SIX DEGREE OF FREEDOM OPTIMAL TRAJECTORIES FOR SATELLITE RENDEZVOUS

by

John M. Kruep

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

IN

AEROSPACE ENGINEERING

APPROVED:

Frederick H. Lutze, Chair

Mark R. Anderson

Wayne C. Durham

May, 1996

Blacksburg, VA

Key Words: Satellites, Rendezvous, Trajectory, Optimization

SIX DEGREE OF FREEDOM OPTIMAL TRAJECTORIES FOR SATELLITE RENDEZVOUS

by

John M. Kruep

Dr. F. H. Lutze, Chairman

Aerospace and Ocean Engineering

(ABSTRACT)

A method is developed for computing the minimum fuel trajectory for a satellite that moves between two different positions and orientations using a sequence of impulsive burns. The method makes use of the linear Clohessy-Wiltshire equations to describe translational motions, Euler's equations of rigid body motion for describing the attitude motions, and a sequential quadratic programming optimization code. Initial solutions are found assuming no coupling between the translational and rotational motions and with no imposed constraint on the time of the rendezvous. Further solutions are then found by varying the vehicle center of gravity location along one axis, thereby coupling the rotational motions into two axes of translation thrusters, and by imposing time limits on the rendezvous. A discussion of the impact that these parameters have on the optimal solutions for two different models of the satellite thruster systems is then presented.

Acknowledgments

I would like to thank my advisor, Dr. Frederick H. Lutze for all his guidance and encouragement over the course of this project. I would also like to thank Dr. M. R. Anderson and Dr. W. C. Durham for serving on my committee.

I would like to gratefully acknowledge the Department of Aerospace and Ocean Engineering of the Virginia Polytechnic Institute and State University for its financial support during this research.

List of Symbols

$\mathbf{A}(\psi_h, \theta_h)$	Rotation matrix to go from inertial to angular momentum axes
$\mathbf{B}(\psi, \theta, \phi)$	Rotation matrix to go from angular momentum to body axes
$\mathbf{C}(\Psi, \Theta, \Phi)$	Rotation matrix to go from inertial to body axes
c	Cosine
c_{ijf}	Direction cosine and elements of **C** matrix at end of coasting arc
c_{ijo}	Direction cosines and elements of **B** matrix at start of coasting arc
CG	Center of Gravity
g	Gravitational acceleration on the surface of the Earth
g_i	Equality constraint i
h_i	Inequality constraint i
$\vec{H}$	Angular momentum vector
(H_x, H_y, H_z)	Angular momentum vector components
$\Delta\vec{H}$	Change in angular momentum vector from thruster burn
$(\Delta H_x, \Delta H_y, \Delta H_z)$	Change in angular momentum vector components from burn
I_{sp}	Thruster specific impulse
I_x	Moment of inertia about satellite x-axis (and y-axis)
I_z	Moment of inertia about satellite z-axis
J	Cost function
m_f	Mass of satellite after thruster burn

m_p	Mass of fuel used during thruster burn
$\overline{r_o}$	Initial separation distance
(q_o, q_1, q_2, q_3)	Euler parameters representing satellite orientation
$(q_{of}, q_{1f}, q_{2f}, q_{3f})$	Euler parameters at end of trajectory
$(q_{og}, q_{1g}, q_{2g}, q_{3g})$	Goal Euler parameters at end of trajectory
(x, y, z)	Nondimensional position
(x_f, y_f, z_f)	Final nondimensional position
(x_o, y_o, z_o)	Initial nondimensional position
$(\overline{x_o}, \overline{y_o}, \overline{z_o})$	Initial (dimensional) position
(x', y', z')	Nondimensional velocity components
(x'_o, y'_o, z'_o)	Initial nondimensional velocity components
s	Sine
t	Time
τ	Nondimensional time
τ_{max}	Maximum allowed nondimensional time for rendezvous
$\hat{\tau}$	Specified nondimensional rendezvous time
T_{ICW}	Transformation matrix from CW to inertial axes
T_{BI}	Transformation matrix from inertial to body axes
$\vec{V}$	Velocity vector
(V_x, V_y, V_z)	Velocity vector components

$\Delta\vec{V}$	Change in velocity vector from thruster burn
$(\Delta V_x, \Delta V_y, \Delta V_z)$	Change in velocity vector components from thruster burn
V_{dim}	CW nondimensionalization factor for velocity
ω	Orbital angular velocity
(ψ, θ, ϕ)	3-1-3 Euler angle sequence from angular momentum to body axes
$(\psi_o, \theta_o, \phi_o)$	Euler angle sequence from angular momentum to body axes at start of coasting arc
$(\psi_f, \theta_f, \phi_f)$	Euler angle sequence from angular momentum to body axes at end of coasting arc
(Ψ, Θ, Φ)	3-1-3 Euler angle sequence from inertial to body axes
$(\Psi_o, \Theta_o, \Phi_o)$	Euler angle sequence from inertial to body axes at start of coasting arc
$(\Psi_f, \Theta_f, \Phi_f)$	Euler angle sequence from inertial to body axes at end of coasting arc
$(\Psi_g, \Theta_g, \Phi_g)$	Goal orientation Euler angle sequence from inertial to body axes
(ψ_h, θ_h)	Euler angle rotations from inertial to angular momentum axes
$(\dot{\psi}, \dot{\theta}, \dot{\phi})$	Rates of change of angular momentum to body axes Euler angles
Z_{off}	Satellite center of gravity offset from nominal position

Table of Contents

List of Figures

CHAPTER 1: INTRODUCTION AND BACKGROUND

At some future time, it conceivable that unmanned satellites orbiting the earth will be used to service other satellites in need of repair or resupply. These service satellites likely will navigate to, and perhaps dock with, their target satellites autonomously, or without humans to guide them. It is also likely that the target satellites will have a preferred orientation for docking. Therefore, the service satellites will need some type of six degree of freedom guidance scheme to lead them to a proper rendezvous with their target. It is highly desirable for this guidance scheme to accomplish the rendezvous an optimal fashion; that is, one that uses the least amount of propellant.

Using a six degree of freedom trajectory solver offers the opportunity to investigate the effect that an off-nominal vehicle center of gravity (CG) can have of the minimum fuel solution for a rendezvous. Normally, the vehicle center of gravity and divert (translation) thrusters are in a fixed alignment, so that no moment is produced when the divert thrusters are fired. It is possible that the center of gravity will not be in its nominal position, due to mass estimation errors, asymmetric fuel usage, or other factors. If the satellite center of gravity is at an off-nominal location, the divert thrusters will produce a moment whenever they are fired, resulting in a coupling between the translational and rotational motions.

Other researchers have studied problems similar to this one in the past. Clohessy and Wiltshire[1] developed translational equations of motion for use in satellite rendezvous. Simmon, Bergmann, et al.[2] developed a trajectory planner and guidance scheme to help plan optimal autonomous rendezvous, but little was published about the specifics of the solution method. They did publish a reasonable amount about what they did with their algorithm, however. For most of their results, they used a simulation of the Space Shuttle Orbiter and concentrated on studying the effects of obstacles in the satellite's path, gravitational gradients (that are usually neglected), and limitations of the shuttle's thruster jets. They did not discuss much information regarding the effects of misaligning the satellite center of gravity or of significantly constraining the time of the rendezvous.

The purpose of this thesis is to develop a method to find the optimal (minimum fuel) trajectory for moving an orbiting satellite between two different positions and orientations using a set of impulsive burns. This method will then be used to assess the impacts that off nominal CG locations have on the optimal solutions, as well as the effects that imposing time constraints on the rendezvous have on the optimal solutions. The differences that result using two different models of the satellite thruster systems will also be studied. At the present time, no attempt has been made to link this trajectory solver to a guidance scheme.

CHAPTER 2: PROBLEM FORMULATION

2.1 Optimization Overview

Prior to performing any optimization, a number of factors that define the optimization must be established. First, the optimization code user must choose the quantity that is to be minimized by the optimizer, which is called the cost function. In this study, two different cost functions are used, based upon two different models of the satellite thruster systems. Both cost functions are representations of the sum of the magnitudes of the impulsive burns by the satellite's thrusters. Once a cost function has been established, a set of parameters or variables that the optimzer may vary to attempt to minimize the cost must be chosen. For this study, the parameters used are the components of the impulsive thruster burns and times that these burns take place during the trajectory.

In addition to a cost function, many optimizations, including this one, use a set of constraints that somehow restrict the combinations of the parameters that may be used. These constraints typically come in two forms: those that must be satisfied exactly, called equality constraints, and those that prevent only a certain range of values from being used, called inequality constraints. In this study, equality constraints are used to ensure that the satellite reaches the desired position and orientation at the end of the trajectory. Inequality constraints are used to limit the values that the variables may take on to reasonable numbers, and in some cases to restrain the maximum amount of time that may be used to accomplish the rendezvous.

Once the cost function, constraints, and parameterization of the optimization problem have been established, the user must provide a means to compute the values of the cost function and constraints given a specific set of variables. In this case, the optimization code is written in FORTRAN 77 and the author has developed a set of subroutines that can: compute the trajectory that the satellite follows, given a set of thruster burns; provide a measure of how close the satellite is to its target position and orientation at the end of the trajectory; and sum the magnitudes of the thruster burns in a specified fashion for the cost function. The trajectory propagation requires two components: a means of computing the trajectory the satellite follows in between thruster burns, which is done through analytical integration of the equations of motions, and a method to account for the effects of thruster burns on the trajectories. Both are developed later in this chapter.

An overview of the remaining sections of this chapter is as follows. First, the translational trajectory equations of motion and their solutions are presented. Next, the rotational equations are developed. Then, additional equations are derived to allow coupling between the translational thrusters and rotational motions. The optimization code itself is then discussed, including a summary of the specific optimization variables, constraints, and cost functions, and the two satellite thruster models used for this study. Finally, a discussion of using the optimzer to find solutions is presented.

2.2 Translational Equations

The translational trajectory equations assume that the satellite to be serviced (called the target satellite) is in a circular orbit, and sits at the origin of the Clohessy-Wiltshire (CW) coordinate system shown in Figure 1. This coordinate system has its x-axis pointing radially outward from the earth, its y-axis pointing along the direction of the target's motion, and its z-axis perpendicular to the x- and y-axes (and the plane of the orbit) to complete a right-handed coordinate system. The system rotates around the earth with the constant orbital angular velocity of the target, and thus remains fixed to the target at all times. The servicing satellite, henceforth called the chase satellite, is assumed to start at a position $(\overline{x}_o, \overline{y}_o, \overline{z}_o)$ relative to the target in this coordinate system and, for the cases looked at in this study, is assumed to have no initial motion relative to the target. Zero initial relative motion was chosen simply to help limit the number of initial conditions that could be investigated. The algorithms developed herein would work just as well with a nonzero initial relative velocity.

The translational equations of motion used are the linear Clohessy-Wiltshire (CW) equations, as found in reference 3. These equations may be nondimensionalized as follows:

$$\begin{aligned} &\tau = \omega t & \qquad & x = \overline{x} / \overline{r}_o \\ &\frac{d()}{dt} = \omega \frac{d()}{d\tau} & & y = \overline{y} / \overline{r}_o \\ & & & z = \overline{z} / \overline{r}_o \end{aligned} \tag{1}$$

where $\overline{r_o}$ is the initial separation distance, ω is the constant angular velocity of the target satellite's orbit, τ is the nondimensional time, and (x, y, z) are the nondimensional position coordinates. This method of nondimensionalization is not standard, but it allows the solutions to be scaled linearly with the value of $\overline{r_o}$. The nondimensional equations of motion then become:

$$\begin{aligned} x'' - 2y' - 3x &= 0 \\ y'' + 2x' &= 0 \\ z'' + z &= 0 \end{aligned} \qquad (2)$$

where $(\ )' = d/d\tau$. These equations describe the relative motion of the chase satellite with respect to the target satellite (which is at the origin of the CW system). It is worth noting that for this system of equations, the in-plane (x-y) motion is decoupled from the out-of-plane (z) motion, though no real use of this fact will be made in this study. Equations (2) have an analytical solution given by:

$$\begin{aligned} x &= x_o + (3x_o + 2y_o')(1-\cos\tau) + x_o'\sin\tau \\ y &= y_o - (6x_o + 3y_o')\tau + (6x_o + 4y_o')\sin\tau - 2x_o'(1-\cos\tau) \\ x' &= x_o'\cos\tau + (3x_o + 2y_o')\sin\tau \\ y' &= y_o' - (6x_o + 4y_o')(1-\cos\tau) - 2x_o'\sin\tau \\ \\ z &= z_o\cos\tau + z_o'\sin\tau \\ z' &= z_o'\cos\tau - z_o\sin\tau \end{aligned} \qquad (3)$$

By assigning a position (x_f, y_f, z_f) to be achieved at a given nondimensional time $\hat{\tau}$, equations (3) can be solved for the initial velocities (x_o', y_o', z_o') required to achieve that position:

$$
\begin{aligned}
x_o' &= \frac{x_o(3\hat{\tau}\cos\hat{\tau} - 4\sin\hat{\tau}) + 2(y_o - y_f)(1-\cos\hat{\tau}) + x_f(4\sin\hat{\tau} - 3\hat{\tau})}{8(1-\cos\hat{\tau}) - 3\hat{\tau}\sin\hat{\tau}} \\
y_o' &= \frac{2x_o[7(\cos\hat{\tau} - 1) + 3\tau\sin\hat{\tau}] - (y_o - y_f)\sin\hat{\tau} + 2x_f(1-\cos\hat{\tau})}{8(1-\cos\hat{\tau}) - 3\hat{\tau}\sin\hat{\tau}} \\
z_o' &= \frac{z_f - z_o\cos\hat{\tau}}{\sin\hat{\tau}}
\end{aligned}
\qquad (4)
$$

Equations (4) can provide solutions for all times except for cases where the equations' denominators go to zero (i.e., at $\hat{\tau} = 0, \pi, 2\pi$, etc. and when $8(1-\cos\hat{\tau}) - 3\hat{\tau}\sin\hat{\tau} = 0$). These restrictions are of only minor concern, though, because equations (4) are only used to help provide an initial guess to the solution.

Equations (3) and (4) have been coded in FORTRAN 77 for use in finding the optimal trajectories. Equations (4) are used in a routine that can provide an initial guess for a translational trajectory solution that satisfies the optimizer's equality constraints on the satellite's final position. Equations (3) are used in a routine that analytically propagates the translational trajectories between impulsive thruster burns. During a divert thruster burn, the velocity vector $\vec{V}$ (or (V_x, V_y, V_z) in CW axis components) discontinuously changes. The divert burn is characterized as an instantaneous change in the velocity vector, $\Delta\vec{V}$, and is represented in its component form as (ΔV_x, ΔV_y, ΔV_z). When a divert burn occurs, $\vec{V}$, is added to $\Delta\vec{V}$ to produce a new vector $\vec{V}$, i. e.:

$$
\begin{Bmatrix} V_x \\ V_y \\ V_z \end{Bmatrix}^{new} = \begin{Bmatrix} V_x \\ V_y \\ V_z \end{Bmatrix} + \begin{Bmatrix} \Delta V_x \\ \Delta V_y \\ \Delta V_z \end{Bmatrix} \qquad (5)
$$

The trajectory propagation may now proceed with a new initial velocity vector.

To guarantee that the chase satellite will arrive at the origin of the CW coordinate system, a set of equality constraints is used with the optimization code. These constraints are formulated to require that the dimensional position at the end of the flight be (0,0,0), i.e.:

$$
\begin{aligned}
g_1 &= x_f \overline{r_o} = 0 \\
g_2 &= y_f \overline{r_o} = 0 \\
g_3 &= z_f \overline{r_o} = 0
\end{aligned}
\qquad (6)
$$

where (x_f, y_f, z_f) are the nondimensional coordinates at the end of the flight and g_1, g_2, and g_3, are the constraints. The dimensional position is used instead of the nondimensional position to promote a tighter tolerance in the optimizer's solution.

2.3 Rotational Equations

To understand the development of the rotational equations of motion, it is first necessary to define a few coordinate systems. The first system is the inertial coordinate system, which is fixed in space and is assumed to be coincident with the CW coordinate system at the start of the problem. The next system is the angular momentum coordinate system, which is shown relative to the inertial system in Figure 2. This system has the angular momentum vector $\vec{H}$ as its z-axis, and is related to the inertial system through a rotation ψ_h about the inertial z-axis and then a rotation θ_h about the intermediate x-axis. The last system is the body fixed coordinate system, which is related to the angular momentum system by a 3-1-3 Euler angle rotation sequence of (ψ, θ, ϕ), as shown in

Figure 3, and is related to the inertial axes by a similar 3-1-3 Euler rotation of (Ψ, Θ, Φ). Since (Ψ, Θ, Φ) represent an orientation with respect to a fixed coordinate system, these angles will be used to keep track of the satellite's orientation.

The satellite will now be assumed to be axisymmetric about its z-axis. This assumption is made because many satellites are cylindrical in shape and are nearly axisymmetric. The equations of motion for the angular momentum to body axes Euler angles (ψ, θ, ϕ) take on a very simple form for an axisymmetric body with no external torques, and are given in reference 4 as:

$$\begin{aligned}\dot{\psi} &= H / I_x \\ \dot{\phi} &= [1 / I_z - 1 / I_x] H \cos\theta \\ \dot{\theta} &= 0\end{aligned} \tag{7}$$

where H is the magnitude of the angular momentum vector, and I_x and I_z are the satellite mass moments of inertia about the x- and z- axes, respectively. Since H and θ are constant for a given coasting arc, equations (7) may be easily integrated to determine the values of these Euler angles at the end of the coasting arc:

$$\begin{aligned}\psi_f &= \psi_o + (H / I_x)t \\ \phi_f &= \phi_o + [(1 / I_z - 1 / I_x) H \cos\theta]t \\ \theta_f &= \theta_o\end{aligned} \tag{8}$$

where t is the (actual) time of the coasting arc and (ψ_o, θ_o, ϕ_o) are the angular momentum to body axes Euler angles at the start of the coasting arc. Unfortunately, the initial conditions of this problem are given in terms of inertial to body axes Euler angles (Ψ_o, Θ_o, Φ_o) and inertial to angular momentum axes angles (ψ_h, θ_h). To evaluate equations (8), the initial values (ψ_o, θ_o, ϕ_o) must be found given the inertial to angular momentum axes angles (ψ_h, θ_h) and the inertial to body axes Euler angles at the start of the problem (Ψ_o, Θ_o, Φ_o). To find these angles, the rotation matrices that can be derived from the Euler angles will be called $\mathbf{A}(\psi_h, \theta_h)$, $\mathbf{B}(\psi, \theta, \phi)$, and $\mathbf{C}(\Psi, \Theta, \Phi)$. Then, using a standard procedure[4], a transformation matrix between the inertial and body axes may be written two ways and equated, giving:

$$\mathbf{C}(\Psi, \Theta, \Phi) = \mathbf{B}(\psi, \theta, \phi)\mathbf{A}(\psi_h, \theta_h) \tag{9}$$

To isolate the **B** matrix, $\mathbf{A}^{-1}$ (which is the same as $\mathbf{A}^T$) may be used, giving:

$$\mathbf{B} = \mathbf{C}\,\mathbf{A}^T \tag{10}$$

This equation may be evaluated at the start of the coasting arc by equating the last row and last column of $\mathbf{B}(\psi_o, \theta_o, \phi_o)$, and the corresponding elements in $\mathbf{CA}^T$ to yield:

$$\begin{aligned}
c_{13o} &= s\theta_o s\phi_o = s\psi_h s\theta_h(c\Phi_o c\Psi_o - s\Phi_o s\Psi_o c\Theta_o) - c\psi_h s\theta_h(c\Phi_o s\Psi_o + s\Phi_o c\Psi_o c\Theta_o) + c\theta_h s\Phi_o s\Theta_o \\
c_{23o} &= s\theta_o c\phi_o = s\psi_h s\theta_h(-s\Phi_o c\Psi_o - c\Phi_o s\Psi_o c\Theta_o) - c\psi_h s\theta_h(-s\Phi_o s\Psi_o + c\Phi_o c\Psi_o c\Theta_o) + c\theta_h c\Phi_o s\Theta_o \\
c_{33o} &= c\theta_o = s\psi_h s\theta_h s\Psi_o s\Theta_o + c\psi_h s\theta_h s\Theta_o c\Psi_o + c\Theta_o c\theta_h \\
c_{31o} &= s\theta_o s\psi_o = c\psi_h s\Psi_o s\Theta_o - s\psi_h s\Theta_o c\Psi_o \\
c_{32o} &= -s\theta_o c\psi_o = -s\psi_h c\theta_h s\Psi_o s\Theta_o - c\psi_h c\theta_h s\Theta_o c\Psi_o + c\Theta_o s\theta_h
\end{aligned} \tag{11}$$

where c_{ijo} are the direction cosines and elements of the **B** matrix, and *c* and *s* represent the cosine and sine, respectively. The direction cosines are constrained by the equations:

$$\begin{aligned}
c_{13o}^{\;2} + c_{23o}^{\;2} + c_{33o}^{\;2} &= 1 \\
c_{31o}^{\;2} + c_{32o}^{\;2} + c_{33o}^{\;2} &= 1
\end{aligned} \tag{12}$$

Taking the constraints into account, the values of (ψ_o, θ_o, ϕ_o) may be solved for thusly:

$$
\begin{aligned}
\psi_o &= \tan^{-1}(c_{31o}, -c_{32o}) \\
\theta_o &= \cos^{-1}(c_{33o}) \\
\phi_o &= \tan^{-1}(c_{13o}, c_{23o})
\end{aligned}
\tag{13}
$$

Once the values of (ψ_o, θ_o, ϕ_o) have been obtained, equations (8) may be evaluated. A similar method is then needed to obtain the final inertial to body axes Euler angles (Ψ_f, Θ_f, Φ_f), and was taken from reference 4. Once again using that the fact that $\mathbf{C}(\Psi, \Theta, \Phi) = \mathbf{B}(\psi, \theta, \phi)\mathbf{A}(\psi_h, \theta_h)$, the individual matrix elements may be obtained, this time from the **C** matrix:

$$
\begin{aligned}
c_{13f} &= s\Theta_f s\Phi_f = s\theta_h(c\phi_f s\psi_f + s\phi_f c\psi_f c\theta_f) + c\theta_h s\theta_f s\phi_f \\
c_{23f} &= s\Theta_f c\Phi_f = s\theta_h(-s\phi_f s\psi_f + c\phi_f c\psi_f c\theta_f) + c\theta_h s\theta_f c\phi_f \\
c_{33f} &= c\Theta_f = -s\theta_f c\psi_f s\theta_h + c\theta_f c\theta_h \\
c_{31f} &= s\Theta_f s\Psi_f = s\theta_f(s\psi_f c\psi_h + c\psi_f s\psi_h c\theta_h) + c\theta_f s\theta_h s\psi_h \\
c_{32f} &= -s\Theta_f c\Psi_f = s\theta_f(s\psi_f s\psi_h - c\psi_f c\psi_h c\theta_h) - c\theta_f s\theta_h c\psi_h
\end{aligned}
\tag{14}
$$

These equations may be used to obtain the values of (Ψ_f, Θ_f, Φ_f):

$$
\begin{aligned}
\Psi_f &= \tan^{-1}(c_{31f}, -c_{32f}) \\
\Theta_f &= \cos^{-1}(c_{33f}) \\
\Phi_f &= \tan^{-1}(c_{13f}, c_{23f})
\end{aligned}
\tag{15}
$$

Equations (9), (11), (13), (14), and (15) have also been coded in FORTRAN 77 for use with the optimization code. They are used to analytically propagate the angular trajectories between thruster burns. Like the translational trajectories, it will be assumed that the initial angular velocities (and angular momentum vector) are zero. When an attitude thruster burn occurs, the angular momentum vector $\vec{H}$ (or (H_x, H_y, H_z) in inertial axis components) changes discontinuously. The attitude burn is characterized as an instantaneous change in the angular momentum vector, $\Delta \vec{H}$, and is represented in its component form as (ΔH_x, ΔH_y, ΔH_z). During an attitude burn, $\vec{H}$, is added to $\Delta \vec{H}$ to produce a new vector $\vec{H}$, i. e.:

$$\begin{Bmatrix} H_x \\ H_y \\ H_z \end{Bmatrix}^{new} = \begin{Bmatrix} H_x \\ H_y \\ H_z \end{Bmatrix} + \begin{Bmatrix} \Delta H_x \\ \Delta H_y \\ \Delta H_z \end{Bmatrix} \tag{16}$$

Using equations (15), the new values of H, Ψ_h, and Θ_h may be computed:

$$\begin{aligned} H_{new} &= \sqrt{(H_x^{\;new})^2 + (H_y^{\;new})^2 + (H_z^{\;new})^2} \\ \theta_h^{\;new} &= \cos^{-1}\left(\frac{H_z^{\;new}}{H_{new}}\right) \\ \psi_h^{\;new} &= \tan^{-1}(H_x^{\;new}, -H_y^{\;new}) \end{aligned} \tag{17}$$

Trajectory propagation may now proceed with a new angular momentum vector.

To use these rotational equations of motion with the optimization code, a set of constraints is required to insure that the chase satellite reaches the desired final attitude. Unfortunately, the choice of attitude motion constraints proved problematic. Euler angles can discontinuously jump from $-\pi$ to $+\pi$ (or vice versa) for Ψ and Φ, and from 0 to π (or vice versa) for Θ, if the optimizer happens to be in a region that is near the maximum or minimum value for one of these angles. These discontinuities play havoc with the optimizer's assumption of smooth, continuous cost functions and constraints. If the Euler angles are used alone as constraints in a form similar to equations (6), the optimizer is unable to find a solution in many instances.

A number of different forms for the constraints were investigated before Euler parameters were chosen. Euler parameters represent the orientation of a vehicle using four parameters q_0, q_1, q_2, q_3 as follows[5]:

$$
\begin{aligned}
q_o &= \cos\frac{\Theta}{2}\cos\frac{\Psi+\Phi}{2} \\
q_1 &= \sin\frac{\Theta}{2}\cos\frac{\Psi-\Phi}{2} \\
q_2 &= \sin\frac{\Theta}{2}\sin\frac{\Psi-\Phi}{2} \\
q_3 &= \cos\frac{\Theta}{2}\sin\frac{\Psi+\Phi}{2}
\end{aligned}
\tag{18}
$$

Because the Euler parameters are composed only of sines and cosines of angles, they are more closely continuous when (Ψ, Θ, Φ) jump to wrap around. In the computer code, the user is required to specify at set of goal Euler angles (Ψ_g, Θ_g, Φ_g), which are converted to a set of goal Euler parameters (q_{0g}, q_{1g}, q_{2g}, q_{3g}) by the program. After the final attitude is obtained in Euler angles (Ψ_f, Θ_f, Φ_f), it is also converted to Euler parameters (q_{0f}, q_{1f}, q_{2f}, q_{3f}). After some experimentation, the following form of the constraints was found to yield the best convergence properties:

$$
\begin{aligned}
g_4 &= (q_{0f} - q_{0g})^2 + (q_{1f} - q_{1g})^2 + (q_{2f} - q_{2g})^2 = 0 \\
g_5 &= (q_{1f} - q_{1g})^2 + (q_{2f} - q_{2g})^2 + (q_{3f} - q_{3g})^2 = 0 \\
g_6 &= (q_{2f} - q_{2g})^2 + (q_{3f} - q_{3g})^2 + (q_{0f} - q_{0g})^2 = 0
\end{aligned}
\tag{19}
$$

With an acceptable formulation of the final attitude constraints, the optimizer is able to find solutions with fewer problems.

A final note about the rotational equations is in order. To give the flight simulation a more realistic target model, it is assumed that the target satellite rotates about the z-axis of the CW coordinate system with the constant angular rate of its orbit, so that it always keeps the same side facing the earth. This assumption means that the goal orientation is no longer fixed in inertial space, but rather depends on the length of time it takes to accomplish the rendezvous. The user is required to specify the goal set of Euler angles for docking to occur at time = 0, which makes them coincident with the inertial Euler angles, and then the program computes what that orientation will be at the final time by adding the nondimensional time of flight to the first Euler angle, Ψ_g.

2.4 Coupling the Motions

It will now be assumed that when the chase satellite's center of gravity (CG) is in its normal location, the satellite CG and divert thrusters are aligned so that no moment is produced when the divert thrusters are fired. It is conceivable, though, that the center of gravity will not be in its nominal location due to asymmetric fuel usage, mass estimation errors, or other factors. An off-nominal CG location will couple the divert thrusters into the attitude motions of the satellite. This satellite is assumed to be axisymmetric about its z-axis, so in order for it to remain axisymmetric, only CG offsets along the z-axis will be studied.

In order to allow coupling of the translational thrusters into the rotational motions of the chase satellite, it is necessary to develop some additional equations. First, it is convenient to convert the $\Delta\vec{V}$ components in the CW axis system to components in the body axis system. This is accomplished using two rotation matrices, one to go from CW to inertial coordinates, T_{ICW}, and one to go from inertial to body coordinates, T_{BI}, as follows:

$$\begin{Bmatrix} \Delta V_x \\ \Delta V_y \\ \Delta V_z \end{Bmatrix}^B = T_{BI}\, T_{ICW} \begin{Bmatrix} \Delta V_x \\ \Delta V_y \\ \Delta V_z \end{Bmatrix}^{CW} \tag{20}$$

where the B and CW superscripts represent components in the body and CW axes, respectively. The components $(\Delta V_x$ and $\Delta V_y)^B$ will cause moments due to a z CG offset (z_{off}) as follows:

$$\begin{aligned} \Delta H_y{}^B &= -z_{off}\, \Delta V_x{}^B V_{\dim} \\ \Delta H_x{}^B &= z_{off}\, \Delta V_y{}^B V_{\dim} \end{aligned} \tag{21}$$

where $\Delta H_x{}^B$ and $\Delta H_y{}^B$ are the body axis x- and y- components of the angular momentum changes, respectively, and V_{dim} is the CW nondimensionalization factor for velocity, which is equal to $\omega \overline{r_o}$. The body axis angular momentum components must now be converted to the inertial axes, via

$$\begin{Bmatrix} \Delta H_x \\ \Delta H_y \\ \Delta H_z \end{Bmatrix}^I = T_{BI}{}^T \begin{Bmatrix} \Delta H_x \\ \Delta H_y \\ 0 \end{Bmatrix}^B \tag{22}$$

The change in angular momentum due to divert thruster firings may now be added in the same fashion as an attitude burn.

2.5 Optimization Code and Parameters

The optimization code comes from the DNCONF subroutine of the IMSL Math Library[6] and is written in FORTRAN 77. It uses a sequential quadratic programming algorithm to solve the problem:

Given a cost function	$J = f(\vec{x})$,
subject to equality constraints	$\vec{g}(\vec{x}) = 0$
and inequality constraints	$\vec{h}(\vec{x}) \geq 0$

find the design variables $\vec{x}$ that minimize the cost.

The optimization code requires the user to provide a subroutine that will compute the value of the cost function J and constraints $\vec{g}$ and $\vec{h}$ given a set of design variables $\vec{x}$. Using this routine, it computes a gradient of the cost function and constraints, and attempts to proceed down the gradient to a new set of design variables that result in a lower cost function value and/or better constraint satisfaction. The optimizer continues until it finds a minimum cost solution that satisfies the constraint equations. For this problem, the optimization cost function is intended to be a measure of the fuel used by the chase satellite to accomplish the rendezvous with the target. Thus, minimizing the cost function should minimize the fuel used.

To allow the optimizer some flexibility in finding solutions, 5 different burns are allowed: two burns using only the divert (translation) thrusters, two using only the attitude (rotation) thrusters, and one burn using both. Based on the experience gained studying the attitude motions separately, it was decided that one attitude burn should occur at the start of the trajectory. The combined burn is used to stop all motions at the end of the trajectory. The other three burns are allowed to occur at any time between the beginning and the end of the flight, and in any order. The results in Chapter 3 will show that these five burns are more than sufficient for the cases used in this study.

The total number of variables used by the optimizer is 16: two sets of three components of a nondimensional $\Delta\vec{V}$ for the divert burns, specified in the CW coordinate system; two sets of the three $\Delta\vec{H}$ components for the attitude burns, specified in inertial coordinates; the nondimensional times at which each of the three intermediate burns occur; and the positive nondimensional time length between burns 4 and 5. (It was necessary to make the final time variable a positive change in time to prevent the optimizer from using a coasting arc that spanned a negative time.)

Given a specific set of variables, the method by which the variables are used to propagate the satellite's trajectory and compute the cost function and constraints will now be explained. The computer code that was written to perform these calculations for the optimization code is shown in the Appendix.

The subroutine which performs these calculations, called "fcn", is provided with the design variables, initial conditions, and desired final conditions as inputs. Upon entering routine "fcn", the initial velocity and angular momentum vectors and cost function are set to zero, the position (x, y, z) and orientation (Ψ, Θ, Φ) are set to their specified initial values, and the goal Euler angles and parameters are modified (taking into account the nondimensional rendezvous time). Next, the intermediate three burns are passed through a bubble sort algorithm to place them in order by the time each burn occurs. Then a trajectory propagation loop is entered. At the start of each loop iteration, the effects of the next, upcoming burn are implemented. If that burn is an attitude burn, the angular momentum vector is changed using equations (16) and (17). If that burn is a divert burn, the velocity vector is changed using equations (5), and if there is a CG offset, the change in the angular momentum vector is found using equations (20)-(22) and added using equations (16) and (17). At this point, the current burn information is added to the cost function using methods described in subsequent paragraphs. Using the new angular momentum and/or velocity vectors, the translational and rotational trajectories are propagated until the time of the next burn using equations (3), (8), (11), (13), (14), and (15). At this point, the velocity, position, and orientation are updated, and the loop repeats. After the fourth iteration, the loop is exited and the final burn (which is basically the negative of the current velocity and angular momentum vectors) is computed.

Six equality constraints are evaluated at the end of the trajectory and have been mentioned previously. Three constraints (equations (6)) are used to insure the desired final position is attained, and three (equations (19)) are used to insure the desired final attitude is obtained. One optional inequality constraint is used in cases where the user imposes a maximum nondimensional time for the rendezvous to occur. This inequality constraint specifies that the maximum time allowed minus the actual final time must be greater than or equal to zero. There are also de facto inequality constraints which the optimzer evaluates that prevent the variables from taking on unreasonable values (e.g., the times that the burns occur must be greater than 10^{-8}, magnitudes of the burn components must be less than 50 m/s)

Two different models of the chase satellite's thrusters are used in this study, and each has its own cost function. For both cost functions, all of the attitude thrusters were assumed to be at unit distance from the chase satellite's center of gravity, so that the attitude thruster burns can be added directly to the divert burns. Also, the nondimensional velocities used by the CW equations (and the optimzer) need to be redimensionalized by multiplying them by V_{dim}, before they can be added to the attitude burns.

The first thruster model assumes that the satellite's thrusters can be throttled and thrust vectored so that they fire only along the desired direction with the desired amount of impulse. For this case, the cost function is taken to be the sum of the total (dimensional) velocity and angular momentum changes ($\Delta\vec{V}$'s and $\Delta\vec{H}$'s) for each burn, i. e.

$$J = \left|\Delta\vec{V}_1\right| + \left|\Delta\vec{V}_2\right| + \left|\Delta\vec{V}_f\right| + \left|\Delta\vec{H}_1\right| + \left|\Delta\vec{H}_2\right| + \left|\Delta\vec{H}_f\right| \tag{23}$$

where $|\ \ |$ denotes the absolute value, or equivalently the square root of the sum of the squares of the components.

The other satellite thruster model is perhaps a bit more realistic. It assumes that the satellite's thrusters are throttleable but are fixed to its body axes, so that they may only fire along or produce moments about the satellite's body axes. In this case, the cost function must measure the fuel usage by each of the individual body-axis thrusters, i.e.:

$$J = \left(|\Delta V_x| + |\Delta V_y| + |\Delta V_z|\right)_1^B + \left(|\Delta H_x| + |\Delta H_y| + |\Delta H_z|\right)_1^B$$
$$+ \left(|\Delta V_x| + |\Delta V_y| + |\Delta V_z|\right)_2^B + \left(|\Delta H_x| + |\Delta H_y| + |\Delta H_z|\right)_2^B \qquad (24)$$
$$+ \left(|\Delta V_x| + |\Delta V_y| + |\Delta V_z|\right)_f^B + \left(|\Delta H_x| + |\Delta H_y| + |\Delta H_z|\right)_f^B$$

To convert the $\Delta\vec{V}$ and $\Delta\vec{H}$ components to body axis components, the transformation matrices T_{BI} and T_{ICW} must be used, as discussed in section 2.4.

Both of these cost functions are measures of the total impulse imparted to the satellite during its trajectory. To compute the amount of fuel used by a given thruster burn, m_p, the rocket equation as found in reference 7 may be used:

$$m_p = m_f\left(e^{|\Delta V|/I_{sp}g} - 1\right) \qquad (25)$$

where m_f is the mass of the satellite after the burn, I_{sp} is the thruster's specific impulse, and g is the acceleration of gravity at the earth's surface. I_{sp} is typically a number greater than 250 seconds (s), g is about 9.81 m/s^2, and the ΔV's seen in this study are less than 5 m/s. With numbers of these magnitudes in equation (25), it is easy to see that fuel used during a burn is a very small fraction of the satellite's mass and will scale almost linearly with the burn's impulse. Therefore minimizing the sum of the impulses given to the satellite should minimize the fuel used by the satellite during the rendezvous.

2.6 Finding Solutions with the Optimzer

To find a solution using the optimization code, the code must be provided with an initial guess for the solution. The code developed for this study gives the user three options for an initial guess. The first option is simply to manually initialize the array that contains the initial guess, called "xguess." The second requires the user to provide values for the attitude burn components and the times for all of the burns. The code can then provide a guess for a one divert burn solution which will satisfy the position constraints. The final option is to use the results from a previous optimization run. At the end of each run, the code prints out a listing of the first three places of the final values of each variable in a form that is convenient to "cut and paste" into the initialization statement for "xguess."

Each optimization run ends with one of two possibilities: either the optimizer has converged to a solution, or for some reason it has not. Convergence is obtained when the constraints have values less than 10^{-5} and the Kuhn-Tucker optimality parameter, which is a measure of the local flatness of the function/constraint space, is also less than 10^{-5}. If the optimizer has not converged, the user must provide a new initial guess for the solution. Usually, this is done by using the optimizer's final variable values as the new guess (the third method from above). Often, a number of iterations of this method are needed for the optimzer to converge to a solution. Even if the optimzer does converge, more optimization runs may be needed; the optimizer will frequently find a solution that is only locally a minimum in the cost function. There are potentially other solutions that have the true minimum cost function, called a global minimum. To ensure that a solution is a global minimum, it is often advisable to make another optimization run using the solution's final variable set as an initial guess (the third guessing method described above). If the optimizer converges to the same solution again, it reasonable to assume that the solution is a global minimum. If the optimzer converges to a lower cost solution, the first solution is obviously not a global minimum.

For both satellite thruster models, initial guesses were first made for the cases where the Z CG offset (z_{off}) was zero using the second method mentioned above. After finding a solution, an attempt is made to verify that the solution was a global minimum by make additional optimization runs, using both the third guessing method above and other, manually generated guesses. Once a minimum cost solution has been found, that solution is used as a guess (using method three) for cases that have a small positive or negative CG offset. The minimum cost solutions for these CG offsets are then used as guesses for larger CG offsets. Each subsequent minimum cost solution is used as a guess for the next larger CG offset solution, and in this manner solutions are obtained for a range of CG offsets.

A similar technique is used for cases where the rendezvous time is constrained. Initial solutions are found for cases with no time constraint. Then, the time allowed for the rendezvous is reduced slightly, using the unconstrained solution as an initial guess. Again, each subsequent minimum cost solution is used as an initial guess for the next lower time solution to obtain solutions over a range of time constraints.

CHAPTER 3: RESULTS AND DISCUSSION

3.1 Conditions of the Rendezvous

Solutions for both satellite thruster models were obtained assuming the chase satellite's initial distance from the target ($\overline{r_o}$) is 2.5 km, the nondimensional position (x, y, z) is (0.5, 0.7071, 0.5), the initial orientation (Ψ_o, Θ_o, Φ_o) is (0, 0, 0), and the goal orientation (if the rendezvous occurred at time = 0) (Ψ_g, Θ_g, Φ_g) is (45°, 30°, 45°). The chase satellite was chosen to have moments of inertia (about the x-, y-, and z-axes, respectively) of I_x = 3376, I_y = 3376, and I_z = 1057 kilogram-meters^2 (kg-m^2) and a mass of 14,600 kg. These values were based upon an enlarged version of a military satellite which was considered for this type of rendezvous operation. The target satellite is taken to be in a 300 km altitude circular orbit, which has an orbital angular rate of 1.15687 x 10^{-3} radians/second (rad/s). These conditions result in a velocity dimensionalization factor (V_{dim}) of 2.8922 meters/second (m/s).

3.2 Thrust Vectoring Solutions

Figures 4 through 10 present the data obtain assuming the chase satellite uses thrust vectoring thrusters. The overall minimum cost solution found for a nominal CG location has a cost of about 4.88 and a nondimensional flight time of about 4.95 (radians). (The units on the cost function are m/s for the divert burns, and 1 meter*(m/s) for the attitude burns. Hereafter, the units will be reported only as m/s for convenience) This solution indicates that the optimizer chose to collapse both of the allowed attitude burns and one of the divert burns into (essentially) a single burn that occurs at the start of the flight. The other divert burn occurs about 1/3 of the way through the flight. As described in Chapter 2, this nominal minimum cost solution is then used as a guess for the first for the first off-nominal CG solutions. Each subsequent solution is then used as a guess for the next larger CG offset solution. In this manner, solutions are found for a range CG locations.

Figures 4 and 5 present the data obtained by varying the CG offset. Figure 4 shows a plot of the value of the cost function vs. z_{off}. This curve is generally smooth, but has a few noticeable wiggles in it. These wiggles are probably due to the fact that since the optimizer is only really using three burns (but is allowed five), there is more than one way to represent the same solution. A major point revealed from Fig. 4 is that the value of the cost function doesn't change drastically (only about 14% over the entire CG range). Its minimum is about 4.59 m/s at an offset of -15 cm, and its maximum is about 5.23 m/s at an offset of +15 cm. The small variation in the cost function can probably be traced to the fact that the divert thrusters are required to fire more strongly than the attitude thrusters; the attitude thruster firings account for only 17.2%, 22.6%, and 11.7% of the total cost function for the nominal, +15 cm , and -15 cm CG offset cases, respectively. Since the attitude motions account for only a relatively small fraction of the total impulse given to the chase satellite, the effects of coupling the rotational and translational motions are fairly small.

Figure 5 shows a plot of the in-orbit-plane and out-of-plane paths for the zero and +/- 15 cm offset solutions. The in-plane path (Fig. 5 (a)) is plotted to mimic the CW coordinate system, with the x-axis positive in the positive vertical direction and the y-axis in the positive in the negative horizontal direction. (It was necessary to plot x vs. negative y in the plotting software to emulate the CW coordinate system.) The out-of-plane path (Fig. 5 (b)) is plotted with the x-axis horizontal and the z-axis vertical. The intermediate divert thruster burn can be seen in the kink near a y value of 0.2 in all three trajectories. Both the in and out-of -plane trajectories don't proceed directly to their targets (the origin). Rather they overshoot the targets slightly and then spiral in to the them. It is also worth mentioning that there are only small differences in the paths taken by all three cases. The slight differences in the trajectories can probably be tied to difference the times of flight. The -15 cm offset trajectory takes place over about 4.92 (rad), the nominal case takes about 4.95, and the +15 cm case takes about 4.97. There is apparently not much of a cost differential involved when choosing times in this range for the thrust vectored cost function.

Figure 6 shows plots of the angular trajectories (inertial to body axes Euler angles vs. time) for the nominal and +/- 15 cm CG offset cases. For the nominal case, the trajectories are all basically straight lines from their initial to final values, which is indicative of the optimizer's elimination of the intermediate attitude burn. The + 15 cm and - 15 cm offset cases both have noticeable kinks in the trajectories of Ψ and Φ at a time of about 1.7, the time of the intermediate divert thruster burn. As one would expect, the kinks on the positive and negative CG offset trajectories bend the trajectories in opposite directions. For this case, it just happens that the -15 cm CG offset trajectory gets a divert thruster induced kick that is in the direction the satellite needs to go, which results in a lower cost function value.

One point needs to be made about the initial starting point of the angular trajectories. The satellite body axes are taken to be coincident with the inertial axes at the start of the flight, which corresponds to angles (Ψ, Θ, Φ) equal to (0, 0, 0). This location is a singularity for a 3-1-3 rotation sequence because the middle angle is zero. As such, the sequence (0, 0, 0) is equivalent to any rotation ($+\varepsilon$, 0, $-\varepsilon$). The integration of the rotational equations of motion is analytic from the beginning to the end of each coasting arc, so this discontinuity doesn't cause a problem in the computer code. However, the angular trajectories may appear as if they are starting from different places when plotted. In reality, though, all of the angular trajectories start from the same orientation.

The discussion will now turn to the solutions obtained for the cases where the time of the rendezvous was constrained. The solutions for these cases are obtained in a manner similar to the offset CG solutions, as described in section 2.6. From the results obtained, it can be seen that constraining the time of the rendezvous can alter the character of the solutions; as the maximum nondimensional rendezvous time(τ_{max}) drops below about 2.0, the optimizer drops the intermediate divert thruster burn, leaving only a combined burn at the beginning and end of the trajectories. Solutions have been obtained for values of τ_{max} as low as 1.2. Below that value, it becomes increasingly difficult for the optimizer to converge to a solution with the problem formulated to allow five different burns.

Figure 7 shows a plot of cost function vs. the maximum nondimensional rendezvous time allowed for the nominal and +/- 15 cm CG offset cases. This plot features a dramatic rise in the cost function (from about 4.9 to 13.1 m/s at $\tau_{max} = 1.2$) for all cases as the rendezvous time is reduced further and further. The -15 cm offset case, though initially lower in cost than the nominal case, rises in cost more quickly than the nominal and eventually has a higher cost function value. The +15 cm case is initially higher in cost than the nominal, but it grows more slowly than the nominal and is eventually lower in cost. As the time is constrained more and more, all three curves appear to be converging.

Also shown on Figure 7 is an alternate solution for the nominal CG offset case. This alternate solution has a lower cost function (11.9 vs. 13.1 m/s at $\tau_{max} = 1.2$) for very low values of τ_{max}, though as time is increased it quickly becomes a higher cost solution. The alternate solution is from a different solution family from the original solution. A solution family is a group of solutions that use the same type of burn sequence to satisfy the constraints, though the magnitudes of the burns change with z_{off} or τ_{max}. Different solution families often have similar costs, but there is usually one family that has the true minimum cost. For this particular case, the alternate solution family has a virtually identical translational trajectory, but has a rotational trajectory that goes in the opposite direction from the original solution. (This difference will be shown in a discussion of the angular trajectories shortly.) The +/- 15 cm CG offset cases also possessed alternate solutions of a similar cost and character to the nominal case. These alternate solutions are not shown.

Figure 8 shows a plot of the translational trajectories for the most highly time constrained case, $\tau_{max} = 1.2$. It can easily be seen from this figure that all three of the CG offset trajectories are essentially the same. This similarity is probably because there is only a brief amount of time to accomplish the rendezvous, and the optimizer has chosen to use only two burns to do it. As a result, any noticeable deviation from the lowest cost trajectory will cost substantially more and/or add more burns. It is also worth noting that these trajectories take a much more direct route to the target (the origin) than the time free trajectories (Fig. 5). Again, this direct routing is probably because of the severe time restrictions imposed on the trajectories; there simply is no time for much indirect motion.

The angular trajectories for the time constrained ($\tau_{max} = 1.2$) cases are shown in Figure 9. Figures 9 (a), (b), and (c) are the trajectories for the original solution family, and they are all virtually identical. The large spikes are due to the Euler angle Φ jumping from its minimum to maximum value to keep within its range limits. Figure 9 (d) contains the trajectory for the alternate solution. It proceeds in the opposite direction for angles Ψ and Φ from the original solution, and in this case has lower magnitude slopes than the original solution does. The magnitude of the slope of the trajectory is directly related to the size of the angular momentum vector, which is in turn directly related to the cost function. Thus, the alternate lower slope trajectories cost less than the original trajectories.

As a final point about the thrust vectoring solutions, it is interesting to note how much the attitude thruster firings have increased in significance due to the time constraint. The percentages of the total cost function due to attitude burns for the original solution family are 60.2%, 59.6%, and 60.7% for the nominal, +15 cm, and -15 cm offset cases respectively. For the alternate solution family, the percentages are 55.7%, 55.5%, and 55.8%, respectively. (The unrestricted time values are 17.2%, 22.6%, and 11.7%.) The attitude thruster firings have gone from relatively minor effects to become the dominant driver of the cost function. As a result, the alternate solution family can offer a substantial reduction in the cost function.

3.3 Body Fixed Thruster Solutions

The rendezvous conditions for the chase satellite with body fixed thrusters are exactly the same as for the thrust vectored case, as is the solution methodology used to find the optimal rendezvous. Like the thrust vectored case, the optimizer chose to use only three burns to accomplish the rendezvous. However, the time for the intermediate divert thruster burn shifted from about 1.7 to about 2.7 rad, and the total time of flight dropped to about 4.68 rad from 4.95. The results from the body fixed rendezvous are presented in Figures 10 through 15.

Figure 10 shows the cost function vs. z_{off} for the nominal case ($z_{off} = 0$) stepped outward through $z_{off} = +15$ cm and -15 cm. From Figure 10, a few points may be made. First of all, the value of the cost function is significantly higher using body fixed thrusters

(6.10 m/s for the nominal CG case) than using thrust vectoring (4.88 m/s for the nominal case). The cost function shows the same general trend of increasing with increasing CG offset as the thrust vectored case, but in this case the curve has a minimum cost function (5.70 m/s) at an offset of about -12 cm. In looking at the solutions at and around this value of offset, it is apparent that -12 cm is the point where the additional moment caused by the divert thruster coupling has reached its most beneficial value. For more negative CG offsets, the moment caused by the coupling is too large, and extra attitude thrust must be expended to compensate for it. It is also worth noting that this curve has fewer wiggles in it than the thrust vectored curve does. The body fixed curve is probably more smooth because the rotational and translational motions are more closely coupled here, since the direction that the satellite is pointed has a direct impact on the divert thruster firings needed to accomplish the rendezvous.

Figure 11 shows the nondimensional in- and out-of-plane paths for the body fixed thruster case. As one can seen from Fig. 11, all three of the CG offset trajectories lie virtually on top of each other, both in and out of the orbit plane. These trajectories also loop around the target to a considerable degree before hitting the target point. Contrast this trend with the thrust vectored trajectories (Fig. 5), where the trajectories take a more direct approach to the target and are slightly different depending upon the CG offset. The trajectory differences between the thruster models likely occur because when the body fixed burns take place is of much greater importance than when the thrust vectored burns occur, since the direction the body fixed satellite is pointing will help determine the cost of the burn needed. All three body fixed trajectories must need to have their burns at the same time and follow the same path to insure that the body axes components of the thruster firings are minimized.

Figure 12 shows plots of the angular trajectories for the nominal and z_{off} = +/- 15 cm cases. As with the thrust vectored cases (Fig. 6), the trajectories for the nominal CG location are basically straight lines. Unlike the thrust vectored case, however, the trajectories for Ψ and Φ have much lower magnitude slopes. The +/- 15 cm offset trajectories have what appear to be very rapid changes in Euler angles Ψ and Φ at the beginning of their trajectories. Remember, though, that the initial point in the trajectory is a singularity, and in this case the small value of Θ soon after the start make the changes in orientation look bigger than they really are. As one might expect, the positive and negative CG offsets' trajectories bend in opposite directions, due to their opposite CG locations. It is also worth noting that neither one of these offset trajectories has much of a noticeable kink in the Ψ and Φ plots, as the thrust vectored cases did. This lack of kinks is probably because the intermediate divert thruster burn, which causes the kinks, is significantly smaller in magnitude for these body fixed thruster cases than for the thrust vectored cases.

It is also worth mentioning the percentage of the total cost function due to attitude thruster firings. For the time free cases here, the attitude thrusters are responsible for 20.5%, 26.5%, and 15.8% of the total cost for the nominal, +15 cm and -15 cm offset cases, respectively. These numbers are about 4% higher than for the thrust vectored cases.

The discussion will now turn to the time constrained cases run using body fixed thrusters. Figure 13 shows a plot of the cost function vs. maximum nondimensional time allowed for the nominal CG and z_{off} = +/- 15 cm cases. This plot is very similar to the corresponding thrust vectored plot (Fig. 8) in a number of ways. First, this plot features a very dramatic rise in the cost function (from 6.1 to 19.2 m/s at $\tau_{max} = 1.2$) for all cases as the time is reduced more and more. In these cases, though, the cost function rises even more dramatically than the thrust vectored cases. Second, like the thrust vectored cases, the positive and negative CG offset cases swap being higher/lower than the nominal case as the time of the rendezvous is more constrained. And as with the thrust vectored cases, there is an alternative family of solutions that has a lower cost function in the most highly time constrained cases. Finally, the optimizer generally chooses to eliminate the intermediate divert thruster burn when τ_{max} drop below about 2.0, as it did in the thrust vectored cases.

Figure 14 shows a plot of the translational trajectories for the most highly time constrained case, $\tau_{max} = 1.2$. Two of these trajectories, for the +/- 15 cm offset cases are essentially the same as each other and the thrust vectored trajectories in Fig. 9. As in the thrust vectored case, these trajectories are probably the same because the time restriction severely limits the variation possible in a two burn trajectory. The nominal CG offset case here actually uses a three burn solution for this particular point, and that is why it is noticeably different from the others. This three burn solution is probably an anomaly, the result of the optimizer converging to a local minimum solution. The author made numerous attempts to get convergence to the global minimum, but was unsuccessful in this instance because the cost functions are so close together. The intermediate divert thruster burn takes place very near the end of the trajectory and is very weak, but is far enough away from the end to be distinguished as a separate burn.

A plot of the angular trajectories for the time constrained cases is shown in Figure 15. All four of these plots are virtually identical to those found in Figure 10 for the thrust vectored cases. The major difference in these solutions is just the fact that the body fixed thrust solutions cost considerably more than the thrust vectored solution. For the body fixed thruster solutions here, the attitude burns comprise only 61.1%, 59.3%, and 61.5% of the nominal, +15 cm and -15 cm CG offset cases, respectively. The alternate solution family solutions are 56.5%, 56.3%, and 57.1% attitude thruster burns, respectively. These percentages are just slightly higher than the thrust vectored percentages.

CHAPTER 4: FUTURE RESEARCH OPPORTUNITIES

A number of opportunities for future work in this area come to mind. The first is to look at using sets of initial conditions other than the ones used here. Changing the chase satellite's starting position, orientation, and translational and angular velocities will likely have significant effects on the results. A second idea for future work is to add constraints on the path the chase satellite may take to reach the target. Most real world rendezvous and docking situations will only allow an approach from one direction, and this fact could certainly change the character of the optimal trajectories. Another area that could be investigated is to shut down certain divert thrusters on the body fixed thruster satellite, and observe the effects on the solutions. The chase satellite will likely have to orient itself in very different ways to complete the rendezvous without some of its divert thrusters. A final idea for future work is to join this trajectory solver with a guidance scheme that uses some type of feedback control to execute the rendezvous.

CHAPTER 5: CONCLUSIONS

A method to find the optimal (minimum fuel) trajectory for moving a satellite between two different positions and orientations using impulsive burns has been developed. Solutions have been obtained using this method with two different models of the satellite thruster systems. The solutions found with both satellite models for cases with and without coupling between translational and rotational motions, and for cases with a time limit imposed upon the rendezvous, have been discussed. The following conclusions are drawn:

1) For the center of gravity offsets considered in this study, the coupling between rotational and translational motions does not seem to have a large effect on the value of the cost function, or on the trajectories followed to the rendezvous.

2) Rendezvous time constraints appear to have a substantial effect on increase the value of the cost function and on the trajectories taken. The cost function increases dramatically and the trajectories become more direct in cases with severe time constraints.

3) The thrust vectored and body fixed thruster satellites follow markedly different trajectories to the rendezvous for cases with no constraint on the rendezvous time; the trajectories taken by the thrust vectored satellite are considerably more direct. The thrust vector satellite appears to have a substantially lower cost function for all cases considered in this study.

REFERENCES

1. Clohessy, W. H., and Wiltshire, R. S., "Terminal Guidance Systems for Satellite Rendezvous," *Journal of the Aerospace Sciences*, September 1980, pp. 653-658, 674.

2. Simmons, Ronald J., Bergmann, Edward V., Persson, Bruce A., and Hollister, Walter M., *Six Dimensional Trajectory Solver for Autonomous Proximity Operations*, AIAA Paper 90-3459, 1990.

3. Lutze, Frederick H., "Unaided EVA Intercept and Rendezvous Charts," *Journal of Spacecraft and Rockets*, Article 79-4142, Vol. 16, No. 6, pp. 426-431.

4. Cochran, J. E., Jr., Cheng, Y-M, and Lee, S., *Fixed-Time Reorientations of Rigid Bodies and Lambert's Problem*, AAS Paper 93-582, Presented at the AAS/AIAA Astrodynamics Specialist Conference, Victoria, B.C., Canada, August 16-19, 1993.

5. Junkins, J. L., and Turner, J. D., *Optimal Continuous Torque Attitude Maneuvers*, Presented at the AAS/AIAA Astrodynamics Conference, August 1978.

6. *IMSL Math Library, User's Manual*, subroutine DNCONF, January 1989.

7. Kruep, J., Fields, T., Hahn, C., Molinksy, J., Zarnegar, T., and Thai, B. *Small Reusable Launch Vehicle*, Senior Design Report, Department of Aerospace and Ocean Engineering, Virginia Tech, May 1995.

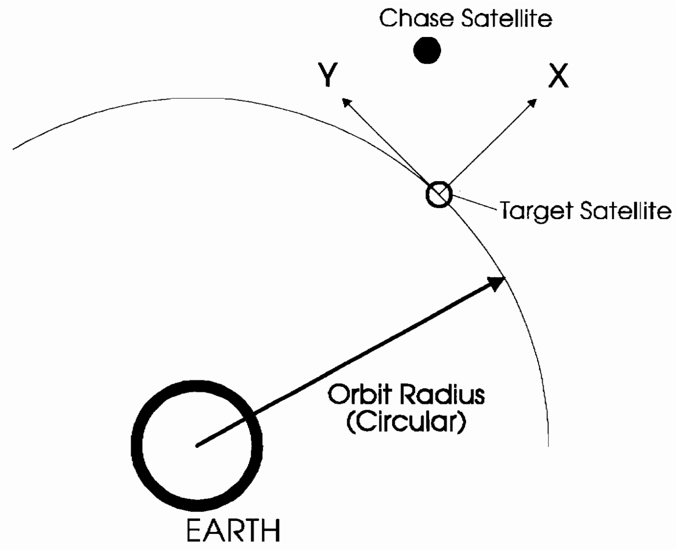


Figure 1. Visual of the Clohessy-Wiltshire (CW) coordinate system.

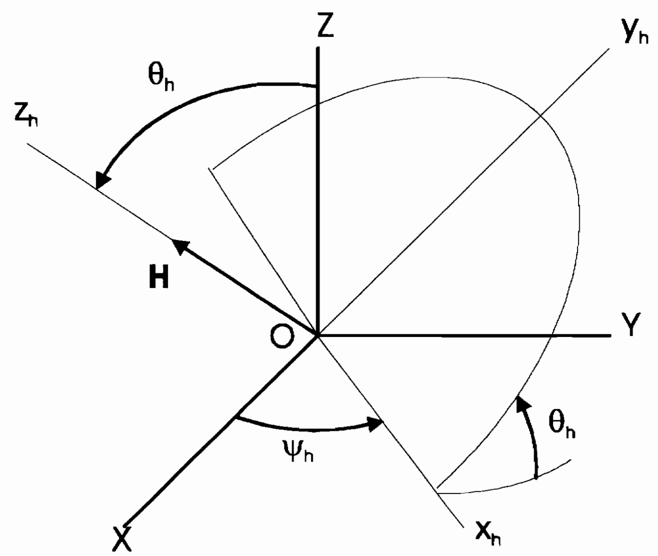


Figure 2. Inertial axes (X Y Z) to angular momentum axes (x_h y_h z_h) rotations.

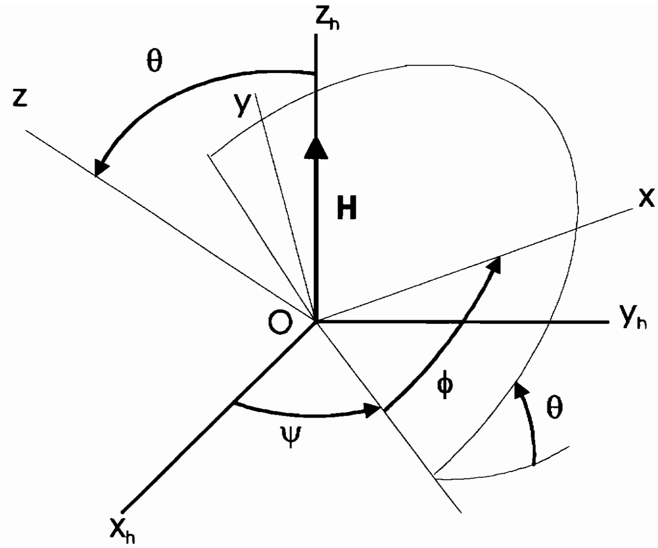


Figure 3. Angular momentum axes to body axes (x y z) rotations.

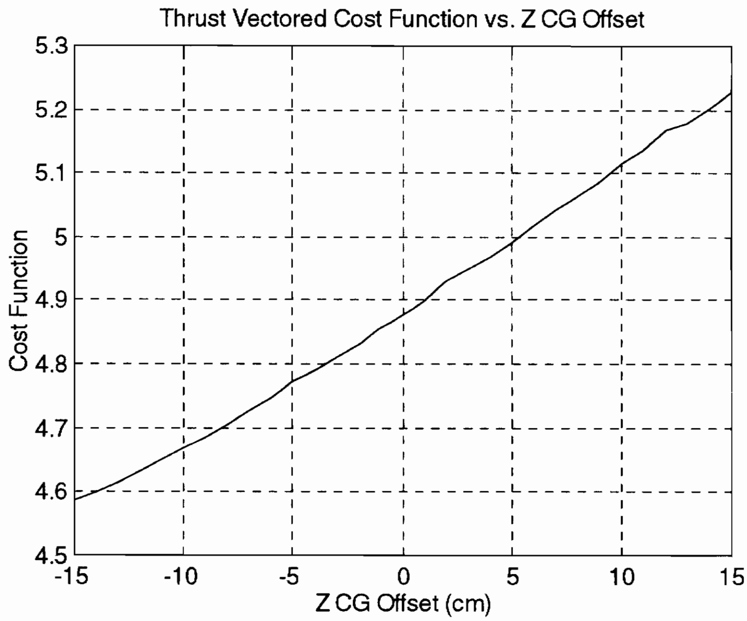


Figure 4. Cost function vs. Z_{off} for the thrust vectoring case.

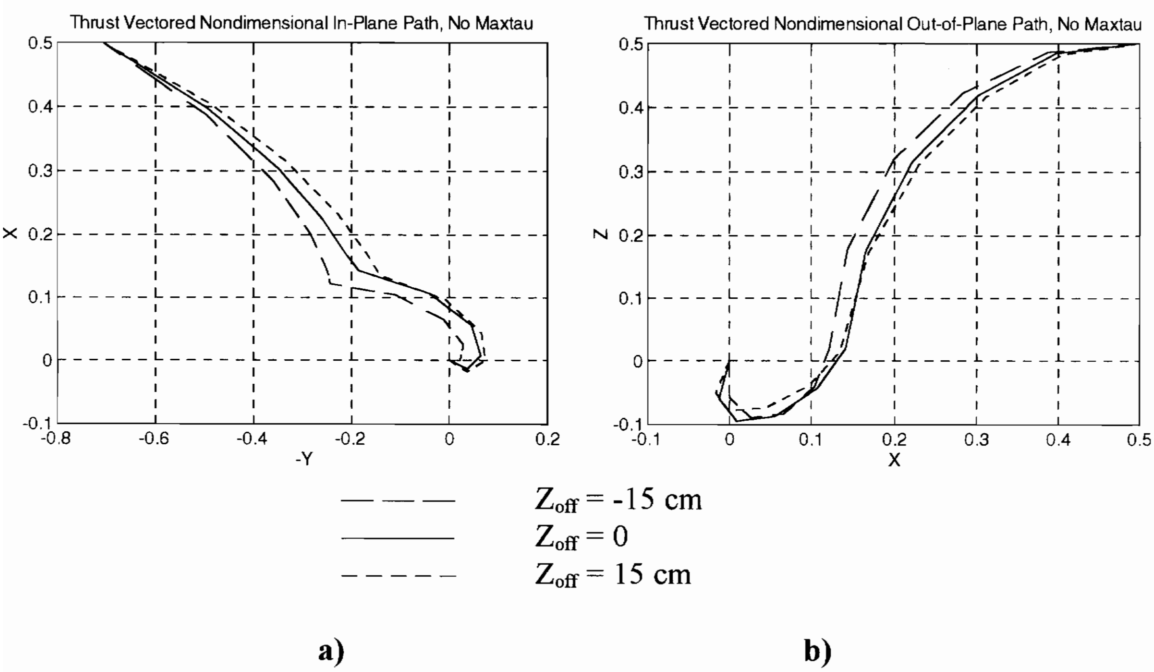
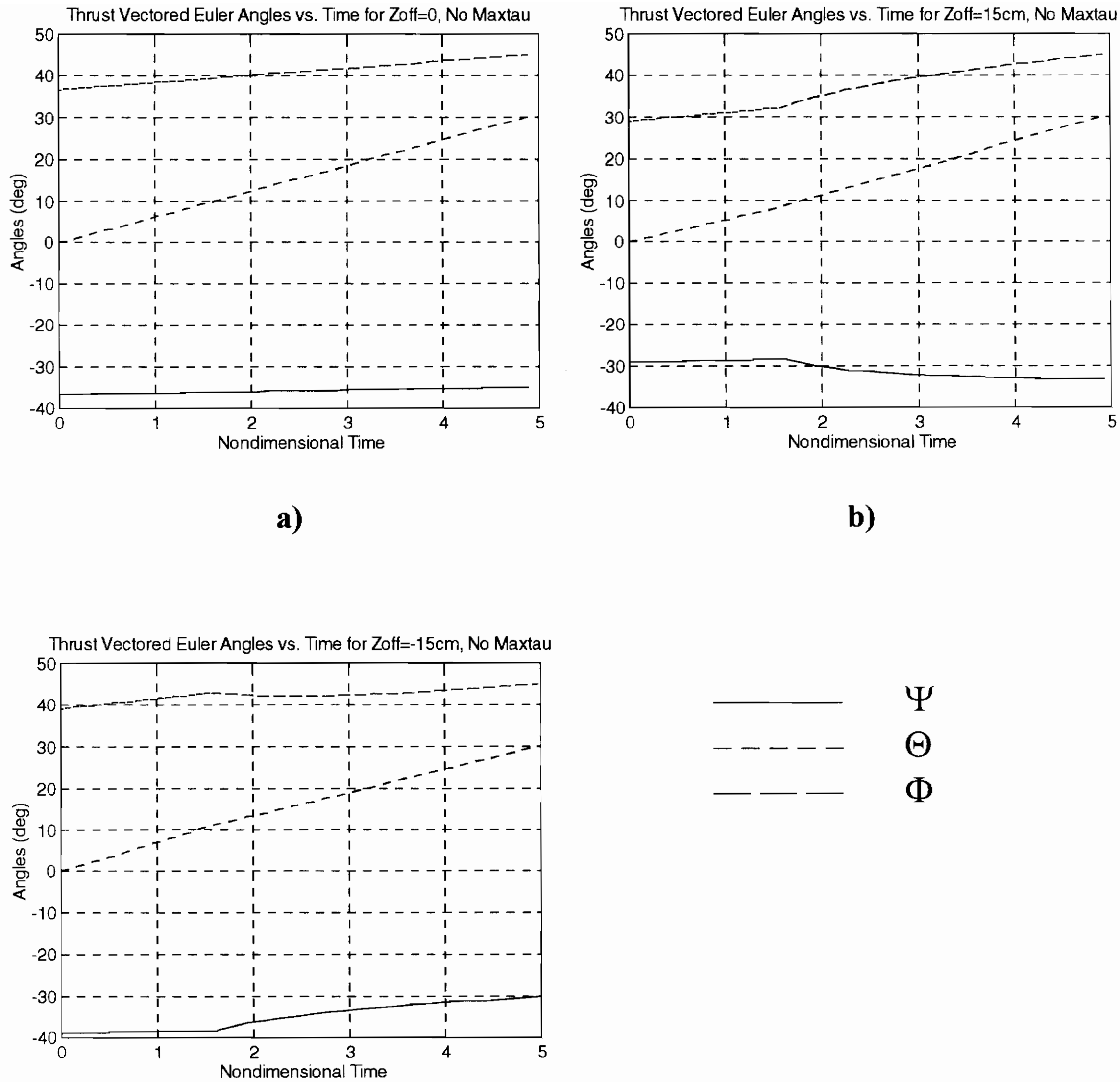


Figure 5. Nondimensional paths of the unrestricted time thrust vectored cases a) in the orbit plane, and b) out of the orbit plane.



c)

Figure 6. Inertial to body axes 3-1-3 Euler angles vs. nondimensional time for the time free thrust vectored cases a) nominal CG, b) Z_{off} = 15 cm, and c) Z_{off} = -15 cm.

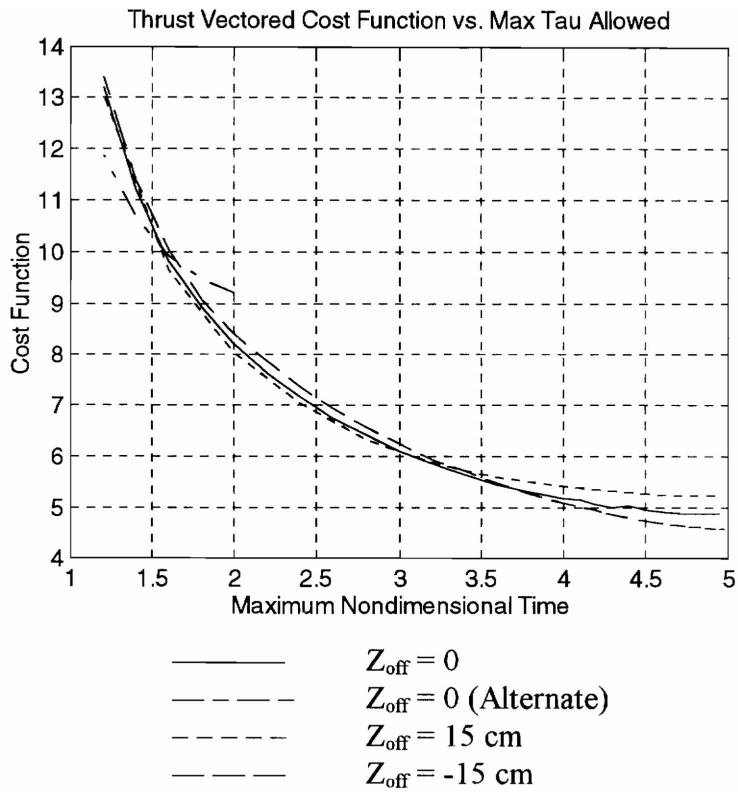


Figure 7. Cost function vs. maximum allowed rendezvous time for the thrust vectored cases.

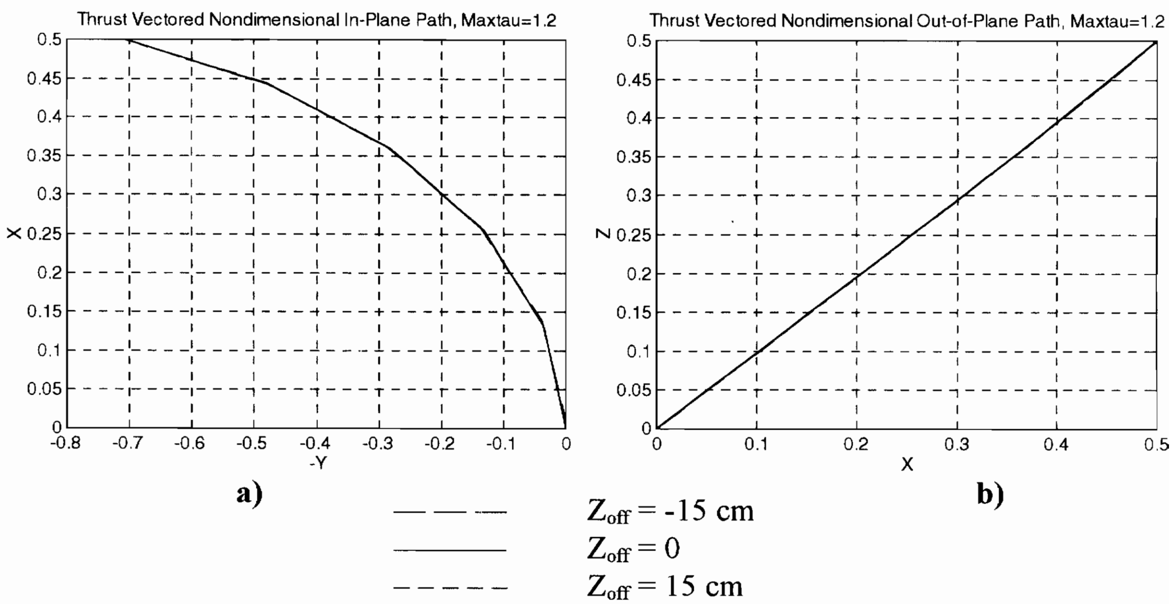
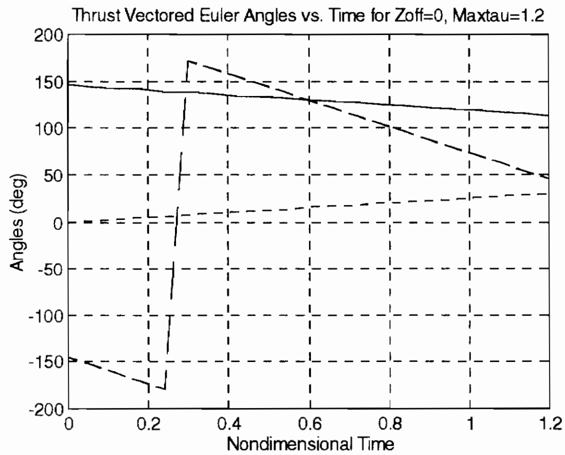
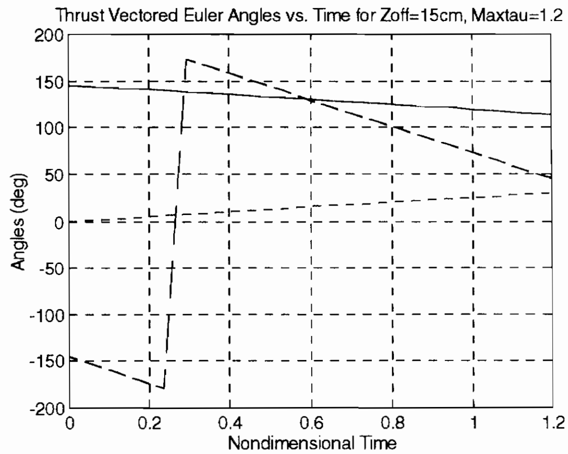


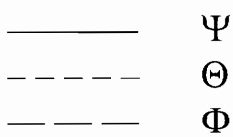
Figure 8. Nondimensional paths of the thrust vectored, time constrained cases a) in the orbit plane, and b) out of the orbit plane.

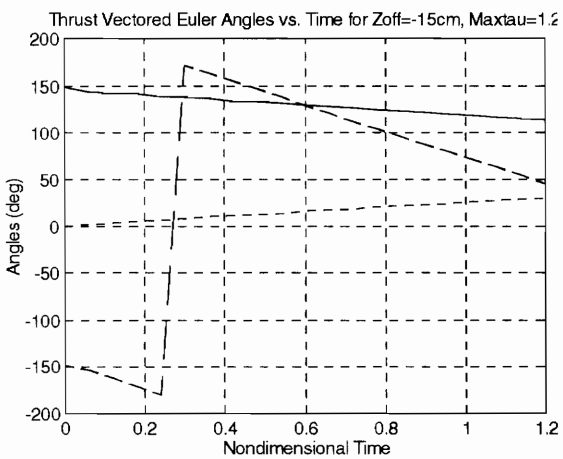


a)

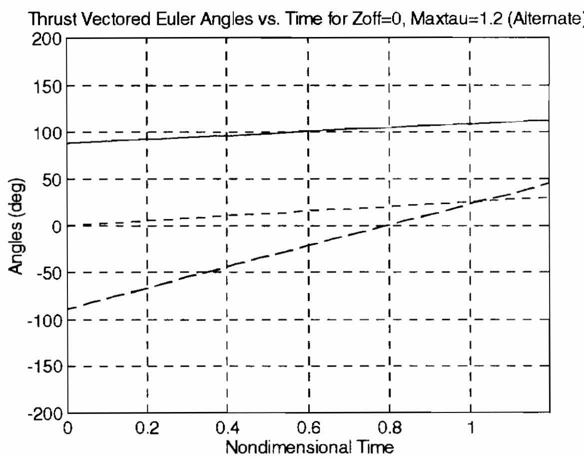


b)





c)



d)

Figure 9. Inertial to body axes Euler angles vs. nondimensional time for the time constrained (τ_{max} = 1.2) cases a) nominal CG location, b) Z_{off} = 15 cm, c) Z_{off} = -15 cm, and d) nominal CG location, alternate solution family.

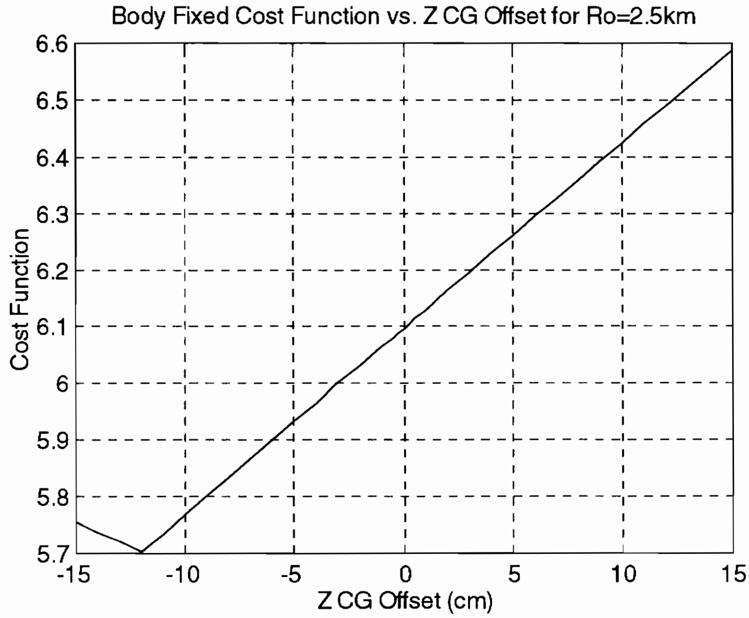


Figure 10. Cost function vs. Z_{off} for the body fixed thruster case.

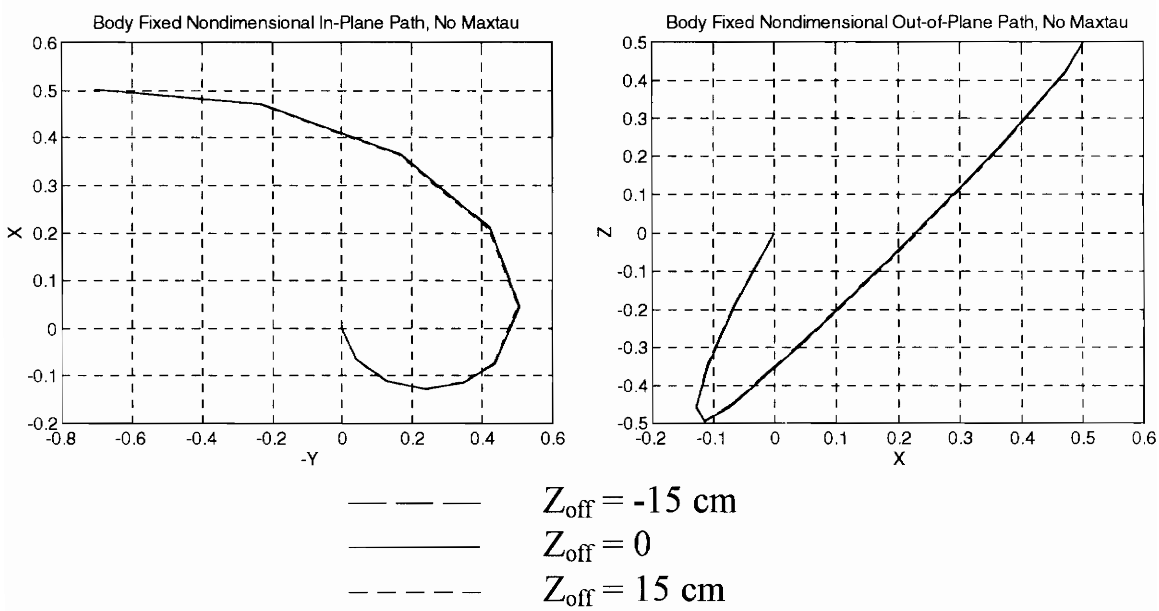
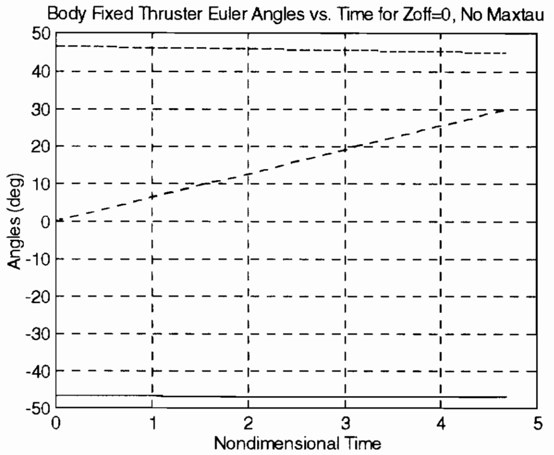
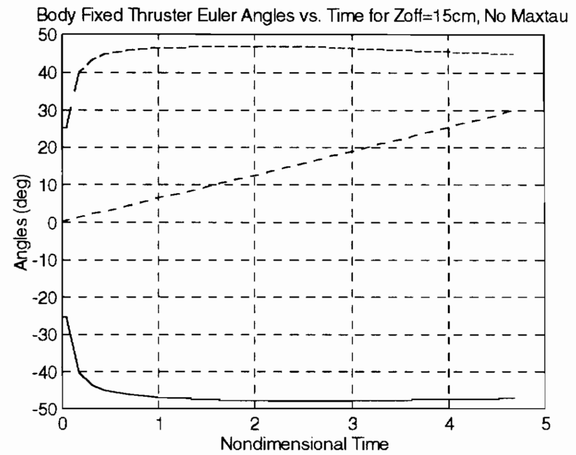


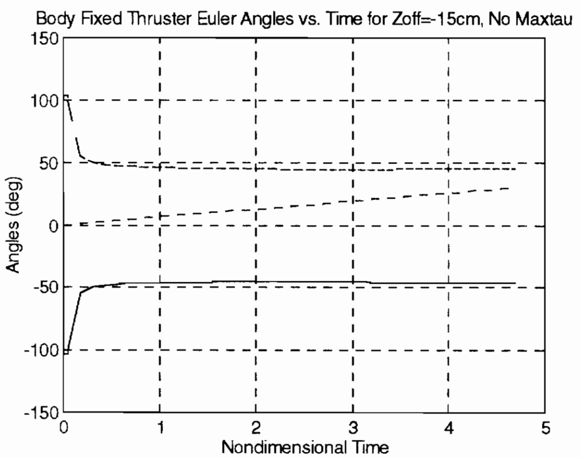
Figure 11. Nondimensional paths of the unrestricted time body fixed thruster cases a) in the orbit plane, and b) out of the orbit plane.



a)



b)



c)

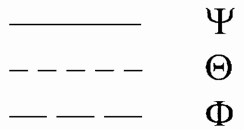
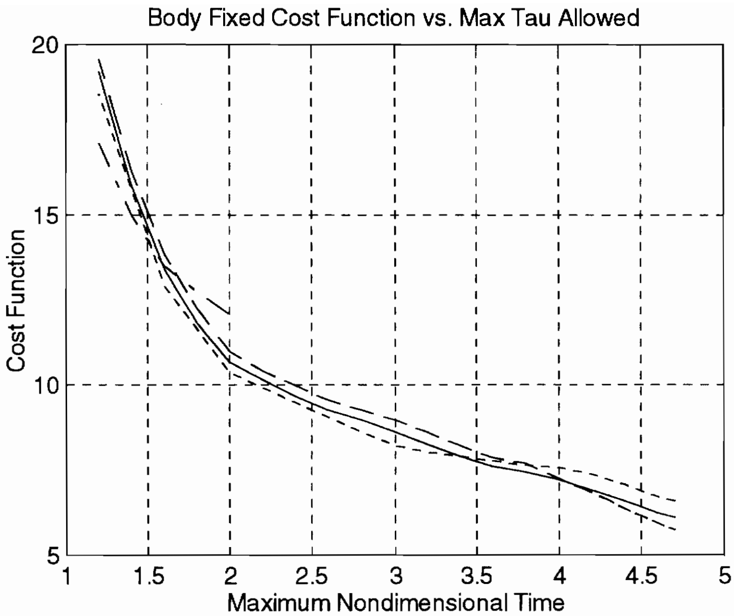
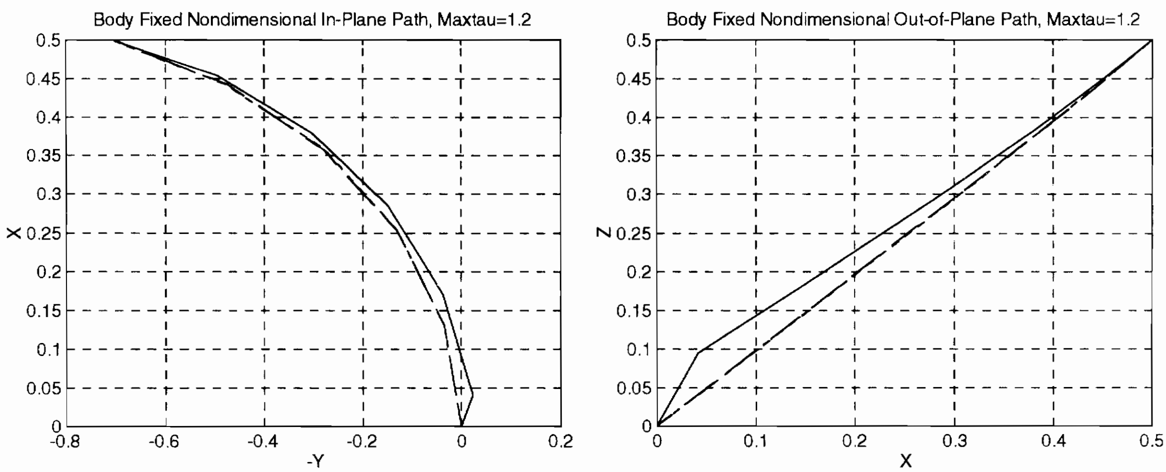


Figure 12. Inertial to body axes 3-1-3 Euler angles vs. nondimensional time for the time free body fixed thruster cases a) nominal CG, b) Z_{off} = 15 cm, and c) Z_{off} = -15 cm.



———— $Z_{off} = 0$
— — — $Z_{off} = 0$ (Alternate)
- - - - - $Z_{off} = 15$ cm
— — — $Z_{off} = -15$ cm

Figure 13. Cost function vs. maximum allowed rendezvous time for the body fixed thruster cases.



a)

— — — $Z_{off} = -15$ cm
———— $Z_{off} = 0$
- - - - - $Z_{off} = 15$ cm

b)

Figure 14. Nondimensional paths of the body fixed thruster, time constrained cases a) in the orbit plane, and b) out of the orbit plane.

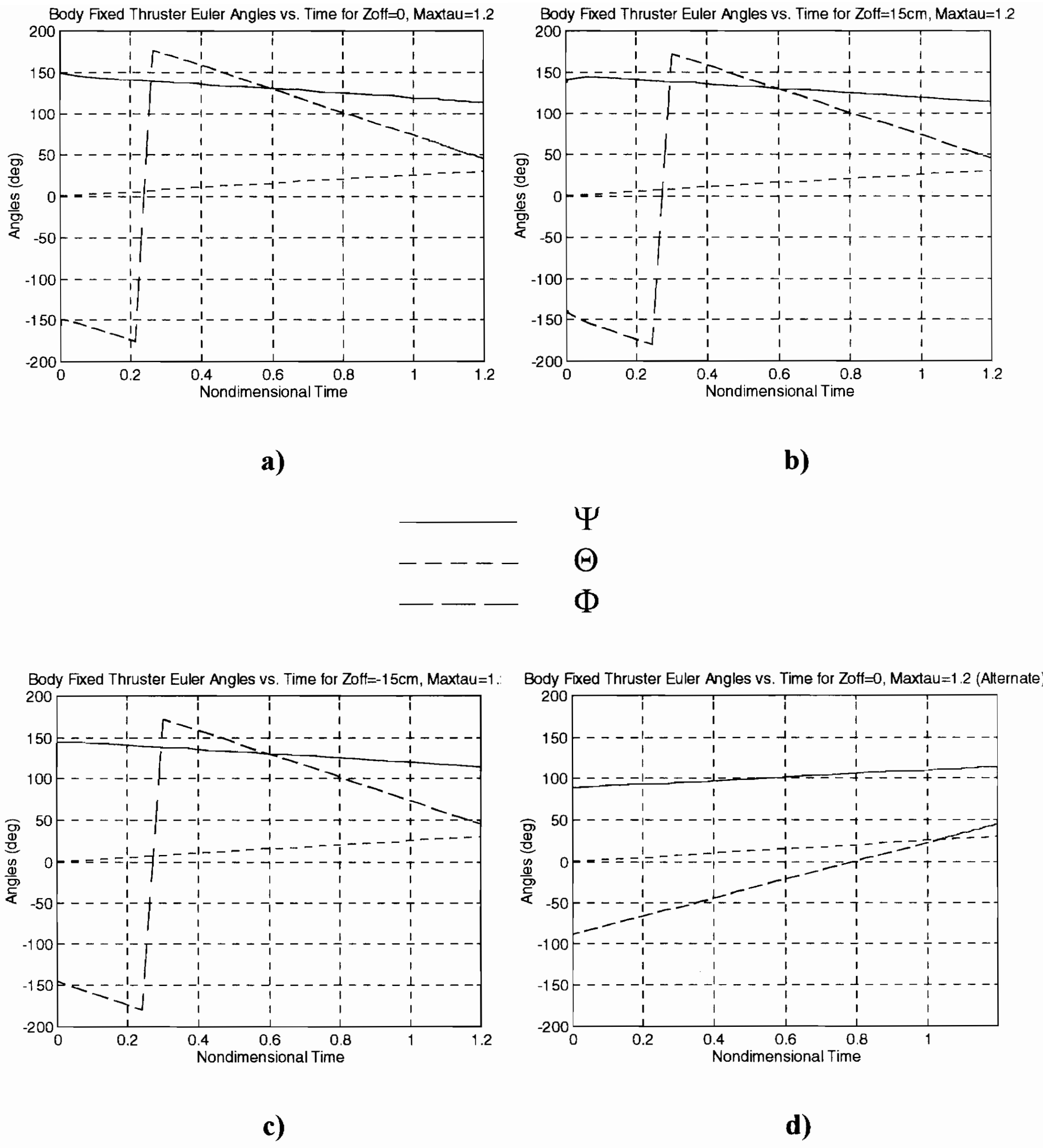


Figure 15. Inertial to body axes Euler angles vs. nondimensional time for the time constrained (τ_{max} = 1.2) body fixed thruster cases a) nominal CG location, b) Z_{off} = 15 cm, c) Z_{off} = -15 cm, and d) nominal CG location, alternate solution family.

APPENDIX: COMPUTER CODE

```
***********************************************************************
*   This program finds the minimum fuel trajectory between two different
*   positions and orientations for a satellite in orbit. This portion of
*   the code is meant to be used with routines SPQ.FOR, QLD.FOR, and
*   NLPQLD.FOR. This code was written by John M. Kruep for use with his
*   Master of Science thesis research at Virginia Tech.
***********************************************************************

      program homein
      implicit double precision (a-h,o-z)

      integer nmax
      parameter (nmax = 16)
      integer   iprint, ibtype, maxitn, maxsch
      integer   i, m, me, n
      real*8 x, xl, xu, xguess, xscale, f, eps,q(0:3),maxtau
      logical active(6),vtype
      common /CMACHE/ eps
      common /scale/ xscale
      common /conditons/ xs,ys,zs,rx,ry,rz,zoff,sifg,thfg,phfg,pi,dtor,
     c           vdim,iprn,epsdv,maxtau
      dimension x(nmax), xl(nmax), xu(nmax), xguess(nmax), xscale(nmax)
      dimension chg(3),r(3),v(3),rnew(3),vnew(3),gset(6),g(6)
      data iprint, ibtype  / 4, 0 /
      data maxitn, maxsch  / 1200, 35 /

      data n,m,me /16,6,6/
      data xl /-50.d0,-50.d0,-50.d0,1.01d-8,-50.d0,-50.d0,-50.d0,
     c   1.01d-8,-50.d0,-50.d0,-50.d0,1.01d-8,-50.d0,-50.d0,-50.d0,1.d-8/
      data xu /50.d0,50.d0,50.d0,7.d0,50.d0,50.d0,50.d0,
     c       7.d0,50.d0,50.d0,50.d0,7.d0,50.d0,50.d0,50.d0,6.d0/
      data xscale /16*1.d0/

      data xguess/  -.415D-01, -.268D+00,   .179D-02, .422D-01,
     c       -.377D-01,  -.905D+00, .740D-03,   .112D-06,
     c       -.332D-01,  -.475D-01,-.827D-02,   .270D+01,
     c        .298D-01,  -.610D-01, .150D-01,   .199D+01/
c      data xguess/   .689D-01, .590D+00,   .210D+00, .269D-06,
c     c -.178D+00,  -.101D+01,  -.194D+00,   .100D-06,
c     c   .413D+00,   .108D+01,   .259D+01,   .120D+01,
c     c   .168D+00,   .545D-01, .113D+00,   .561D-05/

      Open (6,file='five.dat',status='unknown')
      open (8,file='angles.dat')

      eps = 1.d-16
      epsdv=1.d-16
      xs=0.5d0
      ys=dsqrt(0.5d0)
      zs=0.5d0
      rx=1.d0
      ry=1.d0
      rz=1.d0
      zoff=-15.0d-2
c      maxtau=1.2d0
      pi=datan(1.d0)*4.d0
      dtor=pi/180.d0
      sifg=45.d0*dtor
      thfg=30.d0*dtor
      phfg=45.d0*dtor
      vdim=2.8922d0
c      vdim=34.706d0
      iprn=0
      psio=0.d0
      thetao=0.0d0
      phio=0.d0

C Use next line for translational guess
c      call guess(xguess)
C Next line is call to optimizer
      call sqp(m,me,n,xguess,ibtype,xl,xu,
     +       xscale,iprint,maxitn,maxsch,x,f)
c      do 60 i=1,16
c        x(i)=xguess(i)
c 60   continue
      iprn=1
      call fcn(m, me, n, x, active, f, g)
      do 14 i=1,13,4
      write(6,101) x(i),x(i+1),x(i+2),x(i+3)
 14   continue
 101  format(5x,1hc,4(d11.3,1h,))
c
c   To estimate gradient of function/contraints
c      iprn=1
c      open(7,file='gradient.dat')
c      do 60 i=1,16
c        x(i)=xguess(i)
c 60   continue
c      call fcn(m, me, n, x, active, fset, gset)
c      do 65 i=5,5
c        x(i)=xguess(i)+1.d-7
c        call fcn(m, me, n, x, active, f, g)
c        write(7,*) i,xguess(i)
c        write(7,*) x(i),f-fset,((g(j)-gset(j)),j=1,6)
c        x(i)=xguess(i)-1.d-7
c        call fcn(m, me, n, x, active, f, g)
c        write(7,*) x(i),f-fset,((g(j)-gset(j)),j=1,6)
c        x(i)=xguess(i)
c 65   continue
c   end gradient estimate

      end


      subroutine fcn(m, me, n, x, active, f, g)
      implicit double precision(a-h,o-z)
c
c     Routine for evaluating the cost and constraint functions
c     Body axis fixed thrusters
c
      integer m, me, n, i
      logical active,vtype(4),vtemp
      real*8  x,g,f,qf(0:3),qg(0:3)
      real*8 taus,x0p,y0p,z0p,xfp,yfp,zfp,taus2,xfp2,yfp2,zfp2
      real*8 pi,dtor,sif,sifg,phf,phfg,thf,thfg,tof,h,thetah,psih
      real*8 psio,thetao,phio,tof2,h2,thetah2,psih2,sif2,thf2,phf2,df
      real*8 xs,ys,zs,rx,ry,rz,xoff,vdim,maxtau
      real*8 th,ph,h01,dvx,dvy,dvz,dv1,dv2,dv3
      real*8 dxd,dyd,dzd,a1,a2,a3,b1,b2,b3,sifgm
      dimension   x(n), g(m), active(m),xo(0:16),r(3),v(3),rnew(3),
     c          vnew(3),chg(3)
      common /conditons/ xs,ys,zs,rx,ry,rz,zoff,sifg,thfg,phfg,pi,dtor,
     c            vdim,iprn,epsdv,maxtau
      common /curtime/ tau

      hmt=0.d0
C   Set up originial sequence of burns (att,div,att,div)
      vtype(1)=.false.
      vtype(2)=.true.
      vtype(3)=.false.
      vtype(4)=.true.
      xo(0)=0.d0
      do 3 i=1,16
            xo(i)=x(i)
 3    continue

c   Bubble Sort the burns to ensure proper time order
      do 5 i=1,3
       do 6 j=1,2
        if (xo(j*4).gt.xo((j+1)*4)) then
         do 7 k=1,4
          temp=xo(j*4+k-1)
           xo(j*4+k-1)=xo((j+1)*4+k-1)
           xo((j+1)*4+k-1)=temp
 7       continue
         vtemp=vtype(j+2)
         vtype(j+2)=vtype(j+1)
         vtype(j+1)=vtemp
        endif
 6     continue
 5    continue

C   Computes proper final rotation state
      if ((sifg+xo(16)+xo(12)).gt.pi) then
        sifgm=-2.d0*pi+sifg+xo(16)+xo(12)
      else
        sifgm=sifg+xo(16)+xo(12)
      endif
      call angtoparam(sifgm,thfg,phfg,qg)

C   Set up for trajectory propagations
      f=0.d0
      h=0.d0
      thetah=0.d0
      psih=0.d0
      r(1)=xs
      r(2)=ys
      r(3)=zs
      psi=0.d0
      theta=0.d0
      phi=0.d0
      tau=0.d0
      do 10 i=1,3
        v(i)=0.d0
```

```
 10   continue

c   Trajectory propagation loop
      do 20 i=1,4
        do 13 j=1,3
          chg(j)=xo(j+(i-1)*4)
 13     continue
        if (i.lt.4) then
          dt=xo(i*4)-xo((i-1)*4)
        else
          dt=xo(i*4)    ! Last time is a dtau
        endif

        if (iprn.eq.1) then
          write(6,*) 'Stage ',i,' Tau =',tau
          write(6,50) 'Angles',psi/dtor,theta/dtor,phi/dtor
        endif

        call event(f,dt,tau,h,thetah,psih,r,v,vtype(i),chg,
     c    psi,theta,phi,sif,thf,phf,hnew,thetahnew,psihnew,
     c    rnew,vnew,hmt)
        tau=tau+dt
        h=hnew
        thetah=thetahnew
        psih=psihnew
        psi=sif
        theta=thf
        phi=phf
        do 15 j=1,3
          r(j)=rnew(j)
          v(j)=vnew(j)
 15     continue
 20   continue
c   End trajectory propagation loop
 50   format(a6,3f10.5)

      if (iprn.eq.1) then
        write(6,*) 'End      Tau =',tau
        write(6,50) 'Angles',psi/dtor,theta/dtor,phi/dtor
        write(6,50) 'Goals ',sifgm/dtor,thfg/dtor,phfg/dtor
      endif

c   Compute final combined burn
      call tilh(xo(12)+xo(16),v(1),v(2),v(3),a1,a2,a3)
      call tbi(psi,theta,phi,a1,a2,a3,b1,b2,b3)
      f=f+dsqrt(b1**2+epsdv)*vdim+dsqrt(b2**2+epsdv)*vdim
     c    +dsqrt(b3**2+epsdv)*vdim
      dhy=-zoff*(b1)*vdim
      dhx=zoff*(b2)*vdim
      if (iprn.eq.1) write(6,111)'Body axis divert thrusters',b1,b2,b3
      call tib(psi,theta,phi,dhx,dhy,0.d0,b1,b2,b3)
      hx=h*dsin(thetah)*dsin(psih)+b1
      hy=-h*dsin(thetah)*dcos(psih)+b2
      hz=h*dcos(thetah)+b3
      call tbi(psi,theta,phi,hx,hy,hz,b1,b2,b3)
      if (iprn.eq.1) write(6,111)'Body axis moment thrusters',b1,b2,b3
      hmt=hmt+dabs(b1)+dabs(b2)+dabs(b3)
      f=f+dsqrt(b1**2+epsdv)+dsqrt(b2**2+epsdv)
     c    +dsqrt(b3**2+epsdv)
c   End final burn computations

      if (iprn.eq.1) write(6,*) '% of attitude burns ',hmt/f*100.d0
c        No Attitude Thruster distances

 111  format(a26,3f12.7)

c   Constraints
      call angtoparam(psi,theta,phi,qf)
      g(1)=(qf(0)-qg(0))**2+(qf(1)-qg(1))**2+(qf(2)-qg(2))**2
      g(2)=(qf(1)-qg(1))**2+(qf(2)-qg(2))**2+(qf(3)-qg(3))**2
      g(3)=(qf(2)-qg(2))**2+(qf(3)-qg(3))**2+(qf(0)-qg(0))**2
      g(4)=r(1)*2500.d0
      g(5)=r(2)*2500.d0
      g(6)=r(3)*2500.d0
ccc   g(7)=maxtau-tau

      return
      end

      subroutine guess(xguess)
c   Computes a guess for a two burn translational trajectory solution that
c   will statisfy the position constraints. Must set the 2nd divert burn
c   to zero in xguess to work properly.
      implicit double precision (a-h,o-z)
      dimension xguess(*),r(3),v(3),rnew(3),vnew(3),chg(3)
      logical vtype
      common /conditons/ xs,ys,zs,rx,ry,rz,zoff,sifg,thfg,phfg,pi,dtor,
     c        vdim

      vtype=.true.
      f=0.d0
      h=0.01d0
      thetah=0.1d0
      psih=0.01d0
      r(1)=xs
      r(2)=ys
      r(3)=zs
      psi=0.d0
      theta=0.d0
      phi=0.d0
      tau=0.d0
      do 10 i=1,3
        v(i)=0.d0
        chg(i)=0.d0
 10   continue
      call event(f,xguess(4),tau,h,thetah,psih,r,v,vtype,chg,
     c    psio,thetao,phio,sif,thf,phf,hnew,thetahnew,psihnew,
     c    rnew,vnew,hmt)
       do 15 i=1,3
        r(i)=rnew(i)
        v(i)=vnew(i)
        chg(i)=xguess(4+i)
 15   continue

      taus=xguess(16)+xguess(12)-xguess(4)  !last time is dt
      call intercept2(rnew(1),rnew(2),rnew(3),0.d0,0.d0,0.d0,taus,
     c    x0p,y0p,z0p,xfp,yfp,zfp)
      xguess(5)=x0p-vnew(1)
      xguess(6)=y0p-vnew(2)
      xguess(7)=z0p-vnew(3)
       do 18 i=1,3
        chg(i)=xguess(4+i)
 18   continue
      call event(f,xguess(16)+xguess(12)-xguess(4),tau,h,thetah,psih,
     c r,v,vtype,chg,psio,thetao,phio,sif,thf,phf,
     c hnew,thetahnew,psihnew,rnew,vnew,hmt)
      write(*,*) rnew,xguess(5),xguess(6),xguess(7)
      return
      end


      subroutine event(f,dt,tau,h,thetah,psih,r,v,vtype,chg,
     c    psio,thetao,phio,sif,thf,phf,hnew,thetahnew,psihnew,
     c    rnew,vnew,hmt)
c   Computes changes caused by a burn (depending on its type).
c   Propagates angular and translational trajectories between burns.
c   Variable array chg is either Dh in inertial components
c   or Dv in CW components
      implicit double precision(a-h,o-z)
      logical vtype
      dimension chg(3),r(3),v(3),rnew(3),vnew(3)
      common /conditons/ xs,ys,zs,rx,ry,rz,zoff,sifg,thfg,phfg,pi,dtor,
     c    vdim,iprn,epsdv

c   For divert burn
      if (vtype)  then
        vx=v(1)+chg(1)
        vy=v(2)+chg(2)
        vz=v(3)+chg(3)
        call tilh(tau,chg(1),chg(2),chg(3),a1,a2,a3)
        call tbi(psio,thetao,phio,a1,a2,a3,b1,b2,b3)
        f=f+dsqrt(b1**2+epsdv)*vdim+dsqrt(b2**2+epsdv)*vdim
     c    +dsqrt(b3**2+epsdv)*vdim
        dhy=-zoff*(b1)*vdim
        dhx=zoff*(b2)*vdim
        if (iprn.eq.1) write(6,111)'Body axis divert thrusters',b1,b2,b3
        call tib(psio,thetao,phio,dhx,dhy,0.d0,b1,b2,b3)
        hx=h*dsin(thetah)*dsin(psih)+b1
        hy=-h*dsin(thetah)*dcos(psih)+b2
        hz=h*dcos(thetah)+b3
        hnew=dsqrt(hx**2+hy**2+hz**2)
        if (hnew.lt.0.d0) hnew=1.d-14
        thetahnew=dacos(hz/hnew)
        psihnew=datan2(hx,-hy)
        call propagate(dt,vx,vy,vz,r(1),r(2),r(3),
     c        vnew(1),vnew(2),vnew(3),rnew(1),rnew(2),rnew(3),iprn)
      else
c   For attitude burn
c   Still need to work out attitude thruster distance thing
        call tbi(psio,thetao,phio,chg(1),chg(2),chg(3),b1,b2,b3)
        f=f+dsqrt(b1**2+epsdv)+dsqrt(b2**2+epsdv)
     c    +dsqrt(b3**2+epsdv)
        if (iprn.eq.1) write(6,111)'Body axis moment thrusters',b1,b2,b3
        hmt=hmt+dabs(b1)+dabs(b2)+dabs(b3)
        hx=h*dsin(thetah)*dsin(psih)+chg(1)
        hy=-h*dsin(thetah)*dcos(psih)+chg(2)
        hz=h*dcos(thetah)+chg(3)
        hnew=dsqrt(hx**2+hy**2+hz**2)
        if (hnew.lt.0.d0) hnew=1.d-14
        thetahnew=dacos(hz/hnew)
        psihnew=datan2(hx,-hy)
        call propagate(dt,v(1),v(2),v(3),r(1),r(2),r(3),
     c        vnew(1),vnew(2),vnew(3),rnew(1),rnew(2),rnew(3),iprn)
      endif

c   For both burn types
      if (iprn.ne.1) then
       call rotate(dt/1.156872597d-03,hnew,thetahnew,psihnew,
     c     psio,thetao,phio,sif,thf,phf)
      else
       call rotatep(dt/1.156872597d-03,hnew,thetahnew,psihnew,
     c     psio,thetao,phio,sif,thf,phf)
      endif

 111  format(a26,3f12.7)

      return
      end


      subroutine propagate(taus,x0p,y0p,z0p,x0,y0,z0,xp,yp,zp,x,y,z,
     c iprn)
c   Propagtes the nondimensional CW (translational) equations over nondim
c   time taus. Takes initial nondim velocity (x0p, etc) and position
```

```
c   (x0, etc) and returns final velocity and position.
      implicit double precision(a-h,o-z)

c   To print out intermediate positions
      if (iprn.eq.1) then
      do 10 i=0,5
        tau=dfloat(i)/5.d0*taus
        xp=x0p*dcos(tau)+(3.d0*x0+2.d0*y0p)*dsin(tau)
        yp=y0p-(6.d0*x0+4.d0*y0p)*(1.d0-dcos(tau))-2.d0*x0p*dsin(tau)
        zp=-z0*dsin(tau)+z0p*dcos(tau)
        x=x0+(3.d0*x0+2.d0*y0p)*(1.d0-dcos(tau))+x0p*dsin(tau)
        y=y0-(6.d0*x0+3.d0*y0p)*tau+(6.d0*x0+4.d0*y0p)*dsin(tau)-
     c    2.d0*x0p*(1.d0-dcos(tau))
        z=z0*dcos(tau)+z0p*dsin(tau)
        write(6,15) xp,yp,zp,x,y,z
 10   continue
 15   format(6(1x,f11.8))
      endif

c   Actual trajectory propagation
      xp=x0p*dcos(taus)+(3.d0*x0+2.d0*y0p)*dsin(taus)
      yp=y0p-(6.d0*x0+4.d0*y0p)*(1.d0-dcos(taus))-2.d0*x0p*dsin(taus)
      zp=-z0*dsin(taus)+z0p*dcos(taus)
      x=x0+(3.d0*x0+2.d0*y0p)*(1.d0-dcos(taus))+x0p*dsin(taus)
      y=y0-(6.d0*x0+3.d0*y0p)*taus+(6.d0*x0+4.d0*y0p)*dsin(taus)-
     c          2.d0*x0p*(1.d0-dcos(taus))
      z=z0*dcos(taus)+z0p*dsin(taus)

      return
      end

      subroutine interceptw(x0,y0,z0,x1,y1,z1,taus,x0p,y0p,z0p,
     c          xfp,yfp,zfp)

      implicit double precision (a-h,o-z)

      denom=8.d0*(1.d0-dcos(taus))-3.d0*taus*dsin(taus)
      x0p=(x0*(3.d0*taus*dcos(taus)-4.d0*dsin(taus))+
     c   2.d0*(y0-y1)*(1.d0-dcos(taus))+x1*(4.d0*dsin(taus)-3.d0*taus))
     c   /denom
      y0p=(2.d0*x0*(7.d0*(dcos(taus)-1.d0)+3.d0*taus*dsin(taus))-
     c       (y0-y1)*dsin(taus)+2.d0*x1*(1.d0-dcos(taus)))/denom
      xfp=x0p*dcos(taus)+(3.d0*x0+2.d0*y0p)*dsin(taus)
      yfp=y0p-(6.d0*x0+4.d0*y0p)*(1.d0-dcos(taus))-2.d0*x0p*dsin(taus)
      z0p=(z1-z0*dcos(taus))/dsin(taus)
      zfp=-z0*dsin(taus)+z0p*dcos(taus)

      do 10 i=0,5
        tau=dfloat(i)/5.d0*taus
        xp=x0p*dcos(tau)+(3.d0*x0+2.d0*y0p)*dsin(tau)
        yp=y0p-(6.d0*x0+4.d0*y0p)*(1.d0-dcos(tau))-2.d0*x0p*dsin(tau)
        zp=-z0*dsin(tau)+z0p*dcos(tau)
        x=x0+(3.d0*x0+2.d0*y0p)*(1.d0-dcos(tau))+x0p*dsin(tau)
        y=y0-(6.d0*x0+3.d0*y0p)*tau+(6.d0*x0+4.d0*y0p)*dsin(tau)-
     c    2.d0*x0p*(1.d0-dcos(tau))
        z=z0*dcos(tau)+z0p*dsin(tau)
        write(*,15) xp,yp,zp,x,y,z
 10   continue
 15   format(6(1x,f9.5))
      end

      subroutine intercept2(x0,y0,z0,x1,y1,z1,taus,x0p,y0p,z0p,
     c    xfp,yfp,zfp)
c   Intercepts CW coordinates x1,y1,z1 in nondim time taus using one
c   translational trajectory arc.
      implicit double precision (a-h,o-z)

      denom=8.d0*(1.d0-dcos(taus))-3.d0*taus*dsin(taus)
      if (dabs(denom).lt.1.d-17) then
        if (denom.lt.0.d0) denom=-1.d-17
        if (denom.ge.0.d0) denom=1.d-17
      endif
      x0p=(x0*(3.d0*taus*dcos(taus)-4.d0*dsin(taus))+
     c   2.d0*(y0-y1)*(1.d0-dcos(taus))+x1*(4.d0*dsin(taus)-3.d0*taus))
     c   /denom
      y0p=(2.d0*x0*(7.d0*(dcos(taus)-1.d0)+3.d0*taus*dsin(taus))-
     c       (y0-y1)*dsin(taus)+2.d0*x1*(1.d0-dcos(taus)))/denom
      xfp=x0p*dcos(taus)+(3.d0*x0+2.d0*y0p)*dsin(taus)
      yfp=y0p-(6.d0*x0+4.d0*y0p)*(1.d0-dcos(taus))-2.d0*x0p*dsin(taus)
      if (dsin(taus).eq.0.d0) then
       z0p=(z1-z0*dcos(taus))/1.d-17
       else
       z0p=(z1-z0*dcos(taus))/dsin(taus)
      endif
      zfp=-z0*dsin(taus)+z0p*dcos(taus)

      do 10 i=0,5
        tau=dfloat(i)/5.d0*taus
        xp=x0p*dcos(tau)+(3.d0*x0+2.d0*y0p)*dsin(tau)
        yp=y0p-(6.d0*x0+4.d0*y0p)*(1.d0-dcos(tau))-2.d0*x0p*dsin(tau)
        zp=-z0*dsin(tau)+z0p*dcos(tau)
        x=x0+(3.d0*x0+2.d0*y0p)*(1.d0-dcos(tau))+x0p*dsin(tau)
        y=y0-(6.d0*x0+3.d0*y0p)*tau+(6.d0*x0+4.d0*y0p)*dsin(tau)-
     c    2.d0*x0p*(1.d0-dcos(tau))
        z=z0*dcos(tau)+z0p*dsin(tau)
c       write(6,15) xp,yp,zp,x,y,z
 10   continue
 15   format(6(1x,f8.5))
      end


      subroutine rotate(tof,h,thetah,psih,psio,thetao,phio,sif,thf,phf)
```

```
c   Computes the rotational changes that occur over tof due to ang.
c   momentum vector (h,thetah,psih). Specifies input/output 3-1-3 Euler
c   angles from inertial to body coordinate systems. See Cochrane et al.
      implicit double precision(a-h,o-z)

      call orient(psih,thetah,psio,thetao,phio,psi0,theta0,phi0)
      ix=3376.d0
      iz=1057.d0

      psif=psi0+h/ix*tof
      thetaf=theta0
      phif=phi0+(1.d0/iz-1.d0/ix)*h*dcos(theta0)*tof

      cphif=dcos(phif)
      sphif=dsin(phif)
      cpsif=dcos(psif)
      spsif=dsin(psif)
      cthef=dcos(thetaf)
      sthef=dsin(thetaf)
      cpsih=dcos(psih)
      spsih=dsin(psih)
      ctheh=dcos(thetah)
      stheh=dsin(thetah)

      if (stheh.eq.0.d0) stheh=1.d-16

      c13f=(cphif*spsif+sphif*cpsif*cthef)*stheh+ctheh*sthef*sphif
      c23f=(-sphif*spsif+cphif*cpsif*cthef)*stheh+ctheh*sthef*cphif
      c33f=-sthef*cpsif*stheh+cthef*ctheh
      c31f=sthef*(spsif*cpsih+cpsif*ctheh*spsih)+cthef*spsih*stheh
      c32f=sthef*(spsif*spsih-cpsif*ctheh*cpsih)-cthef*cpsih*stheh

      sif=datan2(c31f,-c32f)
c      thf=datan2(dsqrt(c13f**2+c23f**2),c33f)
      if (c33f.gt.1.d0) c33f=1.d0
      thf=dacos(c33f)
      phf=datan2(c13f,c23f)

      return

      end

      subroutine rotatep(tof,h,thetah,psih,psio,thetao,phio,sif,thf,phf)
      implicit double precision(a-h,o-z)
c   Same as rotate, but prints out intermediate values
      common /conditons/ xs,ys,zs,rx,ry,rz,zoff,sifg,thfg,phfg,pi,dtor,
     c    vdim,iprn,epsdv,maxtau
      common /curtime/ time

      call orient(psih,thetah,psio,thetao,phio,psi0,theta0,phi0)
      ix=3376.d0
      iz=1057.d0

      if (iprn.eq.1) then
        if (tof.gt.1.d-1) then

      do 25 i=0,20
        if (i.ne.0) then
         psif=psi0+h/ix*tof*dfloat(i)/20.d0
         thetaf=theta0
         phif=phi0+(1.d0/iz-1.d0/ix)*h*dcos(theta0)*tof*dfloat(i)/20.d0
        else
         psif=psi0+h/ix*tof*0.001d0
         thetaf=theta0
         phif=phi0+(1.d0/iz-1.d0/ix)*h*dcos(theta0)*tof*0.001
        endif

        cphif=dcos(phif)
        sphif=dsin(phif)
        cpsif=dcos(psif)
        spsif=dsin(psif)
        cthef=dcos(thetaf)
        sthef=dsin(thetaf)
        cpsih=dcos(psih)
        spsih=dsin(psih)
        ctheh=dcos(thetah)
        stheh=dsin(thetah)

        if (stheh.eq.0.d0) stheh=1.d-16

        c13f=(cphif*spsif+sphif*cpsif*cthef)*stheh+ctheh*sthef*sphif
        c23f=(-sphif*spsif+cphif*cpsif*cthef)*stheh+ctheh*sthef*cphif
        c33f=-sthef*cpsif*stheh+cthef*ctheh
        c31f=sthef*(spsif*cpsih+cpsif*ctheh*spsih)+cthef*spsih*stheh
        c32f=sthef*(spsif*spsih-cpsif*ctheh*cpsih)-cthef*cpsih*stheh

        sif=datan2(c31f,-c32f)
        if (c33f.gt.1.d0) c33f=1.d0
        thf=dacos(c33f)
        phf=datan2(c13f,c23f)
        write(8,101) time+tof*dfloat(i)/20.d0*1.156872597d-03,
     c   sif/dtor,thf/dtor,phf/dtor
 25   continue
      endif
      endif

      psif=psi0+h/ix*tof
      thetaf=theta0
      phif=phi0+(1.d0/iz-1.d0/ix)*h*dcos(theta0)*tof

      cphif=dcos(phif)
```

```
      sphif=dsin(phif)
      cpsif=dcos(psif)
      spsif=dsin(psif)
      cthef=dcos(thetaf)
      sthef=dsin(thetaf)
      cpsih=dcos(psih)
      spsih=dsin(psih)
      ctheh=dcos(thetah)
      stheh=dsin(thetah)

      if (stheh.eq.0.d0) stheh=1.d-16

      c13f=(cphif*spsif+sphif*cpsif*cthef)*stheh+ctheh*sthef*sphif
      c23f=(-sphif*spsif+cphif*cpsif*cthef)*stheh+ctheh*sthef*cphif
      c33f=-sthef*cpsif*stheh+cthef*ctheh
      c31f=sthef*(spsif*cpsih+cpsif*ctheh*spsih)+cthef*spsih*stheh
      c32f=sthef*(spsif*spsih-cpsif*ctheh*cpsih)-cthef*cpsih*stheh

      sif=datan2(c31f,-c32f)
      if (c33f.gt.1.d0) c33f=1.d0
      thf=dacos(c33f)
      phf=datan2(c13f,c23f)

 101  format(f11.7,3f11.5)
      return

      end


      subroutine orient(psih,thetah,psio,thetao,phio,psi0,theta0,phi0)
      implicit double precision(a-h,o-z)
C
C   Converts inertial to body axis Euler angles to ang mom to body axis
Euler
C   angles.

      cpsih=dcos(psih)
      spsih=dsin(psih)
      cthetah=dcos(thetah)
      sthetah=dsin(thetah)
      cpsio=dcos(psio)
      spsio=dsin(psio)
      cthetao=dcos(thetao)
      sthetao=dsin(thetao)
      cphio=dcos(phio)
      sphio=dsin(phio)

      if (sthetah.eq.0.d0) sthetah=1.d-16

      c13=spsih*sthetah*(cphio*cpsio-sphio*spsio*cthetao)-sthetah*cpsih*
     c     (cphio*spsio+sphio*cpsio*cthetao)+cthetah*sphio*sthetao
      c23=spsih*sthetah*(-sphio*cpsio-cphio*spsio*cthetao)-sthetah*
     c      cpsih*(-sphio*spsio+cphio*cpsio*cthetao)+cthetah*cphio*sthetao
      c33=spsih*sthetah*spsio*sthetao+cpsih*sthetah*sthetao*cpsio+
     c     cthetao*cthetah
      c31=cpsih*spsio*sthetao-spsih*sthetao*cpsio
      c32=-spsih*cthetah*spsio*sthetao-cpsih*cthetah*sthetao*cpsio+
     c     cthetao*sthetah

      psi0=datan2(c31,-c32)
c      theta0=datan2(dsqrt(c13**2+c23**2),c33)
      if (c33.gt.1.d0) c33=1.d0
      theta0=dacos(c33)
      phi0=datan2(c13,c23)

      return
      end


      subroutine angtoparam(psi,theta,phi,q)
c     Changes from 3-1-3 Euler angles to Euler parameters
      implicit double precision(a-h,o-z)
      dimension q(0:3)

      q(0)=dcos(theta/2.d0)*dcos((psi+phi)/2.d0)
      q(1)=dsin(theta/2.d0)*dcos((psi-phi)/2.d0)
      q(2)=dsin(theta/2.d0)*dsin((psi-phi)/2.d0)
      q(3)=dcos(theta/2.d0)*dsin((psi+phi)/2.d0)

      return
      end


      subroutine tbi(psi,the,phi,x1,x2,x3,y1,y2,y3)
c     Changes from inertial to body coords
      implicit double precision(a-h,o-z)

      cthe=dcos(the)
      sthe=dsin(the)
      cpsi=dcos(psi)
      spsi=dsin(psi)
      cphi=dcos(phi)
      sphi=dsin(phi)
      y1=x1*(cphi*cpsi-sphi*spsi*cthe)+x2*(cphi*spsi+sphi*cpsi*cthe)
     c    +x3*(sphi*sthe)
      y2=x1*(-sphi*cpsi-cphi*spsi*cthe)+x2*(-sphi*spsi+cphi*cpsi*cthe)
     c    +x3*(cphi*sthe)
      y3=x1*(spsi*sthe)-x2*(sthe*cpsi)+x3*cthe

      return
      end
```

```
      subroutine tib(psi,the,phi,x1,x2,x3,y1,y2,y3)
c     Changes from body to inertial coords
      implicit double precision(a-h,o-z)

      cthe=dcos(the)
      sthe=dsin(the)
      cpsi=dcos(psi)
      spsi=dsin(psi)
      cphi=dcos(phi)
      sphi=dsin(phi)
      y1=x1*(cphi*cpsi-sphi*spsi*cthe)+x2*(-sphi*cpsi-cphi*spsi*cthe)
     c    +x3*(spsi*sthe)
      y2=x1*(cphi*spsi+sphi*cpsi*cthe)+x2*(-sphi*spsi+cphi*cpsi*cthe)
     c    +x3*(-cpsi*sthe)
      y3=x1*(sphi*sthe)+x2*(sthe*cphi)+x3*cthe

      return
      end


      subroutine tilh(t,x1,x2,x3,y1,y2,y3)
c     Changes from local horiz (CW) to inertial coords
      implicit double precision(a-h,o-z)

      y1=x1*dcos(t)-x2*dsin(t)
      y2=x1*dsin(t)+x2*dcos(t)
      y3=x3

      return
      end



****************************************************************************
*   The following two routines replace routines "fcn" and "event" in
*   the above code when the satellite is using thrust vectored
*   thrusters.
****************************************************************************

      subroutine fcn(m, me, n, x, active, f, g)
      implicit double precision(a-h,o-z)
c
c     routine for evaluating the cost and constraint functions
c     thrust vectoring thrusters
c
      integer m, me, n, i
      logical active,vtype(4),vtemp
      real*8  x,g,f,qf(0:3),qg(0:3)
      real*8 taus,x0p,y0p,z0p,xfp,yfp,zfp,taus2,xfp2,yfp2,zfp2
      real*8 pi,dtor,sif,sifg,phf,phfg,thf,thfg,tof,h,thetah,psih
      real*8 psio,thetao,phio,tof2,h2,thetah2,psih2,sif2,thf2,phf2,df
      real*8 xs,ys,zs,rx,ry,rz,xoff,vdim,maxtau
      real*8 th,ph,h01,dvx,dvy,dvz,dv1,dv2,dv3
      real*8 dxd,dyd,dzd,a1,a2,a3,b1,b2,b3,sifgm
      dimension   x(n), g(m), active(m),xo(0:16),r(3),v(3),rnew(3),
     c         vnew(3),chg(3)
      common /conditons/ xs,ys,zs,rx,ry,rz,zoff,sifg,thfg,phfg,pi,dtor,
     c            vdim,iprn,epsdv,maxtau
      common /curtime/ tau

      hmt=0.d0
      vtype(1)=.false.
      vtype(2)=.true.
      vtype(3)=.false.
      vtype(4)=.true.
      xo(0)=0.d0
      do 3 i=1,16
        xo(i)=x(i)
 3    continue

c   Bubble Sort
      do 5 i=1,3
        do 6 j=1,2
          if (xo(j*4).gt.xo((j+1)*4)) then
            do 7 k=1,4
              temp=xo(j*4+k-1)
              xo(j*4+k-1)=xo((j+1)*4+k-1)
              xo((j+1)*4+k-1)=temp
 7            continue
            vtemp=vtype(j+2)
            vtype(j+2)=vtype(j+1)
            vtype(j+1)=vtemp
          endif
 6        continue
 5    continue

      if ((sifg+xo(16)+xo(12)).gt.pi) then
        sifgm=-2.d0*pi+sifg+xo(16)+xo(12)
      else
        sifgm=sifg+xo(16)+xo(12)
      endif
      call angtoparam(sifgm,thfg,phfg,qg)
c      write(*,*) xo,vtype
c      if (xo(12).gt.xo(16)) write(*,*) 'Violation'

      f=0.d0
```

```fortran
      h=0.d0
      thetah=0.d0
      psih=0.d0
      r(1)=xs
      r(2)=ys
      r(3)=zs
      psi=0.d0
      theta=0.d0
      phi=0.d0
      tau=0.d0
      do 10 i=1,3
        v(i)=0.d0
10    continue
      do 20 i=1,4
        do 13 j=1,3
          chg(j)=xo(j+(i-1)*4)
13      continue
        if (i.lt.4) then
          dt=xo(i*4)-xo((i-1)*4)
        else
          dt=xo(i*4)          ! Last time is a dtau
        endif
        if (iprn.eq.1) then
          write(6,*) 'Stage ',i,' Tau =',tau
          write(6,50) 'Angles',psi/dtor,theta/dtor,phi/dtor
        endif
        call event(f,dt,tau,h,thetah,psih,r,v,vtype(i),chg,
     c    psi,theta,phi,sif,thf,phf,hnew,thetahnew,psihnew,
     c    rnew,vnew,hmt)
        tau=tau+dt
        h=hnew
        thetah=thetahnew
        psih=psihnew
        psi=sif
        theta=thf
        phi=phf
        do 15 j=1,3
          r(j)=rnew(j)
          v(j)=vnew(j)
15      continue
20    continue
50    format(a6,3f10.5)
      if (iprn.eq.1) then
        write(6,*) 'End      Tau =',tau
        write(6,50) 'Angles',psi/dtor,theta/dtor,phi/dtor
        write(6,50) 'Goals ',sifgm/dtor,thfg/dtor,phfg/dtor
      endif
      call tilh(xo(12)+xo(16),v(1),v(2),v(3),a1,a2,a3)
      call tbi(psi,theta,phi,a1,a2,a3,b1,b2,b3)
      dhy=-zoff*(b1)*vdim
      dhx=zoff*(b2)*vdim
      call tib(psio,thetao,phio,dhx,dhy,0.d0,b1,b2,b3)
      hx=h*dsin(thetah)*dsin(psih)+b1
      hy=-h*dsin(thetah)*dcos(psih)+b2
      hz=h*dcos(thetah)+b3
      htot=dsqrt(hx**2+hy**2+hz**2+epsdv)! Thruster distances missing
      f=f+htot+dsqrt(v(1)**2+v(2)**2+v(3)**2+epsdv)*vdim
      hmt=hmt+dabs(htot)
      if (iprn.eq.1) write(6,*) '% of attitude cost ',hmt/f*100.d0

      call angtoparam(psi,theta,phi,qf)
      g(1)=(qf(0)-qg(0))**2+(qf(1)-qg(1))**2+(qf(2)-qg(2))**2
      g(2)=(qf(1)-qg(1))**2+(qf(2)-qg(2))**2+(qf(3)-qg(3))**2
      g(3)=(qf(2)-qg(2))**2+(qf(3)-qg(3))**2+(qf(0)-qg(0))**2
      g(4)=r(1)*2500.d0
      g(5)=r(2)*2500.d0
      g(6)=r(3)*2500.d0
ccc          g(7)=maxtau-tau

      return
      end


      subroutine event(f,dt,tau,h,thetah,psih,r,v,vtype,chg,
     c    psio,thetao,phio,sif,thf,phf,hnew,thetahnew,psihnew,
     c    rnew,vnew,hmt)
c   Computes changes caused by a burn (depending on its type)
c   Variable array chg is either Dh in inertial components
c   or Dv in CW components
      implicit double precision(a-h,o-z)
      logical vtype
      dimension chg(3),r(3),v(3),rnew(3),vnew(3)
      common /conditons/ xs,ys,zs,rx,ry,rz,zoff,sifg,thfg,phfg,pi,dtor,
     c    vdim,iprn,epsdv

      if (vtype)  then
        f=f+dsqrt(chg(1)**2+chg(2)**2+chg(3)**2+epsdv)*vdim
        vx=v(1)+chg(1)
        vy=v(2)+chg(2)
        vz=v(3)+chg(3)
        call tilh(tau,chg(1),chg(2),chg(3),a1,a2,a3)
        call tbi(psio,thetao,phio,a1,a2,a3,b1,b2,b3)
        dhy=-zoff*(b1)*vdim
        dhx=zoff*(b2)*vdim
        call tib(psio,thetao,phio,dhx,dhy,0.d0,b1,b2,b3)
        hx=h*dsin(thetah)*dsin(psih)+b1
        hy=-h*dsin(thetah)*dcos(psih)+b2
        hz=h*dcos(thetah)+b3
        hnew=dsqrt(hx**2+hy**2+hz**2)
        if (hnew.lt.0.d0) hnew=1.d-14
        thetahnew=dacos(hz/hnew)
        psihnew=datan2(hx,-hy)
        call propagate(dt,vx,vy,vz,r(1),r(2),r(3),
     c     vnew(1),vnew(2),vnew(3),rnew(1),rnew(2),rnew(3),iprn)
      else
c       f=f+dsqrt((chg(1)/rx)**2+(chg(2)/ry)**2+(chg(3)/rz)**2)
c   Still need to work out thruster distance thing
        f=f+dsqrt((chg(1))**2+(chg(2))**2+(chg(3))**2+epsdv)
        hx=h*dsin(thetah)*dsin(psih)+chg(1)
        hy=-h*dsin(thetah)*dcos(psih)+chg(2)
        hz=h*dcos(thetah)+chg(3)
        hnew=dsqrt(hx**2+hy**2+hz**2)
        if (hnew.lt.0.d0) hnew=1.d-14
        thetahnew=dacos(hz/hnew)
        psihnew=datan2(hx,-hy)
        call propagate(dt,v(1),v(2),v(3),r(1),r(2),r(3),
     c    vnew(1),vnew(2),vnew(3),rnew(1),rnew(2),rnew(3),iprn)
        hmt=hmt+dsqrt((chg(1))**2+(chg(2))**2+(chg(3))**2+epsdv)
      endif
if (iprn.ne.1) then
        call rotate(dt/1.156872597d-03,hnew,thetahnew,psihnew,
     c  psio,thetao,phio,sif,thf,phf)
      else
        call rotatep(dt/1.156872597d-03,hnew,thetahnew,psihnew,
     c  psio,thetao,phio,sif,thf,phf)
      endif

      return
      end
```

VITA

The author (John M. Kruep) was born in Torrance, California on April 19, 1973. He received a Bachelor of Science degree in Aerospace Engineering from Virginia Polytechnic Institute and State University in May of 1995. In addition to his university studies, the author spent the summers of 1994 and 1995 working in the Vehicle Analysis Branch at NASA Langley Research Center in Hampton, Virginia. After completing his Master of Science degree in May 1996, the author began work at the Lockheed Martin Corporation in Denver, Colorado.

This work was done to partially fulfill the requirements of a Master of Science degree in Aerospace Engineering from Virginia Polytechnic Institute and State University.

John M. Kruep