**User Guide for the Generalized Reconstruction of EDL Trajectory, Atmospheric and Aerodynamics (GRETAA)**

Version 1.0

April 2014

Prepared by:

S. Dutta

waahvsov@yahoo.com

**Table of Contents**

[Introduction 1](#_Toc290320714)

[Extended Kalman Filtering Basics 2](#_Toc290320715)

[Structure of the Program 7](#_Toc290320716)

[Main Directory 7](#_Toc290320717)

[Measurement\_Files 9](#_Toc290320718)

[Process\_Files 12](#_Toc290320719)

[Utilities 17](#_Toc290320720)

[Application\_Master\_Files 25](#_Toc290320721)

[Sample Application Case (MER Simulated Data) 26](#_Toc290320722)

[References 29](#_Toc290320723)

[Index 30](#_Toc290320724)

# Introduction

The Extended Kalman Filter (EKF) program for which this user manual has been written was developed under a research contract with NASA. The code and its user manual is a deliverable for the Research Opportunities in Aeronautics, NASA Research Announcement (NRA) NNH07ZEA001N-ELD1, Subtopic A.6.2, NOI N7-EDL1-0014. The title of the project for which this code was originally developed is “Atmospheric Flight Reconstruction of Mars Robotic Missions to Provide an Aeroassist Data Set that Enables Future High Mass Mars Entry Systems.”

The EKF tool was written in various versions of MATLAB, ranging from 2008a to 2009a. All of the codes provided are backwards compatible up to MATLAB 2008a and have been tested extensively with MATLAB 2009a.

The program contains data and driver files for a few Mars EDL-related datasets. These reconstruction-related applications are the 1997 Pathfinder, 2003 Mars Exploration Rover-A, 2008 Phoenix, Crew Exploration Vehicle (CEV) Simulated data (testing for MSL MEDLI-type dataset) and several versions of simulated Mars EDL trajectory (POST-II simulated MER trajectory) data to test the EKF tool’s capability for atmospheric and aerodynamic reconstruction. The data files are included in the copy of the codes being delivered for the completion of the NRA.

The following document takes the user through the structure of the program, the different directories in the tool and also provides a list of .m files and their input/output behavior. The document also contains some basic information about Extended Kalman filters. This discussion has some results from the Mars Pathfinder reconstruction done using an earlier version of this tool. Finally, a sample application of this tool is also present in this manual.

# Extended Kalman Filtering Basics

Extended Kalman Filter is the statistical filter used here to combine the measurement information with the nominal estimate of the state. A Kalman filter is based on linear filter theory and uses the difference between predicted and measured data to update the estimate of the state. An extended Kalman filter is a modification of the original Kalman filter to express the nonlinearity in the system dynamics that is lost in the linearization needed for the original Kalman filter. Consider the linearization of the state vector at time increment k as a function of the state at time k-1 and a random state noise vector (**w**) as seen in equation below. Note that **x** is the deviation in the state from its nominal estimate, while **X** is the state vector.

The state transition matrix (**Ф**) is the function that propagates the state from k-1 to k. The linear Kalman filter needs a nominal trajectory from the initial state to the end state, and the filter estimates the deviation in the state around this nominal trajectory. The extended Kalman filter does not need a nominal trajectory from the start to the end of the trajectory. Instead, the propagation from k-1 to k is done using the nonlinear equations of state (see equations below). Then, when the state estimate is updated at time k using the measurements, this new estimate is used to propagate to time k+1. Thus, the nonlinearity inherent in the system dynamics can be better handled using the extended Kalman filter algorithm rather than the linearized Kalman filter. Please see reference 1 for more information about the equations of motions.

In addition to the equation that defines the state vector, relationships are also needed to define the uncertainty in the state and how these values propagate over time. In Eq. 4, ε was introduced as the measurement error. The state vector also has a similar error term known as the state error vector (**e**k) which contains the error in each element of the state vector at time k. EKF assumes that the state error is also normally distributed and thus a state covariance matrix (**P**) can be introduced which is defined as E[**e**k**e**kT]. A measurement covariance matrix (**R**k) can be defined at time k where **R**k = E[εεT]. As is the case with the state vector, the state covariance vector must be propagated from time k-1 to k. State transition matrices can be used to accomplish this operation as seen below, where **Q**k is the state noise covariance (i.e. **Q**k = E[**ww**T]).

A Riccatti-type differential equation can also be used to update the covariance vector as seen in the equation below. Here **A** is the Jacobian of the equations of motion with respect to the state vector and produces a matrix similar to what is found in Eq. 6 for the measurement expressions. **B** is the partial derivatives of the equations of state with respect to the state noise vector. All of these matrices are evaluated at the current time k-1 and are used to propagate P to time k.

In order to begin the EKF process, a nominal estimate at the current time must be found. If the current time is k, then the nominal state estimate () can be found from the final estimate at k-1 as described before. The covariance matrix can be similarly estimated at time k ().Then, the best state estimate at time k () is found by equations shown below, where **K**k is the Kalman filter gain (see equation below) and and **y**k is the measurement residual vector. The measurement residual vector (**y**) is defined as the difference between all of the actual measurements at the current time and the corresponding predicted measurements at the nominal state. Within the expression for the Kalman gain, **H**k is the measurement sensitivity matrix and evaluated at time k and **R**k is the measurement noise matrix. Finally, the state covariance for the best estimate () is found as shown below, where **I** is the identity matrix.



The algorithm for this filter can be summarized as follows:

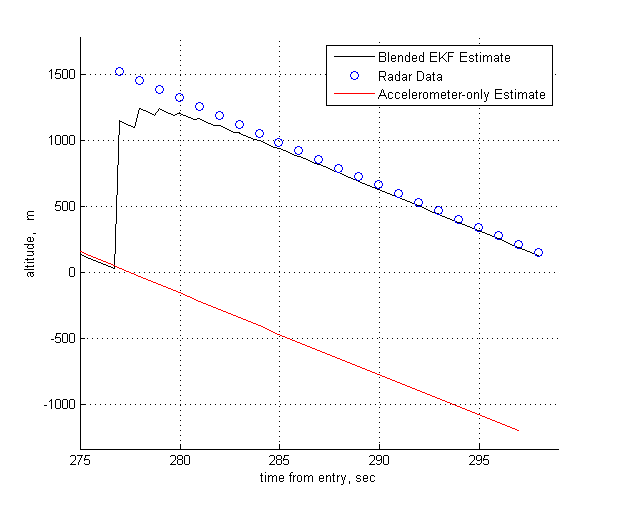
1. Initialize the state vector and the state covariance matrix at time tk-1=t0 and let k =1, where k is an index of the epoch when measurement was taken.
2. Read in measurement at time tk.
3. Calculate a nominal state at tk () by integrating the non-linear equations of motions with as the initial condition.
4. Calculate the nominal state covariance matrix () using a state transition matrix or the Riccati equations.
5. Calculate the measurement residual vector (**y**k)­, the measurement sensitivity matrix (**H**k), and the Kalman gain (**K**k) using the nominal state and state covariance.
6. Calculate the best estimate of the state () and state covariance ().
7. Increment counter k and go back to step 2 until measurements at all times have been read.

A difference between the extended Kalman filter and the standard Kalman filter is highlighted in step 3 of the algorithm where the nominal state is calculated by integrating the non-linear equations and the last best estimate is used as the initial condition. The standard Kalman filter relies on a linearized, nominal trajectory for this propagation. In a highly non-linear problem, large deviations could be propagated through this linear approximation. The extended Kalman filter effectively re-linearizes the state estimate at the last best estimate found whenever a new measurement is processed, thus reducing deviations that can result from linearizing a non-linear problem.

Additionally, an advantage of the extended Kalman filter over some other filters is that it provides an efficient way to incorporate more than one type of measurement. Each measurement type has a unique measurement sensitivity matrix and observation covariance. Thus, when the filter is processing measurement type A, the appropriate H and R matrices are used with the nominal state error covariance. If measurement type B has to be processed at the next time step, only the H and R matrices will change in the algorithm.

Moreover, one can see that the state is affected by three factors by looking at the Kalman gain and the state update equations. The Kalman gain is a function of the current state uncertainty (Pk), the measurement uncertainty (Rk) and the residual between the predicted and actual measurements (**y**). If the state estimate is more certain than the measurements being processed, the filter will be minimally affected by the data. In addition, the filter can “blend” the information from the various data types, and the state estimate will be weighted towards the measurement with the smallest observation error, which can be gleaned from its observation covariance matrix. When two or more data types are being processed sequentially, the state estimate initially may oscillate between the measurements from the differing sources, but the filter quickly uses the weighting information from the R matrices to calculate the blended estimate.

Finally, the residual of the measurements can scale the update of the state. If the predicted measurements were very close to the actual measurements, then the state update will be minimal. To demonstrate this concept, Mars Pathfinder reconstruction data is used below. Details about this trajectory reconstruction can be found in Ref. 2, with additional background information in Ref. 3. This application is also present in the codes provided as ‘Mars Pathfinder.’ Figure 1 demonstrates the effect of data blending by showing the estimate of altitude for Mars Pathfinder when the radar altimeter measurements are included with the accelerometer measurements. One way to compare the effect of uncertainty in the estimate is to vary the weighting factor for the measurements being used. As one can see, the altitude estimate initially oscillates between the accelerometer and radar altimeter observations, but finally the EKF moves the estimate towards the less uncertain measurements, which in this case comes from the radar altimeter.



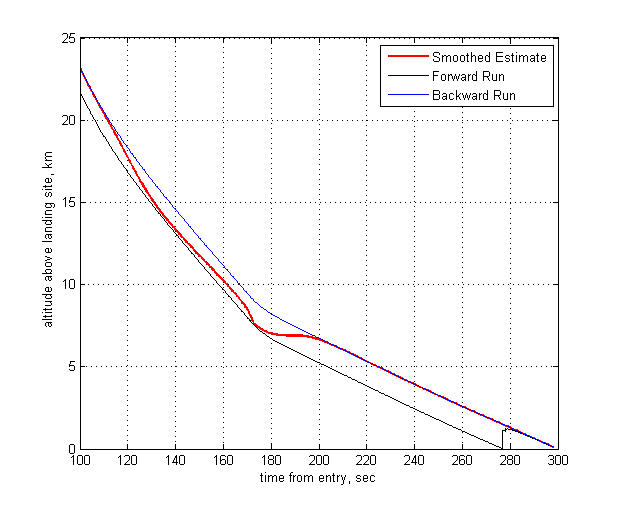
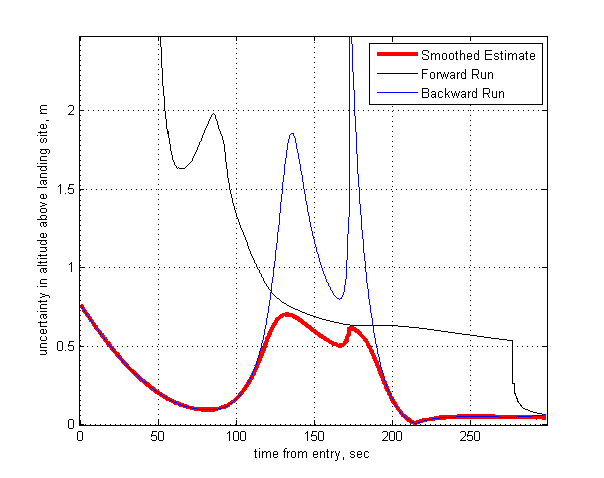
**Time from Entry (sec)**

**Altitude above Mean Radius (m)**

Figure : Effect of blending different data types on the estimate of altitude for Mars Pathfinder. The altitude shown is above Mars mean radius.

Another advantage of the extended Kalman filter is that it can be used to sequentially reconstruct the trajectory in either a forward or backwards manner. The reconstruction can be conducted starting from the atmospheric entry all the way down to the ground (forward pass) or using a projected landing location to estimate the trajectory up to the entry conditions (backwards pass). The forward pass starts its estimate from an initial state and covariance that is found independent of the trajectory reconstruction process. Also, the reconstruction is conducted in a chronological manner. The backwards pass has the advantage of starting at a smaller uncertainty value as it begins from the end of the forward estimate. The forward (f) and backward (b) pass estimates can be combined using the Fraser-Potter smoothing solution,4 which is shown below in Figure 2.

An advantage of combining both the forward and backward estimates is to find an optimal estimate of the trajectory. The forward pass estimate at time k uses the measurement data from tentry to k, while the backward pass uses the measurement data from time tground to k. The combined smoothed estimate can use measurement data at all times to create the estimate at k. Figure 2 shows the forward, backward, and smoothed estimate of the altitude of Mars Pathfinder, which is used to demonstrate the advantage of the smoothing algorithm. The 1σ uncertainties associated with the three estimates are also shown.

**Time from Entry (sec)**

**1σ Uncertainty in Altitude (m)**

**Time from Entry (sec)**

**Altitude above Mean Radius (km)**

1. **Altitude estimate (b) 1σ uncertainty in altitude**

Figure : Forward and backward runs and smoothing on the estimate of altitude for Mars Pathfinder.

# Structure of the Program

## Main Directory

The tool is divided into four subdirectories and three shared files. Figure 3 shows the structure of the structure of the main directory. Every dataset (or application in this nomenclature) has its master files in the ‘Application\_Master\_Files’ subdirectory. However, each of those applications shares some common files. These shared files are present in the three other subdirectories seen in Figure 3, as well as the three MATLAB files: ‘ExtendedKalmanFileter.m’, ‘propogateState.m’ and ‘smoother.m.’ Script files are blue in color and function files are brown in color.

**Application\_Master\_Files**

**Measurement\_Files**

**Process\_Files**

**Utilities**

**GT\_Extended\_Kalman\_Filter**

**ExtendedKalmanFilter.m**

**propogateState.m**

**smoother.m**

Figure : Main directory of the GT Extended Kalman Filter tool.

### ExtendedKalmanFilter.m

ExtendedKalmanFilter.m is the file that takes in the current state and uncertainty estimate and the measurement to be processed and then creates an updated estimate of the state and the state uncertainty. The EKF file already has the handles to populate the H-Matrix (measurement sensitivity matrix) and the R-Matrix (measurement noise matrix) from the function caller. The state is propagated from t1to t2 and then using the new estimate of the state at t2, the state deviation update is calculated using the EKF algorithm (see EKF basics section). Then the state deviation update is added to the propagated estimate of the state to obtain the best estimate of the state at time t2. The state covariance is also updated accordingly. See the inputs and outputs of the function below for detail.

The ‘inputs’ structure that is an input for this function is described in detail later.

***Inputs***

t2 = Time of the next measurement

X1 = State estimate at current time

P1 = State covariance estimate at current time (vector format)

t1 = Current time (t2 – t1 = change in time btw. now and the next meas.)

measurements = vector of measurements at time t2 (can change in length)

HMatrixFcnHandle = handle to the meas. sensitivity matrix for meas. at t2

RMatrixFcnHandle = handle to the meas. sensitivity matrix for meas. at t2

Inputs = structure that contains constants and vectors needed for propagation

error\_level = uncertainty in the measurement (R-matrix value)

***Outputs***

x2estimate = state deviation vector update

X2estimate = state vector update at time t2  
Uncertainty = uncertainty (1-sigma) value for the state estimates

P2estimate = state covariance vector update at time t2 (uncertainty is 1- sigma for these values)

### PropogateState.m

This function uses the non-linear equations of motion to propagate a state from time t1 to time t2. The states are updated using the MATLAB intrinsic 4th order Runge-Kutta propagator (ode45) and uses user-specified tolerances passed through the ‘inputs’ structure. The state vector passed to the propagator contains both the physical states from the equations of motion as well as the state covariance matrix elements which are propagated using the Ricatti-type equations (see EKF basics section).

***Inputs***

x1 = State estimate at current time

P1 = State covariance estimate at current time (vector format)

t1 = Current time (t2 – t1 = change in time btw. now and the next meas.)

t2 = Time of the next measurement

Inputs = structure that contains constants and vectors needed for propagation

***Outputs***

x2 = state vector update at time t2  
P2 = state covariance vector update at time t2

### Smoother.m

This script takes the outputs from the EKF during the forward and backward runs and combines them for a best estimated trajectory. The smoothing is done using the Fraser-Potter algorithm. Note, since this is a script, not a function, this file must be called from a workspace that has a struct called fwdstate’ containing the results of a forward run of the EKF and a ‘bckstate’ struct containing the results of a backward run of the EKF. Each struct has a .state and .Pvec entry (e.g fwdstate.state and fwdstate.Pvec) which contains the time history of the state estimates and the covariance matrix elements. The corresponding fwdstate and bckstate elements should be of the same dimension. Moreover, the time associated with the first element in bckstate should be the same time associated with the first element in fwdstate. That is, even though bckstate contains results from a backward run of the EKF, the time vector has been re-oriented to correspond from the start of the fwdstate.

***Outputs (generated in the workspace since this file is a script not a function)***

smthstate = struct resembling fwdstate and bckstate; two elements: smthstate.state = state vector time history

smthstate.Pvect = state covariance matrix time history

## Measurement\_Files

This tool has the files that support the EKF’s measurement update process. The files that begin with the “meas” prefix calculate what parameters sensors for the current state vector and time. The files that begin with the “H” prefix calculate the measurement sensitivity matrix, i.e. the H-matrix, used in the EKF update. The files that begin with the “R” prefix, calculates the measurement uncertainty matrix or the R-matrix in the EKF update. Figure 4 shows the setup of the measurement files directory.

**Measurement\_Files**

**H\_Port\_pressure.m**

**H\_Radar\_altimeter.m**

**meas\_Port\_pressure.m**

**meas\_Range.m**

**R\_Port\_pressure.m**

**R\_Radar\_altimeter.m**

Figure : Measurement files directory.

### H\_Port\_pressure.m

Calculates the measurement sensitivity matrix and the residual between the predicted and actual measurements for the port pressure measurements such as the ones found in the MEADS dataset. Note that this function uses the total static pressure as the “measurement.” The gage pressure at the transducer is available in the function and it can be turned into the “measurement.”

***Inputs***

t = current time

x = current state vector (this function doesn't care about the size)

measurements = vector with the measurements

inputs = struct containing information needed for state propagation and

EKF update. Contents depend on what application is being run

***Outputs***

Hk = measurement sensitivity matrix (n\_meas by n\_states)

zk = measurement residual (btw. predicted and actual measurements)

### H\_Radar\_altimeter.m

Calculates the measurement sensitivity matrix and the residual between the predicted and actual measurements for the radar altimeter measurements from on-board the lander.

***Inputs***

t = current time

x = current state vector (this function doesn't care about the size)

measurements = vector with the measurements

inputs = struct containing information needed for state propagation and

EKF update. Contents depend on what application is being run

***Outputs***

Hk = measurement sensitivity matrix (n\_meas by n\_states)

zk = measurement residual (btw. predicted and actual measurements)

### meas\_Port\_pressure.m

Computes the pressure measurements at the forebody pressure transducers given the current trajectory and atmospheric states. This function is used by the H\_Port\_Pressure.m file.

***Inputs***

t = current time (not used in the calculations)

x = current estimated state (see EOM file for the full list)

PressureInputs = struct containing the look-up table with Cp data

***Outputs***

PortPress = stagnation pressure at port

GaugePress = dynamic pressure at port

### meas\_Range.m

Computes range measurements from the vehicle to the ground given the current trajectory states. This function is used by the H\_Radar\_altimeter.m file.

***Inputs***

t = time (sec)

x = states

inputs = structure containing information for state propagation

***Outputs***

range = range from station to s/c (m)

azimuth = heading angle from station to s/c (deg)

elevation = elevation angle from station to s/c (deg)

### R\_Port\_pressure.m

Calculates the measurement noise matrix for the port pressure measurement.

***Inputs***

t = current time

x = current state vector

measurements = a vector with the measurements at the current time

inputs = a struct with information for state propagations

uncertainty = a vector with uncertainty in the current measurements

***Output***

Rk = measurement sensitivity matrix (num\_meas by num\_meas)

### R\_Radar\_altimeter.m

Calculates the measurement noise matrix for the radar altimeter.

***Inputs***

t = current time

x = current state vector

measurements = a vector with the measurements at the current time

inputs = a struct with information for state propagations

uncertainty = a vector with uncertainty in the current measurements

***Output***

Rk = measurement sensitivity matrix (num\_meas by num\_meas)

## Process\_Files

The process files directory has functions that calculate the A, B and Q-matrices used in the state and covariance propagation equations in the EKF. There are five types of processes that are included in this tool. These processes are described below. Figure 5 shows the structure of the process files directory.

**SixDOF**

**SixDOFwAtmo**

**SixDOFwAtmoWind**

**ThreeDOF**

**Process\_Files**

**gravity.m**

**ThreeDOFNoAccel**

Figure : Structure of the process files directory.

1. SixDOF – Six degree of freedom equations of motion. The equations are:

The states are geocentric position (radius, latitude and longitude), velocity in the vehicle-carried (local horizontal) frame, and attitude information using quaternion.

1. SixDOFwAtmo – Six degree of freedom equations of motion with atmospheric states. The equations are the same as the SixDOF set before with the addition of the atmospheric states:

The states are geocentric position (radius, latitude and longitude), velocity in the vehicle-carried (local horizontal) frame, attitude information using quaternion and freestream pressure and density. The freestream pressure and density equations have been developed using the perfect gas relation and the hydrostatic equation.

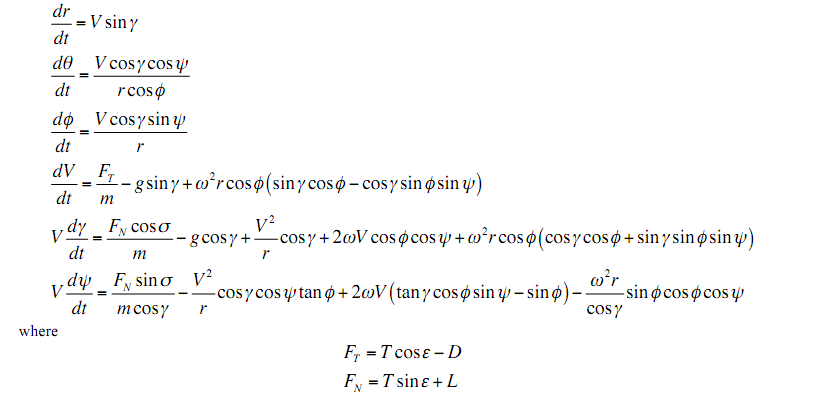
1. SixDOFwAtmoWind – Six degree of freedom equations of motion with atmospheric states and wind. The equations are the same as the SixDOFwAtmo set before with the addition of the wind velocity states. This set of process equations is currently not used with any of the application files and has not been validated. The time rate of change equations for the wind velocities are all zero, since wind velocity is “calculated” using the EKF measurement update states and pressure port sensor data.

The states are geocentric position (radius, latitude and longitude), velocity in the vehicle-carried (local horizontal) frame, attitude information using quaternion, freestream pressure and density, and wind velocity in the vehicle-carried frame.

1. ThreeDOF – Three degree of freedom equations of motion. The equation set is shown below. There is some ambiguity if IMU data is used for the sensed acceleration values. Since attitude information is not propagated by these equations of motion, then the information needed to move IU acceleration measurements from the body axis/IMU axis to the vehicle-carried frame needs to be known beforehand or some attitude sequence needs to be assumed.

The states are geocentric position (radius, latitude and longitude) and velocity in the vehicle-carried (local horizontal) frame.

1. ThreeDOFNoAccel – Three degree of freedom equations of motion without using IMU acceleration. Using this EOM, IMU can be treated as a measurement not part of the process. The equation set is shown below.5 This equation set uses pre-determined information about the aerodynamic coefficients (such as CD, CL) to propagate the velocity vector. The aerodynamic coefficients are usually a function of M∞, α and β in regimes of flight where this tool will be used. Thus, the Mach number, angle of attack and sideslip number has to be known even before the trajectory reconstruction is complete! This continuity problem is solved by doing a deterministic (i.e. not EKF based) reconstruction based on the IMU data only. Then the angle of attack and sideslip history can be reconstructed using the deterministic reconstruction, IMU data available and a force coefficient look-up table. Finally, the known angle of attack history can be used to find the correct CL and CD to put in the ThreeDOFNoAccel equations. This procedure is described in Ref. 2.



The states are geocentric position (radius, latitude and longitude) and velocity in the vehicle-carried (local horizontal) frame. The velocity vector can be constructed using the velocity magnitude (V), flight path angle (γ) and the heading angle (ψ). Note that the heading angle is not the same as azimuth angle. Ψ = 0 is due East. This angle can be best described as angle between local parallel of latitude and projection of the velocity vector on horizontal plane.

### Gravity.m

This function outputs the gravity vector in the vehicle-carried (local horizontal) frame. There are two options currently available for gravity calculations: ‘r-squared”-type model or ‘J2’-type model.

***Inputs***

r = planet-centric radius (m)

long = longitude (currently not used) (rad)

lat = planet-centric latitude (rad)

mu = planetary gravitational constant (SI units)

J2 = planetary zonal harmonic of the second order constant (SI units)

gravmodel = flag for what kind of gravitational model (‘rsquared’ or ‘J2’)

***Output***

g = gravitational acceleration vector in the vehicle-carried frame

The structure of the subdirectories of each type of process is same. Figure 6 shows the structure is a generic process subdirectory. Explanations for each function are given below.

**\_DOF\_files**

**\_DOFEntry.m**

**\_DOFEntryCOV.m**

**\_DOFEntryEOM.m**

**\_DOFEntryEOMjac.m**

**\_DOFEntryEOMNoiseJac.m**

**\_DOFEntryEOMProcessNoise.m**

Figure : Structure that is common between each \_DOF files.

### \_DOFEntry.m

This file will create an augmented state vector equation of motion. This the main propagation file called by the propagateState.m file to generate xdot and Pdot information. This file in turn calls the EntryEOM and EntryCOV files to calculate xdot and Pdot.

***Inputs***

T = current time

x = current states and covariance matrix elements

inputs = struct containing information used for propagation (specific for each application)

***Outputs***

Xdot = augmented state vector that includes statedot and covdot

### \_DOFEntryCOV.m

This file will create the Pdot information. This file in returns the information to the \_DOFEntry function.

***Inputs***

t = current time

xcov = current covariance matrix elements

xstates = current state vector

inputs = struct containing information used for propagation (specific for each application)

***Outputs***

covdot = covariance element rate (Pdot) information

### \_DOFEntryEOM.m

This file will create the xdot information. This file in returns the information to the \_DOFEntry function.

***Inputs***

T = current time

x = current states and covariance matrix elements

inputs = struct containing information used for propagation (specific for each application)

***Outputs***

Xdot = augmented state vector that includes statedot and covdot

### \_DOFEntryEOMjac.m

This file contains the Jacobian of the equations of motion (A-matrix) present in the \_DOFEntyEOM file. This file is used in the \_DOFEntryCOV function to calculate the Pdot values using the Riccati-type equations.

***Inputs***

T = current time

x = current states and covariance matrix elements

inputs = struct containing information used for propagation (specific for each application)

***Outputs***

F = state equations’ Jacobian

### \_DOFEntryEOMNoiseJac.m

This file contains the Jacobian of the process noise (i.e B-matrix). This file is used in the \_DOFEntryCOV function to calculate the Pdot values using the Riccati-type equations.

***Inputs***

T = current time

x = current states and covariance matrix elements

inputs = struct containing information used for propagation (specific for each application)

***Outputs***

B = process noise’s Jacobian

### \_DOFEntryEOMProcessNoise.m

This file contains the process noise matrix (i.e Q-matrix). This file is used in the \_DOFEntryCOV function to calculate the Pdot values using the Riccati-type equations.

***Inputs***

T = current time

x = current states and covariance matrix elements

inputs = struct containing information used for propagation

***Outputs***

Q = process noise matrix

## Utilities

The utilities directory contains functions and script files that are used at various places in EKF tool. Figure 7 shows a list of all the files in the utilities subdirectory.

**Utilities**

**Quat2EulUncer.m**

**AeroRecon.m**

**R\_iner2veh.m**

**calculateAOABeta.m**

**R\_veh2body.m**

**calculateMachUncer.m**

**R\_veh2body\_wq.m**

**CFDInterpolate.m**

**Rot321\_angles.m**

**Eul2Quat.m**

**sinterp3.m**

**Eul2QuatUncer.m**

**generate\_MEADS\_Data.m**

**skew.m**

**vec\_veh2body.m**

**load\_cfd\_data.m**

**vec\_Vw\_2\_alphabeta.m**

**makemat.m**

**Velvec2VGammaAzimuth.m**

**makevec.m**

**VGammaAzimuth2VelVec.m**

**norm\_vec.m**

**Vw\_2\_alphabeta.m**

**processCovariance.m**

**yz2ConeandClockCapsule.m**

**qmult\_left.m**

**yz2ConeandClockSphCone.m**

**Quat2Eul.m**

Figure : List of files in the Utilities directory.

### AeroRecon.m

This file is a script that can be called from the “\_Data\_driver.m” file in the Applications directory. This file will process the fwdstate struct and calculate the aerodynamic coefficients, such as CA, CN, CD and CL. The script also has an in-built plotter routine to graph the calculated coefficients and the actual coefficients from the output of a POSTII run.

***Inputs***

None (this is a script file). The workspace should have the results of an EKF run and the actual trajectory from a POST II run.

***Outputs***

None (this is a script file). Aerodynamic coefficients are calculated from the reconstructed states and these coefficients are added to the workspace.

### CalculateAOABeta.m

Calculate angle of attack (alpha) and sideslip angle (beta) and their 1σ uncertainties.

***Inputs***

state = state vector [position;velocity;quaternions;pressure;density]

P\_vec = vector representation of upper-tri of covariance matrix

vi, ve = index for start and end of velocity in state vector

qi, qe = index for start and end of quaternions in state vector

wind\_vel = wind speed in body frame

***Outputs***

AOA = angle of attack

Sideslip = sideslip angle

UAOA = uncertainty in the angle of attack

USideslip = uncertainty in sideslip angle

### CalculateMachUncer.m

Calculate the uncertainty in the Mach number

***Inputs***

Pvec = vector representation of covariance matrix

Vvec = velocity vector

i, e = index of start and stop of the velocities in the covariance matrix

***Outputs***

Machuncer = uncertainty in the Mach number

### CFDInterpolate.m

Looks up the coefficient of the pressure at the pressure port locations

***Inputs***

Clock = clock angle (rad)

Cone = cone angle (rad)

alpha = angle-of-attack (rad)

beta = side-slip angle (rad)

Mach = mach number of the freestream

aero\_data = Cp tables from CFD

***Outputs***

Cp = coefficient of pressure

### Eul2Quat.m

Takes the flight dynamics Euler angles (roll, pitch, yaw) and convert to quaternion

***Inputs***

bank, pitch, azimuth = Euler angles (rad) in 1,2,and 3 axis

***Outputs***  
q0, q1, q2, q3 = quaternions with q0 being the "scalar" value

### Eul2QuatUncer.m

Takes the flight dynamics Euler angles (roll, pitch, yaw) and convert to uncertainties in quaternion

***Inputs***

bank, pitch, azimuth = Euler angles (rad)in 1, 2, and 3 axis

ubank, upitch, uazimuth = Euler angles uncertainties (rad)in 1,2,and 3 axis

***Outputs***

uq0,uq1,uq2,uq3 = quaternion uncertainties (q0 being the "scalar" value)

### Generate\_MEADS\_Data.m

This function will generate pressure measurements like the MEADS data type.

***Inputs***

actual = simulated POST2 output struct

PressureInputs = struct with pressure port information

***Outputs***

Pressure\_measurement = vector of measurements for time history

### Load\_cfd\_data.m

Loads CFD data containing Cp table used to generate pressure measurements for MEADS datatype. This function is copied from NASA LaRC-Analytical Mechanics Associates

supplied files as it contains additional data to be added to the MSL Cp look-up table.

***Inputs***

filename = name of the file containing the CFD data

gamma = specific heat ratio -- used to generate pressure

***Outputs***

CFDdata = structure that has Cp table from the CFD data

### Makemat.m

Takes a vector and makes a symmetric matrix for that vector.

***Inputs***

vec = vector containing the upper triangle elements of a symmetric matrix

***Outputs***

mat = symmetric matrix

### Makevec.m

Takes the upper triangle of a symmetric matrix and makes it into a vector.

***Inputs***

mat = symmetric matrix

***Outputs***

vec = vector containing the upper triangle elements of a symmetric matrix

### Norm\_vec.m

Function takes the norm of the components of a vector at every timeline.

***Inputs***

vec = vector with components

***Outputs***

normvec = norm of the components

### ProcessCovariance.m

Calculate uncertainties (i.e. 1 sigma values) from covariance matrix.

***Inputs***

Calculate uncertainties (i.e. 1 sigma values) from covariance matrix

***Outputs***

Uncertainty = vector of 1-sigma uncertainties

### Qmult\_left.m

Returns the left matrix needed for quaternion multiplication.

***Inputs***

q0,q1,q2,q3 = quaternions (q0 = scalar)

***Outputs***

qmat = left matrix

### Quat2Eul.m

Takes quaternions and calculates the equivalent Euler angles.

***Inputs***

q0,q1,q2,q3 = quaternions with q0 being the "scalar" value

***Outputs***

bank,pitch,azimuth = Euler angles (rad) in 1,2,and 3 axis

### Quat2EulUncer.m

Calculate uncertainty in the aerospace Euler angles based on the uncertainty in the quaternions.

***Inputs***

q0,q1,q2,q3 = quaternions with q0 being the "scalar" value

bank,pitch,azimuth = Euler angles (rad) in 1,2,and 3 axis

uq0,uq1,uq2,uq3 = uncertainty in quaternion

***Outputs***

Ubank,Upitch,Uazimuth = uncertainties in Euler angles (rad) the 1,2,& 3 axis

### R\_iner2veh.m

Rotation vector to take a vector from inertial frame to vehicle-carried frame.

***Inputs***

lat = latitude of the vehicle, deg (assumed to be geocentric latitude)

lon = longitude of the vehicle, deg

***Output***

rotVI = rotation matrix to go from inertial to vehicle-carried frame

### R\_veh2body.m

Rotation matrix to take vectors from the vehicle frame to the body frame.

***Inputs***

bank = bank or roll angle (rotation angle in x-axis), deg

pitch = pitch angle (rotation angle in y-axis), deg

azimuth = azimuth or yaw angle (rotation angle in z-axis), deg

***Output***

rotBV = rotation matrix to take matrices from vehicle frame to body frame

### R\_veh2body\_wq.m

Rotation matrix from vehicle-fixed frame to body-frame using quaternions.

***Inputs***

q0,q1,q2,q3 = quaternions (q0 = scalar)

***Output***

rotMat = rotation matrix from veh. to b-frame

### Rot321\_angles.m

Takes a 321 angle rotation matrix and calculates the Euler rotation angles from it.

***Input***

rotMat = rotation matrix

***Outputs***

Roll = angle is the x axis

Pitch = angle in the y axis

Azimuth = angle in the z axis

### Sinterp3.m

This function is a 3D look-up table routine. This code is the exact 3D look-up code developed by Analytical Mechanics Associates (AMA) and used by them extensively for Cp lookup tables to generate MEADS-like data. This function was provided by AMA when Cp tables were provided for this NRA.

***Input***

x,y,z = list of the 3 variables which are used for the look-up process

v = list of the value being looked-up in this subroutine

xi,yi,zi = list of the known values of the 3 variables

***Output***

vi = value looked-up by the subroutine

### Skew.m

Create a skew matrix used in cross products.

***Input***

vec = vector of 3 x 1

***Output***

skewmat = skew matrix

### specialInterp.m

Function uses linear interpolation to do 1D interpolation.

***Input***

t = vector of variables used in the 1D interpolation

a = vector of values being interpolated

time\_a = variable values used in the interpolation

***Outputs***

b = interpolated variable

### Vec\_veh2body.m

Converts the components of a vector in the vehicle-carried frame (i.e. local horizontal) to the body frame.

***Inputs***

vecxin = x-component of the vector in vehicle frame

vecyin = y-component of the vector in vehicle frame

veczin = z-component of the vector in vehicle frame

azimuthin = azimuth angle (or yaw angle), deg

pitchin = pitch angle, deg

bankin = bank angle (or roll angle), deg

***Output***

vecrotx = x-component of the vector in body frame

vecroty = y-component of the vector in body frame

vecrotz = z-component of the vector in body frame

### Vec\_Vw\_2\_alphabeta.m

Takes a vector of the vehicle’s velocity (as in the velocity time history) with respect to the wind in the body frame and finds the angle of attack (alpha) and sideslip angle (beta).

***Input***

Vwvec = velocity column vector of vehicle vel. wrt to the wind in body-fixed

Frame. Assumed that Vwvec is a column matrix.

***Output***

alpha = angle-of-attack, deg (also alpha\_x as opposed to total AOA)

beta = sideslip angle, deg

### Velvec2VGammaAzimuth.m

Takes the 3 x 1 velocity vector in the local horizontal (vehicle carried) coordinate frame and computes the velocity magnitude, flight path angle (+ is up) and azimuth angle.

***Input***

Velvec = velocity column vector of vehicle vel in the vehicle carried frame

***Output***

Vmag = velocity magnitude

Gamma = Flight path angle (+ is up) (rad)

Azimuth = azimuth angle (0 = due North, pi/2 = due East) (rad)

### VGammaAzimuth2VelVec.m

Takes a velocity magnitude (V), flight path angle (gamma) and azimuth angle (azimuth) and converts it into a velocity vector in the vehicle-carried (local horizontal) frame.

***Input***

V = velocity magnitude

Gamma = Flight path angle (+ is up) (rad)

Azimuth = azimuth angle (0 = due North, pi/2 = due East) (rad)

***Output***

velocityVec = velocity column vector of vehicle vel in the vehicle carried frame

### Vw\_2\_alphabeta.m

Takes the vehicle’s velocity (as in a single velocity vector) with respect to the wind in the body frame and finds the angle of attack (alpha) and sideslip angle (beta). This file is different from Vwvec\_2\_alphabeta as this function can only handle one velocity vector at a time, while the Vwvec file can go through an array full of velocity vectors at different time steps.

***Input***

Vw = velocity column vector of vehicle vel. wrt to the wind in body-fixed

Frame. Assumed that Vwvec is a column vector.

***Output***

alpha = angle-of-attack, deg (also alpha\_x as opposed to total AOA)

beta = sideslip angle, deg

### Yz2ConeandClockCapsule.m

Calculate clock and cone angle for various points on the aeroshell of a capsule.

***Input***

yloc = y coordinates of port (origin at tip of heatshield, axial-out-and

down coordinate frame)

zloc = z coordinates of port (origin at tip of heatshield, axial-out-and

down coordinate frame)

rnose = nose radius

deltamax = cone half angle (deg.)

***Output***

Clock = clock angle (angle in the standard flight dynamics y-z plane) (rad)

Cone = clock angle (angle in the standard flight dynamics x-z plane) (rad)

### Yz2ConeandClockSphCone.m

Calculate clock and cone angle for various points on the aeroshell of a sphere cone.

***Input***

yloc = y coordinates of port (origin at tip of heatshield, axial-out-and

down coordinate frame)

zloc = z coordinates of port (origin at tip of heatshield, axial-out-and

down coordinate frame)

rnose = nose radius

rshell = max shell diameter for sphere cone (meters)

deltamax = cone half angle (deg.)

***Output***

Clock = clock angle (angle in the standard flight dynamics y-z plane) (rad)

Cone = clock angle (angle in the standard flight dynamics x-z plane) (rad)

## Application\_Master\_Files

The Application\_Master\_Files directory has functions and scripts that are the driver files for the reconstruction of different datasets. This is the directory in which the user should create a folder for a new application using the functions and files from the other directories. Currently, this directory has files for seven applications. These applications are:

1. CEV\_Ballistic\_Range\_Simulated – the dataset is simulated from POST output of a Crew Exploration Vehicle (CEV) model being fired from a ballistic range gun.
2. Mars\_Exploration\_Rover\_A – the dataset is the actual data from MER-A that has since been reconstructed using an earlier version of the GT\_EKF tool. See Ref. 6 for details about that specific reconstruction. All of the necessary files needed for the MER reconstruction are present in this folder and they do not need any files outside of this subdirectory.
3. Mars\_Pathfinder – the dataset is the actual data from Pathfinder that has since been reconstructed using an earlier version of the GT\_EKF tool. See Ref. 2 for details about that specific reconstruction. All of the necessary files needed for the Pathfinder reconstruction are present in this folder and they do not need any files outside of this subdirectory.
4. Mars\_Phoenix – the dataset us the actual data from Phoenix. Reconstruction results will be published soon. Ref. 7 will show results from this reconstruction.
5. MER\_Simulated\_nodisp – the dataset is simulated from POST output of a Mars entry vehicle with conditions similar to the Mars Exploration rovers. This dataset is generated from POST files that have no dispersions in aerodynamics, wind or atmosphere.
6. MER\_Simulated\_withaeroatmowinddisp – the dataset is simulated from POST output of a Mars entry vehicle with conditions similar to the Mars Exploration rovers. This dataset is generated from POST files that have dispersions in aerodynamics, wind and atmosphere.
7. MER\_Simulated\_withaerodisp – the dataset is simulated from POST output of a Mars entry vehicle with conditions similar to the Mars Exploration rovers. This dataset is generated from POST files that have no dispersions in wind and atmosphere, but some dispersion in the aerodynamics.

Cases 1 and 5-7 were used to test the aerodynamic and atmospheric reconstruction ability of the tool, since the simulated datasets contained MEADS-like pressure data. The results from actual CEV ballistic range data for trajectory and atmospheric reconstruction has been already presented at a conference (see Ref. 1). The reconstruction of the MER simulated data allows validation for new functionalities included in the EKF tool since Ref. 1 was presented. The new reconstruction results will be shortly presented at a conference, although the paper has yet to be submitted to a specific conference. All of these verifications will help the tool with reconstruction of the actual MEADS dataset when that information becomes available.

# Sample Application Case (MER Simulated Data)

It would be tedious to go through the master files and setup information for each and every application presented in this code. Thus, this manual will go through the set-up of one application (MER\_Simulated\_nodisp), and hopes the user can draw similarities from this example to figure out the structure of the other applications or how to create a new application. Figure 8 shows the contents of this specific application. Other applications have similar structure.

**MER\_Simulated\_nodisp**

**UserInputs.m**

**CFD\_Data**

**Meas\_nodisp.mat**

**Plotter**

**RandomSeed.mat**

**addpaths.m**

**Generate\_MER\_data.m**

**SpeedSound.mat**

**MER\_Simulated\_Data\_driver.m**

**Test\_nodisp.mat**

**mixMatch.m**

**Process\_and\_intialize.m**

Figure : Structure of the MER\_Simulated\_nodisp application.

The main script file is the “MER\_Simulated\_Data\_driver.m” file. This is the script file that should be run to generate all results. This script has the flags and conditions set up to process the data in one of three ways: (1) Deterministic mode, (2) Forward run of EKF, (3) Forward, Backward and Smoothed run of EKF. These conditions are controlled using the ‘deterministicRun’ and ‘smoother\_flag.’ ‘deterministicRun’ flag overwrites the control from the smoother\_flag if EKF mode is turned-off.

Next, the simulated data is generated using the ‘UserInputs.m,’ ‘generate\_MER\_data.m’ and ‘Process\_and\_initialize.m’ script files. Whereas the ‘\_Data\_driver.m’ can be re-used wholly in another application, the scripts generating and processing the data are different in each application. These files ultimately have to create a vector for the initial state conditions, and if EKF is being used, have to give the initial covariance matrix and the measurements to be processed arranged in chronological order. Additionally, an ‘inputs’ struct has to be created for propagation purposes. This struct has been mentioned before. Figure 9 shows an example of contents in the inputs struct. It will be very time consuming to explain each and every element of this struct. The user is advised to look through the data generating and processing scripts to find out about each and every entry of the inputs struct. Note, that this struct is by no means limited to only the entries shown for this application. The user may add or subtract entries to suit his or her needs.

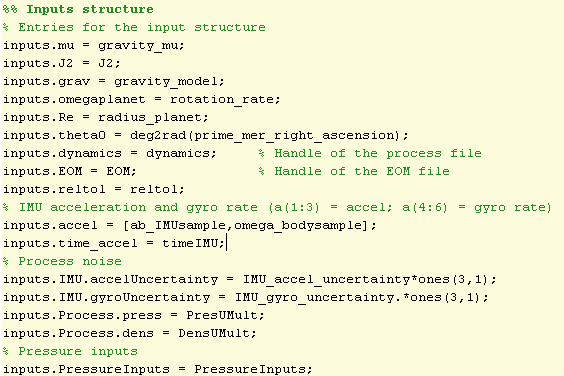


Figure : Sample elements of the inputs struct. This example is from the MER\_Simulated\_Data\_nodisp application.

‘UserInputs.m’ is the first script that is run from the inside of the ‘\_Data\_driver.m’ file. This file has conditions the user will set, such as initial conditions, name of the .mat file that has the Cp look-up table, or the .mat file with the POST output.

‘generate\_ MER\_data’ is run next. This file will generate the various measurements that need to be processed by the filter. The IMU measurements will at the very least need to be generated as for most process files, sensed acceleration data is needed. In this application, the pressure port measurements and the radar altimeter data is also generated. This data is re-sampled (using interpolation schemes) to achieve the desired sampling rate of the measurements. Additionally, noise can be added here to the data to simulate the randomness of the sensor measurements. Finally, the function ‘mixMatch.m’ is used to arrange the data chronologically and also create a vector that has flags telling the filter what type of measurement will be processed at each time increment. This function was created for this application. Parts of it can be re-used for other situations, but in general, the user should customize his or her own code here.

Finally, the ‘process\_and\_initialize.m’ script is run to generate structs called ‘obs,’ ‘initial,’ and ‘inputs.’ The obs struct includes information related to the measurements that will be processed by the EKF. This struct will be ignored if deterministic mode is used. The ‘initial’ struct contains the initial state and covariance values. The ‘inputs’ struct has been described earlier.

The data files shown in this subdirectory contain the outputs from the POST file (‘meas\_nodisp.mat’ and ‘test\_nodisp.mat’), random seed used to generate the noisy data (‘RandomSeed.mat’), and a table of the speed of sound versus altitude (‘SpeedSound.mat’). The last data file is specific to this application where pressure measurements had to be generated and Mach number (and in turn the speed of sound) was a necessary input.

The subdirectory ‘CFD\_data’ contains .mat files that have the Cplook-up tables. In this case, this data is specific to the MSL/Mars entry-like conditions. The CEV simulated data application has Cp look-up tables specific for that situation. The ‘Plotter’ subdirectory has plotting routines to take the reconstructed results and plot some parameters of interest. These scripts can be run from the inside of the ‘\_Data.driver.m’ files.

In the condition the code has been delivered at the time of writing this manual, running the ‘MER\_Simulated\_Data\_driver.m’ file will result in a forward run of the EKF and then results will be plotted using the ‘plot\_forward\_states.m’ file. The aerodynamic coefficients will also be reconstructed and plotted using the ‘AeroRecon.m’ script described earlier.

# References

1. Dutta, S. and Braun, R.D., "Mars Entry, Descent, and Landing Trajectory and Atmosphere Reconstruction," AIAA 2010-1210, 48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition, Orlando, FL, January 2010.
2. Christian, J.A., Verges, A.M., and Braun, R.D., “Statistical Reconstruction of Mars Entry, Descent, and Landing Trajectories and Atmospheric Profiles,” AIAA SPACE Conference and Exposition, Long Beach, CA, 18-20 Sep. 2007.
3. Spencer, D.A., Blanchard, R.C., Braun, R.D., Kallemeyn, P.H., and Thurman, S.W., “Mars Pathfinder Entry, Descent, and Landing Reconstruction,” Journal of Spacecraft and Rockets, Vol. 36, No. 3 May-June 1999, pp 357-366.
4. Fraser, D.C., and Potter, J.E., “The Optimum Linear Smoothers as a Combination of Two Optimum Linear Filters,” IEEE Transactions on Automatic Control, Vol. 14, No. 8, 1969, pp. 387-390.
5. Vinh, N.X., Busemann, A., and Culp, R.D., *Hypersonic and Planetary Entry Flight Mechanics*, The University of Michigan Press, Ann Arbor, MI, 1980.
6. Wells, G.W. and Braun, R.D., "Reconstruction of the Spirit Mars Exploration Rover Entry, Descent and Landing Performance," AA-3-2008-16, **2nd International ARA Days Conference**, Arachon, France, October 2008.
7. Dutta, S., Clark, I.G., Russell, R.P., and Braun, R.D., “Statistical Entry, Descent and Landing Performance Reconstruction of the Mars Phoenix Lander.” International Planetary Probe Workshop 8, Portsmouth, VA, June 2011. (submitted).

# Index

\_Data\_driver.m 26

\_DOFEntry.m 15

\_DOFEntryCOV.m 15

\_DOFEntryEOM.m 15

\_DOFEntryEOMjac.m 16

\_DOFEntryEOMNoiseJac.m 16

\_DOFEntryEOMProcessNoise.m 16

‘inputs’ structure 7, 27

AeroRecon.m 17

Application\_Master\_Files 25

CalculateAOABeta.m 18

CalculateMachUncer.m 18

CEV\_Ballistic\_Range\_Simulated 25

CFDInterpolate.m 18

Eul2Quat.m 19

Eul2QuatUncer.m 19

ExtendedKalmanFilter.m 7

Generate\_MEADS\_Data.m 19

Gravity.m 14

H\_Port\_pressure.m 9

H\_Radar\_altimeter.m 9

Load\_cfd\_data.m 19

Makemat.m 19

Makevec.m 20

Mars\_Exploration\_Rover\_A 25

Mars\_Pathfinder 25

meas\_Port\_pressure.m 10

Measurement\_Files 9

MER\_Simulated 25, 26

Norm\_vec.m 20

Phoenix 25

Process\_Files 12

ProcessCovariance.m 20

PropogateState.m 8, 15

Qmult\_left.m 20

Quat2Eul.m 20

Quat2EulUncer.m 21

R\_iner2veh.m 21

R\_Port\_pressure.m 10

R\_Radar\_altimeter.m 11

R\_veh2body.m 21

R\_veh2body\_wq.m 21

Rot321\_angles.m 21

Sinterp3.m 22

SixDOF 12

SixDOFwAtmo 12

SixDOFwAtmoWind 13

Skew.m 22

Smoother.m 8

specialInterp.m 22

ThreeDOF 13

Utilities 17

Vec\_veh2body.m 22

Vec\_Vw\_2\_alphabeta.m 23

Velvec2VGammaAzimuth.m 23

VGammaAzimuth2VelVec.m 23

Vw\_2\_alphabeta.m 23

Yz2ConeandClockCapsule.m 24

Yz2ConeandClockSphCone.m 24