Space Launch Systems, American Institute of Aeronautics and Astronautics, Inc., New York, NY, 1991.

(17) McAdams, Jim V., "Mission Options for Rendezvous with the Most Accessible Near-Earth Asteroid - 1989 ML", AAS 91-397, 1991.

(18) Frisbee, R. H., Blandino, J. J., Sercel, J. C., Sargent, M. G., Gowda, N., "Advanced Propulsion Options for the Mars Cargo Mission", JPL D-6620 Rev. A, Jet Propulsion Laboratory, September 1989.

(19) Garvey, J. M., "Space Station Options for Constructing Advanced Solar Sails Capable of Multiple Mars Missions, AIAA 87-1902, AIAA/SAE/ASME/ASEE 23rd Joint Propulsion Conference, San Diego, CA, 29 June - 2 July 1987.

(20) Staehle, Robert L., "An Expedition to Mars Employing Shuttle-Era Systems, Solar Sail and Aerocapture", Journal of the British Interplanetary Society, Vol. 35, 1982.

(21) Danby, J. M. A., Fundamentals of Celestial Mechanics, Willmann-Bell, Inc., Richmond, VA, 1988.

(22) Bate, Roger R., Mueller, Donald D., and White, Jerry E., Fundamentals of Astrodynamics, Dover publications, Inc., New York, NY, 1971.

APPENDIXES

APPENDIX A

Ephemeris Data For Eros

Mean anomaly (Mo) on June 19, 1986 = 170.451

Argument of the periapsis (ω) = 178.510

Eccentricity (e) = 0.2229

Semi-major axis (a) = 1.4582

Inclination (i) = 10.832

Longitude of the ascending node (Ω) = 304.496

From Fundamentals of Celestial Mechanics by J.M.A. Danby (21)

Constants and Conversions

Solar Gravitational Constant (GMsol)* = 2.959083388 E-4 AU^3 / Day^2

Solar Gravitational Constant (GMsol)* = 3.96396415708 E-14 AU^3 / s^2

Solar Radiation Pressure at 1 A.U. (P_{sol}) = 9 N / km^2

Earth Gravitational Constant (GMearth)* = 398600 km^3 / s^2

Gravitational Constant (g) = 9.81 m/s^2

1 AU* = 1.4959965 E+8 km

1 radian = 57.29577951 degrees;

* From Fundamentals of Astrodynamics by Bate, Muller and White (22)

APPENDIX B

Impulsive Program

```
program Asteroid Rendezvous (input, output, C3, DeltaV, orbit);

{_____________________________________________________________________}
{*********************************************************************}
{Program:          Impulsive Intrerplanetary Trajectories}
{}
{By:               A. Allen McCarley and Brian K. Funk}
{}
{Date:             May 21, 1990}
{}
{Last Revised:     October 2, 1991}
{}
{Description:      This program calculates the three dimensional conic}
{                  trajectory for any two objects in the solar system}
{                  using Battin's method. Launch dates and arrival dates}
{                  and the respective planets involved are the only }
{                  required inputs.}
{*********************************************************************}
{---------------------------------------------------------------------}

      uses
            Ephemeris, Graphical, Indirect, Quadrant, Verification, Parking;

{****************************** UNITS ******************************}

{*********************************************************************}
unit Ephemeris;

interface

      type
            planet = array[1..7] of extended;
            storage = array[1..7] of extended;

      const
            GMSol = 2.959083388E-4;    {AU^3/Day^2}

      procedure Mercury (var choice: planet; time: extended);
      procedure Venus (var choice: planet; time: extended);
      procedure earth (var choice: planet; time: extended);
      procedure Mars (var choice: planet; time: extended);
      procedure XB (var choice: planet; Jt: extended);
      procedure Orpheus (var choice: planet; Jt: extended);
      procedure McAuliffe (var choice: planet; Jt: extended);
      procedure Seleucus (var choice: planet; Jt: extended);
      procedure Eros (var choice: planet; Jt: extended);
```

```
implementation

procedure Mercury (var choice: planet; time: extended);

                { This procedure determines the orbital}
                { elements for the planet Mercury at}
                { the date chosen.}

        begin
                choice[1] := (14732.4197380 - 0.000001355 * time);
                choice[2] := (178 + 10 / 60 + 44.68 / 3600) + (538106654.80 / 3600) *
                                time + (1.084 / 3600) * time * time;
                choice[3] := 75 + 53 / 60 + 58.91 / 3600 + (5599.76 / 3600) * time +
                                (1.061 / 3600) * time * time;
                choice[4] := 0.20561421 + 0.00002046 * time - 0.000000030 * time *
                                time;
                choice[5] := 0.38709860;
                choice[6] := 7 + 10.37 / 3600 + (6.699 / 3600) * time - (0.066 / 3600) *
                                time * time;
                choice[7] := 47 + 8 / 60 + 45.4 / 3600 + (4266.75 / 3600) * time + (0.626 /
                                3600) * time * time;
        end;{Freddy Mercury}

{=============================================================}
procedure Venus (var choice: planet; time: extended);

                { This procedure determines the orbital}
                { elements for the planet Venus at the}
                { date chosen.}

        begin
                choice[1] := (5767.6697692 + 0.0000002628 * time);
                choice[2] := 342 + 46 / 60 + 1.39 / 3600 + (210669162.88 / 3600) * time +
                                (1.1148 / 3600) * time * time;
                choice[3] := 130 + 9 / 60 + 49.8 / 3600 + (5068.93 / 3600) * time - (3.515 /
                                3600) * time * time;
                choice[4] := 0.00682069 - 0.00004774 * time + 0.000000091 * time *
                                time;
                choice[5] := 0.7323162;
                choice[6] := 3 + 23 / 60 + 37.07 / 3600 + (3.621 / 3600) * time - (0.0035 /
                                3600) * time * time;
                choice[7] := 75 + 46 / 60 + 46.73 / 3600 + (3239.46 / 3600) * time +
                                (1.476 / 3600) * time * time;
        end;

{=============================================================}
procedure earth (var choice: planet; time: extended);

                { This procedure determines the orbital}
                { elements for the planet Venus at the}
                { date chosen.}
```

```
        begin
                choice[1] := (3548.1928323 - 0.000001103 * time);
                choice[2] := 99 + 41 / 60 + 48.04 / 3600 + (129602768.13 / 3600) * time +
                                (1.089 / 3600) * time * time;
                choice[3] := 101 + 13 / 60 + 15 / 3600 + (6189.03 / 3600) * time + (1.63 /
                                3600) * time * time + (0.012 / 3600) * time * time * time;
                choice[4] := 0.01675104 - 0.00004180 * time - 0.000000126 * time * time;
                choice[5] := 1.00000023;
                choice[6] := 0.0;
                choice[7] := 0.0;
        end;

{============================================================}
procedure Mars (var choice: planet; time: extended);

                { This procedure determines the orbital}
                {  elements for the planet Mars at the}
                {  date chosen.}

        begin
                choice[1] := (1886.5186207 + 0.000000463 * time);
                choice[2] := 293 + 44 / 60 + 51.46 / 3600 + (68910103.83 / 3600) * time +
                                (1.1184 / 3600) * time * time;
                choice[3] := 334 + 13 / 60 + 5.53 / 3600 + (6626.73 / 3600) * time +
                                (0.4675 / 3600) * time * time - (0.0043 / 3600) * time * time
                                * time;
                choice[4] := 0.09331290 + (0.000092064) * time - (0.000000077) * time *
                                time;
                choice[5] := 1.52368840;
                choice[6] := 1 + 51 / 60 + 1.2 / 3600 - (2.43 / 3600) * time + (0.0454 /
                                3600) * time * time;
                choice[7] := 48 + 47 / 60 + 11.19 / 3600 + (2775.57 / 3600) * time - (0.005
                                / 3600) * time * time - (0.0192 / 3600) * time * time * time;
        end;{Mars bar (with almonds)}

{============================================================}
procedure XB (var choice: planet; Jt: extended);

                {This procedure gives the ephimeral data}
                for the asteroid 1982 XB}

        var
                n,                      {Asteroid mean motion}
                tau: extended;          {time since perigee in days}

        begin
                choice[1] := 0;
                choice[2] := 56.89879;
                choice[3] := 16.77969 + 74.52644;
                choice[4] := 0.4466558;
                choice[5] := 1.8355863;
                choice[6] := 3.87466;
```

```
        choice[7] := 74.52644;
        n := sqrt(GMSol / (choice[5] * choice[5] * choice[5]));
        tau := -(choice[2] / 57.29577951) / n + 2448200;
        choice[2] := n * (Jt - tau) * 57.29577951;
    end;

{============================================================}
procedure Orpheus (var choice: planet; Jt: extended);

        {This procedure gives the ephimeral data }
        {for the asteroid Orpheus}

    var
        n,                      {Asteroid mean motion}
        tau: extended;          {time since perigee in days}

    begin
        choice[1] := 0;
        choice[2] := 197.58723;
        choice[3] := 301.56931 + 189.13898;
        choice[4] := 0.3226346;
        choice[5] := 1.2093243;
        choice[6] := 2.68775;
        choice[7] := 189.13898;
        n := sqrt(GMSol / (choice[5] * choice[5] * choice[5]));
        tau := -(choice[2] / 57.29577951) / n + 2448200;
        choice[2] := n * (Jt - tau) * 57.29577951;
    end;

{============================================================}
procedure McAuliffe (var choice: planet; Jt: extended);

        {This procedure gives the ephimeral data }
        {for the asteroid McAuliffe.}

    var
        n,                      {Asteroid mean motion}
        tau: extended;          {time since perigee in days}

    begin
        choice[1] := 0;
        choice[2] := 283.00658;
        choice[3] := 15.59117 + 106.90142;
        choice[4] := 0.3694649;
        choice[5] := 1.8787307;
        choice[6] := 4.77743;
        choice[7] := 106.90142;
        n := sqrt(GMSol / (choice[5] * choice[5] * choice[5]));
        tau := -(choice[2] / 57.29577951) / n + 2448200;
        choice[2] := n * (Jt - tau) * 57.29577951;
    end;
```

```
{============================================================}
procedure Seleucus (var choice: planet; Jt: extended);

        {This procedure gives the ephimeral data }
        {for the asteroid Seleucus.}

    var
        n,                      {Asteroid mean motion}
        tau: extended;          {time since perigee in days}

    begin
        choice[1] := 0;
        choice[2] := 343.98574;
        choice[3] := 349.18871 + 218.12603;
        choice[4] := 0.4571202;
        choice[5] := 2.0325801;
        choice[6] := 5.93181;
        choice[7] := 218.12603;
        n := sqrt(GMSol / (choice[5] * choice[5] * choice[5]));
        tau := -(choice[2] / 57.29577951) / n + 2448200;
        choice[2] := n * (Jt - tau) * 57.29577951;
    end;

{============================================================}
procedure Eros (var choice: planet; Jt: extended);

        {This procedure gives the ephimeral data }
        {for the asteroid Eros.}

    var
        n,                      {Asteroid mean motion}
        tau: extended;          {time since perigee in days }

    begin
        choice[1] := 0;
        choice[2] := 170.451;
        choice[3] := 304.496 + 178.510;
        choice[4] := 0.2229;
        choice[5] := 1.4582;
        choice[6] := 10.832;
        choice[7] := 304.496;
        n := sqrt(GMSol / (choice[5] * choice[5] * choice[5]));
        tau := -(choice[2] / 57.29577951) / n + 2446600;
        choice[2] := n * (Jt - tau) * 57.29577951;
    end;
end.

{*************************************************************************}
unit Graphical;

        {This unit contains the mathematical procedures for projecting}
        {the paths of the planets and spacecraft in their own pericentric}
```

```
            {coordinate systems into the ecliptic plane.}
interface
        uses
            Ephemeris, Indirect;

        type
            matrix = array[1..7] of array[1..3] of extended;
            placement = array[1..7] of extended;
            vec = array[1..20] of extended;


        var
            PlPort: Grafport;
            MyWindow2: GrafPtr;
            MyWindow1: WindowPtr;
            wMgrPort: GrafPtr;
            MyRect1, MyRect2: Rect;
            event: EventRecord;
            ihor, iver: integer;
            scra, scrb, scrc, scrd: extended;      {Machine resident graphics}
            y: vec;
            rock1,                  {The graphical region for the departure planet}
            rock2,                  {The graphical region for the destination body}
            rock3,                  {The graphical region for Venus}
            rock4,                  {The graphical region for Mercury}
            rock5,                  {The graphical region for the spacecraft}
            rock6: RgnHandle;       {The graphical region for Mars}
            xlast,                  {Keeps track of a planets last x position in the local
                                    graphport coordinate system}
            ylast: placement;       {Keeps track of a planets last y position in the local
                                    graphport coordinate system}
            Trans: matrix;          {The transformation matrix}

        procedure Gscale (Xmin, Xmax, Ymin, Ymax: extended);
        procedure Gmove (x, y: extended);
        procedure Gdraw (x, y: extended);
        procedure Gxaxis (Ycoord, TickSpace, Origin: extended);
        procedure Gyaxis (Xcoord, TickSpace, Origin: extended);
        procedure Gerase;
        procedure Gwindow1 (left, top, right, bottom: integer; name: string);
        procedure Gwindow2 (left, top, right, bottom: integer; name: string);
        procedure Gimove (ix, iy: integer);
        procedure Gidraw (ix, iy: integer);
        procedure Gpause;
        procedure BuildTransform (var body: planet; var Trans: matrix);
        procedure CoordConvert (var body: planet; a, E, b, nu: extended; var Eclipt, Peri,
                                Trans: matrix; i: integer);
        procedure PlanetIcon1 (x1, y1: extended; var rock1: RgnHandle);
        procedure PlanetIcon2 (x1, y1: extended; var rock2: RgnHandle);
        procedure PlanetIcon3 (x1, y1: extended; var rock3: RgnHandle);
        procedure PlanetIcon4 (x1, y1: extended; var rock4: RgnHandle);
        procedure PlanetIcon5 (x1, y1: extended; var rock5: RgnHandle);
```

```
        procedure ShipIcon1 (x1, y1: extended; var rock6: RgnHandle);
        procedure GoGraphics (var a: extended; var b, E, nu: extended; i: integer; var body:
                                        planet; animate: boolean; var Trans: matrix; ship:
                                        boolean);

implementation

{Some of these graphical procedures were written by Paul Finlaysson}

procedure Gscale (Xmin, Xmax, Ymin, Ymax: extended);
        begin
                scra := ihor / (Xmax - Xmin);
                scrb := -scra * Xmin;
                scrc := iver / (Ymin - Ymax);
                scrd := -scrc * Ymax
        end;
{============================================================}
procedure Gmove (x, y: extended);
        begin
                MoveTo(trunc(scra * x + scrb), trunc(scrc * y + scrd))
        end;

{============================================================}
procedure Gdraw (x, y: extended);
        begin
                LineTo(trunc(scra * x + scrb), trunc(scrc * y + scrd))
        end;
{============================================================}
procedure Gxaxis (Ycoord, TickSpace, Origin: extended);
                var
                        iy: integer;
                        xi, x, dx, s: extended;
        begin
                iy := trunc(scrc * Ycoord + scrd);
                MoveTo(0, iy);
                Line(700, 0);
                if TickSpace > 0.0 then
                        begin
                                xi := Origin * scra + scrb;
                                dx := TickSpace * scra;
                                MoveTo(trunc(xi), iy - 8);
                                Line(0, 16);
                                s := -1;
                                repeat
                                        x := xi;
                                        repeat
                                                x := x + s * dx;
                                                MoveTo(trunc(x), iy - 4);
                                                Line(0, 8);
                                        until (x > 700.0) or (x < 0.0);
                                        s := s + 2;
                                until s > 1;
```

```
                end;
        end;
{=============================================================}
procedure Gyaxis (Xcoord, TickSpace, Origin: extended);
        var
                ix: integer;
                yi, y, dy, s: extended;
        begin
                ix := trunc(scra * Xcoord + scrb);
                MoveTo(ix, 0);
                Line(0, 500);
                if TickSpace > 0.0 then
                        begin
                                yi := Origin * scrc + scrd;
                                dy := TickSpace * scrc;
                                MoveTo(ix - 8, trunc(yi));
                                Line(16, 0);
                                s := -1;
                                repeat
                                        y := yi;
                                        repeat
                                                y := y + s * dy;
                                                MoveTo(ix - 4, trunc(y));
                                                Line(8, 0);
                                        until (y > 500.0) or (y < 0.0);
                                        s := s + 2;
                                until s > 1;
                        end;
        end;
{=============================================================}
procedure Gerase;
        var
                box: rect;
        begin
                box.top := 0;
                box.left := 0;
                box.bottom := iver;
                box.right := ihor;
                eraserect(box);
        end;
{=============================================================}
procedure Gwindow1 (left, top, right, bottom: integer; name: string);

        begin
                ihor := right - left;
                iver := bottom - top;
                InitGraf(@ThePort);
                GetWMgrPort(wMgrPort);
                MyRect1 := wMgrPort^.portRect;
                SetRect(MyRect1, left, top, right, bottom);
                MyWindow1 := NewWindow(nil, MyRect1, name, TRUE, 0, POINTER(-
                1), FALSE, 0);
```

```
            SetPort(MyWindow1);
            Gscale(0, 1, 0, 1);
        end;
{=============================================================}
procedure Gwindow2 (left, top, right, bottom: integer; name: string);

        begin
            ihor := right - left;
            writeln('have set ihor');
            readln;
            iver := bottom - top;
            writeln('have set iver');
            readln;
            SetRect(MyRect2, left, top, right, bottom);
            writeln('have set rect2');
            readln;
            New(MyWindow2);
            OpenPort(MyWindow2);
            writeln('have opened port2');
            readln;
            SetPort(MyWindow2);
            writeln('have set port2');
            readln;
            ClipRect(MyRect2);
            writeln('have clipped port2');
            readln;
            Gscale(0, 1, 0, 1);
            writeln('have scaled port2');
            readln;
        end;
{=============================================================}
procedure Gimove (ix, iy: integer);
        begin
            MoveTo(ix, iy)
        end;
{=============================================================}
procedure Gidraw (ix, iy: integer);
        begin
            LineTo(ix, iy)
        end;
{=============================================================}
procedure Gpause;

        var
                dummy: boolean;
        begin
            repeat
                dummy := GetNextEvent(everyevent, event);
                { This allows screenshots }
            until button;
            DisposeWindow(MyWindow1);
        end;
```

```
{=============================================================}
procedure BuildTransform (var body: planet; var Trans: matrix);

        {This procedure builds the matrix which serves }
        {as the transformation matrix between the }
        {ecliptic and pericentric coordinate system}

    var
        SmOmega: extended;          {argument of periapsis}

    begin
        SmOmega := body[3] - body[7];
        Trans[1, 1] := cos(body[7] / R) * cos(SmOmega / R) - sin(body[7] / R) *
                sin(SmOmega / R) * cos(body[6] / R);
        Trans[1, 2] := -cos(body[7] / R) * sin(SmOmega / R) - sin(body[7] / R) *
                cos(SmOmega / R) * cos(body[6] / R);
        Trans[1, 3] := sin(body[7] / R) * sin(body[6] / R);
        Trans[2, 1] := sin(body[7] / R) * cos(SmOmega / R) + cos(body[7] / R) *
                sin(SmOmega / R) * cos(body[6] / R);
        Trans[2, 2] := -sin(body[7] / R) * sin(SmOmega / R) + cos(body[7] / R) *
                cos(SmOmega / R) * cos(body[6] / R);
        Trans[2, 3] := -cos(body[7] / R) * sin(body[6] / R);
        Trans[3, 1] := sin(SmOmega / R) * sin(body[6] / R);
        Trans[3, 2] := cos(SmOmega / R) * sin(body[6] / R);
        Trans[3, 3] := cos(body[6] / R);
    end;

{=============================================================}
procedure CoordConvert (var body: planet; a, E, b, nu: extended; var Eclipt, Peri, Trans:
                matrix; i: integer);

        {This procedure converts from pericentric to ecliptic}
        {coordinates using the transformation matrix built }
        {BuildTransform.}
    begin
        Peri[i, 1] := (body[5] * (1 - body[4] * body[4]) / (1 + body[4] * cos(nu))) *
                cos(nu);
        Peri[i, 2] := (body[5] * (1 - body[4] * body[4]) / (1 + body[4] * cos(nu))) *
                sin(nu);
        Peri[i, 3] := 0;
        Eclipt[i, 1] := Trans[1, 1] * Peri[i, 1] + Trans[1, 2] * Peri[i, 2] + Trans[1,
                3] * Peri[i, 3];
        Eclipt[i, 2] := Trans[2, 1] * Peri[i, 1] + Trans[2, 2] * Peri[i, 2] + Trans[2,
                3] * Peri[i, 3];
        Eclipt[i, 3] := Trans[3, 1] * Peri[i, 1] + Trans[3, 2] * Peri[i, 2] + Trans[3,
                3] * Peri[i, 3];
    end;

{=============================================================}
procedure PlanetIcon1 (x1, y1: extended; var rock1: RgnHandle);

        { This procedure draws icons for the launch}
```

```
        { planets at launch and arrival dates, respectively}

    var
        angle, rd, x, y: extended;      {angle,radius,horizontal position,vertical
                                        position}
        Leftx, Lefty, Rightx, Righty: integer;
        tempRect: Rect;
        response: integer;

    begin
        angle := 0;
        rd := 0.05;
        x := x1 + rd * cos(angle);
        y := y1 + rd * sin(angle);
        xlast[1] := scra * x1 + scrb;
        ylast[1] := scrc * y1 + scrd;
        GMove(x, y);
        rock1 := NewRgn;
        OpenRgn;
        repeat
            angle := angle + 5 * 0.017453292;
            x := x1 + rd * cos(angle);
            y := y1 + rd * sin(angle);
            GDraw(x, y);
        until (angle >= (6.28));
        CloseRgn(rock1);
        FrameRgn(rock1);
        eraseRgn(rock1);
        InvertRgn(rock1);
    end;

    {PlanetIcon}
{============================================================}
procedure PlanetIcon2 (x1, y1: extended; var rock2: RgnHandle);

        { This procedure draws icons for the destination}
        { planets at launch and arrival dates, respectively}

    var
        angle, rd, x, y: extended;      {angle,radius,horizontal position,vertical
                                        position}
    begin
        angle := 0;
        rd := 0.05;
        x := x1 + rd * cos(angle);
        y := y1 + rd * sin(angle);
        xlast[2] := scra * x1 + scrb;
        ylast[2] := scrc * y1 + scrd;
        GMove(x, y);
        rock2 := NewRgn;
        OpenRgn;
        repeat
```

```
                angle := angle + 5 * 0.017453292;
                x := x1 + rd * cos(angle);
                y := y1 + rd * sin(angle);
                GDraw(x, y);
            until (angle >= (6.28));
            CloseRgn(rock2);
            InvertRgn(rock2);
        end;

{============================================================}
procedure PlanetIcon3 (x1, y1: extended; var rock3: RgnHandle);

            { This procedure draws icons for the launch}
            { planets at launch and arrival dates, respectively}

        var
            angle, rd, x, y: extended;      {angle,radius,horizontal position,vertical
                                             position}
        begin
            angle := 0;
            rd := 0.05;
            x := x1 + rd * cos(angle);
            y := y1 + rd * sin(angle);
            xlast[3] := scra * x1 + scrb;
            ylast[3] := scrc * y1 + scrd;
            GMove(x, y);
            rock3 := NewRgn;
            OpenRgn;
            repeat
                angle := angle + 5 * 0.017453292;
                x := x1 + rd * cos(angle);
                y := y1 + rd * sin(angle);
                GDraw(x, y);
            until (angle >= (6.28));
            CloseRgn(rock3);
            InvertRgn(rock3);
        end;

{============================================================}
procedure PlanetIcon4 (x1, y1: extended; var rock4: RgnHandle);

            { This procedure draws icons for the launch}
            { planets at launch and arrival dates, respectively}

        var
            angle, rd, x, y: extended;      {angle,radius,horizontal position,vertical
                                             position}
        begin
            angle := 0;
            rd := 0.05;
            x := x1 + rd * cos(angle);
            y := y1 + rd * sin(angle);
```

```
        xlast[4] := scra * x1 + scrb;
        ylast[4] := scrc * y1 + scrd;
        GMove(x, y);
        rock4 := NewRgn;
        OpenRgn;
        repeat
            angle := angle + 5 * 0.017453292;
            x := x1 + rd * cos(angle);
            y := y1 + rd * sin(angle);
            GDraw(x, y);
        until (angle >= (6.28));
        CloseRgn(rock4);
        InvertRgn(rock4);
    end;

{============================================================}
procedure PlanetIcon5 (x1, y1: extended; var rock5: RgnHandle);

        { This procedure draws icons for the launch}
        { planets at launch and arrival dates, respectively}

    var
        angle, rd, x, y: extended;      {angle,radius,horizontal position,vertical
                                         position}
    begin
        angle := 0;
        rd := 0.05;
        x := x1 + rd * cos(angle);
        y := y1 + rd * sin(angle);
        xlast[5] := scra * x1 + scrb;
        ylast[5] := scrc * y1 + scrd;
        GMove(x, y);
        rock5 := NewRgn;
        OpenRgn;
        repeat
            angle := angle + 5 * 0.017453292;
            x := x1 + rd * cos(angle);
            y := y1 + rd * sin(angle);
            GDraw(x, y);
        until (angle >= (6.28));
        CloseRgn(rock5);
        InvertRgn(rock5);
    end;

{============================================================}
procedure ShipIcon1 (x1, y1: extended; var rock6: RgnHandle);

        { This procedure draws icons for the launch}
        { planets at launch and arrival dates, respectively}

    var
```

```
        angle, rd, x, y: extended;      {angle,radius,horizontal position,vertical
                                        position}
    begin
        angle := 0;
        rd := 0.03;
        x := x1 + rd * cos(angle);
        y := y1 + rd * sin(angle);
        xlast[6] := scra * x1 + scrb;
        ylast[6] := scrc * y1 + scrd;
        GMove(x, y);
        rock6 := NewRgn;
        OpenRgn;
        repeat
            angle := angle + 5 * 0.017453292;
            x := x1 + rd * cos(angle);
            y := y1 + rd * sin(angle);
            GDraw(x, y);
        until (angle >= (6.28));
        CloseRgn(rock6);
        InvertRgn(rock6);
    end;

{============================================================}
procedure GoGraphics (var a: extended; var b, E, nu: extended; i: integer; var body: planet;
                      animate: boolean; var Trans: matrix; ship: boolean);

        {This procedure runs the graphics routines}

    var

        Peri,                   {The pericentral coordinate system}
        Eclipt: matrix;         {The ecliptical plane coordinate system projection}
        final: extended;        {The final eccentric anomoly after a 2*pi rotation}

    begin
        BuildTransform(body, Trans);
        CoordConvert(body, a, E, b, nu, Eclipt, Peri, Trans, i);
        GMove(Eclipt[i, 1], Eclipt[i, 2]);
        position(E, nu, body[5], body[4], body, ship);
        CoordConvert(body, a, E, b, nu, Eclipt, Peri, Trans, i);
        if (animate = FALSE) then
            Gdraw(Eclipt[i, 1], Eclipt[i, 2]);
        if (animate = TRUE) then
            case i of
                1:
                    begin
                        InvertRgn(rock1);
                        OffsetRgn(rock1, trunc((scra * Eclipt[i, 1] +
                                scrb)) - trunc(xlast[i]),
                                trunc((scrc * Eclipt[i, 2] +
                                scrd)) - trunc(ylast[i]));
                        InvertRgn(rock1);
```

```
                xlast[i] := scra * Eclipt[i, 1] + scrb;
                ylast[i] := scrc * Eclipt[i, 2] + scrd;
            end;
        2:
            begin
                InvertRgn(rock2);
                OffsetRgn(rock2, trunc((scra * Eclipt[i, 1] +
                        scrb)) - trunc(xlast[i]),
                        trunc((scrc * Eclipt[i, 2] +
                        scrd)) - trunc(ylast[i]));
                InvertRgn(rock2);
                xlast[i] := scra * Eclipt[i, 1] + scrb;
                ylast[i] := scrc * Eclipt[i, 2] + scrd;
            end;
        3:
            begin
                InvertRgn(rock3);
                OffsetRgn(rock3, trunc((scra * Eclipt[i, 1] +
                        scrb)) - trunc(xlast[i]),
                        trunc((scrc * Eclipt[i, 2] +
                        scrd)) - trunc(ylast[i]));
                InvertRgn(rock3);
                xlast[i] := scra * Eclipt[i, 1] + scrb;
                ylast[i] := scrc * Eclipt[i, 2] + scrd;
            end;
        4:
            begin
                InvertRgn(rock4);
                OffsetRgn(rock4, trunc((scra * Eclipt[i, 1] +
                        scrb)) - trunc(xlast[i]),
                        trunc((scrc * Eclipt[i, 2] +
                        scrd)) - trunc(ylast[i]));
                InvertRgn(rock4);
                xlast[i] := scra * Eclipt[i, 1] + scrb;
                ylast[i] := scrc * Eclipt[i, 2] + scrd;
            end;
        5:
            begin
                InvertRgn(rock5);
                OffsetRgn(rock5, trunc((scra * Eclipt[i, 1] +
                        scrb)) - trunc(xlast[i]),
                        trunc((scrc * Eclipt[i, 2] +
                        scrd)) - trunc(ylast[i]));
                InvertRgn(rock5);
                xlast[i] := scra * Eclipt[i, 1] + scrb;
                ylast[i] := scrc * Eclipt[i, 2] + scrd;
            end;
        6:
            begin
                InvertRgn(rock6);
                OffsetRgn(rock6, trunc((scra * Eclipt[i, 1] +
                        scrb)) - trunc(xlast[i]),
```

```
                                                    trunc((scrc * Eclipt[i, 2] +
                                                    scrd)) - trunc(ylast[i]));
                                            InvertRgn(rock6);
                                            xlast[i] := scra * Eclipt[i, 1] + scrb;
                                            ylast[i] := scrc * Eclipt[i, 2] + scrd;

                                    end;
                            end;
        end;
end.

{*************************************************************************}
unit Indirect;

interface
        uses
                Ephemeris;

        const
                GM = 3.96396415708E-14;         {au3/s2}
                R = 57.29577951;
                root = 1.990970657E-7;
                stepsize = 2;
        var
                t,                      {time since perifocal passage}
                newtime,                {one day (step size)}
                w,                      {mean motion}
                delta: extended;        {Newton's method error term}


        procedure position (var EA, Nu: extended; a, ecc: extended; body: planet; ship:
                                boolean);

implementation

procedure position (var EA, Nu: extended; a, ecc: extended; body: planet; ship: boolean);

        var
                verify,                 {verifies the quadrant of nu}
                Mo,                     {Previous value for Mean in iteration}
                E,                      {temporary holding variable for EA}
                rotations,              {The number of 2*pi multiples in E}
                term,                   {just a silly simplification term}
                Mean: extended;         {mean anomaly}
                counter: integer;       {counter}

        begin
                if (ship = TRUE) then
                        begin
                                t := (sqrt(a * a * a) / root) * (EA - ecc * sin(EA));
                                newtime := t + stepsize * 86400;
                                w := sqrt(GM) / sqrt(a * a * a);
```

```
                        Mean := w * newtime;
                        counter := 0;
                        repeat
                                counter := counter + 1;
                                delta := (-EA + ecc * sin(EA) + Mean) / (1 - ecc *
                                        cos(EA));
                                EA := EA + delta;
                        until ((delta < 0.0000000000001) or (counter = 40));
                        term := (cos(EA) - ecc) / (1 - ecc * cos(EA));
                        Nu := pi / 2 - ArcTan(term / (sqrt(1 - term * term)));
                        verify := EA - trunc(EA / (2 * pi)) * 2 * pi;
                        if (verify > pi) then
                                Nu := 2 * pi - Nu;
                end
        else
                begin
                        if (body[1] <> 0) then
                                Mean := pi / 180 * (body[2] - body[3])
                        else
                                Mean := body[2] / R;
                        if (Mean < 0) then
                                Mean := Mean + (2 * 3.141592654);
                        E := Mean + ecc * (sin(Mean) + ecc * sin(2 * Mean) / 2);
                        repeat
                                Mo := E - ecc * sin(E);
                                delta := (Mean - Mo) / (1 - ecc * cos(E));
                                E := E + delta;
                        until (abs(delta) <= 1E-14);
                        Nu := 2 * ArcTan(sqrt((1 + ecc) / (1 - ecc)) * ((sin(E / 2) /
                                cos(E / 2))));
                        if (Nu < 0) then
                                Nu := Nu + 2 * pi;
                        rotations := E / (2 * pi);
                        EA := E - trunc(rotations) * 2 * pi;
                end;
        end;
end.

{*************************************************************************}
unit Quadrant;


interface
        uses
                Ephemeris, indirect;


        procedure QuadrantCheck (var Omega: extended; delLong, Lat1, Lat2, L1, L2:
                                extended);
        procedure PieQuadrant (var Pie: extended; CelesLat, long: storage; inclination,
                                Omega: extended);
```

```
implementation

    procedure QuadrantCheck (var Omega: extended; delLong, Lat1, Lat2, L1, L2:
                                extended);

        { This procedure utilizes geometric arguments within}
        { the celestial sphere to ensure that Omega has been}
        { assigned in the correct quadrant}

    begin
        delLong := delLong * R;
        Lat1 := Lat1 * R;
        Lat2 := Lat2 * R;
        L1 := L1 * R;
        L2 := L2 * R;
        if (delLong <= 180) then
            begin
                if ((Lat1 >= 0) and (Lat2 >= 0)) then
                    begin
                        if ((L2 <= 180) and ((L2 < Omega) and
                            (Omega < (L2 + 180)))) then
                            Omega := Omega + 180;
                        if ((L2 > 180) and (((L2 <= Omega) and
                            (Omega <= 360)) or ((0 <= Omega)
                            and (Omega < (L2 + 180 - 360)))))
                        then
                            Omega := Omega + 180;
                    end;
                if ((Lat1 <= 0) and (Lat2 <= 0)) then
                    begin
                        if ((L2 <= 180) and not ((L2 <= Omega) and
                            (Omega < (L2 + 180)))) then
                            Omega := Omega + 180;
                        if ((L2 > 180) and not (((L2 <= Omega) and
                            (Omega <= 360)) or ((0 <= Omega)
                            and (Omega < (L2 + 180 - 360)))))
                        then
                            Omega := Omega + 180;
                    end;
                if ((Lat1 > 0) and (Lat2 > 0)) then
                    begin
                        if ((L1 <= 180) and ((L1 <= Omega) and
                            (Omega < (L1 + 180)))) then
                            Omega := Omega + 180;
                        if ((L1 > 180) and (((L1 <= Omega) and
                            (Omega <= 360)) or ((0 <= Omega)
                            and (Omega < (L1 + 180 - 360)))))
                        then
                            Omega := Omega + 180;
                    end;
                if ((Lat1 < 0) and (Lat2 > 0)) then
```

```
                    begin
                        if ((L1 <= 180) and not ((L1 <= Omega) and
                            (Omega < (L1 + 180)))) then
                            Omega := Omega + 180;
                        if ((L1 > 180) and not (((L1 <= Omega) and
                            (Omega <= 360)) or ((0 <= Omega)
                            and (Omega < (L1 + 180 - 360)))))
                        then
                            Omega := Omega + 180;
                    end;
            end
    else
            begin
                if ((Lat2 >= 0) and (Lat1 >= 0)) then
                    begin
                        if ((L1 <= 180) and ((L1 < Omega) and
                            (Omega < (L1 + 180)))) then
                            Omega := Omega + 180;
                        if ((L1 > 180) and (((L1 <= Omega) and
                            (Omega <= 360)) or ((0 <= Omega)
                            and (Omega < (L1 + 180 - 360)))))
                        then
                            Omega := Omega + 180;
                    end;
                if ((Lat2 <= 0) and (Lat1 <= 0)) then
                    begin
                        if ((L1 <= 180) and not ((L1 <= Omega) and
                            (Omega < (L1 + 180)))) then
                            Omega := Omega + 180;
                        if ((L1 > 180) and not (((L1 <= Omega) and
                            (Omega <= 360)) or ((0 <= Omega)
                            and (Omega < (L1 + 180 - 360)))))
                        then
                            Omega := Omega + 180;
                    end;
                if ((Lat2 > 0) and (Lat1 > 0)) then
                    begin
                        if ((L2 <= 180) and ((L2 <= Omega) and
                            (Omega < (L2 + 180)))) then
                            Omega := Omega + 180;
                        if ((L2 > 180) and (((L2 <= Omega) and
                            (Omega <= 360)) or ((0 <= Omega)
                            and (Omega < (L2 + 180 - 360)))))
                        then
                            Omega := Omega + 180;
                    end;
                if ((Lat2 < 0) and (Lat1 > 0)) then
                    begin
                        if ((L2 <= 180) and not ((L2 <= Omega) and
                            (Omega < (L2 + 180)))) then
                            Omega := Omega + 180;
```

```
                                        if ((L2 > 180) and not (((L2 <= Omega) and
                                                (Omega <= 360)) or ((0 <= Omega)
                                                and (Omega < (L2 + 180 - 360)))))
                                        then
                                                Omega := Omega + 180;
                                end;
                        end;
        end;
{=============================================================}
procedure PieQuadrant (var Pie: extended; CelesLat, long: storage; inclination, Omega:
                                extended);

        var
                PieSin,                 {The sine of the sum of Omega and SmOmega}
                PieCos: extended;       {The cosine of the sum of Omega and SmOmega}

        begin
                PieSin := sin(CelesLat[2]) / sin(inclination);
                if (Celeslat[2] = 0) then
                        PieSin := sin(CelesLat[1]) / sin(inclination);
                PieCos := cos(Long[2] - Omega / R) * cos(CelesLat[2]);
                if (Celeslat[2] = 0) then
                        PieCos := cos(Long[1] - Omega / R) * cos(CelesLat[1]);
                if ((PieSin > 0) and (PieCos < 0)) then
                        Pie := pi - Pie;
                if ((PieSin < 0) and (PieCos < 0)) then
                        Pie := pi - Pie;
        end;
end.

{*************************************************************************}
unit verification;

interface

        uses
                Ephemeris, indirect;

        function tan (argument: extended): extended;
        function asin (argument: extended): extended;
        function acos (argument: extended): extended;

        procedure Verify (a, dt, H, p, ecc: extended; var eccentric1, eccentric2: extended;
                                var radius: storage; var nu1, nu2: extended);
        procedure HyperVerify (var nu1, nu2: extended; radius: storage; ecc, H, p, a, dt:
                                extended);

implementation

{***************************** Functions ********************************}

function tan (argument: extended): extended;
```

```
                { This function returns the tangent of the input angle }

        begin
                tan := sin(argument) / cos(argument);
        end;

{============================================================}
function asin (argument: extended): extended;

                {This function returns the arcsine value of the input angle in radians}

        begin
                asin := ArcTan((argument) / (sqrt(1 - argument * argument)));
        end;

{============================================================}
function acos (argument: extended): extended;

                {This function returns the arcosine value of the input angle in radians}

        begin
                acos := pi / 2 - ArcTan((argument) / (sqrt(1 - argument * argument)));
        end;

{************************ Procedures ***********************************}

procedure Verify (a, dt, H, p, ecc: extended; var eccentric1, eccentric2: extended; var
                        radius: storage; var nu1, nu2: extended);

                {This procedure verifies that the value for nu1 has been assigned}
                {to the proper quadrant in the case of a nonhyperbolic trajectory.}
                {The time of flight is solved for using Kepler's equations, and the }
                {quadrant of nu1 is alterred until this time matches the input time }
                {of flight.  nu2 is assigned, dependant upon the value of nu1}

        var
                time1,          {initial time since perigee passage}
                time2,          {final time since perigee passage}
                time,           {time of flight}
                Period,         {orbit period}
                w,              {mean motion}
                arg1,           {cosine of eccentric anomoly at r1}
                arg2,           {cosine of eccentric anomoly ar r2}
                check1,
                check2: extended;{comparison values for time of flight}
                loop: integer;  {an indexing variable}

        begin
                loop := 0;
                nu1 := acos((p / radius[1] - 1) / ecc);
                repeat
```

```
                loop := loop + 1;
                case loop of

                        1:
                                nu1 := nu1;
                        2:
                                nu1 := 2 * pi - nu1;
                        3:
                                nu1 := 2 * pi - nu1;
                end;
                nu2 := nu1 + H;
                if (nu2 > 2 * pi) then
                        nu2 := nu2 - 2 * pi;
                arg1 := (ecc + cos(Nu1)) / (1 + ecc * cos(Nu1));
                arg2 := (ecc + cos(Nu2)) / (1 + ecc * cos(Nu2));
                eccentric1 := acos(arg1);
                eccentric2 := acos(arg2);
                if (Nu1 > pi) then
                        eccentric1 := 2 * pi - eccentric1;
                if (Nu2 > pi) then
                        eccentric2 := 2 * pi - eccentric2;
                Period := (2 * pi * sqrt(a * a * a)) / (sqrt(GM));
                w := 2 * pi / Period;
                time1 := (eccentric1 - ecc * sin(eccentric1)) / w;
                time2 := (eccentric2 - ecc * sin(eccentric2)) / w;
                time := time2 - time1;
                if (eccentric1 > eccentric2) then
                        time := Period + time;
                check1 := time / 3600 / 24;
                check2 := dt / 3600 / 24;
        until ((abs(check1 - check2) < 0.001) or (loop = 3));
        if (loop = 3) then
                writeln('Not!!!!!!!(????)');
    end;
{============================================================}
procedure HyperVerify (var nu1, nu2: extended; radius: storage; ecc, H, p, a, dt:
                        extended);

        {This procedure uses time of flight computations}
        {in order to verify that the correct sign of nu1}
        {has been selected in the case of a hyperbolic}
        {trajectory. Corrections are made if  they are}
        { needed as done in Verify, above.}

    var
        loop: integer;          {an indexing variable}
        F1,                     {the initial hyperbolic eccentric anomaly}
        F2,                     {the final hyperbolic eccentric anomaly}
        t1,                     {the initial time past perigee}
        t2,                     {the final time past perigee}
        time,                   {the time of flight}
```

```
        Numax,                  {The maximum true anomaly possible within the
                                hyperbola}
        argument: extended;     {Holding value for arcsinh terms}

    begin
        loop := 0;
        nu1 := pi / 2 - ArcTan(((p / radius[1] - 1) / ecc) / (sqrt(1 - ((p / radius[1] - 1)
            / ecc * (p / radius[1] - 1) / ecc))));
        Numax := pi / 2 - ArcTan((-1 / ecc) / (sqrt(1 - ((-1 / ecc) * (-1 / ecc)))));
        if ((nu1 + H) > Numax) then
            begin
                nu1 := -nu1;
                nu2 := nu1 + H;
            end
        else
            begin
                repeat
                    loop := loop + 1;
                    case loop of

                        1:
                            nu1 := nu1;
                        2:
                            nu1 := -nu1;
                        3:
                            nu1 := -nu1;
                    end;
                    nu2 := nu1 + H;
                    argument := (sqrt(ecc * ecc - 1) * sin(nu1)) / (1 +
                            ecc * cos(nu1));
                    F1 := ln(argument + sqrt(argument * argument +
                        1));
                    t1 := (ecc * ((exp(F1) - exp(-F1)) / 2) - F1) /
                        (sqrt(GM / (abs(a * a * a))));
                    argument := (sqrt(ecc * ecc - 1) * sin(nu2)) / (1 +
                            ecc * cos(nu2));
                    F2 := ln(argument + sqrt(argument * argument +
                        1));
                    t2 := (ecc * ((exp(F2) - exp(-F2)) / 2) - F2) /
                        (sqrt(GM / (abs(a * a * a))));
                    time := t2 - t1;
                until ((abs(dt - time) < 0.00001) or (loop = 3));
                if (loop = 3) then
                    writeln('Not!!!!!!!(????)');
            end;
    end;
end.

{*************************************************************************}
unit Parking;
```

```
interface

    uses
        Ephemeris, indirect, Graphical, verification;

    const
        GMearth = 398600;                                   {km^3/s^2}

    var
        EclipVel: matrix;       {Spacecraft's velocity vector in the ecliptic
                                coordinate system}
        tilt,                   {earth's tilt}
        parki: extended;        {inclination of Cape Canaveral parking orbit}
        EclipVpx,               {Planet's x velocity}
        EclipVpy,               {Planet's y velocity}
        EclipVPz: storage;      {Planet's z velocity}
        long: storage;          {Longitude}

    procedure EarthParking (var Rpark1, COne, DeltaV1: extended);
    procedure ArrivalParking (var CTwo, DeltaV2: extended);

implementation

procedure EarthParking (var Rpark1, COne, DeltaV1: extended);

        {This procedure allows the user to select the}
        {altitude of the circular parking orbit}
    var
        Vinfx,          {x-component of hyperbolic excess velocity}
        Vinfy,          {y-component of hyperbolic excess velocity}
        Vinfz,          {z-component of hyperbolic excess velocity}
        Vinearthx,      {x-component of hyperbolic excess velocity (earth centered
                        coord sys)}
        Vinearthy,      {y-component of hyperbolic excess velocity (earth centered
                        coord sys)}
        Vinearthz,      {z-component of hyperbolic excess velocity (earth centered
                        coord sys)}
        Vinf,           {hyperbolic excess velocity}
        B,              {Angle between north pole and the hyperbolic excess
                        velocity}
        Bhold,          {Angle between north pole and the hyperbolic excess
                        velocity}
        ahyp,           {semi-major axis of escape hyperbola}
        ehyp,           {eccentricity of escape hyperbola}
        rho,            {pi/2 - turn angle}
        relinc,         {angle between parking orbit plane and hyperbolic escape
                        plane}
        DeltaVinc,      {Delta V required for plane change}
        Omeg1,          {Ascending node of parking orbit measured relative to the
                        hyperbolic velocity vector}
```

```
        Omeg2,          {Ascending node of parking orbit measured relative to the
                        hyperbolic velocity vector}
        omeg,           {}
        term,           {long term in an equation}
        Vreq1,          {Required departure velocity}
        Vcirc1: extended;{Circular parking orbit velocity}

begin
        writeln('Enter the desired circular earth parking orbit altitude in km.');
        readln(Rpark1);
        parki := 28.5 / R;
        Vreq1 := sqrt(2 * GMearth / (Rpark1 + 6378.145) + COne);
        Vcirc1 := sqrt(GMearth / (Rpark1 + 6378.145));
        DeltaV1 := Vreq1 - Vcirc1;
        Vinfx := EcliptVel[1, 1] - EcliptVPx[1];
        Vinfy := EcliptVel[1, 2] - EcliptVPy[1];
        Vinfz := EcliptVel[1, 3] - EcliptVPz[1];
        Vinearthx := -Vinfx * sin(long[1]) + (Vinfy * cos(tilt) - Vinfz * sin(tilt)) *
                        cos(long[1]);
        Vinearthy := -Vinfx * cos(long[1]) + (-Vinfy * cos(tilt) + Vinfz * sin(tilt)) *
                        sin(long[1]);
        Vinearthz := Vinfy * sin(tilt) + Vinfz * cos(tilt);
        Vinf := sqrt(Vinearthx * Vinearthx + Vinearthy * Vinearthy + Vinearthz *
                        Vinearthz);
        B := acos(Vinearthz / Vinf);
        Bhold := B;
        if (B > pi / 2) then
                Bhold := pi - Bhold;
        writeln('B: ', Bhold * R : 1 : 5);
        if ((Bhold + parki) < pi / 2) then
                parki := 51.0 / R;
        if ((Bhold + parki) < pi / 2) then
                begin
                        writeln;
                        writeln('Nontangential injection.  Plane change reqiured.');
                        writeln;
                        ahyp := -GMearth / (Vinf * Vinf);
                        ehyp := 1 - Rpark1 / ahyp;
                        rho := acos(1 / ehyp);
                        if ((Bhold + parki) > (pi / 2 - rho)) then
                                begin
                                        relinc := asin(cos(Bhold + parki) / cos(pi / 2 -
                                                rho));
                                        DeltaVinc := Vinf * sin(relinc / 2);
                                        writeln('Inclination change: ', relinc * R : 1 :
                                                5);
                                        writeln('Delta V for plane change: ',
                                                DeltaVinc : 1 : 5);
                                end
                        else
                                writeln('This is a non-horizontal injection case.');
                end;
```

```
            if ((Bhold + parki) >= pi / 2) then
                begin
                    writeln;
                    writeln('Tangential injection.  No plane change reqiured.');
                    writeln;
                    parki := pi / 2 - Bhold;
                    if (parki < 28.5 / R) then
                        parki := 28.5 / R;
                    term := (1 / (tan(B) * tan(parki)));
                    if (term > 1) then
                        begin
                            Omeg1 := pi + pi / 2;
                            Omeg2 := 2 * pi - pi / 2;
                        end;
                    if (term < -1) then
                        begin
                            Omeg1 := pi - pi / 2;
                            Omeg2 := 2 * pi + pi / 2;
                        end;
                    if (term > -1) and (term < 1) then
                        begin
                            Omeg1 := pi + asin(1 / (tan(B) * tan(parki)));
                            Omeg2 := 2 * pi - asin(1 / (tan(B) *
                                tan(parki)));
                        end;
                    if (Omeg1 > 2 * pi) then
                        Omeg1 := Omeg1 - 2 * pi;
                    if (Omeg2 > 2 * pi) then
                        Omeg2 := Omeg2 - 2 * pi;
                    writeln('Ascending Node of Parking Orbit: ', Omeg1 * R : 1
                        : 5);
                    writeln('Ascending Node of Parking Orbit: ', Omeg2 * R : 1
                        : 5);
                end;
            writeln('Inclination of Parking Orbit: ', parki * R : 1 : 5);
        end;

{================================================================}
procedure ArrivalParking (var CTwo, DeltaV2: extended);

            {This procedure finds the DeltaV needed to}
            {rendevous with the target asteroid.}

        begin
            DeltaV2 := sqrt(CTwo);
        end;
end.

{*****************************************************************}
{-----------------------------------------------------------------}
{************************ END OF UNITS ***************************}
```

```
type
        name = array[1..5] of string;
        vector = array[1..3] of extended;

const
        AU = 1.4959965E+8;          {km/au}

var
        path,                       {the path of the transport vehicle}
        spacecraft,                 {the transport vehicle}
        other,                      {Mercury}
        other2,                     {Mars}
        flyby,                      {Venus}
        departure,                  {The planet of departure}
        target: planet;             {The target asteroid}
        x,                          {The Battin independant variable}
        L,                          {Initial guess for x}
        m,                          {Transfer time equation term}
        Rop,                        {Radius to mean point between r1 and r2}
        c,                          {Cord between endpoints of r1 and r2}
        s,                          {Semi-perimeter of triangle r1,r2,c}
        a,                          {Semi major axis}
        ecc,                        {Eccentricity of the transfer trajectory}
        e1,                         {Battin variable used to calculate eccentricity}
        e2,                         {Battin variable used to calculate eccentricity}
        eccentric1,                 {the eccentric anomoly at r1}
        eccentric2,                 {the eccentric anomoly at r2}
        eccalt,                     {eccentric1 holding variable for drawing the
                                    ship's trajectory}
        Xi,                         {Battin continued fraction}
        K,                          {Continued fraction used to solve the cubic}
        p,                          {Transfer orbit parameter}
        angmom,                     {Transfer orbit angular momentum}
        n,                          {Mean daily motion of the spacecraft}
        COne,                       {Square of the hyperbolic excess speed at launch}
        CTwo,                       {Square of the hyperbolic excess speed at target}
        VOne,                       {Heliocentric speed at departure}
        VTwo,                       {Heliocentric speed at target}
        gamma1,                     {Spacecraft path angle at launch}
        gamma2,                     {Spacecraft path angle at approach}
        H,                          {Heliocentric angle traverse parameter}
        launchT,                    {The date of departure}
        arriveT,                    {The outbound trip time of flight}
        arriveT2,                   {The return trip time of flight}
        JlaunchT,                   {The Julian date of departure from earth}
        JLaunchT2,                  {The Julian date of departure from Mars}
        Jcentury1,                  {Julian centuries from Jan. 1, 1900 to launch}
        JarriveT,                   {The Julian date of arrival at Mars}
        JarriveT2,                  {The Julian date of arrival at earth}
        Jcentury2,                  {Julian centuries from Jan. 1, 1900 to arrival}
```

```
        Jcentury,           {Julian centuries from Jan. 1,1900 to presently
                             considered date}
        CalDateL,           {Calendar Launch Date}
        CalDateA,           {Calendar Arrival Date}
        duration,           {The date of arrival at the target planet}
        nu1,                {True anomaly at time of launch}
        nu2,                {True anomaly at time of landing}
        nualt,              {Holding variable for nu1 or nu2 while drawing the
                             ship's path}
        inclination,        {inclination of the transfer plane}
        xpo,                {spacecraft's x position}
        ypo,                {spacecraft's y position}
        RPark1,             {Radius of the parking orbit around the departure
                             planet}
        DeltaV1,            {Delta V to leave the departure planet}
        DeltaV2,            {Delta V to rendezvous with target}
        Days,               {A counting variable that adds up towards the time of
                             flight}
        dt: extended;       {Time of flight}

        CelesLat,           {The celestial latitude}
        radius,             {Radius of the planetary orbit}
        Vplanet,            {Planet velocity}
        phi,                {Planet flight path angle}
        E,                  {eccentric anomaly}
        Nu,                 {true anomaly}
        SMinAx,             {Semi Minor Axis}
        Vpx,                {Planet x-axis velocity component}
        Vpy: storage;       {Planet y-axis velocity component}
        check,              {Like Days}
        i: integer;         {Indexing variables}
        place: name;        {Launch and target planets, enterred from keyboard}
        valid: boolean;     {Checks for proper input format of destination
                             asteroid designation}


{*************************  Procedures  ***********************}
procedure ChooseAsteroid (var choice: planet; entry: string; Jt: extended; var valid:
                          boolean);

        {This procedure checks input responses to}
        {determine which planet they correspond}
        {to, and sets the departure and target}
        {arrays appropriately}


    var
        index,                  {The planet's number from the sun}
        i: integer;             {An indexing variable}

    begin
        for i := 1 to 7 do
```

```
                choice[i] := 0;
        index := 0;
        valid := TRUE;
        if (entry = '1982 XB') then
                XB(choice, Jt)
        else if (entry = 'Orpheus') then
                Orpheus(choice, Jt)
        else if (entry = 'McAuliffe') then
                McAuliffe(choice, Jt)
        else if (entry = 'Seleucus') then
                Seleucus(choice, Jt)
        else if (entry = 'Eros') then
                Eros(choice, Jt)
        else
                begin
                        write('Please enter 1982 XB, Orpheus, McAuliffe,');
                        writeln('or Seleucus.  Be careful of ');
                        writeln('spelling, and capitilization ');
                        valid := FALSE;
                end;
end;

{============================================================}
procedure Convert (var date: extended);

        {This procedure converts any date enterred in }
        {the standard calender into a Julian date.}

    var
        Month,          {The portion of the enterred date that represents the month}
        Day,            {The portion of the enterred date that represents the day}
        Year,           {The protion of the enterred date that represents the year}
        a, b: extended;
    begin
        Month := trunc(date);
        Day := trunc(100 * (date - Month));
        Year := Round((1000000 * date) - (Month * 1000000) - (Day * 10000));
        if (month <= 2) then
                begin
                        year := year - 1;
                        month := month + 12;
                end;
        a := year / 100;
        b := 2 - a + a / 4;
        date := trunc(365.25 * year) + trunc(30.6001 * (month + 1)) + day +
                1720994.5 + b;
    end;

{============================================================}

procedure InvConvert (var date: extended);
```

```
        {This procedure converts any Julian date into a calendar date. }
        {Taken from Astronomical Formulae for Calculators, Meeus, p. 24.}

    var
        Month,      {The portion of the enterred date that represents the month}
        Day,        {The portion of the enterred date that represents the day}
        Year,       {The portion of the enterred date that represents the year}
        a, b, c, d, e, f: extended;

    begin
        f := date + 0.5;
        a := trunc(f);
        f := f - a;
        if (a >= 2299161) then
            begin
                b := (a - 1867216.25) / 36524.25;
                a := a + (1 + b - b / 4);
            end;
        b := a + 1524;
        c := trunc((b - 122.1) / 365.25);
        d := trunc(365.25 * c);
        e := trunc((b - d) / 30.6001);
        Day := b - d - trunc(30.6001 * e) + f;
        if (e <= 13) then
            Month := e - 1
        else
            Month := e - 13;
        if (Month <= 2) then
            Year := c - 4715
        else
            Year := c - 4716;
        date := trunc(Month) + 0.01 * round(Day) + 0.000001 * trunc(Year);
    end;

{============================================================}
procedure CaseEntry (var departure, target: planet; var launchT, JlaunchT, arriveT,
            JarriveT, dt, Jcentury1, Jcentury2: extended; var place: name; var
            valid: boolean);

        {This procedure prompts for the names of the}
        {target and launch planets and the dates of}
        {these events.  It also calls the procedure}
        {ChoosePlanet to read the planetary orbital}
        {elements and the procedure Convert to}
        {convert all dates to the Julian calendar.}

    var
        i: integer;     {An indexing variable}

    begin
        writeln('Enter the launch date as mm.ddyyyy or as a Julian date.');
        readln(launchT);
```

```
            JlaunchT := launchT;
            if ((JlaunchT / 1000) < 1) then
                  Convert(JlaunchT);
            writeln('Enterred JlaunchT is', JlaunchT : 8 : 0);
            Jcentury1 := (JlaunchT - 2415021) / (36525);
            place[1] := 'earth';
            for i := 1 to 7 do
                  departure[i] := 0;
            earth(departure, Jcentury1);
            writeln('Enter the desired time of flight to destination in days.');
            readln(arriveT);
            JarriveT := JlaunchT + arriveT;
            Jcentury2 := (JarriveT - 2415021) / (36525);
            writeln('Please enter the designation of the asteroid you wish to consider.');
            repeat
                  readln(place[2]);
                  ChooseAsteroid(target, place[2], JarriveT, valid);
            until (valid = TRUE);
            dt := (JarriveT * 24 * 3600) - (JlaunchT * 24 * 3600);
            place[3] := 'Venus';
            for i := 1 to 7 do
                  flyby[i] := 0;
            Venus(flyby, Jcentury1);
            place[4] := 'Mercury';
            for i := 1 to 7 do
                  other[i] := 0;
            Mercury(other, Jcentury1);
            place[5] := 'Mars';
            for i := 1 to 7 do
                  other2[i] := 0;
            Mars(other2, Jcentury1);
      end;

{============================================================}
procedure ComputePositions (var body: planet; time: extended; index: integer; var check:
                        integer; var long, CelesLat, radius, phi, Vplanet, EA, nu,
                        SMinAx, VPr, VPt: storage; var EcliptVPx, EcliptVPy,
                        EcliptVPz: storage);

            { This procedure finds the positions of each}
            {of the planets within their orbits on the}
            {dates specified.}


      var
            Trans,          {a temporary transformation matrix}
            Peri,           {The pericentral coordinate system}
            Eclipt: matrix; {The ecliptical plane coordinate system projection}
            M,              {mean anomoly}
            Mo,             {storage variable}
            D,              {slope}
            x,              {position variable}
```

```
        rotations,      {number of body rotational periods since epoch}
        E,              {Eccentric Anomaly}
        SmOmega,        {argument of the periapsis}
        final,          {final nu angle}
        initx,          {initial x}
        inity,          {initial y}
        arg: extended;  {a holding variable for arcsin argument}

begin
        check := check + 1;
        if (body[1] <> 0) then
                M := pi / 180 * (body[2] - body[3])
        else
                M := body[2] / R;
        if (M < 0) then
                M := M + (2 * 3.141592654);
        E := M + body[4] * (sin(M) + body[4] * sin(2 * M) / 2);
        repeat
                Mo := E - body[4] * sin(E);
                D := (M - Mo) / (1 - body[4] * cos(E));
                E := E + D;
        until (abs(D) <= 1E-14);
        nu[index] := 2 * ArcTan(sqrt((1 + body[4]) / (1 - body[4])) * ((sin(E / 2) /
                        cos(E / 2))));
        if (nu[index] < 0) then
                nu[index] := nu[index] + 2 * pi;
        SmOmega := (body[3] - body[7]) / R;
        CelesLat[index] := ArcTan((sin(SmOmega + nu[index]) * sin(body[6] / R))
                                / (sqrt(1 - (sin(SmOmega + nu[index]) * sin(body[6]
                                / R)) * (sin(SmOmega + nu[index]) * sin(body[6] /
                                R)))));
        if body[6] = 0 then
                CelesLat[index] := 0;
        radius[index] := body[5] * (1 - body[4] * body[4]) / (1 + body[4] *
                                cos(nu[index]));
        x := radius[index] / body[5];
        phi[index] := pi / 2 - ArcTan((sqrt((1 - body[4] * body[4]) / (x * (2 - x)))) /
                        sqrt(1 - (sqrt((1 - body[4] * body[4]) / (x * (2 - x)))) *
                        (sqrt((1 - body[4] * body[4]) / (x * (2 - x))))));
        Vplanet[index] := sqrt(GM * (2 - x) / radius[index]) * AU;
        if (nu[index] < pi) then
                phi[index] := -phi[index];
        SMinAx[index] := body[5] * sqrt(1 - body[4] * body[4]);
        rotations := E / (2 * pi);
        EA[index] := E - trunc(rotations) * 2 * pi;

        if ((check = 1) or (check = 2)) then    {Draws radial lines to initial and final
                                                 positions in ecliptic plane}
                begin
                        BuildTransform(body, Trans);
                        CoordConvert(body, body[5], EA[index], SMinAx[index],
                                                nu[index], Eclipt, Peri, Trans, index);
```

```
                    GMove(0, 0);
                    Gdraw(Eclipt[index, 1], Eclipt[index, 2]);
                end;

{--------Draws orbits of all the planets--------}

            if (check = 1) then
                begin
                    initx := Eclipt[index, 1];
                    inity := Eclipt[index, 2];
                    GMove(Eclipt[index, 1], Eclipt[index, 2]);
                    final := nu[index] + 2 * pi;
                    repeat
                        nu[index] := nu[index] + 5 * 0.017453292;
                        CoordConvert(body, body[5], EA[index],
                                SMinAx[index], nu[index], Eclipt,
                                Peri, Trans, index);
                        Gdraw(Eclipt[index, 1], Eclipt[index, 2]);
                    until (nu[index] >= final);
                    Eclipt[index, 1] := initx;
                    Eclipt[index, 2] := inity;
                    xlast[1] := Eclipt[index, 1];
                    ylast[1] := Eclipt[index, 2];
                    PlanetIcon1(Eclipt[index, 1], Eclipt[index, 2], rock1);
                    Eclipt[6, 1] := Eclipt[index, 1];
                    Eclipt[6, 2] := Eclipt[index, 2];
                    ShipIcon1(Eclipt[6, 1], Eclipt[6, 2], rock6);
                    InvertRgn(Rock6);
                end;

            if (check = 2) then
                begin
                    BuildTransform(body, Trans);
                    CoordConvert(body, body[5], EA[index], SMinAx[index],
                            nu[index], Eclipt, Peri, Trans, index);
                    initx := Eclipt[index, 1];
                    inity := Eclipt[index, 2];
                    GMove(Eclipt[index, 1], Eclipt[index, 2]);
                    final := nu[index] + 2 * pi;
                    repeat
                        nu[index] := nu[index] + 5 * 0.017453292;
                        CoordConvert(body, body[5], EA[index],
                                SMinAx[index], nu[index], Eclipt,
                                Peri, Trans, index);
                        Gdraw(Eclipt[index, 1], Eclipt[index, 2]);
                    until (nu[index] >= final);
                    Eclipt[index, 1] := initx;
                    Eclipt[index, 2] := inity;
                    xlast[2] := Eclipt[index, 1];
                    ylast[2] := Eclipt[index, 2];
                    PlanetIcon2(Eclipt[index, 1], Eclipt[index, 2], rock2);
                end;
```

```
if (check = 3) then
        begin
                BuildTransform(body, Trans);
                CoordConvert(body, body[5], EA[index], SMinAx[index],
                        nu[index], Eclipt, Peri, Trans, index);
                initx := Eclipt[index, 1];
                inity := Eclipt[index, 2];
                GMove(Eclipt[index, 1], Eclipt[index, 2]);
                final := nu[index] + 2 * pi;
                repeat
                        nu[index] := nu[index] + 5 * 0.017453292;
                        CoordConvert(body, body[5], EA[index],
                                SMinAx[index], nu[index], Eclipt,
                                Peri, Trans, index);
                        Gdraw(Eclipt[index, 1], Eclipt[index, 2]);
                until (nu[index] >= final);
                Eclipt[index, 1] := initx;
                Eclipt[index, 2] := inity;
                xlast[3] := Eclipt[index, 1];
                ylast[3] := Eclipt[index, 2];
                PlanetIcon3(Eclipt[index, 1], Eclipt[index, 2], rock3);
        end;

if (check = 4) then
        begin
                BuildTransform(body, Trans);
                CoordConvert(body, body[5], EA[index], SMinAx[index],
                        nu[index], Eclipt, Peri, Trans, index);
                initx := Eclipt[index, 1];
                inity := Eclipt[index, 2];
                GMove(Eclipt[index, 1], Eclipt[index, 2]);
                final := nu[index] + 2 * pi;
                repeat
                        nu[index] := nu[index] + 5 * 0.017453292;
                        CoordConvert(body, body[5], EA[index],
                                SMinAx[index], nu[index], Eclipt,
                                Peri, Trans, index);
                        Gdraw(Eclipt[index, 1], Eclipt[index, 2]);
                until (nu[index] >= final);
                Eclipt[index, 1] := initx;
                Eclipt[index, 2] := inity;
                xlast[4] := Eclipt[index, 1];
                ylast[4] := Eclipt[index, 2];
                PlanetIcon4(Eclipt[index, 1], Eclipt[index, 2], rock4);
        end;

if (check = 5) then
        begin
                BuildTransform(body, Trans);
                CoordConvert(body, body[5], EA[index], SMinAx[index],
                        nu[index], Eclipt, Peri, Trans, index);
```

```
                        initx := Eclipt[index, 1];
                        inity := Eclipt[index, 2];
                        GMove(Eclipt[index, 1], Eclipt[index, 2]);
                        final := nu[index] + 2 * pi;
                        repeat
                                nu[index] := nu[index] + 5 * 0.017453292;
                                CoordConvert(body, body[5], EA[index],
                                        SMinAx[index], nu[index], Eclipt,
                                        Peri, Trans, index);
                                Gdraw(Eclipt[index, 1], Eclipt[index, 2]);
                        until (nu[index] >= final);
                        Eclipt[index, 1] := initx;
                        Eclipt[index, 2] := inity;
                        xlast[5] := Eclipt[index, 1];
                        ylast[5] := Eclipt[index, 2];
                        PlanetIcon5(Eclipt[index, 1], Eclipt[index, 2], rock5);
                end;
        long[index] := ArcTan(Eclipt[index, 2] / Eclipt[index, 1]);
        if ((Eclipt[index, 1] < 0) and (Eclipt[index, 2] > 0) and (long[index] < 0))
                then
                        long[index] := long[index] + pi;
        if ((Eclipt[index, 1] < 0) and (Eclipt[index, 2] < 0) and (long[index] > 0))
                then
                        long[index] := long[index] + pi;
        if ((Eclipt[index, 1] > 0) and (Eclipt[index, 2] < 0) and (long[index] < 0))
                then
                        long[index] := long[index] + 2 * pi;
        BuildTransform(body, Trans);
        VPx[index] := (-body[5] * sin(E) * sqrt(GM) / (sqrt(body[5] * body[5] *
                        body[5]) * (1 - body[4] * cos(E)))) * AU;
        VPy[index] := (SMinAx[index] * cos(E) * sqrt(GM) / (sqrt(body[5] *
                        body[5] * body[5]) * (1 - body[4] * cos(E)))) * AU;
        EcliptVPx[index] := Trans[1, 1] * VPx[index] + Trans[1, 2] * VPy[index];
        EcliptVPy[index] := Trans[2, 1] * VPx[index] + Trans[2, 2] * VPy[index];
        EcliptVPz[index] := Trans[3, 1] * VPx[index] + Trans[3, 2] * VPy[index];
end;

{============================================================}
procedure Parameters (var H, c, s, Rop, m, L, e1, e2: extended; var CelesLat, long:
                storage; radius: storage);

        { This procedure defines several parameters depending upon}
        {whether the problem being solved is that for a body}
        {in heliocentric or geocentric orbit. The distance units}
        {are astronomical units or kilometers, respectively.}

    var
        x, y, z: storage;       {planetary position in the ecliptic}
        H_One,                  {preliminary HCA variable}
        k0, k1, k2, k3,         {computational parameters}
        k4, k5, k6, k7: extended;
    begin
```

```
        x[1] := cos(CelesLat[1]) * cos(long[1]);
        x[2] := cos(CelesLat[2]) * cos(long[2]);
        y[1] := cos(CelesLat[1]) * sin(long[1]);
        y[2] := cos(CelesLat[2]) * sin(long[2]);
        z[1] := sin(CelesLat[1]);
        z[2] := sin(CelesLat[2]);
        inclination := arctan((z[2] - z[1]) / (sqrt((x[2] - x[1]) * (x[2] - x[1])) +
                        ((y[2] - y[1]) * (y[2] - y[1]))));
        H_One := pi / 2 - ArcTan((x[1] * x[2] + y[1] * y[2] + z[1] * z[2]) / (sqrt(1 -
                        sqr(x[1] * x[2] + y[1] * y[2] + z[1] * z[2]))));
        if ((sin(long[2] - long[1])) >= 0) then
                H := H_One
        else
                H := 2 * pi - H_One;
        c := sqrt(radius[1] * radius[1] + radius[2] * radius[2] - 2 * radius[1] *
                radius[2] * cos(H));
        s := (radius[1] + radius[2] + c) / 2;
        Rop := (radius[1] + radius[2] + 2 * sqrt(radius[1] * radius[2]) * cos(H / 2))
                / 4;
        m := (GM * (dt) * (dt)) / (8 * Rop * Rop * Rop);
        k0 := radius[2] / radius[1];
        k1 := sqrt(k0);
        k2 := (k0 - 1);
        k3 := k2 * k2 / (4 * (k1 + k0 * (2 + k1)));
        k4 := (sin(H / 4)) * (sin(H / 4)) + k3;
        k5 := (cos(H / 4)) * (cos(H / 4)) + k3;
        k6 := cos(H / 2);
        k7 := 4 * k0 * (sin(H / 2) * sin(H / 2));
        e1 := k2 * k2;
        e2 := k7;
        if ((0 < H) and (H < pi)) then
                L := k4 / (k4 + k6)
        else
                L := (k5 - k6) / k5;
end;

{============================================================}
procedure XiFunction (x: extended; var Xi: extended);

        { This procedure uses the continued fraction scheme}
        {to approximate the function Xi.}

var
        eta, c, c2, A, A1, A2, B,B1, B2,
        d, Xilast: extended;    {calculation parameters}
        counter: integer;       {counts the number of repeat loop iterations}
        exit: boolean;          {determines when to exit the procedure}


begin
        eta := x / ((sqrt(1 + x) + 1) * (sqrt(1 + x) + 1));
        c2 := 8 * (sqrt(1 + x) + 1);
```

```
        A1 := 5 + eta;
        B1 := 1;
        A2 := 1;
        B2 := 0;
        counter := 0;
        exit := FALSE;
        Xi := 0;
        repeat
                counter := counter + 1;
                if (counter = 1) then
                        c := 9 * eta / 7
                else
                        c := eta * ((counter + 2) * (counter + 2)) / ((2 * counter + 5)
                                * (2 * counter + 3));
                d := 1;
                A := d * A1 + c * A2;
                B := d * B1 + c * B2;
                A1 := A1 / B;
                B1 := B1 / B;
                A := A / B;
                A2 := A1;
                B2 := B1;
                A1 := A;
                B1 := 1;
                Xilast := Xi;
                Xi := A;
                if (abs(Xi) = abs(Xilast)) then
                        begin
                                Xi := c2 / (3 + (1 / Xi));
                                exit := TRUE;
                        end;
        until ((counter = 100) or (exit = TRUE));
end;

{=============================================================}
procedure KFunction (u: extended; var K: extended);

        { This procedure uses a continued fraction scheme to}
        {approximate the function K.  This function is in}
        {turn used to solve the cubic equation of y.}

var
        exit: boolean;          {determines when to exit procedure}
        a, c, d,
        Klast: extended;        {calculation parameters}
        coefficient: extended;  {coefficient of u in the continued fraction}
        n: extended;            {the independant variable in the coeff eqn's}
        counter: integer;       {counts the number of iterations}
        index: extended;        {half-step counter that advances n}

begin
        a := 1 / 3;
```

```
        d := 1;
        c := a;
        K := c;
        index := 0;
        exit := FALSE;
        counter := -1;
        repeat
                counter := counter + 1;
                index := index + 0.5;
                n := trunc(index);
                if ((counter mod 2) = 0) then
                        coefficient := (2 * (3 * n + 2) * (6 * n + 1)) / (9 * (4 * n + 1)
                                * (4 * n + 3))
                else
                        coefficient := (2 * (3 * n + 1) * (6 * n - 1)) / (9 * (4 * n - 1)
                                * (4 * n + 1));
                a := coefficient * u;
                d := 1 / (1 + a * d);
                c := c * (d - 1);
                Klast := K;
                K := K + c;
                if ((abs(K) = abs(Klast)) or (n = 100)) then
                        exit := TRUE;
        until (exit = TRUE);
end;

{=============================================================}
procedure Cubic (Xi, L, m: extended; var y, K: extended);

        { This procedure calls the KFunction procedure}
        {in order to solve the cubic equation for y.}

var
        u, B, h1, h2: extended;         {calculation parameters}

begin
        h1 := ((L + x) * (L + x) * (1 + 3 * x + Xi)) / ((1 + 2 * x + L) * (4 * x + Xi
                * (3 + x)));
        h2 := (m * (x - L + Xi)) / ((1 + 2 * x + L) * (4 * x + Xi * (3 + x)));
        B := (27 * h2) / (4 * (1 + h1) * (1 + h1) * (1 + h1));
        if ((1 + B) < 0) then
                writeln('This mission crashes the program. Ignore the output
                        data.');
        if ((1 + B) > 0) then
                begin
                        u := B / (2 * (sqrt(1 + B) + 1));
                        KFunction(u, K);
                        y := ((1 + h1) / 3) * (2 + (sqrt(1 + B)) / (1 + 2 * u * K *
                                K));
                end;
end;
```

```
{=========================================================}
procedure TrajectoryComputations (var x, Xi, K, dt, H, c, s, Rop, m, L, e1, e2,
                                  inclination: extended; var radius, CelesLat, long:
                                  storage);


        {This procedure iterates the Battin procedures until x is calculated}

    var
        xlast,          {Holding variable for x}
        y: extended;    {Battin dependant variable}

    begin
        Parameters(H, c, s, Rop, m, L, e1, e2, CelesLat, long, radius);
        x := L;
        repeat
            XiFunction(x, Xi);
            Cubic(Xi, L, m, y, K);
            xlast := x;
            x := sqrt(((1 - L) / 2) * ((1 - l) / 2) + m / (y * y)) - (1 + L) / 2;
        until ((abs(x - xlast)) < 1.0E-15);
    end;

{=========================================================}
procedure ComputeOrbit (var L, a, ecc, eccentric1, eccentric2, p, angmom, n, gamma1,
                        gamma2, COne, CTwo, nu1, nu2: extended; Rop, e1, e2, c,
                        s, H, dt, inclination: extended; var radius, SMinAx,
                        CelesLat, long: storage; var spacecraft: planet; var Trans:
                        matrix; var EcliptVpx, EcliptVpy, EcliptVpz: storage);

        {This procedure calculates the orbital}
        {parameters of the spacecraft}

    var
        V1t,            {Tangential velocity at departure}
        V2t,            {Tangential velocity at arrival}
        V1r,            {Radial velocity at departure}
        V2r,            {Radial velocity at arrival}
        V1x,            {x- axis velocity at departure}
        V1y,            {y-axis velocity at departure}
        V2x,            {x-axis velocity at arrival}
        V2y,            {y-axis velocity at arrival}
        Vinfinity,      {Hyperbolic excess velocity}
        SmOmega,        {argument of the periapsis for the spacecraft}
        NuAlt,          {true anomaly reduced to a single quadrant}
        LongAlt,        {longitude}
        delLong,        {the difference between the longitudes of the launch
                        and target planets}
        delLat,         {the difference between the latitudes of the launch
                        and target planets}
        snapperhead,    {Hypotenuse of spherical triangle}
        Period: extended;   {The period of the orbit ('nuff said!)}
```

```
        descend: boolean;       {checks if orbit climbs or descends in CelesLat}
        i, j: integer;          {indexing variables}


begin
        descend := FALSE;
        j := 1;
        a := Rop * (1 + x) * (L + x) / (2 * x);                 {au}
        ecc := sqrt((e1 + e2 * ((L - x) / (L + x)) * ((L - x) / (L + x))) / (e1 + e2));
        p := a * (1 - ecc * ecc);                               {au}
        angmom := sqrt(GM * p) * AU * AU;                       {km2/s}
        n := sqrt((GM) / abs(a * a * a));                       {rad/s}
        VOne := sqrt(GM * (2 / radius[1] - 1 / a)) * AU;        {km/s}
        VTwo := sqrt(GM * (2 / radius[2] - 1 / a)) * AU;        {km/s}
        V1t := angmom / (radius[1] * AU);                       {km/s}
        V2t := angmom / (radius[2] * AU);                       {km/s}
        V1r := sqrt(VOne * VOne - V1t * V1t);                   {km/s}
        V2r := sqrt(VTwo * VTwo - V2t * V2t);                   {km/s}
        gamma1 := arctan(V1r / V1t);                            {rad}
        gamma2 := arctan(V2r / V2t);                            {rad}
        if (a > 0) then
                begin
                        Period := n / (2 * pi);                 {s}
                        Verify(a, dt, H, p, ecc, eccentric1, eccentric2, radius, nu1,
                                nu2);
                        if (nu1 > pi) then
                                gamma1 := -gamma1;
                        if (nu2 > pi) then
                                gamma2 := -gamma2;
                end;
        if (a < 0) then
                begin
                        writeln('This is a hyperbolic orbit');
                        Hyperverify(nu1, nu2, radius, ecc, H, p, a, dt);
                        if (nu1 < 0) then
                                gamma1 := -gamma1;
                        if (nu2 < 0) then
                                gamma2 := -gamma2;
                        Vinfinity := sqrt(-GM / a) * AU;        {km/s}
                        writeln('V infinity = ', Vinfinity : 12 : 4, ' km/s' : 5);
                end;
        spacecraft[4] := ecc;
        spacecraft[5] := a;
        delLong := long[2] - long[1];
        spacecraft[7] := ArcTan((((tan(Celeslat[1]) / tan(Celeslat[2])) * sin(long[2]))
                        - sin(long[1])) / (((tan(Celeslat[1]) / tan(Celeslat[2]))
                        * cos(long[2])) - cos(long[1])));
        if (Celeslat[2] = 0) then
                spacecraft[7] := ArcTan((((tan(Celeslat[2]) / tan(Celeslat[1])) *
                                sin(long[1])) - sin(long[2])) /
                                (((tan(Celeslat[2]) / tan(Celeslat[1])) *
                                cos(long[1])) - cos(long[2])));
```

```
        spacecraft[7] := spacecraft[7] * R;
        if (spacecraft[7] < 0) then
                spacecraft[7] := spacecraft[7] + 360;
        QuadrantCheck(spacecraft[7], delLong, CelesLat[1], CelesLat[2], long[1],
                        long[2]);
        if (spacecraft[7] > 360) then
                spacecraft[7] := spacecraft[7] - 360;
        if (Celeslat[j] = 0) then
                j := 2;
        spacecraft[6] := ArcTan((tan(Celeslat[j])) / (sin(long[j] - spacecraft[7] / R)));
        SMinAx[6] := spacecraft[5] * sqrt(1 - spacecraft[4] * spacecraft[4]);
        snapperhead := asin(sin(CelesLat[j]) / sin(spacecraft[6]));
        PieQuadrant(snapperhead, CelesLat, long, spacecraft[6], spacecraft[7]);
        if (snapperhead < 0) then
                snapperhead := snapperhead + 2 * pi;
        spacecraft[3] := snapperhead * R - nu2 * R + spacecraft[7];
        if CelesLat[2] = 0 then
                spacecraft[3] := snapperhead * R - nu1 * R + spacecraft[7];
        if (spacecraft[6] < 0) then
                spacecraft[6] := -spacecraft[6];
        spacecraft[6] := spacecraft[6] * R;
        V1x := (-a * sin(eccentric1) * sqrt(GM) / (sqrt(a * a * a) * (1 - ecc *
                cos(eccentric1)))) * AU;
        V1y := (SMinAx[6] * cos(eccentric1) * sqrt(GM) / (sqrt(a * a * a) * (1 - ecc
                * cos(eccentric1)))) * AU;
        V2x := (-a * sin(eccentric2) * sqrt(GM) / (sqrt(a * a * a) * (1 - ecc *
                cos(eccentric2)))) * AU;
        V2y := (SMinAx[6] * cos(eccentric2) * sqrt(GM) / (sqrt(a * a * a) * (1 - ecc
                * cos(eccentric2)))) * AU;
        BuildTransform(spacecraft, Trans);
        EcliptVel[1, 1] := Trans[1, 1] * V1x + Trans[1, 2] * V1y;
        EcliptVel[1, 2] := Trans[2, 1] * V1x + Trans[2, 2] * V1y;
        EcliptVel[1, 3] := Trans[3, 1] * V1x + Trans[3, 2] * V1y;
        EcliptVel[2, 1] := Trans[1, 1] * V2x + Trans[1, 2] * V2y;
        EcliptVel[2, 2] := Trans[2, 1] * V2x + Trans[2, 2] * V2y;
        EcliptVel[2, 3] := Trans[3, 1] * V2x + Trans[3, 2] * V2y;
        COne := (EcliptVel[1, 1] - EcliptVPx[1]) * (EcliptVel[1, 1] - EcliptVPx[1])
                + (EcliptVel[1, 2] - EcliptVPy[1]) * (EcliptVel[1, 2] - EcliptVPy[1])
                + (EcliptVel[1, 3] - EcliptVPz[1]) * (EcliptVel[1, 3] -
                EcliptVPz[1]);                                  {km^2/s^2}
        CTwo := (EcliptVel[2, 1] - EcliptVPx[2]) * (EcliptVel[2, 1] - EcliptVPx[2])
                + (EcliptVel[2, 2] - EcliptVPy[2]) * (EcliptVel[2, 2] - EcliptVPy[2])
                + (EcliptVel[2, 3] - EcliptVPz[2]) * (EcliptVel[2, 3] -
                EcliptVPz[2]);                                  {km^2/s^2}
end;

{=============================================================}
procedure GiveOutput (var place: name; var H, launchT, JlaunchT, arriveT, JarriveT,
                VOne, VTwo, COne, CTwo, ZOne, ZTwo, a, ecc, p, angmom, n,
                nu1, nu2, DeltaV1, DeltaV2: extended; radius, Long: storage;
                spacecraft: planet);
```

```
        { This procedure writes the output to the screen}
        {and to the external file, data.}

    begin
        writeln;
        writeln;
        writeln('Departure Planet:  ', place[1]);
        writeln('Launch Date:  ', CalDateL : 1 : 6);
        writeln('Julian Launch Date = ', JlaunchT : 1 : 0);
        writeln('R1 = ', radius[1] : 1 : 3, ' AU                    Long = ',
            Long[1] * R : 1 : 3, 'o');
        writeln('Celestial Latitude at time of launch = ', CelesLat[1] * R : 1 : 4);
        writeln('Launch True Anomaly = ', R * Nu1 : 1 : 6, 'o');
        writeln('V1 = ', VOne : 1 : 3, ' km/s' : 5, '  F1 = ', R * ZOne : 1 : 3, 'o');
        writeln('C3 = ', COne : 1 : 3, ' (km/sec)^2   DeltaV1 = ', DeltaV1: 1 : 4);
        writeln('Mlong: ', departure[2] - trunc(departure[2] / (360)) * 360 : 1 : 5);
        writeln;
        writeln('HELIOCENTRIC CONIC:');
        writeln('a = ', a : 1 : 3, '  e = ', ecc : 1 : 4);
        writeln('HCA = ', R * H : 1 : 2);
        writeln('p = ', p : 1 : 4, ' h = ', angmom : 1 : 4, ' n = ', n * 180 / pi * 24 *
            3600 : 1 : 4, 'o/day');
        writeln('sc omega = ', spacecraft[7] : 1 : 6);
        writeln('sc pi = ', spacecraft[3] : 1 : 6);
        writeln('sc SmOmega = ', spacecraft[3] - spacecraft[7] : 1 : 6);
        writeln('sc inclination = ', spacecraft[6] : 1 : 6);
        writeln;
        writeln('Destination:  ', place[2]);
        writeln('Time of Flight:  ', arriveT : 1 : 0, ' days');
        writeln('Arrival Date:  ', CalDateA : 1 : 6);
        writeln('Julian Arrival Date:  ', JarriveT : 1 : 0);
        writeln('R2 = ', radius[2] : 1 : 3, ' AU                    Long = ',
            Long[2] * R : 1 : 3, 'o');
        writeln('Celestial Latitude at time of arrival = ', CelesLat[2] * R : 1 : 4);
        writeln('Arrival True Anomaly = ', R * Nu2 : 1 : 6, 'o');
        writeln('V2 = ', VTwo : 1 : 3, ' km/s', '  F2 = ', R * ZTwo : 1 : 3, 'o');
        writeln('C3 = ', CTwo : 1 : 3, ' (km/sec)^2   DeltaV2 = ', DeltaV2 : 1 : 4);
    end;

{****************************** Main Program ******************************}

begin
    tilt := 23.443597 / R;
    Gwindow1(10, 10, 350, 350, 'x,y');
    Gscale(-3, 3, -3, 3);
    Gxaxis(0, 1, 0);
    Gyaxis(0, 1, 0);
    check := 0;
    CaseEntry(departure, target, launchT, JlaunchT, arriveT, JarriveT, dt, Jcentury1,
            Jcentury2, place, valid);
```

```
ComputePositions(departure, JlaunchT, 1, check, long, CelesLat, radius, phi,
                        VPlanet, E, nu, SMinAx, VPx, VPy, EcliptVPx, EcliptVPy,
                        EcliptVPz);
ComputePositions(target, JarriveT, 2, check, long, CelesLat, radius, phi, VPlanet,
                        E, nu, SMinAx, VPx, VPy, EcliptVPx, EcliptVPy,
                        EcliptVPz);
ComputePositions(flyby, JlaunchT, 3, check, long, CelesLat, radius, phi, Vplanet,
                        E, nu, SMinAx, VPx, VPy, EcliptVPx, EcliptVPy,
                        EcliptVPz);
ComputePositions(other, JlaunchT, 4, check, long, CelesLat, radius, phi, Vplanet,
                        E, nu, SMinAx, VPx, VPy, EcliptVPx, EcliptVPy,
                        EcliptVPz);
ComputePositions(other2, JlaunchT, 5, check, long, CelesLat, radius, phi,
                        Vplanet, E, nu, SMinAx, VPx, VPy, EcliptVPx, EcliptVPy,
                        EcliptVPz);
TrajectoryComputations(x, Xi, K, dt, H, c, s, Rop, m, L, e1, e2, inclination,
                                radius, CelesLat, long);
ComputeOrbit(L, a, ecc, eccentric1, eccentric2, p, angmom, n, gamma1, gamma2,
                COne, CTwo, nu1, nu2, Rop, e1, e2, c, s, H, dt, inclination,
                radius, SMinAx, CelesLat, long, spacecraft, Trans, EcliptVpx,
                EcliptVpy, EcliptVpz);
EarthParking(Rpark1, COne, DeltaV1);
ArrivalParking(CTwo, DeltaV2);
CalDateL := JlaunchT;
CalDateA := JarriveT;
InvConvert(CalDateL);
InvConvert(CalDateA);
GiveOutput(place, H, launchT, JlaunchT, arriveT, JarriveT, VOne, VTwo, COne,
                CTwo, gamma1, gamma2, a, ecc, p, angmom, n, nu1, nu2,
                DeltaV1, DeltaV2, radius, Long, spacecraft);
for i := 1 to 7 do
        target[i] := 0;
ChooseAsteroid(target, place[2], JlaunchT, valid);
ComputePositions(target, JlaunchT, 2, check, long, CelesLat, radius, phi, VPlanet,
                        E, nu, SMinAx, VPx, VPy, EcliptVPx, EcliptVPy,
                        EcliptVPz);
eccalt := eccentric1;
nualt := nu1;
for i := 1 to 7 do
        path[i] := spacecraft[i];
SMinAx[7] := SMinAx[6];
Days := JlaunchT;
repeat
        Days := Days + stepsize;
        GoGraphics(path[5], SMinAx[7], eccalt, nualt, 7, path, FALSE, Trans,
                        TRUE);
until (Days >= JarriveT);
Days := JlaunchT;
repeat
        Days := Days + stepsize;
        Jcentury := (Days - 2415021) / (36525);
```

```
            GoGraphics(spacecraft[5], SMinAx[6], eccentric1, nu1, 6, spacecraft,
                    TRUE, Trans, TRUE);
            earth(departure, Jcentury);
            GoGraphics(departure[5], SMinAx[1], E[1], nu[1], 1, departure, TRUE,
                    Trans, FALSE);
            ChooseAsteroid(target, place[2], Days, valid);
            GoGraphics(target[5], SMinAx[2], E[2], nu[2], 2, target, TRUE, Trans,
                    FALSE);
            Venus(flyby, Jcentury);
            GoGraphics(flyby[5], SMinAx[3], E[3], nu[3], 3, flyby, TRUE, Trans,
                    FALSE);
            Mercury(other, Jcentury);
            GoGraphics(other[5], SMinAx[4], E[4], nu[4], 4, other, TRUE, Trans,
                    FALSE);
            Mars(other2, Jcentury);
            GoGraphics(other2[5], SMinAx[5], E[5], nu[5], 5, other2, TRUE, Trans,
                    FALSE);
        until (Days >= JarriveT);
        SaveDrawing('orbit');
        Gpause;
end.
```

Low Thrust Program

```
program Solar_Sail (input, output, Aster3dOut, AMom3dOut);
{_____}
{**********************************************************************}
{Program: 3-D Solar Sail Program}

{By:  Brian K. Funk, A. Allen McCarley, H. Dan Thomas}

{Date:  12/03/92}

{Last Revised:  2/03/93}

{Description:  This program uses Pontryagin's Maximization Principle to calculate the
               minimum time Solar Sail 3-D interplanetary trajectory.}
{**********************************************************************}
{----------------------------------------------------------------------------------------}

type
      vec = array[1..20] of double;
      mat = array[1..20,1..20] of double;

const {Astronautical constants taken from U.S.A.F. Academy text}

      AU = 1.4959965E+8;              {km/au}
      GM = 3.96396415708E-14;         {au3/s2}
      GMearth = 398600;               {km3/s2}
      RAD = 57.29577951;              {degrees/radian}
      dt = 0.005;
      pi = 3.1415926535897;
      inf = 1e+15;

var
      JlaunchT,     {The launch date on the Julian calendar}
      Epoch,        {The date corresponding to the asteroid ephemeris data}
      ErosEpoch,    {The date corresponding to Eros's ephemeris data}
      Tfinal,       {The date of arrival at the target planet}
      tstar,        {Non-dimensionalizing factor}
      JarriveT,     {Tfinal in the Julian calendar}
      JEpoch,       {The Epoch date on the Julian calendar}
      Jcentury1,    {Launch date in Julian Centuries}
      Jcentury2,    {Arrival date in Julian Centuries}
      Vr,           {Radial velocity}
      Vphi,         {Tangential velocity}
      Vpsi,         {Normal velocity}
      r,            {Radius}
      Phi,          {Celestial Latitude in 2-D spherical}
      Psi,          {Celestial Longitude in 2-D spherical}
      PVr,          {Lagrange multiplier conjugate to Vr}
      PVphi,        {Lagrange multiplier conjugate to Vphi}
      PVpsi,        {Lagrange multiplier conjugate to Vpsi}
      Pr,           {Lagrange multiplier conjugate to radius}
```

```
Pphi,          {Lagrange multiplier conjugate to Phi}
Ppsi,          {Lagrange multiplier conjugate to Psi}
rearth,        {Initial radius of earth}
rfinal,        {Final radius}
Phiearth,      {Initial Phi of earth}
Phifinal,      {Final Phi of solar sail}
Psiearth,      {Initial Psi of earth}
Psifinal,      {Final Psi}
Vrearth,       {Initial Radial Velocity of earth}
Vrfinal,       {Final Radial Velocity}
Vphiearth,     {Initial Tangental Velocity of earth}
Vphifinal,     {Final Tangental Velocity}
Vpsiearth,     {Initial Normal Velocity of earth}
Vpsifinal,     {Final Normal Velocity}
Theta,         {The sail thrust cone angle}
Beta,          {The sail thrust clock angle}
Hamil,         {The Hamiltonian}
t,             {Time}
inc,           {Earth's tilt}
Parki,         {Inclination of Cape Canaveral parking orbit}
rPark,         {Parking orbit radius}
VcPark,        {Parking orbit velocity}
Vpara,         {Parabolic velocity}
DeltaV,        {Velocity increment required for parabolic escape}
Caldate,       {Launch date in calendar form}
ao: double;    {Characteristic acceleration}

f,             {External file reference variable}
elk: text;     {External file reference variable}
departure,     {Stores the orbital elements of the departure planet}
target,        {Stores the orbital elements of the target planet}
Phiplanet,     {Celestial Latitude of the planets}
Psiplanet,     {Celestial Longitude of the planets}
Rplanet,       {Radius of the Planets}
Vrplanet,      {Radial Velocity of the Planets}
Vphiplanet,    {Tangential Velocity of the Planets}
Vpsiplanet,    {Normal Velocity of the Planets}
x,             {Initial guess conditions}
y: vec;        {Final conditions to be met}

k1: integer;

doit:  boolean;


{**************************************************************************}
                              FUNCTIONS
{**************************************************************************}

function tan (argument: double): double;
```

```
                { This function returns the tangent of the input angle }
begin
        tan := sin(argument) / cos(argument);
end;

{============================================================}

function asin (argument: double): double;

                {This function returns the arcsine value of the input angle in radians}

begin
        asin := ArcTan((argument) / (sqrt(1 - argument * argument)));
end;

{============================================================}

function acos (argument: double): double;

                {This function returns the arcosine value of the input angle in radians}

begin
        acos := pi / 2 - ArcTan((argument) / (sqrt(1 - argument * argument)));
end;

{============================================================}

function CalcBeta (arg10, arg11: double): double;

                {This function calculates the optimal sail clock angle}

var
        Betac,
        Betas:  double;

begin
        Betac := arg11 / (sqrt(sqr(arg10) + sqr(arg11)));
        Betas := arg10 / (sqrt(sqr(arg10) + sqr(arg11)));
        CalcBeta := acos(Betac);
        if (Betas < 0) then
                CalcBeta := 2*pi - acos(Betac);

end;

{============================================================}

function CalcTheta (arg9, arg10, arg11, Beta: double): double;

                {This function calculates the optimal sail cone angle}

var
        B1:     double;
```

```
begin
        B1 := arg10*sin(Beta)+arg11*cos(Beta);
        CalcTheta := ArcTan((sqrt(9*arg9*arg9 + 8*sqr(B1)) - 3*arg9)/(4*B1));
end;

{=================================================================}

                        Differential Equations

{=================================================================}

function Dr (arg4: double): double;

begin
        Dr := arg4;
end;

{=================================================================}

function Dphi (arg1, arg3, arg5: double): double;

begin
        Dphi := arg5/(arg1*cos(arg3));
end;

{=================================================================}

function Dpsi (arg1, arg6: double): double;

begin
        Dpsi := arg6/arg1;
end;

{=================================================================}

function DVr (arg1, arg5, arg6, ao, Theta: double): double;

begin
        DVr := (sqr(arg5)+sqr(arg6))/arg1 +
                (ao*cos(Theta)*cos(Theta)*cos(Theta) - 1)/(arg1*arg1);
end;

{=================================================================}

function DVphi (arg1, arg3, arg4, arg5, arg6, ao, Theta, Beta: double): double;

begin
        DVphi := -(arg4*arg5/arg1) + (arg5*arg6*tan(arg3)/arg1) +
                (ao*sin(Theta)*cos(Theta)*cos(Theta)*sin(Beta))/(arg1*arg1);
end;
```

```
{================================================================}

function DVpsi (arg1, arg3, arg4, arg5, arg6, ao, Theta, Beta: double): double;

begin
        DVpsi := -(arg4*arg6/arg1) - (arg5*arg5*tan(arg3)/arg1) +
                (ao*sin(Theta)*cos(Theta)*cos(Theta)*cos(Beta))/(arg1*arg1);
end;

{================================================================}

function DPvr (arg1, arg5, arg6, arg7, arg10, arg11: double): double;

begin
        DPvr := arg10*arg5/arg1 + arg11*arg6/arg1 - arg7;
end;

{================================================================}

function DPvphi (arg1, arg3, arg4, arg5, arg6, arg9, arg10, arg11: double): double;

begin
        DPvphi := -(2*arg9*arg5/arg1) + arg4*arg10/arg1 - arg10*arg6*tan(arg3)/arg1 +
                        2*arg11*arg5*tan(arg3)/arg1;
end;

{================================================================}

function DPvpsi (arg1, arg3, arg4, arg5, arg6, arg8, arg9, arg10, arg11: double): double;

begin
        DPvpsi := -(2*arg9*arg6/arg1) + arg4*arg11/arg1 - arg10*arg5*tan(arg3)/arg1 -
                        arg8/arg1;
end;

{================================================================}

function DPr (arg1, arg3, arg4, arg5, arg6, arg8, arg9, arg10, arg11, ao,
                Theta, Beta: double): double;

var
        C1, C2, C3, C4, C5, C6, C7, C8, C9, C10:  double;

begin
        C1 := arg8*arg6/(sqr(arg1));
        C2 := arg9*(sqr(arg5)+sqr(arg6))/(arg1*arg1);
        C3 := 2*arg9/(arg1*arg1*arg1);
        C4 := 2*ao*arg9*cos(Theta)*cos(Theta)*cos(Theta)/(arg1*arg1*arg1);
        C5 := arg10*arg4*arg5/(arg1*arg1);
        C6 := arg10*arg5*arg6*tan(arg3)/(arg1*arg1);
        C7 := 2*arg10*ao*cos(Theta)*cos(Theta)*sin(Theta)*
```

```
            sin(Beta)/(arg1*arg1*arg1);
        C8 := arg11*arg4*arg6/(sqr(arg1));
        C9 := arg11*arg5*arg5*tan(arg3)/(sqr(arg1));
        C10 := 2*arg11*ao*cos(Theta)*cos(Theta)*sin(Theta)*
            cos(Beta)/(arg1*arg1*arg1);
        DPr := C1 + C2 - C3 + C4 - C5 + C6 + C7 - C8 - C9 + C10;
end;

{=================================================================}

function DPpsi (arg1, arg3, arg5, arg6, arg10, arg11:double): double;

begin
        DPpsi := -(arg10*arg5*arg6/(arg1*cos(arg3)*cos(arg3))) +
                    arg11*sqr(arg5)/(arg1*cos(arg3)*cos(arg3));
end;

{*****************************************************************}

                                PROCEDURES

{*****************************************************************}

procedure Mercury (var choice: vec; time: double);

                { This procedure determines the orbital      }
                { elements for the planet Mercury at         }
                { the date chosen.                           }

begin
        choice[1] := (14732.4197380 - 0.000001355 * time);
        choice[2] := (178 + 10 / 60 + 44.68 / 3600) + (538106654.80 / 3600) * time +
                    (1.084 / 3600) * time * time;
        choice[3] := 75 + 53 / 60 + 58.91 / 3600 + (5599.76 / 3600) * time +
                    (1.061 / 3600) * time * time;
        choice[4] := 0.20561421 + 0.00002046 * time - 0.000000030 * time * time;
        choice[5] := 0.38709860;
        choice[6] := 7 + 10.37 / 3600 + (6.699 / 3600) * time -
                    (0.066 / 3600) * time * time;
        choice[7] := 47 + 8 / 60 + 45.4 / 3600 + (4266.75 / 3600) * time +
                    (0.626 / 3600) * time * time;
end;

{=================================================================}

procedure Venus (var choice: vec; time: double);

                { This procedure determines the orbital      }
                { elements for the planet Venus at the       }
                { date chosen.                               }

begin
```

```
        choice[1] := (5767.6697692 + 0.0000002628 * time);
        choice[2] := 342 + 46 / 60 + 1.39 / 3600 + (210669162.88 / 3600) * time +
                     (1.1148/3600) * time * time;
        choice[3] := 130 + 9 / 60 + 49.8 / 3600 + (5068.93 / 3600) * time -
                     (3.515 / 3600) * time * time;
        choice[4] := 0.00682069 - 0.00004774 * time + 0.000000091 * time * time;
        choice[5] := 0.7323162;
        choice[6] := 3 + 23 / 60 + 37.07 / 3600 + (3.621 / 3600) * time -
                     (0.0035 / 3600) * time * time;
        choice[7] := 75 + 46 / 60 + 46.73 / 3600 + (3239.46 / 3600) * time +
                     (1.476 / 3600) * time * time;
end;

{=============================================================}

procedure earth (var choice: vec; time: double);

                { This procedure determines the orbital      }
                { elements for the planet earth at the       }
                { date chosen.                               }

begin
        choice[1] := (3548.1928323 - 0.000001103 * time);
        choice[2] := 99 + 41 / 60 + 48.04 / 3600 + (129602768.13 / 3600) * time +
                     (1.089 / 3600) * time * time;
        choice[3] := 101 + 13 / 60 + 15 / 3600 + (6189.03 / 3600) * time +
                     (1.63 / 3600) * time * time + (0.012 / 3600) * time * time * time;
        choice[4] := 0.01675104 - 0.00004180 * time - 0.000000126 * time * time;
        choice[5] := 1.00000023;
        choice[6] := 0.0;
        choice[7] := 0.0;
end;

{=============================================================}

procedure Mars (var choice: vec; time: double);

                { This procedure determines the orbital      }
                { elements for the planet Mars at the        }
                { date chosen.                               }

begin
        choice[1] := (1886.5186207 + 0.000000463 * time);
        choice[2] := 293 + 44 / 60 + 51.46 / 3600 + (68910103.83 / 3600) * time +
                     1.1184 * time * time / 3600;
        choice[3] := 334 + 13 / 60 + 5.53 / 3600 + (6626.73 / 3600) * time +
                     (0.4675 / 3600) * time * time -
                     (0.0043 / 3600) * time * time * time;
        choice[4] := 0.09331290 + (0.000092064) * time - (0.000000077) * time * time;
        choice[5] := 1.52368840;
        choice[6] := 1 + 51 / 60 + 1.2 / 3600 - (2.43 / 3600) * time +
                     (0.0454 / 3600) * time * time;
```

```
        choice[7] := 48 + 47 / 60 + 11.19 / 3600 + (2775.57 / 3600) * time -
                        (0.005 / 3600) * time * time -
                        (0.0192 / 3600) * time * time * time;
end;

{=============================================================}

procedure XB (var choice: vec);

                { This procedure determines the orbital      }
                { elements for the asteroid 1982 XB at the   }
                { date chosen.                               }

begin
        choice[1] := 16.77969;
        choice[2] := 56.89879/RAD;
        choice[3] := 91.30613;
        choice[4] := 0.4466558;
        choice[5] := 1.8355863;
        choice[6] := 3.87446;
        choice[7] := 74.52644;
end;

{=============================================================}

procedure Orpheus (var choice: vec);

                { This procedure determines the orbital      }
                { elements for the asteroid Orpheus at the   }
                { date chosen.                               }

begin
        choice[1] := 301.56931;
        choice[2] := 197.58723/RAD;
        choice[3] := 130.70829;
        choice[4] := 0.3226346;
        choice[5] := 1.2093243;
        choice[6] := 2.68775;
        choice[7] := 189.13898;
end;

{=============================================================}

procedure McAuliffe (var choice: vec);

                { This procedure determines the orbital      }
                { elements for the asteroid McAuliffe at the }
                { date chosen.                               }

begin
        choice[1] := 15.59117;
        choice[2] := 283.00658/RAD;
```

```
        choice[3] := 122.49259;
        choice[4] := 0.3694649;
        choice[5] := 1.8787307;
        choice[6] := 4.77743;
        choice[7] := 106.90142;
end;

{=============================================================}
procedure Seleucus (var choice: vec);

                { This procedure determines the orbital      }
                { elements for the asteroid Seleucus at the  }
                { date chosen.                               }

begin
        choice[1] := 349.18871;
        choice[2] := 343.98574/RAD;
        choice[3] := 207.31474;
        choice[4] := 0.4571202;
        choice[5] := 2.0325801;
        choice[6] := 5.93181;
        choice[7] := 218.12603;
end;

{=============================================================}

procedure Eros (var choice: vec);

                { This procedure determines the orbital      }
                { elements for the asteroid Seleucus at the  }
                { date chosen.                               }

begin
        choice[1] := 178.510;
        choice[2] := 170.451/RAD;
        choice[3] := 123.006;
        choice[4] := 0.2229;
        choice[5] := 1.4582;
        choice[6] := 10.832;
        choice[7] := 304.496;
end;

{=============================================================}
procedure Convert (var date: double);

                { This procedure converts any date entered in        }
                { the standard calender into a Julian date.          }
                { Taken from Astronomical Formulae for Calculators   }
                { Meeus, p. 24.                                      }

var
        Month,          {The portion of the enterred date that represents the month}
```

```
        Day,            {The portion of the enterred date that represents the day}
        Year: double;   {The protion of the enterred date that represents the year}
        a,b: double;

begin
        Month := trunc(date);
        Day := trunc(100 * (date - Month));
        Year := Round((1000000 * date) - (Month * 1000000) - (Day * 10000));
        if (Month <= 2) then
                begin
                        Year := Year - 1;
                        Month := Month + 12;
                end;
        a := Year/100;
        b := 2 - a + a/4;
        date := trunc(365.25 * Year) + trunc(30.60001*(Month + 1.0)) + Day +
                1720994.5 + b;
end;

{================================================================}
procedure InvConvert (var date: double);

                { This procedure converts any Julian date     }
                { into a calendar date.                       }
                { Taken from Astronomical Formulae for        }
                { Calculators, Meeus, p. 24.                  }

var
        Month,          {The portion of the enterred date that represents the month}
        Day,            {The portion of the enterred date that represents the day}
        Year: double;   {The protion of the enterred date that represents the year}
        a,b,c,d,e,
        f: double;

begin
        f := date + 0.5;
        a := trunc(f);
        f := f - a;
        if (a >= 2299161) then
                begin
                        b := (a - 1867216.25)/36524.25;
                        a := a + (1 + b - b/4);
                end;
        b := a + 1524;
        c := trunc((b - 122.1)/365.25);
        d := trunc(365.25*c);
        e := trunc((b - d)/30.6001);
        Day := b - d - trunc(30.6001*e) + f;
        if (e <= 13) then
                Month := e - 1
        else
                Month := e - 13;
```

```
        if (Month <= 2) then
                Year := c - 4715
        else
                Year := c - 4716;
        date := trunc(Month) + 0.01*round(Day) + 0.000001*trunc(Year);

end;

{================================================================}
procedure CaseEntry (var JlaunchT, tstar, ao: double; var x:vec);

                { This procedure prompts the user for the initial guess      }
                { launch date and flight time. It also prompts the user      }
                { for the sail's characteristic acceleration.                }

begin

        writeln('Enter the initial guess launch date as mm.ddyyyy or as a Julian date.');
        readln(JlaunchT);
        if ((JlaunchT / 1000) < 1) then
                Convert(JlaunchT);
        x[1] := JlaunchT;
        writeln('Enter the initial guess for time of flight in days.');
        readln(x[2]);
        x[2] := x[2]/tstar;
        writeln('Enter the characteristic acceleration in (mm/s^2).');
        readln(ao);
        ao := ((0.000001 * ao)/AU)/GM;
end;

{================================================================}

procedure BuildTrans1 (var SmOmega: double; var body: vec; var Transec: mat);

                {This procedure builds the matrix which serves       }
                {as the transformation matrix between the            }
                {ecliptic and pericentric coordinate system          }

begin
        Transec[1, 1] := cos(body[7] / RAD) * cos(SmOmega / RAD) -
                         sin(body[7] / RAD)*sin(SmOmega / RAD) * cos(body[6] / RAD);
        Transec[1, 2] := -cos(body[7] / RAD) * sin(SmOmega / RAD) -
                         sin(body[7] / RAD)*cos(SmOmega / RAD) * cos(body[6] / RAD);
        Transec[1, 3] := sin(body[7] / RAD) * sin(body[6] / RAD);
        Transec[2, 1] := sin(body[7] / RAD) * cos(SmOmega / RAD) +
                         cos(body[7] / RAD)*sin(SmOmega / RAD) * cos(body[6] / RAD);
        Transec[2, 2] := -sin(body[7] / RAD) * sin(SmOmega / RAD) +
                         cos(body[7] / RAD)*cos(SmOmega / RAD) * cos(body[6] / RAD);
        Transec[2, 3] := -cos(body[7] / RAD) * sin(body[6] / RAD);
        Transec[3, 1] := sin(SmOmega / RAD) * sin(body[6] / RAD);
        Transec[3, 2] := cos(SmOmega / RAD) * sin(body[6] / RAD);
        Transec[3, 3] := cos(body[6] / RAD);
```

```
end;

{================================================================}

procedure BuildTrans2 (var Phiplanet, Psiplanet: vec; index:integer; var Transsp: mat);

                {This procedure builds the matrix which serves      }
                {as the transformation matrix between the           }
                {spherical and ecliptic coordinate system           }

begin
        Transsp[1, 1] := cos(Phiplanet[index])*cos(Psiplanet[index]);
        Transsp[1, 2] := sin(Phiplanet[index])*cos(Psiplanet[index]);
        Transsp[1, 3] := sin(Psiplanet[index]);
        Transsp[2, 1] := -sin(Phiplanet[index]);
        Transsp[2, 2] := cos(Phiplanet[index]);
        Transsp[2, 3] := 0;
        Transsp[3, 1] := -sin(Psiplanet[index])*cos(Phiplanet[index]);
        Transsp[3, 2] := -sin(Psiplanet[index])*sin(Phiplanet[index]);
        Transsp[3, 3] := cos(Psiplanet[index]);
end;

{================================================================}

procedure GetEaEarth (var body: vec; var Ea: double);

                { This procedure finds the eccentric anomaly of earth   }
                { at the launch date.                                   }
var
        Ma,
        Mo,                             {Initial mean anomaly}
        D:      double;                 {Newton's method delta}


begin
        Ma := pi / 180 * (body[2] - body[3]);
        if (Ma < 0) then
                Ma := Ma + (2 * 3.141592654);
        Ea := Ma + body[4] * (sin(Ma) + body[4] * sin(2 * Ma) / 2);
        repeat
                Mo := Ea - body[4] * sin(Ea);
                D := (Ma - Mo) / (1 - body[4] * cos(Ea));
                Ea := Ea + D;
        until (abs(D) <= 1E-10);
end;

{================================================================}

procedure GetEaAster (var body: vec; var Ea, JEpoch, JarriveT: double);

                { This procedure finds the eccentric anomaly of the     }
                { target asteroid at the arrival date.                  }
```

```
var
        wo,                             {Mean angular motion}
        Mo,                             {Initial mean anomaly}
        Tepoch,                         {Time after periapsis passage at the Epoch}
        Tdelta,                         {Time difference between Arrival and Epoch}
        Ttotal,                         {Time after periapsis passage at the Arrival Time}
        D:      double;                 {Newton's method delta}

begin
        wo := sqrt(GM/(body[5]*body[5]*body[5]));
        Tepoch := body[2]/wo;
        Tdelta := (JarriveT - JEpoch)*(24)*(3600);
        Ttotal := Tepoch + Tdelta;
        Mo := wo * Ttotal;
        Ea := Mo;
        repeat
                D := (-Ea + body[4]*sin(Ea) + Mo) / (1 - body[4] * cos(Ea));
                Ea := Ea + D;
        until (abs(D) <= 1E-10);
end;

{================================================================}

procedure ComputePositions (var body, Rplanet, Phiplanet, Vrplanet, Vphiplanet: vec;
                            var JEpoch, JarriveT: double; index: integer);

                { This procedure calculates the earth's position and velocity at      }
                { launch, and the target asteroid's position and velocity at arrival  }

var
        Ea,                             {Eccentric anomaly}
        SminAx,                         {Semi-minor axis}
        SmOmega,                        {Longitude of periapsis}
        Perix,                          {Pericentric x position}
        Periy,                          {Pericentric y position}
        Vpx,                            {Pericentric x velocity component}
        Vpy,                            {Pericentric y velocity component}
        Vearthx,                        {Earth centered x velocity component}
        Vearthy,                        {Earth centered y velocity component}
        Vearthz,                        {Earth centered z velocity component}
        B,                              {Angle between the North pole and the}
                                        {hyperbolic excess velocity vector}
        Omeg1,                          {Longitude of the parking orbit's ascending node}
        Omeg2,                          {Longitude of the parking orbit's ascending node}
        relinc,                         {Required inclination change}
        Vel:            double;         {Velocity}

        nu,                             {true anomaly}
        Ecliptx,                        {Ecliptical x position}
        Eclipty,                        {Ecliptical y position}
        Ecliptz,                        {Ecliptical z position}
```

```
        EcliptVx,                   {Ecliptical x velocity component}
        EcliptVy,                   {Ecliptical y velocity component}
        EcliptVz:    vec;           {Ecliptical z velocity component}

        Transec,                    {Pericentric to ecliptical coordinate conversion}
                                    {matrix}
        Transsp:     mat;           {Ecliptical to spherical coordinate conversion}
                                    {matrix}
begin
        if (index = 1) then
                GetEaEarth(body, Ea);
        if (index = 2) then
                GetEaAster (body, Ea, JEpoch, JarriveT);
        nu[index] := 2 * ArcTan(sqrt((1+body[4]) / (1-body[4])) *
                        ((sin(Ea/2) / cos(Ea/2))));
        if (nu[index] < 0) then
                nu[index] := nu[index] + 2 * pi;
        SminAx := body[5]*sqrt(1 - body[4]*body[4]);
        SmOmega := body[3] - body[7];
        if (index = 2) then
                SmOmega := body[1];
        Rplanet[index] := body[5] * (1-body[4]*body[4]) / (1+body[4]*cos(nu[index]));
        Perix := Rplanet[index] * cos(nu[index]);
        Periy := Rplanet[index] * sin(nu[index]);


        VPx := (-body[5] * sin(Ea) / (sqrt(body[5] * body[5] * body[5]) *
                (1 - body[4] * cos(Ea))));
        VPy := (SminAx * cos(Ea)/ (sqrt(body[5] * body[5] * body[5]) *
                (1 - body[4] * cos(Ea))));

        Vel := sqrt(2/Rplanet[index]-1/body[5]);

        BuildTrans1 (SmOmega,body,Transec);

        Ecliptx[index] := Transec[1, 1] * Perix + Transec[1, 2] * Periy;
        Eclipty[index] := Transec[2, 1] * Perix + Transec[2, 2] * Periy;
        Ecliptz[index] := Transec[3, 1] * Perix + Transec[3, 2] * Periy;

        EcliptVx[index] := Transec[1, 1] * VPx + Transec[1, 2] * VPy;
        EcliptVy[index] := Transec[2, 1] * VPx + Transec[2, 2] * VPy;
        EcliptVz[index] := Transec[3, 1] * VPx + Transec[3, 2] * VPy;

        Phiplanet[index] := ArcTan(Eclipty[index] / Ecliptx[index]);
        if ((Ecliptx[index] < 0) and (Eclipty[index] > 0) and (Phiplanet[index] < 0)) then
                Phiplanet[index] := Phiplanet[index] + pi;
        if ((Ecliptx[index] < 0) and (Eclipty[index] < 0) and (Phiplanet[index] > 0)) then
                Phiplanet[index] := Phiplanet[index] + pi;
        if ((Ecliptx[index] > 0) and (Eclipty[index] < 0) and (Phiplanet[index] < 0)) then
                Phiplanet[index] := Phiplanet[index] + 2 * pi;

        Psiplanet[index] := asin(sin(SmOmega/RAD+nu[index]) * sin(body[6]/RAD));
```

```
        BuildTrans2 (Phiplanet, Psiplanet, index, Transsp);

        Vrplanet[index] := Transsp[1, 1]*EcliptVx[index] +
                           Transsp[1, 2]*EcliptVy[index] +
                           Transsp[1, 3]*EcliptVz[index];
        Vphiplanet[index] := Transsp[2, 1]*EcliptVx[index] +
                           Transsp[2, 2]*EcliptVy[index] +
                           Transsp[2, 3]*EcliptVz[index];
        Vpsiplanet[index] := Transsp[3, 1]*EcliptVx[index] +
                           Transsp[3, 2]*EcliptVy[index] +
                           Transsp[3, 3]*EcliptVz[index];

        if (index = 1) and (doit = TRUE) then
                begin
                        Vearthx := -EcliptVx[index]*sin(Phiplanet[index]) +
                                (EcliptVy[index]*cos(inc) - EcliptVz[index]*sin(inc))*
                                cos(Phiplanet[index]);
                        Vearthy := -EcliptVx[index]*cos(Phiplanet[index]) +
                                (- EcliptVy[index]*cos(inc) + EcliptVz[index]*sin(inc))*
                                sin(Phiplanet[index]);
                        Vearthz := EcliptVy[index]*sin(inc) + EcliptVz[index]*cos(inc);
                        B := acos(Vearthz/Vel);
                        if ((B + Parki)<pi/2) then
                                begin
                                        writeln('Nontangential injection.');
                                        relinc := asin((cos(B+Parki))/(cos(89.9999/RAD)));
                                        writeln('relinc: ',relinc*RAD:1:6);
                                end;
                        if ((B + Parki)>=pi/2) then
                                begin
                                        writeln('Tangential injection.');
                                        writeln('No plane change required.');
                                        Omeg1 := pi + asin(1/(tan(B)*tan(Parki)));
                                        Omeg2 := 2*pi - asin(1/(tan(B)*tan(Parki)));
                                        writeln;
                                        writeln('Omega 1: ',Omeg1*RAD:1:6);
                                        writeln('Omega 2: ',Omeg2*RAD:1:6);
                                        write('(Omega is measured relative to the earth''s');
                                        writeln(' velocity vector.)');
                                end;
                        Vearthx := Vearthx*sqrt(GM)*AU;
                        Vearthy := Vearthy*sqrt(GM)*AU;
                        Vearthz := Vearthz*sqrt(GM)*AU;
                        EcliptVx[index] := EcliptVx[index]*sqrt(GM)*AU;
                        EcliptVy[index] := EcliptVy[index]*sqrt(GM)*AU;
                        EcliptVz[index] := EcliptVz[index]*sqrt(GM)*AU;

                end;

end;
```

```
{================================================================}

procedure InitialConditions (var JlaunchT,Jcentury1,Jcentury2,JarriveT,t,ao,Vr,Vphi,
                             r,phi,PVr,PVphi,Pr,Tfinal,Phiearth,Phifinal: double;
                             var x,departure,target,Phiplanet: vec);

        {This procedure sets the initial conditions for the problem}

var i:integer;  {indexing variable}

begin
    JlaunchT := x[1];
    Jcentury1 := (JlaunchT - 2415021) / (36525);
    for i := 1 to 7 do
        departure[i] := 0;
    earth(departure, Jcentury1);
    JarriveT := x[1] + x[2] * tstar;
    Jcentury2 := (JarriveT - 2415021) / (36525);
    for i := 1 to 7 do
        target[i] := 0;

{********** Target Asteroid **********}

    {XB(target);}
    {Orpheus(target);}
    {McAuliffe(target);}
    {Seleucus(target);}
    Eros(target);

    ComputePositions (departure,Rplanet,Phiplanet,Vrplanet,Vphiplanet,JEpoch,
                      JarriveT,1);
    ComputePositions (target,Rplanet,Phiplanet,Vrplanet,Vphiplanet,JEpoch,
                      JarriveT,2);
    rearth := Rplanet[1];
    rfinal := Rplanet[2];
    Phiearth := Phiplanet[1];
    Phifinal := Phiplanet[2];
    if (Phiearth >= 2 * pi) then
        Phiearth := Phiearth - 2 * pi;
    if (Phifinal >= 2 * pi) then
        Phifinal := Phifinal - 2 * pi;
    Psiearth := Psiplanet[1];
    Psifinal := Psiplanet[2];
    Vrearth := Vrplanet[1];
    Vrfinal := Vrplanet[2];
    Vphiearth := Vphiplanet[1];
    Vphifinal := Vphiplanet[2];
    Vpsiearth := Vpsiplanet[1];
    Vpsifinal := Vpsiplanet[2];

    t := 0.0;
    Vr := Vrearth;
```

```
        Vphi := Vphiearth;
        Vpsi := Vpsiearth;
        r := rearth;
        phi := Phiearth;
        Psi := Psiearth;
        Tfinal := x[2];
        Pr := x[3];
        PVr := x[4];
        PVphi := x[5];
        Ppsi := x[6];
        PVpsi := x[7];
        Pphi := 0.0;
end;

{================================================================}

procedure RungeKutta (var r, Phi, Psi, Vr, Vphi, Vpsi, PVr, PVphi, PVpsi, Pr, Ppsi,
                      Theta, Beta, ao, Hamil: double);

        { This procedure uses the forth order Runge-Kutta procedure      }
        { to integrate the equations of motion                           }

var
        K1, K2, K3, K4, L1, L2, L3, L4, M1, M2, M3, M4, N1, N2, N3, N4: double;
        O1, O2, O3, O4, P1, P2, P3, P4, Q1, Q2, Q3, Q4, R1, R2, R3, R4: double;
        S1, S2, S3, S4, U1, U2, U3, U4, W1, W2, W3, W4: double;

begin
        Beta := CalcBeta (PVphi, PVpsi);
        Theta := CalcTheta (PVr, PVphi, PVpsi, Beta);
        K1 := dt*Dr(Vr);
        L1 := dt*Dphi(r, Psi, Vphi);
        M1 := dt*Dpsi(r, Vpsi);
        N1 := dt*DVr(r, Vphi, Vpsi, ao, Theta);
        O1 := dt*DVphi(r, Psi, Vr, Vphi, Vpsi, ao, Theta, Beta);
        P1 := dt*DVpsi(r, Psi, Vr, Vphi, Vpsi, ao, Theta, Beta);
        Q1 := dt*DPvr(r, Vphi, Vpsi, Pr, PVphi, PVpsi);
        R1 := dt*DPvphi(r, Psi, Vr, Vphi, Vpsi, PVr, PVphi, PVpsi);
        S1 := dt*DPvpsi(r, Psi, Vr, Vphi, Vpsi, Ppsi, PVr, PVphi, PVpsi);
        U1 := dt*DPr(r, Psi, Vr, Vphi, Vpsi, Ppsi, PVr, PVphi, PVpsi, ao, Theta, Beta);
        W1 := dt*DPpsi(r, Psi, Vphi, Vpsi, PVphi, PVpsi);

        Beta := CalcBeta (PVphi+R1/2.0, PVpsi+S1/2.0);
        Theta := CalcTheta(PVr + Q1/2.0, PVphi + R1/2.0, PVpsi+S1/2.0, Beta);
        K2 := dt*Dr(Vr+N1/2.0);
        L2 := dt*Dphi(r+K1/2.0, Psi+M1/2.0, Vphi+O1/2.0);
        M2 := dt*Dpsi(r+K1/2.0, Vpsi+P1/2.0);
        N2 := dt*DVr(r+K1/2.0, Vphi+O1/2.0, Vpsi+P1/2.0, ao, Theta);
        O2 := dt*DVphi(r+K1/2.0, Psi+M1/2.0, Vr+N1/2.0, Vphi+O1/2.0, Vpsi+P1/2.0,
                       ao, Theta, Beta);
        P2 := dt*DVpsi(r+K1/2.0, Psi+M1/2.0, Vr+N1/2.0, Vphi+O1/2.0, Vpsi+P1/2.0,
                       ao, Theta, Beta);
```

```
Q2 := dt*DPvr(r+K1/2.0, Vphi+O1/2.0, Vpsi+P1/2.0, Pr+U1/2.0, PVphi+R1/2.0,
                PVpsi+S1/2.0);
R2 := dt*DPvphi(r+K1/2.0, Psi+M1/2.0, Vr+N1/2.0, Vphi+O1/2.0, Vpsi+P1/2.0,
                PVr+Q1/2.0, PVphi+R1/2.0, PVpsi+S1/2.0);
S2 := dt*DPvpsi(r+K1/2.0, Psi+M1/2.0, Vr+N1/2.0, Vphi+O1/2.0, Vpsi+P1/2.0,
                Ppsi+W1/2.0, PVr+Q1/2.0, PVphi+R1/2.0, PVpsi+S1/2.0);
U2 := dt*Dpr(r+K1/2.0, Psi+M1/2.0, Vr+N1/2.0, Vphi+O1/2.0, Vpsi+P1/2.0,
                Ppsi+W1/2.0, PVr+Q1/2.0, PVphi+R1/2.0, PVpsi+S1/2.0,
                ao, Theta, Beta);
W2 := dt*DPpsi(r+K1/2.0, Psi+M1/2.0, Vphi+O1/2.0, Vpsi+P1/2.0,
                PVphi+R1/2.0,PVpsi+S1/2.0);

Beta := CalcBeta (PVphi+R2/2.0, PVpsi+S2/2.0);
Theta := CalcTheta(PVr + Q2/2.0, PVphi + R2/2.0, PVpsi+S2/2.0, Beta);
K3 := dt*Dr(Vr+N2/2.0);
L3 := dt*Dphi(r+K2/2.0, Psi+M2/2.0, Vphi+O2/2.0);
M3 := dt*Dpsi(r+K2/2.0, Vpsi+P2/2.0);
N3 := dt*DVr(r+K2/2.0, Vphi+O2/2.0, Vpsi+P2/2.0, ao, Theta);
O3 := dt*DVphi(r+K2/2.0, Psi+M2/2.0, Vr+N2/2.0, Vphi+O2/2.0, Vpsi+P2/2.0,
                ao, Theta, Beta);
P3 := dt*DVpsi(r+K2/2.0, Psi+M2/2.0, Vr+N2/2.0, Vphi+O2/2.0, Vpsi+P2/2.0,
                ao, Theta, Beta);
Q3 := dt*DPvr(r+K2/2.0, Vphi+O2/2.0, Vpsi+P2/2.0, Pr+U2/2.0, PVphi+R2/2.0,
                PVpsi+S2/2.0);
R3 := dt*DPvphi(r+K2/2.0, Psi+M2/2.0, Vr+N2/2.0, Vphi+O2/2.0, Vpsi+P2/2.0,
                PVr+Q2/2.0, PVphi+R2/2.0, PVpsi+S2/2.0);
S3 := dt*DPvpsi(r+K2/2.0, Psi+M2/2.0, Vr+N2/2.0, Vphi+O2/2.0, Vpsi+P2/2.0,
                Ppsi+W2/2.0, PVr+Q2/2.0, PVphi+R2/2.0, PVpsi+S2/2.0);
U3 := dt*Dpr(r+K2/2.0, Psi+M2/2.0, Vr+N2/2.0, Vphi+O2/2.0, Vpsi+P2/2.0,
                Ppsi+W2/2.0, PVr+Q2/2.0, PVphi+R2/2.0, PVpsi+S2/2.0,
                ao, Theta, Beta);
W3 := dt*DPpsi(r+K2/2.0, Psi+M2/2.0, Vphi+O2/2.0, Vpsi+P2/2.0,
                PVphi+R2/2.0,PVpsi+S2/2.0);

Beta := CalcBeta (PVphi+R3, PVpsi+S3);
Theta := CalcTheta(PVr + Q3, PVphi + R3, PVpsi+S3, Beta);
K4 := dt*Dr(Vr+N3);
L4 := dt*Dphi(r+K3, Psi+M3, Vphi+O3);
M4 := dt*Dpsi(r+K3, Vpsi+P3);
N4 := dt*DVr(r+K3, Vphi+O3, Vpsi+P3, ao, Theta);
O4 := dt*DVphi(r+K3, Psi+M3, Vr+N3, Vphi+O3, Vpsi+P3, ao, Theta, Beta);
P4 := dt*DVpsi(r+K3, Psi+M3, Vr+N3, Vphi+O3, Vpsi+P3, ao, Theta, Beta);
Q4 := dt*DPvr(r+K3, Vphi+O3, Vpsi+P3, Pr+U3, PVphi+R3, PVpsi+S3);
R4 := dt*DPvphi(r+K3, Psi+M3, Vr+N3, Vphi+O3, Vpsi+P3,PVr+Q3,
                PVphi+R3, PVpsi+S3);
S4 := dt*DPvpsi(r+K3, Psi+M3, Vr+N3, Vphi+O3, Vpsi+P3,Ppsi+W3, PVr+Q3,
                PVphi+R3,PVpsi+S3);
U4 := dt*Dpr(r+K3, Psi+M3, Vr+N3, Vphi+O3, Vpsi+P3, Ppsi+W3, PVr+Q3,
                PVphi+R3,PVpsi+S3, ao, Theta, Beta);
W4 := dt*DPpsi(r+K3, Psi+M3, Vphi+O3, Vpsi+P3, PVphi+R3, PVpsi+S3);

{********************** Final Runge Kutta Equations ********************}
```

```
        r := r + (1.0/6.0)*(K1 + 2.0*K2 + 2.0*K3 + K4);
        Phi := Phi + (1.0/6.0)*(L1 + 2.0*L2 + 2.0*L3 + L4);
        if (Phi >= 2 * pi) then
                Phi := Phi - 2 * pi;
        Psi := Psi + (1.0/6.0)*(M1 + 2.0*M2 + 2.0*M3 + M4);
        Vr := Vr + (1.0/6.0)*(N1 + 2.0*N2 + 2.0*N3 + N4);
        Vphi := Vphi + (1.0/6.0)*(O1 + 2.0*O2 + 2.0*O3 + O4);
        Vpsi := Vpsi + (1.0/6.0)*(P1 + 2.0*P2 + 2.0*P3 + P4);
        PVr := PVr + (1.0/6.0)*(Q1 + 2.0*Q2 + 2.0*Q3 + Q4);
        PVphi := PVphi + (1.0/6.0)*(R1 + 2.0*R2 + 2.0*R3 + R4);
        PVpsi := PVpsi + (1.0/6.0)*(S1 + 2.0*S2 + 2.0*S3 + S4);
        Pr := Pr + (1.0/6.0)*(U1 + 2.0*U2 + 2.0*U3 + U4);
        Ppsi := Ppsi + (1.0/6.0)*(W1 + 2.0*W2 + 2.0*W3 + W4);
        Beta := CalcBeta (PVphi, PVpsi);
        Theta := CalcTheta(PVr, PVphi, PVpsi, Beta);

        Hamil := Pr*Vr + Ppsi*Vpsi/r;
        Hamil := Hamil + PVr*((sqr(Vphi)+sqr(Vpsi))/r - 1/(r*r)) +
                        PVr*(ao*cos(Theta)*cos(Theta)*cos(Theta)/(r*r));
        Hamil := Hamil + PVphi*(-Vr*Vphi/r + Vphi*Vpsi*tan(Psi)/r) +
                        PVphi*(ao*sin(Theta)*cos(Theta)*cos(Theta)*sin(Beta)/(r*r));
        Hamil := Hamil + PVpsi*(-Vr*Vpsi/r - Vphi*Vphi*tan(Psi)/r) +
                        PVpsi*(ao*sin(Theta)*cos(Theta)*cos(Theta)*cos(Beta)/(r*r));
end;

{=============================================================}

procedure Solution (var r, Phi, Psi, Vr, Vphi, Vpsi, PVr, PVphi, PVpsi, Pr, Ppsi,
                        Theta, Beta, ao: double);

                { This procedure calls the initial conditions and runs the      }
                { Runge-Kutta procedure                                         }
var
        time1: double;

begin
        InitialConditions(JlaunchT,Jcentury1,Jcentury2,JarriveT,t,ao,Vr,Vphi,r,phi,PVr,
                                PVphi,Pr,Tfinal,Phiearth,Phifinal,
                                x,departure,target,Phiplanet);
        time1 := 0.0;
        while (t <= Tfinal) do
                begin
                        RungeKutta(r,Phi,Psi,Vr,Vphi,Vpsi,PVr,PVphi,PVpsi,Pr,Ppsi,
                                        Theta,Beta,ao,Hamil);
                        t := t + dt;
                        time1 := t * tstar;
                        if (time1 > 1000) then
                                return;
                end;
end;
```

```
{============================================================}

procedure Vecset (var x,y: vec);

                { This procedure calculates the difference between the current    }
                { and desired values at the endpoint of the trajectory            }

begin
        Solution (r,Phi,Psi,Vr,Vphi,Vpsi,PVr,PVphi,PVpsi,Pr,Ppsi,Theta,Beta,ao);
        y[1] := r - rfinal;
        y[2] := Phi - Phifinal;
        y[3] := Psi - Psifinal;
        y[4] := Vr - Vrfinal;
        y[5] := Vphi - Vphifinal;
        y[6] := Vpsi - Vpsifinal;
        y[7] := Hamil - 1;
end;

{============================================================}

procedure GaussianElim (var A: mat; var b,x: vec; var n: integer);

                {This procedure is a Gaussian Elimination routine used in  }
                { the two point boundary value problem solvers             }

var
        SltnCheck:  boolean;


procedure AugMatrix (var A: mat; var b: vec);

var
        i:  integer;

begin (* AugMatrix *)
        for i := 1 to n do
                A[i,n+1] := b[i];
end; (* AugMatrix *)


procedure PartialPiv (var A: mat; var n, k: integer);

var
        row, i, j: integer;
        max, temp: double;

begin (* PartialPiv *)
        max := Abs(A[k,k]);
        row := k;

        for i := k+1 to n do
                begin (* i *)
```

```
                        if (Abs(A[i,k]) > max) then
                                begin (* if *)
                                        row := i;
                                        max := Abs(A[i,k]);
                                end (* if *)
                end; (* i *)

        if (row <> k) then
                begin (* if *)
                        for j := k to n+1 do
                                begin (* j *)
                                        temp := A[k,j];
                                        A[k,j] := A[row,j];
                                        A[row,j] := temp
                                end (* j *)
                end (* if *)

end; (* PartialPiv *)


procedure BackSubst (var A: mat; var n: integer; var x: vec);

var
        i,j: integer;
        sum: double;

begin (* BackSubst *)
        x[n] := A[n,n+1]/A[n,n];
        for i := n-1 downto 1 do
                begin (* i *)
                        sum := 0.0;
                        for j := i+1 to n do
                                sum := sum + A[i,j]*x[j];
                        x[i] := (A[i,n+1] - sum)/A[i,i]
                end (* i *)
end; (* BackSubst *)


procedure Gauss (var A: mat; var n: integer; var SltnCheck: boolean);

var
        i, j, k:  integer;
        m, tol:  double;

begin (* Gauss *)
        tol := 1.0e-16;
        for k := 1 to n-1 do
                begin (* k *)
                        if (SltnCheck = TRUE) then
                                begin (* if SltnCheck *)
                                        k1 := k;
                                        PartialPiv (A, n, k1);
```

```
                                        if (Abs(A[k,k]) > tol) then
                                                begin (* if *)
                                                        for i := k+1 to n do
                                                        begin (* i *)
                                                        m := A[i,k]/A[k,k];
                                                        for j := k+1 to n+1 do
                                                                A[i,j] := A[i,j] - m*A[k,j]
                                                        end; (* i *)
                                                        for i := k+1 to n do
                                                                A[i,k] := 0.0
                                                end (* if *)
                                        end (* if SltnCheck *)
                                else
                                        SltnCheck := FALSE
                        end; (* k *)
                if (Abs(A[n,n]) < tol) then
                        SltnCheck := FALSE;
end; (* Gauss *)


begin (* GaussianElim *)
        SltnCheck := TRUE;
        AugMatrix(A,b);
        Gauss(A,n,SltnCheck);
        if (SltnCheck = TRUE) then
                BackSubst(A, n, x);
end; (* GaussianElim *)

{================================================================}

procedure Steep(var x, ybase: vec; ndim: integer);

                { This procedure utilizes the Steepest Descent method to        }
                { improve the initial guesses for the launch date, time of flight,  }
                { and the Lagrange undetermined multipliers such that Newton's  }
                { method will converge to a solution                            }

var
        y, xbase, z:  vec;
        jacob, jacobT:  mat;
        tol1:  double;
        g0, g1, g2, g3, h1, h2, h3:  double;
        a0, a1, a2, a3, z0, gmin, amin, delx:  double;
        i, j, k, w, q:  integer;

begin

        tol1 := 1e-15;
        for w := 1 to 15 do
                begin
                        writeln;
                        writeln('Steepest descent:');
```

```
writeln;
writeln('w = ',w:2);
writeln;
Vecset(x, ybase);
for i := 1 to ndim do
        xbase[i] := x[i];

{ Calculate Jacobian }

writeln;
for j := 1 to ndim do
        begin
                delx:=x[j]/100;
                if (j=1) then delx := 5;
                x[j] := x[j] + delx;
                write('Calculating Jacobian:');
                writeln(' varying initial condition ',j:1);
                Vecset(x,y);
                for i := 1 to ndim do
                        jacob[i,j] := (y[i] - ybase[i])/delx;
                x[j] := x[j] - delx;
        end;
writeln;

g1 := 0.0;
for i := 1 to ndim do
        g1 := g1 + ybase[i] * ybase[i];

{ Find Transpose of Jacobian }

for i := 1 to ndim do
        begin
                for j := 1 to ndim do
                        jacobT[i,j] := jacob[j,i];
        end;

{ Obtain 2*jacobT*ybase (gradient of g) }

for j := 1 to ndim do
        begin
                z[j] := 0.0;
                for k := 1 to ndim do
                        z[j] := z[j] + 2 * jacobT[j,k] *ybase[k];
        end;

z0 := 0.0;
for i := 1 to ndim do
        z0 := z0 + z[i] * z[i];
if (z0 < tol1 / 2) then
        begin
                writeln('exit Steep');
                readln;
```

```
                return;
        end;
for i := 1 to ndim do
        z[i] := z[i] / z0;
a1 := 0.0;
a3 := 1.0;
for i := 1 to ndim do
        begin
        x[i] := xbase[i] - a3 * z[i];
        end;
Vecset(x,y);
g3 := 0.0;
for i := 1 to ndim do
        begin
        g3 := g3 + y[i] *y[i];

        end;

while ((abs(g3) - abs(g1)) > tol1 / 10) do
        begin
                a3 := a3 / 2;
                for i := 1 to ndim do
                        x[i] := xbase[i] - a3 * z[i];
                Vecset(x,y);
                g3 := 0.0;
                for i := 1 to ndim do
                        g3 := g3 + y[i] *y[i];
                writeln('Snapperhead');
                for i := 1 to ndim do
                        writeln('x[',i:1,'] = ', x[i]:1:15);
                writeln('g1 = ',g1:1:15);
                writeln('g3 = ',g3:1:15);
                writeln('g3 - g1 = ',abs(g3) - abs(g1));
                writeln;
                if (a3 < 1e-15) then
                        return;
        end;
a2 := a3 / 2;
for i:= 1 to ndim do
        x[i] := xbase[i] - a2 * z[i];
Vecset(x,y);
g2 := 0.0;
for i := 1 to ndim do
        g2 := g2 + y[i] * y[i];
h1 := (g2 - g1) / a2;
h2 := (g3 - g2) / (a3 - a2);
h3 := (h2 - h1) / a3;
writeln('g1 = ',g1:1:15);
writeln('g2 = ',g2:1:15);
writeln('g3 = ',g3:1:15);
a0 := 0.5 * (a2 - h1/h3);
for i := 1 to ndim do
```

```
                        x[i] := xbase[i] - a0 * z[i];
                Vecset(x,y);
                g0 := 0.0;
                for i := 1 to ndim do
                        g0 := g0 + y[i]*y[i];
                gmin := g0;
                amin := a0;
                if (g1 < gmin) then
                        begin
                                gmin := g1;
                                amin := a1;
                        end;
                if (g2 < gmin) then
                        begin
                                gmin := g2;
                                amin := a2;
                        end;
                if (g3 < gmin) then
                        begin
                                gmin := g3;
                                amin := a3;
                        end;
                for i := 1 to ndim do
                        x[i] := xbase[i] - amin*z[i];
                writeln('gmin = ',gmin);
                for i := 1 to ndim do
                        writeln('xbest[',i:1,'] = ',x[i]:1:15);
                for i := 1 to 3 do
                        writeln;
                if (gmin < tol1) then
                        begin
                                writeln('exit Steep');
                                readln;
                                return;
                        end;
        end;

    writeln('Steepest descent did not work');
    readln;
end;

{=============================================================}

procedure Vecroot(var x, ybase: vec; ndim: integer);

        { This procedure utilizes Newton's method to modify the          }
        { guesses for the launch date, time of flight, and Lagrange      }
        { undetermined multipliers such that the interplanetary          }
        { trajectory meets the target body's final position and velocity. }

var
    y,
```

```
        x1,
        xbest: vec;             {The value of x that gives the least error}
        jacob: mat;             {Jacobian matrix}
        errlast,                {Error on the previous iteration}
        err,                    {Error on the current iteration}
        delx:  double;          {The change in x for partial differentials}
        i, j, k, w,             {Counter and indexing variables}
        bad: integer;           {The successive increases in error}

begin
        writeln;
        writeln('ybase at entry:');
        for i:= 1 to ndim do
                writeln('ybase[',i:1,'] = ':4,ybase[i]);
        writeln;
        errlast := inf;
        for k := 1 to ndim do
                xbest[k] := x[k];
        for w := 1 to 25 do
                begin
                        writeln('Vecroot');
                        writeln;
                        write('Phifinal = ',Phifinal:1:4);
                        writeln(' or ',Phifinal*180/pi:1:4,' deg.');
                        writeln;
                        for i := 1 to ndim do
                                writeln('x[', i:1, '] = ', x[i]:1:15);
                        Vecset(x, ybase);
                        writeln;
                        writeln('ybase after first Vecset; w = ',w:1);
                        for i:= 1 to ndim do
                                writeln('ybase[',i:1,'] = ':4,ybase[i]);
                        writeln;
                        err := 0;
                        for i := 1 to ndim do
                                err := err + sqr(ybase[i]);
                        err := sqrt(err);

                        writeln;
                        writeln('Vecroot Error: ', err:1:19);
                        writeln;
                        if (errlast > err) then
                                begin
                                        for j := 1 to ndim do
                                                xbest[j] := x[j];
                                end;
                        if (err >= 10.0) then
                                begin
                                        for j := 1 to ndim do
                                                x[j] := xbest[j];
                                        writeln('Error too large: Aborted.');
                                        return;
```

```
                end;

            if (w > 5) and (err >= errlast) then
                begin
                bad:=bad+1;
                if (bad=2) then
                    begin
                        for j := 1 to ndim do
                            x[j] := xbest[j];
                        for i := 1 to ndim do
                            begin
                                write('xbest[', i:1, '] = ');
                                writeln(x[i]:1:15);
                            end;
                        readln;
                        return;
                    end;
                end
            else bad:=0;

            if (errlast > err) then
                errlast := err;

            writeln;
            for j := 1 to ndim do
                begin
                    delx:=x[j]/100;
                    if (j=1) then delx:=1;
                    x[j] := x[j] + delx;
                    write('Calculating Jacobian:  ');
                    writeln('varying initial conditions = ',j:1);
                    Vecset(x, y);
                    for i := 1 to ndim do
                        jacob[i,j] := (y[i] - ybase[i])/delx;
                    x[j] := x[j] - delx;

                end;
            writeln;

            for i:= 1 to ndim do ybase[i]:=-ybase[i];
            GaussianElim(jacob, ybase, x1, ndim);
            for i := 1 to ndim do
                x[i] := x[i] + x1[i];
        end;
    writeln('Solution not found in "Vecroot"');
end;

{*************************************************************************}
```

Main Program

```
{*************************************************************************}

begin
        rewrite(f,'Aster3dOut');
        rewrite(elk,'AMom3dOut');

                { The x-values are the initial guesses used by the two point      }
                { boundary problem solver procedures                              }

{x[1] = Launch Date}
{x[2] = Time of Flight}
{x[3] = Pr}
{x[4] = PVr}
{x[5] = PVphi}
{x[6] = Ppsi}
{x[7] = PVpsi}

{=================== Initial Condition Solutions ======================}

{******************** For 2 mm/s^2 acceleration ********************}

{======================= Earth to Eros ========================}

{-------------------------------------------------------------}

{Vecroot Error: 0.0001725047340392750
Tfinal = 6.98770 = 406.21462
JarriveT: 2451305.922404}

{xbest[1] = 2450899.707781539700000
xbest[2] = 6.987698280041766
xbest[3] = 2.278961645327300
xbest[4] = 2.159274427781875
xbest[5] = 2.852886154793199
xbest[6] = -2.116801372463481
xbest[7] = -2.605091641891992}

{-------------------------------------------------------------}

{Vecroot Error: 0.0000000000006838246
Tfinal = 6.28761 = 365.51623
JarriveT: 2452047.477645}

{xbest[1] = 2451681.961418402800000
xbest[2] = 6.287605037317054
xbest[3] = 2.418927405149455
xbest[4] = 2.777735035449584
xbest[5] = 2.475577147228649
xbest[6] = -2.231662565480368
```

xbest[7] = 0.718223413337819}

{----------}

{Vecroot Error: 0.0000000000007250947
Tfinal = 4.92201 = 286.13019
JarriveT: 2452766.237633}

{xbest[1] = 2452480.107440463700000
xbest[2] = 4.922007583418802
xbest[3] = 1.358808317197987
xbest[4] = 1.721597214585108
xbest[5] = 0.714675262079025
xbest[6] = 1.351887813729902
xbest[7] = 5.317699374919390}

{----------}

{Vecroot Error: 0.0001801027294678451
Tfinal = 6.84820 = 398.10521
JarriveT: 2453841.057578}

{xbest[1] = 2453442.952372001000000
xbest[2] = 6.848200220366335
xbest[3] = 2.194156353966996
xbest[4] = 1.939352589120784
xbest[5] = 2.813604608698123
xbest[6] = -1.860112723102528
xbest[7] = -3.194060935652314}

{----------}

{Vecroot Error: 0.0000000000011815839
Tfinal = 6.49128 = 377.35640
JarriveT: 2454607.168759}

{xbest[1] = 2454229.812363040100000
xbest[2] = 6.491279474887119
xbest[3] = 2.434391476117421
xbest[4] = 2.742327203458784
xbest[5] = 2.610151093307849
xbest[6] = -2.363445498290722
xbest[7] = 0.024063351036179}

{----------}

{Vecroot Error: 0.0000000000007697013
Tfinal = 5.20565 = 302.61923
JarriveT: 2455320.992483}

{xbest[1] = 2455018.373248561700000

xbest[2] = 5.205651857041241
xbest[3] = 1.676255879393927
xbest[4] = 2.069150484351515
xbest[5] = 1.205563472500338
xbest[6] = 0.062564213694901
xbest[7] = 4.525472684903297}

{----------}

{Vecroot Error: 0.0000641388477953920
Tfinal = 6.36728 = 370.14823
JarriveT: 2456350.746018}

{xbest[1] = 2455980.597787703800000
xbest[2] = 6.367284710229947
xbest[3] = 2.043338781085809
xbest[4] = 1.653849692791382
xbest[5] = 2.698644405540325
xbest[6] = -1.465700608252043
xbest[7] = -3.877750063533898}

{----------}

{********** For 1 mm/s^2 acceleration **********}

{========== Earth to Eros ==========}

{----------}

{Vecroot Error: 0.0001874307356185348
Tfinal = 8.93651 = 519.50461}

{xbest[1] = 2450845.844906746400000
xbest[2] = 8.936511156513333
xbest[3] = 3.897025970244485
xbest[4] = -0.337273150375379
xbest[5] = 8.601649665374314
xbest[6] = -2.372716922435655
xbest[7] = -13.602780230315307}

{----------}

{Vecroot Error: 0.0000000000003196663
Tfinal = 8.09820 = 470.77123}

{xbest[1] = 2451632.053823401200000
xbest[2] = 8.098200143872099
xbest[3] = 3.699188092588651
xbest[4] = 2.248625059954152
xbest[5] = 7.679548349641885
xbest[6] = -19.244559682147457
xbest[7] = -10.296504191812763}

{----------}

{Vecroot Error: 0.0003118975264085687
Tfinal = 6.83125 = 397.11969}

{xbest[1] = 2452427.111974170800000
xbest[2] = 6.831247436873251
xbest[3] = 1.863768268542354
xbest[4] = 2.836110914043703
xbest[5] = 1.982596694947442
xbest[6] = -13.652115349787424
xbest[7] = 11.802911911001345}

{----------}

{Vecroot Error: 0.0005631537405202619
Tfinal = 8.90836 = 517.86800}

{xbest[1] = 2453393.557834652700000
xbest[2] = 8.908358063753981
xbest[3] = 4.604554220185918
xbest[4] = -0.199921867457571
xbest[5] = 9.276148861541371
xbest[6] = 0.579332764991332
xbest[7] = -12.614898719053798}

{----------}

{Vecroot Error: 0.0000000000003096127
Tfinal = 8.33351 = 484.45048}

{xbest[1] = 2454176.524940265300000
xbest[2] = 8.333510350993803
xbest[3] = 3.056290623307565
xbest[4] = 1.184295842435545
xbest[5] = 7.302668990924715
xbest[6] = -16.018439839239551
xbest[7] = -12.464945383471043}

{----------}

{Vecroot Error: 0.0000000000006739922
Tfinal = 7.01397 = 407.74192}

{xbest[1] = 2454970.481742481700000
xbest[2] = 7.013970849541767
xbest[3] = 4.011608293854890
xbest[4] = 4.225268789650134
xbest[5] = 5.049210990753889
xbest[6] = -18.159206148693524
xbest[7] = 7.974579378307928}

```
{-------------------------------------------------------}

{Vecroot Error: 0.0005292705391496099
Tfinal = 8.74096 = 508.13654}

{xbest[1] = 2455939.135251565800000
xbest[2] = 8.740957765669005
xbest[3] = 5.324524861445238
xbest[4] = -0.077065640126640
xbest[5] = 9.939163604346628
xbest[6] = 3.842202709947721
xbest[7] = -11.491162575010282}

{-------------------------------------------------------}

{Vecroot Error: 0.0000000000011611136
Tfinal = 7.23824 = 420.77913}

{xbest[1] = 2457515.507579755900000
xbest[2] = 7.238236705207489
xbest[3] = 5.162516308093598
xbest[4] = 4.904960622216813
xbest[5] = 7.065955713019433
xbest[6] = -21.555095825831980
xbest[7] = 3.459708149619061}

{-------------------------------------------------------}

{*************************************************************************}

doit := FALSE;
        inc := 23.443597/RAD;
        Parki := 28.5/RAD;
        rPark := 6378.145 + 500;

        Vcpark := sqrt(GMearth/rPark);
        Vpara := sqrt(2*GMearth/rPark);
        DeltaV := (Vpara-Vcpark);

        Epoch := 11.051990;
        ErosEpoch := 6.191986;

        {JEpoch := Epoch;}
        JEpoch := ErosEpoch;

        Convert(JEpoch);
        tstar := sqrt(1.0/GM)/86400.0;

{ao := ((0.000001)/AU)/GM;}
ao := ((0.000002)/AU)/GM;
        {CaseEntry (JlaunchT, tstar, ao, x);}
```

```
        {Steep(x, y, 7);}
        {Vecroot(x, y, 7);}

        writeln;

doit := TRUE;
        InitialConditions(JlaunchT,Jcentury1,Jcentury2,JarriveT,t,ao,Vr,Vphi,r,phi,PVr,
                          PVphi,Pr,Tfinal,Phiearth,Phifinal,
                          x,departure,target,Phiplanet);
        Beta := CalcBeta (PVphi, PVpsi);
        Theta := CalcTheta(PVr, PVphi, PVpsi, Beta);
        Caldate := JlaunchT;
        InvConvert (Caldate);

        Hamil := Pr*Vr + Ppsi*Vpsi/r;
        Hamil := Hamil + PVr*((sqr(Vphi)+sqr(Vpsi))/r - 1/(r*r)) +
                         PVr*(ao*cos(Theta)*cos(Theta)*cos(Theta)/(r*r));
        Hamil := Hamil + PVphi*(-Vr*Vphi/r + Vphi*Vpsi*tan(Psi)/r) +
                         PVphi*(ao*sin(Theta)*cos(Theta)*cos(Theta)*sin(Beta)/(r*r));
        Hamil := Hamil + PVpsi*(-Vr*Vpsi/r - Vphi*Vphi*tan(Psi)/r) +
                         PVpsi*(ao*sin(Theta)*cos(Theta)*cos(Theta)*cos(Beta)/(r*r));


        writeln('Launch Date: ',Caldate:1:6);
        writeln('Julian Launch Date: ',trunc(JlaunchT));
        writeln;
        writeln('Required Delta V: ',DeltaV:1:5,' km/s');
        writeln;
        writeln('Tfinal = ',Tfinal:1:5, ' = ',Tfinal*tstar:1:5);

        writeln(f,'Time', 'radius':10, 'Phi':12, 'Psi':12,'Vr':10, 'Vphi':10, 'Vpsi':10);
        write(elk,'Theta', 'Beta':10, 'Pr':10, 'Ppsi':10,'PVr':11, 'PVphi':11);
        writeln(elk,'PVpsi':10,'H':7);

        writeln('Theta:', Theta*RAD:1:2, '  Beta: ', Beta*RAD:1:2, '  H:',Hamil:1:5);
        writeln;
        writeln('JarriveT: ',trunc(JarriveT));
        readln;

        write('Time:', (t*tstar):1:4, '  r:', r:1:6, '  Phi:', Phi*RAD:1:2);
        write('  Psi:', Psi*RAD:1:2,'  Vr:', Vr:1:4, '  Vphi:', Vphi:1:4);
        writeln('  Vpsi:', Vpsi:1:4, '  H:',Hamil:1:5);
        write(f, (t*tstar):3:2,' ', r:10:6,' ', Phi*RAD:12:6,' ', Psi*RAD:12:6,' ', Vr:10:6);
        writeln(f,' ', Vphi:10:6,' ', Vpsi:10:6);
        write(elk, Theta*RAD:2:6,' ', Beta*RAD:10:6,' ', Pr:10:6,' ', Ppsi:10:6);
        writeln(elk,' ', PVr:10:6,' ', PVphi:10:6,' ', PVpsi:10:6,' ', Hamil:8:2);

        t:=0.0;

        while (t <= Tfinal) do
                begin
```

```
                RungeKutta(r,Phi,Psi,Vr,Vphi,Vpsi,PVr,PVphi,PVpsi,Pr,Ppsi,
                                Theta,Beta,ao,Hamil);
                write('Time:', (t*tstar):1:4, ' r:', r:1:6, ' Phi:', Phi*RAD:1:2);
                write(' Psi:', Psi*RAD:1:2,' Vr:', Vr:1:4, ' Vphi:', Vphi:1:4);
                writeln(' Vpsi:', Vpsi:1:4, ' H:',Hamil:1:5);
                write(f, (t*tstar):3:2,' ', r:10:6,' ', Phi*RAD:12:6,' ', Psi*RAD:12:6);
                writeln(f,' ', Vr:10:6,' ', Vphi:10:6,' ', Vpsi:10:6);
                write(elk, Theta*RAD:2:6,' ', Beta*RAD:10:6,' ', Pr:10:6,' ', Ppsi:10:6);
                writeln(elk,' ', PVr:10:6,' ', PVphi:10:6,' ', PVpsi:10:6,' ', Hamil:8:2);
                t := t + dt;
            end;
        write('R Final: ',rfinal:1:5, ' Phi Final: ',Phifinal*RAD:1:5);
        writeln(' Psi Final: ',Psifinal*RAD:1:5);
        write('Vr Final: ',Vrfinal:1:5 , '    Vphi Final: ',Vphifinal:1:5);
        writeln(' Vpsi Final:  ', Vpsifinal:1:5);

        close(f);
        close(elk);
end.
```

VITA

Brian Kendell Funk was born on June 9, 1968 in Ridgecrest, California. At the time, his father, Kennerly W. Funk, was employed by the Naval Weapons Center in China Lake, California. His mother, Janet K. Funk, was an elementary school teacher. Brian's sister, Stacie J. Funk was born on October 9, 1971. In 1976, Brian moved to Monterey, California where his father pursed a Master's degree at the Naval Post Graduate School. Brian's family moved to Vienna, Virginia in 1978. His parents are still residents of the area. His father works in the Air Systems Office at the Pentagon, and his mother is the principal at Riverside Elementary School. Brian graduated from James Madison High School in 1986, where he met his future fiancee Carolyn Marie Cleary. After high school, Brian attended Georgia Tech and worked for the BDM Corporation as a cooperative student. In 1991, Brian graduated with a Bachelor of Aerospace Engineering degree with highest honors. Immediately after graduation, Brian began pursuing a Master's degree in Aerospace Engineering at the University of Tennessee Space Institute in Tullahoma, Tennessee. He worked as a Graduate Research Assistant to Dr. Gary A. Flandro. The primary area of his research was orbital mechanics and interplanetary mission design. Brian received his Master of Science degree in May of 1993, and began working for Texas Instruments shortly thereafter.