REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	21.Jul.99	THESIS

4. TITLE AND SUBTITLE
GROUND BASED INTERCEPT OF A BALLISTIC MISSILE: BATTLE MANAGEMENT

5. FUNDING NUMBERS

6. AUTHOR(S)
2D LT ROXBURGH MICHELLE I

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
UNIVERSITY OF COLORADO AT COLORADO SPRINGS

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
THE DEPARTMENT OF THE AIR FORCE
AFIT/CIA, BLDG 125
2950 P STREET
WPAFB OH 45433

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
FY99-195

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
Unlimited distribution
In Accordance With AFI 35-205/AFIT Sup 1

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

DISTRIBUTION STATEMENT A
Approved for Public Release
Distribution Unlimited

14. SUBJECT TERMS	15. NUMBER OF PAGES	16. PRICE CODE
	25	

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

DTIC QUALITY INSPECTED 2

GROUND BASED INTERCEPT OF A BALLISTIC MISSILE:

BATTLE MANAGEMENT

by

MICHELLE IRENE ROXBURGH

B.S., United States Air Force Academy, 1998

A creative investigation to the Graduate Faculty of the

University of Colorado at Colorado Springs

in partial fulfillment of the

requirements for the degree of

Master of Engineering in Space Operations

Department of Mechanical and Aerospace Engineering

1999

Roxburgh, Michelle Irene (M.E., Space Operations)

Ground Based Intercept of a Ballistic Missile: Battle Management

Creative investigation directed by Dr. Don Caughlin

This creative investigation is based on work completed for ASE 583, Engineering Simulation. As a group project, the class designed and simulated a ballistic missile intercept system. This particular paper covers the battle management aspects of this simulation. Specifically, it addresses issues of infrared data processing, launch message timing, initial track generation, and track updating.

19990804 192

© Copyright by Michelle Irene Roxburgh 1999
All Rights Reserved

CONTENTS

TABLES

None.

FIGURES

Figure

I. INTRODUCTION

This paper is based on a group simulation project completed for ASE 583, Engineering Simulation. The class designed and simulated a ballistic missile intercept system; technical issues associated with the detection, acquisition, and hit of an incoming missile were primary concerns. Specific components modeled in this simulation include space-based sensors, ground based radars, battle management, the interceptor missile, and the global positioning system (GPS). This particular paper covers the battle management aspects of the simulation. The four main topics of discussion will include infrared data processing, launch message timing, initial track generation, and track updating.

II. BACKGROUND

Battle management is the capability for a designated operational commander to plan, coordinate, direct, and control weapons and sensors. Battle management technology includes the development of tool sets that assist in planning, reasoning, and decision-making, given uncertainty and incomplete information. The Ballistic Missile Defense Organization (BMDO) groups battle management with command, control, and communications in its Ballistic Missile Defense (BMD) Program; it is commonly referred to as BM/C3. BM/C3 has been identified by BMDO as one of the most difficult issues

associated with missile defense systems. Unlike other elements, BM/C3 is a software development problem rather than a hardware development problem. The primary software challenge is making use of apriori, inferred, and most likely information to make correct decisions for a given situation based on known information. Development of collaborative software tools connecting geographically distributed staff in near real-time fashion is a secondary issue. Other challenges for BM/C3 include keeping the system adaptable so that all imaginable situations are included and so that new capabilities provided by the rapidly evolving telecommunications structure may be used. Human-system interface is also an issue. In light of these technical problem areas, BMDO is conducting numerous exercises and war games to validate its BM/C3 concepts. These concepts continue to evolve through each iteration of testing[1].

In the ASE 583 simulation, battle management takes on a much more limited scope than the above BM/C3 description. Only selected behaviors are modeled due to the complexity of the problem. Behaviors were chosen based on preliminary guidance provided by a requirements list and based on the overall necessity of the behavior in the simulation. With this in mind, certain behaviors were selected; the responsibilities of the Battle Manager were determined to

include infrared data processing, interceptor missile launch timing, initial track determination of the target missile, and relay of track target updates to the interceptor missile. The Battle Manager is a central location of data and information distribution; as a result, data flow is of utmost importance. The following diagram shows the data flow to and from the Battle Manager:

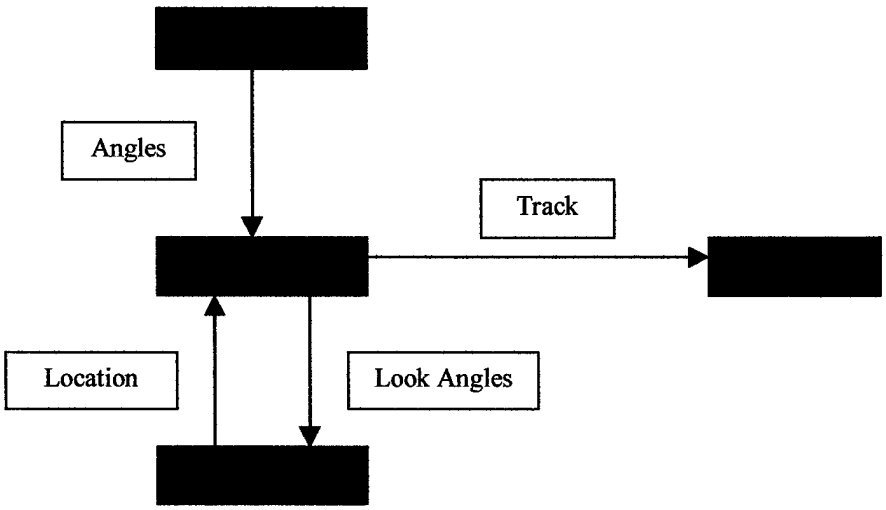

Figure 1 - Data Flow Diagram

The infrared sensors send data to the Battle Manager, which is then processed and sent to the search radar. Next, the track radar sends data to the Battle Manager, which is processed into an initial track or an updated track, depending on the number of radar observations. The initial track or updated track is then sent to the interceptor missile. The launch message is sent to the interceptor at the appropriate time.

The remainder of this paper is dedicated to the modeled behaviors and is divided into four sections:

1. IR Data Processing
2. Launch Message Timing
3. Initial Track Generation
4. Track Updates

Each section discusses requirements, inputs and outputs, algorithms, and assumptions.

III. IR DATA PROCESSING

The infrared sensors send data to the Battle Manager once a launch is detected. Information must then be conveyed to the search radar so that ground based radar systems can track the target missile. Because the infrared data is incompatible with radar architecture, the Battle Manager must process it into a usable form.

Infrared data processing depends primarily on the type of data provided by the infrared sensors. Originally, the plan was to include only one space-based infrared sensor in the simulation. Because one sensor alone cannot determine the altitude of the target missile, altitude would be assumed or modeled based on the type of missile. Calculations showed, however, that for a missile ranging between 0 and 1000 kilometers, latitude and longitude calculations could be several degrees off if invalid assumptions were made. This was unacceptable for the system. It was then determined that two infrared sensors at separate locations would provide a better solution. Two separate infrared sensors allow the position of the target missile to be determined through vector geometry. Azimuth and elevation for the search radar can then be calculated from the position vector.

Both infrared sensors are aboard geostationary satellites, which means their position is always known. Using

Figure 2, it is obvious that the position of the target missile (R_{ICBM}) can be found with simple vector addition.

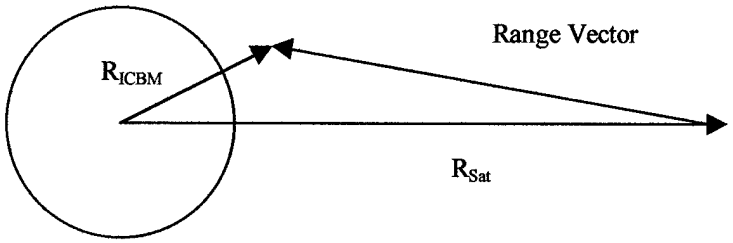

Figure 2 - Vector Geometery

Finding the range vector is the complex part of the process. This is why two infrared sensors at different locations are needed. Each sensor provides a nadir angle (η) and a rotation angle (θ) to the Battle Manager, as shown by Figure 3.

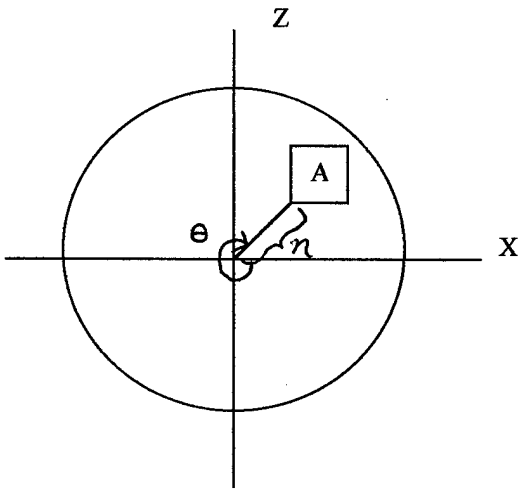

Figure 3 - Nadir and Rotation Angle Diagram

The nadir angle is the angle between the origin and the target missile (point A) from the geostationary satellite; the

rotation angle is the angle between the X axis and target missile, going clockwise.

The nadir and rotation angles from the infrared sensors can be used to calculate line of sight vectors from each satellite to the target missile; these are calculated in a North-East-Down (NED) coordinate frame with each satellite at the origin of its own frame. The line of sight vectors are unit vectors of the range vectors, and as a result, the range magnitude must be determined to find the components of the range vector. From the Figure 4, it is easy to see that the two triangles of satellites 1 and 2 share a common side, R_{ICBM}.

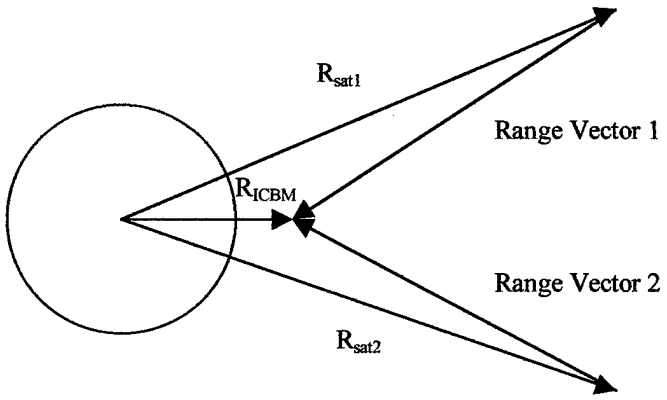

Figure 4 - Two Satellite Vector Geometry

Using this geometry, R_{sat1} plus Range Vector 1 can be set equal to R_{sat2} plus Range Vector 2.

$$R_{sat1} + R_1 L1 = R_{sat2} + R_2 L2$$

Where R_1 is the magnitude of Range Vector 1

R_2 is the magnitude of Range Vector 2

L_1 is the line of sight vector 1

L_2 is the line of sight vector 2

Then, either range magnitude can be solved for and substituted into the respective equation side to calculate R_{ICBM}. The final outputs of the process are an azimuth and elevation angle for the search radar site. The process as a whole is outlined in greater detail in the steps below:

Step 1: Calculate the line of sight vector from each infrared sensor to the target missile. This is done using the nadir (η) and rotation (θ) angles provided by the infrared sensors.

$$L = \begin{bmatrix} SIN(\eta) * SIN(\theta) \\ SIN(\eta) * COS(\theta) \\ COS(\eta) \end{bmatrix}$$

This line of sight vector is in a NED frame, which is useful for the given infrared data.

Step 2: By rearranging the geometry equation derived from Figure 4, R_2 can be calculated. The second satellite must be rotated into the first's NED frame, which is taken into account in the equation for R_2. As a note, the equation for R_2 is formulated especially for geostationary satellites.

$$R_{sat1} + R_1 L1 = R_{sat2} + R_2 L2$$

$$R_2 = \frac{SIN(-\Delta\lambda) * Sat2}{\frac{-L2(1) * L1(2)}{L1(1)} + L2(2) * COS(-\Delta\lambda) + L2(3) * SIN(-\Delta\lambda)}$$

Where λ is the IR sensor's longitude

$\Delta\lambda$ is equal to $\lambda_2 - \lambda_1$

Sat2 is the radius a geostationary satellite

Step 3: Calculate R_{ICBM} in the NED frame.

$$R_{ICBM}(NED) = [R_{sat2}] + R_2 * L2$$

Step 4: Rotate R_{ICBM} into the IJK frame.

$$R_{ICBM}(IJK) = ROT3(-\Delta\lambda)\, ROT2(90°)\, R_{ICBM}(NED)$$

Step 5: Calculate azimuth and elevation.

1. Calculate the position vector of the site in ECI coordinates, assuming a non-rotating and spherical earth.

$$\bar{r}_{siteIJK} = \begin{bmatrix} (R_{Earth} + h)Cos(\phi_{gc})Cos(\lambda_E) \\ (R_{Earth} + h)Cos(\phi_{gc})Sin(\lambda_e) \\ (R_{Earth} + h)Sin(\phi_{gc}) \end{bmatrix}$$

Where R_{Earth} is the radius of the earth

ϕ_{gc} is the geocentric latitude of the radar site

λ is the longitude of the site (east is positive)

h is the altitude of the site

2. Calculate the range vector from the radar site to the target missile in IJK coordinates.

$$\rho_{IJK} = R_{ICBM} - R_{Site}$$

3. Rotate the above range vector into SEZ coordinates.

$$\rho_{SEZ} = ROT2(90^\circ - \phi_{gc})\, ROT3(\lambda_E)\, \rho_{IJK}$$

4. Calculate azimuth and elevation.

$$AZ = TAN^{-1}\left(\frac{\rho_{SEZ}(2)}{-\rho_{SEZ}(1)}\right)$$

$$EL = SIN^{-1}\left(\frac{\rho_{SEZ}(3)}{\rho_{SEZ}}\right)$$

This process is coded in Matlab and is attached as Appendix A. The original code was completed in Fortran before transferring to the more user-friendly Matlab, which is compatible with the simulation software, Simulink and StateFlow.

Because there is only one radar site, azimuth and elevation were chosen as outputs for simplicity. Although less robust than outputting the position vector of the target, it consolidates the processing at the Battle Manager and lets the Radar Engineer concentrate on the radar system itself.

The infrared data processing, as well as the three issues yet to be discussed, are based two main assumptions: a non-rotating earth and a spherical earth. The non-rotating earth

assumption is valid because the earth rotates once every twenty-four hours and the simulation takes place in approximately thirty minutes, which means the earth will have completed a little over 2% of a rotation. Using this assumption simplifies calculations of Local Sidereal Time (LST). LST is the addition of Greenwich Sidereal Time (GST) and radar site longitude; because of the non-rotating earth assumption, GST is constant and chosen to be zero, which means LST is simply radar site longitude. This makes the problem easier in that LST is constant. The substitution of site longitude for LST has already been made in the above algorithm.

The second assumption is that of a spherical earth. In reality, the earth is not a perfect sphere. It is flattened at the poles, and gravitational perturbations differ at different locations around the earth. Geodetic latitude is normally used; for spherical earth calculations, however, geocentric latitude is used. Geocentric latitude (φ_{gc}) is the angle measured at the Earth's center from the plane of the equator to the point of interest; geodetic latitude (φ_{gd}) is the angle between the equatorial plane and the normal to the surface of the ellipsoid. This angle difference can cause discrepancies of up to several kilometers at high latitudes. Because all other simulation systems are assuming a spherical

earth, this assumption works. If needed, an oblate earth could be modeled in the Battle Manager portion of the simulation by simply replacing the spherical site vector function with an oblate site vector function.

It is understood that these assumptions make the simulation unrealistic. Their resulting simplicity, however, allow for a working simulation in a shorter amount of time. Because simulations typically take longer than planned, this is key. If time allows or further study is needed, these assumptions will be the first to be taken out.

Validation is a key issue in modeling behaviors. It is important to know that a process is producing expected results before it is used in a simulation. The IR data processing code went through several steps of validation. First, it was verified by inputting test data from the infrared sensors and then using previously coded functions to convert the outputted azimuth and elevation back into a position vector for the target missile. This position vector was then checked with the initial truth position vector for accuracy. Second, many numbers were inputted to the code as sanity checks. For example, if the target missile was directly over the radar station, elevation should be 90° and azimuth should be 0°; these expected results were verified with the computer code. Finally, even after the code was validated as an individual

entity, it had to be validated as a part of Simulink for the simulation. Because Matlab functions in Simulink accept only one vector as input and one vector as output, the code had to be tested yet again to check if the right inputs were being sent in the right order. This was accomplished using a test module in Simulink, inputting known numbers, and checking the output with previously calculated numbers.

IV. LAUNCH MESSAGE TIMING

The Battle Manager is responsible for the launch of the interceptor missile, which means the Battle Manager must send a launch message to the interceptor missile at a specified time. The time must allow the kill vehicle to intercept the target exoatmospherically. The interceptor directs itself toward the target missile using the target's current state; therefore, the launch message must be sent when the target missile is above the horizon relative to the interceptor launch site.

There are many ways the launch message timing could have been approached. An intercept point could have been chosen beforehand to meet this particular scenario's requirements; a launch window could have been determined given the range constraints of the interceptor and the exoatmospheric intercept requirement; or the interceptor missile could have been sent the launch message immediately after the initial track was acquired. Initially, the plan was to launch immediately after the initial track was acquired. This option, however, turned out to be unacceptable due to the kill vehicle controls. Because the interceptor directs itself toward the current position of the target missile, positions that are below the horizon relative to the kill vehicle cause it to fly through the earth and thus crash. When the initial

track is acquired, the target missile is well below the horizon relative to the interceptor. As a result, the launch message will be sent to the kill vehicle under the condition that the elevation angle relative to the interceptor launch site is greater than or equal to zero.

The algorithm for determining the elevation angle at the launch site is almost exactly the same as Step 5 of IR Data Processing. The only difference is that launch site data, including latitude, longitude, and altitude, is used instead of radar site data. Input includes the position and velocity vector of the target missile, and the only output value is elevation. The velocity vector is not needed in the calculation; however, it is easier to include it for the simulation architecture. The launch message is sent to the interceptor the first time elevation is greater than or equal to zero.Like the infrared data processing, this process is coded in Matlab; it is attached as Appendix B. It was validated in the infrared data processing section.

The same assumptions of a non-rotating, spherical earth still apply. The spherical earth assumption affects the elevation calculation by means of the site vector computation. The assumption is still valid, however, because all elements of the simulation are using it.

V. INITIAL TRACK GENERATION

An initial track of the target missile must be determined. For a given time, the track radar will send range, azimuth, and elevation to the Battle Manager. The Battle Manager is responsible for determining the position and velocity vectors of the target given this information.

To accomplish this task, three observations of range, azimuth, and elevation are needed. Because an initial track must be established as soon as possible, the first three observations from the track radar are used in this process.

To begin, each observation of range, azimuth, and elevation is converted into a position vector in the Earth Centered Inertial (ECI) coordinate frame. This initial orbit determination problem then becomes one of three position vectors and time. There are two primary methods for this type of problem: Gibbs method and Herrick-Gibbs method. Herrick-Gibbs method of initial orbit determination is a variation of the Gibbs method. Both find the velocity vector of the middle observation and require three nonzero, coplanar vectors, which represent three time-sequential vectors of an orbit. Gibbs method is the more robust method; it works best when the vectors are more than a degree apart. In the simulation problem, however, data is processed at close intervals because of its critical nature. As a result, the position vectors are

less than a degree apart; this is where Herrick-Gibbs' more limited application is useful. Herrick-Gibbs method works best when the position vectors are closely spaced. The following outlines a step-by-step procedure on how this process is implemented:

Step 1: Calculate the position vector of the radar site in ECI coordinates, assuming a non-rotating and spherical earth, in the same manner as Step 5 of IR Data Processing.

Step 2: Calculate the range vector of all three observations in SEZ coordinates.

$$\bar{\rho}_{SEZ} = \begin{bmatrix} -\rho Cos(El)Cos(\beta) \\ \rho Cos(El)Sin(\beta) \\ \rho Sin(El) \end{bmatrix}$$

Step 3: Using the nonrotating, spherical earth assumption, transform the range vector of all three observations into ECI coordinates.

$$\bar{\rho}_{IJK} = ROT3(-\lambda_E)ROT2(-(90° - \phi_{gc}))$$

Step 4: Calculate the position vector of all three observations in ECI coordinates.

$$\bar{r}_{IJK} = \bar{\rho}_{IJK} + \bar{r}_{siteIJK}$$

Step 5: Use Herrick-Gibbs method of initial orbit determination to determine the velocity vector of the second observation.

1. Calculate the changes in time.

$$\Delta t_{31} = t_3 - t_1$$
$$\Delta t_{32} = t_3 - t_2$$
$$\Delta t_{21} = t_2 - t_1$$

2. Calculate the middle velocity vector.

$$\vec{V}_2 = -\Delta t_{32}\left(\frac{1}{\Delta t_{21}\Delta t_{31}} + \frac{\mu}{12r_1^3}\right)\vec{r}_1 + (\Delta t_{32} - \Delta t_{21})\left(\frac{1}{\Delta t_{21}\Delta t_{32}} + \frac{\mu}{12r_2^3}\right)\vec{r}_2$$
$$+ \Delta t_{21}\left(\frac{1}{\Delta t_{32}\Delta t_{31}} + \frac{\mu}{12\vec{r}_3^3}\right)\vec{r}_3$$

The output is an orbit track of position and velocity at the middle observation time. The actual Matlab code is attached as Appendix C; it was validated by inputting test data and matching the output with known results. Initially, it was not known if Herrick-Gibbs could handle observations as close together as one second or less; tests indicated that Herrick-Gibbs could be used with observations 0.0001 seconds apart or less, which is more than the simulation needs. Herrick-Gibbs method actually increased its accuracy with smaller time intervals and angle separations.

VI. TRACK UPDATES

The interceptor missile needs the most current target track possible. As a result, the Battle Manager must update the target track at intervals required by the kill vehicle. The updated track must then be sent to the kill vehicle.

When new information, including range, azimuth, and elevation, is available from the track radar, the target track is updated using an Extended Kalman Filter. An Extended Kalman Filter is ideal for the simulation because it estimates the current state of the target missile. There are times, however, when new radar data is not available. This happens because the kill vehicle needs updates more often than the track radar sends new information. Because the kill vehicle needs up-to-date information, the track must be updated using another method. For these circumstances, the track must be updated using another method.

Inputs for the Extended Kalman Filter include initial position and velocity vectors, an initial covariance matrix, and the time change between states. In addition, range, azimuth, and elevation for the second state are required. Outputs include position and velocity vectors and a covariance matrix. The Extended Kalman Filter will be implemented in the following manner:

Step 1: Propagate the initial state forward from $time_K$ to $time_{K+1}$ using a two-body propagator. This code is based on David A. Vallado's PKEPLER algorithm in Fundamentals of Astrodynamics and Applications[2]. The initial position and velocity vectors are propagated through the time of flight to determine the current position and velocity vectors.

$$R_k, V_k, TOF \Rightarrow R_{K+1}, V_{K+1}$$

The code is included as Appendix D.

Step 2: Use Step 5 of IR Data Processing to calculate the nominal observations (range, azimuth, and elevation) of the predicted state vector.

Step 3: Calculate the 6 x 6 state transition matrix, ϕ.

$$\phi = I + F\Delta t + 0.5(F^2\Delta t)$$

Where I is the identity matrix (6 x 6)

F is a two body partial derivative matrix

Δt is the time difference between states

Step 4: Calculate the predicted covariance matrix, P_{K+1}.

$$P_{K+1} = \phi P_K \phi^T + Q$$

Where P_K is the initial covariance matrix

Q is the modeling error

Step 5: Calculate H, a 3 x 6 matrix. For this simulation, the H matrix is computed numerically. Each element of the predicted position vector is perturbed individually,

corresponding to the first three columns. Perturbed range, azimuth, and elevation are then calculated for the observation. The difference between the perturbed and nominal values represents the numerator in each H matrix value. The denominator is simply the difference between the perturbed R element and its nominal value; this process is called finite differencing. The last three columns of the H matrix are zero.

$$H = \begin{bmatrix} \frac{\Delta\rho_1}{\Delta r_I} \frac{\Delta\rho_1}{\Delta r_J} \frac{\Delta\rho_1}{\Delta r_K} ..0..0..0 \\ \frac{\Delta az_1}{\Delta r_I} \\ \frac{\Delta el_1}{\Delta r_I} .....0..0..0 \end{bmatrix}$$

Step 6: Calculate the Kalman Gain, a 6 x 3 matrix.

$$K_{K+1} = P_{K+1}H^T_{K+1}[H_{K+1}P_{K+1}H^T_{K+1} + R_{K+1}]^{-1}$$

Where R is the inverse of the weighting matrix

Step 7: Calculate the residual, Z.

$$Z_{K+1} = \begin{bmatrix} \rho_{Actual} - \rho_{Nominal} \\ Az_{Actual} - Az_{Nominal} \\ El_{Actual} - El_{Nominal} \end{bmatrix}$$

Step 8: Calculate the new state vector, X_{NEW} and the new covariance matrix, P_{NEW}.

$$\Delta x_{K+1} = K_{K+1}Z_{K+1}$$

$$X_{new} = X_{K+1} + \Delta x_{K+1}$$

$$P_{NEW} = [P_{K+1} - K_{K+1}H_{K+1}]P_{K+1}$$

This process has been coded and verified in Matlab; it is attached as Appendix E. Problems can occur in any Kalman Filter when the filter becomes smug, meaning the error is so small new observations are being disregarded by the filter. This can be fixed by adjusting the modeling error, Q, in the filter. In addition, the initial covariance matrix is assumed. This matrix can have an effect on how fast the Kalman filter converges on a solution.

For times when no new radar information is available, an alternative method is used to update the track. Inputs include position and velocity vectors, a covariance matrix, and a change in time; outputs include the new position and velocity vectors and a new covariance matrix. This method is implemented in the following steps:

Step 1: Calculate the 6 x 6 state transition matrix, ϕ.

$$\phi = I + F\Delta t + 0.5(F^2\Delta t)$$

Where I is the identity matrix (6 x 6)

F is a two body partial derivative matrix

Δt is the time difference between states

Step 2: Calculate the new (predicted) covariance matrix, P_{K+1}. This is just an estimate, since no new radar measurements are available.

$$P_{K+1} = \phi P_K \phi^T + Q$$

Where P_K is the initial covariance matrix

Q is the modeling error

Step 3: Calculate the new (predicted) position and velocity state using the two-body propagator of Step 1 of the Kalman filter algorithm.

This process has been coded and verified in Matlab; it is attached as Appendix F; additional battle management code used throughout all of the algorithms is attached as Appendix G. Because of the small time increments, this process is fairly simple and accurate even with the spherical, non-rotating earth assumption.

VII. CONCLUSION

This paper covered aspects of battle management simulated in a ballistic missile defense scenario designed by the Spring Semester class of ASE 583. Topics covered include infrared data processing, launch message timing, initial track generation, and track updates. The surface of battle management was hardly scratched modeling these four behaviors and knowing the path of the target missile with no uncertainty; the complexity of an actual battle management system is readily apparent. It is easy to see why an actual missile defense system has not yet been successfully developed.

BIBLIOGRAPHY

[1]"Ballistic Missile Defense Program," Ballistic Missile Defense Organization. Internet. Available from http//www.acq.osd.mil/bmdo/bmdolink/html/ccc.html.

[2]Vallado, David A., Fundamentals of Astrodynamics and Applications, New York: McGraw-Hill Companies, Inc, 1997.

Appendix A:

IR Data Processing Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%   function ir                                                                 %
%                                                                               %
%     Michelle I. Roxburgh                        11 May 1999                   %
%                                                                               %
%     This function determines the azimuth and elevation angle of               %
%          the target missile at a predetermined radar site, given              %
%          a nadir angle and a rotation angle from two separate IR              %
%          sensors.                                                             %
%                                                                               %
%     Input is a vector of width 4.                                             %
%                                                                               %
%     Output is a vector of width 2.                                            %
%                                                                               %
%     Input:                                                                    %
%        Rotation1 - Rotation angle from sensor 1 (deg)                         %
%        Nadir1    - Nadir angle from sensor 1 (deg)                            %
%        Rotation2 - Rotation angle from sensor 2 (deg)                         %
%        Nadir2    - Nadir angle from sensor 2 (deg)                            %
%                                                                               %
%     Output:                                                                   %
%        AZ        - Azimuth from the given radar site (deg)                    %
%        EL        - Elevation from the given radar site (deg)                  %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [OUTPUT] = ir(INPUT);

format long

% INPUT = [Rotation1,Nadir1,Rotation2,Nadir2]

Rotation1 = double(INPUT(1,1)*pi/180);
Nadir1 = double(INPUT(1,2)*pi/180);
Rotation2 = double(INPUT(1,3)*pi/180);
Nadir2 = double(INPUT(1,4)*pi/180);

REarth = 6378137.0;
GST = 0.0;
F = 0.006694385;

%******** Satellites at GEO

Sat2 = 42241100.0;
Lon1 = 0.0 * pi/180;
Lon2 = -35.0 * pi/180;
DeltaLon = Lon2 - Lon1;

%******** Radar Site

Sitlat = 62.0 * pi/180;
Sitlon = -47.0 * pi/180;
Sitalt = 5.0;
LST = GST + Sitlon;

Sitlat = atan((1 - F)*tan(Sitlat));
```

```
[RS] = Site (Sitlat,LST,Sitalt);

%******** IR Sensor Output

L1(1,:) = sin(Nadir1) * sin(Rotation1);
L1(2,:) = sin(Nadir1) * cos(Rotation1);
L1(3,:) = cos(Nadir1);
L1_mag = sqrt((L1(1)^2) + (L1(2)^2) + (L1(3)^2));

L2(1,:) = sin(Nadir2) * sin(Rotation2);
L2(2,:) = sin(Nadir2) * cos(Rotation2);
L2(3,:) = cos(Nadir2);
L2_mag = sqrt((L2(1)^2) + (L2(2)^2) + (L2(3)^2));

Temp1 = sin(-DeltaLon) * Sat2;
Temp2 = -L2(1) * L1(2) / L1(1);
Temp3 = L2(2) * cos(-DeltaLon) + L2(3) * sin(-DeltaLon);
Range2 = Temp1 / (Temp2 + Temp3);

RTemp1(1,:) = Range2 * L2(1);
RTemp1(2,:) = Range2 * L2(2);
RTemp1(3,:) = -Sat2 + Range2 * L2(3);
RTemp1_mag = sqrt((RTemp1(1)^2) + (RTemp1(2)^2) + (RTemp1(3)^2));

RTemp2 = ROT2(RTemp1, pi/2);
R = ROT3(RTemp2, -Lon2);
R_mag = sqrt(R(1)^2 + R(2)^2 + R(3)^2);

[RHO,AZ,EL] = RAZEL (R,RS,Sitlat,Sitalt,LST);

AZ = double(AZ*180/pi);
EL = double(EL*180/pi)

% OUTPUT = [AZ,EL]

OUTPUT = [AZ,EL];

return;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                             %
%   function site                                                             %
%                                                                             %
%     Michelle I. Roxburgh                        11 May 1999                 %
%                                                                             %
%     This function calculates the position location, given site              %
%          latitude, local sidereal time, and site altitude.  It              %
%          assumes a spherical earth.                                         %
%                                                                             %
%     Input:                                                                  %
%        Sitlat - Site latitude (rad)                                         %
%        LST    - Local sidereal time (rad)                                   %
%        Sitlon - Site longitude (rad)                                        %
%                                                                             %
%     Output:                                                                 %
%        RS     - Site position vector - 3 (m)                                %
%                                                                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [RS] = Site (Sitlat,LST,Sitalt);

format long

% ------------------------- Define Constants --------------------------

REarth = 6378137.0;

RS(1,:) = (REarth+Sitalt)*cos(Sitlat)*cos(LST);
RS(2,:) = (REarth+Sitalt)*cos(Sitlat)*sin(LST);
RS(3,:) = (REarth+Sitalt)*sin(Sitlat);

return
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%   function razel                                                              %
%                                                                               %
%     Michelle I. Roxburgh                      11 May 1999                     %
%                                                                               %
%     This function determines range, azimuth, and elevation, given             %
%          a position vector, a site position vector, site                      %
%          site latitude, site altitude, and local sidereal time.               %
%                                                                               %
%     Input:                                                                    %
%        R       - Position vector (m)                                          %
%        RS      - Site position vector (m)                                     %
%        Sitlat  - Site latitude (rad)                                          %
%        Sitalt  - Site altitude (m)                                            %
%        LST     - Local sidereal time (rad)                                    %
%                                                                               %
%     Output:                                                                   %
%        RHO     - Range (m)                                                    %
%        AZ      - Azimuth (rad)                                                %
%        EL      - Elevation (rad)                                              %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [RHO,AZ,EL] = RAZEL (R,RS,Sitlat,Sitalt,LST);

format long

% Calculate RHO IJK

RHOIJK(1,:) = double(R(1) - RS(1));
RHOIJK(2,:) = double(R(2) - RS(2));
RHOIJK(3,:) = double(R(3) - RS(3));
[RHOIJKmag] = MAG (RHOIJK);

% Rotate RHO to SEZ

Colat = double(pi/2.0 - Sitlat);
[Temp] = ROT3(RHOIJK,LST);
[RHOSEZ] = ROT2(Temp,Colat);
RHOSEZmag = MAG (RHOSEZ);

% Calculate RHO, AZ, and EL

RHO = RHOSEZmag;
AZ = atan2(RHOSEZ(2),-RHOSEZ(1));
EL = asin(RHOSEZ(3)/RHO);

if AZ < 0.0
   AZ = AZ + 2.0D0*pi;
end

return
```

Appendix B:

Launch Message Timing Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%   function launch                                                             %
%                                                                               %
%     Michelle I. Roxburgh                      11 May 1999                     %
%                                                                               %
%     This function determines elevation, given a position vector.              %
%          It also accepts the velocity vector, which makes                     %
%          integration with Simulink easier.                                    %
%                                                                               %
%     Input is a vector of width 6.  It includes the target                     %
%          position vector (3) and the target velocity vector (3).              %
%                                                                               %
%     Output is elevation.                                                      %
%                                                                               %
%     Input:                                                                    %
%        R   - Position vector (m)                                              %
%        V   - Velocity vector (m/s)                                            %
%                                                                               %
%     Output:                                                                   %
%        El  - Elevation (deg)                                                  %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [OUTPUT] = launch (INPUT);

format long

R = [INPUT(1,1);
   INPUT(2,1);
   INPUT(3,1)];

V = [INPUT(4,1);
   INPUT(5,1);
   INPUT(6,1)];

F = 0.006694385;

LaunchLat = double(40.75*pi/180.0);
LaunchLon = double(-74.1*pi/180.0);
LaunchAlt = 230
GST = 0.0;
LST = double (GST + LaunchLon);

LaunchLat = atan((1 - F)*tan(LaunchLat));

[RS] = Site (LaunchLat,LST,LaunchAlt);

% Calculate RHO IJK

RHOIJK(1,:) = double(R(1) - RS(1));
RHOIJK(2,:) = double(R(2) - RS(2));
RHOIJK(3,:) = double(R(3) - RS(3));
[RHOIJKmag] = MAG (RHOIJK);

% Rotate RHO to SEZ
```

```
Colat = double(pi/2.0 - Sitlat);
[Temp] = ROT3(RHOIJK,LST);
[RHOSEZ] = ROT2(Temp,Colat);
RHOSEZmag = MAG (RHOSEZ);

% Calculate EL

EL = double(asin(RHOSEZ(3)/RHO));

EL = double(EL*180/pi);

OUTPUT = EL;

return
```

Appendix C:

Initial Track Generation Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                             %
%   function hgibbs                                                           %
%                                                                             %
%     Michelle I. Roxburgh                        11 May 1999                 %
%                                                                             %
%     This function implements Herrick-Gibbs method of initial               %
%          orbit determination, given three observations of range,            %
%          azimuth, elevation, and time.                                      %
%                                                                             %
%     The input is a vector of width 12.  It includes range,                  %
%         azimuth, elevation, and time of three observations.                 %
%                                                                             %
%     Output is a vector of width 6.  It includes the position and            %
%         and velocity vectors at the middle observation time.                %
%                                                                             %
%     Input:                                                                  %
%        RHO1 - Range at observation 1                                        %
%        AZ1  - Azimuth at observation 1                                      %
%        EL1  - Elevaton at observation 1                                     %
%        RHO2 - Range at observation 2                                        %
%        AZ2  - Azimuth at observation 2                                      %
%        EL2  - Elevaton at observation 2                                     %
%        RHO3 - Range at observation 3                                        %
%        AZ3  - Azimuth at observation 3                                      %
%        EL3  - Elevaton at observation 3                                     %
%        T1   - Time at observation 1                                         %
%        T2   - Time at observation 2                                         %
%        T3   - Time at observation 3                                         %
%                                                                             %
%     Output:                                                                 %
%        R2   - Position vector at the middle observation time                %
%        V2   - Velocity vector at the middle observation time                %
%                                                                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [OUTPUT] =  HGIBBS (INPUT);

format long

% INPUT = [Range1,AZ1,EL1,T1,...,EL3,T3];

RHO1 = INPUT(1,1);
AZ1 = INPUT(2,1)*pi/180;
EL1 = INPUT(3,1)*pi/180;
T1 = INPUT(4,1);
RHO2 = INPUT(5,1);
AZ2 = INPUT(6,1)*pi/180;
EL2 = INPUT(7,1)*pi/180;
T2 = INPUT(8,1);
RHO3 = INPUT(9,1);
AZ3 = INPUT(10,1)*pi/180;
EL3 = INPUT(11,1)*pi/180;
T3 = INPUT(12,1);

F = 0.006694385;
Mu = double(398600.5 * 1000^3);
```

```
Sitlat = 62.0 * pi/180;
Sitlon = -47.0 * pi/180;
Sitalt = 5.0;

%Sitlat = atan((1 - F)*tan(Sitlat));

GST = 0.0
LST = double(GST + Sitlon);

[R1] = SiteTrack (Sitlat,LST,Sitalt,RHO1,AZ1,EL1);
[R2] = SiteTrack (Sitlat,LST,Sitalt,RHO2,AZ2,EL2);
[R3] = SiteTrack (Sitlat,LST,Sitalt,RHO3,AZ3,EL3);

[R1Mag] = MAG (R1);
[R2Mag] = MAG (R2);
[R3Mag] = MAG (R3);

% Determine changes in time

T31 = double(T3 - T1);
T32 = double(T3 - T2);
T21 = double(T2 - T1);

% Calculate V2

Term1 = double(-T32*((1.0/(T21*T31)) + Mu/(12.0*R1Mag^3)));
Term2 = double((T32-T21)*((1.0/(T21*T32)) + Mu/(12.0*R2Mag^3)));
Term3 = double(T21*((1.0/(T32*T31)) + Mu/(12.0*R3Mag^3)));

Temp1 = double(Term1*R1);
Temp2 = double(Term2*R2);
Temp3 = double(Term3*R3);

V2 = double(Temp1 + Temp2 + Temp3);

% OUTPUT = [R2,V2]

OUTPUT = [R2',V2']

return
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                             %
%   function sitetrack                                                        %
%                                                                             %
%     Michelle I. Roxburgh                      11 May 1999                   %
%                                                                             %
%     This function determines a position vector, given range                 %
%          azimuth, elevation, site latitude, local sidereal                  %
%          time, and site altitude.                                           %
%                                                                             %
%     Input:                                                                  %
%        Sitlat - Site latitude (rad)                                         %
%        LST    - Local sidereal time (rad)                                   %
%        Sitalt - Site altitude (m)                                           %
%        RHO    - Range (m)                                                   %
%        AZ     - Azimuth (rad)                                               %
%        EL     - Elevation (rad)                                             %
%                                                                             %
%     Output:                                                                 %
%        RS     - Site position vector - 3 (m)                                %
%                                                                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [R] = SiteTrack (Sitlat,LST,Sitalt,RHO,AZ,EL);

format long

[RS] = Site (Sitlat,LST,Sitalt);

RHOSEZ(1,:) = double(-RHO*cos(EL)*cos(AZ));
RHOSEZ(2,:) = double(RHO*cos(EL)*sin(AZ));
RHOSEZ(3,:) = double(RHO*sin(EL));

Colat = double(pi/2.0 - Sitlat);

[Temp] = ROT2 (RHOSEZ,-Colat);
[RHOIJK] = ROT3 (Temp,-LST);

R = double(RHOIJK + RS);

return
```

Appendix D:

Two-Body Propagator (PKEPLER) Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                             %
%   function pkepler                                                          %
%                                                                             %
%     Michelle I. Roxburgh                      11 May 1999                   %
%                                                                             %
%     This function propagates a position and velocity forward                %
%          for a specified time.                                              %
%                                                                             %
%     Input:                                                                  %
%        R1    - Position vector - 3 (m)                                      %
%        V1    - Velocity vector - 3 (m/s)                                    %
%        TOF   - Time of flight (s)                                           %
%                                                                             %
%     Output:                                                                 %
%        R2    - Position vector - 3 (m)                                      %
%        V2    - Velocity vector - 3 (m/s)                                    %
%                                                                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [R2,V2] = PKepler (R1,V1,TOF);

format long

Mu = double(398600.5 * 1000^3);
J = 0.00108263;
Re = 6378137.0;
Limit = 0.015;

[P,A,Ecc,Inc,Omega0,Argp0,Nu0,Mean0,U0,L0,CapPi0] = ELORB (R1,V1);

% Determine NBar, OmgDot, ArgDot

Local1 = double(J*(Re^2)/(P^2));
Local2 = double(1.0 - (1.5*((sin(Inc))^2)));
Local3 = double(sqrt(1.0-(Ecc^2)));

N0 = double(sqrt(Mu/(A^3)));
NBar = double(N0*(1.0+1.5*Local1*Local2*Local3));

OmgDot = double(-1.5*Local1*NBar*cos(Inc));
ArgDot = double(1.5*Local1*NBar*(2.0 - 2.5*(sin(Inc))^2));

% Take into account circular and equatorial orbits

if Ecc >= Limit & Inc >= Limit
   Omega = Omega0 + OmgDot*TOF;
   Argp = Argp0 + ArgDot*TOF;
   Mean = Mean0 + NBar*TOF;
   U = 'Undefined';
   L = 'Undefined';
   CapPi = 'Undefined';
end

if Ecc >= Limit & Inc < Limit
   CapPi = CapPi0 + (OmgDot+ArgDot)*TOF;
   Mean = Mean0 + NBar*TOF;
```

```
   U = 'Undefined';
   L = 'Undefined';
   Omega = 'Undefined';
   Argp ='Undefined';
end

if Ecc < Limit & Inc < Limit
   L = L0 + (OmgDot+ArgDot+NBar)*TOF;
   CapPi = 'Undefined';
   Mean = 'Undefined';
   U = 'Undefined';
   Argp = 'Undefined';
   Omega = 'Undefined';
end

if Ecc < Limit & Inc >= Limit
   Omega = Omega0 + OmgDot*TOF;
   U = U0 + (ArgDot + NBar)*TOF;
   L = 'Undefined';
   CapPi = 'Undefined';
   Mean = 'Undefined';
   Argp = 'Undefined';
end

% Determine R and V from updated COE's

[Nu] = NEWTONR (Ecc,Mean);

[R2,V2] = RANDV (P,Ecc,Inc,Omega,Argp,Nu,U,L,CapPi);

return
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%   function elorb                                                              %
%                                                                               %
%     Michelle I. Roxburgh                        11 May 1999                   %
%                                                                               %
%     This function converts ECI position and velocity vectors                  %
%          into classical orbital elements.                                     %
%                                                                               %
%     Input:                                                                    %
%        R       - Position vector - 3 (m)                                      %
%        V       - Velocity vector - 3 (m/s)                                    %
%                                                                               %
%     Output:                                                                   %
%        P       - Semi-latus rectum (m)                                        %
%        A       - Semi-major axis (m)                                          %
%        Ecc     - Eccentricity                                                 %
%        Inc     - Inclination (rad)                                            %
%        Omega   - Longitude of the ascending node (rad)                        %
%        Argp    - Argument of perigee (rad)                                    %
%        Nu      - True anomaly (rad)                                           %
%        M       - Mean anomaly (rad)                                           %
%        U       - Argument of latitude (rad)                                   %
%        L       - True longitude (rad)                                         %
%        CapPi   - True longitude of perigee (rad)                              %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [P,A,Ecc,Inc,Omega,Argp,Nu,M,U,L,CapPi] = ELORB (R,V);

format long

Mu = double(398600.5 * 1000^3);
REarth = 6378137.0;
Small = 0.000001;
Smallei = 0.015;

[RMag] = MAG (R);
[VMag] = MAG (V);

HBar = cross(R,V);
[HMag] = MAG (HBar);

NBar = cross([0 0 1],HBar);
[NMag] = MAG (NBar);

Temp1 = double((VMag^2 - Mu/RMag)/Mu);
Temp2 = double(dot(R,V)/Mu);
EBar = double(Temp1*R - Temp2*V);
[Ecc] = MAG (EBar);

SME = double((VMag^2 * 0.5) - Mu/RMag);
if abs(SME) > Small
   A = double(-Mu/(2.0*SME));
else
   A = 'Infinite';
end
```

```
P = double(HMag^2/Mu);
Inc = double(acos(HBar(3)/HMag));

TypeOrbit = 'EI';
if Ecc < Small
   if Inc < Smallei or abs(Inc-pi) < Smallei
      TypeOrbit = 'CE';
   else
      TypeOrbit = 'CI';
   end
else
   if Inc < Smallei or abs(Inc-pi) < Smallei
      TypeOrbit = 'EE';
   end
end

if NMag > Small
   Omega = double(acos(NBar(1)/NMag));
   if NBar(2) < 0.0
      Omega = double(2.0*pi - Omega);
   end
else
   Omega = 'Undefined';
end

if TypeOrbit == 'EI'
   Argp = double(acos(dot(NBar,EBar)/(NMag*Ecc)));
   if EBar(3,:) < 0.0
      Argp = double(2.0*pi - Argp);
   end
else
   Argp = 'Undefined';
end

if (TypeOrbit == 'EI') | (TypeOrbit == 'EE')
   Nu = double(acos(dot(EBar,R)/(Ecc*RMag)));
   if double(dot(R,V)) < 0.0
      Nu = double(2.0*pi - Nu);
   end
else
   Nu = 'Undefined';
end

if TypeOrbit == 'CI'
   U = double(acos(dot(NBar*R)/(NMag*RMag)));
   if R(3,:) < 0.0
      U = double(2.0*pi - U);
   end
else
   U = 'Undefined';
end

if Ecc > Smallei & TypeOrbit == 'CE'
   CapPi = double(acos(EBar(1)/Ecc));
   if EBar(2,:) < 0.0
      CapPi = double(2.0*pi - CapPi);
```

```
   end
else
   CapPi = 'Undefined';
end

if RMag > Small & TypeOrbit == 'CE'
   L = double(acos(R(1)/RMag));
   if R(2,:) < 0.0
      L = double(2.0*pi - L);
   end
else
   L = 'Undefined';
end

if double(Ecc - 1.0) > Smallei
   F = double(acosh(A-RMag/(A*Ecc)));
   M = Ecc*sinh(F) - F;
else
   if abs(Ecc-1.0) < Smallei
      D = double(sqrt(P)*tan(Nu*0.5));
      M = double((1.0/6.0)*(3.0*D + D^3));
   else
      if Ecc > Smallei
         Temp = double(1.0 + Ecc*cos(Nu));
         if abs(Temp) < Small
            M = 0.0
         else
            sinE = double(sqrt(1.0 - Ecc^2)*sin(Nu)/Temp);
            cosE = double((Ecc + cos(Nu))/Temp);
            if abs(sinE) > 1.0
               sinE = sign(1.0,sinE);
            end
            if abs(cosE) > 1.0
               cosE = sign(1.0,cosE);
            end
            E = double(atan2(sinE,cosE));
            M = double(E - Ecc*sin(E));
         end
      else
         if TypeOrbit == 'CE'
            M = L;
         else
            M = U;
         end
      end
   end
   if M < 0.0
      M = double(M + 2.0*pi);
   end
end

return
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%   function randv                                                              %
%                                                                               %
%     Michelle I. Roxburgh                      11 May 1999                     %
%                                                                               %
%     This function determines a position and velocity vector,                  %
%          given classical orbital elements.                                    %
%                                                                               %
%     Input:                                                                    %
%        P       - Semi-latus rectum (m)                                        %
%        Ecc     - Eccentricity                                                 %
%        Inc     - Inclination (rad)                                            %
%        Omega   - Longitude of the ascending node (rad)                        %
%        Argp    - Argument of perigee (rad)                                    %
%        Nu      - True anomaly (rad)                                           %
%        U       - Argument of latitude (rad)                                   %
%        L       - True longitude (rad)                                         %
%        CapPi   - True longitude of perigee (rad)                              %
%                                                                               %
%     Output:                                                                   %
%        R       - Position vector - 3 (m)                                      %
%        V       - Velocity vector - 3 (m/s)                                    %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Michelle I. Roxburgh
% Function RandV
% This function determines a position and velocity vector, given classical
%      orbitial elements.

function [R,V] = RANDV (P,Ecc,Inc,Omega,Argp,Nu,U,L,CapPi);

format long

Small = 0.015;
Mu = double(398600.5 * 1000^3);

if Ecc < Small
   if Inc < Small or abs(Inc - pi) < Small
      Argp = 0.0;
      Omega = 0.0;
      Nu = L;
   else
      Argp = 0.0;
      Nu = U;
   end
else
   if Inc < Small or abs(Inc - pi) < Small
      Argp = CapPi;
      Omega = 0.0;
   end
end

Temp = P/(1.0 + Ecc*cos(Nu));
RPQW(1,:) = Temp*cos(Nu);
RPQW(2,:) = Temp*sin(Nu);
RPQW(3,:) = 0.0;
```

```
RPQWmag = sqrt(RPQW(1,:)^2+RPQW(2,:)^2+RPQW(3,:)^2);

VPQW(1,:) = -sin(Nu)*sqrt(Mu/P);
VPQW(2,:) = (Ecc + cos(Nu))*sqrt(Mu/P);
VPQW(3,:) = 0.0;
VPQWmag = sqrt(VPQW(1,:)^2+VPQW(2,:)^2+VPQW(3,:)^2);

[TempVec1] = ROT3 (RPQW,-Argp);
[TempVec2] = ROT1 (TempVec1,-Inc);
[R] = ROT3 (TempVec2,-Omega);

[TempVec3] = ROT3 (VPQW,-Argp);
[TempVec4] = ROT1 (TempVec3,-Inc);
[V] = ROT3 (TempVec4,-Omega);

return;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%   function NEWTONR                                                            %
%                                                                               %
%     Michelle I. Roxburgh                        11 May 1999                   %
%                                                                               %
%     This function determines true anomaly using eccentricity                  %
%          and mean anomaly.                                                    %
%                                                                               %
%     Input:                                                                    %
%        Ecc - Eccentricity                                                     %
%        M   - Mean anomaly (rad)                                               %
%                                                                               %
%     Output:                                                                   %
%        Nu  - True anomaly (rad)                                               %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Nu] = NEWTONR (Ecc,M);

format long

E0 = M;
I = 1;

E1 = E0 - ((E0 - Ecc*sin(E0) - M)/(1.0 - Ecc*cos(E0)));

while abs(E1-E0) >= 0.0000001 & I <= 20
   E0 = E1;
   E1 = E0 - ((E0 - Ecc*sin(E0) - M)/(1.0 - Ecc*cos(E0)));
   I = I + 1;
end

sinv = (sqrt(1.0 - Ecc^2)*sin(E1))/(1.0 - Ecc*cos(E0));
cosv = (cos(E1) - Ecc)/(1.0 - Ecc*cos(E1));
Nu = atan2(sinv,cosv);

if Nu < 0.0
   Nu = 2.0*pi + Nu;
end

return
```

Appendix E:

Track Updating with New Measurements Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%   function radar                                                              %
%                                                                               %
%     Michelle I. Roxburgh                        11 May 1999                   %
%                                                                               %
%     This function executes an Extended Kalman Filter.                         %
%                                                                               %
%     Input is a vector of width 46.                                            %
%                                                                               %
%     Output is a vector of width 42.                                           %
%                                                                               %
%     Input:                                                                    %
%        Rinit     - Initial position vector - 3 (m)                            %
%        Vinit     - Initial velocity vector - 3 (m/s)                          %
%        P         - Covariance matrix (6 x 6)                                  %
%        Range     - Range (m)                                                  %
%        Azimuth   - Azimuth (deg)                                              %
%        Elevation - Elevation (deg)                                            %
%                                                                               %
%     Output:                                                                   %
%        R         - Position vector - 3 (m)                                    %
%        V         - Velocity vector - 3 (m/s)                                  %
%        PNew      - Covariance matrix (6 x 6)                                  %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [OUTPUT] = radar (INPUT);

format long

INPUT = [R(1)...V(3), P(1,1)...P(6,6),Range,AZ,EL,DeltaT]

Rinit(1,1) = INPUT(1,1);
Rinit(2,1) = INPUT(2,1);
Rinit(3,1) = INPUT(3,1);

Vinit(1,1) = INPUT(4,1);
Vinit(2,1) = INPUT(5,1);
Vinit(3,1) = INPUT(6,1);

P = [INPUT(7:12,1)';
     INPUT(13:18,1)';
     INPUT(19:24,1)';
     INPUT(25:30,1)';
     INPUT(31:36,1)';
     INPUT(37:42,1)'];

Obs = [INPUT(43,1);
   double(INPUT(44,1)*pi/180);
   double(INPUT(45,1)*pi/180)];

DeltaT = INPUT(46,1);

Radar (1,1) = 62.0*pi/180;
Radar (2,1) = -47.0*pi/180;
Radar (3,1) = 5.0;
```

```
F = 0.006694385;

Radar(1,1) = atan((1 - F)*tan(Radar(1,1)));

LST = Radar(2);

Noise(1,1) = 0.026;
Noise(2,1) = 0.026*pi/180.0;
Noise(3,1) = 0.022*pi/180.0;

% Calculate the Weight Matrix

R = zeros(3);
R(1,1) = 1.0/Noise(1)^2;
R(2,2) = 1.0/Noise(2)^2;
R(3,3) = 1.0/Noise(3)^2;

% Calculate the Predicted State

[RPredict,VPredict] = PKEPLER (Rinit,Vinit,DeltaT);

[RS] = SITE (Radar(1),LST,Radar(3));

[RHONom,AZNom,ELNom] = RAZEL (RPredict,RS,Radar(1),Radar(3),LST);

XPredict(1,1) = RPredict(1,1);
XPredict(2,1) = RPredict(2,1);
XPredict(3,1) = RPredict(3,1);
XPredict(4,1) = VPredict(1,1);
XPredict(5,1) = VPredict(2,1);
XPredict(6,1) = VPredict(3,1);

Z(1,1) = double(Obs(1) - RHONom);
Z(2,1) = double(Obs(2) - AZNom);
Z(3,1) = double(Obs(3) - ELNom);

% Find the PHI Matrix

[PHIMatrix] = PHI (Rinit,DeltaT);

% Calculate the Predicted Covariance, assume no Q

PBar = PHIMatrix*P*PHIMatrix';

% Find the H Matrix

[H] = HMatrix (RPredict,LST,Radar);

% Calculate K

KMatrix = PBar * H' * inv(H * PBar * H' + R);

% Calculate DeltaX and the New State Vector

DeltaXHat = KMatrix*Z;

XNew = XPredict + DeltaXHat;
```

```
% Calculate the New Covariance Matrix

PNew = PBar - (KMatrix*H*PBar);

% OUTPUT = [R(1)...V(3),P(1,1)...P(6,6)]

OUTPUT = [XNew',PNew(1,1:6),PNew(2,1:6),PNew(3,1:6),PNew(4,1:6),...
          PNew(5,1:6),PNew(6,1:6)];

return;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%   function PHI                                                                %
%                                                                               %
%     Michelle I. Roxburgh                        11 May 1999                   %
%                                                                               %
%     This function determines the state transition matrix needed               %
%          needed for the Extended Kalman Filter.                               %
%                                                                               %
%     Input:                                                                    %
%        R       - Position vector - 3 (m)                                      %
%        DeltaT  - Time between filtering (s)                                   %
%                                                                               %
%     Output:                                                                   %
%        PHIMatrix - State transition matrix (6 x 6)                            %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [PHIMatrix] = PHI (R,DeltaT);

format long

% Get the F Matrix

[F] = FMatrix(R);

% Do calculations for PHI Matrix

PHIMatrix = eye(6) + F*DeltaT + (F*F*DeltaT^2)/2.0;

return
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%   function FMatrix                                                            %
%                                                                               %
%     Michelle I. Roxburgh                        11 May 1999                   %
%                                                                               %
%     This function determines the two-body partial derivative                  %
%          matrix needed for the Extended Kalman Filter.                        %
%                                                                               %
%     Input:                                                                    %
%        R  - Position vector - 3 (m)                                           %
%                                                                               %
%     Output:                                                                   %
%        F  - Two body partial derivative matrix, F (6 x 6)                     %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [F] = FMatrix (R);

format long

Mu = double(398600.5 * 1000^3);

[RMag] = MAG (R);

% Set initial F entries to 0

F = zeros(6);

% Set nonzero entries of F to their values

F(1,4) = 1.0;
F(2,5) = 1.0;
F(3,6) = 1.0;

F(4,1) = double((-Mu/RMag^3) + (3.0*R(1)^2)/RMag^5);
F(5,1) = double(3.0*Mu*R(1)*R(2)/RMag^5);
F(6,1) = double(3.0*Mu*R(1)*R(3)/RMag^5);
F(4,2) = double(3.0*Mu*R(1)*R(2)/RMag^5);
F(5,2) = double((-Mu/RMag^3) + (3.0*Mu*R(2)^2)/RMag^5);
F(6,2) = double(3.0*Mu*R(2)*R(3)/RMag^5);
F(4,3) = double(3.0*Mu*R(1)*R(3)/RMag^5);
F(5,3) = double(3.0*Mu*R(2)*R(3)/RMag^5);
F(6,3) = double((-Mu/RMag^3) + (3.0*Mu*R(3)^2)/RMag^5);

return
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%   function HMatrix                                                            %
%                                                                               %
%     Michelle I. Roxburgh                        11 May 1999                   %
%                                                                               %
%     This function determines the H matrix need for the Extended               %
%          Kalman Filter.                                                       %
%                                                                               %
%     Input:                                                                    %
%        RNom    - Position vector - 3 (m)                                      %
%        LST     - Local sidereal time (rad)                                    %
%        Radar   - Vector of site latitude, longitude, and altitude             %
%                                                                               %
%     Output:                                                                   %
%        H       - H Matrix                                                     %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [H] = HMatrix (RNom,LST,Radar);

format long

% Calculate Nominal Observation Values

[RS] = Site (Radar(1),LST,Radar(3));
[RHONom,AZNom,ELNom] = RAZEL (RNom,RS,Radar(1),Radar(3),LST);

% Calculate values of H matrix entries

H = zeros(3,6);

for I = 1:3
      RPert = double(1.005*RNom(I,:));
      DeltaR = double(0.005*RNom(I,:));
   if I == 1
      RNomTemp(1,:) = RPert;
      RNomTemp(2,:) = RNom(2,:);
      RNomTemp(3,:) = RNom(3,:);
   elseif I == 2
      RNomTemp(1,:) = RNom(1,:);
      RNomTemp(2,:) = RPert;
      RNomTemp(3,:) = RNom(3,:);
   elseif I == 3
      RNomTemp(1,:) = RNom(1,:);
      RNomTemp(2,:) = RNom(2,:);
      RNomTemp(3,:) = RPert;
   end
   RNomTemp(4,:) = sqrt(RNomTemp(1,:)^2+RNomTemp(2,:)^2+RNomTemp(3,:)^2);
   [RHOPert,AZPert,ELPert] = RAZEL (RNomTemp,RS,Radar(1),Radar(3),LST);
   DeltaRHO = double(RHOPert - RHONom);
   DeltaAZ = double(AZPert - AZNom);
   DeltaEL = double(ELPert - ELNom);
   H(1,I) = double(DeltaRHO/DeltaR);
   H(2,I) = double(DeltaAZ/DeltaR);
   H(3,I) = double(DeltaEL/DeltaR);
end
```

```
return
```

Appendix F:

Track Updating without New Measurements Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%   function propagate                                                          %
%                                                                               %
%     Michelle I. Roxburgh                        11 May 1999                   %
%                                                                               %
%     This function updates the target state and covariance                     %
%          matrix between updated information from the track                    %
%          radar.                                                               %
%                                                                               %
%     Input is a vector of width 43.                                            %
%                                                                               %
%     Output is a vector of width 42.                                           %
%                                                                               %
%     Input:                                                                    %
%        DeltaT     - Time since the last update                                %
%        Rinit      - Initial position vector - 3 (m)                           %
%        Vinit      - Initial velocity vector - 3 (m/s)                         %
%        P          - Covariance matrix (6 x 6)                                 %
%                                                                               %
%     Output:                                                                   %
%        R          - Position vector - 3 (m)                                   %
%        V          - Velocity vector - 3 (m/s)                                 %
%        PNew       - Covariance matrix (6 x 6)                                 %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [OUTPUT] = propagate (INPUT);

format long

INPUT = [DeltaT,R(1)...V(3),P(1,1)...P(6,6)]

DeltaT = INPUT(1,1);

Rinit(1,1) = INPUT(2,1);
Rinit(2,1) = INPUT(3,1);
Rinit(3,1) = INPUT(4,1);

Vinit(1,1) = INPUT(5,1);
Vinit(2,1) = INPUT(6,1);
Vinit(3,1) = INPUT(7,1);

P = [INPUT(8:13,1)';
     INPUT(14:19,1)';
     INPUT(20:25,1)';
     INPUT(26:31,1)';
     INPUT(32:37,1)';
     INPUT(38:43,1)'];

Radar (1,1) = 62.0*pi/180;
Radar (2,1) = -47.0*pi/180;
Radar (3,1) = 5.0;
F = 0.006694385;

Radar(1,1) = atan((1 - F)*tan(Radar(1,1)));
```

```
% Find the PHI Matrix

[PHIMatrix] = PHI (Rinit,DeltaT);

% Calculate the Predicted Covariance, assume no Q

PNew = PHIMatrix*P*PHIMatrix';

OUTPUT = [XNew',PNew(1,1:6),PNew(2,1:6),PNew(3,1:6),PNew(4,1:6),...
          PNew(5,1:6),PNew(6,1:6)];

return;
```

Appendix G:

Additional Functions

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%   function ROT1                                                               %
%                                                                               %
%     Michelle I. Roxburgh                      11 May 1999                     %
%                                                                               %
%     This function rotates about the X axis, given a vector and                %
%          an angle.                                                            %
%                                                                               %
%     Input:                                                                    %
%        Vector1  - Input vector of width 3                                     %
%        RotAngle - Rotation angle (rad)                                        %
%                                                                               %
%     Output:                                                                   %
%        Vector2  - Output vector of width 3                                    %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Vector2 = ROT1(Vector1, RotAngle);

M_1_2 = [1, 0, 0;
   0, cos(RotAngle), sin(RotAngle);
   0, -sin(RotAngle), cos(RotAngle)];

Vector2 = M_1_2 * Vector1;

return
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%   function ROT2                                                               %
%                                                                               %
%     Michelle I. Roxburgh                      11 May 1999                     %
%                                                                               %
%     This function rotates about the Y axis, given a vector and                %
%          an angle.                                                            %
%                                                                               %
%     Input:                                                                    %
%        Vector1  - Input vector of width 3                                     %
%        RotAngle - Rotation angle (rad)                                        %
%                                                                               %
%     Output:                                                                   %
%        Vector2  - Output vector of width 3                                    %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Vector2 = ROT2(Vector1, RotAngle);

M_1_2 = [cos(RotAngle), 0, -sin(RotAngle);
   0, 1, 0;
   sin(RotAngle), 0, cos(RotAngle)];

Vector2 = M_1_2 * Vector1;

return
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%   function ROT3                                                               %
%                                                                               %
%     Michelle I. Roxburgh                      11 May 1999                     %
%                                                                               %
%     This function rotates about the Z axis, given a vector and                %
%          an angle.                                                            %
%                                                                               %
%     Input:                                                                    %
%        Vector1  - Input vector of width 3                                     %
%        RotAngle - Rotation angle (rad)                                        %
%                                                                               %
%     Output:                                                                   %
%        Vector2  - Output vector of width 3                                    %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Vector2 = ROT3(Vector1, RotAngle);

M_1_2 = [cos(RotAngle), sin(RotAngle), 0;
   -sin(RotAngle), cos(RotAngle), 0;
   0, 0, 1];

Vector2 = M_1_2 * Vector1;

return
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%   function mag                                                                %
%                                                                               %
%     Michelle I. Roxburgh                      11 May 1999                     %
%                                                                               %
%     This function determines the magnitude of a vector (3).                   %
%                                                                               %
%     Input:                                                                    %
%        Vector   - Vector of width 3                                           %
%                                                                               %
%     Output:                                                                   %
%        Magnitude - Magnitude of the vector                                    %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Magnitude] = MAG (Vector);

format long

Magnitude = sqrt(Vector(1)^2+Vector(2)^2+Vector(3)^2);

return
```